Practical Malware Analysis & Triage

Malware Analysis Report

WannaCry Ransomware

Nov 2023 | Muhammad Osama Khalid | v1.0

Table of Contents

Executive Summary

MD5 Hash	db349b97c37d22f5ea1d1841e3c89eb4
SHA1 Hash	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA256 Hash	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

WannaCry is a ransomware that came into light in May 2017, when it spread across the world and encrypts over hundreds of thousands of computers in more than 150 countries. It targets the systems running Microsoft Windows by encrypting the data on the infected system and demanding a ransom payment in Bitcoin in order to decrypt the data. It also has the capability to spread itself through networks by exploiting a vulnerability in Microsoft Windows known as EternalBlue.

The WannaCry malware was written in C++. When the malware is executed, it attempts to connect to a URL that has been hardcoded into its code. If the connection to the URL is successful, the malware stops its execution. However, if it fails to connect to the URL, the malware will continue to execute. After executing the malware, it creates a file named "tasksche.exe" that contains the main payload of the malware that is used to encrypt the data on the infected system.

The YARA signature rule was created for malware and attached in the Rules and Signatures section. The malware sample and hashes were submitted to VirusTotal for further analysis.

High-Level Technical Summary

WannaCry consists following steps:

1. Upon execution, the malware attempts to connect to a domain called "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com". If the connection to the URL is successful, the malware stops its execution.
2. If the connection to the URL fails, then the malware creates a service named “mssecsvc2.0” with the DisplayName “Microsoft Security Centre (2.0) Service” and then uses this service to locate and infect other systems on the network using SMB port 445.
3. It then unpacks a resource named "1831" that is packed inside it, drops it into the "C:\Windows" folder, and renames it to "tasksche.exe". After that, it executes the dropped binary with the “/i” argument.
4. Upon execution "tasksche.exe" checks to see if the mutex “MsWinZonesCacheCounterMutexA” exists, and will stop execution if the mutex is present.
5. If the mutex is not present, then "tasksche.exe" obtains the NetBIOS name of the local computer and then obfuscates it.
6. After that "tasksche.exe" creates a folder with the obfuscated name within "C:\ProgramsData" and saves its copy to the folder. Also, it starts unpacking the malware files in the folder, which are then used to encrypt the files on the system.
7. Once the malware files are unpacked and saved in the specified directory, the program executes the "attrib.exe" utility with the "attrib +h" command to hide the folder. Furthermore, it utilizes the "icacls.exe" utility with the "icacls ./grant Everyone:F /T /C /Q" command to provide full access to all users for all files in the folder.
8. "tasksche.exe" creates a service named after the folder it makes in "C:\ProgramsData" and sets it to run "tasksche.exe" on startup.
9. After completing the encryption routine it executes "@WanaDecryptor@.exe" and also replaces the wallpaper with "@WanaDecryptor@.bitmap".

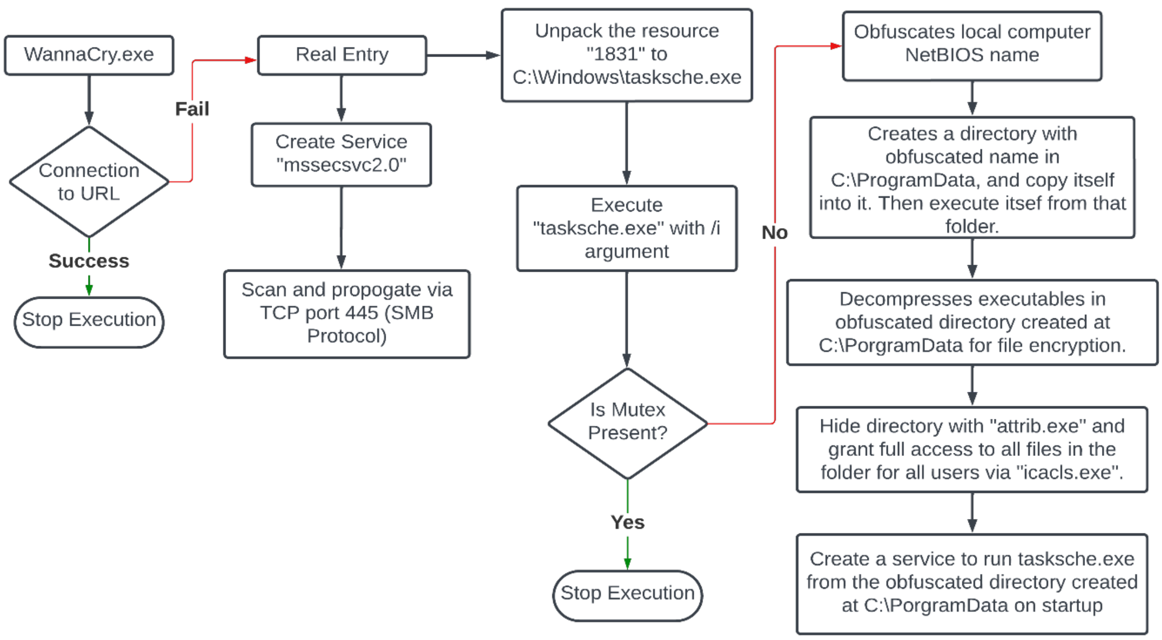


Figure 1: Basic Flow of Execution

Malware Composition

WannaCry consists of the following components:
1. Ransomware.wannaCry.exe
2. tasksche.exe

1. WannaCry.exe

This is the initial part of the malware that runs and checks the connection to the domain. If the connection to the URL is Successful it stops the execution, otherwise, it unpacks the resource “1831” and drops it to C:\Windows, and renames it to “tasksche.exe”.

File Name	SHA256 Hash
Ransomware.wannaCry.exe	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

2. Tasksche.exe

This is the second stage of the malware that is responsible for creating a hidden folder in "C:\ProgramData", unpacks the malware files into the folder, and uses them to encrypt files on the system.

File Name	SHA256 Hash
tasksche.exe	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

Basic Static Analysis

Basic static analysis is a method of examining a malware sample without actually executing it. This includes identifying the type of binary, extracting any readable strings, inspecting file headers, comprehending machine-specific information, and identifying any potential dynamic link libraries (DLLs) and functions that the malware may use.

1. Magic Byte

A "magic byte" is a sequence of bytes that appears at the beginning of a file, serving as a signature to quickly recognize its structure, without relying on file extensions. In other words, it acts as a unique identifier for every file, enabling the system to determine its format accurately. Below are some examples of the magic bytes of commonly used files.

File	Magic Byte	Hex
EXE	MZ	4D 5A
PNG	PNG	50 4E 47
ZIP	PK	50 4B

To find the magic byte of the malware, I used HxD, and found that the magic byte of the binary is “MZ”.

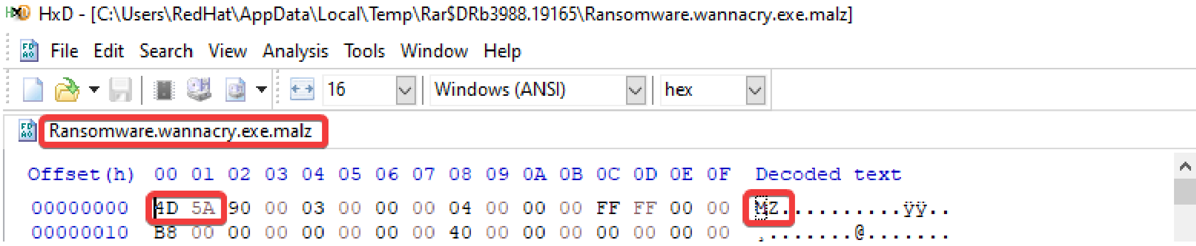


Figure 2: Magic Byte

2. VirusTotal Analysis

As we have already obtained the hash of the malware sample, therefore, I have submitted it to the VirusTotal and found that the hash is triggered as malicious by 70 out of 72 antivirus engines.

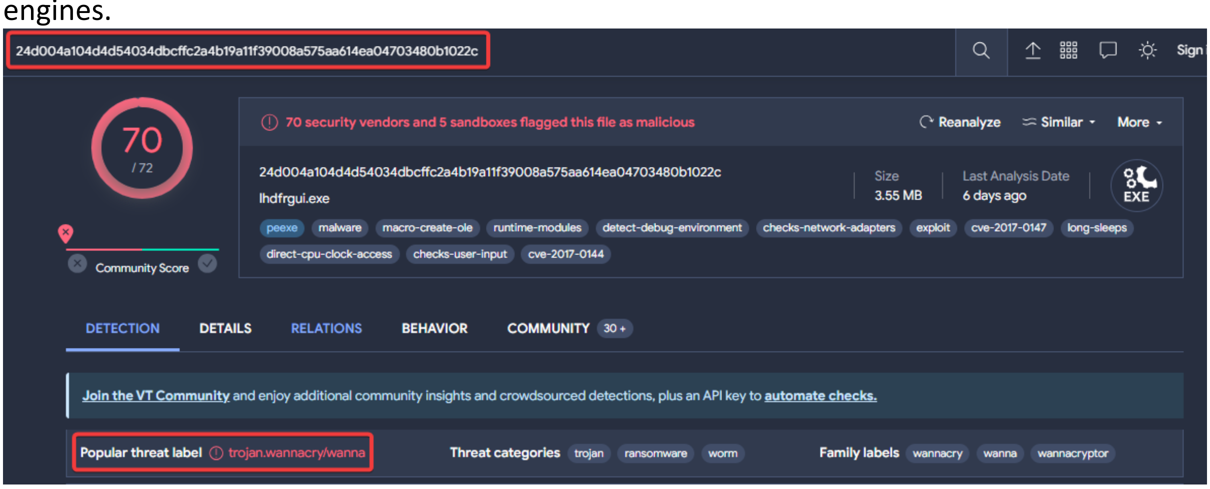


Figure 3: VirusTotal Analysis

3. Strings Analysis

Strings are the readable characters extracted from the malware file and are used to identify the functionality of the malware. To extract the strings from the malware file I use a tool called “Floss”. To extract the strings from the malware file and save them in the text file I run the following command

```
C:\Users\RedHat\Desktop
λ floss.exe Ransomware.wannacry.exe.malz > wannacry.txt
INFO: floss: extracting static strings...
finding decoding function features: 100%|████| 87/87 [00:00<00:00, 222.14 functions/s
INFO: floss.stackstrings: extracting stackstrings from 55 functions
INFO: floss.results: SMBu
```

Figure 4: Command to run Floss

Some of the important strings from the floss output are as follows

1. While analyzing the floss output, I found the name of the service that the malware might use along with some random paths, and the program named “tasksche.exe”.

```
mssecsvc2.0
Microsoft Security Center (2.0) Service
%s -m security
C:\%s\qeriuwjhrf
C:\%s\%s
WINDOWS
tasksche.exe
```

Figure 5: Floss Output

2. Find the hardcoded URL that the malware might try to connect with.

```
CreateFileA
CreateProcessA
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
!This program cannot be run in DOS mode.
```

Figure 6: Floss Output

3. Find some unzip libraries used by the malware.

```
 inflate 1.1.3 Copyright 1995-1998 Mark Adler
 n;^
Qkkbal
i]Wb
9a&g
MGiI
wn>Jj
#.zf
+o*7
- unzip 0.15 Copyright 1998 Gilles Vollant
```

Figure 7: Floss Output

4. Find the name of the mutex, that the malware uses to identify whether it is already running on the system or not.

```
%s%d
Global\MsWinZonesCacheCounterMutexA
tasksche.exe
```

Figure 8: Floss Output

5. Find the list of API calls associated with cryptography.

```
Microsoft Enhanced RSA and AES Cryptographic Provider
CryptGenKey
CryptDecrypt
CryptEncrypt
CryptDestroyKey
CryptImportKey
CryptAcquireContextA
```

Figure 9: Floss Output

6. Find the command line to call “cmd.exe”, along with hardcoded Bitcoin addresses.

```
cmd.exe /c "%s"
115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
%s%d
```

Figure 10: Floss Output

7. Find the commands to modify access of a file or folder along with the command to hide the file or folder.

```
icacls . /grant Everyone:F /T /C /Q
attrib +h .
```

Figure 11: Floss Output

8. Find the names of some files and executables that are possibly dropped by the malware.

```
r.wnry
JCg4K_
s.wnry
t.wnry
*($:{
taskdl.exe
taskse.exe
1N$D
u.wnry
```

Figure 12: Floss Output

9. Find some IP Addresses that the malware might use.

```
Windows 2000 5.0
\\172.16.99.5\IPC$
Windows 2000 2195
Windows 2000 5.0
\\192.168.56.20\IPC$
kernel32.dll
WanaCrypt0r
```

Figure 13: Floss Output

4. Packed or Unpacked Analysis

Malware developers often use packing techniques to hide the malicious code of their malware. When packed, the malware first unpacks itself before executing its intended actions. The information about the packing is stored in the malware file, which helps the malware to unpack itself during execution. I used the tool "Exeinfo" to check whether the malware was packed or not. After analyzing the output of "Exeinfo," I found that the malware was not packed. Additionally, "Exeinfo" indicated that the malware had four sections along with a zip archive embedded in it, containing a total of nine files.

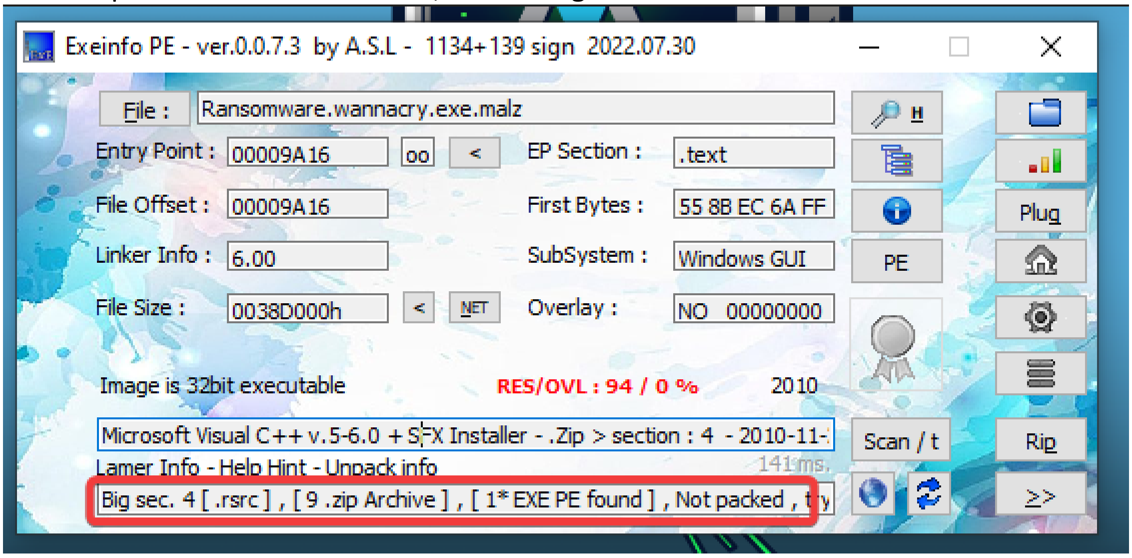


Figure 14: Exeinfo Output

5. PEStudio File Analysis

For further analysis of the malware, I used the tool called “PEStudio”. PEStudio is a powerful tool that provides detailed insights into the characteristics and components of Portable Executable (PE) files. This includes executable files (.exe), dynamic link libraries (.dll), drivers (.sys), and other Windows binary files. While analyzing the malware with PEStudio, I was able to identify the exact date and time when the malware was compiled and written.

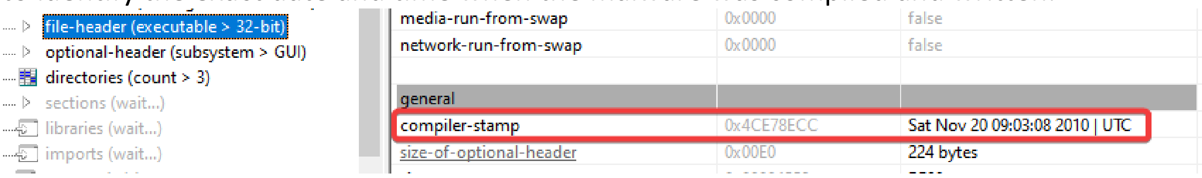


Figure 15: PEStudio: Compiler-Stamp

Moreover, PEStudio also identified that the malware has disguised itself as "Microsoft® Disk Defragmenter" with the filename "lhdfrgui.exe".

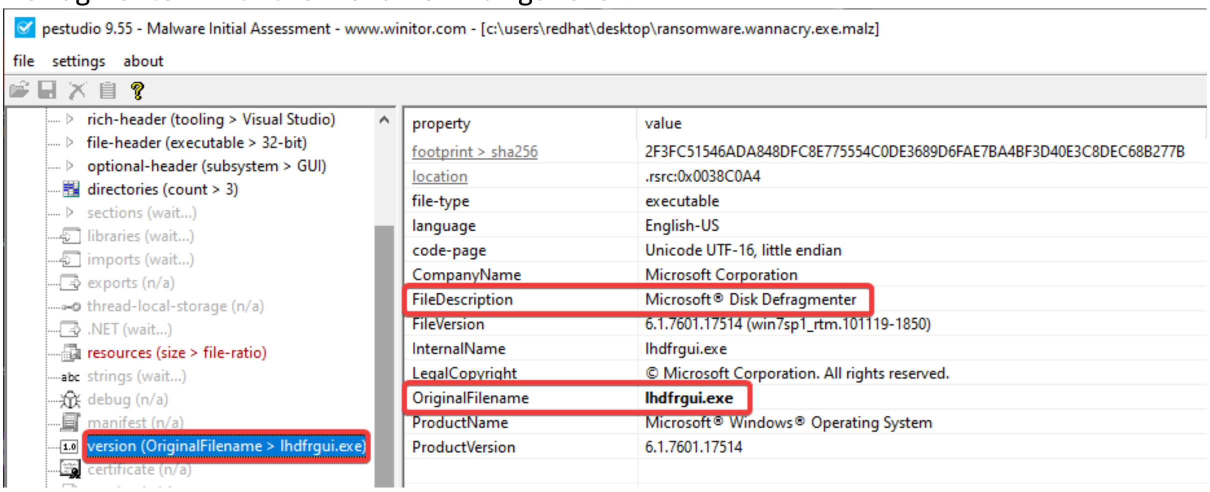


Figure 16: PEStudio: Original File Name

Also, it identifies that the malware has a resource named ‘1831’ embedded inside itself.

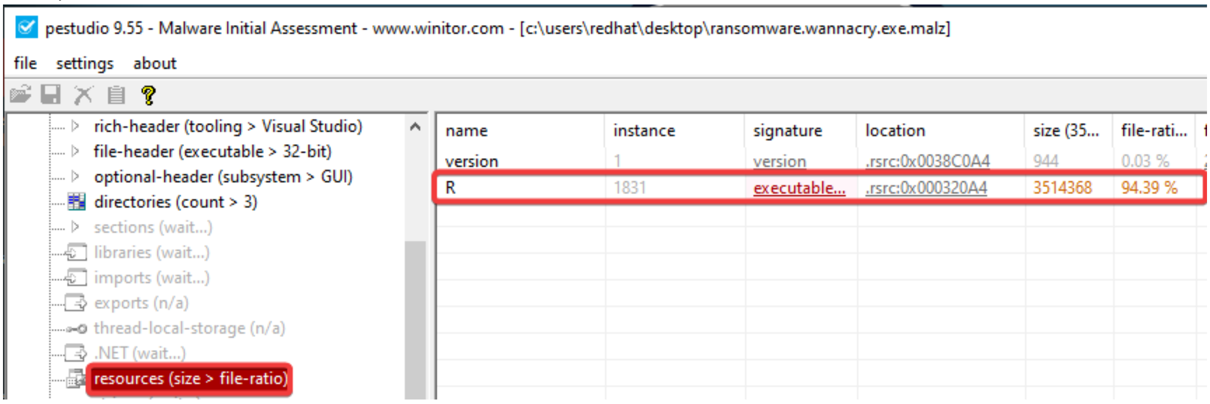
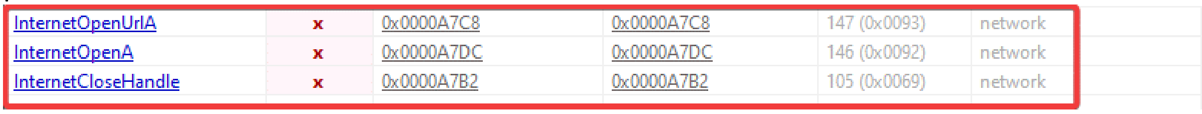


Figure 17: PEStudio: Resource Details

5.1 Imports

The imports are the specific functions that are used by the executables. The name of these functions gives us an idea about what the executable does when executed. While analyzing the malware in PEStudio, I analyze the following interesting import functions used by the malware.

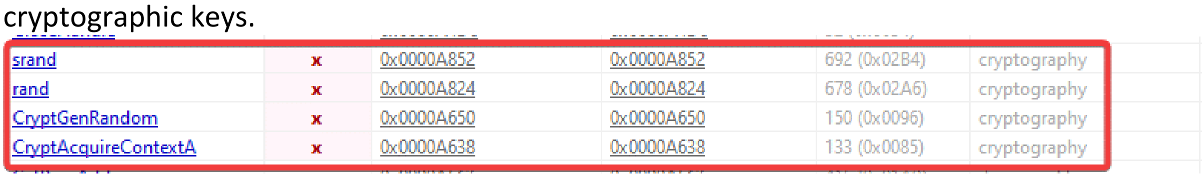
1. The following functions indicate potential network-related activities that the malware performs when executed.



InternetOpenUrlA	x	0x0000A7C8	0x0000A7C8	147 (0x0093)	network
InternetOpenA	x	0x0000A7DC	0x0000A7DC	146 (0x0092)	network
InternetCloseHandle	x	0x0000A7B2	0x0000A7B2	105 (0x0069)	network

Figure 18: PEStudio: Import Functions

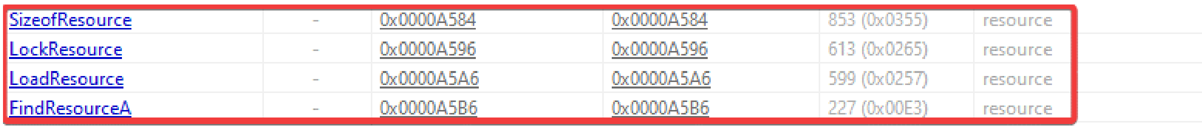
2. The following functions indicate that the malware uses cryptographic operations when executed, which could include encryption, decryption, or generating random cryptographic keys.



srand	x	0x0000A852	0x0000A852	692 (0x02B4)	cryptography
rand	x	0x0000A824	0x0000A824	678 (0x02A6)	cryptography
CryptGenRandom	x	0x0000A650	0x0000A650	150 (0x0096)	cryptography
CryptAcquireContextA	x	0x0000A638	0x0000A638	133 (0x0085)	cryptography

Figure 19: PEStudio: Import Functions

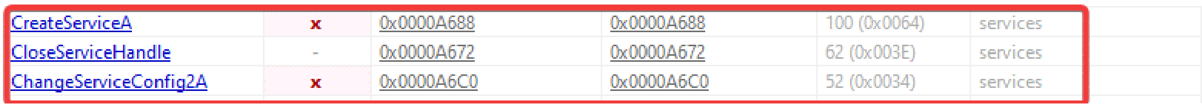
3. The following functions indicate that malware will load some of the resources embedded inside itself when executed.



SizeofResource	-	0x0000A584	0x0000A584	853 (0x0355)	resource
LockResource	-	0x0000A596	0x0000A596	613 (0x0265)	resource
LoadResource	-	0x0000A5A6	0x0000A5A6	599 (0x0257)	resource
FindResourceA	-	0x0000A5B6	0x0000A5B6	227 (0x00E3)	resource

Figure 20: PEStudio: Import Functions

4. The following functions indicate that malware will create some service when it is executed.



CreateServiceA	x	0x0000A688	0x0000A688	100 (0x0064)	services
CloseServiceHandle	-	0x0000A672	0x0000A672	62 (0x003E)	services
ChangeServiceConfig2A	x	0x0000A6C0	0x0000A6C0	52 (0x0034)	services

Figure 21: PEStudio: Import Functions

Basic Dynamic Analysis

Basic Dynamic analysis is a technique that involves running a malware sample in a controlled environment to gain insight into its behavior and interactions with the system. Basic Dynamic analysis, in particular, focuses on real-time observation of the malware's activities such as file operations, network communications, and registry modifications. By using this method, we can better understand the functionality and behavior of the malware on a system.
For the analysis of WannaCry, I will conduct two distinct analyses: first, while running INetSim (internet simulator), and second, without its execution. Additionally, I will utilize several tools such as Process Explorer, Procmon, Regshot, Wireshark, and TCPView to examine its behavior and impact on the system.

1. With INetSim running

When WannaCry is executed while INetSim is running, it attempts to establish a connection with the domain "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com". Since INetSim is active, the malware receives a positive HTTP response with code "200", indicating a successful connection, therefore malware stops its execution.

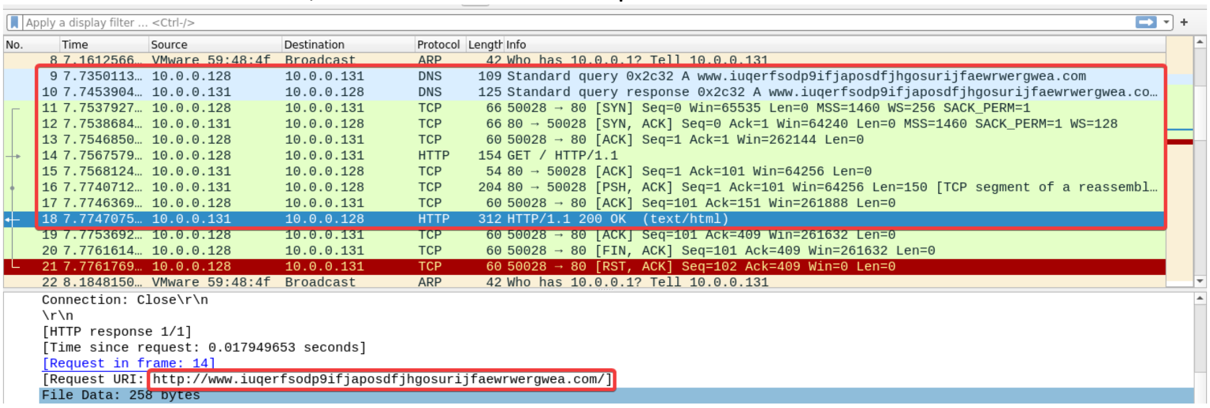


Figure 22: Wireshark: Network traffic on WannaCry Execution

2. Without running Inetsim

When WannaCry is executed without running INetSim, it attempts to establish a connection with the domain "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com". Since the INetSim is inactive, the requests to the domain are unreachable.

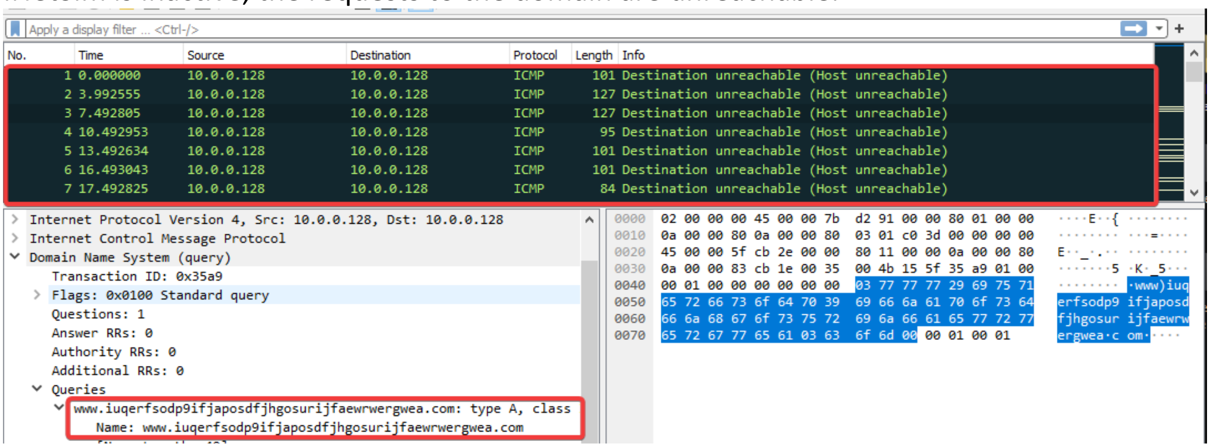


Figure 23: Wireshark: Network traffic on WannaCry Execution

As the connection to the URL fails, the malware creates a service named “mssecsvc2.0” with the DisplayName “Microsoft Security Centre (2.0) Service”.

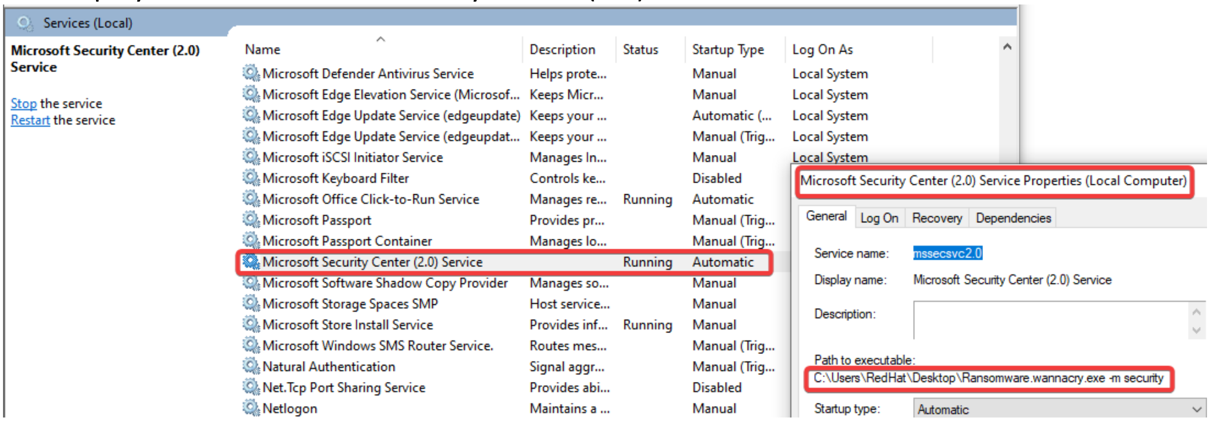


Figure 24: Service created by WannaCry

Then WannaCry uses this service to locate and infect other systems on the network using SMB port 445. We can analyze this by using TCPView.

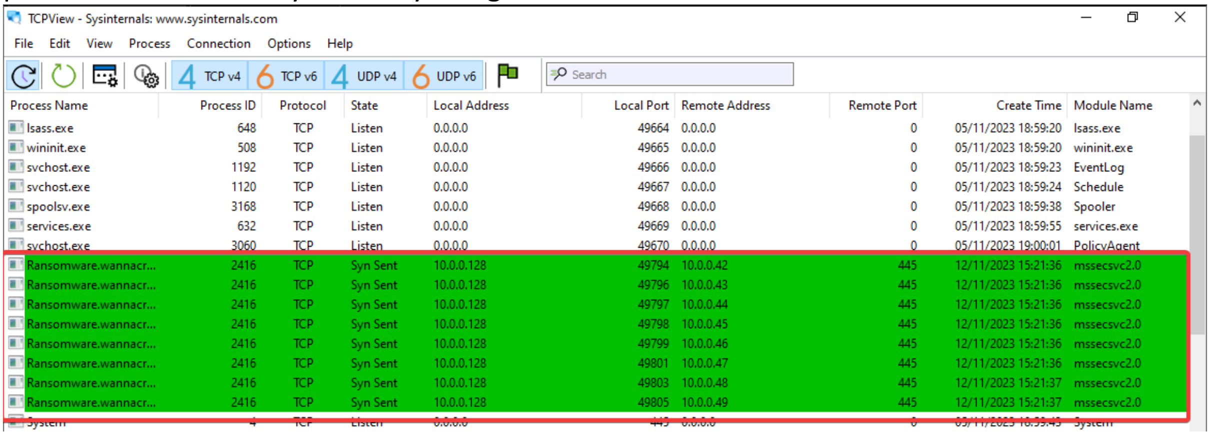


Figure 25: TCPView: WannaCry tries to propagate over the network using SMB Port 445

Along with that WannaCry will unpack the resources named "1831" that are packed inside it, drop it into the "C:\Windows" folder, and rename it to "tasksche.exe".

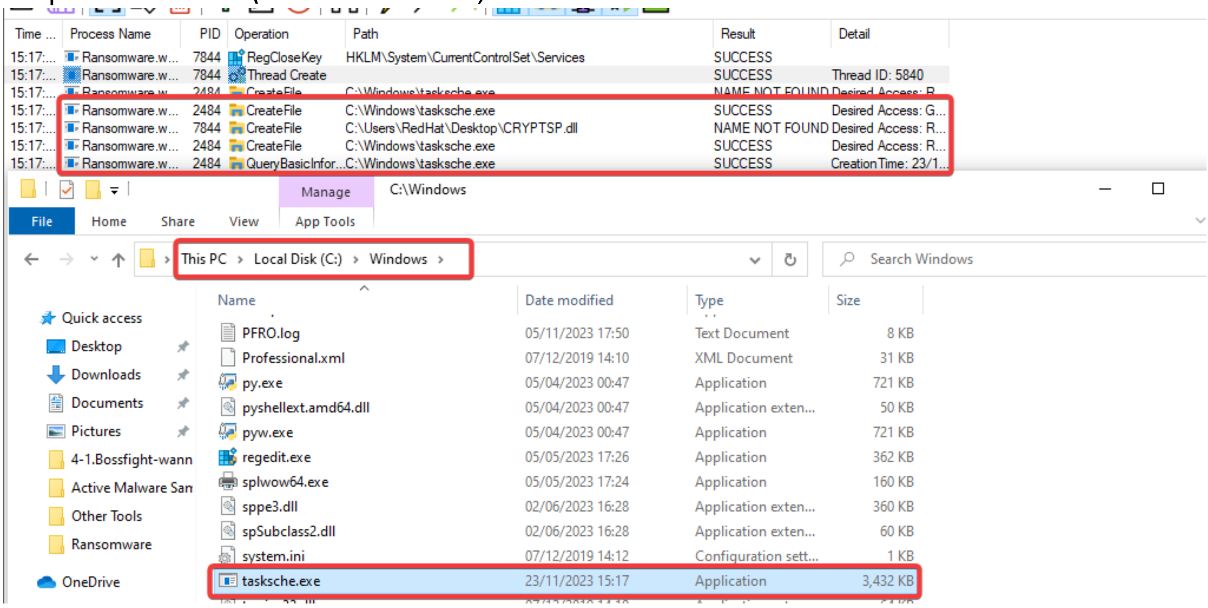


Figure 26: Procmon: File Dropped

After that, it executes the “tasksche.exe” with the “/i” argument.

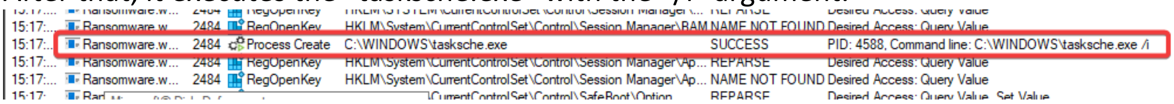


Figure 27: Procmon: WannaCry Loads “tasksche.exe” with “/i” argument

Upon execution “tasksche.exe” checks to see if the mutex “MsWinZonesCacheCounterMutexA” already exists, and will stop execution if the mutex is present.

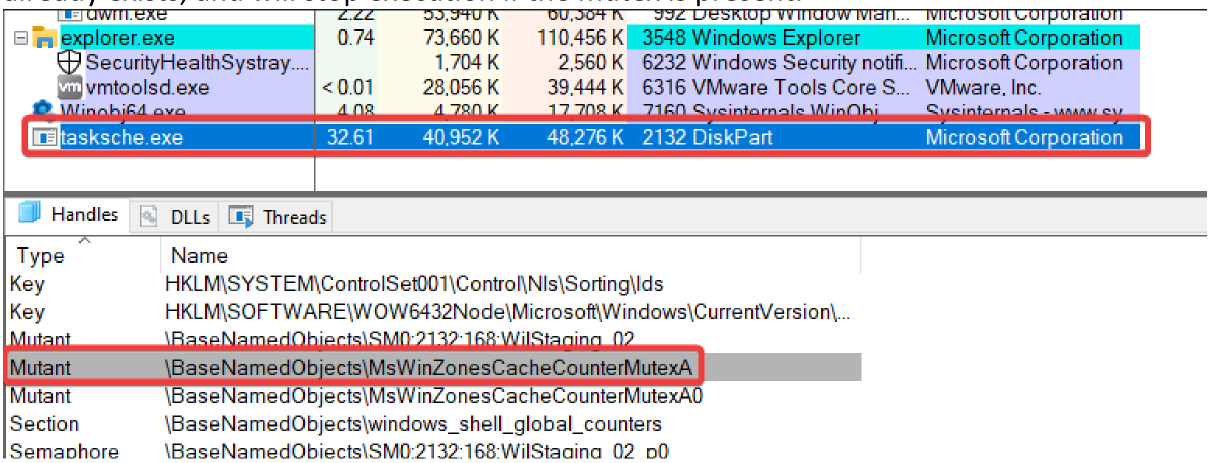


Figure 28: Process Explorer: "tasksche.exe" checks for mutex

If the mutex is not present, then “tasksche.exe” obtains the NetBIOS name of the local computer and then obfuscates it.

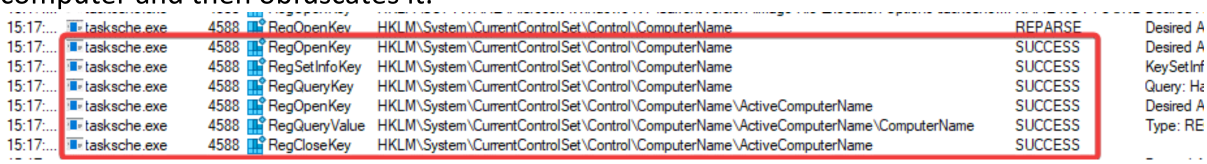


Figure 29: Procmon: "tasksche.exe" checking for local computer NetBIOS name

After that “tasksche.exe” creates a folder with the obfuscated name within “C:\ProgramsData” and saves its copy to the folder.

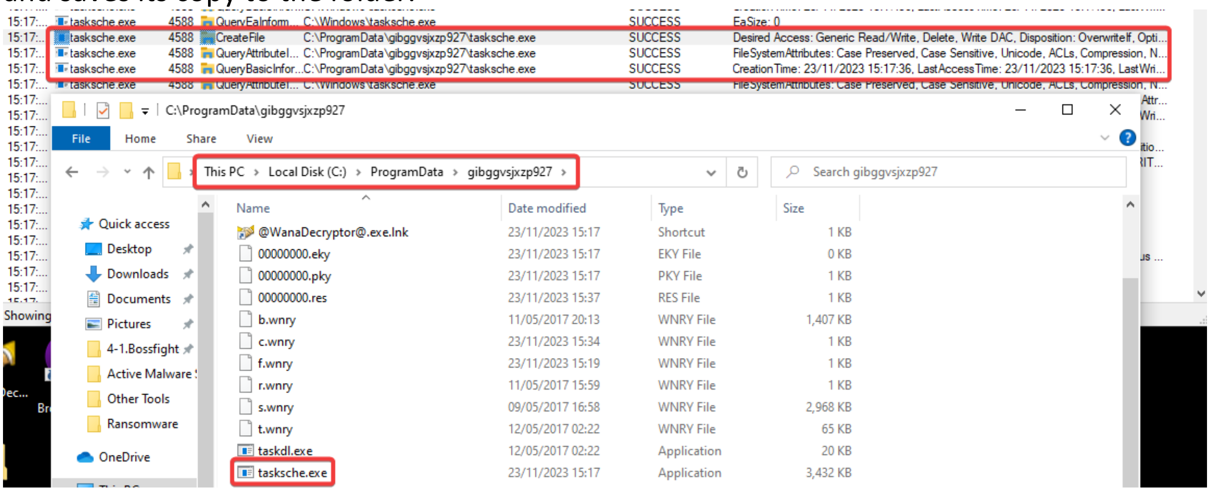


Figure 30: Procmon: "tasksche.exe" saves its copy to the obfuscated folder

Then it exits and loads itself from that folder.

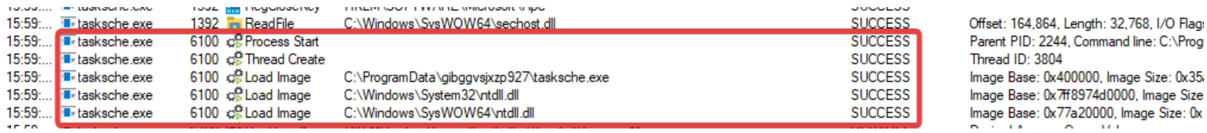
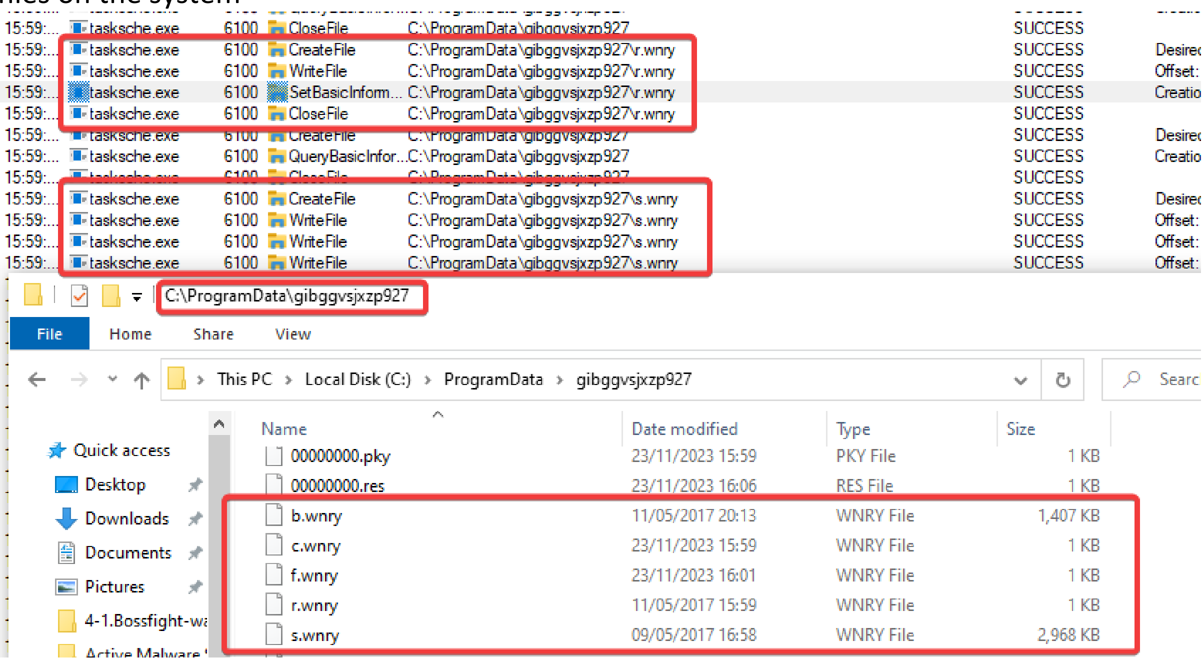


Figure 31: Procmon: "tasksche.exe" loads itself from the folder

Then it starts unpacking the malware files in the folder, which are then used to encrypt the files on the system



I analyzed the files dropped by "tasksche.exe" in the obfuscated folder at "C:\ProgramData\" using the hex editor "HxD". The unpacked files are:

1. b.wnry: It is the bitmap image that is the background screen after the infection.

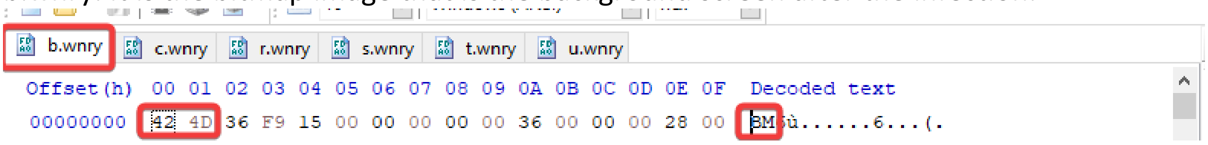


Figure 32: HxD: b.wnry analysis

2. c. wnry: It contains the address of the onion sites and a Zip file to install the Tor browser. The address of onion sites are as follows:

No.	Onion Sites Addresses
1	gx7ekbenv2riucmf.onion
2	57g7spgrzlojinas.onion
3	xxlvbrloxvriy2c5.onion
4	76jdd2ir2embyv47.onion
5	cwwnhwhlz52maqm7.onion

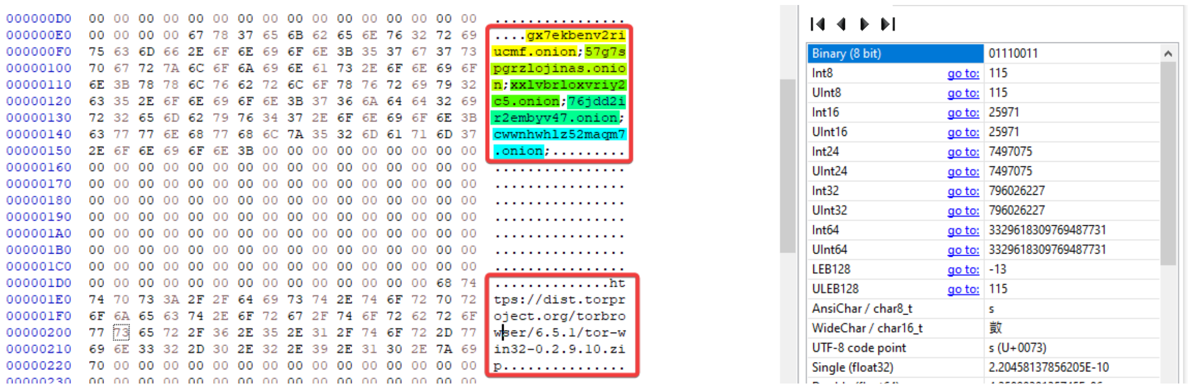


Figure 33: HxD: c.wnry analysis

3. r.wnry: It contains instructions for the user in English about the decryption process.

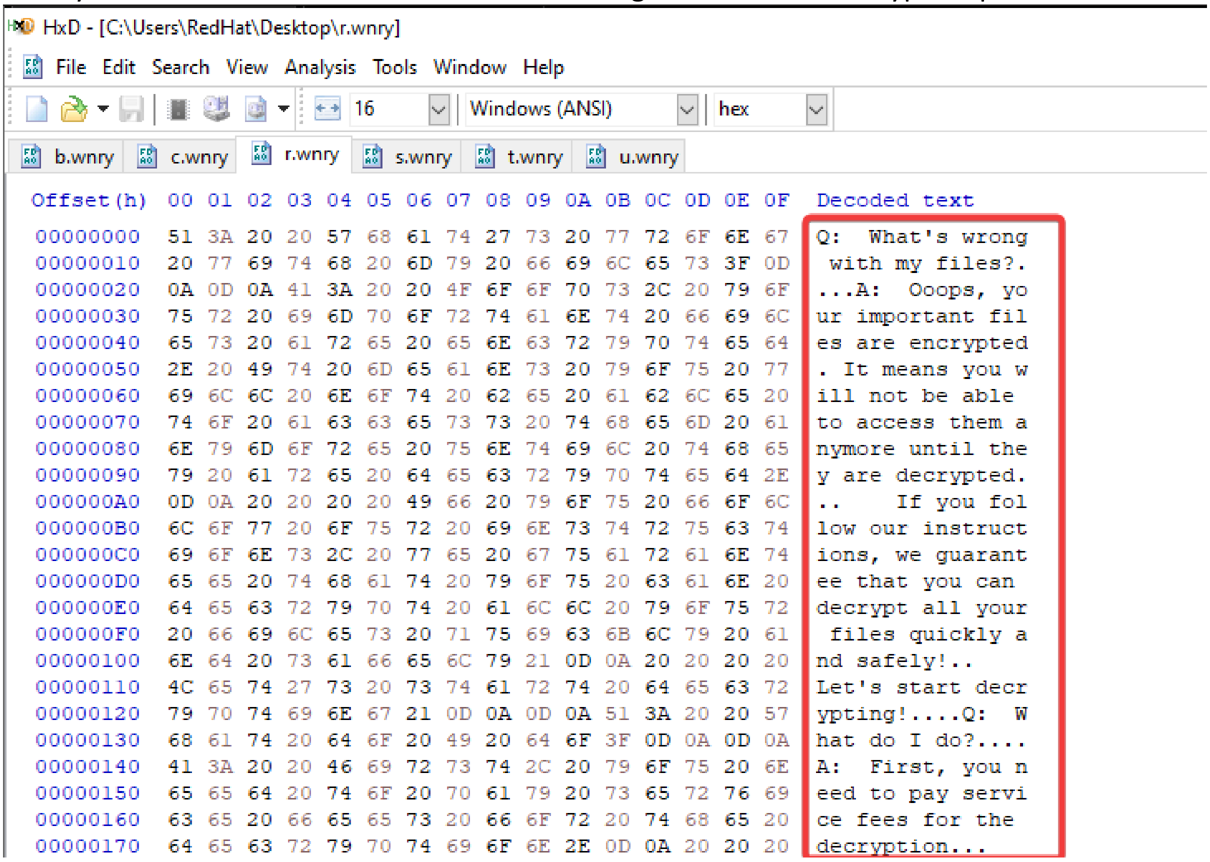


Figure 34: HxD: r.wnry analysis

4. s.wnry: It contains a zip file containing the legitimate Tor software executable.

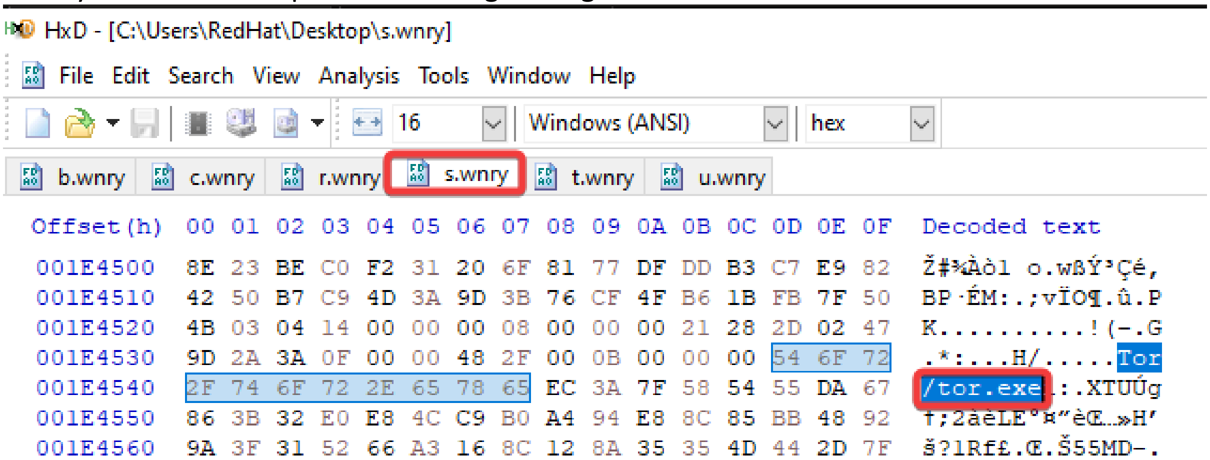


Figure 35: HxD: s.wnry analysis

5. t.wnry: This file is encrypted using the WANACRY! encryption format. The file header of the file is "WANACRY!".

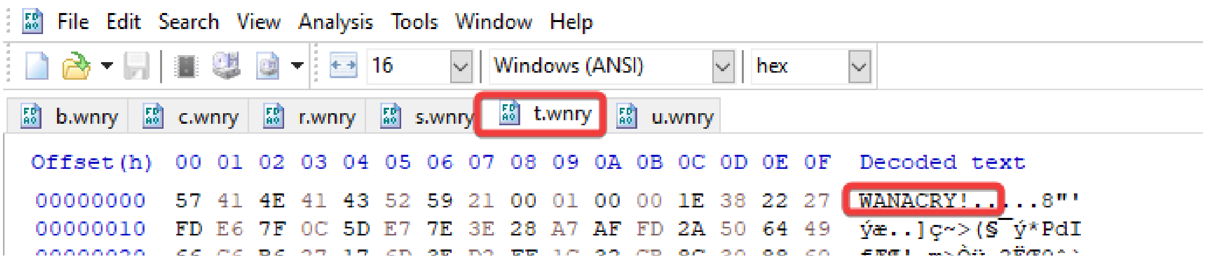


Figure 36: HxD: t.wnry analysis

6. u.wnry: It is the @WanaDecryptor@.exe file used to decrypt files.

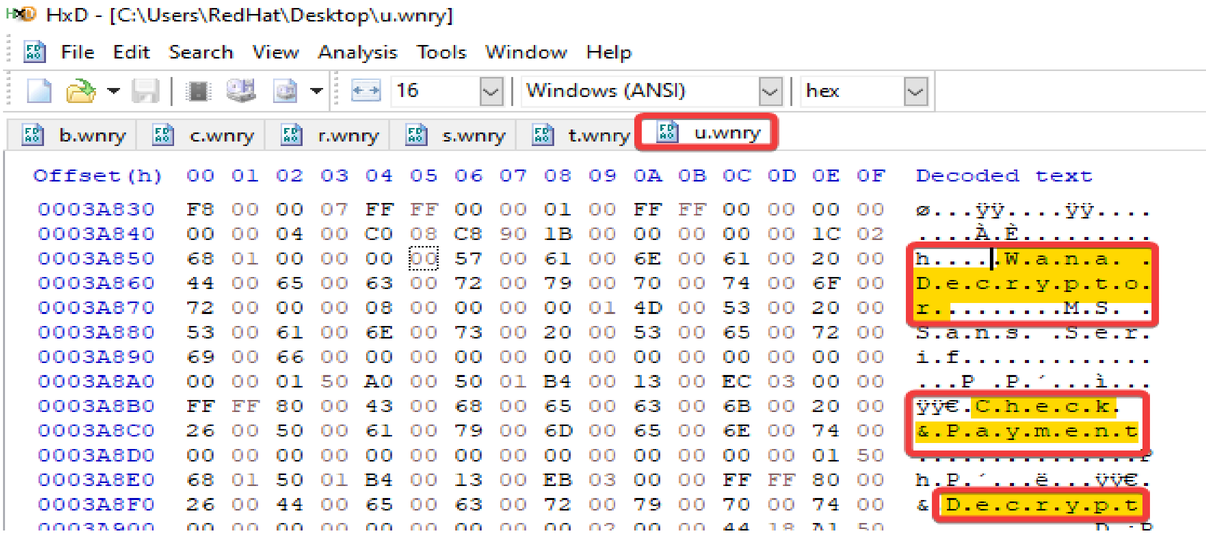


Figure 37: HxD: u.wnry analysis

7. f.wnry: This file is created during the execution. It contains a list of randomly chosen encrypted files by the malware that it will use to demonstrate the decryption process to the victims.

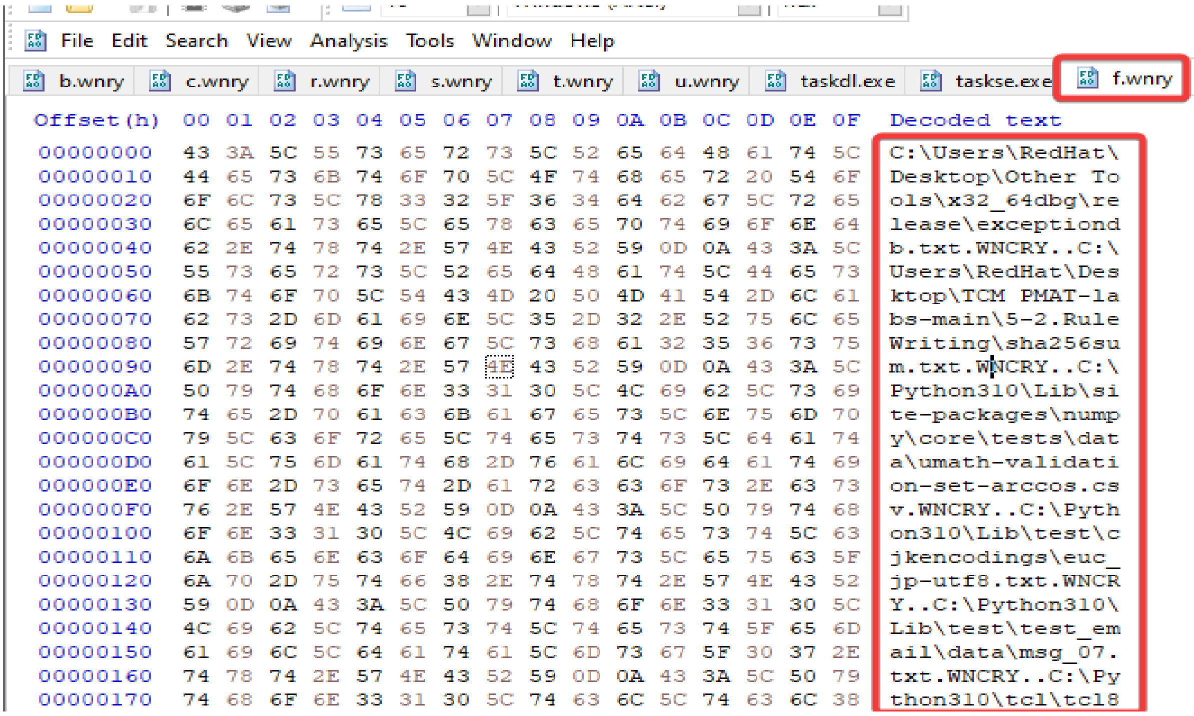


Figure 38: HxD: u.wnry analysis

8. taskdl.exe: This is used for the cleanup of WNCRYT temporary files.

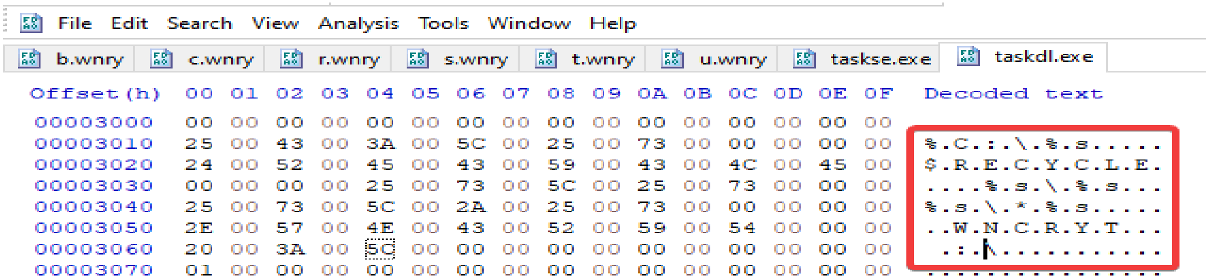


Figure 39: HxD: "taskdl.exe" analysis

9. taskse.exe: This is used to open “@WanaDecryptor@.exe” in RDP sessions.

Once the malware files are unpacked in the designated folder, the program initiates the "attrib.exe" with the "attrib +h" command to hide the directory. Additionally, it runs "icacls.exe" with the "icacls ./grant Everyone:F /T /C /Q" command to grant full access to all files in the folder to all users.

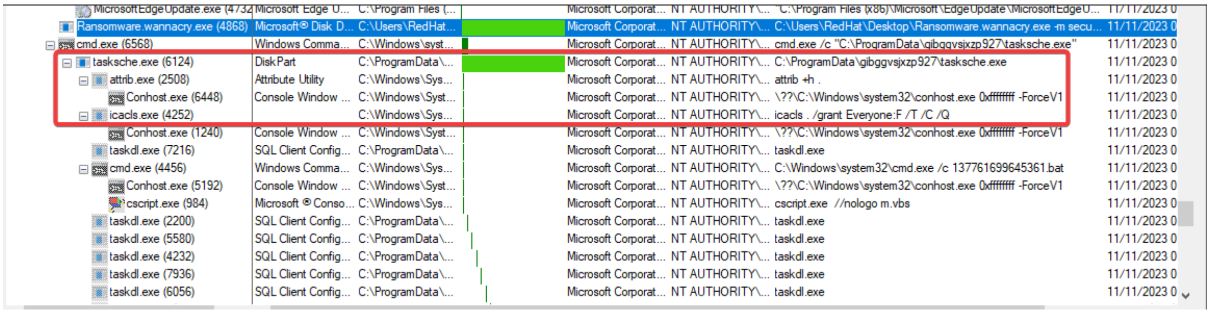


Figure 40: Procmon: Process Tree

After mawlare unpacks the files in the designated folder, it starts encrypting the files on the system.

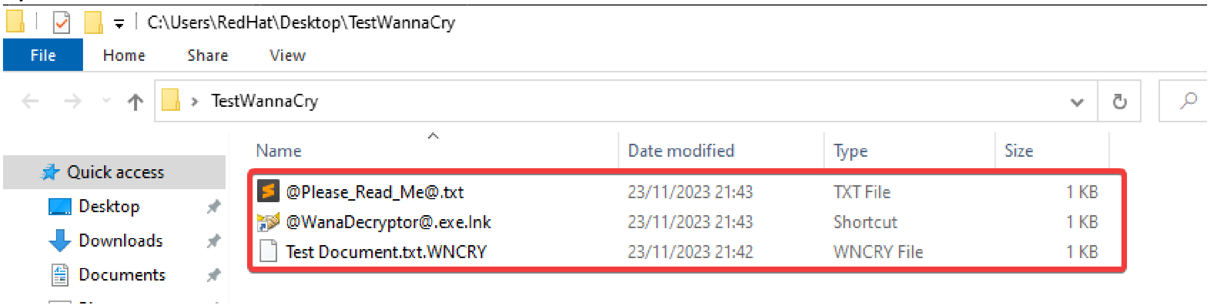


Figure 41: Malware encrypting the files

After completing the encryption routine it executes "@WanaDecryptor@.exe" and also replaces the wallpaper with "@WanaDecryptor@.bitmap".

Figure 42: "tasksche.exe" executes WannaDecryptor after encryption is completed

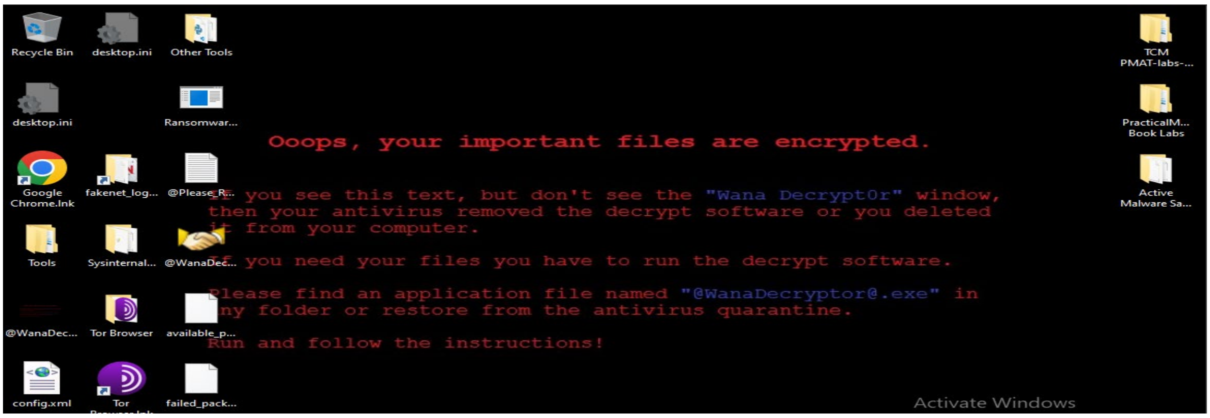


Figure 43: "tasksche.exe" changes the background

Now to analyze the registry changes that the malware made during the execution, I use the tool called “Regshot”. Upon analyzing the results, I found that the malware had added the following keys and values to the registry:

1. During execution malware has created a registry value "gibggvsjxzp927" at "HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Run" that points to a file in "C:\ProgramData".

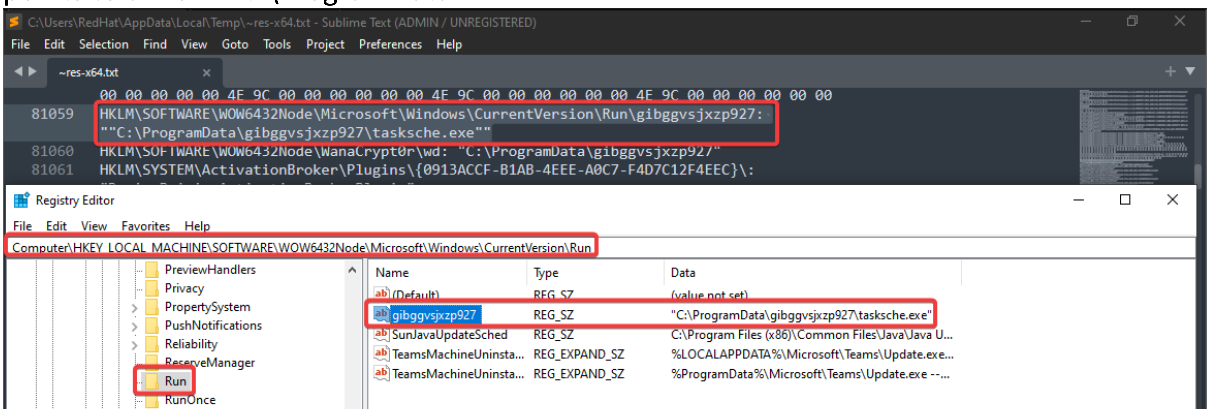


Figure 44: Regshot: Registry Value added by malware

2. Malware has created a registry key "WanaCrypt0r" with the value "wd" at "HKLM\Software\WOW6432Node" that points to the folder created by the binary at "C:\ProgramData".

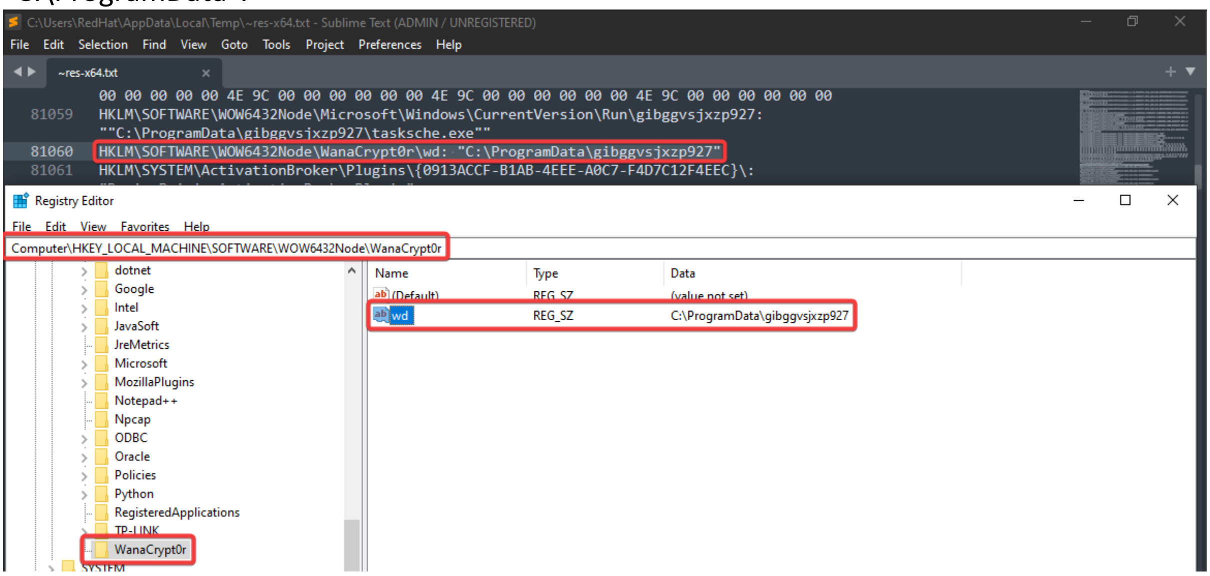


Figure 45: Regshot: Registry Key and Value added by malware

3. Malware has created a service named "gibggvsjxzp927" and configured it to execute a command that involves the execution of "tasksche.exe" at startup from the folder created by the binary in "C:\ProgramData". Moreover, binary has also created a registry key for the service as well.

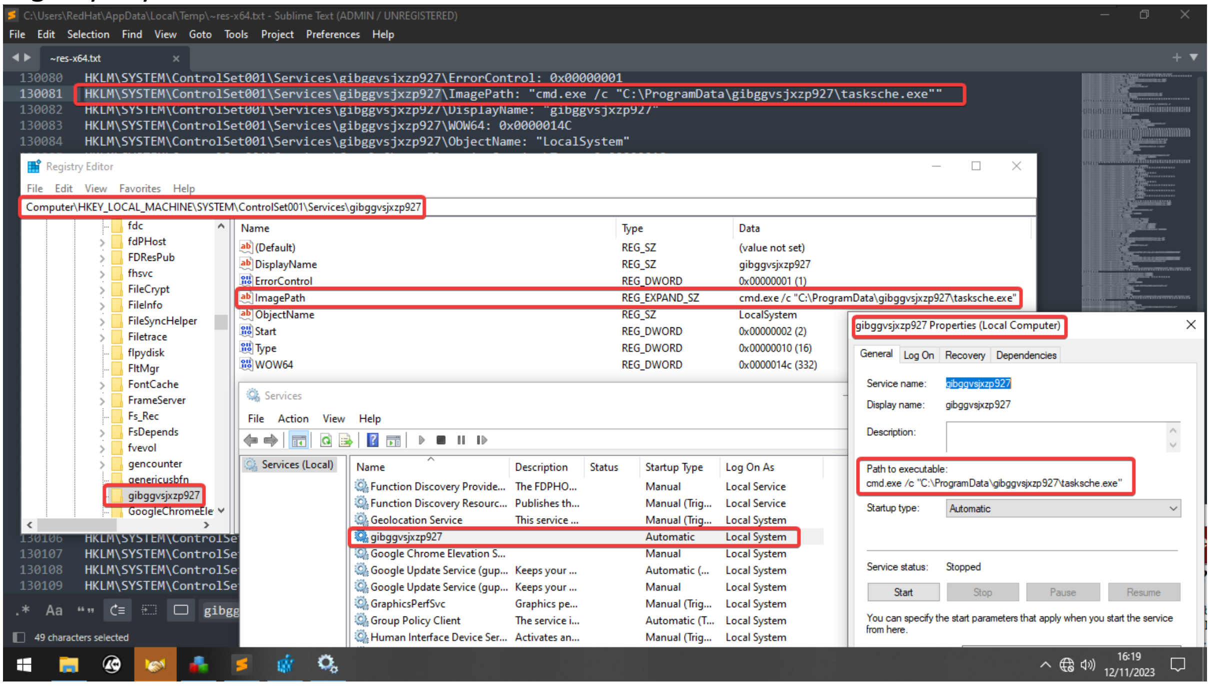


Figure 46: Regshot: Registry Value added and a Service created by malware

Advance Static Analysis

Advanced static analysis involves an in-depth examination of malware code and structures without executing them. It is a process that involves a detailed inspection of the code to understand how the malware operates when executed. To perform advanced static analysis on malware I utilize a disassembler tool called "cutter" to conduct an in-depth static analysis on malware and the following is the analysis of WannaCry Ransomware.

1. This is the main function of the malware where it tries to connect to the URL.
    a. If the malware is able to communicate with the domain it will return some non-zero value to the EAX register, which is then moved to the EDI register. After that the test instruction will perform Bitwise AND operation of the EDI register and will set the ZF based on the result which in this case is ZF = 0, this means that the values of EDI are not equal hence it sets the ZF=0. Now JNE (Jump If not equal) will check the ZF and as the ZF=0 it will take the jump and as a result, the malware will not execute its main payload.
    b. If the malware is unable to communicate with the domain it will return some zero value to the EAX register, which is then moved to the EDI register. After that, the test instruction will perform Bitwise AND operation of the EDI register and will set the ZF based on the result which in this case is ZF = 1, this means that the values of EDI are equal hence it set the ZF=1. Now JNE (Jump If not equal) will check the ZF and as the ZF=1 it will not take the jump and as a result, the binary will execute its main payload.

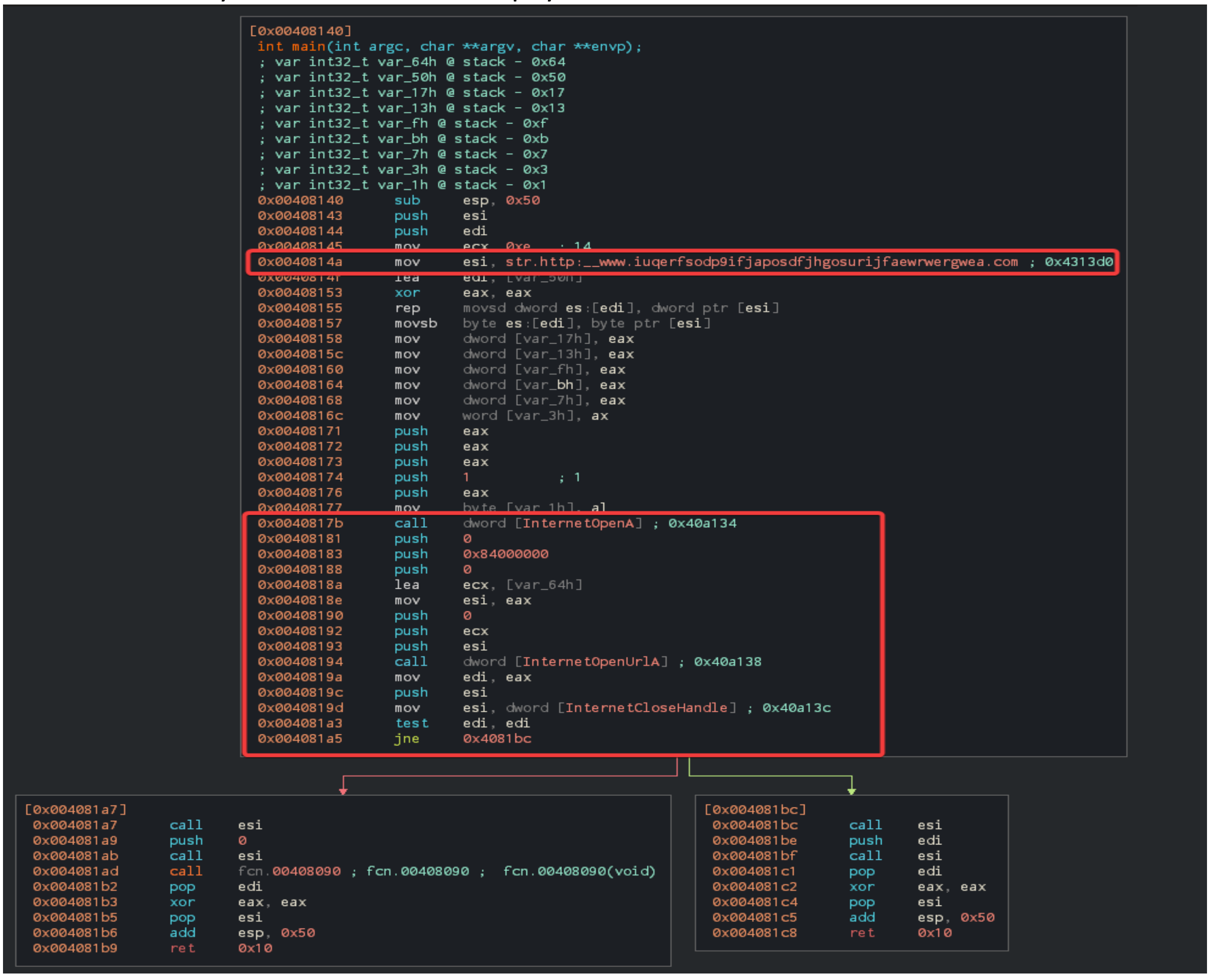


Figure 47: Cutter: Main Function

2. The following code creates a service "mssecsvc2.0" with the display name of "Microsoft Security Center 2.0 Service" to disguise itself as a Microsoft Service.

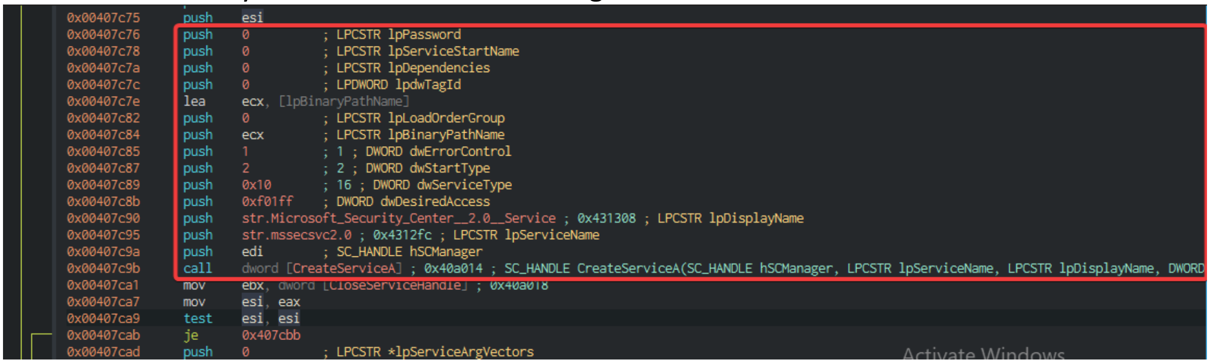


Figure 48: Cutter: Code for "mssecsvc2.0" service creation

3. The following code unpacks the embedded resource "1831", renames it to "tasksche.exe" and drop it in the C:\Windows directory, and loads it.

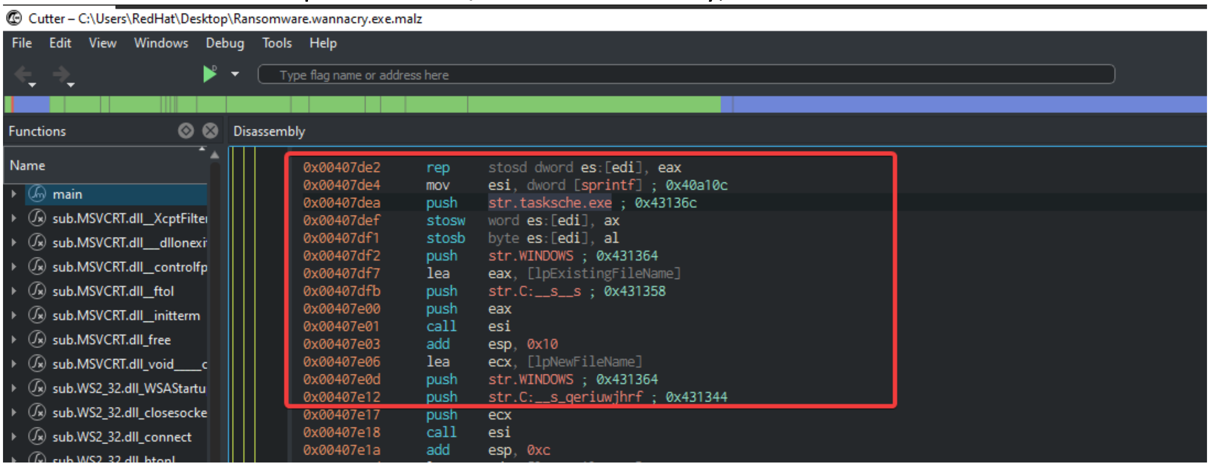


Figure 49: Cutter: Code to unpack the embedded resource "1831"

4. The following code checks for the mutex, if the mutex is present then tasksche.exe exits because the presence of this mutex indicates that binary is already running on the system.

Figure 50: Cutter: Code to check for Mutex

5. The following code gets the NetBIOS name of the local computer and obfuscates it

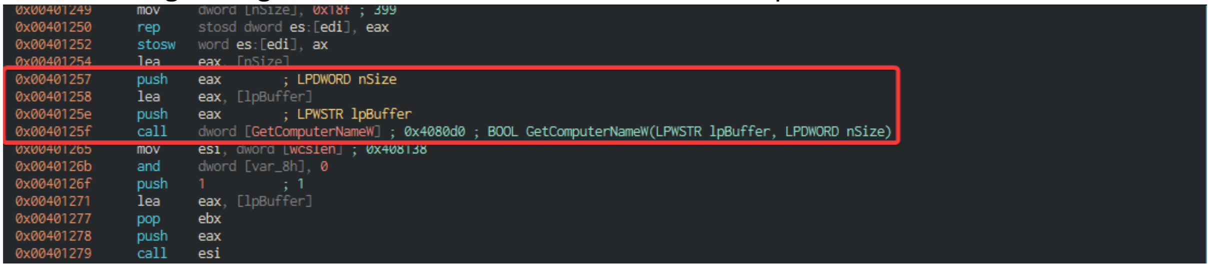


Figure 51: Cutter: Code to get NetBIOS name of the computer

6. The following code creates a folder in "C:\ProgramData" with the obfuscated computer name

Figure 52: Cutter: Code to create an obfuscated folder in "C:\ProgramData"

7. The following code extracts the content of the embedded zip file to the directory created by "tasksche.exe" at "C:\ProgramData" using the password "WNcry_2ol7".

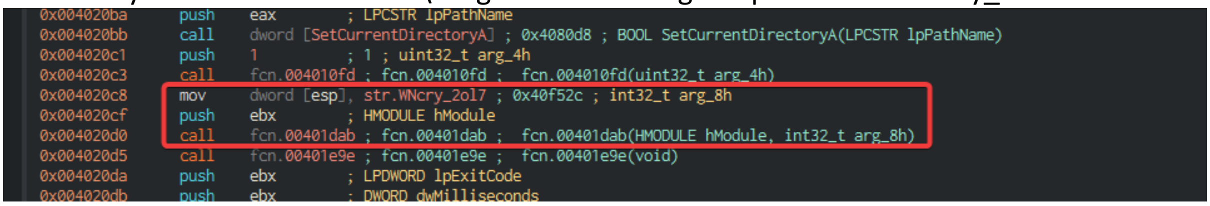


Figure 53: Cutter: "tasksche.exe" unpack resources at the obfuscated folder

8. The following code makes the directory hidden that it creates at "C:\ProgramData" and also grants full access to all files in the directory to all users.

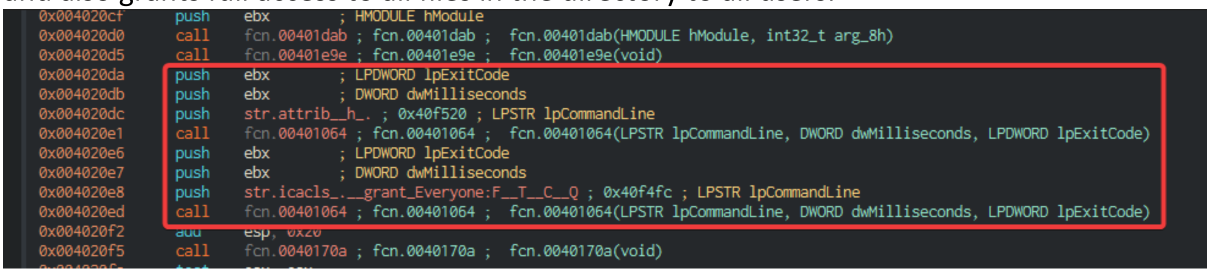


Figure 54: Cutter: Code to make the obfuscated folder hidden and change the access of the folder to everyone

9. The following code loads the Bitcoin addresses.

```
fcn.00401e9e();
; var int32_t var_31ch @ stack - 0x31c
; var char *dest @ stack - 0x26a
; var const char *var_10h @ stack - 0x10
; var const char *var_ch @ stack - 0xc
; var const char *var_8h @ stack - 0x8
0x00401e9e      push    ebp
0x00401e9f      mov     ebp, esp
0x00401ea1      sub     esp, 0x318
0x00401ea7      lea     eax, [var_31ch]
0x00401ead      push    1               ; 1 ; int32_t arg_4h
0x00401eaf      push    eax             ; int32_t arg_8h
0x00401eb0      mov     dword [var_10h], 0x40f488 ; str.13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
0x00401eb7      mov     dword [var_ch], str.12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw ; 0x40f464
0x00401ebe      mov     dword [var_8h], 0x40f440 ; str.115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
0x00401ec5      call    fcn.00401000 ; fcn.00401000
```

Figure 55: Cutter: code to load the hardcoded Bitcoin addresses

10. The following code creates the registry Key "WanaCrypt0r".

```
0x0040113e      stosb   byte es:[edi], al
0x0040113f      lea     eax, [s1]
0x00401145      push    str.WanaCrypt0r ; 0x40e034 ; wchar_t *s2
0x0040114a      push    eax             ; wchar_t *s1
0x0040114b      call    dword [wcscat] ; 0x408134 ; wchar_t *wcscat(wchar_t *s1, wchar_t *s2)
0x00401151      and     dword [var_ch], 0
0x00401155      pop     ecx
0x00401156      pop     ecx
0x00401157      mov     edi, data.0040e030 ; 0x40e030
0x0040115c      lea     eax, [hKey]
0x0040115f      xor     esi, esi
0x00401161      cmp     dword [var_ch], esi
0x00401164      push    eax
0x00401165      lea     eax, [s1]
0x0040116b      push    eax
0x0040116c      jne     0x401175
0x0040116e      push    0x80000002
0x00401173      jmp     0x40117a
0x00401175      push    0x80000001
0x0040117a      call    dword [RegCreateKeyW] ; 0x408014 ; LSTATUS RegCreateKeyW(HKEY hKey, LPCWSTR lpSubKey, PHKEY phkResult)
0x00401180      cmp     dword [hKey], esi
0x00401183      je      0x40120d
0x00401189      cmp     dword [arg_4h], esi
0x0040118c      je      0x4011cc
0x0040118e      lea     eax, [lpPathName]
0x00401194      push    eax             ; LPSTR lpBuffer
0x00401195      push    0x207           ; 519 ; DWORD nBufferLength
0x0040119a      call    dword [GetCurrentDirectoryA] ; 0x4080d4 ; DWORD GetCurrentDirectoryA(DWORD nBufferLength, LPSTR lpBuffer)
0x004011a0      lea     eax, [lpPathName]
0x004011a6      push    eax             ; const char *s
0x004011a7      call    sub.MSVCRT.dll_strlen ; sub.MSVCRT.dll_strlen ; size_t strlen(const char *s)
```

Figure 56: Cutter: Code to create "WanaCrypt0r" registry key

Advance Dynamic Analysis

Advanced dynamic analysis is a comprehensive process that involves examining the behaviour of malware while running in a controlled environment. It allows analysts to gain an in-depth understanding of the malware's activities and how it interacts with the system. This insight helps them understand how the malware operates. For the analysis of WannaCry, I will conduct this analysis by running INetSim and making the malware to execute even if the connection to the URL is successful. Additionally, I will utilize a debugger tool called "x32dbg" to conduct an in-depth dynamic analysis of malware.

1. As analyzed in the Advance Static Analysis, the main function of the malware contains a hardcoded URL, and the malware will check for the connection to the URL in order to proceed further therefore, we need to put a breakpoint on the URL.

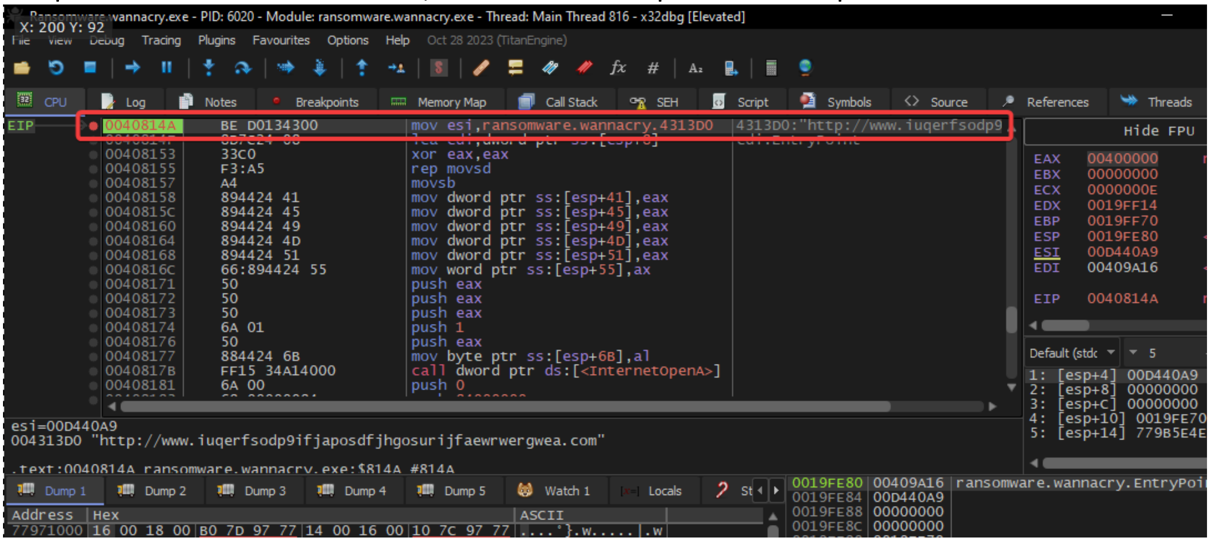


Figure 57: x32dbg: Main Function

2. Since the connection to the URL is successful, therefore, the "EDI" register is not zero. As a result, the instruction "test edi, edi" returns 1, indicating that the value of edi is not equal. Consequently, the ZF is set to 0. This means that if we execute the jump statement, the program will take the jump, and the instruction pointer will move to the “004081BC” address, and the malware will not execute its main payload.

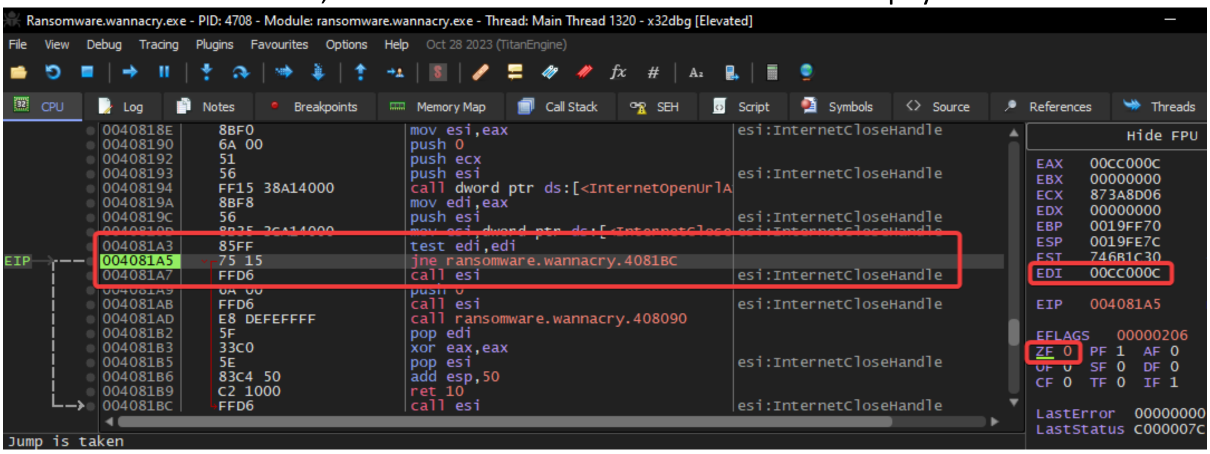


Figure 58: x32dbg: Evaluation of test statement

3. If we want to stop the program from making a jump, we can simply change the value of ZF to 1 by clicking on it. By doing so, we can prevent the malware from taking the jump to the "004081BC" address, and thereby allow it to carry out its main payload. This means that if we execute the jump statement, the program will not take the jump, and the instruction pointer will move to the next statement "call esi".

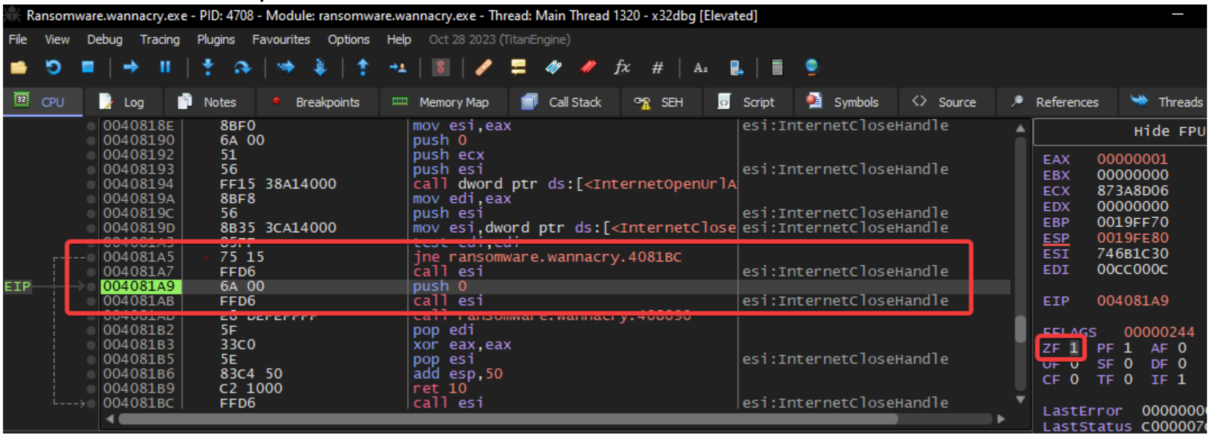


Figure 59: x32dbg: Change zero flag to prevent jump

Indicators of Compromise (IOCs)

Upon execution, the malware performs various actions such as reaching out to the domain, dropping files on the system, creating services, and registry keys and values as observed during the analysis, therefore below is a comprehensive list of Indicators of Compromise (IOCs).

1. File Hashes

The following are the file hashes

File Name	Hash Algorithm	Hashes
Ransomware. wannaCry.exe	MD5	db349b97c37d22f5ea1d1841e3c89eb4
	SHA1	e889544aff85ffaf8b0d0da705105dee7c97fe26
	SHA256	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
tasksche.exe	MD5	84c82835a5d21bbcf75a61706d8ab549
	SHA1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
	SHA256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

2. Callback Domain and Onion Links

The following is the hardcoded

No.	Domain	Description
1	hxxp://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com	Hardcoded URL
2	gx7ekbenv2riucmf.onion	Onion Links
3	57g7spgrzlojinas.onion	
4	xxlvbrloxvriy2c5.onion	
5	76jdd2ir2embyv47.onion	
6	cwwnhwhlz52maqm7.onion	

3. Commands

The following is the are the commands that malware executes

No.	Commands
1	icacls . /grant Everyone:F /T /C /Q
2	attrib +h .
3	cmd.exe /c "%s"
4	C:\%s\qeriuwjhrf

4. IP Addresses

The following is the are the callback IP Address of the malware

No.	IP Addresses
1	172.16.99.5
2	192.168.56.20

5. Files Dropped

The following are the files dropped by malware on execution

File Name	Hash Algorithm	Hashes
tasksche.exe	MD5	84c82835a5d21bbcf75a61706d8ab549
	SHA1	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
	SHA256	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
b.wnry	MD5	c17170262312f3be7027bc2ca825bf0c
	SHA1	f19eceda82973239a1fdc5826bce7691e5dcb4fb
	SHA256	d5e0e8694ddc0548d8e6b87c83d50f4ab85c1debadb106d6a6a794c3e746f4fa
c.wnry	MD5	ae08f79a0d800b82fcbe1b43cdbdbefc
	SHA1	f6b08523b1a836e2112875398ffefffde98ad3ca
	SHA256	055c7760512c98c8d51e4427227fe2a7ea3b34ee63178fe78631fa8aa6d15622
r.wnry	MD5	3e0020fc529b1c2a061016dd2469ba96
	SHA1	c3a91c22b63f6fe709e7c29cafb29a2ee83e6ade
	SHA256	402751fa49e0cb68fe052cb3db87b05e71c1d950984d339940cf6b29409f2a7c
s.wnry	MD5	ad4c9de7c8c40813f200ba1c2fa33083
	SHA1	d1af27518d455d432b62d73c6a1497d032f6120e
	SHA256	e18fdd912dfe5b45776e68d578c3af3547886cf1353d7086c8bee037436dff4b
t.wnry	MD5	5dcaac857e695a65f5c3ef1441a73a8f
	SHA1	7b10aaeee05e7a1efb43d9f837e9356ad55c07dd
	SHA256	97ebce49b14c46bebc9ec2448d00e1e397123b256e2be9eba5140688e7bc0ae6
u.wnry	MD5	7bf2b57f2a205768755c07f238fb32cc
	SHA1	45356a9dd616ed7161a3b9192e2f318d0ab5ad10
	SHA256	b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25
f.wnry	MD5	30bb42c9f63bec26b405aa3f951da18f
	SHA1	d89756b6fa6d728734ad75392a692d283e94182c
	SHA256	d89756b6fa6d728734ad75392a692d283e94182c
taskdl.exe	MD5	4fef5e34143e646dbf9907c4374276f5
	SHA1	47a9ad4125b6bd7c55e4e7da251e23f089407b8f
	SHA256	4a468603fdcb7a2eb5770705898cf9ef37aade532a7964642ecd705a74794b79
taskse.exe	MD5	8495400f199ac77853c53b5a3f278f3e
	SHA1	be5d6279874da315e3080b06083757aad9b32c23
	SHA256	2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d
@WanaDecryptor@.exe	MD5	7bf2b57f2a205768755c07f238fb32cc
	SHA1	45356a9dd616ed7161a3b9192e2f318d0ab5ad10
	SHA256	b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25

6. Services Created

The following services are created by malware on execution

No.	Services Created
1	mssecsvc2.0
2	gibggvsjxzp927 (obfuscated directory name created by tasksche.exe at C:\ProgramData)

7. Registry Keys Added

The following registry keys and values are added by malware on execution

No.	Registry Keys
1	HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\gibggvsjxzp927
2	HKLM\Software\WOW6432Node\WanaCrypt0r
3	HKLM\SYSTEM\ControlSet001\Services\gibggvsjxzp927

8. File Strings

The following are the file strings

No.	Type	File Strings
1	String	1831
2	String	inflate 1.1.3 Copyright 1995-1998 Mark Adler
3	String	unzip 0.15 Copyright 1998 Gilles Vollant
4	String	WNcry@2ol7
5	String	WanaCrypt0r
6	Mutex Name	Global\MsWinZonesCacheCounterMutexA
7	Bitcoin Address	115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
8	Bitcoin Address	12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
9	Bitcoin Address	13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94

YARA Rule

The following is the Yara rule created for the detection of the malware

```
rule Ransomware_WannaCry{
    meta:
        description = "YARA Rule for WannaCry Ransomware Detection"
        author = "Muhammad Osama Khalid"
        last_updated = "14 November 2023"
        sha256 =
"24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c"
    strings:
        $PE_magic_byte = "MZ"
        $URL = "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
        $cmd1 = "attrib +h ." fullword ascii
        $cmd2 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii
        $cmd3 = "cmd.exe /c \"%s\"" fullword ascii
        $cmd4 = "C:\\%s\\qeriuwjhrf" fullword ascii
        $bitcoin_address1 = "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
        $bitcoin_address2 = "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
        $bitcoin_address3 = "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
        $string1 = "WNcry@2ol7" fullword ascii
        $string2 = "inflate 1.1.3 Copyright 1995-1998 Mark Adler" fullword ascii
        $string3 = "unzip 0.15 Copyright 1998 Gilles Vollant" fullword ascii
        $string4 = "Global\\MsWinZonesCacheCounterMutexA" fullword ascii
        $payload = "tasksche.exe" fullword ascii
    condition:
        $PE_magic_byte at 0 and ( $URL or 1 of ($cmd*) or 1 of($bitcoin_address*)
or 1 of ($string*) or $payload)
}
```