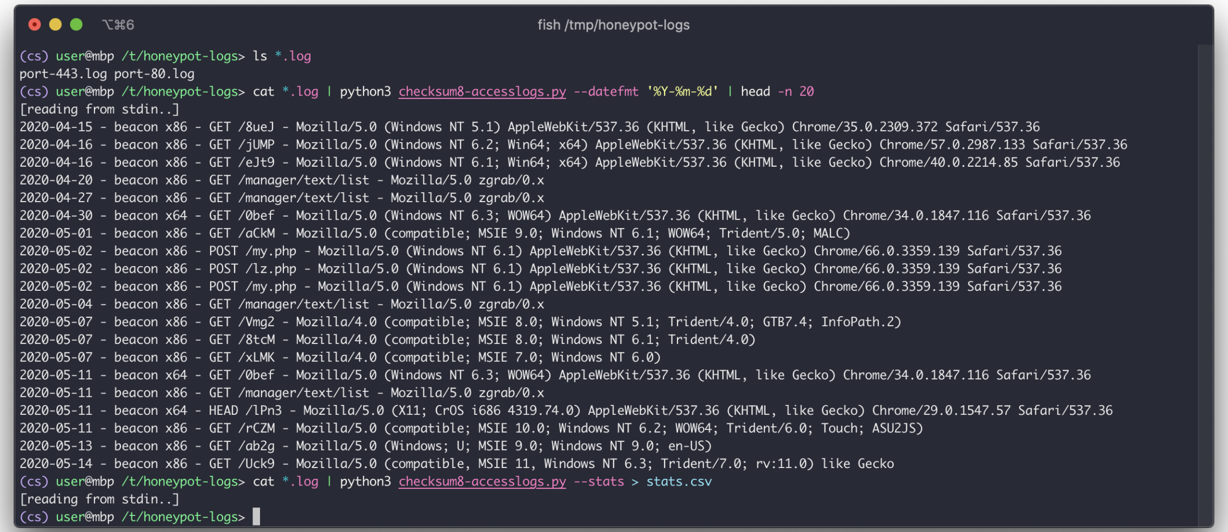
Mining data from Cobalt Strike beacons

research.nccgroup.com/2022/03/25/mining-data-from-cobalt-strike-beacons/

March 25, 2022



```
fish /tmp/honeypot-logs
(cs) user@mbp /t/honeypot-logs> ls *.log
port-443.log port-80.log
(cs) user@mbp /t/honeypot-logs> cat *.log | python3 checksum8-accesslogs.py --datefmt '%Y-%m-%d' | head -n 20
[reading from stdin..]
2020-04-15 - beacon x86 - GET /8ueJ - Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.2309.372 Safari/537.36
2020-04-16 - beacon x86 - GET /jUMP - Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36
2020-04-16 - beacon x86 - GET /eJt9 - Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.85 Safari/537.36
2020-04-20 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-04-27 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-04-30 - beacon x64 - GET /0bef - Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36
2020-05-01 - beacon x86 - GET /aCkM - Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; MALC)
2020-05-02 - beacon x86 - POST /my.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-02 - beacon x86 - POST /lz.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-02 - beacon x86 - POST /my.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-04 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-05-07 - beacon x86 - GET /Vmg2 - Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; InfoPath.2)
2020-05-07 - beacon x86 - GET /8tcM - Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
2020-05-07 - beacon x86 - GET /xLMK - Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
2020-05-11 - beacon x64 - GET /0bef - Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36
2020-05-11 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-05-11 - beacon x64 - HEAD /lPn3 - Mozilla/5.0 (X11; CrOS i686 4319.74.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.57 Safari/537.36
2020-05-11 - beacon x86 - GET /rCZM - Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; Touch; ASU2JS)
2020-05-13 - beacon x86 - GET /ab2g - Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
2020-05-14 - beacon x86 - GET /Uck9 - Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
(cs) user@mbp /t/honeypot-logs> cat *.log | python3 checksum8-accesslogs.py --stats > stats.csv
[reading from stdin..]
(cs) user@mbp /t/honeypot-logs>
```

Since we published about identifying Cobalt Strike Team Servers in the wild just over three years ago, we've collected over 128,000 beacons from over 24,000 active Team Servers. Today, RIFT is making this extensive beacon dataset publicly available in combination with the open-source release of `dissect.cobaltstrike`, our Python library for studying and parsing Cobalt Strike related data.

The published dataset contains historical beacon metadata ranging from 2018 to 2022. This blog will highlight some interesting findings you can extract and query from this extensive dataset. We encourage other researchers also to explore the dataset and share exciting results with the community.

Cobalt Strike Beacon dataset

The dataset `beacons.jsonl.gz` is a GZIP compressed file containing 128,340 rows of beacon metadata as JSON-lines. You can download it from the following repository and make sure to also check out the accompanying Jupyter notebook:

https://github.com/fox-it/cobaltstrike-beacon-data

The dataset spans almost four years of historical Cobalt Strike beacon metadata from July 2018 until February 2022. Unfortunately, we lost five months' worth of data in 2019 due to archiving issues. In addition, the dataset mainly focuses on x86 beacons collected from active Team Servers on HTTP port 80, 443 and DNS; therefore, it does not contain any beacons from other sources, such as VirusTotal.

The beacon payloads themselves are not in the dataset due to the size. Instead, the different beacon configuration settings are stored, including other metadata such as:

- Date the beacon was collected and from which IP address and port
- GeoIP + ASN metadata
- TLS certificate in DER format
- PE information (timestamps, magic_mz, magic_pe, stage_prepend, stage_append)
- If the payload was XOR encoded, and which XOR key was used for config obfuscation
- The raw beacon configuration bytes; handy if you want to parse the beacon config manually. (e.g. using `dissect.cobaltstrike` or another parser of choice)

While there are some trivial methods to identify cracked/pirated Cobalt Strike Team Servers from the beacon payload, it's difficult to tell for the non-trivial ones. Therefore the dataset is unfiltered, full disclosure and contains all beacons we have collected.

Cobalt Strike Team Servers that are properly hidden or have payload staging disabled are, of course, not included. That means Red Teams (and sadly threat actors) with good OPSEC have nothing to worry about being present in this dataset. 🙂

Beacons, and where to find them

The Cobalt Strike beacons were acquired by first identifying Team Servers on the Internet and then downloading the beacon using a `checksum8` HTTP request. This method is similar to how the company behind Cobalt Strike did their own Cobalt Strike Team Server Population Study back in 2019.

Although the *anomalous space* fingerprint we used for identification was since fixed, we found other reliable methods for identifying Team Servers. In addition, we are sure that the original author of Cobalt Strike intentionally left some indicators in there to help blue teams. For that, we are grateful and hope this doesn't change in the future.

Via our honeypots, we can tell that RIFT is not alone in mining beacons. Everyone is doing it now. The increased blog posts and example scripts on how to find Cobalt Strike probably attribute to this, next to the increased popularity of Cobalt Strike itself, of course.

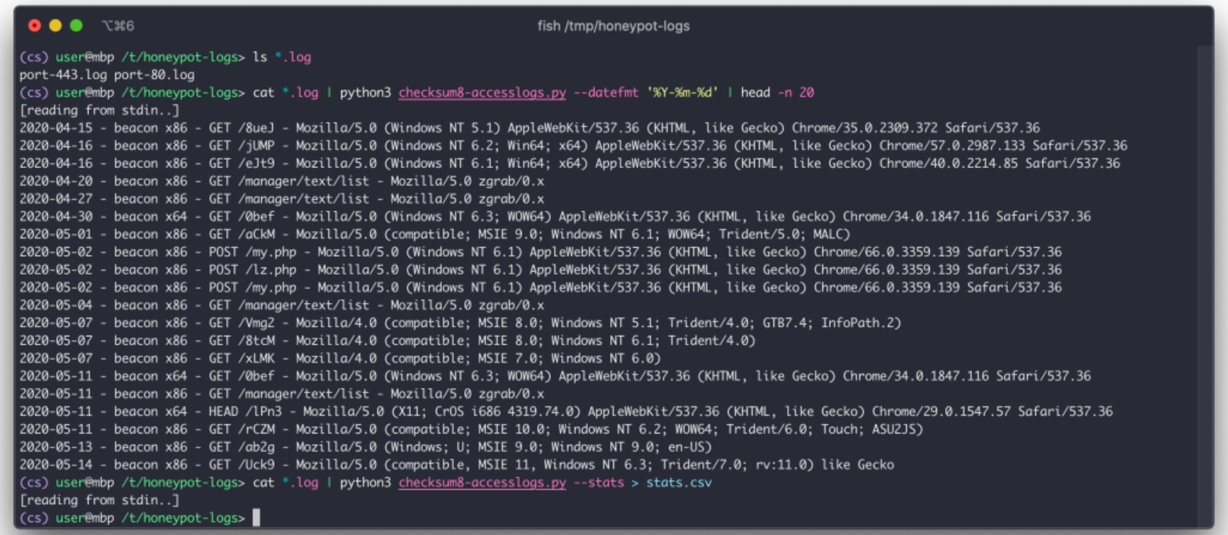
The development of increased scanning for Cobalt Strike is fascinating to witness, including the different techniques and shotgun approaches for identifying Team Servers and retrieving beacons. Some even skip the identification part and go directly for the beacon request! As you can imagine, this can be noisy, which surely doesn't go unnoticed for some threat actors.

If you run a public-facing web server, you can easily verify this increased scanning by checking the HTTP access logs for common `checksum8` like requests, for example, by using the following *grep* command:

```
$ zgrep -hE "GET /[a-zA-Z0-9]{4} HTTP" /var/log/nginx/*.gz
172.x.x.x - - [23/Feb/2021:18:xx:08 +0100] "GET /0bef HTTP/1.0” 404 162 "-" "-"
172.x.x.x - - [24/Feb/2021:09:xx:40 +0100] "GET /0bef HTTP/1.0” 404 162 "-" "-"
139.x.x.x - - [25/Feb/2021:05:xx:39 +0100] "GET /bag2 HTTP/1.1” 404 193 "-" "-"
134.x.x.x - - [25/Feb/2021:15:xx:12 +0100] "GET /ab2g HTTP/1.1” 400 166 "-" "-"
134.x.x.x - - [25/Feb/2021:15:xx:22 +0100] "GET /ab2h HTTP/1.1” 400 166 "-" "-"
```

The requests shown above are `checksum8` requests (for x86 and x64 beacons), hitting a normal webserver hosting a real website in February 2021.

You can also use our `checksum8-accesslogs.py` script, which does all these things in one script and more accurately by verifying the `checksum8` value. It can also output statistics. Here is an example of outputting the x86 and x64 beacon HTTP requests hitting one of our honeypots and generating the statistics:



```
(cs) user@mbp /t/honeypot-logs> ls *.log
port-443.log port-80.log
(cs) user@mbp /t/honeypot-logs> cat *.log | python3 checksum8-accesslogs.py --datefmt '%Y-%m-%d' | head -n 20
[reading from stdin..]
2020-04-15 - beacon x86 - GET /8ueJ - Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.2309.372 Safari/537.36
2020-04-16 - beacon x86 - GET /jUMP - Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36
2020-04-16 - beacon x86 - GET /eJt9 - Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/40.0.2214.85 Safari/537.36
2020-04-20 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-04-27 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-04-30 - beacon x64 - GET /0bef - Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36
2020-05-01 - beacon x86 - GET /aCkM - Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; MALC)
2020-05-02 - beacon x86 - POST /my.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-02 - beacon x86 - POST /lz.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-02 - beacon x86 - POST /my.php - Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.139 Safari/537.36
2020-05-04 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-05-07 - beacon x86 - GET /Vmg2 - Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB7.4; InfoPath.2)
2020-05-07 - beacon x86 - GET /8tcM - Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
2020-05-07 - beacon x86 - GET /xLMK - Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
2020-05-11 - beacon x64 - GET /0bef - Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.116 Safari/537.36
2020-05-11 - beacon x86 - GET /manager/text/list - Mozilla/5.0 zgrab/0.x
2020-05-11 - beacon x64 - HEAD /lPn3 - Mozilla/5.0 (X11; CrOS i686 4319.74.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.57 Safari/537.36
2020-05-11 - beacon x86 - GET /rCZM - Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; WOW64; Trident/6.0; Touch; ASU2JS)
2020-05-13 - beacon x86 - GET /ab2g - Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
2020-05-14 - beacon x86 - GET /Uck9 - Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
(cs) user@mbp /t/honeypot-logs> cat *.log | python3 checksum8-accesslogs.py --stats > stats.csv
[reading from stdin..]
(cs) user@mbp /t/honeypot-logs>
```

checksum8-accesslog.py script finds possible Beacon stager requests in access logs
In the output, you can also see the different beacon scanning techniques being used, which we will leave as an exercise for the reader to figure out.

We can see an apparent increase in beacon scanning on one of our honeypots by plotting the statistics:

So if you ever wondered why people are requesting these weird four-character URLs (or other strange-looking URLs) on your web server, check the `checksum8` value of the request, and you might have your answer.

We try to be a bit stealthier and won't disclose our fingerprinting techniques, as we also know threat actors are vigilant and, in the long run, will make it harder for everyone dealing with Threat Intelligence.

Cobalt Strike version usage over time

Because we have beacon metadata over multiple years, we can paint a pretty good picture of active Cobalt Strike servers on the Internet and which versions they were using at that time.

To extract the Cobalt Strike version data, we used the following two methods:

- Using the Beacon Setting constants
  When a new Cobalt Strike beacon configuration setting is introduced, the Setting constant is increased and then assigned. It's possible to deduce the version based on the highest available constant in the extracted beacon configuration.
- Using the PE export timestamp
  This is also documented by BlackBerry in their Finding Beacons In the Dark: A Guide to Cyber Threat Intelligence book and is a more accurate way of determining the exact version.

Our Python library `dissect.cobaltstrike` supports both methods for deducing version numbers and favours the PE export timestamp when available.

The dataset already contains the `beacon_version` field for your convenience and is based on the PE export timestamp. Using this field, we can generate the following graph showing the different Cobalt Strike versions used on the Internet over time:

We can see that in April 2021, there was quite a prominent peak of online Cobalt Strike servers and unique beacons, but we are not sure what caused this except that there was a 3% increase of modified (likely malicious) beacons that month.

The following percentage-wise graph shows a clearer picture of the adoption and popularity between the different versions over time:

We can see that Cobalt Strike 4.0 (released in December 2019) remained quite popular from January 2020 to January 2021.

Beacon watermark statistics

Since Cobalt Strike 3.10 (released December 2017), the beacons contain a setting called `SETTING_WATERMARK`. This watermark value should be unique per Cobalt Strike installation, as the license server issues this.

However, cracked/pirated versions usually patch this to a fixed value, making it easy to identify which beacons are more likely to be malicious (i.e. not a penetration tester). This likelihood aligns with our incident response engagements so far, where beacons related to the compromise used known-bad watermarks.

Note that requesting a trial or buying a legitimate copy of Cobalt Strike is difficult for malicious actors as every user is vetted and screened. Because of these measures, there is a high asking price for a Cobalt Strike copy on the dark web. For example, Conti invested $60.000 to acquire a valid copy of Cobalt Strike.

If you find a beacon with a watermark in this top 50, then it's most likely malicious!

Customized Beacons

While parsing collected beacons, we found that some were modified, for example, with a custom shellcode stub, non-default XOR keys or reassigned Beacon settings.

Therefore, the beacons with heavy customizations could not be dumped properly and are not included in the dataset.

The configuration block in the beacon payload is usually *obfuscated* using a single byte XOR key. Depending on the Cobalt Strike version, the default keys are 0x2e or 0x69.

The use of non-default XOR keys requires the user to modify the beacon and or Team Server, as it's not configurable by default. Here is an overview of seen XOR keys over the unique beacon dataset:

Using a custom XOR key makes you an outlier though, but it does protect you against some existing Cobalt Strike config dumpers. Our Python library `dissect.cobaltstrike` supports trying all XOR keys when the default XOR keys don't work. For example, you can pass the command line flag `--all-xor-keys` to the `beacon-dump` command.

Portable Executable artifacts

While most existing Cobalt Strike dumpers focus on the beacon settings, some settings from the Malleable C2 profiles will not end up in the embedded beacon config of the payload. For example, some Portable Executable (PE) settings in the Malleable C2 profile are applied directly to the beacon payload. Our Python library `dissect.cobaltstrike` supports extracting this information, and our dataset includes the following extracted PE header metadata:

- magic_mz — MZ header
- magic_pe — PE header
- pe_compile_stamp — PE compilation stamp
- pe_export_stamp — timestamp of the export section

- stage_prepend – (shellcode) bytes prepended to the start of the beacon payload
- stage_append — bytes appended to the end of the beacon payload

We created an overview of the most common `stage_prepend` bytes that are *all ASCII* bytes. These bytes are prepended in front of the MZ header, and has to be valid assembly code but resulting in a no-operation as it's executed as shellcode. Some are quite creative:

If we disassemble the example stage_prepend shellcode `JFIFJFIF` we can see that it increases the ESI and decreases the EDX registers and leaves it modified as a result; so it's not fully a no-operation shellcode but it most likely doesn't affect the staging process either.

```
$ echo -n JFIFJFIF | ndisasm -b 32 /dev/stdin
00000000  4A                dec edx
00000001  46                inc esi
00000002  49                dec ecx
00000003  46                inc esi
00000004  4A                dec edx
00000005  46                inc esi
00000006  49                dec ecx
00000007  46                inc esi
```

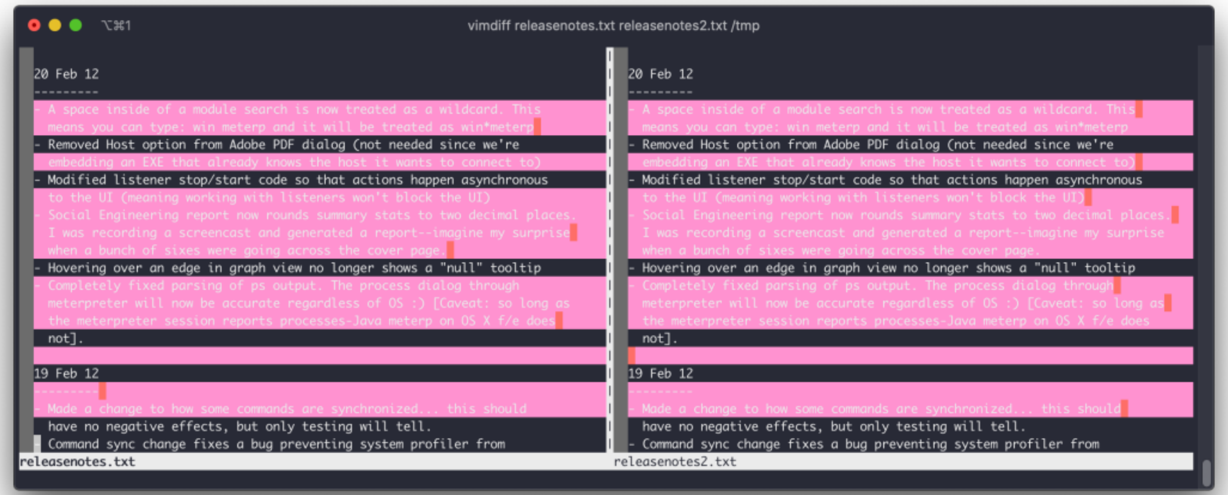
You can check our Jupyter notebook for an overview on the rest of the PE artifacts, such as `magic_mz` and `magic_pe`.

Watermarked releasenotes.txt using whitespace

The author of Cobalt Strike must really like spaces, after the *erroneous space* in the HTTP server header, there is now also a (repurposed) beacon setting called `SETTING_SPAWNTO` that is now populated with the MD5 hash of the file `releasenotes.txt` (or accidentally another file in the same directory if that matches the same `checksum8` value of 152 and filename length!).

The `releasenotes.txt` is automatically downloaded from the license server when you activate or update your Cobalt Strike server. To our surprise, we discovered that this file is most likely watermarked using whitespace characters thus making this file and MD5 hash unique per installation. The license server probably keeps track of all these uniquely generated files to help combat piracy and leaks of Cobalt Strike.

While this is pretty clever, we found that in some pirated beacons this field is all zeroes, or not available. Meaning they knew about this file and decided not to ship it in the pirated version or the field value was patched out. Nevertheless, this field is still useful for hunting or correlating beacons when it is available.


```
vimdiff releasenotes.txt releasenotes2.txt /tmp
```

```
20 Feb 12
---------
- A space inside of a module search is now treated as a wildcard. This
  means you can type: win meterp and it will be treated as win*meterp
- Removed Host option from Adobe PDF dialog (not needed since we're
  embedding an EXE that already knows the host it wants to connect to)
- Modified listener stop/start code so that actions happen asynchronous
  to the UI (meaning working with listeners won't block the UI)
- Social Engineering report now rounds summary stats to two decimal places.
  I was recording a screencast and generated a report--imagine my surprise
  when a bunch of sixes were going across the cover page.
- Hovering over an edge in graph view no longer shows a "null" tooltip
- Completely fixed parsing of ps output. The process dialog through
  meterpreter will now be accurate regardless of OS :) [Caveat: so long as
  the meterpreter session reports processes-Java meterp on OS X f/e does
  not].

19 Feb 12
---------
- Made a change to how some commands are synchronized... this should
  have no negative effects, but only testing will tell.
- Command sync change fixes a bug preventing system profiler from
releasenotes.txt
```

```
20 Feb 12
---------
- A space inside of a module search is now treated as a wildcard. This
  means you can type: win meterp and it will be treated as win*meterp
- Removed Host option from Adobe PDF dialog (not needed since we're
  embedding an EXE that already knows the host it wants to connect to)
- Modified listener stop/start code so that actions happen asynchronous
  to the UI (meaning working with listeners won't block the UI)
- Social Engineering report now rounds summary stats to two decimal places.
  I was recording a screencast and generated a report--imagine my surprise
  when a bunch of sixes were going across the cover page.
- Hovering over an edge in graph view no longer shows a "null" tooltip
- Completely fixed parsing of ps output. The process dialog through
  meterpreter will now be accurate regardless of OS :) [Caveat: so long as
  the meterpreter session reports processes-Java meterp on OS X f/e does
  not].

19 Feb 12
---------
- Made a change to how some commands are synchronized... this should
  have no negative effects, but only testing will tell.
- Command sync change fixes a bug preventing system profiler from
releasenotes2.txt
```

Note the subtle whitespace changes at the end of the lines between the two releasenotes.txt files.

Analyze Beacon payloads with dissect.cobaltstrike

We are also proud to open-source our Python library for dissecting Cobalt Strike, aptly named `dissect.cobaltstrike`. The library is available on PyPI and requires Python 3.6 or higher. You can use pip to install it:

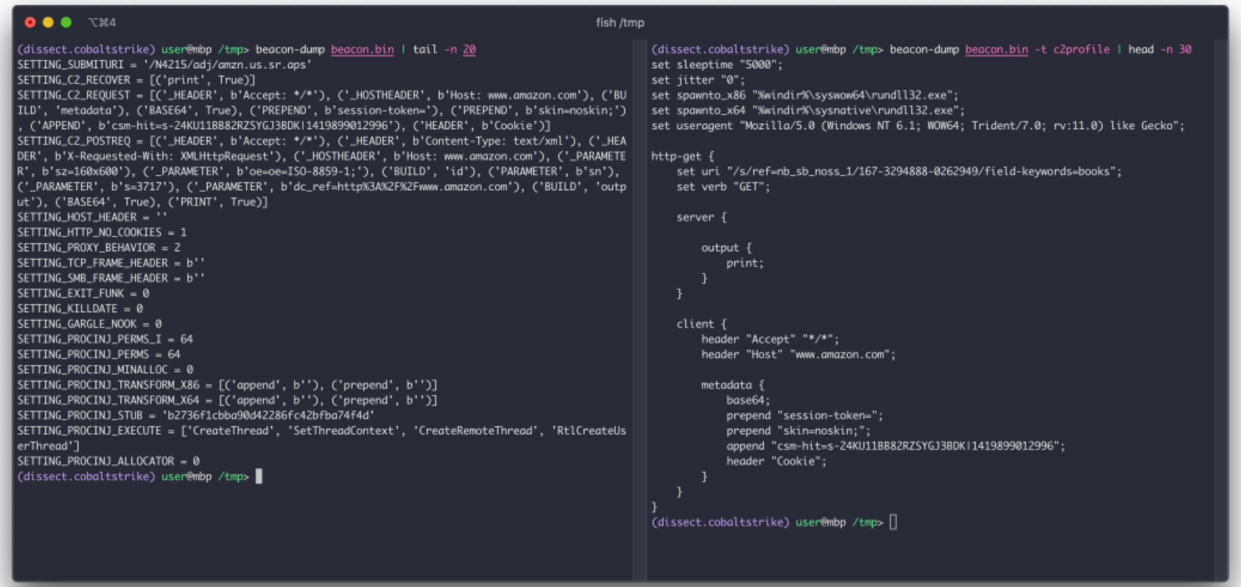
```
$ pip install dissect.cobaltstrike
```

The project's GitHub repository: https://github.com/fox-it/dissect.cobaltstrike

It currently installs three command line tools for your convenience:

- `beacon-dump` – used for dumping configuration from beacon payloads (also works on memory dumps)
- `beacon-xordecode` – a standalone tool for decoding xorencoded payloads
- `c2profile-dump` – use this to read and parse Malleable C2 profiles.

A neat feature of `beacon-dump` is to dump the beacon configuration back as it's Malleable C2 profile compatible equivalent:



```
(dissect.cobaltstrike) user@mbp /tmp> beacon-dump beacon.bin | tail -n 20
SETTING_SUBMITURI = '/N4215/adj/amzn.us.sr.aps'
SETTING_C2_RECOVER = [('print', True)]
SETTING_C2_REQUEST = [('_HEADER', b'Accept: */*'), ('_HOSTHEADER', b'Host: www.amazon.com'), ('BU
ILD', 'metadata'), ('BASE64', True), ('PREPEND', b'session-token='), ('PREPEND', b'skin=noskin;')
, ('APPEND', b'csm-hit=s-24KU11BB82RZSYGJ3BDK|1419899012996'), ('HEADER', b'Cookie')]
SETTING_C2_POSTREQ = [('_HEADER', b'Accept: */*'), ('_HEADER', b'Content-Type: text/xml'), ('_HEA
DER', b'X-Requested-With: XMLHttpRequest'), ('_HOSTHEADER', b'Host: www.amazon.com'), ('_PARAMETE
R', b'sz=160x600'), ('_PARAMETER', b'oe=oe=ISO-8859-1;'), ('BUILD', 'id'), ('PARAMETER', b'sn'),
('_PARAMETER', b's=3717'), ('_PARAMETER', b'dc_ref=http%3A%2F%2Fwww.amazon.com'), ('BUILD', 'outp
ut'), ('BASE64', True), ('PRINT', True)]
SETTING_HOST_HEADER = ''
SETTING_HTTP_NO_COOKIES = 1
SETTING_PROXY_BEHAVIOR = 2
SETTING_TCP_FRAME_HEADER = b''
SETTING_SMB_FRAME_HEADER = b''
SETTING_EXIT_FUNK = 0
SETTING_KILLDATE = 0
SETTING_GARGLE_NOOK = 0
SETTING_PROCINJ_PERMS_I = 64
SETTING_PROCINJ_PERMS = 64
SETTING_PROCINJ_MINALLOC = 0
SETTING_PROCINJ_TRANSFORM_X86 = [('append', b''), ('prepend', b'')]
SETTING_PROCINJ_TRANSFORM_X64 = [('append', b''), ('prepend', b'')]
SETTING_PROCINJ_STUB = 'b2736f1cbba90d42286fc42bfba74f4d'
SETTING_PROCINJ_EXECUTE = ['CreateThread', 'SetThreadContext', 'CreateRemoteThread', 'RtlCreateUs
erThread']
SETTING_PROCINJ_ALLOCATOR = 0
(dissect.cobaltstrike) user@mbp /tmp>
```

```
(dissect.cobaltstrike) user@mbp /tmp> beacon-dump beacon.bin -t c2profile | head -n 30
set sleeptime "5000";
set jitter "0";
set spawnto_x86 "%windir%\syswow64\rundll32.exe";
set spawnto_x64 "%windir%\sysnative\rundll32.exe";
set useragent "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko";

http-get {
    set uri "/s/ref=nb_sb_noss_1/167-3294888-0262949/field-keywords=books";
    set verb "GET";

    server {

        output {
            print;
        }
    }

    client {
        header "Accept" "*/*";
        header "Host" "www.amazon.com";

        metadata {
            base64;
            prepend "session-token=";
            prepend "skin=noskin;";
            append "csm-hit=s-24KU11BB82RZSYGJ3BDK|1419899012996";
            header "Cookie";
        }
    }
}
(dissect.cobaltstrike) user@mbp /tmp>
```

Dumping beacons settings as a Malleable C2 Profile
While these command line tools provide most of the boilerplate for working with Beacon payloads, you can also import the library in a script or notebook for more advanced use cases. See our notebook and documentation for some examples.

Closing thoughts

The beacon dataset has proved very useful to us, especially the historical aspect of the dataset is insightful during incident response engagements. We use the dataset daily, ranging from C2 infrastructure mapping, actor tracking, threat hunting, high-quality indicators, detection engineering and many more.

We hope this dataset and Python library will be helpful to the community as it is for us and are eager to see what kind of exciting things people will come up with or find using the data and tooling. What we have shown in this blog is only the tip of the iceberg of what you can uncover from beacon data.

Some ideas for the readers:

- Cluster beacon and C2 profile features using a clustering algorithm such as DBSCAN.
- Improve the classification of *malicious* beacons. You can find the current classification method in our notebook.
- Use the GeoIP ASN data to determine where the most malicious beacons are hosted.
- Analysis on the x509 certificate data, such as self-signed or not.
- Determine if a beacon uses domain fronting and which CDN.

All the statistics shown in this blog post can also be found in our accompanying Jupyter notebook including some more statistics and overviews not shown in this blog.

We also want to thank Rapid7 for the Open Data sets, without this data the beacon dataset would be far less complete!

Final links for convenience:

- Beacon dataset and notebook – https://github.com/fox-it/cobaltstrike-beacon-data
- `dissect.cobaltstrike` Python library – https://github.com/fox-it/dissect.cobaltstrike
- `dissect.cobaltstrike` Documentation – https://dissect-cobaltstrike.readthedocs.io