SPRING 2023 CS390R Final Project - Checkpoint #2

WannaCry Ransomware Research

Everything that is described in this research that was conducted on existing documentation of the ransomware WannaCry and the various components that allow for the ransomware to work. Our goal is to be able to search for the sections in the ransomware binary that correspond to these different functionalities.

WannaCry depends on two different things: spreading itself through the target's network and the encryption of files on the target's machine.

As mentioned in our previous checkpoint, we know that WannaCry depends on a vulnerability found in Windows' ***Server Message Block (SMB)*** version 1.0 protocol for the spread of itself in the network.

The Server Message Block (SMB) Protocol

The SMB protocol is a response-request communication protocol that is used to share access for resources on a network. It operates over TCP ports 139 and 445. The SMB protocol allow for things called SMB transactions that give permissions for read and write to be performed by a SMB client and server. These transactions become topical when it comes to exploiting the vulnerability in SMB. SMBv1 was developed in the 1980s and can be vulnerable to pre-authentication integrity, encryption, and other exploits if not patched. This got patched on March 14, 2017 in security bulletin MS17-010.

- What is the Server Message Block (SMB) protocol? How does it work? (techtarget.com)

SMBv1 Vulnerabilities

SMBv1's MD5 Algorithm

SMBv1 uses the MD5 (message-digest) algorithm that is a cryptographic protocol that is used to authenticate messages, content verification, and digital signatures. It takes some message of arbitrary length as an input and returns a fixed-length digest hash value as an output that is used to authenticate the original input message. Since this hashing algorithm was used to encrypt files, files that were encrypted using this method have some sort of long hash value as an output.

MD5 is vulnerable to attackers because they can take an expected hash value for one file and create an entirely different file that has the exact same hash.

SMBv1's MD5 algorithm is something that we learned about while learning about SMBv1 and its potential vulnerabilities, but as per the existing documentation of WannaCry, it does not seem that the MD5 algorithm was used for encryption purposes in the ransomware but instead just to keep track of file authenticity.

- How to disable SMB v1 (Server Message Block) (manageengine.com)
- What Is the MD5 Hashing Algorithm & How Does It Work? | Avast

SMB Transaction Buffer Overflow

Buffer overflows allow for attackers to move to certain parts of a program by sending large size payloads and then obtaining remote code execution.

As mentioned earlier, SMB transactions allow for read/write to be performed between a SMB client and server. If the message that is communicated between the two is greater than the SMB maximum buffer size `MaxBufferSize` , then the remaining messages are sent as secondary requests (`Secondary Trans2`), affecting the `srv2.sys` kernel driver.

- SMB Exploited: WannaCry Use of "EternalBlue" | Mandiant

WannaCry's SMBv1 Exploit and EternalBlue

In the malware lecture, we talked about Eternal Blue, which is a computer exploit that was developed by the NSA to exploit the SMB protocol. Later on, this exploit was repackaged into WannaCry. This malware spreads to unpatched pre-MS17-010 Windows systems on a network that has SMBv1 enabled, eventually allowing remote code execution. WannaCry utilizes EternalBlue by creating custom SMB session requests with specialized packets that contained hardcoded local IP values found in the ransomware.

The exploit first checks to see if the kill switch domain is available (iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com). If it is not available, then it does the following:

The ransomware scans (port scan) the target network to see if traffic is allowed on port 455 (to determine if SMB can be ran).

Afterwards, an initial SMB handshake (request/response) is done. Then, WannaCry connects to the IPC$ share -- a null session connection where Windows allows anonymous users to perform certain activities -- on the machine.

An `NT Trans` request gets sent out after the handshaking is done, which is the transaction buffer overflow that was mentioned earlier. This buffer overflow that contains a sequence of `NOP` s moves the SMB server machine to a state where another payload can be sent for exploitation. Specifically, after the request, multiple `Secondary Trans2 Requests` attempt to accommodate the large payload size. These `Secondary Trans2` requests are malformed and act as triggers for the vulnerability where the request data portion can contain shellcode. This shellcode launches the ransomware.

- WannaCry Malware Profile | Mandiant
- SMB Exploited: WannaCry Use of "EternalBlue" | Mandiant
- How does the WannaCry malware work (tutorialspoint.com)

WannaCry's Encryption System

We know that the WannaCry ransomware depends on asymmetric and symmetric encryption for encrypting files. Although we outlined this in our initial checkpoint, this checkpoint will delve more into the intricacies of how these types of encryption work in general as well as how WannaCry utilizes it.

Before Encryption: Identifying Specific Files

Here are the file extensions that are targeted, with a focus on productivity and database applications like Microsoft Excel, etc.

```
.der .pfx .key .crt .csr .p12 .pem .odt .ott .sxw .stw .uot .3ds .max .3dm .ods .ots
.sxc .stc .dif .slk .wb2 .odp .otp .sxd .std .uop .odg .otg .sxm .mml .lay .lay6 .asc
.sqlite3 .sqlitedb .sql .accdb .mdb .dbf .odb .frm .myd .myi .ibd .mdf .ldf .sln .suo
.cpp .pas .asm .cmd .bat .ps1 .vbs .dip .dch .sch .brd .jsp .php .asp .java .jar
.class .mp3 .wav .swf .fla .wmv .mpg .vob .mpeg .asf .avi .mov .mp4 .3gp .mkv .3g2
.flv .wma .mid .m3u .m4u .djvu .svg .psd .nef .tiff .tif .cgm .raw .gif .png .bmp .jpg
.jpeg .vcd .iso .backup .zip .rar .tgz .tar .bak .tbk .bz2 .PAQ .ARC .aes .gpg .vmx
.vmdk .vdi .sldm .sldx .sti .sxi .602 .hwp .snt .onetoc2 .dwg .pdf .wk1 .wks .123 .rtf
.csv .txt .vsdx .vsd .edb .eml .msg .ost .pst .potm .potx .ppam .ppsx .ppsm .pps .pot
.pptm .pptx .ppt .xltm .xltx .xlc .xlm .xlt .xlw .xlsb .xlsm .xlsx .xls .dotx .dotm
.dot .docm .docb .docx .doc
```

Symmetric Encryption (AES)

The symmetric encryption that WannaCry uses is AES-128. This system takes data, divides it into 10 fixed-size blocks (10 since it's 128bit), processes each block using mathematical operations called rounds, and ultimately transforms all of it into ciphertext via some secret key. In WannaCry, there is only one key that is used for both encryption and decryption in the AES cryptosystem.

Asymmetric Encryption (RSA public-key)

The asymmetric encryption that WannaCry uses is a private RSA-2048 key pair specific to each individual infection. RSA numbers are sets of large numbers with exactly two prime factors. RSA-2048 has 2048 bits, or 617 decimal digits, and is the largest of the RSA numbers. The cryptosystem involves key generation, key distribution, encryption, and decryption.

The following modular exponentiation for all integers m where $0 \leq m < n$ is: $(m^e)^d \equiv m \pmod{n}$

where the modulus *n* and public/encryption exponent *e* correspond to the public key, the modulus *n* and private/decryption exponent *d* for private key, and *m* for the message.

Encryption works with the following modular exponentiation:

$$c \equiv m^e \pmod{n}$$

where c is the ciphertext and decryption works with the following:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}.$$

WannaCry's Encryption Process

As detailed in the last checkpoint, this is the order of events that WannaCry takes as the ransomware runs:

1. makes symmetric and asymmetric keys -- *ATTACKER SIDE*
2. encrypts files with asymmetric key -- *ATTACKER SIDE*
3. encrypts asymmetric key with symmetric key -- *VICTIM SIDE*
4. send private key to attacker then delete -- *VICTIM SIDE*
5. delete private key -- *VICTIM SIDE*

The Spread of WannaCry

Once a machine is compromised with the ransomware, the ransomware automatically spreads itself via the target's network to other devices that are susceptible to this vulnerability.

After encryption of one machine, it installs **DoublePulsar**, a backdoor tool that runs in kernel mode allowing attackers control over a target system, as a payload that spreads copies of WannaCry onto more systems of vulnerable TCP port 455 machines.

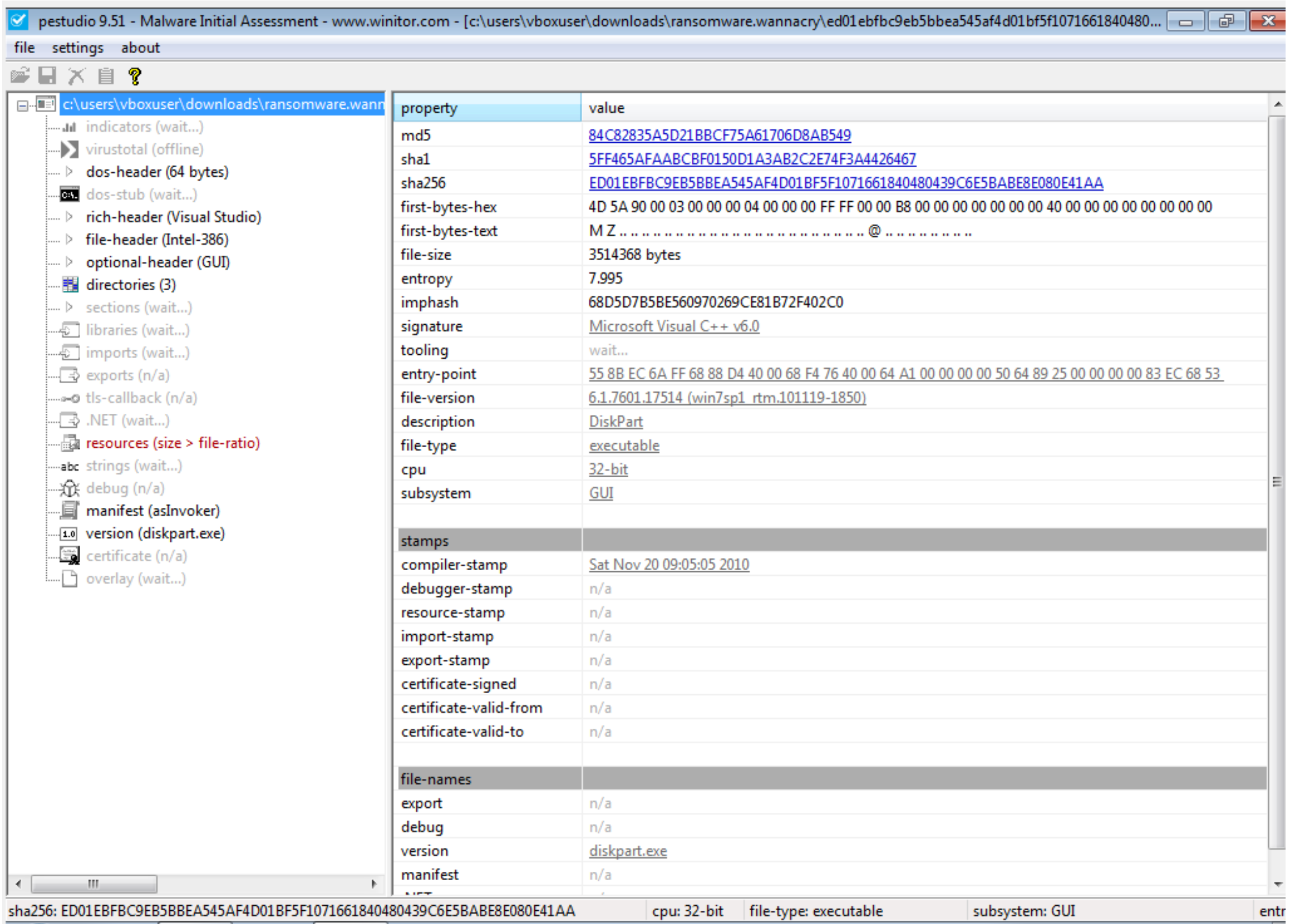
- What is the WannaCry Ransomware Attack? | UpGuard

Scripting Attempts

Began working on a Ghidra script to view the control flow of the program along with the total amount of times each function is called but ran into issues finding info from the API. Another idea was to find all referenced functions from DLLs but not entirely sure if this needs to be automated at all. Most of my time went into watching Youtube tutorials on how to write scripts for Ghidra or trying to search the API for methods to get certain information.

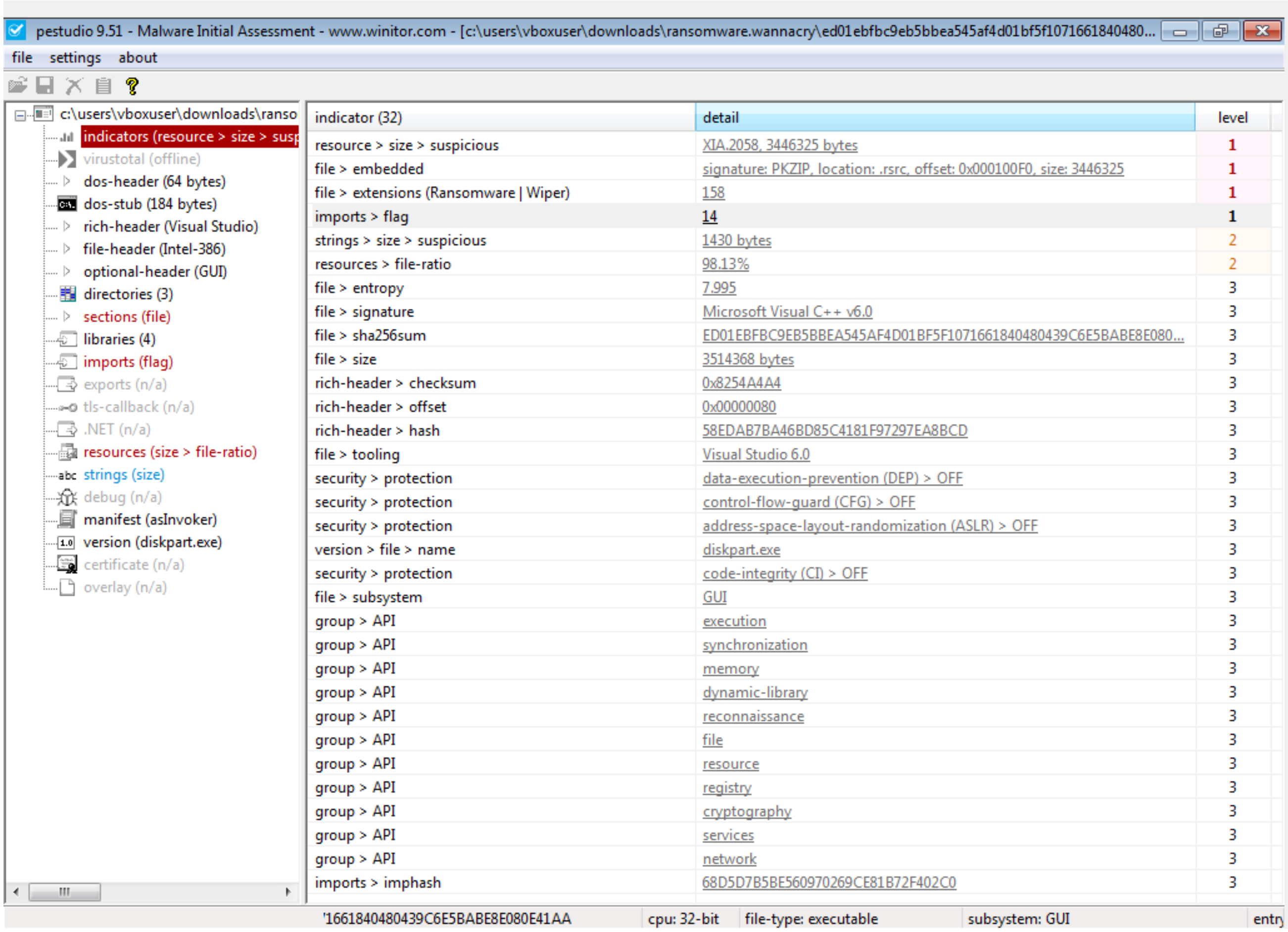
=======

PeStudio Analysis



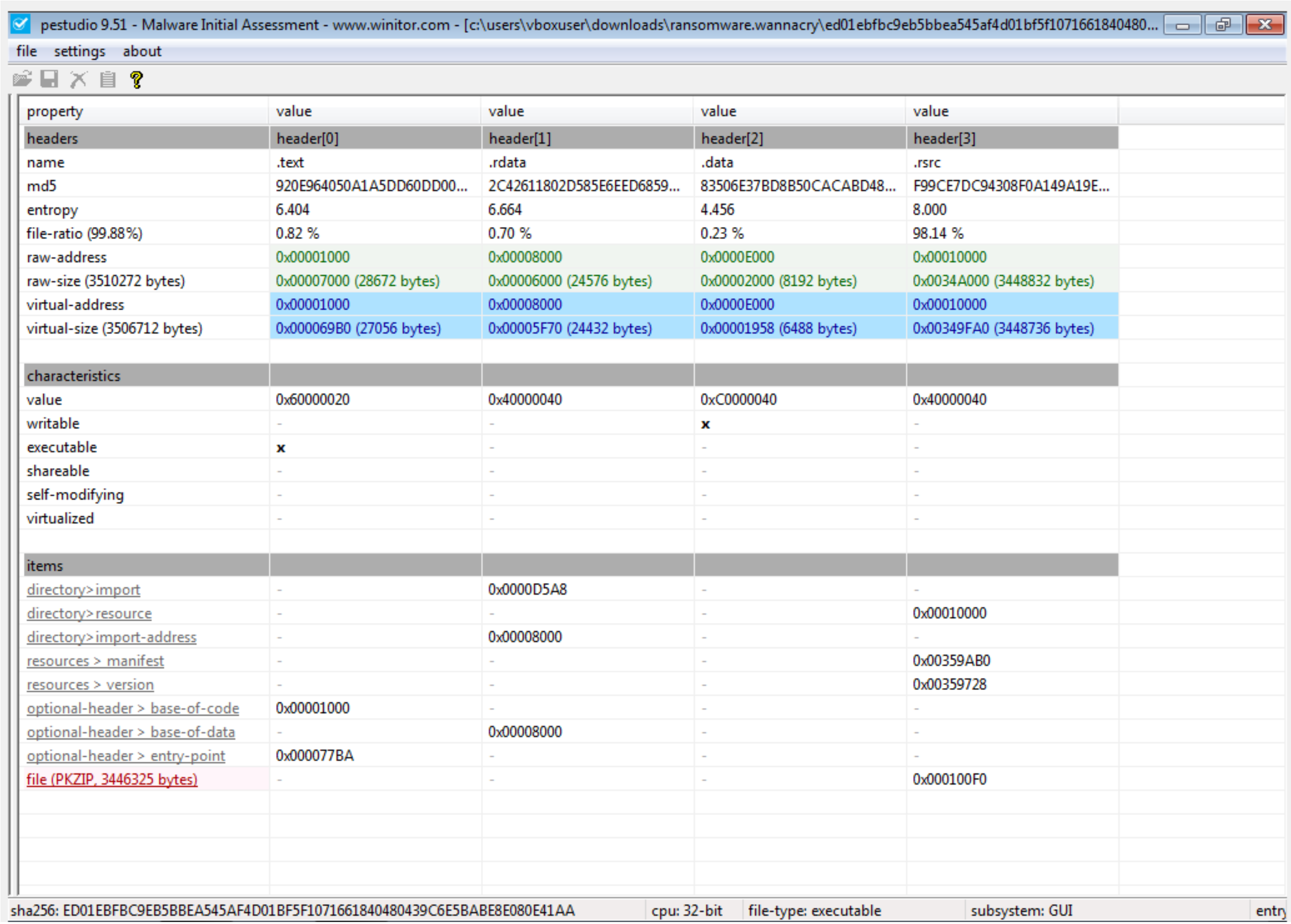
We used PeStudio to analyze WannaCry. PeStudio is used to analyze Windows executables for malware and has a host of different features. Running it on the WannaCry executable we are told that its entropy, a measure of a malware's obfuscation, is 7.995 on a range of 0-8. Which means the malware is heavily obfuscated and packed. After letting PeStudio run, it returns with a few malicious indicators. The top 4 are:

- Resource sizes
- Embedded files
- File extensions
- Imports



Resource Sizes

The executable consists of 4 headers: .text, .data, .rdata, and .rsrc. The .rscr section contains the resources required by the program. In this executable, the .rsrc header consists of 98.14% of the entire program, a common sign of malware as a method of obfuscation.



pestudio 9.51 - Malware Initial Assessment - www.winitor.com - [c:\users\vboxuser\downloads\ransomware.wannacry\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480...

file settings about

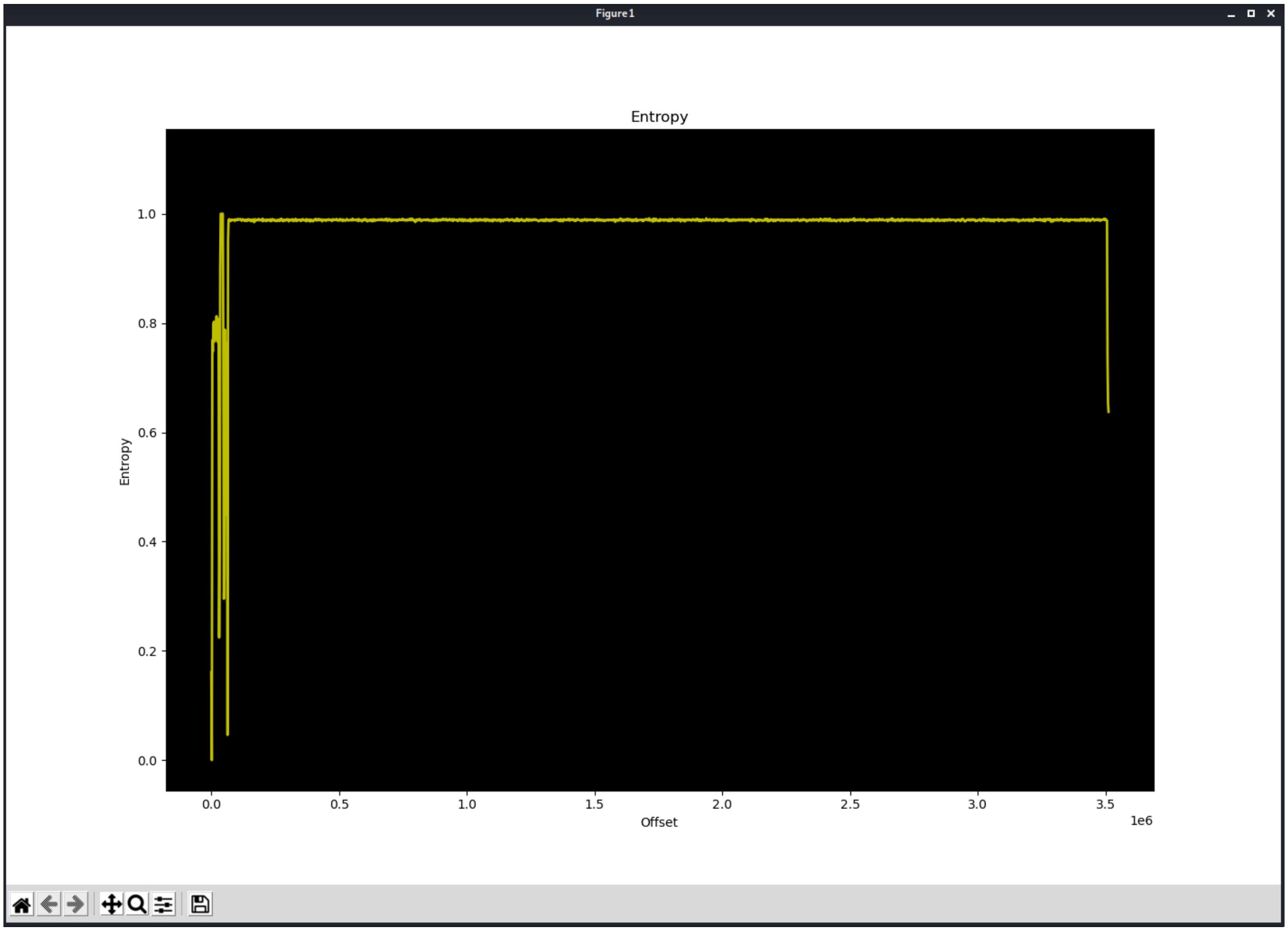
property	value	value	value	value
headers	header[0]	header[1]	header[2]	header[3]
name	.text	.rdata	.data	.rsrc
md5	920E964050A1A5DD60DD00...	2C42611802D585E6EED6859...	83506E37BD8B50CACABD48...	F99CE7DC94308F0A149A19E...
entropy	6.404	6.664	4.456	8.000
file-ratio (99.88%)	0.82 %	0.70 %	0.23 %	98.14 %
raw-address	0x00001000	0x00008000	0x0000E000	0x00010000
raw-size (3510272 bytes)	0x00007000 (28672 bytes)	0x00006000 (24576 bytes)	0x00002000 (8192 bytes)	0x0034A000 (3448832 bytes)
virtual-address	0x00001000	0x00008000	0x0000E000	0x00010000
virtual-size (3506712 bytes)	0x000069B0 (27056 bytes)	0x00005F70 (24432 bytes)	0x00001958 (6488 bytes)	0x00349FA0 (3448736 bytes)
characteristics				
value	0x60000020	0x40000040	0xC0000040	0x40000040
writable	-	-	x	-
executable	x	-	-	-
shareable	-	-	-	-
self-modifying	-	-	-	-
virtualized	-	-	-	-
items				
directory>import	-	0x0000D5A8	-	-
directory>resource	-	-	-	0x00010000
directory>import-address	-	0x00008000	-	-
resources > manifest	-	-	-	0x00359AB0
resources > version	-	-	-	0x00359728
optional-header > base-of-code	0x00001000	-	-	-
optional-header > base-of-data	-	0x00008000	-	-
optional-header > entry-point	0x000077BA	-	-	-
file (PKZIP, 3446325 bytes)	-	-	-	0x000100F0

sha256: ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA | cpu: 32-bit | file-type: executable | subsystem: GUI | entry

Embedded Files

PeStudio found that the executable is hiding an embedded file in the .rsrc section called PKZIP. PKZIP is what is taking up most of the file space across the entire program, so this is our actual malware. Putting the malware here instead of in .text where the program would generally go is another method of obfuscation and packing, making it harder for the programs intent to be discovered.

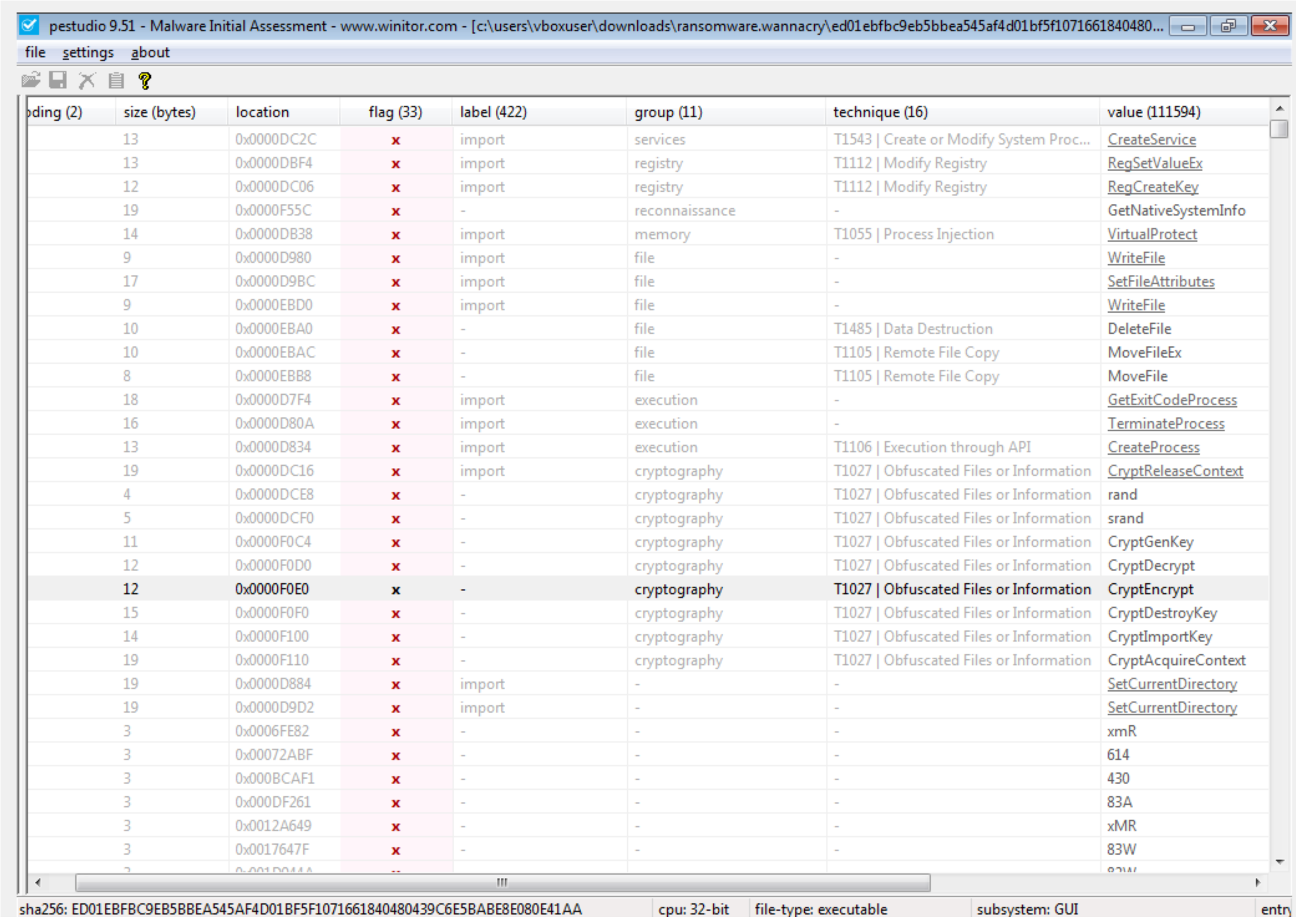
This is supported by performing entropy analysis using binwalk. The command used being : binwalk -E wannacry.exe



This entropy graph helps validate the findings through PeStudio. We can see sections of low entropy in the start of the binary, those being the text and data sections. However we can see the vast majority of the binary has extremely high entropy, meaning it's either compressed or encrypted. This helps validate the findings of the PeStudio analysis and confirm that the majority of the binary is compressed as well as where that compression actually is whithin the binary. This helps speed up analysis of the binary as we know which sections have readable data and which are compressed and difficult to decipher.

File Extensions

This is where PeStudio caught many malicious strings in the program. We can see that these strings are representing different function calls and Windows Systems calls. These include modifying registry values, copying, writing to, and destroying files, creating and modifying system services and processes, creating and destroying cryptographic keys, and encrypting file data. From these files, PeStudio tells us that the program is most likely ransomware or wiper malware, which WannaCry most certainly is.



pestudio 9.51 - Malware Initial Assessment - www.winitor.com - [c:\users\vboxuser\downloads\ransomware.wannacry\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480...

file settings about

oding (2)	size (bytes)	location	flag (33)	label (422)	group (11)	technique (16)	value (111594)
	13	0x0000DC2C	x	import	services	T1543 | Create or Modify System Proc...	CreateService
	13	0x0000DBF4	x	import	registry	T1112 | Modify Registry	RegSetValueEx
	12	0x0000DC06	x	import	registry	T1112 | Modify Registry	RegCreateKey
	19	0x0000F55C	x	-	reconnaissance	-	GetNativeSystemInfo
	14	0x0000DB38	x	import	memory	T1055 | Process Injection	VirtualProtect
	9	0x0000D980	x	import	file	-	WriteFile
	17	0x0000D9BC	x	import	file	-	SetFileAttributes
	9	0x0000EBD0	x	import	file	-	WriteFile
	10	0x0000EBA0	x	-	file	T1485 | Data Destruction	DeleteFile
	10	0x0000EBAC	x	-	file	T1105 | Remote File Copy	MoveFileEx
	8	0x0000EBB8	x	-	file	T1105 | Remote File Copy	MoveFile
	18	0x0000D7F4	x	import	execution	-	GetExitCodeProcess
	16	0x0000D80A	x	import	execution	-	TerminateProcess
	13	0x0000D834	x	import	execution	T1106 | Execution through API	CreateProcess
	19	0x0000DC16	x	import	cryptography	T1027 | Obfuscated Files or Information	CryptReleaseContext
	4	0x0000DCE8	x	-	cryptography	T1027 | Obfuscated Files or Information	rand
	5	0x0000DCF0	x	-	cryptography	T1027 | Obfuscated Files or Information	srand
	11	0x0000F0C4	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptGenKey
	12	0x0000F0D0	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptDecrypt
	12	0x0000F0E0	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptEncrypt
	15	0x0000F0F0	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptDestroyKey
	14	0x0000F100	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptImportKey
	19	0x0000F110	x	-	cryptography	T1027 | Obfuscated Files or Information	CryptAcquireContext
	19	0x0000D884	x	import	-	-	SetCurrentDirectory
	19	0x0000D9D2	x	import	-	-	SetCurrentDirectory
	3	0x0006FE82	x	-	-	-	xmR
	3	0x00072ABF	x	-	-	-	614
	3	0x000BCAF1	x	-	-	-	430
	3	0x000DF261	x	-	-	-	83A
	3	0x0012A649	x	-	-	-	xMR
	3	0x0017647F	x	-	-	-	83W

sha256: ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA | cpu: 32-bit | file-type: executable | subsystem: GUI | entry

Imports

Finally, we have the imports. Many malwares rely on importing various DLLs to perform different functions just like any other program. But by studying which imports a program uses, we can determine if it is a malicious actor or not. In this case, the program is using imports to create services and processes, edit registry values, write to files, and calling rand and srand to generate cryptographic keys. One of the specific imports it uses is VirtualProtect.

VirtualProtect

Virtual protect is a function that changes the protection of a memory address in the process's space. This allows the malware to read, write, and execute from various parts of the memory, which lets a malicious actor write and execute code anywhere they want if they have the permissions.

Scripting Attempts

Began working on a Ghidra script to view the control flow of the program along with the total amount of times each function is called but ran into issues finding info from the API. Another idea was to find all referenced functions from DLLs but not entirely sure if this needs to be automated at all.

Continued Analysis

Additionally found the winMain function from entry and discovered more functionality further in. Found where a directory is created and a file that seems to have the attributes set to those of tasksche.exe.