NewTrapsPlan

Here's the plan for hosting a replacement SpamAssassin spamtrap at EC2.

USE CASES

- Organisation A runs a set of spamtrap domains, and wants to be able to receive spam for those domains, and save that spam to collections which can be downloaded for later analysis (e.g. SpamAssassin rule generation and pattern analysis).
- Person B wants to be able to share traffic from his existing spamtraps. He wants to set up an /etc/aliases entry pointing to the new spamtraps. He doesn't need access to the end result.
- Person C just wants to redirect a frequently-spammed domain entirely to the spamtrap system. He doesn't want access to the end result either.
- Organisation D wants to share traffic from their existing spamtraps, and also wants access to download chunks of the combined results.
- Organisation E wants to redirect a frequently-spammed domain in its entirety, on the basis that multiple consumer organisations can access them (if they want).

A-C are the use cases catered for by the current spamtrap system. D and E are new, and would be useful in order to provide a common, shared "meeting place" of spamtrap data.

SMTP LISTENER

Each trap box runs an SMTP listener which delivers all incoming messages with size under a set size limit, to a Gearman queue. A set of Gearman worker processes then take messages from that queue, processing them as follows:

- discarding obvious viruses/malware
- discarding too-large messages (over 4MB in size?)
- removing forwarding headers, for spams forwarded from third-party addresses
- extracting attached message/rfc822 parts (for certain forwarders)
- discarding messages from now-unreliable addresses, identified using header regexps
- discarding frequently-appearing ham, identified using header regexps
- discarding bounces (scan with SpamAssassin for ANY_BOUNCE_MESSAGES)

COLLATION

Remaining messages are categorized into output sets, by searching for distinctive header regexps. For example, "private" and "less private" spamtrap addresses can be differentiated this way, with the "less private" shared further with other organisations, but the "private" set kept separate. Each message can be sent to *multiple* output sets.

Messages for each output set are collated into mbox files, gzipped, and uploaded as an S3 "object" (think file) to an S3 "bucket" (think directory). The current mbox is closed and a new mbox opened once the mbox reaches a certain size, or number of messages (let's see which works better).

It's possible to specify that an output set should produce more than one mbox per period; the number to produce is specified in the configuration, and messages will be distributed randomly between them. (This allows massive volumes to be spread out, so that every output mbox from a high-volume trap doesn't just wind up just containing 100 identical dictionary attack messages in each mbox.)

The messages in the mbox are stored with additional metadata regarding the connection, SMTP transaction etc., as header field(s) prepended to the message.

Once each mbox is uploaded, notification of that upload is sent to interested subscribers for that set (see below).

NAMING

Each bundle is named by date/time in ISO-8601 format ("2007-12-10T12:06:00", UTC timezone, with a pseudo-random unique string appended.

(Note that there's no need to break this into subdirectories – "directory listings" of S3 buckets use a prefix search – S3 has no real concept of "subdirectories". So it is possible for a client with sufficient privileges to "list" bundles that arrived during 12pm on 2007-12-10, for example, by searching with the prefix "2007-12-10T12". see http://docs.amazonwebservices.com/AmazonS3/2006-03-01/ListingKeys.html)

Here's a sample output file name:

setname/2007-12-10-T12:06:00.machinename.4wHkcYIyIeNMVh2H.mbox.gz

SCALING

I've seen typical upload speeds from EC2 to S3 of 5MB/sec, so I doubt upload speed will be a problem. I haven't seen scaling problems with large numbers of parallel uploads from multiple EC2 nodes to the same bucket. Bandwidth use between S3 and EC2 is free of charge. (In this kind of setup, this is the #1 selling point for EC2, since its per-month charges are higher than a generic dedicated server.)

However, CPU power may be the issue, since EC2 nodes are not very powerful. If a machine becomes too busy to complete a bundle's upload before the next N-minute trigger time, we let the upload complete and simply skip the second, or later, uploads entirely – there's always more spam.

Scaling horizontally with this setup is trivial – simply start more EC2 nodes with the appropriate AMI (disk image), and ensure they join the appropriate pool of instances to handle new spamtrap SMTP connections, at boot.

Distribution of load can be done using round-robin DNS records; if that doesn't spread the load sufficiently, we may need to set up a load-balancer. If that doesn't work, we can assign multiple Gearman workers to a single incoming SMTP listener and balance at that level.

PERMISSIONS

Output files are available over unauthenticated HTTP from Amazon S3. This allows low-overhead (and fast) access for consumers.

Directory listing operations on S3 require authentication, so the URLs of the trapped mails can be relatively secret with the addition of some randomness in their names (see NAMING above). Only receivers of the subscriptions (below), or authenticated S3 users, will gain access to those URLs.

SUBSCRIPTION

We need a way to notify a pool of "subscribers" each time a new bundle of messages is uploaded, so that they can find new data without having to repeatedly perform slow directory listing operations against S3.

I'd suggest the simplest is for the uploading EC2 nodes to send a mail to a mailing list once each upload completes, containing:

- the newly-uploaded object's URL
- its size
- possibly other metadata; number of messages, time period covered, etc.

in a machine-readable format – namely RFC-822 "header: value".

Each output set has its own mailing list, so that different subscribers can subscribe to different subsets of the trapped spam.