CyberThreatIntel/China/APT/Chimera/Analysis.md at master · StrangerealIntel/CyberThreatIntel

By StrangerealIntel

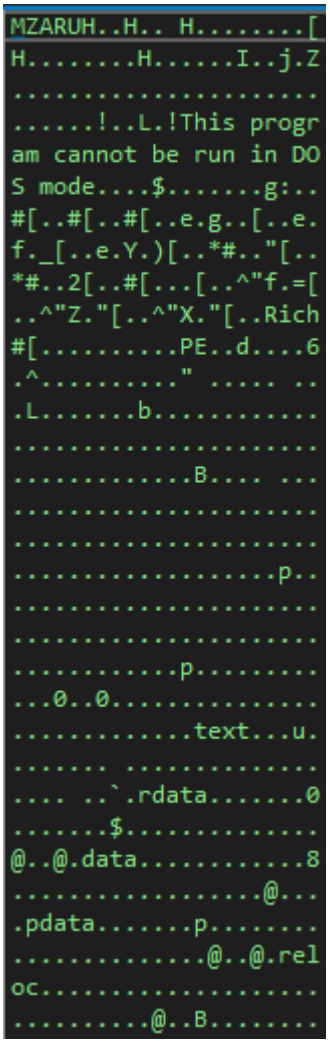
Archived: 2026-04-02 11:14:30 UTC

Chimera, APT19 under the radar ?

Initital approach

At the beginning I studied a suspicious DLL uploaded on Anyrun, this one have been tagged as "Malformatted PE header". By the fact that some Threat Actor let theirs DLL with an invalid header for avoiding to correctly run in sandbox or in AV sandbox and modify it for run by a loader (side-loading with multiples files [Header + DLL], script for rebuilding the header...).

As the first look, we can see the anomaly on the PE header based on redirection to a part of malware.



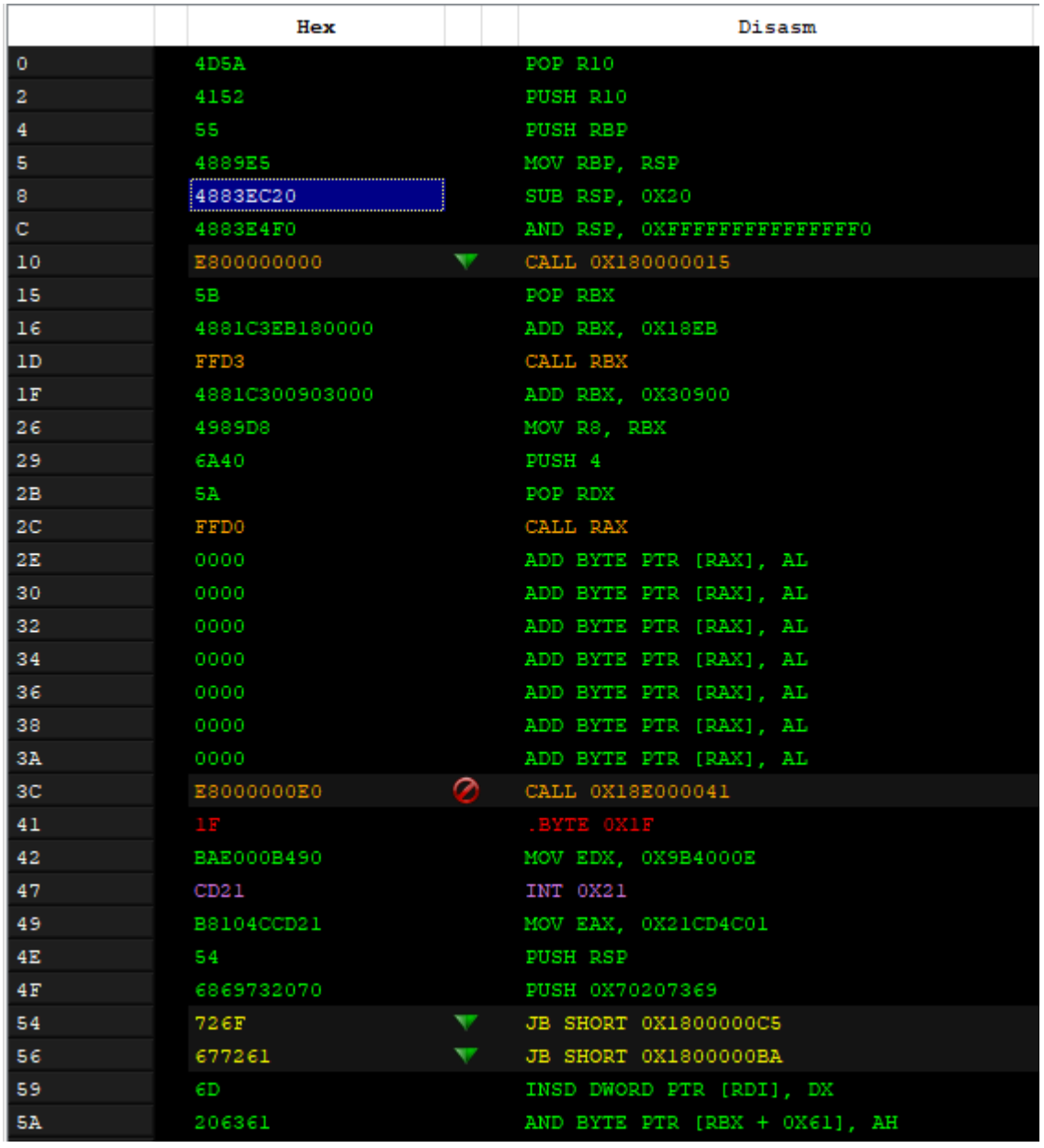
The timestamp is valid if we compare to the other sections (proving that doesn't modified), the internal name in the import section and the exported functions are the same that used by Meterpreter as reflective loader method.

Offset	Name	Value	Meaning
2A8C0	Characteristics	0	
2A8C4	TimeDateStamp	5E9A36EC	vendredi, 17.04.2020 23:08:28 UTC
2A8C8	MajorVersion	0	
2A8CA	MinorVersion	0	
2A8CC	Name	2B812	metsrv.dll
2A8D0	Base	1	
2A8D4	NumberOfFunctions	51	
2A8D8	NumberOfNames	51	
2A8DC	AddressOfFunctions	2B4E8	
2A8E0	AddressOfNames	2B62C	
2A8E4	AddressOfNameOrdinals	2B770	

Exported Functions [81 entries]

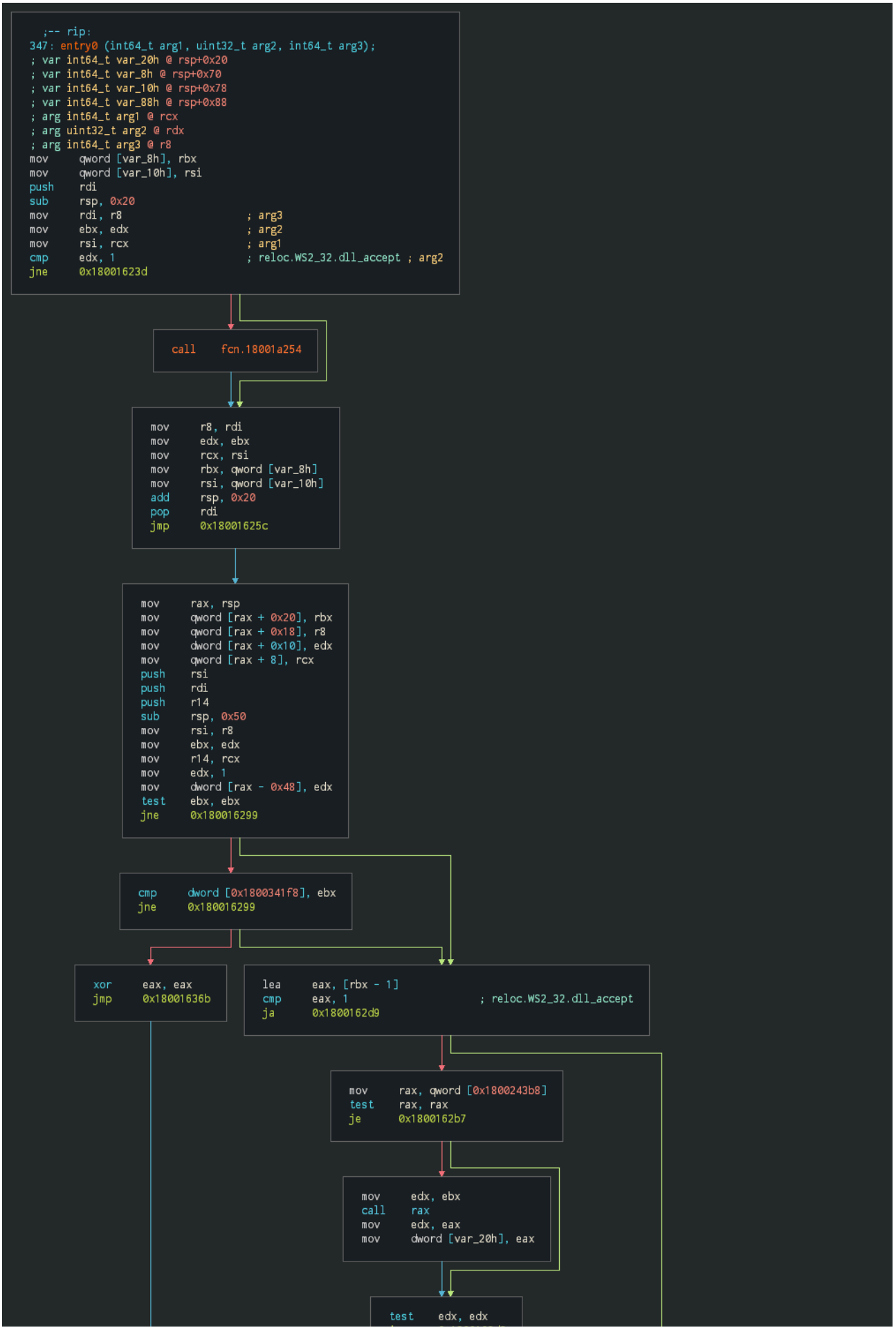
Offset	Ordinal	Function RVA	Name RVA	Name
2A8E8	1	18F0	2B81D	Init
2A8EC	2	2500	2B822	ReflectiveLoader
2A8F0	3	75C8	2B833	buffer_from_file
2A8F4	4	76D4	2B844	buffer_to_file
2A8F8	5	12D68	2B853	channel_close
2A8FC	6	125A8	2B861	channel_create
2A900	7	126A8	2B870	channel_create_datagram
2A904	8	12704	2B888	channel_create_pool
2A908	9	1264C	2B89C	channel_create_stream
2A90C	A	12A24	2B8B2	channel_default_io_handler
2A910	B	1275C	2B8CD	channel_destroy
2A914	C	12F6C	2B8DD	channel_exists
2A918	D	12F4C	2B8EC	channel_find_by_id
2A91C	E	12A00	2B8FF	channel_get_buffered_io_context
2A920	F	12850	2B91F	channel_get_class
2A924	10	1287C	2B931	channel_get_flags
2A928	11	127F0	2B943	channel_get_id
2A92C	12	12A18	2B952	channel_get_native_io_context
2A930	13	12844	2B970	channel_get_type
2A934	14	12E50	2B981	channel_interact
2A938	15	12868	2B992	channel_is_flag
2A93C	16	12894	2B9A2	channel_is_interactive
2A940	17	12A74	2B9B9	channel_open
2A944	18	12B3C	2B9C6	channel_read
2A948	19	129BC	2B9D3	channel_read_from_buffered
2A94C	1A	129F0	2B9EE	channel_set_buffered_io_handler
2A950	1B	1285C	2BA0E	channel_set_flags

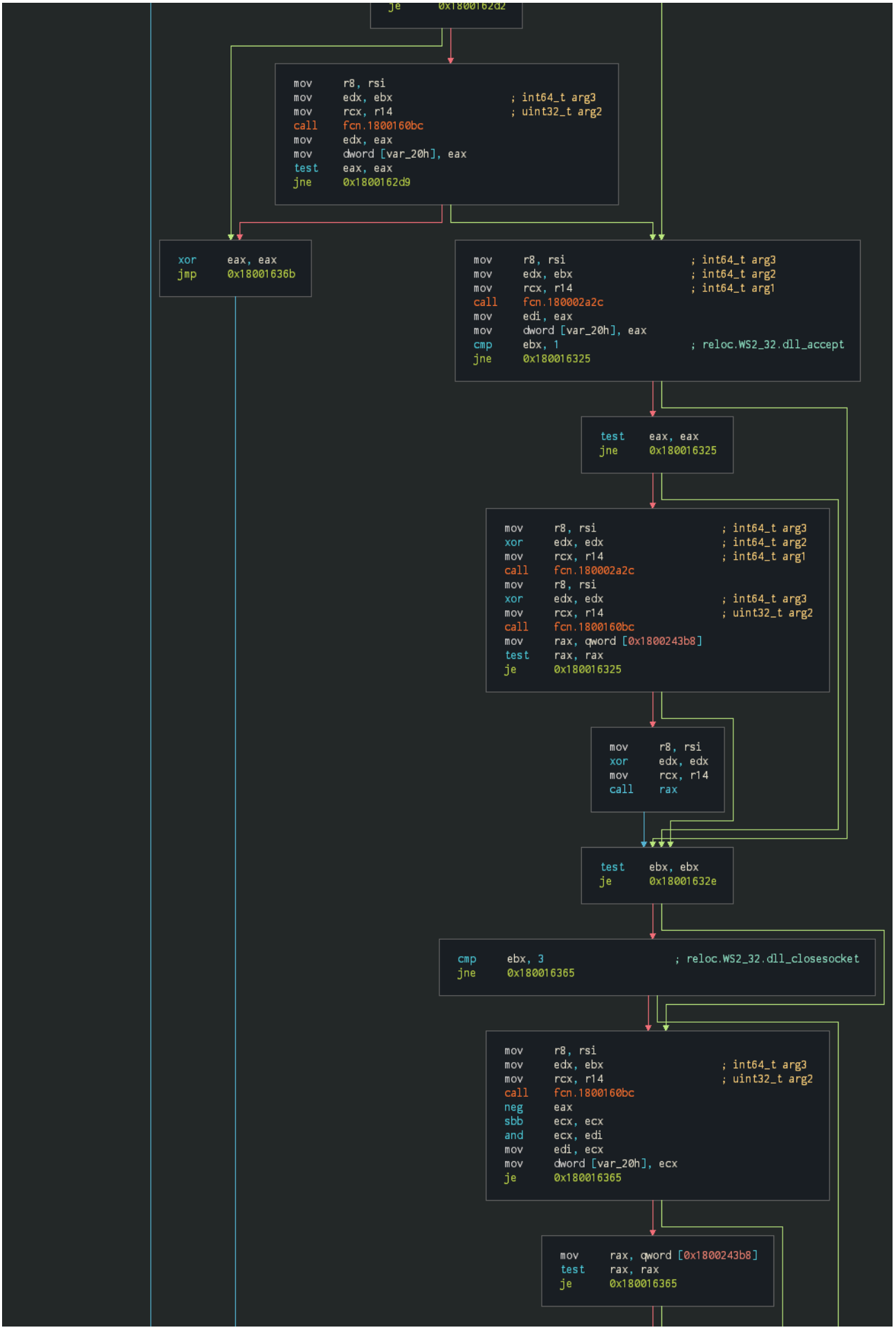
On seeing the assembly code of the header, we can see the multiples operation for parse by the stack pointer for load the export section which content the Meterpreter shellcode.

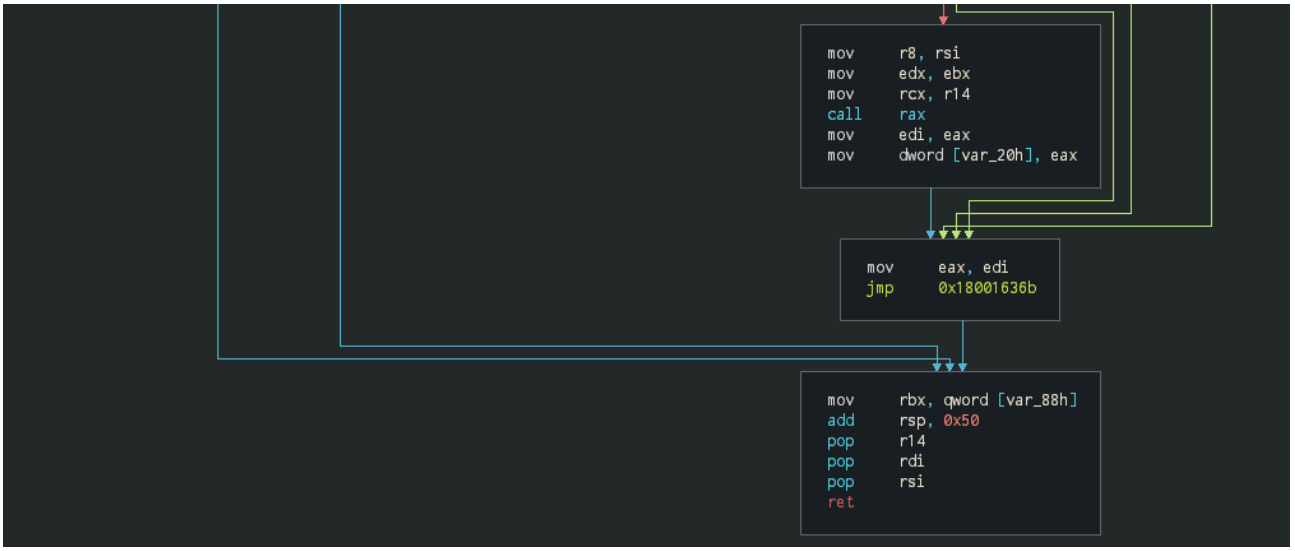


	Hex	Disasm
0	4D5A	POP R10
2	4152	PUSH R10
4	55	PUSH RBP
5	4889E5	MOV RBP, RSP
8	4883EC20	SUB RSP, 0X20
C	4883E4F0	AND RSP, 0XFFFFFFFFFFFFFFF0
10	E800000000	CALL 0X180000015
15	5B	POP RBX
16	4881C3EB180000	ADD RBX, 0X18EB
1D	FFD3	CALL RBX
1F	4881C300903000	ADD RBX, 0X30900
26	4989D8	MOV R8, RBX
29	6A40	PUSH 4
2B	5A	POP RDX
2C	FFD0	CALL RAX
2E	0000	ADD BYTE PTR [RAX], AL
30	0000	ADD BYTE PTR [RAX], AL
32	0000	ADD BYTE PTR [RAX], AL
34	0000	ADD BYTE PTR [RAX], AL
36	0000	ADD BYTE PTR [RAX], AL
38	0000	ADD BYTE PTR [RAX], AL
3A	0000	ADD BYTE PTR [RAX], AL
3C	E8000000E0	CALL 0X18E000041
41	1F	.BYTE 0X1F
42	BAE000B490	MOV EDX, 0X9B4000E
47	CD21	INT 0X21
49	B8104CCD21	MOV EAX, 0X21CD4C01
4E	54	PUSH RSP
4F	6869732070	PUSH 0X70207369
54	726F	JB SHORT 0X1800000C5
56	677261	JB SHORT 0X1800000BA
59	6D	INSD DWORD PTR [RDI], DX
5A	206361	AND BYTE PTR [RBX + 0X61], AH

We can note the characteristic entrypoint of Cobalt Strike with the three accepts calls and one close socket.


;-- rip:
347: entry0 (int64_t arg1, uint32_t arg2, int64_t arg3);
; var int64_t var_20h @ rsp+0x20
; var int64_t var_8h @ rsp+0x70
; var int64_t var_10h @ rsp+0x78
; var int64_t var_88h @ rsp+0x88
; arg int64_t arg1 @ rcx
; arg uint32_t arg2 @ rdx
; arg int64_t arg3 @ r8
mov qword [var_8h], rbx
mov qword [var_10h], rsi
push rdi
sub rsp, 0x20
mov rdi, r8 ; arg3
mov ebx, edx ; arg2
mov rsi, rcx ; arg1
cmp edx, 1 ; reloc.WS2_32.dll_accept ; arg2
jne 0x18001623d
call fcn.18001a254
mov r8, rdi
mov edx, ebx
mov rcx, rsi
mov rbx, qword [var_8h]
mov rsi, qword [var_10h]
add rsp, 0x20
pop rdi
jmp 0x18001625c
mov rax, rsp
mov qword [rax + 0x20], rbx
mov qword [rax + 0x18], r8
mov dword [rax + 0x10], edx
mov qword [rax + 8], rcx
push rsi
push rdi
push r14
sub rsp, 0x50
mov rsi, r8
mov ebx, edx
mov r14, rcx
mov edx, 1
mov dword [rax - 0x48], edx
test ebx, ebx
jne 0x180016299
cmp dword [0x1800341f8], ebx
jne 0x180016299
xor eax, eax
jmp 0x18001636b
lea eax, [rbx - 1]
cmp eax, 1 ; reloc.WS2_32.dll_accept
ja 0x1800162d9
mov rax, qword [0x1800243b8]
test rax, rax
je 0x1800162b7
mov edx, ebx
call rax
mov edx, eax
mov dword [var_20h], eax
test edx, edx


je 0x1800162d2
mov r8, rsi
mov edx, ebx ; int64_t arg3
mov rcx, r14 ; uint32_t arg2
call fcn.1800160bc
mov edx, eax
mov dword [var_20h], eax
test eax, eax
jne 0x1800162d9
xor eax, eax
jmp 0x18001636b
mov r8, rsi ; int64_t arg3
mov edx, ebx ; int64_t arg2
mov rcx, r14 ; int64_t arg1
call fcn.180002a2c
mov edi, eax
mov dword [var_20h], eax
cmp ebx, 1 ; reloc.WS2_32.dll_accept
jne 0x180016325
test eax, eax
jne 0x180016325
mov r8, rsi ; int64_t arg3
xor edx, edx ; int64_t arg2
mov rcx, r14 ; int64_t arg1
call fcn.180002a2c
mov r8, rsi
xor edx, edx ; int64_t arg3
mov rcx, r14 ; uint32_t arg2
call fcn.1800160bc
mov rax, qword [0x1800243b8]
test rax, rax
je 0x180016325
mov r8, rsi
xor edx, edx
mov rcx, r14
call rax
test ebx, ebx
je 0x18001632e
cmp ebx, 3 ; reloc.WS2_32.dll_closesocket
jne 0x180016365
mov r8, rsi
mov edx, ebx ; int64_t arg3
mov rcx, r14 ; uint32_t arg2
call fcn.1800160bc
neg eax
sbb ecx, ecx
and ecx, edi
mov edi, ecx
mov dword [var_20h], ecx
je 0x180016365
mov rax, qword [0x1800243b8]
test rax, rax
je 0x180016365



We can observe the SMB pipe used as pivoting method for the implant to run.

```
0x18000a7be lea r8, [rbx + 8]
0x18000a7c2 lea r9, str.s__pipe___s ; 0x180023a08
0x18000a7c9 lea rdx, [rbx + 9]
0x18000a7cd mov rcx, rax
0x18000a7d0 mov qword [rsp + 0x28], r14
0x18000a7d5 mov qword [rsi], rax
0x18000a7d8 mov qword [rsp + 0x20], r15
0x18000a7dd call fcn.180015054
0x18000a7e2 mov rcx, qword [rsi + 8]
0x18000a7e6 mov ebx, 0x57     ; 'W' ; 87
0x18000a7eb lea rax, [rcx - 1]
0x18000a7ef cmp rax, 0xfffffffffffffffd
0x18000a7f3 ja 0x18000a812
0x18000a7f5 lea rdx, [rsp + 0x450]
0x18000a7fd xor r9d, r9d
0x18000a800 xor r8d, r8d
0x18000a803 mov dword [rsp + 0x450], edi
0x18000a80a call qword [SetNamedPipeHandleState] ; 0x1800233f8 ; BOOL SetNamedPipeHandleState(HANDLE
0x18000a810 jmp 0x18000a84b
```

This collects the system informations and format for send it the previous node.

```
0x180007ff7 lea rcx, [rsp + 0x40]
0x180007ffc mov edx, 0x104    ; 260
0x180008001 call qword [GetSystemDirectoryW] ; 0x1800232a0 ; UINT GetSystemDirectoryW(LPWSTR lpBuffe
0x180008007 test eax, eax
0x180008009 je 0x1800080ba
0x18000800f lea edx, [rsi + 0x5c]
0x180008012 lea rcx, [rsp + 0x40]
```

```
0x180008017 mov dword [rsp + 0x480], 0x104 ; 260
0x180008022 call fcn.180015078
0x180008027 mov dword [rsp + 0x38], esi
0x18000802b mov qword [rsp + 0x30], rsi
0x180008030 lea r9, [rsp + 0x488]
0x180008038 lea rcx, [rsp + 0x40]
0x18000803d xor r8d, r8d
0x180008040 xor edx, edx
0x180008042 mov qword [rsp + 0x28], rsi
0x180008047 mov word [rax + 2], si
0x18000804b mov qword [rsp + 0x20], rsi
0x180008050 call qword [GetVolumeInformationW] ; 0x1800232b0 ; BOOL GetVolumeInformationW(LPCWSTR lp
0x180008056 lea rdx, [rsp + 0x480]
0x18000805e lea rcx, [rsp + 0x250]
0x180008066 call qword [GetComputerNameW] ; 0x1800232b8 ; BOOL GetComputerNameW(LPWSTR lpBuffer, LPD
0x18000806c mov ecx, dword [rsp + 0x488]
0x180008073 lea r8, [rsp + 0x250]
0x18000807b movzx eax, cx
0x18000807e mov qword [rsp + 0x30], r8
0x180008083 shr ecx, 0x10
0x180008086 mov edx, 0x104    ; 260
0x18000808b mov dword [rsp + 0x28], eax
0x18000808f mov dword [rsp + 0x20], ecx
0x180008093 lea rcx, [rsp + 0x40]
0x180008098 lea r9, str.04x__04x:_s ; 0x180023940 ; Format the data
0x18000809f lea r8d, [rdx - 1]
```

Looking at the TTPs and the anomaly on the PE header, I make the parallel with the APT chimera report, a group that targeted the semi-conductor sector in Taiwan. I had written the Yara rule with the full part of the anomaly and posted on Twitter.

Hunting

Few times after release a compact analysis, I think to use my Yara rule for hunting additionals samples with differents levels on condition, for detect if by example, a new variant reuse a part of the indicators (which can be the oldest or more recent). By the way of improving this specter of results and reduce the load on the Yara rule, I have removed a part of the anomaly just before the manipulation of the RSP (stack pointer).

Due to the numbers of results, I had only got last month of hits on Virustotal but quickly some different types of Cobalt Strike are identified in two major famillies :

- With the standard ReflectiveLoader reference in export table.
- Have not the reference but use custom way by ordinal or execute function.

The last one has been splited between recent (2019-2020) and old (2017-2018) for links to the period of samples analyzed on the chimera report (maybe a variant not analysed).

The first result in the compiled the informations on the samples in the different groups, show that multiple pairs of samples can observed with the same VHash, date of compilation of the DLL and size of the files. VHash being based on imports, exports and the header for the PE, this insensitive unlike a simple modification of an IP address of a payload and allow to confirm that reuse the code.

SHA-256	Vhash	File type	File size
c50a67746b3b10a5961f1dfbd1acccd52f0a9ff049fb47edf6e973c8f90bc185	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)
681412c7ead2a551a6ff11f0a25288629322a32d73009e07efd6a81972465260	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
c9e649f7ca9790834148caef3718b55ddd964ed9ea6c3b90e0fa5f34cfba37da	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
222a38b7a34bf52dea4bcd6b39d30a25b8b2485a684c42f702d237f2e09bfb29	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	196.57 KB (201283 bytes)
aae6502e18ec751262b79beadb77eb5d40ce6928484425bb5ef6c417189deecf	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
f9cebbde1d4c61fdce981c73d24274dbe3f2707f6f42f76fcabe689ebcb1965d	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
3d842f42a7caa4e088a4c7a28ef866a9ac1e0f75be929beed99cc73838ad8507	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
e00f032ddecf958b9ed4fbdd9ca52f44ed7b25a260ab08e842f8d4f174f8c344	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
b2ebbcd9700e0ac2e0b54e3599f95f389a6c206c2c1236287de48757c89b8f80	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
10b5ede60b9c5d7857a4462c4c3fd531b1793a37bd366f9cb6cb675289858aab	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
76e6b9102e44d048fcdcb4e567cdd50754fd3e952f76a5c1b4cfcec8ccbe129b	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	195.50 KB (200192 bytes)
e8b94f00131ffad10638c7f3e323ae501e2164b101f9544eb91678ffcf8eb6b9	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
f2de9a3fc0c1fc82ce1aa5c22bac552302da903840124b899131842a98f01bd6	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	198.00 KB (202752 bytes)
879ec7c5e7340c99c6a1380342cdc4d8440e94ea00ec5ba314fbf31bcad25003	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
f78d609f632431eeadfe724a9c2b050fc6cc17a1a7fd5363bf35b4391e0df5bc	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
339bd08af13367befff6cde3a8b32d863735710cac210758cb9d276ee43991e8	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	196.57 KB (201283 bytes)
cfc7b6a8ad0959f4ea3f6b6f09492ea93961938008b61279567f1bddf1a7bc06	125056655d15551158z8drza00166z1	Win32 DLL	254.88 KB (261000 bytes)
dd2192fa412326fdd33451a9329f3bc6e1d81808fcfb5648a7cad0cdf50393ce	125056651d15555143z42z78z1dz31z900156z1	Win32 DLL	197.00 KB (201728 bytes)
f6d89ff139f4169e8a67332a0fd55b6c9beda0b619b1332ddc07d9a860558bab	125056655d15555153z42z737z1dz31z900185z51	Win32 DLL	202.00 KB (206848 bytes)
7d4feeeb7bd6e05d4c15c1fbd892e7d3ff9ea8eedd02d5f426d30c8a1ba8a957	125056655d15555153z42z737z1dz31z900185z51	Win32 DLL	202.00 KB (206848 bytes)
56bdf2077e8e54e47f8a5a3102c4052a90db2d0871ce49afb293d78f42297222	125056655d1555129z8frza00166z1	Win32 DLL	254.50 KB (260609 bytes)
d03f975148e13019971f60857322ce49b923ae0cabd477cd282b97fdf3f906a3	125056655d1555129z8frza00166z1	Win32 DLL	256.00 KB (262144 bytes)
d352c4b9852fb132913f526cd9ae8d68291b288a30a3c5dfe810a1ea9ae851b1	125056655d1555129z8frza00166z1	Win32 DLL	256.00 KB (262144 bytes)

Now, this the time that each analyst hate, the time to found the samples (Ask to Virustotal theirs prices for get the samples and cry). Fortunately, almost a sample of each pair could be found on the public sandbox (36 samples on 74).

At the first sample analysed, the sample content the same combo Cobalt Strike and Meterpreter but have a persistence method by .NET client by local IP, localhost (in the infrastructure) or with an external IP or domain (initial compromisation point).

```
0x18002ca03  %s.4%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s
0x18003c104  樧,-/./...........................................................................................&.-/.
0x18002e0f0  R6031\r\n- Attempt to initialize the CRT more than once.\nThis indicates a bug in your application.\r\n
0x18002ca7b  %s.3%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s
0x18002ef6f   !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~
0x180032880   !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~
0x180032a00   !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~
0x18002d1cc  ppid %d is in a different desktop session (spawned jobs may fail). Use 'ppid' to reset.
0x18002d9e0  HTTP/1.1 200 OK\r\nContent-Type: application/octet-stream\r\nContent-Length: %d\r\n\r\n
0x18002d4ee  IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/'); %s
0x18002cade  %s.2%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s
0x18002d480  IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/')
0x18003c685  KVK.................................0.-.n\vYG@JG\\\vr]W]@OZGXKr\\[@JBB
0x18002cc6f  could not run command (w/ token) because of its length of %d bytes!
0x180038a50  ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
0x180038b00  0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz+/
0x18002cb24  %s.2%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x.%x%x.%s
0x180037bf0  SOFTWARE\\Wow6432Node\\Microsoft\\VisualStudio\\11.0\\Setup\\VC
0x180038aa0  abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq
0x1800378f0  A local variable was used before it was initialized\n\r
0x180037980  Runtime Check Error.\n\r Unable to display RTC Message.
0x18002cb62  %s.2%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x.%x%x.%s
0x18002dec0  R6024\r\n- not enough space for _onexit/atexit table\r\n
0x18002df80  R6026\r\n- not enough space for stdio initialization\r\n
0x18002dff0  R6027\r\n- not enough space for lowio initialization\r\n
0x18002e1c0  R6032\r\n- not enough space for locale information\r\n
0x18002e420  R6034\r\n- inconsistent onexit begin-end variables\r\n
0x18002d538  powershell -nop -exec bypass -EncodedCommand "%s"
0x18002cdcf  Could not open service control manager on %s: %d
0x18002d2e8  %d is an x64 process (can't inject x86 content)
0x18002d318  %d is an x86 process (can't inject x64 content)
0x18002d5fb  Failed to duplicate primary token for %d (%u)
0x18002d629  Failed to impersonate logged on user %d (%u)
0x0000004d   !This program cannot be run in DOS mode.\r\r\n$
```

In searching in the archives that match with the TTPs and the strings, I found the Yara rule of APT19 that use a combo Cobalt Strike + Meterpreter as implant for pivoting the infrastructure of the victim.

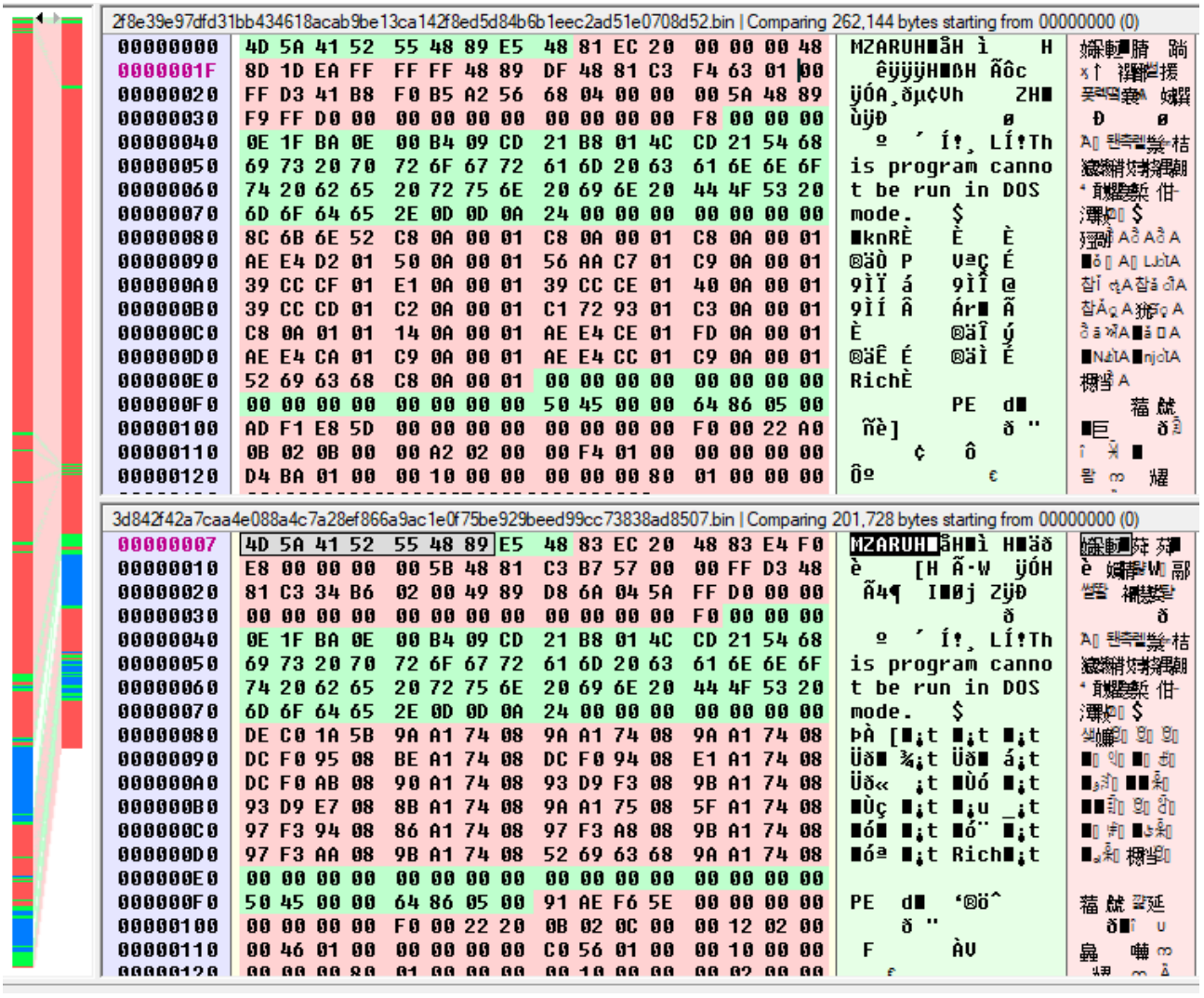
```
/*
   Yara Rule Set
   Author: Ian.Ahl@fireeye.com @TekDefense, modified by Florian Roth
   Date: 2017-06-05
   Identifier: APT19
   Reference: https://www.fireeye.com/blog/threat-research/2017/06/phished-at-the-request-of-counsel.html
*/

rule Beacon_K5om {
   meta:
      description = "Detects Meterpreter Beacon - file K5om.dll"
      license = "https://creativecommons.org/licenses/by-nc/4.0/"
      author = "Florian Roth"
      reference = "https://www.fireeye.com/blog/threat-research/2017/06/phished-at-the-request-of-counsel.html"
      date = "2017-06-07"
      hash1 = "e3494fd2cc7e9e02cff76841630892e4baed34a3e1ef2b9ae4e2608f9a4d7be9"
   strings:
      $x1 = "IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:%u/'); %s" fullword ascii
      $x2 = "powershell -nop -exec bypass -EncodedCommand \"%s\"" fullword ascii
      $x3 = "%d is an x86 process (can't inject x64 content)" fullword ascii

      $s1 = "Could not open process token: %d (%u)" fullword ascii
      $s2 = "0fd00b.dll" fullword ascii
      $s3 = "%s.4%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%08x%08x%08x%08x%08x%08x%08x.%x%x.%s" fullword ascii
      $s4 = "Could not connect to pipe (%s): %d" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and filesize < 600KB and ( 1 of ($x*) or 3 of them ) )
}
```

This uses an well-known fileless UAC bypass using Event Viewer technique and maintain the persistence in the key, this spawn a Meterpreter instance in loading the DLL inside the beacon, we can recognize the part for initiating the communication in getting the system informations.


196: InitComToC2 (int64_t arg1, int64_t arg2);
; var sockaddr *name @ rsp+0x20
; var u_short var_22h @ rsp+0x22
; var int64_t var_24h @ rsp+0x24
; var int64_t var_30h @ rsp+0x30
; var int64_t var_18h @ rsp+0x70
; var int64_t var_20h @ rsp+0x78
; arg int64_t arg1 @ rcx
; arg int64_t arg2 @ rdx
mov qword [var_18h], rbx
mov qword [var_20h], rbp
push rsi
push rdi
push r14
sub rsp, 0x40
mov rax, qword [0x18003d030]
xor rax, rsp
mov qword [var_30h], rax
xor r8d, r8d ; int protocol
mov esi, edx ; arg2
mov rdi, rcx ; arg1
lea r14d, [r8 + 2]
lea edx, [r8 + 1]
mov ecx, r14d ; int af
call sub.WS2_32.dll_socket ; SOCKET socket(int af, int type, int protocol)
or rbp, 0xffffffffffffffff
mov rbx, rax
cmp rax, rbp
jne 0x18000ee02
mov rcx, rdi ; const char *name
call sub.WS2_32.dll_gethostbyname ; hostent *gethostbyname(const char *name)
test rax, rax
je 0x18000edfd
mov rdx, qword [rax + 0x18]
movsx r8, word [rax + 0x12]
lea rcx, [var_24h]
mov rdx, qword [rdx]
call fcn.180019c80
movzx ecx, si ; u_short hostshort
mov word [name], r14w
call sub.WS2_32.dll_htons ; u_short htons(u_short hostshort)
lea rdx, [name] ; const sockaddr *name
mov r8d, 0x10 ; reloc.WS2_32.dll_recv ; int namelen
mov rcx, rbx ; SOCKET s
mov word [var_22h], ax
call sub.WS2_32.dll_connect ; int connect(SOCKET s, const sockaddr *name, int namelen)
test eax, eax
je 0x18000ee59
mov rcx, rbx ; SOCKET s
call sub.WS2_32.dll_closesocket ; int closesocket(SOCKET s)
jmp 0x18000edfd
mov rax, rbx
mov rax, rbp
jmp 0x18000ee5c
mov rcx, qword [var_30h]
xor rcx, rsp
call fcn.18001a200

```
mov     rbp, qword [var_20h]
add     rsp, 0x40
pop     r14
pop     rdi
pop     rsi
ret
```

But now this beginning to become interesting, in comparing the both PE, we can observe a lot of differences on the structures of the payload due to the comparison is between each byte on the sequence order but the structure have common bytes in the anomaly in the header path.



We can see the differences on the implementation of the stack pointer in using destination index for copy the data of the instructions for load the shellcode of the Meterpreter DLL.

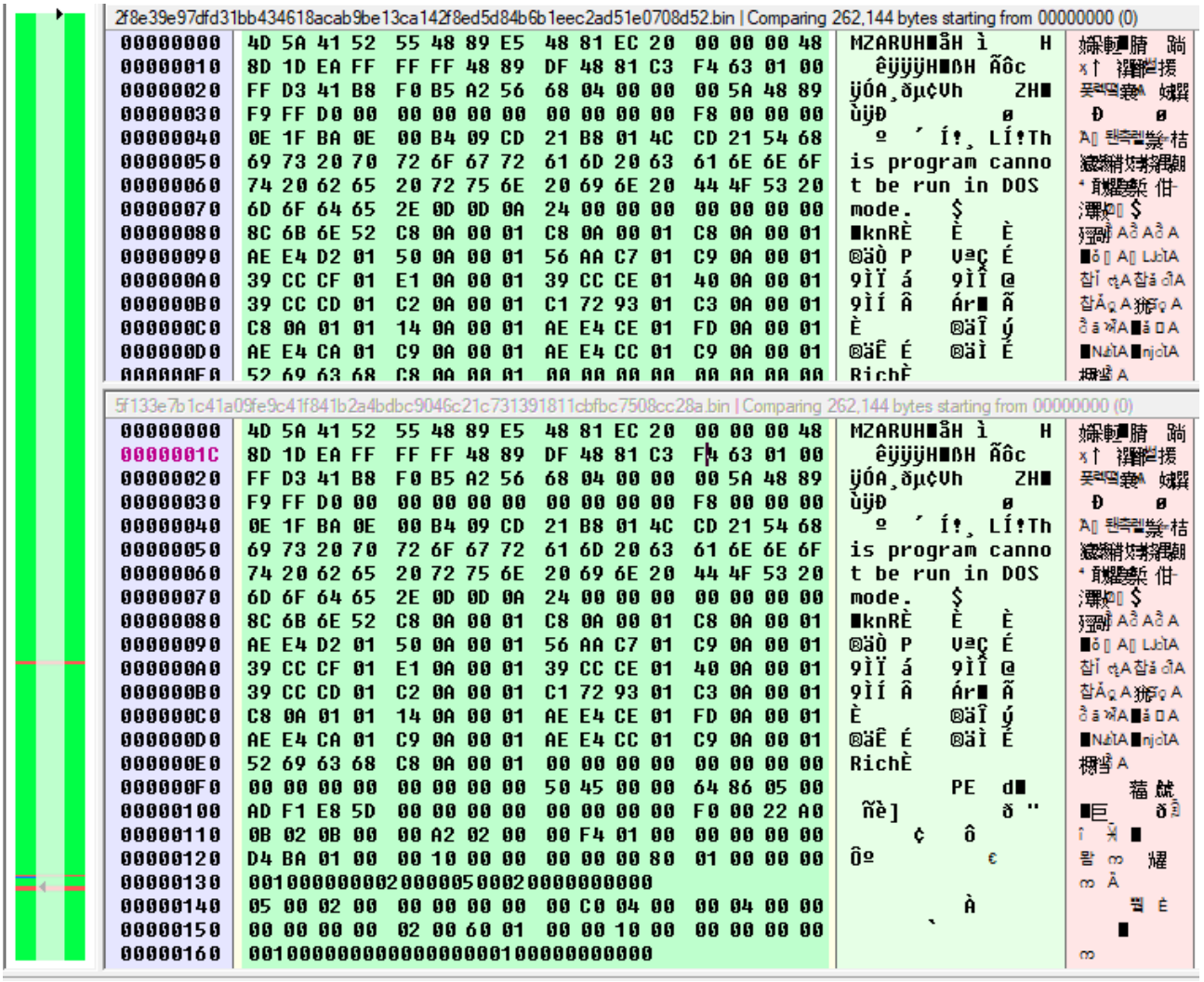
	Hex	Disasm
0	4D5A	POP R10
2	4152	PUSH R10
4	55	PUSH RBP
5	4889E5	MOV RBP, RSP
8	4881EC20000000	SUB RSP, 0X20
F	488D1DEAFFFFFF	LEA RBX, [RIP - 0X16]
16	4889DF	MOV RDI, RBX
19	4881C33C6E1000	ADD RBX, 0X16E3C
20	FFD3	CALL RBX
22	41B8F0B5A256	MOV R8D, 0X56A2B5F0
28	6840000000	PUSH 4
2D	5A	POP RDX
2E	4889F9	MOV RCX, RDI
31	FFD0	CALL RAX
33	0000	ADD BYTE PTR [RAX], AL
35	0000	ADD BYTE PTR [RAX], AL
37	0000	ADD BYTE PTR [RAX], AL
39	0000	ADD BYTE PTR [RAX], AL
3B	00F0	ADD AL, DH
3D	0000	ADD BYTE PTR [RAX], AL
3F	005C55C0	ADD BYTE PTR [RBP + RDX*2 + 0XC], BL
43	7563	JNE SHORT 0X1800000A8
45	EF	OUT DX, EAX
46	98	CWDE
47	B07FC4	OR EDI, DWORD PTR [RDI - 0X3C]
4A	5C	POP RSP
4B	BA57568E5A	MOV EDX, 0X5A8E5657
50	1DA09E83F3	SBB EAX, 0XF3839EA0
55	AE	SCASB AL, BYTE PTR [RDI]
56	36A054D878B23AAFEC…	MOVABS AL, BYTE PTR SS:[0XD4ECAF3AB278D854]
60	13C1	ADC EAX, ECX
62	4641CAE059	RETF 0X590E
67	B76B	MOV BH, 0X6B

After this I have created a little script for extract each first part of PE header (4D 5A to 00 00 00 0E), get all unique the signature, attribute an ID to the signature an this time, attribute all the ID generated to the samples that have the same signature for display pairs of samples with the same modifications. On the results, we note all the samples have splited in two sections in having the same similarities in the header of the PE (here on the samples with content the ReflectiveLoader reference).

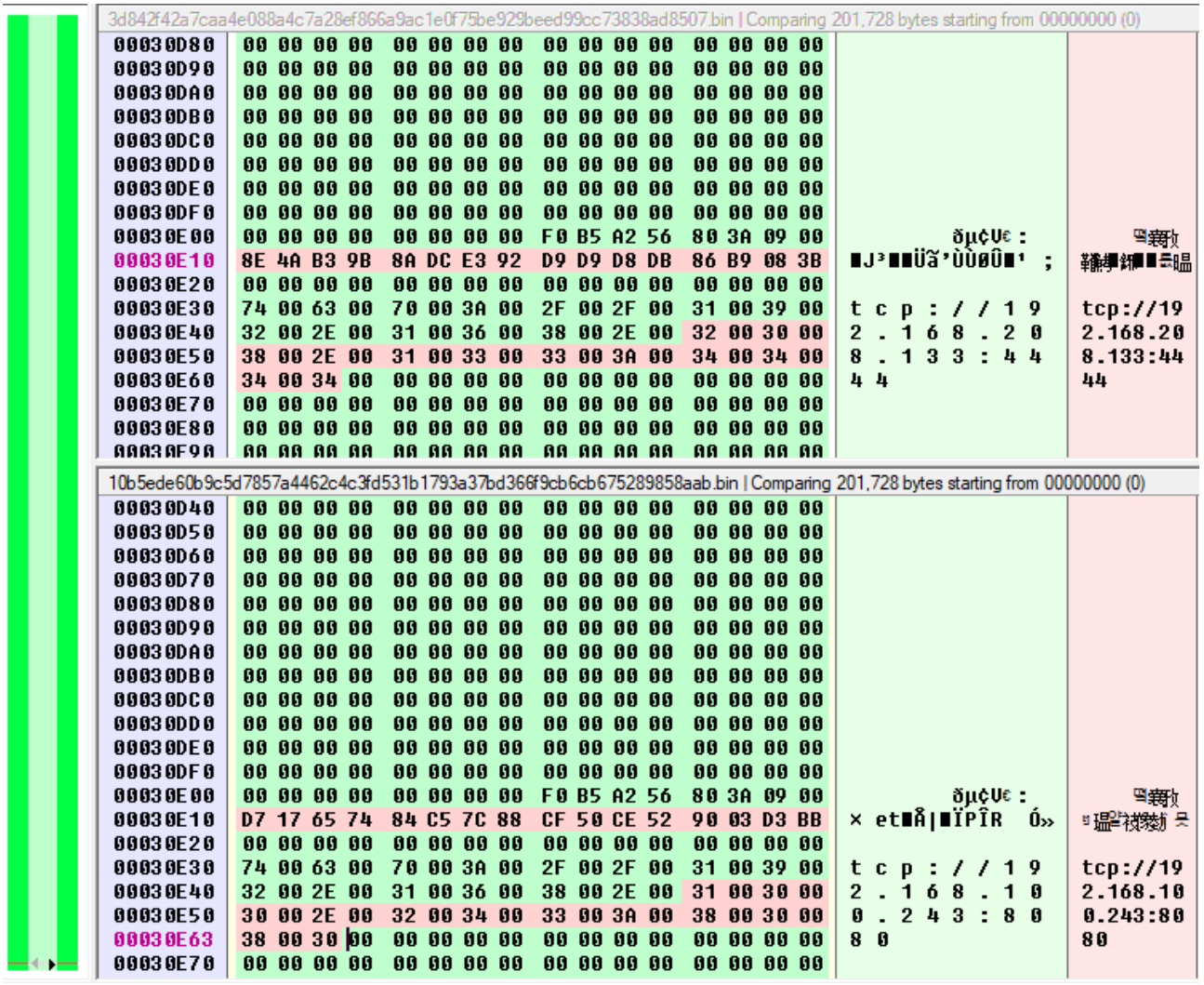
ID	Hash	Signature
0	ed4043b9a410016fb57c57cefb8bda4eeef1b222194fd68eb17650e353a4eea4	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
1	cfc7b6a8ad0959f4ea3f6b6f09492ea93961938008b61279567f1bddf1a7bc06	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
2	d352c4b9852fb132913f526cd9ae8d68291b288a30a3c5dfe810a1ea9ae851b1	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
2	d03f975148e13019971f60857322ce49b923ae0cabd477cd282b97fdf3f906a3	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
2	5f133e7b1c41a09fe9c41f841b2a4bdbc9046c21c731391811cbfbc7508cc28a	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
2	2f8e39e97dfd31bb434618acab9be13ca142f8ed5d84b6b1eec2ad51e0708d52	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
2	f625ac3b2c790e92810a05823a5ea8ce4c9741278a377c3f7e69b65a33affa04	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d
3	c50a67746b3b10a5961f1dfbd1acccd52f0a9ff049fb47edf6e973c8f90bc185	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	e8b94f00131ffad10638c7f3e323ae501e2164b101f9544eb91678ffcf8eb6b9	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	f9cebbde1d4c61fdce981c73d24274dbe3f2707f6f42f76fcabe689ebcb1965d	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	e00f032ddecf958b9ed4fbdd9ca52f44ed7b25a260ab08e842f8d4f174f8c344	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	3d842f42a7caa4e088a4c7a28ef866a9ac1e0f75be929beed99cc73838ad8507	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	222a38b7a34bf52dea4bcd6b39d30a25b8b2485a684c42f702d237f2e09bfb29	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	76e6b9102e44d048fcdcb4e567cdd50754fd3e952f76a5c1b4cfcec8ccbe129b	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	10b5ede60b9c5d7857a4462c4c3fd531b1793a37bd366f9cb6cb675289858aab	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
4	b2ebbcd9700e0ac2e0b54e3599f95f389a6c206c2c1236287de48757c89b8f80	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
5	f6d89ff139f4169e8a67332a0fd55b6c9beda0b619b1332ddc07d9a860558bab	4d 5a 41 52 55 48 89 e5 48 83 ec 20 48 83 e4 f0 e8
6	399a07f32a3d29c3feac66fe71fc6694d456f8de4894f92743f4e9031500b9e9	4d 5a 41 52 55 48 89 e5 48 81 ec 20 00 00 00 48 8d

By seeing the comparison between several samples of the same pair, we can note a code reuse at 98% between each sample, only the 2% which remains are due to the declaration or not of the IP address or domain for the pivot. This explains by the fact of the sample as compiled at the same time or use the same template like Cobalt Strike is a template that can be edited for use a custom DLL to load. Here on a pair of the Chimera samples :

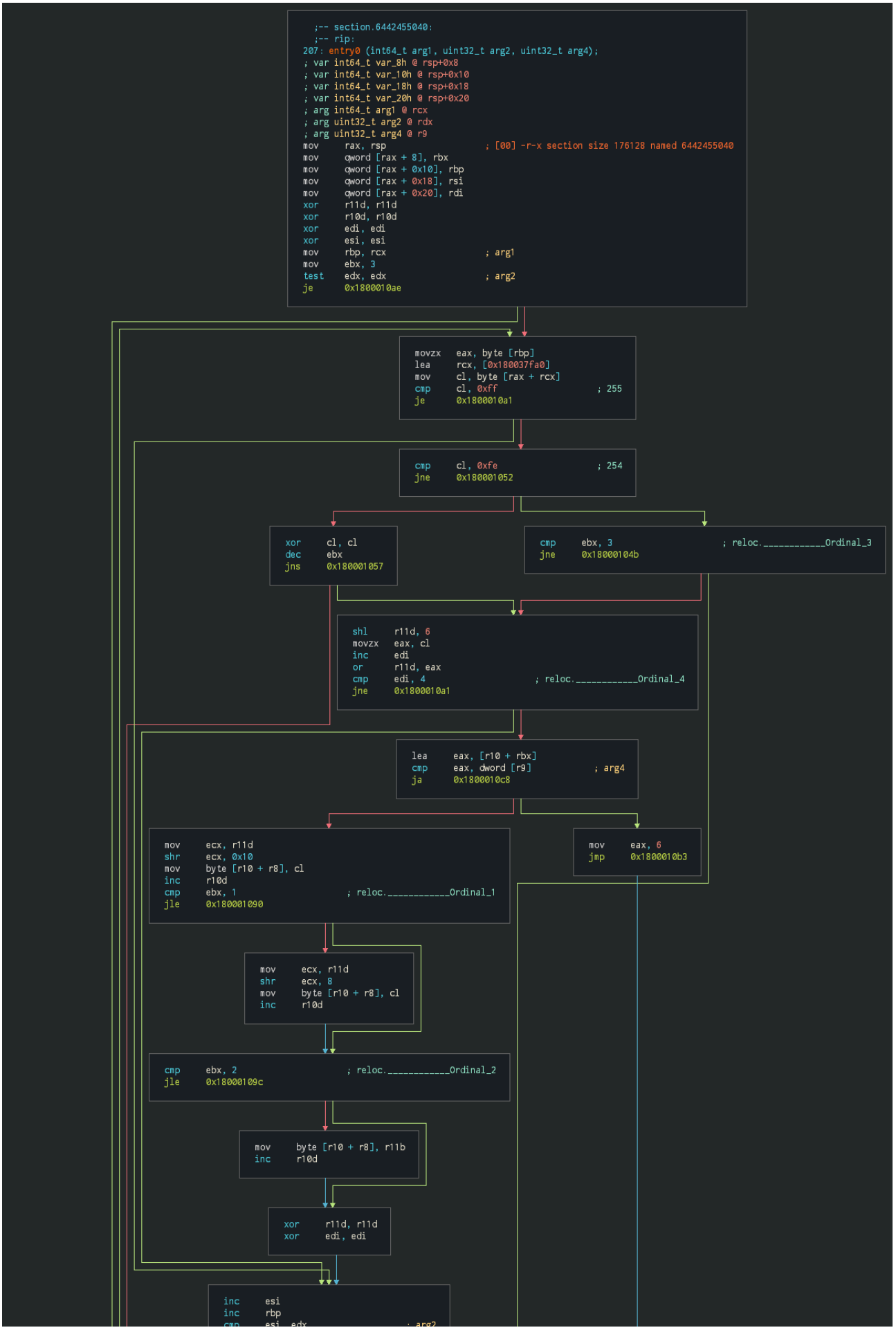
Same result on a pair of the APT19 samples :

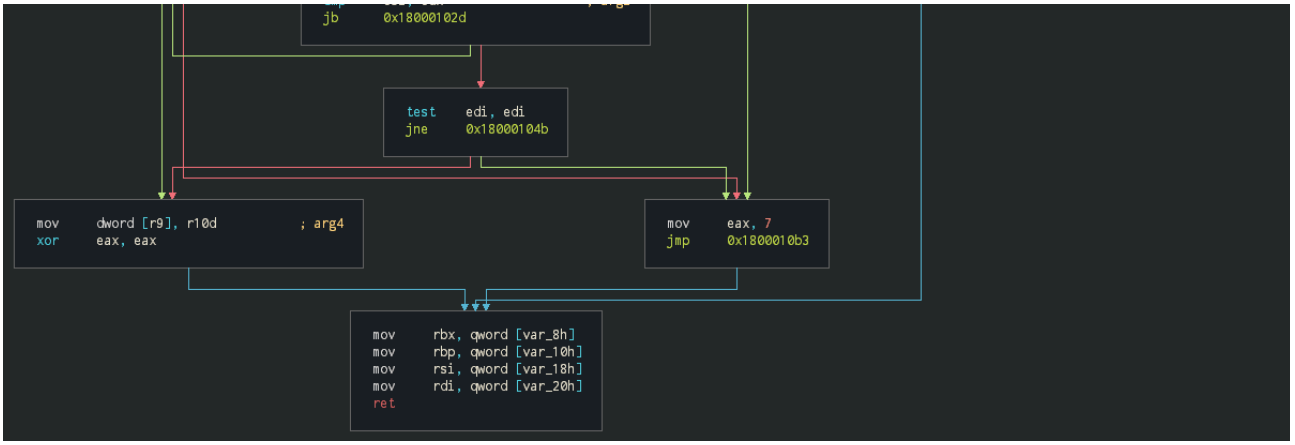


Liking said previously only the configuration change but the rest is the same due to this build on a template.

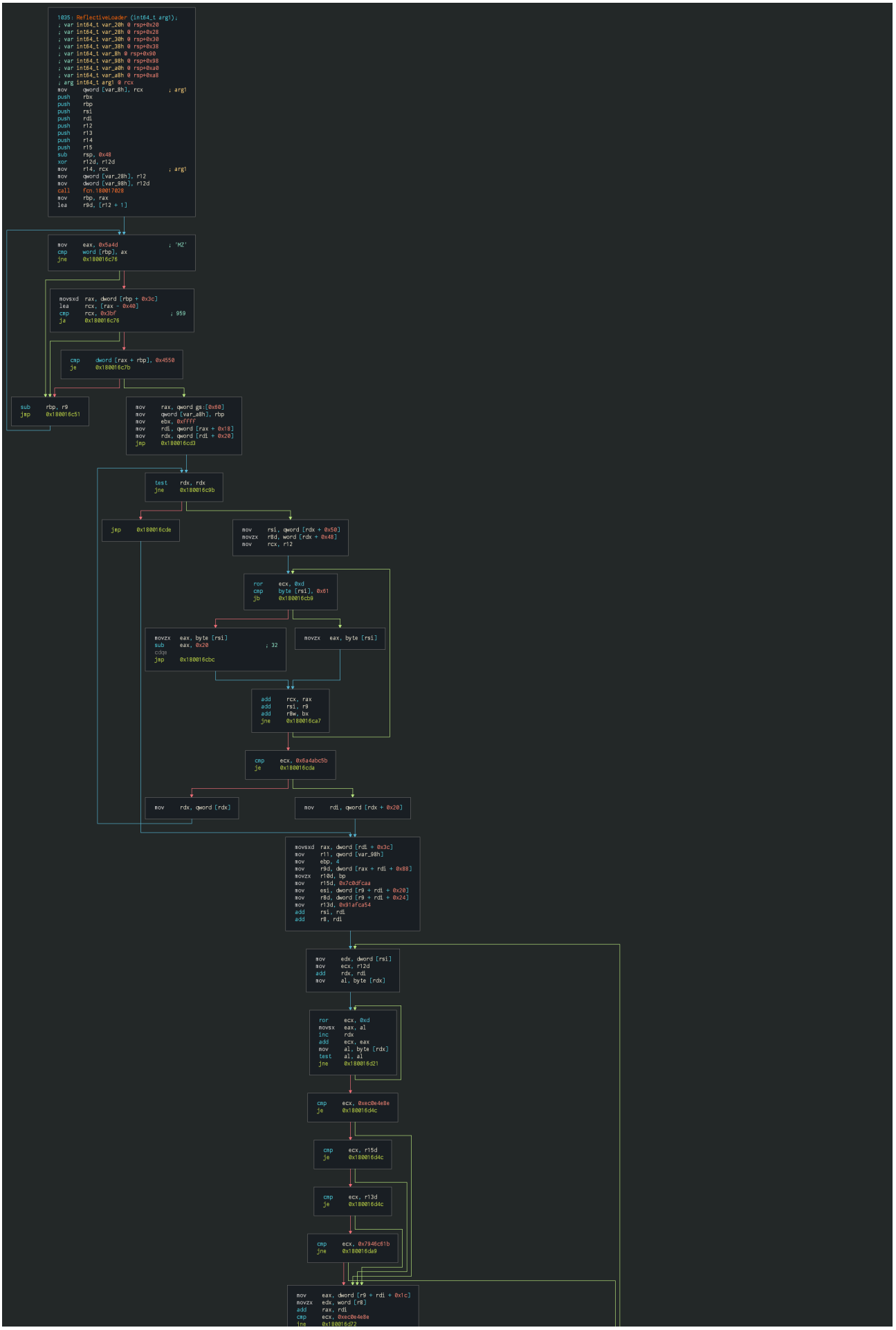


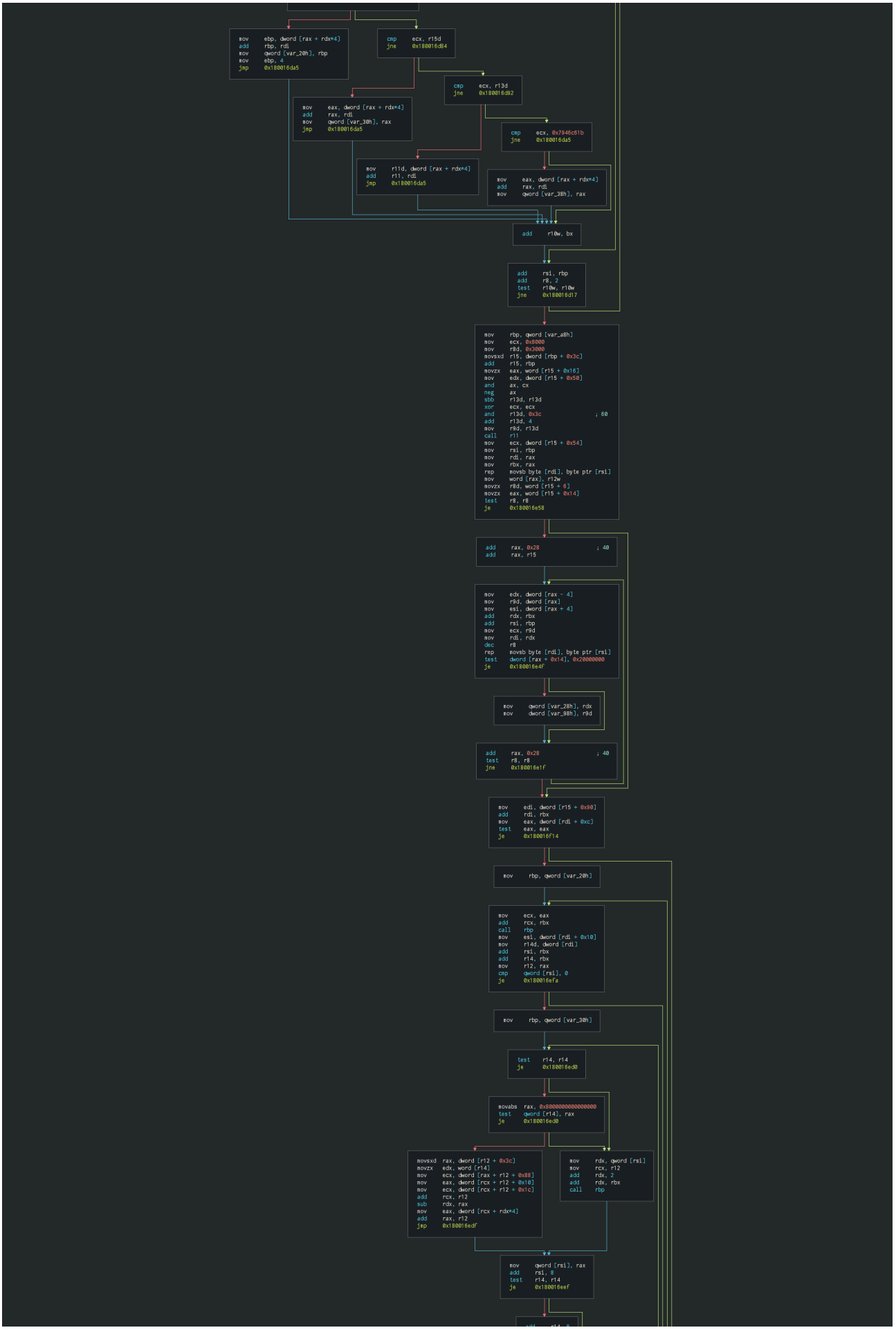
Few times after the report of APT19, the group have deleted the export reference in using ordinal way used for allow to use the beacon of Cobalt Strike with a custom DLL. This has by example rename as "execute".

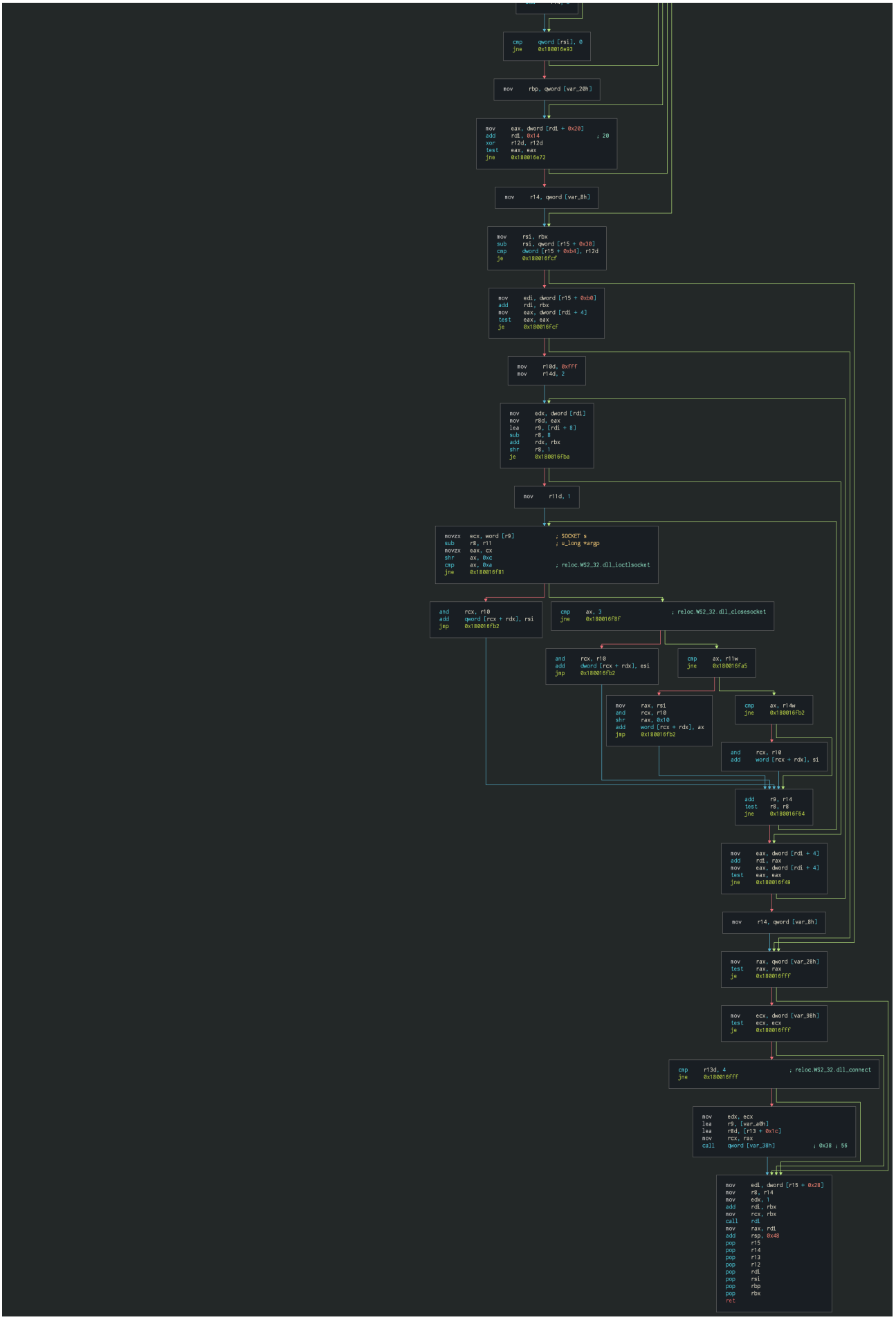

;-- section.6442455040:
;-- rip:
207: entry0 (int64_t arg1, uint32_t arg2, uint32_t arg4);
; var int64_t var_8h @ rsp+0x8
; var int64_t var_10h @ rsp+0x10
; var int64_t var_18h @ rsp+0x18
; var int64_t var_20h @ rsp+0x20
; arg int64_t arg1 @ rcx
; arg uint32_t arg2 @ rdx
; arg uint32_t arg4 @ r9
mov rax, rsp ; [00] -r-x section size 176128 named 6442455040
mov qword [rax + 8], rbx
mov qword [rax + 0x10], rbp
mov qword [rax + 0x18], rsi
mov qword [rax + 0x20], rdi
xor r11d, r11d
xor r10d, r10d
xor edi, edi
xor esi, esi
mov rbp, rcx ; arg1
mov ebx, 3
test edx, edx ; arg2
je 0x1800010ae
movzx eax, byte [rbp]
lea rcx, [0x180037fa0]
mov cl, byte [rax + rcx]
cmp cl, 0xff ; 255
je 0x1800010a1
cmp cl, 0xfe ; 254
jne 0x180001052
xor cl, cl
dec ebx
jns 0x180001057
cmp ebx, 3 ; reloc._____Ordinal_3
jne 0x18000104b
shl r11d, 6
movzx eax, cl
inc edi
or r11d, eax
cmp edi, 4 ; reloc._____Ordinal_4
jne 0x1800010a1
lea eax, [r10 + rbx]
cmp eax, dword [r9] ; arg4
ja 0x1800010c8
mov ecx, r11d
shr ecx, 0x10
mov byte [r10 + r8], cl
inc r10d
cmp ebx, 1 ; reloc._____Ordinal_1
jle 0x180001090
mov eax, 6
jmp 0x1800010b3
mov ecx, r11d
shr ecx, 8
mov byte [r10 + r8], cl
inc r10d
cmp ebx, 2 ; reloc._____Ordinal_2
jle 0x18000109c
mov byte [r10 + r8], r11b
inc r10d
xor r11d, r11d
xor edi, edi
inc esi
inc rbp



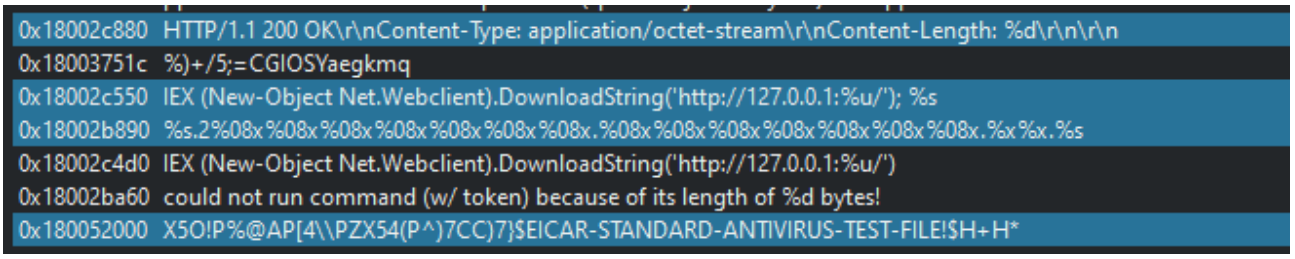
The group use this way only for changing the static reference in the export table but kept the Meterpreter DLL as implant to run.


1035: ReflectiveLoader (int64_t arg1);
mov eax, 0x5a4d ; 'MZ'
cmp word [rbp], ax
jne 0x180016c76
movsxd rax, dword [rbp + 0x3c]
lea rcx, [rax - 0x40]
cmp rcx, 0x3bf ; 959
ja 0x180016c76
cmp dword [rax + rbp], 0x4550
je 0x180016c7b
sub rbp, r9
jmp 0x180016c51
mov rax, qword gs:[0x60]
mov qword [var_a8h], rbp
mov ebx, 0xffff
mov rdi, qword [rax + 0x18]
mov rdx, qword [rdi + 0x20]
jmp 0x180016cd3
test rdx, rdx
jne 0x180016c9b
jmp 0x180016cde
mov rsi, qword [rdx + 0x50]
movzx r8d, word [rdx + 0x48]
mov rcx, r12
ror ecx, 0xd
cmp byte [rsi], 0x61
jb 0x180016cb9
movzx eax, byte [rsi]
movzx eax, byte [rsi]
sub eax, 0x20 ; 32
cdqe
jmp 0x180016cbc
add rcx, rax
add rsi, r9
add r8w, bx
jne 0x180016ca7
cmp ecx, 0x6a4abc5b
je 0x180016cda
mov rdx, qword [rdx]
mov rdi, qword [rdx + 0x20]
movsxd rax, dword [rdi + 0x3c]
mov r11, qword [var_98h]
mov ebp, 4
mov r9d, dword [rax + rdi + 0x88]
movzx r10d, bp
mov r15d, 0x7c0dfcaa
mov esi, dword [r9 + rdi + 0x20]
mov r8d, dword [r9 + rdi + 0x24]
mov r13d, 0x91afca54
add rsi, rdi
add r8, rdi
mov edx, dword [rsi]
mov ecx, r12d
add rdx, rdi
mov al, byte [rdx]
ror ecx, 0xd
movsx eax, al
inc rdx
add ecx, eax
mov al, byte [rdx]
test al, al
jne 0x180016d21
cmp ecx, 0xec0e4e8e
je 0x180016d4c
cmp ecx, r15d
je 0x180016d4c
cmp ecx, r13d
je 0x180016d4c
cmp ecx, 0x7946c61b
jne 0x180016da9
mov eax, dword [r9 + rdi + 0x1c]
movzx edx, word [r8]
add rax, rdi
cmp ecx, 0xec0e4e8e
jne 0x180016d22





Some samples tagged as APT 19 have the EICAR-TEST string to suggest a detection of a test software for the SOC managers of the targeted companies.We must not forget that if now this technique can be trival and should be notified to fight against distraction measures towards the detection of the tool, in 2016 - 2017, it isn't so well known and was very effective during the pentests so for APT, I'll let you guess.



The most recent samples on the same family of APT 19 hide theirs references to the ReflectiveLoader reference in going to the Ordinal way for the custom DLL few time after have been reported by Threat Intelligences companies on theirs reports.The most recent Chimera samples have done the same modification since 1st August 2020 in using External domain or IP, Internal IP or localhost for have an elevated session like on Active Directory machines.

SHA-256	Vhash	File type	File size	Filename	Creation Time	First Submission
4644e922a0a46e560f1115b8078ee6978568d2d838645b84293cdb6f8c797fff	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.puti	2020-09-04 19:37:33	2020-09-30 18:23:00
ccd14c31dd98d9c7c3de77437d440c8120eb445ecac828a7fed984c991ad65cd	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.mrmv	2020-09-04 19:37:33	2020-09-30 17:47:33
8a343368941ce2c500224256a96aec952b00786b2500746ac184553d99b9f912	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	197.00 KB (201728 bytes)	.ebtn	2020-09-04 19:37:33	2020-09-22 16:06:56
822c5be1861c4df935db5d0b7b045f9bc7847f06b2f626a798905899e3f0a1b5	115056651d15555143z32z717z1dz31z900157z	Win32 DLL	194.50 KB (199168 bytes)	upload.bin	2020-09-04 19:37:33	2020-09-18 08:28:49
da0d8dc8a3c034275d3a98471009dc65fc54afda5fc4f36a778c060e4113c429	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.fgdk	2020-09-04 19:37:33	2020-09-16 20:55:21
57557d0f6a3989d9676e92607b6d6f700930c26f41f12d47bee79c5df0913334	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	197.00 KB (201728 bytes)	.lqjj	2020-09-04 19:37:33	2020-09-15 02:52:27
cc02448dbfe5290451ff2f7f13f96b96590d31774c3c72e6b2e236e7755dbd31	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	195.57 KB (200262 bytes)	Xeexe.raw	2020-08-01 03:10:57	2020-09-27 08:07:44
2a9523e7dbe78ae48f1f46f8c549e1163e46f534a6567c9b41fde8c6d1936be1	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	197.00 KB (201728 bytes)	.mmxr	2020-08-01 03:10:57	2020-09-23 22:31:04
44f04b808cff6ed4143ba65e5ce84624eb9811abb0e8338bcbf5d41382aee5a3	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.59 KB (201308 bytes)	Sample.bin	2020-08-01 03:10:57	2020-09-22 08:32:53
fbe327350c11038f64cec12eb7343ac2dcfcc66ced70a8216f9f8053479edbb3	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.vonl	2020-08-01 03:10:57	2020-09-22 02:03:41
24dc59a7ea8f08318200eacc44b4044d984e68d86f3f98f72477059789ea0466	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	197.00 KB (201728 bytes)	.tung	2020-08-01 03:10:57	2020-09-11 10:10:51
f7d8e3458210963963742f5c66527ed3a9e465e2410a3343fe5487a934e85d44	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	197.00 KB (201728 bytes)	.lfev	2020-08-01 03:10:57	2020-09-08 20:52:56
6785dfb411255cb0a0d16bdcca68f3bb71e193694b34e997562acfc4a9baedae	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.nojv	2020-08-01 03:10:57	2020-09-07 16:37:19
801cac0879575ea2cf5dafd72d1676836c3ac8bc4264635c4461c3ee90a79297	125056651d15555143z32z717z1dz31z900157z	Win32 DLL	196.00 KB (200704 bytes)	.ydbp	2020-08-01 03:10:57	2020-09-07 08:14:05

```
https://112.213.98.44:8443/yolZSbt0qhZjjGKOPOXInwsGAF4fh-ug_DJWthkcIw248sAYaksYdEMF9AfLWAxNLZeL0cqpKH90RWpcWyu
tcp://192.168.233.129:4444
tcp://hash-37257.portmap.io:37257
```

Difficult to say if the both groups are the same but a lot of commons behaviour and TTPs can be observed. I estimated that more 200 samples have been detected by the Thor rule as Chimera in the last six months can be also linked to APT19 samples that detected by the common part of the anomaly on the header. On compiling all the data, we can see the common part and the little variant code but also that match with the VHash and pairs that we have detected at the beginning of the analysis.

A list of data that can be queried is available [here]

It should be remembered that the way groups linked to government work are sporadic groups linked only to a project like small teams. With this in mind, it is easy to recognize similarities because they are probably the same people and as soon as there is a different news, this classed as a new APT group but nobody remembers Thrip, Calypso ... that use lot similatires with APT10 or APT3 but have just a RAT or a small modification of a PE ?

References

- Meterpreter - Memory Indicators, Detection & Tooling
- Ref Yara APT19
- Privileges and Credentials: Phished at the Request of Counsel
- Custom DLL injection with Cobalt Strike's Beacon Object Files

---

Source: https://github.com/StrangerealIntel/CyberThreatIntel/blob/master/China/APT/Chimera/Analysis.md