SpamTrapBackend

Spam Trap Backend Details

Chris Hastie asked, on the mailing list: *Is there some nice efficient fast code out there for spamtraps?*

jmason's followup: Yep! 😉

First off, you're creating a new SpamAssassin object for each mail. creating that and parsing config is by far the most heavyweight part of processing.

Here's what to do. Install my own IPC::DirQueue perl module. Create a userid, "trapper". Set it up to deliver to /home/trapscripts/maildelivery.pl . In that script, do something like this:

```
my $queue_incoming = "/tmpfs/trapperqueue/incoming";
use IPC::DirQueue;
my $dq = IPC::DirQueue->new ({ dir => $queue_incoming });
while (<STDIN>) {
  $msg .= $_;
}
$dq->enqueue_string ($msg);
```

(I'm omitting basic stuff like skipping overlarge messages, error handling etc., just to keep these examples brief.) It's important to keep this script extremely fast and low-impact, since this is what the MTA runs.

Then, you have a "qproc-incoming" script which processes the "/tmpfs/trapperqueue/incoming" queue:

```
my $queue_incoming = "/tmpfs/trapperqueue/incoming";

use strict;
use IPC::DirQueue;
mkdir ($queue_incoming, 01777);
my $dq_incoming = IPC::DirQueue->new ({ ordered => 0,
                dir => $queue_incoming });

use lib '/home/trapscripts/trapsa/lib';
use Mail::SpamAssassin;
my $spamtest = new Mail::SpamAssassin(
  {
    rules_filename      => '/home/trapscripts/trapsa/rules',
    dont_copy_prefs     => 1
  }
);
$spamtest->init(1);

while (1) {
  my $job = $dq_incoming->wait_for_queued_job();
  eval {                    # catch die()s
    process_job ($job);
  };
  $@ and warn $@;           # warn about die()s
  $job->finish();
}
```

And then process_job takes the "job" file, reads it into a string, discards obvious bad stuff (viruses, bounces, etc.), then uses the $spamtest object to parse it using SA.

It can then do more stuff there and then, or alternatively distribute it on to further qprocs. (The further qprocs angle is useful when you're dealing with stuff that can seriously lag, like DCC reporting to remote DCC servers.)

This script can do heavyweight tasks, since only a limited, statically-sized pool of qproc daemons is allowed to run, therefore it's CPU/IO limited.

This is the backend for how the SpamAssassin traps have been running for a while. Using lightweight queueing, as provided by IPC::DirQueue, is essential – this way they can cope with an insane load, deal with some tasks as soon as the mail arrives, but simply queue stuff that they can't process immediately. Basically, it allows you to specify how much CPU/IO power to throw at the problem, and it's limited to that. Nowadays, of course, Amazon and Google have revealed that they use queueing infrastructure in their backends too, but I didn't know that when I wrote IPC::DirQueue 😉

Finally, the queues are cleared every night, so if insane load keeps up for more than 24 hours, it can't cause a serious backlog.