Администрирование Linux. Часть4.

Пособие для дистанционных курсов.
http://linuxnavigator.ru.

03.07.2009

ОГЛАВЛЕНИЕ

Настройка firewall....3
Внутреннее устройство. 4
Цепочки. 5
Таблицы. 5
Порядок прохождения таблиц и цепочек. 7
Пограмма iptables....8
Команды программы iptables. 8
Опции программы iptables. 10
Критерии отбора пакетов. 10
Действия и переходы. 19
NAT преобразования. 21
Управление netfilter. 26
Стартовые скрипты. 26
Примеры. 30
Пример 1. 30
Пример 2. 31
Небольшое задание на закрепление материала. 32

НАСТРОЙКА FIREWALL.

При написании этого раздела, я сначала очень внимательно изучил вот эту страницу в Интернет: http://www.opennet.ru/docs/RUS/iptables/ и даже имел смелость использовать много материала из русского перевода.

Эта часть посвящена конфигурации firewall в Linux. Тема большая, сложная. Пока мы её будем изучать, наш сервер будет стоять открытым. Это не хорошо. Поэтому сначала создадим небольшой файлик, закроем в firewall всё, что надо закрыть. Вернее все закроем и откроем только то, что надо открыть. И только после этого, не торопясь рассмотрим, что мы сделали и как все это работает.

Пока вы до конца не будете понимать, как устроен firewall, рекомендую все действия по его настройке производить в локальной консоли, доступной через клиента виртуальной машины.

При помощи клиента подключитесь к вашей виртуальной машине. Зайдите в систему как пользователь root. Перейдите в свою домашнюю директорию. Создайте в ней директорию *bin*.

```
# mkdir ~/bin
```

Перейдите в директорию *bin*. И создайте файл *rc.fw*, следующего содержания.

```
#!/bin/bash
IPT=/sbin/iptables
IPTR=/sbin/iptables-restore
IPTS=/sbin/iptables-save

reset()
{
$IPT -F
$IPT -t nat -F
$IPT -t mangle -F
$IPT -P INPUT DROP
$IPT -P FORWARD DROP

# STATE RULES ======
$IPT -A INPUT -m state --state INVALID -j DROP
$IPT -A FORWARD -m state --state INVALID -j DROP
$IPT -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# LOCALHOST ===========
$IPT -A FORWARD -i ! lo -s 127.0.0.1 -j DROP
$IPT -A INPUT -i ! lo -s 127.0.0.1 -j DROP
$IPT -A FORWARD -i ! lo -d 127.0.0.1 -j DROP
$IPT -A INPUT -i ! lo -d 127.0.0.1 -j DROP
$IPT -A INPUT -d 127.0.0.1 -j ACCEPT
$IPT -A INPUT -s 127.0.0.1 -j ACCEPT

# ICMP ENABLED =====
$IPT -A INPUT -p icmp -j ACCEPT
```

```
# SSH enable =======
$IPT -A INPUT -p tcp -m state --state NEW --dport 22 -m recent --update \
       --seconds 20 -j DROP
$IPT -A INPUT -p tcp -m state --state NEW --dport 22 -m recent --set \
       -j ACCEPT
}

init()
{
echo -n "Init firewall... "
reset
####### INSERT you rules

#######
# SAVE rules in file ==
$IPTS -c > /etc/sysconfig/iptables
echo "Done"
}

case $1 in
   reset) reset;;
   init) init;;
   *) echo "Usage: rc.fw reset|init"
esac
```

Сделайте файл исполняемым.

```
# chmod u+x ~/bin/rc.fw
```

И запустите скрипт.

```
# ~/bin/rc.fw init
Init firewall... Done
#
```

Если на экран выводятся сообщения об ошибках, исправьте файл и запустите его снова.

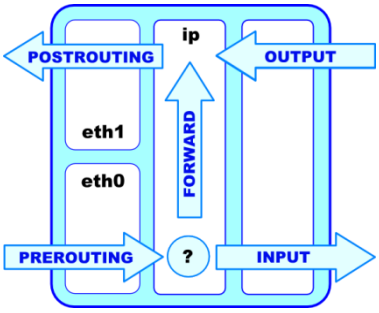
После того, как все заработает, попробуйте подключиться к вашему серверу при помощи putty.

Скрипт, который мы запустили, ограничивает количество подключений по ssh. Не чаще чем один раз в 20 секунд. Поэтому, если у вас была неудачная попытка подключения, подождите 20 секунд и попробуйте войти снова.

ВНУТРЕННЕЕ УСТРОЙСТВО.

Для создания firewall в Linux был создан отдельный проект — netfilter (http://www.netfilter.org). Firewall работает на уровне ядра, поэтому, сначала мы посмотрим внутреннее устройство.

Для облегчения понимания, нарисую небольшую схему, которую необходимо запомнить. Схема весьма условная и адаптированная. На самом деле все немного сложнее. Я нарисую стек TCP/IP, но не так как это принято в классической литературе, где уровни рисуют снизу вверх: от канального, до уровня приложений. Я нарисую его боком.



Слева находятся сетевые интерфейсы. На схеме их два: *eth0* и *eth1*. Для облечения понимания, предположим, что все пакеты приходят к нам через *eth0* (*PREROUTING*), а уходят через *eth1*(*POSTROUTING*).

Затем идет уровень IP, уровень принятия решения о маршрутизации пакетов. Входящий пакет попадает в точку принятия решения (*обозначена символом ?*) и может быть передан программному обеспечению, работающему на нашей машине (*INPUT*). В случае если он не предназначен для наших программ, он будет передан на выходной интерфейс (*FORWARD — POSTROUTING*).

Так же существуют пакеты, которые генерирует наше программное обеспечение (*OUTPUT*). Мы предположим, что эти пакеты сразу уходят с нашей машины через выходной интерфейс. Но вы должны понимать, что на самом деле они тоже попадают в точку принятия решения о маршрутизации и могут быть переданы обратно на нашу машину или отправлены на выходной интерфейс.

ЦЕПОЧКИ.

В firewall в Linux существует понятие цепочки. Цепочка — это некоторая часть пути пакета, на которую можно подключать правила обработки (фильтрации). На нашей схеме, стрелки — это и есть цепочки.

Каждая цепочка имеет свое уникальное имя, которое пишется большими буквами:

PREROUTING — пакеты до принятия решения о маршрутизации. Т.е. пакеты, которые только пришли на нашу машину и мы еще не знаем, куда они будут направлены дальше, нашему программному обеспечению или уйдут через другой интерфейс.
POSTROUTING — пакеты, которые гарантированной уходят с нашей машины. Это могут быть как транзитные пакеты, так и сгенерированные нашим программным обеспечением.
INPUT — пакеты, предназначенные для программ, работающих на нашей машине.
OUTPUT — пакеты, сгенерированные программами, работающими на нашей машине. Причем, мы еще не знаем, куда они будут направлены дальше: обратно к нам или уйдут с нашей машины.
FORWARD — транзитные пакеты.

ТАБЛИЦЫ.

Кроме цепочек, в netfilter существует такое понятие как таблица. То, что я нарисовал на предыдущей схеме — это таблица. В ней есть цепочки. Таблиц четыре штуки:

raw
filter
nat
mangle

Каждая таблица имеет индивидуальный набор цепочек и применяется для определенных действий с пакетами.

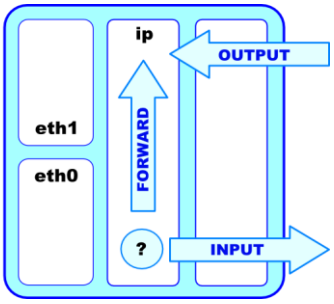


Рисунок 1. Таблица filter

Таблица *filter* используется для фильтрации пакетов. Т.е. выполняет функции firewall. В ней используются три цепочки: *INPUT*, *FORWARD* и *OUTPUT*.

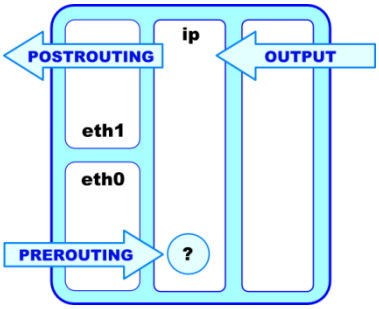


Рисунок 2. Таблица nat.

Таблица *nat* предназначена для описания правил NAT (Network Address Translations). В ней используются три цепочки: *PREROUTING*, *POSTROUTING* и *OUTPUT*.

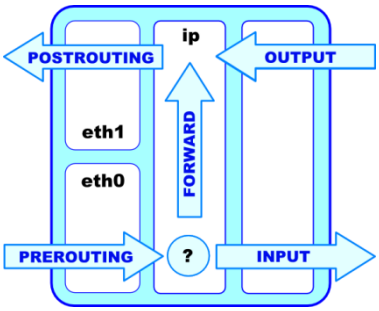


Рисунок 3. Таблица mangle

Таблица *mangle* предназначена для внесения изменений в остальные заголовки пакетов и нестандартные действия с пакетами, свойственные Linux. В таблице можно использовать все пять цепочек: *PREROUTING*, *POSTROUTING*, *INPUT*, *FORWARD* и *OUTPUT*.

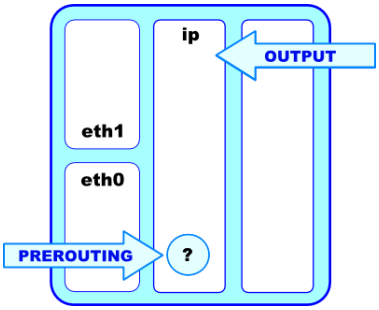
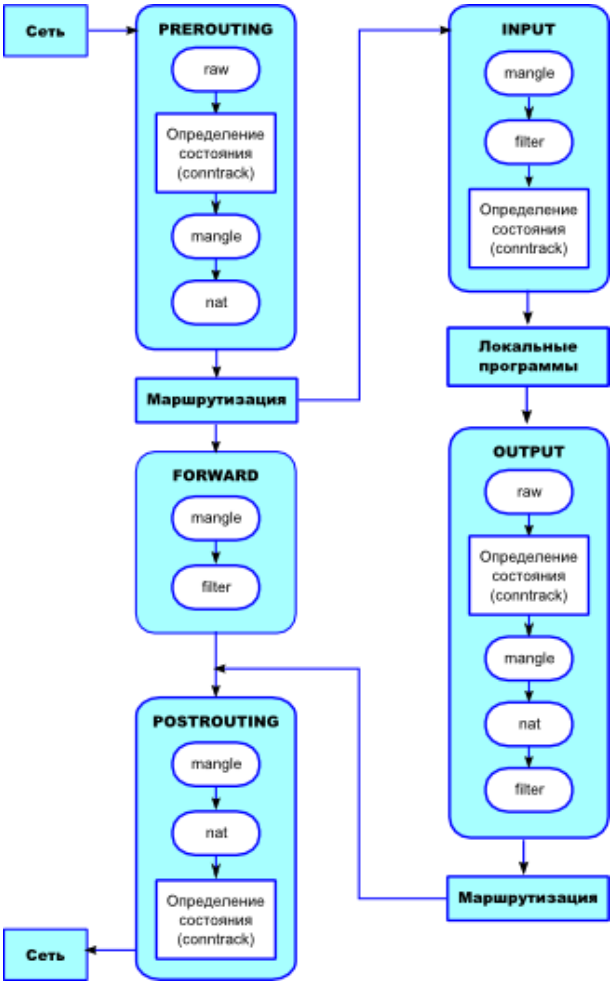


Рисунок 4. Таблица raw.

Таблица *raw* просматривается до передачи пакета системе определения состояний. Используется редко, например, для маркировки пакетов, которые *не* должны обрабатываться системой определения состояний (подробнее об этой системе рассказывается в описании модуля state). Для этого в правиле указывается действие *NOTRACK*. Содержит цепочки *PREROUTING* и *OUTPUT*.

ПОРЯДОК ПРОХОЖДЕНИЯ ТАБЛИЦ И ЦЕПОЧЕК.



Несмотря на то, что в netfilter используется несколько таблиц и цепочек, пакет не может параллельно проходить через все таблицы и цепочки. Он проходит через них последовательно.

Сначала рассмотрим порядок прохождения транзитных пакетов. Запомнить порядок достаточно просто. В первую очередь пакет всегда попадает в таблицу *mangle* и только потом в оставшиеся таблицы этой цепочки.

Таблица	Цепочка
raw	PREROUTING
mangle	PREROUTING
nat	PREROUTING
mangle	FORWARD
filter	FORWARD
mangle	POSTROUTING
nat	POSTROUTING

Порядок прохождения входящих пакетов, предназначенных для программного обеспечения, работающего на нашей машине.

Таблица	Цепочка
mangle	PREROUTING
nat	PREROUTING
mangle	INPUT
filter	INPUT

И наконец, порядок прохождения пакетов, сгенерированных нашим программным обеспечением.

Таблица	Цепочка
raw	OUTPUT
mangle	OUTPUT
nat	OUTPUT
filter	OUTPUT
mangle	POSTROUTING
Nat	POSTROUTING

ПОГРАММА IPTABLES.

Netfilter — это код, который выполняется на уровне ядра операционной системы. Мы должны каким то образом изменять правила фильтрации и его поведение. Для этих целей, на уровне пользователя используется программа *iptables*.

Iptables — это программа, которой все параметры передаются в виде аргументов командной строки. За один вызов программы — одна команда.

Программа может влиять только на одну из таблиц за один вызов. Для определения, с какой таблицей будет работать программа, используется опция *-t*, после которой указывается соответствующая таблица: *filter*, *nat* или *mangle*. Если опция не указана, программа по умолчанию работает с таблицей *filter*. И это правильно. На таблицу *filter* ложится основная нагрузка, в ней располагаются правила firewall. В *nat* будет оно, два правила, а *mangle* может быть совсем пустой.

КОМАНДЫ ПРОГРАММЫ IPTABLES.

Для определения действия, которое программа *iptables* должна выполнить, ей необходимо указать команду.

В таблице приведены команды, которые можно использовать. За один вызов программы можно указать только одну команду.

Команда	Описание
-A, **--append**	Добавляет правило в конец цепочки. При выполнении команды необходимо обязательно указать цепочку. Пример: `iptables -A INPUT -s 10.10.100.100 -j ACCEPT`
-D, **--delete**	Удаляет правило из цепочки. Существуют два способа удаления: по номеру правила или с указанием критериев отбора (*правило записывается точно так же, как оно*

	добавлялось, только указывается команда D).
	
	Пример:
	`iptables -D INPUT 1`
	`iptables -D INPUT --dport 80 -j DROP`

-R, **--replace**	Команда заменяет одно правило другим. Пример: `iptables -R INPUT 1 -s 192.168.0.1 -j DROP`
-I, **--insert**	Команда вставляет правило в указанное место в цепочке. Пример: `iptables -I INPUT 1 --dport 80 -j ACCEPT`
-L, **--list**	Если не указывать имя цепочки, команда показывает все правила текущей таблицы. Если указать имя цепочки — показывает все правила в указанной цепочке. Пример: `iptables -L INPUT`
-F, --flush	Если не указывать имя цепочки, команда удаляет все правила из текущей таблицы. Если указать имя цепочки — будут удалены все правила из указанной цепочки. Пример: `iptables -F INPUT`
-Z, --zero	Каждому правилу в таблице соответствуют два счётчика: количество пакетов и количество байт, сработавших на данном правиле. Команда сбрасывает счетчики в текущей таблице, или, если указано имя цепочки, в заданной цепочке. Пример: `iptables -Z INPUT`
-N, **--new-chain**	Команда создаёт цепочку с заданным именем в текущей таблице. Имя должно быть уникальным и не должно совпадать с зарезервированными именами цепочек и действий. Пример: `iptables -N userchain`
-X, **--delete-chain**	Команда удаляет цепочку пользователя из таблицы. Удалять можно только цепочки, не содержащие правил. Так же на эти цепочки не должно быть ссылок из других цепочек. Пример: `iptables -X userchain`
-P, **--policy**	Команда задает политику по умолчанию для цепочки. Политика определяет, что следует сделать с пакетом, на котором не сработало ни одно правило цепочки. В качестве значения политики можно использовать значения *ACCEPT* и *DROP*. Пример:

	`iptables -P INPUT DROP`

ОПЦИИ ПРОГРАММЫ IPTABLES.

Программе iptables можно указывать дополнительные опции, влияющие на работу программы. Некоторые опции имеет смысл давать только с соответствующими командами.

Опция	Описание
-v, --verbose	Опция заставляет программу выводить дополнительную информацию. Например, в сочетании с командой *-L* будут показаны не только правила, но и округленные значения счётчиков.
-x, --exact	Опция заставляет программу выводить точные значения счётчиков.
-n, --numeric	Опция заставляет программу не делать обратного преобразования из IP адреса в имя машины, если в выводе программы встречаются IP адреса.
--line-numbers	Опция заставляет программу выводить номера правил в цепочке.
-c, --set-counters	Опция позволяет установить значения счётчиков при добавлении правила в цепочку.

КРИТЕРИИ ОТБОРА ПАКЕТОВ.

При добавлении правила необходимо указывать критерии отбора пакетов, на которых это правило будет работать. Правила можно условно разделить на три группы:

- Общие критерии — не зависят от типа протокола и не требуют загрузки специальных модулей.
- Неявные критерии — зависят от типа протокола и не требуют загрузки специальных модулей.
- Явные критерии — перед использованием требуют явной загрузки специальных модулей.

В одном правиле можно указывать сразу несколько разных критериев отбора пакетов. Один и тот же критерий указывать нельзя.

Если в правиле отбора пакета критерий не определён, значит возможно любое значение критерия.

Когда мы говорим о модуле программы iptables, имеются в виду не модули ядра Linux (хотя модули netfilter существуют). Имеется в виду внутренне понятие модуля. После указания «внутреннего» имени модуля появляется возможность использовать критерии, предоставляемые этим модулем.

ОБЩИЕ КРИТЕРИИ.

Общие критерии не зависят от типа протокола и для их использования не требуется загрузка специальных модулей при помощи опции *-m*.

Критерий	Описание
-p	Критерий служит для определения типа протокола. В качестве дополнительного параметра необходимо указывать имя, номер протокола (*см. файл /etc/protocols*) или ключевое слово *ALL*. При указании протокола можно использовать символ *!*, который инвертирует значение критерия. Примеры: `-p tcp`

	-p ! icmp
-s	Критерий определяет IP адрес источника. В качестве дополнительного параметра можно указывать IP адрес машины или сети. В последнем случае необходимо указывать маску подсети. Символ *!* инвертирует значение параметра. Примеры: **-s 192.168.0.1** **-s 10.10.100.0/24** **-s ! 10.10.100.0/255.255.255.0**
-d	Критерий определяет IP адрес получателя. В качестве дополнительного параметра можно указывать IP адрес машины или сети. В последнем случае необходимо указывать маску подсети. Символ *!* инвертирует значение параметра. Примеры: **-d 192.168.0.1** **-d 10.10.100.0/24** **-d ! 10.10.100.0/255.255.255.0**
-i	Критерий определяет интерфейс, с которого был получен пакет. *Использование этого критерия допускается только в цепочках INPUT, FORWARD и PREROUTING, в любых других случаях будет вызывать сообщение об ошибке.* Если имя интерфейса завершается символом +, то критерий задает все интерфейсы, начинающиеся с заданной строки. Символ *!* инвертирует значение параметра. Примеры: **-i eth0** **-i ppp+** **-i ! eth1**
-o	Критерий определяет выходной интерфейс. *Этот критерий допускается использовать только в цепочках OUTPUT, FORWARD и POSTROUTING, в противном случае будет генерироваться сообщение об ошибке.* Если имя интерфейса завершается символом +, то критерий задает все интерфейсы, начинающиеся с заданной строки. Символ *!* инвертирует значение параметра. Примеры: **-o eth0** **-o ppp+** **-o ! eth1**
-f	Критерий определяет все фрагменты фрагментированного пакета, кроме первого.

	Символ *!* инвертирует значение параметра.
	Примеры:
	`-f`
	`! -f`

НЕЯВНЫЕ КРИТЕРИИ.

Для того, чтобы воспользоваться явными критериями, необходимо при помощи критерия *-p* указать протокол и только после этого определять явные критерии.

TCP КРИТЕРИИ.

TCP критерии можно использовать только после определения протокола TCP: *-p tcp*.

Критерий	**Описание**

--sport	Критерий определяет порт, с которого был отправлен пакет. В качестве дополнительного параметра можно указывать номер порта или имя службы (*см. файл /etc/services*). Так же можно указывать диапазон портов, разделенных символом *:*.
	Символ *!* инвертирует значение параметра.
	Примеры:
	`--sport 1024:65535`
	`--sport 80`
	`--sport ! www`

--dport	Критерий определяет порт назначения. В качестве дополнительного параметра можно указывать номер порта или имя службы (*см. файл /etc/services*). Так же можно указывать диапазон портов, разделенных символом *двоеточие*.
	Символ *!* инвертирует значение параметра.
	Примеры:
	`--dport 1024:65535`
	`--dport 80`
	`--dport ! www`

--tcp-flags	Критерий позволяет контролировать флаги TCP пакета. В качестве дополнительных параметров указывают список интересующих флагов и через пробел список флагов, значения которых должны быть равны 1. Так же можно использовать ключевые слова *ALL* — все, *NONE* — ни одного. Символ *!* инвертирует значение параметра. Пример: `--tcp-flags SYN,ACK,FIN SYN`
--syn	Критерий определяет TCP пакет с установленным флагом SYN и со сброшенными флагами ACK и FIN, что соответствует запросу на соединение. Примеры: `--syn`

	`! --syn`

UDP КРИТЕРИИ.

UDP критерии можно использовать только после определения протокола UDP: *-p udp*.

Критерий	Описание

--sport	Критерий определяет порт, с которого был отправлен пакет. В качестве дополнительного параметра можно указывать номер порта или имя службы (*см. файл /etc/services*). Так же можно указывать диапазон портов, разделенных символом *двоеточие*.
	
	Символ *!* инвертирует значение параметра.
	
	Примеры:
	`--sport 1024:65535`
	`--sport 21`

--dport	Критерий определяет порт назначения. В качестве дополнительного параметра можно указывать номер порта или имя службы (*см. файл /etc/services*). Так же можно указывать диапазон портов, разделенных символом *двоеточие*.
	
	Символ *!* инвертирует значение параметра.
	
	Примеры:
	`--dport 1024:65535`
	`--dport 21`

ICMP КРИТЕРИЙ.

ICMP критерий можно использовать только после определения протокола ICMP: *-p icmp*.

Существует всего лишь один icmp критерий: *--icmp-type*, определяющий тип ICMP пакета. В качестве дополнительного параметра можно указывать имя или номер ICMP пакета.

Например:

```
--icmp-type echo-request
```

Чтобы посмотреть список всех ICMP пакетов, можно воспользоваться программой *iptables*:

```
iptables -p icmp --help
```

ЯВНЫЕ КРИТЕРИИ.

Для того, чтобы воспользоваться любым явным критерием, необходимо при помощи параметра *-m* загрузить необходимый модуль и только затем можно указывать явные критерии.

ЯВНЫЕ КРИТЕРИИ. КРИТЕРИЙ LIMIT.

На самом деле существует довольно подробное объяснение как работает этот критерий, но мы ограничимся простым вариантом объяснения работы — критерий *limit* ограничивает количество срабатываний правила.

Перед использованием критерия необходимо воспользоваться параметром *-m limit*.

Критерий	Описание

--limit-burst	Критерий определяет количество пакетов, на которых правило будет работать. Если это параметр не указан явно, его значение принимается равным 5. Пример: --limit-burst 1
--limit	Критерий определяет, сколько раз в единицу времени будет срабатывать правило. В качестве дополнительного параметра, указывающего единицу времени, можно использовать: *N/s* (секунда), *N/m* (минута), *N/h* (час), *N/d* (день). Вместо буквы *N* необходимо подставить число. Примеры: --limit 1/s

Например, установлены следующие критерии отбора пакетов:

```
-p tcp --dport 80 -m limit --limit 1/s --limit-burst 4 \
-j ACCEPT
```

Это значит, что четыре пакета (*--limit-burst 4*) протокола TCP (*-p tcp*), направленные на порт 80 (*--dport 80*), будут приниматься (*-j ACCEPT*) один раз в секунду (*--limit 1/s*).

Критерий *limit* применяют при защите от syn flood атак или подобных им. Так же его рекомендуют применять при отсылке пакетов в систему Syslog.

ЯВНЫЕ КРИТЕРИИ. КРИТЕРИЙ MULTIPORT.

Критерий *multiport* позволяет в качестве параметра указывать несколько портов в любой последовательности.

Его нельзя использовать совместно с критериями *--sport* и *--dport*.

Критерий	**Описание**
--source-port **или** **--sports**	Критерий служит для указания списка исходящих портов. С его помощью можно указать до 15 различных портов. Названия портов в списке должны отделяться друг от друга запятыми, пробелы не допустимы. Данное расширение может использоваться только совместно с критериями *-p tcp* или *-p udp*. Пример: `--source-port 21,23,53`
--destination-port **или** **--dports**	Критерий служит для указания списка входных портов. В остальном этот критерий аналогичен критерию *--source-port*.

Например, нам потребуется пропускать пакеты протокола *TCP* на цепочке *INPUT* на порты 22, 25 и 53. Правило может выглядеть следующим образом:

```
iptables -A INPUT -p tcp –m multiport --dports 22,25,53 -j ACCEPT
```

ЯВНЫЕ КРИТЕРИИ. КРИТЕРИЙ STATE.

Netfilter позволяет отслеживать соединения, причем даже для таких протоколов, как ICMP и UDP. В пределах netfilter соединение может иметь одно из 4-х базовых состояний: *NEW*, *ESTABLISHED*, *RELATED* и *INVALID*. Для управления прохождением пакетов, основываясь на их состоянии, используется критерий *state*.

Трассировка соединений производится специальным кодом в пространстве ядра — трассировщиком (conntrack). В большинстве случаев требуется более специфичная информация о соединении, чем та, которую поставляет трассировщик по умолчанию. Поэтому трассировщик включает в себя обработчики различных протоколов, например TCP, UDP или ICMP. Собранная ими информация затем используется для идентификации и определения текущего состояния соединения. Например — соединение по протоколу UDP однозначно идентифицируется по IP-адресам и портам источника и приемника.

В предыдущих версиях ядра имелась возможность включения/выключения поддержки дефрагментации пакетов. Однако после того как трассировка соединений была включена в состав netfilter, надобность в этом отпала. Причина в том, что трассировщик не в состоянии выполнять возложенные на него функции без поддержки дефрагментации и поэтому она включена постоянно. Её нельзя отключить иначе, как отключив трассировку соединений. Дефрагментация выполняется всегда, если трассировщик включен.

Трассировка соединений производится в цепочке *PREROUTING*, исключая случаи, когда пакеты создаются локальными процессами на брандмауэре. В этом случае трассировка производится в цепочке *OUTPUT*. Это означает, что netfilter производит все вычисления, связанные с определением состояния, в пределах этих цепочек. Когда локальный процесс на брандмауэре отправляет первый пакет из потока, то в цепочке *OUTPUT* ему присваивается состояние *NEW*, а когда возвращается пакет ответа, то состояние соединения в цепочке *PREROUTING* изменяется на *ESTABLISHED*, и так далее. Если же соединение устанавливается извне, то состояние *NEW* присваивается первому пакету из потока в цепочке *PREROUTING*. Таким образом, определение состояния пакетов производится в пределах цепочек *PREROUTING* и *OUTPUT* таблицы nat.

Критерий	Описание
--limit	Критерий определяет состояние: *NEW*, *EASTABLISHED*, *RELATED* и *INVALID*. Пример: `--state ESTABLISHED,RELATED`

СОСТОЯНИЯ.

Состояние	Описание
NEW	Признак *NEW* сообщает о том, что пакет является первым для данного соединения. Это означает, что это первый пакет в данном соединении, который увидел модуль трассировщика. Например, если получен *SYN* пакет, являющийся первым пакетом для данного соединения, то он получит статус *NEW*. Однако пакет может и не быть *SYN* пакетом, и тем не менее получить статус *NEW*.
ESTABLISHED	Признак *ESTABLISHED* говорит о том, что пакет принадлежит уже установленному соединению.
RELATED	Соединение получает статус *RELATED*, если оно связано с другим соединением, имеющим признак *ESTABLISHED*. Это означает, что соединение получает признак *RELATED* тогда, когда оно инициировано из уже установленного соединения, имеющего признак *ESTABLISHED*. Хорошим примером соединения, которое может рассматриваться как *RELATED*, является соединение *FTP-data*, которое является связанным с портом *FTP control*, а так же *DCC* соединение, запущенное из *IRC*. Обратите внимание на то, что большинство протоколов TCP и некоторые из протоколов UDP весьма сложны и передают информацию о соединении через область данных TCP или UDP пакетов и поэтому требуют наличия

	специальных вспомогательных модулей для корректной работы.
INVALID	Признак *INVALID* говорит о том, что пакет не может быть идентифицирован и поэтому не может иметь определенного статуса. Это может происходить по разным причинам, например при нехватке памяти или при получении ICMP-сообщения об ошибке, которое не соответствует какому-либо известному соединению.

Критерий state позволяет сократить количество правил фильтрации, необходимых для работы.

Обычно самым первым в цепочках *INPUT* и *FORWARD* определяют правила, запрещающие приём пакетов с состоянием *INVALID*:

```
iptables -A INPUT -m state --state INVALID -j DROP
iptables -A FORWARD -m state --state INVALID -j DROP
```

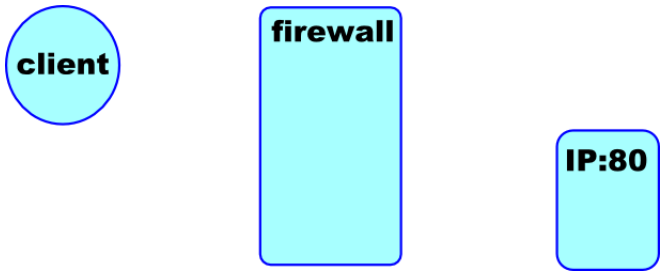
Затем определяют правила, принимающие пакеты уже установленных соединений:

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

После указания этих правил, достаточно разрешить выход в Internet на необходимые порты или машины, например:

```
iptables -A FORWARD -p tcp --dport 80 -j ACCEPT
```

Как это будет работать. Предположим, что нам необходимо из внутренней сети пропускать пакеты в интернет на 80 порт. Для этого нам придется написать как минимум три правила в цепочке INPUT.



Самыми первыми в цепочки будут правила с использованием модуля *state*. Эти правила пишутся только один раз в начале цепочки.

```
iptables -A FORWARD -m state --state INVALID -j DROP
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

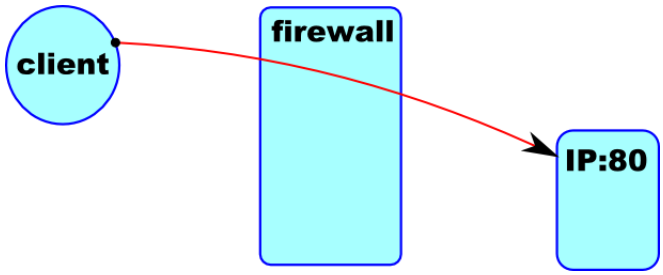
А затем правило, явным образом разрешающее хождение из внутренней сети наружу.

```
iptables -A FORWARD -p tcp --dopt 80 -o eth1 -j ACCEPT
```

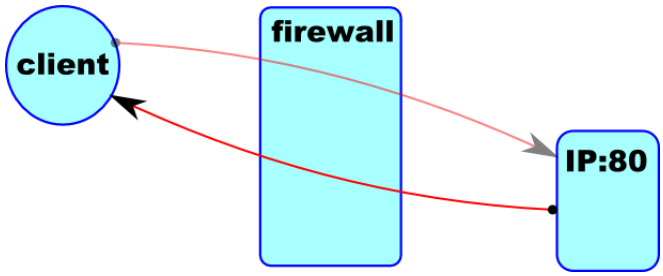
Обратите внимание на то, что исходящий интерфейс был указан явным образом (*-o eth1*). Если этого не сделать, то это правило сможет пропускать пакеты и наружной сети во внутреннюю, а нам этого не надо.

Теперь рассмотрим, как будет работать наш firewall.

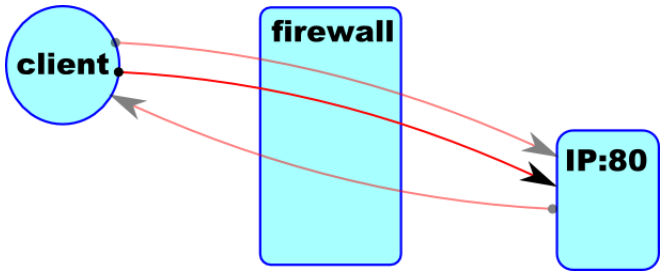
Клиентская машина из внутренней сети посылает первый пакет (запрос на соединение) на WEB браузер. Это правильный пакет, соответствующий всем стандартам, поэтому первое правило (*--state INVALID*) не срабатывает. Информации об этом соединении в таблице трассировки нет, поэтому второе правило (*--state ESTABLISHED,RELATED*) тоже не работает. Пакет доходит до третьего правила. Это протокол TCP? Да. На 80 порт? Да. Выходной интерфейс eth1? Да. Пакет пропускается и в таблицу трассировки заносится соответствующая информация.



Пакет дошел до WEB сервера. Сервер посылает ответный пакет. Это пакет правильный? Да, поэтому первое правило не работает. Информация об этом соединении (*IP:порт -> IP:порт*) есть в таблице трассировки? Да, есть. Срабатывает второе правило по состоянию *ESTABLISHED* (пакет принадлежит уже установленному соединению) и пакет пропускается.



Клиент посылает пакет обратно на WEB сервер. Это пакет правильный? Да, поэтому первое правило не работает. Информация об этом соединении (*IP:порт -> IP:порт*) есть в таблице трассировки? Да, есть. Срабатывает второе правило по состоянию *ESTABLISHED* (пакет принадлежит уже установленному соединению) и пакет пропускается.



В дальнейшем все пакеты этого соединения будут пропускаться по второму правилу (*--state ESTABLISHED,RELATED*).

Таким образом, для конфигурации firewall вам потребуется дописывать правила, пропускающие пакет только в одну сторону. Это правило сработает один раз, все остальные пакеты этого соединения будут пропускаться вторым правилом по состоянию *ESTABLISHED*.

Например, для разрешения подключения по ssh на вашу машину, достаточно будет добавить правило разрешающее такое подключение. Это можно сделать так:

```
iptables -A INPUT -p tcp --dopt 22 -j ACCEPT
```

Все остальные пакеты этого соединения будут проходить по правилу, которое вы напишите в начале цепочки *INPUT*:

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Обратите внимание на скрипт, который мы по быстрому написали в самом начале. Присмотритесь внимательно к этим строкам:

```
# STATE RULES ======
$IPT -A INPUT -m state --state INVALID -j DROP
```

```
$IPT -A FORWARD -m state --state INVALID -j DROP
$IPT -A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
$IPT -A FORWARD -m state –state ESTABLISHED,RELATED -j ACCEPT
```

Вместо $IPT, подставляется программа iptables. Что мы сделали в первую очередь? Мы добавили по два правила в цепочки *INPUT* и *FORWARD*, работающие с трассировкой пакетов. В дальнейшем, в этом файле можно просто разрешать хождение пакетов только в одну сторону.

ЯВНЫЕ КРИТЕРИИ. КРИТЕРИЙ RECENT.

Как я уже показал в предыдущем разделе, для разрешения подключения по ssh можно воспользоваться правилом:

```
iptables -A INPUT -p tcp --dopt 22 -j ACCEPT
```

Это правило будет работать, вы сможете подключаться по ssh. Но! Это правило позволяет братьям китайцам свободно подбирать пароли пользователей по ssh. Ради интереса попробуйте использовать именно это правило и посмотрите файлы журнальной регистрации. Вы удивитесь, как много китайцев есть на свете. В среднем, в день у вас будет тратиться около 4-5 Мбайт трафика только на попытки подбора пароля!

Я бы рекомендовал ограничить количество подключений на 22 порт. Ведь с той стороны работают программы (роботы). Программы, с задержкой в 2-3 секунды, открывают новое соединение и пытаются подобрать пароль. Достаточно разрешить одно соединение в течении 10-20 секунд и робот отвалится сам.

Для этих целей можно использовать модуль *limit*, который ограничивает количество срабатываний правила. Но, сожалению, он не учитывает IP откуда пришел пакет. Поэтому вы встанете в общую очередь с китайцами на подключение к ssh и достаточно долго не сможете подключится к своему же серверу.

Для ограничения доступа с учетом IP адресов можно использовать модуль *recent*. Модуль позволяет создавать динамический список IP адресов и если IP присутствует в списке, делать какие либо дополнительные действия с пакетами.

Основные критерии модуля *recent*:

Критерий	Описание
--name	Присваивает или определяет имя списку IP адресов. Можно делать несколько разных списков и использовать их в различных правилах и ситуациях. `--name ssh`
--set	Добавляет IP источника в список.
--rcheck	Проверяет, есть ли указанный IP адрес в списке. Если есть — возвращает истину. Если нет — ложь.
--update	Тоже самое что и *--rcheck*, но попутно обновляет информацию в списке.
--remove	Если IP адрес есть в списке, удаляет его оттуда.
--seconds	Если IP адрес находится в списке, и еще не прошло указанное количество секунд, возвращает истину. *Этот параметр следует использовать совместно с --rcheck и --update*. `--seconds 20`

Например, для ограничения количества соединений на 22 порт, мы можем написать следующие два правила:

```
iptables -A INPUT -p tcp -m state --state NEW --dport 22 \
-m recent --name ssh --update --seconds 20 -j DROP
```

```
iptables -A INPUT -p tcp -m state --state NEW --dport 22 \
-m recent --name ssh --set -j ACCEPT
```

Первое правило проверяет, есть ли в списке с именем ssh (*--name ssh --update*) IP адрес с которого пришел к нам пакет, и не прошло ли с момента попадания его в список 20 секунд (*--seconds 20*). Если IP есть в списке и не прошло 20 секунд, тогда пакет не пропускается (*-j DROP*). Если нет в списке или уже прошло 20 секунд, правило не срабатывает, смотрим следующее правило.

Второе правило заносит IP адрес источника в список с именем ssh (*--name ssh --set*) и пакет пропускается (*-j ACCEPT*).

Обратите внимание на то, что эти правила обрабатывают только первые пакеты в соединении (*--state NEW*). И на остальные пакеты не распространяются. Да и не смогли бы, потому что все остальные пакеты проходят по *--state ESTABLISHED*, описанном в первых правилах в цепочке *INPUT*.

ДЕЙСТВИЯ И ПЕРЕХОДЫ.

После определения критериев отбора пакетов, обычно указывается, что сделать с отобранным пакетом. Для этого используется параметр *-j* и соответствующее действие. Так же, в качестве аргумента параметру можно указать имя цепочки пользователя.

Действие — это предопределенная команда, описывающая действие, которое необходимо выполнить, если пакет совпал с заданными критериями отбора. Например, можно применить действие *DROP* или *ACCEPT*. Существуют и другие типы действий, некоторые из них будут рассмотрены ниже.

ДЕЙСТВИЕ ACCEPT.

Если выполняется действие *ACCEPT*, то пакет прекращает движение по цепочке и всем вызвавшим её цепочкам (если текущая цепочка была вложенной) и считается принятым.

Но, очень важно понимать, что пакет пропускается *только* в текущей цепочке. И если он попадает в другую цепочку, то в ней его могут и не пропустить. Т.е. *ACCEPT* не даёт 100%-й гарантии на прохождения пакета через ваш firewall.

ДЕЙСТВИЕ DROP.

Действие просто сбрасывает пакет и netfilter забывает о его существовании. Сброшенные пакеты прекращают свое движение полностью, т.е. они не передаются в другие таблицы, как это происходит в случае с действием *ACCEPT*.

Следует помнить, что данное действие может иметь негативные последствия, поскольку может оставлять незакрытые *мертвые* соединения как на стороне сервера, так и на стороне клиента. Поэтому, в случае если вы изменяете правила firewall динамически (на лету), использовать *DROP* не рекомендуется. Используйте действия *REJECT*.

ДЕЙСТВИЕ REJECT.

REJECT используется, как правило, в тех же самых ситуациях, что и *DROP*, но в отличие от *DROP*, возвращает icmp сообщение об ошибке на хост, передавший пакет.

Действие *REJECT* может использоваться только в цепочках *INPUT*, *FORWARD* и *OUTPUT* (и во вложенных в них цепочках).

Параметр	Описание
--reject-with	Указывает, какое сообщение необходимо передать в ответ, если пакет совпал с заданным критерием. При применении действия *REJECT* к пакету, сначала на хост отправителя будет отослан указанный ответ, а затем пакет будет сброшен. Допускается использовать следующие типы ответов: *icmp-net-unreachable, icmp-host-unreachable, icmp-port-unreachable, icmp-proto-unreachable,*

	icmp-net-prohibited и icmp-host-prohibited. По умолчанию передается сообщение *port-unreachable*. Ещё один тип ответа *--tcp-reset*, который используется только для протокола TCP. Если указано значение *--tcp-reset*, то действие *REJECT* передаст в ответ пакет *TCP RST*. Пакеты *TCP RST* используются для закрытия TCP соединений. Пример: `-j REJECT --reject-with icmp-host-unreachable`

ДЕЙСТВИЕ RETURN.

Действие *RETURN* прекращает движение пакета по текущей цепочке правил и производит возврат в вызывающую цепочку, если текущая цепочка была вложенной, или, если текущая цепочка лежит на самом верхнем уровне (например, *INPUT*), то к пакету будет применена политика по умолчанию.

ДЕЙСТВИЕ LOG.

Действие *LOG* передаёт копию заголовка пакета в систему журнальной регистрации.

Если необходимо передавать копию содержимого пакета — пользуйтесь действием *ULOG*.

При использовании действия LOG настоятельно рекомендуется включать критерий limit для ограничения информации, попадающей в систему журнальной регистрации.

Параметр	Описание
--log-level	Используется для задания уровня важности передаваемого сообщения. Все сообщения передаются средством *kernel*. Пример: `-j LOG --log-level debug`
--log-prefix	Параметр задает текст, с которого будет начинаться сообщение, передаваемое в систему журнальной регистрации. Сообщения со специфичным префиксом затем легко можно найти, к примеру, с помощью *grep*. Префикс может содержать до 29 символов, включая и пробелы. Пример: `-j LOG --log-prefix "Alert: "`

Например, нам потребуется посылать в систему журнальной регистрации копию заголовков пакетов, при попытке сканирования портов на нашей машине. Причем, если сканер портов присылает новый TCP, но это не запрос на соединение. При этом нас интересует сам факт такой попытки, а не количество попыток.

```
iptables -A INPUT -p tcp --tcp-flags ALL NONE -m limit \
--limit 1/m --limit-burst 1 -j LOG \
--log-prefix "Scan: " --log-level info
```

Рассмотрим, приведенное выше правило. Нас интересуют пакеты TCP, в заголовке которых все флаги равны нулю (*-p tcp --tcp-flags ALL NONE*).

Поскольку мы заранее не знаем, как часто такие пакеты будут приходить на нашу машину, ограничимся одним пакетом в минуту. Для такого ограничения используем модуль *limit* (*-m limit --limit 1/m --limit-burst 1*).

Скажем, что строка должна начинаться со *Scan* (*--log-prefix "Scan: "*). И посылаться на уровне важности *info* (*--log-level info*).

Очень важно запомнить, что если срабатывает LOG, пакет все равно продолжает свое продвижение по цепочке!

NAT ПРЕОБРАЗОВАНИЯ.

NAT преобразования позволяют заменить в заголовках пакетов IP адрес источника или назначения, а также порт источника или назначения. В netfilter существуют два основных действия, позволяющие осуществить NAT преобразования:

SNAT — замена адреса и/или порта источника.
DNAT — замена адреса и/или порта назначения.

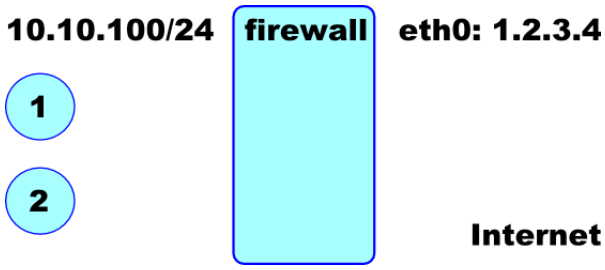
Кроме того, у *SNAT* и *DNAT* существуют частные случаи:

MASQUERADE — частный случай *SNAT*.
REDIRECT — частный случай *DNAT*.

Не смотря на то, что firewall и NAT находятся в netfilter — это разные подсистемы, предназначенные для разных целей. Для firewall используется таблица *filter*, для NAT преобразований — таблица *nat*. Не надо писать правила firewall в цепочках таблицы *nat*! Это не правильно!

SNAT.

Предположим, что существует сеть 10.10.100.0/24 и необходимо сделать так, чтобы компьютеры, находящиеся в этой сети, могли получить доступ в Интернет.



Недостаточно просто разрешить хождение пакетов через роутер. Все пакеты, отправляемые с машин клиентов, в поле IP источника будут содержать IP адрес внутренней сети. Первый же роутер вашего провайдера такие пакеты будет отбрасывать. Даже если предположить, что пакет дойдёт, например, до WEB сервера, сервер не будет знать, как ответить клиенту. Ведь на роутерах в Интернет не прописаны маршруты к частным сетям.

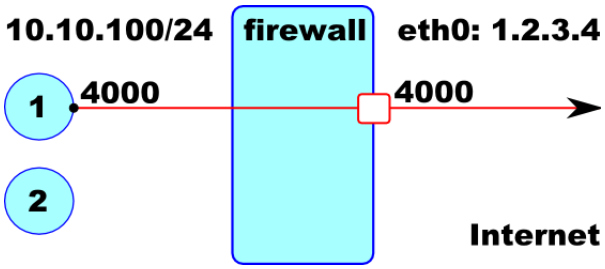
Наша задача при выходе пакета из нашего роутера, заменить IP адрес частной сети на реальный IP внешнего интерфейса роутера. И обеспечить механизм обратного преобразования, когда пакеты будут возвращаться на роутер из Интернет для передачи их машинам во внутреннюю сеть.

Если выходной интерфейс имеет статический IP адрес, можно применять действие *SNAT*, которое позволяет осуществлять замену IP адреса и/или порта источника. Предположим, что роутер подключён к Интернет через интерфейс *eth0* с IP адресом 1.2.3.4, а все пакеты посылаются из внутренней сети с машины с адресом 10.10.100.1. Тогда для замены IP адреса источника воспользуемся следующей командой:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 1.2.3.4
```

Все действия с NAT преобразованиями следует производить только в таблице *nat* (*-t nat*). *SNAT* преобразования можно делать только в цепочке *POSTROUTING* (*-A POSTROUTING*). Действия по замене будем производить только над теми пакетами, которые должны выходить в Интернет, т.е. пакеты, выходящие через интерфейс *eth0* (*-o eth0*). IP адрес источника будем менять на IP адрес внешнего интерфейса — 1.2.3.4 *(--to-source 1.2.3.4*).

Для иллюстрации того, что будет происходить с пакетами, можно представить следующую картину:

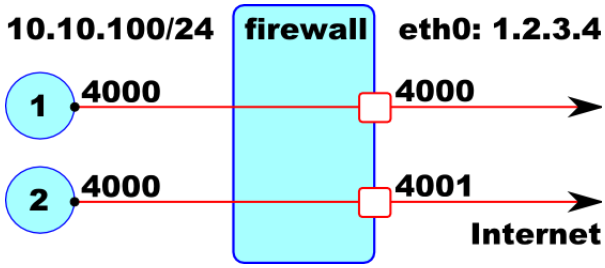


С машины с IP адресом 10.10.100.1 открывается новое соединение с порта 4000. После NAT преобразования, IP источника заменяется на IP внешнего интерфейса *eth0*. Всем машинам в Интернет будет казаться, что пакет пришел с машины с IP адресом 1.2.3.4 с порта 4000. И все пакеты будут возвращаться на IP 1.2.3.4. Обратное преобразование происходит автоматически, и вам не потребуется явно преобразовывать IP адреса.

Это происходит потому, что netfiler отслеживает соединения. В оперативной памяти образуется таблица, в которой заносятся IP адреса и номера портов до и после NAT. Согласно записей в этой таблице происходят обратные преобразования.

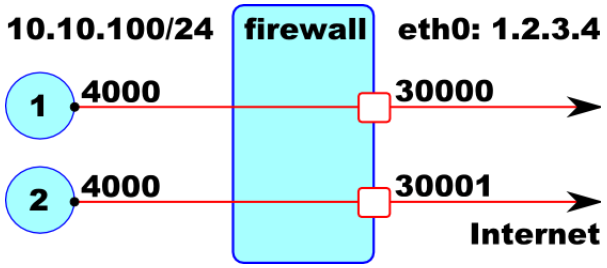
До SNAT		**После SNAT**	
IPs	Port source	IPs	Port source
10.10.100.1	4000	1.2.3.4	4000

Теперь предположим, что с другой машины в сети открывается ещё одно соединение в Интернет и тоже с порта 4000. А первое соединение в это время продолжает работать.



Получается неприятная ситуация. Все пакеты, которые будут выходить с роутера в Интернет, будут видны как пакеты соединения открываемого роутером с 4000 порта. И получается, что два соединения открываются с одного и того же порта. Такого быть не должно. Поэтому при *SNAT* необходимо менять не только IP адрес источника, но и порт. Вообще то, netfilter для нового соединения будет автоматически подставлять следующий свободный порт, например 4001. Но ему можно явным образом указать диапазон портов, которые он может использовать. Например, следующим образом:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 1.2.3.4:30000-35000
```



Теперь при открытии соединения будет меняться и порт источника. Причём будет выбираться следующий свободный порт из указанного диапазона.

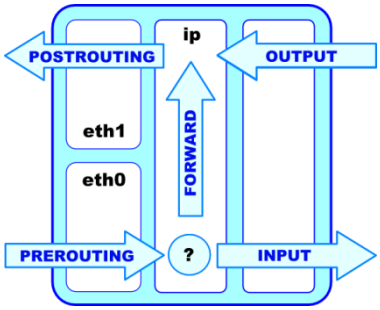
И таблица состояний будет выглядеть по другому.

До SNAT		После SNAT	
IPs	Port source	IPs	Port source
10.10.100.1	4000	1.2.3.4	30000
10.10.100.2	4000	1.2.3.4	30001

Поскольку netfilter отслеживает соединения, во всех пакетах приходящих обратно, IP адрес назначения автоматически меняется на IP адрес внутренней сети, и пакет попадает по назначению.

Хочу напомнить вам, что NAT — это NAT, а firewall — это firewall. Это различные действия с пакетом. И то, что эти две *разных* подсистемы находятся в одном netfiltre, ещё не означает, что достаточно сделать только NAT преобразование и всё будет хорошо.

Еще раз покажу рисунок, на котором показаны цепочки.



Смотрите, прежде чем попасть в цепочку *POSTROUTING* таблицы *nat*, пакет сначала проходит по цепочке *FORWARD*, таблицы *filter*. И именно в *FORWARD* необходимо принимать решение о том, пропускать или не пропускать пакет. NAT преобразование будет производиться только с теми пакетами, которые были пропущены в цепочке *FORWARD*.

Что я этим хотел сказать?

Никогда не принимайте решение о прохождении пакета в таблице *nat*! Некоторые *чудо* администраторы очень часто допускают серьёзную ошибку.

Предположим, что во внутренней сети есть две машины, с IP адресами: 192.168.1.1 и 192.168.1.2. Первую машину надо пропускать в Интернет, второй надо запретить хождение в Интернет. *Чудо* администратор напишет правило:

```
iptables -t nat -A POSTROUTING -o eth1 -s 192.168.1.1 -j SNAT --to-source 1.2.3.4
```

и будет считать, что все хорошо. Да, ему будет казаться, что все получилось, потому, что клиент на 192.168.1.1 будет получать доступ в Интернет, а 192.168.1.2 нет.

В чём ошибка? Дело в том, что клиент 192.168.1.2 действительно не сможет подключиться ни к одному серверу в Интернет, потому что у его пакетов не будет заменяться IP источника и сервера не смогут возвращать пакеты обратно. Но! *Его пакеты все равно будут уходить с вашей машины и провайдер будет считать вам трафик*! *Чудо* администратор — это чистая прибыль провайдеров.

Не надо в таблице *nat* писать правила firewall! В *nat* следует делать только NAT преобразования. А вот разрешать или запрещать доступ в Интернет вы будете в таблице *filter* цепочка *FORWARD*. Она как раз для этого и предназначена. Все пропущенные в этой цепочке пакеты будут подвергаться преобразованию. Поэтому правильно решение предыдущего примера будет такое.

В *nat* пишем общее правило:

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source 1.2.3.4
```

А в *filter*, разрешающее прохождение пакетов правило:

```
iptables -A FORWARD -s 192.168.1.1 -j ACCEPT
```

MASQUERADE.

Действие *MASQUERADE* является частным случаем *SNAT* преобразования. Его рекомендуют применять в тех случаях, когда IP адрес получается динамически: при ppp соединении или от DHCP сервера.

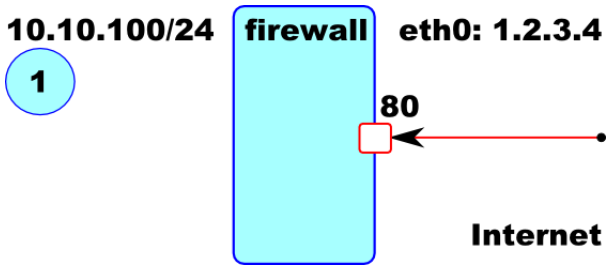
Маскарадинг подразумевает получение IP адреса от заданного сетевого интерфейса вместо прямого его указания, как это делается с помощью ключа *--to-source* в действии *SNAT*. Действие *MASQUERADE* имеет хорошее свойство — «забывать» соединения при остановке сетевого интерфейса. В случае же SNAT, в этой ситуации, в таблице трассировщика остаются данные о потерянных соединениях, и эти данные могут сохраняться до суток, поглощая ценную память. Эффект *забывчивости* связан с тем, что при остановке сетевого интерфейса с динамическим IP адресом есть вероятность на следующем запуске получить другой IP адрес, но в этом случае любые соединения все равно будут потеряны, и было бы глупо хранить трассировочную информацию.

Невзирая на положительные черты, маскарадинг не следует считать предпочтительным, поскольку он дает большую нагрузку на систему.

Для примера, описанного в предыдущей главе, вместо SNAT можно использовать следующую команду:

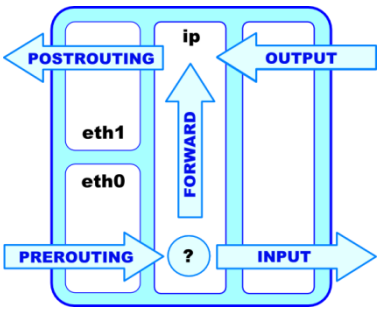
```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

DNAT.



Предположим, что в DMZ (10.10.100.0/24) находится WEB сервер (10.10.100.1) и необходимо организовать доступ к этому серверу из Интернет. Внешний интерфейс роутера (*eth0*) имеет IP адрес 1.2.3.4. В DNS, машине *www* соответствует IP 1.2.3.4, т.е. все пакеты, предназначенные нашему WEB серверу, будут приходить на этот IP на порт 80. Нам необходимо организовать пересылку этих пакетов на машину 10.10.100.1 на порт 80.

Если ничего не делать, то пакет в точке принятия решения о маршрутизации будет передан на транспортный уровень нашей машины. Но на нашей машине нет WEB сервера и будет отклонен со стандартным сообщением об ошибке. Ещё раз посмотрите на рисунок.



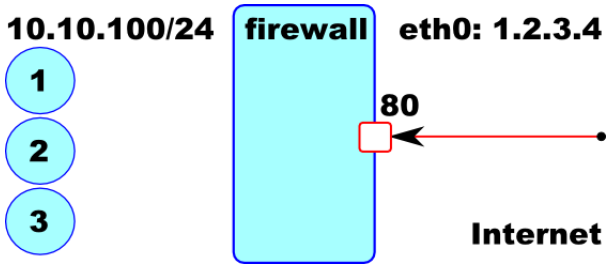
До точки принятия решения о маршрутизации у нас есть цепочка *PREROUTING* таблицы *nat*. И именно в этой цепочке нам следует заменить IP назначения на 10.10.100.1. В таблице маршрутизации есть маршрут к сети 10.10.100.0/24 и пакет будет направлен во внутреннюю сеть на наш WEB сервер.

Для решения этой задачи будем пользоваться *DNAT*, при помощи которого у проходящих пакетов меняются IP адрес и/или порт назначения.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT \
--to-destination 10.10.100.1
```

Теперь у всех пакетов TCP, направленных на порт 80, IP адреса назначения будут меняться на IP 10.10.100.1. Так как netfilter отслеживает соединения, в памяти организуется таблица, похожая на таблицу, которую мы рассматривали в случае *SNAT*. У пакетов идущих обратно, IP адрес источника меняется автоматически.

Предположим, что у нас есть несколько WEB серверов, обслуживающих один и тот же сайт. По какой то причине мы купили только один реальный IP адрес — 1.2.3.4. И нам нужно распределить соединения между серверами.



При помощи *DNAT* можно организовать равномерное распределение соединений между несколькими серверами, обслуживающими одну и ту же службу. Для этого надо написать диапазон IP адресов.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT \
--to-destination 10.10.100.1-10.10.100.3
```

Теперь новое соединение будет перенаправлено на сервер, у которого в данный момент времени наименьшее количество открытых соединений. Правда у этого метода есть один существенный недостаток — если один из серверов перестанет работать, netfilter это не увидит и будет пытаться новые соединения перенаправлять на не работающий сервер.

Но мало сделать *DNAT*, необходимо еще пропустить пакет в цепочке *FORWAR* таблицы *filter*. Для первого случая, например вот так:

```
iptables -A FORWARD -p tcp --dport 80 -d 10.10.100.1 -j ACCEPT
```

Для второго случая, где у нас используется диапазон IP адресов, придётся писать три правила в цепочке *FORWARD*. Для каждого IP отдельно.

REDIRECT.

При помощи *REDIRECT* мы можем заменить порт назначения пакета.

Параметр	Описание

--to-ports	Определяет порт или диапазон портов назначения.
	Пример:
	`--to-ports 8008`
	`--to-ports 8001-8009`

Действие *REDIRECT* можно использовать только в цепочках *PREROUTING* и *OUTPUT* таблицы *nat*.

Приведу пример из жизни. Мой домашний провайдер, по какой то причине, не пропускает пакеты из внутренней сети, отправленные на 25 порт. Это конечно не правильно и он должен пропускать все, что идет с машин клиентов. Но факт остается фактом. И я не могу из дома отправлять почту через любые почтовые сервера. Но поскольку у меня есть свой собственный сервер, я поступил очень просто. На нем прописал простейшее правило *REDIRECT*, которое перенаправляет все входящие пакеты с порта 2525 на порт 25.

```
iptables -t nat -A PREROUTING -p tcp --dport 2525 -j REDIRECT --to-ports 25
```

После этого в почтовом клиенте указал порт 2525 и теперь я могу отправлять почту из нашей сети. Но это, к сожалению, не поможет, если мне придется отправлять почту через любой другой сервер. Поэтому лучшим решением данной проблемы является смена провайдера.

УПРАВЛЕНИЕ NETFILTER.

После создания правил фильтрации необходимо сделать так, чтобы эти правила загружались автоматически после старта компьютера. С пакетом *iptables* поставляются две программы, при помощи которых можно сохранять правила фильтрации в специальном файле, а затем восстанавливать их из этого файла.

```
iptables-save [-c] > file
```

Программа выводит текущие правила на стандартный вывод. Если указать опцию *-c*, с каждым правилом будут показаны значения счётчиков.

```
iptables-restore [-c] file
```

Программа устанавливает текущие правила фильтрации из указанного файла. Опция *-c* восстанавливает значения счётчиков.

В CentOS информация о правилах фильтрации храниться в файле */etc/sysconfig/iptables*. Для сохранения информации можно воспользоваться стандартным стартовым скриптом:

```
service iptables save
```

СТАРТОВЫЕ СКРИПТЫ.

Для автоматического включения правил фильтрации при старте компьютера и для его автоматического выключения, при остановке компьютера, необходимо написать или использовать готовые стартовые скрипты.

В случае RedHat (CentOS) Linux, стартовый скрипт уже существует. Поскольку в системе инициализации SystemV нельзя вносить изменения в стартовые скрипты, вам придется дописать небольшую программу, для первоначальной инициализации firewall. В SuSE Linux такой скрипт тоже существует (*susefirewall2*), но я бы рекомендовал его удалить и написать свой собственный вариант. В Slackware Linux скрипт не поставляется и его приходится писать самому.

Скрипт следует писать с учетом особенностей системы инициализации System V. То есть, он должен поддерживать как минимум две функции:

- *start* — запуск firewall.
- *stop* — останов firewall.

Кроме перечисленных функций рекомендуется использовать еще две:

- *reset* — сбрасывает firewall в какое либо первоначальное состояние. Функция применяется, когда вы редактируете правила firewall на удаленной машине.
- *init* — используется для первоначальной инициализации firewall.

Прежде чем начинать писать скрипт, вы должны четко представлять себе какие задачи должен решать firewall. Лучше всего нарисовать графически план прохождения пакетов. И только после этого приступать к написанию скрипта.

Напишем стартовый скрипт для Slackware Linux. Несмотря на то, что в этом дистрибутиве используется система инициализации BSD, скрипт будет написан с учетом системы инициализации System V. Таким образом с минимальными переделками он может быть использован в дистрибутиве SuSE и RedHat Linux. В RedHat Linux следует оставить только две функции: *reset* и *init*.

Сначала укажем интерпретатор и для удобства определим несколько переменных.

```
#!/bin/bash
IPT=/usr/sbin/iptables
IPTR=/usr/sbin/iptables-restore
IPTS=/usr/sbin/iptables-save
```

Функция *start* должна организовать первоначальную инициализацию firewall. Для этого будет использована программа *iptables-restore* и файл */etc/iptables*, который будет создаваться автоматически.

```
start()
{
echo -n "Starting firewall... "
$IPTR -c /etc/iptables
echo "Done"
}
```

Функция *stop* должна записывать текущие правила фильтрации в файл */etc/iptables*.

```
stop()
{
echo -n "Stop firewall... "
$IPTS -c > /etc/iptables
echo "Done"
}
```

Функция *reset* должна сбрасывать правила в первоначальное состояние. В этом состоянии должен быть запрещен доступ ко всем службам, кроме удаленного соединения по ssh.

Как уже говорилось выше, эту функцию необходимо использовать в случае удаленного управления компьютером, при изменении в правилах фильтрации. Запуск скрипта с параметром *reset* (параметр будет описан ниже) необходимо производить раз в 10-15 минут, при помощи CRON. Это необходимо для того, что бы в случае ошибки, правила фильтрации вернулись в первоначальное состояние и вы получили доступ к удаленному компьютеру. После изменения правил, не забудьте убрать из CRON вызов скрипта.

```
reset()
{
$IPT -F
$IPT -t nat -F
$IPT -t mangle -F
$IPT -P INPUT DROP
$IPT -P FORWARD DROP

# STATE RULES ======
$IPT -A INPUT -m state --state INVALID -j DROP
$IPT -A FORWARD -m state --state INVALID -j DROP
$IPT -A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
$IPT -A FORWARD -m state –state ESTABLISHED,RELATED -j ACCEPT

# LOCALHOST ===========
$IPT -A FORWARD -i ! lo -s 127.0.0.1 -j DROP
$IPT -A INPUT -i ! lo -s 127.0.0.1 -j DROP
$IPT -A FORWARD -i ! lo -d 127.0.0.1 -j DROP
$IPT -A INPUT -i ! lo -d 127.0.0.1 -j DROP
$IPT -A INPUT -d 127.0.0.1 -j ACCEPT
$IPT -A INPUT -s 127.0.0.1 -j ACCEPT

# ICMP ENABLED =====
```

```
$IPT -A INPUT -p icmp -j ACCEPT


# SSH enable =======
$IPT -A INPUT -p tcp -m state --state NEW --dport 22 \
-m recent --update --seconds 20 -j DROP
$IPT -A INPUT -p tcp -m state --state NEW --dport 22 \
-m recent --set -j ACCEPT
}
```

В функции сбрасываются все правила, устанавливаются политики по умолчанию. Обычно я по умолчанию запрещаю хождение пакетов только в цепочках *INPUT* и *FORWARD*. Можно еще закрывать *OUTPUT*, но это вопрос спорный и его обсуждение выходит за рамки этого руководства. Все остальные цепочки, во всех таблицах должны по умолчанию пропускать пакеты.

Затем устанавливаются правила, связанные с уже знакомым вам модулем *state*.

Разрешается хождение пакетов через интерфейс *lo*. Это делать обязательно, потому, что этот интерфейс необходим для корректной работы достаточно большого количества приложений.

Разрешается трафик icmp но только в цепочке *INPUT*. Не закрывайте пинги, не надо. Вы сами усложните себе работу. А вот пропускать их во внутрь сети — это не правильно, но мы этого как раз этого и не делаем.

В заключении разрешается подключение к 22 порту (ssh). Но для разрешения используется модуль *recent*, который позволяет ограничить количество соединений с одного IP адреса. В нашем примере не более одного соединения в 20 секунд. Таким способом закрывается подбор паролей по ssh *роботами*.

В функции *init* вы должны написать все остальные правила. Например, вот такие:

```
init()
{
echo -n "Init firewall... "
reset
# DNS ENABLE ==========
$IPT -A INPUT -p tcp --dport 53 -j ACCEPT
$IPT -A INPUT -p udp --dport 53 -j ACCEPT
# Apache ==============
$IPT -A INPUT -p tcp --dport 80 -j ACCEPT
# SMTP ================
$IPT -A INPUT -p tcp --dport 25 -j ACCEPT


# SAVE rules in file ==
$IPTS -c > /etc/iptables
echo "Done"
}
```

Какие правила использовать в функции *init* зависит от задач, решаемых вашим сервером. В приведенном примере был открыт до ступ к серверам: DNS, WEB, SMTP. В конце функции происходит запись текущих правил в файл */etc/iptables*.

В заключении необходимо написать код, который будет обрабатывать параметры командной строки и вызывать соответствующие функции.

```
case $1 in
start) start ;;
```

```
stop) stop ;;
init) init ;;
reset) reset ;;
*)
echo "Usage rc.fw start|stop|init|reset"
exit 88
esac
```

Для того, что бы скрипт вызывался при старте компьютера, в Slackware Linux необходимо изменить содержимое файла */etc/rc.d/rc.inet1*.

```
#! /bin/sh
start()
{
/etc/rc.d/rc.fw start
echo "ifconfig lo 127.0.0.1"
echo "ifconfig lo 127.0.0.1" | /usr/bin/logger
/sbin/ifconfig lo 127.0.0.1
/sbin/route add -net 127.0.0.0/8 lo
echo "ifconfig eth0 192.168.10.69"
echo "ifconfig eth0 192.168.10.69" | /usr/bin/logger
/sbin/ifconfig eth0 192.168.10.69
/sbin/route add default gw 192.168.10.1
}
stop()
{
echo "ifconfig eth0 down"
echo "ifconfig eth0 down" | /usr/bin/logger
/sbin/ifconfig eth0 down
echo "ifconfig lo down"
echo "ifconfig lo down" | /usr/bin/logger
/sbin/ifconfig lo down
/etc/rc.d/rc.fw stop
}
case $1 in
stop) stop;;
*) start;;
esac
```

Вы должны будете подставить свои параметры инициализации сетевых интерфейсов. Обратите внимание на то, что запуск firewall происходит до инициализации сетевых интерфейсов. А его останов, только после выключения интерфейсов.

```
/etc/rc.d/rc.fw start
/etc/rc.d/rc.fw stop
```

Изменение правил firewall происходит следующим образом. Сначала вы дописываете новое правило в функции *init*. Затем запускаете скрипт с параметром *init*. Правила записываются в оперативную память и сохраняются в файле iptables.

Ну а теперь посмотрите, что мы с вами сделали в самом начале этого занятия. Мы создали скрипт *rc.fw*, в котором написали две функции: *reset* и *init*. *Start* и *stop* в этом скрипте отсутствуют, потому, что они реализованы в стандартном скрипте системы инициализации */etc/rc.d/init.d/iptables*.

Теперь, если вы захотите изменять правила firewall, вам нужно будет дописывать их внутри функции *init*. Что мы и будем делать по мере добавления поддержки новых программ.

ПРИМЕРЫ.

Для закрепления материала рассмотрим несколько примеров.

ЕСЛИ ПАКЕТЫ, ПО КАКОЙ ЛИБО ПРИЧИНЕ НЕ ПРОХОДЯТ ЧЕРЕЗ LINUX РОУТЕР, ПРОВЕРЬТЕ, РАЗРЕШЕНА ЛИ ПЕРЕСЫЛКА ПАКЕТОВ МЕЖДУ СЕТЕВЫМИ ИНТЕРФЕЙСАМИ (IP FORWARDING).

Для этого выполните следующую команду:

```
cat /proc/sys/net/ipv4/ip_forward
```

Если на экране будет показан ноль, значит пересылка запрещена и ее надо включить. Включение производится следующим способом. Сначала отредактируйте файл /etc/sysctl.conf

```
net.ipv4.ip_forvard = 1
```

Запустите программу sysctl.

```
sysctl -p
```

ПРИМЕР 1.

Linux сервер имеет два сетевых интерфейса:

- *eth0* — подключен к Интернет и имеет реальный IP адрес 1.2.3.4.
- *eth1* — подключен ко внутренней сети 192.168.0.0/24 и имеет IP адрес 192.168.0.1.

На сервере работают: DNS сервер и почтовый сервер. Для доступа к почтовым ящикам используются протоколы *pop3s* (995 порт), *imap* (143 порт) и *imaps* (993 порт). Для отправки почты используется протокол *smtp* (25 порт) и *ssmtp* (465 порт).

Пользователей из внутренней сети необходимо пропускать в Интернет только для просмотра WEB сайтов по протоколам *http* (80 порт) и *https* (443 порт).

Функция init будет выглядеть следующим образом.

```
init()
{
echo -n "Init firewall... "
reset
####### INSERT you rules
# DNS server
$IPT -A INPUT -p tcp --dport 53 -j ACCEPT
$IPT -A INPUT -p tcp --dport 53 -j ACCEPT

# mail server
$IPT -A INPUT -p tcp -m multiport --dports \
            25,143,465,993,995 -j ACCEPT

# Web transite
$IPT -A FORWARD -p tcp -m multiport --dports 80,443 \
```

```
            -o eth0 -j ACCEPT

# NAT
$IPT -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 1.2.3.4

#######
# SAVE rules in file ==
$IPTS -c > /etc/sysconfig/iptables
echo "Done"
}
```

ПРИМЕР 2.

Linux сервер имеет два сетевых интерфейса:

- *eth0* — подключен к Интернет и имеет реальный IP адрес 1.2.3.4.
- *eth1* — подключен ко внутренней сети 192.168.0.0/24 и имеет IP адрес 192.168.0.1.

Внутри сети работают две Windows машины 192.168.0.10 и 192.168.0.11, с установленными на них программами Radmin (порт 4899).

Необходимо дать доступ из Интернет к программе radmin на обе машины.

Поскольку в Интернет «виден» только один IP адрес 1.2.3.4, клиенты должны подключаться к разным портам: 4898 и 4899. При подключении к порту 4898 необходимо пересылать пакеты на машину 192.168.0.10 порт 4899. При подключении к порту 4899 необходимо пересылать пакеты на машину 192.168.0.11.

Пример функции init.

```
init()
{
echo -n "Init firewall... "
reset
####### INSERT you rules
# RADMIN
$IPT -A FORWARD -p tcp --dport 4899 -d 192.168.0.10 -j ACCEPT
$IPT -A FORWARD -p tcp --dport 4899 -d 192.168.0.11 -j ACCEPT

# NAT
$IPT -t nat -A PREROUTING -i eth0 -d 1.2.3.4 -p tcp \
            --dport 4898 -j DNAT --to-destination 192.168.0.10:4899
$IPT -t nat -A PREROUTING -i eth0 -d 1.2.3.4 -p tcp \
            --dport 4899 -j DNAT --to-destination 192.168.0.11

#######
# SAVE rules in file ==
$IPTS -c > /etc/sysconfig/iptables
echo "Done"
}
```

НЕБОЛЬШОЕ ЗАДАНИЕ НА ЗАКРЕПЛЕНИЕ МАТЕРИАЛА.

В вашем распоряжении есть вторая виртуальная машина, которую мы собирались использовать в качестве клиента. Она имеет всего один сетевой интерфейс, подключенный ко второму интерфейсу виртуальной машины, в которой вы сейчас работаете.

Сделайте так, что бы машина клиент, могла бы выходить в Интернет через виртуальную машину сервер.