Gelato@UNSW

Performance and Scalability on Itanium

www.gelato.unsw.edu.au

I Hate Spam

Like everyone else, I get a lot of unwanted email sent to me. The problem used to be more-or-less manageable — maybe ten or twenty a day — but matters started to get out of hand, one day back in 2004.

My habit had been to decode the `Received:` lines, and report spammers to their ISP. I managed to get a few spammers shut down, and was feeling pretty pleased with myself.

That's when we started to get bounce messages. Thousands and thousands of bounce messages. *Millions* of bounce messages every hour! Enough to make the University's CSE mail servers glacially slow.

One of the ISPs had forwarded my entire complaint email to the spammer, who had arranged for several batches of spam to be sent out using addresses in my domain as the `From:` and `Sender:` addresses.

I Hate Spam!

and what I've done about it

Peter Chubb

Gelato Project

National ICT Australia

and

The University of New South Wales

January 2009

Lessons

- Don’t report spammers directly
  - You may get DOSed
- Don’t generate bounces
  - you may be DOSing some other poor sysadmin’s system

I’m not going to cover DNS blacklists, greylisting or how to set up spamassassin — there are plenty of HOWTOs on the web.

SpamCop

- You report spam to SpamCop
  - `www.spamcop.net`
- It automatically parses the `Received:` lines
  - It can tell forgeries faster than I can
- And generates munged spam reports that hide your address.
- Also keeps a DNS blacklist based on spam reports received.

SpamCop is pretty neat. It maintains a database of spammers; it also maintains a database of spam reporting addresses. It can parse a set of `received:` headers faster than I can, and generate an email to the appropriate complaints address. And you can query its spammers DB as a DNS blacklist.

References

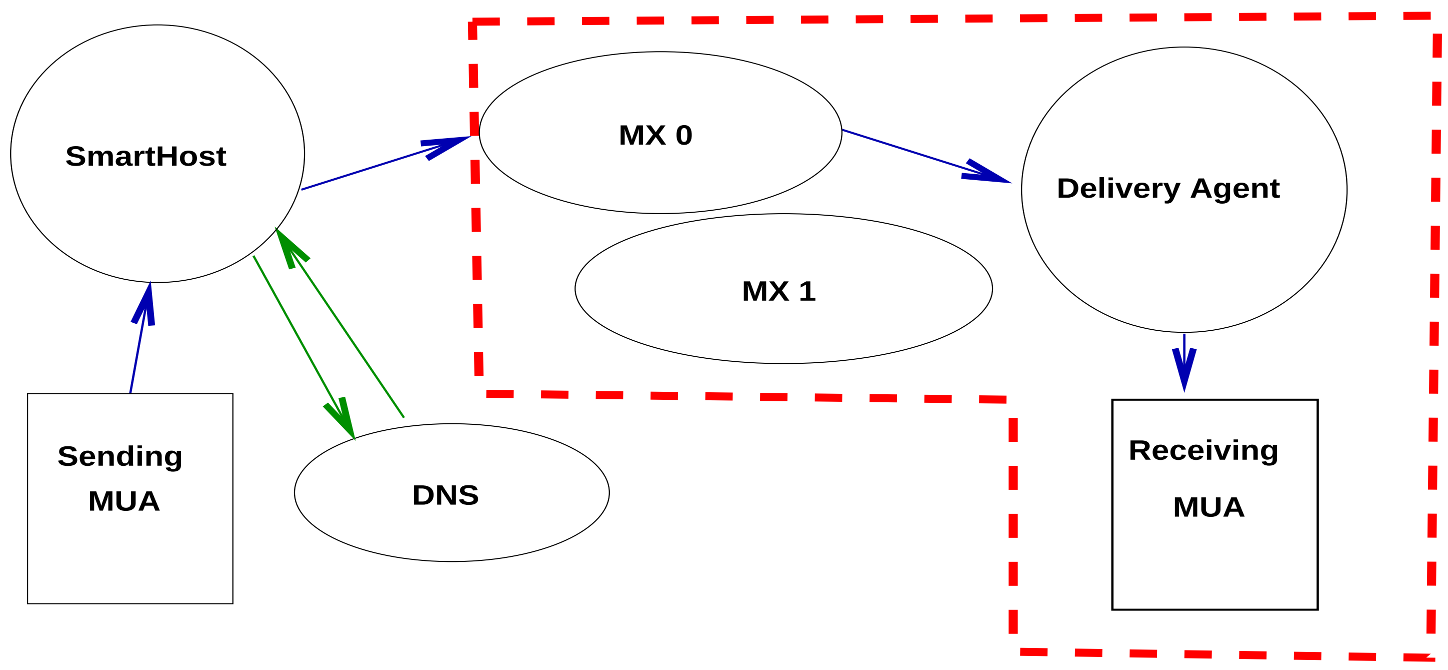
- RFC5321 — main RFC for SMTP
- RFC5322 — description of standard header fields
- `http://www.sput.nl/spam/` – good place for Exim ACLs

When I get carried away, I start just talking about RFC numbers. The ones you need to know for this talk are RFC5321, ‘Simple Mail Transfer Protocol’ and RFC5322, ‘Internet Message Format’.

These superseded RFC2821 and 2822 which in turn superseded 821 and 822.

An RFC is a ‘Request for Comment’, circulated within the community as a specification of how to do things. Many RFCs, including the ones for email, are effectively standards.

How SMTP Works – I



When you hit `send` after you've typed in a new email, your Mail User Agent (MUA) either talks directly to an outgoing email gateway (a *smarthost*) or invokes a program such as `/usr/lib/sendmail` to do that for you. (It could also bypass the smarthost and talk directly to the destination, but this is in general a *bad idea*).

The result is an SMTP conversation with a smarthost.

The smarthost then works out how to deliver the email. It may validate the sender (using TLS or source IP), to work out whether relaying is allowed. If it is authorised to do so, it will then look at the recipient address, and do a DNS query to find out how to deliver it. It tries to look up MX (Mail Exchanger) records. Each such record has a *priority*; the MTA must choose the lowest numbered MX record. There can be several MX records with the same priority (in which case the MTA is required to select one at random). If the lowest numbered MX cannot be contacted, then the next highest number is chosen. If there is no MX record, then an A record is looked up.

Some spammers always choose the highest numbered MX, which suggests a technique for blocking them: set up a couple of very high numbered MX records (priority 100 or more) that lead to a black hole spam honeypot.

The smarthost then has an SMTP conversation with the receiving gateway, which then hands off the email to a delivery agent — which may be another mail relay on another machine, behind a firewall, or could be mailman list, or could be a file in a spool directory somewhere.

Finally the user's MUA can collect the email (by grabbing it from the spool file, or via IMAP or POP).

If you're running a spam detector, it's usually run just before delivery. This is *too late* ... your machine has done an awful lot before then.

You *can* run spamassassin at SMTP conversation time. But in general, spamassassin is too big and hungry: it can grow to use all of memory (and then some); and the kernel's OOM killer is likely to choose, say, Apache as the memory hog by mistake. So you need to reduce the amount of potential spam before running spamassassin.

We want to stop spam at the boundary of the area we control — the bright red dotted part.

How SMTP works – II

```
220 lemon.gelato.unsw.edu.au ESMTP ready to rumble.
HELO joe.com
250 lemon.gelato.unsw.edu.au HELO joe.com [1.2.3.4]
MAIL FROM: <fred@joe.com>
250 OK
rcpt to:  <freddy@gelato.unsw.edu.au>
250 Accepted
data
354 Enter message, ending with "." on a line by itself
```

The SMTP (simple mail transfer protocol) is described in RFC 5321. Important features are that in its simplest form, it consists of a plain text conversation, where each part of the conversation is acknowledged separately by the server.

Some things to note:

- The 'HELO' name should be a valid domain name associated in some way with the IP address that the client is using. It MUST be a valid Fully-qualified domain name — not a CNAME (RFC5321, section 3.6).
- The envelope sender (`mail from:` argument) is the address to which bounce messages may be sent.
- Once the server has accepted a message and replied `250 OK`,

it must either deliver it, or if it cannot be delivered, send a message with empty sender field to the envelope sender.

- Although the example shows a standard SMTP conversation, there is also an extended one, introduced with `EHLO` instead of `HELO`.

The envelope sender and receiver are *not* the ones you see in your MUA. The addresses in `To:`, `From:` and `CC:` headers are defined in RFC5322, and will usually have some relationship with the envelope the first time an email is sent. But `BCC:` lists, and bounced messages, mean that for relayed email they will not necessarily be the same.

Reply codes 2xx are successes. Sometimes, the server can reply with 3xx (more information needed), 4xx (delivery deferred, try again later) or 5xx (permanent negative completion ... do not retry).

How SMTP works – II

```
220 lemon.gelato.unsw.edu.au ESMTP ready to rumble.
EHLO joe.com
250 lemon.gelato.unsw.edu.au Hello joe.com [1.2.3.4]
250-SIZE 52428800
250-8BITMIME
250-STARTTLS
250-PIPELINING
250 HELP
```

If you respond EHLO to the 220 message, the server will give you a list of options. The most interesting one from an anti-spam POV is PIPELINING ... which, if turned on, allows multiple requests to be queued up without waiting for server responses. Spammers *love* this — they want to jam out as much as possible, then get off the line. So if you turn it off, it

1. slows the spammers down, and
2. for spam software that doesn't check that the request to turn on pipelining failed (and there seems to be a lot of that about), you can terminate the conversation early.

Also note the STARTTLS line. This allows encrypted conversations, and also can allow authentication. In general, you should relay only for authenticated clients.

Observations

- Spammers don't care about RFCs but neither do some ISPs
- Enforcing strict conformance can help a lot.
  - Whitelist broken ISPs

A simple observation is that most spammers don't go through their ISP's smarthost. If they did, they'd get shut down pretty quickly. Instead, they tend to use ratware to take over other people's machines to send their spam, or use ISPs that allow direct access to port 25. As such, they tend not to be running full-blown MTAs that obey all the RFCs. Likewise, some of the things they do to hide themselves bend the RFC.

Some more Observations

- RFC 2821: accepted email must be delivered, or a delivery failure notice sent.
- Spammers forge envelope senders
- Don’t want to DOS some other poor sysadmin
- reject all email that you’re not going to deliver

We’re going to try to obey the RFCs, especially RFC2821. To give the same strong guarantee that RFC5321 gives, while trying not to send unwelcome could-not-deliver reports, we *must not accept mails we’re not intending to deliver*. This implies detecting spam and spammers early on. At SMTP conversation time, you have two choices: reject the email, or accept it. And if you accept it, you must either deliver it, or return a bounce.

First Steps

- Gain control of your email
  - you choose what to accept
- blackhole MX records
  - Some spammer software always sends to high MX

First step is to gain control of your email. Otherwise, you end up running a spam checker after the best time to reject the email is past; your ISP's mail gateway may not have the same rules you do. For my home domain, I hire a virtual host for around $19 a month; you can do that, or find some other solution.

Then you can put in a few fake MX records, with very high priorities. Some spam software always sends to the highest MX or next-to-highest MX, rather than the lowest priority one. Anything sent to such an MX (assuming that your lower numbered MXes are always available) is therefore spam, and the sender can be blacklisted.

Strict RFC compliance

- `HELO`
- Don’t advertise Pipelining
- Authorised users

There are a number of checks possible on the `HELO` line. However, we don’t finish the conversation straight away; rather dump the result in an acl for checking later — it could be, for example, me trying to connect from my laptop in an hotel in Beijing. Rather, delay until after we can check authentication, and allow all authenticated connexions.

Spamware tends to ignore the fact that pipelining isn’t advertised, and tries to pipeline anyway. Simply turning it off loses 10% of the spam.

- PTR record
- Warn, not deny

I used to reject emails from senders without a PTR record, or where the PTR record wasn’t a plausible match. Too many legitimate senders were blocked by this, so I don’t do it anymore. It tends to catch people sending from dynamic DNS locations, or from wireless cafés. However, it’s worth adding as a message header, so that a Bayesian spam filter can match on it. The sender IP may be lost by the time spamassassin runs.

- Verifying Sender
  - Check MX or A record
  - verify sender
  - verify random
  - Don’t upset innocent third parties!

Remember, we want to be able to deliver a bounce message if the mail delivery fails after we accept it; if we can’t deliver such a message, then we’re not going to accept the email. The simplest check is whether the sender envelope address has an A or MX record. If it doesn’t, reject the email.

You can go further, and do a *callout*. This is the first part of an SMTP conversation, to see if the MTA will accept an email for the envelope sender. If it doesn’t, reject the email.

Some MTAs will accept *anything*. You can detect these by attempting an email to a randomly generated address.

There are two reasons *not* to do this. For a start, many spammers forge the sender envelope address. So by doing a callout, you’re participating in a DDOS. For a second, *this breaks some (broken) cgi-generated email from major companies*.

- Verifying Sender

```
callout_positive_expire = 7d
callout_domain_positive_expire = 7d
callout_negative_expire = 2h
```

If you absolutely have to do this, use generous cache timeouts.

- Verifying Sender

```
# Avoid backscatter: don’t do sender callout verificati
#  deny
#    hosts = !+relay_from_hosts
#    !acl = acl_whitelist_local_deny
#    senders = ${if exists{CONFDIR/local_sender_callout
#                            {CONFDIR/local_sender_callout
#                      {}}
#    !verify = sender/callout=120s,defer_ok
#
#

 deny
```

```
hosts = !+relay_from_hosts
message = Sender verification failed
!acl = acl_whitelist_local_deny
!verify = sender
```

And here’s how to do it. The `defer_ok` means that you don’t get into a greylisting war: if the sender’s MTA says ‘defer’ or is down, you treat it as OK.

- Accept only verified recipient addresses. That may mean a callout!!!

```
accept domains = +relay_to_domains
 endpass
 message = "Unknown User"
 verify = recipient/callout=30s,use_sender,defer_ok
```

In my case, the receiving MTA is inside the firewall, the MX is outside. So the MX has no list of valid users. It does a callout to the inside to see if the email can be delivered. Once again, `defer_ok` means that if the final MTA is overloaded, or otherwise temporarily not accepting the email, you treat it as OK.

Because the destination may have blacklists as well, we reuse the original sender address.

Now we start to get hairier.

- RFC5321 says don’t reject on faulty RFC5322 headers.
- We want to reject bogus bounces.

One major cause of problem to us is the backscatter from spammers using our domain. We want to reject emails that appear to be bounces, that were never ours. There are a number of ways to do this; the simplest is to scan for a `Received:` line in the body of the text that says this was ours. Strictly speaking, this is a violation of RFC2821.

```
# Block fake bounces on ip address
 deny hosts     = !+relay_from_hosts : !+spf_white_hosts
      senders   = :
      message   = This is a fake (joe job) or sub standa
      condition = \
      ${if !match{${lc:$message_body}}\
      {\N( |ˆ)received: from .+\.gelato\.unsw\.id\.au.+2
      {yes}{no}}
```

DSN is Denial of Service Notification. SPF is Sender Policy Framework.

Other things to think about

(see `http://www.sput.nl/spam/`)

- Headers should be US ASCII
  - I'm now (reluctantly) accepting UTF8 as well
- ALWAYS accept email to `postmaster`
  - and `abuse` if you want to be up-to-date.
- Greylisting can help
- Don't filter your secondary MXs
- Feed to spamassassin *after* rejecting 99% of the messages

The RFC says that all the headers should be US ASCII. I'm now reluctantly accepting UTF8 as well, because of so many correspondents with non-compliant MTAs.

You *must* accept email to postmaster. If you do not, you'll be added to the RFC-ignorant blacklist, and have difficulty getting email out.

Greylisting is the practice of deferring emails from unknown, untrusted senders, but remembering the message ID. When the sending MTA retries, it'll get through. It delays email, and can, if other MTAs use badly configured sender verification callouts, prevent email getting through at all. One variant on greylisting only defers emails after using Spamassassin to check if they're likely to be spam.

You do need to be careful not to delay or deny email from hosts you relay for, such as your secondary MXs.

Mailing Lists

- Don’t bounce from mailing lists
  - They tend to unsubscribe users
- Tag mailing list SPAM with spamassassin and leave it up to the users.

Likewise, it’s better not to reject email from mailing lists. Whitelist hosts such as `vger.kernel.org`, `yahoogroups.com` or `lists.herald.co.u` that support many mailing lists. Otherwise, spam that passes the mailing list host’s fitlers will cause messages to be rejected, which will result in list unsubscriptions, and your users on your back.

Situation now

TOTAL	Volume	Messages	Delayed	Failed
Received	2065KB	265	0.8%	62.3%
Delivered	2038KB	272		
Rejects		2063		
Temp Rejects		1		
Ham		259		
Spam		71		

This is the summary from a four-hour slice late last year. The spamassassin and Exim logs aren't fully aligned, which is why the numbers don't add up. There are more emails delivered than received; this is because we run mailman lists that explode messages; and also some emails (from root) are generated locally.

Of the 2330 messages received, we ran spamassassin on only 265 of them — around 11%. And we rejected 2063 messages that were almost certainly spam. There may have been one or two ham messages in there.

Getting your email through

- Use your ISP’s smarthost
- Or (preferably) your own smarthost on a static IP.
- Configure your outgoing MTA correctly.

If you want to get your email through to me, you need to use a properly configured outgoing MTA that is not listed in any DNS blacklist.