

Filtering E-Mail with Postfix and Procmail, Part Three

Brian Hatch 2002-07-11

Filtering E-Mail with Postfix and Procmail, Part Three

by Brian Hatch

last updated July 11th, 2002

This is the third installment in a four-part series on filtering e-mail with Postfix and Procmail. The first two parts of this series focused on how you can stop receiving spam by configuring Postfix for spam prevention. This segment will introduce you to the methods of stopping spam with Procmail.

Stopping Spam with Procmail

Procmail is a local mail delivery agent, meaning that its only job is to deliver mail to a local user mailbox. Procmail can deliver mail to local mailboxes that are in mbox files, MH folders, or maildir folders, as well as sending messages to other e-mail addresses or local programs. Unless you have a specific mailbox format preference, you should probably stick with the default on your system, which is most likely mbox format. For the purposes of this article, I'll assume mbox folders are used.

Postfix uses its own internal local mail delivery agent by default. However, you can easily configure it to use Procmail instead, either globally or on a per-user basis.

Using Procmail Globally

The easiest way to use procmail as your delivery agent is to add the following line to your `/etc/postfix/main.cf`:

```
mailbox_command = /usr/bin/procmail
```

You should also verify that mail for the root user is forwarded to a user account. For example, to forward all root mail to "bree" you'd use:

```
# echo 'root: bree' >> /etc/aliases
```

Finally, rebuild the aliases database and reload postfix:

```
# newaliases
```

```
# postfix reload
```

That's it. Now procmail is in use for all local users.

Using Procmail for Specific User Accounts

If you prefer that only certain users get Procmail for their local delivery agent, or if you are not the Postfix administrator, then you can set up `.forward` files to launch Procmail for specific users. All you need to do is create a file called `.forward` in your home directory that looks like this:

```
"|IFS=' ' && exec /usr/bin/procmail -f- || exit 75 #username"
```

Substitute "username" with your actual username and you're all set. And note that the quotes **do** belong in the file, they are not just there for this Web page.

If you are the administrator and want to enable Procmail for a specific user, you can create their forward file for them. Just make sure to chown the file appropriately when done.

What to do with Your Mail

Procmail's default purpose is to deliver your e-mail to your mailbox, usually `/var/mail/USERNAME` or `/var/spool/mail/USERNAME`. However, there are several other options that we have for filtering our e-mail.

Dedicated Spam Box

If you have a mailer that supports multiple inboxes, such as Mutt, Pine, or any IMAP-based mail client, then you can have mail delivered to multiple places. This is perfect for having a mailbox dedicated to each list to which you subscribe. This makes it easier to read your e-mail because you are in the appropriate mindset - you won't have messages from Great Grandma George in the same mailbox as your buffer overflow discussions for example. I prefer to create a mailbox that will be the repository for all the mails that are most likely spam. That way I can check this mailbox once a day when I'm bored and make sure nothing important ended up there by mistake, and can delete the rest. No need to look at any of the message content in general - the "From" and "Subject" lines are usually sufficient to distinguish them.

Many folks (myself included) prefer to name all mailboxes that can receive e-mail (rather than just being places you manually move e-mail) with a prefix such as 'IN.'. How you set up your mailer to check these alternate mailboxes is client specific. In Mutt, for example, you can add the following to your `~/.muttrc`

```
mailboxes =IN.fanmail =IN.family =IN.vpns =IN.hle \  
           =IN.bugtraq =IN.flames =IN.deaththreats
```

Or if you want to slurp all mailboxes in alphabetically:

```
mailboxes = `echo $HOME/Mail/IN.* | sed -e "s#\$HOME/Mail/#=#g" `
```

This is good to make sure you don't accidentally forget to check an inbound mailbox that you set up via Procmail later. You can even merge the two in such a way that you have a specific order of mailboxes to check, followed by all others not specifically named:

```
mailboxes =IN.fanmail =IN.family =IN.vpns =IN.hle \
          =IN.bugtraq =IN.flames =IN.deaththreats \
          `echo $HOME/Mail/IN.* | sed -e "s#$HOME/Mail/#=#g" `
```

In Mutt you can check the alternate mailboxes by using the "c" command. Mutt will tell you when new mail has been delivered to these periodically too. If you're using a different mail program, you're on your own.

Adding Mail Headers

Another option would be to insert new mail headers for any message that is likely to be spam. You can then use this header for additional Procmail rules (to shuttle the offending mail to a dedicated spam box, for example) or use it within your mail client. Many mail clients are able to look for specific mail headers and then flag the messages differently.

Mutt, for example, can use mail headers as part of its scoring algorithms to put spam messages at the bottom of your mailbox where you don't need to look at them. Other programs can color code mails with this header, letting you identify and delete them faster.

/dev/null

Instead of a dedicated spam mailbox, you can send mails that are determined to be spam directly to the bit bucket. If you do this then the message will never reach your e-mail spool whatsoever, and you will have no trace of what the message was. The sender will have no idea that you did not get the mail.

I generally don't like this option because if a legitimate message was dropped to /dev/null then I will never know what it said, and the sender won't know that I didn't get it since they get no error message. I strongly recommend against using this.

Exiting with an Error

If Procmail exits with an error, then the mail server (Postfix) will assume there is some problem delivering the message, and it will either retry later, or give up and send back an error to the envelope sender. This is somewhat better than using /dev/null because the sender will get an error message indicating that their e-mail was not properly received. In the case of a legitimate e-mail, the sender will know to retry. In the case of a spam, the spammer may even take your address off their list, although this is unlikely.

Procmail Recipes

Procmail reads global configuration from /etc/procmailrc, and then user configuration from \$HOME/.procmailrc. These files contain "recipes" that tell the program what to do. These recipes could tell Procmail to deposit the mail somewhere, forward it to another e-mail address, or filter or pipe it through an external process. The procmailrc has both and/or style logic available, as well as grouping with { } for control flow. You can create procmailrc files that do exactly what you need that are almost as unreadable as perl code.

I'll be keeping the Procmail commands pretty light here. If you wish to hurt your brain, then investigate the full power of Procmail by reading the `procmailrc` and `procmailex` manual pages. You can (and should) comment your `procmailrc` files in the standard shell way, by using lines beginning with "#".

The most important thing to understand is that there are two kinds of recipes. A delivering recipe is one that will save, forward, or pipe the e-mail somewhere. At that point procmail considers its work done, and it stops processing the e-mail. (Unless you specifically tell it to continue with the "c" flag, which is discussed below.) Anything else is a non-delivering recipe - including things like message rewriting via the filtering mechanism I discuss later - and the processing of the mail continues until a delivering recipe is found or the end of the `procmailrc` is reached.

So, let's see what these recipes look like.

Recipe Heading

All recipes start with a line that looks like this:

```
:0 [flags] [ : [lockfilename] ]
```

There are a bunch of flags available to you. The ones you'll use most often are:

Condition-modifying flags, such as:

- **H** - egrep the header for the conditions listed below. This is the default unless other flags are specified.
- **B** - egrep the body for the conditions listed below
- **D** - tell egrep to be case sensitive. (By default it is case insensitive, which is usually what you want.)

Action-related flags, such as:

- **h** - pipe the header to an external program
- **b** - pipe the body to an external program
- **f** - filter the message through an external program. Procmail discards the existing message and replaces it with the output of the program. Excellent for adding or removing headers, often with the `formail` program that is included with Procmail. This is **not** a delivering recipe as it only modifies the message.
- **W** - wait for the external program to finish. (Including it is a good idea when when using `h`, `b`, or `f`.)

Flow control, such as:

- **a** - execute this recipe if the previous one matched and completed successfully - basic "AND" logic.
- **E** - execute this recipe if the previous one was not executed at all - basic 'ELSE' logic.
- **c** - Generate a new copy of the message. This is used if you want to have a message delivered somewhere (say to a special mailbox or bounced to another e-mail address) and have Procmail continue processing the

procmailrc file to deliver it somewhere else too.

There are others, and I even hand-waved away some of the additional features of the flags I listed above. See the procmailrc man page if you're interested.

All the flags you list are "and"-ed together. For example:

```
:0 Bdfh
```

would mean to `egrep` the body of the message case sensitively for the patterns found on the following lines, and if the patterns match then filter the header of the message through some program.

If you put a colon after the flags then Procmail will create a lockfile before performing your actions. This is important on any recipe that will end up saving the e-mail somewhere, as you don't want two copies of Procmail writing to the same file at the same time. You can specify a name for the lockfile if you wish, such as:

```
:0 Bf:busy_lock
```

If you don't specify a filename, then Procmail will generate one by using "*filename.lock*", where "*filename*" is the name of the target file. Because of this, there's no need to specify a lock file name in general.

Recipe Conditions

The most common conditions are simple `egrep` pattern match strings. You tell Procmail to look at a portion of the message (header or body) and see if the regular expressions match. Each regular expression (you can specify more than one) is listed on it's own line with "*" at the beginning, such as:

```
* regular_expression
```

For example, the following recipe will match e-mails with those two Subject and X-Mailer headers. (The default is to `egrep` headers, unless flags are specified.)

```
: 0
* ^Subject: make money fast
* ^X-Mailer: AOL 6.0
```

Since frequently you'll want to be able to determine if a message is to a specific destination, Procmail includes two handy shortcuts:

```
^TO      Match a destination (To/Cc/Bcc/etc) containing
```

the specified word.

```
^TO_    Match a destination (To/Cc/Bcc/etc) containing
        the specified address.
```

Thus, the following is a good condition to add to match only messages addressed to you specifically:

```
* ^TO_myaddress@my_isp.net
```

You also have the ability to negate any regular expression using the **!** character, thus the following would match messages **not** specifically addressed to you:

```
* !^TO_myaddress@my_isp.net
```

When you include multiple regular expressions, each must be matched for the action to be taken. However, Procmail also has the ability to use multiple regular expressions with scoring, enabling each to have different weight. You include a score and exponent (used to affect the score when found multiple times) before the actual regular expression. Procmail tallies the score and if the result is positive, it considers it a match:

```
:0
* -200^0 ^Content-Type: text/html
* -300^0 ^Subject:.*(ADV|Sex|Viagra|Enlargement)
* -300^0 ^From:.*(hotmail|aol).com
* 100^0 ^Subject:.*Re:
* 300^0 ^User-Agent: Mutt
* 501^0 ^TO_myaddress@my_isp.net
```

```
IN.spam
```

The previous recipe gives negative weights to various common spam characteristics and positive weights for indications of legitimate e-mail.

Procmail's scoring is rather powerful, but I won't go into it further because we'll see later that most of the spam-related scoring algorithms are better performed in external products like SpamAssassin, which we will call from Procmail. However, if you are interested in using scoring algorithms, consult the `procmails` man page.

Recipe Actions

The last part of each recipe is the action to take. There are four actions you can have, which I will explain in detail below.

1. Delivery to local file

Deliver the e-mail to a local file. If the action line begins with anything other than a "!", "{", or "|" (which are the remaining possibilities described below) then the action is simply the filename to which the mail should be delivered. An example would be:

```
:0:
* ^TO_bri@hackinglinuxexposed.com
IN.hle
```

In the example above, mail to my HLE address is automatically dropped into the ~/Mail/IN.hle mailbox. The type of mail spool used depends on the suffix you use for the filename.

```
filename          standard mbox file format
filename/         maildir folder
filename/.        MH folder
```

Again, unless you prefer one form or another, stick with the default for your system, which is likely mbox format. Procmail will create the file if it doesn't already exist.

One thing to note: if the target is a directory (as opposed to a proper maildir or MH folder directory structure), then the messages will be stored in that directory as separate files named "msg.XXXXX" (for varying values of XXXXX.) So make sure you type your filenames correctly.

2. Begin nesting block

It's often helpful to take multiple actions based on some conditions. You create a nesting block using curlies, inside of which are more recipes that are only executed if the conditions are matched.

For example you may want to group all your logic that is based on the presence of a list header into a block, rather than listing each combination set separately. An example would be:

```
:0 h
* ^Subject: security alert
```

```
{
    :0 c
    $DEFAULT

    :0
    ! my-pager-address@example.com
}
```

Since the "Subject:" line matched, the two recipes inside the curlies are executed. True, this logic could have been written as:

```
:0 hc

* ^Subject: security alert

$DEFAULT

:0 h

* ^Subject: security alert

! my-pager-address@example.com
```

But by grouping them together it was more readable. Advanced rules may require (and will definitely be more readable with) curlies, especially when using and/else flags.

(\$DEFAULT is your default mail destination, by the way. I'll cover some handy variables in a minute.)

3. Forward the message

You can e-mail a copy of the message simply by listing the e-mail address or addresses after a "!", such as:

```
:0 h

* ^TO_helpdesk@our_company.net

* ^Subject:.* Windows(95|98|ME|NT|2000|XP| )

! bill_gates@micro_soft.com
```

If you're exclusively a Unix administrator, you may find that this recipe correctly routes help desk e-mails to the true offending party. Since this is a delivering recipe, you would never see these mails at all.

4. Pipe the message into an external program

You can pipe a message into a separate program in two different ways. The first is to use the program as a final delivery, such as this:

```
:0 bhW
* ^Subject: Re-sync DNS
| /opt/bin/resync-dns
```

In that case, the resync-dns command will receive the e-mail and do something with it. Procmail's job is finished, and no other copy of the e-mail goes anywhere. Some places have used SMTP as a "reliable" messaging service to trigger tasks remotely just like this example.

The other option is to use the external program as a filter, which will be used to re-process the mail for Procmail, such as:

```
:0 bfW
* Greeting card
| /opt/bin/snag-card-from-web
```

Assuming you have some program that understands the format of the e-mail sent by those on-line greeting cards your Grandfather just loves, it can extract the URL from it, run it through 'lynx --dump -force_html \$URL' and you'll never need to actually click to get to the card at all. The output of your program will replace whatever was filtered through it (in this case the body, due to the "b" flag). A filtering recipe is not a delivering recipe. Thus your filter will replace the ad-laden greeting card info with the actual greeting card contents, and then Procmail will continue processing, eventually delivering that new e-mail somewhere. The previous example may have been better written as:

```
:0 bfW
* Greeting card
| /opt/bin/snag-card-from-web
```

```
:0 A:
```

```
$DEFAULT
```

This will filter the e-mail and deposit it instantly into the default mail spool. There's no need to continue looking through the remaining Procmail recipes, just deliver it.

One of the most convenient filters you can use is the Formail program, which comes with Procmail. You can use it to add or delete message headers, amongst many other features. For example you could use the following:

```
:0 Wf
* ^Subject:.*Internet SIC Codes
|formail -A "X-Spam: yes" -A "X-Spam-Identifier: formail rocks"
```

to add two "X-Spam" headers to the e-mail, which could be used by your mail client to filter or sort the e-mail as you see fit. See the formail man page to learn all it can do.

Procmail Variables

There are a number of variables that Procmail uses, and you can define any number of them you like for later. You define a variable simply by putting

```
VARIABLENAME=value
```

at the top of your `.procmailrc`. The most important variables used by Procmail itself are:

```
VERBOSE          If set to 'on' then you'll get lots of verbose logging.
                  (Defaults to 'off'.)

LOGFILE          Place to write procmail logs (/dev/null by default.)

MAILDIR          Directory to which all relative pathnames are located.

DEFAULT          Default mail spool (/var/mail/username, for example.)
```

When testing, it's best to enable logging by defining a LOGFILE and setting VERBOSE to "on". If you plan to filter e-mail to multiple folders within the same directory, it's easiest to set MAILDIR to an appropriate value in your home directory, such as:

```
MAILDIR=$HOME/Mail
```

Then any filenames you use as delivery actions will reside in \$HOME/Mail unless they begin with a "/". All the other Procmail variables have pretty sane defaults that you won't usually bother with. You can set your own variables as

well. The one that I use the most is:

```
SPAM=IN.spam
```

I use variables for all my frequently used mailbox names, both for convenience and to avoid my frequent typos. For example:

```
# Catch commonly used spam-sending programs.

# Thanks for acknowledging yourselves in the headers...

:0 H:

* ^(Cyber-Bomber|E-Broadcaster|Ellipse Bulk E-mailer|E-mailBlaster|MailKing)

$SPAM

# I'm fine, thank you. No need. Really.

:0 H:

* ^Subject.* Viagra .*

$SPAM

# Why am I in so many people's address books?

:0 BD:

* I send you this file in order to have your advice

$VIRUS
```

Mail Filtering Methodology

Detecting and filtering spam at the mail delivery agent takes CPU time and will slow down your mail delivery. It's completely unavoidable. Because of this, you want to perform your filters and tests in the most CPU-efficient manner possible, without compromising your detection rules. I filter my mail through the following steps:

1. Take any messages that are from moderated lists I've subscribed to and put them in their own folders. Presumably moderated lists won't allow spam through, so let them through right away:

```
# deposit bugtraq mails
```

```
:0:
```

```
* ^List-Id: <bugtraq.list-id.securityfocus.com>
```

```
IN.bugtraq
```

2. Accept other messages that are typically not spam:

```
:0:
```

```
* ^From: .*@current_client.com
```

```
IN.current_client
```

```
:0:
```

```
* ^From:.*great_grandpa@his_e-mail.com
```

```
$DEFAULT
```

3. Identify spam based through procmail natively:

```
# Drop messages which are pure HTML - Probably spam
```

```
:0:
```

```
* ^Content-Type: text/html
```

```
$SPAM
```

```
# I've got a real one, thanks.
```

```
:0:
```

```
* ^Subject:.*(real|university|authentic).*DIPLOMAS
```

```
$SPAM
```

```
# No From header?  What are you hiding?
```

```
:0:
```

```
* !^From:
```

```
$SPAM
```

4. Perform more time-consuming but robust spam-detection. In this case I have Razor determine if the message is spam, and deposit it in the \$SPAM mailbox if that's the case. (Razor is discussed in the next installment in this series.)

```
:0 Wc
```

```
| /usr/bin/razor-check
```

```
:0 Wa
```

```
$SPAM
```

5. Deposit mail from unmoderated mailing lists and other messages that don't go to \$DEFAULT:

```
:0:
```

```
* ^X-List-Name: openssl-users
```

```
IN.openssl
```

6. Drop all other messages to \$DEFAULT

```
:0:
```

```
$DEFAULT
```

You don't actually need this rule, since it's the default action anyway.

This system gets messages that are very likely not spam to the proper destinations as quickly as possible, minimizing the CPU usage of the tests. I also exclusively use header-based rules early on, since matching against the headers will take much less time than checking the entire body of the mails, in general.

Interfacing Procmail with External Spam Detection Programs

Procmail's power is in its simplicity. It allows you to make sophisticated decisions based on message content with both static expressions and scoring decisions. However, implementing all your spam blocking inside Procmail itself means you must maintain thousands of rules to make your friend-or-foe decision. This is not only a hassle, but it also means that you'll spend more time tweaking your spam rules than you may save by simply hitting the delete key.

For this reason you may wish to call external programs designed specifically for spam detection. You tell Procmail to pipe the e-mail to the spam detection program, and this program either modifies the message or exits with an error code indicated the message is spam. Procmail then filters the message accordingly.

Conclusion

That concludes the general discussion on filtering e-mail using Procmail. In the next (and final) installment in this series, we will discuss the use of two tools that are available for use with Procmail: Razor, an automated spam tagging and filtering tool, and SpamAssassin, a mail filter that contains hundreds of different spam tests.

To read the Fourth and final installment of this series, click [here](#).

Brian Hatch is an obsessive security freak and lead author of [Hacking Linux Exposed](#) and co-author of [Building Linux VPNs](#). While he frequently stays up late to write or hack code, he thinks it's much more fun to go to the park and push his daughter in the swing as he delivers horrible puns to his fiancée.

Relevant Links

[Filtering E-Mail with Postfix and Procmail, Part One](#)

Brian Hatch, SecurityFocus

[Filtering E-Mail with Postfix and Procmail, Part Two](#)

Brian Hatch, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus