

## Filtering E-Mail with Postfix and Procmail, Part Four

Brian Hatch 2002-07-25

### Filtering E-Mail with Postfix and Procmail, Part Four

by Brian Hatch

last updated July 25, 2002

---

This is the fourth and final installment in a 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. The third segment introduced methods of stopping spam with Procmail. This installment will discuss 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.

## Distributed Spam Detection and Reporting with Razor

The whole theory behind spamming is to send huge numbers of messages to everyone the world over, using as little local resources as needed. The least amount of CPU and network usage occurs when you send out many identical messages. Any variation, such as custom From/To/Subject lines or message body requires additional processing which spammers would prefer to avoid.

We can bank on the fact that unsophisticated spammers send the exact same message to everyone on the planet. It'd be nice if I could tag a message as spam, and tell all my friends not to bother reading it. However how do you do this? Obviously sending the spam itself doesn't solve anything, it just makes the matter worse.

Instead, if I were to calculate a digest of the message body, I'd have a short string that I could send you, and you could delete, ignore, or refile any messages that had the same digest. However that still isn't great, because I need to somehow communicate those digests to you.

Enter Vipul's Razor, an automated spam tagging and filtering tool. It allows users across the Internet to rely on each other to correctly identify spam messages. Installation is pretty painless. Simply download the razor-agents tarball from <http://razor.sourceforge.net/> and install:

```
$ tar xzvf razor-agents-VERSION.tar.gz
$ cd razor-agents-VERSION
$ perl Makefile.PL
```

There are a few perl modules that are required, namely Net::Ping, Net::DNS, Time::HiRes, Digest::SHA1, and Mail::Internet. You can install them manually, via CPAN like this:

```
# perl -MCPAN -e "install 'Net::Ping'"
```

or download the razor-agents-sdk tarball from the Razor page, which has them all bundled up for you.

Once installed, you have two main programs that you'll use: Razor-report and Razor-check.

## Razor-report

When you get a spam message, you will use the razor-report program to compute a SHA Digest of the message body and report that digest to a Razor server. A SHA Digest is simply a small hex string that represents the data. SHA Digests are strong enough that it is extremely unlikely that two different messages will have the same digest.

The razor-report program will connect to a Razor server and submit this digest to the database, where it can be used by other people. It simply takes the message on the command line or standard input. You can test it like this:

```
$ razor-report -d -s definately_spam_file
debug: Razor Agents 1.19, protocol version 2.
debug: Discovering closest server in the razor-report.vipul.net zone
debug: Sorted (closest first) list of available servers & RTTs:
debug: 64.90.187.2 (0.0915) 194.109.217.74 (0.1510)
debug: Wrote server list to .razor-report.lst
debug: Closest server is 64.90.187.2
FATAL: Razor Error 4: This is a simulation. Won't connect to 64.90.187.2.
debug: Agent terminated
```

The '-d' options selects more debugging info, and '-s' tells it to simulate the report, such that it doesn't actually contact the server and add the digest to the database. It's important to not add spurious digests, because then legitimate mail that matches the digest may be rejected down the road.

Razor-report automatically determines which of the razor servers are closest to you to allow the best response times. The razor servers share the digests between them automatically.

## Razor-check

Razor-check queries the Razor databases to determine if a message you've received is spam. It takes a mail message on the command line or standard input, computes the digest, and looks up the digest in the closest razor server:

```
$ razor-check -d potential_spam_file
debug: Razor Agents 1.19, protocol version 2.
debug: 168968 seconds before closest server discovery
debug: Closest server is 64.90.187.2
debug: Connecting to 64.90.187.2...
debug: Connection established. Returning self
debug: Signature: a8e0ade8d5037db329f464b2ec62cbccdab13612
debug: Server version: 1.11, protocol version 2
debug: Server response: Negative a8e0ade8d5037db329f464b2ec62cbccdab13612
debug: Message 1 NOT found in the catalogue.
```

```
debug: Agent terminated
$ echo "Razor-check returned: $?"
Razor-check returned: 1
```

In this case, the digest was not found in the database, and the razor-check program return the value 1. If a message is found, then it will return 0.

## Integrating Vipul's Razor with Procmail

Since razor-check returns 0 or 1 depending on if the message is spam or not, we have a simple way to have procmail tell the difference. Say we want to have all Razor-identified spam go to a special mailbox, we'd simply add the following to our .procmailrc:

```
:0 Wc
| razor-check

:0 a
IN.razor
```

This will make all Razor spam go to the IN.razor mailbox. Of course we could do other things with the mail, such as mangling the subject line instead:

```
:0 Wc
| razor-check

:0 af
| formail -i "Subject: SPAM (Identified by Vipul's Razor)"
```

or adding spam headers, which could be used by your mail client:

```
:0 Wc
| razor-check

:0 af
| formail -A "X-Spam-Identifier: Razor" -A "X-Spam-Probability: High"
```

Razor-check is all you need to tag your message as spam. However, if you want to give back to the community that is helping, you'll want to submit your own spam digests.

## Reporting Spam to Vipul's Razor

All you need to do when you get a piece of spam is to run it through razor-report. However this still requires saving the mail and running razor-report, or piping it directly if your mail client supports it. To make it easier to report spam, and save us all from seeing it, you have a couple other options:

## Setting up a dedicated razor-report mailbox

Set up a dummy user on your system with the following `.procmailrc`:

```
:0c
| razor-report
```

Then any time you want to report spam, simply bounce it to this e-mail account. If you want to be even more proactive, you could link to this e-mail address from the Web, post to newsgroups using it, and eventually spambots will learn it and send spam to it, automatically reporting it. For example, I set up [razor@ifokr.org](mailto:razor@ifokr.org) for this purpose. (Don't send mail there, you have been warned.)

## Configuring your mailer to report spam

If you use a configurable mail client, you can probably map a key to bounce the message automatically. For example in Mutt I mapped the F8 key in mutt by adding the following to my `.muttrc`:

```
macro pager 'b razor@ifokr.org^Myd'
macro index 'b razor@ifokr.org^Myd'
```

The `^M` in the two strings above is a literal carriage return. To generate it, you'd type "Ctrl-V Ctrl-M" in vi, for example. This bounces the e-mail and then deletes it, and I have it mapped to both the index and pager.

I separate my Razor-identified spam to `IN.razor`. I direct other probable spam to `IN.spam`. I have the following mutt commands in my `.muttrc` to allow me to report and delete all the spam in `IN.spam` by hitting F8:

```
# We don't want f8 in any folders except IN.spam,
# so replace any f8 hook with 'refresh screen'
# (ctrl-l) by default.
folder-hook . "macro index <f8> '^L' "

# Enable f8 for IN.spam only
folder-hook IN.spam "macro index <f8> 'T.^M;b razor@ifokr.org^My;d' "
```

This macro will tag all messages, bounce them all to the reporting mailbox, and delete them. It's only enabled in the `IN.spam` mailbox for safety.

If you don't have a dedicated mailbox to report spam, you could instead just run `razor-report` locally, for example using the following macro bindings:

```
macro index "|razor-report"
macro pager "|razor-report"
```

## Drawbacks of Vipul's Razor

As spam preventionists come up with new ways to detect spam, the spammers adapt their software to

compensate. Spammers are now beginning to alter the body of their messages to prevent Vipul's Razor from functioning. Since even a one byte change in the body will result in a different SHA1 digest, none of the spam messages will be blocked. They can do this in a number of ways. They sometimes include your e-mail address as part of the message, such as

```
"This message was intended for razor@ifokr.org, as part
our supposed opt-in commitment to hawking our wares..."
```

Sometimes their "remove yourself from our database" link contains an ID associated with your e-mail address, or they address you by name at the top of the message. Any of these modifications will render Vipul's Razor useless. In fact, there's no point in reporting this spam, because all it will do is add a digest to the database for an e-mail that no one else will get.

One of the most transparent methods that a spammer can use to vary an e-mail to avoid detection by Vipul's Razor is to add a line of random text at the bottom of each e-mail. Since spams are frequently HTML, spammers can even add this random text after the end of the html and the user will never see it. This method requires minimal resources on the spammer to produce unique messages that are undetectable by Razor.

Some spammers go so far as to radically alter the messages. I've seen an upswing of HTML-formatted spam that litters the message with HTML comments, such as:

```
We w<!--church-->ant to help you get lo<!--jesus-->wer
HOUSE pa<!--dads trailer-->yments
```

In a browser, this simply reads "We want to help you get lower HOUSE payments," but due to all the random HTML comments it remains unique from the point of a SHA digest. You could write Procmail rules to check for these tricks, but it'd be difficult. SpamAssassin, which I cover next, has checks that may thwart this form of message obfuscation. However, I have a very simple solution I've used for years: block HTML e-mail.

No one I care to talk to sends mail as HTML. At worst folks send it as both HTML and text. Thus I have the following procmail rule to direct HTML mail to my spambox:

```
:0:
* ^Content-Type: text/html
IN.spam
```

I check my spam box around once a day for misfiled messages from people with bad e-mail netiquette. But I'd have to say that 49 out of 50 HTML messages are spam. Send 'em where they belong.

## Razor in the Future

At the time of this writing, Vipul's Razor v2 was just entering beta. This new version promises a plethora of exciting new features to better analyze and match spam, even when spammers employ tricks to defeat our checksumming algorithms, such as interspersing HTML comments, as seen previously. Among the new features

are

- **Text Preprocessors** - New message processors which can convert HTML to text, decode quoted-plain and Base64 encoded messages, will cleanse the message of many checksum-defeating tricks employed by the spammers.
- **Revocation** - Razor V1 had no way to revoke a result from razor-report. You can revoke messages you accidentally reported, or messages that you believe others reported in error.
- **Fuzzy Signatures** - Razor can use Nilsimsa signatures, a fuzzy algorithm which compares how close a message is to one in the database.
- **Truth Evaluation System** - Users who wish to report spam to Razor must now register and authenticate. The system will be able to track the quality of the reports, and can use this to keep questionable checksums out of the database.
- **Performance Improvements** - Razor v2 uses a better network protocol, the ability to use a single pipelined connection instead of multiple individual connections, and other enhancements to make it faster and more efficient.
- **Message Submission** - Razor servers will accept the entire spam message when submitted. The server can then perform its own checksums and other similar ephemeral signatures – hashes on periodically changing sections of the e-mail message.

## SpamAssassin

I've shown you how you can create Procmail recipes that will help weed out spam, and how Razor can catch messages that other users have flagged as spam. Unfortunately, there are a lot of messages that still sneak through the cracks. Most spams contain similar formatting and phrases. The problem is that it's a lot of work for individuals to maintain their own Procmail recipes to match spam characteristics.

SpamAssassin is a mail filter that contains hundreds of different spam tests. It is available at <http://spamassassin.org>, as well as through CPAN, and is included in some Linux distributions. Installation instructions are contained on the Web site, but boil down to:

```
$ tar xzvf Mail-SpamAssassin-VERSION.tar.gz
$ cd Mail-SpamAssassin-VERSION
$ perl Makefile.pl
$ make
$ make test
# make install
```

This installs the SpamAssassin software itself. You should test out the installation by running the sample messages through SpamAssassin:

```
$ spamassassin -t < sample-spam.txt | less
$ spamassassin -t < sample-nonspam.txt | less
```

The first message should be recognized as spam, and SpamAssassin will prepend '\*\*\*\*\*SPAM\*\*\*\*\*' to the Subject: line. The second message, though it contains some spam-like characteristics, does not exceed the spam threshold. Both messages will have new X-Spam headers inserted, and a report of all the spam tests that matched. In normal usage you will only get this report when a message is spam, but we get it even for legitimate e-mails due to the "-t" test mode.

## SpamAssassin Features

SpamAssassin has hundreds of tests to help it determine if a message is spam or not. It weights these differently, depending on how accurately it fits spam. It even has tests that indicate non-spam messages (pgp signatures, output from "diff") that can counteract spam-like characteristics in legitimate e-mail. For a terse list of the tests and their scores, see <http://spamassassin.org/tests.html>. SpamAssassin sums the results of all tests and computes a final score for the message. By default any message of score 5 or higher is considered spam, although you can change this value if you wish.

SpamAssassin can perform DNSBL lookups based on the IP addresses and host names listed in the headers. It has whitelisting capabilities to allow legitimate but misflagged e-mail. It has an automated whitelisting feature, which keeps track of sender addresses and their spam vs non-spam history to automatically adjust scores. It can even report spam to Razor automatically if a threshold is reached. If you decide to use SpamAssassin, make sure you read all the features that are available, because I don't have enough space to cover them all here.

## X-Spam Headers

SpamAssassin is traditionally used as Procmail filter and will edit the headers of the message to include the results of it's spam tests. Simply add the following to your .procmailrc, or to /etc/procmailrc if you wish it to apply to all users:

```
:0fw
| spamassassin -P
```

SpamAssassin will analyze the e-mail and create several new headers:

- **X-Spam-Flag:** YES or NO, is the message spam (was the hit threshold reached.) This header is the easiest to use with Procmail.
- **X-Spam-Level:** Includes one asterisk (\*) for each hit. Useful for matching messages with X or greater hits by counting the asterisks in Procmail rules.
- **X-Spam-Status:** Includes the actual number of hits for this message, as well as the individual tests that matched. Can allow more sensitive Procmail selections.
- **X-Spam-Checker-Version:** SpamAssassin version.
- **X-Spam-Prev-Content-Type:** If SpamAssassin is configured to 'defang mime' (the default) then if the message is spam, it will replace the existing mime type with text/plain. This will stop things like

JavaScript or HTML bugs from being effective.

## Redirecting Tagged Spam

Both the Subject (which is modified if a message is spam) and the new X-Spam headers can be used by subsequent Procmail rules. For instance:

```
# Have SpamAssassin analyze the mail and insert headers
:0fw
| spamassassin -P

# Redirect definitive spam
:0:
* ^X-Spam: YES
$SPAM

# Redirect things that we consider spam,
# even if SpamAssassin doesn't agree by default
:0:
* ^X-Spam-Status:.*CTYPE_JUST_HTML
$SPAM

# Put messages with scores of 3 or more into probable spam
# mailbox. The three dots here match three or more asterisks.
:0:
* ^X-Spam-Level: ...
$PROBABLY_SPAM
```

## Changing User Preferences

Every SpamAssassin user can create his or her own preference file, which tweaks how it functions. This can help you rid yourself of historical Procmail rules by integrating them into SpamAssassin natively. Preferences live in the `.spamassassin/user_prefs` file in each user's home directory. For example:

```
$ cat ~/.spamassassin/user_prefs

# The default is way too low. Let's bump it up.
score CTYPE_JUST_HTML 4.5

# Grandpa gets flagged a lot due to his sticky shift key
whitelist_from grandpa@his_e-mail.com

# Lower the number of hits required:
required_hits 3

# Turn off re-write subject if you don't like *****SPAM*****
# put in your Subject: line.
rewrite_subject 0
```

```

# I get a lot of spam to this address, but haven't
# used it in several years
header    TO_RETIRED_E_MAIL      To =~ /retired\@e-mail_address.org/
describe  TO_RETIRED_E_MAIL      Mail to an old retired e-mail address
score     TO_RETIRED_E_MAIL      3.00

```

Here I tweaked a few parameters to lower the spam threshold, turned off subject rewriting (which is not really needed if you redirect spams to their own mailbox), created a whitelist address, increased the score for CTYPE\_JUST\_HTML (and can now remove the special case in the .procmailrc in the last section), and even created my own new TO\_RETIRED\_E\_MAIL rule. These changes will only affect this particular user, allowing a high degree of customizability. Of course if you wanted to make global changes, you can add or change rules in the default SpamAssassin rules, which are typically in /usr/share/spamassassin/ or /etc/spamassassin/.

### Using the SpamAssassin Daemon

SpamAssassin is a perl script with many rules and, as such, takes a bit to compile internally each time it is run. This leads to a performance hit of a few seconds before the message is even analyzed, which will slow down your message delivery.

SpamAssassin comes with a daemonized version, named "spamd", and a small efficient client program named "spamc". Instead of running the SpamAssassin perl script from Procmail, you'd start the spamd daemon and put the following into your procmailrc:

```

:0fw
| spamc

```

Spamc will pipe the mail into the SpamAssassin daemon, and will rewrite the result back to stdout, filtering the e-mail just as if you had run SpamAssassin directly. However, since spamc has little startup overhead the processing goes much quicker. In my tests, using spamc with spamd resulted in a fourfold performance improvement over running SpamAssassin alone.

The spamd daemon still honors user's user\_prefs file, so you only need to run one daemon to service everyone on the machine. However, you must make sure that the user\_prefs file is readable by the spamd process. If you run spamd as a dummy user instead of as root, as suggested, this means you need to have at minimum the following permissions for users with user\_prefs files:

```

$ chmod go+x ~user
$ chmod go+x ~user/.spamassassin
$ chmod go+r ~user/.spamassassin/user_prefs

```

If you make global changes to the SpamAssassin rules, you will need to restart spamd, but user preferences are read each time automatically.

## Conclusion

In this series of articles I've shown you some straightforward steps you can take to keep spam off of your network and out of your mailbox. By implementing various spam protection mechanisms at the optimal points, you can have a spam-free experience without overburdening your systems.

Blocking spam does require a significant amount of education and configuration up front as you create your rules on the SMTP server (Postfix), local delivery agent (Procmail), and any external spam-detection software (Razor, SpamAssassin) that you decide to use. However, once you have the pieces in place, it is largely a system that you can sit back and forget about entirely.

There are many other spam-blocking software packages out there besides Razor and SpamAssassin. Below is a list of mature spam and filtering software that you may want to look at when picking the right tools for you.

- [Distributed Checksum Clearinghouse](#)
- [Spam Bouncer](#)
- [Junkfilter](#)
- [DeSpam](#)
- [Tagged Message Delivery Agent \(TMDA\)](#)
- [Active Spam Killer \(ASK\)](#)

There are doubtless many more, but those are some packages that I've played with and can recommend.

Happy filtering!

*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*

[Filtering E-Mail with Postfix and Procmail, Part Three](#)  
*Brian Hatch, SecurityFocus*

[Privacy Statement](#)

Copyright 2006, SecurityFocus