Hotcobalt – New Cobalt Strike DoS Vulnerability That Lets You Halt Operations

labs.sentinelone.com/hotcobalt-new-cobalt-strike-dos-vulnerability-that-lets-you-halt-operations/

Gal Kristal

SentinelLABS
Hotcobalt - New Cobalt Strike DoS Vulnerability That Lets You Halt Operations
By Gal Kristal
READ BLOG >

Executive Summary

- Versions 4.2 and 4.3 of Cobalt Strike's server contain multiple Denial of Service vulnerabilities (CVE-2021-36798).
- The vulnerabilities can render existing Beacons unable to communicate with their C2 server, prevent new beacons from being installed, and have the potential to interfere with ongoing operations.
- We have released a new Python library to help generically parse Beacon communication in order to help the research security community.

Introduction

Cobalt Strike is one of the most popular attack frameworks designed for Red Team operations. At the same time, many APTs and malicious actors also use it.

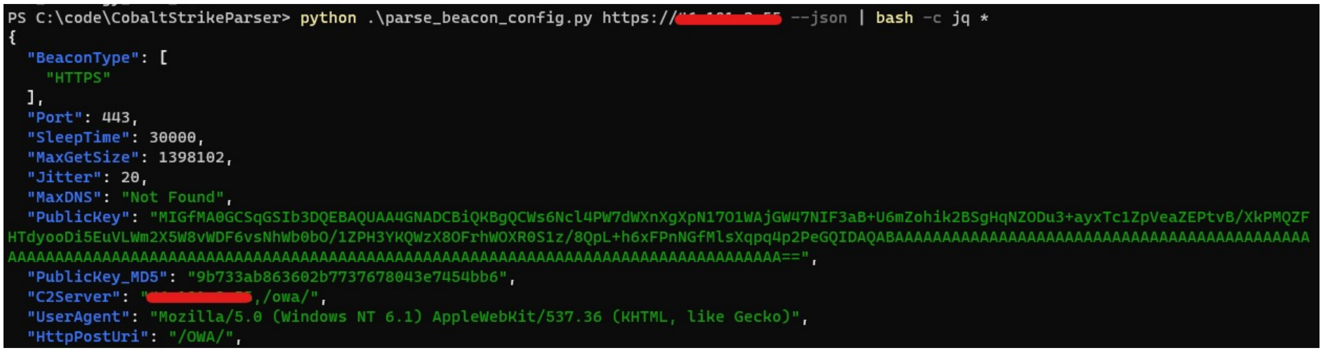
SentinelOne has seen numerous attacks involving Cobalt Strike Beacons across our customer base. SentinelOne detects Cobalt Strike Beacon and we are constantly rolling out new ways to detect modifications or novel ways to load Beacon in memory.

Given its rampant adoption by red teams and attackers alike, we wanted to better understand the operational security of Cobalt Strike. This led us to discover the vulnerabilities reported in CVE-2021-36798 and which we describe below.

Beacon Communications

To understand the vulnerabilities we found, we will briefly cover how Cobalt Strike Beacon communication works.

The first time the Cobalt Strike server runs, it creates randomly generated RSA keys, private and public, stored in a file named ".Cobalt Strike.beacon_keys". Every Beacon stager has the public key embedded in it.



```
PS C:\code\CobaltStrikeParser> python .\parse_beacon_config.py https://[redacted] --json | bash -c jq *
{
  "BeaconType": [
    "HTTPS"
  ],
  "Port": 443,
  "SleepTime": 30000,
  "MaxGetSize": 1398102,
  "Jitter": 20,
  "MaxDNS": "Not Found",
  "PublicKey": "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCWs6Ncl4PW7dWXnXgXpN17O1WAjGW47NIF3aB+U6mZohik2BSgHqNZODu3+ayxTc1ZpVeaZEPtvB/XkPMQZF
HTdyooDi5EuVLWm2X5W8vWDF6vsNhWb0bO/1ZPH3YKQWzX8OFrhWOXR0S1z/8QpL+h6xFPnNGfMlsXqpq4p2PeGQIDAQABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA==",
  "PublicKey_MD5": "9b733ab863602b7737678043e7454bb6",
  "C2Server": "[redacted],/owa/",
  "UserAgent": "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko)",
  "HttpPostUri": "/OWA/",
```

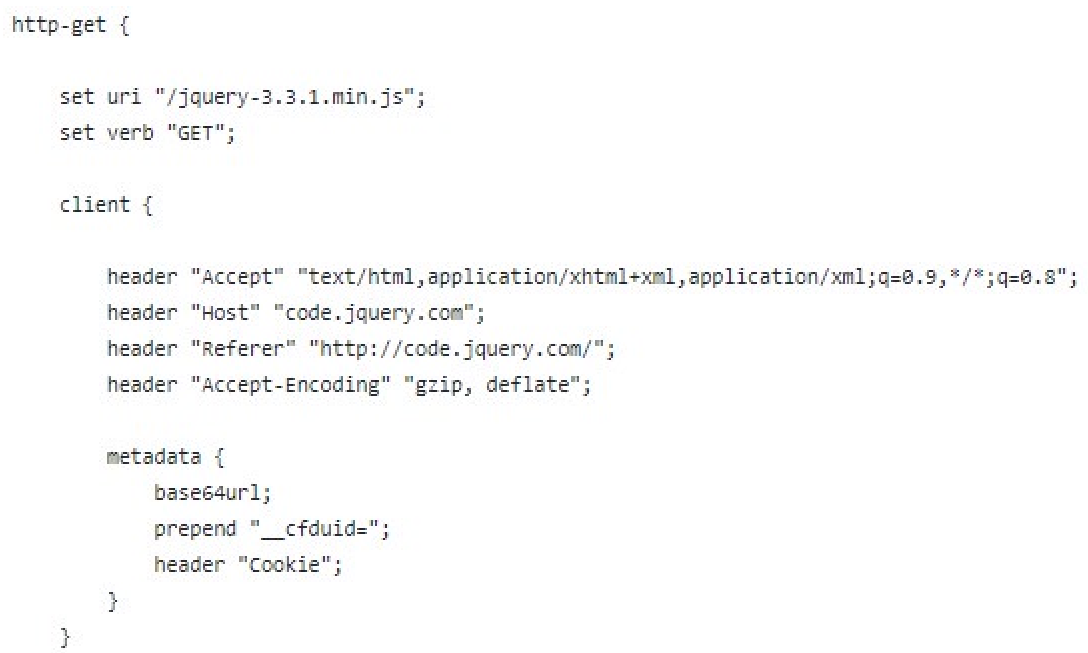
We can get the Beacon's public RSA key by parsing its configuration
When a Beacon stager runs, it gathers information about the computer it is running on (CPU architecture, keyboard layout, internal IP, etc.), encrypts that info using the public key, and sends it to the server in an HTTP GET request. We will refer to that part as "Beacon registration".

After the Beacon has registered with the server, the attacker can interact with the Beacon. From this point, the Beacon works by receiving and replying to "tasks". Tasks can, for example, be used to get a process list, run a command, conduct lateral movement, and many other things of interest to the attacker.

Receiving tasks generally happens over HTTP GET requests and the Beacon replies with the task data over HTTP POST requests. Tasks are encrypted using an AES key sent by the Beacon in the registration request. The entire communication flow is explained in the official documentation, but the outline above should suffice for what follows.

One of the most famous features of Cobalt Strike is its Malleable C2. In short, this feature lets the attacker encode ("transform" in Cobalt's language) all the beacon's HTTP communications. The entire process described above is wrapped in the chosen Malleable profile's transformation steps, which are also embedded in the stager itself.

Below is an example of a popular Malleable C2 profile that masquerades traffic as a normal request for the jquery code (source):



```
http-get {

    set uri "/jquery-3.3.1.min.js";
    set verb "GET";

    client {

        header "Accept" "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8";
        header "Host" "code.jquery.com";
        header "Referer" "http://code.jquery.com/";
        header "Accept-Encoding" "gzip, deflate";

        metadata {
            base64url;
            prepend "__cfduid=";
            header "Cookie";
        }
    }
```

An example of a popular Malleable C2 profile

Vulnerabilities

First, it should be noted that there was already one known vulnerability in Cobalt Strike that was previously reported. A great write-up written by nccgroup is worth reading for a more in-depth understanding of Beacon’s communication internals. In practice, that vulnerability allowed for remote code execution on the server.

We’re not interested in remote code execution vulnerability here as it would be overkill for our purposes. Considering that the server’s code is written in Java and isn’t very large, it wasn’t too hard to find bugs there.

For example, in the Screenshot and Keylogger task replies, there’s an interesting behavior when reading the reply’s data:

```
public void process_beacon_callback_decrypted(final String beaconID, final byte[]
responseBytes) {
...
// Sanity checks here
...
try {
final DataInputStream responeBytesStream = new DataInputStream(new
ByteArrayInputStream(responseBytes));
cmd = responeBytesStream.readInt();
if (cmd == 0) {...}
...
else if (cmd == 3) {
        final DataParser dp = new
DataParser(CommonUtils.readAll(responeBytesStream));
        dp.little();
        final byte[] scData = dp.readCountedBytes();    // Bug #1 here
        final int scDesktop = dp.readInt();
        final String scTitle = this.getCharsets().process(beaconID,
dp.readCountedBytes());
        final String process6 = this.getCharsets().process(beaconID,
dp.readCountedBytes());
        if (scData.length == 0) {
                output(BeaconOutput.Error(beaconID, "screenshot from desktop " +
scDesktop + " is empty"));
                return;
        }
        ...
        output(BeaconOutput.OutputB(beaconID, "received screenshot of " + scTitle + "
from " + process6 + " (" + CommonUtils.formatSize(scData.length) + ")"));
        ...
}}}
```

In this example, we see the parsing of a screenshot task reply. To read the screenshot's data, it calls the function readCountedBytes, which reads an integer from the first four bytes of the data and treats it as the screenshot's size without any sanity checks.

Then, before reading the screenshot's data, it allocates a buffer big enough to hold it:

```
byte[] array = new byte[ReplySize];
```

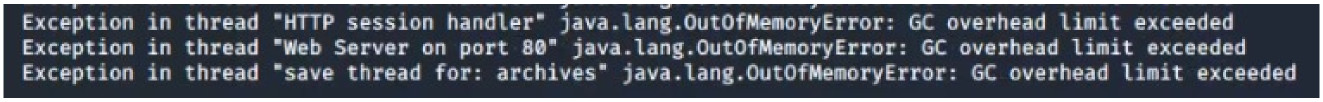
By manipulating the screenshot's size we can make the server allocate an arbitrary size of memory, the size of which is totally controllable by us. However, in order to trigger this piece of code, we need to be able to talk to the server like a Beacon.

By combining all the knowledge of Beacon communication flow with our configuration parser, we have all we need to fake a Beacon.

We've published a POC python script that does just that: it parses a Beacon's configuration and uses the information stored in it to register a new random Beacon on the server. After registering the Beacon, it's pretty trivial to use the primitive found above to iteratively send fake task replies that squeeze every bit of available memory from the C2's web server thread:

```
size = 1000000000
while True:
    try:
        if size < 20:
            break
        send_task(fake_beacon, size)
    except KeyboardInterrupt:
        break
    except Exception as e:
        size = size // 2
```

This leads to the crashing of the server's web thread that handles HTTP stagers and Beacon communication:



```
Exception in thread "HTTP session handler" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "Web Server on port 80" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "save thread for: archives" java.lang.OutOfMemoryError: GC overhead limit exceeded
```

Crashing the server's web thread

This would allow an attacker to cause memory exhaustion in the Cobalt Strike server (the "Teamserver") making the server unresponsive until it's restarted. This means that live Beacons cannot communicate to their C2 until the operators restart the server.

Restarting, however, won't be enough to defend against this vulnerability as it is possible to repeatedly target the server until it is patched or the Beacon's configuration is changed.

Either of these will make the existing live Beacons obsolete as they'll be unable to communicate with the server until they're updated with the new configuration. Therefore, this vulnerability has the potential to severely interfere with ongoing operations.

Although used every day for malicious attacks, Cobalt Strike is ultimately a legitimate product, so we have disclosed these issues responsibly to HelpSystems and they have fixed the vulnerabilities in the last release.

Utilities

On our Cobalt Strike parser repository, we've added new modules and code examples that implement:

- Parsing of a Beacon's embedded Malleable profile instructions
- Parsing of a Beacon's configuration directly from an active C2 (like the popular nmap script)
- Basic code for communicating with a C2 as a fake Beacon

Other than registering a fake Beacon with the server, the code we are releasing makes it easier to parse captured Beacon communications in a generic way.

Let’s take, for example, a case of a captured unencrypted Beacon communication from malware-traffic-analysis and decode it using the new communication module:

```
from urllib import parse
from pcaper import PcapParser
from parse_beacon_config import *
from comm import *

conf = cobaltstrikeConfig(r"beacon.bin").parse_config()
pparser = PcapParser()
reqs = pparser.read_pcap({'input': r"2019-07-25-Hancitor-style-Amadey-with-Pony-and-
Cobalt-Strike.pcap"})

t = Transform(conf['HttpPost_Metadata'])
for req in reqs:
        if conf['HttpPostUri'] in req.uri:
                params = {k: v[0] for k, v in
parse.parse_qs(parse.urlsplit(req.uri).query).items()}
                print('nnFound beacon reply:n', t.decode(req.body, req.headers,
params)[1])
```

Output:

```
...
Found beacon reply:
 ♠r↓10.7.25.101:445 (platform: 500 version: 6.1 name: HIDDENROAD-PC domain:
WORKGROUP)
Scanner module is complete
"))

Found beacon reply:
 ☺►[*] Wrote hijack DLL to 'C:UsersSARAH~1.RUTAppDataLocalTemp745f.dll'
[+] Privileged file copy success! C:WindowsSystem32sysprepCRYPTBASE.dll
[+] C:WindowsSystem32sysprepsysprep.exe ran and exited.
[*] Cleanup successful
...
```

It parses the Malleable C2 instructions embedded in the Beacon’s configuration and uses it to decode Beacon replies from the captured HTTP requests.

There’s a lot that can be done with this new communication library and it will be interesting to see what other researchers from the community will do with it.

Conclusion

Research into attack frameworks like Cobalt Strike and Cellebrite is still a niche area. We hope that this research and the tools we have released help to further encourage research into the robustness of attack frameworks and expand the range of available options when facing their consistent abuse.

Disclosure Timeline

We would like to thank HelpSystems for their approach to our disclosure and for remediating the vulnerabilities.

04/20/2021 - Initial contact with HelpSystems for issue disclosure.
04/22/2021 - Issue details disclosed to HelpSystems.
04/23/2021 - HelpSystems confirmed the issue and asked for an extension until August 3rd.
04/28/2021 - SentinelOne accepted the extension.
07/18/2021 - Submitted CVE request to MITRE.
07/19/2021 - CVE-2021-36798 was assigned and reserved for the specified issue.
08/02/2021 - SentinelOne shared the publication date and post for review.
08/02/2021 - HelpSystems reviewed and confirmed the post for publication.
08/04/2021 - HelpSystems released Cobalt Strike 4.4, which contains a fix for CVE-2021-36798.

All issues found by SentinelOne are disclosed to the relevant third party according to our Responsible Disclosure Policy for Third Parties.