CS4238 Malware Analysis Report - WannaCry

Joyce Yeo Shuhui
Jonathan Cheng

Date submitted: 1 Dec 2020

Basic Information

For brevity, we give shorter names to the malware samples in [“brackets”].

Name	09a46b3e1be080745a6d8d88d6b5bd351b1c7586ae0dc94d0c238ee36421cafa .bin [***“file1”***]
Size	3432 KB
Type [PE View]	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	509c41ec97bb81b0567b059aa2f50fe8
SHA1 [HashCalc]	87420a2791d18dad3f18be436045280a4cc16fc4
SHA256 [HashCalc]	09a46b3e1be080745a6d8d88d6b5bd351b1c7586ae0dc94d0c238ee36421cafa
Compiled [PE View]	2010/11/20 Sat 09:05:05 UTC
Packers [PEiD]	Microsoft Visual C++ 6.0, Entry Point 000077BA
Imported DLLs [IDA]	ADVAPI32, KERNEL32, MSVCRT, USER32

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	920E964050A1A5DD60DD00083FD541A2	28672	6.404

.rdata	2C42611802D585E6EED68595876D1A15	24576	6.664
.data	83506E37BD8B50CACABD480F8EB3849B	8192	4.456
.rsrc	4DDC9B76F38B5823F69E1F505A0A152D	3448832	8.000

Resources [Resource Hacker + HashCalc]

Name	MD5
XIA	5b225149abb8c8eb245445f707e6f0d2
Version Info	0e14014289c29078069237196bd3ea72
Manifest	a31cf56465371581763e9f0a86d41987

Name	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c.bin [***"file2"***]
Size	3636 KB
Type [PE View]	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	db349b97c37d22f5ea1d1841e3c89eb4
SHA1 [HashCalc]	e889544aff85ffaf8b0d0da705105dee7c97fe26
SHA256 [HashCalc]	24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c
Compiled [PE View]	2010/11/20 Sat 09:03:08 UTC
Packers [PEiD]	Microsoft Visual C++ 6.0, Entry Point 00009A16
Imported DLLs [IDA]	ADVAPI32, KERNEL32, MSVCP60, MSVCRT, WININET, WS2_32, iphlpapi

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	C7613102E2ECEC5DCEFC144F83189153	36864	6.135
.rdata	D8037D744B539326C06E897625751CC9	4096	3.504
.data	22A8598DC29CAD7078C291E94612CE26	159744	6.100
.rsrc	12E1BD7375D82CCA3A51CA48FE22D1A9	3518464	7.995

Resources [Resource Hacker + HashCalc]

Name	MD5
R	84c82835a5d21bbcf75a61706d8ab549
Version Info	1ebdc36976dd611e1a9e221a88e6858e

Name	201f42080e1c989774d05d5b127a8cd4b4781f1956b78df7c01112436c89b2c9.bin [***"file3"***]
Size	232 KB
Type [PE View]	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	8dd63adb68ef053e044a5a2f46e0d2cd
SHA1 [HashCalc]	1bc604573ceab106e5a0e9c419ade38739228707
SHA256 [HashCalc]	201f42080e1c989774d05d5b127a8cd4b4781f1956b78df7c01112436c89b2c9
Compiled [PE View]	2017/03/27 Mon 05:17:07 UTC
Packers [PEiD]	Microsoft Visual C++ 6.0, Entry Point 00008A44
Imported DLLs [IDA]	ADVAPI32, KERNEL32, MSVCRT, USER32, WS2_32

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	952E19045301532623681F7A72758663	32768	6.390
.rdata	3E0E0E9555BBC4CD2B9C1B122B77602D	24576	6.537
.data	4DA32D4E7477776E8F496E2D8D55594E	8192	3.923
.rsrc	7455F1A2D4B6BA840E5BBCD0144EC488	167936	7.930

Resources [Resource Hacker + HashCalc]

Name	MD5
CFG	3989c6075c13ff5ec2c00e55ba7371d5
Manifest	a31cf56465371581763e9f0a86d41987

Name	R / tasksche.exe / qeriuwjhrf
Size	3432 KB
Type [PE View]	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	84c82835a5d21bbcf75a61706d8ab549
SHA1 [HashCalc]	5ff465afaabcbf0150d1a3ab2c2e74f3a4426467
SHA256 [HashCalc]	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa
Compiled [PE View]	2010/11/20 Sat 09:05:05 UTC
Packers [PEiD]	Microsoft Visual C++ 6.0, Entry Point 000077BA
Imported DLLs [IDA]	ADVAPI32, KERNEL32, MSVCRT, USER32

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	920E964050A1A5DD60DD00083FD541A2	28672	6.404
.rdata	2C42611802D585E6EED68595876D1A15	24576	6.664
.data	83506E37BD8B50CACABD480F8EB3849B	8192	4.456
.rsrc	F99CE7DC94308F0A149A19E022E4C316	3448832	8.000

Resources [Resource Hacker + HashCalc]

Name	MD5
XIA	b576ada3366908875e5ce4cb3da6153a
Version Info	0e14014289c29078069237196bd3ea72
Manifest	a31cf56465371581763e9f0a86d41987

Name	@WanaDecryptor@.exe
Size	240 KB
Type [PE View]	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	7bf2b57f2a205768755c07f238fb32cc
SHA1 [HashCalc]	45356a9dd616ed7161a3b9192e2f318d0ab5ad10
SHA256 [HashCalc]	45356a9dd616ed7161a3b9192e2f318d0ab5ad10
Compiled [PE View]	2009/07/13 Mon 23:19:35 UTC
Packers [PEiD]	Microsoft Visual C++ 6.0, Entry Point 00013102

Imported DLLs [IDA]	ADVAPI32, COMCTL32, GDI32, KERNEL32, MFC42, MSVCP60, MSVCRT, OLEAUT32, SHELL32, USER32, WININET, WS2_32, urlmon

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	C9EDE1054FEF33720F9FA97F5E8ABE4	81920	6.241
.rdata	5A89AAC6C8259ABBBA2FA2AD3FCEFC6E	40960	5.872
.data	05DA32043B1E3A147DE634C550F1954D	12288	4.727
.rsrc	8E97637474AB77441AE5ADD3F3325753	106496	5.635

Resources [Resource Hacker + HashCalc]

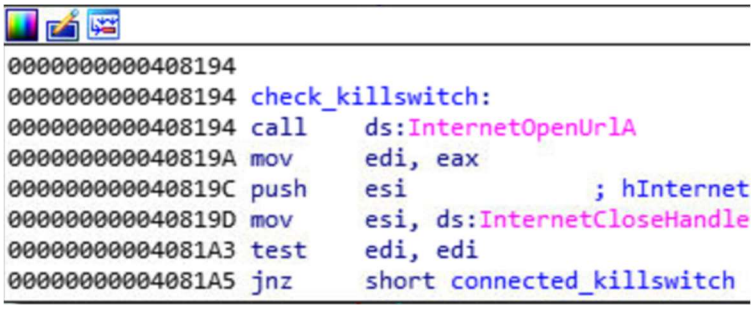
Name	MD5
Bitmap134	053cb9b42aa55f9f01a2e777e44589f9
Bitmap135	e177012ab89bc2b6d47b3c2b8ee423e2
Bitmap136	4e4daa5c1c2954c306267524c93033a0
Icon1	4ae749d3104f6a1f4cfb34c5c0428f35
Icon2	74df1cc56549ee28f3fb8015def915f9
Icon3	1976f46ba4fccdfb9487f471509b65a9
Dialog102	d50e408ca75519b1a9bcbc6a2cfaf324
Dialog137	f9fc70f49f772b6d2652e0eb0a91a614
Dialog138	a4edae0fe79803f8329051e0461c80f9
Dialog141	c095671bad49e32e7f76750294495d40
Dialog143	712cb93371d130b8b50465d13133b0ad
IconGroup128	1ca559e52ba2941be4c2e87cc5728277
IconGroup139	00ea652113ed90803b15f6167124172d
VersionInfo	fe953411cfb9a4a566e0727e096c9eeb

Manifest	0585548eef8226c74c1d523108ed81f3
240	b485116506cc664bac6886aaac65d756

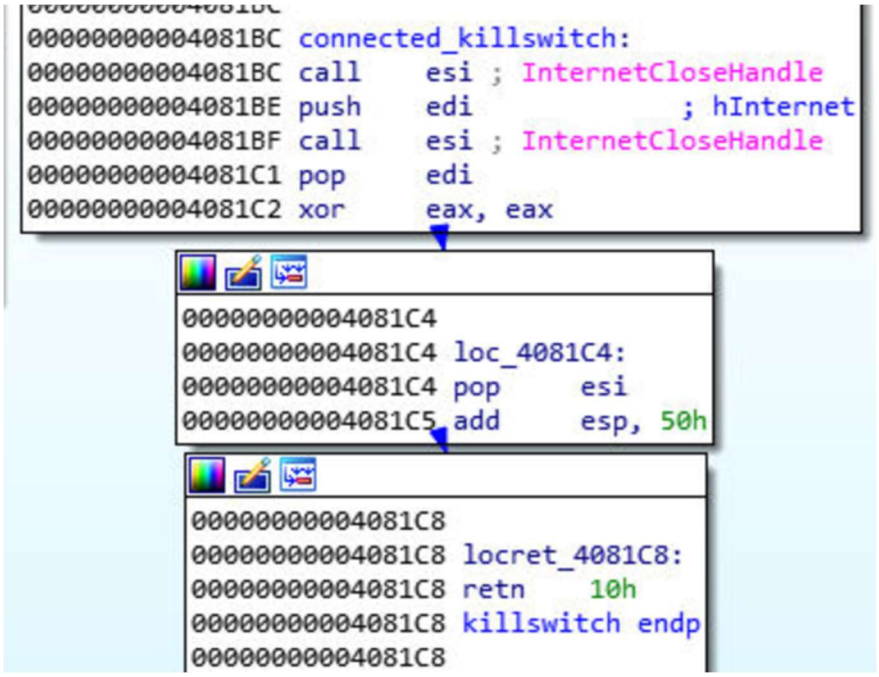
Killswitch

The killswitch mechanism is implemented in file2. Analysing from the function at 0x408140, we see that there is a random URL that the malware attempts to visit using InternetOpenUrl.

The result of visiting the domain is tested. If the domain is alive and the malware can connect to the site, the return value to InternetOpenUrl (stored in eax and then edi) is 0. Then, the jnz jump is taken.



At the jump location, the entire process of checking for the domain again is started. The same function at 0x408140 is called again and the checking repeats indefinitely.



This means that the domain www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com functions as WannaCry's killswitch. When the domain is alive, the malicious payload will not be executed.

Currently, the domain is active and sinkholed by KryptosLogic.

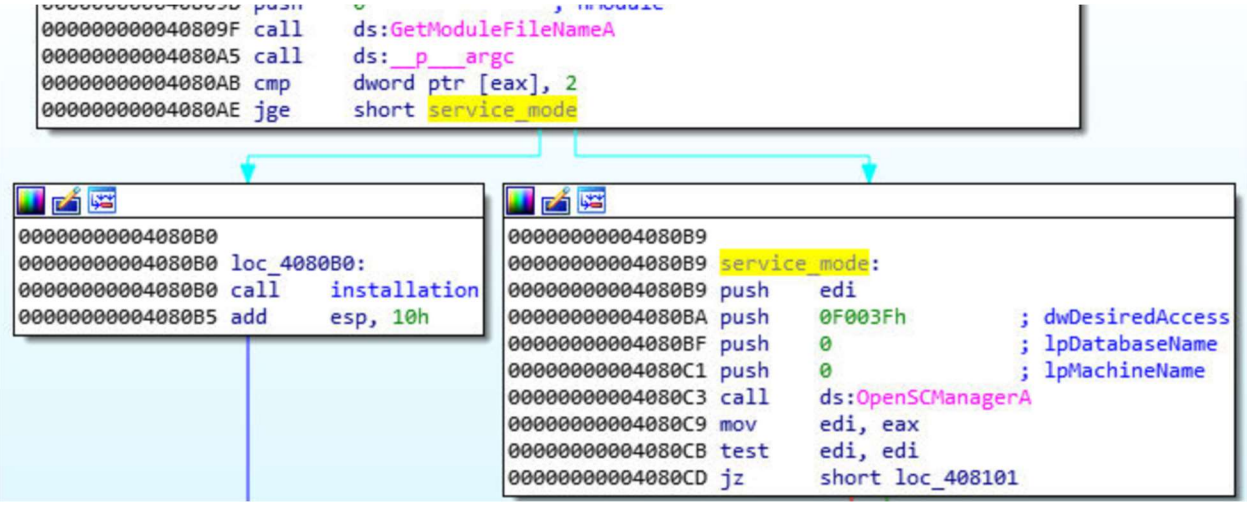
This sinkhole effectively prevents any WannaCry malware on hosts with internet connection to execute the malicious payload. Hence, when running WannaCry in the sandbox environment, we need to ensure not to use any network simulation services. Else, the malware can successfully connect to the site and will not execute the malicious payload for analysis.

Installation and Persistence

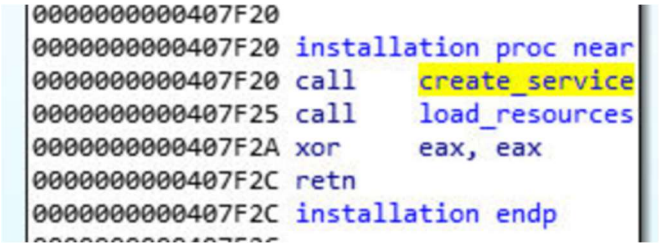
Should the killswitch test in file2 pass (i.e. the killswitch is still inactive), then the malicious payload at 0x408090 begins to execute. At this point, the malware will do a check for the number of arguments passed into args.

If there is an argument passed into the executable, then service mode is started. Service mode involves the malware running its payload and propagating across the network via the Eternal Blue SMB exploit. Even though Microsoft patched the exploit a month before WannaCry was released by The Shadow Brokers, the infection rate of WannaCry was still very high due to users not updating their systems. In this report, we analyse the behaviour of WannaCry should a system get infected. The propagation mechanism is not examined in detail.

If there is no argument to the executable. the installation step is done.



Installation function is at 0x407F20 and has two key parts (1) create the service at 0x407C40 and (2) loading the resources at 0x407CE0. More will be explained below.



(1) create_service function at 0x407C40. The malware first creates a service for itself, called “mssecsvc2.0”

```
0000000000407C8B push    0F01FFh         ; dwDesiredAccess
0000000000407C90 push    offset DisplayName ; "Microsoft Security Center (2.0) Service"
0000000000407C95 push    offset ServiceName ; "mssecsvc2.0"
0000000000407C9A push    edi             ; hSCManager
0000000000407C9B call    ds:CreateServiceA
0000000000407CA1 mov     ebx, ds:CloseServiceHandle
0000000000407CA7 mov     esi, eax
0000000000407CA9 test    esi, esi
0000000000407CAB jz      short loc_407CBB
```

The service points to the current working directory (cwd), with “-m security” as an additional argument. The entire string is created using the sprintf function and stored as dwDesiredAccess. The variable dwDesiredAccess is used in the CreateServiceA function shown above.

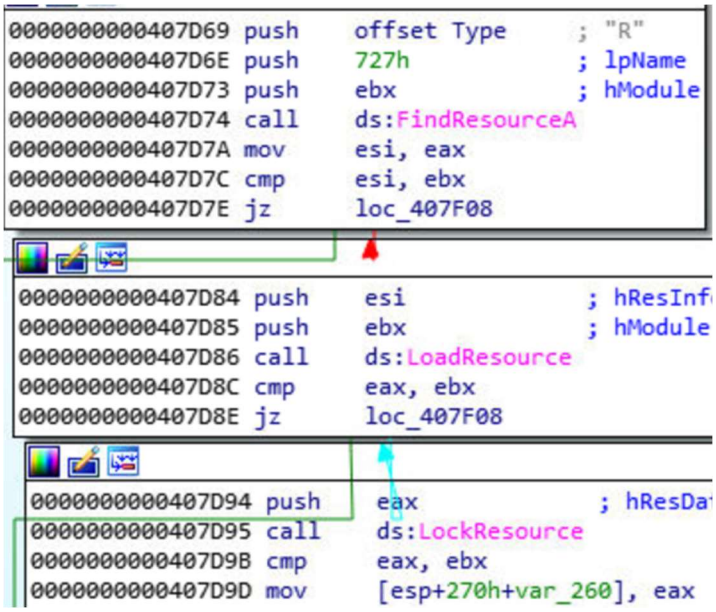
```
0000000000407C40 sub_407C40 proc near
0000000000407C40
0000000000407C40 Dest= byte ptr -104h
0000000000407C40
0000000000407C40 sub     esp, 104h
0000000000407C46 lea     eax, [esp+104h+Dest]
0000000000407C4A push    edi
0000000000407C4B push    offset Filename
0000000000407C50 push    offset aSMSecurity ; "%s -m security"
0000000000407C55 push    eax             ; Dest
0000000000407C56 call    ds:sprintf
0000000000407C5C add     esp, 0Ch
0000000000407C5F push    0F003Fh         ; dwDesiredAccess
```

Then, file2 also starts the service.

```
00000000004080D0
00000000004080D0 loc_4080D0:
00000000004080D0 push    esi
00000000004080D1 push    0F01FFh         ; dwDesiredAccess
00000000004080D6 push    offset ServiceName ; "mssecsvc2.0"
00000000004080DB push    edi             ; hSCManager
```

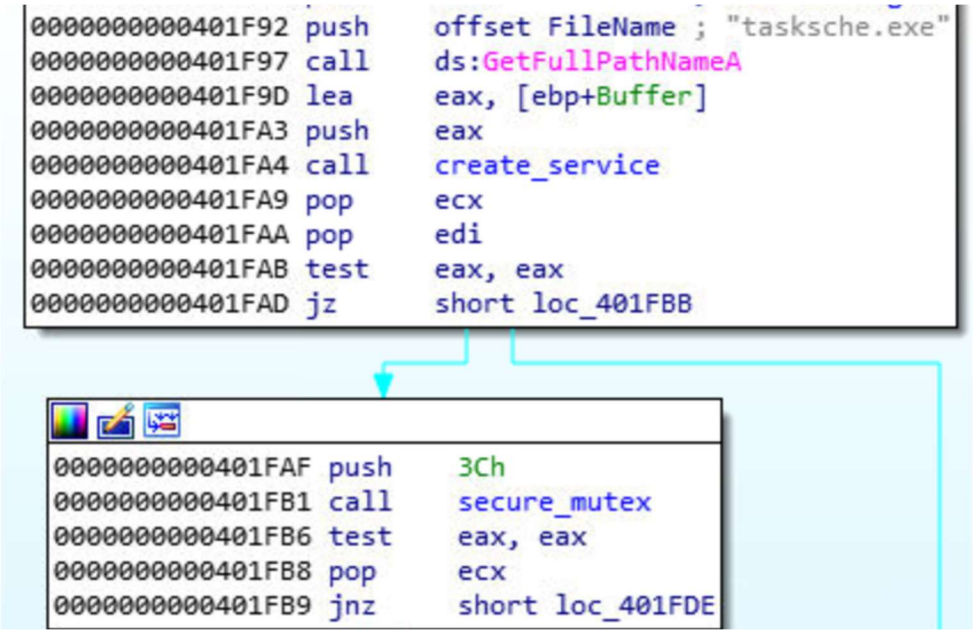
```
00000000004080DC
00000000004080DC loc_4080DC:
00000000004080DC call    ds:OpenServiceA
00000000004080E2 mov     ebx, ds:CloseServiceHandle
```

(2) load_resources function at 0x407CE0. The malware loads its own resource **R** into memory. More information on “R” can be found in “Basic Information”. Interestingly, R has the same resource names as file1 though the hashes differ. This finding suggests that file1 could be a variant of R developed and spread.

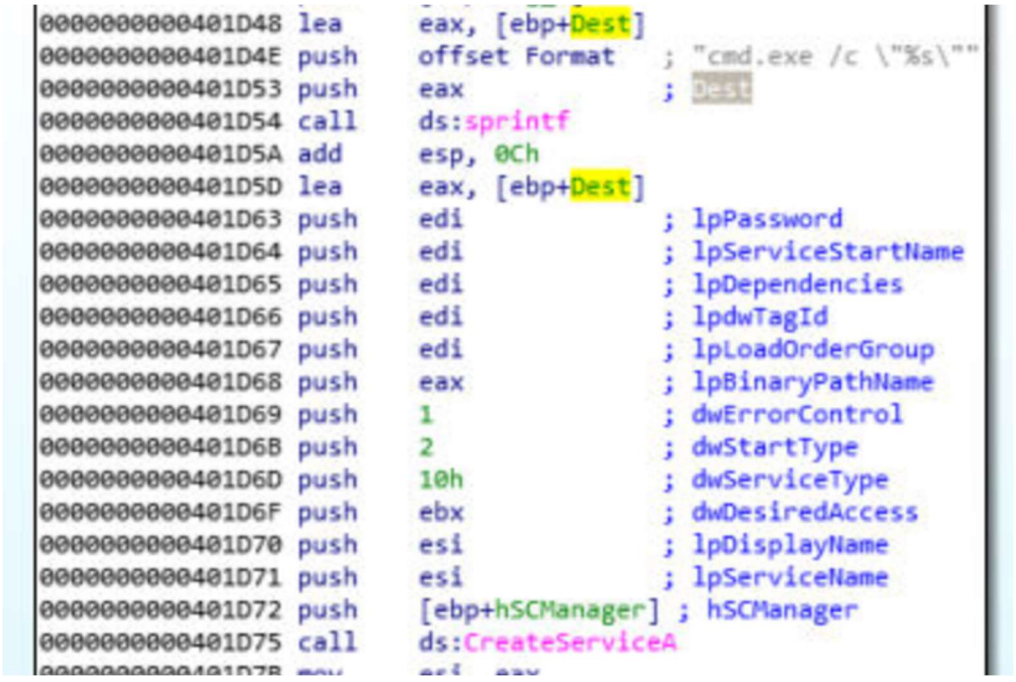


The resources are then stored in C:\\WINDOWS\\tasksche.exe, and then moved to C:\\WINDOWS\qeriuwjhrf. (Image is blur as we had to zoom out on the VM.)

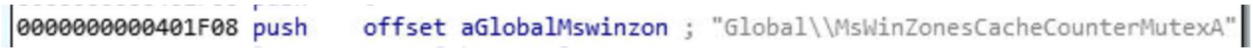
Installing the payload as a service ensures **persistence** of the malware. The malware loader (file1) will then check if it was executed with the /i argument. If the argument /i is present, the malware will interact with the service manager, then has mutex activity.



First, it checks if a required service (implanted earlier) exists. If it doesn't exist, then the malware creates a new service pointing to "cmd /c <path to tasksche.exe>"



The malware then attempts to secure the mutex shown below:



After securing the installation, the malware will run the main payload. It first updates the current directory.

```
00000000004020B4 main_payload:
00000000004020B4 lea     eax, [ebp+Filename]
00000000004020BA push    eax             ; lpPathName
00000000004020BB call    ds:SetCurrentDirectoryA
00000000004020C1 push    1
00000000004020C3 call    setup
```

The function “setup” at 0x4010FD involves registry functions. The subkey “wd” is altered and set to the current directory.

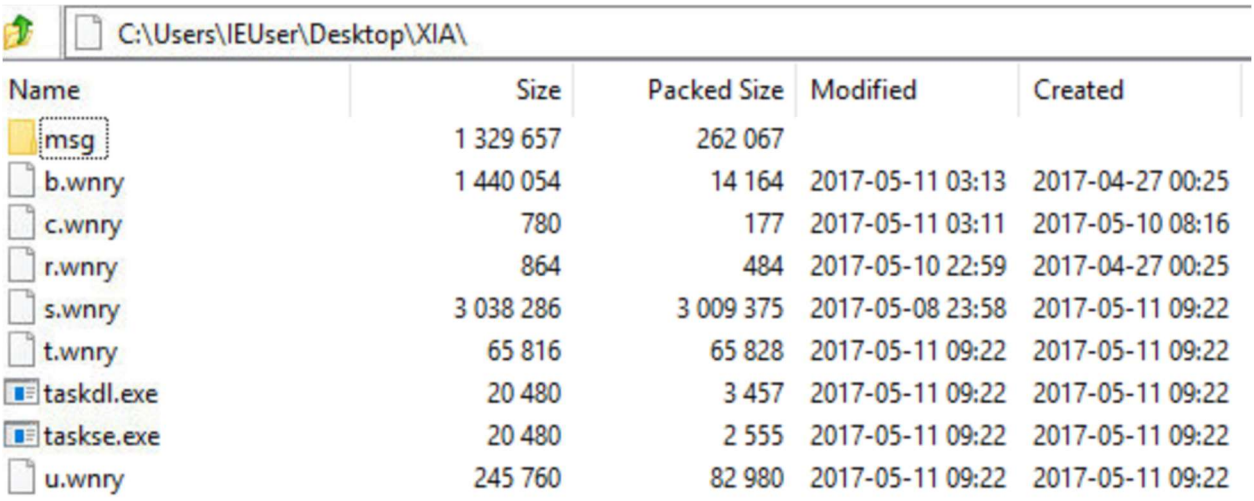
```
0000000000401157 mov     edi, offset ValueName ; "wd"
0000000000401195 push    207h            ; nBufferLength
000000000040119A call    ds:GetCurrentDirectoryA
00000000004011A0 lea     eax, [ebp+Buffer]
00000000004011A6 push    eax             ; Str
00000000004011A7 call    strlen
00000000004011AC pop     ecx
00000000004011AD inc     eax
00000000004011AE push    eax             ; cbData
00000000004011AF lea     eax, [ebp+Buffer]
00000000004011B5 push    eax             ; lpData
00000000004011B6 push    1               ; dwType
00000000004011B8 push    esi             ; Reserved
00000000004011B9 push    edi             ; lpValueName
00000000004011BA push    [ebp+phkResult] ; hKey
00000000004011BD call    ds:RegSetValueExA
00000000004011C3 mov     esi, eax
```

Encryption

The XIA resource of file1 is a ZIP file, whose contents is similar to file3. After the setup above is done, the unzip_XIA function at 0x401DAB is executed. The ransomware will first extract the resource XIA, and then unzip the resource with the password “WNcry@2ol7”.

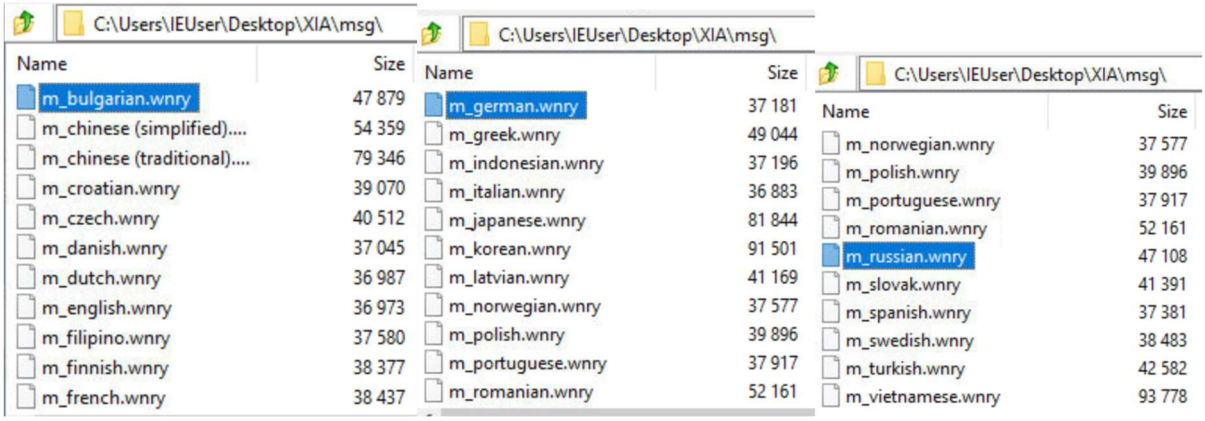
```
00000000004020C3 call    setup
00000000004020C8 mov     [esp+6F4h+var_6F4], offset aWncry2ol7 ; "WNcry@2ol7"
00000000004020CF push    ebx             ; hModule
00000000004020D0 call    unzip_XIA
00000000004020D5 call    write_config
```

The contents of the zip file are shown below.



Name	Size	Packed Size	Modified	Created
msg	1 329 657	262 067		
b.wnry	1 440 054	14 164	2017-05-11 03:13	2017-04-27 00:25
c.wnry	780	177	2017-05-11 03:11	2017-05-10 08:16
r.wnry	864	484	2017-05-10 22:59	2017-04-27 00:25
s.wnry	3 038 286	3 009 375	2017-05-08 23:58	2017-05-11 09:22
t.wnry	65 816	65 828	2017-05-11 09:22	2017-05-11 09:22
taskdl.exe	20 480	3 457	2017-05-11 09:22	2017-05-11 09:22
tasksе.exe	20 480	2 555	2017-05-11 09:22	2017-05-11 09:22
u.wnry	245 760	82 980	2017-05-11 09:22	2017-05-11 09:22

The msg folder contains the ransom message in various languages.

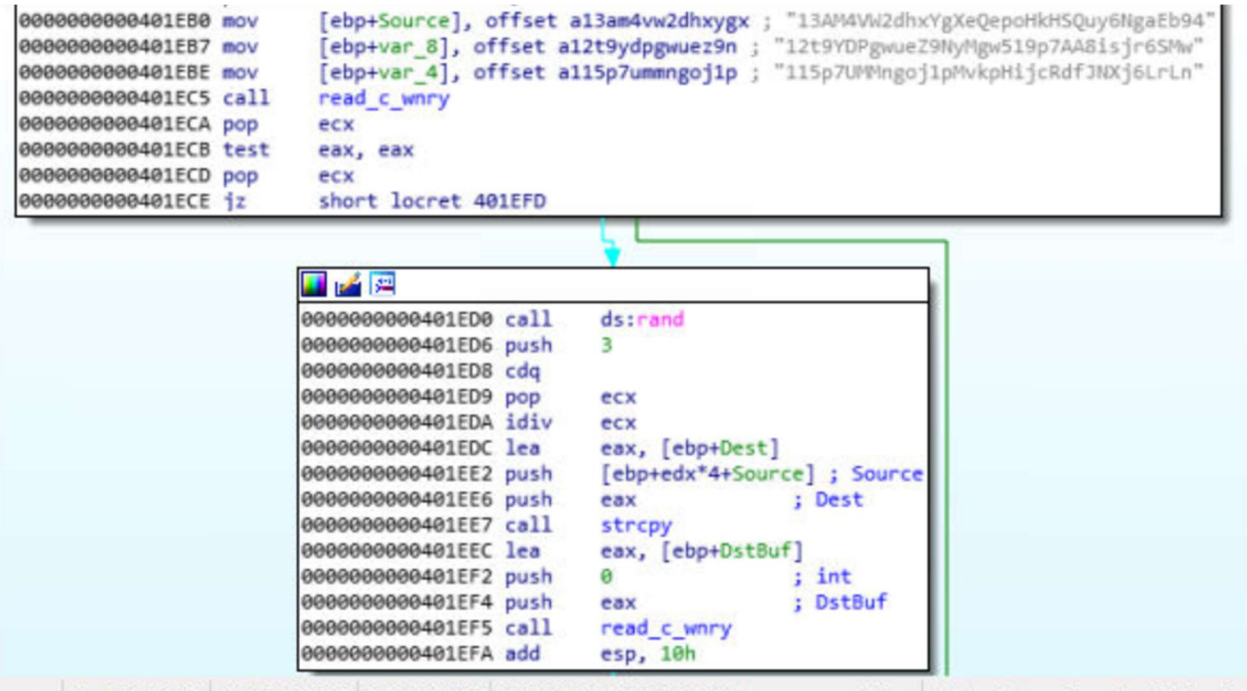


Once the file is unzipped, the c.wnry file is used and loaded in memory.

```
0000000000401E4F lea     eax, [ebp+Str1]
0000000000401E55 push    offset c_wnry_file ; "c.wnry"
0000000000401E5A push    eax                ; Str1
0000000000401E5B call    strcmp
0000000000401E60 add     esp, 14h
0000000000401E63 test    eax, eax
0000000000401E65 jnz     short loc_401E79
```

```
0000000000401E67 lea     eax, [ebp+Str1]
0000000000401E6D push    eax                ; lpFileName
0000000000401E6E call    ds:GetFileAttributesA
```

The “encryption” function at 0x401E9E is called. The function is shown below.



One of the three bitcoin addresses shown is chosen. The information is then saved into c.wnry. c.wnry also contains multiple tor sites.This file is likely a configuration file.

```
eg000:000000B4 a5p7ummngoj1pmv db '5p7UMMngoj1pMvkpHijcRdfJNXj6LrLn',0
eg000:000000D5                 align 4
eg000:000000D8                 dd 3 dup(0)
eg000:000000E4 aGx7ekbenv2riuc db 'gx7ekbenv2riucmf.onion;57g7spgrzlojinas.onion;xxlvbrloxvriy2c5.or
eg000:000000E4                 db 'ion;76jdd2ir2embyv47.onion;cwwnhwhlz52maqm7.onion;',0
eg000:00000158                 dd 21h dup(0)
```

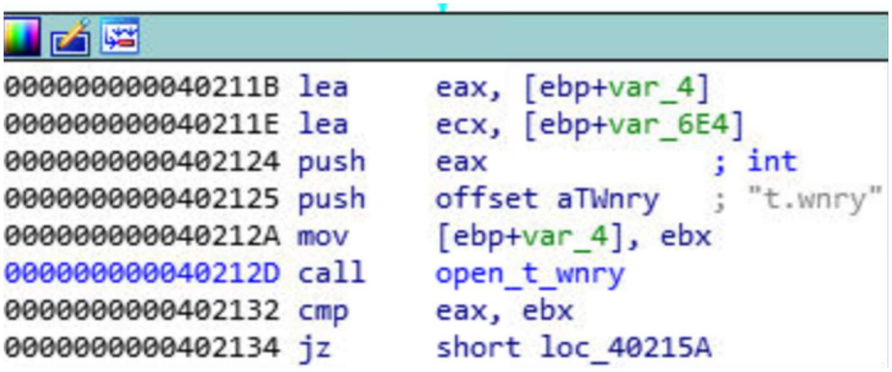
A helper method create_process is then called. The helper process makes calls to CreateProcessA.

```
00000000004020DA push    ebx             ; lpExitCode
00000000004020DB push    ebx             ; dwMilliseconds
00000000004020DC push    offset CommandLine ; "attrib +h ."
00000000004020E1 call    create_process
00000000004020E6 push    ebx             ; lpExitCode
00000000004020E7 push    ebx             ; dwMilliseconds
00000000004020E8 push    offset aIcaclsGrantEve ; "icacls . /grant Everyone:F /T /C /Q"
```

“attrib +h .“ - this command sets the hidden attribute on the current directory, making the directory a hidden directory.

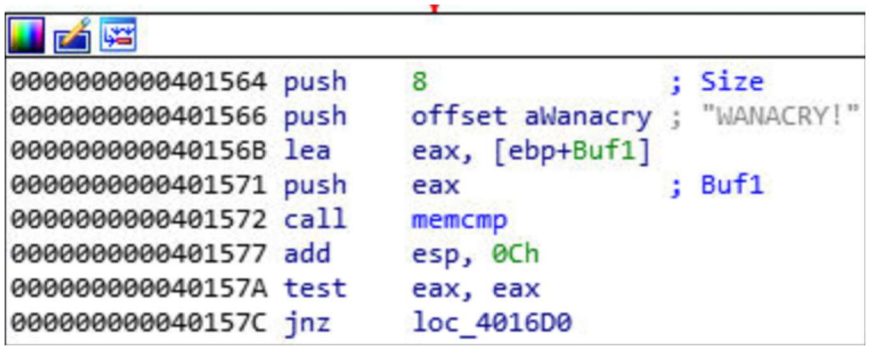
“icacls . /grant Everyone:F /T /C /Q” - this command grants Everyone all the permissions to the current directory and all the subdirectories.

The file t.wnry is opened next.



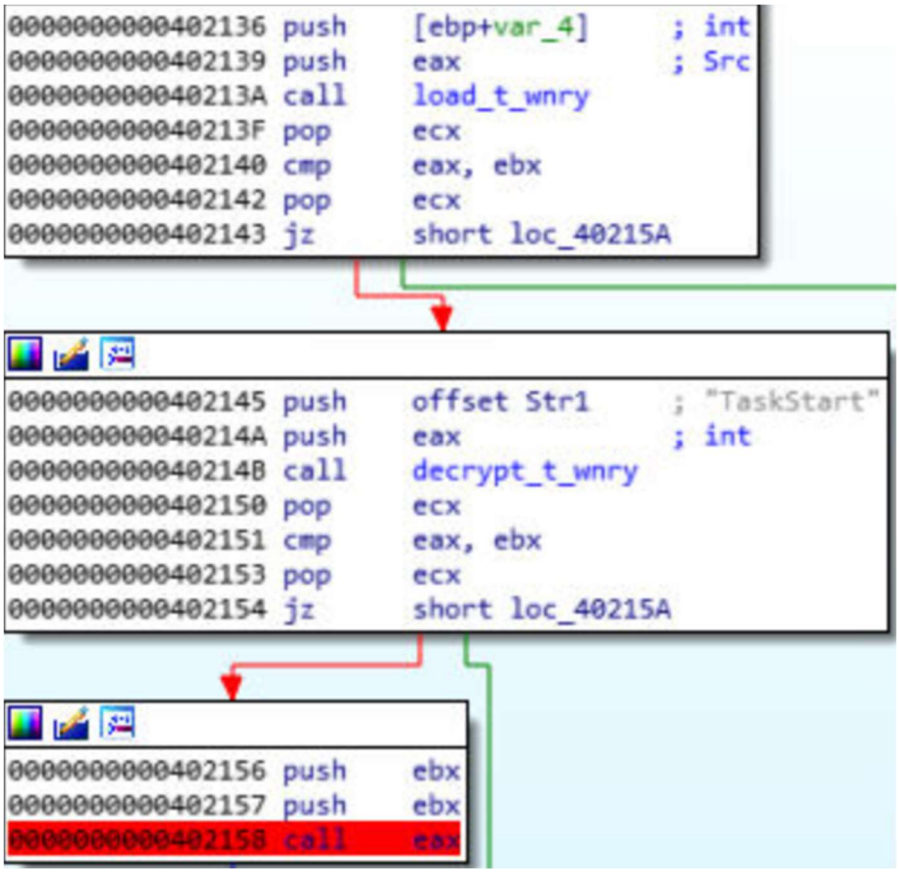
```
000000000040211B lea     eax, [ebp+var_4]
000000000040211E lea     ecx, [ebp+var_6E4]
0000000000402124 push    eax             ; int
0000000000402125 push    offset aTWnry   ; "t.wnry"
000000000040212A mov     [ebp+var_4], ebx
000000000040212D call    open_t_wnry
0000000000402132 cmp     eax, ebx
0000000000402134 jz      short loc_40215A
```

The malware checks that the first 8 bytes of t.wnry matches “WANACRY!”



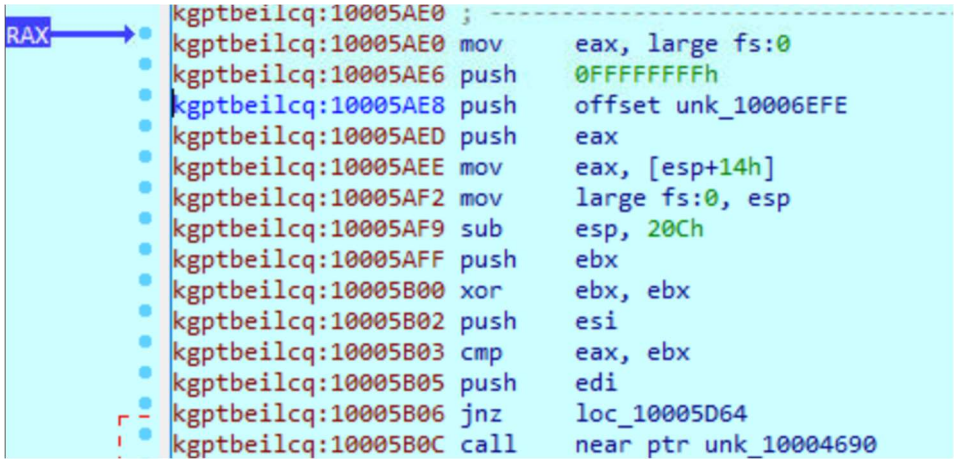
```
0000000000401564 push    8               ; Size
0000000000401566 push    offset aWanacry ; "WANACRY!"
000000000040156B lea     eax, [ebp+Buf1]
0000000000401571 push    eax             ; Buf1
0000000000401572 call    memcmp
0000000000401577 add     esp, 0Ch
000000000040157A test    eax, eax
000000000040157C jnz     loc_4016D0
```

Other than the magic 8 bytes, there is nothing else in t.wnry immediately intelligible. This means that the file is encrypted. The file is decrypted in memory, and the result is a DLL. From the DLL, the TaskStart export is called.



When the call eax at 0x402158 is called, the ransomware will do the encryption process. To analyse the behaviour, we retrieve the value of eax at that point first.

We dynamically run until the call instruction. At that point, the encryption code is decrypted and in memory. We then jump to that location and view the code.



The code jumps to 0x10004690, which creates the mutex MsWinZonesCacheCounterMutexA. The error value is then retrieved and the mutex handle is closed.

```
kgptbeilcq:10004690
kgptbeilcq:10004690 loc_10004690:                          ; CODE XREF: kgptbeilcq:10005B0C↓p
kgptbeilcq:10004690 push    esi
kgptbeilcq:10004691 push    offset aMswinzonescach          ; "MsWinZonesCacheCounterMutexA"
kgptbeilcq:10004696 push    1
kgptbeilcq:10004698 push    0
kgptbeilcq:1000469A call    off_10007094
kgptbeilcq:100046A0 mov     esi, eax
kgptbeilcq:100046A2 test    esi, esi
kgptbeilcq:100046A4 jz      short loc_100046C1
kgptbeilcq:100046A6 call    off_100070F4
kgptbeilcq:100046AC cmp     eax, 0B7h
kgptbeilcq:100046B1 jnz     short loc_100046C1
kgptbeilcq:100046B3 push    esi
kgptbeilcq:100046B4 call    off_100070F8
kgptbeilcq:100046BA mov     eax, 1
kgptbeilcq:100046BF pop     esi
kgptbeilcq:100046C0 retn
kgptbeilcq:100046C1 ; ---------------------------------------------------------------------------
kgptbeilcq:10007094 off_10007094 dd offset kernel32_CreateMutexA
kgptbeilcq:100070F4 off_100070F4 dd offset kernel32_GetLastError
kgptbeilcq:100070F8 off_100070F8 dd offset kernel32_CloseHandle
```

The file c.wnry from the current directory is opened and read. (Renamed the functions msvcrt_fopen and mscvrt_fread).

```
kgptbeilcq:10001016 loc_10001016:                          ; CODE XR
kgptbeilcq:10001016 push    offset aCWnry                  ; "c.wnry"
kgptbeilcq:1000101B call    msvcrt_fopen
kgptbeilcq:10001021 mov     esi, eax
kgptbeilcq:10001023 add     esp, 8
kgptbeilcq:10001026 test    esi, esi
kgptbeilcq:10001028 jz      short loc_1000105F
kgptbeilcq:1000102A push    esi
kgptbeilcq:1000102B push    1
kgptbeilcq:1000102D test    edi, edi
kgptbeilcq:1000102F push    30Ch
kgptbeilcq:10001034 jz      short loc_10001043
kgptbeilcq:10001036 mov     eax, [esp+18h]
kgptbeilcq:1000103A push    eax
kgptbeilcq:1000103B call    msvcrt_fread
kgptbeilcq:10001041 jmp     short loc_1000104E
kgptbeilcq:10001043 ; ---------------------------------------------
```

The encryption and decryption files are created.

```
kgptbeilcq:10005BB0 push    offset a08xRes                  ; "%08X.res"
kgptbeilcq:10005BB5 push    offset unk_1000DCF0
kgptbeilcq:10005BBA call    esi ; msvcrt_sprintf
kgptbeilcq:10005BBC add     esp, 0Ch
kgptbeilcq:10005BBF push    ebx
kgptbeilcq:10005BC0 push    offset a08xPky                  ; "%08X.pky"
kgptbeilcq:10005BC5 push    offset unk_1000DD24
kgptbeilcq:10005BCA call    esi ; msvcrt_sprintf
kgptbeilcq:10005BCC add     esp, 0Ch
kgptbeilcq:10005BCF push    ebx
kgptbeilcq:10005BD0 push    offset a08xEky                  ; "%08X.eky"
kgptbeilcq:10005BD5 push    offset unk_1000DD58
kgptbeilcq:10005BDA call    esi ; msvcrt_sprintf
```

The mutex MsWinZonesCacheCounterMutexW is accessed.

```
kgptbeilcq:10004600 loc_10004600:                           ; CODE XREF: kgptbeilcq:10005BDD↓p
kgptbeilcq:10004600 sub     esp, 64h
kgptbeilcq:10004603 push    esi
kgptbeilcq:10004604 push    offset aGlobalMswinzon_0        ; "Global\\MsWinZonesCacheCounterMutexW"
kgptbeilcq:10004609 push    1
kgptbeilcq:1000460B push    100000h
kgptbeilcq:10004610 call    kernel32_OpenMutexA_0
kgptbeilcq:10004616 test    eax, eax
kgptbeilcq:10004618 jz      short loc_1000462B
kgptbeilcq:1000461A push    eax
kgptbeilcq:1000461B call    kernel32_CloseHandle_0
```

Afterwards, the following files are created in the current directory.

- 00000000.res
- 00000000.pky
- 00000000.eky
- 00000000.dky
- @Please_Read_Me@.txt
- @WanaDecryptor@.exe
- @WanaDecryptor@.bmp
- TaskData (empty folder)
- 216931606739926.bat (later deleted)
- 216931606739926.bat.WNCRY (the number can change)
- M.vbs (later deleted)
- m.vbs.WNCRY

During this process, multiple commands are entered. For instance, the malware adds a hidden subdirectory “$RECYCLE” to drives it wishes to encrypt.

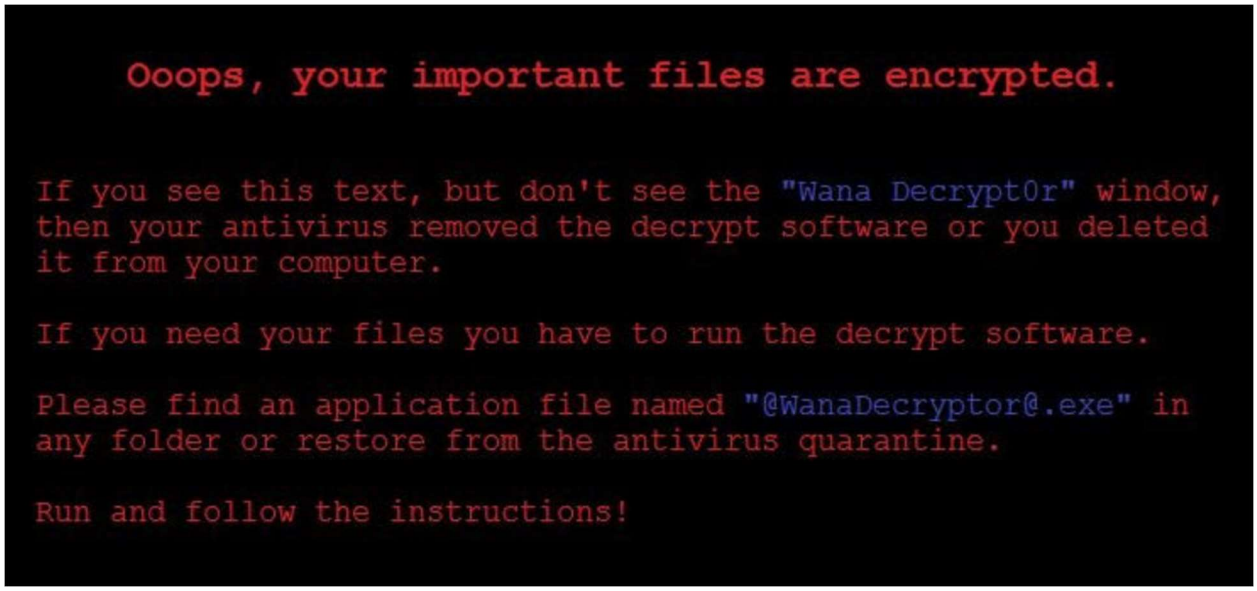
```
debug018:009E2028 aAttribHSCS db 'attrib +h +s %C:\%s',0
debug018:009E203C aRecycle db '$RECYCLE',0
```

Traces of commands can also be found in the memory.

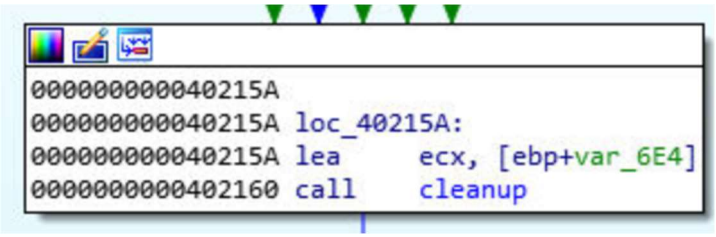
```
kgptbeilcq:1000D7C8 aCmdExeCStartBS db 'cmd.exe /c start /b %s vs',0
kgptbeilcq:1000D7E2 db    0
kgptbeilcq:1000D7E3 db    0
kgptbeilcq:1000D7E4 aSCo db '%s co',0
kgptbeilcq:1000D7EA db    0
kgptbeilcq:1000D7EB db    0
kgptbeilcq:1000D7EC aTaskkillExeFIm db 'taskkill.exe /f /im mysqld.exe',0
kgptbeilcq:1000D80B db    0
kgptbeilcq:1000D80C aTaskkillExeFIm_0 db 'taskkill.exe /f /im sqlwriter.exe',0
kgptbeilcq:1000D82E db    0
kgptbeilcq:1000D82F db    0
kgptbeilcq:1000D830 aTaskkillExeFIm_1 db 'taskkill.exe /f /im sqlserver.exe',0
kgptbeilcq:1000D852 db    0
kgptbeilcq:1000D853 db    0
kgptbeilcq:1000D854 aTaskkillExeFIm_2 db 'taskkill.exe /f /im MSExchange*',0
kgptbeilcq:1000D874 aTaskkillExeFIm_3 db 'taskkill.exe /f /im Microsoft.Exchange.*',0
```

Afterwards, the ransom message appears. This ransom message in a “WanaDecrypt0r 2.0” window periodically pops up again.

The desktop background is changed as well.

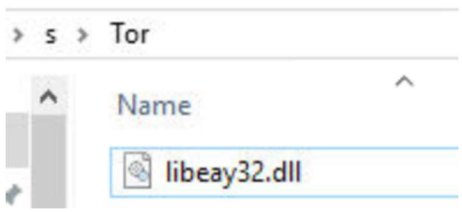
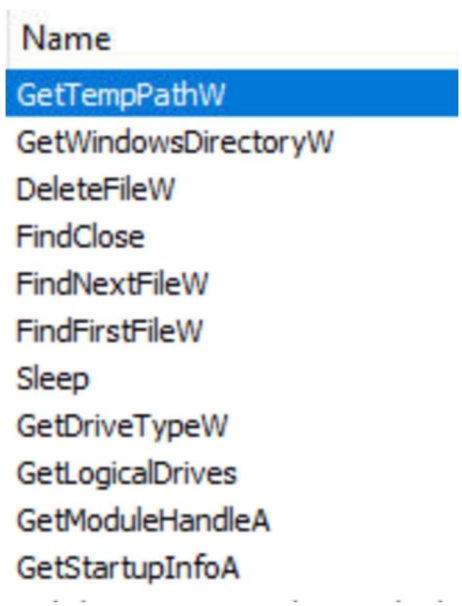


The entirety of the “encryption mechanics” is only loaded into memory and decrypted at runtime. After the code in the imported DLL runs (via call eax at 0x402158), the cleanup happens. The cleanup involves calling GlobalFree and DeleteCriticalSection to clear the mutexes and shared objects. The malware then exits here. However, the service still remains running and can ensure persistence of its behaviour, including propagation and encrypting new drives.



We analyse the remaining files in the XIA zip resource.

File	MD5	Notes
b.wnry	c17170262312f3be7027bc2ca825bf0c	Desktop image
c.wnry	ae08f79a0d800b82fcbe1b43cdbdbefc	Configuration file
r.wnry	3e0020fc529b1c2a061016dd2469ba96	Ransom note
s.wnry	30ff54edefdfec2b1d3f2c61bfcc14e1	ZIP file containing libeay32.dll. Used in interacting with tor server.

		
t.wnry	5dcaac857e695a65f5c3ef1441a73a8f	Encryption code
u.wnry	7bf2b57f2a205768755c07f238fb32cc	Wana Decrypt0r 2.0 exe tool
taskdl.exe	4fef5e34143e646dbf9907c4374276f5	 Contains imports interacting with the file system.
taskse.exe	8495400f199ac77853c53b5a3f278f3e	Dynamically loads and calls wtsapi32.dll

CS4238 Malware Analysis Report - NotPetya

Joyce Yeo Shuhui A0171426J
Jonathan Cheng A0121749A

Date submitted: 1 Dec 2020

Basic Information

Name	f593b73c91003518c20cdc8be04f3a1f8a68ca3ded04700f675a543ac278ab07.bin [“file4”]
Size	353 KB
Type	PE32 executable (GUI) Intel 80386, for MS Windows
MD5 [HashCalc]	dba9c11e2f0b6a3ded91c9c87ce79f72
SHA1 [HashCalc]	da39a3ee5e6b4b0d3255bfef95601890afd80709
Compiled [PE View]	Sun Jun 18 07:14:36 2017 UTC
Packers [PEiD]	None found
Imported DLLs [PE View]	KERNEL32, USER32, ADVAPI32, SHELL32, OLE32, CRYPT32, SHLWAPI, IPHLPAPI, WS2_32, MPR, NETAPI32, DHCPSAPI, MSVCRT

PE Sections [PE Studio]

Name	MD5	Raw Size (bytes)	Entropy
.text	fad30112bbae268f2e473c7b35585828	48640	6.547
.rdata	46418e52b546c1f696eb8a524f18c56e	34304	6.992
.data	5216f0c62d1fd41b1d558e129e18d0fe	20992	5.427
.rsrc	f07e68575f50a62382d99e182baa05d5	247808	7.998

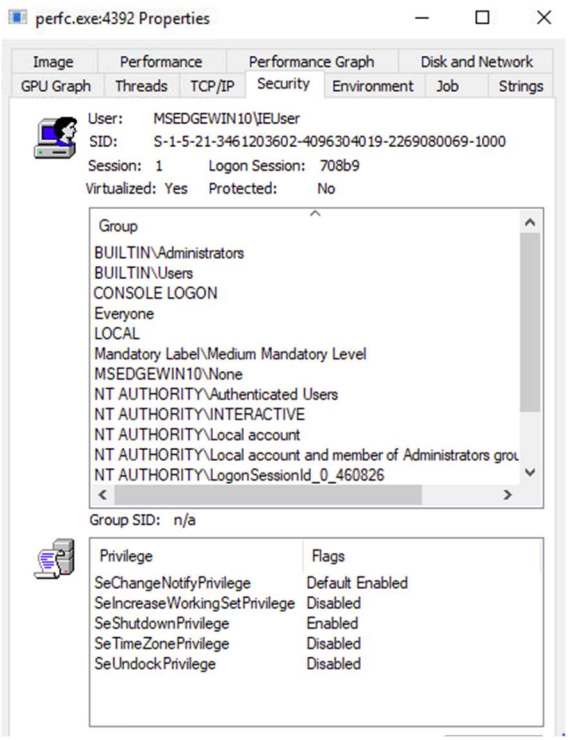
.reloc	c5d1d4cdade7dcfbe14ec10dcf66cfb1	3584	4.772

Resources [Resource Hacker + HashCalc]

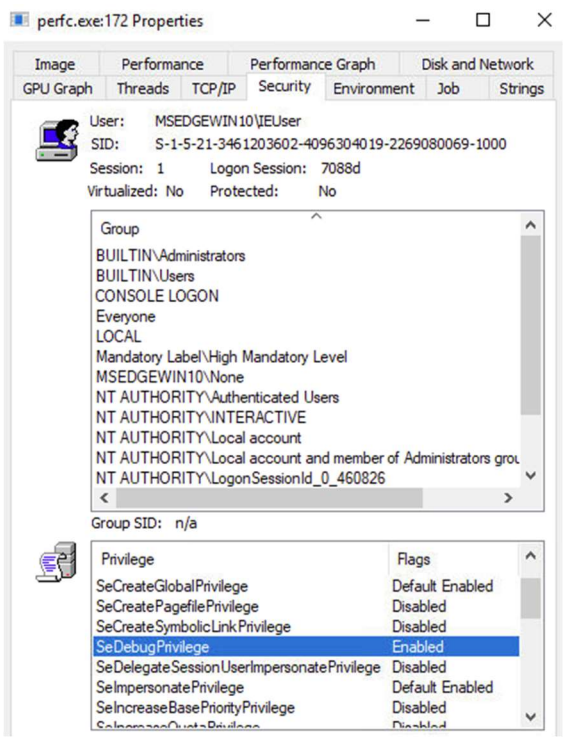
Name	MD5
1	5273a3494495f2278d4415b999debec3
2	ffe1ee50b87d6c569c92e3d847f2fb9a
3	779d952f314c92881abfc4980a7269ea
4	f56fe3b66610dea29473d997792f1ac6

Dynamic Analysis

A common property stated by most reports on NotPetya is that its behaviour is dependent on the privileges it is run with. Therefore, basic dynamic analysis was done twice: once with administrator privileges and once without. The binary was also renamed *perfc.exe* to be consistent with most other reports.



***Without** admin privileges*

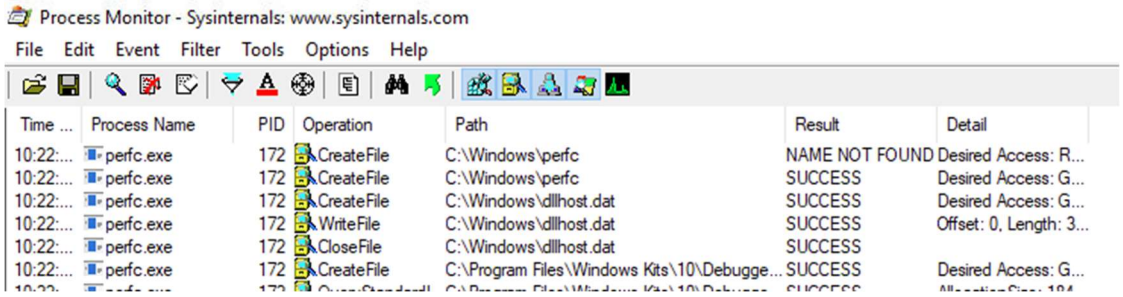


***With** admin privileges*

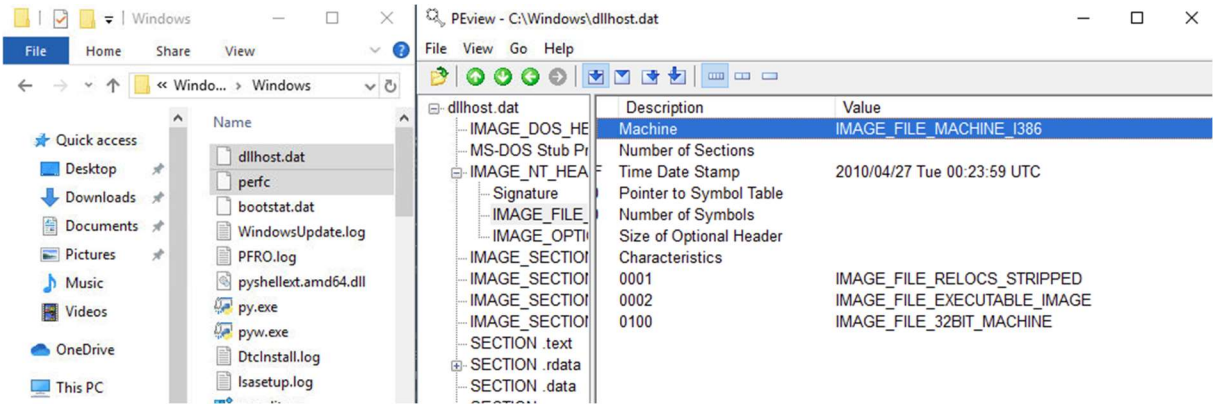
Disk

<u>Admin</u>

Two files commonly brought up in reports are these two shown below; *perfc* and *dllhost.dat*.



dllhost.dat is a PE executable, verified by running it through PEview, while perfc is an empty file.



Another behaviour to verify from reports is the writing to the boot records and file tables. Below shows the sequence of events that corroborates this:

10:22:...	perfc.exe	172	CreateFile	C:\Windows\perfc	NAME NOT FOUND Desired Access: Read Attributes, Disposition: O...
10:22:...	perfc.exe	172	CreateFile	C:\Windows\perfc	SUCCESS Desired Access: Generic Write, Read Attributes,...
10:22:...	perfc.exe	172	CreateFile	C:	SUCCESS Desired Access: Generic Write, Read Attributes,...
10:22:...	perfc.exe	172	DeviceIoControl	C:	SUCCESS Control: IOCTL_DISK_GET_DRIVE_GEOMETRY
10:22:...	perfc.exe	172	WriteFile	C:	SUCCESS Offset: 512, Length: 512, I/O Flags: Non-cache...

Getting disk geometry

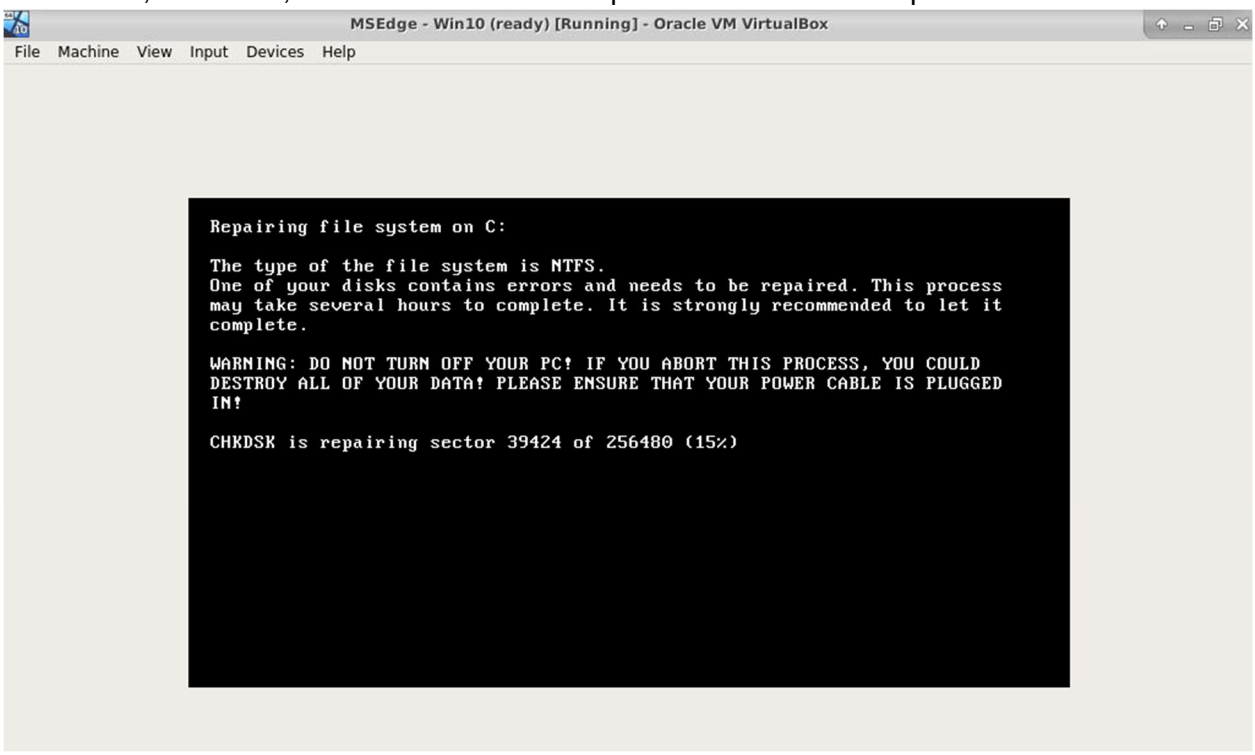
10:22:...	perfc.exe	172	WriteFile	C:\$ConvertToNonresident	SUCCESS Offset: 0, Length: 16,384, I/O Flags: Non-cache...
10:22:...	perfc.exe	172	ReadFile	C:	SUCCESS Offset: 0, Length: 344, I/O Flags: Non-cached, ...
10:22:...	perfc.exe	172	WriteFile	C:\Users\IEUser\AppData\Local\Packages\Microsoft....	SUCCESS Offset: 0, Length: 49,152, I/O Flags: Non-cache...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 8,192, Length: 4,096, I/O Flags: Non-ca...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 0, Length: 4,096, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 282,624, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 675,840, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 20,480, Length: 4,096, I/O Flags: Non-c...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 196,608, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 286,720, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 28,672, Length: 4,096, I/O Flags: Non-c...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 110,592, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 131,072, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:	SUCCESS Offset: 16,384, Length: 4,096, I/O Flags: Non-c...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 0, Length: 8,192, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 0, Length: 8,192, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 0, Length: 4,096, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 8,192, Length: 4,096, I/O Flags: Non-ca...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 110,592, Length: 4,096, I/O Flags: Non-...
10:22:...	perfc.exe	172	WriteFile	C:	SUCCESS Offset: 0, Length: 4,096, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:	SUCCESS Offset: 0, Length: 4,096, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	WriteFile	C:\$Directory	SUCCESS Offset: 4,096, Length: 4,096, I/O Flags: Non-ca...
10:22:...	perfc.exe	172	WriteFile	C:	SUCCESS Offset: 0, Length: 4,096, I/O Flags: Non-cached...
10:22:...	perfc.exe	172	CloseFile	C:	SUCCESS
10:22:...	perfc.exe	172	RegOpenKey	HKLM\System\CurrentControlSet\Services\FileInfo\IN...	REPARSE Desired Access: Read

Writing to file table

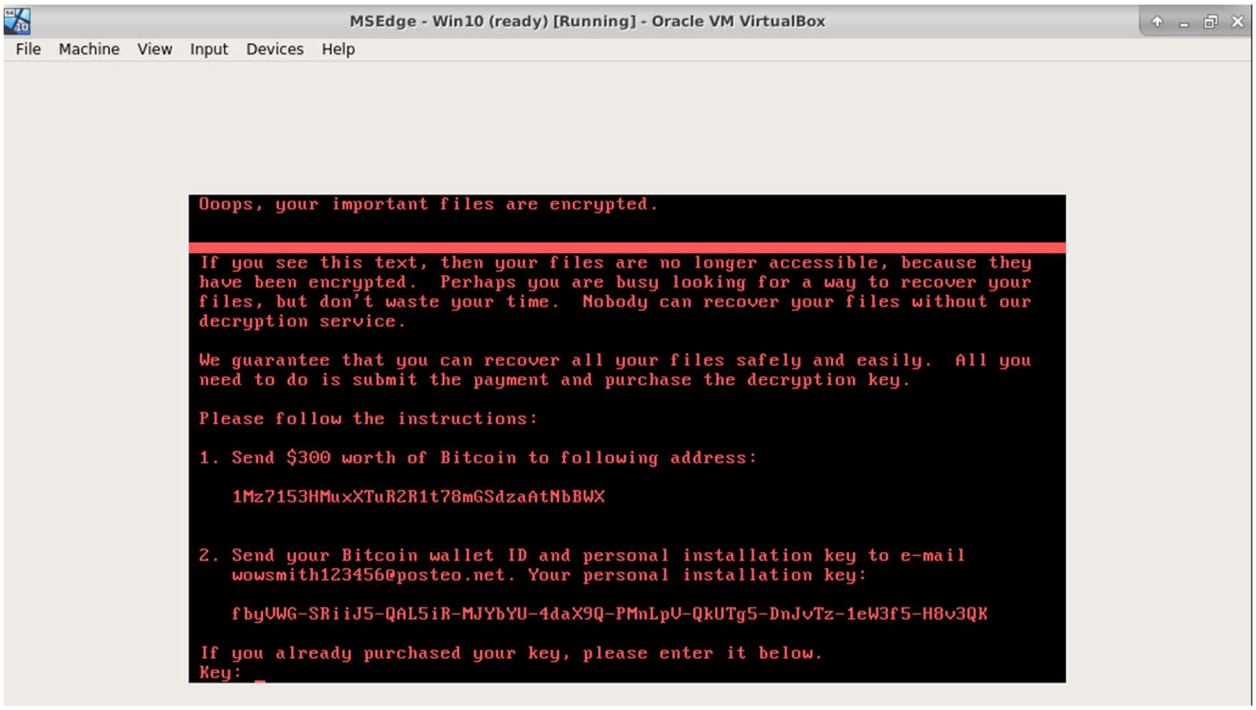
10:22:...	perfc.exe	7552	CreateFile	\Device\Harddisk0\DR0	SUCCESS Desired Access: Generic Read/Write, ...
10:22:...	perfc.exe	7552	WriteFile	\Device\Harddisk0\DR0	SUCCESS Offset: 16,896, Length: 512, I/O Flags: ...
10:22:...	perfc.exe	7552	CloseFile	\Device\Harddisk0\DR0	SUCCESS
10:22:...	perfc.exe	7552	CreateFile	\Device\Harddisk0\DR0	SUCCESS Desired Access: Generic Read/Write, ...
10:22:...	perfc.exe	7552	WriteFile	\Device\Harddisk0\DR0	SUCCESS Offset: 17,408, Length: 512, I/O Flags: ...
10:22:...	perfc.exe	7552	CloseFile	\Device\Harddisk0\DR0	SUCCESS
10:22:...	perfc.exe	7552	RegQueryValue	HKLM\System\CurrentControlSet\Servi...	NAME NOT FOUND Length: 40
10:22:...	perfc.exe	7552	RegQueryValue	HKLM\System\CurrentControlSet\Servi...	NAME NOT FOUND Length: 40
10:22:...	perfc.exe	7552	CreateFile	C:\Windows\dllhost.dat	SUCCESS Desired Access: Generic Write, Read A...
10:22:...	perfc.exe	7552	WriteFile	C:\Windows\dllhost.dat	SUCCESS Offset: 0, Length: 381,816, Priority: Nor...
10:22:...	perfc.exe	7552	CloseFile	C:\Windows\dllhost.dat	SUCCESS
10:22:...	perfc.exe	7552	CreateFile	C:\Program Files\Windows Kits\10\Deb...	SUCCESS Desired Access: Generic Read/Write, ...
10:22:...	perfc.exe	7552	QueryStandardI...	C:\Program Files\Windows Kits\10\Deb...	SUCCESS AllocationSize: 184,320, EndOfFile: 184...
10:22:...	perfc.exe	7552	CreateFileMapp...	C:\Program Files\Windows Kits\10\Deb...	FILE LOCKED WI... SyncType: SyncTypeCreateSection, P...

Writing to boot record

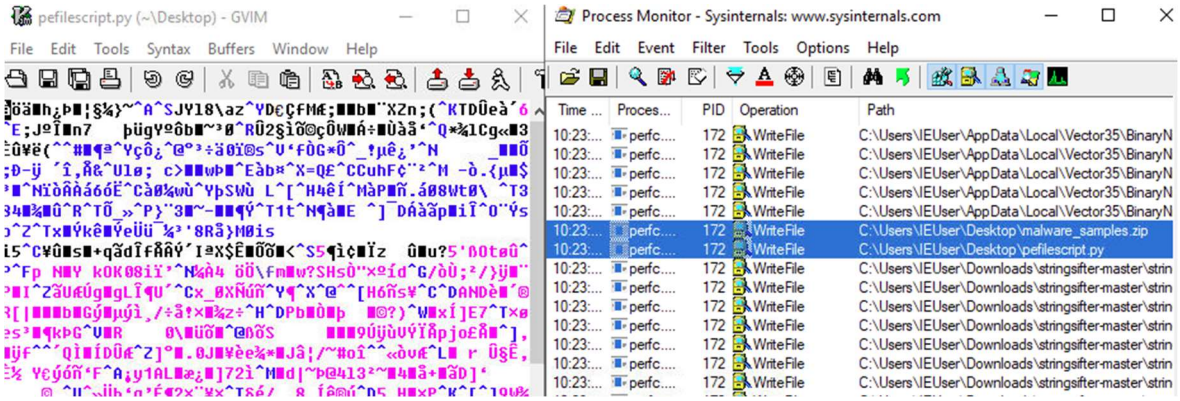
As a result, on reboot, we are faced with the reported fake CHKDSK splash:



Ignoring the warning and restarting the system again causes this splash image to show. It is the ransom note.

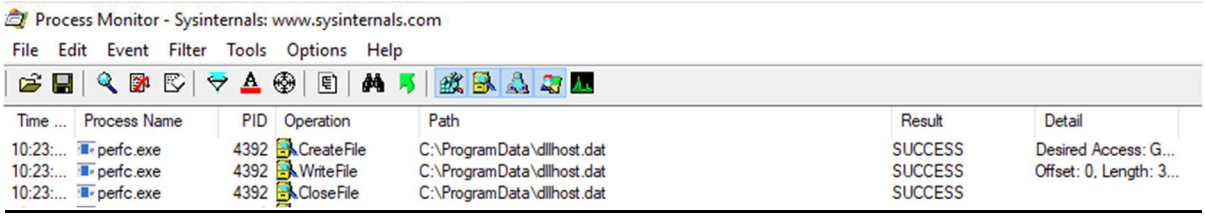


On top of modifying the MBR, the malware also encrypts local files. Below shows the encryption of a local *.py* file on the Desktop.



Non-admin:

The key differences when running without admin privileges is shown below. First, *perfc* is not written anywhere. Second, the directory where *dllhost.dat* is written to is different as well. This is likely because writing to the *C:\Windows* directory requires elevated permissions.

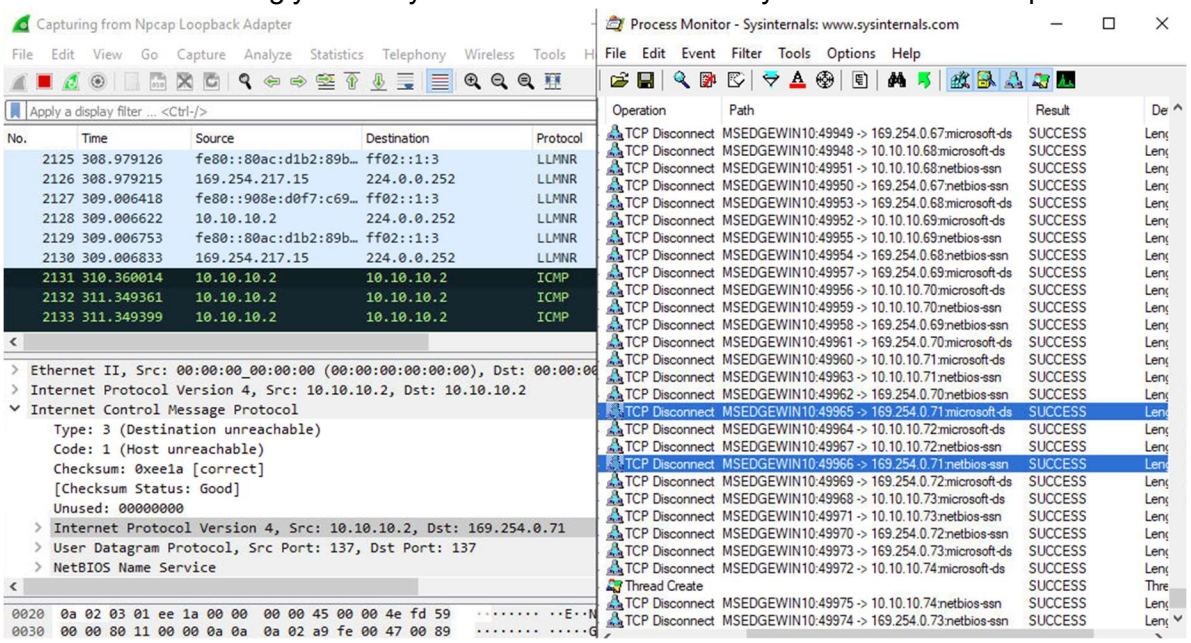


Also, the MBR is not written to. This means that the system can be restarted without facing the fake CHKDSK.

Otherwise, local files are similarly encrypted as in the admin case.

Network Connections

Connections to seemingly arbitrary IP addresses. This is likely how the malware spreads itself.

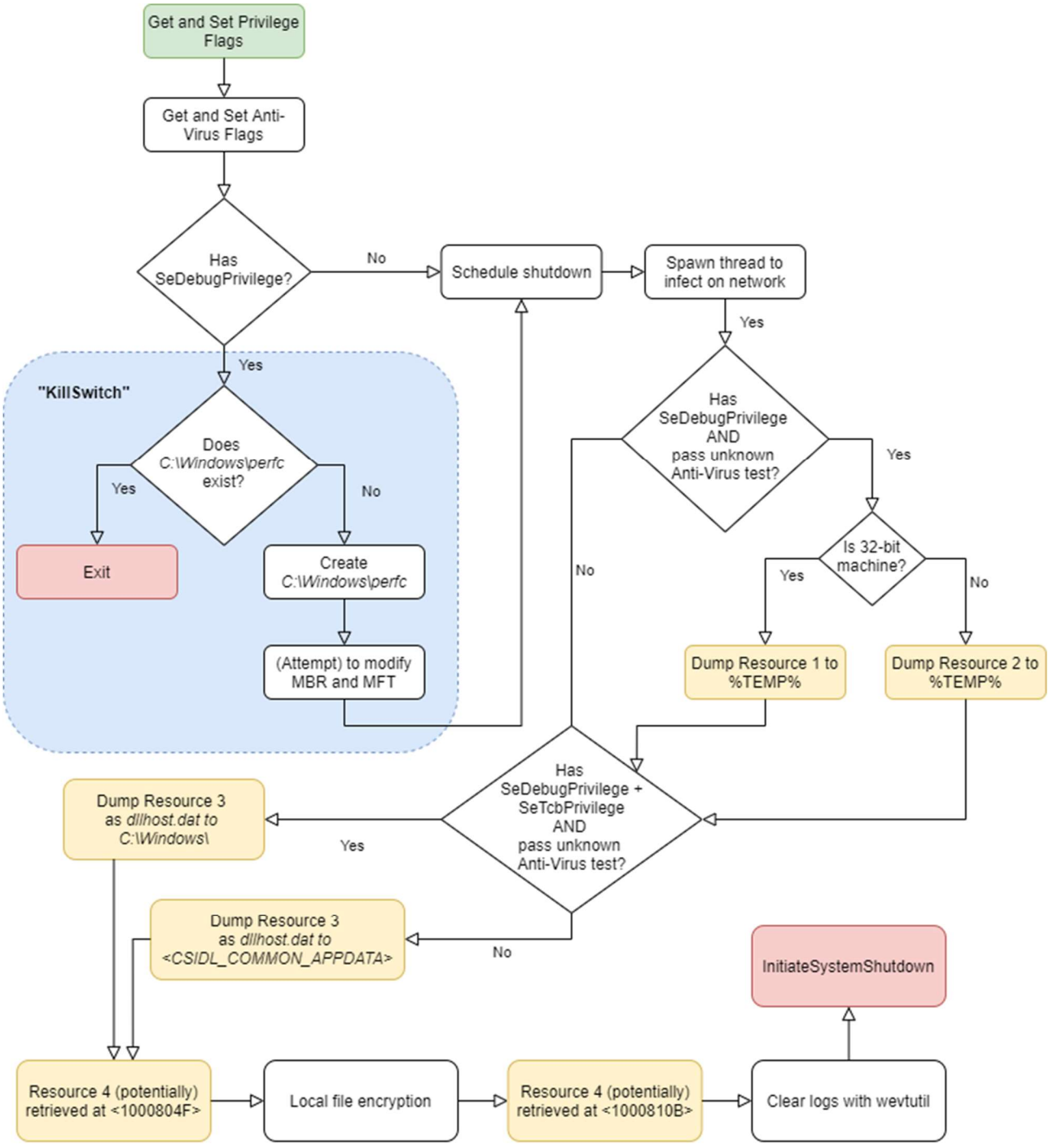


The above image shows the malware attempting to connect to ranges of IP addresses *169.254.0.**. Without faking a network and having other simulated hosts to connect to, not much further analysis can be done, since none of the IP addresses the malware attempts to connect to are available. There is also no perceivable difference between behaviours when run as admin or non-admin.

Binary Analysis

Control Flow Graph

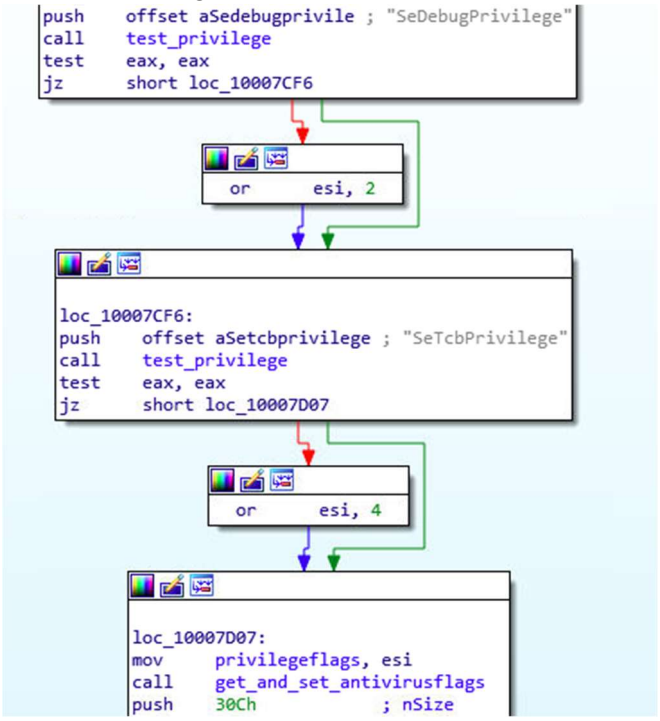
The following is the CFG of the malware based on reports and binary analysis.



The rest of the report will follow closely the execution path of the malware, based on the above.

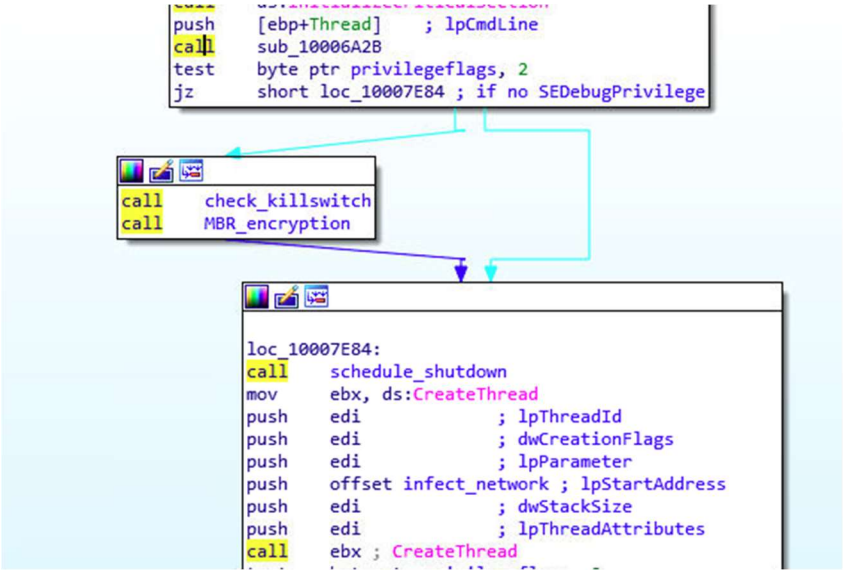
Privilege and Anti-Virus Flags

The malware first retrieves the privileges it is running with. Shown below, privileges are tested using *test_privilege* and stored in the global variable *privilegeflags*.



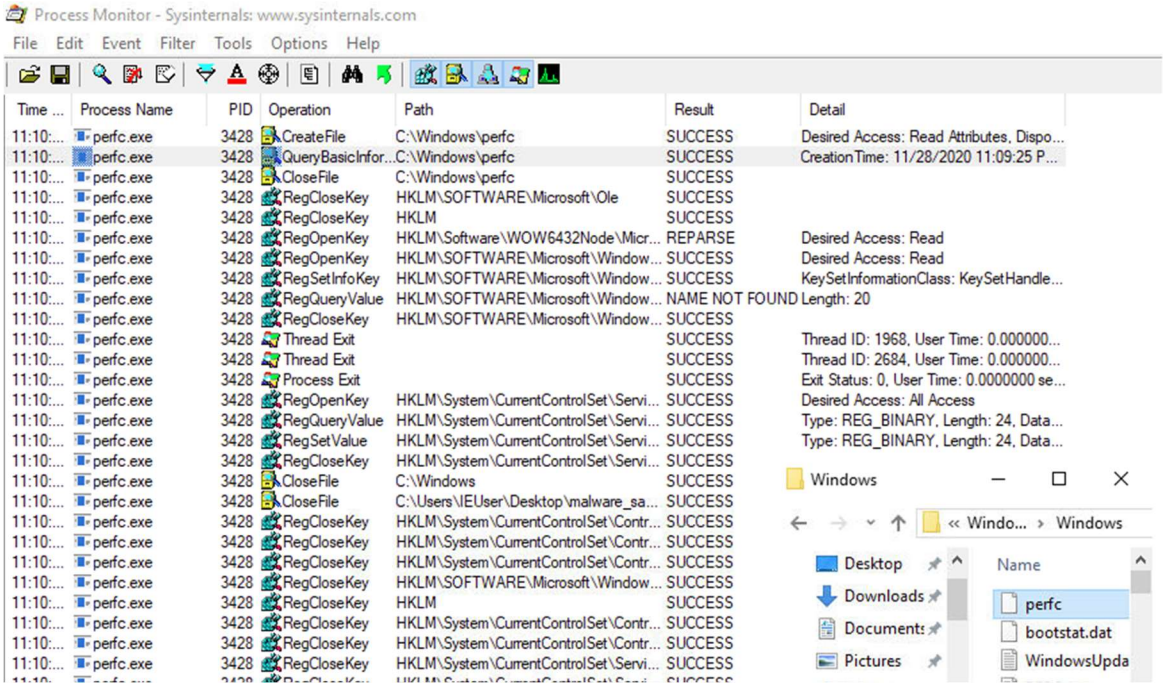
Afterwhich, *get_and_set_antivirusflags* is called to similarly set a global flag (not in image, renamed *antivirusflags*) depending on what anti-viruses are running on the system.

After setting global flags, the malware then checks if it has the SeDebugPrivilege. If it does, it executes the blue highlighted code path. Else, it proceeds with scheduling a shutdown and spawning a thread to spread itself on the network.

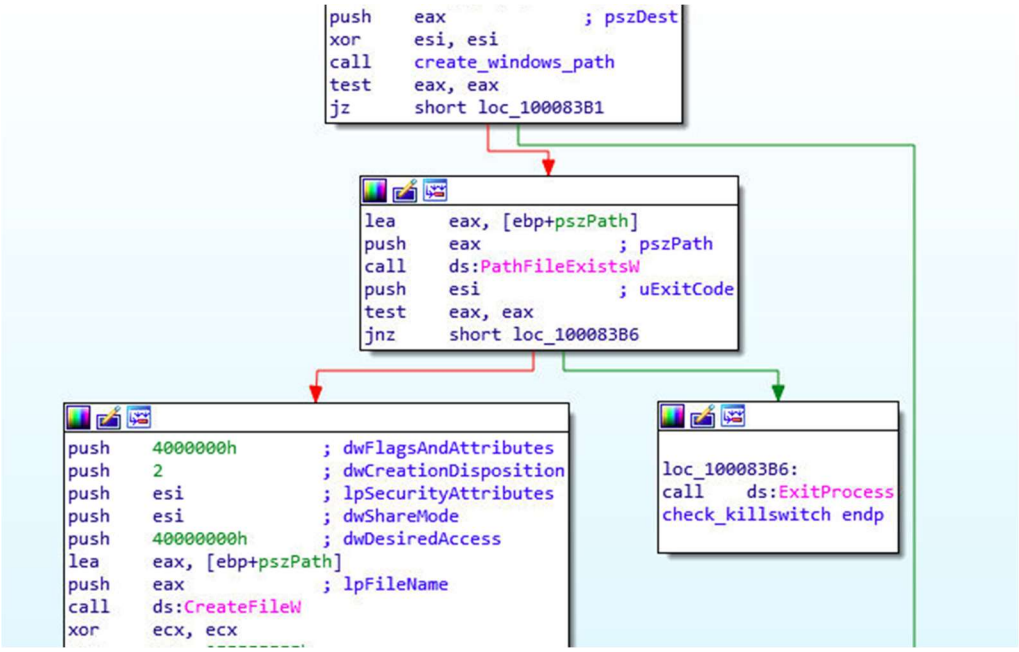


Killswitch

Highlighted in blue is the execution path that checks for the frequently mentioned “killswitch”. Creating a file *C:\Windows\perfc* results in early termination of the malware, as shown below.

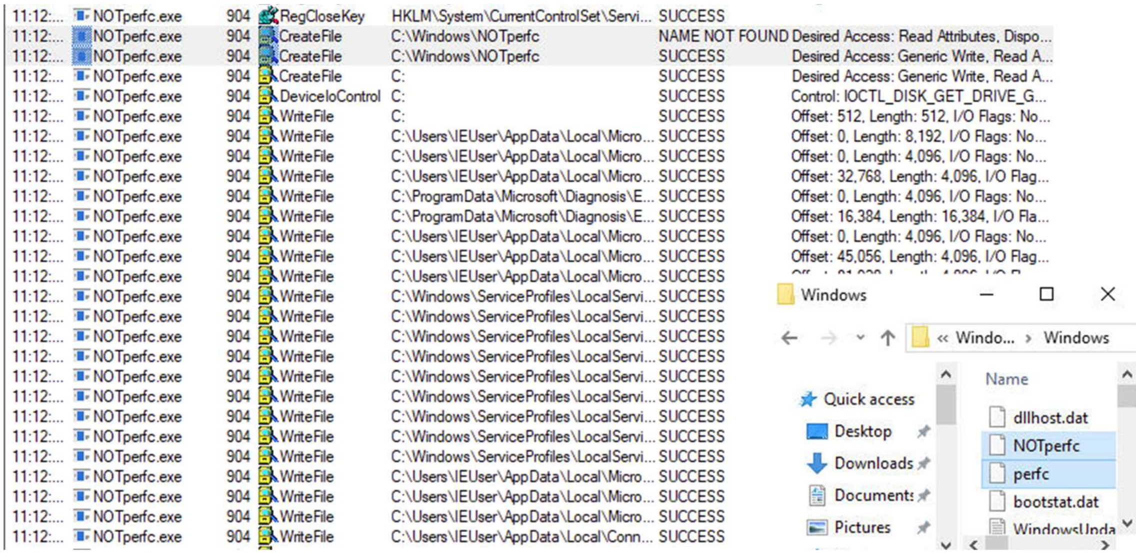


However, the effectiveness of the “killswitch” is limited because the name of the file *perfc* depends on the name of the process the malware is running under.



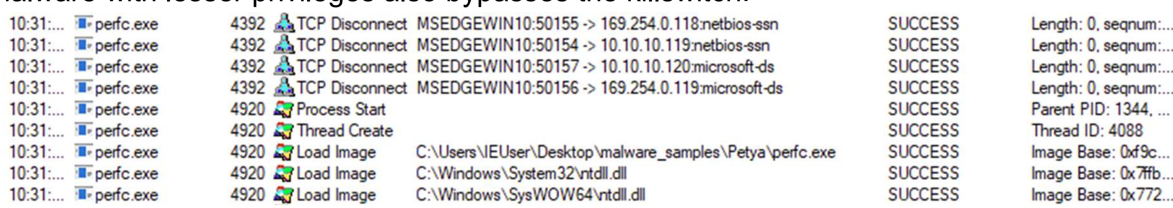
create_windows_path creates a qualified path name from the name of the process. This will be the path and name of the killswitch file.

Renaming the malware executable *NOTperfc.exe* and running it bypasses the “killswitch”, and instead creates another file *C:\Windows\NOTperfc*.



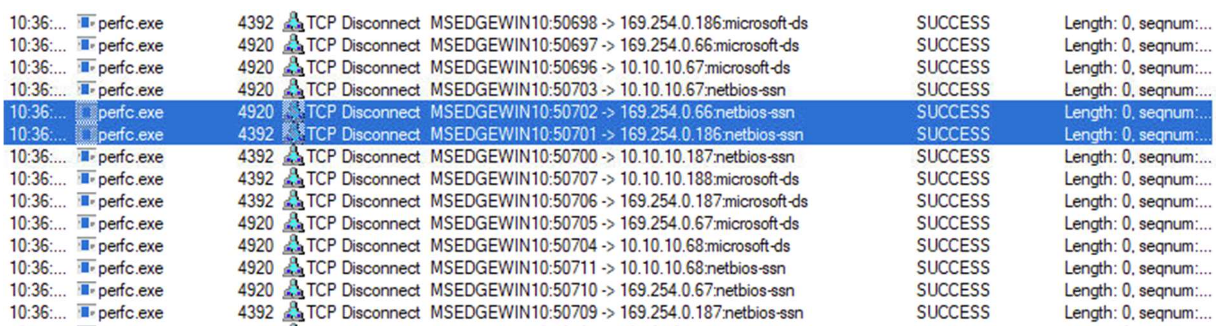
11:12:...	NOTperfc.exe	904	RegCloseKey	HKLM\System\CurrentControlSet\Servi...	SUCCESS	
11:12:...	NOTperfc.exe	904	CreateFile	C:\Windows\NOTperfc	NAME NOT FOUND	Desired Access: Read Attributes, Dispo...
11:12:...	NOTperfc.exe	904	CreateFile	C:\Windows\NOTperfc	SUCCESS	Desired Access: Generic Write, Read A...
11:12:...	NOTperfc.exe	904	CreateFile	C:	SUCCESS	Desired Access: Generic Write, Read A...
11:12:...	NOTperfc.exe	904	DeviceIoControl	C:	SUCCESS	Control: IOCTL_DISK_GET_DRIVE_G...
11:12:...	NOTperfc.exe	904	WriteFile	C:	SUCCESS	Offset: 512, Length: 512, I/O Flags: No...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	Offset: 0, Length: 8,192, I/O Flags: No...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	Offset: 0, Length: 4,096, I/O Flags: No...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	Offset: 32,768, Length: 4,096, I/O Flag...
11:12:...	NOTperfc.exe	904	WriteFile	C:\ProgramData\Microsoft\Diagnosis\E...	SUCCESS	Offset: 0, Length: 4,096, I/O Flags: No...
11:12:...	NOTperfc.exe	904	WriteFile	C:\ProgramData\Microsoft\Diagnosis\E...	SUCCESS	Offset: 16,384, Length: 16,384, I/O Fla...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	Offset: 0, Length: 4,096, I/O Flags: No...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	Offset: 45,056, Length: 4,096, I/O Flag...
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Windows\ServiceProfiles\LocalServi...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Micro...	SUCCESS	
11:12:...	NOTperfc.exe	904	WriteFile	C:\Users\IEUser\AppData\Local\Conn...	SUCCESS	

Another point not often brought up in reports is the fact that the “killswitch” path highlighted in blue is only taken when the malware has the SeDebugPrivilege. This means that running the malware with lesser privileges also bypasses the killswitch.



10:31:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50155 -> 169.254.0.118:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:31:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50154 -> 10.10.10.119:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:31:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50157 -> 10.10.10.120:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:31:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50156 -> 169.254.0.119:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:31:...	perfc.exe	4920	Process Start		SUCCESS	Parent PID: 1344, ...
10:31:...	perfc.exe	4920	Thread Create		SUCCESS	Thread ID: 4088
10:31:...	perfc.exe	4920	Load Image	C:\Users\IEUser\Desktop\malware_samples\Petya\perfc.exe	SUCCESS	Image Base: 0xf9c...
10:31:...	perfc.exe	4920	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7ffb...
10:31:...	perfc.exe	4920	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x772...

Two instances of the malware running without admin privileges, PID4392 and PID4920



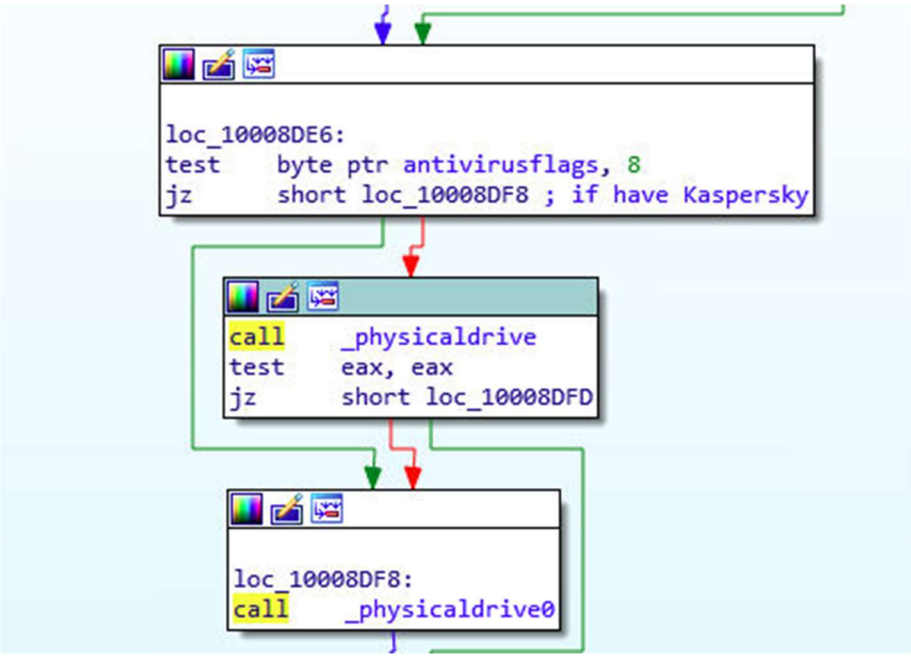
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50698 -> 169.254.0.186:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50697 -> 169.254.0.66:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50696 -> 10.10.10.67:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50703 -> 10.10.10.67:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50702 -> 169.254.0.66:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50701 -> 169.254.0.186:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50700 -> 10.10.10.187:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50707 -> 10.10.10.188:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50706 -> 169.254.0.187:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50705 -> 169.254.0.67:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50704 -> 10.10.10.68:microsoft-ds	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50711 -> 10.10.10.68:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4920	TCP Disconnect	MSEDGEWIN10:50710 -> 169.254.0.67:netbios-ssn	SUCCESS	Length: 0, seqnum:...
10:36:...	perfc.exe	4392	TCP Disconnect	MSEDGEWIN10:50709 -> 169.254.0.187:netbios-ssn	SUCCESS	Length: 0, seqnum:...

In the end, both are still running and attempting to spread on the network.
Would not have been possible if both were ran with admin privileges

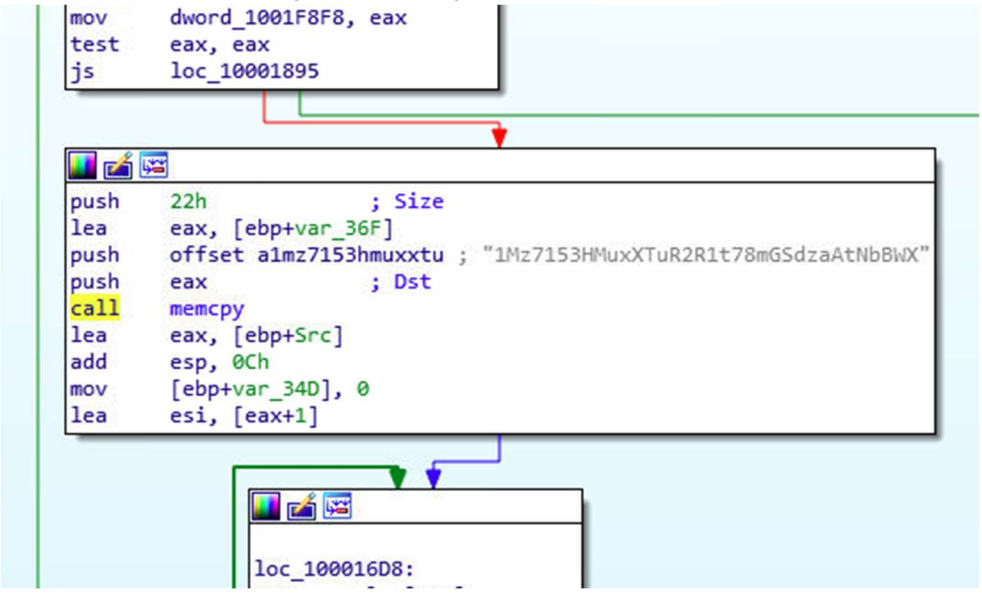
To this end, it seems like the “killswitch” is less of a killswitch, but more of a mutex to prevent two instances of the malware to perform modifications to the boot records concurrently in the next function called.

MBR Modifications

Inside the function *MBR_encryption*, we have a test for the *antivirusflags*. In particular, based on other reports, if *Kaspersky* is running on the system then it skips the function call to *_physicaldrive*.



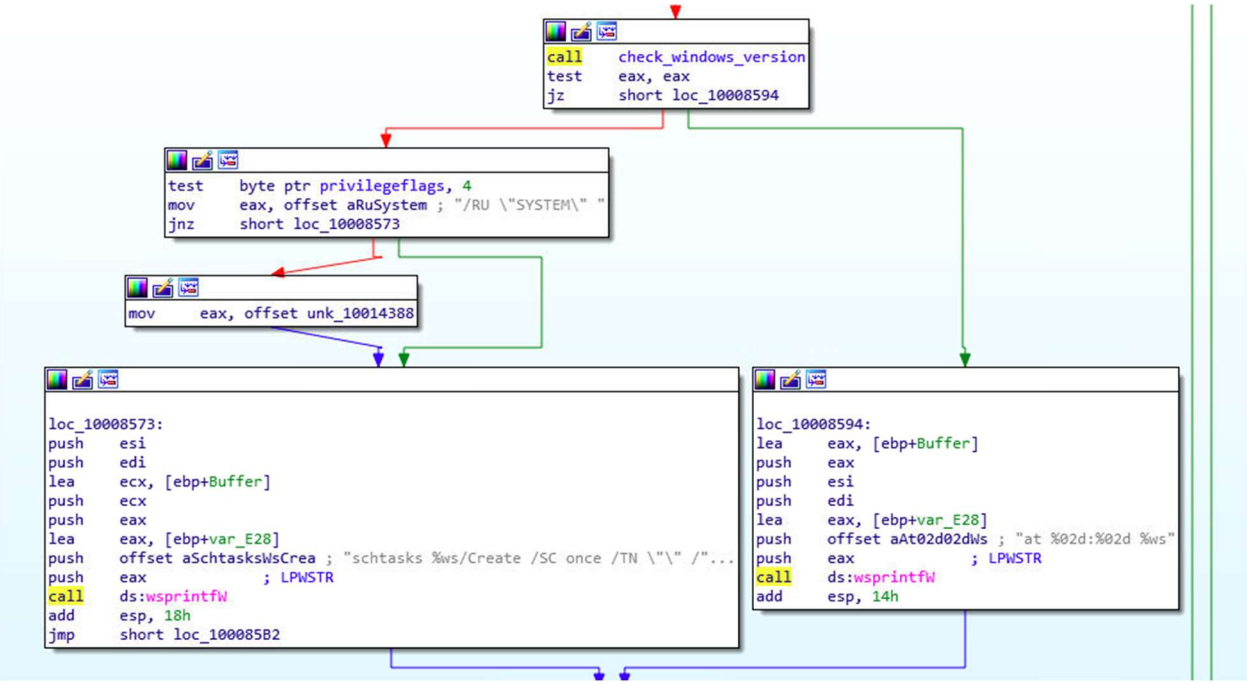
_physicaldrive seems to be the function that modifies the MBR. This guess is made because of the use of the following string "*1Mz7153...*" inside the function. Referring back to the ransom note after the fake CHKDSK in Dynamic Analysis: Disk, this is the Bitcoin address.



At this point, further analysis is required to determine what *_physicaldrive0* does.

Schedule Shutdown

The way the shutdown is scheduled depends on the version of Windows the system is running. This is checked using the *check_windwos_version* function. The right branch is taken when running an old Windows system like XP, while the left branch is taken when running anything newer.

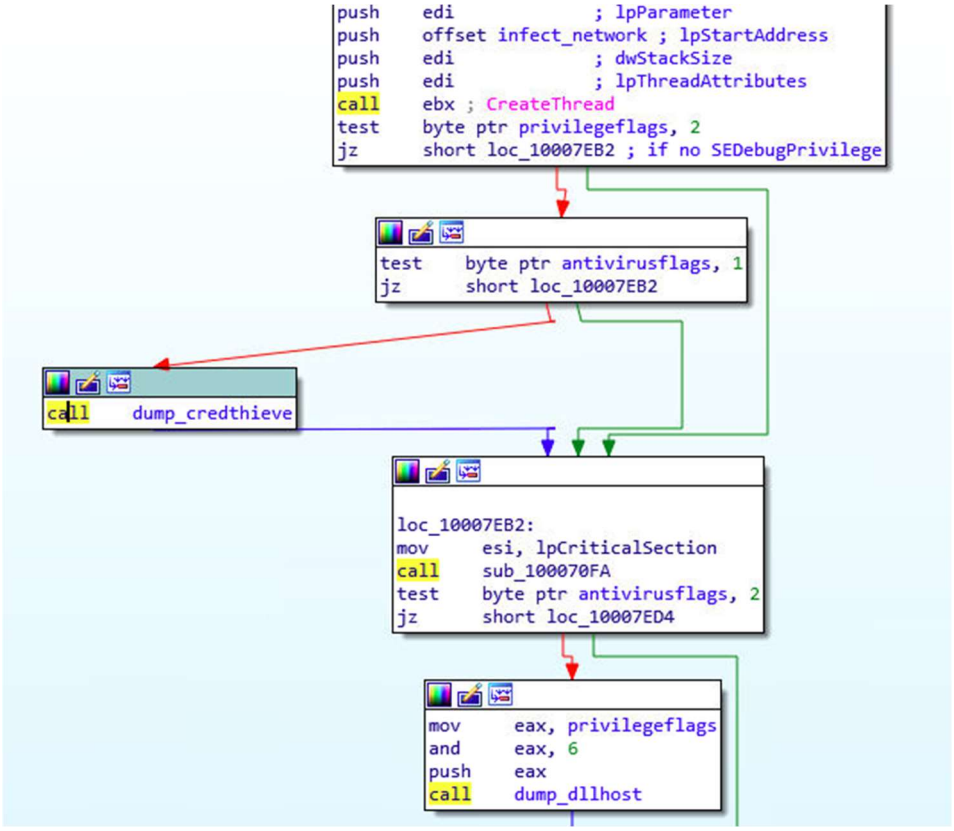


Network Infection

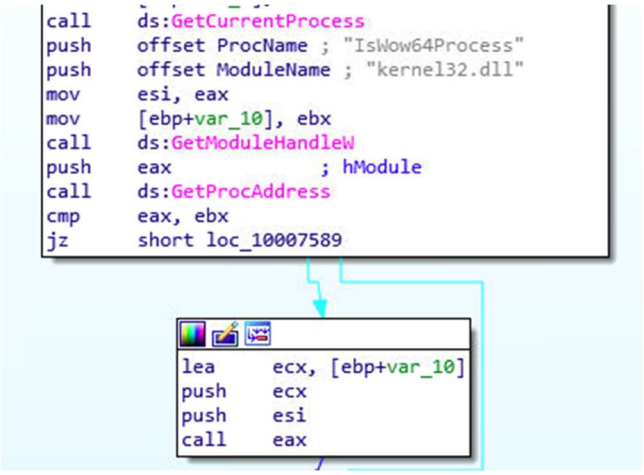
This section of the binary is a bit more involved, and difficult to analyse without a decompiler. Based on other reports, the sub-functions *find_IP_** shown below are identified to be functions that search for IP addresses to infect.

Extracting Resources

Afterwhich, the malware proceeds to dump some of its resources. The two functions involved and called in the main body of the malware is *dump_credthieve* and *dump_dllhost*.

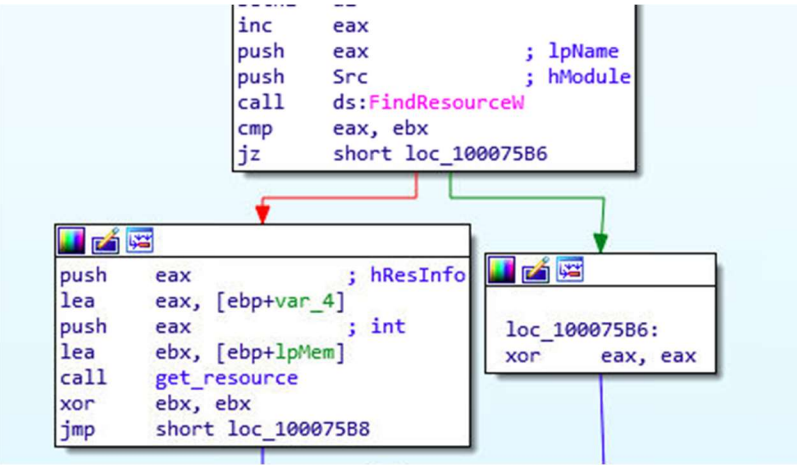


First looking into *dump_credthieve*, it checks whether the system is 32bit or 64bit by calling the function *IsWow64Process* from *kernel32.dll.*



Depending on the result, it will then extract either Resource 1 or Resource 2.

Any retrieval of the resources is done by finally calling *get_resource* at 100085D0.

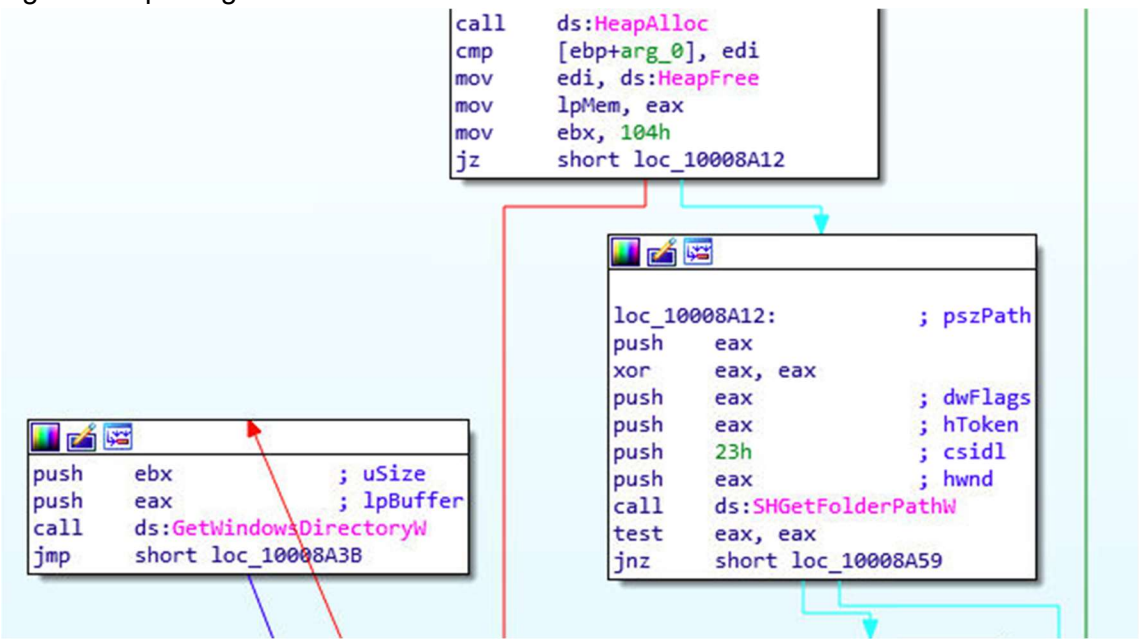


This is because the resources are packed and need to be decompressed before dumping. This function calls a sub-function *unpacker* at 1000A520, which (based on reports) unpacks using the library zlib 1.2.8.

```
000000001000A520                unpacker proc near
000000001000A520
000000001000A520                var_38= dword ptr -38h
000000001000A520                var_34= dword ptr -34h
000000001000A520                var_2C= dword ptr -2Ch
000000001000A520                var_28= dword ptr -28h
000000001000A520                var_24= dword ptr -24h
000000001000A520                var_18= dword ptr -18h
000000001000A520                var_14= dword ptr -14h
000000001000A520                arg_0= dword ptr  8
000000001000A520                arg_4= dword ptr  0Ch
000000001000A520                arg_8= dword ptr  10h
000000001000A520                arg_C= dword ptr  14h
000000001000A520
000000001000A520 55             push    ebp
000000001000A521 8B EC          mov     ebp, esp
000000001000A523 83 EC 38       sub     esp, 38h
000000001000A526 8B 45 10       mov     eax, [ebp+arg_8]
000000001000A529 83 65 E8 00    and     [ebp+var_18], 0
000000001000A52D 83 65 EC 00    and     [ebp+var_14], 0
000000001000A531 57             push    edi
000000001000A532 8B 7D 0C       mov     edi, [ebp+arg_4]
000000001000A535 89 45 C8       mov     [ebp+var_38], eax
000000001000A538 8B 45 14       mov     eax, [ebp+arg_C]
000000001000A53B 89 45 CC       mov     [ebp+var_34], eax
000000001000A53E 8B 45 08       mov     eax, [ebp+arg_0]
000000001000A541 89 45 D4       mov     [ebp+var_2C], eax
000000001000A544 8B 07          mov     eax, [edi]
000000001000A546 6A 38          push    38h
000000001000A548 89 45 D8       mov     [ebp+var_28], eax
000000001000A54B 8D 45 C8       lea     eax, [ebp+var_38]
000000001000A54E 68 C8 D2 00 10 push    offset a128     ; "1.2.8"
000000001000A553 50             push    eax
```

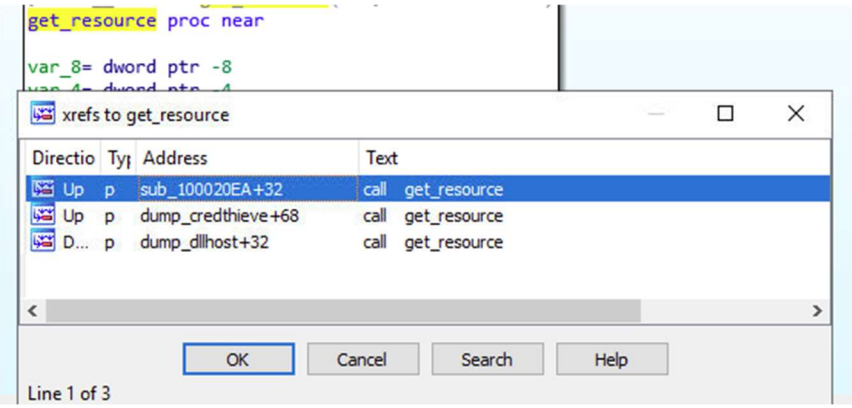
Shown above, the string “1.2.8”

Next in *dump_dllhost*, the directory the file *dllhost.dat* is dumped to is dependent once again on privileges. The privilege check is done outside of the function.



With sufficient privileges, it will use the *C:\Windows* directory. Otherwise, it calls *SHGetFolderPathW* with *CSIDL* of 0x23. This resolves to *CSIDL_COMMON_APPDATA*, which in turn resolves to *C:\ProgramData,* as observed in Dynamic Analysis: Disk. The resource extracted is Resource 3, and similar to *dump_credthieve* uses the decompressing functions *get_resource -> unpacker*.

There is one last resource, Resource 4. Knowing that all resources extracted uses the function *get_resource*, cross-references and tracebacks reveals that the function at 1000804F and 1000810B called in the main body eventually extracts this resource. While the function of the other three resources (1,2 for credential extraction similar to *Mimikatz*, 3 is basically *psexec*) is well-reported, the 4th resource's function needs further analysis.



Function at 100020EA calls get_resource, and extracts Resource 4.
Doing recursive xrefs reveals that functions 1000804F and 1000810B eventually call 100020EA.

File Encryption

It is well reported that the file encryption only encrypts files of certain extensions. The following snippet shows the relevant section of the function *enumerate_and_encrypt_files* that corroborates this.

In addition, the function recursively calls itself, probably as it recursively descends into directories. If the extension matches, it calls *encrypt_file*.