Stats from Hunting Cobalt Strike Beacons

svch0st.medium.com/stats-from-hunting-cobalt-strike-beacons-c17e56255f9b

svch0st May 6, 2021

svc h0st

svch0st

May 6, 2021

.

4 min read

Some Statistics on Cobalt Strike Configs in April and May 2021

Collected from over 1000 configurations, here are some high-level statistics that demonstrate some of the common trends among one of the most popular tools in an adversary's arsenal. These configs were collected from live servers around early May 2021.

If you are interested in how the data was collected, scroll to the bottom of the article.

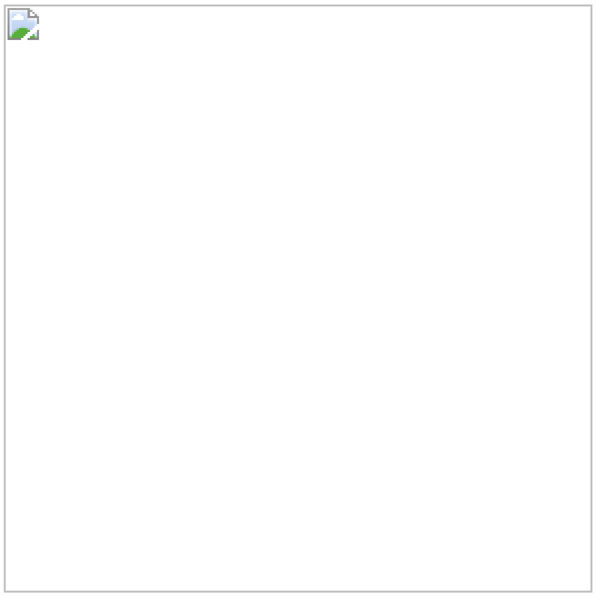
If you want to read more about how the configurations are structured in Cobalt Strike payloads his article is a good start:

Cobalt Strike Staging and Extracting Configuration Information

By default Cobalt Strike exposes its stager shellcode via a valid checksum8 request (the same request format used in…

blog.securehat.co.uk

Most common watermark


Unsurprisingly most common watermark was 0. The watermark of 0 is indicative of cracked versions for Cobalt Strike which are commonly used by threat actors in their campaigns. More interestingly is 305419896, 1359593325, and 1580103814, all had configuration counts above 100.

The watermark 305419896 has been associated with the Maze ransomware:

Enter the Maze: Demystifying an Affiliate Involved in Maze (SNOW) - SentinelLabs

By Jason Reaves and Joshua Platt Maze continues to be one of the most dangerous and actively developed ransomware…

labs.sentinelone.com

User Agents

Besides the standard user agents imitating web browsers, several configurations had the user agent of “Shockwave Flash”

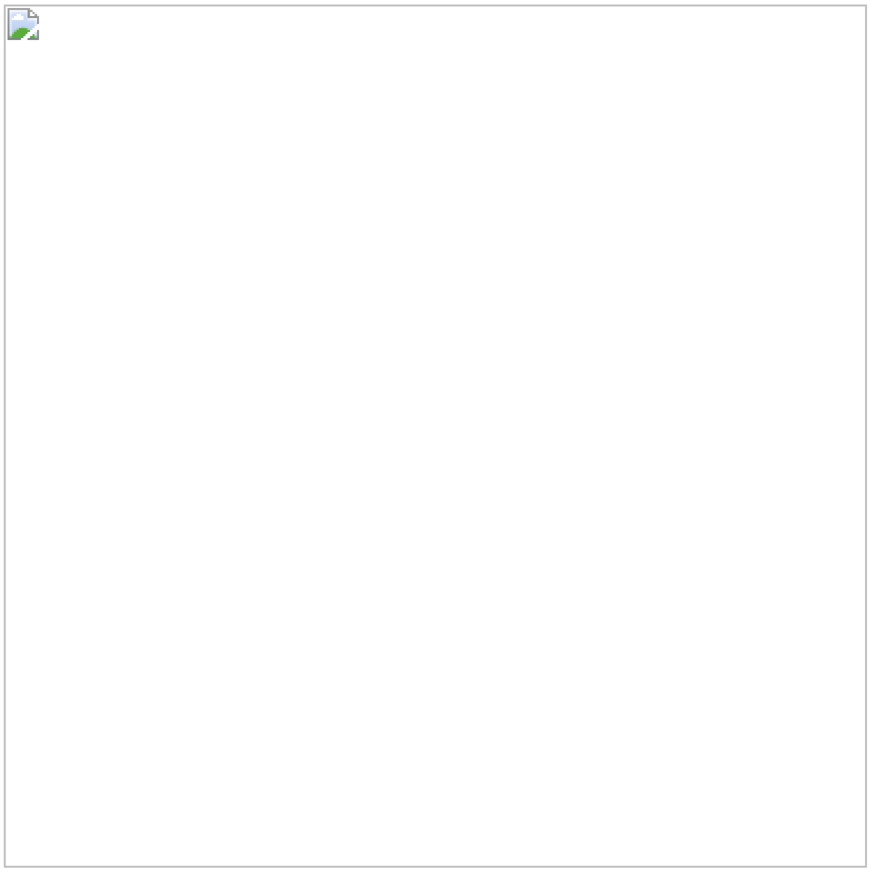
Interesting URI

The more standard URIs of */submit.php* and */jquery-3.3.2.min.js* were the most common but this one stood out to me:

> /r/webdev/comments/95lyr/slow_loading_of_google

Most common process spawn targets

The default values of rundll32.exe were the most common.



Looking at the information, I was most interested in some of the more custom values such as:

- %windir%\syswow64\eventvwr.exe
- %windir%\syswow64\backgroundtaskhost.exe
- %windir%\system32\mobsync.exe
- %windir%\sysnative\adobe64.exe

How I collected the Data

I used 2 main queries to get as many C2 IPs as quickly as possible.

- RiskIQ prebuild component to search for (requires a free account) (~8k IPs)
- A search on JARM hashes that I had found in a recent case (~10k IPs):

```
JARMFuzzy: 07d14d16d21d21d07c42d41d00041d
```

If you want to learn more about JARM, which is developed by the Salesforce team, this is a great article:

Easily Identify Malicious Servers on the Internet with JARM

JARM is an active Transport Layer Security server fingerprinting tool that provides the ability to identify and group…

engineering.salesforce.com

This data contained many IPs that were burnt by the time of analysis; however, it still provided a decent enough dataset to get some results.

Retreiving the Configs

I had two ideas for harvesting configs; one would be downloading the binary payload from sources like Virustotal and MalwareBazaar and parsing them.

But if we know the IPs already, let's just ask the servers for the config?

There are a bunch of awesome tools that exist for extracting and parsing Cobalt beacon configs, but the one I used for this was a Nmap NSE script by Wade Hickey.

whickey-r7/grab_beacon_config

Permalink Failed to load latest commit information. No description, website, or topics provided. You can't perform that…

github.com

I had added some error exception handling and most importantly an extra line to pull out the beacon Watermark (or license number) which is very helpful for threat intelligence.

See my fork here:

[svch0stz/grab_beacon_config]

[Contribute to svch0stz/grab_beacon_config development by creating an account on GitHub.]

[github.com]

I then used the IP lists I had as input and ran the Nmap script.

```
nmap --script=grab_beacon_config.nse -p 80,443,8080 -iL jarmfuzzy.txt -oA jarmfuzzy -T4
```

The output of the script will look something like this:

```
|_grab_beacon_config: {"x86": {"uri_queried": "\/DxRN", "md5":
"7118007ad133a9dcd59419beef0896a5", "config": {"Jitter": 0, "Spawn To x64":
"%windir%\\system32\\mobsync.exe", "Max DNS": 255, "C2 Server":
"thefaithfulamerican.com,\/s\/ref=nb_sb_noss", "DNS Sleep": 0, "HTTP Method Path 2":
"\/gp\/product\/sessionCacheUpdateHandler.html", "Method 2": "POST", "Pipe Name": "",
"Polling": 5000, "Spawn To x86": "%windir%\\syswow64\\nslookup.exe", "Method 1": "GET",
"Port": 443, "Beacon Type": "8 (HTTPS)", "DNS Idle": "0.0.0.0", "C2 Host Header": "",
"User Agent": "5.0 (Windows NT 10.0) AppleWebKit\/537.36 (KHTML, like Gecko)
Chrome\/99.0.7113.93 Safari\/537.36", "Header 2": "", "Watermark": 1, "Header 1": ""},
"sha1": "590d700e79f71222c959cc2be46070725c5f76a1", "time": 1619933006124.8, "sha256":
"f89869fd338c4ef527f31f836308f9906a3991ac45d6f56d4004eea8f91e6ca3"}, "x64":
{"uri_queried": "\/1aZq", "md5": "bf1ca95090d0870729c72b216b7964b5", "config":
{"Jitter": 0, "Spawn To x64": "%windir%\\system32\\mobsync.exe", "Max DNS": 255, "C2
Server": "thefaithfulamerican.com,\/s\/ref=nb_sb_noss", "DNS Sleep": 0, "HTTP Method
Path 2": "\/gp\/product\/sessionCacheUpdateHandler.html", "Method 2": "POST", "Pipe
Name": "", "Polling": 5000, "Spawn To x86": "%windir%\\syswow64\\nslookup.exe", "Method
1": "GET", "Port": 443, "Beacon Type": "8 (HTTPS)", "DNS Idle": "0.0.0.0", "C2 Host
Header": "", "User Agent": "5.0 (Windows NT 10.0) AppleWebKit\/537.36 (KHTML, like
Gecko) Chrome\/99.0.7113.93 Safari\/537.36", "Header 2": "", "Watermark": 1, "Header 1":
""}, "sha1": "f4cf79e7735053836fe7e435d7b0c0a511aa2ea6", "time": 1619933013556.5,
"sha256": "bf4ee9664fba51a1bbbdad13a598688914a48465fac3993c096c6d2cc0c2c021"}}
```

From there, it was an exercise of cleaning up the data into something useable and using Excel-fu to get some ugly pie graphs :)