CSEC 202-02
Final Project

Malware: WannaCry
December 14, 2019

Table of Contents

Background of Malware **3**

Testing Environment **4**

Static Analysis **5**

VirusTotal 6

Strings and FLOSS 9

PEiD 12

Dependency Walker 13

PEview 16

Resource Hacker 19

Dynamic Analysis **21**

Procman 22

Process Explorer 24

Regshot 26

ApateDNS 28

Netcat 29

Wireshark 30

Advanced Static Analysis **32**

IDA Pro 33

Advanced Dynamic Analysis **38**

x32dbg 39

Potential Dangers of the Malware **44**

How to Remove the Malware **45**

What I Learned **46**

Background of Malware

Name: WannaCry
Type of Malware: Ransomware
Executable Name:
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe

Discovered On: May 12, 2017

Description: WannaCry is part of the ransomware family, encrypting the files on the system and demanding a certain amount of ransom, in Bitcoin, to decrypt the files. After WannaCry encrypts the files on the system, a window will appear on the screen stating to pay the ransom to the Bitcoin wallet to decrypt the files on the system. It also shows two countdowns, one displaying that the payment will be raised after the timer runs out and the other displaying that the files will be lost if the ransom isn't paid within that time frame. The reports from MITRE, Symantec, and FireEye below will give a more accurate explanation on what the malware does.

Sources:

- WannaCry Download from theZoo: https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Ransomware.WannaCry
- MITRE Report - https://attack.mitre.org/software/S0366/
- Symantec Report - https://www.symantec.com/security-center/writeup/2017-051310-3522-99
- FireEye Report - https://www.fireeye.com/blog/threat-research/2017/05/wannacry-malware-profile.html

Testing Environment

Physical Machine:

Operating System: Windows 10 Enterprise
Version: 1903
OS Build: 18362.267
Processor: Intel(R) Core(™) i7-6700 CPU @ 3.40GHz 3.41GHz
RAM: 16.0 GB
System Architecture: 64-bit operating system, x64-based processor

Virtual Machine:

Operating System: Windows 10 Enterprise
Version: 1703
OS Build: 15063.540
Processor: Intel(R) Core(™) i7-6700 CPU @ 3.40GHz 3.41GHz
RAM: 4.00 GB
System Architecture: 64-bit operating system, x64-based processor

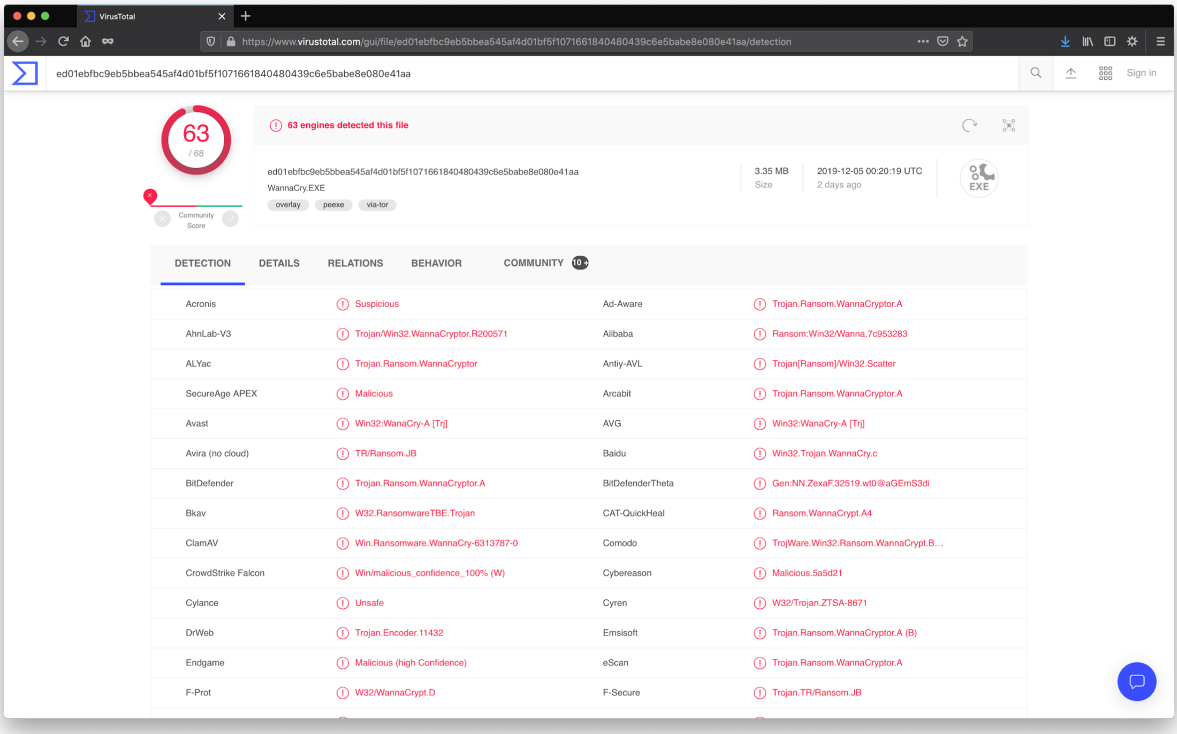
Environment: When running the tools for dynamic analysis and advanced dynamic analysis, I executed the WannaCry malware on one of the computers in the AirGap lab and within a Windows 10 virtual machine (VM). An AirGap environment is an environment where the computers within it cannot access or communicate with other computers or networks, including the ones within the AirGap environment. This prevents worm-like viruses from infecting other computers on the network. DeepFreeze is also installed on all the systems so restarting the computer will revert the computer back to its original state, before the malware was installed. Although I didn't have to go inside a VM because of the AirGap environment, I wanted to practice good habits in case I wasn't in an AirGap environment. The advantage of working on a VM in this scenario, was I could keep all of my screenshots, tools, and the executable on the physical machine. Once the VM was infected and I could no longer do any analysis, I would just revert to a snapshot of the VM I took prior to running the executable.

Static Analysis

Static analysis tools are tools that can be used to analyze malicious executables without running the executable. These tools can be used to get a base knowledge about what the malware might do or how it might act. Some malicious executables will pack their PE file headers rendering a tool, like PEview and Resource Hacker, useless until it's unpacked. The first step in any reverse engineering process should be static analysis, because the information an analyst could obtain without actually running the executable can be very useful when writing a report and understanding how the malware might act. The tools I'll be using for my static analysis of WannaCry are VirusTotal, Strings and Floss, PEiD, Dependency Walker, PEview, and Resource Hacker.

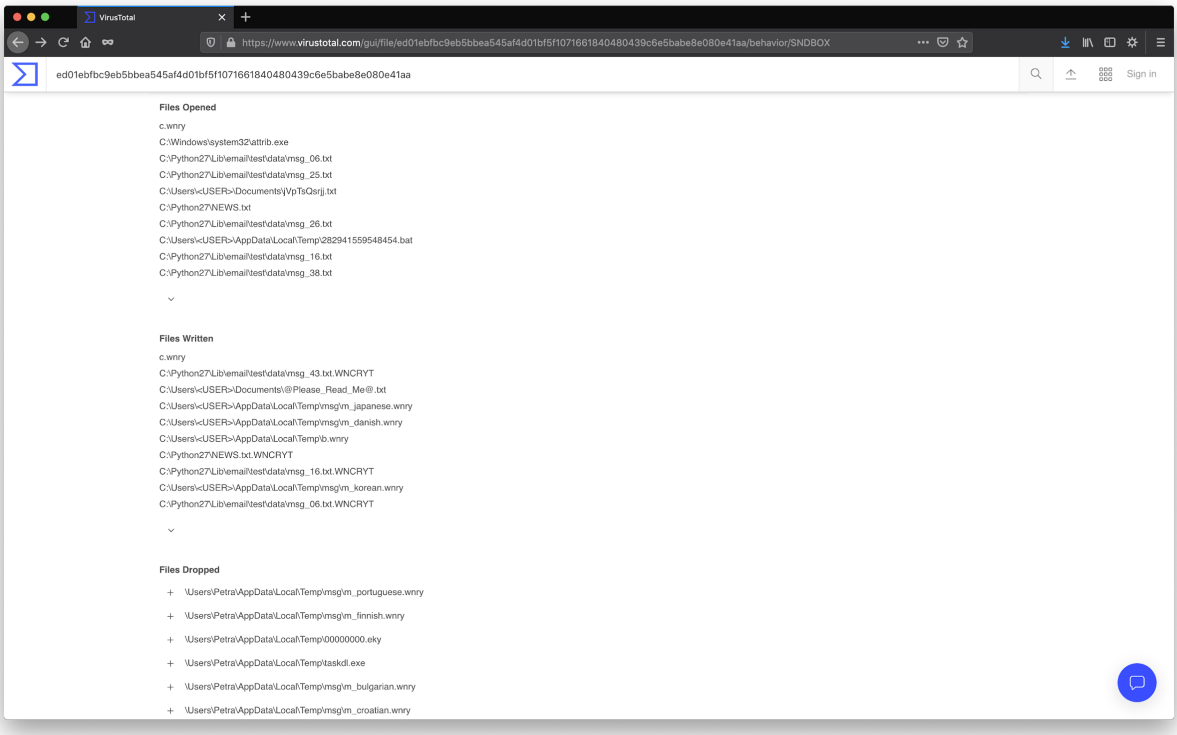
VirusTotal

VirusTotal is a very helpful static analysis tool if an analyst is unaware of what an executable may do. After uploading the executable to VirusTotal it will compare the executable's hash to hashes stored in the database. If there is a match, a full report on the executable will be given along with how it will act once executed and how dangerous it really is. If there isn't a match, it means that no one has uploaded the executable before and the engines will attempt to determine how malicious the executable is.

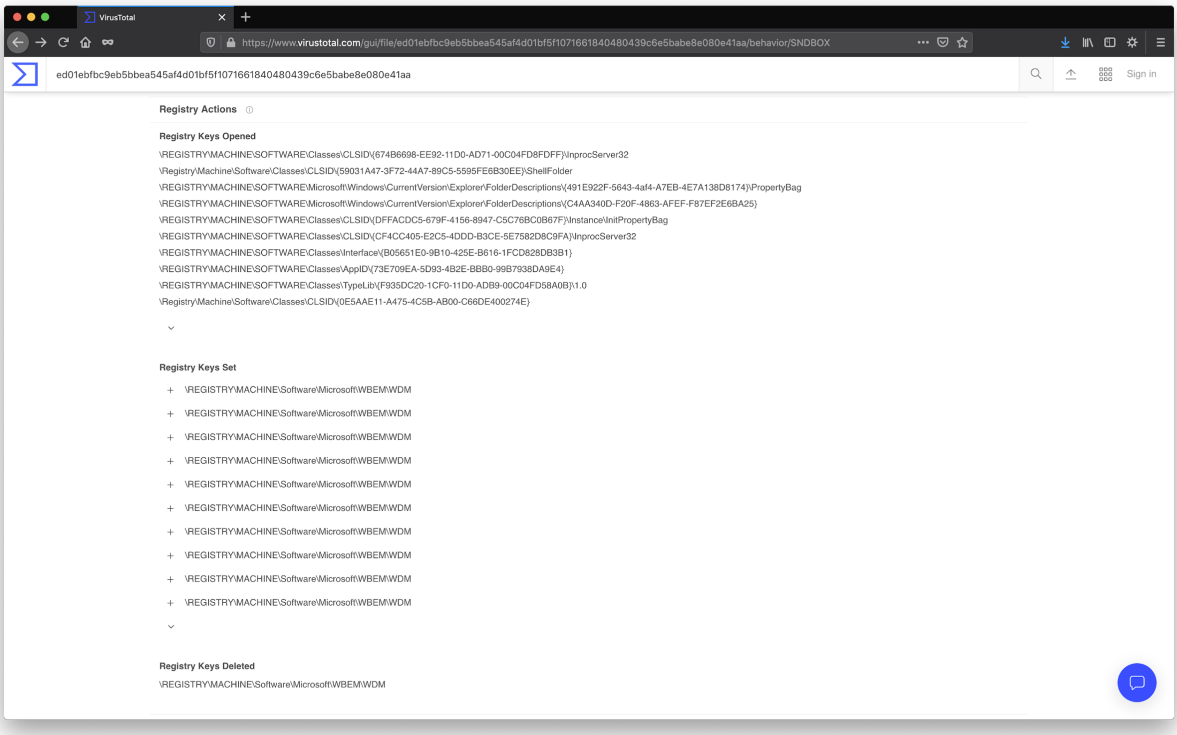


AegisLab	Undetected	Avast-Mobile	Undetected
CMC	Undetected	Kingsoft	Undetected
SUPERAntiSpyware	Undetected	ESET-NOD32	Timeout
FireEye	Timeout	Symantec Mobile Insight	Unable to process file type
Trustlook	Unable to process file type		

These are the results from the VirusTotal scan which tells me that the executable is definitely malicious. 63/68 engines detected this executable as malware. Three of the five engines couldn't detect the executable as malware, one of the engines timed out before it could reach a conclusion, and another one of the engines couldn't process the executable.



This comes from the “Behavior” section of VirusTotal, and tells me what files are going to be opened, written, and dropped. This will be useful because it shows what files it creates and where so that I can analyze them to figure out more information about WannaCry or potentially find the configuration file.



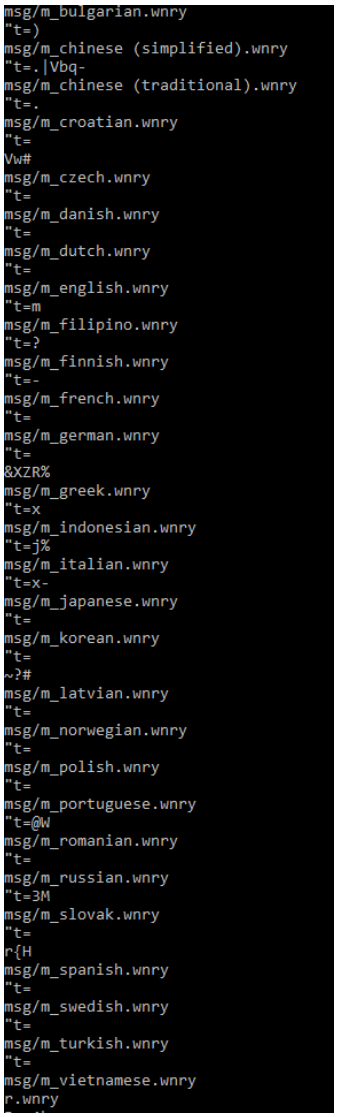
This also comes from the “Behavior” section of VirusTotal. It tells me what registry keys are opened, keys that are having their values set/updated, and what registry keys will be deleted. This will be useful later when looking at a dynamic tool like RegShot and seeing if similar keys are created, set, and/or deleted. It is also useful to look at tools like Regedit and analyze the individual keys created or updated.

Link to VirusTotal Scan:
https://www.virustotal.com/gui/file/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa/detection

Strings and FLOSS

Strings is a static analysis command line interface (CLI) tool that will search the executable for strings, which may or may not be found useful in later stages of the analysis. FLOSS is a tool developed by FireEye that has the same function as strings, however, it will also attempt to decode strings found in the executable.

Strings:
strings.exe ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe



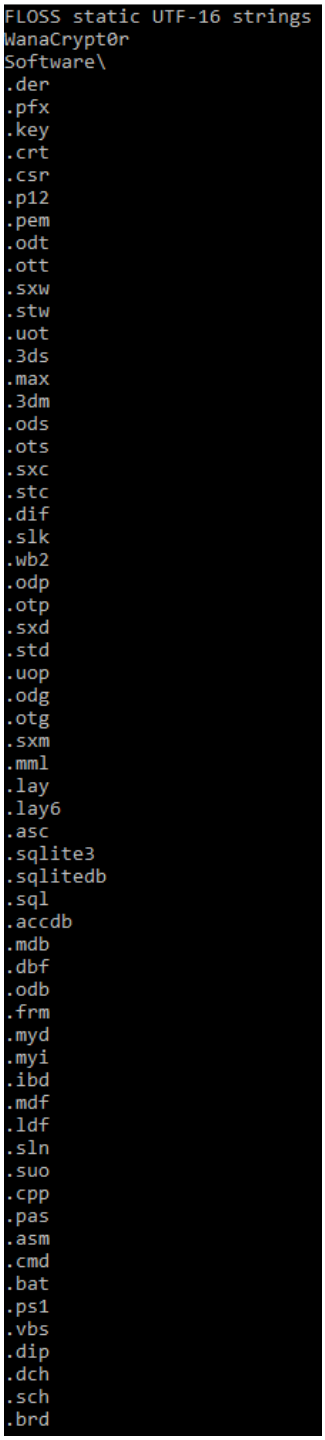
Strings showed a lot of random text when looking at the WannaCry executable. However, I did find this section that contains the names of different languages followed by a .wnry. I'm assuming these refer to the different languages WannaCry will translate itself to so those

infected that speak another language can still read the “Wana Decrypt0r 2.0” and pay the ransom.

There is another screenshot from Strings, however, I’m using it down below with PEview to show a pattern.

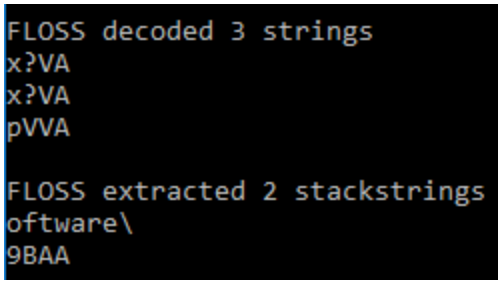
FLOSS:

floss64.exe ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe



As shown in the screenshot above, these are static UTF-16 strings (not shown with Strings). I also noticed, near the top, WanaCrypt0r Software. The majority of the screenshot is in the format .x, where x represents file extensions (like sql). I'm assuming these are the file

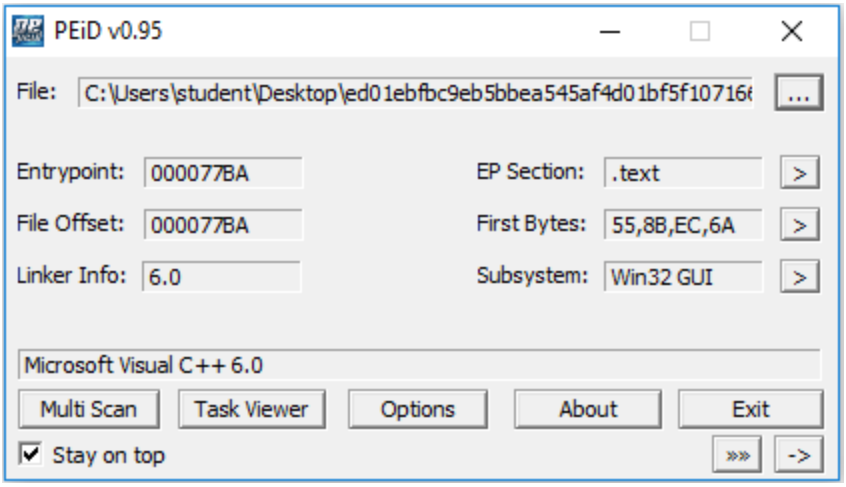
extensions WannaCry, or WanaCrypt0r Software, will look for to encrypt. There are many more extensions listed that are not shown in the screenshot.



These are strings that FLOSS decoded (also not shown with Strings). I'm not really sure as to what these five strings refer to.

PEiD

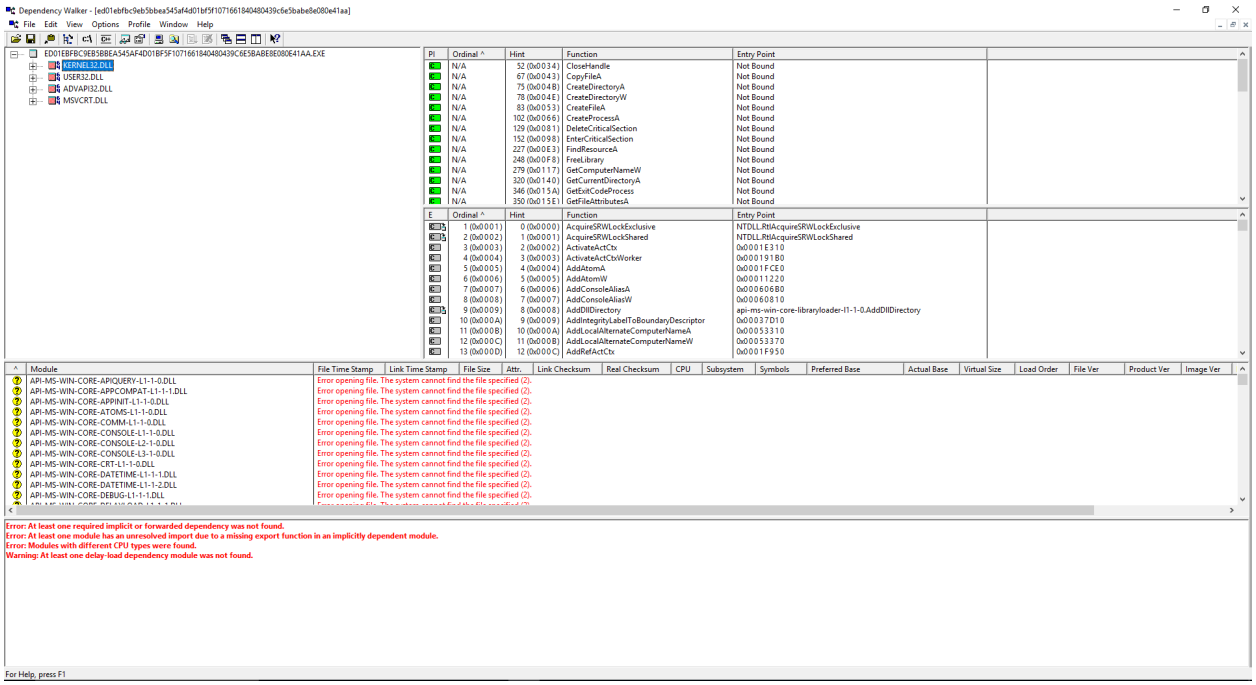
PEiD is a static analysis tool that can determine what an executable is compiled with and whether or not the executable is packed. This tool should be used before PEview because if the executable is packed PEview will not be able to read the headers.



After running the executable through PEiD, I now know that the WannaCry executable is not packed and is compiled with Microsoft Visual C++ 6.0. Since the executable is not packed, I'll be able to look at the data, text, and resource section in PEview.

Dependency Walker

Dependency Walker is a static analysis tool used to look at the DLLs a malware will be utilizing and what the DLLs are doing, such as system calls. An analyst can then look for any calls they deem valuable later in the advanced static or dynamic phases.



In the screenshot above, kernel32.dll is selected and in the right pane are some of the system calls it'll utilize during runtime. Some of the interesting system calls I noticed were CreateFileA, CreateDirectoryA, and CreateProcessA. This tells me that WannaCry is going to be creating files, directories, and processes on the machine once executed.

The kernel32 DLL "is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware." (malwareanalysis Slides)

The user32 DLL "contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions." (malwareanalysis Slides)

The advapi32 DLL "provides access to advanced core Windows components such as the Service Manager and Registry." (malwareanalysis Slides)

PI	Ordinal ^	Hint	Function	Entry Point
C++	N/A	7 (0x0007)	??0exception@@QAE@ABQBD@Z	Not Bound
C++	N/A	8 (0x0008)	??0exception@@QAE@ABV0@@Z	Not Bound
C++	N/A	13 (0x000D)	??1exception@@UAE@XZ	Not Bound
C++	N/A	14 (0x000E)	??1type_info@@UAE@XZ	Not Bound
C++	N/A	15 (0x000F)	???2@YAPAXI@Z	Not Bound
C++	N/A	16 (0x0010)	???3@YAXPAX@Z	Not Bound
C	N/A	65 (0x0041)	_CxxThrowException	Not Bound
C	N/A	72 (0x0048)	_XcptFilter	Not Bound
C	N/A	73 (0x0049)	__CxxFrameHandler	Not Bound
C	N/A	88 (0x0058)	__getmainargs	Not Bound
C	N/A	98 (0x0062)	__p___argc	Not Bound
C	N/A	99 (0x0063)	__p___argv	Not Bound
C	N/A	106 (0x006A)	__p__commode	Not Bound
C	N/A	111 (0x006F)	__p__fmode	Not Bound
C	N/A	129 (0x0081)	__set_app_type	Not Bound
C	N/A	131 (0x0083)	__setusermatherr	Not Bound
C	N/A	143 (0x008F)	_acmdln	Not Bound
C	N/A	157 (0x009D)	_adjust_fdiv	Not Bound
C	N/A	183 (0x00B7)	_controlfp	Not Bound
C	N/A	202 (0x00CA)	_except_handler3	Not Bound
C	N/A	211 (0x00D3)	_exit	Not Bound
C	N/A	271 (0x010F)	_initterm	Not Bound
C	N/A	316 (0x013C)	_local_unwind2	Not Bound
C	N/A	380 (0x017C)	_mbsstr	Not Bound
C	N/A	449 (0x01C1)	_stricmp	Not Bound
C	N/A	576 (0x0240)	calloc	Not Bound
C	N/A	585 (0x0249)	exit	Not Bound
C	N/A	588 (0x024C)	fclose	Not Bound
C	N/A	599 (0x0257)	fopen	Not Bound
C	N/A	605 (0x025D)	fread	Not Bound
C	N/A	606 (0x025E)	free	Not Bound
C	N/A	614 (0x0266)	fwrite	Not Bound
C	N/A	657 (0x0291)	malloc	Not Bound
C	N/A	662 (0x0296)	memcmp	Not Bound
C	N/A	663 (0x0297)	memcpy	Not Bound
C	N/A	665 (0x0299)	memset	Not Bound
C	N/A	678 (0x02A6)	rand	Not Bound
C	N/A	679 (0x02A7)	realloc	Not Bound
C	N/A	690 (0x02B2)	sprintf	Not Bound

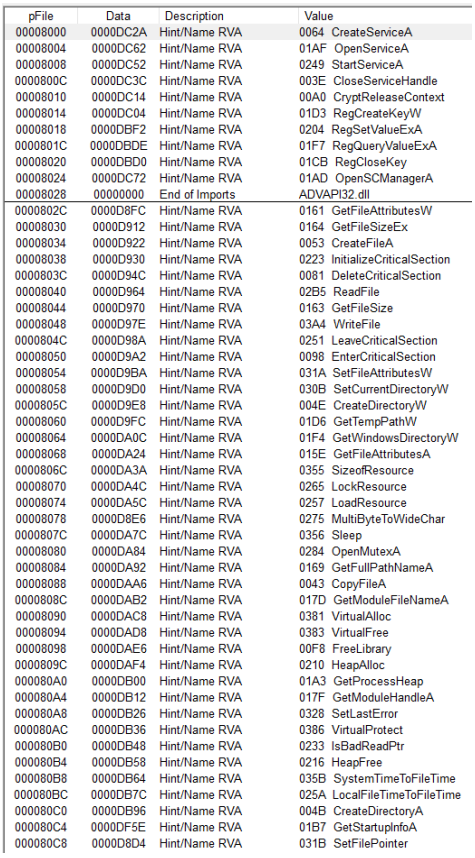
From online research, the msvcrt DLL contains C library functions. The screenshot above shows the function calls only from msvcrt.dll. I'm not sure as to what the first six lines of function calls refer to as they don't look like normal C library functions.

Sources:

- malwareanalysis Slides. Slide 28 - https://mycourses.rit.edu/d2l/le/content/767361/viewContent/5833348/View?ou=767361

PEview

PEview is a static analysis tool useful for looking at PE headers within an executable. PE headers are broken up into a .text, .rdata, .data, .idata, .edata, .pdata, .rsrc, and .reloc. For my analysis I looked at the .rdata, which “holds read-only data that is globally accessible within the program,” .data which “stores global data accessed throughout the program,” .rsrc which “stores resources needed by the executable.”



pFile	Data	Description	Value
00008000	0000DC2A	Hint/Name RVA	0064 CreateServiceA
00008004	0000DC62	Hint/Name RVA	01AF OpenServiceA
00008008	0000DC52	Hint/Name RVA	0249 StartServiceA
0000800C	0000DC3C	Hint/Name RVA	003E CloseServiceHandle
00008010	0000DC14	Hint/Name RVA	00A0 CryptReleaseContext
00008014	0000DC04	Hint/Name RVA	01D3 RegCreateKeyW
00008018	0000DBF2	Hint/Name RVA	0204 RegSetValueExA
0000801C	0000DBDE	Hint/Name RVA	01F7 RegQueryValueExA
00008020	0000DBD0	Hint/Name RVA	01CB RegCloseKey
00008024	0000DC72	Hint/Name RVA	01AD OpenSCManagerA
00008028	00000000	End of Imports	ADVAPI32.dll
0000802C	0000D8FC	Hint/Name RVA	0161 GetFileAttributesW
00008030	0000D912	Hint/Name RVA	0164 GetFileSizeEx
00008034	0000D922	Hint/Name RVA	0053 CreateFileA
00008038	0000D930	Hint/Name RVA	0223 InitializeCriticalSection
0000803C	0000D94C	Hint/Name RVA	0081 DeleteCriticalSection
00008040	0000D964	Hint/Name RVA	02B5 ReadFile
00008044	0000D970	Hint/Name RVA	0163 GetFileSize
00008048	0000D97E	Hint/Name RVA	03A4 WriteFile
0000804C	0000D98A	Hint/Name RVA	0251 LeaveCriticalSection
00008050	0000D9A2	Hint/Name RVA	0098 EnterCriticalSection
00008054	0000D9BA	Hint/Name RVA	031A SetFileAttributesW
00008058	0000D9D0	Hint/Name RVA	030B SetCurrentDirectoryW
0000805C	0000D9E8	Hint/Name RVA	004E CreateDirectoryW
00008060	0000D9FC	Hint/Name RVA	01D6 GetTempPathW
00008064	0000DA0C	Hint/Name RVA	01F4 GetWindowsDirectoryW
00008068	0000DA24	Hint/Name RVA	015E GetFileAttributesA
0000806C	0000DA3A	Hint/Name RVA	0355 SizeofResource
00008070	0000DA4C	Hint/Name RVA	0265 LockResource
00008074	0000DA5C	Hint/Name RVA	0257 LoadResource
00008078	0000D8E6	Hint/Name RVA	0275 MultiByteToWideChar
0000807C	0000DA7C	Hint/Name RVA	0356 Sleep
00008080	0000DA84	Hint/Name RVA	0284 OpenMutexA
00008084	0000DA92	Hint/Name RVA	0169 GetFullPathNameA
00008088	0000DAA6	Hint/Name RVA	0043 CopyFileA
0000808C	0000DAB2	Hint/Name RVA	017D GetModuleFileNameA
00008090	0000DAC8	Hint/Name RVA	0381 VirtualAlloc
00008094	0000DAD8	Hint/Name RVA	0383 VirtualFree
00008098	0000DAE6	Hint/Name RVA	00F8 FreeLibrary
0000809C	0000DAF4	Hint/Name RVA	0210 HeapAlloc
000080A0	0000DB00	Hint/Name RVA	01A3 GetProcessHeap
000080A4	0000DB12	Hint/Name RVA	017F GetModuleHandleA
000080A8	0000DB26	Hint/Name RVA	0328 SetLastError
000080AC	0000DB36	Hint/Name RVA	0386 VirtualProtect
000080B0	0000DB48	Hint/Name RVA	0233 IsBadReadPtr
000080B4	0000DB58	Hint/Name RVA	0216 HeapFree
000080B8	0000DB64	Hint/Name RVA	035B SystemTimeToFileTime
000080BC	0000DB7C	Hint/Name RVA	025A LocalFileTimeToFileTime
000080C0	0000DB96	Hint/Name RVA	004B CreateDirectoryA
000080C4	0000DF5E	Hint/Name RVA	01B7 GetStartupInfoA
000080C8	0000D8D4	Hint/Name RVA	031B SetFilePointer

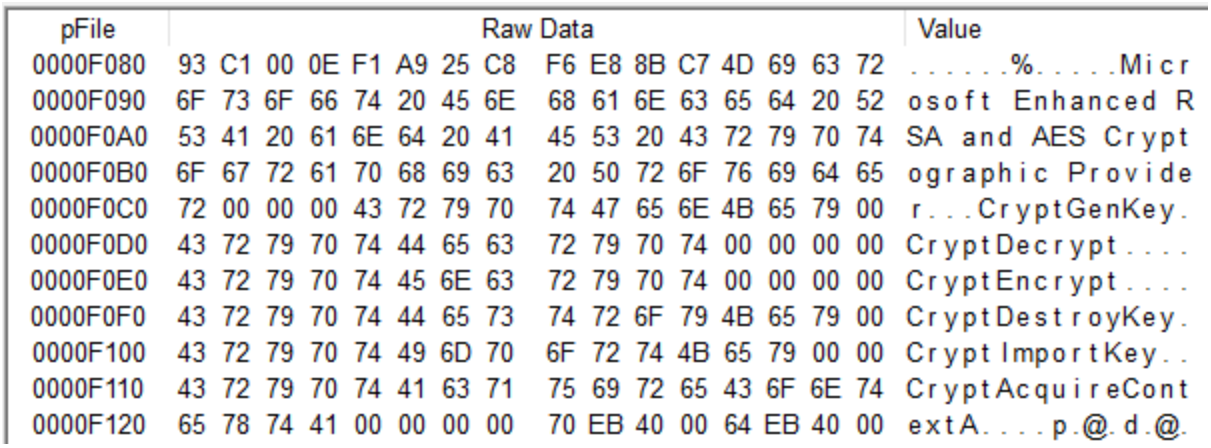
This is what PEview showed for the “SECTION .rdata - IMPORT Address Table.” In the screenshot I noticed the system calls WannaCry will use. Some valuable calls that are noticable; CreateServiceA, OpenServiceA, StartServiceA, RegCreateKeyW, RegSetValueExA, and many more. This tells me that WannaCry is going to be creating services, starting services, creating registry keys, and setting values to those registry keys. There are many more important system calls that are listed within the screenshot and some not listed within the screenshot, but can be seen with PEview.

pFile	Data	Description	Value
0000D5A8	0000D638	Import Name Table RVA	
0000D5AC	00000000	Time Date Stamp	
0000D5B0	00000000	Forwarder Chain	
0000D5B4	0000DBAA	Name RVA	KERNEL32.dll
0000D5B8	0000802C	Import Address Table RVA	
0000D5BC	0000D7DC	Import Name Table RVA	
0000D5C0	00000000	Time Date Stamp	
0000D5C4	00000000	Forwarder Chain	
0000D5C8	0000DBC4	Name RVA	USER32.dll
0000D5CC	000081D0	Import Address Table RVA	
0000D5D0	0000D60C	Import Name Table RVA	
0000D5D4	00000000	Time Date Stamp	
0000D5D8	00000000	Forwarder Chain	
0000D5DC	0000DC84	Name RVA	ADVAPI32.dll
0000D5E0	00008000	Import Address Table RVA	
0000D5E4	0000D714	Import Name Table RVA	
0000D5E8	00000000	Time Date Stamp	
0000D5EC	00000000	Forwarder Chain	
0000D5F0	0000DE88	Name RVA	MSVCRT.dll
0000D5F4	00008108	Import Address Table RVA	
0000D5F8	00000000		
0000D5FC	00000000		
0000D600	00000000		
0000D604	00000000		
0000D608	00000000		

This is what PEview showed for the “SECTION .rdata - IMPORT Directory Table.” The directory table is one of the many ways an analyst can see what DLLs a malware will be utilizing. In the case of WannaCry, it’ll be utilizing kernel32.dll, user32.dll, advapi32.dll, and msvcrt.dll at the minimum.

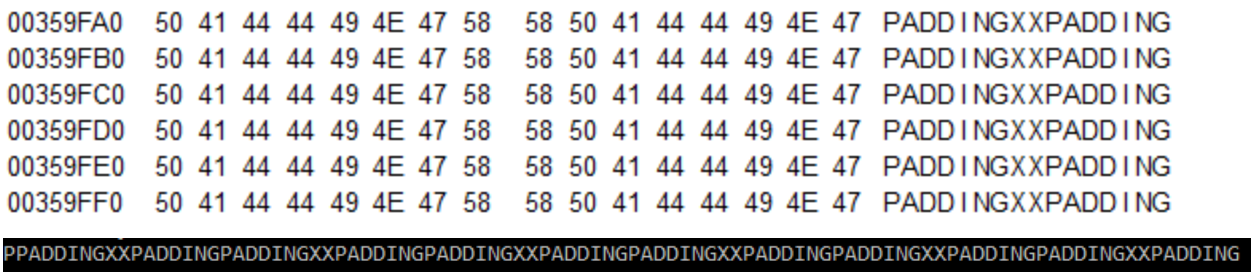
pFile	Raw Data																Value
0000EB70	2E	00	64	00	6F	00	63	00	00	00	00	00	57	41	4E	41	..d.o.c.....WANA
0000EB80	43	52	59	21	00	00	00	00	25	00	73	00	5C	00	25	00	CRY!....%.s.\.%.
0000EB90	73	00	00	00	43	6C	6F	73	65	48	61	6E	64	6C	65	00	s...CloseHandle.
0000EBA0	44	65	6C	65	74	65	46	69	6C	65	57	00	4D	6F	76	65	DeleteFileW.Move
0000EBB0	46	69	6C	65	45	78	57	00	4D	6F	76	65	46	69	6C	65	FileExW.MoveFile
0000EBC0	57	00	00	00	52	65	61	64	46	69	6C	65	00	00	00	00	W...ReadFile....
0000EBD0	57	72	69	74	65	46	69	6C	65	00	00	00	43	72	65	61	WriteFile...Crea
0000EBE0	74	65	46	69	6C	65	57	00	6B	65	72	6E	65	6C	33	32	teFileW.kernel32
0000EBF0	2E	64	6C	6C	00	00	00	00	07	02	00	00	00	A4	00	00	.dll..........

This is what PEview showed for the “SECTION .data”. This screenshot shows the name of the malware, WANACRY, the name of some system calls, and the kernel32.dll.



pFile	Raw Data	Value
0000F080	93 C1 00 0E F1 A9 25 C8 F6 E8 8B C7 4D 69 63 72	......%.....Micr
0000F090	6F 73 6F 66 74 20 45 6E 68 61 6E 63 65 64 20 52	osoft Enhanced R
0000F0A0	53 41 20 61 6E 64 20 41 45 53 20 43 72 79 70 74	SA and AES Crypt
0000F0B0	6F 67 72 61 70 68 69 63 20 50 72 6F 76 69 64 65	ographic Provide
0000F0C0	72 00 00 00 43 72 79 70 74 47 65 6E 4B 65 79 00	r...CryptGenKey.
0000F0D0	43 72 79 70 74 44 65 63 72 79 70 74 00 00 00 00	CryptDecrypt....
0000F0E0	43 72 79 70 74 45 6E 63 72 79 70 74 00 00 00 00	CryptEncrypt....
0000F0F0	43 72 79 70 74 44 65 73 74 72 6F 79 4B 65 79 00	CryptDestroyKey.
0000F100	43 72 79 70 74 49 6D 70 6F 72 74 4B 65 79 00 00	CryptImportKey..
0000F110	43 72 79 70 74 41 63 71 75 69 72 65 43 6F 6E 74	CryptAcquireCont
0000F120	65 78 74 41 00 00 00 00 70 EB 40 00 64 EB 40 00	extA....p.@.d.@.

This is another screenshot from “SECTION .data.” I included this screenshot because WannaCry is a type of ransomware that encrypts files on the system and in the screenshot there are some encryption system calls like CryptGenKey, CryptEncrypt, CryptDestoryKey, and more.



00359FA0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING
00359FB0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING
00359FC0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING
00359FD0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING
00359FE0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING
00359FF0	50 41 44 44 49 4E 47 58 58 50 41 44 44 49 4E 47	PADDINGXXPADDING

```
PPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDING
```

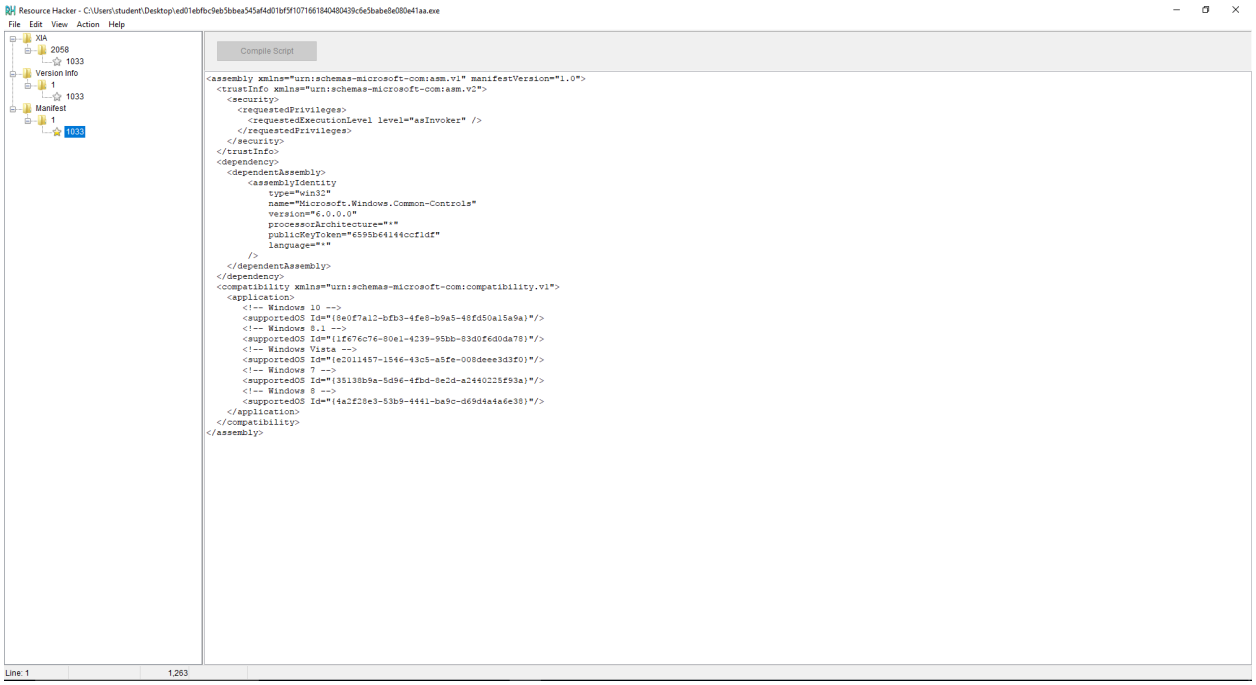
I found a pattern of “PADDINGXXPADDING” at the end of the resource section in PEview. I also found this at the end of the Strings command line tool, which is what the second screenshot is showing. I’m not exactly sure as to what this means.

Sources:

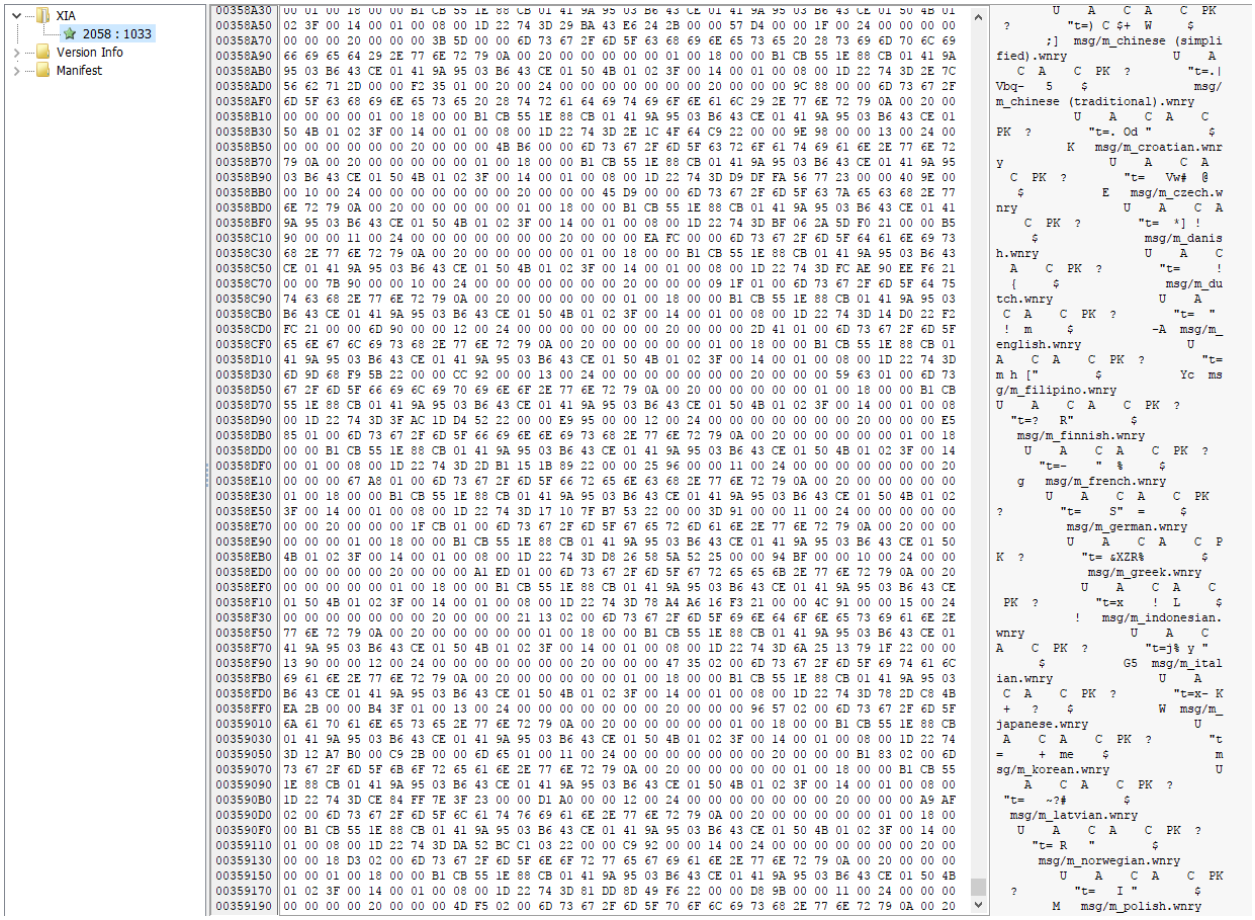
- malwareanalysis Slides. Slide 32 - https://mycourses.rit.edu/d2l/le/content/767361/viewContent/5833348/View

Resource Hacker

Resource Hacker looks at the resource section of PE headers. An analyst can review these files and find something useful about the malware, not always. However, if the executable is packed a tool like Resource Hacker is useless because it'll have no resource section until unpacked.



ResourceHacker showed three other files WannaCry has within it, "XIA," "VERSION," and "MANIFEST," I also noticed these earlier in PEview. They were contained within the resource section (.rsrc). In the Manifest section above there are tags setting its name and type to make it look like it's a Windows product.



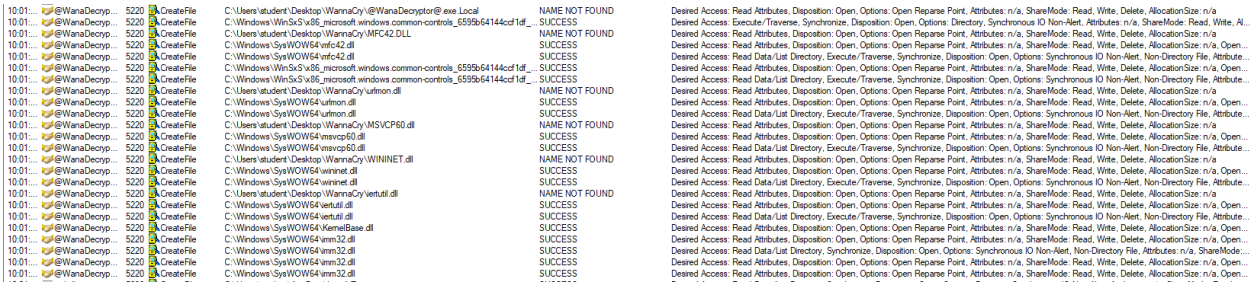
Towards the end of the XIA resource file are the languages seen earlier with FLOSS.

Dynamic Analysis

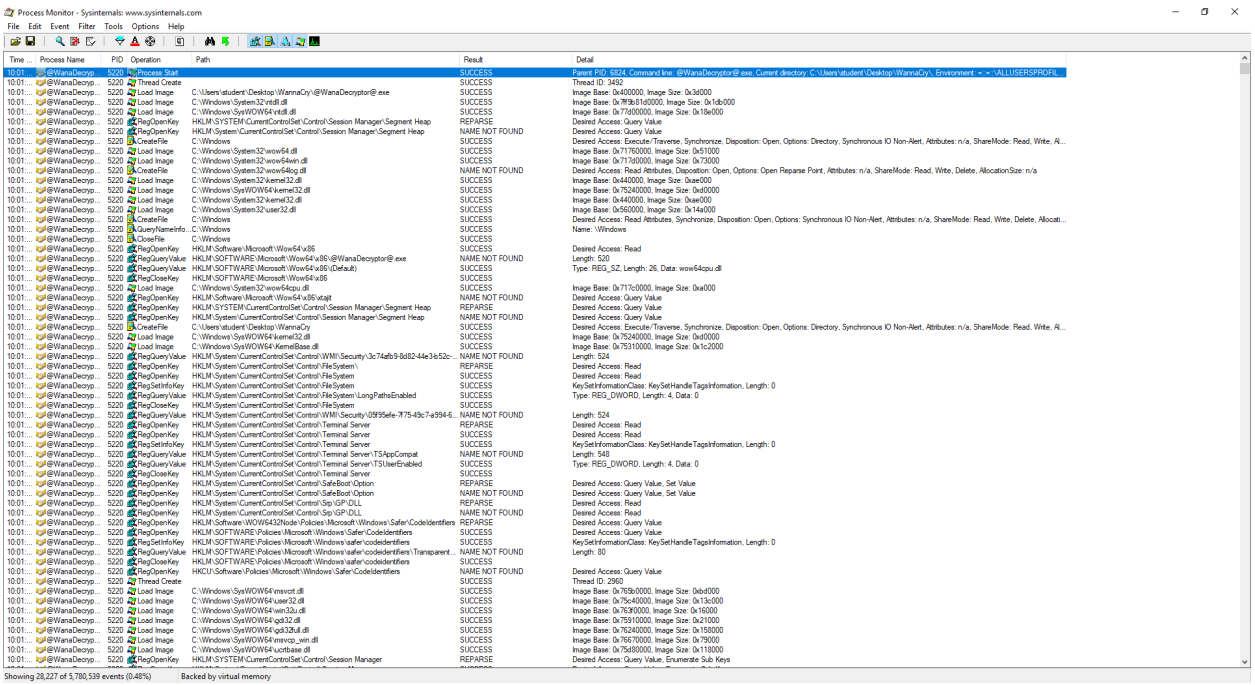
In the dynamic analysis phase an analyst is running the malware and reporting on how it acts. In this phase it is vital that an analyst is in a safe environment, such as an AirGab, where the malware cannot affect their physical machine or other computers on the network. If an analyst isn't careful, the malware could potentially escape the virtual machine and harm their computer or other computers on the network. The tools I'll be using for my dynamic analysis of WannaCry are Procman, Process Explorer, Regshot, ApateDNS, Netcat, and Wireshark.

Procman

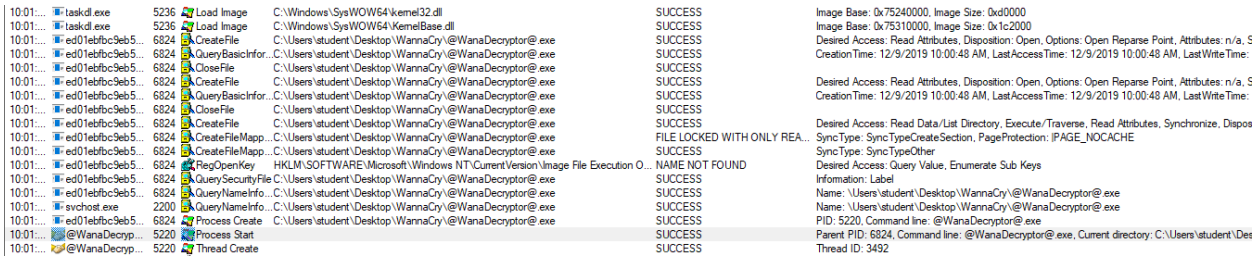
Procman, or Process Monitor, is a very useful dynamic analysis tool that allows an analyst to look at all processes on the system and how they are interacting with the system. It also has a very powerful filter option to only search for certain interactions, like only searching for files that were created or processed by a company name.



I used the filter “Operation is “CreateFile” then Include” for this screenshot. This showed me all the files WannaCry attempted to create and whether or not they were successful.



I used the filter “Command Line is “@WanaDecryptor@.exe” then Include” for this screenshot. Although this is a big screenshot, it contains most of what WannaCry is doing behind the scenes. There are images loaded, processes started, registry keys opened, setting values to those registry keys and so much more.

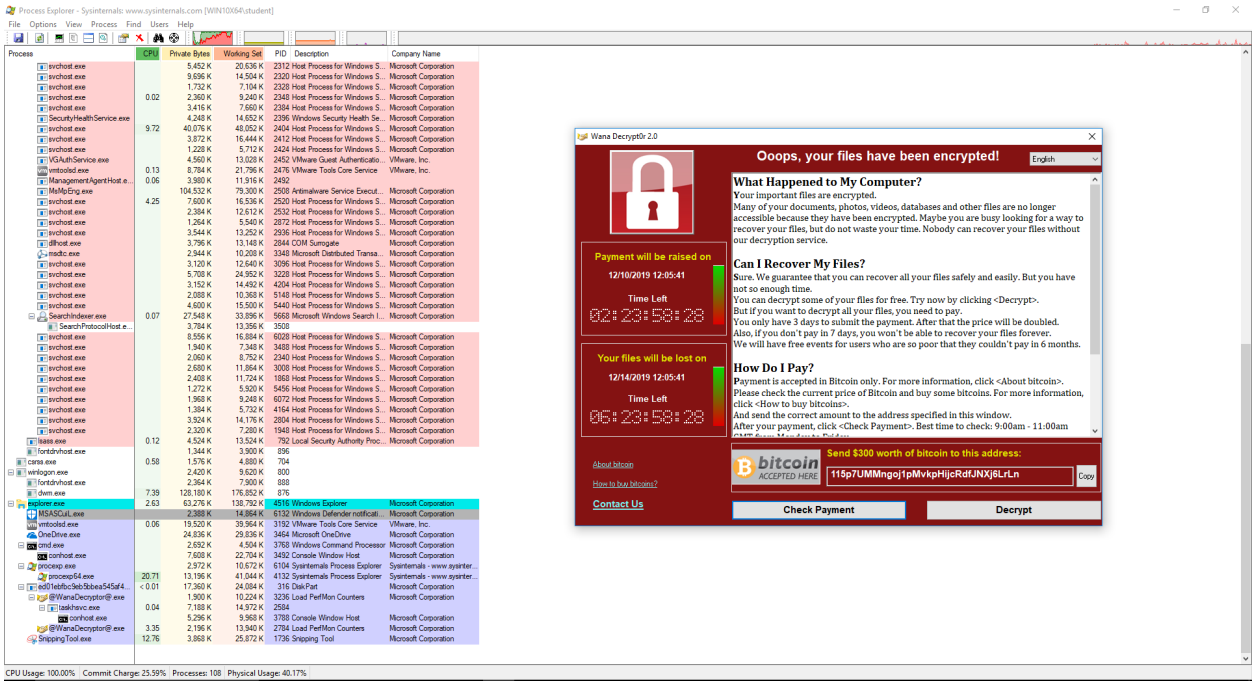


10:01:...	taskdl.exe	5236	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS	Image Base: 0x75240000, Image Size: 0xd0000
10:01:...	taskdl.exe	5236	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Image Base: 0x75310000, Image Size: 0x1c2000
10:01:...	ed01ebfbc9eb5...	6824	CreateFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Desired Access: Read Attributes, Disposition: Open, Options: Open Reparse Point, Attributes: n/a, S
10:01:...	ed01ebfbc9eb5...	6824	QueryBasicInfor...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	CreationTime: 12/9/2019 10:00:48 AM, LastAccessTime: 12/9/2019 10:00:48 AM, LastWriteTime:
10:01:...	ed01ebfbc9eb5...	6824	CloseFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	
10:01:...	ed01ebfbc9eb5...	6824	CreateFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Desired Access: Read Attributes, Disposition: Open, Options: Open Reparse Point, Attributes: n/a, S
10:01:...	ed01ebfbc9eb5...	6824	QueryBasicInfor...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	CreationTime: 12/9/2019 10:00:48 AM, LastAccessTime: 12/9/2019 10:00:48 AM, LastWriteTime:
10:01:...	ed01ebfbc9eb5...	6824	CloseFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	
10:01:...	ed01ebfbc9eb5...	6824	CreateFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Desired Access: Read Data/List Directory, Execute/Traverse, Read Attributes, Synchronize, Dispos
10:01:...	ed01ebfbc9eb5...	6824	CreateFileMapp...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	FILE LOCKED WITH ONLY REA...	SyncType: SyncTypeCreateSection, PageProtection: |PAGE_NOCACHE
10:01:...	ed01ebfbc9eb5...	6824	CreateFileMapp...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	SyncType: SyncTypeOther
10:01:...	ed01ebfbc9eb5...	6824	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution O...	NAME NOT FOUND	Desired Access: Query Value, Enumerate Sub Keys
10:01:...	ed01ebfbc9eb5...	6824	QuerySecurityFile	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Information: Label
10:01:...	ed01ebfbc9eb5...	6824	QueryNameInfo...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Name: \Users\student\Desktop\WannaCry\@WanaDecryptor@.exe
10:01:...	svchost.exe	2200	QueryNameInfo...	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	Name: \Users\student\Desktop\WannaCry\@WanaDecryptor@.exe
10:01:...	ed01ebfbc9eb5...	6824	Process Create	C:\Users\student\Desktop\WannaCry\@WanaDecryptor@.exe	SUCCESS	PID: 5220, Command line: @WanaDecryptor@.exe
10:01:...	@WanaDecryp...	5220	Process Start		SUCCESS	Parent PID: 6824, Command line: @WanaDecryptor@.exe, Current directory: C:\Users\student\Des
10:01:...	@WanaDecryp...	5220	Thread Create		SUCCESS	Thread ID: 3492

I used the filter “Company is “Microsoft Corporation” then Include” for this screenshot. This screenshot is very similar to the one above however this shows WannaCry hiding itself by claiming its a product made by the Microsoft Corporation.

Process Explorer

Process Explorer is another dynamic analysis tool that is looking at processes that are currently running on the system. The difference between Process Explorer and Procman is that Procman will tell an analysis exactly what the process is doing whereas Process Explorer is just telling an analysis CPU time, private bytes, and working set. Process Explorer also doesn't have a powerful filter option like Procman, however, Process Explorer does have an option to show VirusTotal results of the process which can be helpful in some scenarios.



In the screenshot the name of the executable, ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe, is running under process 316. The executable also spawned some child processes like @WanaDecryptor@.exe, taskhsvc.exe, and conhost.exe. @WanaDecryptor@.exe is the window to the right, in the screenshot above, telling the infected user what happened, how long until the payment is raised, when their files will be lost, how much they have to pay currently, and the address of the Bitcoin wallet. I'm not sure at this point in the analysis what taskhsvc.exe and conhost.exe do. I'm assuming since conhost.exe has a command prompt icon it's executing some commands behind the scenes. The WannaCry processes also have the company name "Microsoft Corporation." I believe this correlates to the <assembly> tags I saw in Resource Hacker.

However, these are most likely not all the processes WannaCry is running. It could be hiding some of the processes to make it harder to reverse and fully understand what's going on. It is also very much possible that WannaCry ran a few quick processes that were executed and terminated before I could open and capture the screen of the processes.

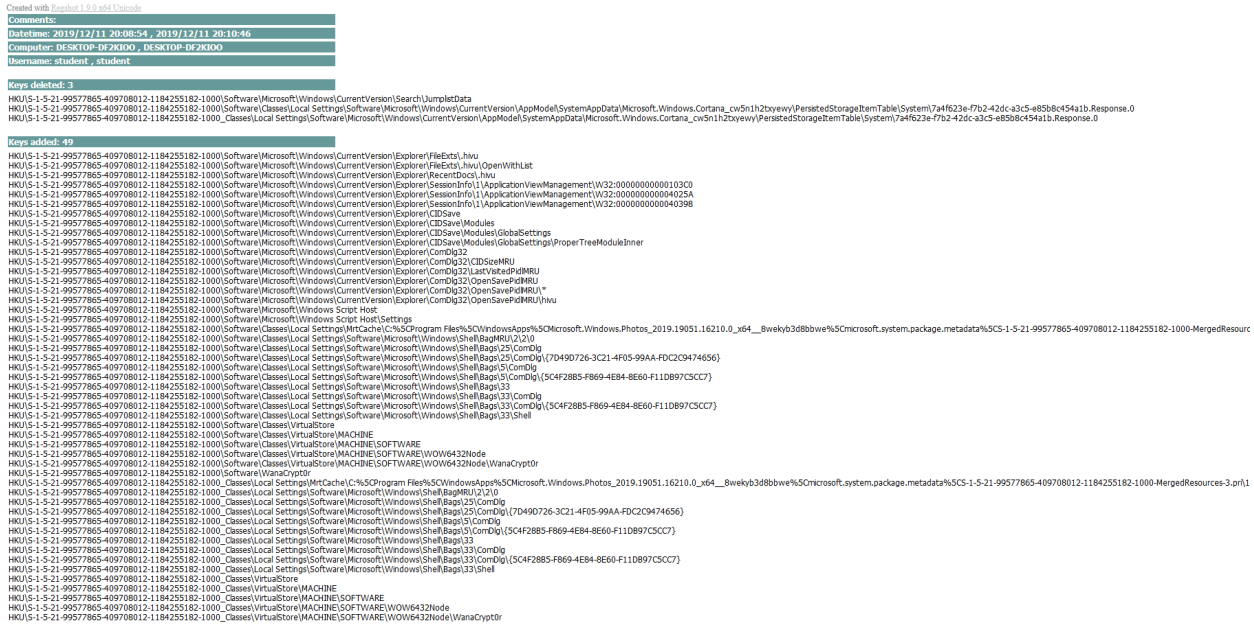
Regshot

Regshot is a dynamic analysis tool that is useful for looking at registry changes within the “two shots.” Prior to running the malware an analyst should take the first shot of the system to capture what is already on the system, in terms of registry keys. An analyst should then run the malware on the system and anytime after that take the second shot. Regshot will then compare the two shots and save the output to a plain text file or a HTML document. The output will show any keys deleted or added and values deleted, added, or modified from registry keys.

Both of the screenshots come from the same Regshot report. I took the first shot before executing the malware and took the second shot one to two minutes after the malware was done executing (Assuming the malware stopped executing after the background was changed and the decrypt window appeared). There were a total of 207 changes on the machine between the shots (The only thing I don’t show is the values being modified).

Total: 207 changes

- Keys deleted: 3
- Keys added: 49
- Values deleted: 14
- Values added: 97
- Values modified: 44



The screenshot above shows WannaCry deleting 3 keys and adding 49 keys. The information above could be useful to see if WannaCry is deleting any important keys from the system and the adding of keys should also be looked at to see what values are being added to them.

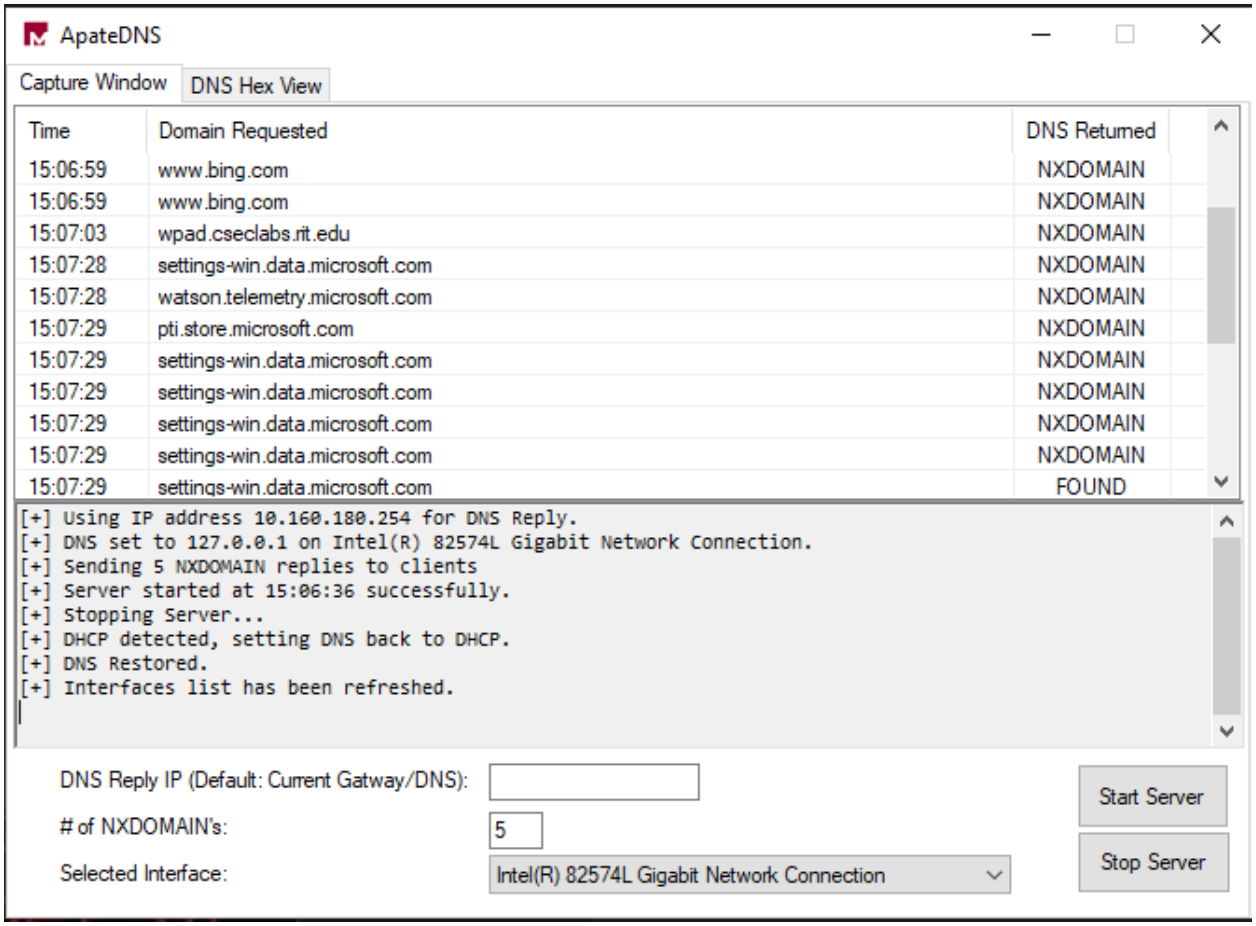
The screenshot above shows 14 values being deleted from registers and 97 values being added to registers. Again, the information above could be useful for looking at regedit and seeing what values have been added to these keys and if they have any significance.

ApateDNS

ApateDNS is a dynamic analysis tool used “for controlling DNS responses through an easy-to-use GUI. As a phony DNS server, ApateDNS spoofs DNS responses to a user-specified IP address by listening on UDP port 53 on the local machine.” - FireEye (https://www.fireeye.com/services/freeware/apatedns.html)

15:06:48	www.bing.com	NXDOMAIN
15:06:49	www.bing.com	NXDOMAIN
15:06:49	www.bing.com	NXDOMAIN
15:06:59	www.bing.com	NXDOMAIN
15:06:59	www.bing.com	NXDOMAIN

After starting up the apateDNS server, setting the “# of NXDOMAIN’s” to 5, and executing the malware I got exactly 5 rows of WannaCry trying to go out to www.bing.com. It also tries to reach out to Bing many other times throughout the course of the server’s uptime.



Above is my entire screen after the three www.bing.com outreaches. Besides trying to reach out to the csec server, it tries to reach out to settings-win.data.microsoft.com very frequently.

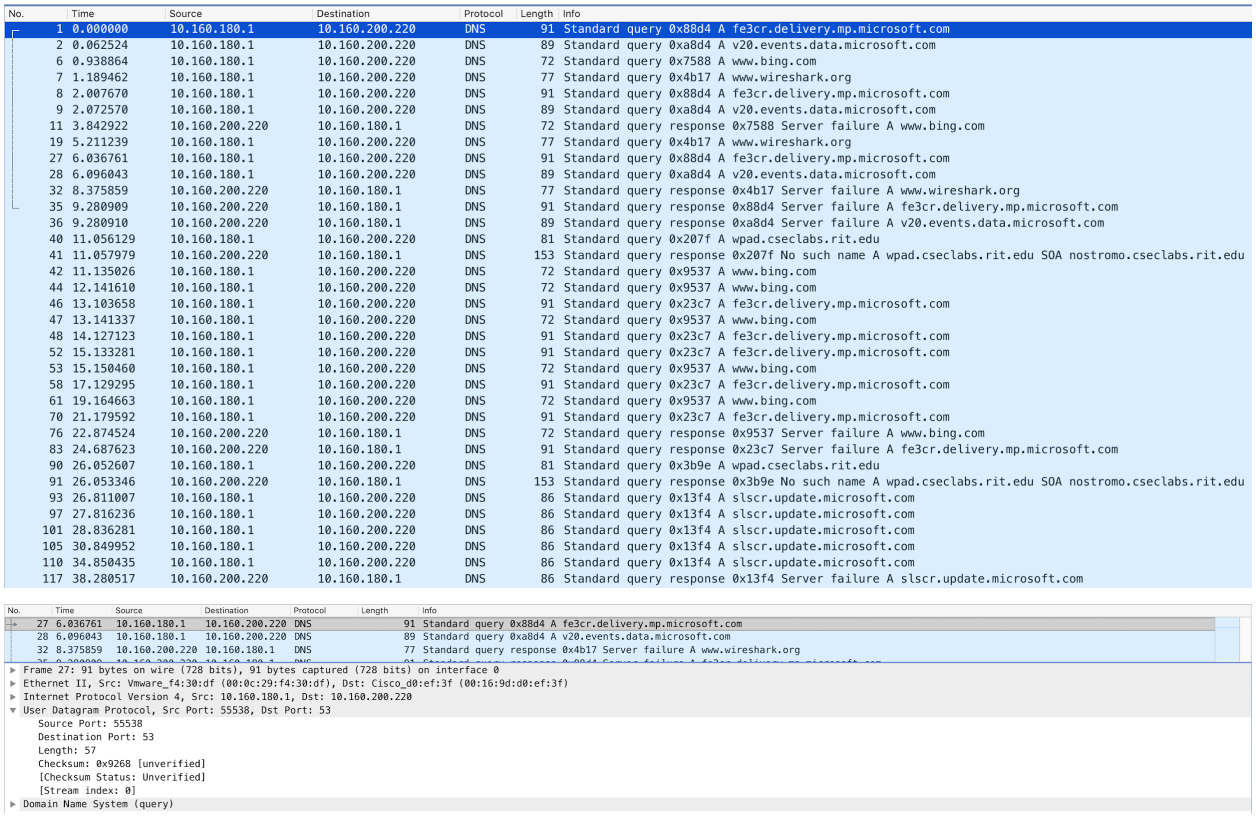
Netcat

I could not use netcat for this malware due to it not using the Hypertext Transfer Protocol (HTTP). This could be because of the AirGab, in other words, the malware potentially detected there was no Internet connection and didn't attempt to connect to any services using HTTP. Since there was no HTTP traffic, netcat was not useful in this situation, since in class we mainly used netcat to read HTTP headers.

However if there was HTTP traffic from WannaCry, netcat would be useful for looking at the HTTP header, as mentioned above. This is useful because an analyst can determine what websites a malware is trying to connect to and whether or not the request is a GET or a POST.

Wireshark

Wireshark is a packet sniffer that shows an analyst packets entering or leaving the system. Wireshark can be useful to show an analyst what protocols the malware is using and what other servers or websites the malware might be interacting with.

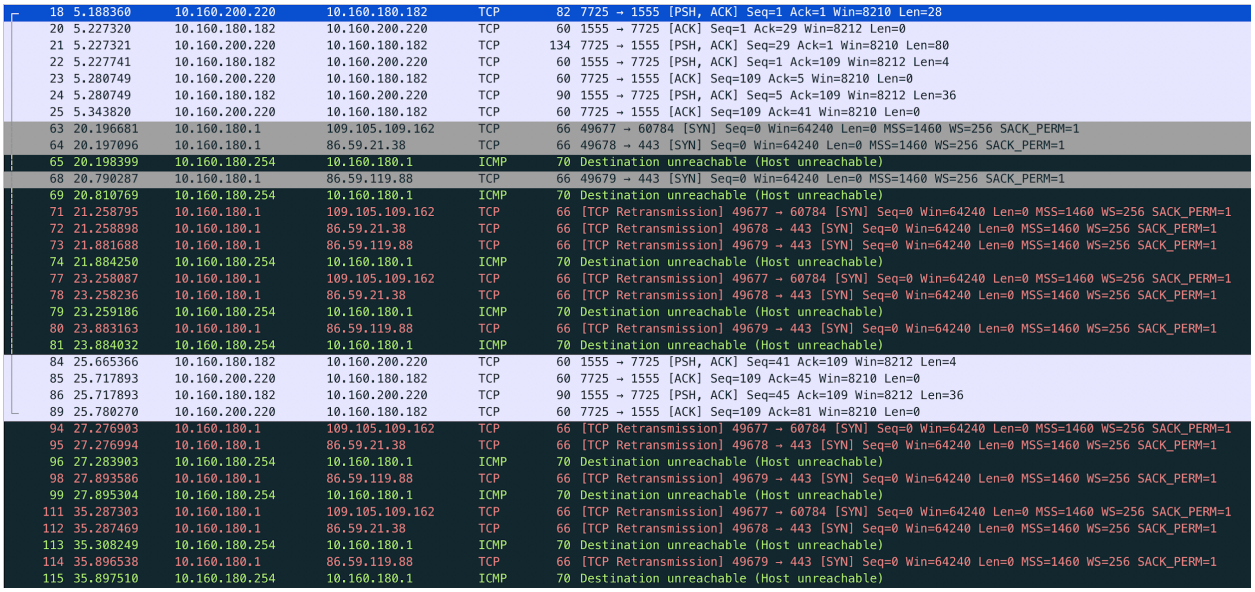


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x88d4 A fe3cr.delivery.mp.microsoft.com
2	0.062524	10.160.180.1	10.160.200.220	DNS	89	Standard query 0xa8d4 A v20.events.data.microsoft.com
6	0.938864	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x7588 A www.bing.com
7	1.189462	10.160.180.1	10.160.200.220	DNS	77	Standard query 0x4b17 A www.wireshark.org
8	2.007670	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x88d4 A fe3cr.delivery.mp.microsoft.com
9	2.072570	10.160.180.1	10.160.200.220	DNS	89	Standard query 0xa8d4 A v20.events.data.microsoft.com
11	3.842922	10.160.200.220	10.160.180.1	DNS	72	Standard query response 0x7588 Server failure A www.bing.com
19	5.211239	10.160.180.1	10.160.200.220	DNS	77	Standard query 0x4b17 A www.wireshark.org
27	6.036761	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x88d4 A fe3cr.delivery.mp.microsoft.com
28	6.096043	10.160.180.1	10.160.200.220	DNS	89	Standard query 0xa8d4 A v20.events.data.microsoft.com
32	8.375859	10.160.200.220	10.160.180.1	DNS	77	Standard query response 0x4b17 Server failure A www.wireshark.org
35	9.280909	10.160.200.220	10.160.180.1	DNS	91	Standard query response 0x88d4 Server failure A fe3cr.delivery.mp.microsoft.com
36	9.280910	10.160.200.220	10.160.180.1	DNS	89	Standard query response 0xa8d4 Server failure A v20.events.data.microsoft.com
40	11.056129	10.160.180.1	10.160.200.220	DNS	81	Standard query 0x207f A wpad.cseclabs.rit.edu
41	11.057979	10.160.200.220	10.160.180.1	DNS	153	Standard query response 0x207f No such name A wpad.cseclabs.rit.edu SOA nostromo.cseclabs.rit.edu
42	11.135026	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x9537 A www.bing.com
44	12.141610	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x9537 A www.bing.com
46	13.103658	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x23c7 A fe3cr.delivery.mp.microsoft.com
47	13.141337	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x9537 A www.bing.com
48	14.127123	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x23c7 A fe3cr.delivery.mp.microsoft.com
52	15.133281	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x23c7 A fe3cr.delivery.mp.microsoft.com
53	15.150460	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x9537 A www.bing.com
58	17.129295	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x23c7 A fe3cr.delivery.mp.microsoft.com
61	19.164663	10.160.180.1	10.160.200.220	DNS	72	Standard query 0x9537 A www.bing.com
70	21.179592	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x23c7 A fe3cr.delivery.mp.microsoft.com
76	22.874524	10.160.200.220	10.160.180.1	DNS	72	Standard query response 0x9537 Server failure A www.bing.com
83	24.687623	10.160.200.220	10.160.180.1	DNS	91	Standard query response 0x23c7 Server failure A fe3cr.delivery.mp.microsoft.com
90	26.052607	10.160.180.1	10.160.200.220	DNS	81	Standard query 0x3b9e A wpad.cseclabs.rit.edu
91	26.053346	10.160.200.220	10.160.180.1	DNS	153	Standard query response 0x3b9e No such name A wpad.cseclabs.rit.edu SOA nostromo.cseclabs.rit.edu
93	26.811007	10.160.180.1	10.160.200.220	DNS	86	Standard query 0x13f4 A slscr.update.microsoft.com
97	27.816236	10.160.180.1	10.160.200.220	DNS	86	Standard query 0x13f4 A slscr.update.microsoft.com
101	28.836281	10.160.180.1	10.160.200.220	DNS	86	Standard query 0x13f4 A slscr.update.microsoft.com
105	30.849952	10.160.180.1	10.160.200.220	DNS	86	Standard query 0x13f4 A slscr.update.microsoft.com
110	34.850435	10.160.180.1	10.160.200.220	DNS	86	Standard query 0x13f4 A slscr.update.microsoft.com
117	38.280517	10.160.200.220	10.160.180.1	DNS	86	Standard query response 0x13f4 Server failure A slscr.update.microsoft.com

No.	Time	Source	Destination	Protocol	Length	Info
27	6.036761	10.160.180.1	10.160.200.220	DNS	91	Standard query 0x88d4 A fe3cr.delivery.mp.microsoft.com
28	6.096043	10.160.180.1	10.160.200.220	DNS	89	Standard query 0xa8d4 A v20.events.data.microsoft.com
32	8.375859	10.160.200.220	10.160.180.1	DNS	77	Standard query response 0x4b17 Server failure A www.wireshark.org

```
▶ Frame 27: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface 0
▶ Ethernet II, Src: Vmware_f4:30:df (00:0c:29:f4:30:df), Dst: Cisco_d0:ef:3f (00:16:9d:d0:ef:3f)
▶ Internet Protocol Version 4, Src: 10.160.180.1, Dst: 10.160.200.220
▼ User Datagram Protocol, Src Port: 55538, Dst Port: 53
    Source Port: 55538
    Destination Port: 53
    Length: 57
    Checksum: 0x9268 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
▶ Domain Name System (query)
```

This is all the DNS traffic Wireshark sniffed while WannaCry was executing. After I wasn't getting anything from netcat I opened Wireshark to see if HTTP traffic was even showing up. Low and behold, no HTTP traffic and I figured out that most of the traffic on the system was using UDP ports, which is what the second screenshot is showing.



18	5.188360	10.160.200.220	10.160.180.182	TCP	82	7725 → 1555 [PSH, ACK] Seq=1 Ack=1 Win=8210 Len=28
20	5.227320	10.160.180.182	10.160.200.220	TCP	60	1555 → 7725 [ACK] Seq=1 Ack=29 Win=8212 Len=0
21	5.227321	10.160.200.220	10.160.180.182	TCP	134	7725 → 1555 [PSH, ACK] Seq=29 Ack=1 Win=8210 Len=80
22	5.227741	10.160.180.182	10.160.200.220	TCP	60	1555 → 7725 [PSH, ACK] Seq=1 Ack=109 Win=8212 Len=4
23	5.280749	10.160.200.220	10.160.180.182	TCP	60	7725 → 1555 [ACK] Seq=109 Ack=5 Win=8210 Len=0
24	5.280749	10.160.180.182	10.160.200.220	TCP	90	1555 → 7725 [PSH, ACK] Seq=5 Ack=109 Win=8212 Len=36
25	5.343820	10.160.200.220	10.160.180.182	TCP	60	7725 → 1555 [ACK] Seq=109 Ack=41 Win=8210 Len=0
63	20.196681	10.160.180.1	109.105.109.162	TCP	66	49677 → 60784 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
64	20.197096	10.160.180.1	86.59.21.38	TCP	66	49678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
65	20.198399	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
68	20.790287	10.160.180.1	86.59.119.88	TCP	66	49679 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
69	20.810769	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
71	21.258795	10.160.180.1	109.105.109.162	TCP	66	[TCP Retransmission] 49677 → 60784 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
72	21.258898	10.160.180.1	86.59.21.38	TCP	66	[TCP Retransmission] 49678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
73	21.881688	10.160.180.1	86.59.119.88	TCP	66	[TCP Retransmission] 49679 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
74	21.884250	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
77	23.258087	10.160.180.1	109.105.109.162	TCP	66	[TCP Retransmission] 49677 → 60784 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
78	23.258236	10.160.180.1	86.59.21.38	TCP	66	[TCP Retransmission] 49678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
79	23.259186	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
80	23.883163	10.160.180.1	86.59.119.88	TCP	66	[TCP Retransmission] 49679 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
81	23.884032	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
84	25.665366	10.160.180.182	10.160.200.220	TCP	60	1555 → 7725 [PSH, ACK] Seq=41 Ack=109 Win=8212 Len=4
85	25.717893	10.160.200.220	10.160.180.182	TCP	60	7725 → 1555 [ACK] Seq=109 Ack=45 Win=8210 Len=0
86	25.717893	10.160.180.182	10.160.200.220	TCP	90	1555 → 7725 [PSH, ACK] Seq=45 Ack=109 Win=8212 Len=36
89	25.780270	10.160.200.220	10.160.180.182	TCP	60	7725 → 1555 [ACK] Seq=109 Ack=81 Win=8210 Len=0
94	27.276903	10.160.180.1	109.105.109.162	TCP	66	[TCP Retransmission] 49677 → 60784 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
95	27.276994	10.160.180.1	86.59.21.38	TCP	66	[TCP Retransmission] 49678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
96	27.283903	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
98	27.893586	10.160.180.1	86.59.119.88	TCP	66	[TCP Retransmission] 49679 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
99	27.895304	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
111	35.287303	10.160.180.1	109.105.109.162	TCP	66	[TCP Retransmission] 49677 → 60784 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
112	35.287469	10.160.180.1	86.59.21.38	TCP	66	[TCP Retransmission] 49678 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
113	35.308249	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)
114	35.896538	10.160.180.1	86.59.119.88	TCP	66	[TCP Retransmission] 49679 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
115	35.897510	10.160.180.254	10.160.180.1	ICMP	70	Destination unreachable (Host unreachable)

This is all the TCP traffic Wireshark sniffed while WannaCry was executing. There is also ICMP traffic Wireshark sniffed telling me that the destination was unreachable, which makes sense because of the AirGab

Advanced Static Analysis

Advanced static analysis is the third phase and involves the user of a disassembler. While using the disassembler an analyst isn't actually running the executable, just looking at the assembly code. As of now IDA is still the most popular disassembler available. However on April 4, 2019, the National Security Agency (NSA) released their disassembler, Ghidra, which they developed. Ghidra might find itself becoming a very popular disassembler due to the fact that the NSA relied on it for disassembly purposes. The tool I'll be using for my advanced static analysis of WannaCry is IDA Pro.

IDA Pro

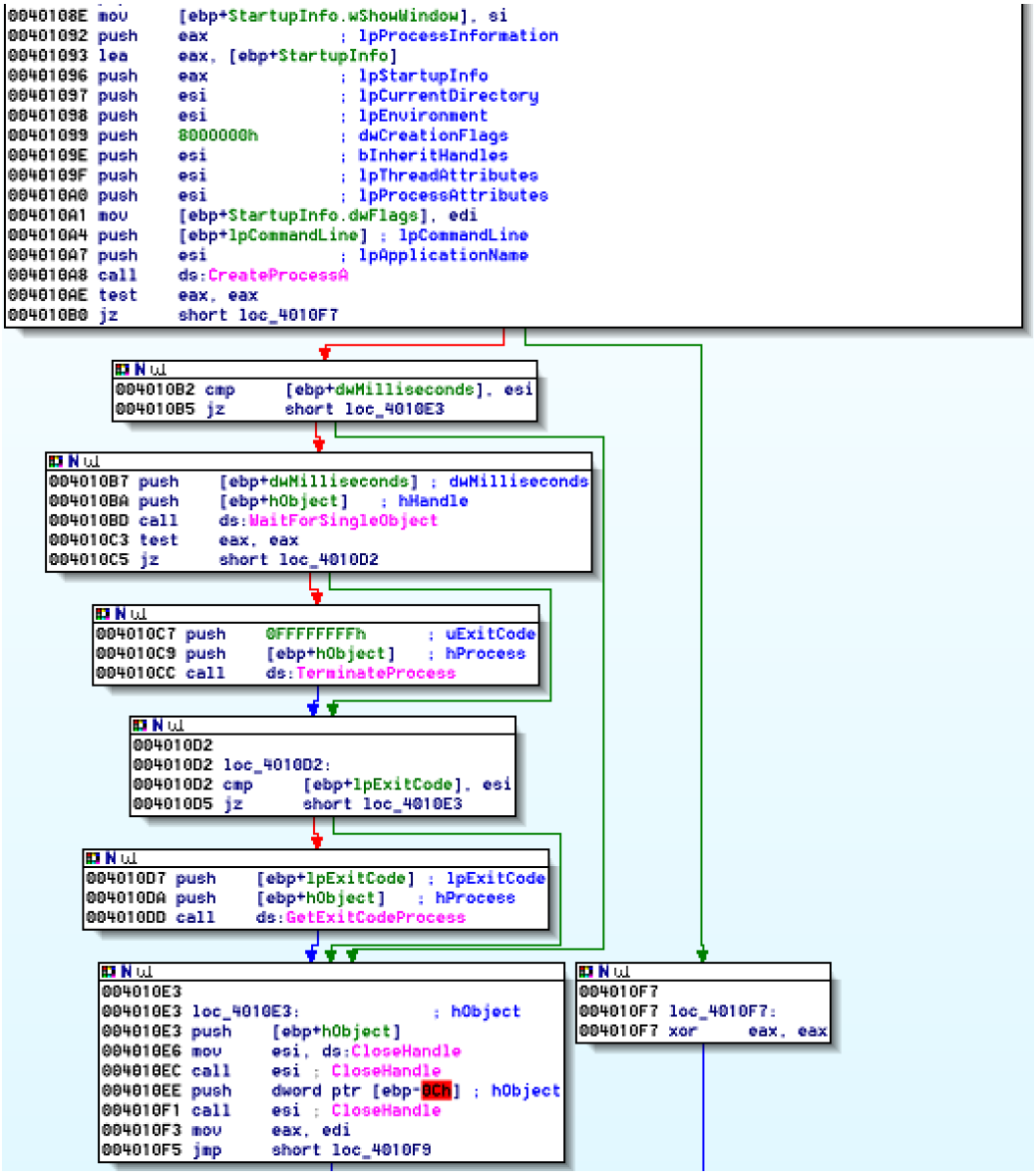
IDA Pro is a disassembler used to look at the assembly code of a malware, without running it. A disassembler is useful to look at before a debugger because of the fact it is an advanced static tool, the malware isn't running, and an analyst can find key memory locations to later view in x32dbg.

```
00000000004020C8 mov     [esp+6F4h+var_6F4], offset aWncry2ol7 ; "WNcry@2ol7"
00000000004020CF push    ebx             ; hModule
00000000004020D0 call    sub_401DAB
```

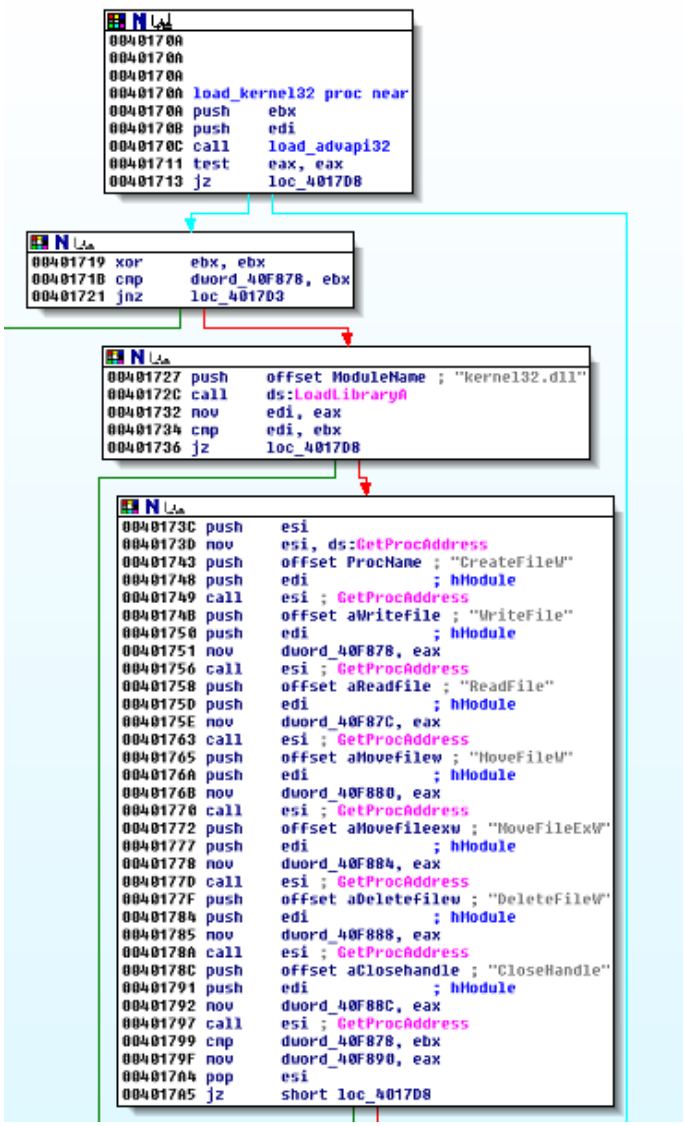
I found the string "WNcry@2ol7" within the main method of the executable and it gets passed to a subroutine, sub_401DAB. This subroutine opens the XIA resources and extracts something from the file. I will be looking into this function with x32dbg.

```
004020DA push    ebx             ; lpExitCode
004020DB push    ebx             ; dwMilliseconds
004020DC push    offset CommandLine ; "attrib +h ."
004020E1 call    modify_file_perms
004020E6 push    ebx             ; lpExitCode
004020E7 push    ebx             ; dwMilliseconds
004020E8 push    offset aIcacls_GrantEv ; "icacls . /grant Everyone:F /T /C /Q"
004020ED call    modify_file_perms
004020F2 add     esp, 20h
```

After further analysis, I noticed the sub_401064 was modifying the permissions on all of the files on the system. Which is also why I changed the name of the subroutine to modify_file_perms. Referring to documentation, attrib "displays, sets, or removes attributes assigned to files or directories." The +h after the attrib sets the Hidden file attribute. Again referring to documentation, icacls "displays or modifies discretionary access control lists (DACLs) on specified files, and applies stored DACLs to files in specified directories." The F means full control, the /T "performs the operation on all specified files in the current directory and its subdirectories," the /C "continues the operation despite any file errors. Error messages will still be displayed", and the /Q "suppresses success messages." In short when the icacls command is executed with these parameters, it's going to apply those permissions to all folders, subfolders, and files on the system. (MSDN Documentation)



The screenshot above shows, almost the entire, code execution for the modify_file_perms subroutine. This is the subroutine that “attrib +h” and “icacls . /grant Everyone:F /T /C /Q” are passed to. The subroutine starts by creating a process and then calling the WaitForSingleObject function which will “wait until the specified object is in the signaled state or the time-out interval elapses.” (MSDN Documentation)



This subroutine is used to load in the kernel32.dll and the system calls it'll be using, such as CreateFileW, WriteFile, DeleteFileW, and more. Also in the first code block there is a call to another subroutine, renamed to load_advapi32, shown in the next screenshot.

This subroutine is used to load in the advapi32.dll and the system calls it'll be using, such as CryptEncrypt, CryptDecrypt, CryptGenKey, and more.

Helpful Memory Addresses:

These memory addresses will help locate key subroutine, or functions, to observe step by step in x32dbg

- main - 401FE7
- modify_file_perms - 401064
- load_kernel32 - 4020F5
- load_advapi32 - 40170C
- Opens XIA resource and writes to c.wnry - 401DAB

Sources:

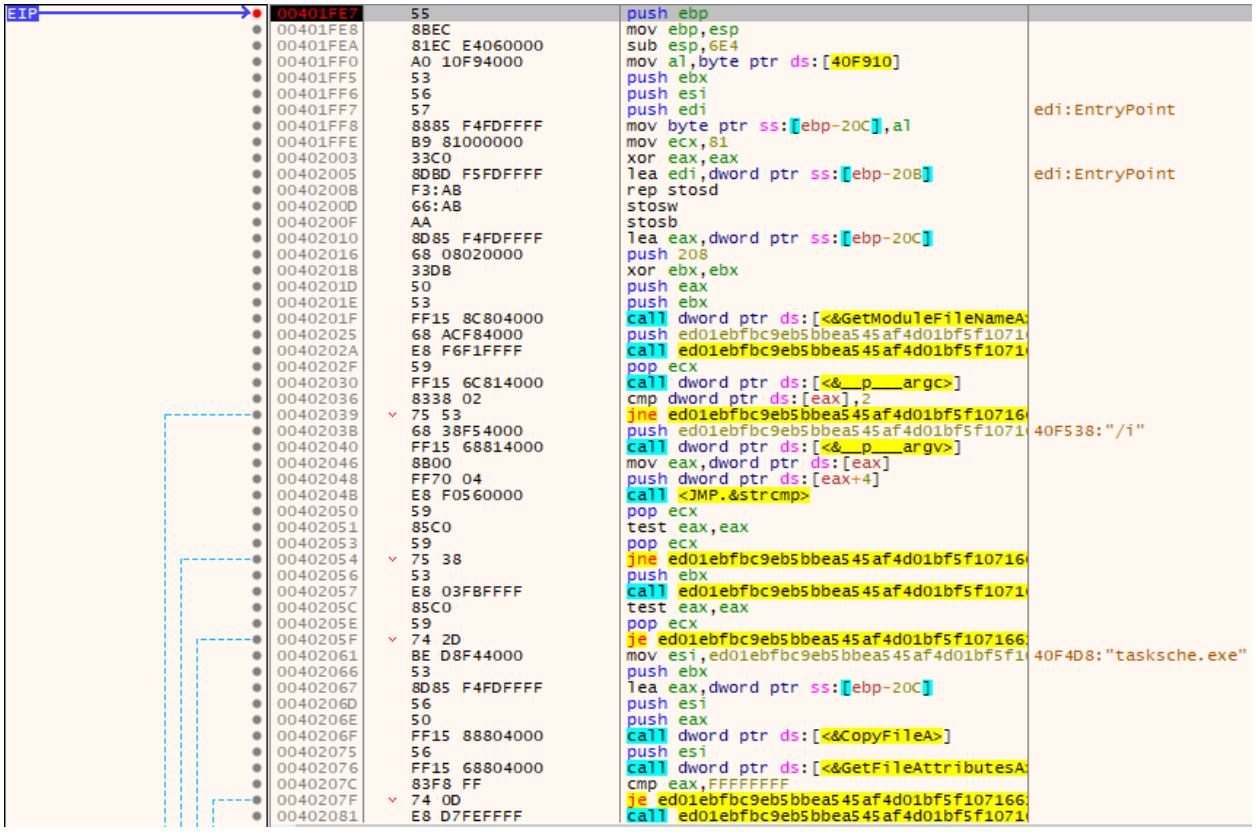
- MSDN Documentation icacls - https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/icacls
- MSDN Documentation attrib - https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/attrib
- MSDN Documentation WaitForSingleObject - https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject

Advanced Dynamic Analysis

Advanced dynamic analysis is another phase where an analyst is running the malware on their system. Again, it is vital that when dealing with live malware to be in a safe environment where the malware cannot affect the physical machine or computers on the network. The tool I'll be using for my advanced dynamic analysis of WannaCry is x32dbg.

x32dbg

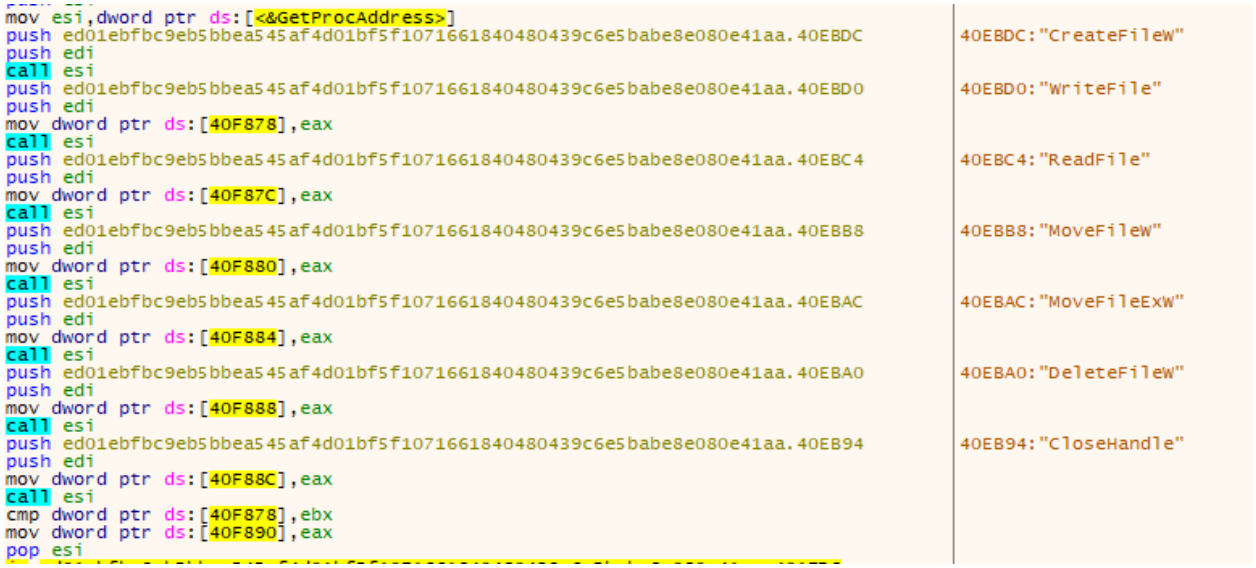
x32dbg is a debugging tool where an analyst can run the malware and see what happens, every step of the way. A debugger is useful for an analyst if they aren't sure what a certain function does; they can step through it step by step and determine what it does.



The code section above shows the main method. Some important things I noticed in this screenshot is the development of the /i command (which may be used to build the icacls commands) and the building of tasksche.exe, which is an executable that will be installed on the system.

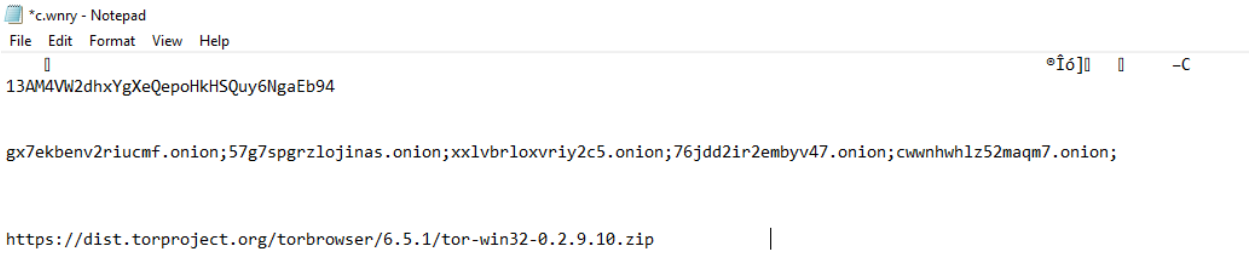
```
xor eax,eax
pop ecx
lea edi,dword ptr ss:[ebp-50]
mov dword ptr ss:[ebp-54],44
xor esi,esi
rep stosd
lea edi,dword ptr ss:[ebp-C]
mov dword ptr ss:[ebp-10],esi
stosd
stosd
stosd
push 1
lea eax,dword ptr ss:[ebp-10]
pop edi
mov word ptr ss:[ebp-24],si
push eax
lea eax,dword ptr ss:[ebp-54]
push eax
push esi
push esi
push 8000000
push esi
push esi
push esi
mov dword ptr ss:[ebp-28],edi
push dword ptr ss:[ebp+8]
push esi
call dword ptr ds:[<&CreateProcessA>]
test eax,eax
je ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.4010F7
cmp dword ptr ss:[ebp+C],esi
je ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.4010E3
push dword ptr ss:[ebp+C]
push dword ptr ss:[ebp-10]
call dword ptr ds:[<&WaitForSingleObject>]
test eax,eax
je ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.4010D2
push FFFFFFFF
push dword ptr ss:[ebp-10]
call dword ptr ds:[<&TerminateProcess>]
cmp dword ptr ss:[ebp+10],esi
je ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.4010E3
push dword ptr ss:[ebp+10]
push dword ptr ss:[ebp-10]
call dword ptr ds:[<&GetExitCodeProcess>]
push dword ptr ss:[ebp-10]
mov esi,dword ptr ds:[<&CloseHandle>]
```

Besides the encryption methods, I'd argue that this function is one of the most important in WannaCry. I discovered this function through IDA Pro, named modify_file_perms, and what the function does is takes the command that is passed to it and applies that command to every file on the system. From my analysis I've only seen "attrib +h" and "icacls . /grant Everyone:F /T /C /Q" (refer to IDA Pro section for a more detailed explanation on what these commands do).



The screenshot above is the function that loads kernel32.dll. The reason I posted this screenshot is because it shows actually how the system calls are loaded in, by getting the process address first through GetProcAddress. The same thing happens when the avapi32.dll is loaded in just before kernel32.dll.

This function is called to open the XIA resource, seen in PEview, with the password "WNcry@2ol7." The password can be seen being passed to this function in main. Refer back to the first screenshot in the IDA Pro to see the password and the calling of this subroutine. The functions procedure starts by unlocking the XIA resource and taking resources from it. After obtaining all the desired resources, it locks the XIA resource and writes the resources to a c.wnry file. With some online research, I figured out the c.wnry file is a configuration file that holds important information WannaCry refers to overtime, like a way to connect to the server, five .onion sites, and a download for the Tor browser. I've searched for .onion links and a system call to InternetOpenA, but no results showed. Several reports by security firms have reported a link to http://www.iuqssfsodp9ifjaposdfjhgosurijfaewrwergwea.com with the InternetOpenA system call.



(Note: I did change the contents of c.wnry for visibility purposes. Originally everything was on one line with a lot of spaces between each entry)

I couldn't find the .onion websites looking through IDA Pro or Resource Hacker, however, I did find them looking through the c.wrnry file. There is also a download for the Tor browser.

Potential Dangers of the Malware

The WannaCry ransomware is not a malware you want on your system. It starts by looping through the files on your system giving all users access to them and after that it encrypts each file with a different 128 bit AES key. For more information on the encryption, read FireEye's report on it below. WannaCry encrypts mostly every file on your computer, avoiding the files necessary for the computer to startup and run. I'm not sure if it was the virtual machine or if it was truly WannaCry, however, I also noticed after running the executable, the computer would tend to slow down at times. Most antiviruses can now detect the WannaCry malware, so it is crucial to always have an antivirus running in case the executable ever finds a way onto your machine.

"The files are encrypted with a randomly generated 128-bit AES key in CBC mode with a NULL initialization vector. The key is generated per file, is encrypted with the generated RSA public key, and included in the encrypted file header. Each file encrypted by the malware starts with the string *WANACRY!* and has the *WNCRY* extension. Depending on the file properties, the malware may also stage files in a *WNCRYT* extension." - FireEye

Sources:

- FireEye's Report - https://www.fireeye.com/blog/threat-research/2017/05/wannacry-malware-profile.html

How to Remove the Malware

I have found mixed reviews online and I'm not sure as to which actually work. Malwarebytes, Avast, and Symantec are among the few companies that claim that their antivirus (AV) software has the capability to remove the WannaCry ransomware. However, I'm not sure if their AV software actually reverses the effect of the malware or just gets rid of the executable. WannaCry also creates a file on the desktop telling an infected user that their AV software could potentially remove the decrypting software. If the AV software removes the executable after execution it isn't doing much for the user. It also doesn't help the user if they're debating on paying the ransom, to receive their files back, and their AV removes the decrypting software. More people need to be educated on how to secure their machines from malware, what phishing emails are, and how to distinguish a real email from a fake one. A lot of malware is spread through phishing emails because users don't think twice before clicking on a link or an attachment in an email.

The WannaCry ransomware also had a worm-like feature to it. Reverse engineers found out that it was an unregistered domain that was helping the virus spread. The domain was quickly registered and WannaCry could no longer spread. However, many different versions of WannaCry have sprung up, using different domains.

What I Learned

What I Learned on this Project:
Throughout the course of this project, I learned how to safely reverse the ransomware WannaCry through all the tools mentioned above. I also learned how the use of some tools can be useful for finding out more in other tools later on. For example, using IDA Pro to find out key memory locations to jump to and follow in x32dbg. Another example would be looking at VirusTotal's report and then looking for their findings in other static analysis tools or dynamic analysis tools.

What I Learned this Semester:
Besides everything I learned on this project and the tools we used throughout the course of the semester, I also learned more about the assembly language, how to convert assembly code to C code, and the differences between an x86 and x64 architecture.