

Используем check_relay в Sendmail для борьбы со спамом

Сергей Супрунов

Борьба со спамом вечна, как и сам спам. Способов защиты придумано множество, но ничего по-настоящему эффективного до сих пор не реализовано. Поэтому иногда приходится прибегать к «полумерам» типа фильтрации по именам хостов, «типичным» для спамеров. Такие почтовые серверы, как Postfix и Exim, предлагают сравнительно прозрачные способы фильтрации соединений. Владельцы же Sendmail вынуждены попотеть, чтобы раскрыть весь потенциал этого MTA.

Защита от спама на сервере провайдера по сравнению с «корпоративным» почтовым сервером имеет свои особенности.

Во-первых, здесь нет жёстких правил документооборота, позволяющих быть уверенным, что никто не ждёт писем в одной из японских кодировок или идущих с израильских доменов.

Во-вторых, среднестатистический абонент обычно не настолько владеет компьютером, чтобы эффективно осуществлять «обучение» различных систем контентной фильтрации, а сидимину заниматься анализом карантина негоже (в том числе и по этическим соображениям). Всё это делает системы типа Spamassassin или DSPAM не очень эффективными либо требующими повышенного внимания со стороны администратора.

К тому же контентная фильтрация совсем не способствует экономии трафика. Более того, если значительная часть писем будет «рубиться» на сервере после загрузки и не попадёт к конечным пользователям, то этот трафик приходится рассматривать как прямые убытки компании (то есть компания «покупает» спам, но клиенту уже не «перепродаёт»).

Из методов же, позволяющих трафик экономить, более-менее безопасно можно использовать разве что грейлистинг. Хотя в моей практике встречались клиенты, головные офисы которых оказывались не в состоянии обеспечить должную работу почтового сервера (а виноват, как обычно, провайдер, поскольку «в другие районы всё отправляется»).

Эффективность грейлистинга сравнительно высока (на моём сервере он отсекал до 95% «мусора»), тем не менее есть и некоторые недостатки:

- его не так уж сложно обойти;
- каждое соединение отнимает определённое время, поскольку решение принимается лишь после этапа «RCPT TO» SMTP-диалога (а то и после команды «DATA», если включены опции, призванные обеспечить взаимодействие с серверами, использующими конкурирующий алгоритм защиты от спама – callback);
- затрачиваются определённые ресурсы на ведение базы триплетов.

Про «фальшивые» имена

О блокировании соединений, разрешение имени для которых возвращает FORGED, следует сказать особо. Злоумышленники иногда используют следующий приём. Соединение устанавливается с некоторого IP-адреса, обратную зону для которого контролирует спамер (например, ему делегирована данная сеть адресов). Указать в PTR-записи можно всё что угодно, хоть barackobama.com. И поэтому имеет смысл дополнительно запросить IP-адрес хоста barackobama.com (на этот запрос уже должен ответить DNS-сервер, обслуживающий данный домен) и сравнить полученный IP-адрес с исходным. Правда, как показывает практика, в наше время под правило FORGED чаще попадают хосты, для которых по возвращённому в PTR-записи имени вообще невозможно определить IP-адрес (т.е. отсутствует соответствующая A-запись в прямой зоне):

Однако иногда можно встретить замечания, что подобная блокировка может привести к проблемам в случае «парковки» на один IP-адрес нескольких доменов. Но поскольку сервер сначала запрашивает PTR-

запись, а затем по полученному имени хоста – IP-адрес, то проблем быть не должно – в итоге при любом «раскладе» мы выйдем на единственный, исходный IP-адрес. Несоответствие же теоретически может возникнуть лишь в случае, когда возвращённое имя хоста указывает на несколько IP-адресов. На практике такое встречается крайне редко, когда используется балансировка нагрузки (показаны только интересные нас строки):

Но и в этом случае Sendmail, похоже, правильно отрабатывает ситуацию, проверяя, чтобы исходный IP-адрес соответствовал хотя бы одному из адресов (если их несколько), возвращённых функцией sm_gethostbyname (см. src/daemon.c). Так что есть все основания надеяться, что правило FORGED будет срабатывать только в случае действительных проблем с настройками DNS-сервера отправителя. Впрочем, если у вас остались сомнения, правило FORGED можно совсем удалить.

К тому же используемый мною milter-greylist (в связке с Sendmail) под нагрузкой иногда переставал вовремя отвечать на запросы, что приводило к тому, что соединения ещё дольше оставались открытыми, увеличивая число одновременно работающих экземпляров sendmail до максимально возможного, нарушая тем самым обслуживание клиентов.

Когда ресурсы сервера оказались практически исчерпаны, было принято решение о дополнительной фильтрации на первой линии обороны, призванной снизить нагрузку на milter-greylist и используемый для проверки на вирусы ClamAV, пусть даже и ценой «разумного риска» ложных срабаты-

ваний. От «чёрных списков» на основе DNSBL я почти отказался – слишком уж часто какой-то из «клиентских» серверов куда-нибудь попадает, и приходится разбираться с тамошними админами и клиентами. Поэтому я решил таки залезть в дебри sendmail.cf и воспользоваться способностью Sendmail выполнять фильтрацию сообщений на основе регулярных выражений.

Sendmail позволяет «вмешаться» в процесс приёма сообщения на различных этапах (см. рисунок). Поскольку наша задача – отсеять вероятный спам как можно раньше, то интерес для нас представляет в первую очередь набор правил check_relay, получающий в качестве параметра выраже-

```
$ grep "dns rule 2" /var/log/maillog | tail -1
```

```
Sep  8 15:29:11 myhost sm-mta[32267]: m88BTATA032267:
ruleset=check_rcpt, arg1=<energo@mydom.ru>,
relay=18971199139.user.veloxzone.com.br [189.71.199.139]
(may be forged), reject=550 5.7.1 <energo@mydom.ru>... Relaying
denied (dns rule 2). IP name possibly forged [189.71.199.139]
```

```
$ nslookup 18971199139.user.veloxzone.com.br
```

```
Server:      10.161.193.5
Address:     10.161.193.5#53

** server can't find 18971199139.user.veloxzone.com.br: NXDOMAIN
```

```
$ dig -x 77.88.21.3
```

```
3.21.88.77.in-addr.arpa. 13943 IN PTR www.yandex.ru.
```

```
$ dig www.yandex.ru
```

```
www.yandex.ru.      5865 IN A 93.158.134.3
www.yandex.ru.      5865 IN A 77.88.21.3
```

Синтаксис правил sendmail.cf

Весь синтаксис этого файла мы рассматривать не будем (при желании вы всегда найдёте нужную информацию в Интернете). Ограничимся лишь правилами проверки. Каждое правило начинается с символа R (он обязательно должен быть первым в строке, без ведущих пробелов). Далее следует «левая часть» (шаблон, напоминающий регулярное выражение) и через символ табуляции (не пробелы!) – «правая часть», то есть преобразование, выполняющее, что нужно сделать со входными данными, соответствующими шаблону. В шаблоне могут использоваться следующие лексемы:

- ☑ **\$*** – ноль и более токенов;
- ☑ **\$+** – один и более токенов;
- ☑ **\$-** – ровно один токен;
- ☑ **\$@** – ни одного токена;
- ☑ **\$=X** – соответствует некоторому элементу класса X;
- ☑ **\$~X** – не соответствует ни одному элементу из класса X.

Токен – это подстрока, разделённая пробелом или одним из символов, перечисленных в опции OperatorChars. По умолчанию это «.:%@!^/[]+». Поэтому не рассматривайте шаблон как чистое регулярное выражение – они работают с отдельными символами, а «левая часть» правил в sendmail.cf – с токенами. То есть строка «qwe_asd» будет соответствовать регулярному выражению «.*_*», но не шаблону «\$*_ \$*», поскольку символ «_» не является разделителем токенов, и, следовательно, входная строка представляет собой один-единственный токен.

Преобразование можно рассматривать как некоторую процедуру, которая, получая на входе строку, совпавшую с шаблоном, выполняет над ней некоторые дейст-

вия и возвращает результат. Типы возврата могут быть следующими (указывается в начале «правой части»):

- ☑ **\$:** – перейти на следующее правило;
- ☑ **\$@** – выйти из данного набора правил (вернуться к точке вызова набора);
- ☑ **\$#** – прекратить дальнейшую проверку правил.

Возврат \$# OK или \$@ OK (в зависимости от того, требуется ли проверка по другим наборам правил) означает «принять данное сообщение». Если возвращается \$error, то соединение разрывается с выдачей следующего за \$error сообщения (оно же записывается и в лог-файл). Кроме того, можно задать преобразование входящих данных, результат которого будет передан дальше. С помощью лексем вида \$1, \$2 и т. д. в правой части можно ссылаться на фрагменты шаблона (соответствующие каждой отдельной лексеме). Например, так можно преобразовать адрес вида domain.ru!user в user@domain.ru:

```
R$+!$+          $: $2@$1
```

Также в правой части часто используются следующие лексемы:

- ☑ **>ruleset** – вызов указанного набора правил;
- ☑ **\$(macro)** – выполнение указанного макроса;
- ☑ **\$(db args \$)** – проверка аргументов args по карте db.

Правила исполняются последовательно (за исключением выходов из набора по \$@ и \$#), каждое следующее использует в качестве входных данных выход предыдущего правила. Если входные данные шаблону не соответствуют, они проходят дальше без преобразований.

ние вида «<hostname>\$! <ip-address>», где hostname – имя хоста, ip-address – IP-адрес источника соединения. (Наборы check_mail и check_rcpt срабатывают после SMTP-команд «MAIL FROM» и «RCPT TO», получая в качестве параметра соответственно адрес отправителя или адрес получателя. Также опцией FEATURE('compat_check') в mc-файле можно подключить не показанный на рисунке «общий» набор check_compat, принимающий в качестве параметра и адрес отправителя, и адрес получателя и выполня-

ющийся после приёма всего сообщения. Сейчас эти наборы нам не столь интересны.) По умолчанию check_relay (точнее, Basic_check_relay) отвечает за «прогон» соединений по базе access и за работу «чёрных списков» (dnsbl) и выполняется до того, как в работу включатся внешние фильтры (работающие через milter-интерфейс). Определять свои правила обработки лучше всего в наборе правил Local_check_relay (по умолчанию наборы Local_check_* пусты), который будет вызываться первым, до

Basic_check_relay. Синтаксис рассмотрим ниже, после того как определимся с общими правилами фильтрации.

Определяем правила фильтрации

В большинстве случаев источником спама являются «пользовательские» подключения к сети Интернет, зачастую с динамическими IP-адресами. Это могут быть как машины самих спамеров, так и (что происходит гораздо чаще) заражённые какой-нибудь гадостью компьютеры ничего не подозревающих простых абонентов. Так что наша задача – определить правила, позволяющие предположить, является ли клиент «законным» почтовым сервером, или это обычный абонентский компьютер, с которого прямая отправка почты не предусматривается (скорей бы уж SPF начал приносить реальную пользу...). Попытаемся найти наиболее характерные отличия в именах «правильных» и «неправильных» хостов.

Но сначала ненадолго отойдём от собственно имён, чтобы рассмотреть другой вопрос. В идеале каждый почтовый сервер должен иметь доменное имя, тем более что таково требование RFC2821 (хотя теоретически можно отправлять почту и на user@[1.2.4.5], но на практике с таким сталкиваться не приходилось), в то время как адреса, динамически назначаемые клиентам, зачастую в DNS не представлены. Конечно, имя почтового домена и доменное имя SMTP-сервера – вещи несколько различные, и к имени сервера особых требований не предъявляется, но всё же серверы без PTR-записи мне пока не встречались. Так что хотя вероятность ложных срабатываний и остаётся, но ею, на мой взгляд, можно пренебречь, поэтому первым правилом введём «Блокировать почту с адресов, не имеющих PTR-записи». Здесь нам поможет не регулярное выражение, а способность Sendmail выполнять DNS-разрешение клиентского адреса и осуществлять то или иное действие в зависимости от результата (см. ниже листинг конфигурации, четыре строки, начиная с обращения к макросу &(client_resolve)). В данном примере я показал блокировку всех трёх возможных ошибок – TEMP, FORGED и FAIL, то есть помимо отсутствия PTR-

записи (FAIL) блокируются также временные ошибки DNS и ситуации, когда запрос А-записи по доменному имени возвращает IP-адрес, отличный от исходного (FORGED).

Кстати говоря, проверку на разрешение имени в последних версиях Sendmail можно включить и просто опцией FEATURE('require_rdns'). Но настройка вручную позволит лучше контролировать как порядок, так и состав проверок.

Как показывает практика, львиная доля спама идёт с адресов вида 123-45-67-89-nekiy.domen.provaidera.ru или m123.dialup.domain.com. Именно такие имена хостов провайдеры, не мудрствуя лукаво, обычно назначают динамическим соединениям. То есть либо в имени присутствует в том или ином виде сам IP-адрес (цифры, разделённые точками, подчёркиваниями, дефисами и иногда символами «х»), либо фигурируют некоторые характерные «слова» – dial, rpp, adsl, dynamic, dhcp, pool, client и т. п. Оформим эти случаи в следующие правила (поскольку на второй линии обороны у нас стоит miller-greylist, то правила будут достаточно строгими, чтобы свести к минимуму риск блокировки нормальных соединений).

Правило 1: Имя хоста содержит четыре группы цифр, разделённых символами [._x-]

Соответствующее регулярное выражение будет выглядеть так: `([0-9]{1,3}[._x-]){4}`. Под это выражение будут подпадать имена следующего вида (показанные здесь и далее имена хостов взяты из лог-файла и соответствуют хостам, с которых реально поступал спам; если кто увидит свой домен – я не специально):

- cpe-98-27-181-209.neo.res.rr.com;
- dialup-88-147-133-182.san.ru;
- 109.251-224-87.telenet.ru;
- 125x103x16x176.ap125.ftth.ucom.ne.jp;
- 219-180-207-85.zapcechy.adsl-llu.static.bluetone.cz.

К сожалению, нередко имена, включающие меньшее число цифровых групп:

- 100-240-90.adsl.terra.cl;
- 103-185.trifle.net.

Можно, конечно, в предыдущем правиле просто уменьшить число ожидаемых групп до двух, но по соображениям удобства отладки (учитывая, что вероятность ложных срабатываний по данному правилу несколько выше) выделим его отдельно.

Правило 2: Имя хоста содержит как минимум две группы цифр

В порядке ужесточения также потребуем, чтобы это не было частью имени домена 2-го уровня (но при этом уберём

Тестирование правил

Как известно, от ошибок никто не застрахован. Синтаксис регулярных выражений, если можно так сказать, «поощряет» опечатки – один пропущенный символ экранирования перед точкой, и результат может уже довольно сильно отличаться от ожидаемого. Поэтому каждое правило имеет смысл предварительно протестировать (тем более, что разрабатывать правила вы, скорее всего, будете, ориентируясь на вполне определённые имена хостов). Сделать это можно следующим образом:

```
$ sendmail -bt
```

```
ADDRESS TEST MODE
(ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
```

```
> /map block4 pool-127-165.gen.ua
```

```
map_lookup: block4 (pool-127-165.gen.ua)
no match (0)
```

```
> /map block5 pool-127-165.gen.ua
```

```
map_lookup: block5 (pool-127-165.gen.ua)
returns @MATCH (0)
```

```
> /quit
```

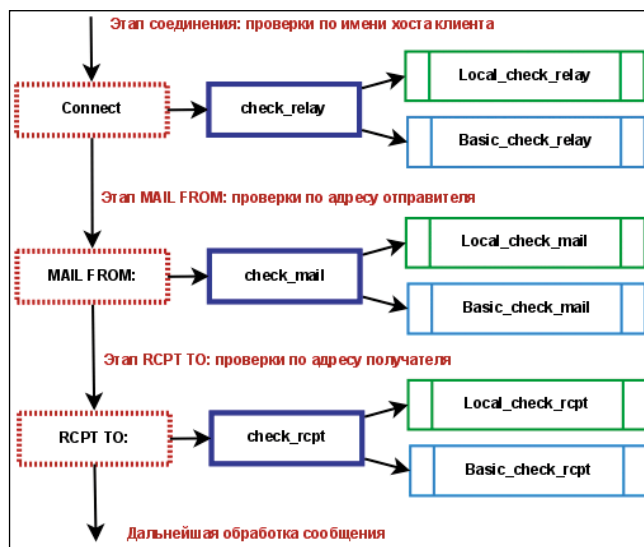
Команда `/map` проверяет имя хоста на соответствие той или иной «карте». Наши правила тоже являются картами. Если соответствие найдено, возвращается `@MATCH`, если нет – вы получите сообщение «no match».

Проводить тестирование лучше всего после компиляции нового конфигурационного файла (`make`), но до его инсталляции (`make install`) и перезапуска работающего почтового сервера (`make restart`). В этом случае потребуются явно указать новый конфигурационный файл (`sendmail -bt -C mydom.ru.cf`), но зато в случае ошибок вы сможете их исправить до того, как они скажутся на реальной работе.

верхнюю границу на число цифр в группе, так как цифры в данном случае не всегда означают часть IP-адреса): `([0-9]+[._x-]){2,}.*(\.){2,}`. Продолжим.

Правило 3: Имя хоста содержит более восьми шестнадцатеричных цифр подряд

Иногда встречаются и такие имена, как `bd21a1bd.virtua.com.br` (обычно цифр 8, т.е. записывается IP-адрес в hex-формате, но может встречаться и больше). Поскольку здесь участвуют и буквы, то можно «нарваться» на добропорядочный адрес типа `affecscade.cho-to.tld`, так что дополнительно потребуем, чтобы такая группа символов была частью домена не менее четвёртого уровня: `[0-9a-f]{8,}(\.){3,}`. Ну и для себя отметим, что здесь требуется дополнительное внимание к логам (по-хорошему, это правило лучше бы вообще исключить, но мне пока оно проблем не создавало; вас же я предупредил).



Порядок вызова наборов правил `check_*`

Что ответить на HELO?

Наверняка вы обратили внимание, что в показанной схеме один этап отсутствует – команда HELO/EHLO, которой клиент (под «клиентом» здесь и далее подразумевается программа, устанавливающая соединение с вашим сервером) приветствует сервер сразу после подключения. Согласно стандарту в HELO/EHLO клиент должен передать своё полное имя (FQDN), которое, очевидно, должно соответствовать IP-адресу клиента. Однако на практике здесь может быть множество «подводных камней».

Во-первых, у различных клиентов могут быть различные понятия о стандартах (особенно это касается собственно почтовых клиентов типа Outlook, The Bat! и т. п., так что в первую очередь фильтрация по HELO может сказаться на ваших же пользователях) – не всегда в HELO будет указано именно то, что нужно. Во-вторых, клиент может соединяться через какой-нибудь NAT- или прокси-сервер и, передавая HELO, может понятия не иметь, с какого IP-адреса фактически будет идти соединение. Наконец, в-третьих, у клиента может быть несколько интерфейсов, и не факт, что FQDN будет соответствовать именно тому, с которого осуществляется подключение к серверу. В общем, на мой взгляд, «безусловная» фильтрация по HELO слишком опасна, и её лучше исключить.

Но, с другой стороны, такая проверка может быть весьма эффективной. Так что если вы всё-таки хотите «наказать» кли-

ентов, не соблюдающих стандарт (кстати говоря, в RFC 2821 ясно обозначено, что сервер не должен отказывать в соединении на основании неправильного имени в HELO/EHLO), можете воспользоваться рядом приёмов, которые можно найти в Интернете. Обычно ограничиваются блокировкой клиентов, которые сообщают в приветствии имя сервера, к которому подключаются (такая блокировка сравнительно безопасна, но тоже могут быть нюансы). Вот несколько ссылок:

- ☑ <http://www.opennet.ru/openforum/vsluhforumID1/57895.html>;
- ☑ <http://www.mta.org.ua/sendmail/stuff/www.cs.niu.edu/~rickert/cf/bad-ehlo.html>;
- ☑ http://www.mta.org.ua/sendmail/stuff/www.cs.niu.edu/~rickert/cf/hack/block_bad_helo.m4;
- ☑ <http://www.vttoth.com/heloehlo.htm>.

Наконец, можно (целиком положившись на разработчиков Sendmail) добавить в mc-файл опцию FEATURE('block_bad_helo'). Любопытных отсылаю к файлу cf/m4/proto.m4, начиная со строки «ifdef(`_BLOCK_BAD_HELO_', `dnl», где можно посмотреть, какие именно правила будут применяться.

Однако, на мой взгляд, проверку по HELO лучше использовать исключительно в комплексных системах типа Spamassassin как один из критериев «спамности». Но эта тема уже далеко выходит за рамки статьи.

pat добавим сюда же, поскольку в данном контексте они уже более «безопасны».

И ещё одно правило предусмотрим для явного указания конкретных доменов, почту от которых мы не хотели бы принимать ни при каких обстоятельствах. Пример можно посмотреть ниже, в листинге конфигурации.

Правила-исключения

Пока будем считать, что этого достаточно. (Думаю, вы понимаете, что дальше вам предстоит не один месяц кропотливого анализа лог-файлов на предмет выявления ложных срабатываний и пропуска спама с последующей корректировкой данных правил.) Но нужно предусмотреть ещё кое-что.

Во-первых, рано или поздно вы столкнётесь с тем, что некоторые «хорошие» почтовые серверы попадут под ваши правила, а корректировка самих правил с целью учесть эту ошибку либо сделает правила излишне громоздкими, либо приведёт к увеличению числа спама, просачивающегося сквозь фильтрацию.

Во-вторых, если своим собственным клиентам вы назначаете DNS-имена так, что они попадают под одно из правил (либо не назначаете имён вообще), то может получиться так, что они не смогут отправлять почту через ваш сервер. Таким образом, мы приходим к необходимости создать правила-исключения, которые будут стоять до рассмотренных выше правил блокировки и пропускать те соединения, которые, безусловно, должны быть пропущены.

Таких правил для начала будет три:

- `^(mailrelay|mx|smtp|mta).*[0-9]*(\.)*{2,}` – то есть будем пропускать адреса вида relay2.mydom.com. (Сейчас они и без того будут проходить через наши правила без проблем, но, во-первых, лучше обработать потенциально частые соединения пораньше, чтобы не прогонять каждое через все правила, а, во-вторых, это позволит уменьшить тяжесть последствий ошибки в будущем, когда нам захочется «закрутить гайки».)

- `\.(mydom\.rulrambler\.rulyandex\.rulmail\.ru)$` – это правило для доменов-исключений. Сюда имеет смысл занести свой домен, а также домены, с которых получить спам –

Правило 4: Имя хоста содержит одно из указанных сочетаний

Сочетания будем отслеживать следующие: dsl, dial, dynamic, static, ppp, pool, client, user, dhcp, gprs. В порядке ужесточения правила потребуем, чтобы перед указанным сочетанием в имени фигурировала, как минимум, одна цифра (поскольку динамические имена, как правило, похожи, отличаясь лишь некоторым номером). Кроме того, будем считать, что искомое сочетание не может быть частью домена второго уровня (чтобы избежать блокировки адресов типа relay2.dynamic-hosting.org). Примеры:

- 1.pool85-58-70.dynamic.orange.es;
- 0x5735c879.esnx7.dynamic.dsl.tele.dk;
- 122.208.c10008-a53.dsl-dynamic.vsi.ru;
- 17-4.gprs.tmcz.cz;
- abfx26.neoplus.adsl.tpnet.pl.

Регулярное выражение получается довольно длинным: `[0-9].*(dsl|dial|dynamic|static|ppp|pool|client|user|dhcp|gprs).*(\..)*{2,}`. К сожалению, в «динамических» именах часто попадают также сочетания dyn, vpn и ip, но здесь уже вероятность ложных срабатываний повышается, так что в этом случае будем полагаться на грейлистинг.

Бывает также, что цифры следуют не перед, а после указанного сочетания в одной части доменного имени: pool3.provider.ru. Учтём это следующим правилом.

Правило 5: Имя хоста содержит указанные сочетания с последующими цифрами

Правило: `(dsl|dial|dyn|static|ppp|pool|client|user|dhcp|gprs|ip|vpn|nat)[a-z_-]*[0-9]+.*(\..)*{2,}`. Сочетания dyn, ip, vpn,

меньшее зло, нежели отвергнуть нужное сообщение. Рассматриваются лишь поддомены и отдельные хосты указанных доменов (предполагается, что сами доменные имена ни под одно из последующих правил блокировки не попадают). Не забывайте экранировать символы «.».

■ **10.5\.[0-9]{1,3}\.[0-9]{1,3}** – здесь мы будем пропускать соединения с IP-адресов вида 10.5.x.y, предполагая, что данные адреса принадлежат нашим клиентам, для которых имена хостов в DNS не прописаны (а значит, в качестве таковых будут передаваться IP-адреса). Если «своих» сетей несколько, лучше для каждой из них указать отдельное правило.

Изменяем конфигурацию

Теперь добавим всё это в файл конфигурации Sendmail. Редактировать непосредственно sendmail.cf – не самая лучшая идея, так как этим вы обречёте себя на необходимость выполнять абсолютно всю настройку именно в этом файле, поскольку при использовании тс-файла внесённые в cf-файл изменения будут теряться. К счастью, «cf-фрагменты» при необходимости можно добавлять и в тс-файл, где они будут учитываться препроцессором m4 и переноситься в итоговый cf-файл.

Для этого в самом конце тс-файла (после опций MAILER) добавим следующее (длинные строки для удобства разбиты на несколько, строка-продолжение начинается с пробелов):

```
LOCAL_CONFIG

# Правила-исключения
Krelays      regex -a@MATCH ^(\mail|relay|mx| \
smtp|mta).*[0-9]*(\.)*{2,}
Kexdoms      regex -a@MATCH \.(mydom\.ru|hotmail\.com| \
rambler\.ru|yandex\.ru|mail\.ru)$
Kournet1     regex -a@MATCH 10\5\.[0-9]{1,3}\.[0-9]{1,3}

# Правила блокировки
Kblock1      regex -a@MATCH ([0-9]{1,3}[\. x-]) {4}
Kblock2      regex -a@MATCH ([0-9]{1,3}[\. x-]) {2,}.*(\.)*{2,}
Kblock3      regex -a@MATCH [0-9a-f]{8,}(\.)*{3,}
Kblock4      regex -a@MATCH [0-9].*(dsl|dial|dynamic| \
static|ppp|pool|client|user|dhcp|gprs).*(\.)*{2,}
Kblock5      regex -a@MATCH (dsl|dial|dyn|static| \
ppp|pool|client|user|dhcp|gprs|ip| \
vpn|nat)[a-z_-]*[0-9]+.*(\.)*{2,}
Kblock6      regex -a@MATCH (vectranet\.pl| \
mediating\.barrier\.volia\.net)$

LOCAL_RULESETS

SLocal_check_relay

# Пропускаем соединения со своей подсети
R$*          $: $(ournet1 ${client_name} $)
R@MATCH      $@ OK
```

Стандарт есть стандарт

Согласно RFC822 и RFC1123 каждый почтовый сервер должен безоговорочно принимать почта на адрес postmaster@<domain.tld>. RFC2142 добавляет к этому списку также адрес abuse@<domain.tld>. Это обусловлено тем, что у отправителя всегда должна быть возможность сообщить вам о проблеме. В нашем примере это требование стандарта не выполняется, т.к. в случае «неправильного» имени хоста наш сервер сразу разорвёт соединение, не спросив даже имени отправителя (MAIL FROM), не говоря уже о том, чтобы узнать, кому тот хотел написать письмо (RCPT TO).

Как было показано, проверки check_relay выполняются до проверок check_mail и check_rcpt. Однако этот порядок можно изменить. Для этого в тс-файл нужно добавить следующую директиву:

```
FEATURE(`delay_checks', `friend')
```

Это, во-первых, отложит проверки check_relay и check_mail до того момента, пока не пройдёт check_rcpt. Во-вторых, аргумент «friend», переданный директиве, позволяет задать в базе access правило SPAMFRIEND для нужных адресов:

```
To:postmaster@      SPAMFRIEND
To:abuse@            SPAMFRIEND
```

Пересобрав базу access (командой make в каталоге /etc/mail), вы обеспечите безоговорочное прохождение сообщений на указанные адреса, и лишь после

этого будут выполняться наши проверки check_relay. Кстати, в лог-файл информация о блокировках отныне будет заноситься с пометкой «ruleset=check_rcpt», поскольку теперь наборов правил check_relay и check_mail в sendmail.cf нет – вместо них при использовании вышеуказанной директивы появляются «нестандартные» checkrelay и checkmail соответственно, которые вызываются из check_rcpt. Побочным эффектом этого будет то, что в daily-сообщениях везде будет фигурировать ваш домен:

```
Checking for rejected mail hosts:
8252 mydom.ru (554... 1))
1492 mydom.ru (554... 2))
668 mydom.ru (554... 3))
534 mydom.ru (554... 4))
424 mydom.ru (554... 5))
111 mydom.ru (550... [218.9.79.68])
108 mydom.ru (550... [89.38.12.29])
91 mydom.ru (550... dul.ru)
76 mydom.ru (451... resolve)
...
```

Происходит это из-за того, что скрипт /etc/periodic/daily/460.status-mail-rejects (напомню, что речь идёт о FreeBSD) выбирает домен из подстроки arg1, записываемой в лог-файл для каждого факта блокировки, а для check_rcpt туда передаётся адрес получателя (т.е. адрес вашего пользователя). Кроме того, такое изменение порядка следования правил увеличивает нагрузку на систему. Но зато у отправителей будет больше шансов сообщить вам о допущенной вами ошибке.

```
# Выполняем разрешение имени клиента
R$*          $: < ${client_resolve} >
R<TEMP>      $$error $@ 4.4.0 $: \
"450 Temp. (dns rule 1). Cannot resolve PTR for " \
${client_addr}
R<FORGED>     $$error $@ 5.7.1 $: \
"550 Denied (dns rule 2). IP name forged " \
${client_name}
R<FAIL>       $$error $@ 5.7.1 $: \
"550 Denied (dns rule 3). IP name lookup failed " \
${client_name}

# Применяем оставшиеся правила-исключения
R$*          $: $(relays ${client_name} $)
R@MATCH      $@ OK
R$*          $: $(exdms ${client_name} $)
R@MATCH      $@ OK

# Блокируем соединения согласно правилам блокировки
R$*          $: $(block1 ${client_name} $)
R@MATCH      $$error $@ 5.7.1 $: "554 Bad hostname \
for mailserver: " ${client_name} " (regex rule 1)"
R$*          $: $(block2 ${client_name} $)
R@MATCH      $$error $@ 5.7.1 $: "554 Bad hostname \
for mailserver: " ${client_name} " (regex rule 2)"
R$*          $: $(block3 ${client_name} $)
R@MATCH      $$error $@ 5.7.1 $: "554 Bad hostname \
for mailserver: " ${client_name} " (regex rule 3)"
R$*          $: $(block4 ${client_name} $)
R@MATCH      $$error $@ 5.7.1 $: "554 Bad hostname \
for mailserver: " ${client_name} " (regex rule 4)"
R$*          $: $(block5 ${client_name} $)
R@MATCH      $$error $@ 5.7.1 $: "554 Bad hostname \
for mailserver: " ${client_name} " (regex rule 5)"
```

```
R$*      $: ${block6 ${client_name} $}
R@MATCH  $#error $@ 5.7.1 $: "554 Bad hostname ↵
        for mailserver: " ${client_name} " (regex rule 6)"
```

Итак, что здесь происходит?

В секции LOCAL_CONFIG прописаны все наши правила. Строки, начинающиеся с «K» – это так называемые «карты» (maps), в нашем случае имеющие тип regex и проверяющие соответствие входящих аргументов указанному регулярному выражению.

Далее, в секции LOCAL_RULESETS, определяются локальные наборы правил преобразования. В нашем случае такой набор один – Local_check_relay. И здесь мы последовательно проверяем соответствие имени клиента (\${client_name}) нашим правилам.

Сперва безоговорочно пропускаем (\$@ OK) клиентов, подпадающих под правило ournet1. Затем выполняем разрешение имени клиента (\${client_resolve}) с выдачей соответствующих сообщений об ошибках в зависимости от результата. Если проверку на ournet1 делать после разрешения имени, то соединения наших клиентов, не имеющих PTR-записи, будут отклонены.

Клиенты, успешно прошедшие проверку на соответствие PTR- и A-записей, переходят к дальнейшим правилам. Сначала применяем наши правила-исключения, пропуская клиентов, имя которых соответствует данным регулярным выражениям (если у вашего домена есть проблемы с разрешением имён, то эти правила можно поставить сразу после ournet1; хотя лучше навести порядок в DNS).

Ну а затем блокируем соединения по нашим правилам блокировки. Для удобства последующего анализа лог-файла каждое сообщение об ошибке помечаем номером правила.

Запускаем и тестируем

Теперь можно ещё раз всё проверить (обратите особое внимание, чтобы левая и правая части каждой строки разделялись символами табуляции, а не пробелами), пересобрать sendmail.cf и поставить его на «боевое дежурство» (в данной статье я имею в виду FreeBSD; на вашей системе могут потребоваться другие команды):

```
# cd /etc/mail
# make && make install
# make restart
```

Результат работы можно будет получить, проанализировав лог-файлы (maillog.7.bz2 собран, когда данные правила фильтрации ещё не использовались; maillog.0.bz2 – уже с работающими регулярными выражениями):

```
$ cd /var/log
$ bzcat maillog.7.bz2 | grep "Greylisting in action" | ↵
wc -l
```

203944

```
$ bzcat maillog.7.bz2 | grep "stat=Sent" | wc -l
```

1188

```
$ bzcat maillog.7.bz2 | grep "5.7.1" | wc -l
```

1179

```
$ bzcat maillog.0.bz2 | grep "Greylisting in action" | wc -l
```

5760

```
$ bzcat maillog.0.bz2 | grep "5.7.1" | wc -l
```

44931

```
$ bzcat maillog.0.bz2 | grep "stat=Sent" | wc -l
```

1243

При желании можно получить статистику по конкретным сработавшим правилам, например, командой:

```
grep 'regex rule 3' maillog | wc -l
```


можно получить количество соединений, заблокированных третьим правилом. Поскольку делать подобный анализ нужно будет довольно часто, имеет смысл создать для этих целей shell-скрипт (один из возможных вариантов вы найдёте на сайте журнала, в разделе «Исходный код»):

```
$ ./getsendmailstat.sh /var/log/maillog.0.bz2
```

```
[Extracting /var/log/maillog.0.bz2 to /tmp/getsendmailstat...]
DNS rule 1:      704
DNS rule 2:     11913
DNS rule 3:     21472

Regex rule 1:    8248
Regex rule 2:    1492
Regex rule 3:     668
Regex rule 4:     534
Regex rule 5:     424
Regex rule 6:       0

Total 5.7.1:     44931
Greylisting:     5760
Total sent:      1243
[/tmp/getsendmailstat removed.]
```

Итак, вместо более чем 200 тысяч срабатываний greylisting за сутки, после включения правил фильтрации по имени хоста на долю milter-greylist осталось лишь 5-7 тысяч подключений. (При анализе этих данных нужно учитывать тот факт, что если сообщение отправляется сразу на несколько адресов получателей, то в случае greylisting создаётся запись в лог-файле для каждого адреса, блокировка же сопровождается одной записью на всё соединение. Так что можно считать, что одна строка «5.7.1» идёт за три-четыре «Greylisting in action».) Нагрузка на систему заметно снизилась, так что поставленную задачу можно считать решённой. Первое время придётся, конечно, критично просматривать лог-файлы и более чутко реагировать на жалобы клиентов, зато необходимость поиска нового «железа» для сервера ещё на некоторое время теряет свою актуальность. 

1. Супрунов С. Greylisting: панацея от спама или «мыльный пузырь»? //Системный администратор, №7, 2006 г. – С. 12-15. – http://www.samag.ru/art/07.2006/07.2006_04.html.
2. Клаус Асман. Using check_* in sendmail 8.8 – <http://www.sendmail.org/~ca/email/check.html>.
3. Эрик Олман. Anti-Spam Configuration Control – http://www.sendmail.org/m4/anti_spam.html.
4. Д. Сени. Considerations for the use of DNS Reverse Mapping draft-ietf-dnsop-reverse-mapping-considerations-06 – <http://tools.ietf.org/html/draft-ietf-dnsop-reverse-mapping-considerations-06>.