A Forensic Analysis of a Distributed Two-Stage Web-Based Spam Attack

Daniel V. Klein – LoneWolf Systems

ABSTRACT

Open mail relays have long been vilified as one of the key vectors for spam, and today – thanks to education and the blocking efforts of open relay databases (ORDBs) – relatively few open relays remain to serve spammers. Yet a critical and widespread vulnerability remains in an as-yet unaddressed arena: web-based email forms. This paper describes the effects of a distributed proxy attack on a vulnerable email form, and proposes easy-to-implement solutions to an endemic problem. Based on forensic evidence, we observed a well-designed and intelligently implemented spam network, consisting of large number of compromised intermediaries that receive instructions from an effectively untraceable source, and which attack vulnerable CGI forms. We also observe that although the problem can be easily mitigated, it will only get worse before it gets better: the vast majority of freely available email scripts all suffer from the same vulnerability; the load on most proxies is relatively very low and hard to detect; and many sites exploited by the compromised proxy machines may never notice that they have been attacked.

Introduction

As new defenses against spam are developed, spammers have been forced get more inventive in their attacks. Much as antibiotic resistant bacteria force the development of new drugs, so do spam-resistant mail servers drive the spammers to continually create new methods of disseminating their advertisements. A modern anti-spam arsenal provides defense with a combination of Bayesian filters, keyword matching, fuzzy checksums, header and source IP checks.

Spammers have responded with text that defeats filters, omits key words with unusual spelling or use of graphics, random strings to confound checksums, and more and more legitimate-looking headers. The one thing that can truly defeat spammer is to deny them the means to send their email. Since recent legislation has made it undesirable to have a traceable source IP address (since getting caught can now mean stiff fines and/or jail time[1]), spammers used open relays in foreign countries as an indirect (and difficult to trace) means of promulgating their wares.

Indeed, for a while it was thought that closing off open relays and/or denying them the ability to send mail might be a solution. Although it took time to spread, updates to the default configurations of mail transfer agents (MTAs) went a long way towards alleviating the open-relay problem, while education through a distributed denial of service (in the form of ORDBs) have made open relays, if not a thing of the past, at least a non-problem.

[1]On September 5, 2006 the Virginia Court of Appeals (Record No. 1054-05-4) upheld the felony conviction of Jeremy Jaynes under the state's anti-spamming law (18.2-152.3:1). The sentence (which was affirmed in the appeal) was nine years imprisonment.

However, open relays fall into two classes: MTAs and MUAs (mail user agents), and the ORDBs only deal with the former class. MUAs are often thought of as applications that send mail for an interactive user, but they can also be applications that send mail on behalf of a script. In this case, the application is a CGI script, and an improperly configured CGI script can be exploited to send a payload to locations that were not envisioned by their designers.

Vulnerability Summary

The vulnerability in CGI scripts that send email is easy to spot, and trivial to fix. It may best be summarized in the following highly simplified example (see Example 1).

At first blush, the form and the CGI script look innocuous enough. A surfer can enter a single line of text for their name, another for their email address, and a multi-line complaint. The script then sends an email from the user to a fixed address (in this case, mine), and lets me know their complaint. Compliant browsers all provide this functionality, but the problem is not in browsers, but rather in the fact that it does not take a browser to surf the web! Surely the notion of robots is no mystery (all of the search engines use them to harvest information from the web), so it should likewise be no surprise that a robot can submit forms. Combine this with a fundamental feature of the HTTP and SMTP protocols, and you have a trivial means for a spammer to abuse this simple form.

Although browsers only allow a single line of data to be supplied for an input item of type "text," the HTTP protocol allows arbitrary text, and robot can easily provide that arbitrary text. When connecting to Sendmail (and other MTAs), mail headers are separated

from the body of the message by a blank line. So if a legitimate surfer enters "Dan Klein" for the Who field, "dan@klein.com" for the Addr field, and "This form\nis really dumb!" for the Complaint field, then the following would be handed to Sendmail:

```
To: dvk@lonewolf.com
From: Dan Klein <dan@klein.com>
Subject: Whine whine whine

This form
is really dumb!
(whew)
```

Notice that the newline in the Complaint field is accurately translated when sending the email. So now consider that a spammer has instead supplied the value in Example 2 for the Addr field. Here is what is handed to Sendmail:

```
To: dvk@lonewolf.com
From: Dan Klein <dan@klein.com>
Bcc: your@address.com, target@victim.com,
   another@mark.com, more@suckers.com
Subject: Increase your pennies

Get bigger pennies! The 19th century
English pennies are HUGE! Impress
your woman!

>Subject: Whine whine whine

This form
is really dumb!
(whew)
```

Since the spammer cannot see the script (only its results), s/he must treat it as a black-box, making a number of attempts until the proper formatting is discovered. Notice the careful placement of the '>' character to guarantee (in this case) that Sendmail properly parses the input. The spammer typically creates a short-term-use email account and uses that address as a test target, trying a collection of well-known formats with specially coded messages with an annotation (representing the format type) included. Once one email is received at the test address, the spammer knows from the annotation the correct format to use.[2] Suddenly the innocuous email form is a vehicle for sending spam!

In retrospect, I should have noticed the problem with my form when the spammer first tested their script. I received a short collection of very similar spam messages from all over the world, but like most people, I simply deleted them. Initially, I received them because the spammer had not yet determined how to properly exploit my script (for example, they had appended the Bcc addresses in the Complaint field).

After the exploit was fine-tuned to match my script, I received approximately 5,000 spam emails (because in my script, I was listed as the To address). Each of the messages were also Bcc'd to 380-390 other addresses. Had I not detected the attack, this ultimately would have resulted in close to 2,000,000 spam messages being sent from my mail server. It is interesting to note that my desktop spam blocking software helpfully deleted most of the spam messages addressed to me. Thus I never knew I was being attacked – at least until I checked my various RRD logs, as shown below.

The Larger Problem

A very quick survey using Google reveals the severity of the problem. Searching for "email cgi script," "Perl CGI email" and "Perl email script" yielded the expected plethora of results, but a union of the top-10 of each search yielded 21 unique pages. Of these 21 scripts, one third suffered from the vulnerability described above:

- 7 scripts had obvious flaws that allowed spam email exploitation of the type seen in the example above.

[2] Some of the test emails also include the correct use of MIME headers and separators, some of which (through *multipart/alternative*) appear to be specifically calculated to hide the original legitimate contents of the form-generated email.

```
<FORM ACTION=mail.cgi>
Your name:
<INPUT TYPE=TEXT NAME=Who>
<BR>
Your email address:
<INPUT TYPE=TEXT NAME=Addr>
<BR>
Your problem:
<TEXTAREA NAME=Complaint>
Type your complaint here!
</TEXTAREA>
<INPUT TYPE=SUBMIT>
</FORM>
```

```
#!/usr/bin/perl
use CGI ':all';
import_names;
open MAIL "| sendmail -oi -t";
print MAIL <<"==END==";
To: dvk\@lonewolf.com
From: "$Q::Who" <$Q::Addr>
Subject: Whine whine whine...

$Q::Complaint
(whew)
.
==END==
close MAIL;
```

Example 1: CGI Form and CGI Script.

```
dan@klein.com>\nBcc: your@address.com, target@victim.com,\nanother@mark.com,more@
suckers.com\nSubject: Increase your pennies\n\nGet bigger pennies! The 19th century\n
English pennies are HUGE! Impress your woman!\n\n
```

Example 2: Nefarious text handed to the complaint field.

  - One of the flawed scripts was #1 on a particular Google results page.
  - Another of these scripts "sanitized" the user data by removing brackets and shell wildcard characters from mail headers, but failed to remove line breaks!
  - A third script removed all instances of "\r\n" followed by removing all "\n" characters. This allows a spammer to use "\n\r" as a line break, leaving a bare "\r" after the so-called sanitization is performed.
- 3 scripts had code to successfully prevent spam email exploitation.
- 2 were at perl.com (and thus were assumed by this author to be correct).
- The remainder of the links were located on pay-sites, were too convoluted to search, or had bad pointers to scripts.

Matt's Script Archive (a tremendously popular website for perl scripts) provided a secure script, but looking at his change log, I found that the script was only secured in August of 2001. Although he urges current users to upgrade to the secured version, the site also has the following note: "This script has been downloaded over 2 million times since 1997." I noted that other publicly available scripts were based on the buggy versions of Matt's script, and hesitate to guess how many of the buggy derivative scripts are still in use throughout the world.

At least half of the most likely-to-be-copied scripts were sufficiently flawed as to allow easy exploitation by a spammer. A random sampling of other email scripts listed later in Google's results indicated that on the order of half the scripts were equally vulnerable.

Solution

Fortunately, there is a trivial fix to the problem. An obvious (yet regrettably often overlooked) rule of thumb is "never trust your users" – especially when those users are unvetted and/or from the web. A simple rule that the script could follow is: "don't trust user input," and simply sanitize the user data to accomplish this. Not allowing any user data in email headers is the best solution – fixed header-content guarantees a known result (using Sendmail's `-oi` switch is also highly recommended, or whatever the equivalent is in other MTAs). However, since it is also convenient to be able to reply to emails from forms (and thus allowing the user to specify input that will ultimately be placed in email headers), a less draconian solution is to simply remove newlines (or more generally, replace all contiguous whitespace with a single space character) in any user input that may find its way into an email header. Either action will fix the vulnerability and allow a user to *Reply* to form-based email.

There is no need to go to the extra effort of determining if the user has supplied a valid email address.[3] Sendmail will simply fail to deliver mail with bogus `From`, `To`, `Cc`, or `Bcc` headers, so in general it is better to leave the address parsing to the program most responsible for it.

Discovering the Attack

In addition to reading my email, one of my morning rituals is to scan the SNMP statistics from the various machines on my network. A single click in my tabbed web browser brings all this data readily to hand.[4] These include critical statistics such as network load, free disk space, web server statistics and email traffic, as well as more incidental data including load average, number of active users and processes, machine uptime, and NTP synchronization data. Any anomalies can thus be quickly investigated, and problems can be readily averted or remedied.

My network consists of my laptops and desktops, a web server, an email server, a backup server, a pair of name servers, and a semi-public wireless network. Figure 1 shows the overall traffic on my T1 serial line. The pattern of activity is not particularly noteworthy, and can readily be accounted for by a newly popular page on one of my websites, or one of my wireless customers uploading a particularly large file.

Figure 2 (available to me on the same dashboard) shows network traffic broken down by LAN. Although a second look showed me that the primary load was on the downstairs network (where the mail and DNS servers are located), initially this graph did not look at all ominous. The spikes at 03:50 and 04:20 are normal for a Saturday morning – they result from an upstairs machine performing a remote disk-to-disk backup to the downstairs backup server.

Figure 3 is a more detailed view of the downstairs network, and shows that the backup server (called `cow`) receives backup data from upstairs, and additionally gets backup data from `maxwell` at 03:30 and `samantha` at 05:30. But more ominous (again, only in retrospect) is the outbound data from `maxwell` (which is also my mail server) starting at 06:00, and the same time, the relatively large amount of output from ns. Typically, the name server accounts for a negligible load, and is all but imperceptible in the graphs.

Indeed, except for data recognized in hindsight, there was nothing to indicate that anything was amiss. The network traffic was cursorily "normal" (although the traffic from the name server was "unusual"), and

[3] A myriad of email form scripts get this wrong by forbidding '+' or other legal address characters. For a complete and correct regular expression for parsing email addresses, consult Jeffrey Friedl's excellent book "Mastering Regular Expressions" (ISBN 0-596-00289-0), or look at http://examples.oreilly.com/regex/email-opt.pl

[4] I use Cricket (http://cricket.sourceforge.net/) to collect the data, RRDTool to store it, and drraw (http://web.taranis.org/drraw/) to display it in "dashboards." Drraw is indispensable, because it allows unrelated data to be collected in a single view.

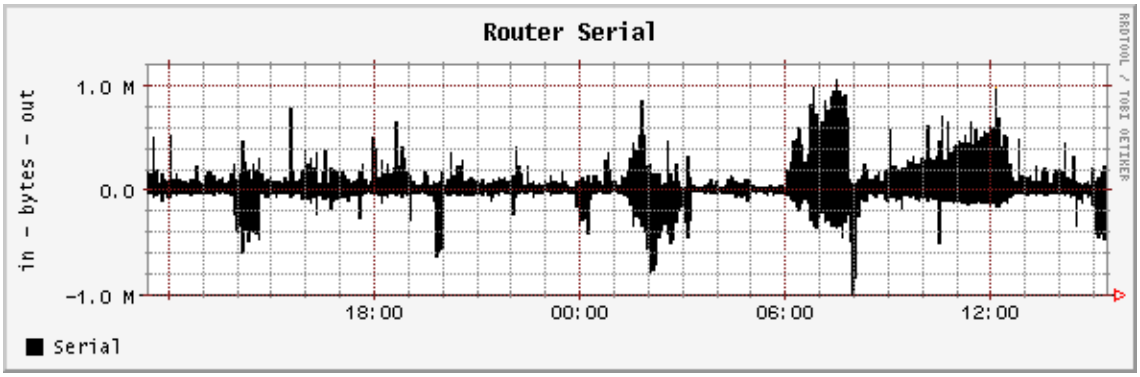

Figure 1: Overall network traffic.

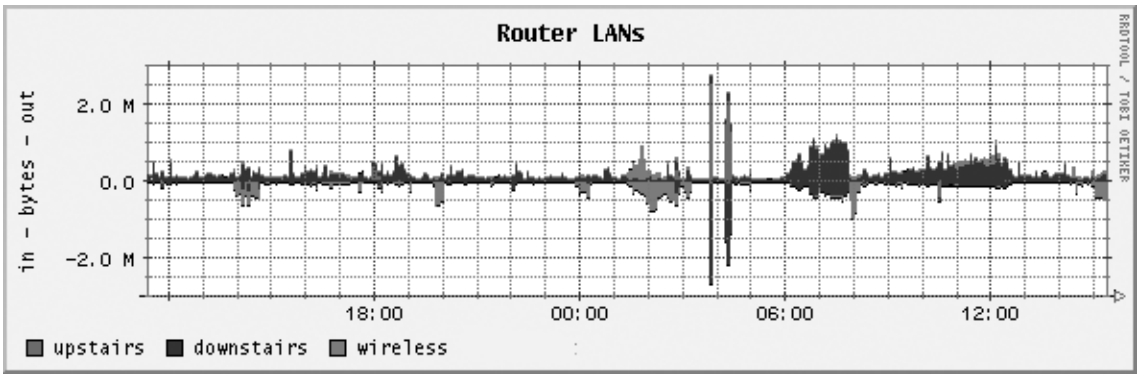

Figure 2: Internal network traffic by LAN.

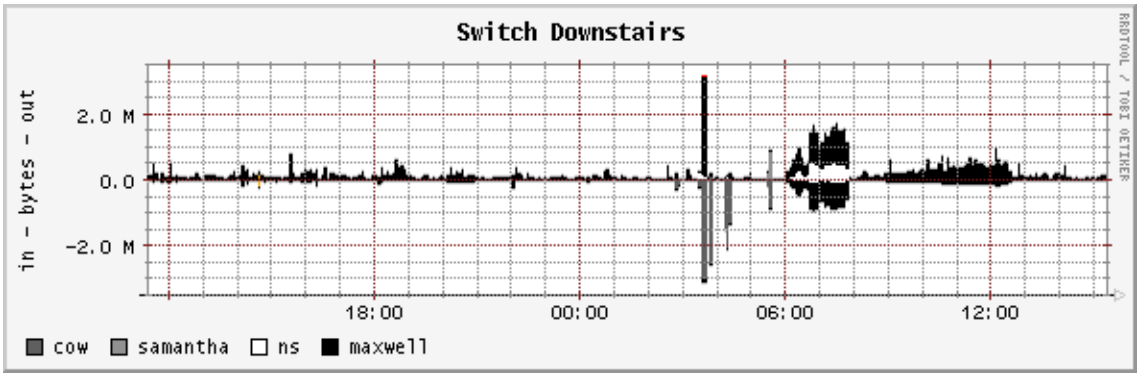

Figure 3: Internal network traffic on the downstairs LAN.

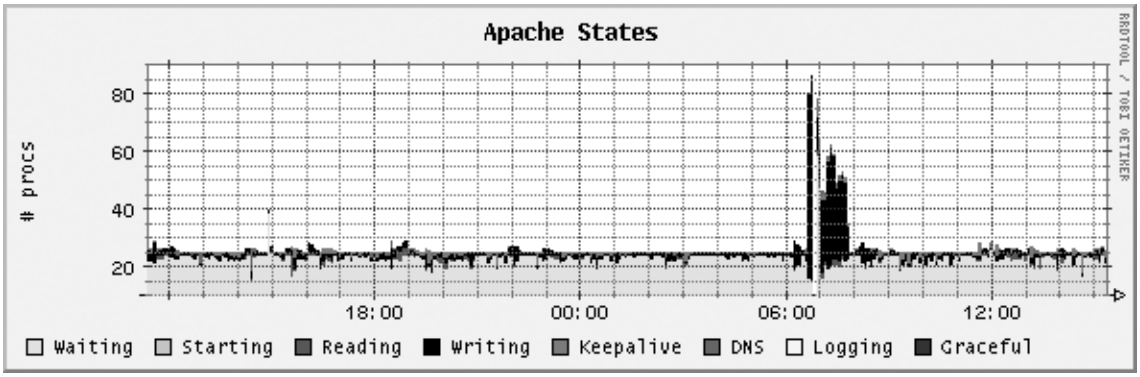

Figure 4: Apache States.

if network traffic had been my only critical indicator, a quick visual survey would have raised no red flags.

I had received a number of odd mail messages from the script that was being attacked, but I wrote those off as "someone messing with the site" (and was unconcerned given that I thought my scripts were secure). I also received one email message for each of the actual attack messages (since the script was *designed* to send me that mail). However, after the first few, my desktop spam filters "helpfully" blocked those messages, and except for the SNMP data, I was otherwise unaware of the thousands of spam messages being sent to AOL from my machine.

What ultimately triggered further analysis was the graph of Apache states, found in Figure 4 (however, this is only after first noticing the highly unusual mail statistics in Figures 10 and 14, below). Mine is a lightly loaded web server, with MinSpareServers and MaxSpareServers set to 15 and 25, respectively. Typically fewer than a half-dozen servers are active serving requests. Thus when Apache reported 40 to 80 active processes for a span exceeding an hour, I determined that something was amiss.

It is especially noteworthy that there was no surge in the requests made of the webserver, nor in the amount of network traffic that that the webserver was generating (seen in Figures 5 and 6). Thus each writing process (shown in Figure 4) was taking a lot of time to perform its job, but did so without generating a lot of data. Since Apache does not block on locked files, the only reasonable explanation was CGI script execution, where each script was taking a long time to complete.

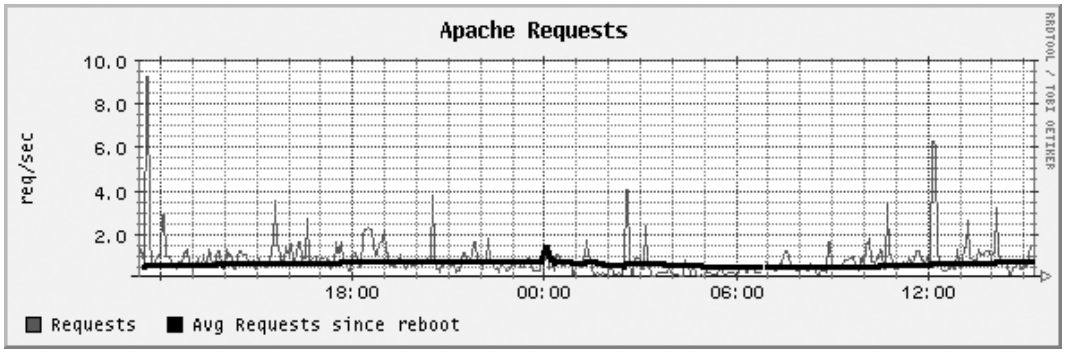

Figure 5: Apache requests from the web.

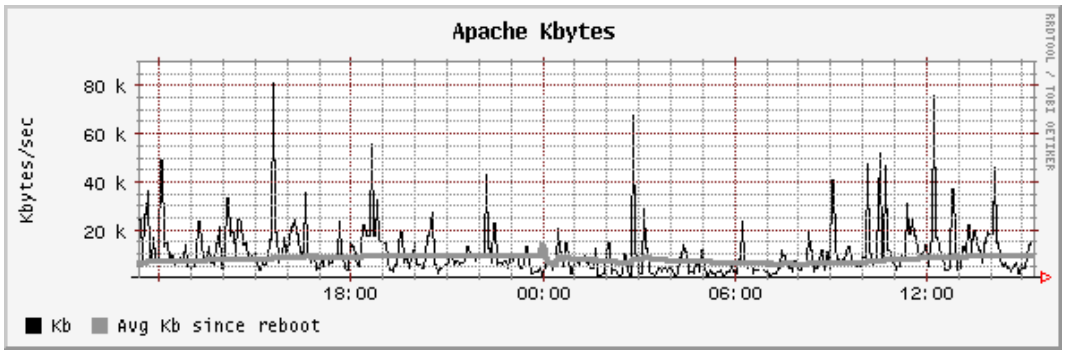

Figure 6: Kbytes served to the web by Apache web server.

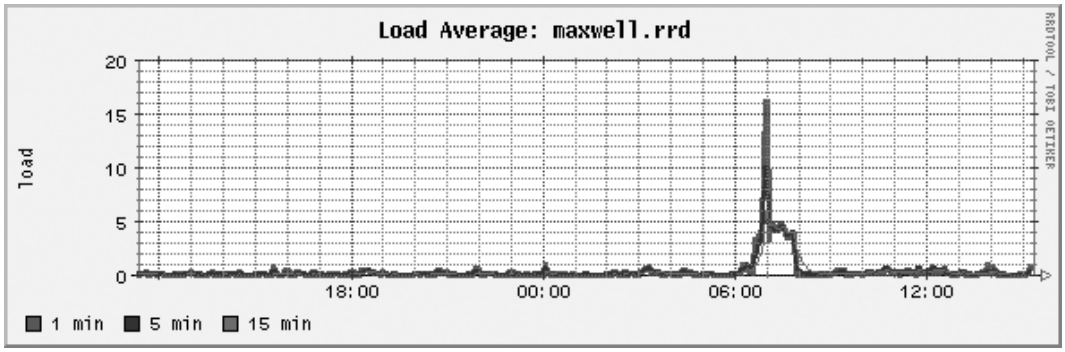

Figure 7: Load Average of Mail Server.

Indeed, as Figures 7 and 8 show, there were a large number of active processes. The load average on the machine (which is typically below 1.0) surged to a high of 16 before Sendmail's RefuseLA safeties cut in, and the number of active processes nearly quadrupled from 200 to 700.

Figure 9 shows the load on the name server machine. Typically, it putters along with a load average of 0.1 or less, but the large volume of email consumed nearly all of the resources of this machine. This small machine has a single purpose, and under ordinary conditions adequately services 15 internal hosts (all of whose webservers are configured to do reverse lookups), and is the authoritative name server for 185 domains. Therefore the load on the name server presented by the spam attack is vastly in excess of any normal load experienced.

The volume of mail delivered (and deferred) by the attacked webserver is shown in Figure 10. From the start of the attack, the messages actually delivered (labeled "forwarded to a remote client") steadily climbs until it reaches a peak of nearly four messages per second. At approximately 08:00, AOL graylisted my server, the deliveries dropped, and the number of deferred deliveries grew steadily.

The peaks in deferred deliveries seen every 30 minutes are Sendmail re-running the queue (because the -q30m option was specified at startup). I estimate that between 15,000-20,000 pieces of spam mail were sent until AOL graylisted my server. As will be seen below,

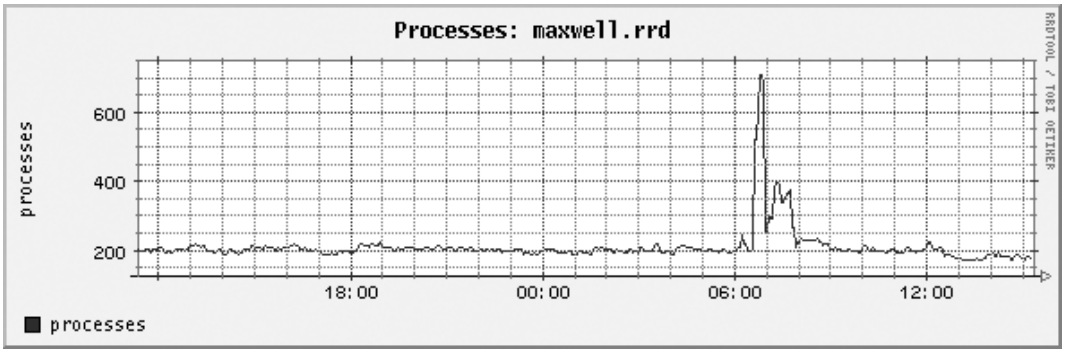

Figure 8: Number of Processes Active on Mail Server.

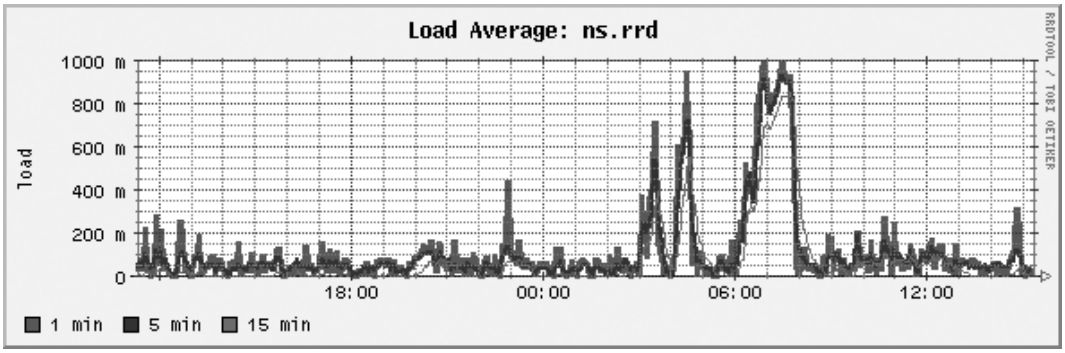

Figure 9: Load on Name Server.

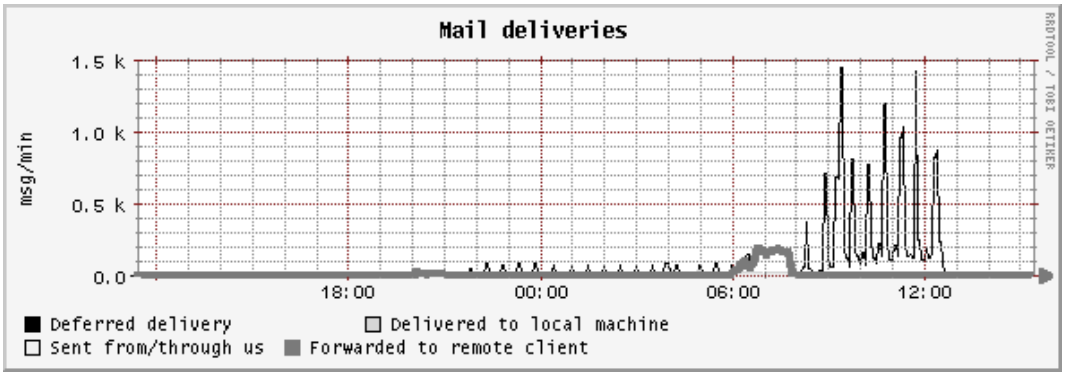

Figure 10: Mail Deliveries Made by Attacked Host.

this is a fraction of the potential impact on both my server and AOL. However, mine was but one of potentially hundreds of thousands of vulnerable email scripts.

Figures 11 and 12 show the number of message in the outbound mail queue, waiting to be sent. The number climbs steadily following the onset of the attack, and peaks at around 07:45, when the attack ceased (the precipitous drop off at 12:30 is due to my manually purging the queue). These numbers may appear small to someone who is used to running a larger site, but the sparseness of these numbers is misleading.

At its peak, the outbound mail queue contained 5,000 messages, with 80 MB of data. This may seem like a pittance, except when you consider that each message was typically addressed to 380-390 victims, so a queue size of 5,000 messages represents nearly *two million* email targets. For a medium to large host, this additional traffic might go completely unnoticed, and I posit that the problem of unprotected scripts is far larger than we suspect!

It is essential to note that no one bit of data would have triggered my awareness of this spam attack (and the search that followed). It is only in aggregate that the data indicated something was amiss. In fact at some sites, an increase in queue size such as this might only suggest a stuck queue, and merely prompt a restart of the mail server. Likewise, if only the queue volume was used as a metric, this graph could easily have been confused as a user emailing a large file (such as a movie or collection of photos) that was not being accepted by the remote site. Plus, 80 MB of outbound mail is a pittance for many ISPs.

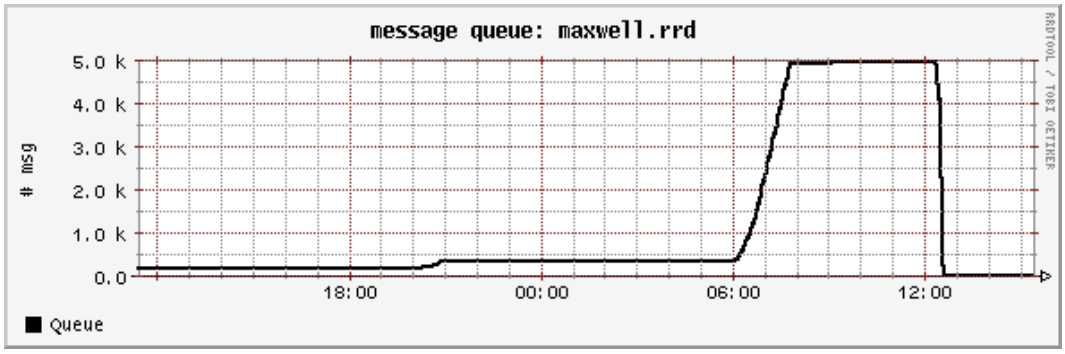

Figure 11: Number of messages in the outbound mail queue.

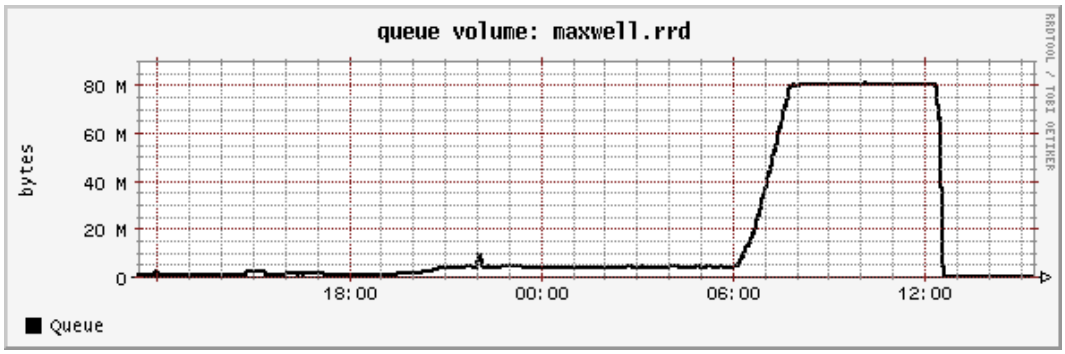

Figure 12: Volume of message in the outbound mail queue.

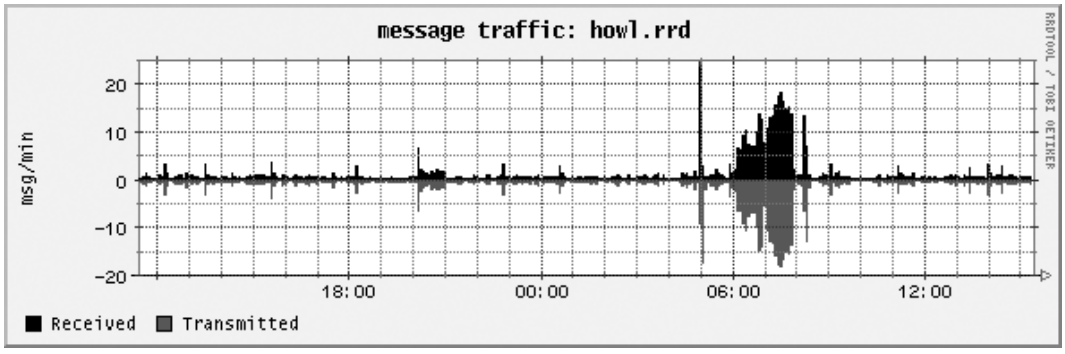

Figure 13: Mail traffic to my desktop.

Because my desktop spam filters deleted the large faction of confirmation and spam bounce emails addressed to me, the volume of these messages was not immediately obvious. However, as Figure 13 shows, the number of email messages that were received (and automatically deleted) is greatly in excess of any ‘‘normal’’ traffic that my desktop would otherwise see.

Yet what is perhaps most interesting is the spam statistics for the machine that was attacked. As you can see in Figure 14 (which tracks types of spam *received*), there is nothing to indicate that spam is being *sent*. During the attack, the spam ratios and rates look completely normal (the spike of ‘‘unknown users’’ at noon is a different, but unrelated attack).

However, Figure 15 is very interesting. This graph tracks the ratio of spam to ham (and also tracks that which is unknown). Surprisingly, the amount of ham goes up during the attack. This is for two reasons. The first is that the exploited script will still send an email to the originally intended email address within our site. This is definitely considered ham, because it originates *inside* our network and is targeted to another machine *inside* our network. The second is because the bounced emails (that fail to reach their intended target at AOL) are unknown. They cannot be classified as either spam or ham, so again the ratio of spam to ham goes down. It was ultimately this graph that caused me to look more closely at what was happening, since nothing (at the time) could immediately explain that erratic behavior.

Analysis of the Attack

The fact that I was attacked (and the solution to the vulnerability) was now obvious. The biggest problem I still had is that I didn’t know exactly *how* I was attacked. I know the reason (sending spam email), the target (a few million email addresses), the vector (my faulty CGI script), but not the source! In the 24-hour period with the most accesses of my vulnerable script, my web logs show a collection of 259 source-IP addresses scattered around the world.

Since web-servers report on the source-IP of the host that made the connection, my initial assumption was a collection of virus-infected Windows machines were attacking me, coordinated through some central database server. These compromised hosts could likely comprise a bot-net, but a problem exists with this hypothesis – a number of the machines were UNIX- or Linux-based machines! I attempted to contact a web server port 80 for each of the 259 machines. 27 of these identified themselves as UNIX/Linux Apache servers, 13 as Windows IIS servers, 6 as Squid proxies, and 6 as other web servers (the remaining 207 machines did not respond to queries on port 80).

I also considered that the attacker was using forged source IP addresses, but this possibility was quickly ruled out. Certainly, the SYN packets could

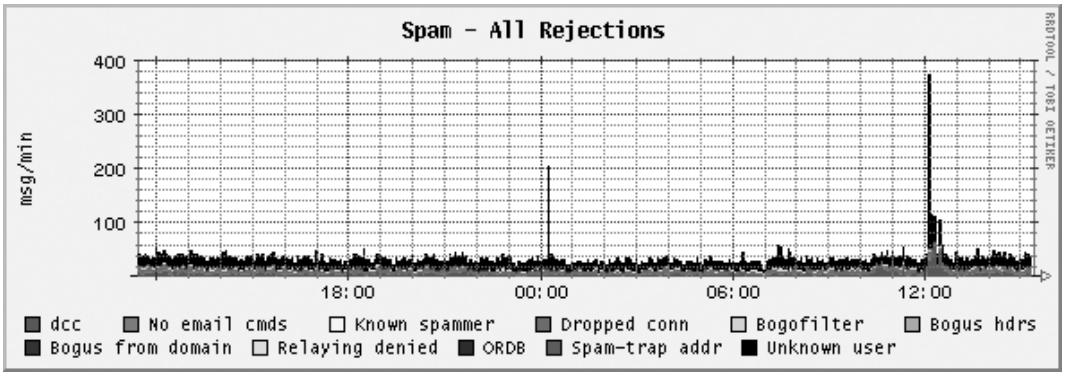

Figure 14: Spam detection.

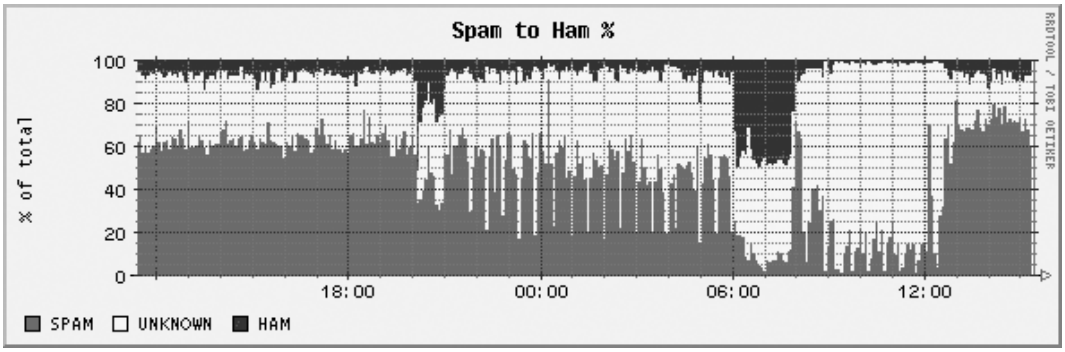

Figure 15: Spam to Ham Ratio.

have come from a forged address, but in order to make a request from my web server, the 3-way TCP handshake needs to be completed. The only way this could be done is for the attacker to be in the routing path of the forged address, and to make sure that the real machine does not respond to any response packets. While theoretically possible, this type of attack is highly unlikely, given the wide distribution of IP addresses.

My next thought was that these machines had open web proxies on them, and this seems the most likely explanation. Using *nmap,* I scanned each of the machines involved. Since my probes were done some months after the attack, it is very likely that some of the machines were either reprovisioned or reconfigured,[5] and approximately 45% of the machines probed were reported as ''down'' (some of these responded very erratically, with long connection delays that often timed out). Of the remainder I determined that a large number (approximately 50%) of the machines were currently running web servers or proxy servers on ''obvious'' ports (80, 3128, 800*x*, or 808*x*). Some of the proxy servers currently required authentication, and although I did not have the proper credentials, it is highly likely that the spammer did. Sniffing software is readily available to find proxy passwords on unencrypted networks. There are also hundreds websites listing ''free proxies,'' and hundreds of others that share passwords – the combination of services is equally likely. I also have no doubt that some of the machines were running proxy servers on unobvious ports as a result of a virus or trojan with an HTTP proxy payload

Finally, I considered compromised logins, especially for those machines running UNIX or Linux. I found that approximately 25% of the machines had open *ssh* or *telnet* ports, so this hypothesis is possible, but it seems much more likely that open proxies are the vector for the attack.

In general, the attacker was smart. A small number of test emails were sent, and once a successful exploitation was found, a relatively small number of spam messages were sent (only a couple of thousand), and then the spammer moved on, presumably to another victim.

Because the attack was distributed through a collection of proxies, it is very difficult to determine the source of the attack. To do so would require accessing the proxy logs from a collection of the participating machines (which would require international cooperation from people who may not even know that they have proxy running), and this effort might in turn discover that the participating proxies were contacted by *other* proxie to further obscure the attack. And although this might eventually lead to a single source ISP, all that we would discover would be a temporary AOL account (to verify that the test emails were received), used only for a day or two and probably created using a forged or stolen credit card.

Further, although the spammer used my script and MTA to send approximately two million spam emails, the volume was not so large that my MTA was swamped, nor would the load have persisted for more than a few days. For a larger ISP, the excess web and email load might even go completely unnoticed![6]

Once the script was used to send emails, renewed probes occurred daily for a few days as two to five tests were sent per host. Since I had by this time fixed the bug in the script, this probing gradually trailed off for the month after the initially discovered attack. However, it seems that the attackers are not fully coordinated, as my script is still probed a few times almost every day from locations around the world. I am ''on their list,'' and it seems that it will take quite some time before I am removed (if ever).

Lessons Learned

A number of lessons can be learned from this attack. Many of them are ''obvious,'' in that they are things that I should have learned and taken to heart ages ago. *Experience* is recognizing a mistake when you make it the second or third time, and I have plenty of that. *Security,* on the other hand, is not allowing oneself to make the same mistake the second time! I pass on my experience in the hope that my readers may have better security.

- Audit your scripts. It is far too easy to fall into the trap of thinking ''if it is available on the web, it must be good.'' The truth is of course ''not by a long shot,'' but internally developed scripts are equally vulnerable, and should also be audited. If you have a colleague who can audit your scripts, have them do so. But even a lay person can help you audit. I call this the ''mannequin theory of programming.'' If I take the time to explain my code to a total dummy, then I slow myself down enough to think about what might go wrong (and can then make notes to fix it).
- Notes are useless without a corresponding action. ''I'll get to it'' has severe consequences when you *don't* get to it! Once you find bugs, fix them!
- Finding a bug in one script probably means that the same bug exists in others. When you find a bug, exhaustively search your site(s) for scripts which have similar behavior to see if the mistake has been repeated.
- Audit your configurations. Most UNIX/Linux systems have Apache and Squid trivially available to them, and both can be (mis?)configured

[5] A webmaster that I contacted directly indicated that on the date of the attack he indeed had had a proxy running, but had since taken it offline due to ''configuration issues.''

[6] The open proxies are perhaps not as lucky. Some of the open proxies I examined had response delays of 45 seconds or more. While they may be intentionally throttled to prevent abuse, they are more likely seriously overloaded by that selfsame abuse.

as an open reverse proxy. While I will not engage in the privacy and anonymity arguments that proxy use can engender, I will only note that with every freedom there is a concomitant potential for abuse of that freedom. If you intentionally run a proxy, make sure it does what you intend it to do.

- Monitor your network. Daily dashboards are great for detecting anomalies, but beware of slowly changing things, too. A slowly filling disk (due to non-deletion of old log files) wasn't noticed for a year, because I only looked at daily, weekly, and monthly logs (and only the yearly log showed the slow trend)!
- Don't rely on automated detection. Tools like IDS systems are only as good as their processing rules (that in general rely on past experience), and they tend to discourage vigilance and encourage laziness. For example, would *cfengine* have found this attack? In a private communication, Mark Burgess (author and designer of *cfengine*) only says ''maybe.'' More accurately, he said:

  > The *cfenvd* daemon would certainly have noticed the rise in SMTP, DNS, processes etc., but the question is – would your monitoring policy have bothered to report it to you? In *cfengine,* you have to tell it what you want to hear about. The default is always – nothing!
  >
  > Cfengine does a kind of ''lazy evaluation'' on the data. If you ask to see anomalies in, say SMTP, you can also ask it look more closely at the distribution of IP sources. It uses the ''informational entropy'' (i.e., how sharp the distribution is) as a policy parameter. So you could say ''tell me only about low entropy SMTP anomalies'' (meaning tell me only about SMTP anomalies that come from one or proportionally few IP sources – a sharp attack from a single source). Or you could say ''show me all of them,'' or ''high entropy'' (wide range of sources). So as with all systems, it depends on how you had it configured.
- Know what you are looking at (and looking for). If you don't know your data, then it is that much harder to recognize an anomaly. In my case, a single anomalous graph would not have triggered my search. Instead, it was the correlation of a number of unusual bits that did so. Your data should provide specific information as well as the overall gestalt of your network.
- Know your systems and your configuration, and where possible, leave yourself clues. Troubleshooting is as much an art as it is a science, and the more you know about your environment, the more likely you are to inuit the source of a problem. In my case, I immediately recognized the 10-year old script I had written from the text of the email messages it was sending. However, even if I hadn't, I had unique text in every email (tagged to the script) that would have enabled me to quickly grep for the offending script. An even better clue would be to put both the URL and the location of the file on your server in the script output.

Author Biography

Daniel Klein has been teaching a wide variety of Unix-related subjects since 1984, and has been involved with Unix since 1976. His experience covers a broad range of disciplines, including the Internals of almost every Unix kernel released in the past 30 years, real-time process control, compilers and interpreters, medical diagnostic systems, system security and administration, web-related systems and servers, graphical user interface management systems, and a racetrack betting system. He contributes regularly to the proceedings of the USENIX Association, and is also their tutorial coordinator. He holds a Masters of Applied Mathematics from Carnegie-Mellon University in Pittsburgh, and in his free time is director of an a cappella group and a member of an improvisational comedy troupe.

Conclusion

Spam is a problem that is going to plague us as long as we provide a means for spammers to send their messages. Crime may not pay, but spam certainly does (it has been estimated that Jeremy Jaynes was earning $400,000 to $750,000 *per month*). While providing an open mail relay or an open web proxy may be viewed as an exercise in free speech, CGI scripts which allow email to be sent to any address other than the intended one can only be viewed as software with a bug. Fixing this particular bug is relatively easy, and has an immediate beneficial effect. While the overall impact of fixing this vulnerability on the volume of spam can only be speculated at, it is unquestionable that the volume *will* be reduced by doing so.