PF: Пакетный фильтр

Оглавление

PF: Начало

Активация:

Для активизации и чтения правил конфигурации pf при начальной загрузке, необходимо в OpenBSD в файле /etc/rc.conf прописать следующее:

```
pf=YES
```

Для вступления изменений в силу, необходимо перезагрузить систему.
Активизировать и остановить pf можно используя программу pfctl(8):

```
# pfctl-e # pfctl-d
```

для запуска и останова соответственно. Обратите внимание, что загрузка правил при этом не производится, их необходимо подгружать отдельно, до или после запуска pf.

В FreeBSD для активации PF во время загрузки в /etc/rc.conf должны быть включены следующие переменные:

```
pf_enable="YES"                 # Включить PF (загрузить модуль если необходимо)
pf_rules="/etc/pf.conf"         # определение правил для pf
pf_flags=""                     # дополнительные флаги для запуска pfctl
pflog_enable="YES"              # запустить pflogd(8)
pflog_logfile="/var/log/pflog"  # где pflogd должен сохранять протокол
pflog_flags=""                  # дополнительные флаги для запуска pflogd
```

Если за межсетевым экраном находится локальная сеть и необходимо передавать пакеты для компьютеров этой сети, или использовать NAT, включите также следующий параметр:

```
gateway_enable="YES"            # Включить сетевой шлюз
```

Конфигурация:

pf читает правила конфигурации из файла /etc/pf.conf во время загрузки, когда выполняются rc.scripts. Обратите внимание, что /etc/pf.conf - название по умолчанию и он представляет собой просто текстовый файл, содержащий наборы правил, загружаемые

и интерпретируемые pfctl(8) и вставляемые в pf(4). Для некоторых приложений наборы правил могут располагаться в других файлах и подгружаться после загрузки системы. Как и другие приложения UNIX, pf обладает большой гибкостью.

Файл pf.conf состоит из семи частей:

1. Макросы: Определяемые пользователем переменные, которые могут содержать адреса IP, имена интерфейсов, и т.д.
2. Таблицы: Применяются для хранения списков IP адресов
3. Опции: Параметры, влияющие на работу pf
4. Scrub: Подготовка пакета к нормализации и дефрагментации
5. Очереди: Обеспечивает управление полосой пропускания и установку приоритетов пакета.
6. Трансляции: Контроль NAT и перенаправлением пакета
7. Правила фильтрации: Осуществляют выборочную фильтрацию пакетов на интерфейсах

За исключением макросов и таблиц порядок следования разделов в файле должен быть таким же, но допускается отсутствие некоторых пунктов.

Пустые строки игнорируются, и строки начинающийся с # считаются комментарием.

Управление:

После начальной загрузки управление pf может осуществляться через программу pfctl. Вот несколько примеров:

```
# pfctl -f /etc/pf.conf     загрузить pf.conf
# pfctl -nf /etc/pf.conf    анализировать файл, но не загружать
# pfctl -Nf /etc/pf.conf    загрузить только правила NAT из файла
# pfctl -Rf /etc/pf.conf    загрузить только правила фильтрации
# pfctl -sn                 показать текущие правила NAT
# pfctl -sr                 показать текущие правила фильтрации
# pfctl -ss                 показать текущее состояние таблиц
# pfctl -si                 показать статистику правил и состояние счетчиков
# pfctl -sa                 показать все
```

Для полного списка команд смотрите man pfctl(8).

Списки и макросы

Списки:

Списки позволяют определять множества, имеющие общие признаки в пределах правила – такие как IP адреса, номера портов и т.д. Таким образом, вместо прописывания нескольких правил фильтрации для каждого IP адреса, который должен быть заблокирован, мы можем определить список IP адресов в пределах одного правила. Списки должны находиться внутри скобок {}.

Когда pfctl(8) доходит до списка при загрузке наборов правил, он раскладывает их на отдельные правила, для каждого элемента списка. Для примера:

```
block out on fxp0 from { 192.168.0.1, 10.5.32.6 } to any
```

Будет преобразован в:

```
block out on fxp0 from 192.168.0.1 to any
block out on fxp0 from 10.5.32.6 to any
```

Множественные списки могут применяться не только для блокировки:

```
rdr on fxp0 proto tcp from any to any port { 22 80 } ->  192.168.0.6
block out on fxp0 proto { tcp udp } from { 192.168.0.1,10.5.32.6 } to any port { ssh telnet }
```

Стоит отметить, что запятая в списке является необязательной.

Макросы:

Макросы – определяемые пользователем переменные, которые могут держать IP адреса, номера портов, имена интерфейсов, и т.д. Макросы позволят облегчить написание наборов правил и сделают поддержание набора правил несравненно легче.

Имена макросов должны начаться с символа и могут содержать символы, цифры, и символы подчеркивания. Названия макросов не могут носить имена зарезервированных слов, типа pass, out, или queue.

```
ext_if = "fxp0"
block in on $ext_if from any to an
```

Это создаст макрос с именем ext_if. При использовании макроса, его имени должен предшествовать знак $.
Макросы также могут быть расширены до списков.

```
friends = "{ 192.168.1.1, 10.0.2.5, 192.168.43.53 }"
```

Макросы могут определяться рекурсивно. В этом случае должен использоваться следующий синтаксис:

```
host1 = "192.168.1.1"
host2 = "192.168.1.2"
all_hosts = "{" $host1 $host2 "}"
```

Теперь макрос $all_hosts расширен до значений 192.168.1.1, 192.168.1.2.

Таблицы

Введение:

Таблицы используются для хранения группы адресов IPv6 и/или IPv4. Поиски в таблице занимают гораздо меньше времени и потребляют меньше ресурсов, чем списки. По этой причине, таблица идеальна чтобы хранить большую группу адресов, поскольку время поиска в таблице, содержащей 50 000 адресов – не намного больше чем для 50 адресов. Таблицы могут использоваться следующими способами:

* источник и/или адрес назначения для filter, scrub, NAT, и redirection rules
* адрес трансляции для правил NAT
* адрес переназначения для правил редиректа
* адрес назначения для правил фильтрации route-to, reply-to, и dup-to

Таблицы определяются в pf.conf или с помощью pfctl(8).

Конфигурация:

В pf.conf таблицы создаются используя директиву table. Следующие атрибуты могут быть определены для каждой таблицы:

* const – содержание таблицы не может быть изменено после ее создания. Когда этот атрибут не определен, pfctl(8) может использоваться, чтобы добавлять или удалять адреса из таблицы в любое время, при выполнении с securelevel(7), равным двум и выше.
* persist – заставляет ядро сохранять таблицу в памяти, даже когда никакие правила к ней не обращаются. Без этого атрибута, ядро автоматически удалит таблицу, когда последнее правило, ссылающееся на нее будет отработано.

Пример:

```
table <goodguys> { 192.0.2.0/24 }
table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }
table <spammers> persist
block in on fxp0 from { <rfc1918>, <spammers> } to any
pass  in on fxp0 from <goodguys> to any
```

Адреса могут также быть определены, используя модификатор типа отрицание (или "не"):

```
table <goodguys> { 192.0.2.0/24, !192.0.2.5 }
```

Таблица goodguys теперь содержит адреса сети 192.0.2.0/24, за исключением адреса 192.0.2.5.
Обратите внимание, что имена таблицы всегда включаются в <>.
Таблицы могут также могут заполняться из файлов, содержащих список адресов IP и сетей:

```
table  persist file "/etc/spammers"
block in on fxp0 from  to any
```

Файл/etc/spammers содержал бы список IP адресов или сетей CIDR. Любая строка начинающаяся с #, будет обработана как комментарий и проигнорирована.

Управление с помощью pfctl:

Таблицы могут управляться на лету, используя pfctl(8). Например, для добавления в таблицу <spammers> :

```
# pfctl -t spammers -Tadd 218.70.0.0/16
```

Это также откроет таблицу, если она не существует. Посмотреть значения в таблице:

```
# pfctl -t spammers -Tshow
```

Аргумент -v может использоваться совместно с -Tshow для отображнения каждой записи в таблице. Для удаления адреса из таблицы:

```
# pfctl -t spammers -Tdelete 218.70.0.0/16
```

Для получения дополнительной информации, смотрите pfctl(8).

Определение адресов:

В дополнение к IP адресу, хосты могут обозначаться по имени. Когда имя хоста разрешается в IP адрес, то оно может быть помещено в таблицу. Также IP адреса могут быть введены в таблицу с использованием имени интерфейса или ключевого слова "self", при этом в таблицу будут добавляться все IP адреса, назначенные интерфейсу.

Соответствие адресов:

Поиск адреса в таблице возвратит наиболее соответствующее значение. Это учитывается при создании таблиц типа:

```
table  { 172.16.0.0/16, !172.16.1.0/24, 172.16.1.100 }
block in on dc0 all
pass  in on dc0 from  to any
```

К любому пакету, входящему через dc0 будут применяться следующие правила:

* 172.16.50.5 – самое точное соответствие - 172.16.0.0/16; пакет соответствует таблице и будет передаваться
* 172.16.1.25 – самое точное соответствие - !172.16.1.0/24; пакет соответствует входу в таблицу, но тот вход инвертирован (использует "!" модификатор); пакет не соответствует таблице и будет блокирован
* 172.16.1.100 – точное значение 172.16.1.100; пакет соответствует таблице и будет передан
* 10.1.4.55 – не соответствует таблице и будет блокирован

Фильтрация пакетов

Введение:

Фильтрация пакета – выборочное принятие или блокирование пакета, проходящего через сетевой интерфейс. pf(4) может использоваться для контроля пакетов на 3 уровне модели OSI (IPv4 и IPv6) и 4 уровне модели OSI (TCP, UDP, ICMP, и ICMPv6). Наиболее часто используется в качестве критериев оценки - протокол, адрес назначения, источник и порт адресата.

Правила фильтра определяют критерии, которым пакет должен соответствовать. Правила просматриваются последовательно. Если пакет не соответствует правилу, содержащему ключевое слово quick, пакет будет оценен, как не соответствующий ни одному правилу, прежде, чем будет просмотрен весь список. Приоритет имеет последнее правило списка, соответствующее пакету. Есть подход "разрешать все" в начале списка, тогда, если пакет не блокируется ни одним последующим правилом, он будет пропущен.

Синтаксис правил:

Полный синтаксис правила:

action direction [log] [quick] on interface [af] [proto protocol] from src_addr [port src_port] to\
dst_addr [port dst_port] [tcp_flags] [state] action

Действие, которое будет применяться к пакету – это pass или block. pass передаст пакет назад к ядру для дальнейшей обработки , в то время как block будет реагировать в соответствии с block-policy. Реакция по умолчанию может быть отменена, определяя или block drop или block return.

direction

Описывает направление движения пакета – in или out.

log

Определяет, что пакет должен быть зарегистрирован через pflogd(8). Если правило определено как keep state, modulate state, или synproxy state, то будет зарегистрирован только первый пакет, создавший правило. Чтобы регистрировать все пакеты, используйте log-all.

quick

Если пакет соответствует правилу quick, правило считают последним правилом соответствия, и предпринимается указанное действие.

interface

Имя сетевого интерфейса, через который проходит пакет.

af

Семейство адреса пакета, или inet для IPv4 или inet6 для IPv6. PF обычно способен определить этот параметр, анализируя источник и/или адрес назначения.

protocol
Протокол пакета:

* tcp
* udp
* icmp
* icmp6
* Имя протокола из /etc/protocols
* Номер протокола от 0 до 255
* Используя список

src_addr, dst_addr

Источник/адрес назначения в заголовке IP. Адреса могут быть определены как:

* Единственный адрес IPv4 или адрес IPv6.
* Блок сети CIDR
* Полностью определенное имя домена, которое будет определено через DNS после загрузки правил. Все определенные имена будут заменены IP адресами.
* Имя сетевого интерфейса. Любые адреса, назначенные интерфейсу, будут вставлены в правило.
* Имя сетевого интерфейса, сопровождаемого / сетевой маской (например /24). Каждый адрес на интерфейсе будет объединен с сетевой маской, чтобы формировать блок сети CIDR
* Имя сетевого интерфейса в круглых скобках (). Это говорит PF модифицировать правило, если IP адрес(а) на названном интерфейсе изменяется. Это полезно на интерфейсе, получающего адрес через DHCP или модемного соединения, поскольку не надо каждый раз перезагружать правила.
* Имя сетевого интерфейса, сопровождаемого ключевыми словами :network или :broadcast. В результате в правила будет загружен адрес сети CIDR (к примеру 192.168.0.0/24) или широковещательный адрес (к примеру 192.168.0.255)
* Таблица.
* Любое вышеупомянутое, но отрицаемое(инвертированное) с использованием ! ("не") модификатора.
* Набор адресов, используя список
* Ключевое слово any
* Ключевое слово all, сокращенное от from any to any.

src_port, dst_port

Порт источника/адресата. Порты могут быть определены как:

* Номер между 1 и 65535
* Имя протокола из /etc/services
* Набор портов, используя список
* Диапазон:
    ! = (не равный)
    < (меньше чем)
    > (больше чем)
    <= (меньше чем или равный)
    >= (больше чем или равный)
    > <(диапазон)
    <> (обратный диапазон)

Последние два – двойные операторы (они берут два параметра) и не включают параметры в диапазон.

tcp_flags

Определяет флаги, которые должны быть установлены в заголовке TCP при использовании proto tcp. Флаги определяются как flags check/mask. Например: flags S/SA - PF будет смотреть на S и A (SYN и ACK) флаги и соответствовать правилу, если установлен флаг SYN.

state

Определяет, сохраняется ли информация о состоянии на пакетах, соответствующих этому правилу.

* keep state – работает с TCP, UDP, и ICMP
* modulate state – работы только с TCP. PF генерирует Initial Sequence Numbers (ISNs) для пакетов, соответствующих этому правилу.
* synproxy state – проксирует входящие TCP соединения, что помогает защищать сервера от TCP SYN флуда и спуфинга. Эта опция включает функциональные возможности keep state и modulate state.

Default Deny:

Рекомендуемой практикой является "default deny".
Для создания политики "default deny":

```
block in  all
block out all
```

Это заблокирует все пакеты, протоколы, IP адреса отовсюду куда угодно.

Прохождение трафика:

Теперь трафику необходимо явно пройти разрешающее правило, чтобы быть пропущеным. Это – то, где критерии пакета типа порта источника/адресата, источника/адреса назначения, и протокола входят в игру. Всякий раз, когда трафику разрешают пройти через систему сетевой защиты, правило(а) должно быть записано настолько конкретным, насколько это возможно. Это должно гарантировать,что только разрешенный трафик пройдет через систему защиты.

Некоторые примеры:

```
# Pass traffic in on dc0 from the local network, 192.168.0.0/24,
# to the OpenBSD machine's IP address 192.168.0.1. Also, pass the
# return traffic out on dc0.
pass in  on dc0 from 192.168.0.0/24 to 192.168.0.1
pass out on dc0 from 192.168.0.1 to 192.168.0.0/24

# Pass TCP traffic in on fxp0 to the web server running on the
# OpenBSD machine. The interface name, fxp0, is used as the
# destination address so that packets will only match this rule if
# they're destined for the OpenBSD machine.
pass in on fxp0 proto tcp from any to fxp0 port www
```

Ключевое слово quick:

Как говорилось выше, каждый пакет оценивается набором правил сверху вниз. По умолчанию, пакет отмечен для прохода, что может быть изменено в соответствии с любым правилом, и это может произойти несколько раз до конца списка правил. Последнее подходящее правило «побеждает». Есть одно исключение: использование ключевого слова quick, которое отменяет любую дальнейшую обработку пакета и осуществляет указанное в правиле действие над пакетом.

Посмотрим на пару примеров:

Неправильно:

```
block in on fxp0 proto tcp from any to any port ssh
pass  in all
```

В этом случае блокирующее правило будет оценено, но никогда применяться не будет, так как следующее правило разрешает любой трафик.

Уже лучше:

```
block in quick on fxp0 proto tcp from any to any port ssh
pass  in all
```

Эти првила оцениваются по другому, так как блокирующее правило указано с параметром quick, что приведет к блокировке и прекращению обработки пакета, пришедшего на порт ssh. Весь остальной трафик будет пропущен.

Keeping State:

Одной из важных способностей PF является "keeping state" или "stateful inspection". Контроль состояния относится к способности PF проследить состояние или прогресс сетевого подключения. информацию о каждом подключении в таблице состояний, PF способен быстро определить, принадлежит ли пакет, проходящий через систему сетевой защиты уже установленному подключению. Если пакет принадлежит уже установленному соединению, то проверки его правилами не происходит.
Keeping state имеет много преимуществ, включая упрощение набора правил и повышение производительности пакетного фильтра. PF способен согласовывать пакеты, двигающиеся в любом направлении, и так как пакеты, соответствующие stateful подключению не проходят ruleset оценку, время, затрачиваемое PF на обработку пакетов может быть значительно уменьшено.

Когда в правиле установлена опция keep state, первый пакет, соответствующий правилу создает "state" между отправителем и получателем. Теперь не только пакеты от отправителя к получателю, но и от получателя к отправителю не проходят проверку правилами фильтрации.

Например:

```
pass out on fxp0 proto tcp from any to any keep state
```

Таким образом любой исходящий трафик TCP от кого угодно куда угодно и обратный трафик будет пропущен и будут созданы временные правила "state" между отправителем и получателем. Это - хорошая особенность, ее использование значительно улучшает работу вашей системы сетевой защиты, поскольку поиски в правилах " state" значительно быстрее чем прохождение пакета через правила фильтра.

modulate state работает точно так же как и keep state, за исключением того, что это работает только с пакетами TCP. С использованием modulate state Initial Sequence Number (ISN) исходящего соединения будет рандомизирован. Это полезно для того, чтобы защитить подключения, инициализированные некоторыми операционными системами, у которых ISN легко предсказуем.

Keep state на исходящих пакетах TCP, UDP, и ICMP с рандомизацией ISN:

```
pass out on fxp0 proto tcp from any to any modulate state
pass out on fxp0 proto { udp, icmp } from any to any keep state
```

Другое преимущество keeping state - это передача ICMP трафика через пакетный фильтр. Для примера, если keep state установлен для TCP соединений, ICMP пакеты в обычном состоянии заблокированы, то после установления соединения ICMP пакеты будут проходить через через систему сетевой защиты.

Важно обратить внимание, что stateful подключения ограничены интерфейсом, на котором они были созданы. особенно важно это на маршрутизаторах и системах сетевой защиты, работающих с pf, особенно когда действует политика "default deny". Если система сетевой защиты сохраняет состояние на всех исходящих подключениях на внешнем интерфейсе, пакетам все еще нужно явно проходить внутренний интерфейс.

Обратите внимание что правила nat, binat, и rdr неявно создают "state" для того, чтобы пакет прошел фильтр.

Keeping State для UDP:

Каждый слышал, что для UDP нельзя создать "state" , так как это протокол без обратной связи! В то время как истинно, что UDP сеанс связи не имеет никакого понятия состояния (явное начало и останов сеанса связи), это не имеет никакого воздействия на способность PF создать состояние для UDP сеанса. В случае протоколов без "начальных" и "конечных" пакетов, PF просто следит, сколько прошло времени с момента прохождения первого пакета. Если время ожидания превышено, правило "state" удаляется. Значения времени ожидания могут быть установлены в разделе вариантов pf.conf

Флаги TCP:

* F: Отправитель закончил посылку байтов.
* S: Флаг для синхронизации номеров сегментов, используется при установлении связи.
* R: Прерывание связи.
* P: Этот сегмент требует выполнения операции push. Получатель должен передать эти данные прикладной программе как можно быстрее.
* A: Номер октета, который должен прийти следующим, правилен.
* U: Флаг важной информации, поле Указатель важной информации имеет смысл, если urg=1.
* E: ECE – Уведомление о перегрузке (пр.п. - не знаю что такое...)
* W: CWR – Уменьшение окна перегрузки (пр.п. - не знаю что такое...)

PF оценивает флаги когда в правиле встречается ключевое слово flags и имеет следующий синтаксис:

```
flags check/mask
```

mask говорит смотреть только указанные флаги и check определяет, какой флаг(и) должен "включен" в заголовке пакета для соответствия правилу.

```
pass in on fxp0 proto tcp from any to any port ssh flags S/SA
```

Вышеупомянутое правило передает трафик TCP с установленным флагом SYN, и только при наличии флагов SYN и ACK.
Контроль флагов очень часто используется вместе с keep state правилами для улучшения контроля за динамическими правилами.

```
pass out on fxp0 proto tcp all flags S/SA keep state
```

Это разрешило бы создание динамического правила на любом уходящем пакете TCP с установленным флагом SYN, и только при наличии флагов SYN и ACK.

Нужно быть внимательным при использовании флагов., понимать что конкретно делается и почему. Советам тоже особо не верьте, так как многие бы посоветовали создавать правило "только если флажок SYN установлен и никакие другие". Это кончится следующим:

```
. . . flags S/FSRPAUEW  bad idea!!
```

Проблема – некоторые сайты, используют флаг ECN и любой сайт, используя ECN, пробуя соединиться с Вами, будет отклонен в соответствии с правилом выше. Намного лучше:

```
. . . flags S/SAFR
```

В то время как это является практическим и безопасным, также ненужно проверять

флаги FIN и RST если трафик scrubbed.

Процесс очистки заставит PF блокировать любые входящие пакеты с незаконными комбинациями флагов TCP (типа SYN и FIN или SYN и RST). Строго рекомендуют всегда вычистить входящий трафик:

```
scrub in on fxp0
.
.
.
pass in on fxp0 proto tcp from any to any port ssh flags S/SA \
  keep state
```

TCP SYN Proxy:

Обычно, когда клиент устанавливает соединение с сервером вне сети, pf передает пакеты установления связи между клиентом и сервером по мере их поступления. PF также имеет способность проксировать процесс создания соединения. При этом PF установит связь с клиентом, инициирует установление связи с сервером и затем соединит клиента и сервер. Преимущества этого процесса в том, что никакие пакеты не посылаются серверу прежде, чем клиент завершит установление связи. Это устраняет угрозу spoofed TCP SYN флуда, та как клиентское подключение будет неспособно завершить установление связи и спуфить сервер.

```
pass in on $ext_if proto tcp from any to $web_server port www \
  flags S/SA synproxy state
```

Теперь соединение с сервером будет проксировано пакетным фильтром.
Так как при работе synproxy state образуются динамические правила, то SYN прокси включает в себя функциональность keep state и modulate state.
SYN прокси-сервер не будет работать, если PF выполняется на bridge(4).

Блокирование спуфинга:

Под термином "spoofing" понимается подмена IP адреса в пакетах с целью скрыть их реальный адрес или выдать себя за другой узел сети. При этом злоумышленник может попытаться получить доступ к сетевым услугам, которые ограничены некоторыми IP адресами или скрывать свой истинный адрес при проведении атаки на сеть.
PF предлагает некоторую защиту против спуфинга с помощью ключевого слова antispoof

```
antispoof [log] [quick] for interface [af]
```

log
Определяет, что соответствующий пакет должен быть зарегистрирован через pflogd (8).

quick
Если пакет соответствует правилу quick, правило считают последним правилом соответствия, и предпринимается указанное действие.

interface
Сетевой интерфейс, на котором активизируется защита от спуфинга. Это может также быть списком интерфейсов.

af
Семейство адреса – или inet для IPv4 или inet6 для IPv6.

Пример:

```
antispoof for fxp0 inet
```

После загрузки наборов правил, там где встречено ключевое слово antispoof правило будет расширено в два правила фильтра. Принимая, что интерфейс fxp0 имеет адрес IP 10.0.0.1 и подсетевую маску 255.255.255.0 (то есть,/24), вышеупомянутое правило расширилось бы до:

```
block in on ! fxp0 inet from 10.0.0.0/24 to any
block in inet from 10.0.0.1 to any
```

Эти правила достигают двух вещей:

* Блокирует весь трафик, исходящий из сети 10.0.0.0/24, который не проходит в через fxp0. Так как сеть 10.0.0.0/24 находится на интерфейсе fxp0, пакеты с исходным адресом в этом сетевом блоке никогда не должны появляться ни на каком другом интерфейсе.
* Блокирует весь входящий трафик с IP адреса 10.0.0.1 на fxp0. Хост никогда не не должен посылать пакеты себе через внешний интерфейс, так что любые входящие пакеты с исходным адресом, принадлежащим хосту можно счесть злонамеренными.

ОБРАТИТЕ ВНИМАНИЕ: При расширении правила antispoof также блокируются пакеты, посланные по петлевому интерфейсу к местным адресам. Эти адреса нужно передать явно. Пример:

```
pass in quick on lo0 all
antispoof for fxp0 inet
```

Использование antispoof должно быть ограничено интерфейсами, которым был назначен IP адрес. Использование antispoof на интерфейсе без IP адреса приведет к правилам фильтрации типа:

```
block drop in on ! fxp0 inet all
block drop in inet all
```

С этими правилами есть риск блокирования всего прибывающего трафика на всех интерфейсах.

Опции IP:

По умолчанию pf блокирует пакеты с установленными опциями IP. Это затрудняет определение типа ОС такими утилитами, как nmap. Если Вы имеете приложение, которое требует прохождения этих пакетов, типа multicast или IGMP, Вы можете использовать директиву allow-opts:

```
pass in quick on fxp0 all allow-opts
```

Пример правил фильтрации:

Ниже – пример фильтрующих правил. Сервер с запущенным pf действует как система сетевой защиты между маленькой, внутренней сетью и Internet. Показаны только фильтрующие правила; правила queueing, nat, rdr и т.д. в этом примере пропущены.

```
ext_if  = "fxp0"
int_if  = "dc0"
lan_net = "192.168.0.0/24"

# scrub incoming packets
scrub in all

# setup a default deny policy
block in  all
```

```
block out all

# pass traffic on the loopback interface in either direction
pass quick on lo0 all

# activate spoofing protection for the internal interface.
antispoof quick for $int_if inet

# only allow ssh connections from the local network if it's from the
# trusted computer, 192.168.0.15. use "block return" so that a TCP RST is
# sent to close blocked connections right away. use "quick" so that this
# rule is not overridden by the "pass" rules below.
block return in quick on $int_if proto tcp from ! 192.168.0.15 to $int_if port ssh flags S/SA

# pass all traffic to and from the local network
pass in  on $int_if from $lan_net to any
pass out on $int_if from any to $lan_net

# pass tcp, udp, and icmp out on the external (Internet) interface.
# keep state on udp and icmp and modulate state on tcp.
pass out on $ext_if proto tcp all modulate state flags S/SA
pass out on $ext_if proto { udp, icmp } all keep state

# allow ssh connections in on the external interface as long as they're
# NOT destined for the firewall (i.e., they're destined for a machine on
# the local network). log the initial packet so that we can later tell
# who is trying to connect. use the tcp syn proxy to proxy the connection.
pass in log on $ext_if proto tcp from any to { !$ext_if, !$int_if } port ssh flags S/SA
synproxy state
```

NAT

Введение:

Трансляция Сетевого адреса (NAT) - отобразить полную сеть (или сети) к единственному IP адресу. NAT необходим, когда количество IP адресов, Выделенный Вам Вашим провайдером меньше чем общее количество компьютеров, для которых Вы желаете обеспечить доступ Internet. NAT описан в RFC 1631.

NAT позволяет Вам использовать в своих интересах зарезервированные блоки адресов, описанные в RFC 1918. Как правило, Вы будете использовать один или несколько блоков адресов из следующего диапазона:

* 10.0.0.0/8 (10.0.0.0 - 10.255.255.255)
* 172.16.0.0/12 (172.16.0.0 - 172.31.255.255)
* 192.168.0.0/16 (192.168.0.0 - 192.168.255.255)

Для работы NAT необходимо иметь как минимум два сетевых интерфейса, один из которых выходит в Internet, а второй во внутреннюю сеть. NAT будет транслировать запросы с внутренней сети, так, как будто они исходят от Вашей системы.

Как работает NAT:

Когда клиент из внутренней сети посылает запрос в Internet, создаются IP пакеты, которые шлются к месту назначения. Эти пакеты содержат всю информацию об обратном адресе, необходимую для получения ответа. NAT заинтересован этими частями информации:

* Исходный адрес IP (например, 192.168.1.35)
* Исходный TCP или UDP порт (например, 2132)

Когда пакеты проходят через NAT, они изменяются таким образом, чтобы они, казалось, исходили от NAT шлюза непосредственно. NAT будет делать запись изменений в таблице состояний так, чтобы это могло,
a), чтобы полностью вернуть изменения на возвращающихся пакетах и
b) гарантировать, что вернувшиеся пакеты пройдут систему сетевой защиты и не будут блокированы.

Например, могут произойти следующие изменения:

* Исходный IP: будет заменен внешним адресом шлюза (например, 24.5.0.5)
* Исходный порт: будет заменен беспорядочно выбранным, неиспользованным портом на шлюзе (например, 53136)

Ни внутренняя машина, ни главный компьютер Internet не знают об этих шагах трансляции. Для внутренней машины NAT система – просто шлюз Internet. Для ресурса, расположенного в Internet пакеты прибывают непосредственно от NAT системы; он не догадывается о существовании клиентской машины. Когда ресурс отвечает на запрос внутренней машины он будет обращаться к внешнему IP адресу NAT шлюза (24.5.0.5) и порту (53136). NAT шлюз будет искать соответствующую запись в таблице , чтобы определить, соответствуют ли пакеты ответа уже установленному подключению. Уникальное соответствие будет найдено на основании комбинации IP/порта, которая говорит PF, что пакеты принадлежат подключению, инициализированному внутренней машиной 192.168.1.35. PF будет тогда делать обратные изменения для приходящих пакетов и отправлять пакеты внутренней машине.

Трансляция ICMP пакетов происходит таким же способом, но без исходной модификации порта.

NAT и фильтрация пакетов:

ОБРАТИТЕ ВНИМАНИЕ: Оттранслированные пакеты должны все еще должны проходить через механизм фильтрации и будут блокированы или переданы на основании правил фильтрации. Единственное исключение будет сделано при наличии ключевого слова pass при использовании в пределах правила NAT. Это заставит NATed пакеты передавать через механизм фильтрации.

Также знайте, что трансляция происходит перед фильтрацией и механизм фильтрации будет видеть оттранслированный пакет с оттранслированным адресом IP и портом.

IP форвардинг:

Так как NAT почти всегда используется на маршрутизаторах и сетевых шлюзах, вероятно будет необходимо допустить форвардинг пакетом между сетевыми интерфейсами машины. Это осуществляется с помощью механизма sysctl(3) :

sysctl -w net.inet.ip.forwarding=1 # sysctl -w net.inet6.ip6.forwarding=1 (if using IPv6)

Для того, чтобы сделать изменения постоянными, внесите в /etc/sysctl.conf:

net.inet.ip.forwarding=1 net.inet6.ip6.forwarding=1

Эти строки присутствуют, но прокомментированны (знаком #) в заданной по умолчанию установке. Удалите #, и сохраните файл. Форвардинг пакетов IP будет будет разрешен после перезагрузки.

Конфигурирование NAT:

Полный формат правила NAT выглядит следующим образом:

nat [pass] on interface [af] from src_addr [port src_port] to \
dst_addr [port dst_port] -> ext_addr [pool_type] [static-port]

nat
Ключевое слово, которое начинает правило NAT pass
Транслировать пакеты, полностью обходя правила фильтрации interface
Имя сетевого интерфейса, на который будут транслироваться пакеты af
Семейство адреса пакета, или inet для IPv4 или inet6 для IPv6. PF обычно способен определить этот параметр, анализируя источник и/или адрес назначения. src_addr
- Внутренний адрес пакетов, которые будут оттранслированы. Исходный адрес может быть определен как: Единственный адрес IPv4 или адрес IPv6.
- Блок сети CIDR
- Полностью определенное имя домена, которое будет определено через DNS после загрузки правил. Все определенные имена будут заменены IP адресами.
- Имя сетевого интерфейса. Любые адреса, назначенные интерфейсу, будут вставлены в правило.
- Имя сетевого интерфейса, сопровождаемого / сетевой маской (например /24). Каждый адрес на интерфейсе будет объединен с сетевой маской, чтобы формировать блок сети CIDR
- Имя сетевого интерфейса, сопровождаемого ключевыми словами :network. В результате в правила будет загружен адрес сети CIDR (к примеру 192.168.0.0/24).
- Таблица.
- Любое вышеупомянутое, но отрицаемое(инвертированное) с использованием ! ("не") модификатора.
- Набор адресов, используя список
- Ключевое слово any
src_port
- Исходный порт. Порты могут быть определены как: Номер между 1 и 65535
- Имя протокола из /etc/services
- Набор портов, используя список
- Диапазон:
+ ! = (не равный)
+ < (меньше чем)
+ > (больше чем)
+ <= (меньше чем или равный)
+ >= (больше чем или равный)
+ > <(диапазон)
+ <> (обратный диапазон)
Последние два - двойные операторы (они берут два параметра) и не включают параметры в диапазон.
Опция port обычно не используется в правилах NAT, потому что весь трафик проходит через правило независимо от используемого порта (ов). dst_addr
Адрес назначения пакетов, которые будут оттранслированы. Адрес назначения имеет те же параметры, что и исходный адрес. dst_port
Порт адресата. Параметры такие же как и в src_port. ext_addr
- Внешний адрес NAT роутера, к которому будут оттранслированы пакеты. Внешний

адрес может быть определен как: Единственный адрес IPv4 или адрес IPv6.
- Блок сети CIDR
- Полностью определенное имя домена, которое будет определено через DNS после загрузки правил. Все определенные имена будут заменены IP адресами.
- Имя внешнего сетевого интерфейса. Любые адреса, назначенные интерфейсу, будут вставлены в правило.
- Имя сетевого интерфейса в круглых скобках (). Это говорит PF модифицировать правило, если IP адрес(а) на названном интерфейсе изменяется. Это полезно на интерфейсе, получающего адрес через DHCP или модемного соединения, поскольку не надо каждый раз перезагружать правила.
- Имя сетевого интерфейса, сопровождаемого ключевыми словами :network. В результате в правила будет загружен адрес сети CIDR (к примеру 192.168.0.0/24).
- Набор адресов, используя список

pool_type
Определяет тип пула адреса, используемый для трансляции. static-port
Говорит PF не транслировать исходный порт в TCP и UDP пакетах.

Это приводит к следующему каноническому виду правила:

```
nat on tl0 from 192.168.1.0/24 to any -> 24.5.0.5
```

Это правило подразумевает то, что на интерфейсе tl0 применяется NAT к любым пакетам, исходящих из сети 192.168.1.0/24 и исходный адрес заменяется на IP 24.5.0.5.

В то время как вышеупомянутое правило правильно, такая форма записи не рекомендуется. Обслуживание таких правил будет затруднено, так как любое изменение внешних или внутренних сетевых адресов будет требовать переделывать все правила в списке. Сравните с этой записью (tl0, является внешним, dc0 внутренним):

```
nat on tl0 from dc0/24 to any -> tl0
```

Преимущество должно быть довольно очевидно: Вы можете изменить IP адрес любого интерфейса, не изменяя это правило.

При указании имени интерфейса для адреса трансляции как в примере выше, IP адрес определяется в pf.conf во время загрузки, а не на лету. Если Вы используете протокол DHCP, чтобы конфигурировать ваш внешний интерфейс, это может быть проблемой. Так как адрес, назначенный интерфейсу может измениться, а в правилах останется старый адрес, пакеты клиентских машин будут уходить в никуда. Чтобы обойти это, Вы можете сказать PF автоматически модифицировать адрес трансляции, помещая имя интерфейса в круглые скобки.

```
nat on tl0 from dc0/24 to any -> (tl0)
```

Есть одно главное ограничение: только первый псевдоним на интерфейсе будет оценен, когда имя интерфейса помещено в круглые скобки.

Bidirectional Mapping (1:1 mapping):

bidirectional mapping - это постоянное соединение, использующее правило binat. Правило binat устанавливает соответствие портов между внутренним адресом и внешним адресом. Это может быть полезно, например, когда внутри сети находится сервер, который должен быть доступен извне. Подключения от Internet до внешнего адреса будут оттранслированы к внутреннему адресу, и подключения от сервера сети (типа запросов DNS) будут оттранслированы к внешнему адресу.

Пример:

```
web_serv_int = "192.168.1.100"
web_serv_ext = "24.5.0.6"
binat on tl0 from $web_serv_int to any -> $web_serv_ext
```

Исключения из правил трансляции:

Исключения могут быть сделаны к правилам трансляции используя ключевое слово no. Например:

```
no nat on tl0 from 192.168.1.10 to any
nat on tl0 from 192.168.1.0/24 to any -> 24.2.74.79
```

В этом случае вся сеть 192.168.1.0/24 за исключением 192.168.1.10 будет транслирована на IP адрес 24.2.74.79.

Обратите внимание, что первое правило побеждает, если указано "no", то пакет не будет транслирован. Ключевое слово "no" нельзя использовать с правилами with binat и rdr.

Проверка состояния NAT:

Просмотреть активные трансляции NAT можно с помощью pfctl(8), используя опцию -s state.

```
# pfctl -s state
TCP 192.168.1.35:2132 -> 24.5.0.5:53136 -> 65.42.33.245:22 TIME_WAIT:TIME_WAIT
UDP 192.168.1.35:2491 -> 24.5.0.5:60527 -> 24.2.68.33:53   MULTIPLE:SINGLE
```

Расшифровка (только верхней строки):

TCP
: Протокол, используемый подключением. 192.168.1.35:2132
: Адрес (192.168.1.35) машины из внутренней сети и исходный порт (2132) . Также это адрес, который заменяется в заголовке IP пакета. 24.5.0.5:53136
: Адрес IP (24.5.0.5) и порт (53136) на шлюзе, к которому пакеты транслируются. 65.42.33.245:22
: Адрес IP (65.42.33.245) и порт (22), с которым внутренняя машина соединяется. TIME_WAIT:TIME_WAIT
: Это указывает на состояние соединения - в состоянии подключения.

Перенаправление трафика

Введение:

Когда работает NAT, Вы можете обеспечить весь офис доступом в Internet. Что делать, когда у Вас есть машина позади NAT, к которой необходимо обращаться снаружи. Это тот случай, когда в игру вступает перенаправление. Перенаправление позволяет входящему трафику быть посланным внутрь сети, стоящей за NAT.

Рассмотрим пример:

```
rdr on tl0 proto tcp from any to any port 80 -> 192.168.1.20
```

Эта строка переадресовывает трафик, приходящий на TCP порт 80 на машину, находящуюся в локальной сети, с адресом 192.168.1.20. from any to any запись в правиле rdr может быть весьма полезна. Но если вы знаете какие IP адреса или подсети могут обращаться к этой машине, то Вы можете ограничить доступ:

```
rdr on tl0 proto tcp from 27.146.49.0/24 to any port 80 -> 192.168.1.20
```

Это правило будет переадресовывать только одну подсеть 27.146.49.0/24. Это наводит на мысль что таким образом можно предоставлять доступ различным хостам внешней сети к различным хостам внутренней сети. Например, Вы могли бы иметь удаленных пользователей, обращающихся к своим собственным машинам внутри локальной сети.

Например:

```
rdr on tl0 proto tcp from 27.146.49.14 to any port 80 -> 192.168.1.20
rdr on tl0 proto tcp from 16.114.4.89 to any port 80 ->  192.168.1.22
rdr on tl0 proto tcp from 24.2.74.178 to any port 80 -> 192.168.1.23
```

Перенаправление и фильтрация пакетов:

ОБРАТИТЕ ВНИМАНИЕ: Оттранслированные пакеты должны все еще должны проходить через механизм фильтрации и будут блокированы или переданы на основании правил фильтрации.

Единственное исключение будет сделано при наличии ключевого слова pass при использовании в пределах правила rdr. Это заставит переадресованные пакеты передавать через механизм фильтрации.

Также учтите, что механизм фильтрации будет видеть оттранслированный пакет, с уже измененными адресами и портами.

Рассмотрим такой сценарий:

* 192.0.2.1 – хост в интернете
* 24.65.1.13 – внешний адрес маршрутизатора
* 192.168.1.5 – внутренний адрес машины в локальной сети

Правило перенаправления:

```
rdr on tl0 proto tcp from 192.0.2.1 to 24.65.1.13 port 80 -> 192.168.1.5 8000
```

Перед правилом rdr пакет имеет характеристики:

* Исходный адрес: 192.0.2.1
* Исходный порт: 4028 (произвольно выбранный операционной системой)
* Адрес назначения: 24.65.1.13
* Порт адресата: 80

Пакет после правила rdr:

* Исходный адрес: 192.0.2.1
* Исходный порт: 4028
* Адрес назначения: 192.168.1.5
* Порт адресата: 8000

Пакет будет обрабатываться правилами фильтрации.

Значение защиты:

Перенаправление потенциально может создать проблемы с безопасностью. Приходится делать отверстие в системе защиты, для того чтобы внутренние сервисы были доступны снаружи. Если трафик был перенаправлен на внутренний web сервер и в CGI скрипте или в сервере была обнаружена уязвимость, это может скомпрометировать машину, а оттуда и всю внутреннюю сеть.

Этот риск может быть уменьшен, если вывести все серверы, к которым будут обращения извне, в отдельную сеть. Такая сеть называется "демилитаризованной зоной"(DMZ) или "частная сервисная сеть"(PSN). Даже если сервер сети будет скомпрометирован, все неприятные последствия будут ограничены этой зоной, используя фильтрацию трафика из и в DMZ.

Перенаправление и отражение:

Часто, правила переназначения используются, чтобы отправить входящие подключения с Internet на местный сервер с частным адресом во внутренней сети, как в данном примере:

```
server = 192.168.1.40
rdr on $ext_if proto tcp from any to $ext_if port 80 -> $server port 80
```

Но при проверке работы этого правила из локальной сети мы потерпим неудачу. Это связано с тем, что правило применяется только к пакетам, которые проходят через указанный интерфейс ($ext_if, внешний интерфейс, в примере). Соединение с внешним адресом системы сетевой защиты с машины внутри локальной сети , однако, не подразумевает, что пакеты фактически пройдут через его внешний интерфейс. Стек TCP/IP на системе сетевой защиты сравнивает адрес назначения входящих пакетов с его собственными адресами и псевдонимами и обнаружит подключение с собой, как только пакеты прийдут на внутренний интерфейс. Такие пакеты физически не проходят через внешний интерфейс. Таким образом, PF никогда не видит эти пакеты на внешнем интерфейсе, и правило не срабатывает.

Добавление второго правила переназначения для внутреннего интерфейса также не будет иметь положительного эффекта. Когда местный клиент соединяется с внешним адресом системы сетевой защиты, начальный пакет установления связи TCP достигает системы сетевой защиты через внутренний интерфейс. Правило переназначения действительно применяется, и адрес назначения заменяется адресом внутреннего сервера. Но исходный адрес не был оттранслирован, и все еще содержит адрес местного клиента, так что сервер посылает свои ответы непосредственно клиенту. Система сетевой защиты никогда не видит ответ и не имеет никакого шанса должным образом полностью контролировать трансляцию. поскольку клиент получает ответ непосредственно от сервера, а запрос на соединение посылается через pf, на сервере сетевой защиты соединение никогда так и не будет установлено.

Однако бывает желательным соединять внутренних клиентов также как и внешних. Решение данной проблемы существует.

Разделение горизонтов DNS:

Возможно такое конфигурирование сервера DNS, чтобы он отвечал на запросы компьютеров локальной сети по другому, чем для запросов из Internet. То есть, чтобы клиенты внутренней сети получали внутренний адрес сервера, тогда они будут соединяться напрямую и система защиты на маршрутизаторе вообще задействоваться не будет. Это уменьшает локальный трафик и снижает нагрузку на маршрутизатор.

Перемещение сервера в DMZ:

Добавив дополнительный сетевой интерфейс к маршрутизатору можно создать демилитаризованную зону. Переместив туда все серверы, к которым требуется обращение извне, можно организовать доступ из локальной сети точно таким же

образом, как и для клиентов из Internet. Использование отдельных сетей имеет несколько преимуществ, включая улучшение качества защиты. Если сервер (который в нашем случае является доступным из Internet), будет скомпрометирован, он не сможет обратиться к компьютерам локальной сети непосредственно, поскольку все подключения должны будут пройти через систему сетевой защиты.

TCP проксирование:

На системе сетевой защиты может быть установлен универсальный прокси-сервер TCP, или слушающий на порту, который будет перенаправлен, или получающий подключения на внутреннем интерфейсе. Когда локальный клиент соединяется с системой сетевой защиты, прокси-сервер принимает подключение, устанавливает второе подключение с внутренним сервером, и передает данные между этими двумя подключениями.

Простое проксирование можно организовать используя inetd(8) и nc(1). Следующие записи в etc/inetd.conf создают сокет, привязанный к петлевому адресу (127.0.0.1) и порту 5000. Подключения перенаправляются на порт 80 сервера 192.168.1.10.

```
127.0.0.1:5000 stream tcp nowait nobody /usr/bin/nc nc -w  20 192.168.1.10 80
```

Следующее правило перенаправляет порт 80 на внутреннем интерфейсе к прокси-серверу:

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 ->   127.0.0.1 port 5000
```

Комбинация RDR и NAT:

С дополнительным правилом NAT на внутреннем интерфейсе может быть решена проблема недостающей переадресации.

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> $server
no nat on $int_if proto tcp from $int_if to $int_net
nat on $int_if proto tcp from $int_net to $server port 80 -> $int_if
```

Это заставит начальный пакет от клиента быть оттранслированным снова, когда он пройдет назад через внутренний интерфейс, заменяя исходный адрес клиента на внутренний адрес системы сетевой защиты. Внутренний сервер ответит системе сетевой защиты, которая может полностью изменить и NAT и RDR трансляции при отправке пакета локальному клиенту. Эта конструкция довольно сложна, поскольку создает два различных соединения. Необходимо проявить осторожность при написании подобных правил, чтобы предотвратить передачу трафика к другим ресурсам в обход системы защиты. Обратите внимание, что правило rdr заставит стек TCP/IP видеть, что пакеты прибывают на внутренний интерфейс с адресом назначения во внутренней сети. Для того, чтобы запретить шлюзу выдавать ICMP сообщение клиенту о том, что сервер доступен напрямую, отключаем эту функцию:

```
# sysctl -w net.inet.ip.redirect=0
# sysctl -w net.inet6.ip6.redirect=0 (if using IPv6)
```

Вообще, не стоит использовать такое решение...

Шаблоны для создания наборов правил

Введение:

PF предлагает много путей упростить наборы правил. Некоторые из них - использование макросов и списков. Кроме того грамматика или язык построения наборов правил предлагает некоторые шаблоны, призванные облегчить написание правил. Как правило, чем проще набор правил, тем легче его поддерживать.

Использование макросов:

Макросы полезны тем, что обеспечивают альтернативу написанию большого количества раз одинаковых адресов, портов, интерфейсов, и т.д., в наборе правил. IP адрес сервера изменился? Никаких проблем, только модифицируйте макрос и используйте свободное время для самосовершенствования.

Общее соглашение в PF состоит в том, чтобы определить макрос для каждого сетевого интерфейса. Если, к примеру, была заменена сетевая карта, то макрос модифицируется и правила выполняются как и прежде. Другой пример – установка одинакового набора правил на многих машинах, которые могут иметь разные сетевые интерфейсы:

```
# define macros for each network interface
IntIF = "dc0"
ExtIF = "fxp0"
DmzIF = "fxp1"
```

Другое соглашение состоит в использовании макросов для определения адресов и сетевых блоков:

```
# define our networks
IntNet = "192.168.0.0/24"
ExtAdd = "24.65.13.4"
DmzNet = "10.0.0.0/24"
```

Если внутренняя сеть была изменена или расширена, то макрос изменяется:

```
IntNet = " {192.168.0.0/24, 192.168.1.0/24} "
```

После перезагрузки набор правил будет функционировать.

Использование списков:

Давайте посмотрим, какие записи в наборе правил должны иметься, чтобы отработать RFC 1918 – о блокировании на внешнем интерфейсе адресов из приватного диапазона сетей.

```
block in  quick on tl0 inet from 127.0.0.0/8 to any
block in  quick on tl0 inet from 192.168.0.0/16 to any
block in  quick on tl0 inet from 172.16.0.0/12 to any
block in  quick on tl0 inet from 10.0.0.0/8 to any
block out quick on tl0 inet from any to 127.0.0.0/8
block out quick on tl0 inet from any to 192.168.0.0/16
block out quick on tl0 inet from any to 172.16.0.0/12
block out quick on tl0 inet from any to 10.0.0.0/8
```

Теперь взглянем на следующее упрощение:

```
block in  quick on tl0 inet from {127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8} to any
block out quick on tl0 inet from any to {127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12,
10.0.0.0/8 }
```

В результате, число записей было уменьшено с восьми до двух. Прилепим сюда еще и список:

```
NoRouteIPs = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }"
ExtIF = "tl0"
block in  quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs
```

Обратите внимание, что макросы и списки упрощают строение файла pf.conf, но при просмотре правил с помощью pfctl(8) они будут расширены. Для примера выше это будет:

```
block in  quick on tl0 inet from 127.0.0.0/8 to any
block in  quick on tl0 inet from 192.168.0.0/16 to any
block in  quick on tl0 inet from 172.16.0.0/12 to any
block in  quick on tl0 inet from 10.0.0.0/8 to any
block out quick on tl0 inet from any to 10.0.0.0/8
block out quick on tl0 inet from any to 172.16.0.0/12
block out quick on tl0 inet from any to 192.168.0.0/16
block out quick on tl0 inet from any to 127.0.0.0/8
```

Как видно – все это способы упрощения написания файла pf.conf, а не выполнения правил непосредственно pf(4).
Макросы также могут использоваться для определения непосредственно правил:

```
pre = "pass in quick on ep0 inet proto tcp from "
post = "to any port { 80, 6667 } keep state"

# David's classroom
$pre 21.14.24.80 $post

# Nick's home
$pre 24.2.74.79 $post
$pre 24.2.74.178 $post
```

Расширяется до:

```
pass in quick on ep0 inet proto tcp from 21.14.24.80 to any  port = 80 keep state
pass in quick on ep0 inet proto tcp from 21.14.24.80 to any  port = 6667 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.79 to any   port = 80 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.79 to any   port = 6667 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.178 to any  port = 80 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.178 to any  port = 6667 keep state
```

Грамматика PF:

Грамматика PF весьма гибка, что определяет гибкость правил фильтрации. PF способен вывести некоторые ключевые слова, что означает, что они не должны быть явно заявлены в правиле.

Устранение ключевых слов:

Политика "default deny", состоящая из двух правил:

```
block in all block out all
```

Может быть сокращена до:

```
block all
```

Когда направление движения пакета не указано, pf предполагает, что правило применяется к пакетам, двигающимся в обоих направлениях.

Точно так же и правила "from any to any" и "all" могут быть сокращены. Для примера:

```
block in on rl0 all pass in quick log on rl0 proto tcp from any to any port 22 keep state
```

Упрощается до:

```
block in on rl0 pass in quick log on rl0 proto tcp to port 22 keep state
```

Упрощение Return:

В наборе правил обычно принято блокировать пакеты ICMP Unreachable и TCP RST :

```
block in all block return-rst in proto tcp all block return-icmp in proto udp all block out all block return-rst out proto tcp all block return-icmp out proto udp all
```

Эта конструкция может быть упрощена до:

```
block return
```

Когда PF видит ключевое слово return, он посылает надлежащий ответ, или вообще ничего не посылает, в зависимости от протокола блокируемого пакета.

Упорядочивание ключевых слов:

Порядок следования следования ключевых слов гибок в большинстве случаев. Например, правило:

```
pass in log quick on rl0 proto tcp to port 22 flags S/SA keep state queue ssh label ssh
```

Может быть записано и так:

```
pass in quick log on rl0 proto tcp to port 22 queue ssh keep state label ssh flags S/SA
```