DAVID MALONE

unwanted HTTP: who has the time?

David is a system administrator at Trinity College, Dublin, a researcher in NUI Maynooth, and a committer on the FreeBSD project. He likes to express himself on technical matters, and so has a Ph.D. in mathematics and is the co-author of *IPv6 Network Administration* (O'Reilly, 2005).

dwmalone@maths.tcd.ie

THANKS
Ian Dowse and I conducted the original investigation. Colm MacCárthaigh provided us with the required Apache changes and Niall Murphy helped improve several drafts of this article.

IN OCTOBER 2002, ONE OF OUR users raised a req ticket with this message:

> When book number p859 is entered in the Web-based Library Catalogue search this error message comes up: Internal Server Error

At first glance, most experienced administrators would give a diagnosis of "broken CGI script"; however, as it would turn out, this was far from the case. Examination of the Web server's logs showed an unusually large number of HTTP requests for our home page, which was causing Apache to hit its per-user process limit. There was no obvious reason for a surge in interest in our Web pages, so we responded:

> It looks like our Web server may be under an attack of some sort! There are lots of people requesting our home page and the server is running out of processes for Webnobody! This isn't leaving enough space to complete the library lookup. I've no idea what is going on at the moment—between 18:00 and 18:59 we saw four times as many requests as we did this time yesterday.

The source of this traffic was far removed from the original error message and was eventually unearthed using a mix of investigation and guesswork. It's the investigation, ultimate causes, and our response to this traffic that I'm going to talk about in this article.

Analyzing the Problem

Our first thought was that our home page had been linked to from some popular Web site or referred to in some piece of spam. Our Apache server had originally been configured to log using the common log file format, so we naturally switched to the combined format, which also records the referring URL and the User-Agent as reported by the browser making the request [1].

To our amazement, the majority of the requests included neither of these pieces of information. However, this seemed to provide a way of easily identifying these unusual requests. Further study showed that the machines making these requests were connecting regularly, some as often as once a minute (to the nearest second), and they rarely requested any page bar our home page. Figure 1 shows the autocorrelation of the requests load, with strong peaks at multiples of 60s. For comparison, the autocorrelation without these unusual requests is also shown [2].

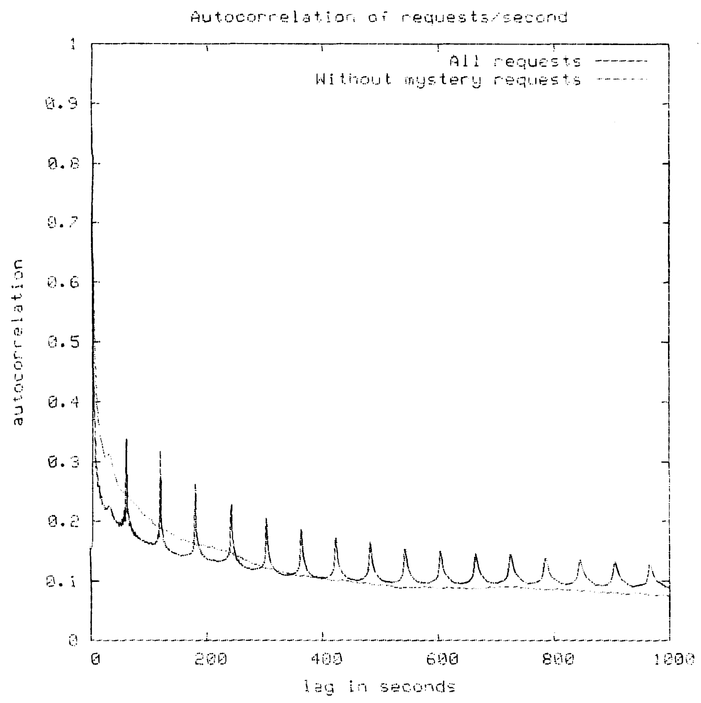

FIGURE 1. AUTOCORRELATION OF THE SERVER'S LOAD

Using *periodic requests for /* as a signature, we found that this had actually been going on for months. The number of hosts making these requests had been gradually increasing. This made it seem less likely that these requests were an orchestrated attack and more likely that it was some sort of quirk or misconfiguration.

Using tcpdump [3] we recorded a number of these requests. A normal TCP three-way handshake was followed by the short HTTP request shown below, contained in a single packet. Our server would reply with a normal HTTP response, headers, and then the content of the home page.

```
GET / HTTP/1.0
Pragma: no-cache
```

The request above is unusual. It does not include the Referer and User-Agent headers mentioned above. Also, it does not include the Host header, which is used to ensure the correct operation of HTTP virtual hosting. Even though the Host header is not part of HTTP 1.0, most HTTP 1.0 clients send a Host header. Virtual hosting is such a common technique that it seemed impossible that this could be any sort of normal Web client. In fact, the lack of a Host header indicated that the software making the request was probably not interested in the content returned.

Other than the content, the only other information returned by Apache was the HTTP headers of the response. These headers were the Date, the Server version, the Content-Type, and a header indicating that the connection would be closed after the page had been sent. After staring at all this for a bit, it occurred to us that something might be using our home page for setting the clocks on machines, as the Date header was the only part of the response that was changing.

We made connections back to several of the IP addresses in question and found that the connecting machines seemed to be Web proxies or gave signs of running Windows. We picked a random sample of 10 to 20 machines and made an effort to identify a contact email address. We sent short queries to these email addresses, but no responses were received.

A post [4] to the comp.protocols.time.ntp Usenet group was more productive. We asked for a list of software that might use the HTTP date header to set the time. This produced a list of possibilities, which we investigated.

Tardis [5] was identified as a likely source of the queries: it had an HTTP time synchronization mode and listed our Web server as a server for this mode. We contacted the support address for Tardis and asked why our server was listed, and why someone would implement an HTTP time synchronization mode when there were other, better protocols available.

Tardis support explained that they had a lot of requests to add a method of setting the time through a firewall, and thus Tardis added a feature using HTTP to do this. At the time they implemented this feature they scanned a list of public NTP servers [6] to find ones also running HTTP servers. The host running our Web server had been a public NTP server around 10 years previously, and due to a misunderstanding had not been removed from the list until mid-2000 [7].

The software only listed four servers for this mode of operation: a host in Texas A&M University, a host in Purdue University, Altavista's address within Digital, and our server. Tardis would initially choose a server at random and then stick with it until no response was forthcoming, when it would select a different server. We suspect that the spike in load that we saw corresponded to the HTTP server on one of these machines being unavailable, resulting in a redistribution of the clients of this machine between the remaining hosts.

Note that the default polling interval in Tardis was once every few hours. However, the software's graphical interface included a slider which allowed the poll interval to be reduced to once per minute.

Tackling the Problem

In the discussions that followed with Tardis support we agreed that future versions of Tardis would only list our official NTP server, and only for Tardis's NTP mode. We also suggested that allowing users to automatically set their clock once per minute was probably a bad idea and suggested modifications to the method used. In particular, using a HEAD rather than a GET request could significantly reduce the amount of data transferred, and setting the User-Agent field would make it easier for server administrators to identify such requests.

We did also suggest that our college be given a complimentary site license for Tardis in exchange for the not inconsiderable traffic shipped to Tardis users. For example, in the first 16 hours after enabling combined logging, we saw 400,000 connections from about 1800 different IP addresses. We estimated the resulting traffic at around 30GB/month.

However, it was some time before a new release of Tardis was planned, so we had to take some action to prevent future incidents of overload on our server. A first simple step was to increase process limits for the Web server, which had plenty of capacity to spare.

A second step was to use FreeBSD's accept filters [8]. Accept filters are a socket option that can be applied to a listening socket that delays the return of an accept system call until some condition is met. Usually the accept system call returns when the TCP three-way handshake is complete. We chose to apply a filter that delays the return of the accept system call until a full HTTP request is available. We knew that the request that Tardis was making arrived in a single packet one round-trip-time later. Thus the filter saves dedicating an Apache process to each request for the duration of the round trip (and avoids a small number of context switches).

While these measures helped prevent our server being overloaded, they did little to reduce the actual number of requests and volume of traffic being served to Tardis users. Using Apache's conditional rewriting rules, as shown below, we were able to match Tardis requests and, rather than returning the full home page (about 3KB), were able to return a much smaller page (about 300 bytes).

```
RewriteCond %{THE_REQUEST} ^GET\ /\ HTTP/1.[01]$
RewriteCond %{HTTP_USER_AGENT} ^$
RewriteCond %{HTTP_REFERER} ^$
RewriteRule ^/$ /Welcome.tardis.asis [L]
```

Using Apache's asis module [9], we were able to return custom headers, including a redirect to our real home page, in case some requests from genuine browsers were accidentally matched by the rewrite rules.

This significantly reduced the amount of data that we returned, but we also wanted to reduce the total number of clients that we were serving. We considered blacklisting the IP addresses of clients making these requests. However, we decided that this was not appropriate, for two reasons. First, a number of the client IPs were the addresses of large HTTP proxy servers and we did not want to exclude users of these proxies from accessing our Web pages. Second, the large number of IPs would make this a high-maintenance endeavor.

Instead, we decided to return a bogus HTTP date header in response to requests matching our Tardis rewrite rule, in the hope that this would encourage Tardis users to reconfigure their clients. By default Apache does not allow the overriding of the date header, but Colm MacCárthaigh of the Apache developer team provided us with a patch to do this. The page was altered to return a date of Fri, 31 Dec 1999 23:59:59 GMT. A link to another page explaining why we were returning an incorrect time was included in the body of this page.

We expected this to cause significant numbers of queries, and so prepared an FAQ entry for our request system to allow our administrators to respond quickly. However, we have only had to reply to a handful of email queries about this anomaly.

This countermeasure had a noticeable impact on the number of clients connecting to our server. Figure 2 shows the number of requests from Tardis users per hour, where we began returning a bogus time at hour number 1609. Alhough the number of requests is quite variable, our countermeasure quickly reduced the number by a factor of roughly five. Tardis support suggests that this is actually users reconfiguring Tardis, rather than some sanity check within Tardis itself. Note that the reduction achieved by this technique is actually more prominent than the impact of the new release of Tardis a year later.

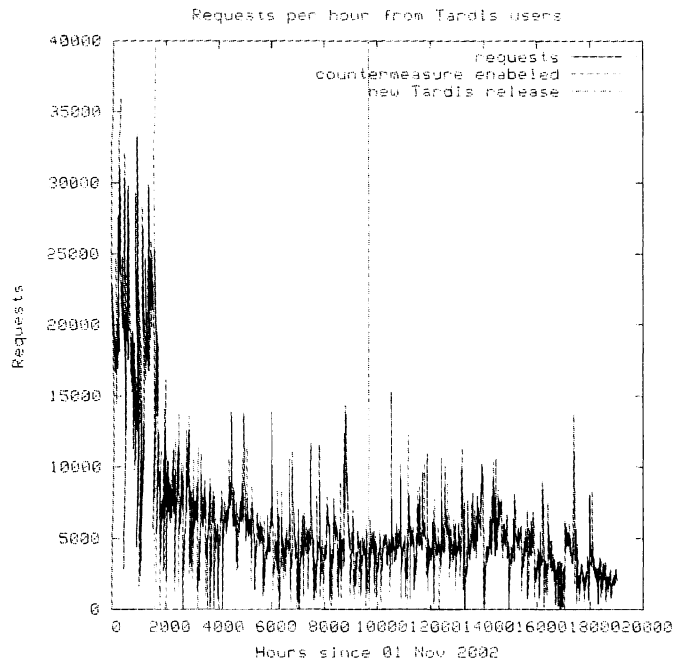

FIGURE 4. NUMBER OF REQUESTS FROM TARDIS USERS PER HOUR

Contemplations...

In dealing with this unwanted traffic, we were fortunate in several respects. Though there were only small hints as to the source of the traffic, they were sufficient to find the origin. The traffic also had a clearly identifiable signature, in that common headers were missing, and was not deliberately designed to be hard to identify. This is in stark contrast to spam, where successful attempts to identify features of spam quickly lead to a new generation of further obfuscated spam.

Though unwittingly inflicted upon us, this attack was quite like a DDoS. The number of hosts involved was in the thousands, making it infeasible to deal with each host by hand. Thankfully we were able to reduce the amount of traffic (both bytes and requests), since the hosts were requesting a valid service via a well-formed protocol dialog, allowing us to tailor the response appropriately.

... ON DIAGNOSIS

At the time, our efforts to diagnose the problem seemed haphazard. On reflection, the steps followed do seem to have been sensible and moderately generic:

- Identify some rogue requests.
- Try to spot a signature that matches these requests.
- Look at all requests matching the signature (and refine the signature if necessary).
- Examine the corpus of requests, looking for indications of their likely origin.

Of course, to identify a signature requires some knowledge of what constitutes a normal request. In our case, had we been using the combined log file format all along, we might have realized sooner that something unusual was going on. In this case, simply monitoring the number of requests to the server would probably not have identified the problem, as the load on a server can plausibly increase gradually without arousing attention. However, as we saw from Figure 1, higher-order statistics make the problem much more obvious.

... ON COUNTERMEASURES

Our initial countermeasure was to send a smaller response to requests matching the signature. This technique has been adopted as a response to being Slashdotted [10] by a number of organizations. For example, Mozilla's Bugzilla database now returns a short static page in response to links from Slashdot. Similarly, it is not uncommon to follow a link from Slashdot to find a page that says, "If you really want to download the 1.5MB PDF report, please go to this page." In the case of one of the other Web servers listed by Tardis, they had no content at /, and were able to create a short page to satisfy the requests.

In our case, we identified that the client making the requests was not a full Web browser. This is why we could use an automatic redirect to accommodate legitimate requests accidentally matched by the rewrite rules. Unfortunately, this option is not available to those who have been Slashdotted.

As a general technique to stop remote sites from linking into specific parts of a Web site, it is possible to generate URLs that have a limited lifetime. However, such systems typically frustrate bookmark systems and search engines alike. Similarly, some people use limited lifetime email addresses to avoid spam. A more extreme version of these techniques could use limited lifetime DNS entries. Options like this were not available to us, as the URL and DNS name in question were too well known.

A consideration that we considered to be important in designing any response to an HTTP problem was that legitimate users and problem users may both be behind the same Web proxy (or NAT device). A student at a local university working on a spidering project repeatedly crawled the Mathworld [11] site. As a response, the Mathworld operators blocked access from the IP they saw the requests coming from. This resulted in blocking the student's entire department!

The final part of our countermeasure was designed to attract the attention of users involved in the problem. Importantly, changing the date returned by our Web server is only likely to attract the attention of users who are using that date for something unusual. It might possibly have confused the caching scheme of some browsers, but we have heard no reports to this effect. Notifying users who are not involved in the problem, as often occurs when virus notifications are returned in response to forged email addresses, can be counterproductive.

. . . ON SIMILAR NTP-RELATED INCIDENTS

There are eerie similarities between this event and a number of other incidents. At about the same time as our incident, CSIRO had to take action because of hundreds of thousands of clients accessing their server from outside Australia. The subsequent incident at Wisconsin [12], where the address of an NTP server was hardwired into a mass-produced DSL router, is probably best known.

Fortunately, our problem was on a smaller scale. Unlike the Wisconsin incident, the extent of the problem had actually been augmented by users configuring the system to poll frequently, rather than simple bad system design (though providing a slider that can be set to poll once per minute probably counts as bad design). It is amusing to note that we actually had to patch the source of Apache to produce our deliberate misconfiguration. This must be a rare example of a Windows GUI providing you with easy-to-use rope to hang yourself, while the config file-based system at the other end requires more work to induce "errors"!

One of the solutions considered at Wisconsin was to abandon the IP address of the host in question; however, this was not the final solution used. There have been incidents of the abandonment of domains because of poor design choices in time synchronization software [13]. We did consider moving our Web server before we had put our countermeasures in place, but this would have placed a much larger burden on our system administration staff.

An interesting question is, why has an apparently innocent service, time synchronization, caused so many problems? A significant part of the problem seems to be misuse of lists of well-known servers [6]. Though the list includes a clear statement that it is "updated frequently and should not be cached," many people serve local copies. A quick search with Google identifies many pages listing our retired server as a current NTP server, even though it has not been on the official list since 2000. Some of these pages include the retired server in example source code.

This suggests that providing standardized dynamic methods for determining an appropriate NTP server might be worth the development effort. Attempts to provide pools of active NTP servers behind one DNS name have proved quite successful in recent years [14]. Multicast NTP would provide a more topologically aware technique for discovering NTP servers; however, the still-limited availability of multicast makes this less practical. Making a number of anycast servers (or, more exactly, shared unicast servers) might also be beneficial. This technique has already been used successfully for the DNS roots and 6to4 relay routers [15]. Anycast NTP has been successfully deployed in the Irish Research and Education Network, HEAnet.

What other protocols/servers may be subject to similar problems? There are obvious parallels with DNS root servers, which are also enumerated with a highly cached list. The high level of bogus queries and attacks arriving at the root servers has been well documented [16].

Conclusions

The investigation of this problem was an interesting exercise in the sorts of problems that sysadmins end up tackling. We had to use many of the standard tools: log files, diagnostic utilities, Usenet, reconfiguration, and a little software hacking.

Our countermeasures remain in place today and seem relatively successful, as the unwanted traffic remains significantly reduced. The incident itself seems to fit into a larger pattern of problems with time synchronization software and statically configured services.

REFERENCES

[1] The common log file format was designed to be a standard format for Web servers and was used by the CERN httpd: see

http://www.w3.org/Daemon/User/Config/Logging .html. More recently the combined format has become more common: http://httpd.apache.org /docs/logs.html.

[2] Autocorrelation is a measure of how much correlation you see when you compare a value now with a value in the future: see http://en.wikipedia .org/wiki/Autocorrelation.

[3] Tcpdump's -X option is good for this kind of thing: http://www.tcpdump.org/.

[4] The thread on comp.protocols.time.ntp can be found at http://groups.google.com/group/comp .protocols.time.ntp/browse_thread/thread /710cc3fb87bd08cc/026820ef0e6b4165.

[5] The home page for Tardis Time Synchronization Software is at http://www.kaska.demon.co.uk/.

[6] David Mills's list of Public NTP Secondary (stratum 2) Time Servers was traditionally at http://www.eecis.udel.edu/~mills/ntp/clock2.htm but now lives in the NTP Wiki at http://ntp.isc.org/ bin/view/Servers/WebHome.

[7] Archive.org is very useful for checking the history of Web pages: http://www.archive.org/.

[8] FreeBSD's accept filters were developed by David Filo and Alfred Perlstein. There are filters that wait until there is data or a complete HTTP request queued on a socket: http://www.freebsd .org/cgi/man.cgi?query=accept_filter.

[9] The Apache Module for sending a file as is, mod_asis, is documented at http://httpd.apache .org/docs-2.0/mod/mod_asis.html.

[10] Being Slashdotted is, of course, being linked to from Slashdot and suffering an unexpected increase in requests as a consequence. Wikipedia has a nice entry at http://en.wikipedia.org/wiki /Slashdotted.

[11] Eric Weisstein's Mathworld is an online encyclopedia of mathematics predating the flurry of Wiki activity: http://mathworld.wolfram.com/.

[12] Dave Plonka's well-known report on routers flooding the University of Wisconsin time server can be found at http://www.cs.wisc.edu/~plonka /netgear-sntp/.

[13] A description of why the UltiMeth.net domain was abandoned can be found at http://www.ultimeth.com/Abandon.html.

[14] The NTP Server Pool project lives at http://www.pool.ntp.org/.

[15] RFC 3258 describes "Distributing Authoritative Name Servers via Shared Unicast Addresses," which is basically making DNS queries to an anycast address. A similar trick for finding a 6to4 relay is described in RFC 3068. Both these techniques seem to work pretty well in practice.

[16] There are a number of studies of requests arriving at the DNS root servers. Check out "DNS Measurements at a Root Server," available on the CAIDA Web site at http://www.caida.org /outreach/papers/bydate/.