

Filtering E-Mail with Postfix and Procmail, Part Two

Brian Hatch 2002-06-26

Filtering E-Mail with Postfix and Procmail, Part Two

by Brian Hatch

last updated June 26, 2002

This article is the second of three articles that will help systems administrators configure SMTP daemons and local mail delivery agents to filter out unwanted e-mails before they arrive in the end-users' in-box. In the [first installment](#), we offered a brief overview of Postfix, and began a discussion of rejecting spam with Postfix by blocking e-mail based on the SMTP client, blocking machines with no reverse DNS, SMTP client map restrictions, DNS blackhole lists, bundling Postfix restriction options, and blocking spam based on SMTP Compliance. In this part, we will look at sender/recipient restrictions, restriction ordering, and map file naming conventions before moving on to Procmail in the final article.

Postfix Sender/Recipient Restrictions

Postfix can easily make spam determinations based on the sender or recipient of the e-mail. It is important to understand that there are potentially two different addresses for the sender and recipient of each e-mail. The first is the envelope address, which is presented during the SMTP conversation. The second is the actual "From:" or "To:" address in the message header.

To illustrate, let's look at an SMTP session, again using our friend netcat:

```
# nc smtp.example.com 25
220 smtp.example.com ESMTP ReegenMail
helo some.host.dom
250 smtp.example.com
mail from: spammer@no_such_domain.com
250 Ok
rcpt to: innocent_bystander@example.com
250 Ok
data
354 End data with <CR><LF>.<CR><LF>
From: FooBar@some_address.net
To: My_Friend@some_other_address.net
Subject: You've just got to see this!

(blahblahblah)
.
250 Ok: queued as 1F75BD8
quit
#
```

The above message has an envelope sender of "spammer@no_such_domain.com". If this mail bounces, this is the address to which an error message will be sent. SMTP servers don't look at the actual contents of the e-mail to determine where the e-mail should go. Thus, although the "From:" line indicates that the message was sent from

"FooBar@some_address.net", this is easily forgeable. Similarly, the envelope recipient here is "innocent_bystander@example.com" while it is listed as "My_Friend@some_other_address.net" in the mail headers themselves. The restrictions immediately available to Postfix are the envelope addresses (MAIL FROM/RCPT TO), not the one's in the mail headers (DATA).

Sender Restrictions

Spammers frequently use bogus sender addresses to avoid dealing with the inevitable bounces and flames. Postfix can try to determine if an address is invalid and, if so, assume the message is spam. Unfortunately, Postfix cannot actually verify that the e-mail address is valid without attempting to send mail to it. However, it can check to see that it could be possible to send e-mail to that address by verifying that there is an MX or A record for the domain itself. To enable this, use the following restriction:

```
smtpd_sender_restriction = reject_unknown_sender_domain
```

If the sender domain is unknown, Postfix responds with a 450 (temporary) error, which means that the client should retry the message later. This is important, as you don't want a temporary DNS problem to result in the loss of legitimate e-mail. However, if there's no way for you to possibly reply to this message, it is most likely spam.

"MAIL FROM" addresses should be fully qualified ("mailserver.example.com", instead of "mailserver" for example) if you want to be sure you can send return e-mail. You can drop mail that is not fully qualified by using:

```
smtpd_sender_restriction = reject_non_fqdn_sender
```

Lastly, you can create a map file listing valid or invalid senders, such as:

```
# allow larry@example.com
larry@example.com          OK

# reject anything else coming from @example.com
example.com                REJECT
```

and integrate it with your spam determination via:

```
smtpd_sender_restriction = check_sender_access maptype:mapname
```

If you wanted to include all these restrictions, the most logical method would be to specify:

```
smtpd_sender_restriction = check_sender_access maptype:mapname,
    reject_non_fqdn_sender, reject_unknown_sender_domain
```

Recipient Restrictions

Our last restriction is based on the message recipient (again, the address listed in the 'RCPT TO' SMTP dialog, not the 'To:' address). The recipient restrictions are similar to the sender restrictions, namely `reject_known_recipient_domain`, `reject_non_fqdn_recipient`, and `check_recipient_access`, and they work in the same

manner. Thus, you could include the following options in addition to any you already have:

```
smtpd_recipient_restriction = (other restrictions here)
    check_recipient_access maptype:mapname,
    reject_non_fqdn_recipient, reject_unknown_recipient_domain
```

Your Postfix configuration should already have some `smtpd_recipient_restriction` value set. In order to prevent your mail machine from acting as an open relay, Postfix requires that you have some rules (using the "check_relay_domains", "reject_unauth_destination", or "reject" options) to dictate which machines are allowed to relay e-mail through your server. The correct settings for these are specific to your environment. A common entry is "permit_mynetworks, check_relay_domains", so to add spam detection rules, you might change this to:

```
smtpd_recipient_restriction = permit_mynetworks,
    check_relay_domains, check_recipient_access maptype:mapname,
    reject_non_fqdn_recipient, reject_unknown_recipient_domain
```

Just as a reminder, make sure that you don't eliminate your current anti-relaying settings while trying to add spam filtering, or your machine will reject all incoming e-mail!

Postfix Restriction Ordering

Postfix processes the various restrictions in the following order:

```
smtpd_client_restrictions
smtpd_helo_restrictions
smtpd_sender_restrictions
smtpd_recipient_restrictions
```

This is, unsurprisingly, the order in which the SMTP session occurs. These restrictions may (and often do) contain multiple checks. Each one of them will result in one of three values:

- OK - The mail is good, let it through;
- DUNNO - Inconclusive results from this restriction. Check the remaining restrictions to see what to do; and,
- REJECT - The mail should be rejected. Don't check any later restrictions.

If a restriction returns "DUNNO", then the later restrictions will be checked. If it returns either "OK" or "REJECT", then the later restrictions are **NOT** checked. As soon as one of the restrictions determines that the message is or is definitely not spam, no further checks are performed.

The problem arises if you want to have later restrictions override a previous restriction. For example, you may have an e-mail address 'spam-troll@example.com' that exists solely to receive spam and send it to a spam-reporting site. You would want all mail to this account to be delivered, even if earlier restrictions (say a HELO check) would normally block the e-mail.

The easiest method is to put all your restrictions into the `smtpd_recipient_restrictions` option. At the point that the recipient has been sent by the client, Postfix has everything but the actual e-mail itself to help make a friend-or-

foe determination.

This may sound surprising - thus far I've been implying that the various options belong to a specific restriction. In actuality, I've been showing you the default place to put the restriction options. Any check can be performed in the earliest or later restrictions. A DNSBL check, for example, is a restriction based on the client IP address, and thus would most "logically" belong as:

```
smtpd_client_restrictions = reject_rbl_domains
```

However, it could be put in any of the later restriction clauses as well, such as

```
smtpd_recipient_restrictions = check_recipient_access
    hash:/etc/postfix/recipients, reject_rbl_domains
```

This would allow our spam-troll@example.com mail (presumably listed as "OK" in the /etc/postfix/recipients map file) to be delivered, even if the system sending the e-mail is listed in the DNSBL domains.

As you can see, it can be a bit tricky to tweak your Postfix restrictions just right, as you have multiple restrictions you can play with, and multiple checks you can do in each restriction. I suggest that you test and retest any changes you make to your configuration.

For a full list of the checks available for each of the restrictions, you should check the Postfix man pages, and also the [Postfix UCE Web page](#). If you need to do advanced restrictions, in cases where sticking them all into smtpd_recipient_restrictions isn't sufficient, check out the [Meng Weng Wong's Postfix UCE guide](#), which provides helpful hints about pitfalls you're likely to encounter.

Header and Body Restrictions

All the restrictions listed thus far are performed during the initial SMTP conversation, before the actual e-mail is accepted. If these restrictions are met, then the client may issue a DATA command, and Postfix will begin slurping up the message data itself. However, Postfix has one more chance to reject the e-mail. After the client sends the "." to signify the end of the message data, Postfix would return a 250 response code if all is well. However, if you wish to reject e-mail based on the message contents, Postfix can respond with an error, and will not accept the message.

This method is useful because the client machine will think your address is invalid, which may get you taken off their spam list. It also means that you haven't sent the message along to your local delivery agent. However, you are receiving the e-mail over the network, so your bandwidth has not been spared.

There are two message content-checking algorithms. The first is header_checks, which compares lines against the message headers (everything from the beginning of the message up until the first blank line.) The other is body_checks, which checks only the body of the message. Using header_checks is faster than body_checks, because message headers are typically 1-2K, while the message body is much larger. Both use regular expression maps instead of the kinds of maps we used in the smtpd_*_restrictions. A regular expression map file looks like this:

```
# comment
```

```

/regular expression1/    ACTION
/regular expression2/    ACTION

# You can also specify two patterns, in which
# case the line must match the first but not
# the second
/must_match_me/!/must_not_match_me/    ACTION

```

The valid actions are:

- REJECT - Rejects the message with a 554 error code, and the generic error message "Message content rejected."
- REJECT text - Rejects the message with a 554 error code, using the supplied text as the error message. (Not all versions of Postfix support this form)
- WARN - logs the header via syslog, but does not actually reject the message. Good for testing. (Not all versions of Postfix support this.)
- IGNORE - Causes the matching line to be deleted from the message. Yes, the term IGNORE is a bit of a misnomer.
- OK - Valid, but useless.

The header_checks and body_checks look through the message until a REJECT, WARN, or the end of the message is reached. For this reason, "OK" doesn't really do anything except waste CPU cycles. Using "OK" will **NOT** short-circuit the logic and accept the message: Postfix will continue to look through the rest and may well find a REJECT line that matches.

There are two kinds of regular expression maps that Postfix can support: regexp, which are standard regular expressions used by grep and friends, and pcre, which are perl-compatible regular expressions. Use whichever one has the functionality you need and you are most comfortable with. Thus your main.cf could look include lines like this:

```

header_checks = pcre:/etc/postfix/header_checks
body_checks = pcre:/etc/postfix/body_checks

# Or, if we prefer regexp instead:
# header_checks = regexp:/etc/postfix/header_checks
# body_checks = regexp:/etc/postfix/body_checks

```

Then simply create your header_checks and/or body_checks files. For example:

```

# Header checks file
/^Subject: Internet Sic Codes/    REJECT
/^Subject: ADV /                    REJECT

# Encourage good Netiquette
/^X-Mailer: Microsoft Outlook/    REJECT Sorry, we don't accept mail from LookOut!

```

```
# Body checks file
/you signed up to receive/ REJECT
/this is not spam/ REJECT Liar.
/rent is overdue/ REJECT Sorry, your message cannot be delivered successfully.
```

The regular expressions are case insensitive by default. If you want to force case sensitivity, include a "i" at the end of the closing slash (yes, this is backwards from the norm). You can also extract sections of the matched pattern using parens, which you can then include in your error message using `${1}`, `${2}`, and so on. Read the sample `regexp` and `pcre` files included in the Postfix distribution for more information and examples.

How to Avoid Bouncing Legitimate E-Mail During Configuration

When you're starting to implement your spam prevention configuration, you're quite likely to make errors that turn your mail server into an anti-social mail-dropping hunk of silicon. You can add "warn_if_reject" options before restrictions to tell Postfix to merely log that the message would have been rejected, without actually rejecting the e-mail. You then need to check your system logs for instances of "reject_warning" and make sure that the rules are acting correctly. Once you feel confident of your changes, remove the "reject_warnings" option and your spam prevention is in place for real. Similarly, when tweaking `header_checks` and `body_checks`, use the `WARN` option instead of `REJECT`.

Map File Naming Conventions

The default Postfix configuration and sample files use the same file for all its maps: `/etc/postfix/access`. This means it would use this file for all its restrictions, which may be more restrictive than you wish. Say I had all my maps in `/etc/postfix/access`, like:

```
smtpd_something_restrictions = ... ,
    check_recipient_access hash:/etc/postfix/access,
    check_sender_access hash:/etc/postfix/access,
    check_client_access hash:/etc/postfix/access
```

and the access file looked like this:

```
# reject mail from bad_domain.com
bad_isp.com REJECT
```

Then I would not be able to receive e-mail from `@bad_isp.com`, any machine that had a reverse DNS entry inside `bad_isp.com`, and I couldn't send any mail to `someaddress@bad_isp.com`.

If all I meant to do was keep people from sending mail from addresses of `@bad_isp.com`, but I do exchange e-mail with people who have their network connectivity through them, then using one file will not work. Similarly, an "OK" restriction could be overly permissive, resulting in spams sneaking through. I prefer to use separate maps for each

restriction, *à la*:

```
smtpd_something_restrictions = ... ,  
    check_recipient_access hash:/etc/postfix/access.recipient,  
    check_sender_access hash:/etc/postfix/access.sender,  
    check_client_access hash:/etc/postfix/access.client
```

This way I can specifically tailor the restrictions I need.

Conclusion: Postfix Summary

Postfix is your first line of defense against spam. It has a myriad of possible restrictions, which may seem a bit overwhelming at first. However, by careful experimentation you can create very useful anti-spam rules to keep unneeded bits off of your machine. In the next (and last) article in this three-part series, we will look at using Procmail to filter e-mail.

To read Part Three of this series, click [here](#).

Relevant Links

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

[Privacy Statement](#)

Copyright 2006, SecurityFocus