

# PROCMAIL: ADD A SPAM FILTER TO YOUR EMAIL SERVER

Filter unwanted mails, keep your inbox clean and make sure you don't pass any viruses on to your Windows-using friends.

## WHY DO THIS?

- Teach your mailserver to keep your inbox Nigerian prince-free.
- Prevent viruses from using your emails as a transmission vector.
- Control the way you communicate!

Last month, we used Arch Linux to build a mail server. It accepts incoming mail and delivers it to users' mailboxes so that they can read it with their favourite IMAP email client. But it accepts all mail, including unwanted spam and virus-ridden ones. This month, we'll add filtering capabilities to our server to help prevent undesirable messages finding their way into our users' mailboxes.

Our server can receive mail in two ways: its Mail Transfer Agent (MTA) accepts mail directly from the internet and its Mail Retrieval Agent (MRA) downloads mail from other external mail servers. We used *Postfix* for our MTA and *Fetchmail* for our MRA.

We'll configure a new Mail Delivery Agent to filter mail from both channels, either delivering it to our IMAP server (also an MDA) or to reject it. We'll use *Procmail* for this new MDA. Install it from the repository (we're using Arch Linux for this project):

```
$ pacman -S procmail
```

The objective of our new MDA is to perform system-wide mail filtering. The system-wide filters will remove spam, viruses and so-on.

*Procmail* takes its instructions from a file, usually called **/etc/procmailrc**. Create a basic file to begin with that just delivers all mail:

```
LMTP_DELIVER="/usr/lib/cyrus/bin/deliver -a $MAILBOX"
NL="
"
:0 w
| $LMTP_DELIVER $MAILBOX
EXITCODE=$?
:0
/dev/null
```

The first line sets up our Cyrus-IMAP delivery command-line. The **NL** variable contains a newline character that we'll use later on when writing to the log file. The blocks beginning with **:0** are recipes – the first recipe delivers mail and the second one tells *Procmail* to dump the message before exiting with an error code.

*Procmail's* processing stops once a delivering recipe succeeds, so the second recipe would only be invoked if there were a problem with delivery. Although *Procmail* dumps the message when there is an error, the agent that invoked *Procmail* would react to its non-zero exit code by bouncing the message.

You can verify that *Procmail* works by sending a test message through it and checking that it appears in our test user's inbox:

```
$ procmail MAILBOX=testuser < testmessage
```

## It's black and white

The simplest filters we can apply either accept or block messages from specific senders. We can create static files containing email addresses or domains and then use those files as black- and white-lists. Add these recipes into the **/etc/procmailrc** before the existing delivery recipe:

```
:0
*? formail -x "From" -x "From:" -x "Sender:" \
-x "Reply-To:" -x "Return-Path:" -x "To:" \
| egrep -is -f /etc/procmail/whitelist
{
LOG="whitelisted$NL"
:0 f
| formail -fA "X-Whitelisted-$$: Yes"
}
:0
* !$X-Whitelisted-$$: Yes
*? formail -x "From" -x "From:" -x "Sender:" \
-x "Reply-To:" -x "Return-Path:" -x "To:" \
| egrep -is -f /etc/procmail/blacklist
{
LOG="blacklisted$NL"
:0
/dev/null
}
```

We can then blacklist a domain, say **example.com** and whitelist a user, say **bob@example.com** by writing entries in the black- and white-list files referenced by the recipes. The rules write a log message when they match. You write to the *Procmail* log by assigning to the **LOG** variable.

Whitelisted messages are marked by adding a header, **X-Whitelisted**, suffixed with *Procmail's* process ID so later recipes can ignore similar headers that we didn't set. The **formail** command that is part of the *Procmail* package is used to read and write message headers. The blacklist passes messages that have this header and otherwise discards messages that match the blacklist rule. We'll also use the presence or absence of the whitelist header in other rules later on.

We can take blacklisting a step further and make use of Realtime Blackhole Lists, or RBL. These are DNS-based address blacklisting databases, also known as DNSBL, that contain IP addresses of known sources of unsolicited bulk email (spam). There is a small utility that checks an IP address against various

blacklists. Install its package:

```
$ pacman -S rblcheck
```

You invoke **rblcheck** with a list of IP addresses and it checks them against lists provided by **sorbs.net**, **spamhaus.org**, **spamcop.net** and some others. It returns a non-zero exit status if a given address is blocked (it also echoes the blocked addresses to standard output). You can use it like this:

```
$ rblcheck -qm 27.20.121.36
```

You need to extract the IP addresses from the message header. One way to do this is with a little *Bash* script, saved as **/etc/procmail/header\_ip** that reads message headers from its standard input:

```
#!/bin/bash
while read line
do
  if [[ $line =~ \[([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)\] ]]
  then
    ip=${BASH_REMATCH[1]}
    [[ $ip =~ ^127\.\.0|^10\.\.|^192.168\.\. ]] || echo -n "$ip "
  fi
done
```

Don't forget **chmod +x** to make it executable. A new recipe uses the script and the **rblcheck** tool to drop mail from addresses on these blackhole lists unless they have also been whitelisted:

```
:0 h
HEADER_IP=/etc/procmail/header_ip
:0
* !^X-Whitelisted-$$: Yes
* ! ? if [ -n "$HEADER_IP" ]; then rblcheck -qm $HEADER_IP; fi
{
  LOG="blackholed$NL"
:0
/dev/null
}
```

## Meet the Assassins

Using realtime blackhole lists prevents a lot of spam from reaching users' mailboxes but some will still get through. We need some additional help and it comes in the form of *SpamAssassin*, which detects spam, and *ClamAV*, which detects viruses. Begin by installing the required packages:

### Is Procmail dead?

The last update to *Procmail* happened on 10 September 2001, quite a long time ago, with the release of version 3.22. Despite this, it is still very widely used and it has been claimed that it does what it needs to do and requires no more development. There are lots of *Procmail* examples on the internet and the **procmail-users** mailing list is still active. We've used *Procmail* for mail filtering because it is well documented and there is a lot of information and community support online. It is also well suited to use with our MTA, MRA and MSA, so has everything covered.

If *Procmail's* status bothers you, consider alternatives like *MailDrop* ([www.courier-mta.org/maildrop](http://www.courier-mta.org/maildrop)) or mail filter applications that interface to *Postfix* (of course, these won't work if you need to filter mail through a mail retrieval agent like *Fetchmail*).

Keep an eye on your Bayes database.

```
$ pacman -S spamassassin razor clamav
```

```
$ pacman -U ~build/clamassassin/clamassassin-1.2.4-5-any.pkg.tar.xz
```

```
$ pacman -U ~build/pyzor/pyzor-0.8.0-1-any.pkg.tar.xz
```

```
$ pacman -U ~build/dcc/dcc-1.3.155-1-x86_64.pkg.tar.xz
```

*ClamAssassin* uses *ClamAV* to virus-check email and adds headers to messages found to contain viruses. Its config file is installed to **/etc/clamav/clamd.conf**, adjust it to include these definitions:

```
LogSyslog yes
LogFacility LOG_MAIL
LogTime yes
PidFile /var/run/clamav/clamd.pid
TemporaryDirectory /tmp
DatabaseDirectory /srv/mail/clamav
LocalSocket /var/lib/clamav/clamd.sock
User clamav
```

A separate daemon called **freshclam** updates the virus database. Review its configuration, in **/etc/clamav/freshclam.conf** too:

```
LogSyslog yes
LogFacility LOG_MAIL
DatabaseDirectory /srv/mail/clamav
DatabaseMirror db.UK.clamav.net
DatabaseMirror database.clamav.net
NotifyClamd /etc/clamav/clamd.conf
```

Create the virus database directory, start the *ClamAV* services and **freshclam** will download the virus database:

```
$ mkdir -m 700 /srv/mail/clamav
$ chown clamav: /srv/mail/clamav
$ systemctl enable clamd freshclamd
$ systemctl start clamd freshclamd
```

You can test your *ClamAV* installation without exposing your system to real viruses. You can instead download files containing the EICAR test string and use those for testing:

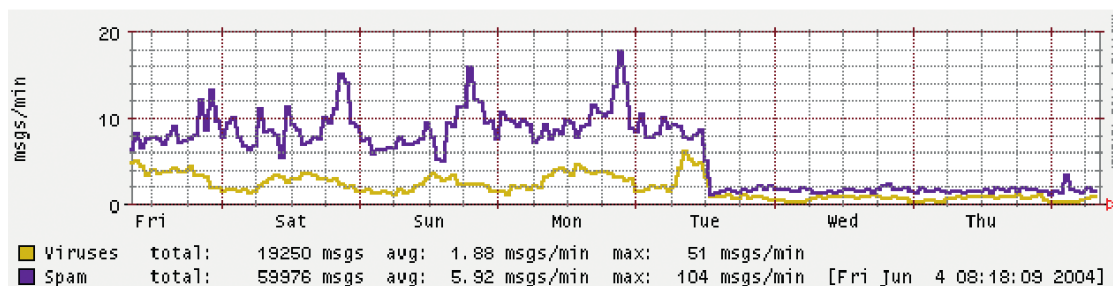
```
$ mkdir /tmp/eicar && pushd /tmp/eicar
$ wget https://secure.eicar.org/eicar.com
$ wget https://secure.eicar.org/eicar_com.zip
$ wget https://secure.eicar.org/eicarcom2.zip
$ popd && clamscan /tmp/eicar
- - - - - SCAN SUMMARY - - - - -
```

```
Infected files: 3
$ clamassassin < /tmp/eicar/eicar.com
X-Virus-Status: Yes
X-Virus-Report: Eicar-Test-Signature FOUND
```

We need to add a *Procmail* recipe that uses

Greylisting makes spam go away.

Image source: <http://postgrey.schweikert.ch>



**clamassassin** to detect viruses.

```
:0 wf
```

```
| /usr/bin/clamassassin
```

Any messages containing detected viruses will have an **X-Virus-Status** header added. We use this header in another *Procmail* recipe to deliver into a quarantine folder. Insert it just before the one that delivers clean mail:

```
:0 w
```

```
* ^X-Virus-Status: Yes
```

```
| $LMTP_DELIVER -m Virus $MAILBOX
```

You can also download a test email that has an attachment containing the EICAR virus and use it to test your *Procmail* configuration. Use your email client to create a Virus folder first.

```
$ wget https://bit.ly/eicar-testmail
```

```
$ procmail MAILBOX=testuser < eicar-testmail
```

*SpamAssassin* is a spam filter that uses various techniques to detect spam. These include message fingerprinting services, like Vipul's Razor, Pyzor and the Distributed Checksum Clearinghouse (DCC), and Bayesian Filtering that can learn what spam looks like if known spam is fed into it.

*SpamAssassin*'s default configuration performs Bayesian Filtering and will also use Pyzor and Razor if they are available. They need some configuration lines added to **/etc/mail/spamassassin/local.cf**:

```
bayes_path /srv/mail/spamassassin/bayes/bayes
```

```
bayes_file_mode 0666
```

```
pyzor_options --homedir /etc/mail/spamassassin
```

```
razor_config /etc/mail/spamassassin/razor/razor-agent.conf
```

You should review the other configuration items, adjusting it to suit your needs. You may like to rewrite the headers of spam messages so they contain a **\*\*\*\*\*SPAM\*\*\*\*\*** prefix.

```
rewrite_header Subject *****SPAM*****
```

*SpamAssassin* doesn't enable DCC, because it isn't open source but, if you want to use it, you can enable it by uncommenting the following line in **/etc/mail/spamassassin/v310.pre**:

```
loadplugin Mail::SpamAssassin::Plugin::DCC
```

and then initialise DCC:

```
$ rm -f /opt/dcc/map
```

```
$ chmod 600 /opt/dcc/map.txt
```

```
$ cdcc load /opt/dcc/map.txt
```

You need to create a Bayes directory for the **spamd** user and also register a Razor identity:

```
$ mkdir -pm 700 /srv/mail/spamassassin/bayes
```

```
$ chown -R spamd: /srv/mail/spamassassin
```

```
$ razor-admin -home=/etc/mail/spamassassin/razor -create
```

```
$ razor-admin -home=/etc/mail/spamassassin/razor -discover
```

```
$ razor-admin -home=/etc/mail/spamassassin/razor -register
```

Register successful. Identity stored in **/etc/mail/spamassassin/razor/identity-ruHZVUhY7x**

Before you can launch the *SpamAssassin* daemon, it needs some spam detection rules to use, which it expects to find under **/var/lib/spamassassin**. Use the **update** tool to download them:

```
$ sa-update
```

With the configuration complete, you can start the daemon and run some tests. You can download GTUBE, the Generic Test for Unsolicited Bulk Email, and use it to test your *SpamAssassin* setup.

```
$ wget http://spamassassin.apache.org/gtube/gtube.txt
```

```
$ systemctl enable spamassassin
```

```
$ systemctl start spamassassin
```

```
$ spamc < gtube.txt
```

X-Spam-Checker-Version: SpamAssassin 3.4.0 (2014-02-07) on mailserver

X-Spam-Flag: YES

X-Spam-Level: \*\*\*\*\*  
\*\*\*

X-Spam-Status: Yes, score=1000.0

You can feed known Spam into the Bayesian filter, however, that it may take a while before enough spam is learnt before Bayesian spam detection gives results:

```
$ sa-learn --spam /path/to/spam/emails
```

Learned tokens from 683 message(s) (683 message(s) examined)

Finally, we need to add a *Procmail* recipe to detect spam. We limit spam checks to emails smaller than 250KiB – most spam is smaller than this and having this rule avoids overloading **spamd**:

```
:0 fw
```

```
* < 256000
```

```
| /usr/bin/spamc
```

Any messages containing detected spam will have a **X-Spam-Status** header added that we use to quarantine them just like we did for viruses:

```
:0 w
```

```
* ^X-Spam-Status: Yes
```

```
| $LMTP_DELIVER -m Spam $MAILBOX
```

You can test your *Procmail* configuration with the **GTUBE** test file. Use your email client to create a Spam folder and then send a test spam into it:

```
$ procmail MAILBOX=testuser < gtube.txt
```

We need to change how our MTAs deliver mail so that it is processed through *Procmail*. For *Fetchmail*,

### LV PRO TIP

Use the **freshclam** command to update the virus database on-demand.

replace the defaults section in `/etc/fetchmailrc`:

```
mda "/usr/bin/procmail MAILBOX=%T"
```

When *Procmail* is invoked by *Fetchmail*, it's the **fetchmail** user that delivers mail. Allow this by adding it to the **mail** group:

```
$ usermod -aG mail fetchmail
```

Two changes are required for *Postfix*. First, in `/etc/postfix/main.cf`, change the **virtual\_transport** so that it reads

```
virtual_transport = procmail
```

This tells *Postfix* to use a transport called **procmail** to deliver mail. We define this transport by adding a new definition to the end of `/etc/postfix/master.cf`. It states that mail should be delivered by launching *Procmail*:

```
procmail unix - n - - pipe
```

```
flags=OR user=cyrus argv=/usr/bin/procmail -t -m MAILBOX=${recipient} /etc/procmailrc
```

Reload *Postfix* and then send some test emails and look for them in your inbox. Congratulations, your incoming emails are now processed through your filtering system for a spam-free life!

## In Submission

In part 1 we mentioned the Message Submission Agent (MSA) that clients should use to send email instead of sending it to the MTA's SMTP port 25. The MSA accepts submissions on port 587 with or without TLS. By implementing MSA, we gain several advantages, including the ability to have separate control over inbound and outbound messages. We need to enable MSA in `/etc/postfix/master.cf`. Uncomment the **submission** daemon and the following lines so that it looks like this:

```
submission inet n - - smtpd
```

```
-o syslog_name=postfix/submission
```

```
-o smtpd_client_restrictions=permit_mynetworks,reject
```

This allows clients on your network to connect and send but any other connections would be rejected. Reconfigure your email client to use port 587 instead of port 25 and send a test message to confirm that you can send. We changed the log name so that the logs label connections to the submission service differently; you can confirm via the logs that your email client is sending to the correct service.

We can now prohibit local clients from sending to port 25. Create a lookup table to list the local networks that should not be able to send via the MTA port. You can also permit specific addresses if necessary. The CIDR (Classless Inter-Domain Routing) table format is suitable for specifying networks; here is an example `/etc/postfix/smtp_access.cidr` that prohibits internal networks except for a specific address (customise yours according to your needs):

```
10.0.1.100      OK
10.0.0.0/8      REJECT
172.16.0.0/12   REJECT
192.168.0.0/16  REJECT
```

Add a rule in `/etc/postfix/main.cf` to check client connections against the access table. The default

## A Procmail primer

The **procmailrc** script is a series of recipes that are applied sequentially to a message. Each recipe begins with the cryptic **:0** character sequence followed by optional conditions and an action to perform if the conditions are met. A recipe is considered matched if its conditions are met.

Besides recipes, you can assign values to variables. The special **LOG** variable writes anything that is assigned to it into the log.

A recipe may have flags after its opening **:0**. We've used these flags in our examples:

- **h** makes the rule process message headers only.
- **w** makes the rule wait for its action to

complete.

- **f** means the rule is a filter and can alter the message.

A condition is an asterisk and a regular expression or a shell command. The condition is satisfied if the expression matches or a command succeeds.

Actions are either a file to write the message to (we've used `/dev/null` in a few places) or they begin with a pipe symbol and launch a command, passing the message into its standard input. Actions are assumed to deliver the message and processing stops on the first **suvh** recipe to be completed (filter recipes are non-delivering).

action is to permit so that genuine SMTP mail delivery from the internet is allowed:

```
smtpd_client_restrictions =
```

```
check_client_access cidr:/etc/postfix/smtp_access.cidr
```

Use **postfix reload** so that your changes take effect, and then perform some tests from an email client to ensure that you can send on port 587 but not port 25. Verify also that incoming messages still work!

Another thing that using a MSA makes easy is outbound filtering. You can also use *Procmail* to filter outbound messages. You first configure a **content\_filter** on the submission service that invokes another service to pass the message into *Procmail*, which must re-inject it back into the message queue using the *Postfix* **sendmail** command. This configuration goes in `/etc/postfix/master.cf`, beginning with the content filter. Add the following to the existing **submission** definition, after the client restrictions:

```
-o content_filter=procmail-outbound
```

Now, define the **procmail-outbound** service; append to the end of the file:

```
procmail-outbound unix - n - - pipe
```

```
flags=Rq user=cyrus argv=/usr/bin/procmail -t -m
```

```
SENDER=${sender} /etc/procmail/outbound-recipes
```

Here's an example **outbound-recipes** that uses **formail** to add a header to the outbound message before queuing it using *Postfix*'s "sendmail" command:


```
:0 f
```

```
| formail -fA "X-Outbound-Content-Filtered: Yes"
```

```
:0 w
```

```
| /usr/bin/sendmail -G -i -t -f $SENDER
```

```
EXITCODE=$?
```

We've now provided ways to separately filter both inbound and outbound mail, and you can build on these concepts to provide filters according to your needs. In part 3 we'll provide a way for end-users to filter their own messages so they can organise them into sub-folders and look at how users can access mail when out of the office. 

John Lane is a technology consultant with a penchant for Linux. He helps new businesses and start-ups make the most of open source software.