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 0x00006b4dcompiled 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$
```

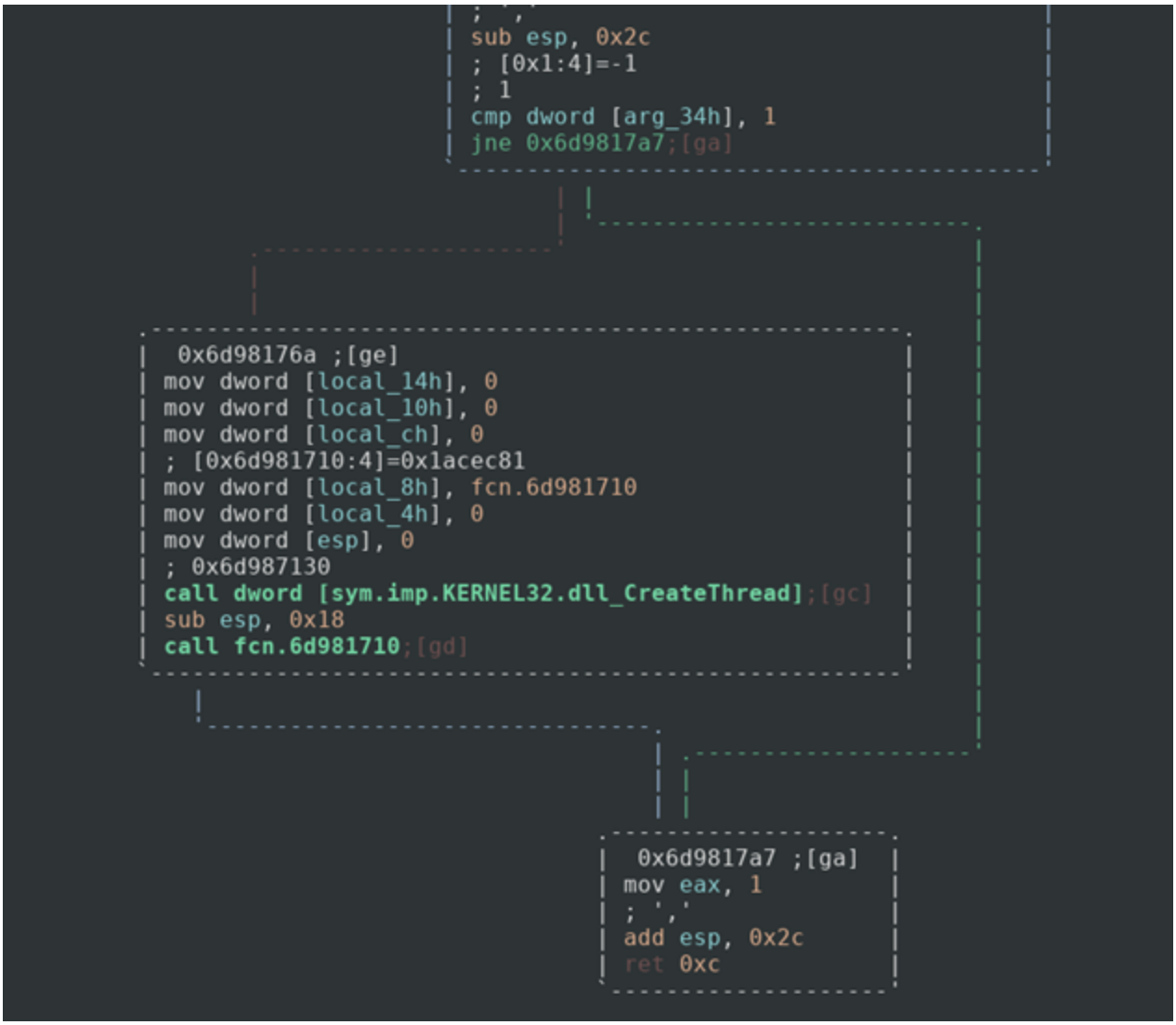
We can see that this is a Windows DLL file. This means it must have some exports that it wants us to run it with. Rabin2 can help us identify these exports.

```
$ 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.

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

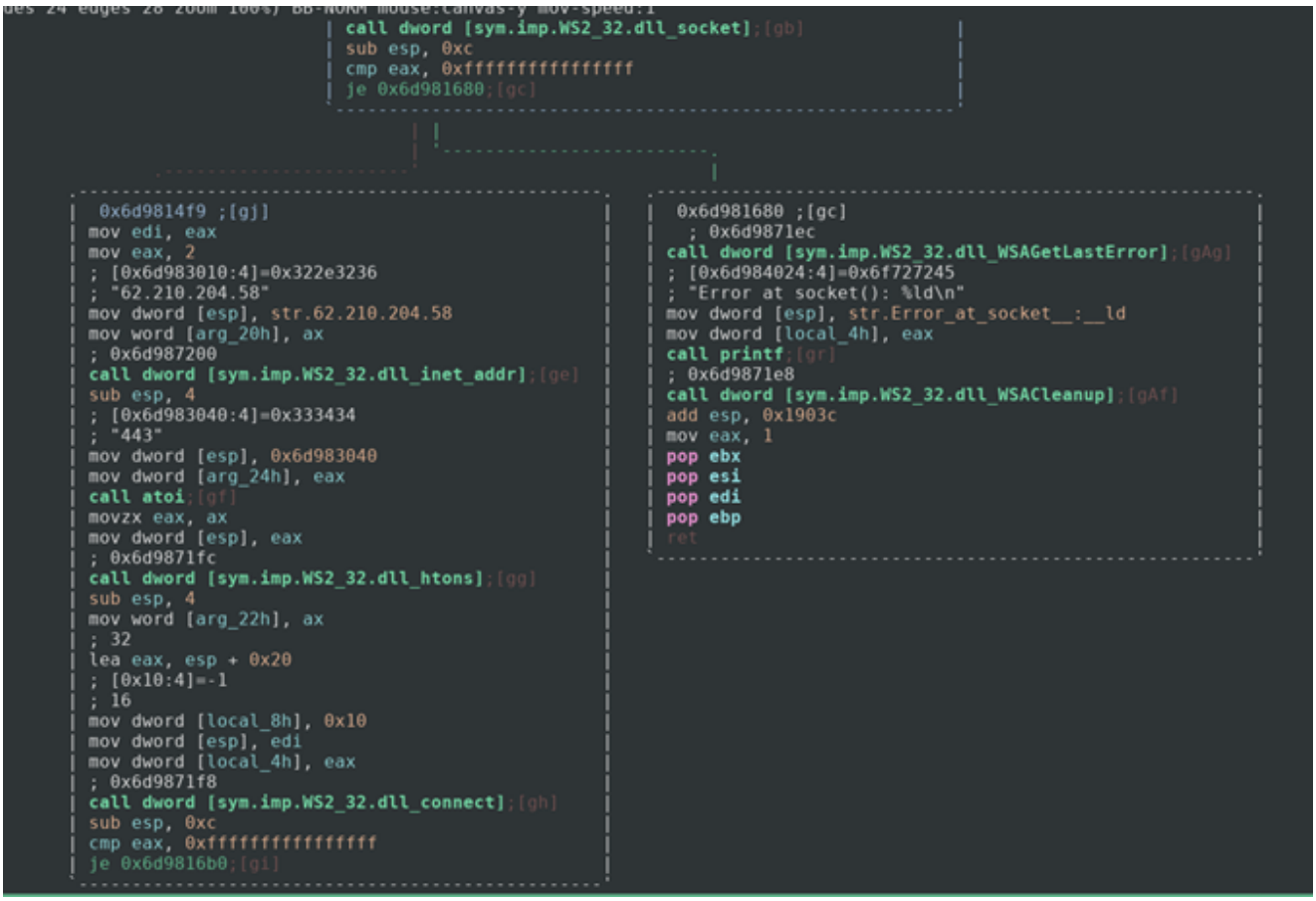


So it looks like it compares an argument to the number 1, if the statement is false then we return and most likely exit the program. If the statement is true, however, then we call the CreateThread function with fcn.6d981710 as a parameter. This would create a thread that would run whatever fcn.6d981710 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.

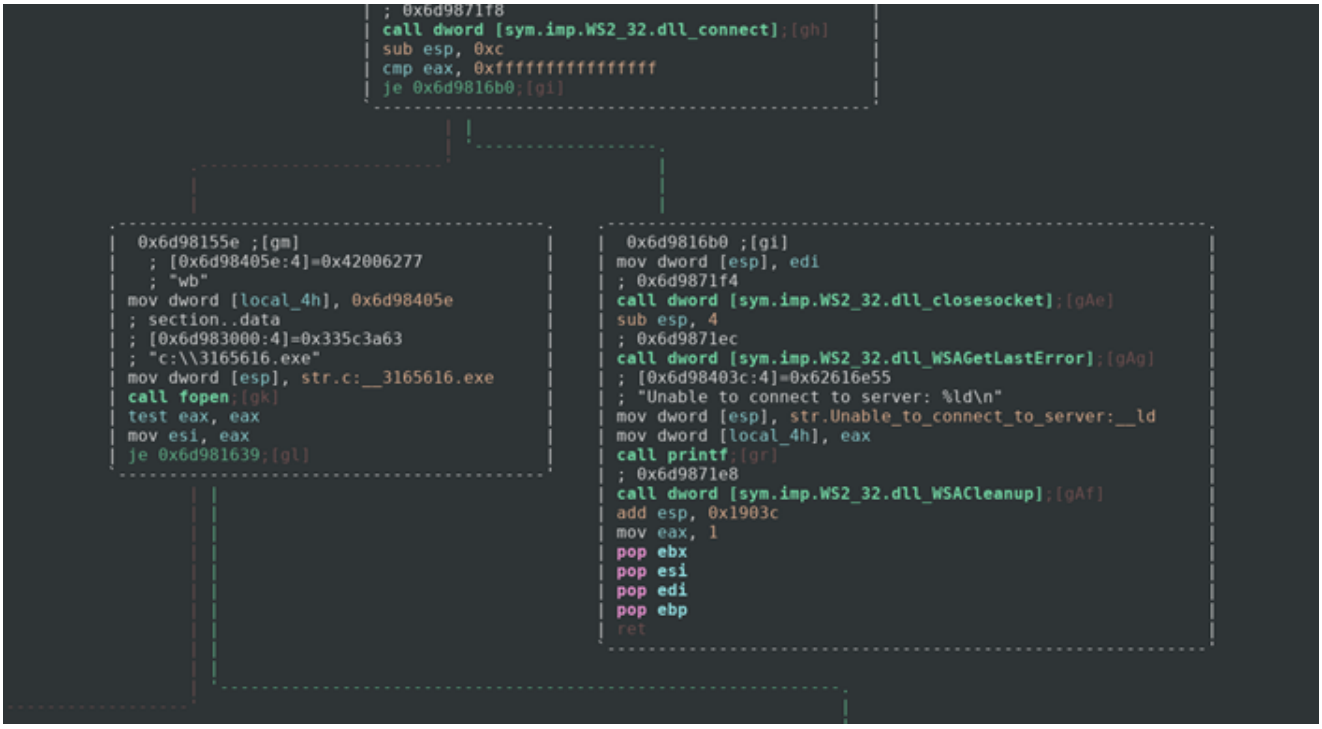
This function calls the WSAStartup function which starts the processes necessary to use the Socket library for Windows. This tells us that the dll must be connecting to some server via an open port. If WSAStartup fails, it pushes the message “WSAStartup failed: %d\n” along with the return value of WSAStartup 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
```

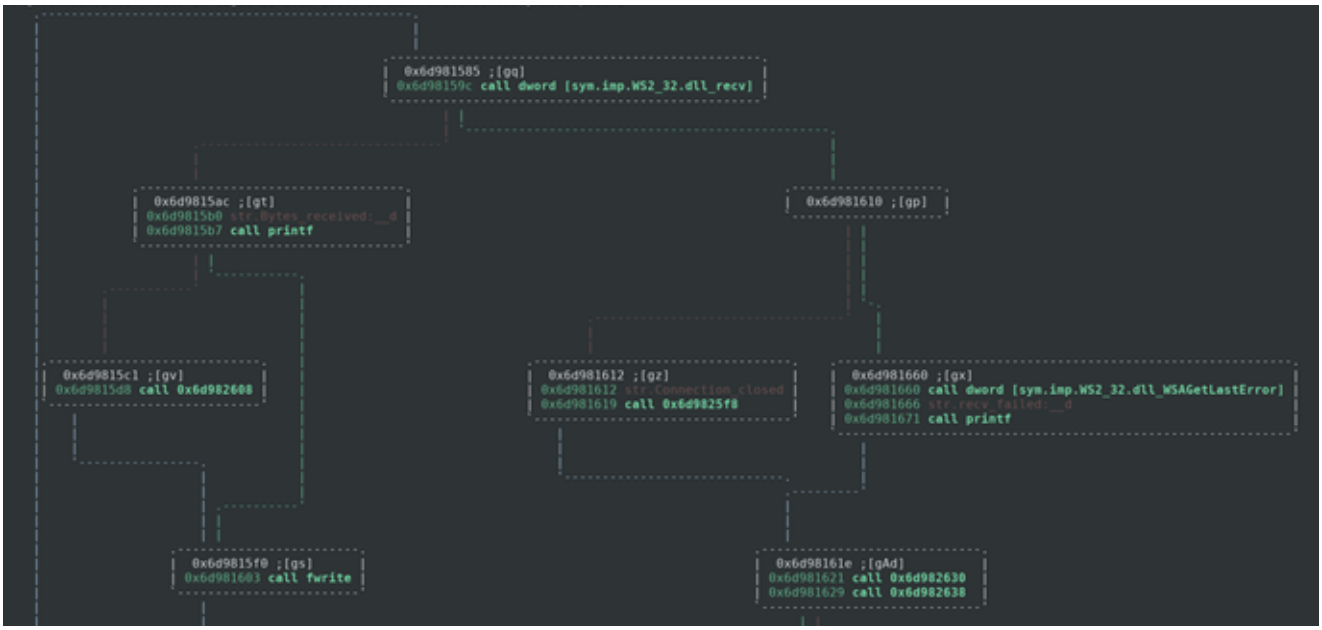
If the process succeeds then it calls the function fcn.6d9814c0.



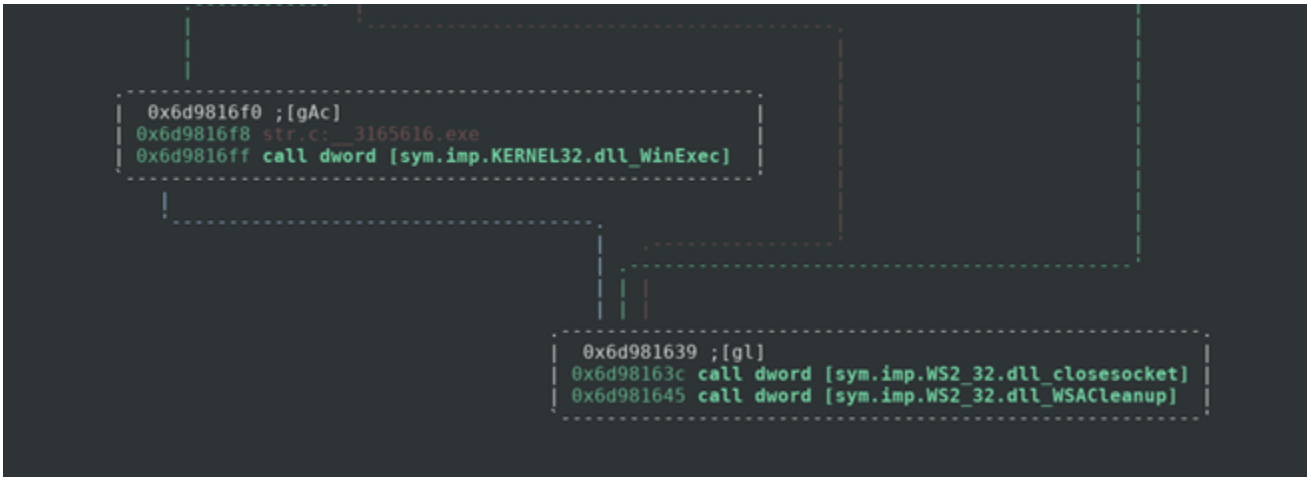
We can see that this function creates a socket. If it succeeded in the creation of the socket then it gives it the ip address 62.210.204.58 and the port 443 and connects. If not then an error is outputted via printf again and the program exits. We would be able to use this ip address and port as a network-based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



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.

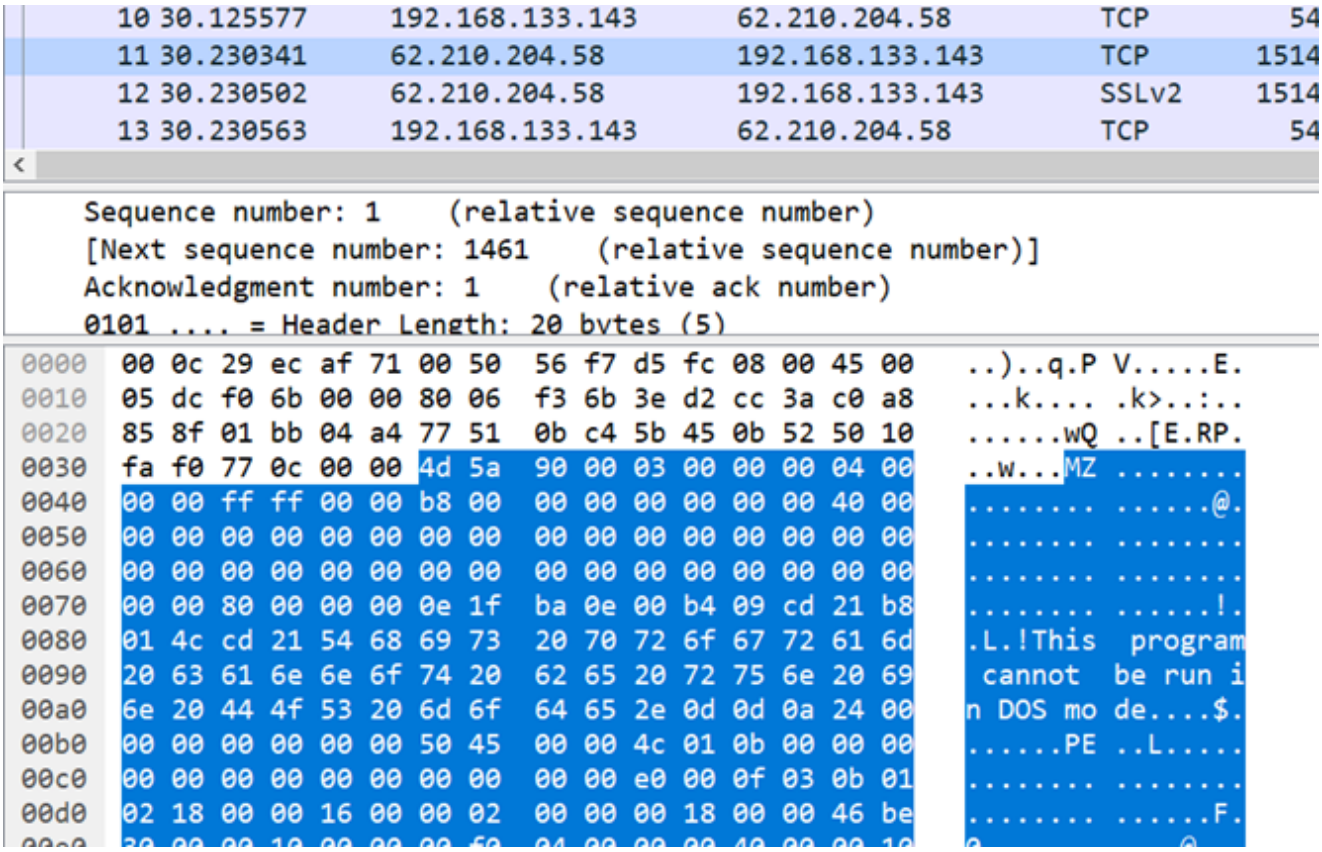


Minigraph illustrating the loop that populates the executable file with data



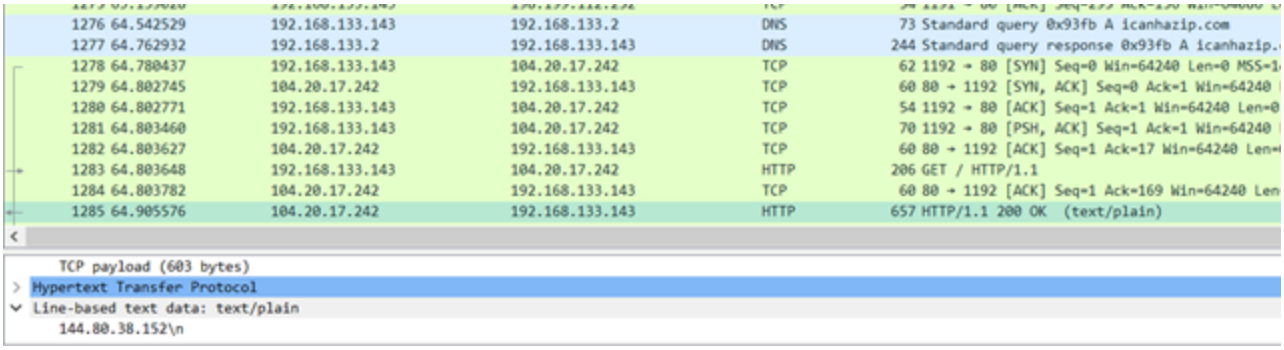
Minigraph illustrating the call to WinExec on the created executable

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.



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.



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!