Zeus Trojan Analysis || Cisco Talos Intelligence Group

By Published by Alex Kirk

Archived: 2026-04-05 17:37:35 UTC

One of the most high-profile pieces of malware in the current threat landscape is Zeus/Zbot, a nasty little trojan that has been employed by botnet operators around the world to steal banking credentials and other personal data, participate in click-fraud schemes, and likely numerous other criminal enterprises. It is the engine behind notorious botnets such as Kneber, which has recently made headlines worldwide. The following is an analysis of the network traffic generated by machines that Talos intentionally infected with known Zeus samples, in order to study post-infection behavior. The machines were all base installations of Windows XP, Service Pack 2, with no patches - i.e., designed to be as vulnerable as possible.

In some ways, a host freshly infected with Zeus is quite predictable. Within milliseconds of the infection, the host will send out an HTTP GET request, which results in the controlling server sending down a binary blob of data marked with a Content-Type of application/octet-stream. The size of this data varies wildly between infections - in the three sample PCAPs referenced at the bottom of this write-up, for example, the data was 3,793, 35,164, and 185,133 bytes, respectively. Judging solely from the declared Content-Type, initially one might suspect that these files are some sort of obfuscated binary; however, a bit of research at the excellent Zeus Tracker site shows that these are in fact encrypted configuration files. In fact, the request made in packet #4 of Sample 3 for hxxp://ishi-bati.com/kartos/kartos.bin matches a sample config file listed on that site, and the config file returned to the victim during Talos' analysis had an identical MD5sum to the one fetched by Zeus Tracker. Given the vast variability in the size of these config files, it is clear that infected hosts will engage in a variety of activities, making detection based on specific network traffic more difficult the longer a host has been infected.

Unfortunately for Snort signature writers of maintainers of web proxies, blocking these requests is not a simple process, due to the randomness of the URL paths being requested at this step of the process. Example URL paths from our research include:

- /Gallery/IMAG0081.GIF
- /btn001/config.bin
- /bugzy/i.cfg
- /cfg2
- /cfg3.bin
- /cnf/trl.jpg
- /config.bin
- /dzen/misc.inc.php
- /film/video.bin
- /ftr/vosmoipoint.bin
- /ftr/vosmoipont.bin
- /gkt/gld44.bin

- /good/tlz/cfg.bin
- /gus/pool.doc
- /ii1IGh.aeL8uf
- /im/cfg.bin
- /img/cfg.bin
- /index_files/4jpg.bin
- /inmake/lds/cfg.bin
- /kartos/kartos.bin
- /ldr/cfg.bin
- /n2.bin
- /norma/cf5.bin
- /ribbn.tar
- /s2/non.bin
- /sell.jpg
- /test/config.bin
- /ukk/cfg.bin
- /web/cfg.bin
- /z/config1.bin
- /z_bot/what.bin
- /zend/cfg.bin
- /zeus/config.bin
- /zs/cfg.bin
- /~am/szkolapanel/zs/config.bin
- /~update/serv/updtsys.bin

The HTTP headers in these config file requests are sparse, and again leave very little for Snort signature writers to key on. While there is some slight variability, typically the headers are as follows:

```
GET /config.bin HTTP/1.1 Accept: */* Connection: Close User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) Host: Pragma: no-cache
```

One behavior that is common among all infected machines that have completed this first step successfully, however, is a pair of HTTP POST requests, sent the same server that the config file was fetched from (the file name being POSTed to is identical between the requests). The requests, which ride on separate TCP streams, occur back to back, with no response from the controlling server between them; they typically occur approximately 30 seconds following the initial request for the configuration file. Again, the URL involved is fairly random, with examples from our research including:

- /4vnrye74mugh.php
- /4vnrye74vmugh.php
- /DZ3LOrAFpl.php
- /back11/stat1.php
- /btn001/gate.php
- /buy.php

- /dd7ejr8ehd8jrf.php
- /dzen/as9965767.php
- /free/wthong.php
- /gate.php
- /good/socialnetworks/all4love/peage.php
- /iXeij7Ai.php
- /im/s.php
- /img/s.php
- /index1.php
- /inmake/page/gate.php
- /kartos/youyou.php
- /test/gate.php
- /trl/gate.php
- /vvn/ci_g.php
- /web/gate.php
- /z/s.php
- /z_bot/bot_adented.php
- /zend/gate.php
- /zs/gate.php

The HTTP headers remain identical to those of the original config file request, plus a standard Content-Length header to specify the size of the data being sent. The POST data itself is encrypted, and of moderately variable size: one of the two is typically close to 270 bytes, and the other is typically close to 350. Once more, this leaves Snort signature writers or proxy maintainers with virtually nothing to key on for a signature.

The data returned in response to these POST requests is the first place where we see behavior anomalous enough that it can form the basis for an Snort signature. The HTTP headers themselves are much more variable than those in the clients' requests, as different controllers use different types of servers, which supply different sets of headers, different values in the Server header, etc. The one header field in these responses that has been consistent across all of our samples is "Content-Type: text/html". By itself, this is a standard header, and a Snort signature based just on this would generate alerts on most legitimate web traffic, thus rendering it useless. The actual data being supplied, which ranges in size between roughly 40 and 100 bytes, is not HTML (or, at the very least, not plaintext HTML), which differentiates these packets from normal, legitimate responses - there is a mismatch between the declared Content-Type and the actual content delivered.

How, you might ask, can you write a Snort rule based on this information, given that there is no functionality for determining the type of data being received from a web server? Obviously, you begin by looking for the string "Content-Type: text/html", which is required for this mismatch to exist. Next, you look for "|0D 0A 0D 0A|" ("\r\n\r\n" / "CR LF CR LF"), which signifies the end of the HTTP headers and the beginning of the data portion of the packet. From there, you use a negated content match, combined with a distance:0 clause, to verify that the string "html" does not appear before the end of the packet. Since all HTML documents must begin with an HTML tag, the string "html" should be present in all normal web responses, causing the rule to fail when evaluating them. Add in a negated content match for "Content-Encoding: gzip" to bail out of packets where the response data is

gzipped (and thus will not contain the plaintext string "html"), and you have a rule that will alert on these Zeus packets - specifically, SID 16460 of the Talos Certified Rules set.

That being said, it's clear that this rule will probably detect packets that are not actually Zeus traffic. Other pieces of malware may exhibit similar behavior, and it is possible that some legitimate applications will have a Content-Type mismatch like this. However, a human analyst can easily identify cases of poorly formed HTML, legitimate servers that behave oddly, etc. - and any server that is sending back encrypted or otherwise malformed traffic will be identified for further inspection. Talos feels that the benefit of detecting Zeus C&C traffic, as well as potentially other malicious traffic, outweighs the burden represented by these potential false positives. That said, knowing that some sites have little to no tolerance for false positives, we have disabled this rule by default, so that individual administrators - who know their networks and their tolerance for potential false positives - can make this decision for themselves on their own.

Beyond this exchange of data, different Zeus samples began to diverge in their behavior. Some of them went silent for the remainder of our pre-determined sample run time, presumably because their controlling servers either had nothing for them to do at that time, because they were waiting for a user to enter credentials that could be stolen and reported back, or for any of a number of possible reasons. Many of them next downloaded a Windows PE executable file (which will be detected by Talos Certified SID 11192); typically, these binaries are updates to the Zeus client software, to ensure that the newly infected host is up-to-date in its capabilities. Again, the filenames associated with these binaries were fairly random, and were typically only minimally associated, if at all, with the name used to request the original configuration file - occasionally they resided in the same directory on the remote server as the config file, but that was the extent of the similarity. Sample names included:

- /40.exe
- /fshit/d.exe
- /good/tlz/server32.exe
- /gus/windir.exe
- /inmake/lds/server32.exe
- /kartos/krt.exe
- /ldr/ldr.exe
- /load.exe
- /money-s2.exe
- /money.exe
- /rhueirh4furh74.exe
- /ser.exe
- /t.exe
- /trhr7y4urjhe83.exe

In most cases, once this new binary was downloaded, the infected machine sent out another POST request to the same URL touched after the initial infection, presumably informing its controlling server that the update had been successfully completed.

A relatively small percentage of the infected machines then proceeded to engage in click fraud, focused on pornographic sites - specifically, visiting hundreds of sites in rapid succession, thus artificially driving up page

views and generating cash for the advertisers on those pages. As these machines were incredibly "noisy" (i.e. they generated a huge volume of traffic), they should be easily identifiable by the sudden spike in traffic associated with the machine in question. Additionally, if any sort of checks for visits to pornographic web sites are being done for a given network, red flags should go up immediately for these particular hosts.

The remainder of the samples we ran remained largely silent. A few visited Google to verify connectivity to the Internet, or touched one or two other sites briefly; some continued to communicate with their controlling servers using the previously discussed method. By and large, though, these systems were very stealthy from a network perspective.

Of course, many of the machines that we infected were unable to reach their pre-assigned control server, as Zeus controllers are being discovered and disabled by law enforcement and/or vigilant service providers on a regular basis. In these cases, we found Zeus to be extremely persistent in its attempts to get a hold of the server in question. In cases where the DNS name of the control server was not found, the bot would send multiple requests for the name in question to all of its DNS servers, often in rapid succession. While a precise threshold for the number of failed DNS lookups in a given period of time is difficult to determine - some samples generated dozens or more requests within just a few seconds, while others would simply retry at random intervals - repeated DNS query failures, particularly for the same name, coming from a single host in a period of minutes would be a good indicator of an infected machine. In cases where the name of the server resolved properly, but the requested URL did not return a configuration file (i.e. a 404 Not Found was generated, a 302 Moved response redirected to something besides a config file, or a 200 OK with a takedown note was received), the same request was made over and over and over again throughout our analysis time frame. Again, the timing of these requests was random between samples, but each request would generally occur between a few seconds and a few mintues apart - i.e., a short enough time frame that, if these requests were detected by a web proxy or other system, they could easily be flagged as suspicious, leading to detection of the compromised host.

Talos will be continuing to run this type of analysis on Zeus-infected machines, and will be updating this document and the Talos Certified Rules as appropriate. In the meantime, we are providing three sample packet captures to the public, in order to help educate security engineers on what Zeus traffic looks like in the wild; see below for the PCAPs.
