by Jeffreys Copeland and Haemer

Not Looking Through Our Mail

Jeffrey Copeland (`copeland@alumni.caltech.edu`) *lives in Boulder, CO, and works at Softway Systems Inc. on UNIX internationalization. He spends his spare time rearing children, raising cats, and being a thorn in the side of his local school board.*
Jeffrey S. Haemer (`jsh@usenix.org`) *works at QMS Inc. in Boulder, CO, building laser printer firmware. Before he worked for QMS, he operated his own consulting firm, and did a lot of other things, like everyone else in the software industry.*

Note: The software from this and past Work columns is available at `http://alumni.caltech.edu/~copeland/work.html`.

Letters,
We get letters,
We get stacks and stacks of letters …
– Perry Como

What? Again? You guys just used this quote.
– Our readers

Dad. You're not funny. I mean it.
– Our children

Look on the bright side. At least you're not seeing the same Perry Como quote over and over again in your email. After you read this column, you'll be able to arrange not to see some other things in your email, too.

Do you get unsolicited junk email? Do you find it tiresome to see people rewrite their email addresses in news postings to avoid spam? ("To get my real email address, remove `NOSPAM` from my address, `<jNOSPAMsh@useNOSPAMnix.org>`") Is there a subset of your mail that you always either delete, or perhaps refile into a separate folder? Are there sites or people that you'd prefer not to hear from, ever? Answers to all these questions come under the header of "mail filtering."

This month, we'll walk you through the basic idea of mail filtering, to give you an idea of how filters work and what's possible.

How's It Work?

Mail is delivered to your mailbox, typically a file named with your login name in a well-known directory, where it has been deposited by a Mail Transfer Agent (MTA), typically Eric Allman's `sendmail`. Jeff Haemer's mailbox, for example, is usually in `/var/spool/mail/jsh` or `/usr/spool/mail/jsh`. After delivery, the mail sits there, waiting for you to look at the mailbox with one of several user agents–such as `elm`, `Mail`, `pico` or even Netscape–that are used to compose and read mail.

If you run a filter over your mailbox before you read it, that filter can file, autoreply to, discard or redirect mail messages without your ever having to see them. This takes some of the burden off of you and puts it onto the software, where it belongs.

Seems pretty obvious, and it is. The two most common forms of mail filtering, `.forward` files and the `vacation` program, are nearly ubiquitous. We'll review them both briefly.

Filtering with .forward Files

If you create a file in your home directory named `.forward`, which contains another email address, all your mail will get forwarded to that address. (If you have never tried this, try it now. Be sure to warn the person to whom you're forwarding the mail first, if you care what they think of you.)

Understand, this does not produce a carbon copy. Suppose, for example, that we have the following:

```
$ cat ~jsh/.forward
jeff@softway.com
```

Upon discovering that a piece of mail has `jsh` as its intended recipient, `sendmail` looks for `~jsh/.forward`, reads it and ships the mail to `jeff@softway.com` instead of depositing it into `/var/spool/mail/jsh`.

At this point, if you're already trying to change your boss' `.forward` file so that all his mail will be sent to you instead, the reason it's not working is that contemporary versions of `sendmail` won't use a `.forward` file that is writable by anyone but the owner.

Nevertheless, `.forward` files are the simplest tool for handling changes of address. (You can also redirect mail with changes to your system `/etc/aliases` file with the help of your systems administrator. For more complicated changes, such as redirecting all the mail for your site, modifying the MX records for your site may be called for, but that's a story for another day.) One of our colleagues, Henry Stiles, told us last week that he's still getting mail forwarded from his QMS address, even though he's been at another company, Artifex, for about a year. Similarly, Jeff Haemer routinely gets email forwarded to him at QMS, from the address `jsh@usenix.org`.

It's a good bet that most of the folks who run ads on the Net offering permanent email addresses ("*Now, a way to let you never have to change your email address, EVER!!!!!*") just have a PC with a lot of home directories, each with a single `.forward` file and nothing more. A single Linux box with a couple of large disks could, in theory, provide this kind of service for the entire world's population, although its `sendmail` process would be very weary at the end of the day.

This is not the only useful trick available with `.forward` files. If an address in a `.forward` file begins with the pipe symbol, `|`, what follows is a program that will act as a filter. Instead of merely forwarding the mail, it is piped as standard input to that filter, which in turn can do all sorts of arbitrary and interesting things to the incoming mail.

Filters on Vacation

The `vacation` program is a commonly used program that makes use of pipes in `.forward` files. When you send someone email, and you get a rapid reply that says something like this:

```
To: jeff@softway.com
From: jsh@boulder.qms.com
Subject: Out of town
Delivered-BY-The-Graces-Of: the vacation program
Precedence: junk
Status: RO

I am in Romania for the last week of October,
for the Romanian Open Systems Exposition.

If you get this and live in Boulder,
don't even think of breaking in.
I've filled my house completely with water
and stocked it with piranhas.

Your message has been saved,
and will be delivered to me on my return,
after I finish draining the children's bedrooms.
```

the reply has probably been automatically generated by the `vacation` program.

The process of generating it goes like this:

1. Jeff Copeland's mail arrives, addressed to Haemer (`jsh`).
2. Our MTA, `sendmail`, reads `~jsh/.forward`, finds something like the following:

```
\jsh, "|/usr/ucb/vacation jsh"
```

and feeds the mail to `vacation` instead of putting it into Haemer's mailbox.

3. The `vacation` program checks to make sure the mail isn't from a mailing list, and that the sender doesn't already know Haemer is on vacation–that is, he hasn't already gotten an "on vacation" message. It also checks to make sure the message isn't "ringing" (see below).
4. If none of these is the case, `vacation` replies to Copeland's mail with the contents of `~jsh/.vacation.msg` and stores his address in a database, generated using the Berkeley DBM library (kept in the files `~jsh/.vacation.pag` and `~jsh/.vacation.dir`), so that he only gets told once that Haemer's out of town.
5. Finally, `vacation` stores the message in `/var/spool/mail/jsh`, so Haemer can read it on his return.

In other words, `vacation` is just a program that handles individual messages in a stereotyped but useful way. And if `vacation` can do that, why stop there? What's to stop us from writing our own code to parse mail messages, and turning it into a back end for `sendmail`?

(The real answer is, "your systems administrator." As you can guess, this sort of thing could be used to create interesting security holes, and some sites configure `sendmail` to prohibit using filters in `.forward` files. We'll talk about that later, too.)

Not Writing a Mail Filter

OK, so the general principle seems easy enough: We write a filter that parses incoming mail messages, figures out what to do based on the contents and the headers, and then does it.

This is a straightforward text-processing application that doesn't require things like GUIs or complex algorithms, so we could probably do the job using a Perl or shell script. Perl even has a suite of mail-manipulating modules, available on the Comprehensive Perl Archive Network (`http:/www.perl.com/CPAN/`). In the March issue ("Looking Through Our Mail," Page 80), we used these modules for a different job, but they can make jobs like this one relatively easy, too.

The problem with this approach is that we risk having to write and maintain an endless string of `sendmail` back ends, each for a slightly different job: filing, discarding spam and so forth.

But what's our alternative? `procmail`.

When we need software, instead of writing our own from scratch, we start by looking to see what's already out there. A quick look around reveals no shortage of general-purpose mail-filtering programs. In all cases, these programs generalize the capability of `vacation`. Through some launching mechanism–typically, `sendmail`'s ability to execute commands in users' `.forward` files–the programs examine mail messages and take actions based on what they find. Nearly all of these programs give users a little language to configure the filter, which pairs search keys with actions–much like `make`'s target/rule pairs or `lex`'s regular-expression/code-block pairs. A short example by Jamie Hoglund, together with a brief analysis, can be found at `http://www.skypoint.com/members/jhoglund/perl/filter.html`.

At the other end of the spectrum, the most mature mail filter is Stephen ven den Berg's `procmail`. (Maturity is important: You don't want bugs in programs that mess with your email.) The syntax for `procmail` is a little painful, but the capabilities are complete. We know of systems administrators for large installations who first arranged to have critical processes notify them of problems by email, then configured `.procmail` to page them whenever things go amiss, or even send them a display message telling them exactly what's gone wrong.

What does the syntax look like? Here's an example rule–one of many in Haemer's `.procmailrc`:

```
1   :0 B
2   * Content-Type:.*name=".*\.doc"
3   {
4       :0 h
5       * !^X-Loop: jsh@woodcock.boulder.qms.com
6       | (formail -r -A"Precedence: junk"\
7       -A"X-Loop: jsh@woodcock.boulder.qms.com" ;\
8       $HOME/Mail/procmail/microsoft ) \
9       | $SENDMAIL -t
10  }
```

In line 1, `:0` begins a rule, and `B` says to examine the body of the message. In line 2, `*` signals a regular expression. This rule is triggered whenever `procmail` sees a message that contains a MIME-encoded Microsoft Corp. `.doc` file.

Lines 3 through 10, enclosed in braces, are an action to take. In this case, we look at the header (lines 4 and 5) to guard against "ringing" (see below), and then auto-generate a reply. The header of the reply is created by a stand-alone utility called `formail` (lines 6 and 7), which comes as part of the `procmail` distribution. We generate the body, in line 8, with a shell script, but we could have just hard-wired the text into the action itself. Finally, in line 9, we pipe the entire message back to `sendmail`, which ships the reply to the original sender, explaining what's happened:

```
The email you sent to Jeff appears to be
a Microsoft .doc file. Jeff cannot read
Microsoft .doc files, and your message has been
automatically discarded.

This reply was automatically generated, so Jeff
is unaware that you sent anything.

If you want to communicate with him, you can do
so by resending your email as flat, ASCII text.
```

Your Email's Ringing

OK, imagine that Haemer has a `.procmailrc` that says "Auto-reply to all mail from Copeland,"

```
:0 c
* ^From.*Copeland# Anything from Copeland
| (formail -r -A"Precedence: junk";\
   echo "Got your mail!") | $SENDMAIL -t
```

and Copeland has a `.procmailrc` that says "Auto-reply to all mail from Haemer,"

```
:0 c
* ^From.*Haemer# Anything from Haemer
| (formail -r -A"Precedence: junk";\
   echo "Got your mail!") | $SENDMAIL -t
```

Any message from Copeland to Haemer will generate a reply from Haemer to Copeland, which will generate a reply from Copeland to Haemer, which will generate a.... If people were doing this, someone would suggest that the participants get a life, but the perpetrators are programs, and can't get a life, no matter what William Gibson and Bruce Sterling tell you. What they can do is create a flurry of email traffic that quickly swamps all other mail processing.

By analogy with circuits, this behavior is called "ringing." To prevent this, a mail filter should insert a sentinel into its reply that will let it detect whether it comes back around.

For `procmail`, an easy solution looks like this:

```
1  :0 c
2  * ^From.*Copeland# Anything from Copeland
3  * !^X-Loop: jsh@boulder.qms.com
4  | (formail -r -A"Precedence: junk"\
5    -A"X-Loop: jsh@boulder.qms.com" ;\
6    echo "Got your mail!") | $SENDMAIL -t
```

The new argument to `formail`, in line 5, inserts a user-defined field into the header. When the message comes back around, the regular expression in line 3 detects that header and doesn't auto-reply to it.

One possibility is to coax your systems administrator into installing procmail as your local mail delivery program... it requires getting on your systems administrator's good side. That's a worthwhile goal in and of itself.

Filters: Just Say No

Security-conscious installations may prohibit the use of filters in `.forward` files. For example, in the incomprehensible configuration file for `sendmail`, `/etc/sendmail.cf`, the variable `Mprog` dictates what program parses the filter command. A line like this:

```
Mprog, P=/bin/sh,  F=lsDFMeuP,
   S=10, R=20, A=sh -c $u
```

means "hand all filter lines to the shell." In contrast,

```
Mprog, P=/bin/true,  F=lsDFMeuP,
   S=10, R=20, A=true -c $u
```

will make `sendmail` ignore filters; `true` will return immediately and successfully without doing anything at all.

If this is your situation, what's to be done? One possibility is to coax your systems administrator into installing `procmail` as your local mail delivery program, by replacing the line that looks like this:

```
Mlocal,    P=/bin/mail, F=rlsDFMmnP,
   S=10, R=20, A=mail -d $u
```

with one like this:

```
Mlocal, P=/usr/local/bin/procmail,
   F=lsSDFMhPfn, S=10, R=20,
   A=procmail -Y -a $h -d $u
```

This option is discussed in some detail in the documentation that comes with the `procmail` distribution, but it requires getting on your systems administrator's good side. That's a worthwhile goal in and of itself, so we encourage this approach.

If that's impossible, all is not lost. First, you have read and write permission for your own mailbox. One possibility is to run a `cron` job through a filter nightly, or even more frequently. This will at least cut down on your work. However, you need to take precautions to keep User Agents and Mail Transfer Agents from modifying your mailbox at the same time your filter is–if two programs are writing to your mailbox at the same time, you're likely to scramble its contents.

A second approach would be to filter on demand. One interesting way to do this is to build a front end for your User Agent. To read your mail, for example, you could invoke a front end that would filter your mail, then invoke your mail reader. This wouldn't work well for an application like Netscape, which typically gets invoked only once, but would work for programs like `pico` or `elm` that get a separate invocation for each mail-reading session. (Another reason we avoid those monolithic what-you-see-is-what-you-get programs: As nice a user interface as Netscape mail provides for some, filtering your mailboxes gets to be a little tricky.)

Now that we've got the tools to automatically defend our battlements against spammers, we need a good heuristic for detecting mail from them. We tried just bouncing back mail we got from anyone at AOL but realized that we were losing correspondence with people we needed to talk to, not to mention Dogbert's father, Scott Adams.

If you'd like to learn more about mail filtering, a fine place to start is the mail-filtering FAQ, which can be found scattered around the Web at places like `http://www.ii.com/ internet/faqs/launchers/mail/filtering-faq/`.

Next month, we'll either talk about that, or about how we shot ourselves in the foot by being too careful when we had a flaky disk, or whatever other interesting problem we trip over in the meantime.

Until next time, happy trails.