Руководство по системе фильтрации спама rspamd.

Стахов Всеволод.

25.09.2009

Оглавление

Глава 1

Общая информация и возможности rspamd

Rspamd - это система, предназначенная для фильтрации спама. Изначально rspamd разрабатывался как фильтр для электронной почты, но он может применяться и для другого типа сообщений (например, для jabber или icq сообщений). В основе rspamd лежит концепция асинхронной обработки входящих сообщений. Для этого применяется библиотека libevent. Это накладывает определенные ограничения на возможности rspamd, так как для любой блокирующей операции (например, чтение из сетевого сокета) необходимо регистрировать отдельное событие и его обработчика, но дает преимущества в скорости работы системы и уменьшает различные служебные затраты (например, на создание процессов или потоков). Rspamd поддерживает встроенные фильтры на языке lua, что позволяет писать собственные фильтры без необходимости пересборки системы. Rspamd настраивается путем редактирования конфигурационного файла. Также имеется управляющий интерфейс, посредством которого можно различным образом управлять работой системы и получать ее текущее состояние. Rspamd поддерживает различные типы фильтров: фильтры на основе регулярных выражений, фильтры на основе DNS запросов, фильтры на основе статистики, фильтры по различным спискам и другие типы фильтров (например, фильтры, написанные на языке lua и выполняющие различные действия по анализу сообщений). Rspamd имеет протокол, совместимый с системой spamassassin (в дальнейшем протокол spamc), а также его расширение - rspamc, позволяющее передавать больше информации фильтру, что ускоряет обработку сообщений. Система rspamd состоит из двух основных частей: монитор процессов и процессы, осуществляющие обработку (workers). Монитор процессов отвечает за старт системы, открытие/закрытие журналов работы, а также обеспечивает непрерывную работу рабочих процессов и их перезапуск при необходимости.

Глава 2

Установка rspamd

2.1 Требования

- GNU C компилятор (работоспособность проверялась на gcc 4.2.1)
- cmake - http://cmake.org/ используется для конфигурации сборки и генерации Makefile. Необходимая версия - не менее 2.6.
- glib - http://ftp.gnome.org/ используется для различного рода утилит и структур хранения данных (хеши, деревья, списки). Необходимая версия - не менее 2.16.
- gmime - http://ftp.acc.umu.se используется для разбора mime структуры сообщений. Необходимая версия 2.2. Работа с gmime 2.4 и старше не проверялась.
- lua - http://www.lua.org/ используется для работы lua плагинов (без liblua работа rspamd возможна, но без поддержки lua плагинов). Версия необходима не меньше, чем 5.1.
- libevent - http://www.monkey.org/~provos/libevent/ используется для кросс-платформенной обработки асинхронных событий, а также для определения DNS имен (также асинхронного).

2.2 Установка

Для сборки rspamd необходимо скачать архив (самая свежая версия может быть найдена на http://cebka.pp.ru/distfiles/). После этого необходимо распаковать архив и скомпилировать код:

```
$ tar xzf rspamd-x.x.x.tar.gz
$ cd rspamd-x.x.x
$ cmake .
$ make
```

Установка осуществляется стандартным

```
# make install
```

В процессе установки копируются исполняемые файлы rspamd: bin/rspamd и bin/rspamc, а также примеры конфигурации и плагины, устанавливающиеся в каталог etc/rspamd/. Также для ОС FreeBSD устанавливается стартовый скрипт rspamd.sh в каталог etc/rc.d.

2.3 Запуск

Rspamd запускается либо из стартового скрипта, либо непосредственно вызовом rspamd. Доступные опции командной строки:

-h: Показать справочную информацию и выйти

-t: Проверить конфигурационный файл и выйти

-C: Показать содержимое кеша символов и выйти

-V Показать все переменные rspamd и выйти

-f: Не выполнять демонизацию

-c: Указать путь до конфигурационного файла (по умолчанию используется /usr/local/etc/rspamd.conf)

-u: Пользователь, под которым осуществлять работу rspamd

-g: Группа, под которой осуществять работу rspamd

Если rspamd запускается от суперпользователя, то после создания лог-файла, PID-файла, а также сокетов, принимающих соединения, осуществляется сброс привиллегий до пользователя и группы, указанных в опциях командной строки (таким образом, все рабочие процессы работают от указанного пользователя и группы).

Глава 3

Общие принципы работы

Прежде чем приступать к настройке rspamd необходимо понять основные принципы функционирования системы.

3.1 Планирование и запуск рабочих процессов

При запуске rspamd происходят следующие действия:

1. Запускается главный процесс (rspamd main)
2. Инициализируются конфигурационные параметры по умолчанию
3. Читаются параметры командной строки
4. Настраивается журналирование ошибок в терминал
5. Читается и парсится конфигурационный файл
6. Инициализируются модули
7. Модули читают свои конфигурационные параметры
8. Устанавливаются лимиты
9. Настраивается журналирование, указанное в конфигурационном файле
10. Происходит демонизация (если не указан флаг -f)
11. Настраивается обработка сигналов головным процессом
12. Записывается PID-файл
13. Инициализируются lua плагины
14. Инициализируется подсистема событий и mime парсер
15. Загружается кеш символов
16. Порождаются рабочие процессы (сброс привиллегий осуществляется сразу же после вызова fork)

17. Начинается цикл обработки сигналов

Головной процесс rspamd реагирует на следующие сигналы:

- SIGTERM - послать всем рабочим процессам SIGTERM, дождаться их завершения и выйти
- SIGINT - то же, что и SIGTERM
- SIGHUP - переинициализировать журналирование и породить новые рабочие процессы, завершив старые (при этом, существующие рабочие процессы завершают работу, обработав уже полученные соединения)
- SIGCHLD - головной процесс получает этот сигнал при завершении работы рабочего процесса. Если рабочий процесс завершился некорректно, то планируется его перезапуск через 2 секунды.
- SIGUSR2 - приходит от рабочего процесса, когда тот успешно инициализируется
- SIGALARM - сигнализирует о необходимости запуска рабочего процесса, который был запланирован после получения SIGCHLD

Таким образом, головной процесс отвечает за инициализацию, конфигурацию, работу с PID-файлом, работу с журналированием, а также за порождение рабочих процессов. В ходе работы головной процесс постоянно следит за работой рабочих процессов и обеспечивает перезапуск некорректно завершившихся рабочих процессов. Для ротации файлов журналирования рабочему процессу необходимо послать сигнал SIGHUP.

3.2 Логика обработки сообщений

Инициализация рабочего процесса предельно проста: происходит переинициализация libevent, а также инициализация DNS resolver'a. После этого рабочий процесс устанавливает обработчик готовности к чтению слушающего сокета (этот сокет создается в головном процессе и передается рабочему процессу как параметр). При готовности к чтению на слущающем сокете рабочий процесс создает новый объект типа worker_task и делается accept на слушающем сокете. После этого rspamd обрабатывает протокол rspamc (или же spamc) и читает сообщение. После окончания получения сообщения rspamd декодирует его содержимое и начинает обработку. Для более простого изложения принципов работы rspamd необходимо описать некоторые понятия:

- Символ - это правило фильтрации rspamd, например, некоторое регулярное выражение или же запрос к DNS или же любое другое действие. Символ имеет собственный вес и имя. Таким образом, символ можно считать результатом работы одного правила фильтрации. Если это правило сработало, то оно добавляет символ с определенным весом и атрибутами, если нет, то символ не добавляется.
- Метрика - это набор логически связанных правил и связанных с ними символов. Такая группа имеет свой предел очков, после набора которых сообщение считается по этой метрике спамом. Очки формируются после

подсчета весов символов, добавленных в метрику (при этом, разумеется, несработавшие правила символов не добавляют и их вес равен нулю) и обработки этих весов функцией консолидации. По умолчанию такой функцией является функция-факторизатор, которая просто считает вес каждого символа равным константе, заданной в конфигурационном файле для этого символа, например, следующие параметры в конфигурационном файле задают вес символа MIME_HTML_ONLY равный одному, а вес символа FAKE_HTML - восьми:

```
"MIME_HTML_ONLY" = 1;
"FAKE_HTML" = 8;
```

- Модуль - это набор правил rspamd, который обеспечивает общие проверки. Например, модуль проверки регулярных выражений или модуль проверки URL'ей по "черным" спискам. Модули также могут быть написаны на языке LUA. Каждый модуль регистрирует символы, соответствующие сконфигурированным в нем правилам, в таблице символов заданной метрики (или метрики по умолчанию "default").

- Таблица символов метрики - это таблица, хранящая данные о зарегистрированных символах, таблица отсортирована, чтобы обеспечить проверку самых "удобных" правил в первую очередь. Критерий "удобности" составляется из трех составляющих: веса правила, частоты его срабатывания и времени его выполнения. Чем больше вес, частота срабатывания и меньше время выполнения, тем раньше будет проверено это правило.

- Классификатор - это алгоритм, обеспечивающий определение принадлежности сообщения к какому-либо классу. Класс определяется символом (например символ SPAM, имеющий вес 5 и символ HAM, имеющий вес -5). Принадлежность к классу обеспечивается либо статистически, путем разбора текста сообщения на токены и сравнения с известными токенами, хранящимися на диске в виде файла токенов (statfile), либо же иным алгоритмом (например, нейросетью). В результате работы классификатора определяется соответствие сообщения какому-либо классу и добавления соответствующего этому классу символа. Классификатор отличается от обычного модуля тем, что он не просто проверяет какие-либо характеристики сообщения, а сравнивает содержание сообщения с известными ему наборами. То есть, классификатор для его работы необходимо обучать на различных наборах. В настоящее время в rspamd реализован алгоритм классификации winnow и разбора на токены OSB. О них будет написано в дальнейшем.

Обработка осуществляется по следующей логике:

- для каждой метрики выбирается таблица символов и выбираются по очереди символы (по степени "удобности")

- для каждого символа вызывается соответствующее правило

- после вызова очередного правила проверяется, не превысил ли результат метрики порогового результата

- при превышении порога сообщение считается по этой метрике спамом и больше символов из этой метрики не проверяется
- для сообщения проверяется принадлежность к какому-либо классу для корректировки результата
- после определения принадлежности к классу происходит окончательный пересчет очков по метрике и при совпадении критериев автообучения происходит автообучение классификатора

После обработки сообщений для каждой из метрик выводится результат. Если используется протокол spamc, то считается только метрика "default", а дополнительные метрики добавляются как заголовки вида X-Spam-Status: metric; result. Для протокола rspamc выводятся результаты всех метрик, что позволяет настраивать различные группы правил и осуществлять фильтрацию сообщений не только как spam/ham, а задавать различные критерии оценки.

Глава 4

Настройка rspamd

4.1 Общие правила настройки

Файл конфигурации rspamd имеет следующий синтаксис:

```
param = value;
```

Точка с запятой является обязательной в конце каждой директивы. Некоторые директивы являются составными и обрамляются фигурными скобками, например:

```
section {
    param = value;
};
```

Также позволяется включать другие файлы (точка с запятой в конце директивы не нужна):

```
.include /path/to/file
```

В конфигурационном файле допускается определять и использовать переменные:

```
$var = "some text";
param = "${var}";
```

Приведенный фрагмент определяет переменную $var и присваивает параметру “param” значение “some text”. Переменные имеют глобальную область действия, обрамление переменных фигурными скобками при использовании (вида ${some_variable}) обязательно. Большинство строк конфигурационного файла обрамляется двойными кавычками. Одинарные кавычки применяются только при конфигурации модуля (это поведение подлежит пересмотру в следующих версиях):

```
.module 'name' {
    param = "value";
};
```

4.1.1 Определения списков

В rspamd многие параметры задаются в виде списков. Списки задаются ссылкой на файл или же http ресурс. Основное отличие таких файлов в том, что rspamd проверяет изменения в таких файлах (примерно раз в минуту, используя случайный разброс) и перегружает списки при их модификации. Таким же образом организована загрузка списков через http, только вместо modification time используется HTTP 1.1 заголовок If-Modified-Since, в ответ на который http сервер может выдать ответ 304: Not modified, в таком случае rspamd не перечитывает список. Списками задаются те параметры, которые могут содержать много значений и которые могут часто меняться. Для того, чтобы не приходилось выполнять перезапуск rspamd списки перечитываются по мере их обновления. Определения списков выглядят следующим образом:

- http список:

```
param = "http://test.ru:81/some/path.php";
param = "http://test.ru/some/other.txt";
```

- file список:

```
param = "file:///var/run/rspamd/some.file";
```

4.2 Общие параметры конфигурации

Общие параметры не принадлежат никакой секции и позволяют задавать общие настройки системы.

- pidfile - путь до PID-файла:

```
pidfile = "/var/run/rspamd.pid";
```

- statfile_pool_size - размер пула файлов статистики в памяти. Может быть с суффиксом, определяющим единицы измерение (по умолчанию байты): K - килобайты, M - мегабайты, G - гигабайты.

```
statfile_pool_size = 40M;
```

- raw_mode - если этот параметр равен “yes”, то rspamd не осуществляет перекодировку сообщений в utf8, в этом режиме проверка сообщений осуществляется быстрее, но при этом одинаковые сообщения в разных кодировках будут обрабатываться как разные.

```
raw_mode = yes;
```

- filters - строка, содержащая список включенных модулей, имена модулей разделяются запятыми и/или пробелами.

```
filters = "surbl,regexp,chartable,emails";
```

4.3 Настройка процессов

Данные секции служат для определения параметров рабочих процессов. Общие параметры рабочего процесса:

- type - тип рабочего процесса:
  - normal - обычный процесс обработки сообщений
  - controller - управляющий процесс
  - lmtp - процесс обработки сообщений по протоколу lmtp
  - fuzzy - хранилище хешей

  ```
  type = "normal";
  ```

- bind_socket - параметры слушающего сокета процесса, может определять либо tcp сокет, либо unix сокет:
  - host:port - осуществляет bind на указанные host и port
  - *:port - осуществляет bind на указанные port на всех локальных адресах
  - /path/to/socket - осуществляет bind на указанный unix socket

  ```
  bind_socket = localhost:11334;
  ```

- count - количество процессов данного типа. По умолчанию это число равно числу логических процессоров в системе.

  ```
  count = 1;
  ```

Для процессов типа "controller" можно также указать пароль для привиллегированных команд параметром password, а для процессов типа "fuzzy" необходимо указать путь к файлу, который будет использован как хранилище хешей параметром hashfile. Пример настройки рабочих процессов:

```
worker {
    type = "normal";
    count = 1;
    bind_socket = *:11333;
};
worker {
    type = "controller";
    bind_socket = localhost:11334;
    count = 1;
    password = "q1";
};
worker {
    type = "fuzzy";
    bind_socket = localhost:11335;
    count = 1;
    hashfile = "/tmp/fuzzy.db";
};
```

4.4 Настройки журналирования

Данные настройки определяют тип журналирования и его параметры.

- log_type - тип журналирования:
  - console - журналирование в stderr
  - syslog - журналирование через syslog
  - file - журналирование в файл

  ```
  log_type = console;
  ```

- log_level - уровень ведения журнала
  - DEBUG - журналирование отладочной информации
  - INFO - журналирование информационных событий
  - WARN - журналирование только предупреждений
  - ERROR - журналирование только ошибок

  ```
  log_level = INFO;
  ```

- log_facility - используется для журналирования в syslog и определяет назначение сообщений. Более подробно об этом можно узнать из man syslog.

  ```
  log_facility = "LOG_MAIL";
  ```

- log_file - используется для журналирования в файл и путь к файлу журнала.

  ```
  log_file = "/var/log/rspamd.log";
  ```

Пример настройки журналирования:

```
logging {
    log_type = file;
    log_level = INFO;
    log_file = "/var/log/rspamd.log"
};
```

4.5 Настройки метрики

Для настроек метрик используются секции “metric”. Основные параметры метрик:

- name - имя метрики.

  ```
  name = "default";
  ```

- required_score - минимальное число очков, необходимое, чтобы сообщение считалось спамом по данной метрике.

```
required_score = 10;
```

- cache_file - путь до файла, содержащего кеш символов метрики (используется, чтобы сохранить статистику "удобности" символов метрики, чтобы при перезапуске rspamd не терять накопленных данных).

```
cache_file = "/var/run/rspamd/metric.cache";
```

Пример настройки метрики:

```
metric {
    name = "default";
    required_score = 10.1;
    cache_file = "/tmp/symbols.cache";
};
```

4.6 Настройка классификаторов

Для настройки классификаторов используются секции "classifier". Общие настройки классфикатора:

- type - алгоритм классификатора (в настоящее время определен только "winnow").

```
type = "winnow";
```

- tokenizer - алгоритм разбиения сообщения на токены (в настоящее время определен только "osb-text").

```
tokenizer = "osb-text";
```

Также каждый классификатор может содержать определения классов и соответствующих им файлов токенов. Для этого используется подсекция statfile, содержащая следующие параметры:

- symbol - имя класса и имя символа, используемого для данного класса.

```
symbol = "WINNOW_SPAM";
```

- path - путь до файла.

```
path = "/var/run/rspamd/winnow.spam";
```

- size - размер данного файла. Также может иметь суффикс размерности.

```
size = 100M;
```

Внутри каждого определения класса можно использовать подсекцию autolearn, определяющую условия, при которых происходит автоматическое обучение данного класса. Секция имеет следующие параметры:

- min_mark - минимальное число очков, при котором осуществляется обучение.

```
min_mark = 10.1;
```

- max_mark - максимальное число очков, при котором осуществляется обучение.

```
max_mark = 0.1;
```

Автообучение происходит, если данное сообщение отвечает данным критериям. То есть, логично обучать класификатор HAM сообщениями, указав максимальное количество очков, близкое к нулю и SPAM сообщениями, указав минимальное число очков, близкое к срабатыванию триггера SPAM для данной метрики. Таким образом, классифицируемые как спам сообщения обучают класс SPAM, а классифицируемые как HAM (то есть, на них не сработали правила метрики) - обучают класс HAM.

Пример определения классификатора:

```
classifier {
    type = "winnow";
    tokenizer = "osb-text";
    statfile {
        symbol = "WINNOW_SPAM";
        path = "/tmp/test.spam";
        size = 10M;
        autolearn {
            min_mark = 10.0;
        };
    };
    statfile {
        symbol = "WINNOW_HAM";
        path = "/tmp/test.ham";
        size = 10M;
        autolearn {
            max_mark = 0.1;
        };
    };
};
```

4.7 Настройка коэффициентов символов

Для настройки коэффициентов применяется секция “factors”. Данная секция состоит из набора определений вида

```
"СИМВОЛ" = вес;
```

например:

```
"R_UNDISC_RCPT" = 5;
"MISSING_MID" = 3;
"R_RCVD_SPAMBOTS" = 3;
"R_TO_SEEMS_AUTO" = 3;
"R_MISSING_CHARSET" = 5;
```

Также существует возможность создавать «составные» символы - символы, которые являются комбинацией других символов. Это нужно для возможности указывать, что комбинация определенных символов имеет больший (или, наоборот, меньший) вес, чем сумма весов нескольких символов. Составные символы представляют собой логические выражения из других символов, например:

```
composites {

    COMPOSITE_SYMBOL1 = «SYMBOL1 & (SYMBOL2 | SYMBOL3)»;
    COMPOSITE_SYMBOL2 = «SYMBOL3 & !SYMBOL4»;

};
```

При добавлении составного символа все символы, входящие в него, удаляются из результата. То есть, при срабатывании COMPOSITE_SYMBOL1 из предыдущего примера символы SYMBOL1, SYMBOL2 и SYMBOL3 в ответе не появятся.

Глава 5

Настройка модулей

5.1 Настройка модуля surbl

Модуль surbl служит для проверки URL'ей в письме на различных "черных" списках. Модуль делает следующее: для каждого из url, найденных в сообщении, извлекает доменный компонент (2-го или 3-го уровня), добавляет суффикс имени surbl и делает dns запрос. При успешном определении такого имени добавляется символ. Пример работы:

```
URL (http://some.test.ru/index.html) -> test.ru + (insecure-bl.rambler.ru) ->
resolve test.ru.insecure-bl.rambler.ru -> 127.0.0.1 -> add symbol
```

Параметры настройки:

```
.module 'surbl' {
    # Определение суффикса SURBL
    # Символы '%b' заменяются на значение определенного бита
    suffix_%b_SURBL_MULTI = "multi.surbl.org";
    # Суффикс для каждого из бит
    bit_2 = "SC"; # sc.surbl.org
    bit_4 = "WS"; # ws.surbl.org
    bit_8 = "PH"; # ph.surbl.org
    bit_16 = "OB"; # ob.surbl.org
    bit_32 = "AB"; # ab.surbl.org
    bit_64 = "JP"; # jp.surbl.org
    # Имя метрики
    metric = "default";
    # Список доменов, для которых необходимо использовать 3 доменных
    # компонента, вместо двух
    2tld = "file:///etc/rspamd/2tld.inc";
    # Список URL'ей, которые не будут проверяться этим модулем
    whitelist = "file:///etc/rspamd/surbl-whitelist.inc";
};
```

Некоторые пояснения по данной конфигурации. Модуль SURBL может осуществлять проверку битов в полученном от DNS сервера ответе, и вставлять соответствующий символ. Это используется для проверки сразу нескольких списков одним DNS запросе. Тогда ответ сервера содержит списки, в которых встретился данный URL в виде битов адреса. Более подробно с этим можно ознакомиться тут: http://www.surbl.org/lists.html#multi. Список 2tld используется для задания списка доменов, для которых необходимо проверять не два уровня доменного имени, а три. Например, это актуально для виртуальных хостингов или же специальных зон для доменов третьего уровня, например org.ru или pp.ru.

5.2 Настройка модуля regexp

Модуль regexp является очень важным в работе rspamd, так как определяет все правила фильтрации сообщений по регулярным выражениям. Модуль работает с логическими выражениями из регулярных выражений, поэтому его настройка выглядит достаточно запутанной. Однако, если пользоваться переменными, то логика работы становится более понятной. При настройке самого модуля используются простые директивы вида:

ИМЯ_СИМВОЛА = "логическое выражение"

Само логическое выражение содержит различные регулярные выражения и функции, объединенные символами логики:

- & - логическое “И”
- | - логическое “ИЛИ”
- ! - логическое отрицание

Приоритет операций может изменяться скобками, например:

```
A & B | C - выполняется слева направо A & B затем | C
A & (B | C) - выполняется как (B | C) затем & A
```

Сами регулярные выражения совместимы с perl regular expressions. Их синтаксис можно изучить в соответствующей литературе: http://perldoc.perl.org/perlre.html. У rspamd есть дополнительные флаги, определяющие, в какой части сообщения искать заданное регулярное выражение:

- r - "сырой" незакодированный в utf8 regexp
- H - ищет по заголовкам сообщения
- M - ищет по всему сообщению (в "сыром" виде, то есть без mime декодинга)
- P - ищет по всем текстовым mime частям
- U - ищет по url
- X - ищет по "сырым" хедерам (опять же без декодирования)

Если в регулярном выражении встречаются символы двойной кавычки (“) или же слэша (/), то их необходимо экранировать обратным слэшем (при этом сам обратный слэш экранировать необязательно):

```
\" \/
```

Для поиска по заголовкам формат регулярного выражения несколько меняется:

```
Имя_заголовка=/регулярное_выражение/H
```

При поиске по заголовкам происходит поиск заголовков с таким именем и сравнение их значений с регулярным выражением, пока это выражение не будет найдено, либо пока не будут проверены все заголовки с таким именем. Для multipart сообщений происходит поиск заголовков по всем частям сообщения. Это справедливо для всех функций, работающих с заголовками. Поиск по “сырым” заголовкам происходит без учета mime частей - только по заголовкам самого сообщения. При этом, хотя и не происходит декодирования заголовков, но происходит их де-фолдинг (фолдинг - перенос заголовков по строчкам). Модуль regexp также может использовать внутри логических выражений встроенные функции rspamd. Встроенные функции всегда возвращат логическое значение (истина или ложь) и могут принимать аргументы (в том числе аргументы, являющиеся логическими выражениями). Список встроенных функций:

- header_exists - принимает в качестве аргумента имя хедера, возвращает true, если такой заголовок существует
- compare_parts_distance - принимает в качестве аргумента число от 0 до 100, которое отражает разницу в процентах между частями письма. Функция работает с сообщениями, содержащими 2 текстовые части (text/plain и text/html) и возвращает true тогда, когда эти части различаются более чем на n процентов. Если аргумент не указан, то по умолчанию ищется различие в 100% (полностью разные части).
- compare_transfer_encoding - сравнивает Content-Transfer-Encoding с заданной строкой
- content_type_compare_param - сравнивает параметр content-type заголовка с регулярным выражением или строкой:

  ```
  content_type_compare_param(Charset, /windows-\d+/)
  content_type_compare_param(Charset, ascii)
  ```

- content_type_has_param - проверяет, есть ли в заголовке content-type определенный параметр
- content_type_is_subtype - сравнивает подтип content-type с регулярным выражением или строкой
- content_type_is_type - сравнивает тип content-type с регулярным выражением или строкой

```
content_type_is_type(text)
content_type_is_subtype(/?.html/)
```

- regexp_match_number - принимает в качестве первого параметра число, которое означает порог сработавших регэкспов и список регэкспов или функций, которые должны проверяться. Если число сработавших регэкспов или функций больше порога, функция возвращает TRUE, иначе - FALSE, например:

```
regexp_match_number(2, ${__RE1}, ${__RE2}, header_exists(Subject))
```

- has_only_html_part - функция возвращает TRUE, если в сообщении есть только одна HTML часть
- compare_recipients_distance - вычисляет процент схожих получателей письма. Принимает аргумент - порог в процентах похожести.
- is_recipients_sorted - возвращает TRUE, если список получателей сортирован (работает только если число получателей $>= 5$).
- is_html_balanced - возвращает TRUE, если теги всех html частей сбалансированы
- has_html_tag - возвращает TRUE, если заданный html тег найден

Данные функции были созданы для решения задач, которые сложно или же невозможно решить при помощи обычных регулярных выражений. При конфигурации модуля regexp целесообразно определить все логические выражения в отдельных переменных, подключить их при помощи директивы .include и задавать символы как:

```
СИМВОЛ="${переменная}";
```

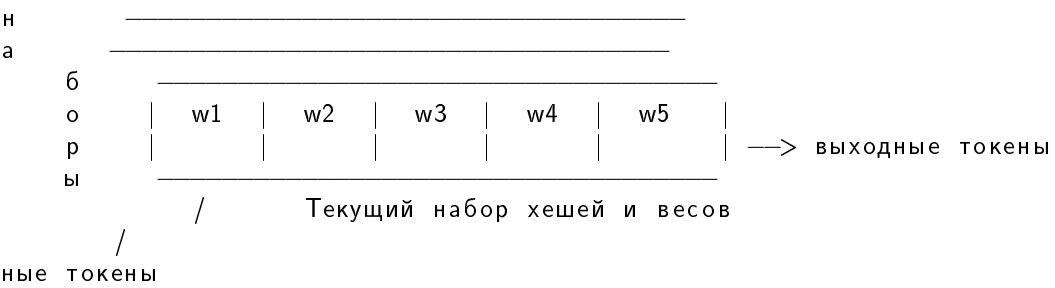
иначе конфигурация модуля будет практически нечитаемой из-за обилия регулярных выражений.

Глава 6

Статистические алгоритмы

6.1 Winnow и OSB

В rspamd используется алгоритм ортогональных разреженных биграмм (OSB), который основан на следующем принципе:



То есть, процесс преобразования можно представить следующим образом: для каждого набора весов (w1..w5) составляется набор хешей. Токены образуются из текста. Например, возьмем некое письмо и наложим на него окно:

```
"Мама мыла раму."
|________|
```

В данном окне создаются 2 токена:
h("Мама"), h("мыла"), где h - хеш функция.
Дальше окно двигается вправо на один токен и опять создаются 2 токена: h("мыла"), h("раму")
В rspamd используется окно в 5 токенов и используются пары:

```
1 - 5 -- h1
2 - 5 -- h2
3 - 5 -- h3
4 - 5 -- h4
```

Каждый такой токен состоит из двух хешей (h1 и h2). То есть каждое слово текста может давать до 5-ти токенов. Это делается для того, чтобы в статистических алгоритмах учитывать не индивидуальные слова, и их сочетания, чтобы уменьшить ошибку.

После этого мы должны вычислить принадлежность потока выходных токенов к некоторому классу. Для этого используется алгоритм Winnow. Идея алгоритма очень проста:

1. Каждый возможный входной токен имеет вес 1.0 (то есть, нас интересуют только те токены, которые не равны 1.0)
2. Для обучения проделываем следующие шаги:
   (a) генерируем набор токенов путем OSB алгоритма
   (b) удаляем все дупликаты
   (c) если данный входной набор принадлежит классу (например, спам или неспам), то умножаем вес каждого встреченного токена на т.н. Promotion Constant, которая равна 1,23
   (d) если данный входной набор не принадлежит классу, то умножаем каждый найденный токен на Demotion Constant в данном классе, которая равна 0,83
   (e) абсолютно неважно, сколько раз встречался данный токен во входном потоке, мы его умножаем на promotion или demotion только один раз
3. Для классификации потока мы поступаем следующим образом:
   (a) генерируем набор токенов путем OSB алгоритма
   (b) удаляем все дупликаты
   (c) суммируем веса всех токенов, найденных в каждом из файлов данных статистики (при этом те токены, которые мы не нашли, имеют вес 1)
   (d) затем мы делим полученную сумму на число токенов и смотрим, какой из классов (файлов данных) набрал больше очков и делаем заключение о принадлежности входного текста к классу

Файлы данных статистики представляют собой следующие структуры:

```
{
Header,
{ feature_block1..feature_blockN }
}
```

Заголовок файла очень прост:

```
struct {

    char magic[3] = { 'r', 's', 'd' };
    u_char version[2] = { '1', '0' };
    uint64_t create_time;
```

```
}
```

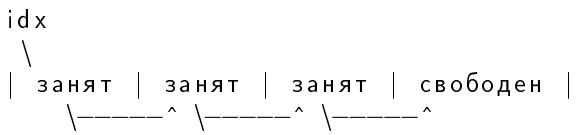
Каждый feature_block состоит из 4-х полей:

```
struct {
    uint32_t hash1;
    uint32_t hash2;
    float value;
    uint32_t last_access;
}
```

Итого 16 байт на каждый feature. 0-е значения показывают свободную ячейку. Значение hash1 используется в качестве индекса:

```
idx = hash1 % filesize;
```

Где filesize - размер в количестве feature_block'ов. При этом данный токен должен помещаться в заданную ячейку или ячейку за ним. При этом образуется цепочка токенов:



При этом, длина такой цепочки должна быть лимитирована некоторым разумным числом, например 128. Тогда максимальное время доступа будет не более 128-и итераций. Если мы не нашли за 128 итераций свободную ячейку, то мы можем поместить новый токен на место того, который меньше всего использовался (min (last_access)). При этом при доступе к ячейке необходимо обновлять last_access:

last_access = now - creation_time.

Такая организация позволяет замещать только наименее используемые токены.

Глава 7

Протокол rspamc

Формат ответа:

```
SPAMD/1.1 0 EX_OK
\ /      \/
Версия Код ошибки
Spam: False ; 2 / 5
```

Это формат совместимости с sa-spamd (без метрик). Новый формат ответа:

```
RSPAMD/1.0 0 EX_OK
Metric: Name; Spam_Result; Spam_Mark / Spam_Mark_Required
Metric: Name2 ; Spam_Result2 ; Spam_Mark2 / Spam_Mark_Required2
```

Заголовков типа metric может быть несколько. Формат вывода символов:

```
SYMBOL1, SYMBOL2, SYMBOL3 -- формат совместимости с sa-spamd
Symbol: Name; Param1,Param2,Param3 -- формат rspamd
```

Формат ответа зависит от формата запроса:

```
PROCESS SPAMC/1.2
\ /        \ /
Команда   Версия
```

В любом из режимов работы поддерживаются следующие заголовки:

- Content-Length - длина сообщения
- Helo - HELO, полученный от клиента
- From - MAIL FROM
- IP - IP клиента
- Recipient-Number - число реципиентов
- Rcpt - реципиент
- Queue-ID - идентификатор очереди

Эти значения могут использоваться в фильтрах rspamd.

Глава 8

Клиент rspamc

Клиент rspamc представляет собой программу, написанную на perl и предназначенную для работы с системой rspamd. Rspamc принимает следующие аргументы:

- -c: определяет путь к конфигурационному файлу rspamd, используется для работы с локальным rspamd
- -h: определяет адрес удаленного rspamd сервера
- -p: определяет порт для удаленного rspamd сервера
- -P: определяет пароль для работы с привиллегированными командами rspamd
- -s: определяет имя символа для обучения классификатора

Последним аргументом rspamc принимает команду. Если команда не задана, то используется команда SYMBOLS. Команды, принимаемые rspamc:

- команды по обработке сообщений:
  - symbols - по данной команде проверяется сообщение, переданное через stdin rspamc
  - check - по данной команде выводится только результат по метрикам без символов
  - process - возвращает не только символы, но и исходное сообщение
  - urls - выводит все найденные url'и
  - emails - выводит все найденные адреса e-mail в сообщении
- команды по работе с управляющим интерфейсом
  - stat - выводит статистику работы
  - learn - обучает классификатор по определенному классу (указанному опцией -s)
  - shutdown - останавливает систему rspamd
  - uptime - выводит время работы rspamd

- counters - выводит значения счетчиков символов
- fuzzy_add - добавляет fuzzy hash в хранилище
- fuzzy_del - удаляет fuzzy_hash из хранилища

Глава 9

LUA API плагинов

Rspamd позволяет реализовывать различную логику в виде lua плагинов, для чего используется директива modules. Данная директива позволяет задавать пути к каталогам, содержащим скрипты на lua:

```
modules {
    module_path = «/some/path/»;
};
```

При инициализации rspamd загружает все файлы вида *.lua и выполняет их (при ошибке в коде плагинов rspamd запускаться не будет, выдавая ошибку конфигурации, при указанной опции -t будет проверяться не только синтаксис конфигурационного файла, но и синтаксис плагинов). При этом, определяется глобальная переменная rspamd_config, позволяющая извлекать опции конфигурации и регистрировать правила и соответствующие им символы. Таким образом, каждый lua плагин условно можно разделить на две части: исполняемый код, выполняющий настройку опций модуля, регистрирующий функции правил (callbacks), и собственно обработчики правил.

9.1 Настройка lua модуля

Для извлечения параметров конфигурации и регистрации обработчиков правил применяется глобальная переменная rspamd_config, которая обладает рядом полезных методов:

- get_module_opt (module_name, option_name) - возвращает значение опции option_name для модуля с именем module_name. То есть, если в конфигурационном файле есть следующая запись:

  ```
  module 'test' {
      param = «value»;
  };
  ```
  То вызов rspamd_config:get_module_opt('test', 'param')
  вернет строку 'value';

- get_all_opts (module_name) - возвращает таблицу из всех опций для данного модуля, ключом служит имя опции:

```
local opts = rspamd_config:get_all_opts('test')
if opts then
    var = opts['param']
    for k,v in pairs opts do

        print («Param: » .. k .. « Value: » .. v)
    end
end
```

- get_metric (name) - возвращает объект метрики с данным именем

Объект метрики используется для регистрации символов:

- register_symbol (symbol, initial_weight, callback) - symbol определяет имя символа, initial_weight - изначальный вес, callback - строка с именем функции:

```
local m = rspamd_config:get_metric('default')
if m then
    m:register_symbol('TEST', 1.0, 'some_callback')
end
```

После регистрации обработчика символа этот обработчик будет вызываться rspamd обычным образом, используя планировщик символов (это подробно описано в 3.2).

9.2 Обработчик правила

Обработчик правила - это функция, реализующая логику правила. В качестве параметра она принимает объект task, из которого можно извлечь различную информацию о сообщении. После выполнения логики работы обработчик может вставить символ, используя тот же объект task. Таким образом типичная функция-обработчик выглядит следующим образом:

```
function some_callback(task)
    if some_condition(task) then
        task:insert_result(metric, symbol, 1)
    end
end
```

Функция insert_result принимает в качестве параметров имя метрики, имя символа, вес и необязательный список строковых параметров, которые будут ассоциированы с этим символом. Объект task предоставляет ряд функций, позволяющих создать логику правила фильтрации:

- get_received_headers() - возвращает массив из обработанных заголовков Received в виде таблицы:
  - h['from_hostname'] - hostname, откуда получено сообщение
  - h['from_ip'] - ip, откуда получено сообщение
  - h['real_hostname'] - hostname, распознанное самим релеем
  - h['real_ip'] - ip, который соответствует real_hostname
  - h['by_hostname'] - hostname самого релея

  received заголовки в массиве идут в обратном порядке, то есть, первые релеи письма будут первыми элементами массива
- get_raw_headers() - возвращает строку, содержащую все заголовки сообщения в неразобранном виде
- get_text_parts() - возвращает массив объектов типа text_part
- get_urls() - возвращает массив строк, содержащих извлеченные из сообщения URL'и
- get_message() - возвращает объект типа сообщение

Объект «message» применяется для манипуляций с заголовками:

- get_header(headername) - возвращает массив всех значений заголовков с таким именем (может быть массив из одного элемента, если такой заголовок в сообщении представлен единожды)
- set_header(headername, headervalue) - устанавливает заданный заголовок

Объект text_part предназначен для манипуляций с текстовым содержимым сообщения:

- get_content() - возвращает текстовое содержимое части
- is_html() - возвращает true, если данная часть представляет собой HTML
- is_empty() - возвращает true, если данная часть не содержит текста
- get_fuzzy() - возвращает строку с нечетким хешем для данного сообщения

Пример функции-обработчика:

```
function received_cb (task)
    local recvh = task:get_received_headers()
    for _,rh in ipairs(recvh) do
        if rh['real_ip'] then

            local parts = task:get_text_parts()
            for _,part in ipairs(parts) do
              if not part:is_empty() then
```

```
                local text = part:get_content()
                if some_filter(text) then
                task:insert_result(metric, symbol, 1)
                end
              end
            end
        end
    end
end
```

9.3 Использование DNS

Для многих задач фильтрации сообщений используются различные DNS запросы, поэтому lua интерфейс предоставляет возможность планирования DNS запросов, используя объект task:

- resolve_dns_a(host, callback) - выполняет прямое преобразование имени host, после чего вызывает обработчик callback
- resolve_dns_ptr(ip, callback) - то же, что и предыдущее, но выполняет обратное преобразование ip

Так как DNS преобразования осуществляются асинхронно, то необходимо задавать обработчик, который будет вызываться по завершению DNS запроса. Обработчик имеет следующий вид:

```
function dns_cb(task, to_resolve, results, err)
```

- task - объект task
- to_resolve - строка, содержащая имя хоста или ip, для которого выполнялось преобразование
- results - массив, содержащий ip адреса (для прямого преобразования) или же имя хоста (для обратного). В случае ошибки этот параметр имеет значение nil
- err - строка, описывающая ошибку (или nil, если преобразование успешно завершилось)

Пример функции-обработчика результатов DNS запроса:

```
function dns_cb(task, to_resolve, results, err)
    if results then

        local _,_,rbl = string.find(to_resolve, '%d+\.%d+\.%d+\.%d+\.(.+)')
        task:insert_result(metric, symbol, 1, rbl)
    end
end
```

Глава 10

Использование HTTP Redirector

TODO

Глава 11

Хранилище нечетких хешей

TODO