Threat Hunting Introduction: Cobalt Strike

By Artem Golubin

Published: 2025-06-23 · Archived: 2026-04-02 12:01:35 UTC

Cobalt Strike is a threat simulation tool that is used by red teams to perform penetration tests (simulate cyber-security attacks). However, it is also used by malicious actors to perform real-world attacks. In this article, I will demonstrate how anyone can find Cobalt Strike servers on the internet and retrieve metadata from them.

After reading this article, you will be able to peek inside the Cobalt Strike servers and see how big companies or malicious actors are using it. Doing this can be pretty fun, it's like being able to hack into hackers or security experts.

I will also introduce you to my tool written in Rust, called SigStrike, which I wrote to increase crawling speed by 20x.

Threat Hunting

Threat hunting is a proactive process of searching for threats that can potentially attack your company. The part of the threat-hunting process consists of the collection of so-called IoCs (Indicators of Compromise). IoCs are artifacts that researchers use to identify malicious activity.

The most common IoCs are:

- IP addresses
- Domain names
- File hashes
- Network traffic patterns
- Binary signatures (e.g., YARA rules)

In this post, I will focus on the IP addresses since they can be obtained by scanning internet. Even if you are not a security researcher, this can be fun!

How do people typically handle IP IoCs in large companies? They collect them from different sources and block them in their firewalls. They also monitor the history of connections to these IP addresses and domains within their networks.

What is Cobalt Strike?

Cobalt Strike, a commercial tool, is used to gain remote access to a system and perform post-exploitation activities. It's important to note that this happens after the system has already been compromised. You can think of it as a remote control tool that is relatively stealthy to deploy and use.

Cobalt Strike consists of two main components:

- **Server** – is used to control the compromised systems as well as serve malware from it.
- **Beacon** – is the agent that is installed on the compromised system (executable file).

The server part is usually hosted on cloud infrastructure, sometimes on a bulletproof hosting provider. It consists of a web server and a command and control (C2) server that communicates with infected machines.

If staging is enabled, the web server hosts beacons (executables) and serves them to the compromised systems. The interesting part here is that oftentimes, the beacon (executable) can be obtained by any person.

In this post, I will demonstrate how to exploit this fact and find Cobalt Strike servers on the internet and get metadata from them.

Finding Cobalt Strike servers

Most of the Cobalt Strike servers can be directly accessed via IPv4 address. There is no need to know a domain name.

But how do you find IP addresses?

If you have the capability to scan the internet, you can use tools like masscan and scan popular ports that Cobalt Strike uses. The default ports are 80 and 443 and it's a good starting point. If you don't have such capability, you can use services like Shodan, Censys, Hunt.io, Fofa, or ZoomEye.

Some of these services already tag servers as Cobalt Strike, so you can just search for them. For example, this is a query for shodan (no registration is required):

Please note that this will only return a portion of Cobalt Strike servers that are easy to detect. Shodan automatically parses the metadata and outputs it in a human-readable format. However, it can't parse and detect some of them when non-default encryption keys are used, or the config is slightly modified. So you will still have to query them for potential candidates and validate them yourself.

Alternatively, you can just grab some sample data from drb-ra/C2IntelFeeds repository.

Scanning the internet

When using mass internet scanners, you only get information about which ports are open and which services they are running. But how do you know that the server is actually a Cobalt Strike server?

As it turns out, the default Cobalt Strike server has a very specific response to HTTP requests.

When you hit the main page of the server, it will return a 404 Not Found response with a specific header:

```
 HTTP/1.1 404 Not Found
Date: Sat, 21 Jun 2025 21:48:05 GMT
Content-Type: text/plain
Content-Length: 0
```

If you scan the whole internet for 404 Not Found responses with zero content length, you will get around 16 million results. You can filter down it to only 30-50k records by also filtering by `Content-Type: text/plain` header. The problem with such an approach is that `Content-Type` header can be easily changed in Cobalt Strike, unlike the HTTP 404 Not Found response with zero content length.

There are more than 500 million unique IP and Port combinations that are open on the internet. Using simple signatures reduces the candidates to a manageable number.

If you want to find as many servers as possible, you will have to probe each response that returns `HTTP 404 Not Found` with zero content length. That's why I wrote a tool that can do it fast.

Most of the search engines (mentioned above) allow you to query such responses as well.

Obviously, not all the servers that return such responses are Cobalt Strike servers.

What if you don't have the capability to scan the whole internet but still want to perform some small scans?

You can always limit the scanning scope to specific hosting providers that are known to host a lot of Cobalt Strike servers. But keep in mind that your can be blocked by your hosting provider if you scan too aggressively.

How to actually find Cobalt Strike servers?

When staging is enabled, Cobalt Strike servers serve small executable files (agents) that later download bigger payloads.

Stager executable can be downloaded directly by sending a GET request with a specific path that contains checksum.

For checksum, Cobalt Strike uses 8-bit checksum algorithm similar to used in meterpreter.

Basically, your request should send a specific 4-character string that returns one of the following checksums:

- 92 - Checksum for x86 binary
- 93 - Checksum for x64 binary

Here are example of such URLs:

```
 http://<ip>:<port>/tdN6 - x86 binary
http://<ip>:<port>/ZrJG - x64 binary
```

Both of these values can be used on any Cobalt Strike server, but you can also randomize them. There are tens of thousands of 4 character combinations that can generate the same checksum values.

When you hit a potential server with such URLs, it should return `HTTP 200 OK` response with a small executable file.

Extracting metadata

Once you have the executable file, you can extract metadata from it.

The metadata itself is usually encrypted within the binary. There are several encryption layers, but they all come down to a simple XOR algorithm with a key that can usually be brute-forced.

Recent versions of Cobalt Strike can also use the Guardrails feature that encrypts config based on the following values:

- IP address
- Username
- Server name
- Domain

So if a machine does not match the values, it will not be able to decrypt the configuration and execute the payload.

Explaining the details of the encryption is beyond the scope of this article. Fortunately, there are tools to do so. So you don't have to worry about it for now.

My tool can crawl and validate millions of potential servers pretty fast. You can find it here: sigstrike. There is also alternative library called dissect.cobaltstrike. I used it to understand the whole encryption and decryption process.

If you want to understand how the decryption works, I would suggest reading the source code of the `dissect.cobaltstrike` library since Python code is easier to read.

Cobalt Strike configs

Each executable file of Cobalt Strike contains a configuration that is used to connect to the server. Since Cobalt Strike is also used by real malicious actors, extracting configs from it makes it even more fun. Sometimes, you can get a peek of big attacks that are happening right now.

Here is what the typical Cobalt Strike config looks like (I've replaced some unique data with dots):

```
{
   "SETTING_BOF_ALLOCATOR": "VirtualAlloc",
   "SETTING_C2_CHUNK_POST": 0,
   "SETTING_C2_POSTREQ": [
       ["_HEADER", "Content-Type: application/octet-stream"],
       ["BUILD", "id"],
       ["PARAMETER", "id"],
       ["BUILD", "output"],
       ["PRINT", true],
   ],
   "SETTING_C2_RECOVER": [["print", true]],
   "SETTING_C2_REQUEST": [["BUILD", "metadata"], ["BASE64", true], ["HEADER", "Cookie"]],
   "SETTING_C2_VERB_GET": "GET",
   "SETTING_C2_VERB_POST": "POST",
```

```
    "SETTING_CFG_CAUTION": 0,
    "SETTING_CLEANUP": 0,
    "SETTING_CRYPTO_SCHEME": 0,
    "SETTING_DOMAINS": "....,/ca",
    "SETTING_DOMAIN_STRATEGY": 0,
    "SETTING_DOMAIN_STRATEGY_FAIL_SECONDS": 4294967295,
    "SETTING_DOMAIN_STRATEGY_FAIL_X": 4294967295,
    "SETTING_DOMAIN_STRATEGY_SECONDS": 4294967295,
    "SETTING_EXIT_FUNK": 0,
    "SETTING_GARGLE_NOOK": 0,
    "SETTING_HOST_HEADER": "",
    "SETTING_HTTP_NO_COOKIES": 1,
    "SETTING_JITTER": 0,
    "SETTING_KILLDATE": 0,
    "SETTING_MAXGET": 1048576,
    "SETTING_MAX_RETRY_STRATEGY_ATTEMPTS": 0,
    "SETTING_MAX_RETRY_STRATEGY_DURATION": 0,
    "SETTING_MAX_RETRY_STRATEGY_INCREASE": 0,
    "SETTING_PORT": 5566,
    "SETTING_PROCINJ_ALLOCATOR": 0,
    "SETTING_PROCINJ_BOF_REUSE_MEM": 1,
    "SETTING_PROCINJ_EXECUTE": ["CreateThread", "SetThreadContext", "CreateRemoteThread", "RtlCreateUserThread"]
    "SETTING_PROCINJ_MINALLOC": 0,
    "SETTING_PROCINJ_PERMS": 64,
    "SETTING_PROCINJ_PERMS_I": 64,
    "SETTING_PROCINJ_STUB": "b50b86d7...4ad8d01781c",
    "SETTING_PROCINJ_TRANSFORM_X64": [["append", ""], ["prepend", ""]],
    "SETTING_PROCINJ_TRANSFORM_X86": [["append", ""], ["prepend", ""]],
    "SETTING_PROTOCOL": ["HTTP"],
    "SETTING_PROXY_BEHAVIOR": 2,
    "SETTING_PUBKEY": "51a8d41b43f9....9f9bae3fb9b82c43e40e7289",
    "SETTING_SLEEPTIME": 60000,
    "SETTING_SMB_FRAME_HEADER": "",
    "SETTING_SPAWNTO": "d7a9ca15a07f8....b63020da38aa16",
    "SETTING_SPAWNTO_X64": "%windir%\\sysnative\\rundll32.exe",
    "SETTING_SPAWNTO_X86": "%windir%\\syswow64\\rundll32.exe",
    "SETTING_SUBMITURI": "/submit.php",
    "SETTING_TCP_FRAME_HEADER": "",
    "SETTING_USERAGENT": "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0; BOIE9;ENIN)",
    "SETTING_WATERMARK": "....",
    "SETTING_WATERMARKHASH": "idv...PjBw=="
}
```


Here are explanations of some settings:

- `SETTING_DOMAINS` - lists server URLs that the beacon will connect to.

- `SETTING_PUBKEY` - a public key that is used to encrypt the communication between the beacon and the server.
- `SETTING_SPAWNTO_...` - the path to the executable that will be used to spawn the beacon.
- `SETTING_USERAGENT` - the user agent that will be used to identify the beacon.

Scanning your first data

You can use the following Python script to obtain a sample of IPs from the drb-ra source:

```
 import json

import requests

URL = "https://raw.githubusercontent.com/drb-ra/C2IntelFeeds/master/C2_configs/cobaltstrike-30day.json"


def fetch_data():
    r = requests.get(URL)
    r.raise_for_status()
    for line in r.text.splitlines():
        try:
            data = json.loads(line)["result"]
        except Exception:
            continue
        if not data["ip"] or not data["Port"]:
            continue
        try:
            host_str = data["ip"] + ":" + str(int(data["Port"]))
        except Exception:
            continue

        yield "http://%s/" % host_str
        yield "https://%s/" % host_str


def write_to_file(data, output_path):
    with open(output_path, mode="wt") as fout:
        for item in data:
            fout.write(item + "\n")


if __name__ == "__main__":
    write_to_file(fetch_data(), "input_30d.json")
```

It will store the URLs in the `input_30d.json` file.

You can then rescan those URLs to find active servers.

First, you need to install the `sigstrike` tool. You can install it via `cargo` (Rust package) or `pip` (Python package):

```
 cargo install sigstrike # Rust
pip install sigstrike # Python 3.9+
```

Then, you can run the following command to crawl the URLs and extract metadata:

```
 sigstrike crawl --input-path input_30d.json --output-path output.json --max-concurrent 1000
Crawl Summary:
  Total URLs processed: 1936
  Found: 301
  Failed: 1635
  Non-matching content type/status: 70
  Unreachable: 1565
```

The result will be stored in the `output.json` file.

Cobalt Strike servers tend to die/go offline quickly, so it's completely normal to have a lot of unreachable servers from the past 30-day scans.

There are services that send abuse reports to hosting providers when they detect Cobalt Strike.

When doing scans, it's better to perform them from an empty VPS server since you are dealing with potentially malicious infrastructure.

Next Steps

It's interesting to see how big companies or malicious actors are using Cobalt Strike. The good thing is that you don't have to be a security expert to give it a try and play with the data.

Professional read teams or experienced threat actors use various techniques to hide their Cobalt Strike servers. It's called Operational Security (OpSec).

In reality, you can still find high-profile servers since people are sometimes lazy and make mistakes.

There are numerous reports on the internet where almost default Cobalt Strike configurations were used in real-world and sophisticated attacks.

Threat hunting is a never-ending process, and there are always new techniques to find more servers.

For example, the scanning process can be improved by:

- Using more ports, especially randomized ports.
- Scanning domains

- Ignoring main page responses and hitting every possible web server with a GET request.

If you don't have the capability to scan the internet, you have to be creative and clever to find more stuff.

Threat hunting is not limited to Cobalt Strike, a similar approach is used to find C2 servers of various malware families.

The typical process for every malware family is the same:

- Obtain a sample of the running malware server
- Find a way to detect it by writing a signature or a query. This can be:
    - HTTP response signature (e.g., 404 Not Found with zero content length)
    - Certificate fingerprint
    - JARM fingerprint
    - JA4x fingerprint.
    - TCP response fingerprint
    - SSH fingerprint
    - Patterns in domain names
- Scan the internet for it or use search engines to find it.

Source: https://rushter.com/blog/threat-hunting-cobalt-strike/