OBSD.RU

Создание межсетевого экрана с помощью PF OpenBSD

printable page

Перевод: Сгибнев Михаил

Предварительные комментарии

В этой лекции будет рассказано о системах сетевой защиты и смежных вопросах, приведено немного теории и большое количество примеров фильтрации и перенаправления трафика. Как и в подавляющем большинстве любых других случаев, все описанное здесь можно сделать более чем одним способом. Ваши комментарии и дополнения приветствуются.

Внимание:

Этот документ является переводом с норвежского, лекции, прокомментированной следующим образом: "Эта лекция - о системах сетевой защиты и связанных с ними вопросах, с примерами из реальной жизни На основе пакетного фильтра PF проекта OpenBSD. Данный пакетный фильтр предлагает использование систем защиты сетей, трансляцию сетевых адресов (Network Address Translation), управление трафиком и управление полосой пропускания в единственной, гибкой и дружественной администратору системе. Питер надеется, что лекция даст вам некоторые идеи о том, как управлять вашим сетевым трафиком, решая все стоящие перед вами задачи. "

PF?

Сначала скажем несколько слов о предмете нашего разговора, пакетном фильтре PF операционной системы OpenBSD.

PF был написан в течении лета-осени 2001 года Дэниелом Хартмеиром(Daniel Hartmeier) и сообществом разработчиков OpenBSD. В декабре 2001 года PF вошел в основной состав ОС OpenBSD 3.0.

Потребность в системе сетевой защиты для OpenBSD назрела тогда, когда Даррен Рид(Darren Reed) обьявил миру, что IPFilter, интегрированный в OpenBSD не будет распространяться по лицензии BSD. На самом деле, в тексте лицензии ничего не изменилось, из слова в слово повторяя текст BSD, было запрещено только делать изменения в исходном коде. OpenBSD версия IPFilter содержала множество изменений и настроек, которые не соответствовали тексту лицензии. IPFilter был удален из дерева исходных текстов OpenBSD 29-ого мая 2001 и в течение нескольких недель OpenBSD-current не содержал программного обеспечения для фильтрации пакетов.

К счастью, в Швейцарии, Дэниел Хартмеир уже экспериментировал с кодом ядра.

Когда случился кризис лицензирования, патч Дэниэла вклинивался в сетевой стэк и у него появились мысли о возможности фильтрации пакетов.

IPFilter был удален из дерева исходных текстов 29 мая. ервый коммит кода PF был сделан в воскресенье, 24

июня 2001 в 19:48:58 UTC.

Затем последовали несколько месяцев интенсивной работы и версия PF в составе OpenBSD 3.0 была уже вполне законченным вариантом пакетного фильтра, включающего в себя возможность NAT.

Дэниел Хартмеир и другие разработчики PF учли опыт работы с IPFilter. На USENIX 2002 были проведены тесты, показавшие, что на OpenBSD 3.1 PF показал производительность сравнимую, а иногда и превосходящую работу IPFilter на той же системе и iptables под Linux на аналогичном оборудовании.

Кроме того, некоторые тесты были проведены на оригинальном PF из OpenBSD 3.0. Они проводились главным образом для сравнения версий 3.0 и 3.1. Информция по этим тестам доступна на странице Дэниела Хартмеира http://www.benzedrine.cx/pf-paper.html.

Я не видел результатов тестов, выполненных недавно, но на собственном опыте могу сказать что PF не требователен к машинным ресурсам. Например, маршрутизацию между сетью Datadok и внешним миром осуществляет Pentium III 450MHz с 384MB RAM. В моменты наблюдений 'idle' составляло не меньше 96%.

Пакетный фильтр? Межсетевой кран?

Ранее по тексту я уже использовал некоторые термины без дачи их определений и вскоре я исправлю это упущение.

PF работает в мире, который состоит из пакетов, протоколов, подключений и портов.

Анализируя порт иточника или назначения, используемый протокол и адреса PF принимает решение о том, куда пропускать пакет и пропускать ли его вообще.

Также возможна фильтрация проходящего сетевого трафика, основанного на содержании пакета, обычно называемом фильтрацией прикладного уровня, но эта функция не является отличительной особенностью PF. В дальшейшем вы увидите, как PF будет делегировать такие задачи стороннему программному обеспечению. Но сперва кратко рассмотрим теоретическую часть.

Мы уже упомянули понятие системы сетевой защиты(firewall). Одна важная особенность PF и ему подобного программного обеспечения заключается в том, что оно способно идентифицировать и блокировать трафик, который из вашей сети наружу или входит в вашу сеть.

NAT?

Другая вещь, о которой мы будем много говорить - это "внутренние" и "внешние" адреса, "маршрутизируемые" и "не маршрутизирумые". Этот вопрос не связан непосредственно с системами сетевой защиты и фильтрации пакетов, но мы вынуждены будем немного его коснуться. Все началось в в начале 1990-ых, когда кто-то подсчитал, сколько всего возможно пользователей Интернета.

Во времена разработки системы адресации компьютеры были большими, очень дорогими и обычно обслуживали большое количество пользователей. Тогда подключенными к сети были лишь университеты и компании с заказами Пентагона. По сути дела 32 битная адресация в 4 октетах позволила бы подключить миллионы машин.

Но тут случилась коммерциализация Интернет и в сеть попали сотни тысяч дешевых маленьких машин, в результате чего адресное пространство стало таять с катастрофической скоростью. Для решения этой проблемы был разработан протокол IP version 6, если коротко - IPv6, использующий 128 битную адресацию. OpenBSD по умолчанию поддерживает IPv6 и PF способен работать с трафиком IPv6.

Кроме того, необходимо было временное решение, так как перевод сети на новую адресацию занял бы значительное время. Найденное решение состояло из двух частей. Одной частью был механизм, позволяющий

перезаписывать адреса источника/назначения налюзе. Другой частью выделялись участки адресного пространства, которые будут использоваться только в сетях, непосредственно не соединенных с Интернетом. Это подразумевает то, что в разных частях света некоторые машины могут иметь одинаковый адрес, но прежде чем их трафик попадет в сеть, эти адреса будут оттранслированы маршрутизаторами.

Если трафик с такого "не маршрутизируемого" адреса захотел бы попасть в Интернет, то маршрутизаторы должны были бы отбрасывать его, как имеющий недопустимый адрес источника.

Это - то, что называют "Трансляцией сетевых адресов"(NAT), иногда упоминается "подмена IP адресов", маскарадинг или нечто подобное.

Узнать диапазоны не маршрутизитуемых сетей можно в "RFC 1918 addresses".

Внимание:

Два документа определяют работу NAT: RFC 1631, "The IP Network Address Translator (NAT)", датированный маем 1994 и RFC 1918, "Address Allocation for Private Internets", ддатированный февралем 1996.

PF сегодня

К этому моменту мы прошли некоторую подготовку и можем непосредственно приступить к рассмотрению основного вопроса. Прошло несколько лет с момента выхода PF и теперь, в составе OpenBSD 3.6 PF стал пакетным фильтром, способным на очень многие вещи.

С одной стороны, PF классифицирует пакеты, основываясь на типе протокола, порта, типе пакета, источнике или адресе назначения и, с некоторой долей уверенности, на исходной операционной системе.

И даже если NAT не является самой необходимой частью пакетного фильтра, по практическим соображениям все же в функции PF включена логика NAT.

PF способен на основе протокола, адреса и других данных пропускать трафик адресатам, таким как удаленные машины или сервисные службы.

Еще до написания PF, OpenBSD содержал код altq для балансировки нагрузки и гибкого управления трафиком, в проследствии они были интегрированы между собой.

В результате этого, все возможности становятся вам доступны через один единственный, легко читаемый файл конфигурации, обычно называемый pf.conf и находящийся в каталоге /etc.

В настоящее время PF доступен как составная часть OpenBSD, в FreeBSD, начиная с 5.3, PF одна из трех доступных систем пакетной фильтрации, PF также доступен в NetBSD и DragonFlyBSD. Непосредственную работу с двумя последними системами я не вел, в виду ограниченности ресурсов, по этому с радостью услышу ваши впечатления.

BSD vs Linux - Конфигурация

Я предполагаю, что нынешние читатели больше знакомы с одним из дистрибутивов Linux, чем с семейством BSD и поэтому кратко упомяну ключевые отличия.

В BSD системах сетевые интерфейсы не обозначаются как eth0 и т.д. Имя каждого интерфейса соответствует имени драйвера устройства и его порядковому номеру, например сетевые карты 3Com, использующие драйвер xl обозначаются как xl0, xl1 и т.д.

Дистрибутивы BSD организованы таким образом, чтобы читать конфигурацию из файла /etc/rc.conf, который читается сценарием/etc/rc при запуске. OpenBSD рекомендует использовать /etc/rc.conf.local для локальных настроек, так как rc.conf содержит значения по умолчанию, в то время как FreeBSD использует /etc/defaults

/rc.conf, чтобы сохранить настройки по умолчанию, используя /etc/rc.conf для хранения пользовательских настроек.

PF конфигурируется редактированием /etc/pf.conf и использованием утилиты командной строки pfctl. pfctl имеет **очень** много опций. Сегодня мы рассмотрим только наиболее используемые.

Если вас интересует вопрос наличия web-интерфейса для управления PF, то предупреждаем, что они не являются частью основной системы. Создатели PF не против наличия таких средств, но они не представляют пока себе графического интерфейса для редактирования pf.conf, передачи опций pfctl и еще некоторых особенностей unix.

Самый простой вариант - одиночная машина (OpenBSD)

Ну, вот мы и подошли к установке PF на выделенную машину в самой простой конфигурации. По условиям вводной машина непосредственно подключена к Интернет.

Для запуска PF при старте системы необходимо внести следующую строку в /etc/rc.conf.local:

```
pf=YES                  # enable pf
```

Очень просто. Там же вы можете указать местоположение файла конфигурации:

```
pf_rules=/etc/pf.conf # specify which file contains your rules
```

После перезагрузки PF должен будет выдать на консоль сообщение "PF enabled", что говорит об активизации пакетного фильтра. Файл конфигурации /etc/pf.conf, устанавливаемый вместе с OpenBSD или FreeBSD, содержит множество полезных примеров, но они все прокомментированы.

Если мы не хотим перегружаться, то можно использовать утилиту pfctl:

```
peter@skapet:~$ sudo pfctl -e ; sudo pfctl -f /etc/pf.conf
```

Внимание:

В качестве отступления от темы, хочу указать, что при необходимости повышения привелегий я использую утилиту **sudo**. Она доступна в коллекции портов BSD систем или поставляется с базовой системой.

Самый простой вариант - одиночная машина (FreeBSD)

В операционной системе FreeBSD вам необходимо внести несколько больше изменений в /etc/rc.conf, руководствуясь FreeBSD Handbook:

```
pf_enable="YES"                 # Enable PF (load module if required)
pf_rules="/etc/pf.conf"         # rules definition file for pf
pf_flags=""                     # additional flags for pfctl startup
pflog_enable="YES"              # start pflogd(8)
pflog_logfile="/var/log/pflog"  # where pflogd should store the logfile
pflog_flags=""                  # additional flags for pflogd startup
```

В FreeBSD, PF по умолчанию скомпилирован как загружаемый модуль ядра. Это означает, что для запуска PF вы должны выполнить команды:

```
$ sudo kldload pf
$ sudo pfctl -e
```

Теперь мы нуждаемся в некотором наборе правил фильтрации. Сейчас мы приведем пример самого простого варианта, для машины, на которой не выполняются никакие сервисы, которая подключена к локальной сети или сразу в Интернет. Итак, наш etc/pf.conf пока будет выглядеть так:

```
block in all
pass out all keep state
```

Этими правилами будет запрещен любой входящий трафик, разрешен наш исходящий и пропущен входящий трафик, являющийся ответным на наши запросы. Если вы готовы к использованию такого набора правил, то загружаем их:

```
$ sudo pfctl -e
$ sudo pfctl -f /etc/pf.conf
```

Чуть более сложный

Для написания несколько более структурированной и законченной системы правил фильтрации мы запретим весь трафик, чтобы впоследствии разрешить только необходимый. Делать это мы будем используя отличительные особенности PF: списки и макросы.

Теперь наш /etc/pf.conf будет начинаться так:

```
block all
```

Макросы требуют определения перед использованием:

```
tcp_services = "{ ssh, smtp, domain, www, pop3, auth, pop3s }"
udp_services = "{ domain }"
```

Теперь мы можем продемонстрировать сразу несколько вещей - то, что макрос может быть списком и то, что PF понимает в качестве номера порта название службы. Именование служб берется из /etc/services. Теперь наши правила будут выглядеть следующим образом:

```
block all
pass out proto tcp to any port $tcp_services keep state
pass proto udp to any port $udp_services keep state
```

Здесь, особо образованные из наших читателей могут указать, что протокол UDP не предусматривает обратной связи, но PF, не смотря на это, может обработать такие соединения. Например, вы, скорее всего, захотите принять ответ от сервера имен DNS.

После выполненных изменений нам необходимо перезагрузить правила:

```
$ sudo pfctl -f /etc/pf.conf
```

Если нет никаких синтаксических ошибок, то pfctl не выдаст никаких сообщений. Установленный флаг -v укажет pfctl сделать более подробный вывод.

Статистика из pfctl

В процессе работы PF возможен просмотр некоторой статистики, это делается с помощью утилиты pfctl. Для этого используется флаг "-s" и тип отображаемой информации.

Следующий пример взят с моего домашнего шлюза, в то время как я готовил эту лекцию:

```
peter@skapet:~$ sudo pfctl -s info
Status: Enabled for 6 days 01:30:14           Debug: Urgent

Hostid: 0x9c6b095b

Interface Stats for xl0               IPv4             IPv6
  Bytes In                       431807046                0
  Bytes Out                       40105602              352
  Packets In
    Passed                          362534                0
    Blocked                          45033                0
  Packets Out
    Passed                          285888                1
    Blocked                              1                4

State Table                          Total             Rate
  current entries                        7
  searches                         1026325            2.0/s
  inserts                            26577            0.1/s
  removals                           26570            0.1/s
Counters
  match                              48962            0.1/s
  bad-offset                             0            0.0/s
  fragment                              10            0.0/s
  short                                 20            0.0/s
  normalize                              0            0.0/s
  memory                                 0            0.0/s
  bad-timestamp                          0            0.0/s
```

В первой строке указывается, что PF работает уже несколько дней, отсчет ведется со времени последней перезагрузки. pfctl -s all даст вам еще более детальную информацию. Для получения дополнительных сведений, обратитесь к man 8 pfctl.

К настоящему моменту мы имеем одну единственную машину, защищенную достаточно для подключения к Интернет.

Нам не хватает еще несколько вещей. Например, все же стоило разрешить проходить некоторому icmp и udp трафику, например, для работы утилит ping и traceroute.

Хотя в настоящее время и доступны более современные и безопасные сервисы, нам скорее всего потребуется нормальная работа ftp.

Вскоре мы вернемся к рассмотрению этих вопросов.

Простой маршрутизатор с NAT

Сейчас мы подошли к рассмотрению более реалистичных и распространенных примеров, где машина с установленной системой сетевой защиты является маршрутизатором для машин локальной сети в сеть Интернет. Если на машинах локальной сети также установлены системы пакетной фильтрации, в нашем примере мы это не учитываем.

Мы предполагаем, что в машину была установлена дополнительная сетевая плата или организовано подключение к внешней сети через дополнительные средства, такие как PPP. В данной лекции мы не будем привязываться к конкретному интерфейсу, поэтому тип подключения не так важен. Для примеров приведенных ниже, в зависимости от типа соединения будет меняться только обозначение интерфейса ppp или ethernet.

Первым делом нам необходимо подготовить машину к транзиту трафика, то есть чтобы трафик принимаемый на один интерфейс мог быть перенаправлен в другую сеть через другой интерфейс. Первоначально мы сделаем это с помощью утилиты sysctl для протокола IPv4:

```
# sysctl net.inet.ip.forwarding=1
```

Для включения маршрутизации протокола IPv6 необходимо выполнить:

```
# sysctl net.inet6.ip6.forwarding=1
```

Для того, чтобы сделать эти изменения постоянными внесем следущие изменения в /etc/sysctl.conf (OpenBSD):

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

Или в /etc/rc.conf (FreeBSD):

```
gateway_enable="YES" #for ipv4
ipv6_gateway_enable="YES" #for ipv6
```

Для FreeBSD необходимые параметры можно выставить и с помощью sysctl, но более предпочтительным было бы использование файла rc.conf.

Для просмотра параметров интерфейсов, которые вы планируете использовать, служит команда ifconfig -a:

Если вы намереваетесь пропускать трафик из вашей внутренней сети наружу, то ваш /etc/pf.conf будет выглядеть следующим образом:

```
ext_if = "xl0"  # macro for external interface - use tun0 for PPPoE
int_if = "xl1"  # macro for internal interface
# ext_if IP address could be dynamic, hence ($ext_if)
nat on $ext_if from $int_if:network to any -> ($ext_if)
block all
pass from { lo, $int_if:network } to any keep state
```

Обратите внимание на использование макросов для определения сетевых карт. Здесь используются сетевые карты 3Com, но в дальнейшем мы убедимся, что тип интерфейса не играет особой роли. Использование макросов позволяет абстрагироваться от типа интерфейса, сетевой карты или ее адреса.

Обратите внимание на правило NAT. В этом правиле мы транслируем "немаршрутизируемые" адреса нашей локальной сети в один внешний адрес сетевого интерфейса нашего маршрутизатора.

Использование выражения ($ext_if) позволяет нам использовать правило в случае, если адрес назначается динамически.

С другой стороны, это позволит проходить большему трафику, нежели могло хотеться. В моем случае используется макрос:

```
client_out = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, \
                https, 446, cvspserver, 2628, cvsup, 8000, 8080 }"
```

Совместно с правилом:

```
pass inet proto tcp from $int_if:network to any port $client_out \
     flags S/SA keep state
```

Выбор портов может показаться довольно специфичным, но это то, что необходимо для работы мне и моим друзьям. Ваши потребности могут отличаться и требовать разрешения других номеров портов.

Кроме того, у нас имеется в запасе еще несколько интересных правил разрешения трафика, их мы покажем ниже. Например, для доступа к нашей машине из интернета по ssh:

```
pass in inet proto tcp from any to any port ssh
```

или так:

```
pass in inet proto tcp from any to $ext_if port ssh
```

Разрешим работу службы DNS. Определим макрос:

```
udp_services = "{ domain, ntp }"
```

И добавим само правило:

```
pass quick inet proto { tcp, udp } to any port $udp_services keep state
```

Обратите внимание на ключевое слово **quick**. Если пакет соответствует правилу с этим ключевым словом то пакет прекращает дальнейшее прохождение по цепочке правил и к нему применяется то действие, которое указано в правиле. Использование ключегого слова **quick** весьма удобно, когда вам требуется сделать несколько правил-исключений.

Вместе со службой DNS мы разрешим службу сетевого времени(NTP), которая используется для синхронизации системных часов. Отличительная особенность обоих протоколов заключается в том, что они посылают данные и по tcp и по udp.

Внимание:

Для пользователей модемного соединения, ADSL или PPP over Ethernet (PPPoE) внешним интерфейсом будет tun0, вместо интерфейса типа ethernet.

Старый грустный FTP

Оглавление:

В списке разрешенных портов мы можем увидеть порты протокола передачи файлов FTP. Этот протокол очень стар и полон загадок, которые вызывают массу проблем, в том числе и с безопасностью. Ключевыми проблемами использования этого протокола:

- Пароль передается открытым текстом
- Протокол требует создания не менее двух tcp подключений(для управления и данных) на разных портах
- Когда сеанс установлен, данные передаются через порты, выбранные наугад

Эти факторы осложняют построение системы сетевой защиты и вызывают проблемы при функционировании системы.

В настоящее время существуют более безопасные и удобные протоколы передачи файлов, такие как sftp или scp, которые обеспечивают идентификацию и передачу данных через зашифрованные соединения. Настоящие профессионалы IT должны предпочесть что-либо отличное от ftp.

Независимо от нашего профессионализма и предпочтения, мы знаем, что сталкиваться с этим протоколом нам придется неоднократно. В этом случае наша задача состоит в переадресации этого типа трафика специализированной программе, написанной для этой цели.

Ftp через NAT: ftp-proxy

ftp-proxy является частью базовой системы OpenBSD и других систем, содержащих PF. Обычно она вызывается "супер-сервером" inetd через конфигурацию /etc/inetd.conf. Приведенная ниже строка запустит ftp-proxy при работе NAT на кольцевом интерфейсе lo0:

```
127.0.0.1:8021 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy -n
```

Эта строка находится по умолчанию в вашем inetd.conf, но она прокомментирована символом # в начале строки. Для того, чтобы изменения вступили в силу, перезапустите сервис inetd. На *BSD дистрибутивах это делается командой:

```
FreeBSD$ sudo /etc/rc.d/inetd restart
```

или аналогичной. Если вы не уверены, то обратитесь к man 8 inetd. В OpenBSD можно сделать таким образом:

```
OpenBSD$ sudo kill -HUP `cat /var/run/inetd.pid`
```

С этого момента inetd выполняется с новыми параметрами настройки.

Теперь разберем реальный случай перенаправления. Правила перенаправления и NAT относятся к одному классу. На эти правила можно сослаться непосредственно другими правилами, и фильтрующие правила могут зависеть от этих правил. Логически, rdr и правила NAT должен быть определен перед правилами фильтрации.

Мы вставляем наше правило rdr сразу после правила NAT в нашем /etc/pf.conf:

```
rdr on $int_if proto tcp from any to any port ftp -> 127.0.0.1 \
        port 8021
```

Также необходимо добавить правило, разрешающее прохождение перенаправленного трафика:

```
pass in on $ext_if inet proto tcp from port ftp-data to ($ext_if) \
   user proxy flags S/SA keep state
```

Сохраним pf.conf и загрузим новые правила:

```
$ sudo pfctl -f /etc/pf.conf
```

После таких вот нехитрых действий ftp должен прекрасно работать.

Этот пример предполагает, что Вы используете трансляцию сетевых адресов (Network Address Translation) на шлюзе и внутри локальной сети используете адреса из диапазона частных сетей.

Ftp через pf с маршрутизируемыми адресами: ftpsesame

В случаях, когда локальная сеть использует официальные, маршрутизируемые адреса, то могут определенные проблемы в защите ftp-соединений. Так как такие случаи достаточно редки, то я не буду их рассматривать, а прямиком направлю вас на http://www.sentia.org/projects/ftpsesame/. ftpsesame присоединяется к набору правил через якорь, называемый sub-ruleset. Документация включает в себя страницу руководства man и множество примеров, которые вы можете использовать.

Решение проблем - утилиты ping и traceroute

Написанный нами набор правил имеет один существенный недостаток - не работают средства диагностики и поиска неисправностей в сети, а именно утилиты ping и traceroute. Это не будет иметь значение для большинства ваших пользователей, особенно работающих с Windows и тех, кто считает, что утилита ping - опасная команда и что icmp должен быть вне закона. Но вам скорее всего эти утилиты будут нужны. Для успешного решения это проблемы выполним несколько простых шагов. Добавляем макрос:

```
icmp_types = "echoreq"
```

И соответствующее правило:

```
pass in inet proto icmp all icmp-type $icmp_types keep state
```

Если вам необходимо попускать и другие типы icmp пакетов, то добавьте их в макрос icmp_types.

traceroute - другая команда, которая является весьма полезной, когда ваши пользователи утверждают, что Internet не работает. Правило ниже работает с командой traceroute на всех Unix, к которым я имел доступ, включая GNU/Linux:

```
# allow out the default range for traceroute(8):
# "base+nhops*nqueries-1" (33434+64*3-1)
pass out on $ext_if inet proto udp from any to any \
            port 33433 >< 33626 keep state
```

Если в каких-либо других операционных системах это правило не будет работать, то вам придется

откорректировать его. Это решение было найдено в списке рассылки openbsd-misc и в случаях проблем с PF я советую вам обратиться к архиву рассылки на http://marc.theaimsgroup.com/.

Работа с внутренними web и почтовыми серверами

Проходит время и нам необходимо внести изменения в наши правила. Довольно обычной является ситуация, когда необходимо предоставлять внешние сервисы, а свободных маршрутизируемых адресов дря размешения серверов попросту нет, а выполнение на одной машине нескольких сервисов не самое безопасное решение.

Механизмы перенаправления в PF довольно просты и позволяют легко разместить серверы внутри локальной сети. Если мы планируем разместить в локальной сети сервера http, https и почтовый сервер для приема и отправки почты, то этот отрезок списка правил будет выглядеть так:

```
webserver = "192.168.2.7"
webports = "{ http, https }"
emailserver = "192.168.2.5"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

rdr on $ext_if proto tcp from any to any port $webports -> $webserver
rdr on $ext_if proto tcp from any to any port $email -> $emailserver

pass in on $ext_if proto tcp from any to $webserver port 80 \
   flags S/SA synproxy state
pass in on $ext_if proto tcp from any to $emailserver port $email \
   flags S/SA synproxy state
pass out on $ext_if proto tcp from $emailserver to any port smtp \
   flags S/SA synproxy state
```

Обратите внимание на флажок "synproxy" в новых правилах. Это означает, что pf будет перехватывать входящие соединения и отправлять их на внутренние сервера, выполняя роль прокси-сервера.Это обеспечивает некоторую защиту против нескольких типов атак.

Правила, описывающие работу с демилитаризованной зоной(DMZ), когда локальная сеть отделаяется от сети, в которой размещены серверы, не будут сильно отличаться от вышеприведенных, достаточно только пределить интерфейсы локальной сети и сети DMZ.

Таблицы, делающие вашу жизнь легче

К моменту прочтения этой главы вы можете придти к мысли, что написанные правила недостаточно гибки, ведь могут быть случаи, когда определенный трафик должен быть разрешен или заблокирован, но нет желания вносить правило в основной список правил фильтрации. PF предлагает очень удобный механизм для решения этой проблемы. Таблица - это структура, главным образом полезная как список правил, которые могут управляться, не вызывая перезагрузки основного набора правил. В нет также можно осуществлять быстрый поиск. Имена таблиц всегда включаются в < >, подобно этому:

```
table <clients> { 192.168.2.0/24, !192.168.2.5 }
```

Здесь в таблице содержатся адреса сети 192.168.2.0/24 за исключением адреса 192.168.2.5, что указано оператором !(логическое NOT). Также возможна загрузка таблицы из выделенного файла, например /etc/clients, значения в котором указываются по одному в строке:

```
192.168.2.0/24
!192.168.2.5
```

Имя файла в свою очередь используется, чтобы инициализировать таблицу в /etc/pf.conf:

```
table <clients> persist file /etc/clients
```

Тогда, например, Вы можете заменить одно из наших более ранних правил на следущее:

```
pass out inet proto tcp from  to any port $client_out \
         flags S/SA keep state
```

Имея в руках такой инструмент, вы можете оперативно изменять правила фильтрации. Например, для редактирования таблицы достаточно воспользоваться командой:

```
$ sudo pfctl -t clients -T add 192.168.1/16
```

Вы могли бы решить сохранять копию таблицы на диск, используя планировщик заданий cron. В качестве альтернативы, вы можете редактировать файл /etc/clients и заменить им находящуюся в памяти содержание таблицы с данными файла:

```
$ sudo pfctl -t clients -T replace -f /etc/clients
```

Для выполнения рутинных действий по добавлению/удалению элементов таблиц советую вам написать соответствующие скрипты. Единственные реальные ограничения лежат в ваших собственных потребностях и вашем творческом потенциале.

Вскоре мы вернемся к другим примерам использования таблиц.

Журнал событий

до сих пор мы не слова не сказали о регистрации событий. PF позволяет вести регистрацию с помощью ключевого слова "log", указываемого после интересующего нас правила. Для ограничения обьема регистрируемых данных можно также указать интересующий нас сетевой интерфейс. Сделать это можно правилом:

```
set loginterface $ext_if
```

И соответствующим образом отредактировать правила:

```
pass out log from  to any port $email \
         label client-email keep state
```

В результате этого, весь трафик, подходящий под это правило будет зарегистрирован в формате вывода утилиты tcpdump. Это может быть довольно удобно при а=выставлении счета за использованный трафика.

Можно было бы разместить такое правило:

```
block log all
```

Для того, чтобы удостовериться что вы не пропускаетет ничего лишнего. Руководство пользователя PF содержит детальное описание того, как привести журнальный файл к читаемому текстовому формату через syslog и это действительно звучит довольно привлекательно. Регистрация событий - очень полезная функция, но она в любом случае должна быть выборочной, так как есть опасность неконтролируемого разрастания файлов журналов.

Следим за всем с помощью pftop

Если вы интересуетесь, что и как ходит по сети, то поможет утилита pftop Эркина Акара. Название довольно символично, поскольку внешний вид очень похож на вывод программы top.

```
pfTop: Up State 1-21/67, View: default, Order: none, Cache: 10000       19:52:28

PR   DIR SRC                  DEST                 STATE   AGE   EXP  PKTS BYTES
tcp  Out 194.54.103.89:3847   216.193.211.2:25       9:9     28    67    29  3608
tcp  In  207.182.140.5:44870  127.0.0.1:8025         4:4     15 86400    30  1594
tcp  In  207.182.140.5:36469  127.0.0.1:8025        10:10   418    75   810 44675
tcp  In  194.54.107.19:51593  194.54.103.65:22       4:4    146 86395   158 37326
tcp  In  194.54.107.19:64926  194.54.103.65:22       4:4    193 86243   131 21186
tcp  In  194.54.103.76:3010   64.136.25.171:80       9:9    154    59    11  1570
tcp  In  194.54.103.76:3013   64.136.25.171:80       4:4      4 86397     6  1370
tcp  In  194.54.103.66:3847   216.193.211.2:25       9:9     28    67    29  3608
tcp  Out 194.54.103.76:3009   64.136.25.171:80       9:9    214     0     9  1490
tcp  Out 194.54.103.76:3010   64.136.25.171:80       4:4     64 86337     7  1410
udp  Out 194.54.107.18:41423  194.54.96.9:53         2:1     36     0     2   235
udp  In  194.54.107.19:58732  194.54.103.66:53       1:2     36     0     2   219
udp  In  194.54.107.19:54402  194.54.103.66:53       1:2     36     0     2   255
udp  In  194.54.107.19:54681  194.54.103.66:53       1:2     36     0     2   271
```

Подключения можно вывести отсортированыым по различным критериям, таким как правил, обьему, возрасту и т.д.

Эта утилита не входит в базовый комплект поставки, но в OpenBSD и FreeBSD она может быть установлена из системы портов - /usr/ports/sysutils/pftop.

Невидимый маршрутизатор - bridge

Бридж, в нашем контексте, машина с двумя или больше сетевыми интерфейсами, расположенная между Интернет и внутренней сетью/сетями, причем интерфейсам не назначены IP адреса. Если на рассматриваемой машине установлена OpenBSD или другая операционная система с поддержкой PF, то все еще остается возможность фильтрации и переадресации трафика. Преимущество такой схемы очевидно - становится значительно труднее атаковать систему сетевой защиты. Недостаток - невозможность удаленного управления и конфигурирования, если вы только не назначите адрес интерфейсу, находящемуся внутри защищенной сети.

Ниже приведен пример для операционной системы OpenBSD, учтите, что если у вас установлена другая ОС, то процесс настройки может измениться. В системе установлено два сетевых интерфейса.

```
/etc/hostname.xl0

up
```

```
/etc/hostname.xl1

up

/etc/bridgename.bridge0

  add xl0 add xl1 blocknonip xl0 blocknonip xl1 up

/etc/pf.conf

ext_if = xl0
int_if  = xl1
interesting-traffic = { ... }
block all
pass quick on $ext_if all
pass log on $int_if from $int_if to any port $interesting-traffic \
     keep state
```

Возможны и более затейливые конфигурации. Опытные люди рекомендуют при использовании моста выполнять фильтрацию и перенаправление только на одном интерфейсе, так как при использовании нескольких интерфейсов правила PF будут довольно сложны.

В дополнение, команда brconfig, входящая в состав OpenBSD предлагает собственный механизм фильтрации. Для получения дополнительной информации, обратитесь к страницам руководства bridge(4) и brconfig(8).

Управление трафиком с помощью altq

ALTQ - сокращение от ALternate Queueing, являющегося очень гибким средством перенаправления, приоритезации и балансировки сетевого трафика и существовавшего отдельным механизмом до интеграции в PF.

Altq использует понятие "очереди"(queue) в качестве главного механизма контроля трафика. Очереди определяются определенной полосой пропускания, выраженной в неком количестве от общей полосы канала и может состоять из подочередей различных типов.

Для красоты картины фильтрации пакетов мы пишем правила фильтрации, которые направляют пакеты на указанные очереди или набор подочередей, где пакеты обрабатываются согласно указанным критериям.

Очереди базируются на классах(CBQ), что на практике означает определение полосы пропускания очереди как количество данных в секунду или процентах и указание приоритета . Приоритеты могут иметь значение от 0 до 7 для cbq очередей и от 0 до 15 для priq очередей, Чем выше значение, тем выше приоритет. Упрощенный синтаксис выглядит следующим образом:

```
altq on interface type [options ... ] main_queue { sub_q1, sub_q2 ..}
  queue sub_q1 [ options ... ]
  queue sub_q2 [ options ... ]
[...]
pass [ ... ] queue sub_q1
pass [ ... ] queue sub_q2
```

Если Вы будете использовать эти возможности в собственных наборах правил, вы должны при любых

обстоятельствах прочитать страницу руководства man pf.conf и руководство пользователя PF. Эти документы содержат очень детальное и доходчивоее объяснение синтаксиса.

ALTQ - распределение пропускной способност

Теперь приведем пример из жизни. Очереди установлены на внешнем интерфейсе. Это, по всей видимости обычный подход, так как ограничения на полосу пропускания на внешнем интерфейсе более неблагоприятны. Хотя принципе, очереди распределения полосы и балансировки трафика можут быть сделаны на любом сетевом интерфейсе. В этом примере правила включают в себя cbq очередь для полной полосы пропускания 640 КБ с шестью sub очередями.

```
altq on $ext_if cbq bandwidth 640Kb queue { def, ftp, udp, http, \
         ssh, icmp }
queue def bandwidth 18% cbq(default borrow red)
queue ftp bandwidth 10% cbq(borrow red)
queue udp bandwidth 30% cbq(borrow red)
queue http bandwidth 20% cbq(borrow red)
queue ssh bandwidth 20% cbq(borrow red) { ssh_interactive, ssh_bulk }
queue ssh_interactive priority 7
queue ssh_bulk priority 0
queue icmp bandwidth 2% cbq
```

Мы видим, что определенная по умолчанию очередь имеет 18 процентов полосы пропускания, в нее будет направляться весь трафик, не подходящий под остальные очереди. Ключевые слова borrow и red подразумевают, что очередь может'заимствовать' полосу пропускания от родительской очереди, в то время как система пытается избегнуть перегрузки, применяя алгоритм RED (Random Early Detection). Другие очереди следуют более или менее тому же образцу, за исключением подочереди ssh, которая имеет две подочереди с индивидуальными приоритетами.

Наконец, правила, в которых указывается, какой трафик назначен в очереди и их критерии:

```
pass log quick on $ext_if proto tcp from any to any port 22 flags S/SA \
    keep state queue (ssh_bulk, ssh_interactive)
pass in quick on $ext_if proto tcp from any to any port 20 flags S/SA  \
    keep state queue ftp
pass in quick on $ext_if proto tcp from any to any port 80 flags S/SA \
    keep state queue http
pass out on $ext_if proto udp all keep state queue udp
pass out on $ext_if proto icmp all keep state queue icmp
```

Мы можем предположить, что такое распределение удовлетворяет потребности сайта.

ALTQ - приоритезация трафика различных типов

Мы переходим к другому примеру, взятому из сети Дэниела Хартмеира. Подобно довольно многим из нас, Дэниел подключен по ADSL , и естественно он хотел бы использовать полосу пропускания более эффективно.

Очевидным путем к достижению совершенства будет уменьшение входящего трафика, занимающего полосу пропускания.

Анализ данных показал, что пакеты ACK для каждого пакета переданных данных вызвали непропорционально большое замедление, возможно из-за очереди FIFO (First In, First Out) в исходящем трафике. То есть,

складывалась ситуация, когда запросы на новые пакеты проходили не самым оптимальным способом.

Сформированная рабочая гипотеза состояла в том, что если бы между большими пакетами данных могли бы проходить маленькие пакеты ACK, то полоса пропускания использовалась бы более эффективно. Для решения были созданы две очереди с разными приоритетами. Вот образцы правил:

```
ext_if="kue0"

altq on $ext_if priq bandwidth 100Kb queue { q_pri, q_def }
queue q_pri priority 7
queue q_def priority 1 priq(default)

pass out on $ext_if proto tcp from $ext_if to any flags S/SA \
        keep state queue (q_def, q_pri)

pass in  on $ext_if proto tcp from any to $ext_if flags S/SA \
        keep state queue (q_def, q_pri)
```

В результате работа канала действительно улучшилась. Статья Дэниела доступна на его домашней страничке http://www.benzedrine.cx/ackpri.html

ALTQ - обработка нежелательного трафика

Наш последний пример использования altq обусловлен событиями последнего времени - проблема спама и вирусных штормов. Практически все машины, зараженные вирусами и троянскими конями работают под управлением ОС Windows. PF имеет довольно точный механизм определения удаленной операционной системы и однажды, один из пользователей OpenBSD написал несколько правил для фильтрации лишнего трафика с зараженных машин:

```
altq on $ext_if cbq queue { q_default q_web q_mail }

   queue q_default cbq(default)
   queue q_web (...)

    ## all mail limited to 1Mb/sec
    queue q_mail bandwidth 1Mb { q_mail_windows }
    ## windows mail limited to 56Kb/sec
    queue q_mail_windows bandwidth 56Kb

    pass in quick proto tcp from any os "Windows" to $ext_if port 25 \
            keep state queue q_mail_windows
    pass in quick proto tcp from any to $ext_if port 25 label "smtp" \
            keep state queue q_mail
```

Эти правила написаны Randal L. Schwartz, 29. januar 2004, http://use.perl.org/~merlyn/journal/17094

В этом примере для почтового трафика выделена полоса пропускания в 1mb/sec, в то время, как почтовый трафик, отправленный с Windows машин ограничивается 56Kb/sec.

Я должен признать, что это достаточно смелый шаг, готорый я всегда хотел сделать, но не смел. Скоро мы рассмотрим другой подход, позволяющий достигнуть почти такого же эффекта.

CARP и pfsync

CARP и pfsync - это два новшества, появившееся в OpenBSD 3.5. CARP - сокращение от Common Address Redundancy Protocol. Он был разработан как альтернатива VRRP (Virtual Router Redundancy Protocol, RFC 2281, RFC 3768), который был весьма близок к стандартизации IETF.

Оба протокола предназначены для создания избыточности сетевых соединений с возможностью автоматического восстановления.

CARP основан на установке группы машин, одна из которых является "главной" и одна или несколько "подчиненной". При падении главной машины одна из подчиненных берет ее IP адрес и, если была правильно настроена синхронизация, активные подключения.

Одна из основных целей CARP заключается в обеспечении функционирования работоспособности сети даже в случае отказа системы сетевой защиты или плановых остановок системы.

При использовании PF pfsync поможет вам разрешить проблему синхронизации машин группы. pfsync - это разработанный специально для обмена между пакетными фильтрами PF виртуальный интерфейс. Интерфейсы pfsync назначаются на физические интерфейсы с помощью команды ifconfig. В сетях, где требования к безотказной работе высоки, число одновременных подключений будет достаточно большим, поэтому будет иметь смысл выделение под pfsync отдельных сетевых интерфейсов.

Более подробно о CARP я расскажу в ближайшем будущем. Для получения дополнительной информации обратитесь к документации OpenBSD, руководствам man и домашней странице Рьяна Макбрайд http://www.countersiege.com/doc/pfsync-carp/

Тяжелое врямя для спамера

Рассмотрев некоторые основные моменты я рад представить вам нечто действительно полезное: PF, как средство сделать жизнь спамера несколько более тяжелой. Базируясь на недавно узнанном пишем правила:

```
table  persist
table  persist file "/var/mail/whitelist.txt"
rdr pass on $ext_if inet proto tcp from  to \
         { $ext_if, $int_if:network } port smtp -> 127.0.0.1 port 8025
rdr pass on $ext_if inet proto tcp from ! to \
         { $ext_if, $int_if:network } port smtp -> 127.0.0.1 port 8025
```

Мы имеем две таблицы, одна из них бередданне из некоторого файла. Трафик от серверов SMTP, находящихся в первой таблице и адреса из другой таблицы переадресовываются к демону, слушающего порт 8025.

Отправная точка, лежащая в основе дизайна spamd - тот факт, что спамеры посылают большое количество сообщений и вероятности, что Вы являетесь первым человеком, принявшим некое сообщение, является невероятно маленькой. Кроме того, спам главным образом посылают через некоторое количество дружественных к спамерам сетей и большое количество зараженных машин. Черные списки зараженных машин и сигнатур сообщений будут пополняться довольно быстро.

Конфигурируется spamd довольно просто. Вы просто редактируете /etc/spamd.conf согласно вашим собственным потребностям. Сам файл некоторым комментарием, на странице руководства man имеется дополнительная информация. Предложенное значение по умолчанию помещает в черный список весьма немного, включая корейские адреса. Я работаю в компании, которая работает с фирмами из Кореи, поэтому из моего файла конфигурации эти адреса удалены. Вы самостоятельно можете указывать, какие черные

списки использовать и что для вас будет источником данных.

Поместите строки для старта spamd и укажите параметров запуска, которые Вы хотите, в вашем /etc/rc.conf или /etc/rc.conf.local. После этого, запустите spamd с указанными опциями и завершите конфигурацию, используя spamd-setup. Наконец, вы создаете cron задание, которое обновляет таблицы в разумных интервалах.

Как только таблицы заполнены, Вы можете управлять их содержанием, используя pfctl, точно так же как любые другие таблицы.

Также вам необходимо написать правила, разрешаюшие почтовый трафик, если у вас в сети раньше уже использовалась почтовая служба, то можно оставить старые правила.

Как практически применяктся spamd? Мы начали использовать spamd в начале декабря 2004 года, после того, как уже некоторое время эксплуатировали spamassassin и clamav в связке с exim. Наш exim был настроен таким образом, что тегирование и доставка осуществляется при установленном spamassassin весе сообщения в интервале от 5 до 9.99 и отбрасывании при весе большем 10. Но в связи с ростом обьема спама, проходить его стало значительно больше.

При введении в строй spamd общее количество обрабатываемых spamassassin сообщений резко уменьшилось. Количество пропущенного спама сейчас составляет примерно пять сообщений в день.

Типичный лог выглядит так:

```
Jan 26 18:52:32 delilah spamd[30816]: 163.125.69.60: connected (4/4),
lists: spamhaus
Jan 26 18:54:14 delilah spamd[30816]: 163.125.69.60: disconnected
after 102 seconds. lists: spamhaus
Jan 26 18:55:28 delilah spamd[30816]: 207.182.142.209: disconnected
after 403 seconds. lists: spamhaus spews1
Jan 26 18:56:03 delilah spamd[30816]: 192.112.102.8: connected (3/2)
Jan 26 18:56:04 delilah spamd[30816]: 192.112.102.8: disconnected
after 1 seconds.
```

В первой строке говорится, что машина соединяется, как четвертое активное подключение из четырех помещенных в черный список. Этот специфический адрес был помещен в черный список spamhaus. Следующие две строки показывают, что двум машинам, отказано в доставке после 1 минуты, 42 секунд и 6 минут, 43 секунд, соответственно. В предпоследней строке показано, как машины из черного списка все еще пытаются доставить почту.

По предварительному заключению, spamd блокирует некоторое количество спама. Но, к сожалению, есть и ложные срабатывания. Все они были связаны с использованием списка spews2 (spews level 2), в связи с чем его использование пришлось прекратить.

Теперь кульминационный момент моего опыта работы со spamd. Моя лучшая запись в логе выглядит так:

```
Dec 11 23:57:24 delilah spamd[32048]: 69.6.40.26: connected (1/1),
lists: spamhaus spews1 spews2
Dec 12 00:30:08 delilah spamd[32048]: 69.6.40.26: disconnected
after 1964 seconds. lists: spamhaus spews1 spews2
```

Речь здесь идет об отправителе в wholesalebandwidth.com. Эта машина сделала 13 попыток доставки почты в

период с 9-ого декабря до 12-ого декабря 2004. Последняя попытка продолжалась 32 минуты, 44 секунды.

Это меньше, чем 47 минут у Дэниела Хартмеира, но я доволен и этим результатом.

Подводя итог можно сказать, что использование черных списков совместно с spamd является мощным средством борьбы со спамом. Потребление ресурсов spamd - минимально. С другой стороны, надежность spamd равна надежности самого слабого элемента, что означает необходимость поддержания актуальности списков и контроль журнальных файлов.