Extracting Cobalt Strike Beacon Configurations

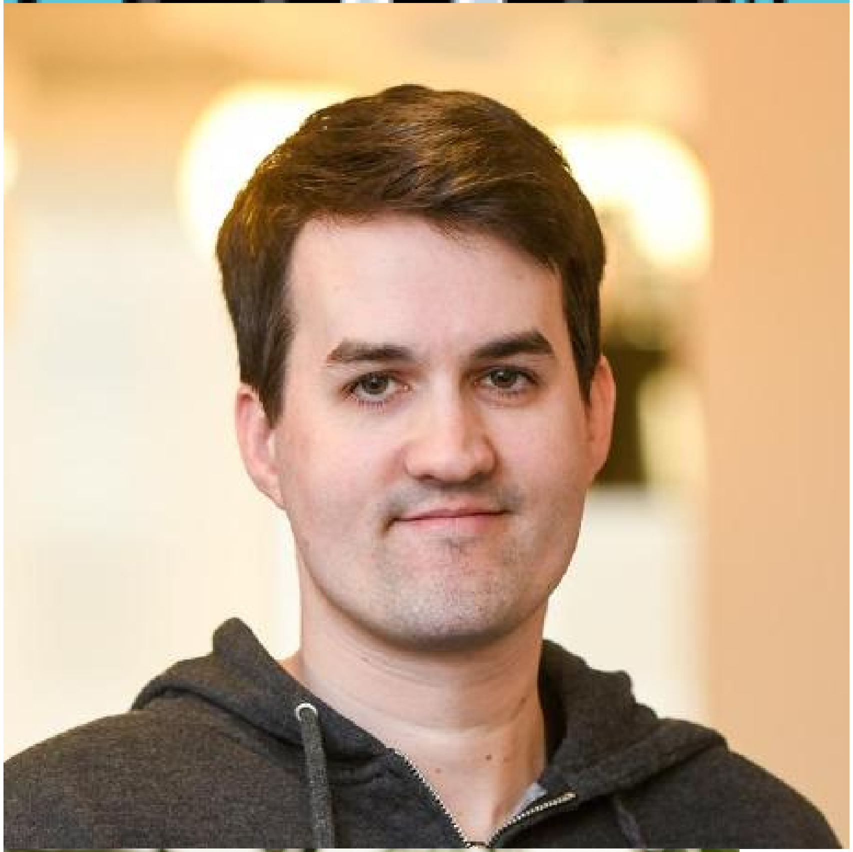
elastic.github.io/security-research/intelligence/2022/01/03.extracting-cobalt-strike-beacon/article/

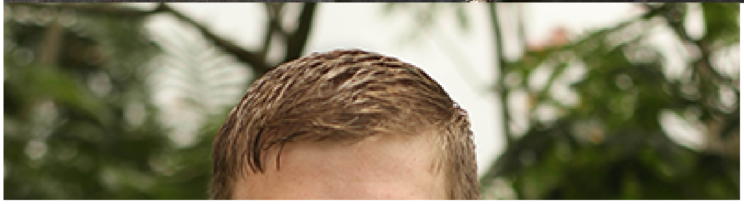
Elastic Security Research

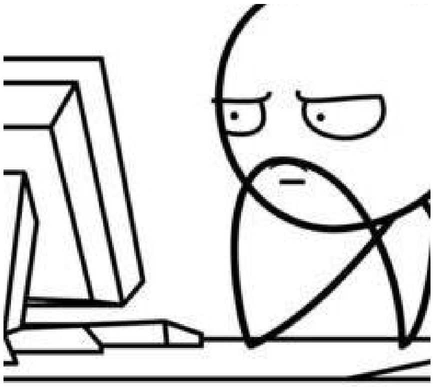
Extracting Cobalt Strike Beacon Configurations

Part 2 - Extracting configurations from Cobalt Strike implant beacons.

Cobalt Strike







2022-01-19

Please check out our previous post on how to collect Cobalt Strike beacon implants. We'll build on that information to extract the configurations from the beacons.

In this post, we'll walk through manually analyzing a Cobalt Strike C2 configuration from a binary beacon payload using the excellent Cobalt Strike Configuration Extractor (CSCE). We'll also cover enabling some newer features of the Elastic Stack that will allow you to do this at scale across all your monitored endpoints, by extracting the beacons from memory.

Shout Out

The team at Blackberry has a tremendous handbook called "Finding Beacons in the Dark" (registration required) that dives extensively into Cobalt Strike beacon configurations. We'll discuss a few fields in the configurations here, but if you're interested in learning about how beacons function, we strongly recommend checking that resource out.

Cobalt Strike Configuration Extractor¶

The Cobalt Strike Configuration Extractor (CSCE) by Stroz Friedberg is a "python library and set of scripts to extract and parse configurations from Cobalt Strike beacons".

To use the CSCE, we'll create a Python virtual environment, activate it, and install the CSCE Python package.

Setting up the Cobalt Strike Configuration Extractor
Next, we can run the CSCE on the beacon payload we extracted from memory to see if there's any interesting information stored we can collect (we'll add the `--pretty` flag to make the output easier to read as a JSON document).

Viewing the atomic indicators of the CS beacon configuration

```
(csce) $ csce --pretty beacon.exe

{
  "beacontype": [
    "HTTPS"
  ],
  "sleeptime": 45000,
  "jitter": 37,
  "maxgetsize": 1403644,
  "spawnto": "GNEtW6h/g4dQzm0dOkL5NA==",
  "license_id": 334850267,
  "cfg_caution": false,
  "kill_date": "2021-12-24",
  "server": {
    "hostname": "clevelandclinic[.]cloud",
    "port": 443,
    "publickey": "MIGfMA0GCSqGSIb3DQEBAQUAA4G...
...truncated...
```

Immediately, we can see that the beacon uses HTTPS to communicate and that the domain is `clevelandclinic[.]cloud`. This gives us an atomic indicator that we can do some analysis on. Looking at the Malleable Command and Control documentation, we can get a description of the configuration variables.

As an example, we can see that the `sleeptime` is `450000` milliseconds, which changes the default beacon check in from every 60-seconds to 450-seconds, or 7 ½ minutes. Additionally, we see a jitter of `37` meaning that there is a random jitter of 37% of `450000` milliseconds (`166,500` milliseconds), so the beacon check-in could be between `283,000` and `450,000` milliseconds (4.7 - 7.5 minutes).

Additionally, the `publickey` field is used by the Cobalt Strike Team Server to encrypt communications between the server and the beacon. This is different from normal TLS certificates used when accessing the C2 domain with a browser or data-transfer libraries, like `cURL`. This field is of note because the Team Server uses the same publickey for each beacon, so this field is valuable in clustering beacons with their perspective Team Server because threat actors often use the same Team Server for multiple campaigns, so this data from the configuration can be used to link threat actors to multiple campaigns and infrastructure.

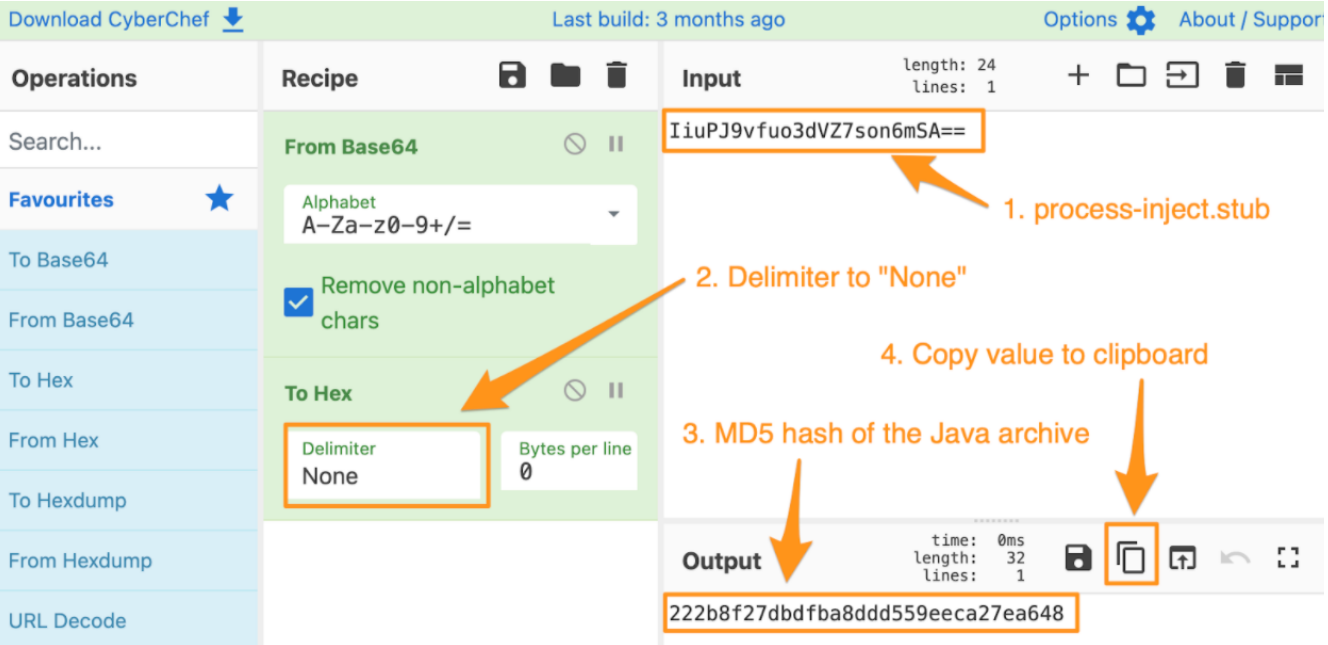
Continuing to look at the configuration output, we can see another interesting section around the `process-inject` nested field, `stub`:

Viewing the process-inject.stub field

```
(csce) $ csce --pretty beacon.exe

...truncated...
  "process-inject": {
    "allocator": "NtMapViewOfSection",
    "execute": [
      "CreateThread 'ntdll!RtlUserThreadStart'",
      "CreateThread",
      "NtQueueApcThread-s",
      "CreateRemoteThread",
      "RtlCreateUserThread"
    ],
    "min_alloc": 17500,
    "startrwx": false,
    "stub": "IiuPJ9vfuo3dVZ7son6mSA==",
    "transform-x86": [
      "prepend '\\x90\\x90'"
    ],
...
```

The `stub` field contains the Base64 encoded MD5 file hash of the Cobalt Strike Java archive. To convert this, we can again use CyberChef, this time add the “From Base64” and “To Hex” recipes.



From Base64
Alphabet: A-Za-z0-9+/=
☑ Remove non-alphabet chars

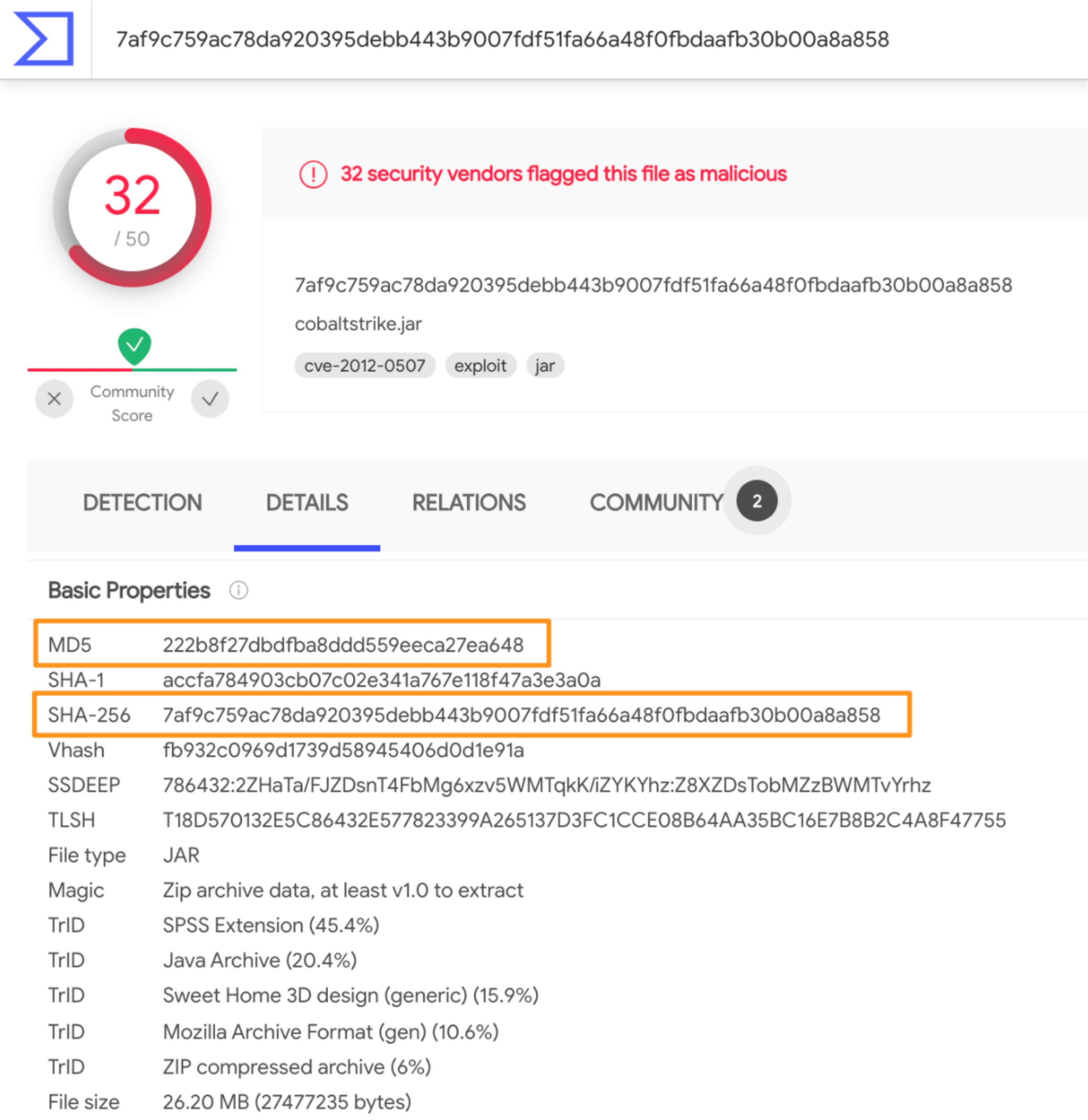
To Hex
Delimiter: None
Bytes per line: 0

Input (length: 24, lines: 1):
`IiuPJ9vfuo3dVZ7son6mSA==`

Output (time: 0ms, length: 32, lines: 1):
`222b8f27dbdfba8ddd559eeca27ea648`

1. process-inject.stub
2. Delimiter to "None"
3. MD5 hash of the Java archive
4. Copy value to clipboard

Now that we have the MD5 value of the Java archive (`222b8f27dbdfba8ddd559eeca27ea648`), we can check that against online databases like VirusTotal to get additional information, specifically, the SHA256 hash (`7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858`).



7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858

32 security vendors flagged this file as malicious

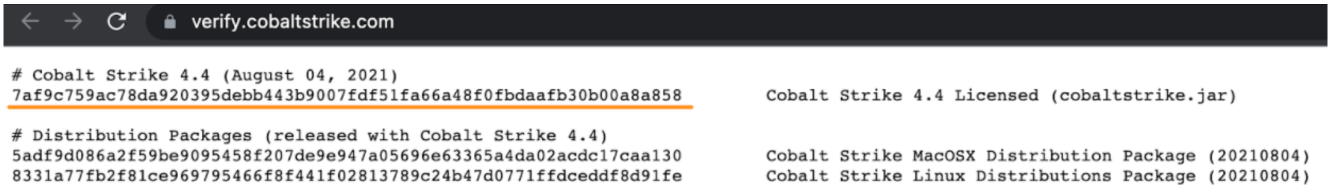
7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858

cobaltstrike.jar

Basic Properties

MD5	222b8f27dbdfba8ddd559eeca27ea648
SHA-1	accfa784903cb07c02e341a767e118f47a3e3a0a
SHA-256	7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858
Vhash	fb932c0969d1739d58945406d0d1e91a
SSDEEP	786432:2ZHaTa/FJZDsnT4FbMg6xzv5WMTqkK/iZYKYhz:Z8XZDsTobMZzBWMTvYrhz
TLSH	T18D570132E5C86432E577823399A265137D3FC1CCE08B64AA35BC16E7B8B2C4A8F47755
File type	JAR
Magic	Zip archive data, at least v1.0 to extract
TrID	SPSS Extension (45.4%)
TrID	Java Archive (20.4%)
TrID	Sweet Home 3D design (generic) (15.9%)
TrID	Mozilla Archive Format (gen) (10.6%)
TrID	ZIP compressed archive (6%)
File size	26.20 MB (27477235 bytes)

Finally, we can verify the SHA256 hash with CobaltStrike to identify the version of the Java archive by going to https://verify.cobaltstrike.com and searching for the hash.



```
# Cobalt Strike 4.4 (August 04, 2021)
7af9c759ac78da920395debb443b9007fdf51fa66a48f0fbdaafb30b00a8a858        Cobalt Strike 4.4 Licensed (cobaltstrike.jar)

# Distribution Packages (released with Cobalt Strike 4.4)
5adf9d086a2f59be9095458f207de9e947a05696e63365a4da02acdc17caa130        Cobalt Strike MacOSX Distribution Package (20210804)
8331a77fb2f81ce969795466f8f441f02813789c24b47d0771ffdceddf8d91fe        Cobalt Strike Linux Distributions Package (20210804)
```

Now we know that this beacon was created using a licensed version of Cobalt Strike 4.4.

Another field from the configuration that is helpful in clustering activity is the `license_id` field.

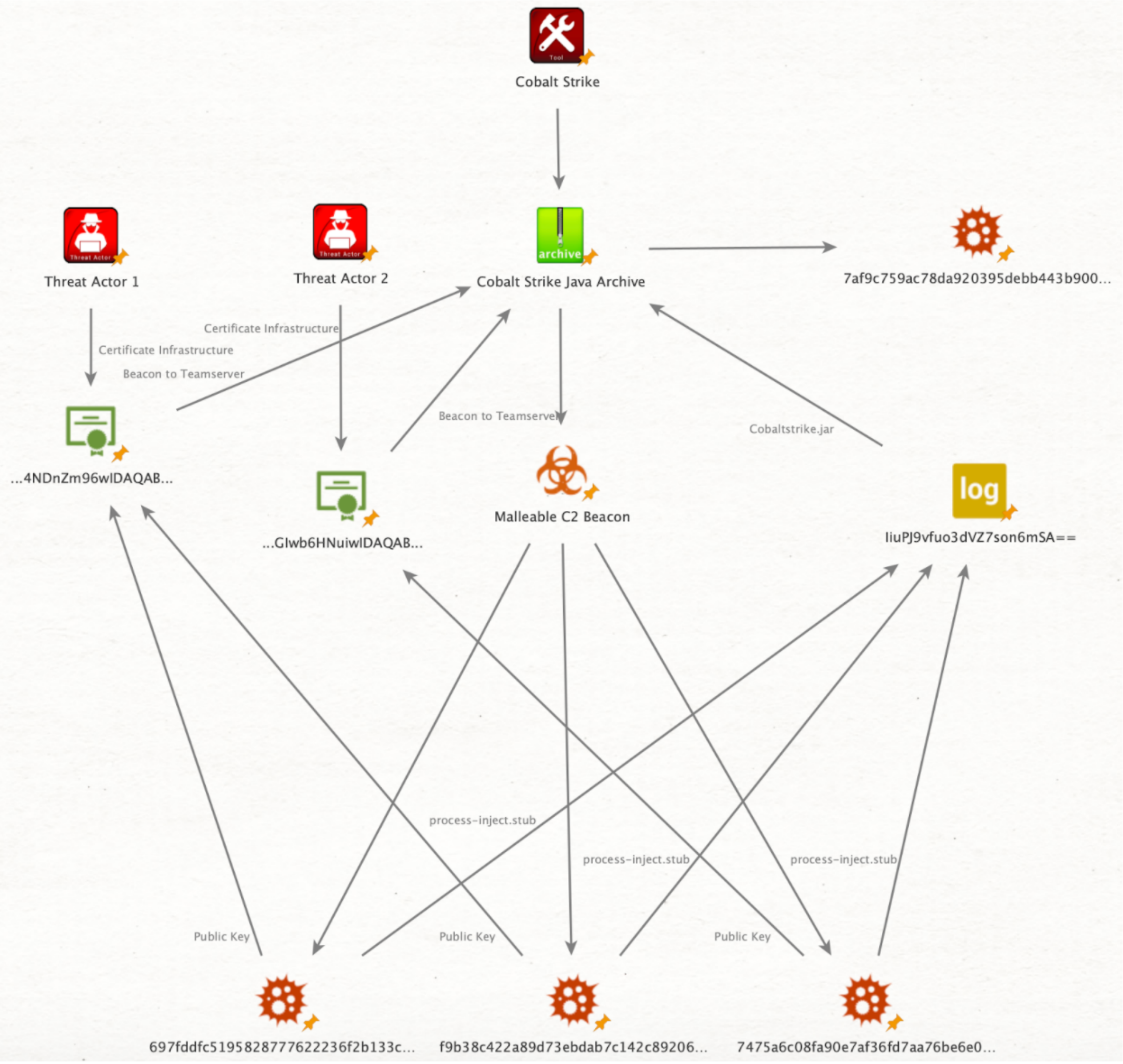
Viewing Cobalt Strike watermark

This is commonly referred to as the Watermark and is a 9-digit value that is unique per license. While this value can be modified, it can still be used in conjunction with the `process-inject.stub` and `publickey` fields (discussed above) to cluster infrastructure and activity groups.

These are just a few fields that can be used to identify and cluster activities using configurations extracted from the Cobalt Strike beacon. If you're interested in a very in-depth analysis of the configuration, we recommend you check out the Finding Beacons in the Dark Cobalt Strike handbook by the team at Blackberry.

Putting Analysis to Action¶

To test out our analyst playbook for collecting Cobalt Strike beacon payloads, their configurations, and metadata contained within; we can apply those to more data to identify clusters of activity.

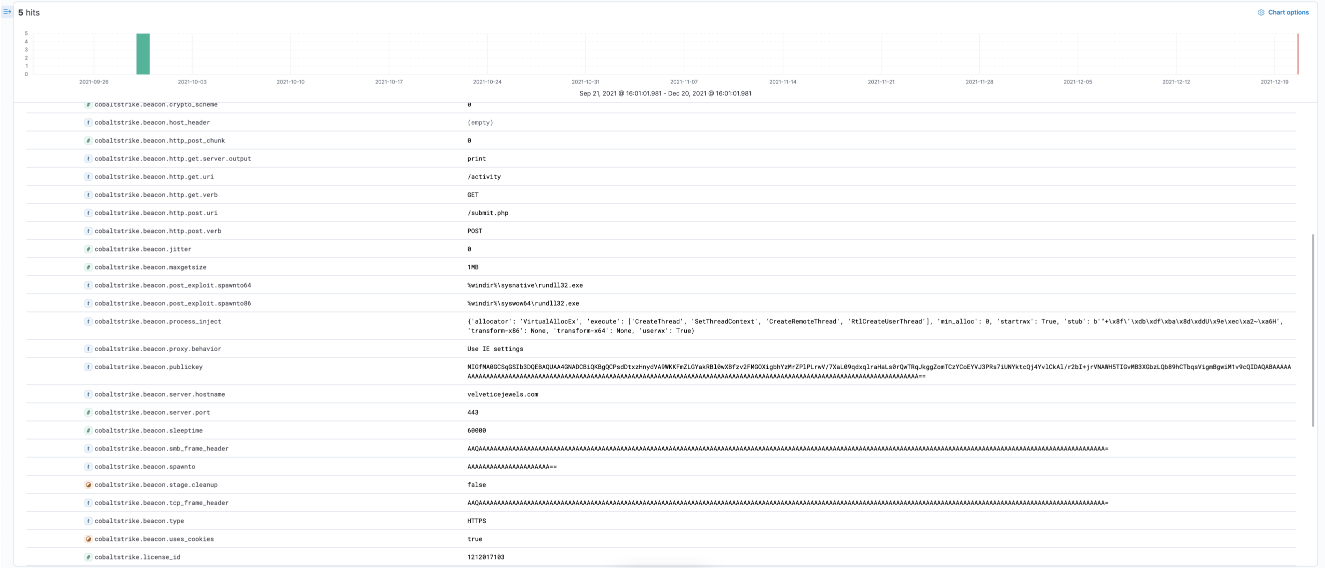


In the above illustration, we can cluster threat actors based on their shared uses of the beacon payload public key, which as we described above, is unique per Team Server. This would allow us to group multiple beacon payload hashes, infrastructure, and campaigns to a single Threat Actor.

As always, using the atomic indicators extracted from the beacon payload configurations (`clevelandclinic[.]cloud` in our example) allow you to identify additional shared infrastructure, target verticals, and threat actor capabilities.

This time at full speed¶

All of the steps that we've highlighted in this release, as well as the previous release, can be automated and written into Elasticsearch using the Cobalt Strike Beacon Extraction project.



cobaltstrike.beacon.crypto_scheme	0
cobaltstrike.beacon.host_header	(empty)
cobaltstrike.beacon.http_post_chunk	0
cobaltstrike.beacon.http.get.server.output	print
cobaltstrike.beacon.http.get.uri	/activity
cobaltstrike.beacon.http.get.verb	GET
cobaltstrike.beacon.http.post.uri	/submit.php
cobaltstrike.beacon.http.post.verb	POST
cobaltstrike.beacon.jitter	0
cobaltstrike.beacon.maxgetsize	1MB
cobaltstrike.beacon.post_exploit.spawnto64	%windir%\sysnative\rundll32.exe
cobaltstrike.beacon.post_exploit.spawnto86	%windir%\syswow64\rundll32.exe
cobaltstrike.beacon.process_inject	{'allocator': 'VirtualAllocEx', 'execute': ['CreateThread', 'SetThreadContext', 'CreateRemoteThread', 'RtlCreateUserThread'], 'min_alloc': 0, 'startrwx': True, 'stub': b'"+\x8f\'\xdb\xdf\xba\x8d\xddU\x9e\xec\xa2~\xa6H', 'transform-x86': None, 'transform-x64': None, 'userwx': True}
cobaltstrike.beacon.proxy.behavior	Use IE settings
cobaltstrike.beacon.publickey	MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCPsdDtxzHnydVA9WKKFmZLGYakRBl0wXBfzv2FMGOXigbhYzMrZPlPLrwV/7XaL09qdxqlraHaLs0rQwTRqJkggZomTCzYCoEYVJ3PRs7iUNYktcQj4YvlCkAl/r2bI+jrVNAWH5TIGvMB3XGbzLQb89hCTbqsVigmBgwiM1v9cQIDAQABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==
cobaltstrike.beacon.server.hostname	velveticejewels.com
cobaltstrike.beacon.server.port	443
cobaltstrike.beacon.sleeptime	60000
cobaltstrike.beacon.smb_frame_header	AAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
cobaltstrike.beacon.spawnto	AAAAAAAAAAAAAAAAAAAAAA==
cobaltstrike.beacon.stage.cleanup	false
cobaltstrike.beacon.tcp_frame_header	AAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
cobaltstrike.beacon.type	HTTPS
cobaltstrike.beacon.uses_cookies	true
cobaltstrike.license_id	1212017103

Summary¶

In this post, we highlighted new features in the Elastic Stack that can be used to collect Cobalt Strike Malleable C2 beacon payloads. Additionally, we covered the processes to build Fleet policies to extract beacon payloads from memory and their configurations.

These Fleet policies and processes enable security analysts to collect Cobalt Strike beacon payloads and their configurations to identify threat actor controlled infrastructure and cluster activity.

Artifacts¶

Observable	Type	Note
`697fddfc5195828777622236f2b133c0a24a6d0dc539ae7da41798c4456a3f89`	SHA256	Cobalt Strike Malleable C2 beacon payload

Observable	Type	Note
`7475a6c08fa90e7af36fd7aa76be6e06b9e887bc0a6501914688a87a43ac7ac4`	SHA256	Cobalt Strike Malleable C2 beacon payload
`f9b38c422a89d73ebdab7c142c8920690ee3a746fc4eea9175d745183c946fc5`	SHA256	Cobalt Strike Malleable C2 beacon payload
`clevelandclinic[.]cloud`	domain-name	Cobalt Strike Malleable C2 domain
`104[.]197[.]142[.]19`	ipv4-addr	Cobalt Strike Malleable C2 IP address
`192[.]64[.]119[.]19`	ipv4-addr	Cobalt Strike Malleable C2 IP address

Artifacts¶

Artifacts are also available for download in both ECS and STIX format in a combined zip bundle.

Download indicators.zip

Last update: January 31, 2022
Created: January 19, 2022