Practical Malware Analysis

WannaCry Ransomware

August 2023 | Xpinux | v1.0

Table of Contents

Executive Summary

MD5 hash	db349b97c37d22f5ea1d1841e3c89eb4
SHA1 Hash	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA256 hash	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
Format	PE
IOC	C:\%s\qeriuwjhrf
IOC	WANACRY!
IOC	http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com

Wannacry.Ransomware, a highly sophisticated and notorious malware, was analyzed, revealing a two-stage structure that underscored its devious capabilities. The first stage boasted a cunning killswitch mechanism, designed to avoid detonation if a specific URL was accessible. In this manner, the malware ensured self-preservation and stealthy behavior.

However, when the URL proved unattainable, the ransomware swiftly transitioned to its second stage - a perilous propagation attempt within the network. This propagation stage raised the stakes significantly, intensifying the threat landscape for organizations.

During analysis, we discovered the ransomware's reliance on tasksche.exe, skillfully employed to unpack files into a mysterious directory nestled within ProgramData. This intelligent maneuver enabled the malware to establish persistence, complicating detection and removal efforts.

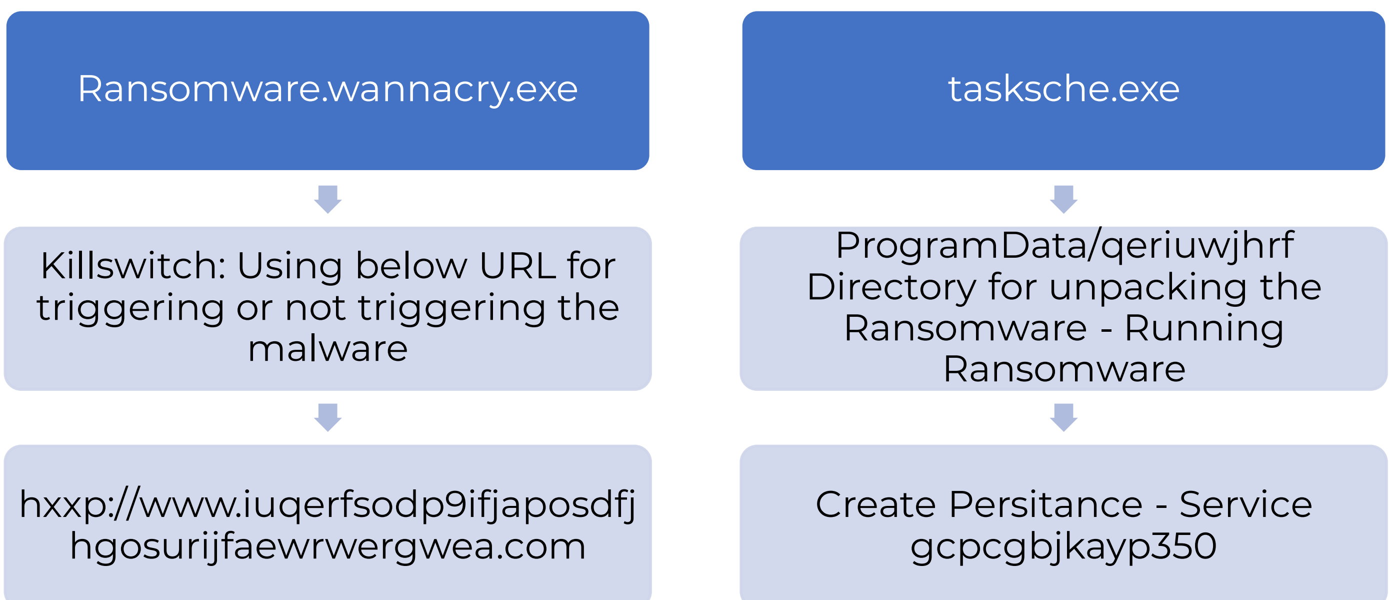
Once the ransomware was in full motion, it executed a relentless encryption process, rendering critical data inaccessible to its victims. To exacerbate matters, it brazenly presented a disconcerting popup, demanding a ransom for the coveted decryption key.

In response to this ominous threat and its potential impact on businesses, we emphasize the urgency of enhancing cybersecurity defenses and fortifying employee awareness. Proactive measures and continuous monitoring are paramount to safeguarding against Wannacry.Ransomware and similar malicious adversaries. By adopting a robust cybersecurity posture, organizations can better protect their digital assets and ensure uninterrupted operations amidst the evolving cyber landscape.

YARA signature rule is attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal with a **Score of 68/71** Detections.

High-Level Technical Summary

Wannacry.Ransomware is a multi-stage malware comprising a killswitch mechanism and a propagation stage. The killswitch checks the reachability of a URL, preventing detonation if successful. However, failure to reach the URL initiates the propagation process within the network. In the second stage, the malware creates a tasksche.exe process to unpack ransomware files into a peculiar directory within ProgramData. Additionally, it establishes a persistent strange service. Subsequently, the ransomware encrypts data and presents a popup demanding ransom for decryption. This sophisticated ransomware poses a significant threat, necessitating robust security measures and vigilant network monitoring to counter its potential impact.



Basic Static Analysis

File Type

Using File Type, we identify that the Malware Sample is a PE32 Executable (32 Bit) Application.

```
C:\Users\Malware\Desktop\PMAT-labs-main\labs\4-1.Bossfight-wannacry.exe\Ransomware.wannacry.exe.malz\Ransomware.wannacry
.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Figure 1: File Type

CAPA Analysis

Using CAPA without any arguments we can gain a first insight of some of the capabilities of the Malware sample. CAPA detects capabilities in executable files. You run it against a PE, ELF, .NET module, or shellcode file and it tells you what it thinks the program can do. For example, it might suggest that the file is a backdoor, is capable of installing services, or relies on HTTP to communicate.

The CAPA output indicates that the malware sample uses ATT&CK tactics, and by analyzing them, we can gain a preliminary understanding of the malware's capabilities.

➢ Defense Evasion: Obfuscated Files or Information (T1027.005)

The malware uses obfuscation techniques to make its files or information harder to detect and analyze. Obfuscation is a common tactic used by malware authors to hide the true intent of their code and avoid detection by security solutions.

➢ Discovery: a. File and Directory Discovery (T1083)

The malware attempts to gather information about files and directories on the infected system. This information can be used to understand the system's structure and locate potential targets for further exploitation or data exfiltration.

b. System Information Discovery (T1082)

The malware conducts actions to collect information about the infected system. This could include details about the operating system, hardware, software, and other relevant system information.

c. System Network Configuration Discovery (T1016)

The malware tries to gather details about the network configuration of the infected system. This information helps the malware to identify available network resources, potential targets, and ways to propagate across the network.

➢ Execution:

a. Shared Modules (T1129)

The malware utilizes shared modules or dynamic link libraries (DLLs) to execute its malicious code. By using shared modules, the malware can avoid raising suspicions since these files are commonly used by legitimate software.

b. System Services::Service Execution (T1569.002)

The malware leverages system services to execute its code. It may interact with legitimate services or create its own service to achieve persistence and maintain a presence on the infected system.

- Persistence: Create or Modify System Process (T1543.003)

The malware employs a technique to establish persistence by creating or modifying system processes. This allows the malware to automatically start each time the system boots or certain events occur, ensuring its continued presence and operation.

- Further Below we can check the Detailed Capabilities of the malware sample:

Notable examples are that it uses Conditional Execution as Service, C2 Communication to send and receive data and the Cryptography API Call.

MBC Objective	MBC Behavior
ANTI-BEHAVIORAL ANALYSIS	Conditional Execution::Runs as Service [B0025.007]
	Debugger Detection::Timing/Delay Check QueryPerformanceCounter [B0001.03
ANTI-STATIC ANALYSIS	Executable Code Obfuscation::Argument Obfuscation [B0032.020]
	Executable Code Obfuscation::Stack Strings [B0032.017]
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002]
	C2 Communication::Send Data [B0030.001]
COMMUNICATION	HTTP Communication::Create Request [C0002.012]
	HTTP Communication::Open URL [C0002.004]
	Socket Communication::Connect Socket [C0001.004]
	Socket Communication::Create TCP Socket [C0001.011]
	Socket Communication::Create UDP Socket [C0001.010]
	Socket Communication::Get Socket Status [C0001.012]
	Socket Communication::Initialize Winsock Library [C0001.009]
	Socket Communication::Receive Data [C0001.006]
	Socket Communication::Send Data [C0001.007]
	Socket Communication::Set Socket Config [C0001.001]
	Socket Communication::TCP Client [C0001.008]
CRYPTOGRAPHY	Generate Pseudo-random Sequence::Use API [C0021.003]
DATA	Compression Library [C0060]
DISCOVERY	Code Discovery::Inspect Section Memory Permissions [B0046.002]
	File and Directory Discovery [E1083]
EXECUTION	Install Additional Program [B0023]
FILE SYSTEM	Move File [C0063]
	Read File [C0051]
PROCESS	Create Thread [C0038]
	Terminate Process [C0018]
	Terminate Thread [C0039]

Figure 2:MBC Objectives

String Analysis

String analysis in malware analysis involves extracting human-readable text (strings) from malware code to reveal C2 communication, encryption keys, file paths, function names, and IOCs. It helps researchers understand malware behavior and develop mitigation strategies.

By utilizing Floss with the "-n 8" argument and directing the output to a text file, we can analyze the strings within the malware sample. The initial observations reveal Win API Calls employed by the malware, with notable detections such as CryptAcquireContextA, CryptGenRandom, CryptGenKey, CryptDecrypt, CryptEncrypt, CryptDestroyKey, CryptImportKey, and CryptAcquireContextA.

Additionally, we notice the recurrent presence of the string "!This program cannot be run in DOS mode." *This suggests that the executable may contain packed other programs*.

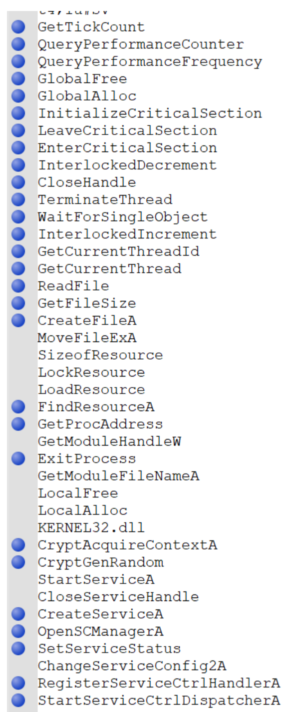


Figure 3:Sample of APIs Used

During the string analysis process, we have discovered intriguing strings, including a notable IOC - a URL***: http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com***. This suggests that the malware sample attempts to connect to this URL. Moreover, we've come across an unusual directory indicated by the "%s" string: **C:%s\qeriuwjhrf**. The usage of **tasksche.exe** is also observed.

Furthermore, we've identified the usage of command lines with the "%s" string, indicating potential command-line arguments being passed: **cmd.exe /c "%s"**. Encoded strings have been detected as well, along with the usage of "**icacls . /grant Everyone:F /T /C /Q**," a command that modifies permissions on directories and files. Lastly, we've encountered the string "WANACRY!".

This string analysis process has provided valuable insights into the behavior and characteristics of the malware sample.

```
advapi32.dll
WANACRY!
CloseHandle
DeleteFileW
MoveFileExW
MoveFileW
ReadFile
WriteFile
CreateFileW
kernel32.dll
2/O-_.X8w.+
Microsoft Enhanced RSA and AES Cryptographic Provider
CryptGenKey
CryptDecrypt
CryptEncrypt
CryptDestroyKey
CryptImportKey
CryptAcquireContextA
cmd.exe /c "%s"
115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
Global\MsWinZonesCacheCounterMutexA
tasksche.exe
TaskStart
icacls . /grant Everyone:F /T /C /Q
attrib +h .
WNcry@2ol7
GetNativeSystemInfo
.?AVexception@@
```

Figure 4: IOC from Strings

PE Studio

Using PE Studio we can get detailed information about this malware Sample

property	value
md5	DB349B97C37D22F5EA1D1841E3C89EB4
sha1	E889544AFF85FFAF8B0D0DA705105DEE7C97FE26
sha256	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	M Z .. .. @ .. ..
file-size	3723264 bytes
entropy	7.964
imphash	n/a
signature	Microsoft Visual C++ v6.0
tooling	Visual Studio 6.0
entry-point	55 8B EC 6A FF 68 A0 A1 40 00 68 A2 9B 40 00 64 A1 00 00 00 00 50 64 89 25 00 00 00 00 83 EC 68 53
file-version	6.1.7601.17514 (win7sp1_rtm.101119-1850)
description	Microsoft® Disk Defragmenter
file-type	executable
cpu	32-bit
subsystem	GUI
compiler-stamp	Sat Nov 20 09:03:08 2010 | UTC
debugger-stamp	n/a
resources-stamp	0x00000000
import-stamp	0x00000000
exports-stamp	n/a

Figure 5: PEstudio Info

Information such as hashes, file size, the first bytes, and the CPU architecture can provide valuable insights into the design of this malware. Additionally, examining indicators allows us to gain immediate insight into the suspicious components of the malware sample. Most importantly, we can cross-reference flagged suspicious libraries from PE studio with the API calls detected during string analysis.

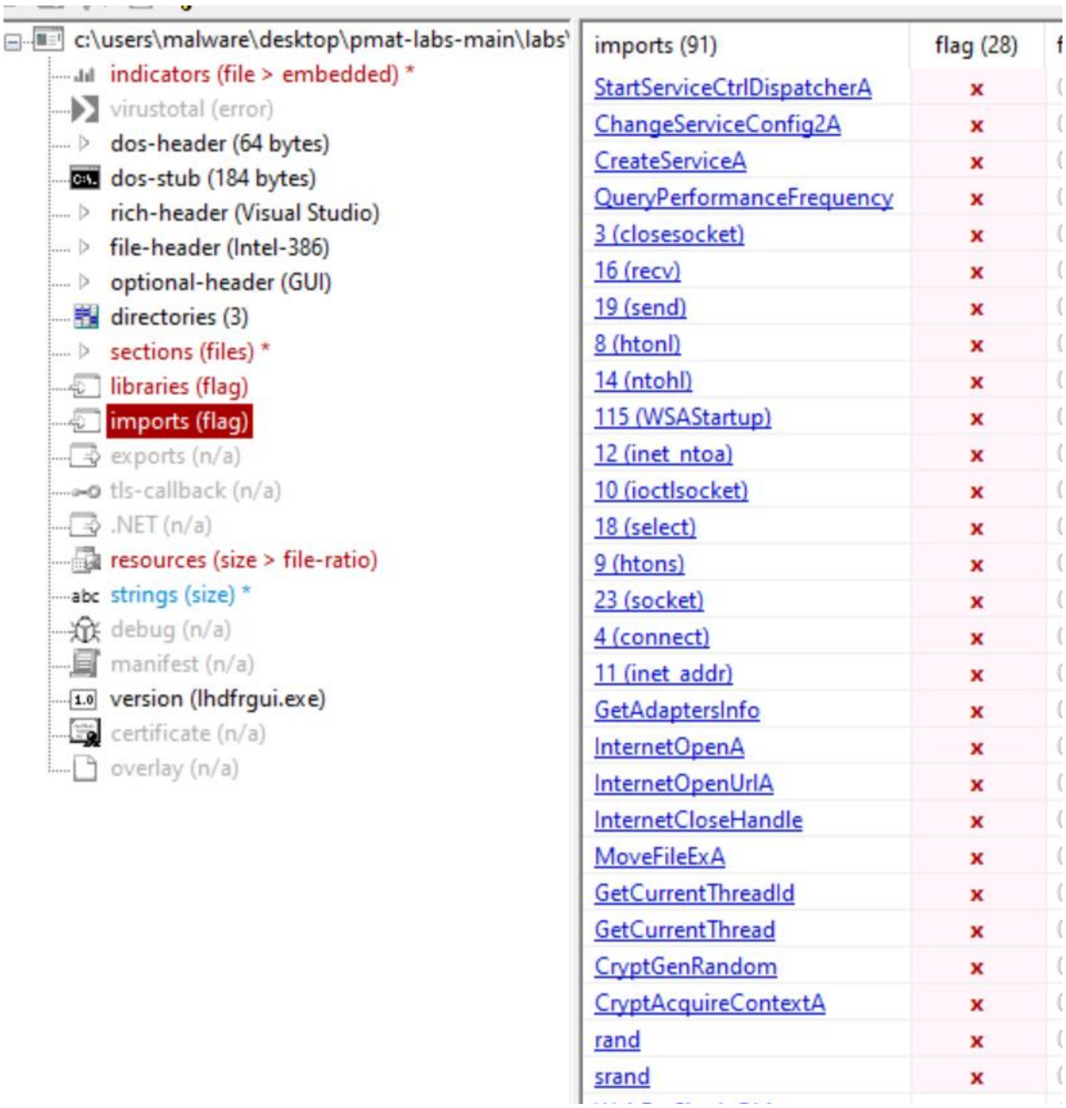


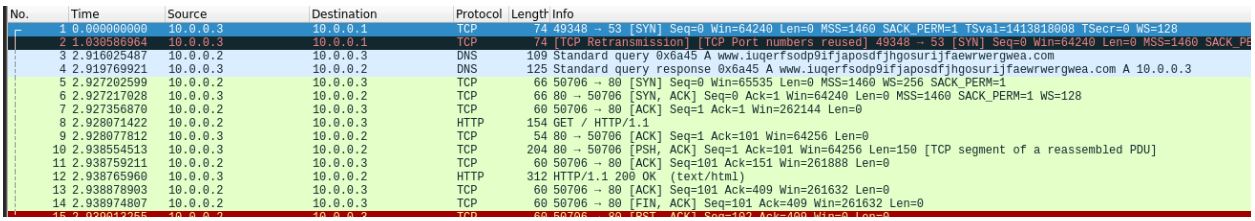
Figure 6: PEstudio Imports Indicators

Basic Dynamic Analysis

Setting the environment for Dynamic Analysis:

We will configure ProcMon, starting with a process name filter for the malware sample. Additionally, we will open TCPView and Procexp. Finally, we will take an initial registry snapshot using RegShot.

- Network Detonation: After the initial detonation of the malware with internet capabilities using inetsim, it appears that the payload is not triggered or activated.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.3	10.0.0.1	TCP	74	49348 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1413818008 TSecr=0 WS=128
2	1.030586964	10.0.0.3	10.0.0.1	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 49348 → 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PE
3	2.916025487	10.0.0.2	10.0.0.3	DNS	109	Standard query 0x6a45 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
4	2.919769921	10.0.0.3	10.0.0.2	DNS	125	Standard query response 0x6a45 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com A 10.0.0.3
5	2.927202599	10.0.0.2	10.0.0.3	TCP	66	50706 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
6	2.927217028	10.0.0.3	10.0.0.2	TCP	66	80 → 50706 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 WS=128
7	2.927356870	10.0.0.2	10.0.0.3	TCP	60	50706 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
8	2.928071422	10.0.0.2	10.0.0.3	HTTP	154	GET / HTTP/1.1
9	2.928077812	10.0.0.3	10.0.0.2	TCP	54	80 → 50706 [ACK] Seq=1 Ack=101 Win=64256 Len=0
10	2.938554513	10.0.0.3	10.0.0.2	TCP	204	80 → 50706 [PSH, ACK] Seq=1 Ack=101 Win=64256 Len=150 [TCP segment of a reassembled PDU]
11	2.938759211	10.0.0.2	10.0.0.3	TCP	60	50706 → 80 [ACK] Seq=101 Ack=151 Win=261888 Len=0
12	2.938765960	10.0.0.3	10.0.0.2	HTTP	312	HTTP/1.1 200 OK (text/html)
13	2.938878903	10.0.0.2	10.0.0.3	TCP	60	50706 → 80 [ACK] Seq=101 Ack=409 Win=261632 Len=0
14	2.938974807	10.0.0.2	10.0.0.3	TCP	60	50706 → 80 [FIN, ACK] Seq=101 Ack=409 Win=261632 Len=0

Figure 7:Network Detonation - Enigmatic URL

The malware sample attempts to communicate with the unusual URL, acting as a killswitch; if reached, the malware will not detonate. Interestingly, even with administrative privileges, the malware fails to trigger.

- Without Network Simulation:

  The malware sample successfully triggers and encrypts our data. Additionally, we encounter the infamous picture associated with WannaCry ransomware, indicating a potential ransomware infection.

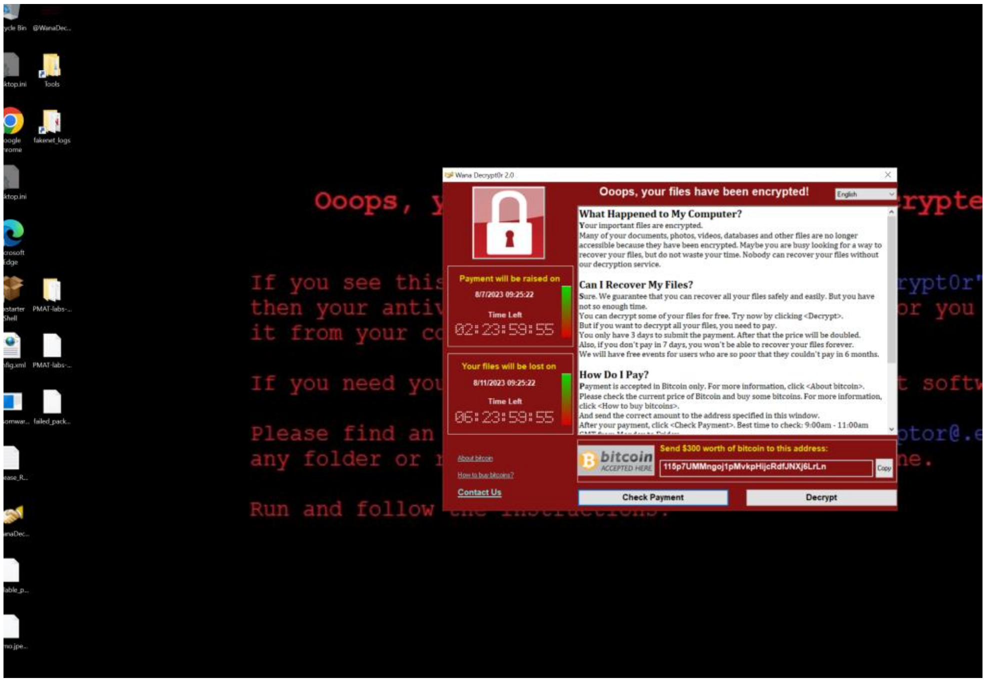


Figure 8: Wannacry Ransomware

Network Analysis:

During the initial activation of the WannaCry ransomware, we can clearly observe the process attempting to communicate with other systems in our network using SMB, aiming to propagate itself and function as a network worm.

Following the ransomware's activation, we observe the WannaCry_Decryptor@exe establishing a connection to remote port 9050.

taskhsvc.exe	1684	TCP	Established	127.0.0.1	61495	127.0.0.1	61496	04/08/2023 09:24:57	taskhsvc.exe
taskhsvc.exe	1684	TCP	Established	127.0.0.1	61496	127.0.0.1	61495	04/08/2023 09:24:57	taskhsvc.exe
taskhsvc.exe	1684	TCP	Established	127.0.0.1	9050	127.0.0.1	21709	04/08/2023 09:28:37	taskhsvc.exe
@WanaDecryptor@.exe	2716	TCP	Established	127.0.0.1	21709	127.0.0.1	9050	04/08/2023 09:28:37	@WanaDecryptor@.exe
System	4	TCP	Listen	10.0.0.2	139	0.0.0.0	0	31/07/2023 13:13:39	System

Figure 9:Immediately after running TCPview

PROCmon

Procmon (Process Monitor) is a Windows tool used for malware analysis. It monitors and logs system activities, providing insights into file system, registry, and process behavior. Analysts use Procmon to understand malware actions, track changes, and identify potential malicious activities. Its real-time monitoring aids in detecting and analyzing malware behavior efficiently.

With Procmon, we filtered out the sample using the executable file's process name. After detonating the malware, we can view its behavior in parts. The first part involves process and thread creation.

From the analysis, we detect that the malware created a new process called "taskshe.exe" with PID 5692.
Furthermore, we can see the usage of other dll files like bcrypt for its ransomware purpose.

:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\urlmon.dll	SUCCESS	Image Base: 0x728...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\netutils.dll	SUCCESS	Image Base: 0x74c...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\srvcli.dll	SUCCESS	Image Base: 0x73a...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\oleaut32.dll	SUCCESS	Image Base: 0x770...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\dnsapi.dll	SUCCESS	Image Base: 0x73f...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\rasadhlp.dll	SUCCESS	Image Base: 0x73f...
:2...	Ransomware.w...	7840	Thread Create		SUCCESS	Thread ID: 8792
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\cryptsp.dll	SUCCESS	Image Base: 0x74a...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\rsaenh.dll	SUCCESS	Image Base: 0x73e...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\bcrypt.dll	SUCCESS	Image Base: 0x75f...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\cryptbase.dll	SUCCESS	Image Base: 0x745...
:2...	Ransomware.w...	7840	Load Image	C:\Windows\SysWOW64\bcryptprimitives.dll	SUCCESS	Image Base: 0x76c...
:2...	Ransomware.w...	7840	Thread Create		SUCCESS	Thread ID: 8456

Figure 10: DLL Used

21:2...	Ransomware.w...	7840	Thread Create		SUCCESS	Thread ID: 7564
21:2...	Ransomware.w...	7840	Thread Create		SUCCESS	Thread ID: 4936
21:2...	Ransomware.w...	5692	Process Create	C:\WINDOWS\tasksche.exe	SUCCESS	PID: 5772, Comma...
21:2...	Ransomware.w...	7840	Thread Create		SUCCESS	Thread ID: 512
21:2...	Ransomware.w...	5692	Thread Exit		SUCCESS	Thread ID: 6024, U...

Figure 11: Creation of taskshe.exe

We can view the process tree of the spawned processes from the WannaCry ransomware.

Ransomware.wannacry.exe (7	Microsoft® Disk D...	C:\Users\Malware\...	Microsoft Corporati...	NT AUTHORITY\S.
cmd.exe (6536)	Windows Comman...	C:\Windows\syste...	Microsoft Corporati...	NT AUTHORITY\S.
tasksche.exe (1128)	DiskPart	C:\ProgramData\g...	Microsoft Corporati...	NT AUTHORITY\S.
attrib.exe (9176)	Attribute Utility	C:\Windows\SysW...	Microsoft Corporati...	NT AUTHORITY\S.
Conhost.exe (6968)	Console Window H...	C:\Windows\Syste...	Microsoft Corporati...	NT AUTHORITY\S.

Figure 12: Process Tree

As we observe, it opens a cmd and the tasksche process, which we previously noticed. Now, we will filter Procmon with the parent PID of taskshe.exe to uncover additional evidence of the malware detonation.

Upon filtering with this parent PID, we obtain the information discovered earlier during string analysis, indicating a peculiar directory in the system.

9:21:2...	tasksche.exe	5772	QuerySecurityFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	SetEndOfFileIn...	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	ReadFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	WriteFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	SetBasicInform...	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	QueryRemotePr...	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	CloseFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	CloseFile	C:\Windows\tasksche.exe
9:21:2...	tasksche.exe	5772	CreateFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	QueryBasicInfor...	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:2...	tasksche.exe	5772	CloseFile	C:\ProgramData\gcpcgbjkayp350\tasksche.exe
9:21:5...	tasksche.exe	5772	CloseFile	C:\Windows
9:21:5...	tasksche.exe	5772	CloseFile	C:\ProgramData\gcpcgbjkayp350

Figure 13: Parent PID Analysis

After inspecting that strange directory in ProgramData, we have come to realize that it is the location where the ransomware unpacked itself and executed the notorious application.

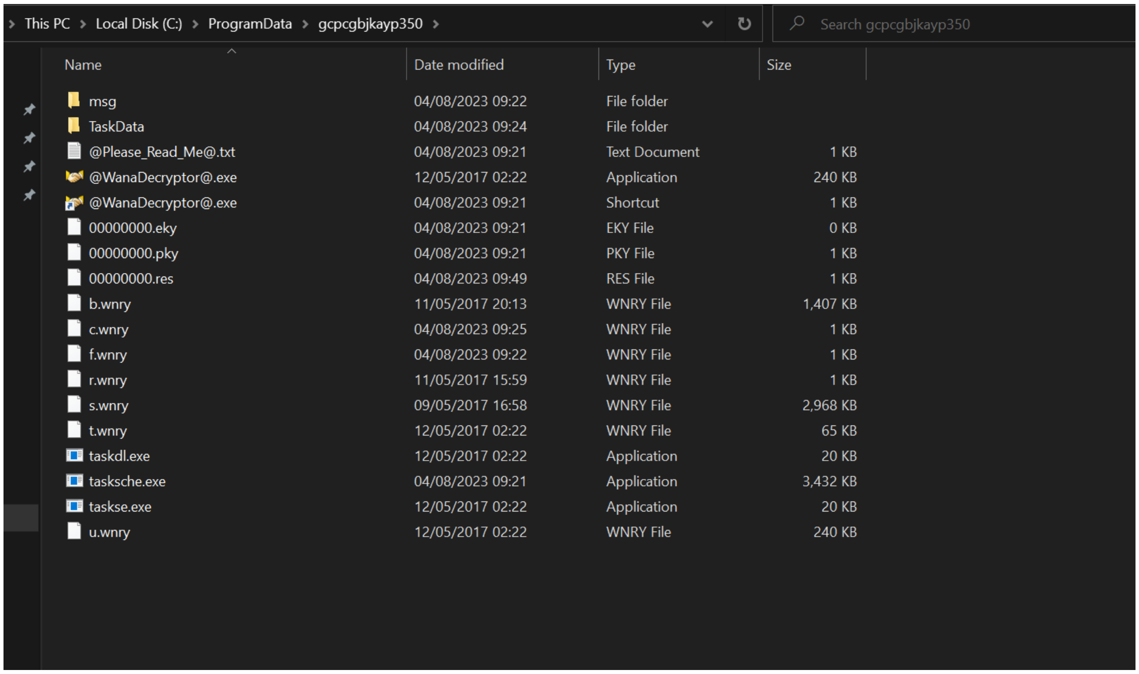


Figure 14: Unpacking Directory

RegShot

Regshot is a utility used in malware analysis to capture and compare system registry snapshots, aiding in identifying changes made by malware to the Windows registry. During our initial static analysis, we noticed that the malware can modify services. By using Regshot, we can discern the specific changes, deletions, and additions made to the registry, comparing a clean snapshot to the one taken after the malware was triggered.

From the comparison, we observed that the ransomware malware deleted 20332 keys.

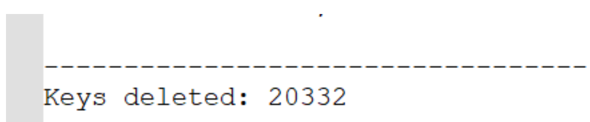


Figure 15: Keys Deleted from Ransomware

We can also observe that it created some keys, and one of them points to the creation of a new service. Furthermore, upon inspecting the Windows services, we can identify the presence of a peculiar service that has been enabled.

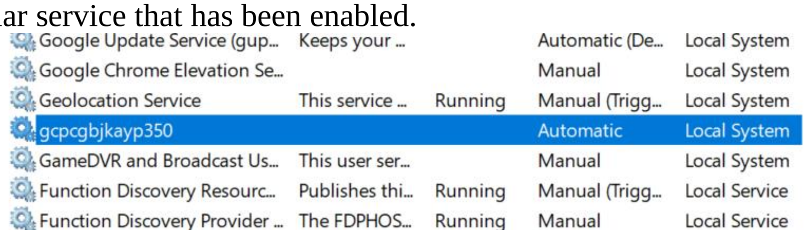


Figure 16: Persistance Service

```
-----------------------------------
Keys added: 49
-----------------------------------
HKLM\SOFTWARE\WOW6432Node\WanaCrypt0r
HKLM\SYSTEM\ControlSet001\Services\gcpcgbjkayp350
HKLM\SYSTEM\ControlSet001\Services\mssecsvc2.0
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0
HKU\.DEFAULT\Software\Microsoft\Windows Script Host
```

Figure 17: Keys Added

```
----------------------------------
Values added: 214
----------------------------------
HKLM\SOFTWARE\WOW6432Node\WanaCrypt0r\wd: "C:\ProgramData\gcpcgbjkayp350"
```

Figure 18: Values Added

It also added values to some keys. One of these values is the new strange directory that unpacks the ransomware, as we analyzed earlier.

Other values include the ones below, indicating that the ransomware is running "cmd" as "taskshe.exe." The service and the values added below serve as the running and persistence mechanism of the ransomware.

```
 69 00 6E 00 73 00 2E 00 74 00 78 00 74 00 00 00 00 00 00 00
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\Type: 0x00000010
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\Start: 0x00000002
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\ErrorControl: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\ImagePath: "cmd.exe /c "C:\ProgramData\gcpcgbjkayp350\tasksche.exe""
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\DisplayName: "gcpcgbjkayp350"
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\WOW64: 0x0000014C
HKLM\SYSTEM\CurrentControlSet\Services\gcpcgbjkayp350\ObjectName: "LocalSystem"
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\Type: 0x00000010
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\Start: 0x00000002
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\ErrorControl: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\ImagePath: "C:\Users\Malware\Desktop\Ransomware.wannacry.exe -m security"
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\DisplayName: "Microsoft Security Center (2.0) Service"
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\WOW64: 0x0000014C
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\ObjectName: "LocalSystem"
HKLM\SYSTEM\CurrentControlSet\Services\mssecsvc2.0\FailureActions: 00 00 00 00 01 00 00 00 01 00 00 00 01 00 00 00 14 00 00 00 01 00 00 00 60 EA 00 00
HKU\S-1-5-21-497346990-3591733918-2170752703-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\FeatureUsage\AppSwitched\C:\Users\Malware\AppData
```

Figure 19: Values Added

Advanced Static Analysis

Let's proceed with the advanced malware analysis using Cutter, a powerful tool that safely disassembles and decompiles executables. With Cutter's capabilities, we can gain detailed insights into the malware's execution, helping us understand its behavior and uncover how it operates.

The first screen we encounter is the dashboard, providing an overview of our malware sample. Here, we gather essential information about its format, class, and type. Additionally, we gain insights into its hashes and receive brief analysis details. This valuable information sets the stage for our in-depth malware analysis.

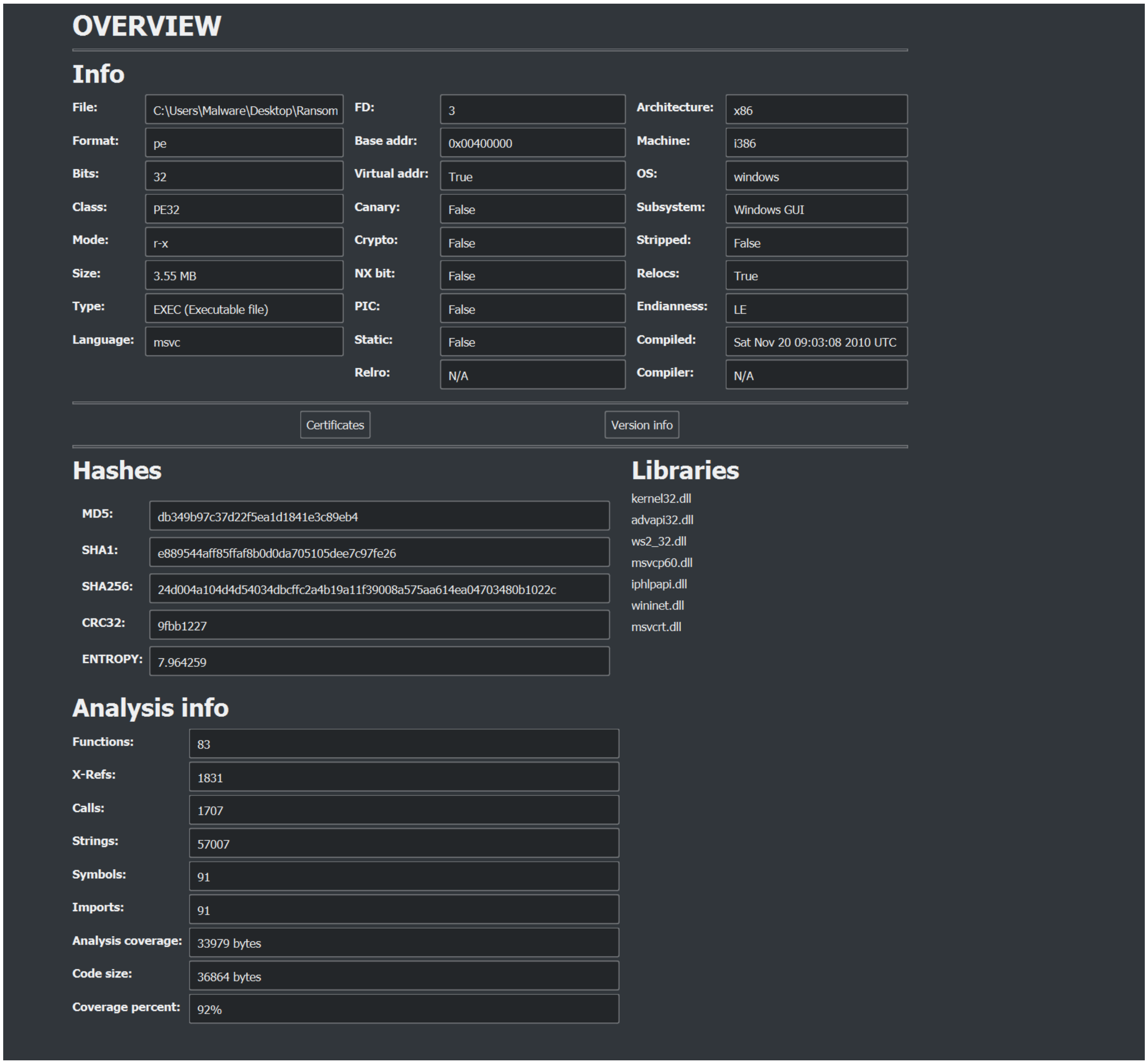


Figure 20: Cutter Dashboard Screen

Upon analyzing the main function of the malware in assembly, we observe the manipulation of the strange URL, moved to the ESI register. Let's note that URL for later use in Advanced Dynamic Analysis. Subsequently, the program invokes the Windows APIs InternetOpenA and InternetOpenAUrlA, utilizing the URL in the ESI register as one of the arguments. Should the URL be successfully reached it returns a bool value and procceds to test EDI against itself and proceeds to the end of the program if the jump is not equal with the zero flag. This behavior suggests that the malware has a kill switch mechanism. If the URL is accessed, it terminates the program, preventing the execution of the ransomware, cleaning the stack and going to ret 0x10 which finished the program. If the URL is reached but is nothing there, it then calls the fuction fcn.00408090.

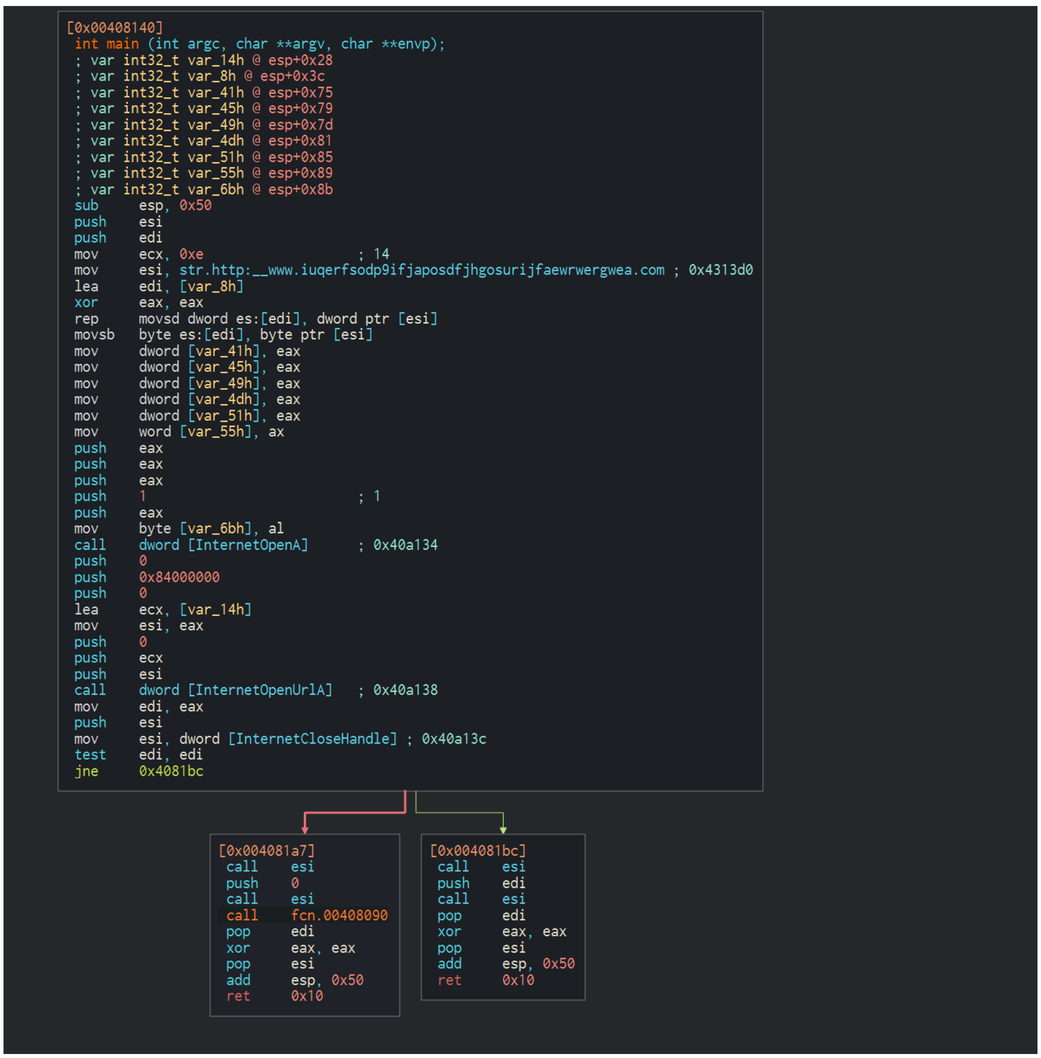


Figure 21: Main of ransomware

Once the malware is successfully executed, it initiates the crucial function call. In a nutshell, this call involves the opening of SCManager and OpenServiceA. Furthermore, a function call to fcn.00407f20 is observed, following a conditional jump (jge). This behavior indicates the malware's attempt to gain control and execute its payload, warranting a closer examination of the involved functions.

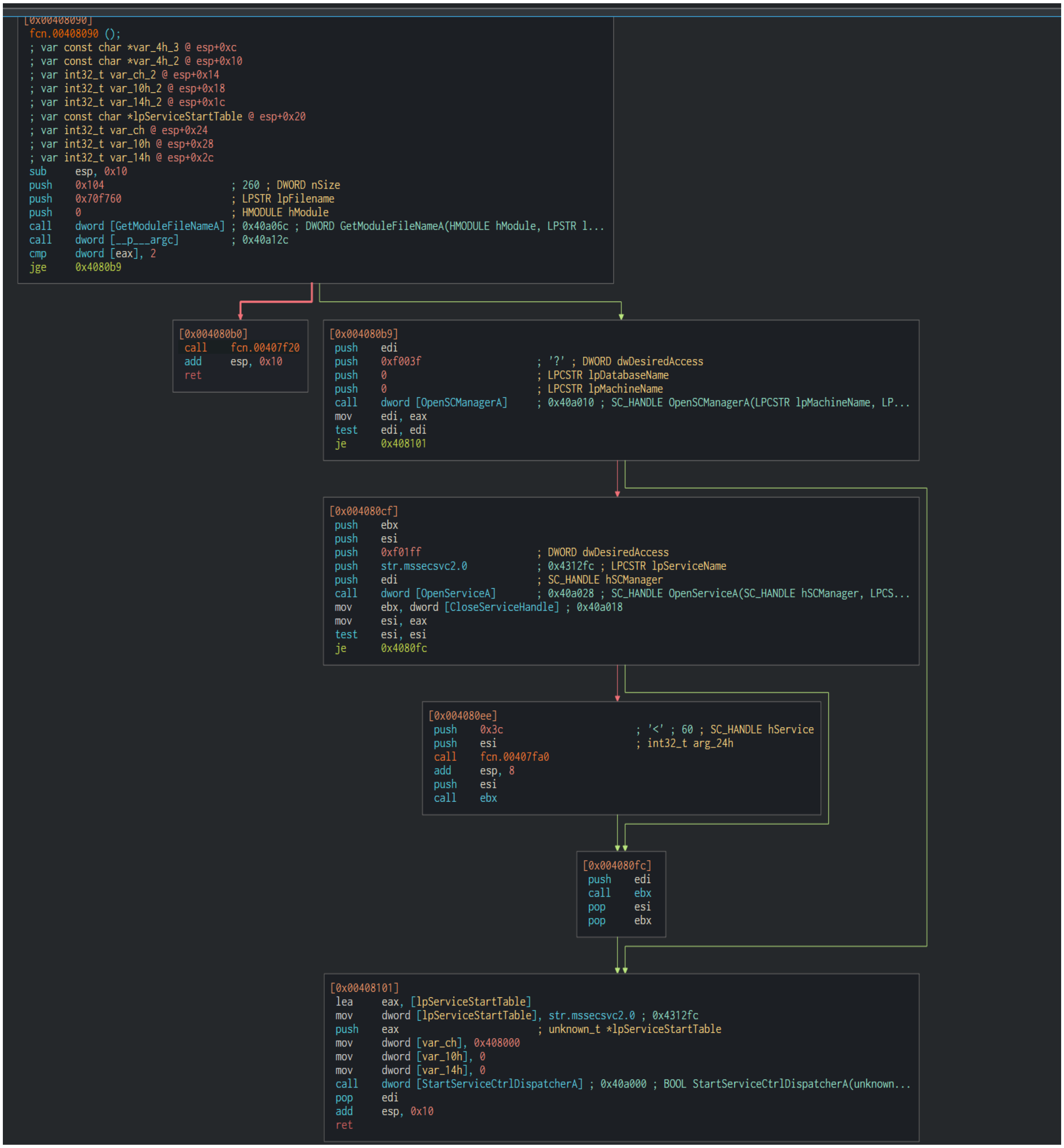


Figure 22:Call fcn.00408090

The call to fcn.00407f20 leads us to a function that invokes two other functions.

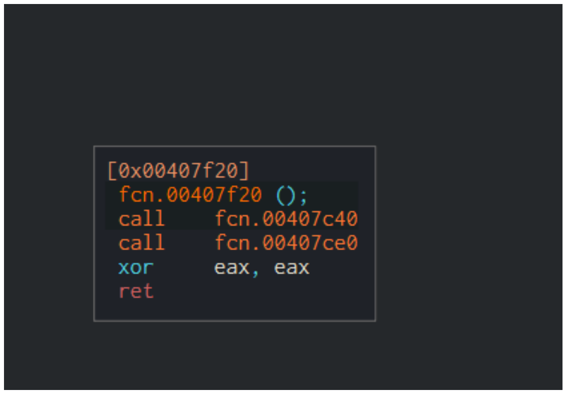


Figure 23: 0040f020 Fuction

The initial function call, fcn.004078c40, is responsible for creating a service with specific characteristics and subsequently starting the service.

Figure 24: First Call

The second call, fcn.00407ce0, represents the ransomware's core payload. This critical function is responsible for orchestrating multiple API calls, including LoadingResources and moveFileExA. These operations suggest that the malware engages in resource loading and file manipulation, which are characteristic behaviors of encryption routines. It's highly likely that this function encrypts files on the system, rendering them inaccessible without the decryption key.

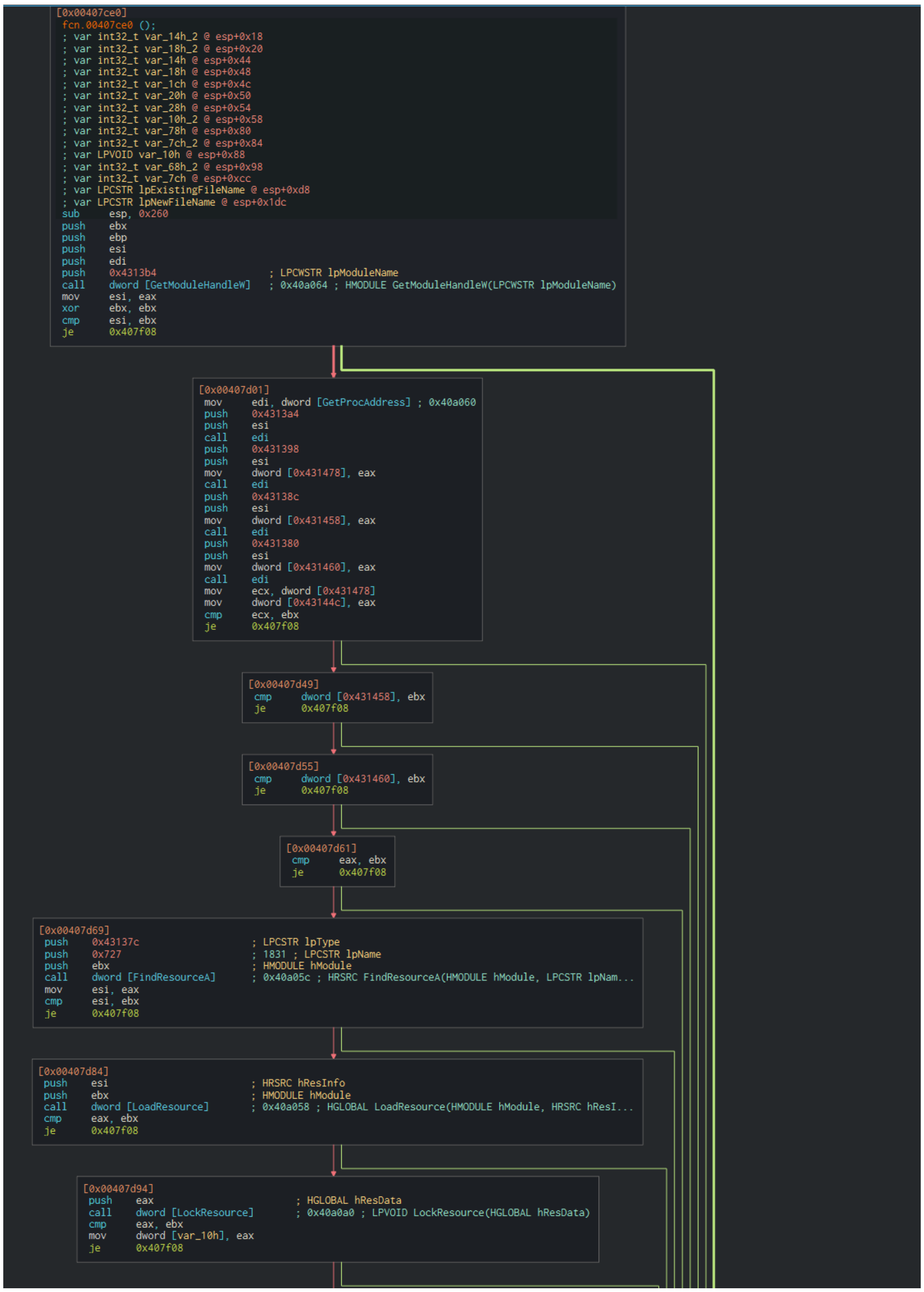


Figure 25: Payload

Figure 26:Payload

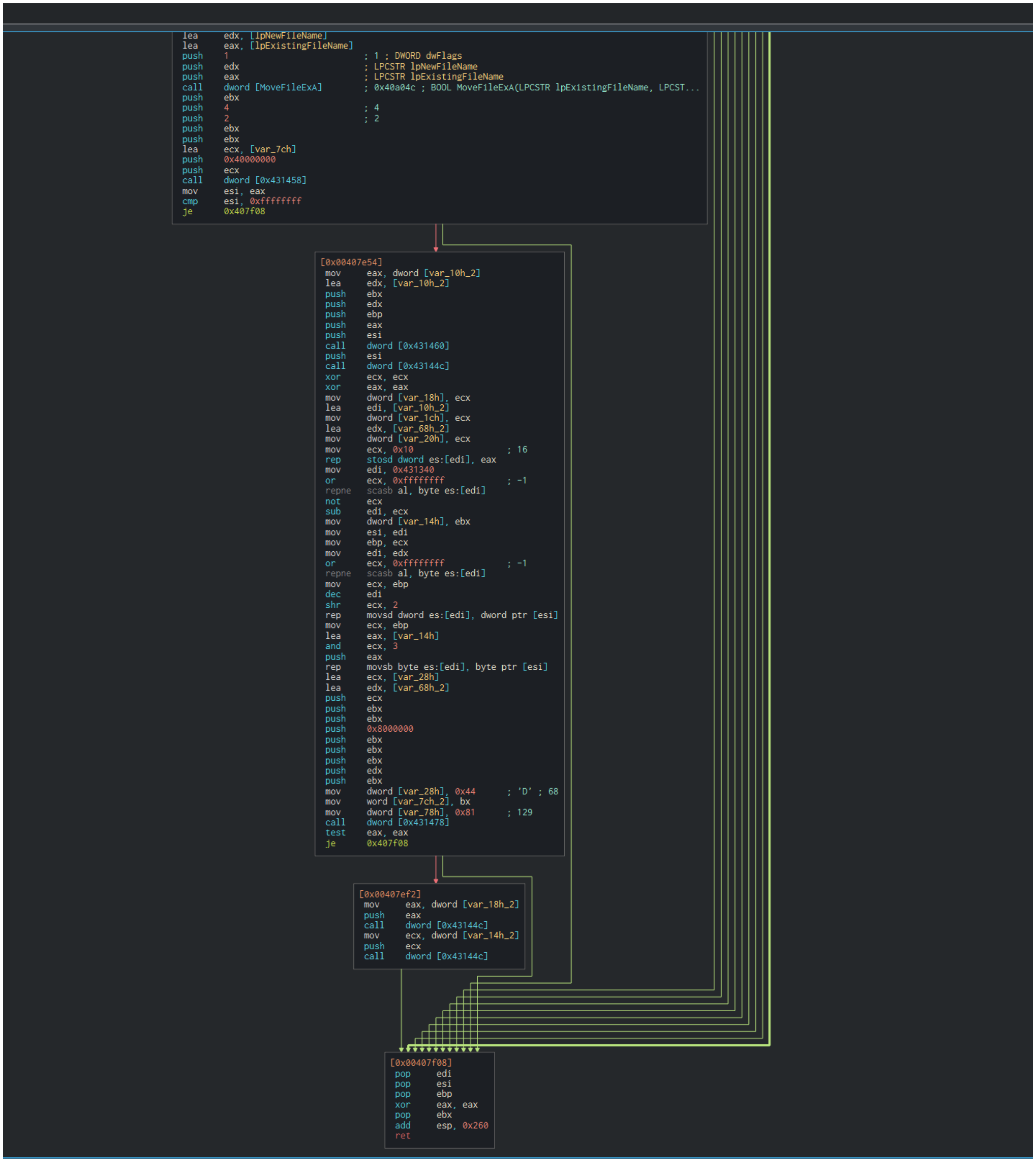


Figure 27: Payload

Advanced Dynamic Analysis

Proceeding with advanced dynamic analysis using debuggers requires utmost caution, as it involves running the program directly on the CPU within the system. This method offers real-time insights into the malware's behavior and interactions with the environment. However, due to its direct execution, there is a risk of unintended consequences and potential system impact. Engaging in controlled environments and employing virtualization is crucial to mitigate risks and maintain a safe testing environment during dynamic analysis.

In this Phase we will use x32dbg.

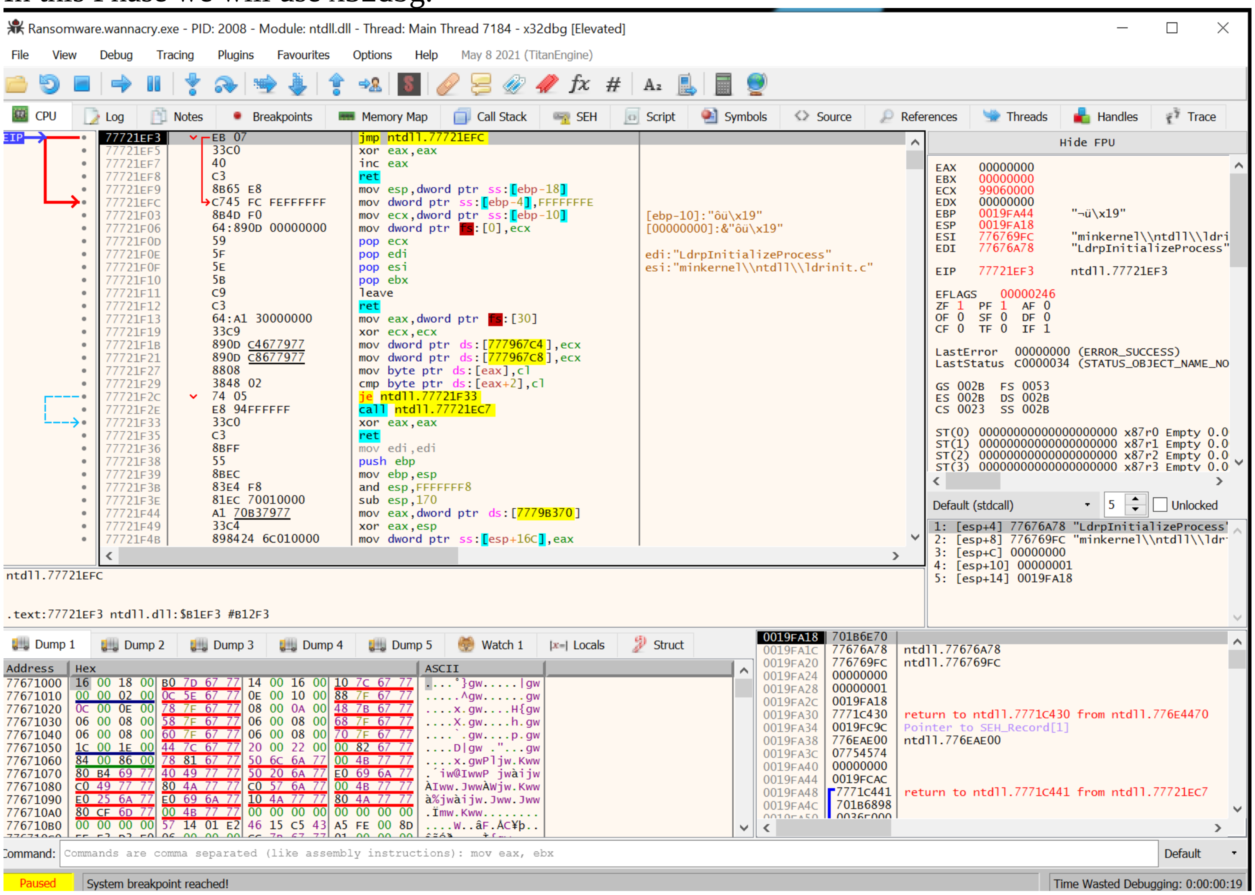


Figure 28: x32dbg

We will press F9 once to jump to the entry point. Previously, we identified the strange URL, and we are aware that there's a comparison (test) near it, determining the jump for the malware's kill switch. To locate this comparison, we will conduct a thorough search in all modules for the string reference of the strange URL. Once found, we will strategically place a breakpoint to examine the malware's behavior at that critical point in the code.

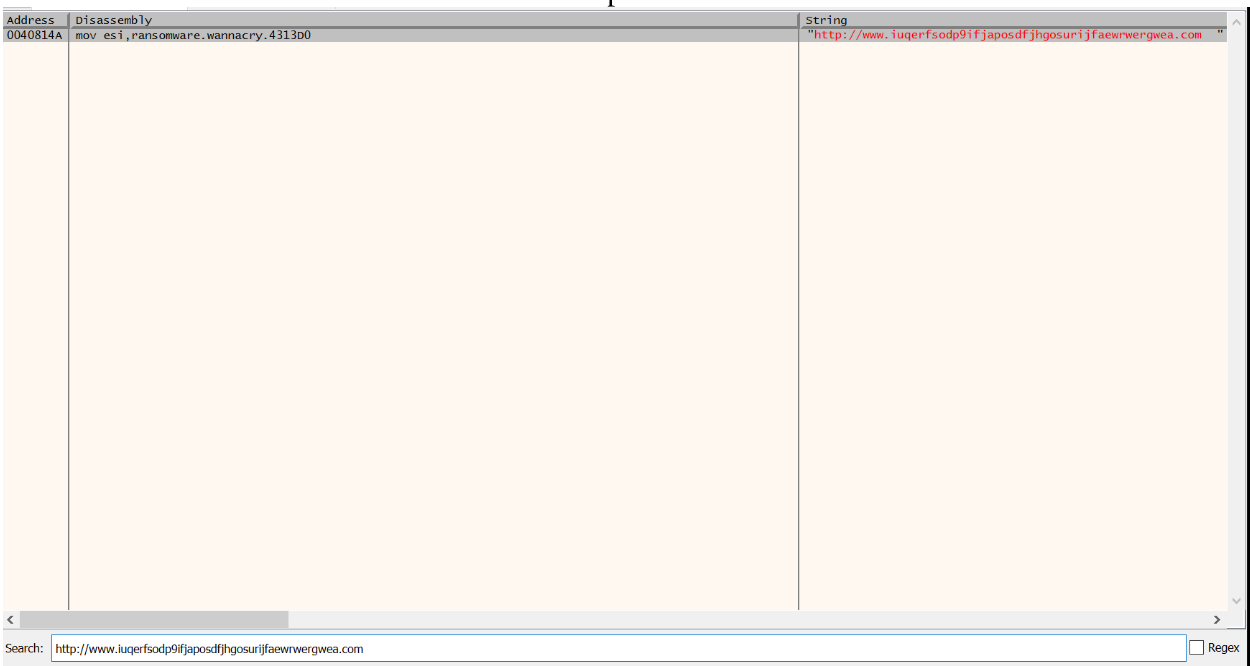


Figure 29: Breaking Point for Enigmatic URL

Upon pressing F9 to execute the program, we will reach the breakpoint we previously set. This strategic breakpoint allows us to pause the execution at a critical moment, enabling us to inspect the malware's behavior and gather valuable insights for further analysis.

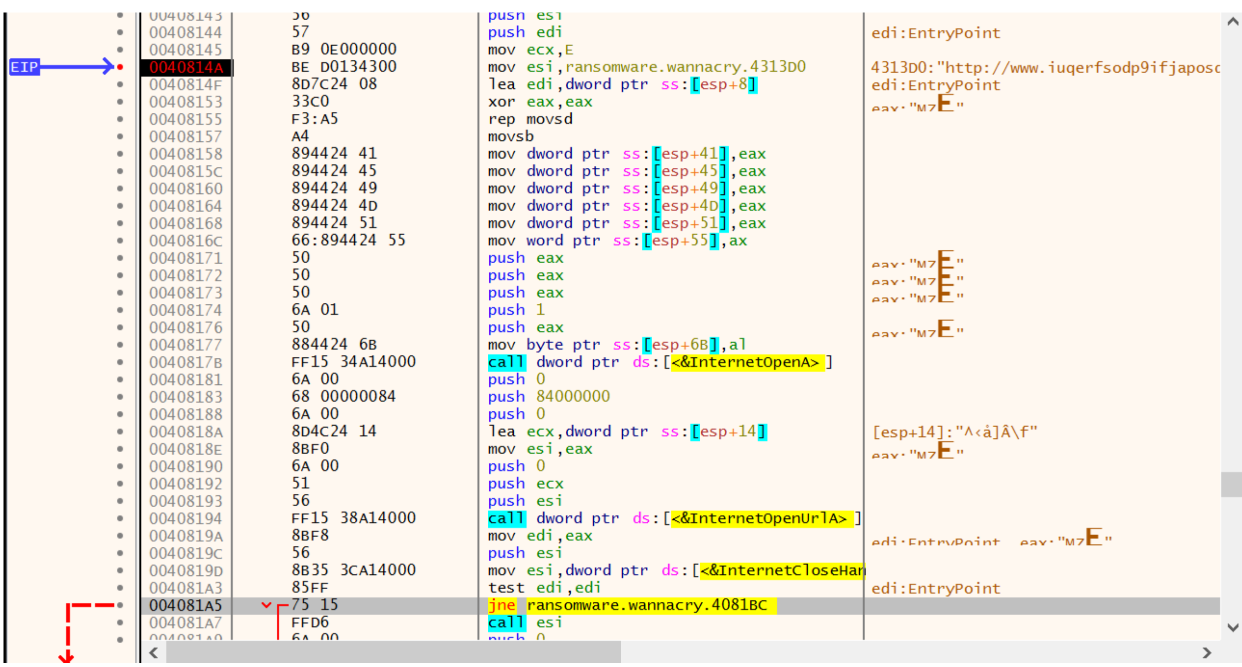


Figure 30: Finding the sweet spot

In the above string, we recognize a familiar sequence of code that we previously encountered in the disassembler. This section of the program employs the InternetOpenA API and

InternetOpenUrlA. Now, we will run the program until we reach the "test edi,edi" assembly instruction.

Upon inspection, we observe that the Zero Flag (ZF) is set to 1, indicating that the malware did not reach the strange URL, and it is prepared to detonate. If we were to change the ZF to 0, the malware would not execute, as the killswitch mechanism would activate, preventing its further progression.

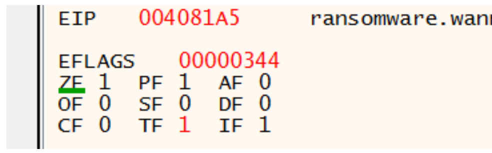


Figure 31: If network is not Enabled - ZF 1

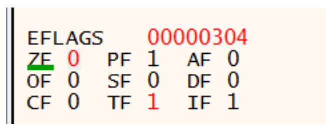


Figure 32: If network is enabled and reaches out to the URL ZF 0

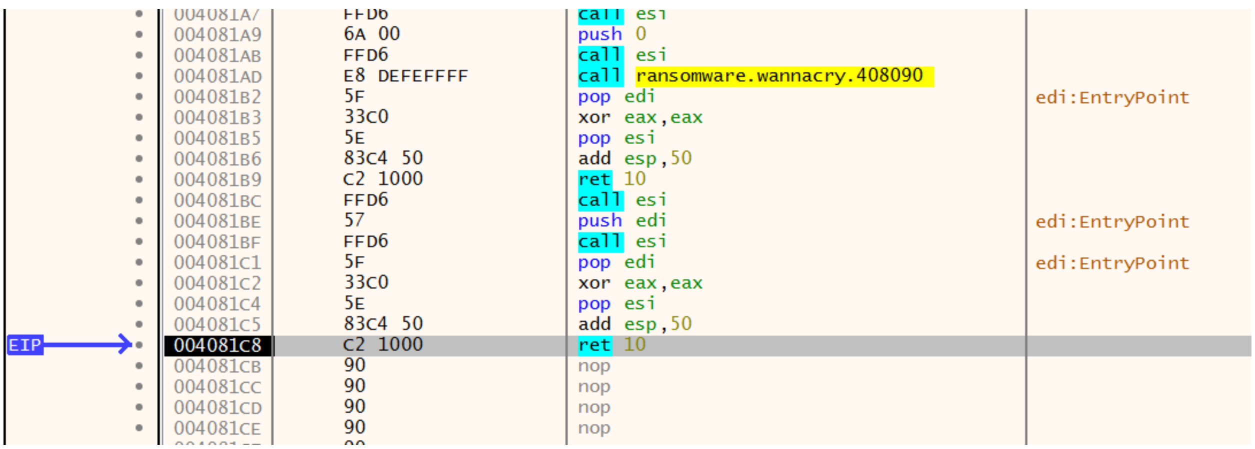


Figure 33: With ZF 0 that we changed earlier, the program finishes

Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

- Detonation with Network:

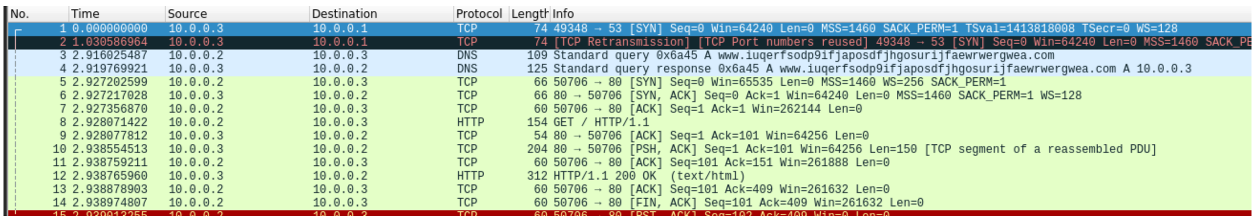


Figure 34: Enigmatic URL 200 OK from inetsim

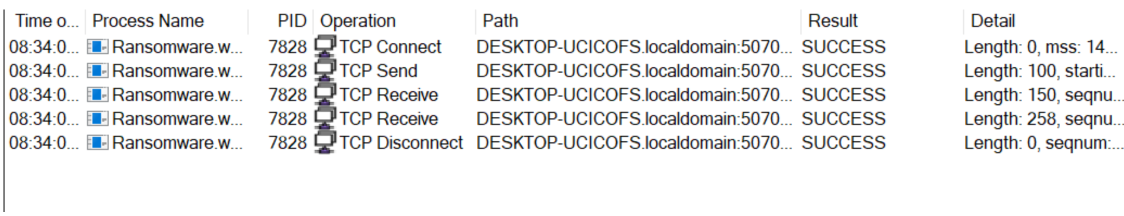


Figure 35: ProcMon Connections

- Detonation Without Network:

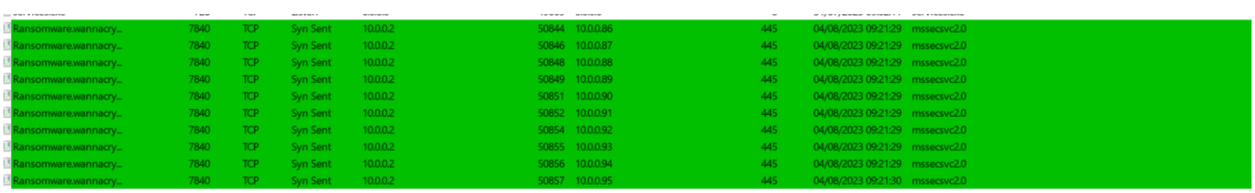


Figure 36: Worm Attributes - SMB

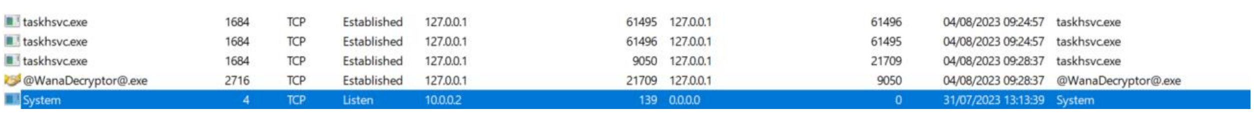


Figure 37: TCPView Wannacry

Host-based Indicators

Figure 38:Creation of taskshe.exe

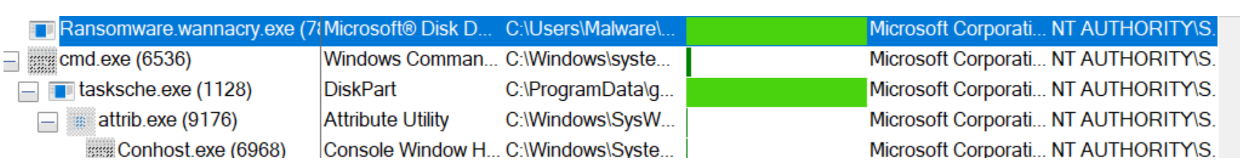


Figure 39: Process Tree

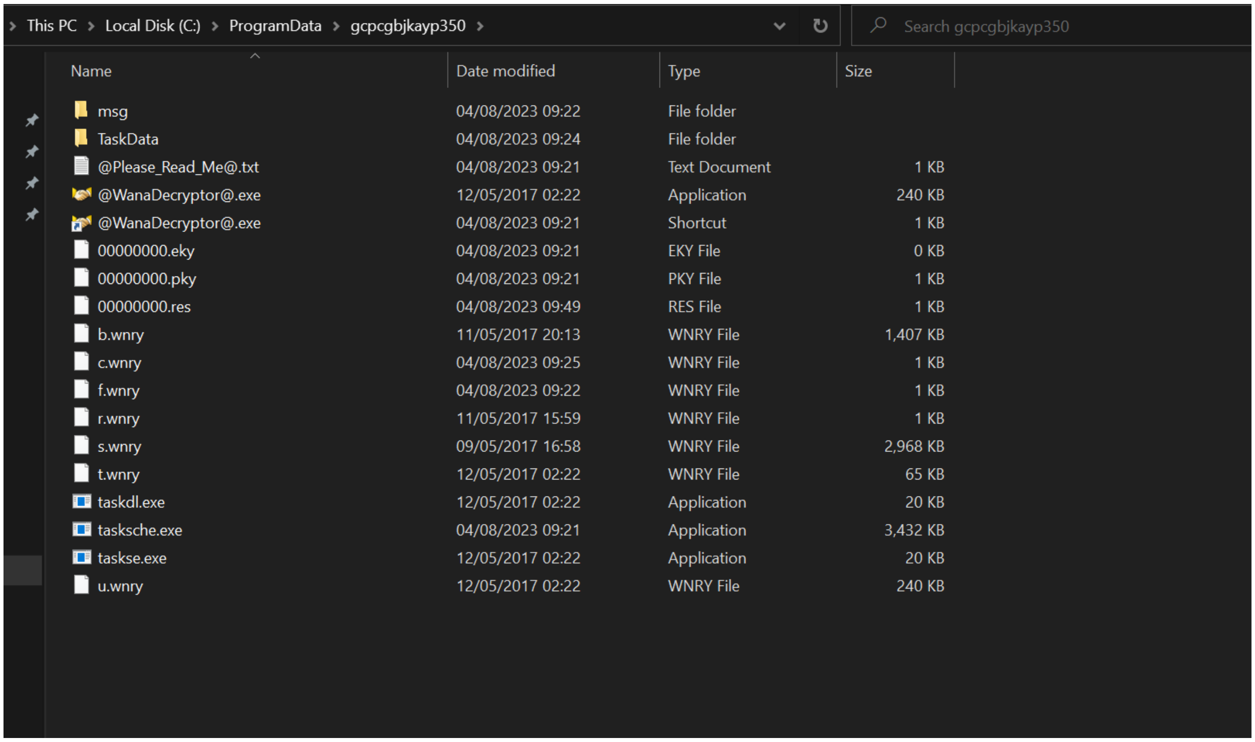


Figure 40: Directory of Ransomware

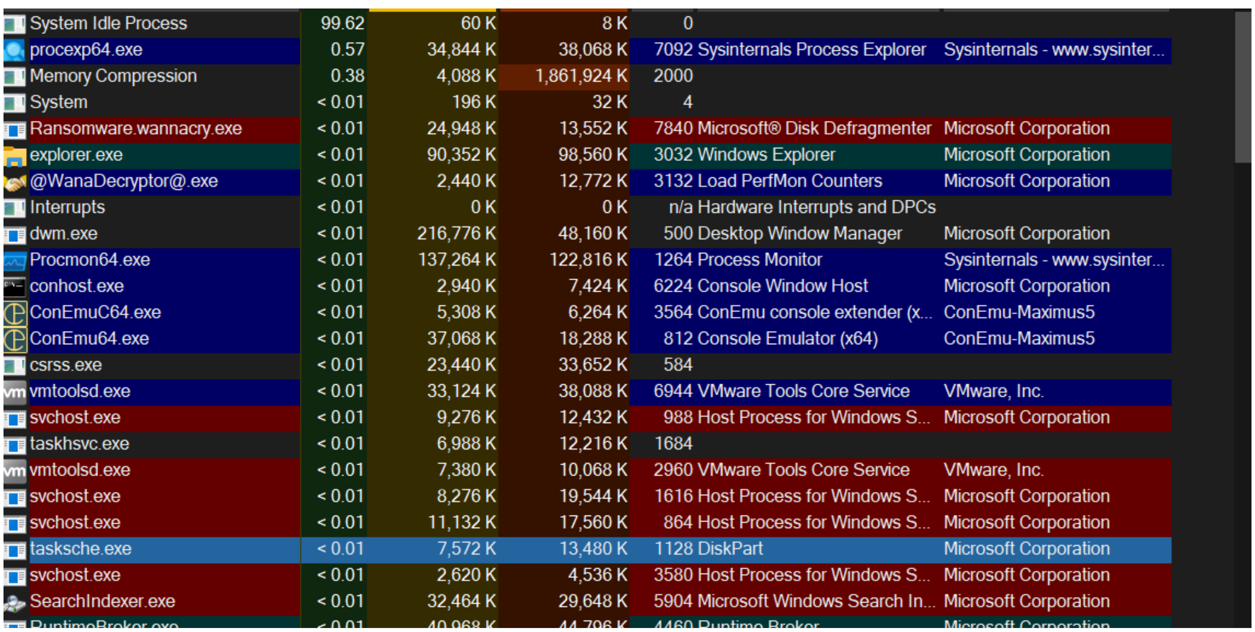
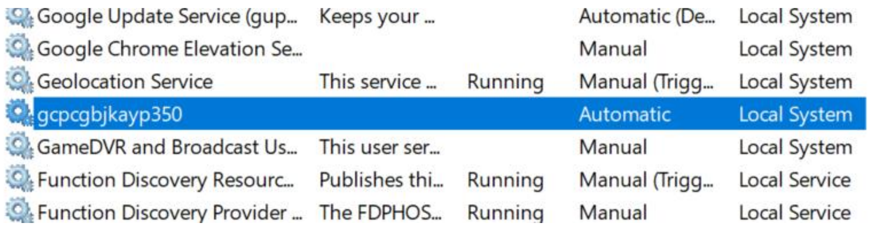


Figure 41: ProcExp of wannacry and taskshe.exe



Google Update Service (gup...	Keeps your ...		Automatic (De...	Local System
Google Chrome Elevation Se...			Manual	Local System
Geolocation Service	This service ...	Running	Manual (Trigg...	Local System
gcpcgbjkayp350			Automatic	Local System
GameDVR and Broadcast Us...	This user ser...		Manual	Local System
Function Discovery Resourc...	Publishes thi...	Running	Manual (Trigg...	Local Service
Function Discovery Provider ...	The FDPHOS...	Running	Manual	Local Service

Figure 42: Service for persistence

Cuckoo Analysis & Yara RULE

Cuckoo Sandbox is an open-source automated malware analysis system. It allows security researchers to execute and analyze suspicious files in a controlled environment. Cuckoo Sandbox provides valuable insights into malware behavior, helping identify potential threats and enhance cybersecurity defenses.

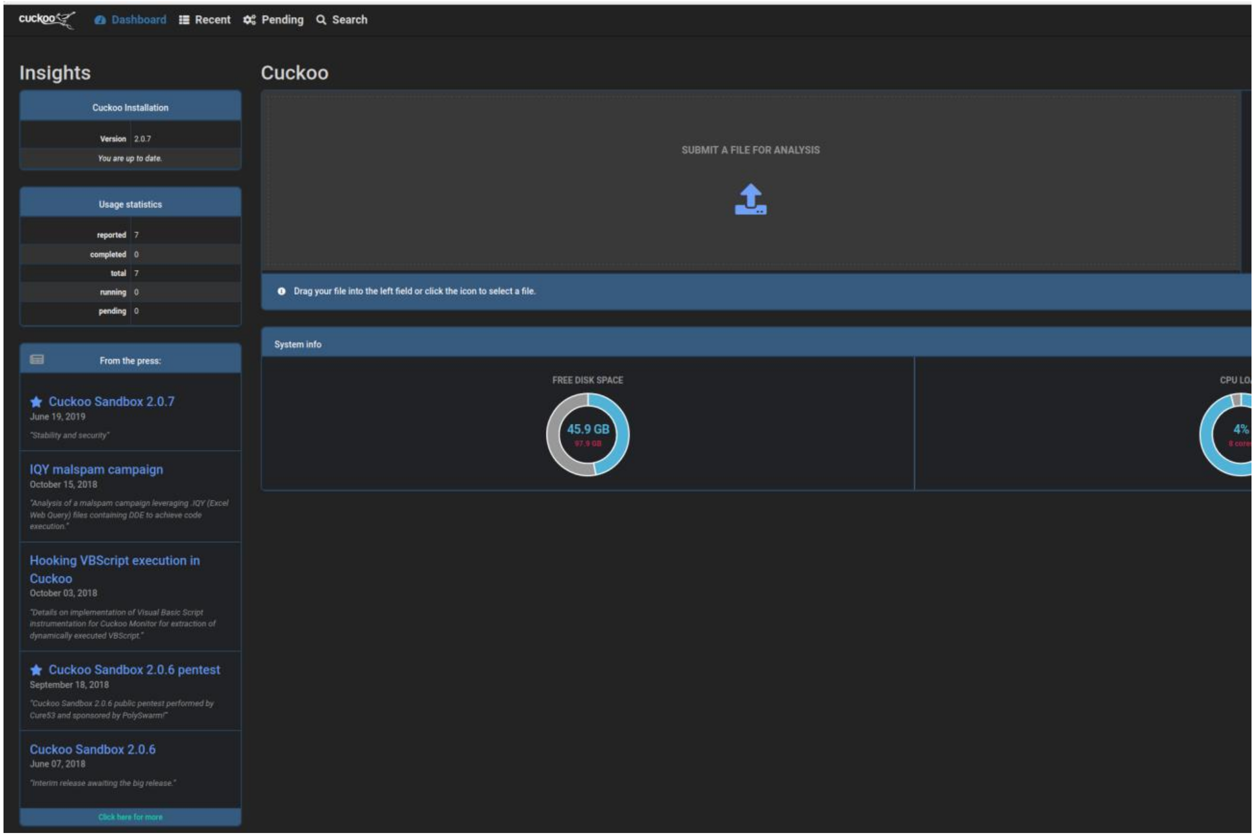


Figure 43: Home Screen Of Cuckoo Sandbox

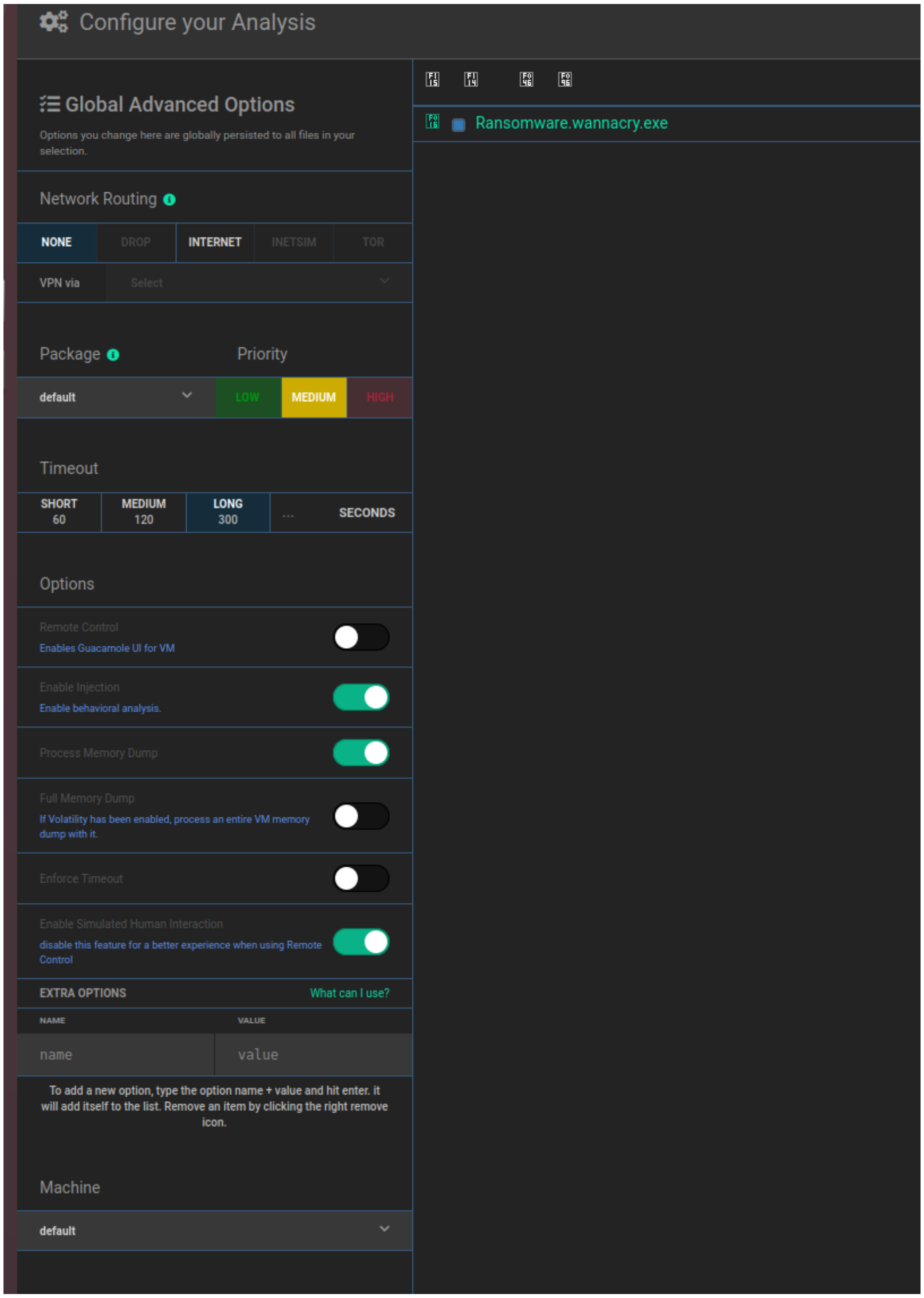


Figure 44: Submitting Wannacry without intenret to Cuckoo

Putting our malware into Cuckoo Sandbox involves submitting the suspicious file to the platform for automated analysis. Cuckoo will execute the malware in a controlled environment, monitor its behavior, and generate detailed reports on its actions. This process helps us gain valuable insights into the malware's capabilities and aids in devising effective countermeasures to protect against similar threats.

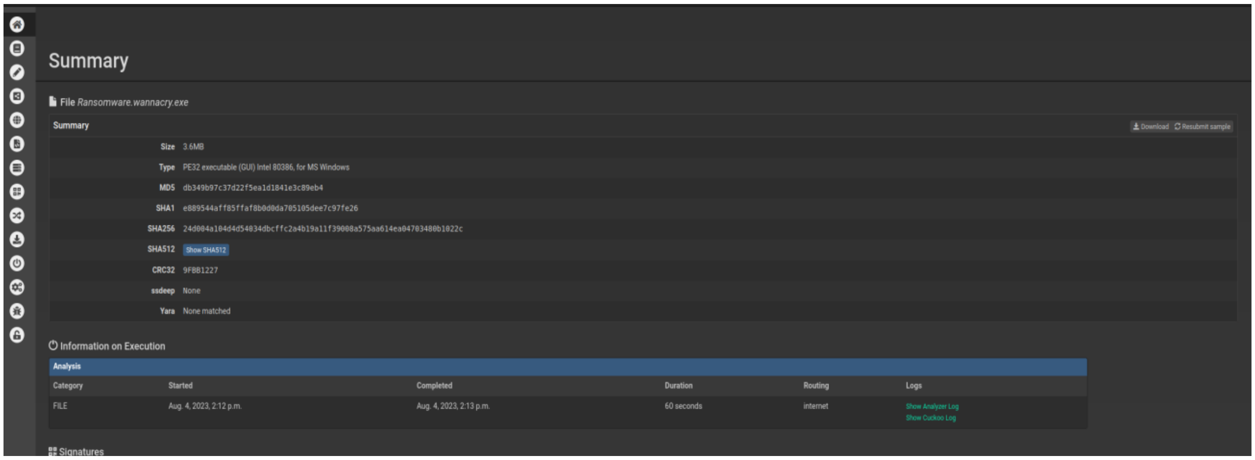


Figure 45: Cuckoo Summary

After analyzing the malware in Cuckoo, we are presented with a comprehensive summary screen containing essential information such as hashes and other details about the analyzed file. Additionally, Cuckoo assigns a score that provides an initial assessment of the file's threat level. In the tab view, we have a range of options to proceed with further analysis, allowing us to delve deeper into the malware's behavior, network interactions, and other critical aspects, aiding us in crafting effective mitigation strategies.

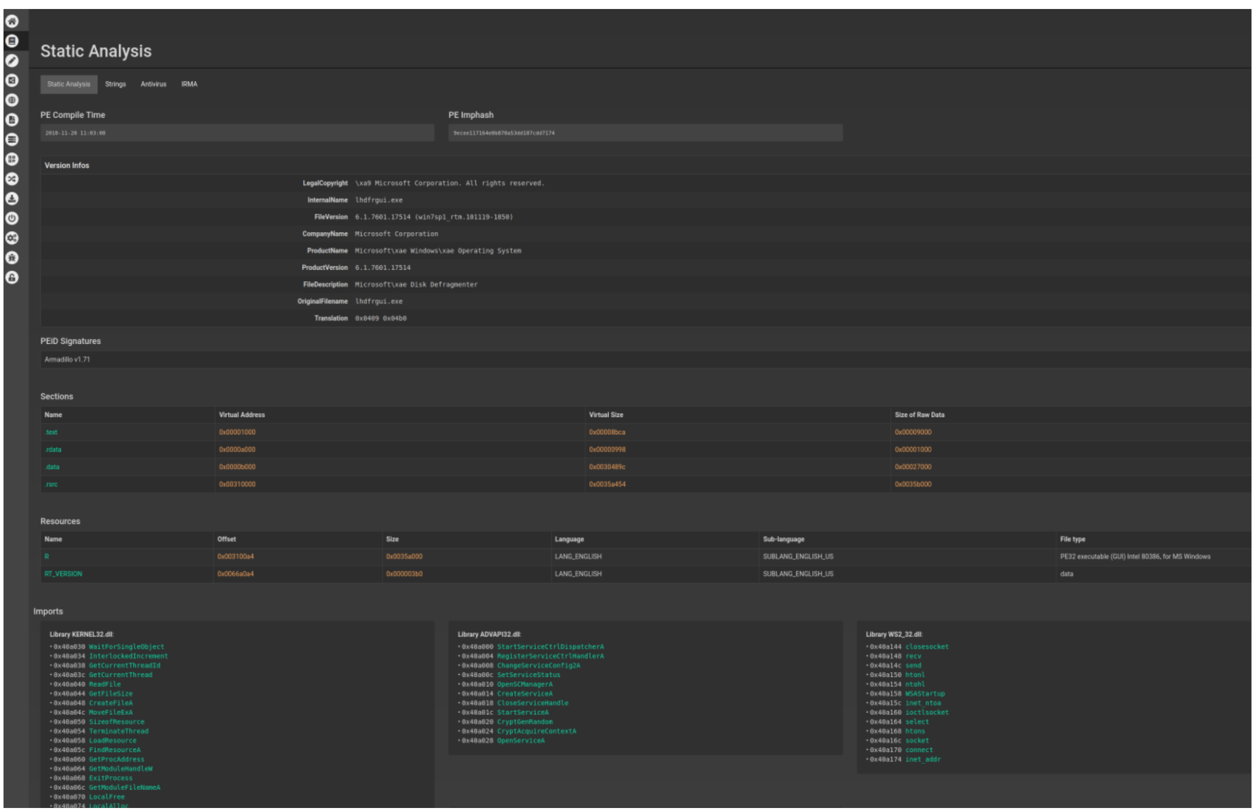


Figure 46: Cuckoo Static Analysis

In Cuckoo Sandbox, during static analysis, we can view the imported APIs to understand the malware's capabilities and interactions with the system. Additionally, we can perform string analysis, extracting and examining strings embedded within the file, which can reveal valuable information about the malware's intent and potential behavior. These insights obtained from static analysis are crucial in assessing the threat and designing appropriate defense mechanisms

During network analysis in Cuckoo, we observed the malware's attempt to access the enigmatic URL. This activity is a critical indicator of potential command and control.

Name	Response
www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com	A → 104.16.173.80
	A → 104.17.244.81

Figure 47: Enigmatic URL

Appendices

A. Yara Rules

```
rule wannacry {

  meta:
    date = "2023-08-04"
    author = "xpinux"
    description = "YARA rule to detect strings associated with WannaCry ransomware"

  strings:
    // Fill out identifying strings and other criteria
    $PE_Magic_Byte = "MZ"
    $str1 = "icacls . /grant Everyone:F /T /C /Q"
    $str2 = "cmd.exe /c \"%s\""
    $str3 = "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
    $str4 = "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
    $str5 = "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
    $str6 = "C:\\%s\\qeriuwjhrf"
    $str7 = “http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com” ascii

  condition:
    // Fill out the conditions that must be met to identify the binary
    $PE_Magic_Byte at 0 and (($str7 and $str1) or ($str2 and $str6) or ($str3 and $str4 and $str5))
}
```

B. Figures