

Practical Malware Analysis & Triage

Malware Analysis Report

WannaCry Ransomware Worm

Oct 2023 | Luis Fernandez Jr. | v1.0

Table of Contents

Table of Contents	2
Executive Summary	3
Malware Composition.....	5
ransomware.wannacry.exe	5
mssecsvc.exe	5
tasksche.exe	5
Basic Static Analysis.....	6
Analysis of Strings (FLOSS)	6
PeView & PeStudio	7
Basic Dynamic Analysis.....	9
INETSIM: On (Killswitch Success)	9
INETSIM: Off (Killswitch Unsuccessful)	10
ProcMon	11
TCPView	13
Advanced Static Analysis.....	13
Cutter	13
Advanced Dynamic Analysis.....	15
x32dbg.....	15
Indicators of Compromise	17
Network Indicators	17
Host-based Indicators	17
Appendices	18
A. Yara Rules	18
B. Callback URLs	18
C. Hash Signatures.....	18

Executive Summary

SHA256 hash	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C
SHA-1 hash	e889544aff85ffaf8b0d0da705105dee7c97fe26
MD5 hash	db349b97c37d22f5ea1d1841e3c89eb4

WannaCry is ransomware, which is a type of cryptographic self-propagating malicious software, first publicly identified on May 12th, 2017. The WannaCry ransomware propagates through the network, impacting computers running Microsoft Windows. Files are encrypted and the impacted victims are extorted for a BitCoin Ransom.

WannaCry, a x32 bit executable, is compiled in C++ and exploits a vulnerability in Windows and the Server Message Block (SMBv1) protocol. There are multiple stages to the WannaCry malware beginning with a dropper which unpacks a malicious payload. During the execution of WannaCry, a ping to the “Killswitch” URL is sent, upon successful ping, no further execution occurs. On the other hand, an unsuccessful ping results in execution and infection.

Signs of infection are:

- “.WNCRY” appended file extensions
- A new desktop background displaying text
- An executable with a GUI for decryption and bitcoin address
- A ransom note “.readme” file

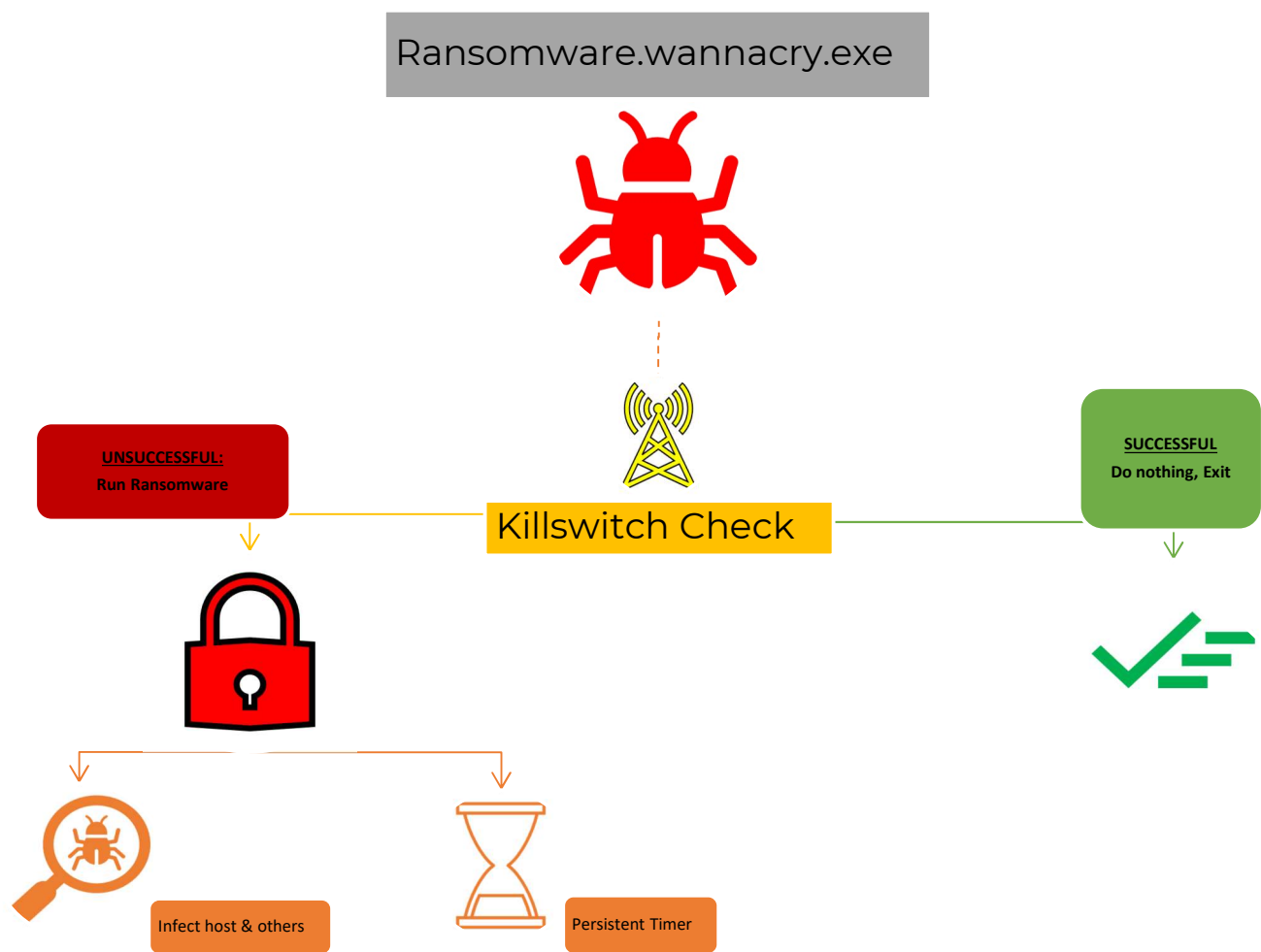
Indicators, YARA rules and signatures are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

Phase 1: The first phase consists of a dropper attempting to reach out to the “Killswitch” URL.

- If the attempt to connect to the URL is successful, then the WannaCry malware halts and the program is terminated.
- If the attempt to connect to the URL is not successful, then the WannaCry Malware continues to encrypt the system.

Phase 2: During the second phase, “tasksche.exe” is extracted to infect and encrypt the hosts files. The malware creates and runs a service to maintain persistence, scans local networks, and tries to propagate to other victims.



Malware Composition

WannaCry consists of the following components:

File Name	SHA256 Hash
ransomware.wannacry.exe	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C
tasksche.exe	ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA
mssecsvc.exe	F8812F1DEB8001F3B7672B6FC85640ECB123BC2304B563728E6235CCBE782D85

ransomware.wannacry.exe

Originally named in this analysis as “lhdfgui.exe” The initial phase of the executable is utilized to detect if the Killswitch URL is accessible. If the Killswitch URL is successful the malware does not execute; otherwise, if the Killswitch URL is not detected, the remaining steps are unpacked.

mssecsvc.exe

Service created to exploit an SMB vulnerability and maintain persistence on the host.

tasksche.exe

A resource extracted to maintain persistence and proceeds to encrypt the host system. A random folder name in ProgramData is created to stage ransomware and further infect systems.

Basic Static Analysis

Analysis of Strings (FLOSS)

The static analysis below is a representation of the strings extracted from the executable utilizing **FLOSS**. The figures below are examples of extracted strings, and utilization of grep pattern match across specific search criteria. Results show readable strings matching on characters higher than 8.

```
C:\Users\asd\Desktop
λ floss -n 8 Ransomware.wannacry.exe > floss_8.txt
INFO: floss: extracting static strings...
\ analyzing program
```

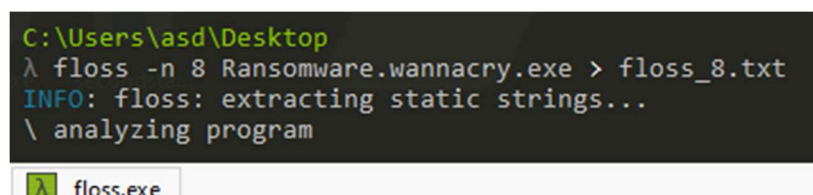


Fig 1: Floss tool matching on at least 8 characters in the file.

```
Microsoft Enhanced RSA and AES Cryptographic Provider
CryptGenKey
CryptDecrypt
CryptEncrypt
CryptDestroyKey
CryptImportKey
CryptAcquireContextA
cmd.exe /c "%s"
115p7UMMngo1pMvKpHijcRdfJNXj6LrLn
12+0YDPm...70NkMwF10c7AA8i...6SM...
```

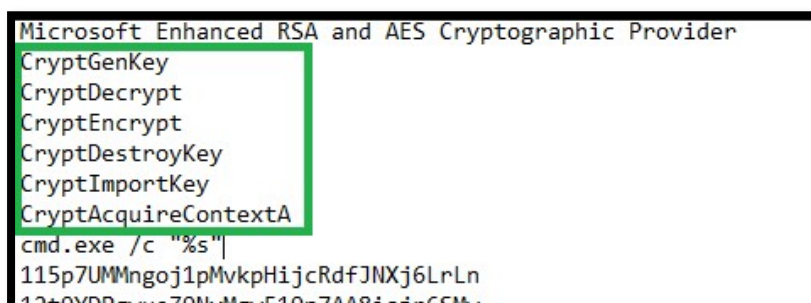


Fig 2: Cryptographic DLLs the binary uses for encryption.

```
tasksche.exe
CloseHandle
WriteFile
CreateFileA
CreateProcessA
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
```

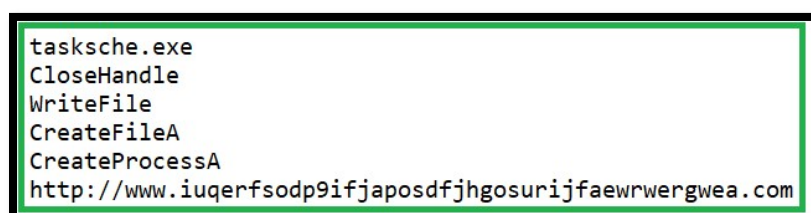


Fig 3: Task Scheduler, Process Creation, and Killswitch URL.

```
icacls . /grant Everyone:F /T /C /Q
attrib +h .
Wncry@2017
```

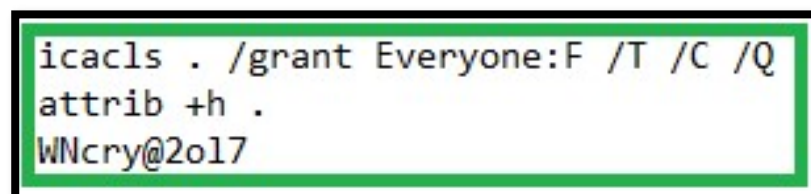


Fig 4: Applied DACLs, Hidden File Attribution, and password.

PeView & PeStudio

Assessing the executable through PeView and PeStudio provides header information, size, hashes, indicators, and bytes. The compiler stamp is shown as Sat, Nov 20th 09:03:08 2010 UTC. An area to check is the .rdata section. Additional files or executables are hidden and can be found in the resource section which are later used as part of malwares design.

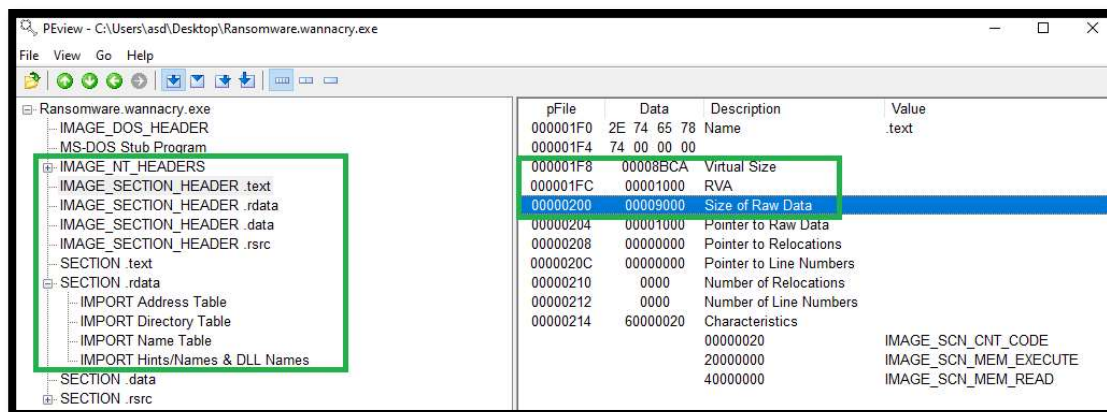


Fig. 5: Analyzing the size and .rdata sections determine packed malware.

Focusing on the resources section (.rsrc) of the file we can see a record labeled R 1831 with a high entropy score, suspicious size, and the magic bytes (4D 5A) referring to an executable. In this malware sample, the hard coded string found “WNcry@2017” is used to extract the resource identified.

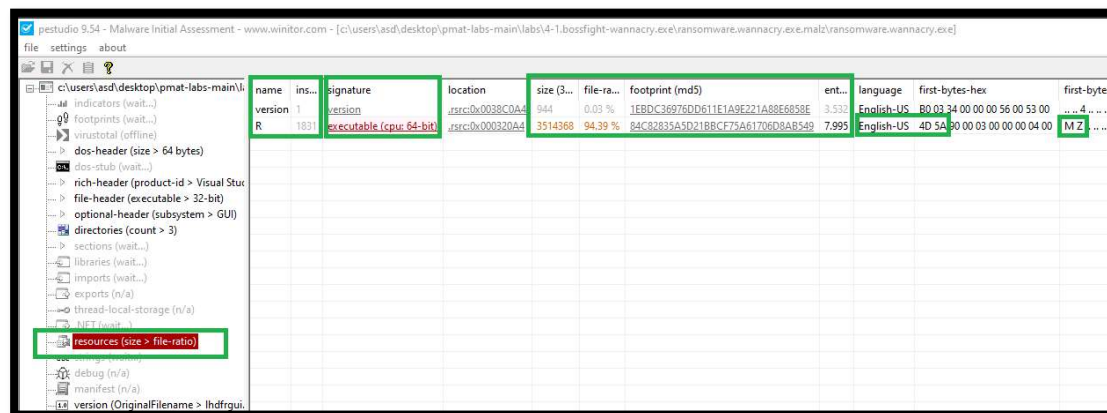


Fig. 6: Highlighting identifiers in the resource sections of the executable.



PeStudio provides us a look at the import address table (IAT) with information on the executable. By default, these imports are not malicious, but they infer the malware attempts to file creation, execution, and other tactics or techniques. The indicator strings also matches url previously identified during our FLOSS workflow.

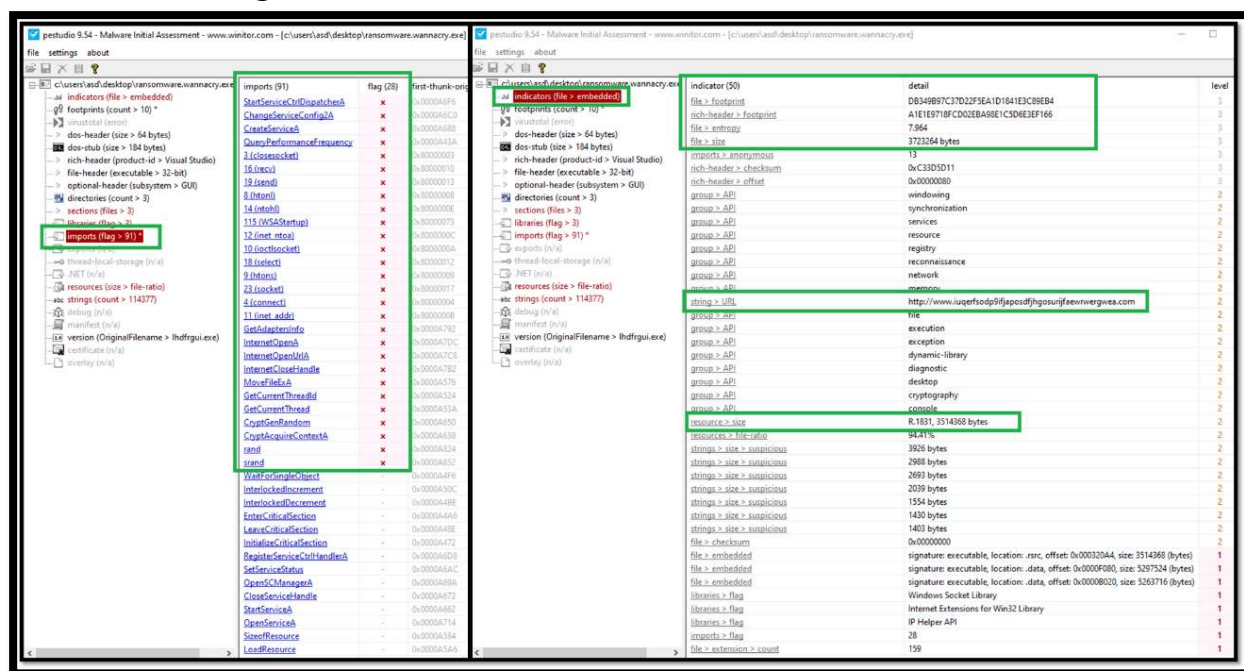


Fig. 7: Highlights from the IAT Table and the indicators section.

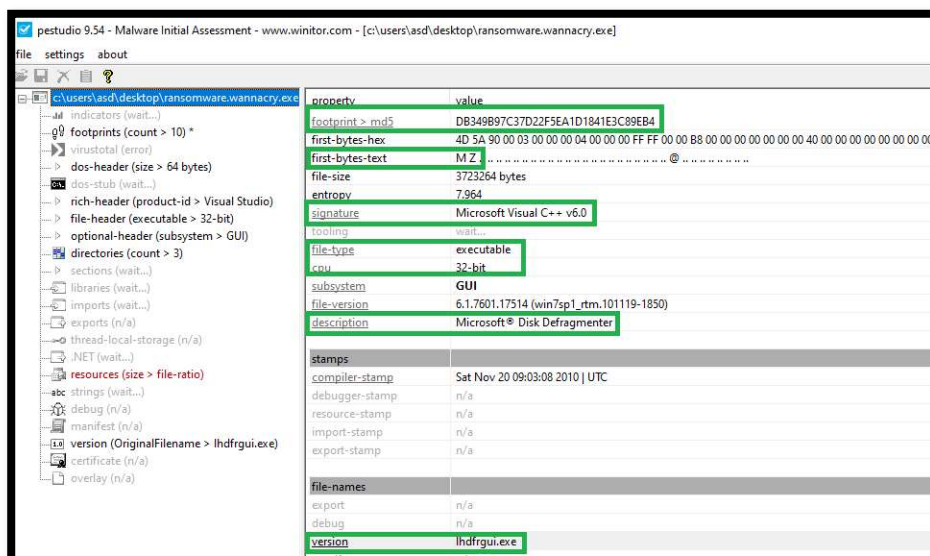


Fig. 8: General overview of Hashes, Bytes, original filename, and description.



Basic Dynamic Analysis

INETSIM: On (Killswitch Success)

The malware behaves differently when our INETSIM configured correctly. INETSIM provides a simulated successful response in our lab. We can also capture traffic requests in REMnux.

Setting up INETSIM and a Wireshark capture provides evidence of a host with the dropper attempting to reach any URLs. The HTTP GET request is resolved which get registered to the program. After knowing that the connection is successful the malware comes to a halt.

```
remnux@remnux:~$ inetsim
INetSim 1.3.2 (2020-05-19) by Matthias Eckert & Thomas Hungenberg
Using log directory: /var/log/inetsim/
Using data directory: /var/lib/inetsim/
Using report directory: /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 1355) ===
Session ID: 1355
Listening on: 10.0.0.4
Real Date/Time: 2023-10-03 22:11:41
Fake Date/Time: 2023-10-03 22:11:41 (Delta: 0 seconds)
Forking services...
* dns_53_tcp_udp - started (PID 1359)
* smtps_465_tcp - started (PID 1363)
* smtp_25_tcp - started (PID 1362)
* pop3s_995_tcp - started (PID 1365)
* pop3_110_tcp - started (PID 1364)
* ftp_21_tcp - started (PID 1366)
* ftps_990_tcp - started (PID 1367)
* https_443_tcp - started (PID 1361)
* http_80_tcp - started (PID 1360)
done.
Simulation running.

remnux@remnux:~$ wireshark
** (wireshark:1584) 22:16:27.380910 [Capture MESSAGE] -- Capture Start ...
** (wireshark:1584) 22:16:27.396274 [Capture MESSAGE] -- Capture started
** (wireshark:1584) 22:16:27.396357 [Capture MESSAGE] -- File: "/tmp/wireshark_enp0s3B1U6B2.pcapng"
```

Fig. 9: REMnux utilizing INETSIM and WireShark capture.

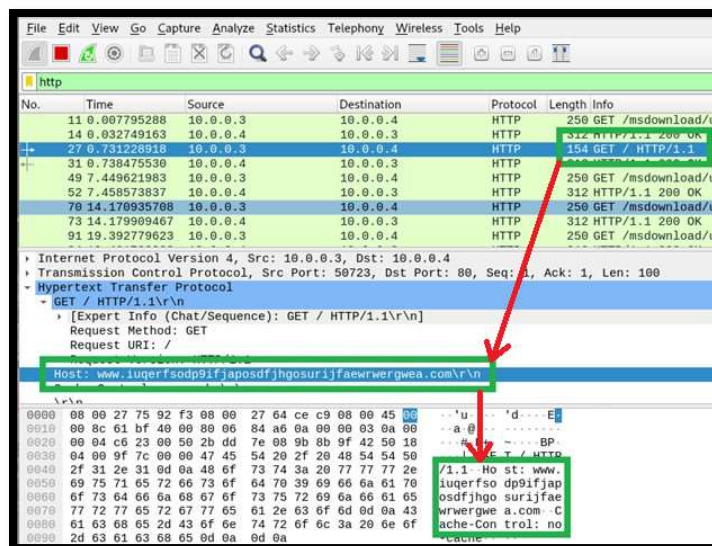


Fig. 10: WireShark Network capture displaying the Killswitch URL.

INETSIM: Off (Killswitch Unsuccessful)

When the malware is running without INETSIM (without the internet), and if the executable is running with **administrative privileges**, the files on the local endpoint will get encrypted. Noticeable symptoms will depict a wallpaper, forensic artifact file (cosmo.jpeg) is no longer usable, files will have a WNCRY extension appended, and a new executable called @WannaDecryptor.exe is added to the desktop.

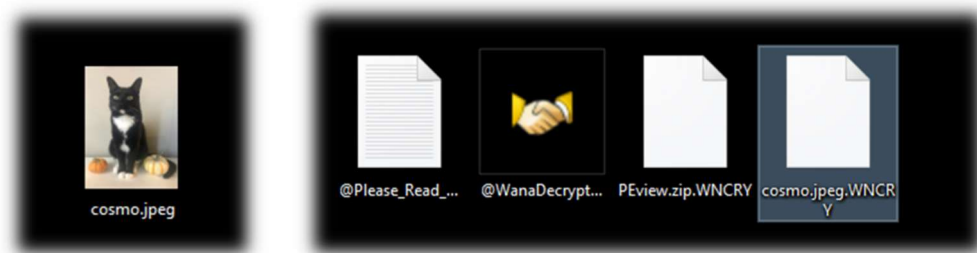


Fig. 11: Files appearing on the desktop and jpeg artifact modification.

An infected system will display a window with a countdown clock, instructions to pay a ransom amount, and a bitcoin address to pay the criminals in order to decrypt their files. A new background and decryptor executable will be persistently popping up along with a BitCoin Address: 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw for payment.



Fig. 12: An infected system with WannaCry.



ProcMon

Executing the Procmon tool can capture a lot of noise. Getting familiar with processes is key to see specific activity and scenarios. Inclusions and/or exclusions highlight relevant events.

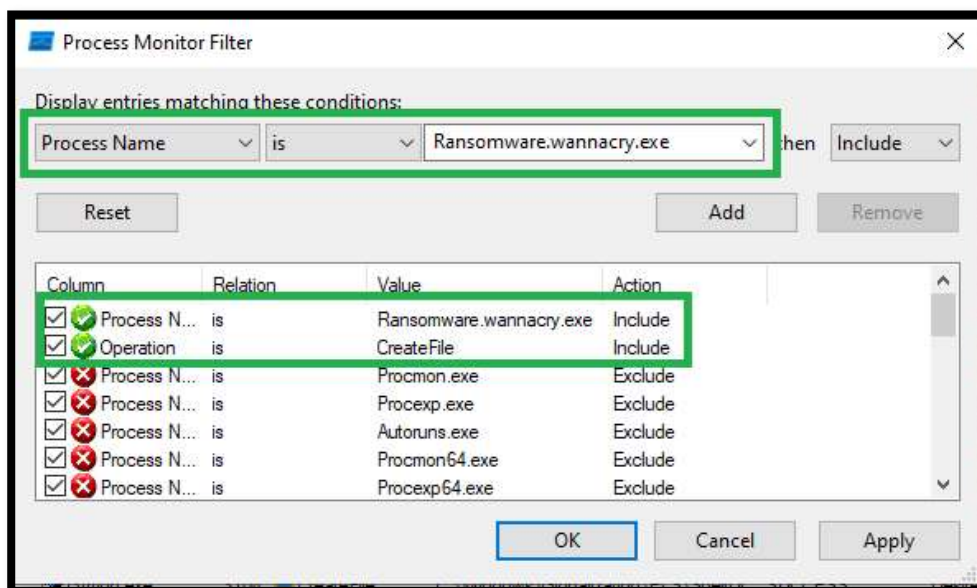


Fig. 13: ProcMon filters for “Operation” with “CreateFile” value and “ProcessName” with “Ransomware.wannacry.exe”.

Notice that the “Ransomware.wannacry.exe” created various processes. We can identify the tasksche.exe along with it's process ID as part of the events.

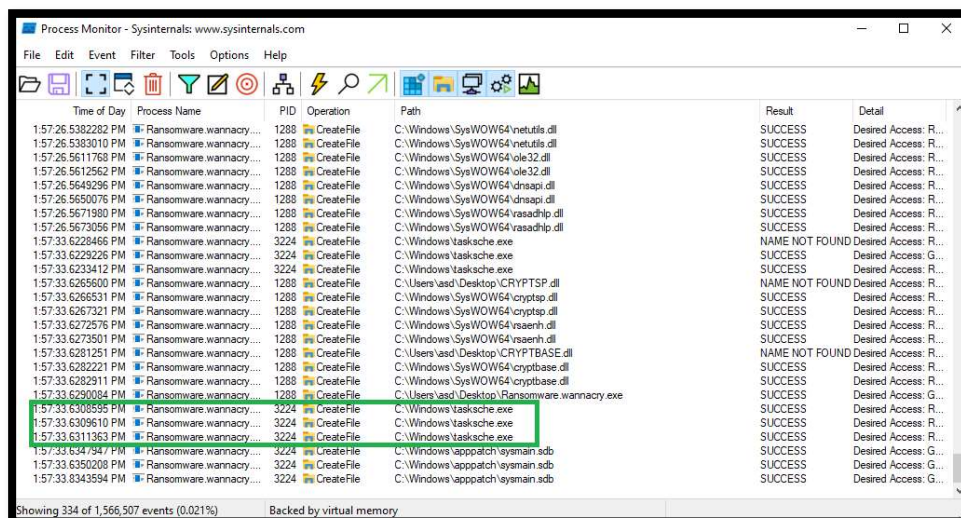


Fig. 14: Results of filter and discovering “tasksche.exe”.

From the identified process (tasksche.exe) we can pivot to the Process Tree and target the process or parent process IDs involved. Additionally, applying new filters in ProcMon such as process ID's, parent process ID's, registry changes or additions can lead us to new events.

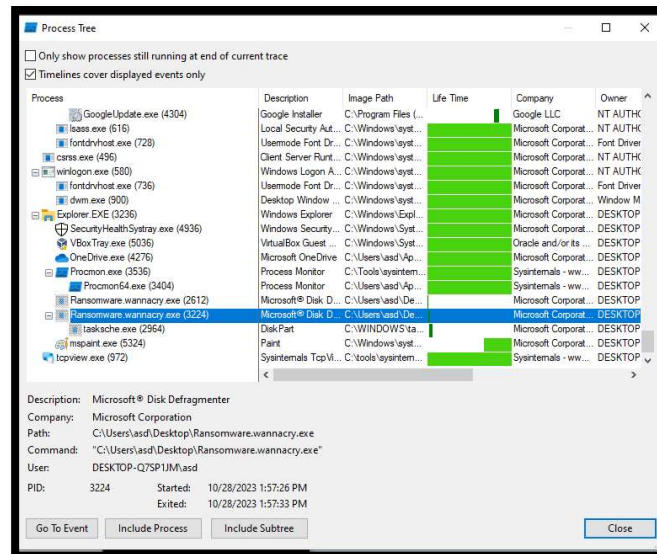


Fig. 15: Process tree depicting path, PID and PPIDs.

Searching for files created and process IDs a “hidden” folder in C:\Programdata can be found. This folder is associated to the “attrib +h” commands found in the strings analysis. There is a service with the same name “mxfszyvkt014” as the hidden folder created.

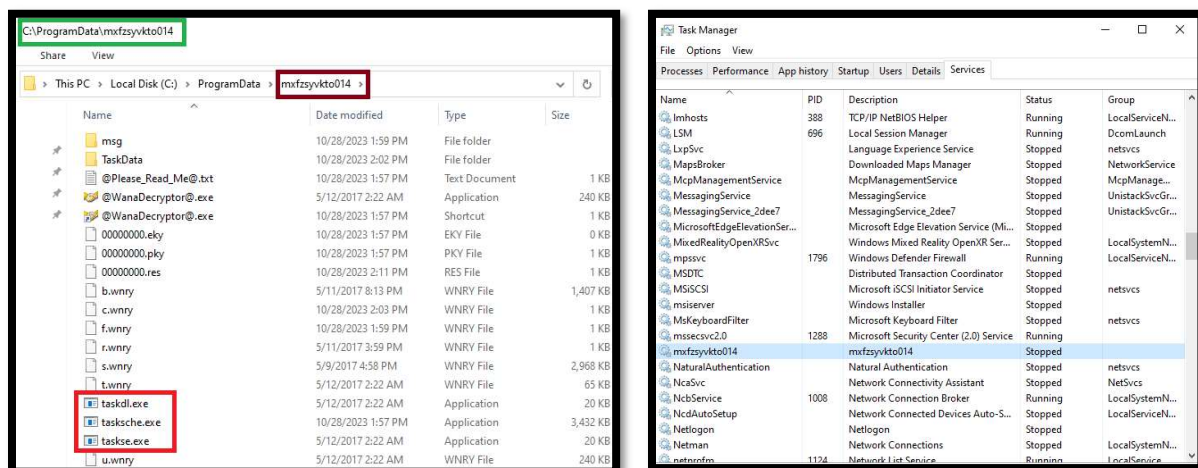


Fig. 16: Location of hidden resources revealed and attribution to Services.

TCPView

From TCPView we can see a flood of connections to port 445, which is utilized by the SMB protocol, targeting variety of addresses through mssecsvc2.0.

Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49716	10.0.0.35	445	10/28/2023 6:03:29 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49717	10.0.0.36	445	10/28/2023 6:03:29 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49718	10.0.0.37	445	10/28/2023 6:03:29 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49719	10.0.0.38	445	10/28/2023 6:03:29 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49720	10.0.0.39	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49721	10.0.0.40	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49722	10.0.0.41	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49724	10.0.0.42	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49726	10.0.0.43	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49727	10.0.0.44	445	10/28/2023 6:03:30 PM	mssecsvc2.0
Ransomware.wannacr...	3192	TCP	Syn Sent	10.0.0.3	49728	10.0.0.45	445	10/28/2023 6:03:30 PM	mssecsvc2.0

Fig. 17: TCPview capturing the binary sending requests.

Advanced Static Analysis

Cutter

Using Cutter, advanced static analysis is conducted to glance at meta-data which is useful to understand how the binary works. With an understanding that the malware checks for the Killswitch URL, we can look at the main function. The main function typically is the starting point of execution and gives more insight on program execution.

OVERVIEW

Info

File:	C:\Users\asd\Desktop\Ransomware.wannacr.exe	FD:	3
Format:	pe	Base addr:	0x00400000
Bits:	32	Virtual addr:	True
Class:	PE32	Canary:	False
Mode:	r-x	Crypto:	False
Size:	3.55 MB	IDX bit:	False
Type:	EXEC (Executable file)	PIC:	False
Language:	msvc	Static:	False
		Relro:	N/A

Architecture: x86
Machine: i386
OS: windows
Subsystem: Windows GUI
Stripped: False
Relocs: True
Endianness: LE
Compiled: Sat Nov 20 01:03:08 2010 UTC-8
Compiler: N/A

Hashes

MD5:	db349b97c37d22f5ea1d1841e3c89eb4
SHA1:	e889544af85f8fb0d0da705105dee7c97fe26
SHA256:	24d004b104d4d54034dbccf2a4b19a11f39008a575aa614ea04703480b1022c
CRC32:	9fbb1227
ENTROPY:	7.964259

Libraries

- kernel32.dll
- advapi32.dll
- ws2_32.dll
- msvcp60.dll
- iphlpapi.dll
- wininet.dll
- user32.dll

Analysis info

Functions:	83
X-Refs:	1831
Calls:	1707
Strings:	50189
Symbols:	91
Imports:	91
Analysis coverage:	33979 bytes
Code size:	36864 bytes
Coverage percent:	92.1739%

- flirt.alldiv
- flirt.allrem
- main
- sub.MSVCRT.dll_XcptFilter
- sub.MSVCRT.dll___dllonexit

Fig. 18 & 19: Overview of Cutter followed by loading into the main function.

Opening the main function in the graph view exposes the low-level assembly code and the highlighted Internet APIs: InternetOpenA, InternetOpenUrlA, and InternetCloseHandle. These APIs were noted during the analysis in PeStudio. InternetOpenUrlA is known to open a specified resource and store the value, which is later used in the stack.



```

[0x00408140]
int main(int argc, char **argv, char **envp);
var int32_t var_64h @ stack - 0x64
var int32_t var_50h @ stack - 0x50
var int32_t var_17h @ stack - 0x17
var int32_t var_13h @ stack - 0x13
var int32_t var_fh @ stack - 0xf
var int32_t var_3h @ stack - 0x3
var int32_t var_7h @ stack - 0x7
var int32_t var_3h @ stack - 0x3
var int32_t var_7h @ stack - 0x7
sub esp, 0x50
push esi
push edi
mov ecx, 0x1
mov esi, str_https_www_luberFood9171qoudf7hazur1f1awer-mea.com ; 0x431308
lea esi, [var_50h]
xor eax, eax
rep movsd dword es:[edi], dword ptr [esi]
movsb byte es:[edi], byte ptr [esi]
mov dword [var_17h], eax
mov dword [var_13h], eax
mov dword [var_fh], eax
mov dword [var_3h], eax
mov word [var_7h], ax
push eax
push eax
push eax
push 1
push eax
call dword [var_1h1_a1] ; 0x40b134
push 0
push 0x400000
push 0
lea ecx, [var_64h]
mov esi, eax
push 0
push ecx
call dword [InternetOpenA] ; 0x40b138
mov esi, eax
push esi
mov esi, 0x00000000 ; 0x40a13c
test edi, edi
jne 0x4081bc

[0x004081a7]
call esi
push 0
call esi
call fcn.00408090 ; fcn.00408090
pop edi
xor eax, eax
pop esi
add esp, 0x50
ret 0x10

[0x004081bc]
call esi
push edi
call esi
pop edi
xor eax, eax
pop esi
add esp, 0x50
ret 0x10
  
```

Fig. 20: Graph view of main function.

Towards the bottom of the main function is a test instruction that performs a bitwise AND on two operands (edi and edi). There is a flag, aka the Zero Flag or ZF which is set to 1 or 0. The ZF value is set to 1 (seen later). Right after test, is a JNE “Jump if not equal” condition.

- When the test of edi against edi is performed, if the results of the test set the ZF value to 0, the program moves to address 0x4081BC which results in exiting the program. (Successful Connection)
- If the results of the test are 1 the program proceeds and jumps to 0x004081a7 address resulting in the continuation of the malware. (Unsuccessful Connection)

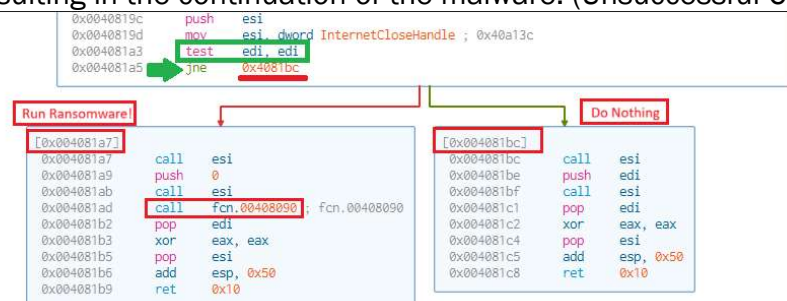


Fig. 21: Killswitch assembly code focus on TEST and JNE instruction.



Advanced Dynamic Analysis

x32dbg

Opening this malware sample in x64dbg says “use x32dbg to debug this file”. It’s worth noting that the architecture of this sample is x86 as detected by Cutter and other tools.

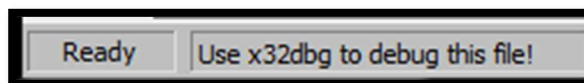


Fig. 22: x64dbg prompting to use x32dbg.

By opening the malware sample in x32dbg we have more control over the code instructions. Opening the malware sample successfully, highlights a lot of information. Immediately, the EIP is pointing to next step of assembly instructions. The ZF value is seen to be set to 1, which can change throughout the program’s instructions. Given our current conditions of the host, we will manipulate the ZF value to try and prevent the program from running further. Otherwise, when the ZF value is set to 1, the conditional jump execution will infect the host.

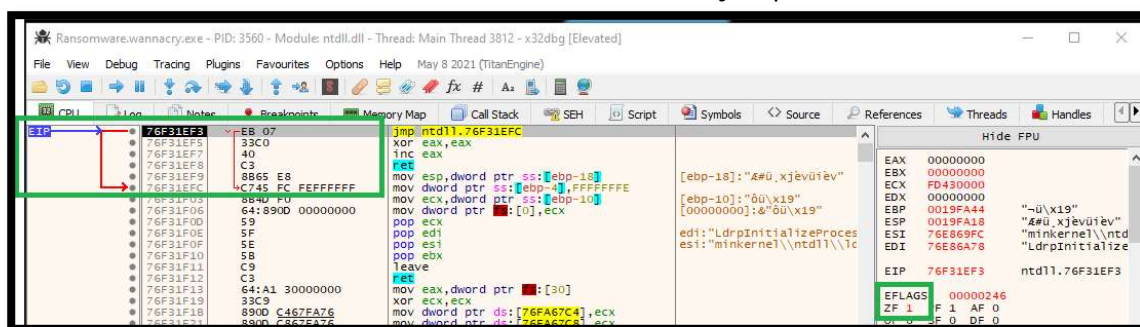


Fig. 23: Instruction pointer and the Zero Flag set to 1.

The infamous URL is continuously seen during analysis so we can use strings built into x32dbg and find the value yet again. The difference in this scenario is setting a breakpoint on the address found and stepping into each set of instructions to observe changes.

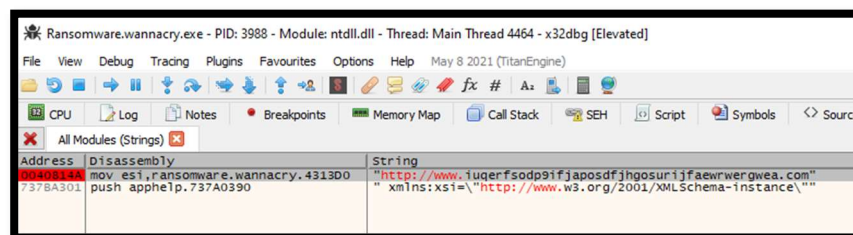


Fig. 24: Setting a breakpoint on the address 0x0040814A



Working with x32dbg, REMnux is not on running, which means INETSIM is off and a failed or unsuccessful connection to the URL would occur, leading to infection on the host.

Setting the breakpoint on the address 0x0040814A, allows us to step and pause into each instruction, following up to the internet API calls. Note that the EDI value stored is 0, which indicates nothing was returned when the internet APIs reach out to the suspicious URL. The ZF value is set to 1, and if the JNE instruction is called while ZF value is set to 1, a jump to address 0x0040817 occurs, followed by another function leading to infect the host.

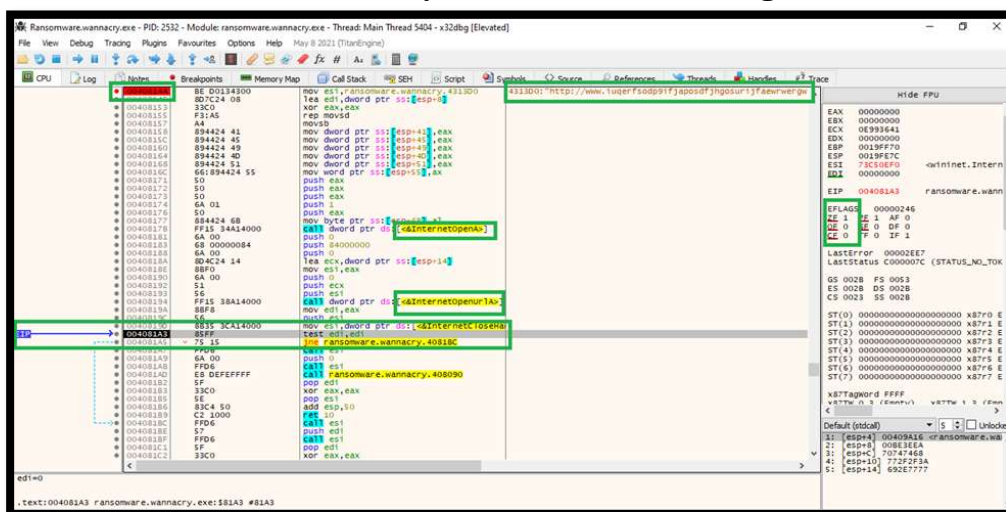


Fig. 25: Setting a breakpoint on the address 0x0040814A

On address 0x004081A5 where the JNE instruction occurs, we can interfere which the programs logic. By double clicking on the ZF value, the value is set to 0, altering the conditions and halting the program.

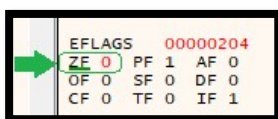


Fig. 26: Changing the ZF value from 1 to 0.

When stepping into the next instruction, we are brought to the memory address known as 0x4081BC. Continuing down this path will proceed to terminate and exit the program. No infection occurs.

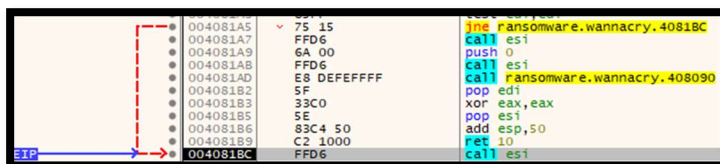


Fig. 27: Changing the ZF value leads to 0x4081BC address.

Indicators of Compromise

The following are samples of indicators observed on an infected host.

Network Indicators

hxxp://www[.]iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com

Host-based Indicators

FilePath:

%ProgramData%\<random directory name>\tasksche.exe

Files created:

00000000.eky
00000000.pky
00000000.res
@Please_Read_Me@.txt
@WanaDecryptor@.exe
b.wnry
c.wnry
f.wnry
r.wnry
s.wnry
t.wnry
taskdl.exe
taskse.exe
u.wnry
@WanaDecryptor@.bmp

Registry:

HKLM\SOFTWARE\Wow6432Node\WanaCrypt0r\wd



Appendices

A. Yara Rules

```
rule Yara_wannacry_ransomware_custom {  
  
    meta:  
        last_updated = "2023-10-02"  
        author = "LF - Malware Analysis"  
        description = " Basic Detection of WannaCry Ransomware on endpoint disk"  
  
    strings:  
        // Strings in WannaCry  
        $string1 = "iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea" ascii  
        $string2 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii  
        $string3 = "WANNACRY"  
        $string3 = "WNcry@2ol7"  
        // Strings in WannaCry  
        $PE_magic_byte = "MZ"  
  
    condition:  
        // Conditions that must be met to identify the binary  
        $PE_magic_byte at 0 and  
        any of ($string*)  
}
```

B. Callback URLs

Domain	Port
hxxp[:]//[i]uqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com	80

C. Hash Signatures

ransomware.wannacry.exe

SHA256 hash	24D004A104D4D54034DBCFFC2A4B19A11F39008A575AA614EA04703480B1022C
SHA-1 hash	e889544aff85ffaf8b0d0da705105dee7c97fe26
MD5 hash	db349b97c37d22f5ea1d1841e3c89eb4

tasksche.exe

SHA256 hash	ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BABE8E080E41AA
SHA-1 hash	5FF465AFAABCBF0150D1A3AB2C2E74F3A4426467
MD5 hash	84C82835A5D21BBCF75A61706D8AB549

msseccsvc.exe

SHA256 hash	F8812F1DEB8001F3B7672B6FC85640ECB123BC2304B563728E6235CCBE782D85
SHA-1 hash	51E4307093F8CA8854359C0AC882DDCA427A813C
MD5 hash	F107A717F76F4F910AE9CB4DC5290594