Malware Analysis Report

WannaCry

Jan 2023 | Alessio Ragazzi

Table of Contents

Introduction

The following report is not meant to be exhaustive. Extensive research has been conducted on WannaCry by teams of experts from all over the world. I've reserved a section at the end of this text to mention some of the best analyses that can be found online today.
My ultimate goal was to demonstrate the methodology acquired upon completion of the TCM Security Practical Malware Analysis & Triage.

Prior to attempting the analysis, I had a very superficial knowledge of the malware. I was well aware of the devastating effects and its crypto-ransom capabilities.
I performed basic and advanced static and dynamic analysis of the sample, including disassembly and debugging limited to the initial dropper. I've collected host-based and network-based IoCs, and I have revealed the malware's major capabilities.

I was able to observe, analyze and report the *KillSwitch* mechanism, the *EternalBlue* exploit, and the role of *tasksche.exe.*
I wasn't able to observe directly some of the components. as well as part of the behavior, such as: Backdoor delivery, Persistence mechanism, and C2 communication.

The following report will only contain the result of my personal analysis of the sample. For further information, please refer to the resources at the end of the text.

Executive Summary

SHA256 hash	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

SHA1 hash	3b669778698972c402f7c149fc844d0ddb3a00e8

MD5 hash	d724d8cc6420f06e8a48752f0da11c66

WannaCry is a crypto-ransomware worm that spread rapidly through across a number of computer networks in May of 2017. After infecting a Windows computer (32bit/64bit), it encrypts files on the PC's hard drive, making them impossible for users to access, demanding a fee of either $300 or $600 worth of bitcoins to an address specified in the instructions displayed after infection.

Utilizing one of the core components of the Microsoft Operating System and a known exploit, it's able to propagate through the network. An estimated of around 230.000 computers being infected have been confirmed up-to-date.

Symptoms of infection include encrypted files, a custom background, a directory containing multiple files dropped in the C:\Windows\ProgramData, and the highly recognizable *Wana Decrypt0r 2.0* program window, containing a timer and information on how to proceed.

Despite security patches and decryption keys having been released, WannaCry is still active. In-depth research and extensive documentation can be found today in this regard.

YARA signature rules are attached in Appendix A, along with the malware's IoCs.

High-Level Technical Summary

The WannaCry ransomware is composed of multiple components. An initial dropper contains the encrypter, named *tasksche.exe*, as an embedded resource; the encrypter component contains a decryption application called *Wana Decrypt0r 2.0*, a password-protected zip containing a copy of Tor, and several individual files with configuration information and encryption keys.

First phase - The "KillSwitch"
Once the payload has been delivered, the dropper performs the initial function. The so-called, KillSwitch. The mechanism by which it can terminate its execution if it's able to establish a connection to a hard-coded domain.
The domain requested by the initial dropper is
http://www[.]iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com.

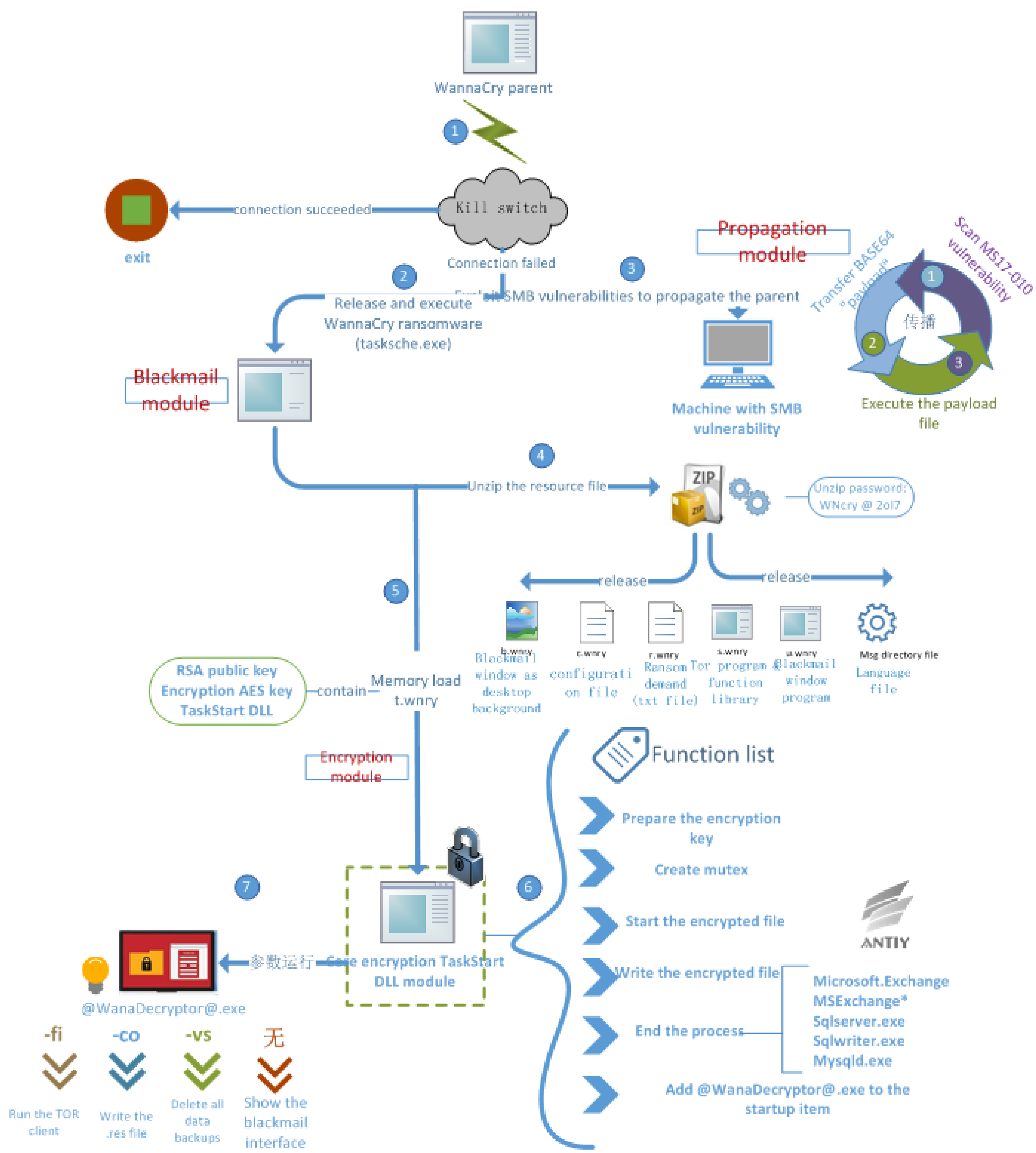
Second phase - The EternalBlue exploit
If the connection fails, the dropper attempts to create a service named "mssecsvc2.0" with the DisplayName "Microsoft Security Center (2.0) Service". WannaCry utilizes windows services to spread, exploiting a vulnerability in the SMB V1.0 protocol via port 445.

Main phase - *Tasksche.exe*
The dropper then extracts the encrypter binary from its resource R/1831, writes it to the hardcoded filename %WinDir%\tasksche.exe, and then executes it. As shown in the static and dynamic analysis section, *tasksche.exe* is responsible for the encryption of the files in the system, and for launching the *WanaDecrypt0r 2.0* program. The encrypter doesn't encrypt executable files such as .exe and .dll to avoid system interruption.

Double Pulsar, C2 communication and Persistence mechanism
WannaCry performs further actions, such as delivering the *Double Pulsar* backdoor after infecting a new host, establishing persistence by creating a new Registry key, and contacting a C2 server via the TOR service. Unfortunately, I wasn't able to detect these actions, and therefore I haven't reported the related documentation. Interestingly, the malware's authors have chosen to implement very few Anti-analysis techniques, allowing disassembly and debugging.

•

Execution Process of WannaCry



Malware Composition

The following components were analyzed or observed by me directly:

File Name	SHA256 Hash
mssecsvc.exe	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
tasksche.exe	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
taskse.exe	2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d
@WanaDecryptor@.exe	b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25

In addition, the following components can also be observed:

File Name	Description
taskdl.exe	Binary used for deleting temporary files
r.wnry	Shows the ransom message
s.wnry	Contains Tor executable
msg/	Contain Language files
f.wnry	N/A
b.wnry	N/A

Analysis Environment

Host	Description	IP address	DNS	Use
VM1	Windows 10 - Flare VM distribution	10.0.0.3	10.0.04	Main Analysis Environment (Infected Host)
VM2	Linux - Remnux distribution	10.0.0.4	N.A.	Internet simulation and Packet Capture

Host	Tool	Description
VM1	Floss	Uses advanced static analysis techniques to automatically deobfuscate strings from malware binaries.
	Pestudio	Spots artifacts of executable files.
	PEview	Views the structure and content of 32-bit Portable Executable (PE) and Component Object File Format (COFF) files.
	Cutter	Disassemble.r
	Capa	Performs reverse engineering to figure out what a program does. It includes different frameworks, including MITRE ATT&CK.
	X64dbg	An open-source x64/x32 debugger for windows.
	TCPview	Shows detailed listings of all TCP and UDP endpoints on the system.
	Process Hacker	Monitors system resources, debug software, and detect malware.
	Procmon	Advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity.
VM2	InetSim	Internet service simulation suite.
	Wireshark	Packet capture.

Basic Static Analysis

WannaCry binary lab's name: Ransomware.wannacry.exe

Before attempting the detonation, I used floss to try to pull some interesting strings out of the binary. I made use of ***floss*** for this purpose. It is possible to observe numerous APIs, .exe, a domain, reference to Windows service, and system paths.

```
*Untitled - Notepad
File Edit Format View Help
CreateServiceA
OpenSCManagerA
SetServiceStatus
ChangeServiceConfig2A
RegisterServiceCtrlHandlerA
StartServiceCtrlDispatcherA
OpenServiceA
InternetCloseHandle
InternetOpenUrlA
InternetOpenA
CloseHandle
WriteFile
CreateFileA
SetCurrentDirectoryA
GetCurrentDirectoryA
RegCloseKey
RegQueryValueExA
RegSetValueExA
RegCreateKeyW
```

```
*Untitled - Notepad
File Edit Format View Help
InitializeCriticalSection
LeaveCriticalSection
EnterCriticalSection
InterlockedDecrement
CloseHandle
TerminateThread
GetCurrentThreadId
GetCurrentThread
ReadFile
GetFileSize
CreateFileA
MoveFileExA
FindResourceA
GetProcAddress
GetModuleHandleW
ExitProcess
GetModuleFileNameA
CryptAcquireContextA
CryptGenRandom
```

```
*Untitled - Notepad
File Edit Format View Help
RegCreateKeyW
CryptReleaseContext
WININET.dll
MSVCRT.dll
C:\%s\%s
!This program cannot be run in DOS mode.
/4%D/4%D/4%D4
D,4%D/4$D
D.4%DRich/4%D
mssecsvc2.0
Microsoft Security Center (2.0) Service
%s -m security
C:\%s\qeriuwjhrf
C:\%s\%s
tasksche.exe
cmd.exe /c "%s"
taskdl.exe
taskse.exe
mssecsvc.exe
```

```
tasksche.exe
cmd.exe /c "%s"
taskdl.exe
taskse.exe
mssecsvc.exe
http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
icacls . /grant Everyone:F /T /C /Q
attrib +h .
```

From the first two snapshots, we can see some interesting APIs:

- ReadFile, CreateFile, WriteFile (Note that these APIs are common in non-malicious binaries too)
- CryptoGenRandom, CryptReleasContext
- OpenSCManagerA
- OpenServiceA, SetServiceStatus, StartServiceCtrlDispatcherA
- RegSetValueA, RegCreateKeyW
- InternetOpenA, InternetOpenUrlA

We can see the following executable: **tasksche.exe, cmd.exe, taskdl.exe, taskse.exe, mssecsvc.exe**

We can also see other strings of interest such as a domain, a system path using token values, more references to windows services, and the string containing ***icacls***.

I have run the tool ***capa*** to see if there's any match in a attempt to understand the malware behavior prior the detonation.
We can start to correlate this output with the strings that we just pulled.
We will expect **service execution** (including Persistence), **reconnaissance activity, C2 and HTTP communication, file creation, cryptography** (fig. 3), **and embedded executables** (fig. 2).

```
C:\Users\Gallianico\Desktop
λ capa Ransomware.wannacry.exe
loading : 100%|██████████████████████████████████████| 485/485 [00:01<00:00, 265.27     rules/s]
matching: 100%|████████████████████████████████████████| 87/87 [00:08<00:00, 10.71 functions/s]
+------------------------+------------------------------------------------------------------------------------+
| md5                    | db349b97c37d22f5ea1d1841e3c89eb4                                                   |
| sha1                   | e889544aff85ffaf8b0d0da705105dee7c97fe26                                           |
| sha256                 | 24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c                   |
| path                   | Ransomware.wannacry.exe                                                            |
+------------------------+------------------------------------------------------------------------------------+

+------------------------+------------------------------------------------------------------------------------+
| ATT&CK Tactic          | ATT&CK Technique                                                                   |
|------------------------+------------------------------------------------------------------------------------|
| DEFENSE EVASION        | Obfuscated Files or Information::Indicator Removal from Tools [T1027.005]          |
| DISCOVERY              | File and Directory Discovery [T1083]                                               |
|                        | System Information Discovery [T1082]                                               |
|                        | System Network Configuration Discovery [T1016]                                     |
| EXECUTION              | Shared Modules [T1129]                                                             |
|                        | System Services::Service Execution [T1569.002]                                     |
| PERSISTENCE            | Create or Modify System Process::Windows Service [T1543.003]                       |
+------------------------+------------------------------------------------------------------------------------+

+-----------------------------+-------------------------------------------------------------------------------+
| MBC Objective               | MBC Behavior                                                                  |
|-----------------------------+-------------------------------------------------------------------------------|
| ANTI-BEHAVIORAL ANALYSIS    | Debugger Detection::Timing/Delay Check GetTickCount [B0001.032]               |
|                             | Debugger Detection::Timing/Delay Check QueryPerformanceCounter [B0001.033]    |
|                             | Execution Guardrails::Runs as Service [E1480.m07]                             |
| ANTI-STATIC ANALYSIS        | Disassembler Evasion::Argument Obfuscation [B0012.001]                        |
| 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]                                      |
```

Fig. 1 Capa output

```
| contain an embedded PE file                      | executable/subfile/pe
| get file size                                    | host-interaction/file-system/meta
| move file                                        | host-interaction/file-system/move
| read file                                        | host-interaction/file-system/read
| get number of processors                         | host-interaction/hardware/cpu
| get networking interfaces                        | host-interaction/network/interface
| terminate process                                | host-interaction/process/terminate
| run as service                                   | host-interaction/service
| create service                                   | host-interaction/service/create
| modify service                                   | host-interaction/service/modify
| start service                                    | host-interaction/service/start
| create thread (4 matches)                        | host-interaction/thread/create
| terminate thread                                 | host-interaction/thread/terminate
| link function at runtime                         | linking/runtime-linking
| linked against ZLIB                              | linking/static/zlib
| inspect section memory permissions               | load-code/pe
| parse PE exports                                 | load-code/pe
| parse PE header                                  | load-code/pe
| persist via Windows service                      | persistence/service
+--------------------------------------------------+------------------------------------
```

Fig .2 Capa output

```
|                          | 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]
| EXECUTION                | Install Additional Program [B0023]
| FILE SYSTEM              | Read File [C0051]
| PROCESS                  | Create Thread [C0038]
|                          | Terminate Process [C0018]
|                          | Terminate Thread [C0039]
+--------------------------+------------------------------------------------------------

+-----------------------------------------------------+---------------------------------------------
| CAPABILITY                                          | NAMESPACE
|-----------------------------------------------------+---------------------------------------------
| check for time delay via GetTickCount               | anti-analysis/anti-debugging/debugger-detection
| check for time delay via QueryPerformanceCounter    | anti-analysis/anti-debugging/debugger-detection
| contain obfuscated stackstrings                     | anti-analysis/obfuscation/string/stackstring
| receive data (5 matches)                            | communication
| send data (5 matches)                               | communication
| connect to URL                                      | communication/http/client
| get socket status                                   | communication/socket
| initialize Winsock library                          | communication/socket
| set socket configuration                            | communication/socket
| create UDP socket (4 matches)                       | communication/socket/udp/send
| act as TCP client                                   | communication/tcp/client
| generate random numbers via WinAPI                  | data-manipulation/prng
| contain a resource (.rsrc) section                  | executable/pe/section/rsrc
| extract resource via kernel32 functions             | executable/resource
| contain an embedded PE file                         | executable/subfile/pe
```

Fig. 3 Capa output

I've opened the file making use of ***PEview*** and ***PEstudio*** to try to grab some artifacts, understand the architecture, observe blacklisted strings and imports. ***PEstudio*** confirms that we are dealing with a ***32bit executable*** (Fig. 4). We can also see the **original filename** (Fig. 5), and the presence of the ***Rich header*** (Fig. 4).

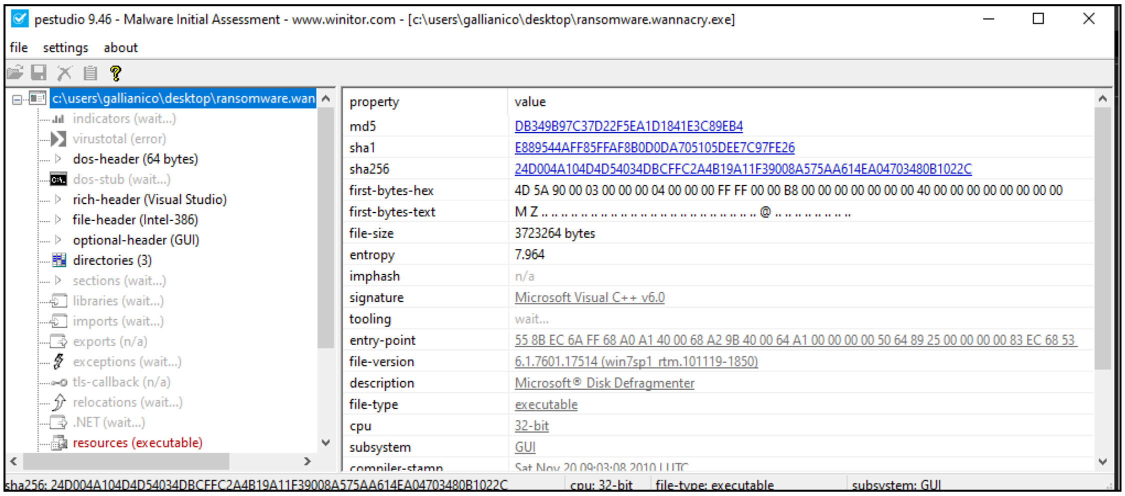


Fig. 4 PEstudio output

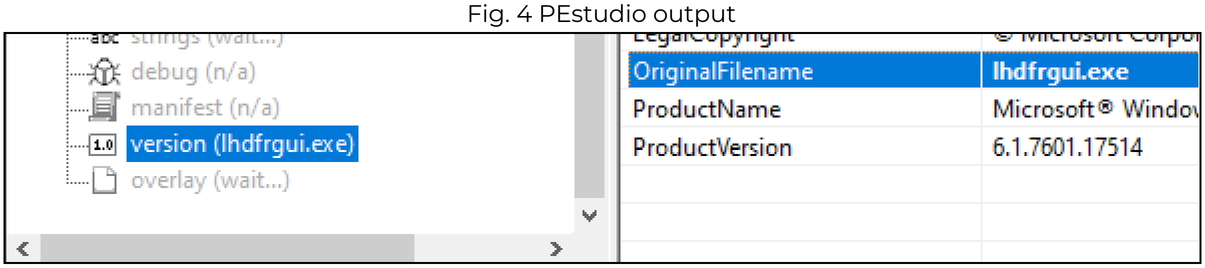
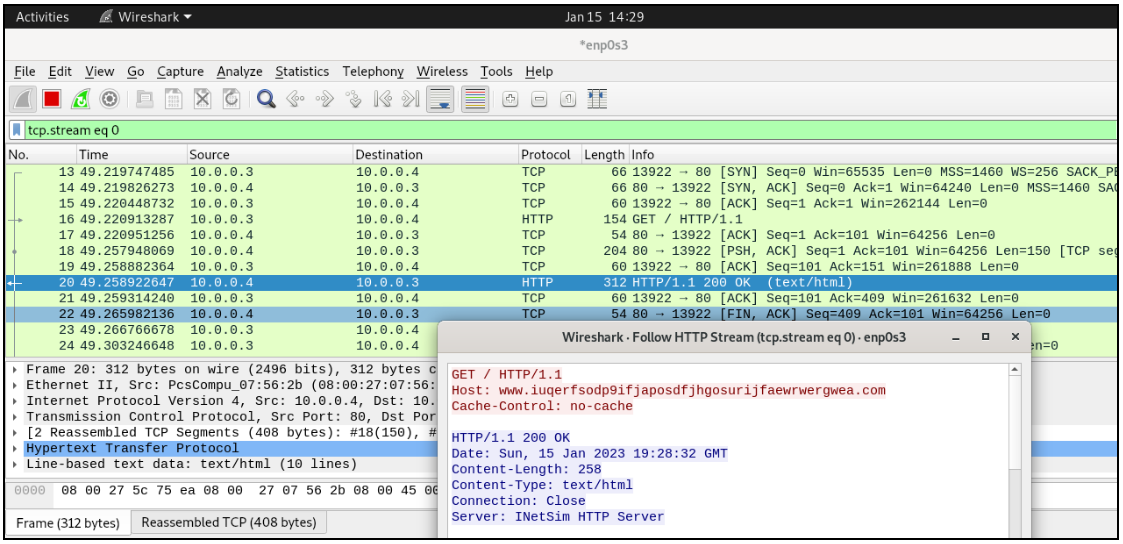


Fig. 5 PEstudio output

Basic Dynamic Analysis

First detonation - Symptoms observation
VM1 : No tools or application running
VM2: Internet Simulation service running - Wireshark running

No symptoms were observed. If we check the packet capture, we can see a DNS request for a type A domain, and the subsequently HTTP/TCP connection established (IoC 1). I'll come back to the ***KillSwitch*** later during disassembly and debugging.



IoC 1. Wireshark packet capture

Second detonation - Symptoms observation
VM1: No tools or application running
VM2: Internet Simulation service NOT running

If we terminate the internet simulation activity on VM2 and we detonate the malware one more time, we can see the first symptoms. The encrypter is deployed and it encrypts the files on the system, appending the .WCRY extension, except the .exe and .dll binaries that will remain unchanged and functional. We can see the @WanaDecrypt0r2.0@ program appearing on the desktop (IoC 2), a custom background image taking the place of our original background, and the notorious window popping up with the instructions to follow if we wish our files to be decrypted.

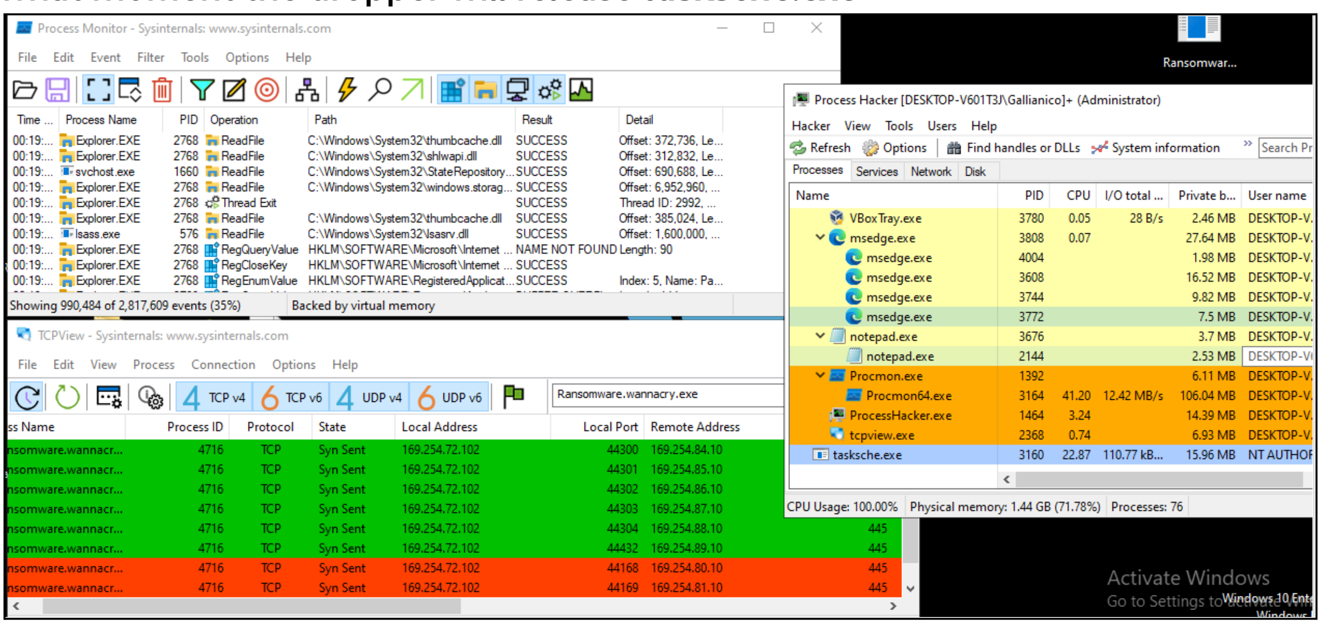
IoC 2. WanaDecrypt0r window appearing on desktop

Lucky for us, we won't have to pay any ransom at this time! At this point I've brought back the virtual machines to a clean state for another detonation, deploying more tool for basic dynamic analysis.

Third detonation - Analysis
VM1 : Procmon, Process Hacker, TCPview
VM2: Internet Simulation service not running

As shown in the picture below, we can see that the first indicators that take our attention are the attempted TCP connections on remote **port 445** (Bottom-left - IoC 3), and the new process called ***tasksche.exe*** (Right - IoC 3) taking the place of our original binary's name (Ransomware.wannacry.exe) in the live process tree.

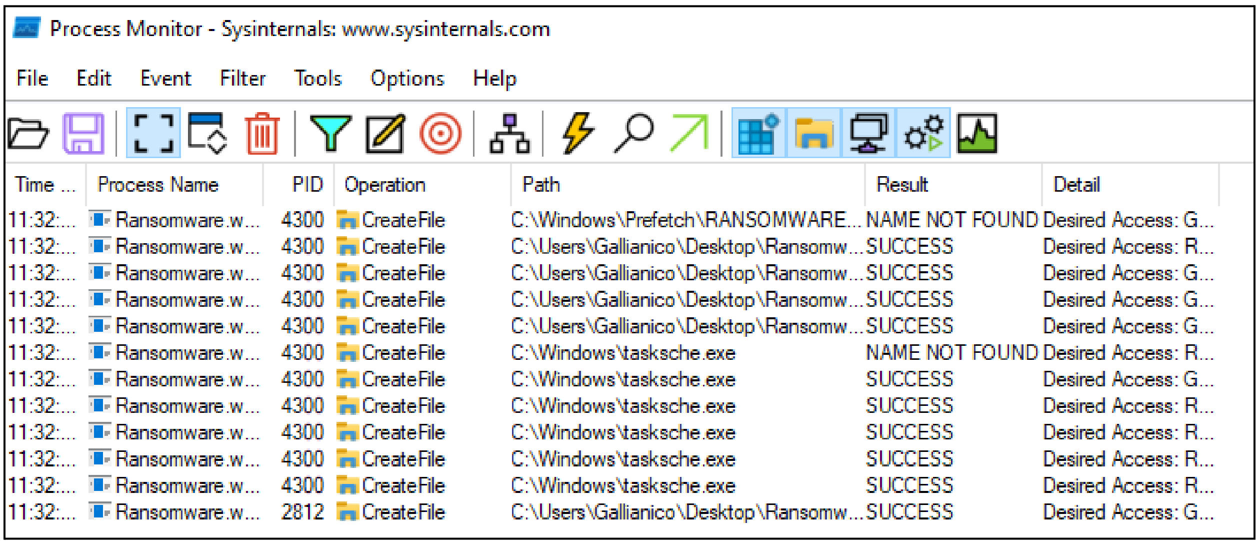
As we'll see later during the disassembly phase, the malware is searching available hosts on the network via open **SMB port 445**. We'll also see in what moment the dropper will release ***tasksche.exe***



IoC 3. A snapshot at the initial detonation. From Top-left to right: Procmon, Process Hacker, TCPview

If we take a look at **procmon** (IoC 4), we can see the file creation of ***tasksche.exe***, by our original dropper, ***Ransomware.wannacry.exe***. For this view, I've applied 3 sets of filters:

- Process name is ***Ransomware.wannacry.exe***
- Operation is **Create file**
- Path contains .exe



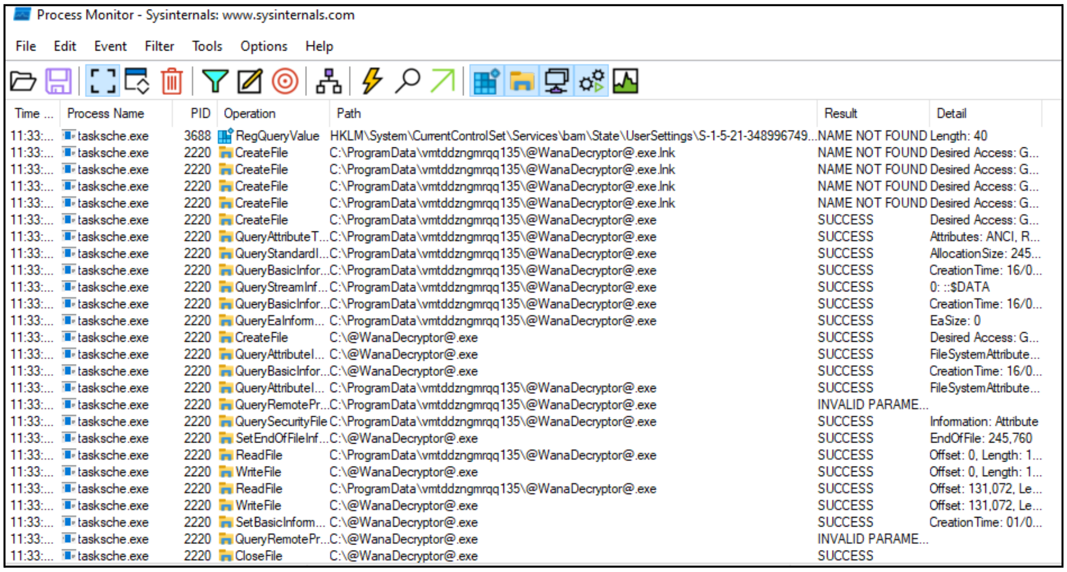
Time ...	Process Name	PID	Operation	Path	Result	Detail
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\Prefetch\RANSOMWARE...	NAME NOT FOUND	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	NAME NOT FOUND	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	2812	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...

IoC 4. A snapshot of procmon after initial detonation

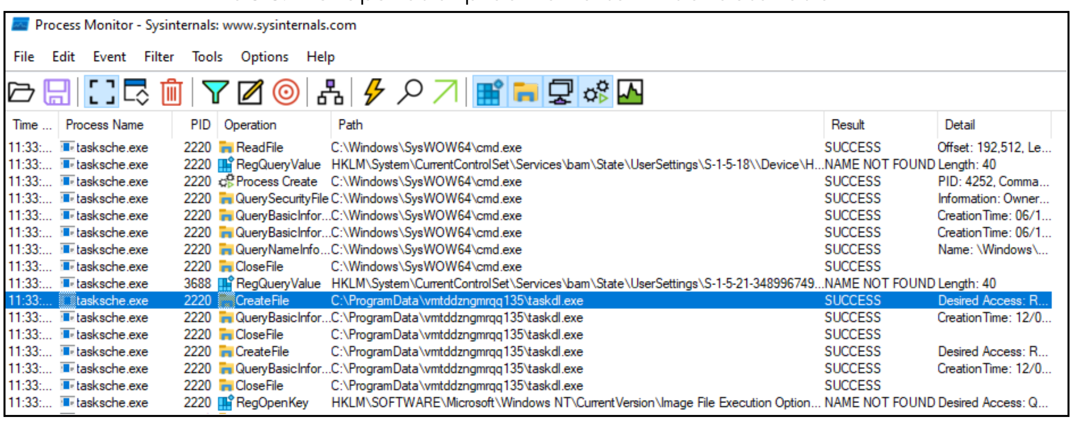
If we change our set of filters, we can see further activity coming from ***tasksche.exe***. The new set of filters is:

- Process name is ***tasksche.exe***
- Operation is **Create file**
- Path contains .exe

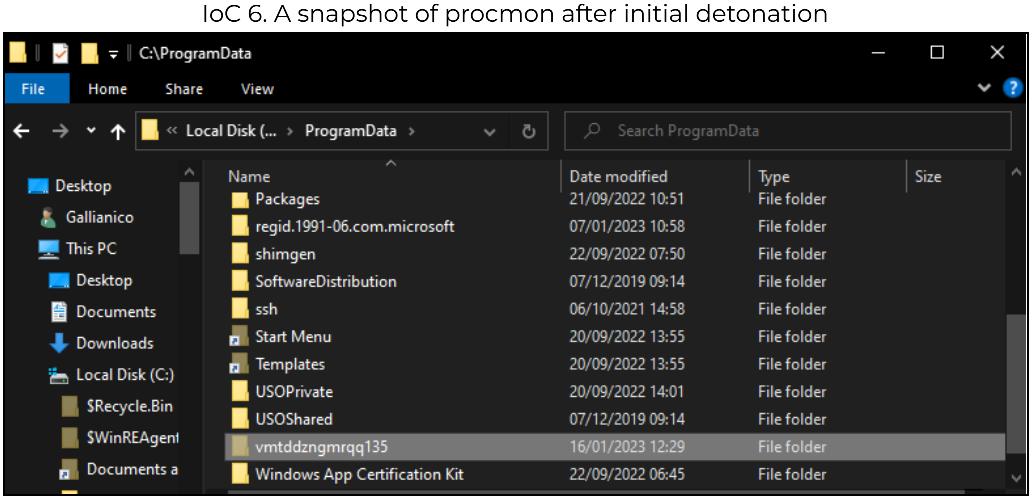
Two new files have been created: ***@WanaDecryptor2.0@.exe*** and ***taskdl.exe*** (IoC 5). We can also see another executable been spawned, ***cmd.exe*** (IoC 6), and a new directory with an obfuscated name under the ***/ProgramData*** directory (IoC 7).



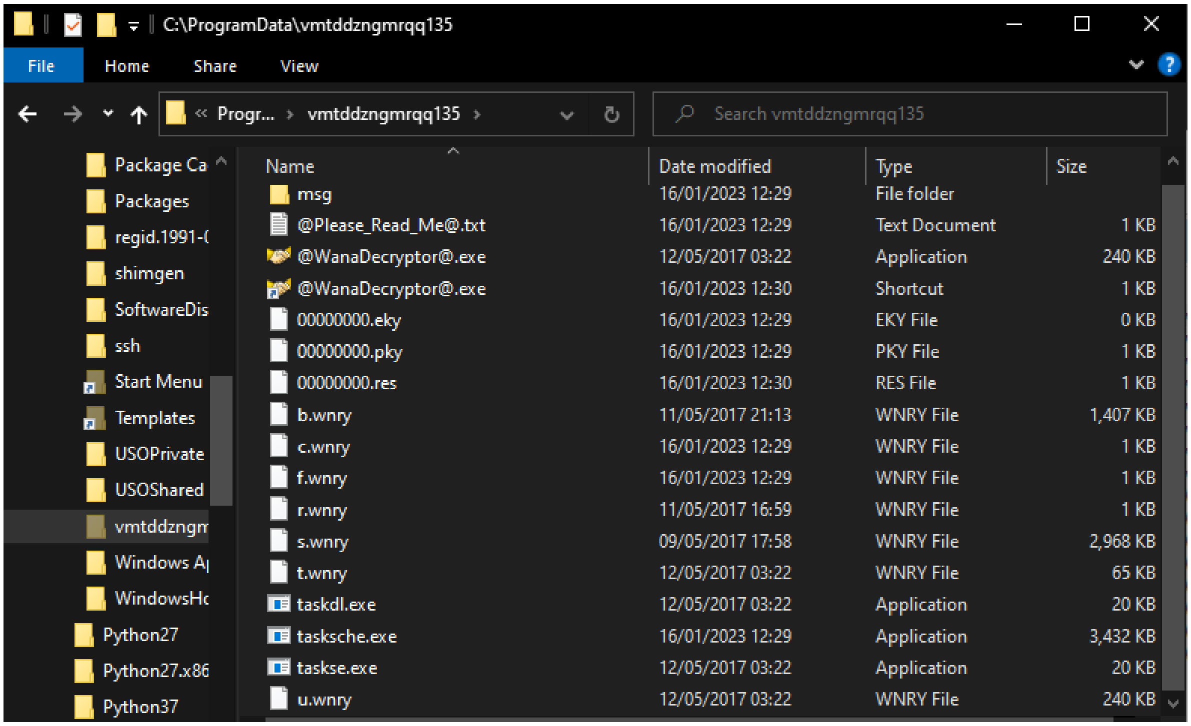
IoC 5. A snapshot of procmon after initial detonation



IoC 6. A snapshot of procmon after initial detonation



IoC 7. A snapshot of the new directory under */ProgramData*

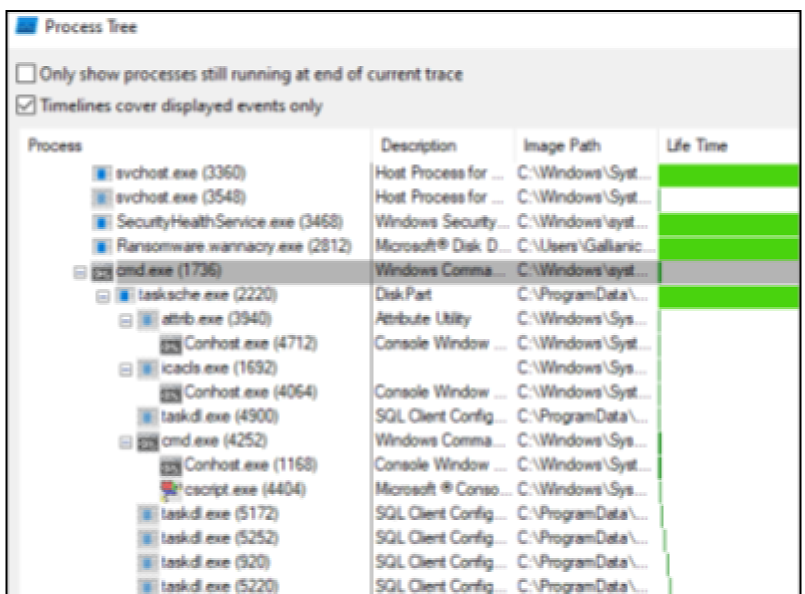


IoC 8. A snapshot of the content of the new directory

It is useful to utilize the **Process tree** utility in **procmon** (IoC 9). We can confirm that ***cmd.exe*** has been run to then execute ***tasksche.exe***. From there, we can see the other executables run: ***attrib.exe***, ***conhost.exe***, ***icacls.exe,*** and ***taskdl.exe***.

icacls is a command-line utility that can be used to modify NTFS file system permissions.

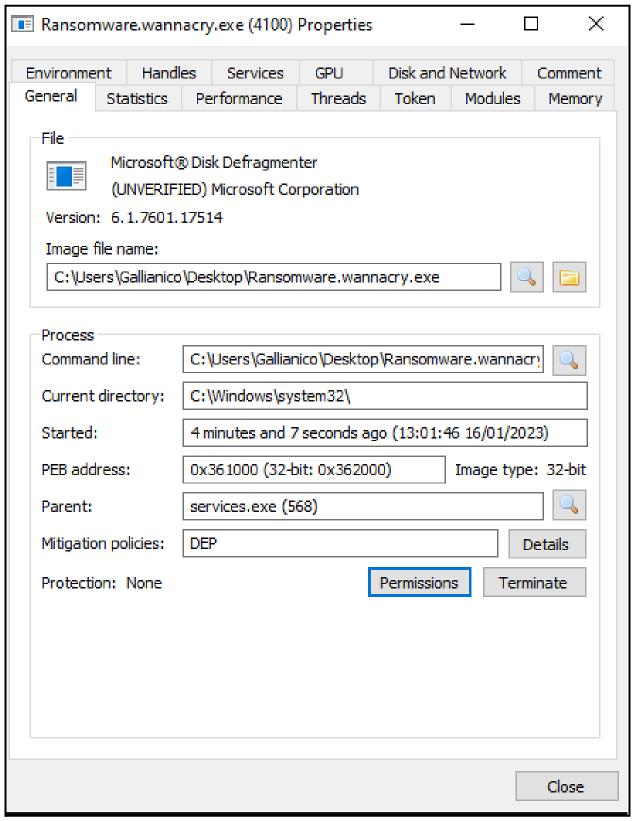
attrib.exe Displays sets, or removes attributes assigned to files or directories.
If we recall the strings that we pulled previously, we can start putting all the pieces together.



IoC 9. A view of Process Tree in Procmon

Another confirmation comes from analyzing the result of **Process Hacker** (IoC 10). We can see that the original dropper it's been executed as a child process of ***services.exe.***

services.exe is a part of the Microsoft Windows Operating System and manages the operation of starting and stopping services.



IoC 10. Detailed view of the Ransomware.wannacry.exe process

Advanced Static Analysis

I've made use of **cutter** to perform the disassembly of the binary. If we analyze the **main** function we can recognize the ***KillSwitch***. From the graph view, we can see the hard-coded domain being passed to the ***esi register*** (Art. 1). The domain is then used during the APIs calls: **InternetOpenA, InternetOpenUrlA** (Art. 2). After attempting the connection, the program performs a test on the ***edi registers***. If the connection is successful, the program terminates, if not, it proceeds with the rest of the program (***fcn. 00408090*** **-** Art. 2).

```
[0x00408140]
 139: int main (int argc, char **argv, char **envp);
 ; var int32_t var_14h @ esp+0x28
 ; var int32_t var_8h @ esp+0x3c
 ; var int32_t var_41h @ esp+0x75
 ; var int32_t var_45h @ esp+0x79
 ; var int32_t var_49h @ esp+0x7d
 ; var int32_t var_4dh @ esp+0x81
 ; var int32_t var_51h @ esp+0x85
 ; var int32_t var_55h @ esp+0x89
 ; var int32_t var_6bh @ esp+0x8b
 sub esp, 0x50
 push esi
 push edi
 mov ecx, 0xe                        ; 14
 mov esi, str.http:__www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com ; 0x4313d0
 lea edi, [var_8h]
 xor eax, eax
 rep movsd dword es:[edi], dword ptr [esi]
 movsb byte es:[edi], byte ptr [esi]
 mov dword [var_41h], eax
 mov dword [var_45h], eax
 mov dword [var_49h], eax
 mov dword [var_4dh], eax
 mov dword [var_51h], eax
 mov word [var_55h], ax
 push eax
 push eax
 push eax
```

Artifact 1. A snippet from the assembly code of the main function

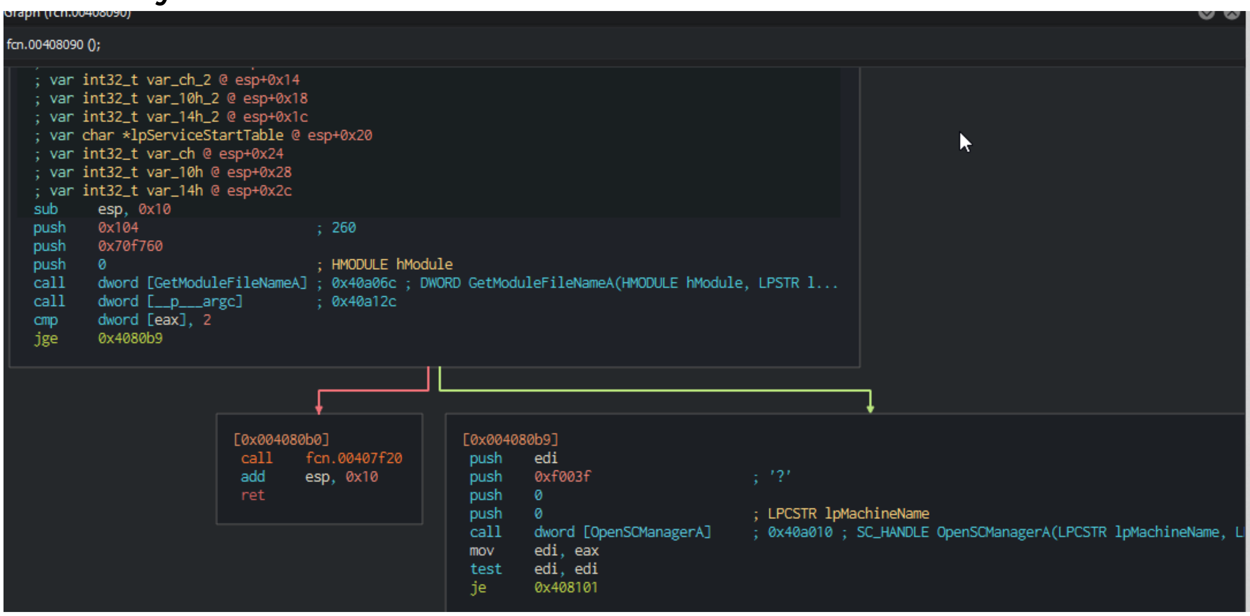
```
mov esi, eax
push 0
push ecx
push esi
call dword [InternetOpenUrlA]      ; 0x40a138
mov edi, eax
push esi
mov esi, dword [InternetCloseHandle] ; 0x40a13c
test edi, edi
jne 0x4081bc
```

```
[0x004081a7]
 call esi
 push 0
 call esi
 call 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
```

Artifact 2. A snippet from the assembly code of the main function

If we jump inside this function, we can see the binary performing a conditional execution by using the comparison (***cmp***). If the program takes the jump, we can see it making a call to the **OpenSCmanagerA** function (Art. 3). This function establishes a connection to the **service control manager** on the specified computer and opens the specified service control manager database.
The binary has now installed itself as a service.



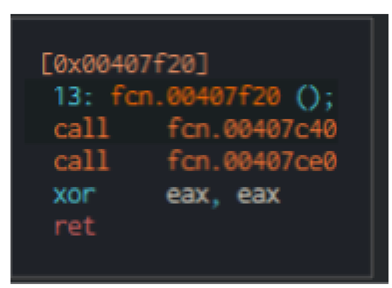
Artifact 3. A snippet from the assembly code. On the right branch, the binary executes OpenSCManagerA

If it doesn't take the jump, it'll call ***fcn. 00407f20***. If we follow the function call, we can see that it'll execute 2 additional function calls: ***fcn. 00407c40*** and ***fcn. 00407ce0*** (Art. 4).

The first one will perform the same function as the previous one. It'll install the binary as a service (Art 5).

The second function will drop the remaining executables that will handle encryption, file creation, and all the rest.

It is possible to dig deeper into the assembly code. It is also possible to perform disassembly on the



Artifact 4

other files too, but this would fall outside the scope of this report, and it would go beyond my capacity to analyze assembly.

```
[0x00407c40]
 148: fcn.00407c40 ();
 ; var int32_t var_1ch @ esp+0x54
 sub     esp, 0x104
 lea     eax, [esp]
 push    edi
 push    0x70f760
 push    str.s__m_security          ; 0x431330 ; const char *format
 push    eax                        ; char *s
 call    dword [sprintf]            ; 0x40a10c ; int sprintf(char *s, const char *format, ...)
 add     esp, 0xc
 push    0xf003f                    ; '?'
 push    0
 push    0                          ; LPCSTR lpMachineName
 call    dword [OpenSCManagerA]     ; 0x40a010 ; SC_HANDLE OpenSCManagerA(LPCSTR lpMachineName, LP...
 mov     edi, eax
 test    edi, edi
 je      0x407cca
```

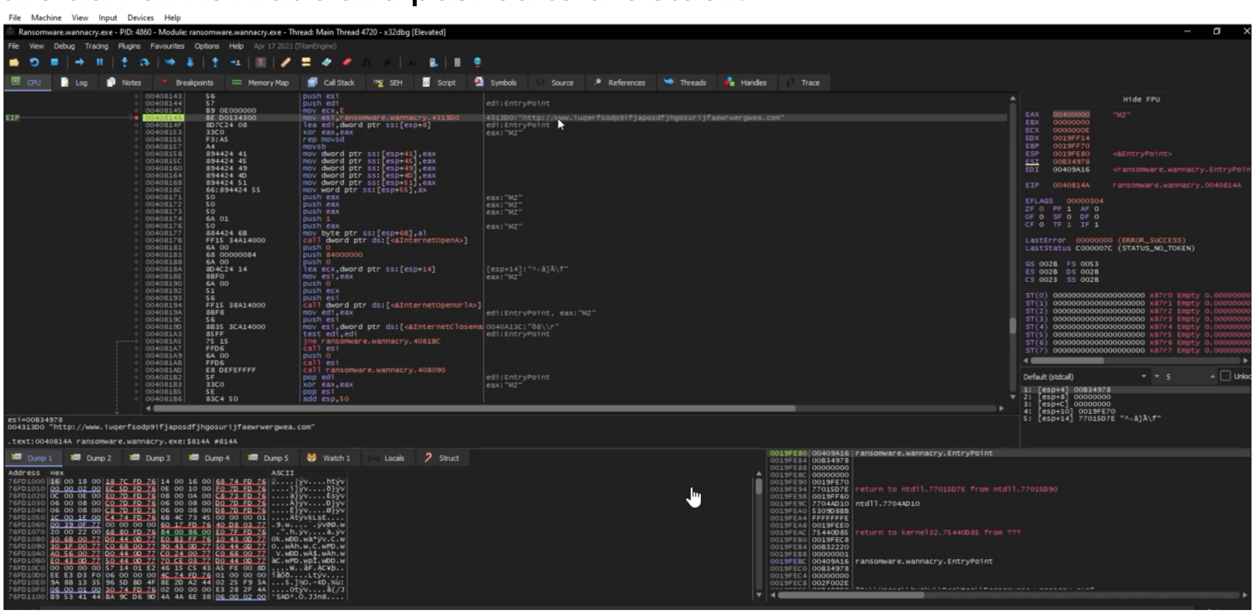
Artifact 5. A snippet from the assembly code. The binary calls again the OpenSCManagerA function

Advanced Dynamic Analysis

To perform debugging of the binary I chose *x86DBG.* From the point where we saw the program testing if the hard-code domain could be reached, it is possible to control the execution flow. By changing the zero flag value, we can make the program run even in the case where the connection can be established.

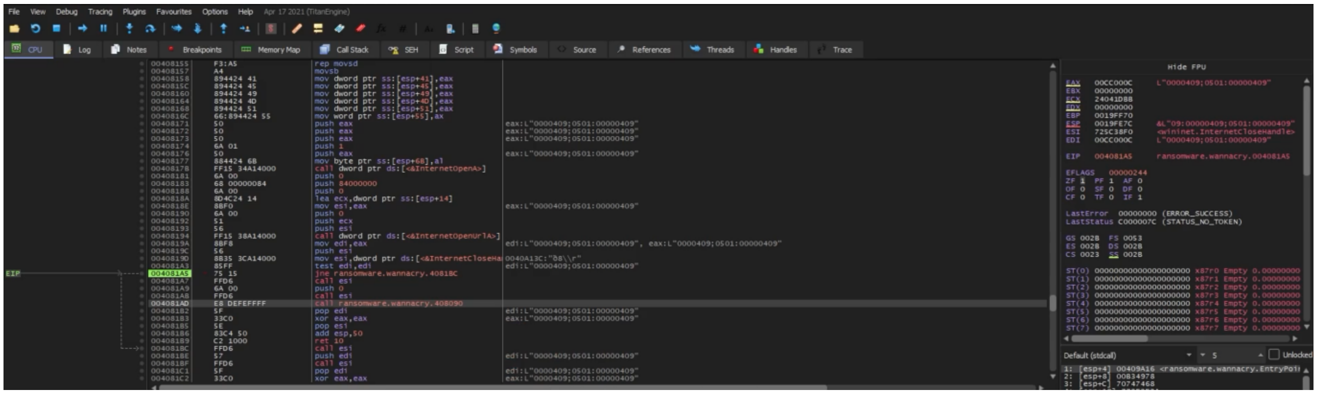
Detonation via Debugger - Controlling the *KillSwitch* workflow
VM1 : x86DBG
VM2: Internet Simulation service running - Wireshark running

As we can see in the Artifact 6, we've place a breakpoint where the call to the domain is made and pushed to the stack.



Artifact 6. A snippet from the debugger. On the top-left corner we can see the domain being called

In the Artifact 7 we can see the point where the zero flag is evaluated. At this point we only need to change the value from 1 to 0 to trick the program into thinking that the connection to the domain wasn't established.



Artifact 7. A snippet from the debugger. On the left we can see the registers being tested, and on the right box we can see the value of the zero flag.

As we can see from Fig 6, the program completed its execution, and the symptoms described in the previous section have appeared again.

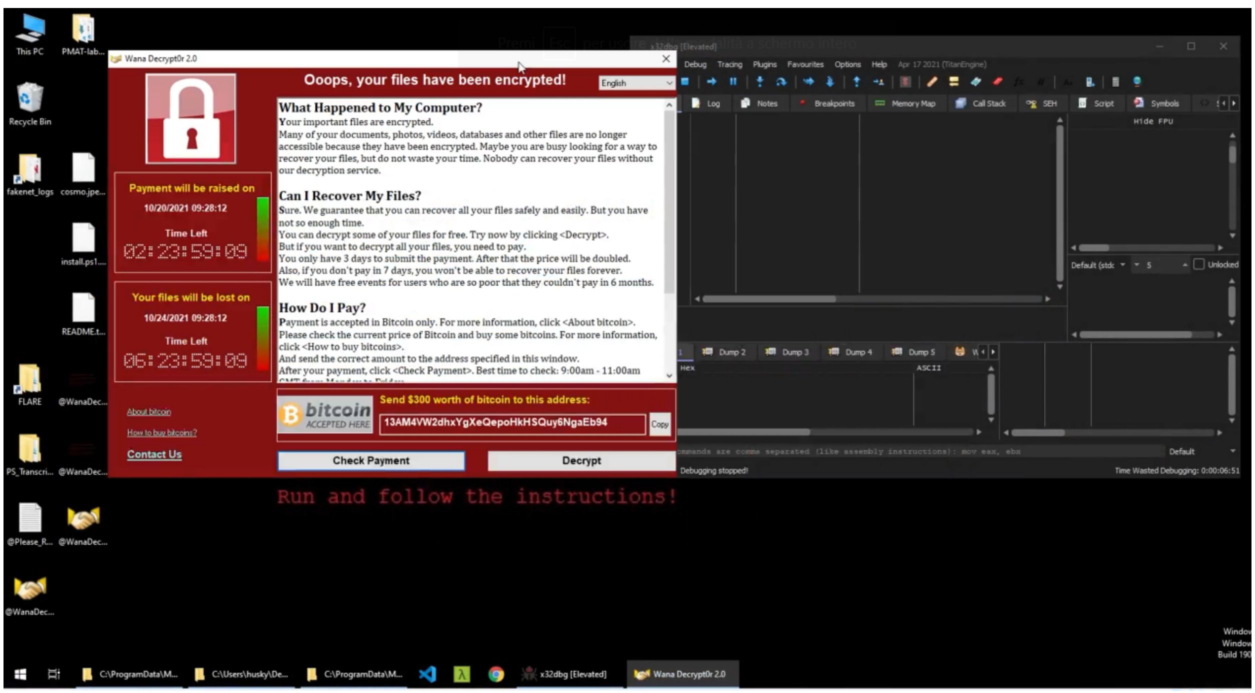
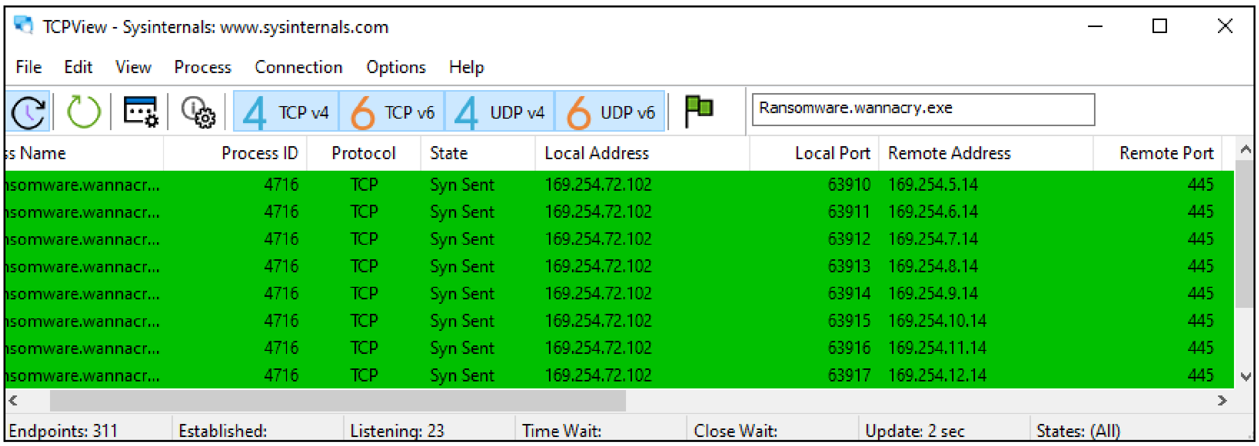


Fig. 6. A snapshot showing the successful execution of the binary

Indicators of Compromise

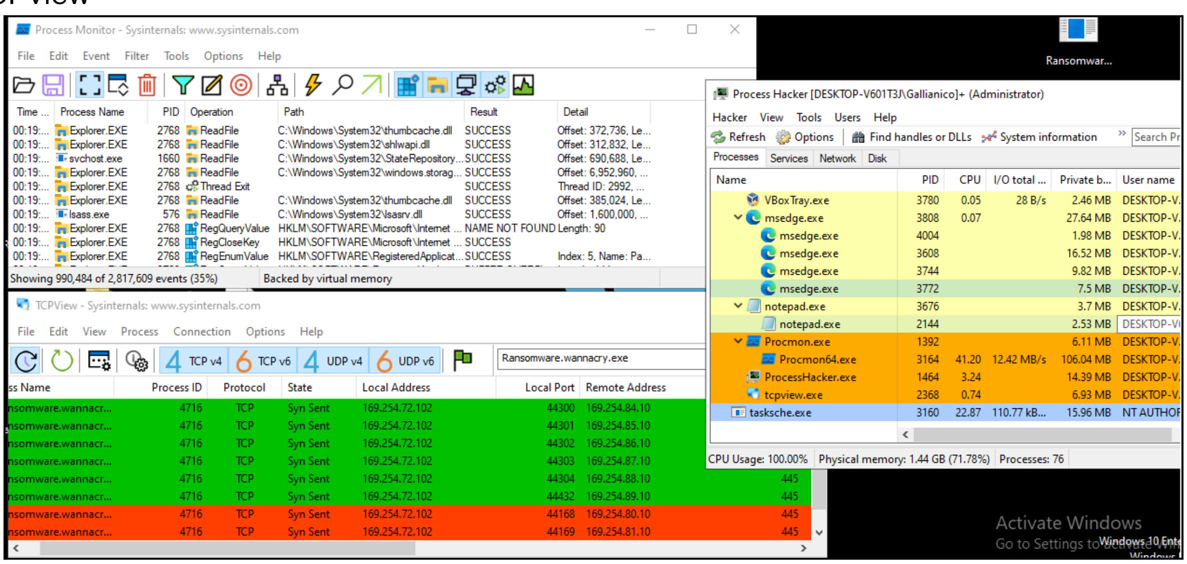
Host-based Network Indicators

IoC 11. A snapshot of TCPview showing the binary scanning available hosts one the network via SMB port 445

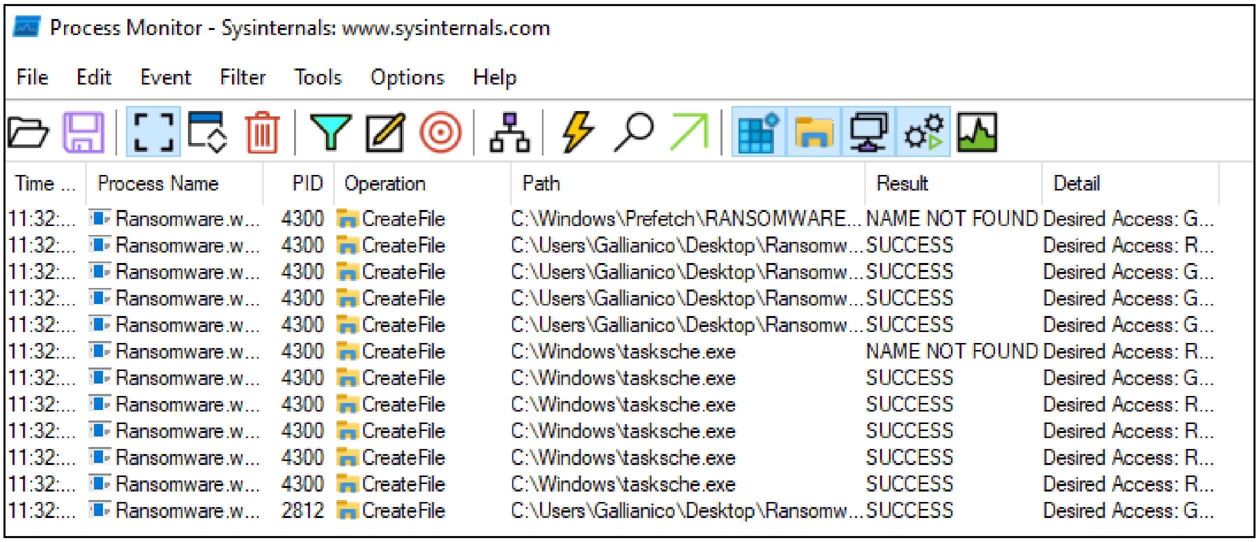


Host-based indicators

IoC 3. A snapshot showing the combined output from Procmon, Process Hacker and TCPview



IoC 4. A snapshot from Procmon showing the creation of *taskche.exe*

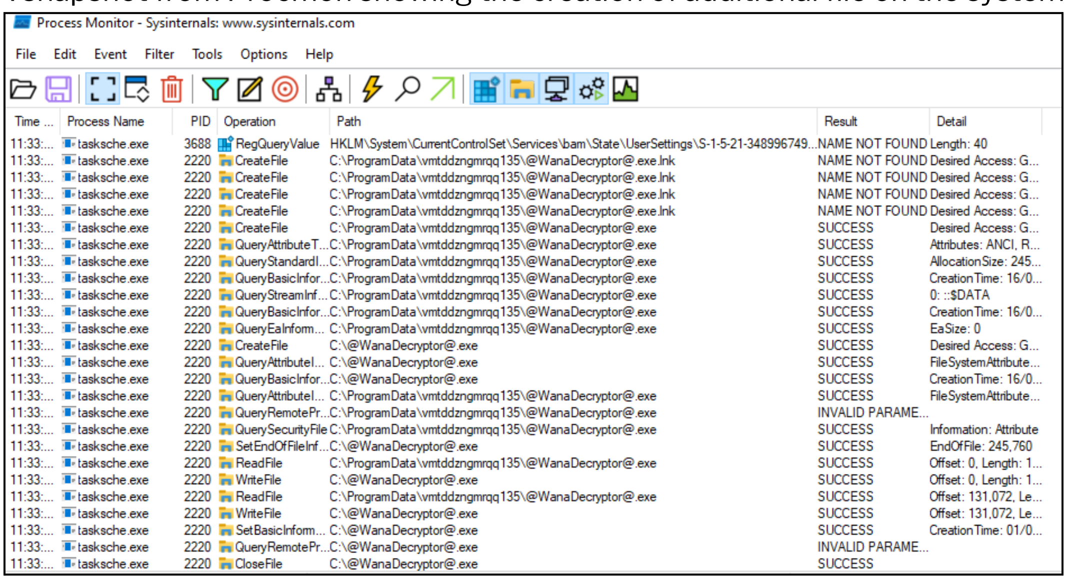


Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\Prefetch\RANSOMWARE...	NAME NOT FOUND	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	NAME NOT FOUND	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: G...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	4300	CreateFile	C:\Windows\tasksche.exe	SUCCESS	Desired Access: R...
11:32:...	Ransomware.w...	2812	CreateFile	C:\Users\Gallianico\Desktop\Ransomw...	SUCCESS	Desired Access: G...

IoC 5. A snapshot from Procmon showing the creation of additional file on the system

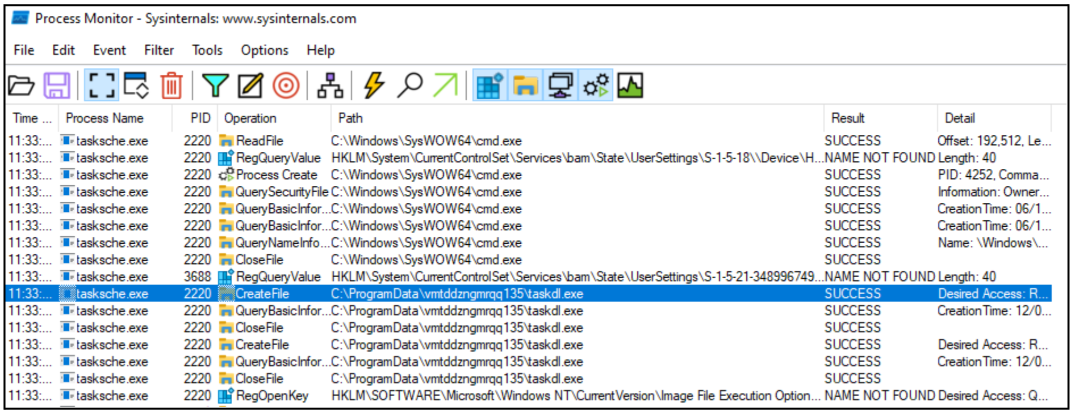


Process Monitor - Sysinternals: www.sysinternals.com

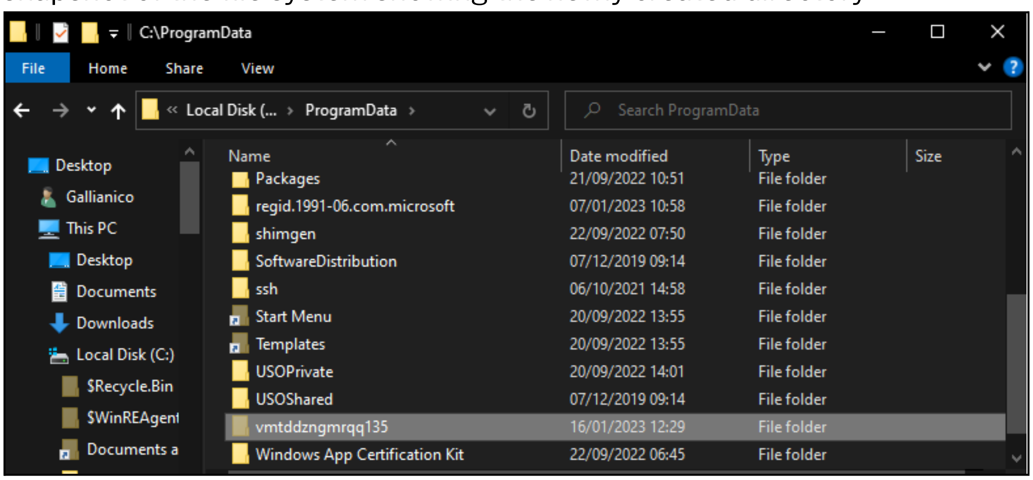
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
11:33:...	tasksche.exe	3688	RegQueryValue	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-348996749...	NAME NOT FOUND	Length: 40
11:33:...	tasksche.exe	2220	CreateFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe.lnk	NAME NOT FOUND	Desired Access: G...
11:33:...	tasksche.exe	2220	CreateFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe.lnk	NAME NOT FOUND	Desired Access: G...
11:33:...	tasksche.exe	2220	CreateFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe.lnk	NAME NOT FOUND	Desired Access: G...
11:33:...	tasksche.exe	2220	CreateFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe.lnk	NAME NOT FOUND	Desired Access: G...
11:33:...	tasksche.exe	2220	CreateFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	Desired Access: G...
11:33:...	tasksche.exe	2220	QueryAttributeT...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	Attributes: ANCI, R...
11:33:...	tasksche.exe	2220	QueryStandardI...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	AllocationSize: 245...
11:33:...	tasksche.exe	2220	QueryBasicInfor...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	CreationTime: 16/0...
11:33:...	tasksche.exe	2220	QueryStreamInf...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	0: ::$DATA
11:33:...	tasksche.exe	2220	QueryBasicInfor...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	CreationTime: 16/0...
11:33:...	tasksche.exe	2220	QueryEaInform...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	EaSize: 0
11:33:...	tasksche.exe	2220	CreateFile	C:\@WanaDecryptor@.exe	SUCCESS	Desired Access: G...
11:33:...	tasksche.exe	2220	QueryAttributeI...	C:\@WanaDecryptor@.exe	SUCCESS	FileSystemAttribute...
11:33:...	tasksche.exe	2220	QueryBasicInfor...	C:\@WanaDecryptor@.exe	SUCCESS	CreationTime: 16/0...
11:33:...	tasksche.exe	2220	QueryAttributeI...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	FileSystemAttribute...
11:33:...	tasksche.exe	2220	QueryRemotePr...	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	INVALID PARAME...	
11:33:...	tasksche.exe	2220	QuerySecurityFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	Information: Attribute
11:33:...	tasksche.exe	2220	SetEndOfFileInf...	C:\@WanaDecryptor@.exe	SUCCESS	EndOfFile: 245,760
11:33:...	tasksche.exe	2220	ReadFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	Offset: 0, Length: 1...
11:33:...	tasksche.exe	2220	WriteFile	C:\@WanaDecryptor@.exe	SUCCESS	Offset: 0, Length: 1...
11:33:...	tasksche.exe	2220	ReadFile	C:\ProgramData\vmtddzngmrqq135\@WanaDecryptor@.exe	SUCCESS	Offset: 131,072, Le...
11:33:...	tasksche.exe	2220	WriteFile	C:\@WanaDecryptor@.exe	SUCCESS	Offset: 131,072, Le...
11:33:...	tasksche.exe	2220	SetBasicInform...	C:\@WanaDecryptor@.exe	SUCCESS	CreationTime: 01/0...
11:33:...	tasksche.exe	2220	QueryRemotePr...	C:\@WanaDecryptor@.exe	INVALID PARAME...	
11:33:...	tasksche.exe	2220	CloseFile	C:\@WanaDecryptor@.exe	SUCCESS	

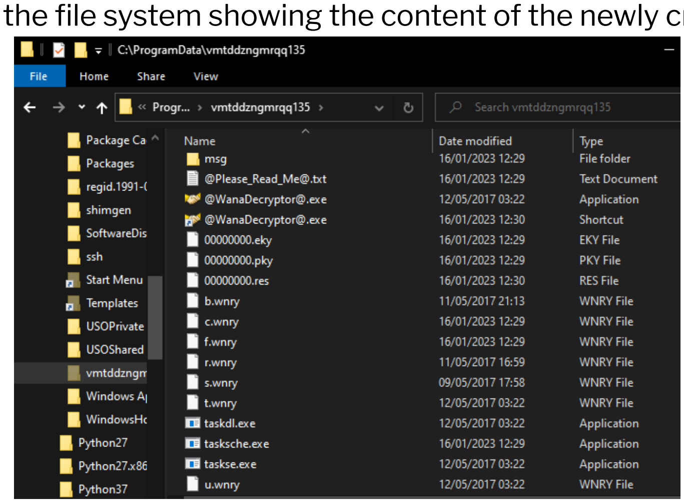
IoC 6. A snapshot from Procmon showing the creation of additional file on the system



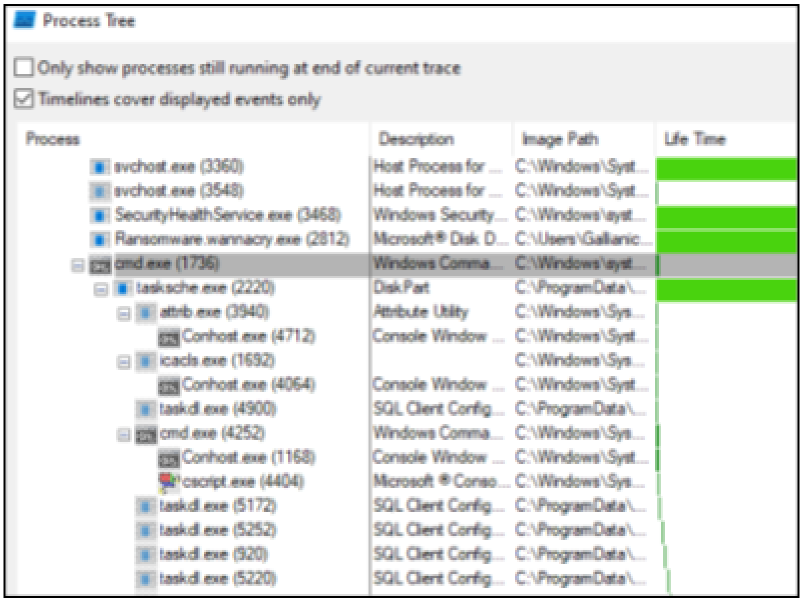
IoC 7. A snapshot of the file system showing the newly created directory



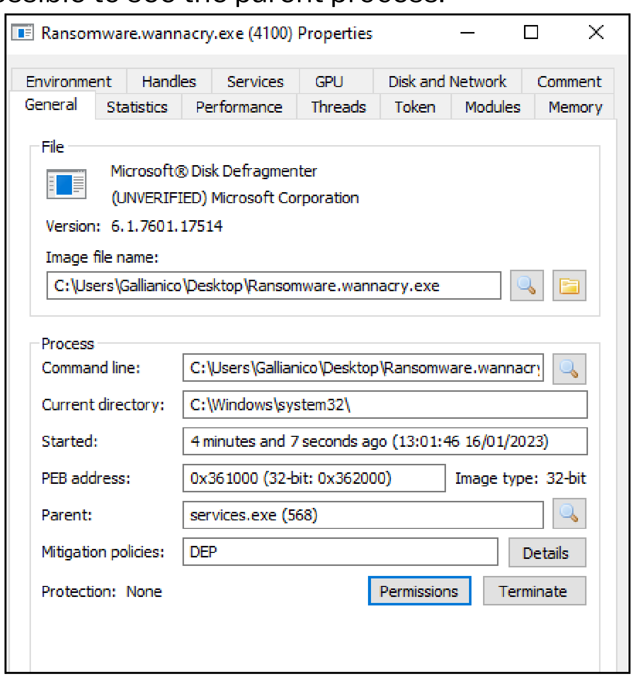
IoC 8. A snapshot of the file system showing the content of the newly created direcory



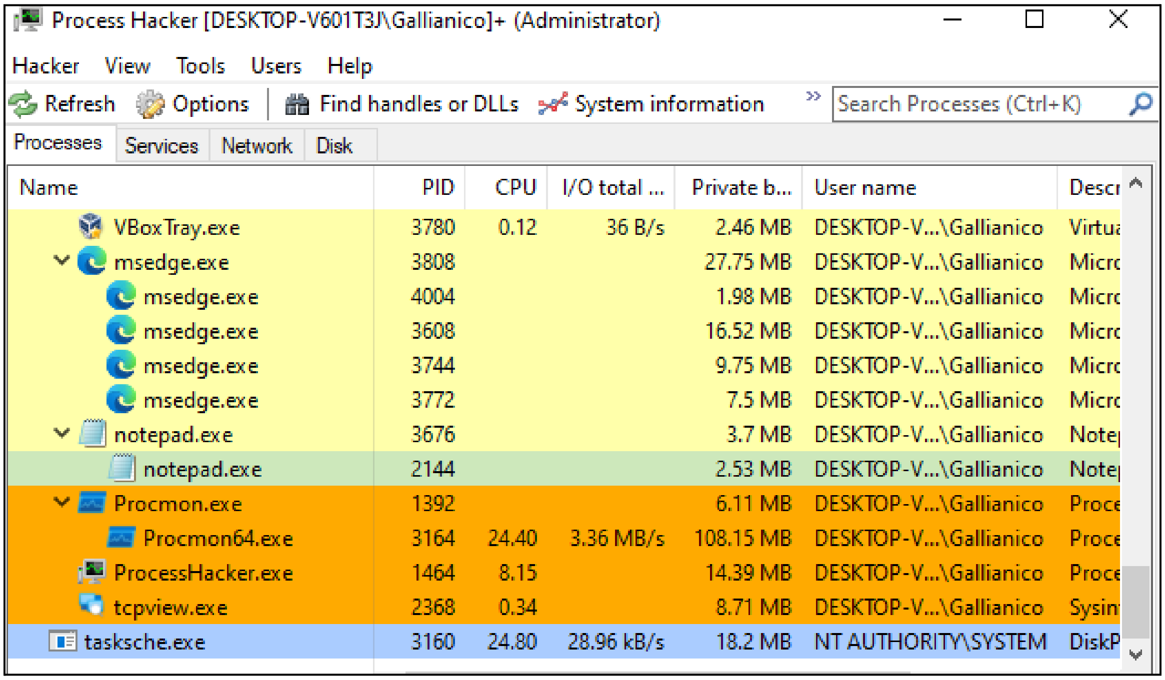
IoC 9. A snapshot from the Process Tree utility showing the processes initiated by the binary



IoC 10. A snapshot from the Process Tree utility showing the Ransomware.wannacry.exe process in detail. It's possible to see the parent process.

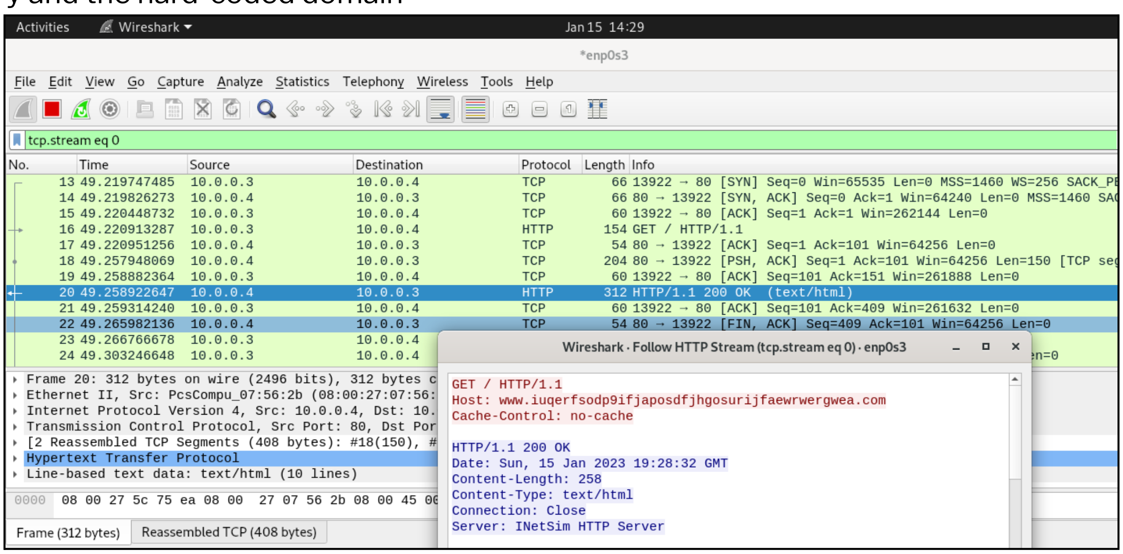


IoC 12. A snapshot from Process hacker showing the *tasksche.exe* running process



Network-based Indicators

IoC 1. A snapshot from Wireshark showing the attempted communication between the binary and the hard-coded domain



YARA Rules & Signatures

```
rule wannacry_detector


    meta

                 = "Alessio Ragazzi"

                     = "18/01/2023"

                       = "18/01/2023"

                      = "Basic yara rules to detect the ransomware WannaCry. Rules are based on the strings that I was able to pull during basic static analysis of the binary"


    strings

        $executable1 = "tasksche.exe"

        $executable2 = "taskdl.exe"

        $string1 = "mssecsvc2.0"

        $string2 = "wannacry"

        $string3 = "wanadecryptor"

        $url1= "iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea"


    condition

        $executable1 and  $executable2 and $string1  or

        $string2 and $executable1 or

        $string3 and $executable1 or

        $url1 and  $executable1 or $executable1 or $string1 or $string2 or $string3
```

Appendices

A. Yara Rules

Full Yara repository located at: https://github.com/ale17ragazzi

B. Further reading

https://www.mandiant.com/resources/blog/wannacry-malware-profile

C. URLs of interest

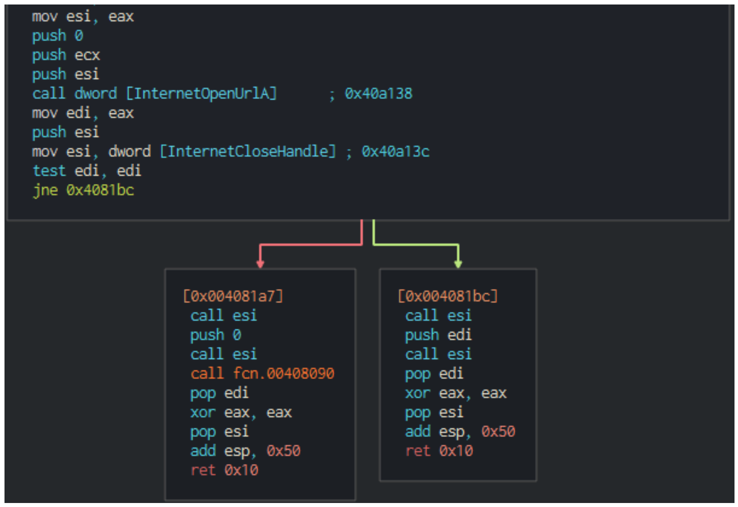
Domain	Port
http://www[.]iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea[.]com	80

D. Disassembled Code Snippets

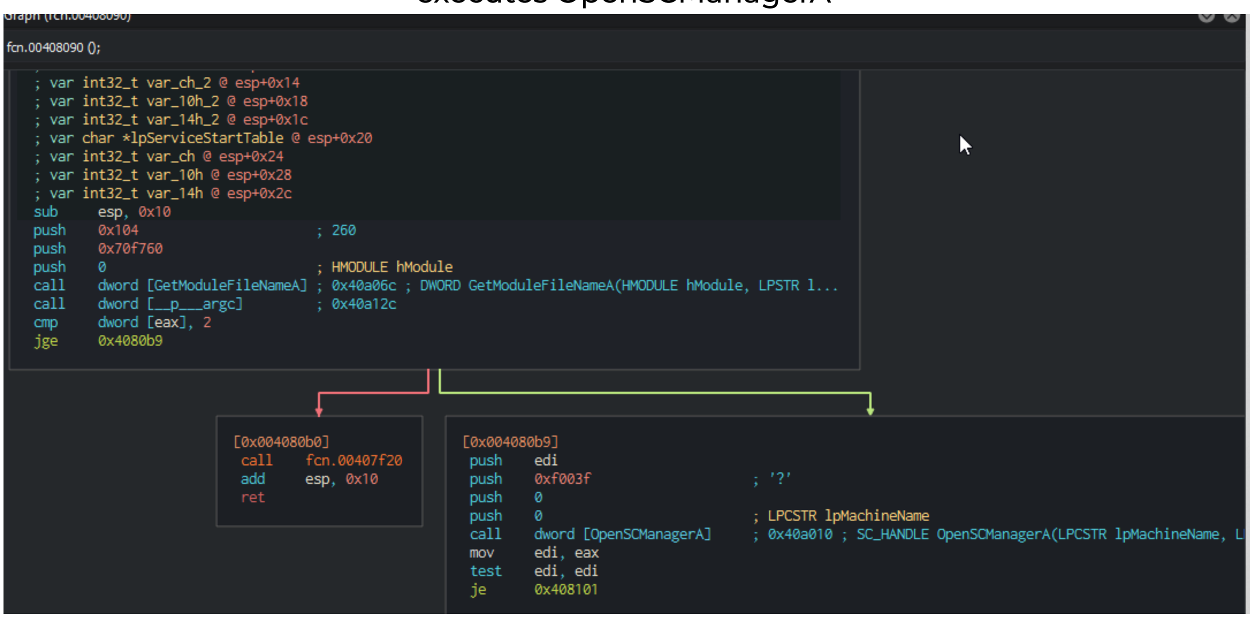
Artifact 1. A snippet from the assembly code of the main function

```
[0x00408140]
 139: int main (int argc, char **argv, char **envp);
 ; var int32_t var_14h @ esp+0x28
 ; var int32_t var_8h @ esp+0x3c
 ; var int32_t var_41h @ esp+0x75
 ; var int32_t var_45h @ esp+0x79
 ; var int32_t var_49h @ esp+0x7d
 ; var int32_t var_4dh @ esp+0x81
 ; var int32_t var_51h @ esp+0x85
 ; var int32_t var_55h @ esp+0x89
 ; var int32_t var_6bh @ esp+0x8b
 sub esp, 0x50
 push esi
 push edi
 mov ecx, 0xe                       ; 14
 mov esi, str.http:__www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com ; 0x4313d0
 lea edi, [var_8h]
 xor eax, eax
 rep movsd dword es:[edi], dword ptr [esi]
 movsb byte es:[edi], byte ptr [esi]
 mov dword [var_41h], eax
 mov dword [var_45h], eax
 mov dword [var_49h], eax
 mov dword [var_4dh], eax
 mov dword [var_51h], eax
 mov word [var_55h], ax
 push eax
 push eax
 push eax
```

Artifact 2. A snippet from the assembly code of the main function



Artifact 3. A snippet from the assembly code. On the right branch, the binary executes OpenSCManagerA



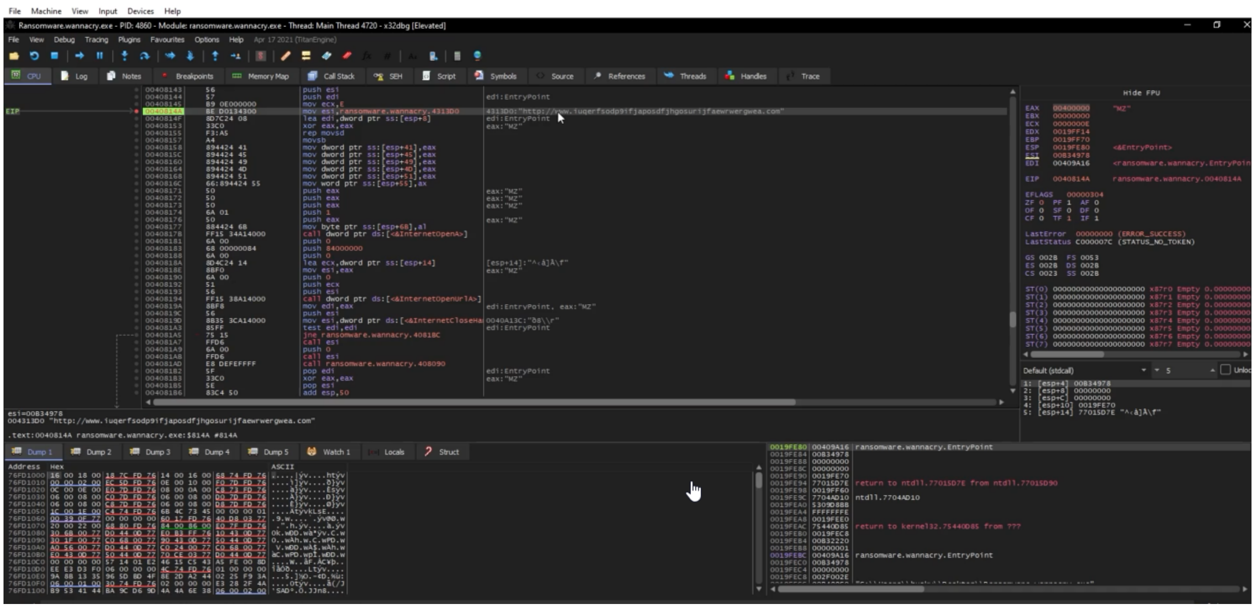
Artifact 4

```
[0x00407f20]
 13: fcn.00407f20 ();
 call      fcn.00407c40
 call      fcn.00407ce0
 xor       eax, eax
 ret
```

Artifact 5

```
[0x00407c40]
 148: fcn.00407c40 ();
 ; var int32_t var_1ch @ esp+0x54
 sub     esp, 0x104
 lea     eax, [esp]
 push    edi
 push    0x70f760
 push    str.s__m_security            ; 0x431330 ; const char *format
 push    eax                          ; char *s
 call    dword [sprintf]              ; 0x40a10c ; int sprintf(char *s, const char *format, ...)
 add     esp, 0xc
 push    0xf003f                      ; '?'
 push    0
 push    0                            ; LPCSTR lpMachineName
 call    dword [OpenSCManagerA]       ; 0x40a010 ; SC_HANDLE OpenSCManagerA(LPCSTR lpMachineName, LP...
 mov     edi, eax
 test    edi, edi
 je      0x407cca
```

Artifact 6. A snippet from the debugger. On the top-left corner we can see the domain being called



Artifact 7. A snippet from the debugger. On the left we can see the registers being tested, and on the right box we can see the value of the zero flag.