Cobalt Strike: Using Known Private Keys To Decrypt Traffic – Part 2

blog.nviso.eu/2021/10/27/cobalt-strike-using-known-private-keys-to-decrypt-traffic-part-2/

October 27, 2021

Blogpost series: Cobalt Strike: Decrypting Traffic
We decrypt Cobalt Strike traffic using one of 6 private keys we found.

In this blog post, we will analyze a Cobalt Strike infection by looking at a full packet capture that was taken during the infection. This analysis includes decryption of the C2 traffic.

If you haven't already, we invite you to read part 1 first: Cobalt Strike: Using Known Private Keys To Decrypt Traffic – Part 1.

For this analysis, we are using capture file 2021-02-02-Hancitor-with-Ficker-Stealer-and-Cobalt-Strike-and-NetSupport-RAT.pcap.zip, this is one of the many malware traffic capture files that Brad Duncan shares on his web site Malware-Traffic-Analysis.net.

We start with a minimum of knowledge: the capture file contains encrypted HTTP traffic of a Cobalt Strike beacon communicating with its team server.

If you want to know more about Cobalt Strike and its components, we highly recommend the following blog post.

First step: we open the capture file with Wireshark, and look for downloads of a full beacon by stager shellcode.

Although beacons can come in many forms, we can identify 2 major categories:

1. A small piece of shellcode (a couple of hundred bytes), aka the stager shellcode, that downloads the full beacon
2. The full beacon: a PE file that can be reflectively loaded

In this first step, we search for signs of stager shellcode in the capture file: we do this with the following display filter: *http.request.uri matches "/….$"*.

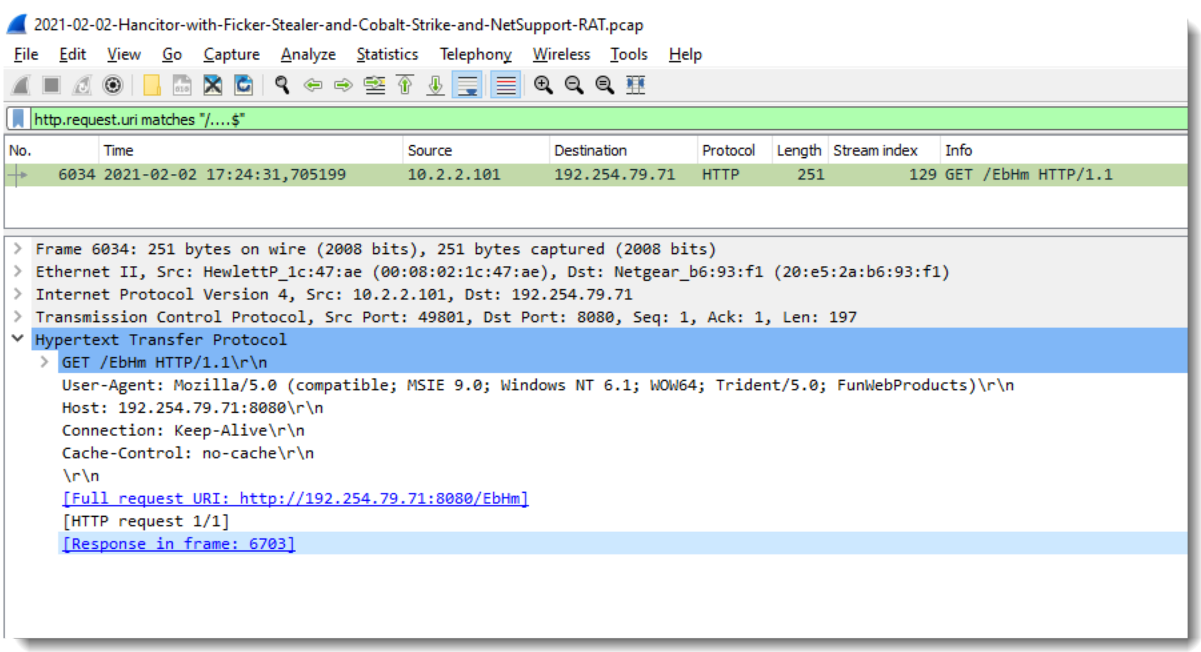


Figure 1: packet capture for Cobalt Strike traffic

We have one hit. The path used in the GET request to download the full beacon, consists of 4 characters that satisfy a condition: the byte-value of the sum of the character values (aka checksum 8) is a known constant. We can check this with the tool metatool.py like this:

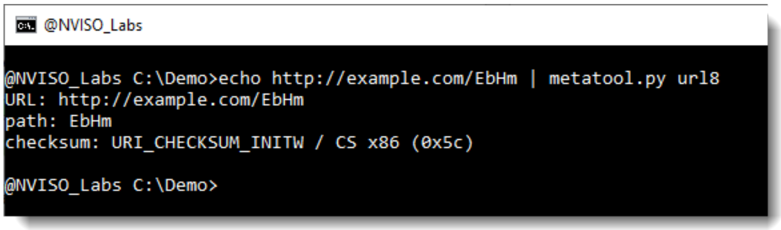


Figure 2: using metatool.py

More info on this checksum process can be found here.
The output of the tool shows that this is a valid path to download a 32-bit full beacon (CS x86).
The download of the full beacon is captured too:

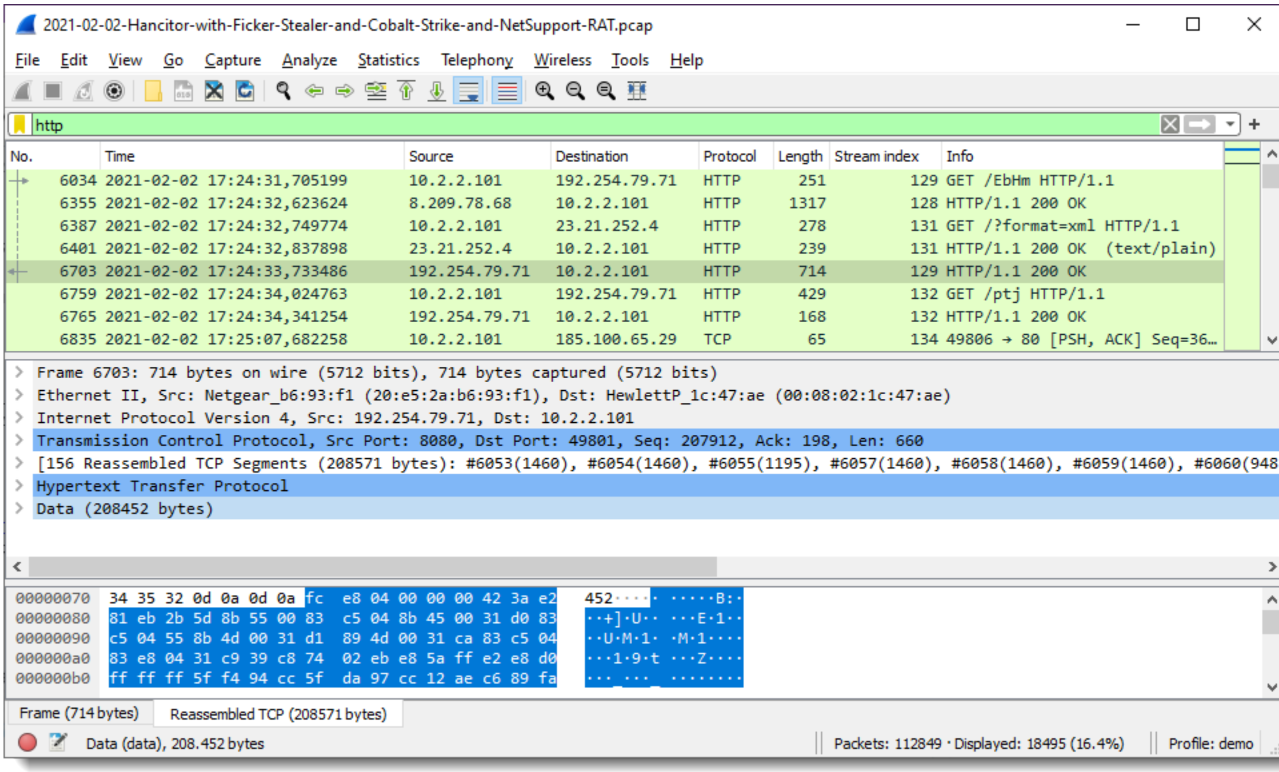


Figure 3: full beacon download
And we can extract this download:

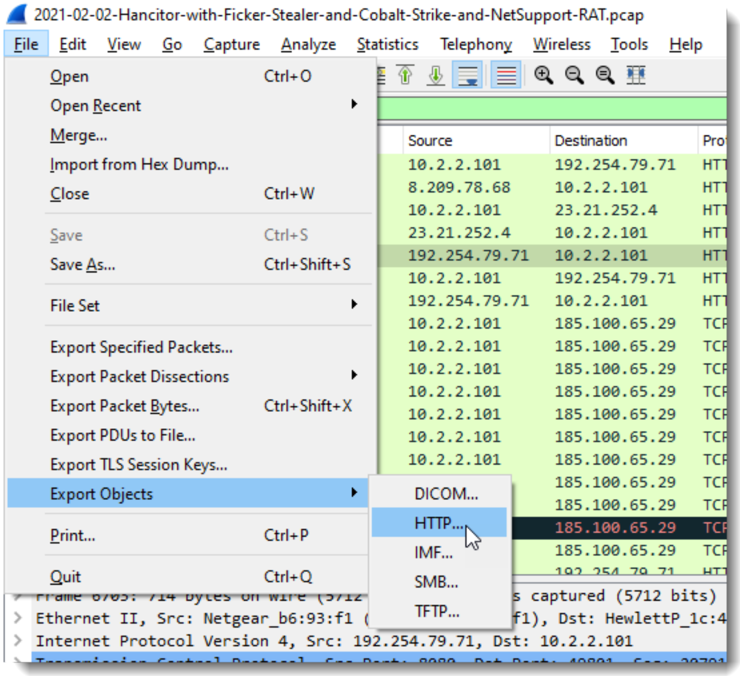


Figure 4: export HTTP objects

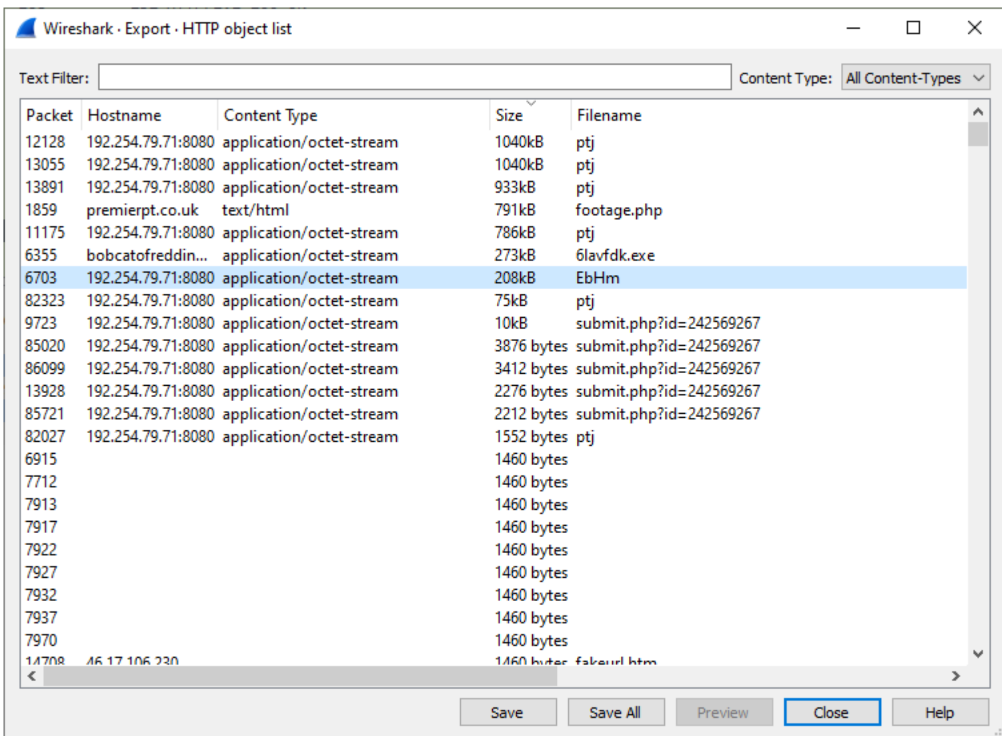


Figure 5: selecting download

EbHm for saving

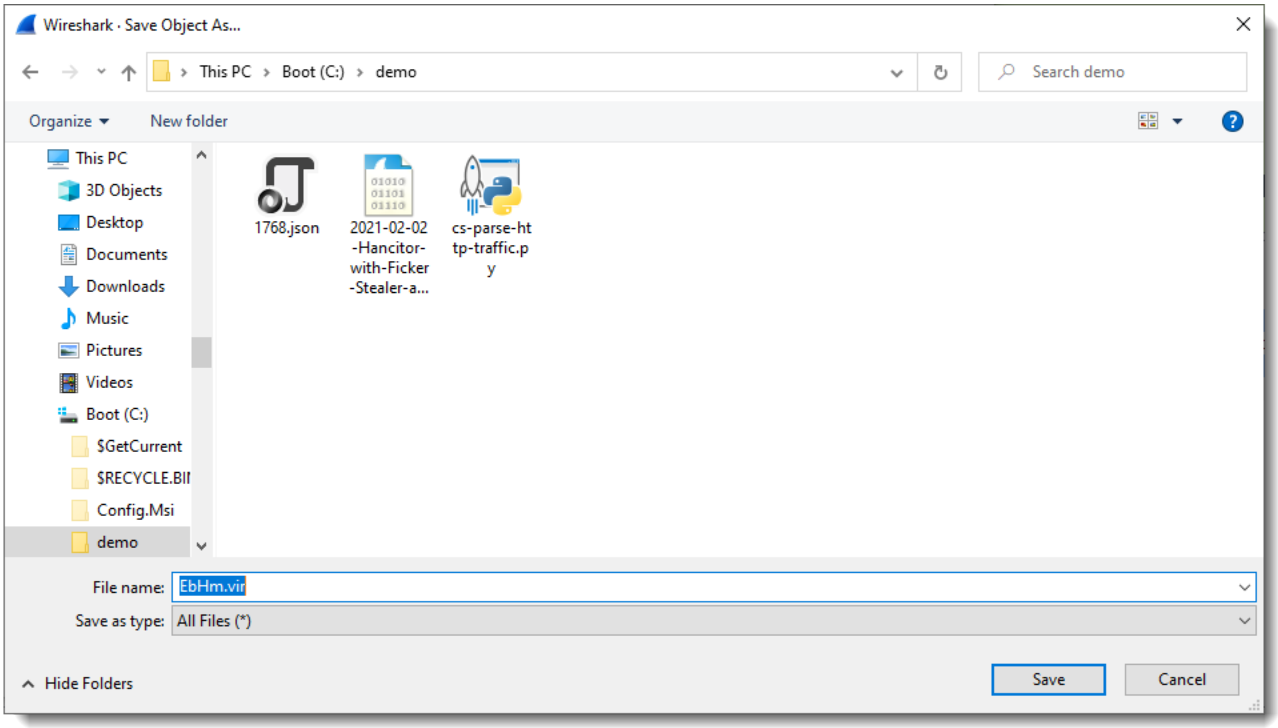


Figure 6: saving selected download to disk

Once the full beacon has been saved to disk as EbHm.vir, it can be analyzed with tool 1768.py. 1768.py is a tool that can decode/decrypt Cobalt Strike beacons, and extract their configuration. Cobalt Strike beacons have many configuration options: all these options are stored in an encoded and embedded table.

Here is the output of the analysis:

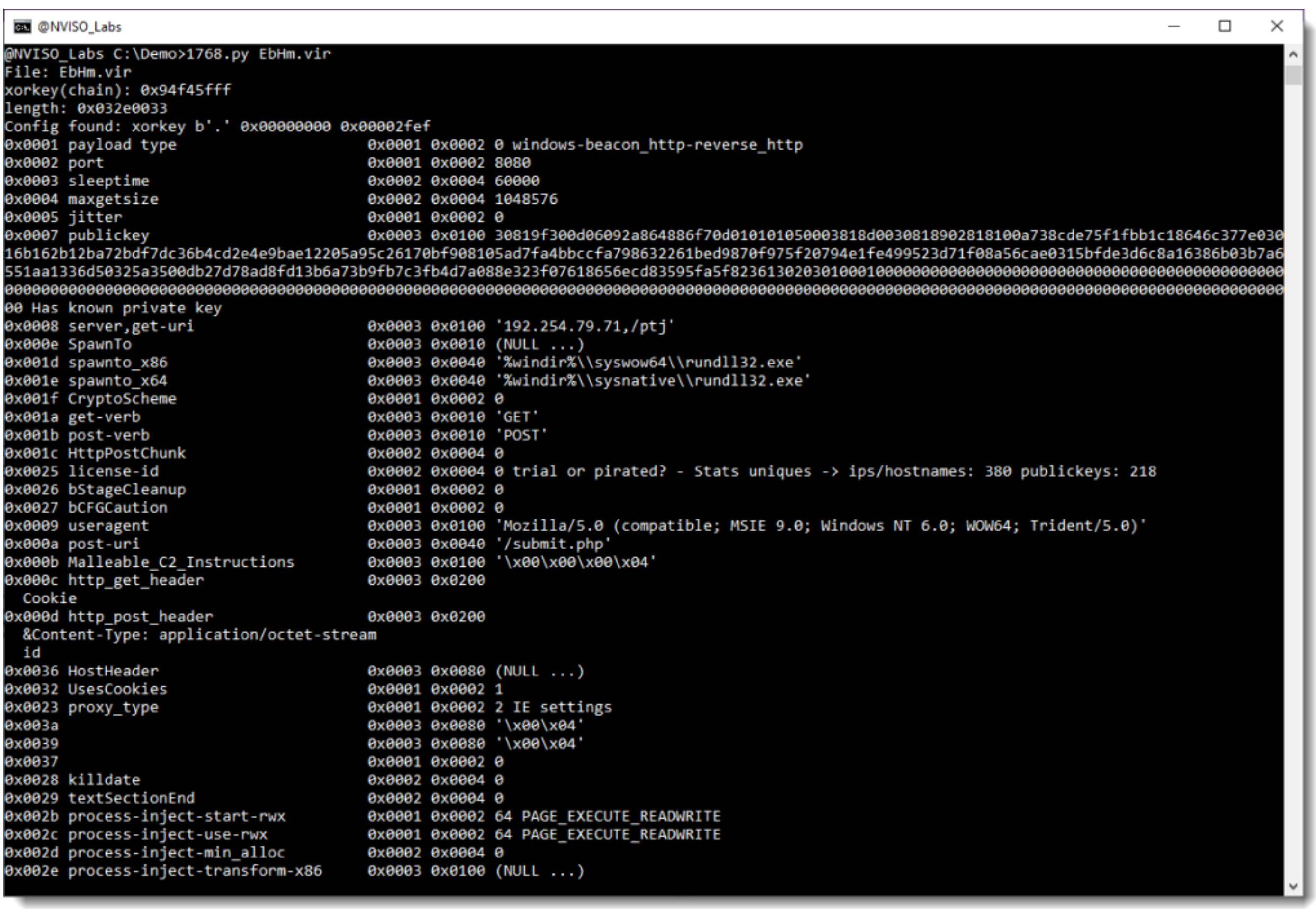


Figure 7: extracting beacon configuration

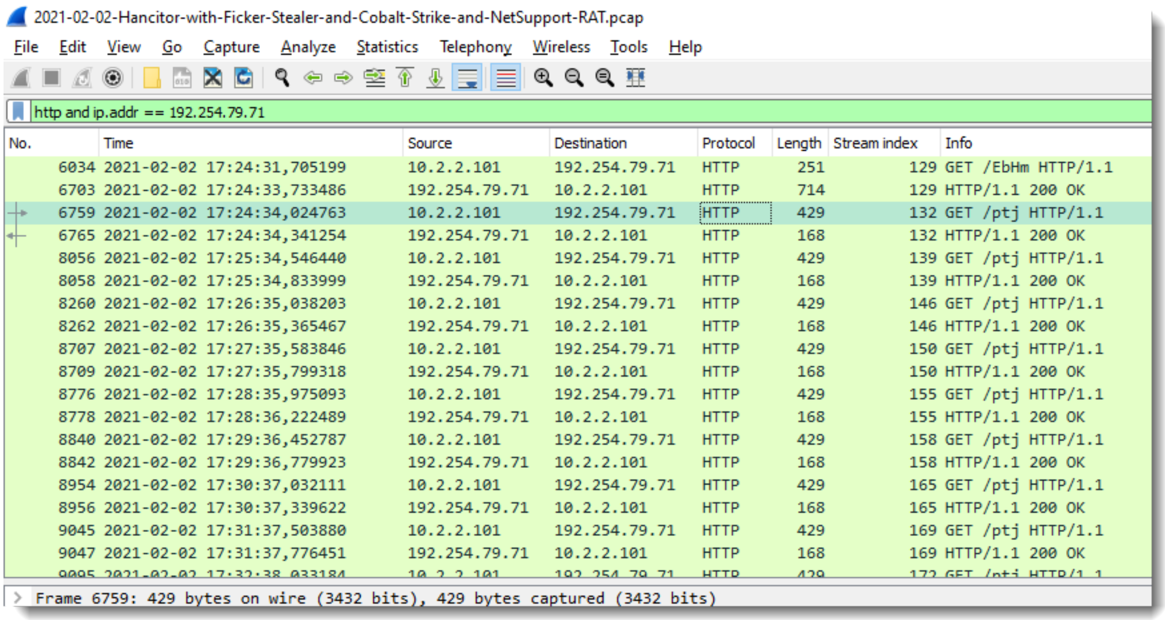
Let's take a closer look at some of the options.

First of all, option 0x0000 tells us that this is an HTTP beacon: it communicates over HTTP.
It does this by connecting to 192.254.79[.]71 (option 0x0008) on port 8080 (option 0x0002).
GET requests use path /ptj (option 0x0008), and POST requests use path /submit.php (option 0x000a)
And important for our analysis: there is a known private key (Has known private key) for the public key used by this beacon (option 0x0007).

Thus, armed with this information, we know that the beacon will send GET requests to the team server, to obtain instructions. If the team server has commands to be executed by the beacon, it will reply with encrypted data to the GET request. And when the beacon has to send back output from its commands to the team server, it will use a POST request with encrypted data.

If the team server has no commands for the beacon, it will send no encrypted data. This does not necessarily mean that the reply to a GET request contains no data: it is possible for the operator, through profiles, to masquerade the communication. For example, that the encrypted data is inside a GIF file. But that is not the case with this beacon. We know this, because there are no so-called malleable C2 instructions in this profile: option 0x000b is equal to 0x00000004 -> this means no operations should be performed on the data prior to decryption (we will explain this in more detail in a later blog post).

Let's create a display filter to view this C2 traffic: *http and ip.addr == 192.254.79[.]71*



No.	Time	Source	Destination	Protocol	Length	Stream index	Info
6034	2021-02-02 17:24:31,705199	10.2.2.101	192.254.79.71	HTTP	251	129	GET /EbHm HTTP/1.1
6703	2021-02-02 17:24:33,733486	192.254.79.71	10.2.2.101	HTTP	714	129	HTTP/1.1 200 OK
6759	2021-02-02 17:24:34,024763	10.2.2.101	192.254.79.71	HTTP	429	132	GET /ptj HTTP/1.1
6765	2021-02-02 17:24:34,341254	192.254.79.71	10.2.2.101	HTTP	168	132	HTTP/1.1 200 OK
8056	2021-02-02 17:25:34,546440	10.2.2.101	192.254.79.71	HTTP	429	139	GET /ptj HTTP/1.1
8058	2021-02-02 17:25:34,833999	192.254.79.71	10.2.2.101	HTTP	168	139	HTTP/1.1 200 OK
8260	2021-02-02 17:26:35,038203	10.2.2.101	192.254.79.71	HTTP	429	146	GET /ptj HTTP/1.1
8262	2021-02-02 17:26:35,365467	192.254.79.71	10.2.2.101	HTTP	168	146	HTTP/1.1 200 OK
8707	2021-02-02 17:27:35,583846	10.2.2.101	192.254.79.71	HTTP	429	150	GET /ptj HTTP/1.1
8709	2021-02-02 17:27:35,799318	192.254.79.71	10.2.2.101	HTTP	168	150	HTTP/1.1 200 OK
8776	2021-02-02 17:28:35,975093	10.2.2.101	192.254.79.71	HTTP	429	155	GET /ptj HTTP/1.1
8778	2021-02-02 17:28:36,222489	192.254.79.71	10.2.2.101	HTTP	168	155	HTTP/1.1 200 OK
8840	2021-02-02 17:29:36,452787	10.2.2.101	192.254.79.71	HTTP	429	158	GET /ptj HTTP/1.1
8842	2021-02-02 17:29:36,779923	192.254.79.71	10.2.2.101	HTTP	168	158	HTTP/1.1 200 OK
8954	2021-02-02 17:30:37,032111	10.2.2.101	192.254.79.71	HTTP	429	165	GET /ptj HTTP/1.1
8956	2021-02-02 17:30:37,339622	192.254.79.71	10.2.2.101	HTTP	168	165	HTTP/1.1 200 OK
9045	2021-02-02 17:31:37,503880	10.2.2.101	192.254.79.71	HTTP	429	169	GET /ptj HTTP/1.1
9047	2021-02-02 17:31:37,776451	192.254.79.71	10.2.2.101	HTTP	168	169	HTTP/1.1 200 OK

Frame 6759: 429 bytes on wire (3432 bits), 429 bytes captured (3432 bits)

Figure 8: full beacon download and HTTP requests with encrypted Cobalt Strike traffic

This displays all HTTP traffic to and from the team server. Remark that we already took a look at the first 2 packets in this view (packets 6034 and 6703): that's the download of the beacon itself, and that communication is not encrypted. Hence, we will filter these packets out with the following display filter:

http and ip.addr == 192.254.79.71 and frame.number > 6703

This gives us a list of GET requests with their reply. Remark that there's a GET request every minute. That too is in the beacon configuration: 60.000 ms of sleep (option 0x0003) with 0% variation (aka jitter, option 0x0005).

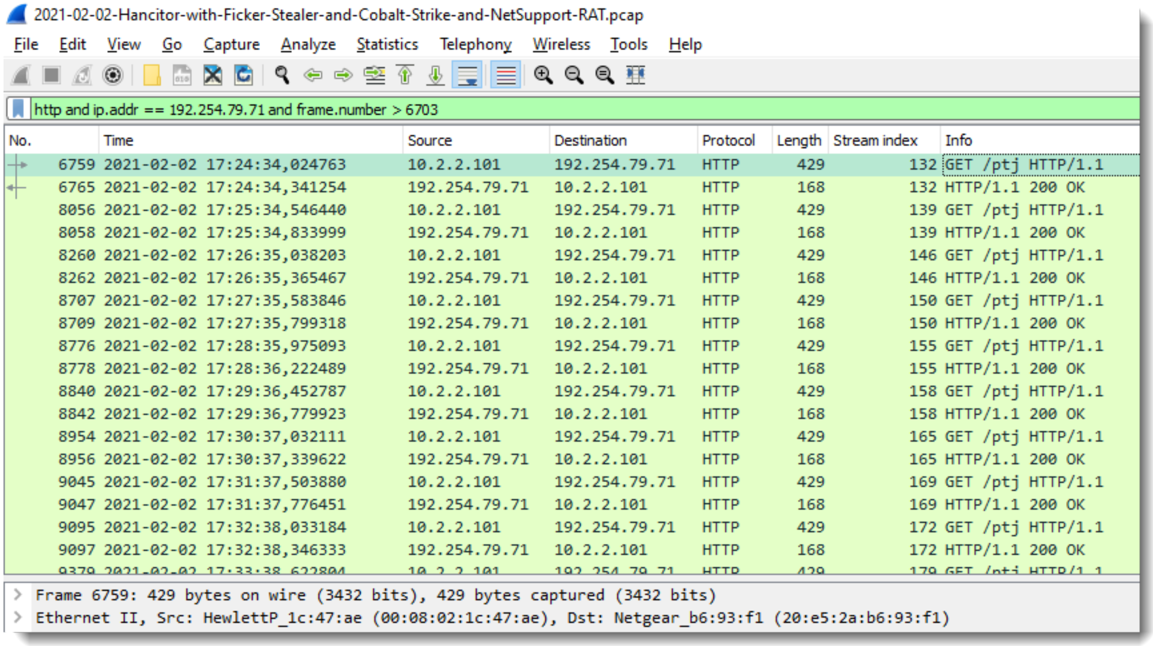


Figure 9: HTTP requests with encrypted Cobalt Strike traffic

We will now follow the first HTTP stream:

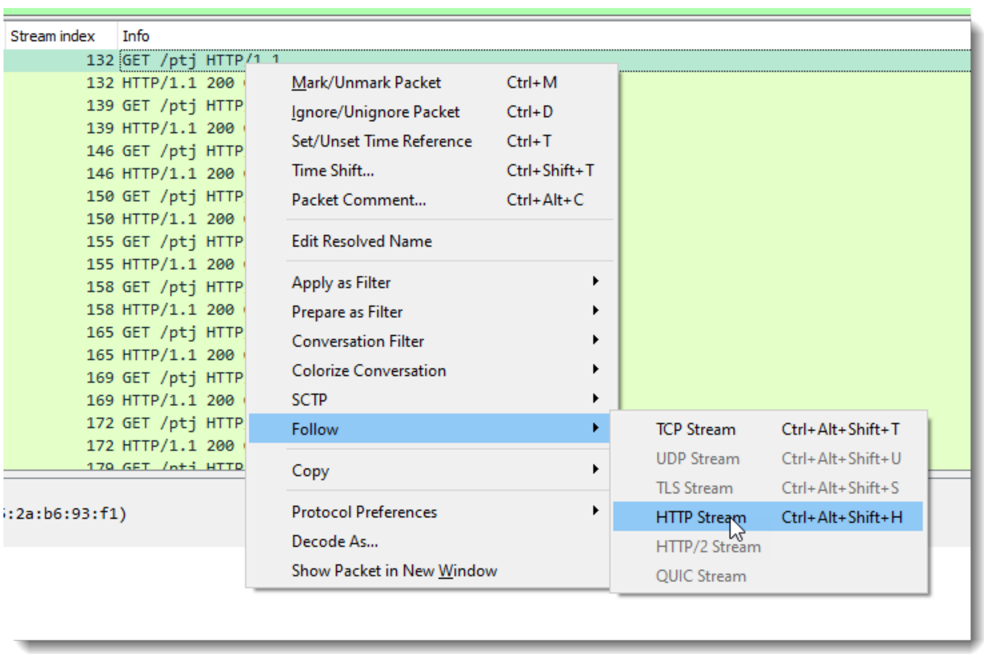


Figure 10: following HTTP stream

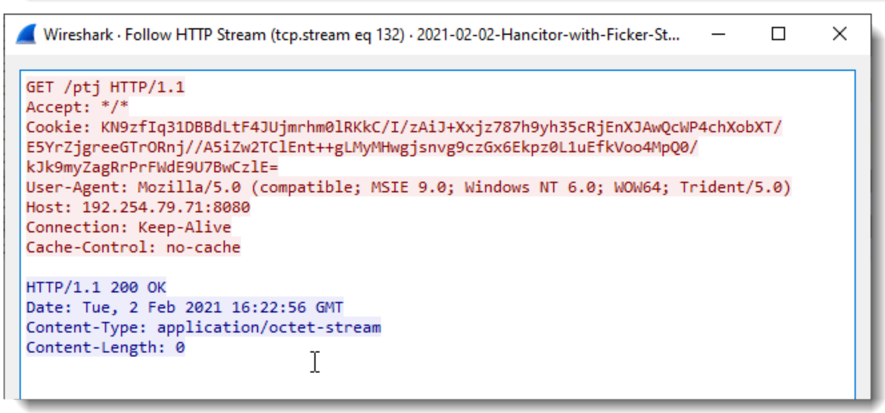


Figure 11: first HTTP stream

This is a GET request for /ptj that receives a STATUS 200 reply with no data. This means that there are no commands from the team server for this beacon for now: the operator has not issued any commands at that point in the capture file.

Remark the Cookie header of the GET request. This looks like a BASE64 string:
KN9zfIq31DBBdLtF4JUjmrhm0lRKkC/I/zAiJ+Xxjz787h9yh35cRjEnXJAwQcWP4chXobXT/E5YrZjgreeGTrORnj//A5iZw2TClEnt++gLMyMHwgjsnvg

That value is encrypted metadata that the beacon sends as a BASE64 string to the team server. This metadata is RSA encrypted with the public key inside the beacon configuration (option 0x0007), and the team server can decrypt this metadata because it has the private key. Remember that some private keys have been “leaked”, we discussed this in our first blog post in this series.

Our beacon analysis showed that this beacon uses a public key with a known private key. This means we can use tool cs-decrypt-metadata.py to decrypt the metadata (cookie) like this:

```
@NVISO_Labs

@NVISO_Labs C:\Demo>cs-decrypt-metadata.py KN9zfIq31DBBdLtF4JUjmrhm0lRKkC/I/zAiJ+Xxjz787h9yh35cRjEnXJAwQcWP4chXobXT/E5YrZjgreeGTrORnj//A5iZw2
TClEnt++gLMyMHwgjsnvg9czGx6Ekpz0L1uEfkVoo4MpQ0/kJk9myZagRrPrFWdE9U7BwCzlE=
Encrypted metadata: KN9zfIq31DBBdLtF4JUjmrhm0lRKkC/I/zAiJ+Xxjz787h9yh35cRjEnXJAwQcWP4chXobXT/E5YrZjgreeGTrORnj//A5iZw2TClEnt++gLMyMHwgjsnvg9c
zGx6Ekpz0L1uEfkVoo4MpQ0/kJk9myZagRrPrFWdE9U7BwCzlE=
Decrypted:
Header: 0000beef
Datasize: 0000005d
Raw key:  caeab4f452fe41182d504aa24966fbd0
 aeskey:  3342f45e6e2f71f5975c998600b11471
 hmackey: 7142dd70f4ec320badac8ca246a9488f
charset: 04e4 ANSI Latin 1; Western European (Windows)
charset_oem: 01b5 OEM United States
bid: 644d8e4 105175268
pid: 1c7c 7292
port: 0
flags: 04
var1: 10
var2: 0
var3: 19042
var4: 0
var5: 1988364896
var6: 1988359504
Internal IPv4: 10.6.9.111
Field: b'DESKTOP-Q21RU7A'
Field: b'maxwell.carter'
Field: b'svchost.exe'


@NVISO_Labs C:\Demo>
```

Figure 12: decrypting beacon metadata

We can see here the decrypted metadata. Very important to us, is the raw key: caeab4f452fe41182d504aa24966fbd0. We will use this key to decrypt traffic (the AES adn HMAC keys are derived from this raw key).

More metadata that we can find here is: the computername, the username, …

We will now follow the HTTP stream with packets 9379 and 9383: this is the first command send by the operator (team server) to the beacon:

```
Wireshark · Follow HTTP Stream (tcp.stream eq 179) · 2021-02-02-Hancitor-with-Ficker-St...

GET /ptj HTTP/1.1
Accept: */*
Cookie: KN9zfIq31DBBdLtF4JUjmrhm0lRKkC/I/zAiJ+Xxjz787h9yh35cRjEnXJAwQcWP4chXobXT/
E5YrZjgreeGTrORnj//A5iZw2TClEnt++gLMyMHwgjsnvg9czGx6Ekpz0L1uEfkVoo4MpQ0/
kJk9myZagRrPrFWdE9U7BwCzlE=
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0; WOW64; Trident/5.0)
Host: 192.254.79.71:8080
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Tue, 2 Feb 2021 16:32:01 GMT
Content-Type: application/octet-stream
Content-Length: 48

.....G.bB&{../.0.......B.....^k@.
..4.g.S..P....
```

Figure 13: HTTP stream with encrypted command

Here we can see that the reply contains 48 bytes of data (Content-length). That data is encrypted:

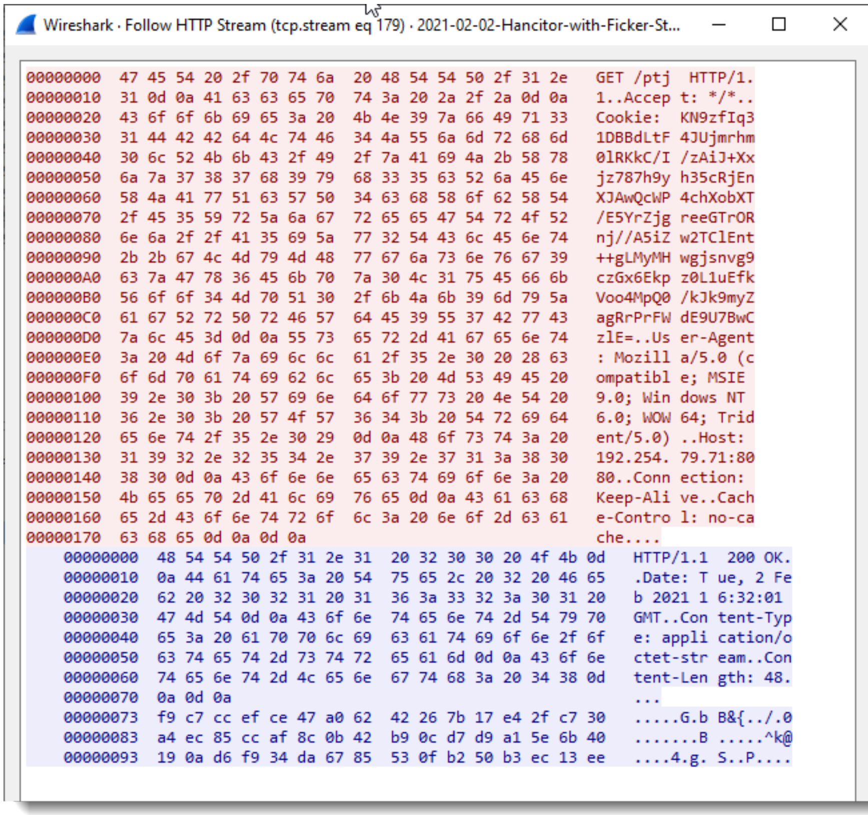


Figure 14: hexadecimal view of HTTP stream with encrypted command

Encrypted data like this, can be decrypted with tool cs-parse-http-traffic.py. Since the data is encrypted, we need to provide the raw key (option -r caeab4f452fe41182d504aa24966fbd0) and as the packet capture contains other traffic than pure Cobalt Strike C2 traffic, it is best to provide a display filter (option -Y http and ip.addr == 192.254.79.71 and frame.number > 6703) so that the tool can ignore all HTTP traffic that is not C2 traffic.

This produces the following output:

```
@NVISO_Labs C:\Demo>cs-parse-http-traffic.py -r caeab4f452fe41182d504aa24966fbd0 -Y "http and ip.addr == 192.254.79.71 and frame.number > 670
3" 2021-02-02-Hancitor-with-Ficker-Stealer-and-Cobalt-Strike-and-NetSupport-RAT.pcap | more
Packet number: 9383
HTTP response
Timestamp: 1612283521 20210202-163201
Data size: 16
Command: 4 SLEEP
 Sleep: 100
 Jitter: 90

Packet number: 9707
HTTP response
Timestamp: 1612283536 20210202-163216
Data size: 19
Command: 53 UNKNOWN
 Arguments length: 11
 b'\x00\x00\x00\xba\x00\x00\x00\x03.\\*'
 MD5: c0747fd03245897c597c4ba783e0ebb7

Packet number: 9723
HTTP request
http://192.254.79.71:8080/submit.php?id=242569267
Counter: 2
Callback: 22 TODO
b'\x00\x00\x00\xba'
----------------------------------------------------------------------------------------------------
C:\Users\maxwell.carter\Documents\*
D       0       01/28/2021 02:15:53     .
D       0       01/28/2021 02:15:53     ..
F       1330688 07/22/2018 05:45:52     13-Using_Powerpoint.ppt
F       691049  07/22/2018 02:20:50     160197-airport-template-16x9.pptx
F       610179  07/22/2018 02:20:32     160228-photo-template-16x9.pptx
F       831703  07/22/2018 02:21:02     160283-hook-template-16x9.pptx
F       206336  12/02/2016 23:11:02     1625.ppt
F       14688   12/02/2016 23:11:02     1625_example.jpg
F       123392  07/22/2018 05:53:02     19-adp.doc
F       7966940 07/22/2018 05:39:44     2016_election_final_report_06-14-18_0.pdf
F       224879  07/22/2018 02:23:28     2018-calendar.xlsx
F       521454  07/22/2018 05:39:26     301FRN.pdf
F       413895  07/22/2018 02:13:18     4Q-report.pptx
F       84649   01/17/2020 20:32:54     actvity-costs-tracker1.xlsx
F       199168  07/22/2018 05:52:14     AID1420-17.doc
F       202085  07/22/2018 05:38:36     amp-a0034857.pdf
F       1035264 07/22/2018 05:45:46     antFableEng.pps
F       62464   07/22/2018 02:35:14     APAFormat.doc
```

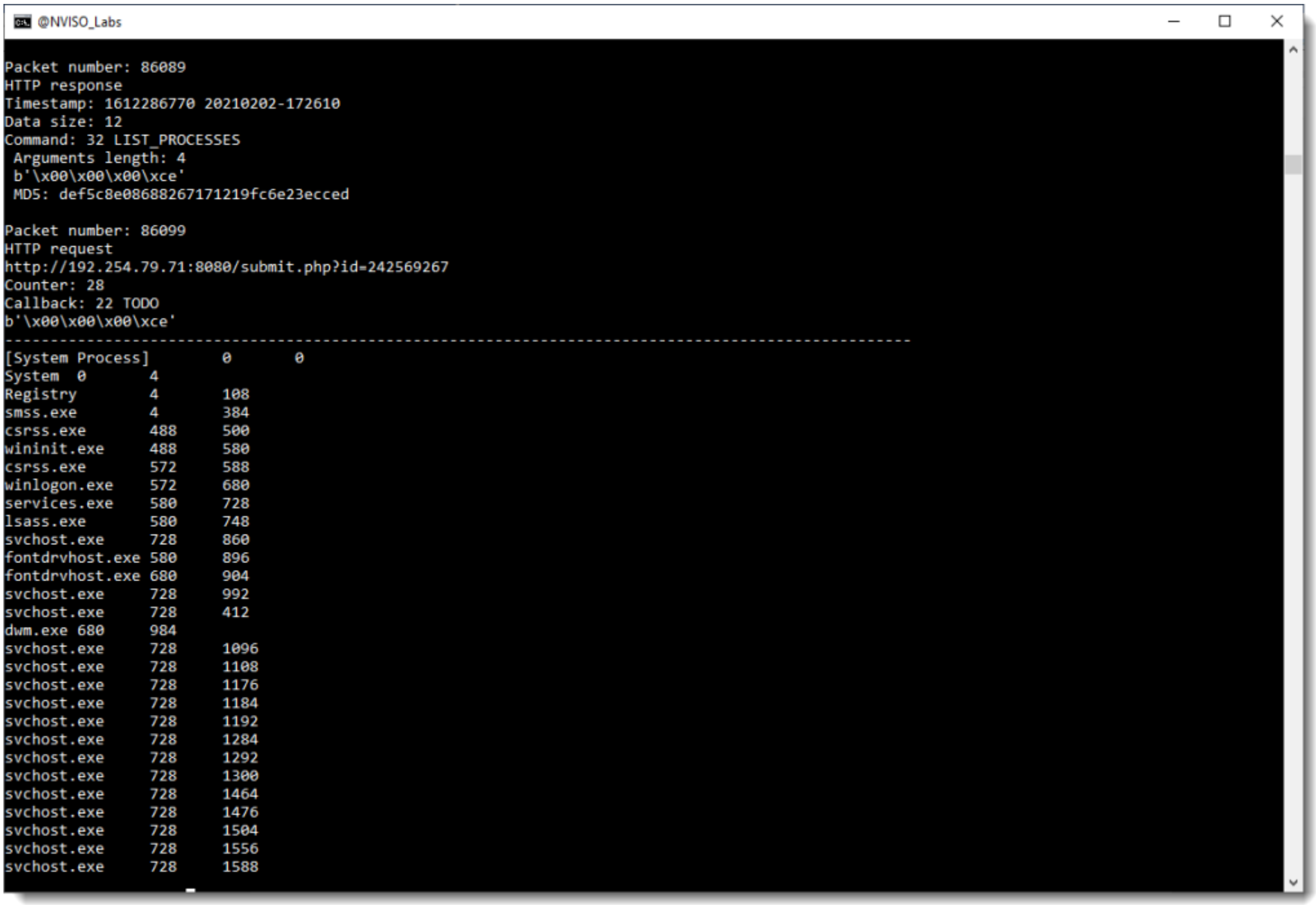
Figure 15: decrypted commands and results

Now we can see that the encrypted data in packet 9383 is a sleep command, with a sleeptime of 100 ms and a jitter factor of 90%. This means that the operator instructed the beacon to beacon interactive.

Decrypted packet 9707 contains an unknown command (id 53), but when we look at packet 9723, we see a directory listing output: this is the output result of the unknown command 53 being send back to the team server (notice the POST url /submit.php). Thus it’s safe to assume that command 53 is a directory listing command.

There are many commands and results in this capture file that tool cs-parse-http-traffic.py can decrypt, too much to show here. But we invite you to reproduce the commands in this blog post, and review the output of the tool.

The last command in the capture file is a process listing command:



```
@NVISO_Labs

Packet number: 86089
HTTP response
Timestamp: 1612286770 20210202-172610
Data size: 12
Command: 32 LIST_PROCESSES
 Arguments length: 4
 b'\x00\x00\x00\xce'
 MD5: def5c8e08688267171219fc6e23ecced

Packet number: 86099
HTTP request
http://192.254.79.71:8080/submit.php?id=242569267
Counter: 28
Callback: 22 TODO
b'\x00\x00\x00\xce'
----------------------------------------------------------------------------------------------------
[System Process]        0       0
System  0       4
Registry        4       108
smss.exe        4       384
csrss.exe       488     500
wininit.exe     488     580
csrss.exe       572     588
winlogon.exe    572     680
services.exe    580     728
lsass.exe       580     748
svchost.exe     728     860
fontdrvhost.exe 580     896
fontdrvhost.exe 680     904
svchost.exe     728     992
svchost.exe     728     412
dwm.exe 680     984
svchost.exe     728     1096
svchost.exe     728     1108
svchost.exe     728     1176
svchost.exe     728     1184
svchost.exe     728     1192
svchost.exe     728     1284
svchost.exe     728     1292
svchost.exe     728     1300
svchost.exe     728     1464
svchost.exe     728     1476
svchost.exe     728     1504
svchost.exe     728     1556
svchost.exe     728     1588
```

Figure 16: decrypted process listing command and result

Conclusion

Although the packet capture file we decrypted here was produced more than half a year ago by Brad Duncan by running a malicious Cobalt Strike beacon inside a sandbox, we can decrypt it today because the operators used a rogue Cobalt Strike package including a private key, that we recovered from VirusTotal.

Without this private key, we would not be able to decrypt the traffic.

The private key is not the only way to decrypt the traffic: if the AES key can be extracted from process memory, we can also decrypt traffic. We will cover this in an upcoming blog post.

About the authors

Didier Stevens is a malware expert working for NVISO. Didier is a SANS Internet Storm Center senior handler and Microsoft MVP, and has developed numerous popular tools to assist with malware analysis. You can find Didier on Twitter and LinkedIn.

You can follow NVISO Labs on Twitter to stay up to date on all our future research and publications.