Technical Analysis of Teardrop Malware

Detections

TearDrop was identified by different antivirus companies with different names regarding their analysis as shown below:

Vendor	Result
Malwarebyes	Backdoor.TearDrop
Microsoft	Trojan: Win64/Cobaltstrike.RN!dha
TrendMicro	Trojan.Win64.TEARDROP.A
McAfee	RDN/Generic.dx
BidDefender	Trojan.Teardrop.C

The Table below contains the artifacts of samples we will analyze in this report

Filename	**MD5**	**PE Timestamp**	**Size (inBytes)**	**Description**
Teardrop.dll	bd842c41b4c1b3c2deb475d7a3876599	(Fri Mar 09 20:23:43 2018)	530432	Type of the malware: Trojan
Cobalt_Strike.bin	a054fa3ae92e26509b88b110e7c932e2	(Tue Feb 14 11:40:46 2017)	262144	Type of the malware: Trojan

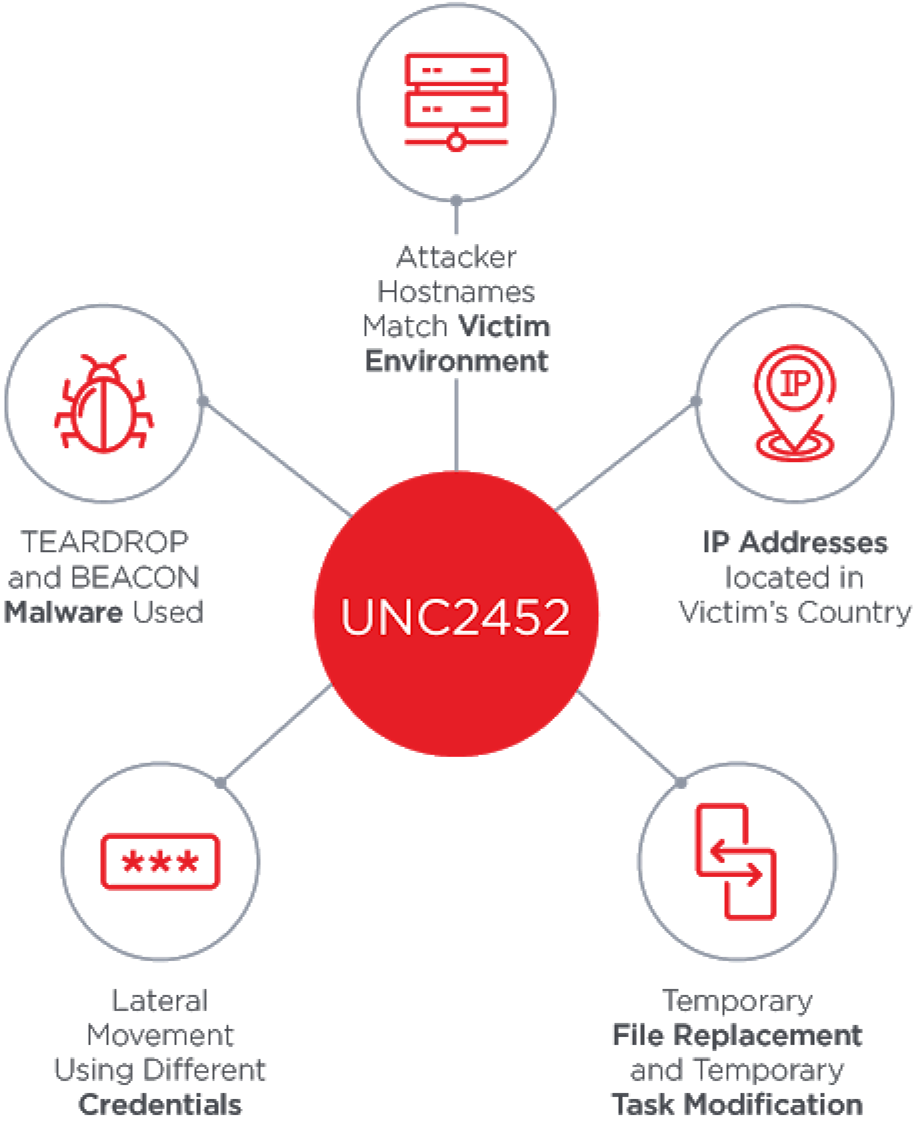
Summary

Multiple samples have been recovered regarding to Sunburst analysis, delivering different payloads. unseen memory-only dropper, TearDrop to deploy Cobalt Strike BEACON.

Introduction

TearDrop is a memory only dropper that runs as a service, a second-stage payload was identified TearDrop. and The Analysis discovered samples showed that TearDrop ultimately loaded a Cobalt Strike beacon into memory.

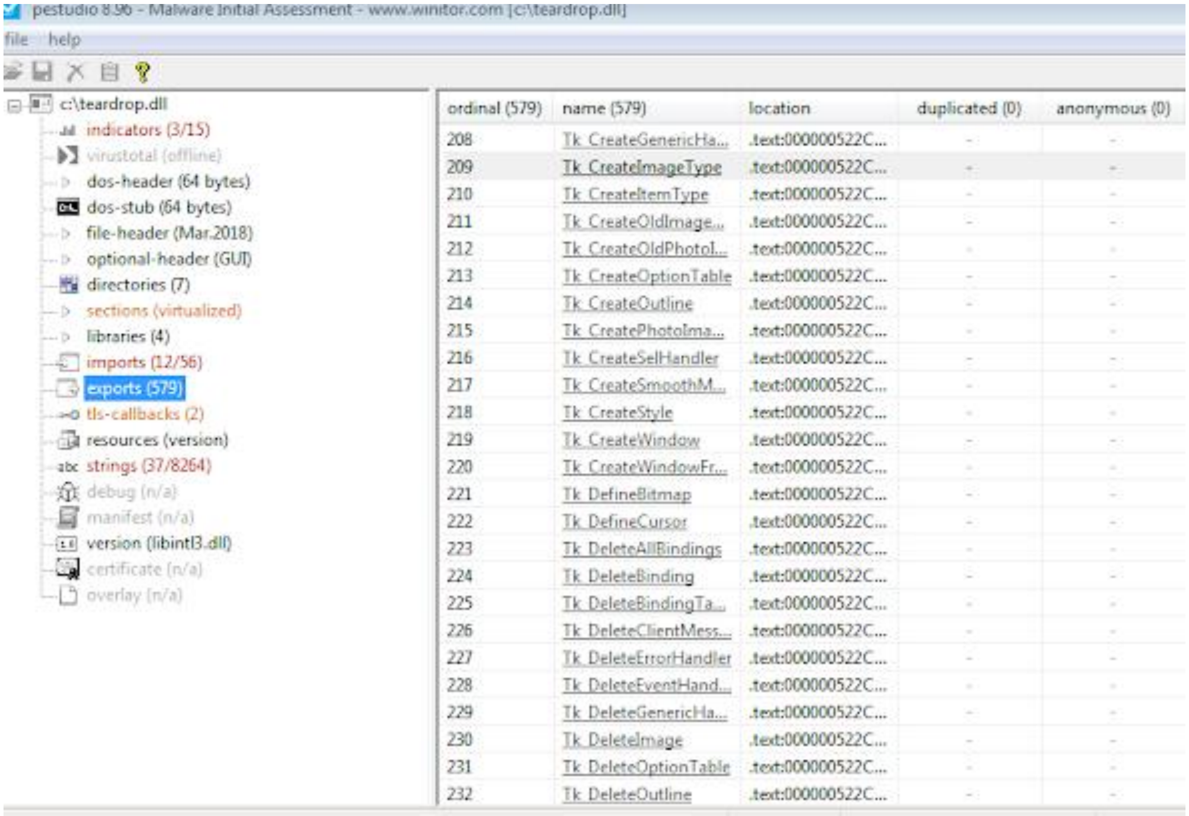
TearDrop and the other custom Cobalt Strike Beacon loaders observed during the Solorigate investigation . Each custom loader loads either a Beacon Reflective Loader or a preliminary loader that subsequently loads the Beacon Reflective Loader. Reflective DLL loading is a technique for loading a DLL into a process memory without using the Windows loader.


Attacker Hostnames Match **Victim Environment**
TEARDROP and BEACON **Malware** Used
IP Addresses located in Victim's Country
UNC2452
Lateral Movement Using Different **Credentials**
Temporary **File Replacement** and Temporary **Task Modification**

Technical Analysis

We're going to analyze Tk_CreateImageType Export to figure out in what exactly it's used.

as shown below Teardrop imported into pestudio:



ordinal (579)	name (579)	location	duplicated (0)	anonymous (0)
208	Tk_CreateGenericHa...	.text:000000522C...	-	-
209	Tk_CreateImageType	.text:000000522C...	-	-
210	Tk_CreateItemType	.text:000000522C...	-	-
211	Tk_CreateOldImage...	.text:000000522C...	-	-
212	Tk_CreateOldPhotoI...	.text:000000522C...	-	-
213	Tk_CreateOptionTable	.text:000000522C...	-	-
214	Tk_CreateOutline	.text:000000522C...	-	-
215	Tk_CreatePhotoIma...	.text:000000522C...	-	-
216	Tk_CreateSelHandler	.text:000000522C...	-	-
217	Tk_CreateSmoothM...	.text:000000522C...	-	-
218	Tk_CreateStyle	.text:000000522C...	-	-
219	Tk_CreateWindow	.text:000000522C...	-	-
220	Tk_CreateWindowFr...	.text:000000522C...	-	-
221	Tk_DefineBitmap	.text:000000522C...	-	-
222	Tk_DefineCursor	.text:000000522C...	-	-
223	Tk_DeleteAllBindings	.text:000000522C...	-	-
224	Tk_DeleteBinding	.text:000000522C...	-	-
225	Tk_DeleteBindingTa...	.text:000000522C...	-	-
226	Tk_DeleteClientMess...	.text:000000522C...	-	-
227	Tk_DeleteErrorHandler	.text:000000522C...	-	-
228	Tk_DeleteEventHand...	.text:000000522C...	-	-
229	Tk_DeleteGenericHa...	.text:000000522C...	-	-
230	Tk_DeleteImage	.text:000000522C...	-	-
231	Tk_DeleteOptionTable	.text:000000522C...	-	-
232	Tk_DeleteOutline	.text:000000522C...	-	-

The malicious code is contained in the export TK_CreateImageType, when executed, that malicious code reads a file named upbeat_anxiety.jpg from the current directory and ensures it has a jpg header. It will also check that the registry key HKCU\Software\Microsoft\CTF exists.

An embedded copy of Cobalt Strike is then extracted and executed. That CobaltStrike sample connects to (infinitysoftwares*com) for command and control.

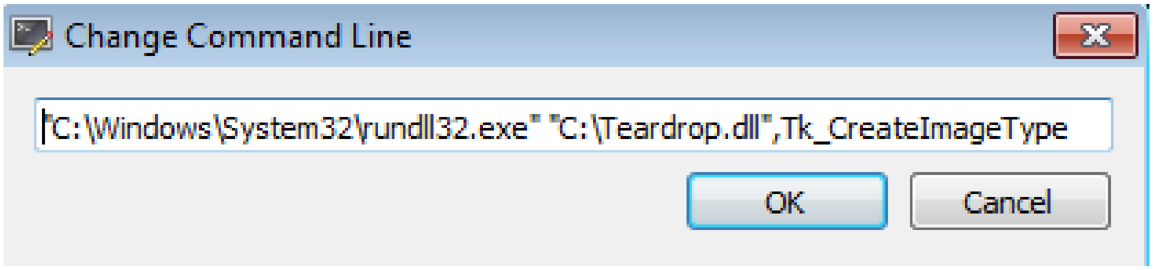
So, let's start analyzing Teardrop.dll and Exactly "TK_CreateImageType" Export to get the embedded Cobalt_Strike.
we will need to get the location of this Export, so we can start with rundll32.exe to be imported into the debugger x64.

Then we can map to Teardrop.dll through it, and Some configurations in x64dbg to break on each DLL during the execution, to do this got to as shown below:

- Import rundll32.exe into x64dbg
- go to Options -> Preferences -> Check DLL Entry
- put breakpoint on the DLL Export and click F9 till get the location

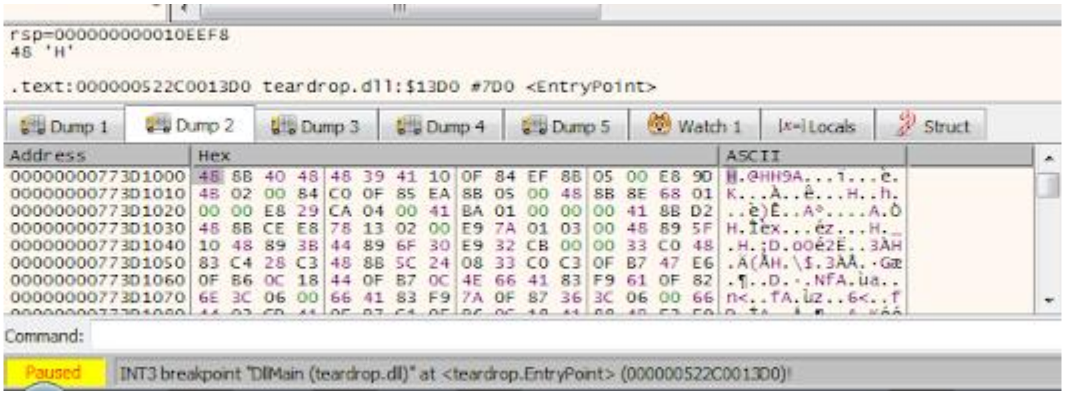
So how we map to the targeted Export through the current running Rundll32.exe:

- Go to File - > click on "Change Command Line" and write as shown below:



```
"C:\Windows\System32\rundll32.exe" "C:\Teardrop.dll",Tk_CreateImageType
```

- Now we can click restart and start Execution of till getinng the Main DLL of Teardrop.



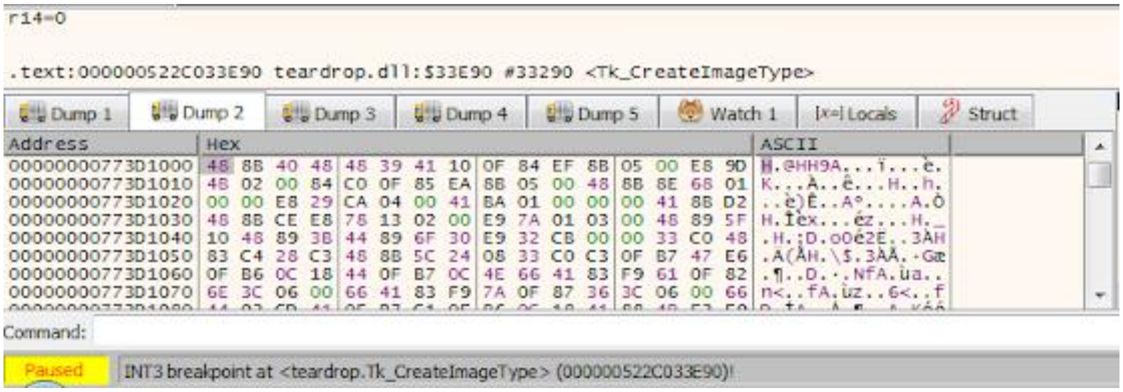
```
rsp=000000000010EEF8
48 'H'

.text:000000522C0013D0 teardrop.dll:$13D0 #7D0 <EntryPoint>
```

```
Address           Hex                                              ASCII
000000000773D1000 48 8B 40 48 48 39 41 10 0F 84 EF 8B 05 00 E8 9D  H.@HH9A...ï...è.
000000000773D1010 48 02 00 84 C0 0F 85 EA 8B 05 00 48 8B 8E 68 01  K...À..ê...H..h.
000000000773D1020 00 00 E8 29 CA 04 00 41 BA 01 00 00 00 41 8B D2  ..è)Ê..A°....A.Ò
000000000773D1030 48 8B CE E8 78 13 02 00 E9 7A 01 03 00 48 89 5F  H.Îèx...éz...H._
000000000773D1040 10 48 89 3B 44 89 6F 30 E9 32 CB 00 00 33 C0 48  .H.;D.o0é2Ë..3ÀH
000000000773D1050 83 C4 28 C3 48 8B 5C 24 08 33 C0 C3 0F B7 47 E6  .Ä(ÃH.\$.3ÀÃ.·Gæ
000000000773D1060 0F B6 0C 18 44 0F B7 0C 4E 66 41 83 F9 61 0F 82  .¶..D.·.NfA.ùa..
000000000773D1070 6E 3C 06 00 66 41 83 F9 7A 0F 87 36 3C 06 00 66  n<..fA.ùz..6<..f
```

Command:

Paused INT3 breakpoint "DllMain (teardrop.dll)" at <teardrop.EntryPoint> (000000522C0013D0)!

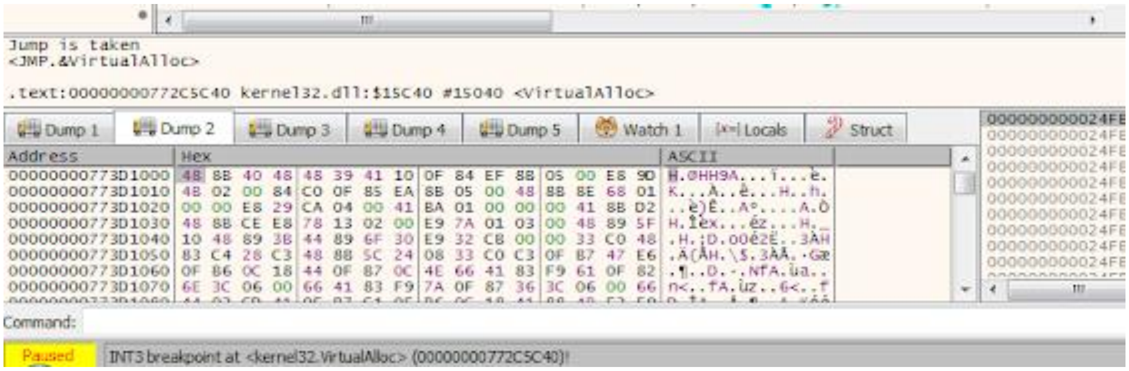
- After getting the Main DLL, We need to set a breakpoint on the targted Export (TK_CreateImageType)
  to that:
  Go to Symbols -> click looking for the targted Export on the left list then you will get all Exports on the right side.
  click on it and rightclick and -> set toggle breakpoint
- And continue running till you get it at the bottom as shown below:



```
r14=0

.text:000000522C033E90 teardrop.dll:$33E90 #33290 <Tk_CreateImageType>

Address           Hex                                               ASCII
000000000773D1000 48 8B 40 48 48 39 41 10 0F 84 EF 8B 05 00 E8 9D  H.@HH9A...ï...è.
000000000773D1010 48 02 00 84 C0 0F 85 EA 8B 05 00 48 8B 8E 68 01  K...À..ê...H..h.
000000000773D1020 00 00 E8 29 CA 04 00 41 BA 01 00 00 00 41 8B D2  ..è)Ê..Aº....A.Ò
000000000773D1030 48 8B CE E8 78 13 02 00 E9 7A 01 03 00 48 89 5F  H.Îèx...éz...H._
000000000773D1040 10 48 89 3B 44 89 6F 30 E9 32 CB 00 00 33 C0 48  .H.;D.o0é2Ë..3ÀH
000000000773D1050 83 C4 28 C3 48 8B 5C 24 08 33 C0 C3 0F B7 47 E6  .Ä(ÃH.\$.3ÀÃ.·Gæ
000000000773D1060 0F B6 0C 18 44 0F B7 0C 4E 66 41 83 F9 61 0F 82  .¶..D.·.NfA.ùa..
000000000773D1070 6E 3C 06 00 66 41 83 F9 7A 0F 87 36 3C 06 00 66  n<..fA.ùz..6<..f

INT3 breakpoint at <teardrop.Tk_CreateImageType> (000000522C033E90)!
```

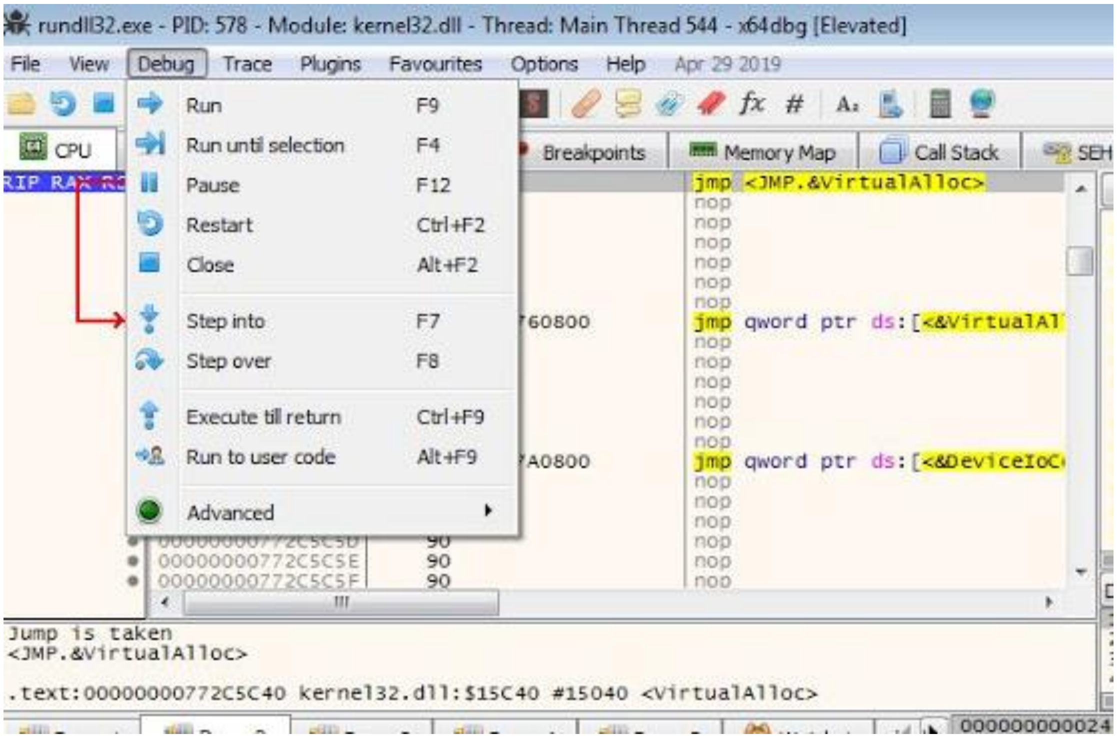
Memory Tracking

- Teardrop.dll is loading cobalt beacon into memory, so we can set breakpoints on common memory APIs to see what we can get.
- Most common Memory APIs may be used like(VirtualAlloc, VirtualProtect,VirtualFree, Virtual Query)
- Write click in disassemble window and click on Go-> Expression-> Write VirtuallAlloc the ok and Press F2.
- repeat last step with each API mentioned above.
- Continue the execution till hitting VirtualAlloc, you can observe that at the bottom.



```
Jump is taken
<JMP.&VirtualAlloc>

.text:00000000772C5C40 kernel32.dll:$15C40 #15040 <VirtualAlloc>

Address           Hex                                               ASCII
000000000773D1000 48 8B 40 48 48 39 41 10 0F 84 EF 8B 05 00 E8 9D  H.@HH9A...ï...è.
000000000773D1010 48 02 00 84 C0 0F 85 EA 8B 05 00 48 8B 8E 68 01  K...À..ê...H..h.
000000000773D1020 00 00 E8 29 CA 04 00 41 BA 01 00 00 00 41 8B D2  ..è)Ê..Aº....A.Ò
000000000773D1030 48 8B CE E8 78 13 02 00 E9 7A 01 03 00 48 89 5F  H.Îèx...éz...H._
000000000773D1040 10 48 89 3B 44 89 6F 30 E9 32 CB 00 00 33 C0 48  .H.;D.o0é2Ë..3ÀH
000000000773D1050 83 C4 28 C3 48 8B 5C 24 08 33 C0 C3 0F B7 47 E6  .Ä(ÃH.\$.3ÀÃ.·Gæ
000000000773D1060 0F B6 0C 18 44 0F B7 0C 4E 66 41 83 F9 61 0F 82  .¶..D.·.NfA.ùa..
000000000773D1070 6E 3C 06 00 66 41 83 F9 7A 0F 87 36 3C 06 00 66  n<..fA.ùz..6<..f

INT3 breakpoint at <kernel32.VirtualAlloc> (00000000772C5C40)!
```

- Mainly, when you hit VirtuallAlloc that means the software executed reserves Memory pages for its execution in Memory
- You can click on Debug-> Run to user code

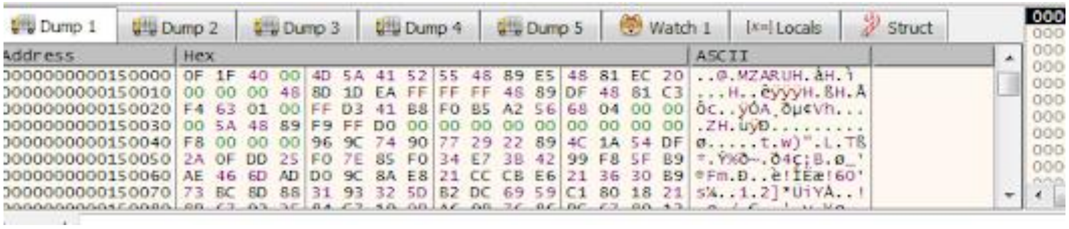


```
jmp <JMP.&VirtualAlloc>
nop
nop
nop
nop
nop
nop
jmp qword ptr ds:[<&VirtualAl
nop
nop
nop
nop
nop
nop
jmp qword ptr ds:[<&DeviceIoC
nop
nop
nop
nop
nop
nop
```

```
00000000772C5C5E 90
00000000772C5C5F 90
```

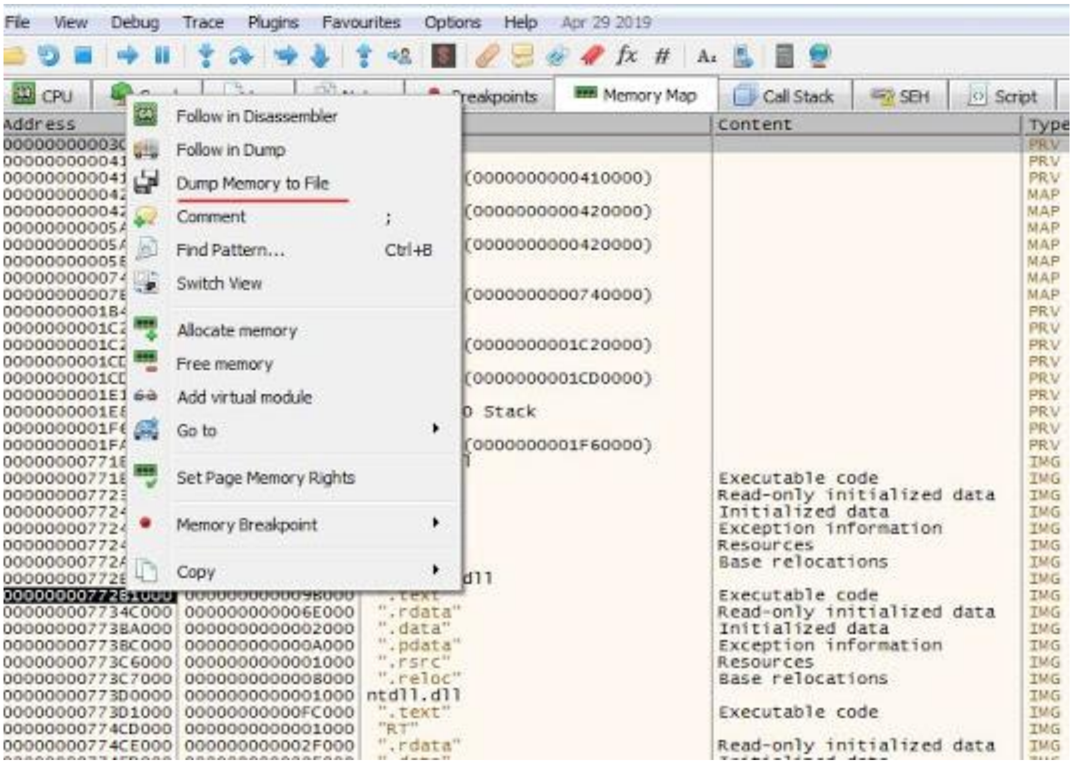
```
Jump is taken
<JMP.&VirtualAlloc>

.text:00000000772C5C40 kernel32.dll:$15C40 #15040 <VirtualAlloc>
```

- Continue running till you get 'VirtualProtect' hit and check register on the right window, writeclick and follow to dump and you will get the Cobalt_Strike copy which we're looking for.



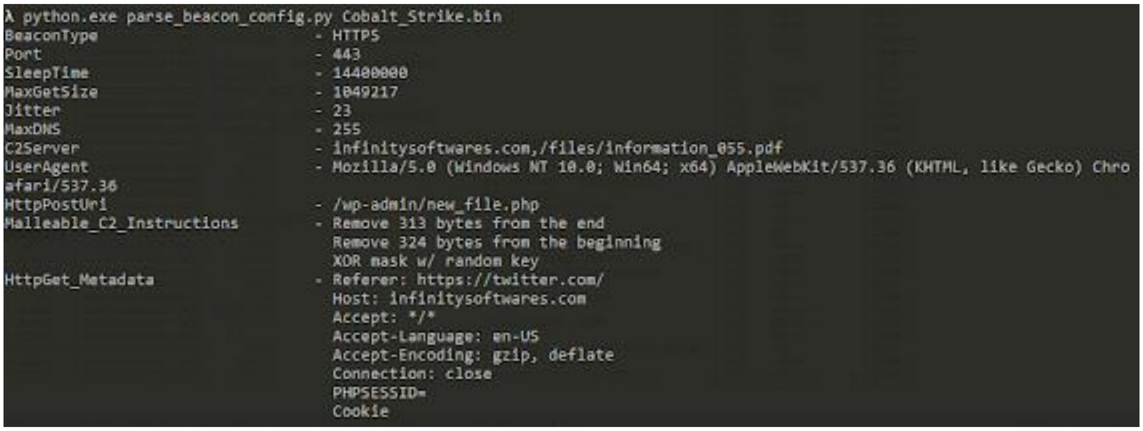
```
Address           Hex                                              ASCII
0000000000150000  0F 1F 40 00 4D 5A 41 52 55 48 89 E5 48 81 EC 20  ..@.MZARUH.åH.ì
0000000000150010  00 00 00 48 8D 1D EA FF FF FF 48 89 DF 48 81 C3  ...H..êÿÿÿH.ßH.Ã
0000000000150020  F4 63 01 00 FF D3 41 B8 F0 B5 A2 56 68 04 00 00  ôc..ÿÓA¸ðµ¢Vh...
0000000000150030  00 5A 48 89 F9 FF D0 00 00 00 00 00 00 00 00 00  .ZH.ùÿÐ.........
0000000000150040  F8 00 00 00 96 9C 74 90 77 29 22 89 4C 1A 54 DF  ø.....t.w)".L.Tß
0000000000150050  2A 0F DD 25 F0 7E 85 F0 34 E7 3B 42 99 F8 5F B9  *.Ý%ð~.ð4ç;B.ø_¹
0000000000150060  AE 46 6D AD D0 9C 8A E8 21 CC CB E6 21 36 30 B9  ®Fm.Ð..è!ÌËæ!60¹
0000000000150070  73 BC 8D 88 31 93 32 5D B2 DC 69 59 C1 80 18 21  s¼..1.2]²ÜiYÁ..!
```

- Now, we need to dump the Cobalt file to check the beacon result
  So we writeclick on the first line in Dump and click -> Dump in Memory Map as show below and then go to Memory Map and writeclick -> Dump Memory to File



```
000000007734C000 000000000006E000  ".rdata"     Read-only initialized data  IMG
00000000773BA000 0000000000002000  ".data"      Initialized data            IMG
00000000773BC000 000000000000A000  ".pdata"     Exception information       IMG
00000000773C6000 0000000000001000  ".rsrc"      Resources                   IMG
00000000773C7000 0000000000008000  ".reloc"     Base relocations            IMG
00000000773D0000 0000000000001000 ntdll.dll                                 IMG
00000000773D1000 00000000000FC000  ".text"      Executable code             IMG
00000000774CD000 0000000000001000  "RT"                                     IMG
00000000774CE000 000000000002F000  ".rdata"     Read-only initialized data  IMG
```

- and finally we get the file we're getting for, So now we need to extract the beacon from the file and check the output to see IOCs of what file does.
- Sentinel-one has created a tool to extract the beacon from the Cobalt_Strike Payload which called: parse_beacon_config.py



```
λ python.exe parse_beacon_config.py Cobalt_Strike.bin
BeaconType                       - HTTPS
Port                             - 443
SleepTime                        - 14400000
MaxGetSize                       - 1049217
Jitter                           - 23
MaxDNS                           - 255
C2Server                         - infinitysoftwares.com,/files/information_055.pdf
UserAgent                        - Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chro
afari/537.36
HttpPostUri                      - /wp-admin/new_file.php
Malleable_C2_Instructions        - Remove 313 bytes from the end
                                   Remove 324 bytes from the beginning
                                   XOR mask w/ random key
HttpGet_Metadata                 - Referer: https://twitter.com/
                                   Host: infinitysoftwares.com
                                   Accept: */*
                                   Accept-Language: en-US
                                   Accept-Encoding: gzip, deflate
                                   Connection: close
                                   PHPSESSID=
                                   Cookie
```

- So we run Python script Parse_beacon_config.py on the extracted file from the memory to get the details of beacon happening in the memory as below:

BeaconType	HTTPS
Port	443
SleepTime	14400000
MaxGetSize	1049217
Jitter	23
MaxDNS	255
C2Server	infinitysoftwares*com,/files/information_055.pdf
UserAgent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.92 Safari/537.36
HttpPostUri	/wp-admin/new_file.php
Malleable_C2_Instructions	Remove 313 bytes from the end Remove 324 bytes from the beginning Remove 324 bytes from the beginning
HttpGet_Metadata	HttpGet_Metadata Host: infinitysoftwares.com Accept: */* Accept-Language: en-US Accept-Encoding: gzip, deflate Connection: close PHPSESSID= Cookie
HttpPost_Metadata	Host: infinitysoftwares*com Accept: */* Accept-Language: en-US Connection: close name="uploaded_1";filename="33139.pdf" Content-Type: text/plain
SpawnTo	b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
PipeName	-

DNS_Idle	208.67.220.220
DNS_Sleep	0
SSH_Host	Not Found
SSH_Port	Not Found
SSH_Username	Not Found
SSH_Password_Plaintext	Not Found
SSH_Password_Pubkey	Not Found
HttpGet_Verb	Get
HttpPost_Verb	POST
HttpPostChunk	0
Spawnto_x86	%windir%\syswow64\print.exe
Spawnto_x64	%windir%\sysnative\msiexec.exe
CryptoScheme	0
Proxy_Config	Not Found
Proxy_User	Not Found
Proxy_Password	Not Found
Proxy_Behavior	Use IE settings
Watermark	943010104
bStageCleanup	True
bCFGCaution	False
KillDate	0
bProcInject_StartRWX	False
bProcInject_UseRWX	False
bProcInject_MinAllocSize	8493

ProcInject_PrependAppend_x86	b'\x90\x90' Empty
ProcInject_PrependAppend_x64	b'\x0f\x1f\x00' Empty
ProcInject_Execute	ntdll:RtlUserThreadStart CreateThread NtQueueApcThread SetThreadContext
ProcInject_AllocationMethod	NtMapViewOfSection
bUsesCookies	True
HostHeader	-

- As per the output, the malware is communicating with infinitysoftwares*com (C2) server.

Contact:

- Name: Mustafa Hussien
- https://www.linkedin.com/in/mustafa-m-hussien-1b3265103/
- E-mail: mustafa12hussien@gmail.com