iptables: правила хорошего тона

Jan Engelhardt (автор)*
Modified Version by Сергей Пташник (русский перевод)

2 января 2012 г.

Аннотация

Наборы правил для Linux-фаервола iptables, написанные новичками, очень часто оказываются не вполне оптимальными с точки зрения производительности. Кроме того, большие, плохо организованные наборы правил очень тяжело читать и отлаживать. Данный документ предлагает ряд полезных рекомендаций, позволяющих создавать удобочитаемые и эффективные наборы правил.

Предполагается, что читатель хорошо знаком с синтаксисом и опциями iptables.

*Первоисточник (на английском языке, оригинальное название «Towards the perfect ruleset») опубликован на сайте автора: http://jengelh.medozas.de/documents/Perfect_Ruleset.pdf

Содержание

1 Вступление

На наш IRC-канал (да и на всякие форумы) регулярно приходят новички и просят объяснить, почему их бессмысленные и беспощадные скрипты для iptables работают не так, как было задумано (именно скрипты, что ужасно само по себе). Абсолютное большинство этих творений страшно даже просто читать, не говоря уже об их эффективности в плане быстродействия.

Данный документ содержит набор советов и рекомендаций, позволяющих очистить ваш набор правил от мусора, и придать ему более-менее читабельный вид. Это упростит жизнь и вам, и тем, кто будет пытаться вам помочь в случае затруднений.

Здесь и далее под термином «iptables» подразумеваются iptables, ip6tables, arptables и ebtables, кроме тех случаев, когда явно указано обратное.

2 Отладка наборов правил

Пытаетесь понять, где теряются пакеты/какое именно правило не срабатывает? Попробуйте включить режим трассировки пакетов:

```
iptables -t raw -A PREROUTING [-m ...] -j TRACE
iptables -t raw -A OUTPUT [-m ...] -j TRACE
```

При этом убедитесь, что загружен соответствующий модуль логгирования: `ipt_LOG`/`ip6t_LOG` (запись в системный журнал) или `nfnetlink_log` (регистрация пакетов через специального демона, например, ulogd). В частности, для записи информации в системный журнал, нужно вызвать команду `modprobe` для соответствующего LOG-модуля.

3 Используйте iptables-restore

По состоянию на июнь 2009 г., iptables работает с ядерными модулями из семейства `ip_tables`, которые накладывают принципиальное ограничение — при изменении набора правил, каждая таблица рассматривается как единое целое, и для того, чтобы изменить одно правило, нужно перестраивать всю таблицу. Написанные в старом стиле скрипты многократно вызывают `/sbin/iptables` — при каждой установке правила по умолчанию (`iptables -P`), при каждой операции сброса таблицы или цепочки (`iptables -F`), при каждом добавлении очередного правила

(`iptables -A`) — вследствие чего, на конфигурирование фаервола уходит много времени. При каждом вызове iptables из пространства ядра загружается весь набор правил, затем над ним производится операция (например, при использовании флага `-A` к нему добавляется одно правило), и затем полученный набор загружается обратно в ядро. В результате, на выполнение n операций, уходит $\mathcal{O}\left(n^2\right)$ времени.

Кроме того, такой подход создает угрозу для безопасности вашей системы: в процессе формирования конфигурации, не исключено появление возможности для прохождения нежелательных пакетов. Даже предварительная установка DROP в качестве действия по умолчанию не всегда может предотвратить такое развитие событий. Рассмотрим простой пример:

```
iptables -P INPUT DROP;
# 2. Разрешаем VPN
iptables -A INPUT -p esp -j ACCEPT;
iptables -P FORWARD DROP;
# 4. Разрешаем пересылку только для определенных VPN-узлов
iptables -A FORWARD -m policy --tunnel-src 1.2.3.4 -j ACCEPT;
```

Этот скрипт создает не вполне очевидное «окно» в безопасности (между выполнением второй и третьей команды). Разумеется, в данном случае вы можете просто поменять местами соответствующие строчки, и «окно» будет закрыто. Однако мой внутренний голос подсказывает, что пользователь всегда найдет способ испортить практически любой набор правил.

Вышеперечисленные проблемы можно решить, перейдя от обычных скриптов к использованию `iptables-save`/`iptables-restore`. Ключевое достоинство `iptables-restore` — возможность сбросить все таблицы, установить действия по умолчанию для всех базовых цепочек, и сформировать полный набор правил *за одну атомарную операцию*. При некотором знакомстве с возможностями современных командных оболочек, вы можете даже использовать отступы, переменные и комментарии, как и в обычных скриптах:

```
#!/bin/sh

management=85.213.68.203/27

iptables-restore <<-EOF;
        *filter
```

```
        :INPUT DROP [0:0]
        :FORWARD DROP [0:0]
        :OUTPUT ACCEPT [0:0]

        # Мы можем использовать комментарии и переменные
        -A INPUT -d $management -p tcp --dport 22 -j ACCEPT
        COMMIT
EOF
```

Комментарии и пустые строки добавлены для улучшения читабельности кода. Отступ реализован средствами оболочки (минус в «`-EOF`»[1]) — перед подачей блока правил на вход `iptables-restore` он автоматически удаляется. Кроме того, в этом блоке будет работать и привычный перенос строк (через добавление обратной косой черты \ в конец строки).

4 Избыточные параметры

Смело удаляйте избыточные параметры, совпадающие со значениями, используемыми по умолчанию. Наиболее яркие примеры таких параметров: `-p all`, `0/0` или `::/0` в адресах (`-s`, `-d`, или где-нибудь еще), `0:65535` при указании портов TCP/UDP/SCTP/DCCP.

5 Устаревшие модули

Следующие модули являются устаревшими:

- `-m state`. Заменен на `-m conntrack`
- `-j NOTRACK`. Заменен на `-j CT --notrack`, начиная с Linux 2.6.35

6 Процедура Path MTU Discovery и TCP MSS

Не надо блокировать ICMP-сообщения с типом «parameter-problem», иначе вы нарушите нормальную работу процедуры Path MTU Discovery (определение оптимального значения TCP MSS), причем, возможно, последствия такой блокировки скажутся не только на вашем хосте. Если

[1] Подробнее об этой возможности см. страницу руководства `bash(1)` (или `sh(1)`).

же эти сообщения не будут блокироваться, применение к пакетам действия TCPMSS станет избыточным, что позволит вам еще немного сократить набор правил.

7 Интерфейс обратной петли

Постарайтесь не блокировать loopback-трафик в цепочках INPUT и OUTPUT — это может создать серьезные проблемы для некоторых локальных процессов.

7.1 Обратная петля — это не только 127.0.0.1

```
iptables -s 127.0.0.1 -j ACCEPT
```

— такой подход является неправильным, как минимум, по двум причинам. Во-первых, представленное правило благополучно пропустит трафик с других хостов, если у пакетов будет подделан исходный адрес (эта возможность может быть исключена добавлением `-i lo`). Во-вторых: 127.0.0.1 — это отнюдь не единственный адрес обратной петли. Вряд ли мы когда-нибудь узнаем, какие соображения сподвигли IANA выделить для обратной петли блок /8 (см. RFC3330), однако факт остается фактом. Многие дистрибутивы Linux используют дополнительные адреса (например, 127.0.0.2) для устранения проблем в некоторых некорректно написанных программах, или просто для удобства[2].

7.2 Даже так: обратная петля — это не только 127.0.0.0/8

Кроме того, есть еще одно «но»: к локальным адресам относится не только подсеть 127/8, но и все адреса, присвоенные всем сетевым интерфейсам данного хоста. И все пакеты, идущие с локального адреса на локальный адрес, маршрутизируются через интерфейс lo[3]. В частности, когда вы обращаетесь к собственному хосту, например, командой «`ssh 192.168.1.1`» (предположим, это адрес вашего хоста), счетчики

[2] См. https://bugzilla.novell.com/show_bug.cgi?id=416964.

[3] Строго говоря, дело даже не в адресах конкретных интерфейсов — пакеты маршрутизируются через lo при наличии соответствующих записей в таблице маршрутов. Вы можете просмотреть эти записи командой «`ip route list table local`». При добавлении адресов на сетевые интерфейсы вашего хоста, соответствующие записи формируются автоматически.

пакетов и байт интерфейса `lo` начинают расти. С учетом всего вышесказанного, корректное разрешение трафика обратной петли выглядит так:

```
iptables -A INPUT -i lo -j ACCEPT
```

8 Фильтрация по обратному пути

... она же — Reverse Path Filtering (некоторые называют ее Reverse Path Forwarding, но мы сейчас обсуждаем именно фильтрацию пакетов, и поэтому будем придерживаться классического термина).

Очень многие наборы правил содержат строки наподобие

```
-A INPUT -i ppp0 -s 192.168.0.0/16 -j DROP
```

(где ppp0 — интерфейс подключения к Интернету, на который не должны приходить пакеты с адресов локальной сети).

После включения RPF (см. `/proc/sys/net/ipv4/conf/all/rp_filter`) подобные правила становятся избыточными, и их можно будет удалить из набора. Однако, не стоит забывать, что бывают ситуации, когда включение RPF может оказать негативный эффект (например, асимметричная маршрутизация, в частности, подключение к Интернету через несколько сетевых интерфейсов).

9 Проверки TCP-флагов

Если вам повезло и у вас есть возможность использовать механизм отслеживания соединений (`nf_conntrack`, разумеется, требует некоторую часть ресурсов, но оно того стоит), вы можете использовать его возможности для проверки корректности TCP-флагов. Вместо того, чтобы писать кучу правил вроде

```
# Блокируем эти опасные пакеты! Все перечисленные здесь
# комбинации TCP-флагов не должны возникать при нормальной
# работе сети, однако порой они встречаются - в частности,
# при атаках на хост. Поэтому блокируем их и заносим
# в системный журнал.
-A INPUT -p tcp --tcp-flags ALL FIN,URG,PSH -j badflags
-A INPUT -p tcp --tcp-flags ALL ALL -j badflags
-A INPUT -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j badflags
-A INPUT -p tcp --tcp-flags ALL NONE -j badflags
```

```
-A INPUT -p tcp --tcp-flags SYN,RST SYN,RST -j badflags
-A INPUT -p tcp --tcp-flags SYN,FIN SYN,FIN -j badflags
```

вы можете просто выделять пакеты с состоянием соединения `INVALID` (только для TCP, или для всех протоколов — как вы сочтете нужным):

```
-A INPUT (-p tcp) -m conntrack --ctstate INVALID -j badflags
```

Упоминаемая здесь цепочка «`badflags`» используется в качестве условного обозначения операций, которые производятся над некорректными пакетам — вместо нее можно использовать просто `-j DROP`, или что-нибудь еще, на ваше усмотрение.

10 Комбинирование критериев

10.1 Расширение масок

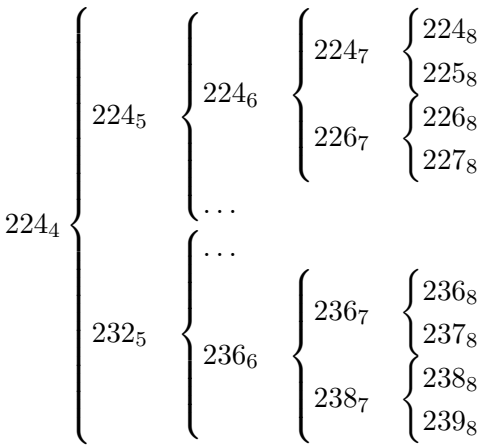
Рассмотрим яркий пример недостаточно оптимального набора правил — списки IP-адресов или блоков, наподобие такого:

```
-A INPUT -d 224.0.0.0/8 -j DROP
-A INPUT -d 225.0.0.0/8 -j DROP
...
-A INPUT -d 239.0.0.0/8 -j DROP
```

Эти правила должны выделять (и блокировать) трафик, идущий на multicast-адреса. Та же структура правил может встречаться и с другими сочетаниями параметров и действий.

Во многих случаях, смежные блоки адресов можно объединить, расширив действие маски так, чтобы она охватывала сразу все требуемые адреса. Последовательно соединяя блоки /8 в блоки /7, затем /6, /5 и /4, получим в итоге подсеть 224/4 (наиболее сообразительные могут сделать

это и за один шаг):



Разумеется, отнюдь не всегда такое объединение возможно. Например, блоки 225/8 и 226/8 нельзя объединить в один блок /7.

10.2 Диапазоны адресов

Нетрудно привести пример ситуации, когда группировка блоков оказывается недостаточно эффективной. Рассмотрим набор из 16 правил:

```
-A INPUT -s 10.10.96.255/32 -j ACCEPT
-A INPUT -s 10.10.97.0/32 -j ACCEPT
-A INPUT -s 10.10.97.1/32 -j ACCEPT
...
-A INPUT -s 10.10.97.14/32 -j ACCEPT
```

Пользуясь методикой, изложенной в разделе 10.1, мы можем свести это к пяти правилам:

```
-A INPUT -s 10.10.96.255/32 -j ACCEPT
-A INPUT -s 10.10.97.0/29 -j ACCEPT
-A INPUT -s 10.10.97.8/30 -j ACCEPT
-A INPUT -s 10.10.97.12/31 -j ACCEPT
-A INPUT -s 10.10.97.14/32 -j ACCEPT
```

Увеличивать блоки дальше мы уже не можем: если взять блоки 10.10.97.**0**/**28**, 10.10.97.**8**/**29**, 10.0.97.**12**/**30**, то под наши правила будет подпадать и адрес 10.10.97.15, которого не было в исходной постановке задачи.

Итак, можем ли мы решить задачу менее, чем пятью правилами? Да, можем — с критерием `iprange` нам будет достаточно только одной строчки:

```
-A INPUT -m iprange --src-range 10.10.96.255-10.10.97.14 -j ACCEPT
```

10.3 Использование произвольных масок

Мало кто знает об одной довольно интересной особенности iptables — возможности использования произвольных масок. Такие маски могут содержать чередующиеся нулевые и единичные биты, и поэтому они не всегда сводятся к классическому CIDR-формату «/n». Возьмем следующий блок из четырех правил, перед которым пасует даже группировка в диапазон адресов (см. раздел 10.2):

```
-A INPUT -s 10.10.97.1/32 -j ACCEPT
-A INPUT -s 10.10.97.3/32 -j ACCEPT
-A INPUT -s 10.10.97.5/32 -j ACCEPT
-A INPUT -s 10.10.97.7/32 -j ACCEPT
```

Специально подобранные для данного примера, «дыры» в последовательности адресов делают абсолютно бесполезной группировку адресов в диапазоны — меньше чем четырьмя диапазонами здесь не обойтись. Однако, поддержка произвольных масок в iptables позволяет упростить даже такой набор правил:

```
-A INPUT -s 10.10.97.1/255.255.255.249 -j ACCEPT
```

Маска /255.255.255.249 позволяет проверять на соответствие заданному адресу (10.10.97.1) все биты, кроме первого и второго (считая от бита номер ноль, последнего в маске). А если мы воспользуемся такой маской в сочетании с критерием `iprange`, мы можем, например, задать правили, под которое будут подпадать только адреса 10.10.97.3 и 10.10.97.5.

10.4 Произвольные наборы адресов

И все же, в жизни могут возникнуть ситуации, когда большой набор адресов не удается сократить до приемлемого размера, даже с использованием всех изложенных выше методик. В таких случаях стоит воспользоваться возможностями критерия `set`, специально оптимизированного для быстрого поиска по большим спискам адресов и подсетей. Это критерий реализован в рамках проекта ipset, обсуждение которого пока выходит за рамки данного документа.

10.5 Списки портов

```
-A INPUT -p tcp --dport 21 -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A INPUT -p tcp --dport 6881:6882 -j ACCEPT
```

Критерий `multiport` позволяет объединить номера и/или диапазоны портов, до 15 в одном списке:

```
-A INPUT -p tcp -m multiport --dports 21,22,6881:6882 -j ACCEPT
```

Это не только смотрится лучше, чем отдельные правила на каждый порт, но и работает гораздо быстрее[4].

Стоит отметить, что вышеупомянутый проект ipset поддерживает работу со списками портов, в том числе, очень большими списками (обычно платой за это является необходимость держать в памяти довольно большую битовую карту).

11 Частота срабатывания

Так как iptables применяет правила в том порядке, в котором они указаны (сверху вниз для каждой цепочки), целесообразно помещать в начало цепочки те правила, которые срабатывают чаще. Разумеется, при этом важно не нарушить логику, которую реализуют правила. Кроме того, не стоит забывать, что правила могут сильно различаться по потреблению системных ресурсов, и поэтому оценка правил только по эмпирическим наблюдениям за счетчиками пакетов/байт не всегда может дать желаемый результат. Я не располагаю точными данными по ресурсоемкости правил (впрочем, для сбора таких данных можно воспользоваться одним из многочисленных профилировщиков ядра), но прежде всего я бы присмотрелся к правилам, в которых много критериев.

11.1 Отслеживание состояния соединений

Почти каждый набор правил содержит строчку

```
-A INPUT -m conntrack --ctstate ESTABLISHED -j ACCEPT
```

[4] В обоих случаях сложность составляет $\mathcal{O}(n)$ (где n — число номеров/диапазонов портов), но у `multiport` это $\mathcal{O}$ меньше.

Аналогичное правило часто встречается в цепочке `FORWARD`, а иногда и в `OUTPUT`. Во многих случаях в него также добавляется состояние `RELATED`. Такое правила обычно обрабатывает бóльшую часть трафика. Например, на нашем многоцелевом выделенном сервере, через это правило прошло 99.02% трафика за последние 37 дней (41134 MB).

12 Еще несколько замечаний

12.1 Так лучше не делать

- Не надо блокировать входящие ICMP-сообщения типа «destination-unreachable», сообщающие о проблемах на другом конце канала связи (закрытый порт, и т.п.).
- Не надо блокировать входящие ICMP-сообщения типа «timeout-exceeded», так как они необходимы для работы сетевых трассировщиков, в частности, `traceroute`.

12.2 Так нельзя делать

- Ни в коем случае не используйте действие `DROP` в таблице `nat`. Действие `DROP` является фильтрующим, а таблица `nat` предназначена исключительно для операций трансляции адресов и портов. Через эту таблицу проходят далеко не все пакеты. Часто упоминаемое утверждение «эта таблица обрабатывает только первый пакет в каждом соединении» является только половиной истины, и поэтому следование ему может привести к неприятным последствиям.

12.3 FTP

Многие наборы правил содержат конструкции, разрешающие доступ для порта 20. С практической точки зрения, это совершенно бессмысленно, так как передача данных по FTP отнюдь не всегда происходит через этот порт. На обеих сторонах соединения, и в активном, и в пассивном режиме — номера портов могут быть выбраны динамически, и, обычно, так и происходит.

Возьмем, например, режим с пассивным клиентом:

```
client> EPSV
server> 229 Entering Extended Passive Mode (|||51458|)
```

Или, наоборот, режим с активным клиентом:

```
client> PORT 134,76,2,119,123,45
server> 200 PORT command successful.
```

В обоих случаях порт 20 не используется. Если в вашем наборе уже есть правило с опцией `--ctstate RELATED`, то оно должно корректно обработать первый пакет, поступивший на выбранный порт передачи данных (разумеется, для этого должен быть загружен модуль ядра `nf_conntrack_ftp`). Правило с опцией `--ctstate ESTABLISHED` обеспечит обработку всех остальных пакетов в этом соединении.

Также существует критерий `-m helper --help ftp`, но я не советую использовать его на трафике по установленным соединениям, который может довольно значительным (этот критерий использует довольно ресурсоемкую операцию сравнения строк) — лучше применять его уже после отбора по состоянию `RELATED`.

13 Заключение

И последний совет: обязательно ознакомьтесь с документацией по iptables и принципам составления наборов правил. Не полагайтесь на готовые скрипты — попытки передать их под конкретные задачи часто приводят к ужасным результатам.

Типичный пример такого «вредного» решения — fwbuilder, в интерфейсе которого различаются только «входящий» и «исходящий» потоки, что приводит к перекрестному замусориванию цепочек INPUT и FORWARD (правила, имеющие смысл только в одной из этих цепочек, попадают в другую, и наоборот).