



Published on Free Software Magazine (<http://www.freesoftwaremagazine.com>)

Filtering spam with Postfix

Effective ways to reduce unwelcome mail

By Kirk Strauser

If you are responsible for maintaining an internet-connected mail-server, then you have, no doubt, come to hate spam and the waste of resources which comes with it. When I first decided to lock down my own mail-server, I found many resources that helped in dealing with these unwanted messages. Each of them contained a trick or two, however very few of them were presented in the context of running a real server, and none of them demonstrated an entire filtering framework. In the end I created my own set of rules from the bits and pieces I found, and months of experimentation and fine-tuning resulted in the system detailed in this article.

This article will show you how to configure a Postfix mail-server in order to reject the wide majority of unwanted incoming “junk email”, whether they contain unsolicited commercial email (UCE), viruses, or worms. Although my examples are specific to Postfix, the concepts are generic and can be applied to any system that can be configured at this level of detail.

For a real world example, I’ll use my server’s configuration details:

Hostname	Kanga.honeypot.net
Public address	208.162.254.122
Postfix configuration path	/usr/local/etc/postfix

My server’s configuration details

Goals

This configuration was written with two primary rules in mind:

1. Safety is important above all else. That is, I would much rather incorrectly allow an unwanted message through my system than reject a legitimate message.
2. The system had to scale well. A perfect system that uses all of my server’s processing power at moderate workloads is not useful.

It’s easy to experiment with Postfix, and it can be customized to your level of comfort

To these ends, several checks—that may have reduced the number of “false negatives” (messages that should have been rejected but were not) at the cost of increasing the number of “false positives” (legitimate messages that were incorrectly rejected)—were avoided. The more resource-intensive tests were moved toward the end of the configuration. This way, the quick checks at the beginning eliminated the majority of UCE so that the longer tests had a relatively small amount of traffic to examine.

HELO restrictions

When a client system connects to a mail-server, it's required to identify itself using the SMTP HELO command. Many viruses and spammers skip this step altogether, either by sending impossibly invalid information, or lying and saying that they are one of your own trusted systems—in the hopes that they will be granted undeserved access. The first step in the filtering pipeline, then, is to reject connections from clients that are severely mis-configured or that are deliberately attempting to circumvent your security. Given their simplicity, these tests are far more effective than might be imagined, and they were implemented in my `main.cf` file with this settings block:

```
1 smtpd_delay_reject = yes
2 smtpd_helo_required = yes
3 smtpd_helo_restrictions =
4   permit_mynetworks,
5   check_helo_access
6     hash:/usr/local/etc/postfix/helo_access,
7   reject_non_fqdn_hostname,
8   reject_invalid_hostname,
9   permit
```

Given their simplicity, these tests are far more effective than might be imagined

Line 1 is a fix for certain broken (but popular) clients, and is required in able to use HELO filtering at all. The second line rejects mail from any system that fails to identify itself. Line 4 tells Postfix to accept connections from any machines on my local network. The next line references an external hash table that contains a set of black- and whitelisted entries; mine looks like this:

```
woozle.honeypot.net      OK
honeypot.net            REJECT You are not me. Shoo!
208.162.254.122        REJECT You are not me. Shoo!
```

The first line in the table explicitly allows connections from my laptop so that I can send mail when connected through an external network. At this point Postfix has already been told to accept connections from all of my local machines and my short list of remote machines, so any other computer in the world that identifies itself as one of my systems is lying. Since I'm not particularly interested in mail from deceptive systems, those connections were flatly rejected.

Lines 6 through 8 complete this stage by rejecting connections from hosts that identify themselves in blatantly incorrect ways, such as "MAILSERVER" and "HOST@192.168!aol.com".

Some administrators also use the `reject_unknown_hostname` option to ignore servers whose hostnames can't be resolved, but in my experience this causes too many false positives from legitimate systems with transient DNS problems or other harmless issues.

You can test the effect of these rules, without activating them on a live system, by using the `warn_if_reject` option to cause Postfix to send debugging information to your maillog without actually processing them. For example, line 6 could be replaced with:

```
warn_if_reject,
reject_non_fqdn_hostname,
```

This way the results can be observed without the risk of inadvertently getting false positives.

Sender restrictions

The next step is to reject invalid senders with these options:

```

9  smtpd_sender_restrictions =
10     permit_sasl_authenticated,
11     permit_mynetworks,
12     reject_non_fqdn_sender,
13     reject_unknown_sender_domain,
14     permit

```

If the sender’s email address is malformed or provably nonexistent, then there’s no reason to accept mail from them

Lines 10 and 11 allow clients that have authenticated with a username and password or Kerberos ticket, or who are hosts on my local network, to continue onward. Lines 12 and 13 work similarly lines 6 and 7 in the “HELO restrictions” section; if the sender’s email address is malformed or provably nonexistent, then there’s no reason to accept mail from them. The next line allows every other message to move on to the next phase of filtering.

Recipient restrictions and expensive tests

By this time, it’s clear that the client machine isn’t completely mis-configured and that the sender stands a reasonable chance of being legitimate. The final step is to see that the client has permission to send to the given recipient and to apply the remaining “expensive” tests to the small number of messages that have made it this far. Here’s how I do it:

```

15 smtpd_recipient_restrictions =
16     reject_unauth_pipelining,
17     reject_non_fqdn_recipient,
18     reject_unknown_recipient_domain,
19     permit_mynetworks,
20     permit_sasl_authenticated,
21     reject_unauth_destination,
22     check_sender_access
           hash:/usr/local/etc/postfix/sender_access,
23     check_recipient_access
           hash:/usr/local/etc/postfix/recipient_access,
24     check_helo_access
           hash:/usr/local/etc/postfix/secondary_mx_access,
25     reject_rbl_client relays.ordb.org,
26     reject_rbl_client list.dsbl.org,
27     reject_rbl_client sbl-xbl.spamhaus.org,
28     check_policy_service unix:private/spfpolicy
29     check_policy_service inet:127.0.0.1:10023
30     permit

```

Many spammers send a series of commands without waiting for authorization, in order to deliver their messages as quickly as possible. Line 16 rejects messages from those attempting to do this.

Line 21 is critically important; without this line, the server would be an open relay!

Options like lines 17 and 18 are probably becoming familiar now, and they work in this case by rejecting mail targeted at domains that don’t exist (or can’t exist). Just as in the “Sender restrictions” section, lines 19 and 20

Filtering spam with Postfix

allow local or authenticated users to proceed—which here means that their messages will not go through any more checks. Line 21 is critically important because it tells Postfix not to accept messages with recipients at domains not hosted locally or that we serve as a backup mail server for; without this line, the server would be an open relay!

The next line defines an access file named `sender_access` that can be used as a black- or whitelist. I use this to list my consulting clients' mail-servers so that none of the remaining tests can inadvertently drop important requests from them. I added line 23, which creates a similar blacklist called `recipient_access` for recipient addresses, as an emergency response to a “joe job”. Once in 2003, a spammer forged an email address from my domain onto several million UCE messages and I was getting deluged with bounce messages to “michelle@honeypot.net”. I was able to reject these by adding an entry like:

```
michelle@honeypot.net REJECT This is a forged account.
```

Although the event was annoying, this allowed my server to continue normal operation until the storm had passed.

Line 24 is the last of the “inexpensive” checks. It compares the name that the remote system sent earlier via the `HELO` command to the list of my secondary mail servers and permits mail filtered through those systems to be delivered without further testing. This is the weak link in my filtering system, because if a spammer were clever enough to claim that they were one of my backup servers then my mail server would cheerfully deliver any message sent to it. In practice, though, I've never seen a spammer that crafty and this line could be removed without side effects should the need arise.

Lines 25 through 27 are somewhat more controversial than most of my techniques, in that they consult external DNS blackhole lists in order to selectively reject messages based on the IP address of the sender. Each of these lists have been chosen because of their good reputation and very conservative policies toward adding new entries, but you should evaluate each list for yourself before using their databases to drop messages.

SPF and greylisting

Lines 28 and 29 add Sender Policy Framework (SPF) and greylisting filtering respectively. SPF works by attempting to look up a DNS record, that domains can publish, which gives the list of addresses allowed to send email for that domain. For example, `webtv.net`'s SPF record is currently “`v=spf1 ip4:209.240.192.0/19 -all`”, which means that a message claiming to be from `joeuser@webtv.net` sent from the IP address `64.4.32.7` is forged and can be safely rejected.

Greylists are pure gold when it comes to rejecting junk email. Whenever a client attempts to send mail to a particular recipient, the greylist server will attempt to find that client's address and the recipient's address in its database. If there is no such entry then one will be created, and Postfix will use a standard SMTP error message to tell the client that the recipient's mailbox is temporarily unavailable and to try again later. It will then continue to reject similar attempts until the timestamp is of a certain age (mine is set to five minutes). The theory behind this is that almost no special-purpose spam sending software will actually attempt to re-send the message, but almost every legitimate mail server in existence will gladly comply and send the queued message a short time later. This simple addition cut my incoming junk email load by over 99% at the small cost of having to wait an extra five minutes to receive email for the first time from a new contact. It has worked flawlessly with the many mailing lists that my clients and I subscribe to and has not caused any collateral problems that I am aware of. If you take nothing else from this article, let it be that greylisting is a *Good Thing* and your customers will love you for using it.

Filtering spam with Postfix

If you take nothing else from this article, let it be that greylisting is a Good Thing and your customers will love you for using it

I use the `smtpd-policy.pl` script that ships with Postfix to handle SPF, and Postgrey as an add-on greylisting policy server. They're defined in my `master.cf` file as:

```
spfpolicy unix#x2014; n n &#x2014; &#x2014; spawn
  user=nobody argv=/usr/bin/perl
  /usr/local/libexec/postfix/smtpd-policy.pl
greypolicy unix#x2014; n n &#x2014; &#x2014; spawn
  user=nobody argv=/usr/bin/perl
  /usr/local/libexec/postfix/greylist.pl
```

Content filtering

The messages remaining at this point are very likely to be legitimate, although all that Postfix has actually done so far is enforce SMTP rules and reject known spammers. Their final hurdle on the way from their senders to my users' mailboxes is to pass through a spam filter and an antivirus program. The easiest way to do this with Postfix is to install AMaViS, SpamAssassin, and ClamAV and then configure Postfix to send messages to AMaViS (which acts as a wrapper around the other two) before delivering them, and line 31 does exactly that:

```
31 content_filter = smtp-amavis:127.0.0.1:10024
```

SpamAssassin is fantastic at identifying spam correctly. Its "hit rate" while used in this system will be much lower than if it were receiving an unfiltered stream, as most of the easy targets have already been removed. I recommend setting AMaViS to reject only the messages with the highest spam scores; I arbitrarily picked a score of 10.0 on my systems. Then, tag the rest with headers that your users can use to sort mail into junk folders.

ClamAV has proven itself to be an effective and trustworthy antivirus filter, and I now discard any message that it identifies as having a viral payload.

Unfortunately, the configuration of these systems is more complicated than I could hope to cover in this article, as it depends heavily on the setup of the rest of your email system. The good news is that literally less than 1% of junk email makes it this far into my system, and I've actually gone for several days without realizing that SpamAssassin or ClamAV hadn't been restarted after an upgrade. Still, these extra filters are very good at catching those last few messages that make it through the cracks.

Other possibilities

If you want more aggressive filtering and can accept the increased risk of false positives, consider some of the other less-conservative blackhole lists such as the ones run by [SPEWS](#) or the various lists of blocks of dynamic IP addresses. You may also consider using the `reject_unknown_hostname` option mentioned in the "HELO restrictions" section, but you can expect a small, measurable increase in false positives.

The ruleset described above should be sufficient on its own to eliminate the vast majority of junk email, so your time would probably be better spent implementing and adjusting it before testing other measures.

Conclusion

None of the techniques I use are particularly difficult to implement, but I faced quite a challenge in assembling them into a coherent system from the scraps I found laying about in various web pages and mailing lists.

It can be difficult to find a good compromise between safety and effectiveness, but I believe I've found a solid combination that should work in almost any situation.

The most important thing I learned from the process was that it's easy to experiment with Postfix, and it can be customized to your level of comfort. When used in my configuration, the most effective filters are:

- Greylisting
- DNS blackhole lists
- HELO enforcement

Greylisting has proven to be an excellent filter and I've deployed it successfully on several networks with nothing but positive feedback from their owners. Even the basic HELO filtering, though, can visibly decrease spam loads and should be completely safe. It can be difficult to find a good compromise between safety and effectiveness, but I believe I've found a solid combination that should work in almost any situation. Don't be afraid to test these ideas on your own and make them a part of your own anti-spam system!

Bibliography

[Postfix Configuration—UCE Controls](#)

[SPF: Sender Policy Framework](#)

[Greylisting.org—a great weapon against spammers](#)

[Postgrey—Postfix Greylisting Policy Server](#)

[AMaViS—A Mail Virus Scanner](#)

[The Apache SpamAssassin Project](#)

[Clam AntiVirus](#)

Biography

[Kirk Strauser](#) (/user/25" title="View user profile.): [Kirk Strauser](mailto:kirk@strauser.com) (mailto:kirk@strauser.com) has a BSc in Computer Science from Missouri State University. He works as a network application developer for The Day Companies, and runs a small [consulting firm](http://www.strausergroup.com/) (http://www.strausergroup.com/) that specializes in network monitoring and email filtering for a wide array of clients. He has released several programs under free software licenses, and is active on several free software support mailing lists and community websites.

Copyright information

This article is made available under the "Attribution-Sharealike" Creative Commons License 2.5 available from <http://creativecommons.org/licenses/by-sa/2.5/>.

Source URL:

http://www.freesoftwaremagazine.com/articles/focus_spam_postfix
