3 SpamAssassin

In this chapter, we introduce SpamAssassin, which is a widely used interface to other anti-spam mechanisms as well a spam classifier in its own right. SpamAssassin has its own rules base, which is used to assign a "score" to each mail message. The methods SpamAssassin uses to classify whether email is spam are as follows:

- **Header analysis**—SpamAssassin can analyze these headers of an email message and generate a score based on them.
- **Body analysis**—The body of a spam message often contains phrases (i.e., "Reduce your debt!") that can easily identify it as spam.
- **Bayesian analysis**—Bayes is a statistical methodology where both the headers and body of a message are compared against a known database of phrases. This book dedicates several chapters to Bayesian analysis due to its effectiveness.
- **Distributed checksums** (Vipul's Razor and DCC)—In this process, a message is checksummed and put into a distributed database along with a count. The higher the count for the checksum associated with the message, the more likely the message is spam.
- **Blackhole listing checking** (MAPS RBL, etc.)—Blackhole lists are ranges of IP addresses where spam originates. If a message comes from such a network, the message score can be adjusted accordingly.
- **Automatic whitelisting/blacklisting** (AWL)—SpamAssassin can automatically add email addresses to databases of whitelists or blacklists.
- **Manual whitelisting/blacklisting**—The software can accept/reject known good/bad email addresses.

SpamAssassin rules can be defined by the administrator and by the user if certain conditions are met. SpamAssassin's power is evident when the individual scores are summed—the final score is worth much more than the individual scores themselves. After SpamAssassin has "scored" a message, the tool can do a number of things:

- Add headers (for example, a spam score like `X-SpamAssassin-Score: 40`)
- Modify existing mail headers (i.e., change the subject line)
- Send a new message regarding the spam (i.e., report the spam to someone)
- Submit the spam to a database for tracking purposes
- Send a message to the sender (spamtrap-related keywords)

Arguably the most useful function in SpamAssassin is the act of scoring the message and adding an appropriate header indicating the score. Adding a score enables email clients (such as Outlook, Mozilla Messenger, etc.) to filter mail into folders as the end user wishes. However, setting up and training end users takes considerable time and effort from the administrator.

Our coverage of SpamAssassin is designed for Unix-based systems. If you are interested in running SpamAssassin on a Microsoft Windows product such as XP, you should read the writeup available at *http://www.openhandhome.com/howtosa260.html*. This web page covers the largely manual process for making SpamAssassin work on MS Windows platforms.

SpamAssassin can be set up to work at mail delivery time (when mail is being written to the users' mailbox, often called "per-user") or transfer time (when mail is coming into the system). Both are useful in the fight against spam, although normally the administrator picks one approach per system. Per-user checking is good for a small number of users and for working out bugs. Figure 3.1 shows a possible SpamAssassin integration with a mail system on a per-user basis.

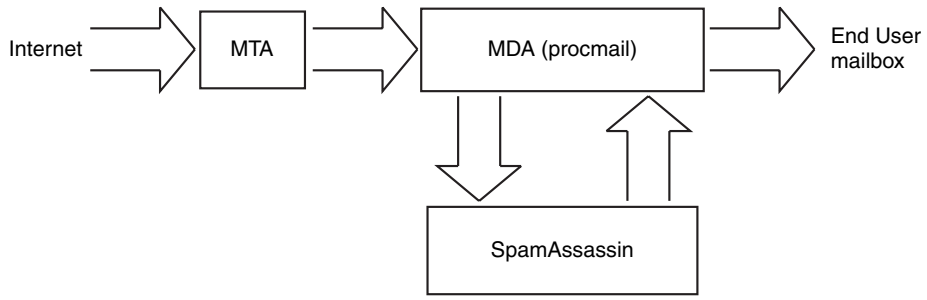

Figure 3.1 SpamAssassin deployed per user.

When you are happy with the per-user installation, you can roll the setup system-wide and implement SpamAssassin for all user accounts at mail transfer time. SpamAssassin documentation refers to this mode as "site-wide," but that is a bit of a misnomer, which is why this book will use the term interchangeably with "system-wide." Figure 3.2 illustrates SpamAssassin integration at MTA time (system-wide).

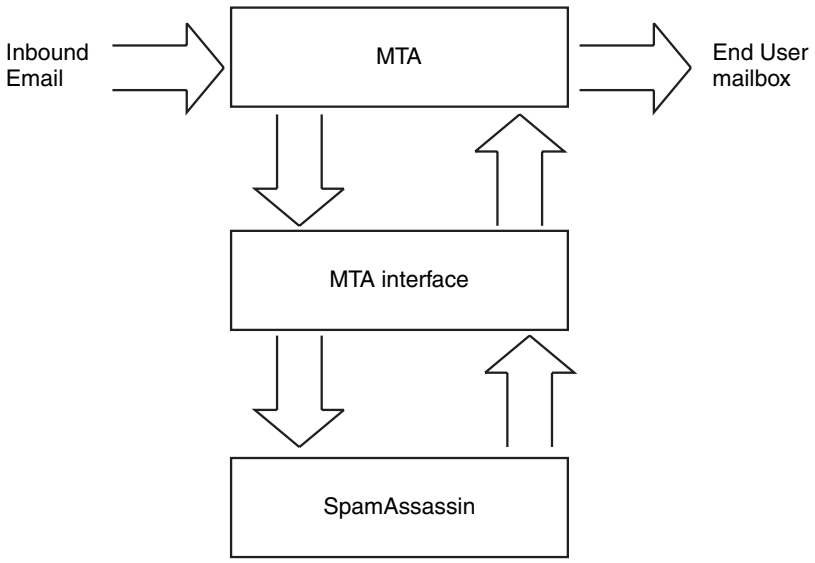

Figure 3.2 SpamAssassin deployed system-wide.

The box labeled *MTA interface* in Figure 3.2 is the mechanism that calls SpamAssassin, which is the topic of the next section.

SPAMASSASSIN AND MTA INTEGRATION

Before we get to the actual installation of SpamAssassin, it is useful to cover how to integrate it into your environment. Each MTA has its own way of adding in third-party components like SpamAssassin. In addition, each MTA itself can have several methods of integrating SpamAssassin into it.

SpamAssassin is often integrated into a large email infrastructure via an MTA filter such as milter, MIMEdefang, and Qmail-Scanner. These programs enable a number of actions on messages, including anti-virus, filtering, and, of course, anti-spam. The other

way to integrate SpamAssassin into your email system is to use a facility such as Procmail, which can be enabled on a per-user basis. Using Procmail might be good for initial testing or for use on a small site. Although this approach will certainly work, the per-user `procmail` method requires more work on the administrator's part. Any site with more than a handful of users is probably going to want to use a site-wide method.

In the case of Sendmail, there is a standard facility called Sendmail Mail Filter (or milter), which allows administrators to call programs to perform desired functions (including filtering and spam detection), which are not part of Sendmail itself. We use milter and MIMEdefang as the method of integration between Sendmail and SpamAssassin.

With Postfix, the options are more varied. SpamAssassin integration methods could include MailScanner, MIMEdefang, or amavisd-new, among others. We chose to use amavisd-new as our Postfix integration method.

Regarding qmail, the options are fewer. We chose the Qmail-Scanner software as our integration. As with other MTAs utilizing Procmail for individual users, qmail's `.qmail` function could be used to call a script to invoke SpamAssassin. This would be good for a small organization, but larger sites will want to use something like Qmail-Scanner.

Table 3-1 summarizes what we use in this chapter for implementing SpamAssassin from each MTA and where to get more information.

Table 3-1

MTA	Interface(s)	More Information
Sendmail	MIMEDefang	http://www.mimedefang.org/
	milter	http://www.milter.org/
Postfix	amavisd-new	http://www.ijs.si/software/amavisd/
qmail	Qmail-Scanner	http://qmail-scanner.sourceforge.net/
	maildrop	http://www.courier-mta.org/maildrop/

INSTALLING SPAMASSASSIN

Before getting to the specifics of installing the supporting software for each MTA, we first cover the installation of SpamAssassin. If SpamAssassin is installed first, several of the packages will automatically determine that SpamAssassin has been installed and adjust their configuration.

The first step is to download the sources, which are available from the SpamAssassin site: *http://useast.spamassassin.org/released/Mail-SpamAssassin-2.63.tar.gz*. After you have

downloaded them to a location such as `/usr/local/src`, uncompress and extract the files as follows:

```
bash$ gzip -d Mail-SpamAssassin-2.63.tar.gz
bash$ tar xf Mail-SpamAssassin-2.63.tar.gz
bash$ cd Mail-SpamAssassin-2.63
```

To install SpamAssassin, perform the following:

```
bash$ perl ./Makefile.PL
What email address or URL should be used in the suspected-spam report
text for users who want more information on your filter installation?
(In particular, ISPs should change this to a local Postmaster contact)
default text: [the administrator of that system] user@mydomain.com

Checking if your kit is complete...
Looks good
Writing Makefile for Mail::SpamAssassin
Makefile written by ExtUtils::MakeMaker 6.03

bash$ make
bash$ sudo su
# make install
```

SpamAssassin is now installed. Configuration of SpamAssassin is covered later in this chapter.

SPAMASSASSIN AND SENDMAIL

To give you an idea of how SpamAssassin is integrated into Sendmail, Figure 3.3 illustrates the flow of email through a Sendmail system filtered with SpamAssassin and MIMEDefang on a system-wide basis.

The first step is to download and install SpamAssassin. After that is accomplished, milter and MIMEDefang must be installed to complete the installation.

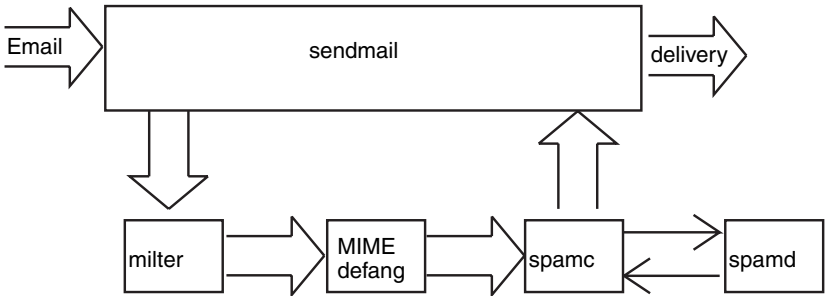

Figure 3.3 SpamAssassin and Sendmail.

Installing milter

The next task is to ensure that milter is installed. Under most Linux/Unix distributions, milter is not enabled, so Sendmail must be recompiled with milter support. To check whether your version of Sendmail was compiled with milter support, run this command: *path-to-sendmail*`/sendmail -bp -d0.4 | grep MILTER`. If the command returns a blank, then you need to install milter. Otherwise, you can skip this section.

We start off by installing milter. In order to enable Sendmail milter functionality, Sendmail must be recompiled. Let's assume the Sendmail sources are located in the `/usr/local/src/sendmail-8.12.10` directory. Create a file in the `devtools/Site` directory of the Sendmail sources called `site.config.m4` with the following contents:

```
dnl Milter
APPENDDEF(`conf_sendmail_ENVDEF', `-DMILTER')
APPENDDEF(`conf_libmilter_ENVDEF', `-D_FFR_MILTER_ROOT_UNSAFE')
```

These lines tell Sendmail to run milter and the environment for its execution. After making the changes to the configuration file, rebuild Sendmail by executing `./Build.sh` from the root of the Sendmail source directory. Install Sendmail by executing the `./Build.sh install` command, and you now have a Sendmail binary that supports milter.

Installing MIMEDefang

The final step is to install MIMEDefang. It is important to note that the MIMEDefang installation automatically detects the SpamAssassin installation and will configure itself appropriately. So, installing SpamAssassin prior to MIMEDefang is a time-saver.

There are two aspects to installing MIMEDefang. MIMEDefang depends upon a number of Perl modules. The developers of MIMEDefang have made a nice package of all the required Perl modules, which can be installed in one shot. Point your browser to *http://www.mimedefang.org/node.php?id=1* and download the `MIME-tools-5.411a-RP-Patched-02.tar.gz` sources. Then install them like this:

```
bash$ tar xzvf MIME-tools-5.411a-RP-Patched-02.tar.gz

bash$ cd MIME-tools-5.411a-RP-Patched-02

bash$ perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for MIME-tools

bash$ make
bash$ sudo su
# make install
```

The second step is to download the MIMEDefang sources from a repository. A good choice would be the MIMEDefang page at *http://www.mimedefang.org/node.php?id=1*. After downloading, the package is installed as follows:

```
bash$ tar xzf mimedefang-2.39.tar.gz
bash$ cd mimedefang-2.39
bash$ ./configure

bash$ sudo groupadd defang

bash$ sudo useradd -c 'MIMEDefang user' -d /var/empty -s /bin/false defang

bash$ make
bash$ sudo su
# make install

# mkdir /var/spool/MIMEDefang
# chmod 700 /var/spool/MIMEDefang

# cp -p examples/init-script /usr/local/bin/mimedefang.sh

# chown defang.defang /usr/local/bin/mimedefang.sh
```

The configuration file `mimedefang-filter` that the installation package includes is reasonable. However, you might need to change some of the parameters, including the following.

To change the email address and name to where notifications go:

```
$AdminAddress = 'postmaster@mydomain.com';
$AdminName = "MIMEDefang Administrator's Full Name";
```

To change the email address MIMEDefang uses to send email, change this:

```
$DaemonAddress = 'mimedefang@mydomain.com';
```

If you want warnings as part of the message instead of an attachment (default is 0), set this variable to 1:

```
$AddWarningsInline = 1;
```

The default action is to send logs via email:

```
md_graphdefang_log_enable(mail,1);
```

Activating MIMEDefang/SpamAssassin

To activate SpamAssassin, edit `sendmail.mc` from your Sendmail source directory to include the following line:

```
INPUT_MAIL_FILTER(`mimedefang', `S=unix:/var/spool/MIMEDefang/mimedefang.sock,
➥F=T, T=S:1m;R:1m')
```

This line tells Sendmail to invoke MIMEDefang, which will in turn call SpamAssassin. The Sendmail configuration file is built by running the following command while in the `$SRC/cf/cf` directory:

```
# m4 ../m4/cf.m4 sendmail.mc > sendmail.cf
```

The resulting `sendmail.cf` file can be installed in `/etc/mail` and the server restarted by issuing `/etc/init.d/sendmail restart`.

SPAMASSASSIN AND POSTFIX

The flow of mail under Postfix integrated with SpamAssassin is diagrammed in Figure 3.4 for a system-wide basis setup.

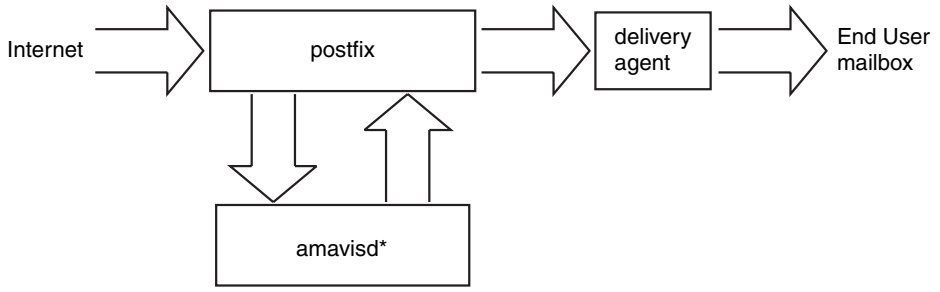

Figure 3.4 SpamAssassin and Postfix.

SpamAssassin is activated under Postfix by using the amavisd-new package. More information is available online for amavisd-new at *http://www.ijs.si/software/amavisd/*. amavisd-new calls the SpamAssassin libraries directly, making the installation a bit simpler than qmail and Sendmail, which both require additional pieces of software to activate SpamAssassin. Calling the SpamAssassin libraries directly also saves some overhead because additional system resources are not required if `spamc` and/or `spamd` are not invoked.

Installing amavisd-new

amavisd-new requires a number of Perl modules to be installed on the target. The `INSTALL` file notes regarding prerequisites from amavisd-new are as follows:

```
Archive::Tar   (Archive-Tar-x.xx)
Archive::Zip   (Archive-Zip-x.xx) (1.09 or later is recommended!)
Compress::Zlib (Compress-Zlib-x.xx)
Convert::TNEF  (Convert-TNEF-x.xx)
Convert::UUlib (Convert-UUlib-x.xxx)
MIME::Base64   (MIME-Base64-x.xx)
MIME::Parser   (MIME-Tools-x.xxxx)
(the patched MIME-tools by David F. Skoll is recommended over 5.411,
  as it better handles broken/bad MIME syntax:
    http://www.mimedefang.org/ -> Download section.
  The new 6.2xx from http://search.cpan.org/dist/MIME-tools/
  also includes these patches, and more.
Mail::Internet (MailTools-1.58 or later have workarounds for Perl 5.8.0 bugs)
Net::Server    (Net-Server-x.xx)
Net::SMTP      (libnet-x.xx)     (use libnet-1.16 or later for performance)
Digest::MD5    (Digest-MD5-x.xx)
```

```
IO::Stringy    (IO-stringy-x.xxx)
Time::HiRes    (Time-HiRes-x.xx) (use 1.49 or later; some older cause problems)
Unix::Syslog   (Unix-Syslog-x.xxx)
```

Make sure all of these Perl modules are installed on the target system. If any are missing, download and install them from *http://www.cpan.org*.

Unfortunately, there is no install script, so the software must be installed and configured manually. To begin, download the amavisd-new sources from *http://www.ijs.si/software/amavisd/amavisd-new-20030616-p6.tar.gz* in a directory, such as `/usr/local/src`. Extract it and `change directory` into the directory by running:

```
# gzip -d amavisd-new-20030615-p6.tar.gz
# tar xvf amavisd-new-20030615-p6.tar.gz
# cd amavisd-new-20030616
```

Then create a directory under `/var` called `amavis` as the `amavisd` home directory:

```
# mkdir /var/amavis
```

Create the group `amavis` and user `amavis`:

```
# groupadd amavis
# useradd -c 'Amavis Daemon' -d /var/amavis -g amavis -s /bin/false amavis
```

Make the permissions and ownership correct on the directory `/var/amavis`:

```
# chown amavis:amavis /var/amavis
# chmod 750 /var/amavis
```

Copy the `amavisd` executable to `/usr/local/sbin` and change the permissions appropriately:

```
# cp amavisd /usr/local/sbin/
# chown root /usr/local/sbin/amavisd
# chmod 755  /usr/local/sbin/amavisd
```

Copy the `amavisd.conf` configuration file to its default location, `/etc`, and make the permissions correct:

```
# cp amavisd.conf /etc/
# chown root /etc/amavisd.conf
# chmod 644  /etc/amavisd.conf
```

(If you change the location, you must start up `amavisd` with the `-c` option to tell it where to read its configuration from.)

Next, you must create the `quarantine` directory (where amavisd stores viruses that are caught) and set the permissions and ownership:

```
# mkdir /var/quarantine
# chown amavis:amavis /var/quarantine
# chmod 750 /var/quarantine
```

Finally, you need to adjust the `amavisd.conf` configuration file to reflect the appropriate settings. If you followed the preceding recommendations, then set the following values as follows:

```
$mydomain = 'example.com';
$daemon_user  = 'amavis';
$daemon_group = 'amavis';
$TEMPBASE = "$MYHOME/tmp";
$forward_method = 'smtp:127.0.0.1:10025'; # for postfix
$notify_method = $forward_method;         # for postfix
$inet_socket_bind = '127.0.0.1';          # improves security
$QUARANTINEDIR = '/var/quarantine';
```

You will want to change `example.com` to the name of the domain you are receiving email for. `$daemon_user` and `$daemon_group` are set to the name of the amavisd-new user—in our case, `amavis`. `$TEMPBASE` is set to the amavisd-new variable `$MYHOME` appended with `/tmp`. You may want to set this to `/var/tmp` or `/tmp`, depending upon your setup. The `$forward_method` setting tells amavisd-new what to do with the message after processing it. In our case, Postfix expects to receive the message on port `10025` of the local machine. `$notify_method` tells amavisd what to do with notify messages—in our case, treat them the same as the `$forward_method`. `$inet_socket_bind` is set to `loopback` in order to restrict the IP addresses that are allowed to connect to amavisd. Finally, the `$QUARANTINEDIR` keyword tells amavisd-new what to do with messages if they are identified as a problem and need to be set aside.

If you are not running virus checks, you will want to enable this line:

```
@bypass_virus_checks_acl = qw( . );
```

This will disable virus checking, if necessary. The log level can be set anywhere from 0 (no logging) to 5 (everything is logged). For debugging purposes, start with 5 and then reduce it down to 2 after everything is running smoothly.

```
$log_level = 2;
```

After all of the settings have been changed, start amavisd with the debug option to check for any missing Perl libraries or other misconfigurations:

```
bash$ sudo su
# /usr/local/sbin/amavisd debug
```

After it starts cleanly, enable amavisd-new to start on bootup by executing the following, assuming you are running a recent version of Linux:

```
# cp amavisd_init.sh /etc/init.d/
# ln -s /etc/rc.d/init.d/amavisd_init.sh /etc/rc.d/init.d/rc2.d/amavisd
```

Configuring Postfix

The Postfix configuration required to activate SpamAssassin and amavisd-new is relatively straightforward. Only a few lines need to be added to your `main.cf` and `master.cf` located by default in `/etc/postfix`.

In `main.cf`, add the following line:

```
content_filter = smtp-amavis:[127.0.0.1]:10024
```

The above line tells Postfix to invoke the amavisd-new content filter by connecting to the loopback interface on port `10024`. In `master.cf`, add the following lines:

```
#
# The amavis interface
#
smtp-amavis unix - - y - 2 smtp
      -o smtp_data_done_timeout=1200
      -o disable_dns_lookups=yes

127.0.0.1:10025 inet n - y - - smtpd
      -o content_filter=
      -o local_recipient_maps=
      -o relay_recipient_maps=
      -o smtpd_restriction_classes=
      -o smtpd_client_restrictions=
      -o smtpd_helo_restrictions=
      -o smtpd_sender_restrictions=
      -o smtpd_recipient_restrictions=permit_mynetworks,reject
      -o mynetworks=127.0.0.0/8
```

The first configuration entry beginning with `smtp-amavis` here tells `smtp` (Postfix's delivery agent) to run in a `chroot`'ed environment with a maximum of two instances. It

invokes `smtpd`, sets the `smtp` done timeout to 1200 seconds, and disables DNS lookups to improve performance. The second configuration entry starting with `127.0.0.1` tells amavisd-new to reinject the filtered results into a `chroot`'ed instance of Postfix's `smtpd` on port 10025 configured with the listed restrictions.

The next step is to tell Postfix to re-read its configuration files:

```
bash$ sudo postfix reload
```

You should now be up and running with SpamAssassin/amavisd-new support in Postfix. You may skip ahead to the "Verifying SpamAssassin Operation" section now.

SPAMASSASSIN AND QMAIL

The flow of mail when utilizing SpamAssassin and qmail on a system-wide basis is shown in Figure 3.5. For the purposes of this book, it is assumed that qmail (and required associated programs) have been previously installed by the administrator.

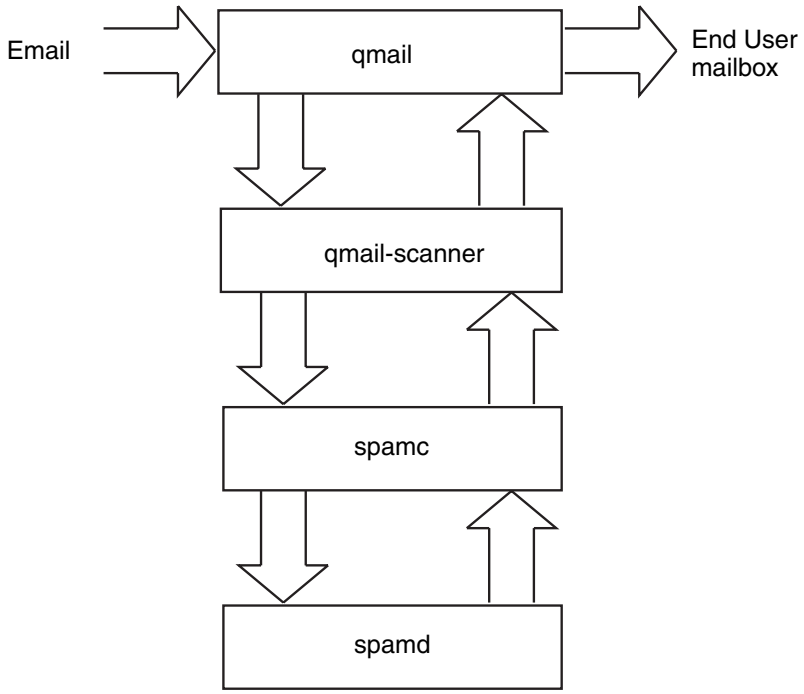

Figure 3.5 SpamAssassin and qmail.

In order to integrate SpamAssassin into qmail, a number of additional packages are required. These include

- Qmail-Scanner (*http://qmail-scanner.sourceforge.net/*)
- qmail-queue patch (*http://www.qmail.org/qmailqueue-patch*)
- maildrop 1.3.8 or higher (*http://www.courier-mta.org/maildrop/*)
- Perl module Time::HiRes (*http://search.cpan.org/dist/Time-HiRes/*)
- Perl module DB_File (may be included in your Perl distribution; if not, it's available at *http://search.cpan.org/dist/DB_File/*)
- Perl module Sys::Syslog (most distributions come with it preinstalled; if not, it's available at *http://search.cpan.org/~nwclark/perl-5.8.2/ext/Sys/Syslog/Syslog.pm*)

Qmail-Scanner allows anti-virus and anti-spam tools such as SpamAssassin to be invoked by qmail. More information on Qmail-Scanner is available at *http://qmail-scanner.sourceforge.net/*. The qmail sources need to have the qmail-queue patch, which is available at *http://www.qmail.org/qmailqueue-patch*, installed. The qmail-queue patch allows filtering, header rewriting, and other functionality required for programs like SpamAssassin. If you are running Linux, a qmail-queue patched version of qmail-1.03 prebuilt RPM prepackaged binary is available at *http://untroubled.org/qmail+patches/*. The instructions here assume that you are building qmail from scratch and not using the RPM.

To enable qmail to invoke SpamAssassin, you must apply a simple patch to qmail-1.03. From the *http://www.qmail.org/qmailqueue-patch* site, copy the lines beginning with `diff` to the last line from the qmail-queue patch page to a file called `patchfile` in your `qmail-1.03` directory. Apply the patch from the directory containing the root of `qmail-1.03` like this:

```
bash$ patch -p0 < patchfile
```

Become root and reconfigure, recompile, and reinstall qmail:

```
bash$ sudo su
# ./config
```

Next, install maildrop. Please note that you need to have GNU `make` installed in order to build maildrop. GNU `make` is installed on most free operating systems such as Linux and BSD variants. To install maildrop, download the sources from *http://www.courier-mta.org/maildrop/* to `/usr/local/src` and unpack them:

```
bash$ bzip2 -d maildrop-1.6.3.tar.bz2
bash$ tar xf maildrop-1.6.3.tar
bash$ cd maildrop-1.6.3
```

Then, compile and install the package as follows:

```
bash$ ./configure
bash$ make
bash$ sudo su
# make install
```

Next, we must prepare to install Qmail-Scanner. The Qmail-Scanner package has a number of prerequisites, namely:

- Perl module Time::HiRes
- Perl module DB_File
- Perl module Sys::Syslog

Be sure they are downloaded and installed prior to continuing the installation. After the environment is set, download qmail-scanner 1.20 from *http://prdownloads.sourceforge.net/qmail-scanner/qmail-scanner-1.20.tgz?download* to a location on your system, such as `/usr/local/src`. Unpack the archive:

```
bash$ gzip -d qmail-scanner-1.20.tgz
bash$ tar xf qmail-scanner-1.20.tar
bash$ cd qmail-scanner-1.20
```

After that is accomplished, compile and install Qmail-Scanner:

```
bash$ ./configure
bash$ sudo su
# ./configure --install
```

After Qmail-Scanner has been installed, you need to update the `tcprules` to have qmail run the `qmail-scanner-queue.pl` executable instead of the default `qmail-queue` binary. This is accomplished by copying the following lines into a file in your local directory called `tcprules.tmp`:

```
127.:allow,RELAYCLIENT="",RBLSMTPD="",QMAILQUEUE="/var/qmail/bin/qmail-queue"
10.:allow,RELAYCLIENT="",RBLSMTPD="",QMAILQUEUE="/var/qmail/bin/qmail-scanner-queue.pl"
:allow,QMAILQUEUE="/var/qmail/bin/qmail-scanner-queue.pl"
```

When loaded into the local `tcprules.tmp` file, update your `tcprules` by executing the following command as root:

```
# tcprules /etc/tcp.smtp.cdb ./tcprules.tmp < /etc/tcp.smtp
```

This command builds a new `tcp.smtp.cdb` file with the new configuration from your `./tcprules.tmp` file. After the `tcprules` have been updated and qmail restarted, qmail will process incoming messages through SpamAssassin.

SPAMASSASSIN SCORING

SpamAssassin's main strength is its ability to score messages for their likelihood to be spam. The set of rules distributed with SpamAssassin is located by default in `/usr/local/share/spamassassin`. The rules are covered here to give you a taste of how the scoring engine works. SpamAssassin rules should be changed only after fully understanding the ramifications of modifying them.

RULES

A rule is a test that determines the spam score of a message. Rules can act on various parts of the message, including the full message and headers, headers only, body only, etc. Rules can be positive (definitely spam) or negative (definitely legitimate email) and can be fractional. An example rule is given in the "Privileged Parameters" section later. An explanation of the default ruleset as shipped with SpamAssassin 2.63 is contained in Appendix C, "Default SpamAssassin Ruleset."

There are a number of rules that incorporate features of SpamAssassin or that make administration easer. One test is called `GTUBE`. This test forces a message to be considered spam by assigning a high score, in the case of `GTUBE`, 1000. `GTUBE` is useful for the administrator when testing a setup. There are also a number of tests associated whitelisting/blacklisting, which implement the whitelisting/blacklisting features of SpamAssassin.

SPAMASSASSIN CONFIGURATION

In the last part of this chapter, we look at the SpamAssassin application: how to use it and how to integrate it into your mail infrastructure.

SpamAssassin is broken down into the following major user-accessible components:

- `spamassassin` —The Perl program meant to be used as a per-user command line interface. It is not designed to be used in high-volume environments; instead, use `spamc/spamd` or call the libraries directly.

- `spamc`—A very efficient interface to the SpamAssassin processor, written in C. It is meant to be called from scripts and designed to communicate with `spamd`, the daemonized version of `spamassassin`.
- `spamd`—The daemonized version of the `spamassassin` program. It is meant to be used with `spamc`, but it doesn't have to be. `spamd` is designed for high-volume environments, where speed is of the essence. It is optimized for running as a standalone program, where clients would connect to it on port 783. There are options for connecting to a MySQL database or LDAP server for large installations.
- `sa-learn`—The program that populates or retrains SpamAssassin's Bayesian classifier database of spam.

Along with these four programs, which can be run by the end-user/administrator, the SpamAssassin libraries themselves are also available. These are the Perl libraries that actually perform the analysis and scoring of email messages. An example of a user program that calls the libraries directly is amavisd-new, one of the SpamAssassin MTA interfaces we cover. The end user can write programs in Perl that execute SpamAssassin library calls, enabling custom programs to be written for processing email. The complete set of command options for `spamassassin`, `spamc`, `spamd` and `sa-learn` commands are contained in Appendix D, "SpamAssassin Command Line Interface Reference."

THE SPAMASSASSIN COMMAND LINE INTERFACE

The `spamassassin` command line interface (CLI) is meant for users who have command line access to the machine that houses their email box. Alternatively, the CLI could be used to process messages for testing purposes or in a setup with a small number of users. The Procmail, forwarding, or `.qmail` functions in MTAs invoke the `spamassassin` command to process messages when SpamAssassin is not set up to be run at MTA time for the entire organization. For example, you would use the CLI if you were not integrating SpamAssassin into your MTA via milter, amavisd-new, or Qmail-Scanner.

The `spamassassin` command line interface can be used for the following purposes:

- To identify a message as spam to collaborative filtering services
- To retrain the internal Bayesian filters
- To send a warning message to spam senders
- To add addresses to whitelists/blacklists

Running SpamAssassin via Procmail

We will use the example of a per-user installation, where we had a small number of users or we wanted to test SpamAssassin prior to rolling it into full production. If you were running Sendmail or Postfix and were set up to call `procmail` as your MDA, this could easily be done by using the following Procmail recipe:

```
:0fw
| /usr/local/bin/spamassassin
```

`/usr/local/bin/spamassassin` could be replaced with `/usr/local/bin/spamc`, the fast interface to spamassassin, if desired. This recipe could alternatively be placed in `/etc/procmailrc`, which would cause `spamassassin` to be invoked for all users. qmail installations would use the `.qmail` functionality outlined next.

Running SpamAssassin in qmail

In a qmail installation, if you wanted to have SpamAssassin process your messages as a regular user, you would place something like this in the `.qmail` file:

```
|/usr/local/bin/spamassassin -P | maildir ./Maildir/
```

This example would route all messages through `spamassassin`, outputting messages in the maildir formatted mailbox located in the user's home directory in folder called `Maildir`.

sa-learn

`sa-learn` is used by a command line user to retrain the Bayesian filters. It accepts a file name as an argument, or messages can be piped to it if desired. It accepts a number of options, the most useful of which are `--ham` for messages that need to be reclassified as non-spam and `--spam` for messages that were mistakenly classified as spam. For example, if you wanted to submit a mail message contained in the file called `spam.txt` in your home directory, you would execute the following command:

```
/usr/local/bin/sa-learn --spam spam.txt
```

This would be used for reclassifying a message that SpamAssassin misidentified as spam.

VERIFYING SPAMASSASSIN OPERATION

In order to be sure that your SpamAssassin installation is working, there are a couple of tests that can be performed. These tests will determine if the SpamAssassin engine is able to identify spam messages with the default installation. The first test is to see if a spam message is identified by the filters correctly, and the second test determines whether a non-spam message is allowed through the program.

TESTING A SPAM MESSAGE

Fortunately, SpamAssassin ships with a message that you can use to easily verify that the filters identify a spam message correctly. The message SpamAssassin ships with is shown in Figure 3.6.

```
Subject: Test spam mail (GTUBE)
Message-ID: <GTUBE1.1010101@example.net>
Date: Wed, 23 Jul 2003 23:30:00 +0200
From: Sender <sender@example.net>
To: Recipient <recipient@example.net>
Precedence: junk
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

This is the GTUBE, the
   Generic
   Test for
   Unsolicited
   Bulk
   Email

If your spam filter supports it, the GTUBE provides a test by which you
can verify that the filter is installed correctly and is detecting incoming
spam. You can send yourself a test mail containing the following string
of characters (in upper case and with no white spaces and line breaks):

XJS*C4JDBQADN1.NSBN3*2IDNEN*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL*C.34X

You should send this test mail from an account outside of your network.
```

Figure 3.6 SpamAssassin test spam message.

The test is run by emailing yourself the sample spam message like this:

```
bash$ mail user@mydomain.com < sample-spam.txt
```

This command assumes that the `sample-spam.txt` file containing the test Spam Assassin message is in the current directory. Replace `user@mydomain.com` with your email address on the machine running SpamAssassin. If this message is caught by the filter, by default you will receive a message (see Figure 3.16) from SpamAssassin telling you about the spam message.

TESTING A NON-SPAM MESSAGE

In order to test a non-spam message, take a message from your inbox and mail it to yourself. For example, from the command line, issue the following command:

```
bash$ mail user@mydomain.com < sample-not-spam.txt
```

We assume that the `sample-not-spam.txt` file, a regular email message with a header and body, is in the current directory. Replace `user@mydomain.com` with your email address as in the previous example. This message should not be caught by the filters but rather should be allowed through SpamAssassin to your inbox.

SPAMASSASSIN CONFIGURATION FILES

SpamAssassin can be run with the as-shipped configuration files, requiring little or no change. Due to the large number of configuration parameters available, only highlights of the configuration files are presented here. A complete description of all configuration parameters is contained in Appendix E, "SpamAssassin Configuration File." Figures 3.7–3.15 and 3.17 are based on information from *http://spamassassin.apache.org*.

CONFIGURATION FILE LOCATIONS

The SpamAssassin configuration files live by default in `/usr/share/spamassassin` and `/etc/mail/spamassassin`. Note that all files with a `.cf` extension are read in alphabetically at `spamassassin` startup time. The site-wide configuration is by convention called `local.cf` and should be placed in `/etc/mail/spamassassin`. You should not place configuration files in `/usr/share/spamassassin`, as those files may get deleted during future SpamAssassin upgrades.

The individual user configuration file is called `user_prefs` and is normally located in the user's home directory in the `.spamassassin` subdirectory. The ability for individual users to have their own `user_prefs` is a tunable parameter. Also, with the virtual support available within `spamd`, user preferences can be located in a variable location, depending upon the mail account's domain and/or username. This makes integration with large mail systems easier.

CONFIGURATION FILE PRECEDENCE

You need to be aware that the SpamAssassin rule processing is "last touched wins." In other words, the last setting is the one that is used. So, it is important to know that the files are processed in this order:

1. SpamAssassin default rules directory, in alphabetic order (defaults to `/usr/share/spamassassin`)
2. Local rules directory, in alphabetic order (by default `/etc/mail/spamassassin`)
3. Local `user_prefs` file (user's home directory, in a directory called `.spamassassin`)

Configuration settings come in different classes. The three classifications of keywords are as follows:

- Those that can be changed globally, by the user and/or system-wide (in other words, both `user_prefs` and `local.cf`)
- Options that are changeable with permission of the SpamAssassin administrator (if `allow_user_rules` is enabled)
- Settings that can only be changed by the SpamAssassin administrator in `local.cf`

The following classes of users are allowed to manipulate the privileged commands:

- The administrator via the system-wide `/etc/mail/spamassassin` file
- Users running `spamassassin` from their `procmailrc` or forward files

`spamd` users are not allowed to change the privileged files unless the administrator has set `allow_user_rules` to 1.

Please consult Appendix E for a complete list and description of available configuration options for all three classes of keywords (global, administrator permitted, and administrator only).

UNPRIVILEGED OR GLOBAL KEYWORDS

Under the unprivileged class of configuration keywords available to anyone, the available options can be broken down as follows:

Scoring

The scoring keywords define what is to be considered spam and the scores assigned by rules (see Figure 3.7).

```
required_score n.nn (default: 5)
score SYMBOLIC_TEST_NAME n.nn [ n.nn n.nn n.nn ]
```

Figure 3.7 Scoring options.

The `required_score` defines what SpamAssassin considers to be spam. This value can be adjusted higher or lower, depending upon the point at which you want SpamAssassin to consider a message to be spam. If you feel SpamAssassin is being too aggressive and creating lots of false positives, adjust the score threshold upward. If it is too conservative, letting a lot of spam get through, adjust it downward.

Whitelist and Blacklist

The settings in Figure 3.8 control how SpamAssassin processes and manages whitelists and blacklists as well as the auto-whitelisting feature.

```
whitelist_from add@ress.com
unwhitelist_from add@ress.com
whitelist_from_rcvd addr@lists.sourceforge.net sourceforge.net
def_whitelist_from_rcvd addr@lists.sourceforge.net sourceforge.net
whitelist_allows_relays add@ress.com
unwhitelist_from_rcvd add@ress.com
blacklist_from add@ress.com
unblacklist_from add@ress.com
whitelist_to add@ress.com
more_spam_to add@ress.com
all_spam_to add@ress.com
blacklist_to add@ress.com
```

Figure 3.8 Whitelist and blacklist options.

These settings are a matter of personal preference and should be set as the user and administrator see fit.

Tagging

The options in Figure 3.9 control how headers are added to spam messages and header processing.

```
rewrite_header { subject | from | to } STRING
add_header { spam | ham | all } header_name string

Defaults to:
add_header spam Flag _YESNOCAPS_
add_header all Status _YESNO_, score=_SCORE__ required=_REQD_ tests=_TESTS_
  autolearn=_AUTOLEARN_ version=_VERSION_
add_header all Level _STARS(*)_
add_header all Checker-Version SpamAssassin _VERSION_ (_SUBVERSION_) on _
  HOSTNAME_

remove_header { spam | ham | all } header_name
clear_headers
report_safe { 0 | 1 | 2 } (default: 1)
```

Figure 3.9 Message tagging and header processing options.

The defaults here are reasonable. If you want to adjust the headers, use the `add_header` keyword. If you want to tell SpamAssassin not to send a separate email message for every message it has identified as spam, set `report_safe` to `0`, and then only `X-Spam-` headers will be added to the message.

Language

The keywords in Figure 3.10 define geographic locations and language.

```
ok_languages xx [ yy zz ... ] (default: all)
ok_locales xx [ yy zz ... ] (default: all)
```

Figure 3.10 Location and language options.

The default is set to allow all locations and languages, which should be acceptable for most situations.

Network Test

These settings include Razor, Pyzor, DCC, RBL, trusted networks, and other related areas (see Figure 3.11).

```
use_dcc ( 0 | 1 ) (default: 1)
dcc_timeout n (default: 10)
dcc_body_max NUMBER (default 999999)
dcc_fuz1_max NUMBER (default: 999999)
dcc_fuz2_max NUMBER (default: 999999)
use_pyzor ( 0 | 1 ) (default: 1)
pyzor_timeout n (default: 10)
pyzor_max NUMBER (default: 5)
pyzor_options [option ...] (default: none)
trusted_networks ip.add.re.ss[/mask] ... (default: none)
clear_trusted_networks
internal_networks ip.add.re.ss[/mask] ... (default: none)
clear_internal_networks
use_razor2 ( 0 | 1 ) (default: 1)
razor_timeout n (default: 10)
skip_rbl_checks { 0 | 1 } (default: 0)
rbl_timeout n (default: 15)
check_mx_attempts n (default: 2)
check_mx_delay n (default: 5)
dns_available { yes | test[: name1 name2...] | no } (default: test)
```

Figure 3.11 Network test options.

These options control the parameters for various network checking and whether checks happen or not. Trusted networks are networks that you know never emit spam and are exempt from certain tests. This series of parameters also allows you to control timeouts and DNS checks. The defaults are reasonable for most installations.

Learning

The options in Figure 3.12 control the operation of the Bayesian analyzer and whitelisting integration with the Bayesian engine.

Most of the Bayesian learning options are acceptable as shipped and don't need to be adjusted. If there are issues with Bayesian scores, the thresholds can be adjusted via the `bayes_auto_learn_threshold` parameters. In case you want to turn off the Bayesian processing totally, set `use_bayes` to 0.

```
use_bayes ( 0 | 1 ) (default: 1)
use_bayes_rules ( 0 | 1 ) (default: 1)
auto_whitelist_factor n (default: 0.5, range [0..1])
auto_whitelist_db_modules Module ... (default: DB_File GDBM_File NDBM_File SDBM_File)
bayes_auto_learn ( 0 | 1 ) (default: 1)
bayes_auto_learn_threshold_nonspam n.nn (default: 0.1)
bayes_auto_learn_threshold_spam n.nn (default: 12.0)
bayes_ignore_header header_name
bayes_ignore_from add@ress.com
bayes_ignore_to add@ress.com
bayes_min_ham_num (Default: 200)
bayes_min_spam_num (Default: 200)
bayes_learn_during_report (Default: 1)
bayes_sql_override_username
bayes_use_hapaxes (default: 1)
bayes_use_chi2_combining (default: 1)
bayes_journal_max_size (default: 102400)
bayes_expiry_max_db_size (default: 150000)
bayes_auto_expire (default: 1)
bayes_learn_to_journal (default: 0)
```

Figure 3.12 Bayesian analysis engine options.

Miscellaneous

These keywords (Figure 3.13) define the templates for functions and other areas not defined previously.

```
lock_method {nfssafe |  flock | win32 } (default: no default)
fold_headers { 0 | 1 } (default: 1)
report_safe_copy_headers header_name ...
envelope_sender_header Name-Of-Header
describe SYMBOLIC_TEST_NAME description ...
report_charset CHARSET (default: unset)
report ...some text for a report...
clear_report_template
report_contact ...text of contact address...
report_hostname ...hostname to use...
unsafe_report ...some text for a report...
clear_unsafe_report_template
```

Figure 3.13 Miscellaneous options.

In the miscellaneous category, `fold_headers` controls whether the headers that SpamAssassin adds are broken up (the default) or kept as one single long line. If a mail client has issues with divided headers, this parameter can be adjusted.

PRIVILEGED PARAMETERS

The end user is allowed to create his or her own spam checking rules if certain conditions apply.

```
allow_user_rules { 0 | 1 } (default: 0)
header SYMBOLIC_TEST_NAME header op /pattern/modifiers [if-unset: STRING]
header SYMBOLIC_TEST_NAME exists:name_of_header
header SYMBOLIC_TEST_NAME eval:name_of_eval_method([arguments])
header SYMBOLIC_TEST_NAME eval:check_rbl('set', 'zone' [, 'sub-test'])
header SYMBOLIC_TEST_NAME eval:check_rbl_txt('set', 'zone')
header SYMBOLIC_TEST_NAME eval:check_rbl_sub('set', 'sub-test')
body SYMBOLIC_TEST_NAME /pattern/modifiers
body SYMBOLIC_TEST_NAME eval:name_of_eval_method([args])
uri SYMBOLIC_TEST_NAME /pattern/modifiers
rawbody SYMBOLIC_TEST_NAME /pattern/modifiers
rawbody SYMBOLIC_TEST_NAME eval:name_of_eval_method([args])
full SYMBOLIC_TEST_NAME /pattern/modifiers
full SYMBOLIC_TEST_NAME eval:name_of_eval_method([args])
meta SYMBOLIC_TEST_NAME boolean expression
meta SYMBOLIC_TEST_NAME boolean arithmetic expression
tflags SYMBOLIC_TEST_NAME [ {net|nice|learn|userconf|noautolearn} ]
priority SYMBOLIC_TEST_NAME n
```

Figure 3.14 Administrator enabled or privileged settings.

A simple rule is shown in Figure 3.14 for illustrative purposes. More complex rules may require changes to configuration files and are beyond the scope of this book.

Example Rule

We want to create a simple rule that scans the message body and adds 10 points to the spam score if the phrase `this is a test of spamassassin` is present. Rules can be placed in the user's configuration file if allowed by the administrator. More commonly, they are placed in `/etc/mail/spamassassin/local.cf` by the administrator, which is the method we use here.

If you want to look at the default rules that are distributed with SpamAssassin, they are located by default in `/usr/local/share/spamassassin`. Although you can add or update the rules located here, this can cause problems when upgrading to future versions of SpamAssassin. The rules files that are distributed with SpamAssassin 2.63 are shown in Figure 3.15.

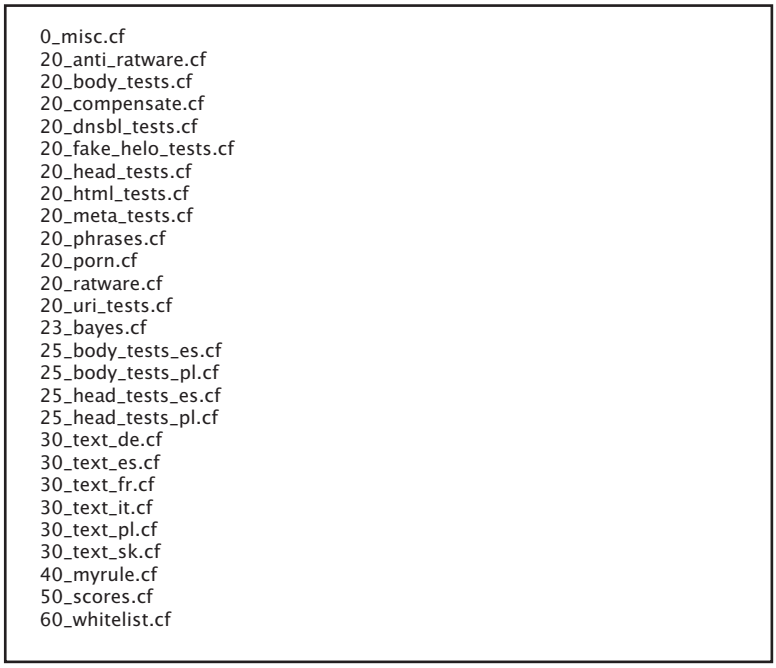

Figure 3.15 SpamAssassin distributed rules.

We update the `local.cf` file in `/etc/mail/spamassassin` with our new rule. The rule we want to add in `local.cf` consists of the following three lines:

```
body MY_SPAMASSASSIN_RULE        /this is a test of spamassassin/is
describe MY_SPAMASSASSIN_RULE    Test of my spamassassin rule
score MY_SPAMASSASSIN_RULE       10
```

The name of the rule is `MY_SPAMASSASSIN_RULE`. The line starting with `body` defines the test to run. In our case, the test is to match the pattern `this is a test of spamassassin`, ignoring the case of the string (indicated by `i`) and matching with embedded newlines (indicated by `s`). The pattern modifiers (our rule has one, `is`) are Perl regular expressions.

The second line gives a human-readable description to our rule—in our case `Test of my spamassassin rule`. Finally, the number of points to be added to the score is indicated by `score`—in our case `10`. Note that the score can be any positive, negative, whole, or fractional value.

If SpamAssassin is configured to run this test, and a message is processed by the server that contains the string `this is a test of spamassassin`, a message similar to that in

Figure 3.16 should be sent to the recipient. Sending a message like this is the default method of spam notification under SpamAssassin. The original message is included at the bottom of the message from SpamAssassin.

```
Subject: Spamassassin rule test
From: somebody@example.com
Date: Tue, 20 Jul 2004 11:07:27 -0400
To: dale@woody.cushman.avacoda.com

Spam detection software, running on the system "woody.cushman.avacoda.com",
has identified this incoming email as possible spam.  The original message
has been attached to this so you can view it (if it isn't spam) or block
similar future email.  If you have any questions, see
the administrator of that system for details.

Content preview:  This is a test of spamassassin. This message should be
  marked as spam. [...]

Content analysis details:   (15.0 points, 5.0 required)

 pts  rule name                 description
 ----  ----------------------   ---------------------------------------------------
 0.3  NO_REAL_NAME              From: does not include a real name
  10  MY_SPAMASSASSIN_RULE      BODY: Test of my spamassassin rule
 3.3  MSGID_FROM_MTA_SHORT      Message-Id was added by a relay
 1.4  DNS_FROM_RFCI_DSN         RBL: From: sender listed in dsn.rfc-ignorant.org



Subject: Spamassassin rule test
From: somebody@example.com
Date: Tue, 20 Jul 2004 11:07:27 -0400
To: dale@woody.cushman.avacoda.com

This is a test of spamassassin.  This message should be marked as spam.
```

Figure 3.16 Example of SpamAssassin notification message.

SpamAssassin assigned this message a score of 15.0 points, activating the following rules and associated scores:

Test	Score
NO_REAL_NAME	0.3
MY_SPAMASSASSIN_RULE	10
MSGID_FROM_MTA_SHORT	3.3
DNS_FROM_RFCI_DSN	1.4

The tests were activated due to the lack of a name in the From: header, the test rule we built containing the test phrase, the fact that a header was generated by a mail relay, and the fact that the From: sender is listed in a BLS blacklist.

ADMINISTRATOR-ONLY SETTINGS

The settings that can be modified only by the administrator include paths, directory locations, library modules, username/password, and similar parameters.

```
test SYMBOLIC_TEST_NAME (ok|fail) Some string to test against
razor_config filename
pyzor_path STRING
dcc_home STRING
dcc_dccifd_path STRING
dcc_path STRING
dcc_options options (default: -R)
use_auto_whitelist ( 0 | 1 ) (default: 1)
auto_whitelist_factory module (default: Mail::SpamAssassin::DBBasedAddrList)
auto_whitelist_path /path/to/file (default: ~/.spamassassin/auto-whitelist)
bayes_path /path/to/file (default: ~/.spamassassin/bayes)
auto_whitelist_file_mode (default: 0700)
bayes_file_mode (default: 0700)
bayes_store_module Name::Of::BayesStore::Module
bayes_sql_dsn DBI::databasetype:databasename:hostname:port
bayes_sql_username
bayes_sql_password
user_scores_dsn DBI:databasetype:databasename:hostname:port
user_scores_sql_username username
user_scores_sql_password password
user_scores_sql_custom_query query
```

Figure 3.17 Administrator-only configuration settings.

Most of these settings are acceptable as distributed and only need to be changed if you use non-standard software installation locations or are using LDAP, MySQL, or other optional features.

CONCLUSION

SpamAssassin is a Perl program that scores email messages on the basis of several different criteria, including heuristics, Bayesian, distributed checksums, blackhole listing

service checks, and blacklists/whitelists. It allows administrators and certain users to generate their own rules for helping to determine whether or not a message should be considered spam.

For our coverage, SpamAssassin is run on a per-user basis by Procmail (Sendmail/Postfix) or `.qmail` files (qmail). It is configured to run on a system-wide basis by milter/MIMEDefang (Sendmail), amavisd-new (Postfix), or Qmail-Scanner (qmail). SpamAssassin has many configuration parameters in three separate classes. The as-distributed settings can produce a reasonable and working implementation for most organizations. Rules can be defined by administrators and certain users, allowing customized scoring rules to be enabled.