Name – Jay Shah
CSEC 202 Final Project

Malware Analysis of WannaCry Ransomware

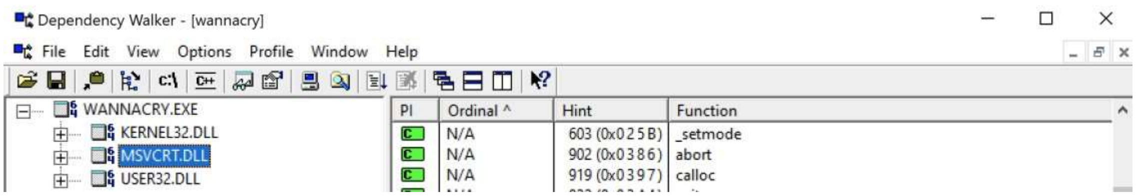
I. INTRODUCTION

The malware I chose to analyze is known as the WannaCry Ransomware. It hit the world by storm in 2017 targeting computers running Windows operating systems, then encrypting the users' files and demanding payment via Bitcoin. This malware was able to propagate through an exploit known as Eternal Blue. Eternal Blue was developed the NSA and stolen by a criminal group known as the Shadow Brokers in 2016. A patch was applied to the Eternal Blue exploit, but the Ransomware spread like wildfire throughout organizations that had yet to apply the patch. This attack infected over 200,000 computers across 150 countries, with damages estimated to have cost 4 billion US dollars. To analyze this complex malware, I first setup my lab with tools to complete Basic Static, Basic Dynamic, Advanced Static, and Advanced Dynamic analysis. I began with Basic Static and Dynamic analysis to get a solid understanding of what was occurring and then moved to advanced static analysis to dig deeper and search for more complex clues. To conclude, I used advanced dynamic analysis to identify code constructs, trace through the program's execution, and tie it all together.

II. ANALYSIS

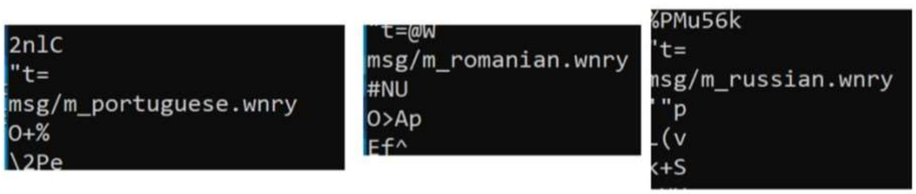
Starting with basic static analysis, I used Dependency Walker to first analyze the malwares imports. Here I found four DLLs imported: KERNEL32, MSVCRT, USER32, and ADVAPI32 **(Figure A)**. I was unfamiliar with MSVCRT, but after examining Windows documentation I found it to be the standard C library for Microsoft's Visual Studio. Much of the imported functions within this DLL referenced string manipulation, memory allocation, and a vast amount of input/output calls. I then investigated the USER32 library and found one imported function to here, *MessageBoxA*. After referencing documentation, I found this call to work with the GUI to display a box containing icons and buttons. I found the most insightful information within the ADVAPI32 linked library. Off the bat I saw StartServiceA, OpenSCManagerA, and OpenServiceA. All of which have to do with running services on the host, which let me know that the malware may be attempting to make itself persistent. I also saw Registry-based functions to close, query, set, and create registry values.

Figure A



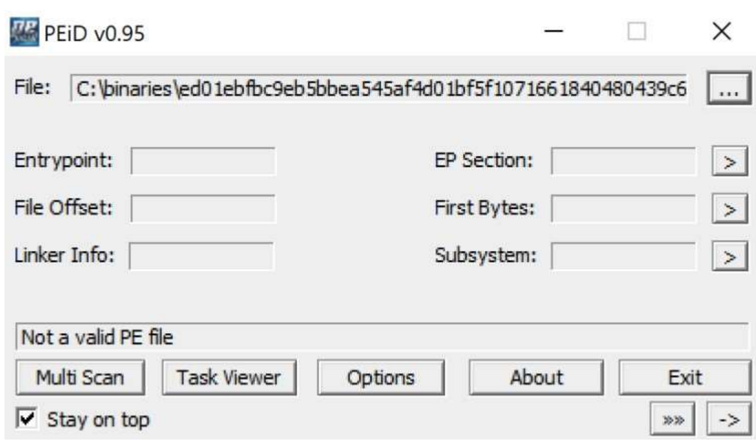
Following this, I ran the executable through another basic static tool, Strings. What I first noticed here was a list of many languages with the form "msg/m_LANGUAGE.wnry" separated by gibberish **(Figure B)**. Knowing this was a ransomware, I could assume that these languages were to be of use later and would most likely match the host machines language in use**.**

Figure B



We also saw the executables "taskdl.exe", "diskpart.exe", and "taskse.exe", another signature left by the malware writer that suggests persistence. To complete this section of analysis, I used PEiD to inspect if the program was packed but came up short here as the program did not recognize the executable as a valid PE file **(Figure C)**. I found this to be odd as the file would later load into IDA pro as a Win-32 PE file just fine.

Figure C



From here I moved onto basic dynamic analysis. First taking a picture of the current Registry values, using Reshot. I then ran the program through Process Explorer and was able to see new processes spawned: WannaDecryptor.exe, conhost.exe, and taskhsvc.exe. At this time, the malware infected my computer (**Figure D)**. From here, the malwares purpose was clear, to take the host machine hostage by encrypting its files and demanding payment via Bitcoin. I then took a second picture of the Registry and compared them to find what else had occurred **(Figure E)**. We saw the addition of Registry key values pertaining to DLL's that manage remote connections (possibly for a C&C server), open files (to encrypt the data) and manage remote access (to input the decryption key when the ransom is paid). The most important registry change was the key that added WanaCrypt0r on the user's desktop. This keeps the malware persistent so that even if the computer is shutdown the machine is still infected. I also noticed the appearance of two new executables on the desktop: taskse and taskdl. At this point of the analysis, I'm not sure what more they could be doing except for potentially aiding in the decryption or acting as another measure of securing a remote access connection for the malware writer.

Figure D

Figure E

```
HKU\S-1-5-21-2435760711-679044203-3364313472-1000\SOFTWARE\Classes\Local Settings\MrtCache\C:%5CProgram Files%5CWindowsApps%5CMicrosoft.Windows.Photos_
HKU\S-1-5-21-2435760711-679044203-3364313472-1000\SOFTWARE\WanaCrypt0r\wd: "C:\Users\Sam Benoist\Desktop"
HKU\S-1-5-21-2435760711-679044203-3364313472-1000_Classes\Local Settings\MrtCache\C:%5CProgram Files%5CWindowsApps%5CMicrosoft.Windows.Photos_2020.2011
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@C:\Windows\System32\wshext.dll,-4802: "VBScript Script File"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@C:\Windows\System32\acppage.dll,-6002: "Windows Batch File"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@%SystemRoot%\system32\windowsudk.shellcommon.dll,-100: "Udk User Service"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@%SystemRoot%\system32\sstpsvc.dll,-200: "Secure Socket Tunneling Protocol Service"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@%Systemroot%\system32\rasmans.dll,-200: "Remote Access Connection Manager"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@%systemroot%\system32\mprmsg.dll,-32011: "Remote Access IP ARP Driver"
HKU\S-1-5-18\Software\Classes\Local Settings\MuiCache\9\52C64B7E\@%SystemRoot%\System32\urlmon.dll,-4200: "Open File - Security Warning"
```

At this point we have a good understanding of what the malware attempts to do and can dig a little deeper, with advanced static analysis. We started by loading the executable into

IDA Pro where we see the start at the address 0x00401FE7 **(Figure F)**. The program then begins by getting its working directory and generating a random string **(Figure G)**.

Figure F

```
sub_401FE7 proc near

var_6F4= dword ptr -6F4h
var_6F4= byte ptr -6F4h
```

Figure G

```
lea     eax, [ebp+Filename] ; get working directory
push    208h                ; nSize
xor     ebx, ebx
push    eax                 ; lpFilename
push    ebx                 ; hModule
call    ds:GetModuleFileNameA
push    offset DisplayName
call    sub_401225          ; function to generate random string
pop     ecx
call    ds:__p___argc       ; check arg #
cmp     dword ptr [eax], 2
```

call to sub_401BF5 **(Figure H)**. Examining sub_401BF5 in more depth, **(Figure I)** we see the directories the malware is attempting to gain permissions to: “%WINDOWS%\ProgramData and “%WINDOWS%\Intel”.

Figure H

```
push    ebx             ; wchar_t *
call    sub_401B5F
test    eax, eax
pop     ecx
jz      short loc_40208E
```

Figure I

```
and     [ebp+var_4D4], 0
lea     eax, [ebp+Buffer]
push    eax             ; Format
lea     eax, [ebp+String]
push    offset aSProgramdata ; "%s\\ProgramData"
push    eax             ; String
call    edi ; swprintf
add     esp, 0Ch
lea     eax, [ebp+String]
push    eax             ; lpFileName
call    ds:GetFileAttributesW
cmp     eax, 0FFFFFFFFh
jz      short loc_401C40

push    [ebp+arg_0]     ; String
lea     eax, [ebp+WideCharStr]
push    eax             ; lpFileName
lea     eax, [ebp+String]
push    eax             ; lpPathName
call    sub_401AF6
add     esp, 0Ch
test    eax, eax
jz      short loc_401C40

loc_401C40:
lea     eax, [ebp+Buffer]
push    eax             ; Format
lea     eax, [ebp+String]
push    offset aSIntel  ; "%s\\Intel"
```

Following the codes execution, we next reach what may be a kill switch of the program. The malware then attempts to make itself replicate with the executable named “tasksche.exe” **(Figure J)**. The executable then moves to check if a specified Mutex exists. If it does, it will sleep and then retry after 1 minute **(Figure K).**

Figure J

```
xt:0040205F                 jz      short loc_40208E
xt:00402061                 mov     esi, offset FileName ; "tasksche.exe"
xt:00402066                 push    ebx             ; bFailIfExists
xt:00402067                 lea     eax, [ebp+Filename]
xt:0040206D                 push    esi             ; lpNewFileName
xt:0040206E                 push    eax             ; lpExistingFileName
xt:0040206F                 call    ds:CopyFileA
xt:00402075                 push    esi             ; lpFileName
xt:00402076                 call    ds:GetFileAttributesA
xt:0040207C                 cmp     eax, 0FFFFFFFFh
xt:0040207F                 jz      short loc_40208E
xt:00402081                 call    sub_401F5D
xt:00402086                 test    eax, eax
xt:00402088                 jnz     loc_402165
```

Figure K

```
xt:00401F26 loc_401F26:                             ; CODE XREF: sub_401EFF+48↓j
xt:00401F26                 lea     eax, [ebp+Dest]
xt:00401F29                 push    eax             ; lpName
xt:00401F2A                 push    1               ; bInheritHandle
xt:00401F2C                 push    100000h         ; dwDesiredAccess
xt:00401F31                 call    ds:OpenMutexA   ; checks if mutex exists
xt:00401F37                 test    eax, eax
xt:00401F39                 jnz     short loc_401F51 ; if Mutex exists, proceed to sleep
xt:00401F3B                 push    3E8h            ; dwMilliseconds
xt:00401F40                 call    ds:Sleep
xt:00401F46                 inc     esi             ; increment counter and compare incrementer to check again after 1 min for mutex existence
xt:00401F47                 cmp     esi, [ebp+arg_0]
xt:00401F4A                 jl      short loc_401F26
xt:00401F4C
```

If the malware proceeds, it continues to a code block where it attempts to create persistence by calling sub_4010FD (renamed persistence) where it attempts to add “WannaCrypt0r” as the software name to the registry **(Figure L)**.

Figure L

```
.text:004010FD                 push    ebp
.text:004010FE                 mov     ebp, esp
.text:00401100                 sub     esp, 2DCh
.text:00401106                 push    esi
.text:00401107                 push    edi
.text:00401108                 push    5
.text:0040110A                 mov     esi, offset aSoftware ; "Software\\"
.text:0040110F                 pop     ecx
.text:00401110                 lea     edi, [ebp+Dest]
.text:00401116                 rep movsd
.text:00401118                 push    2Dh
.text:0040111A                 xor     eax, eax
.text:0040111C                 and     [ebp+Buffer], al
.text:00401122                 pop     ecx
.text:00401123                 lea     edi, [ebp+var_C0]
.text:00401129                 and     [ebp+phkResult], 0
.text:0040112D                 rep stosd
.text:0040112F                 mov     ecx, 81h
.text:00401134                 lea     edi, [ebp+var_2D8]
.text:0040113A                 rep stosd
.text:0040113C                 stosw
.text:0040113E                 stosb
.text:0040113F                 lea     eax, [ebp+Dest]
.text:00401145                 push    offset Source   ; "WanaCrypt0r"
.text:0040114A                 push    eax             ; Dest
.text:0040114B                 call    ds:wcscat
.text:00401151                 and     [ebp+var_8], 0
.text:00401155                 pop     ecx
.text:00401156                 pop     ecx
.text:00401157                 mov     edi, offset ValueName ; "wd"
```

In the same code block that the persistence function is called **(Figure M)** the malware is seen attempting to move files into its working directory, and then trying to change the attributes of the files to "hidden and grant full access to the files. The lines that do this are "attrib +h." and "icalcs
./grant Everyone:F /T /C /Q".

Figure M

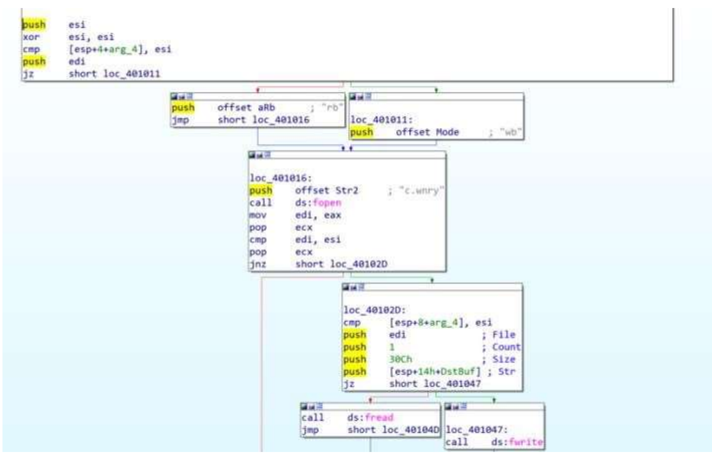
```
loc_4020B4:
lea     eax, [ebp+Filename]
push    eax             ; lpPathName
call    ds:SetCurrentDirectoryA
push    1
call    persistence
mov     [esp+6F4h+var_6F4], offset aWncry2ol7 ; "WNcry@2ol7"
push    ebx             ; hModule
call    sub_401DAB
call    sub_401E9E
push    ebx             ; lpExitCode
push    ebx             ; dwMilliseconds
push    offset CommandLine ; "attrib +h ."
call    cmd_exe
push    ebx             ; lpExitCode
push    ebx             ; dwMilliseconds
push    offset aIcaclsGrantEve ; "icacls . /grant Everyone:F /T /C /Q"
call    cmd_exe
add     esp, 20h
call    get_filehandles
test    eax, eax
jz      short loc_402165
```

Moving a bit backwards, before the file access is manipulated, sub_401E9E **(Figure N)** is a function that contains three hardcoded Bitcoin addresses that then calls the function I renamed to edit_config **(Figure O)**. This function appears to do read/write operations from the file c.wnry, which I inferred to be some sort of configuration file for when the Ransomware part of the malware is executed.

Figure N

```
.text:00401E9E                 push    ebp
.text:00401E9F                 mov     ebp, esp
.text:00401EA1                 sub     esp, 318h
.text:00401EA7                 lea     eax, [ebp+DstBuf]
.text:00401EAD                 push    1               ; int
.text:00401EAF                 push    eax             ; DstBuf
.text:00401EB0                 mov     [ebp+Source], offset a13am4vw2dhxygx ; "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
.text:00401EB7                 mov     [ebp+var_8], offset a12t9ydpgwuez9n ; "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
.text:00401EBE                 mov     [ebp+var_4], offset a115p7ummngoj1p ; "115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn"
.text:00401EC5                 call    edit_config
.text:00401ECA                 pop     ecx
.text:00401ECB                 test    eax, eax
.text:00401ECD                 pop     ecx
.text:00401ECE                 jz      short locret_401EFD
.text:00401ED0                 call    ds:rand
.text:00401ED6                 push    3
.text:00401ED8                 cdq
.text:00401ED9                 pop     ecx
.text:00401EDA                 idiv    ecx
.text:00401EDC                 lea     eax, [ebp+Dest]
.text:00401EE2                 push    [ebp+edx*4+Source] ; Source
.text:00401EE6                 push    eax             ; Dest
.text:00401EE7                 call    strcpy
.text:00401EEC                 lea     eax, [ebp+DstBuf]
.text:00401EF2                 push    0               ; int
.text:00401EF4                 push    eax             ; DstBuf
.text:00401EF5                 call    edit_config
.text:00401EFA                 add     esp, 10h
```

Figure O



From here, I used a tool that had not yet been discussed in class. The file c.wnry appeared in the malware as well as the file t.wnry, curious to know what those were I downloaded the tool Universal Extractor, which allows you to extract files from an executables archive. It extracted the files b.wnry, c.wnry, r.wnry, s.wnry, and u.wnry. I bounced back to basic static analysis and extracted the strings from these files. From r.wnry **(Figure P)**, we found what seems to be messages of what occurs when the Ransomware is run, and instructions for where to send Bitcoin in order for the Ransom to be paid. From c.wnry (**Figure Q)** we see 5 .onion links (potentially C&C server URL's) as well as a link to some tor zip file. From u.wnry (**Figure R)** we saw a lot of gibberish but also information pertaining to the encryption and decryption of the malware.

Figure P

```
C:\binaries>strings r.wnry

Strings v2.51
Copyright (C) 1999-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

Q:  What's wrong with my files?
A:  Ooops, your important files are encrypted. It means you will not be able to access them anymore until they are decry
pted.
    If you follow our instructions, we guarantee that you can decrypt all your files quickly and safely!
    Let's start decrypting!
Q:  What do I do?
A:  First, you need to pay service fees for the decryption.
    Please send %s to this bitcoin address: %s
    Next, please find an application file named "%s". It is the decrypt software.
    Run and follow the instructions! (You may need to disable your antivirus for a while.)

Q:  How can I trust?
A:  Don't worry about decryption.
    We will decrypt your files surely because nobody will trust us if we cheat users.

*   If you need our assistance, send a message by clicking <Contact Us> on the decryptor window.
```

Figure Q

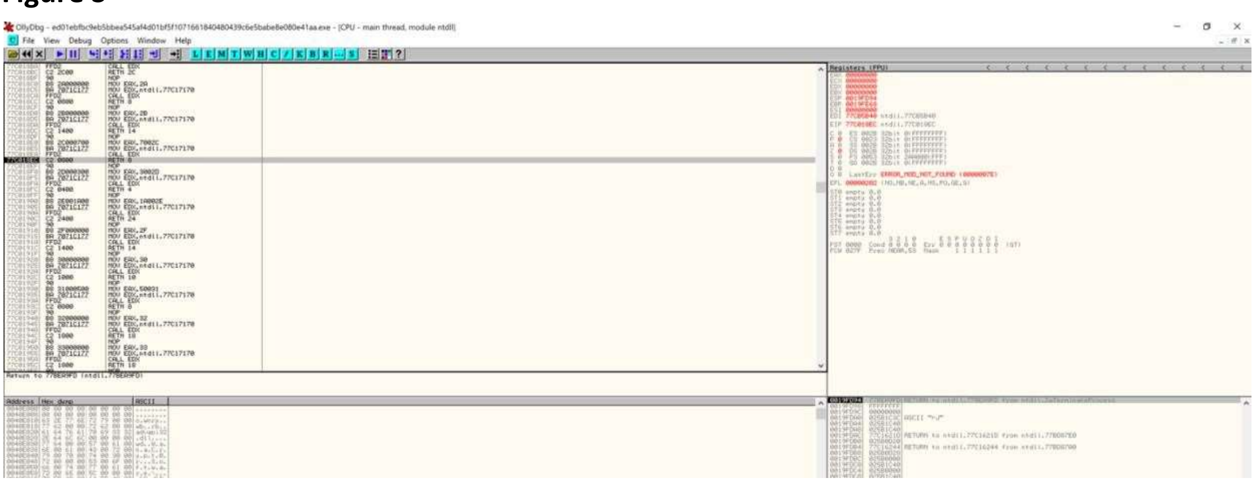
```
C:\binaries>strings c.wnry

Strings v2.51
Copyright (C) 1999-2013 Mark Russinovich
Sysinternals - www.sysinternals.com

gx7ekbenv2riucmf.onion;57g7spgrzlojinas.onion;xxlvbrloxvriy2c5.onion;76jdd2ir2embyv47.onion;cwwnhwhlz52maqm7.onion;
https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip
```

Figure R

```
Microsoft Enhanced RSA and AES Cryptographic Provider
TESTDATA
CryptGenKey
CryptDecrypt
CryptEncrypt
CryptDestroyKey
CryptImportKey
CryptAcquireContextA
Wana Decrypt0r 2.0
```

At this point, advanced static analysis was concluded, and it was time to move into advanced dynamic analysis. To begin, I first ran the program to see where it stops. This occurred at the address 0x77C018EC after the decryptor is installed and the malware has been made persistent **(Figure S)**.

Figure S



I was able to once again trace through the programs execution and saw that once directory permissions were granted and files ending in ".wnry" were modified to edit the configuration the program had ended. Unfortunately, I could not find anything not yet identified.

III. CHALLENGES

I faced many challenges throughout the entirety of the process. The hardest part was not knowing where to look. I often found myself stuck, unsure if I was missing some crucial element. After taking a step back and remembering what was taught in class the clues started to appear. Another problem I ran into was seeing the files ending in ".wnry" and having no knowledge of what they were used for. I looked through all the features in IDA and OllyBDG and couldn't figure it out. I then thought of downloading an external tool and tried many without success. It wasn't until I found Universal Extractor that I was able to scrape the data within the files archived in the executable.

IV. SUMMARY

Modifying the registry, gaining persistence, communicating with a onion C&C server, and encryption of the hosts files were actions identified through all types of analysis used. Through this, I was able to gain a deep understanding of this malware purpose, action, and outcome. The WannaCry ransomware exploits the victim where they are most vulnerable and is executed well. Only charging $300 per machine is a tactic that allowed the malware writers to get the most out of the victims, as it is safe to assume most users value the totality of the files on their computer at well over that price. This speaks to the malwares ability to gain national news coverage with the removal costing an estimated 4 billion USD. While we can acknowledge a strong design by the criminals who developed this, we can avoid future pain by outlining some design flaws. First off, when checking for a mutex, the program temporarily goes to sleep for a set number of seconds. If this value were to be manipulated, then we could cause the malware to sleep forever, never fully executing to which we could then simply kill the process. Another point of interception is when the malware attempts communication with an external C&C server. In theory, we could use a tool to fake the DNS responses and send back messages indicating that the ransom has been paid. These ideas and the findings of my analyzation could be used in the future to design efficient mitigation mechanisms for Ransomware that exhibits similar behavior.