Email Control

By Jerry Peek

For a lot of us these days, "control" of email is more of a sad joke than a reality. New laws may not do much to stop the flood of spam, and viruses and worms keep sneaking in. As incoming messages make your server struggle and users' inboxes groan, what's an administrator — or a user — to do?

System-level mail filtering can help, but it's not always the best answer. For instance, blacklisting entire domains (refusing messages from certain servers) can mean that you miss email from potential customers — customers who don't know (or care) that their postmaster isn't following proper anti-spam procedures. Or a colleague of yours might keep sending late-night emails, asking you for a date, and you'd like to trash those messages automatically, but still receive his other, business-related messages.

This month, we'll look at ways to filter mail using *procmail* (covered in several recent issues of *Linux Magazine,* as well as at http://www.procmail.org), emphasizing ways to detect messages using the contents of the header or body of email messages. (Some of these same techniques also apply to other mail-filtering systems.) Let's get digging for that dirt!

What's the Problem?

One of the first steps to take when you want to filter mail is to see as much of each message as you can. Many mail user agents (MUAs) like Outlook Express don't show what's actually in the message header and body. (The article "Personal Post," available online at http://www.linux-mag.com/2003-07/email_01.html, explains the header and body.) Your MUA may also add new fields, such as `Status:`, that aren't in the message header *procmail* sees. So it's important to get as close to the source as possible.

As you configure *procmail*, it's also possible to "lose" a mes-

Efficiency Counts

Searching through text takes time: CPU time, clock time, disk rotations. So, automatically filtering messages in the mail-delivery chain (before messages arrive in your inbox folder) can cause system delays.

As you add mail filtering — especially filtering for an entire server or organization — you might want to measure the impact it's having on system load and the time it takes to deliver each message. For instance, if you have access to the log for your system MTA (your mail transfer agent, like *sendmail*), see how long incoming messages take to travel from receipt to delivery — especially at busy times of day.

If mail filtering adds too much delay, here are some efficiency tips that may speed things up:

If certain message senders never send spam, or certain message subjects (or other header fields) guarantee a "clean" message — mail from automated processes, for example — add recipes to the top of your *.procmailrc* so those messages bypass filtering and go straight to the destination.

Avoid "big" filter processes like Perl when a simpler one like *formail* or *sed* will do. (Remember that your *procmail* may be running on a busy mail server host instead of your local workstation, so efficiency may really count.)

Look into *scoring,* which is one way to make *procmail* stop searching before the end of a message. We introduce scoring in this article.

Investigate the possibility of queuing huge messages and scanning them later. For example, you might toss some messages into an "unfiltered" folder and scan them on your local workstation instead of on the mail server.

procmail typically runs each time a new message is delivered. Depending on your needs, though, it may not have to. New mail can be delivered immediately to a user's mail store (typically, a spool file, like */usr/mail/jpeek*), then *procmail* can run sometime later, before or during the time a user gets new mail from that file. For instance, if users have their own workstations and use *fetchmail* to grab new messages from a central server, have *fetchmail* invoke *procmail,* or set *procmail* as the users' local delivery agent (from their *sendmail* file *.forward,* or by another method).

The *man* page for *procmail* shows a script for post-processing a message store. You might run it from a *cron* job, or configure your MUA (mail user agent) to invoke the script when you use the "get new mail" command. (For instance, in *Sylpheed,* in the "Filtering" dialog box, run the script with an "execute" condition.)

Each user can fetch a system-wide *procmail* RC file by setting the `INCLUDERC` variable from their own *.procmailrc* files. If none of these ideas help, dream up some others! *procmail* and Linux give you the tools to be incredibly flexible.

sage by storing or forwarding it somewhere you didn't expect. (*procmail* is reliable; mail shouldn't simply vanish.) Therefore, it's a good idea to keep copies of the newest messages by adding a backup recipe to the start of your *.procmailrc* file. The sidebar "Backup Your Mail" shows one way to do this. Once you have the backup files, you can inspect the original message headers and bodies, just as *procmail* receives them: just *cd* into the backup directory and use a Linux program like *less* to read the files directly.

Once you're looking at the messages, decide what you want to do with them. You can leave that decision to message classifiers like POPfile (described in the July article mentioned earlier) or SpamAssassin (covered in "Slicing Up Spam," available online at http://www.linux-mag.com/2003-08/tech-support.ol.html). But you can also make some choices yourself that are both simple and effective.

Simple (but Effective) Techniques

Look at a series of messages over time. (The *less* pager makes this easy: from your *backup* directory, type `less msg.*`, then use the `:n` and `:p` commands to jump between messages.) If you've had *procmail* running for a while, the log file — its pathname is set by the `LOGFILE` variable — also provides a good overview.

What characteristics can you recognize? Here are some ideas:

> Messages from a particular mailing list might have the header field `X-Mailing-List: listname`. You can add a *procmail* recipe to catch those messages.

> Messages in other languages or character sets may have a `Subject:` field starting with the character set, like `Big5`:

```
Subject: =?Big5?B?sdkfjasf?=
```

The next recipe catches messages whose subject is in `koi8-r`, `gb2312`, `big5`, and more, and stores them in the folder *other-language:*

```
:0 :
* ^subject: =\?(koi8-r|gb2312|big5|...)\?
other-language
```

This recipe uses alternation: a series of patterns inside parentheses, separated by `|` characters.

> Does your site have a host that only processes mail — with no valid email addresses there? Many spammers send mail without a valid envelope sender address, or with no `Message-ID:` field, and your mail server may add its hostname at those places in the spammers' messages. For instance, if your mail server is *server.foo.xyz*, and you get a message with one of the following lines in the header, it may be bogus:

```
From auser@server.foo.xyz
...
Message-Id: <Fu077082@server.foo.xyz>
...
From: Ur Friend <friend@server.foo.xyz>
```

A *procmail* recipe like the following may work for you:

```
:0 f
* @server\.foo\.xyz
* !^from: MAILER-DAEMON@server.foo.xyz
* !^from:.*(joe@bar\.xyz|
  someuser@somewhere\.com)
| sed '1,/^$/s/^Subject: /&
  {Possible SPAM?} /'
```

The recipe determines if the message has `@server.foo.xyz` anywhere in its header, and that it's *not* from a system address like *MAILER-DAEMON* on that host, *nor* from either *joe@bar.xyz* or *user@somewhere.com* (because those users' MTAs aren't configured properly, and your mail server has to fix the header). If so, we use the *sed* stream editor to add the string `{Possible SPAM?}` to the start of the subject. (The `f` flag makes this recipe a filter. The *sed* command searches

Backup Your Mail

The following two recipes from the *procmailex man* page are a great way to make backups of incoming email. Put them at the start of your *.procmailrc* file, just after setting any initial variables like `MAILDIR`:

```
:0 c
backup

:0 ic
| cd backup && \
  rm -f dummy 'ls -t msg.* | sed -e 1,100d'
```

The first recipe writes a copy of each message to a unique filename, starting with *msg.,* in a subdirectory named *backup.*

The second recipe does `cd backup` and, if *cd* succeeds, runs `rm -f` to remove the oldest files. The `ls -t` command gets a list with the newest filenames first, then *sed* takes the first 100 names off of that list. The remainder (any filenames that aren't the 100 newest) are passed to *rm* with the shell's command substitution operators (`''`). (The filename `dummy` keeps `rm -f` from complaining if *sed* makes no output — that is, if there aren't more than 100 files.)

LISTING ONE: A scored recipe

```
# If a message has more quoted than unquoted
# lines, add X-overquote-score field:
:0 B f
*  10^1 ^>
* -10^1 ^[^>]
| formail  -A "X-overquote-score: $="
```

the message header for the line starting with `Subject:` and edits just that line.)

> If you look at the `Sender:` header field, or the envelope sender addresses (on the `From` separator line), the `Received:` fields, or sometimes other fields, you may see a hostname that's being used to pour out tons of spam for a while. The hostnames may change over periods of weeks, but getting in the habit of scanning your *procmail* log and/or the backed-up messages can let you find a way to nab spammers. If your site already uses a blacklist, that may not be necessary, but you still might want to block particular sites or addresses that keep sending you product promotions, newsletters, etc. — and who either refuse to stop, or who you don't want to take a risk of acknowledging their messages in case they send even more!

A condition like this (which we've split across several lines for printing) can do the job:

```
* ^(from|reply-to):.*(sese[- ]?seko|
  RealtyNewsBriefs|shark57|
  town(shipvibe|vibes|evibe)?\.(net|com))
```

It's helpful to understand extended regular expressions like the ones that *procmail* supports. The regular expression used here checks both the `From:` and `Reply-to:` fields for addresses including s*eseseko, sese seko, sese-seko, townshipvibe.net, townshipvibe.com, townvibes.net, townevibe.com*, and more.

Scoring Messages

One lesser-known *procmail* technique is *message scoring*. It gives you even more control over message-matching than regular expressions do. It also can speed up *procmail* by making it stop after checking a certain number of lines — for instance, stopping after the first 100 lines.

Traditional recipes either match or they don't: the conditions are either true or false. Message scoring uses a series of conditions, each of which has a negative or positive weight. A scored recipe only matches if its weights add up to a positive number.

Scoring is too complex to explain completely in this short column. The *man* page for *procmailsc* has details and some helpful examples.

Let's look at the simple example in *Listing One*. In the next section, we'll cover a much more useful example.

The `B` flag tells *procmail* to search the message body (the body has the lines we want to test; we don't want to test the header). The `f` flag makes the recipe a filter: it doesn't deliver the message but simply pipes it through the command line at the end of the recipe. (A later recipe, or your MUA, could use the filter's results.)

Both of the conditions include scoring, followed by a regular expression. Scoring has the syntax `w^e`, where `w` is the *weight* and `e` is the *exponent*. So, in the first scored condition, `10^1`, the weight is 10 and the exponent is 1.

If the condition's regular expression matches the message line, the score is raised or lowered by that condition's weight. (Since each exponent is 1, we can always ignore it: the scores don't change exponentially.)

The score always starts at zero. *procmail* tests the regular expression in each condition on each line of the body, and each matching regular expression changes the score.

For instance, if the five-line message body looks like this:

```
> > blah blah blah
> yadda yadda yadda
> etc. etc. etc.

I agree completely, Jerry!
```

The first line (`> > blah blah blah`) matches the first regexp (`^>`), which selects quoted lines — that is, lines starting with a `>` character. This condition has a score of 10. So, now the score is 10.

Next, *procmail* checks the same line against the next regexp, (`^[^>]`), which selects lines that do *not* start with a `>` character. This regexp doesn't match, so the score doesn't change; it's still 10.

LISTING TWO: A sample YAVR recipe that catches the Aliz worm

```
# for Aliz
:0BD
* -1000^0
*   300^0 ^TVoAAAI
*   300^0 Z48GGVZ
*   300^0 kZ8x\+Ak
*   300^0 QCCZAWJ
  {
  LOG="—-=== WORM-ALIZ $DATE ===—-${NL}"
  :0:
  $VIRDIR/virus-Aliz
  }
```

The second and third lines of the message also start with a quote, so they work like the first one did: we add 10 to the score for each one, making the total score 30, so far.

The fourth line is empty: no characters at all. It doesn't match either of the conditions, so the score is still 30.

The fifth line (`I agree ...`) doesn't start with `>`, so it doesn't match the first regexp. However, it does match the second regexp. This condition has a weight of -10. Adding this to our previous score of 30 gives a total score of 20. (In other words, there are two more quoted lines than unquoted lines.)

Because the total score, 20, is a positive number, the recipe matches and the message is passed through the filter. This *formail* filter adds a header field like `X-overquote-score: 20`. It uses the *$=* variable, which *procmail* sets after checking all condition fields of a recipe.

Encoded Message Bodies

One *procmail* recipe file that makes extensive use of scoring is YAVR, from http://agriroot.aua.gr/~nikant/nkvir. This long series of procmail recipes scans message bodies for worms — and, optionally, for the Nigerian scam ("We've found 25 million dollars that we have to hide! Tell us your bank account number!").

Let's see how it works.

The MIME *base64* encoding technique, which is used in a lot of message bodies, lets you email arbitrary data: pictures, word processor documents, and programs, including worms. The encoding is predictable — the same series of bytes always encodes the same way — so you can use *procmail* pattern matching to match certain file types, as well as the code for worms. Many file types start with a certain sequence of bytes, and many worms contain particular bytes at particular places. *procmail*'s powerful pattern matching lets you search for the encoded versions of these strings.

The YAVR file has a series of recipes, where each recipe matches a different worm. The recipe in *Listing Two* catches the "Aliz" worm.

The first condition, `-1000^0`, has a weight of -1000, an exponent of zero, and no regular expression. The exponent 0 means that this condition will match only once, and because it has no regexp, it will match as soon as the message body is read. The effect is to start the message score at -1000.

The other four conditions all have a weight of 300 and an exponent of 0. If more than three of them match (just once each, due to the zero exponent), then the score becomes positive 200, and the recipe succeeds. These four conditions match snippets of the encoded worm; when all four are found, the author believes, it's found the Aliz worm.

The nested curly-braces surround a variable assignment and a simple recipe that writes the suspect message into a folder named *virus-Aliz*. The *procmail* variable `LOG` holds text to be added to the procmail log. Grouping the log line with the recipe that delivers the message makes sure that this log line is written at the same time as the rest of the log for this message. (Users who get lots of email may sometimes have more than one *procmail* process running. This technique keeps the log from becoming jumbled.)

Digging Deeper

We've barely scratched the surface of what you can do with *procmail*. The best way to see what fits your needs is to keep an eye on your incoming mail — and get to know what *procmail* can do. *procmail* is incredibly powerful, and recipes are limited only by your imagination and cleverness.

If you haven't looked through the *man* page for *procmailex*, it's worth a browse for its big variety of examples. Two other good resources come from the SourceForge PM-DOC project, and their huge frame-based tips page at http://pm-doc.sourceforge.net/pm-tips.html, and the searchable archive of the *procmail* discussion list at http://www.rosat.mpe-garching.mpg.de/mailing-lists/procmail.

It's worth mentioning that *procmail* can run other programs to help you classify email. One is Bogofilter, a command-line-oriented Bayesian spam filter, available from http://bogofilter.sourceforge.net. The FAQ shows some *procmail* examples.

Or, if you use a pre-filter that marks your mail with new header fields — such as SpamAssassin (see http://www.spamassassin.org) or *POPFile* (http://popfile.sourceforge.net/) — *procmail* can match the header fields added by those programs and route your mail accordingly.

Let's can that spam!

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for over 20 years. He's happy to hear from readers at jpeek@jpeek.com.

Power Tip: Monitoring *procmail*

For each message that *procmail* handles, it adds three lines to its log file, starting with `From`, `Subject:`, and `Folder:`, respectively. Any other lines are probably errors (unless you've set the `VERBOSE` or `LOG` variables, that is). Errors can happen at any time, caused by a full filesystem, an unmounted directory, or something more unusual.

The simple shell script *check_procmail,* available at http://www.linux-mag.com/downloads/2003-10/power, watches the log file for any errors and emails them to you. It's typically run from a *cron* job.

To investigate an error, read your *procmail* log file with a browser like *less* that makes it easy to search for a string of error text.