Peter Mastropaolo

Reverse Engineering WannaCry

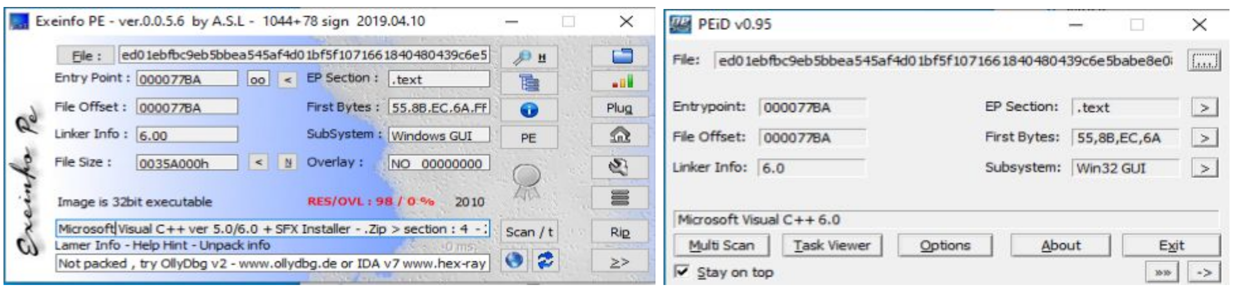
Introduction

The piece of malware that I have chosen to reverse engineer is WannaCry. WannaCry was one of the most prominent pieces of ransomware that the world has ever seen, leading to a keen interest in how this malware was able to inflict the damage it did and spread as it did. I downloaded the WannaCry sample from theZoo on github and placed it on an isolated windows 10 VM on host-only to prevent the malware from breaking out of the VM. In terms of how I will approach reverse engineering the malware, I will begin with basic static analysis in order to first determine if it is packed(using exeinfope and peid) and needs to be unpacked in order to further continue in both basic and advanced static analysis. Then, if it is packed, I will unpack(if they don't develop their own packing system) the malware in order to continue with my static analysis. I will use PEView to determine what imports are being used in order to give me direction for what I should be looking for in further analysis. I will also check strings for any suspicious strings that may also give me a hint as to what is happening(possible connection points and potential process names to look out for). I will also use resource hacker to check for hidden files that can be further analysed. Next I will move onto basic dynamic to start discovering what the malware observably does from the information gathered from basic static analysis, and begin to narrow down what the malware's exact purpose is. Then, in order to begin reverse-engineering exactly how the malware works, I will use advanced static analysis with IDA in order to determine exactly what the malware is doing. Advanced dynamic analysis will be used in conjunction with advanced static in order to fill in the gaps that IDA is unable to determine what is happening.

<u>Analysis</u>

Basic Static

When performing basic static analysis, I first used exeinfope and peid to determine whether or not the executable is packed before I went on to further analysis.



After using both of these tools, I determined that the .exe was not packed and I could continue with basic static analysis. My next step was to determine what imports were being used in this executable so I used the tool PEView to find the imports that I considered relevant to malware usage.

```
0064  CreateServiceA
01AF  OpenServiceA
0249  StartServiceA
003E  CloseServiceHandle
00A0  CryptReleaseContext
01D3  RegCreateKeyW
0204  RegSetValueExA
01F7  RegQueryValueExA
01CB  RegCloseKey
01AD  OpenSCManagerA
ADVAPI32.dll
0161  GetFileAttributesW
0164  GetFileSizeEx
```

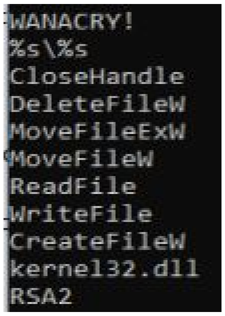
The imports that I noticed as important were CreateServiceA, OpenServiceA, StartServiceA, CloseServiceHandle, RegCreateKeyW, RegSetValueExA, RegQueryValueExA, RegCloseKey, OpenSCManagerA. This may mean that the malware is creating a service that will run on the system as well as creating a registry key and finding a registry value and changing it.

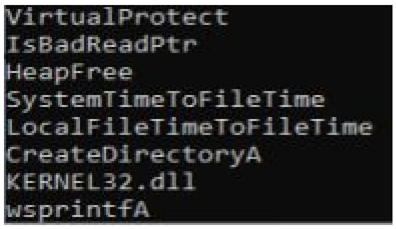
The malware is also involved with file manipulation based on the imports listed below.

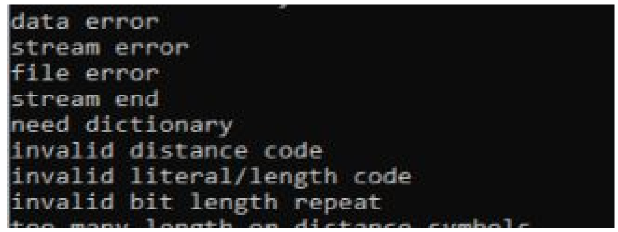
```
0161  GetFileAttributesW
0164  GetFileSizeEx
0053  CreateFileA
0223  InitializeCriticalSection
0081  DeleteCriticalSection
02B5  ReadFile
0163  GetFileSize
03A4  WriteFile
0251  LeaveCriticalSection
0098  EnterCriticalSection
031A  SetFileAttributesW
030B  SetCurrentDirectoryW
004E  CreateDirectoryW
01D6  GetTempPathW
01F4  GetWindowsDirectoryW
015E  GetFileAttributesA
0355  SizeofResource
0265  LockResource
0257  LoadResource
0275  MultiByteToWideChar
0356  Sleep
0284  OpenMutexA
0169  GetFullPathNameA
0043  CopyFileA
017D  GetModuleFileNameA
```

The malware is moving around directories and looking at the files in them as inferred by SetCurrentDirectoryW, GetWindowsDirectoryW, GetFileAttributesA, ReadFile, and GetFileSizeEx. The malware may also be creating copies of these files and modifying them as suggested by CopyFileA, CreateDirectoryW, SetFileAttributesW, WriteFile, InitializeCriticalSection, DeleteCriticalSection, LeaveCriticalSection, EnterCriticalSection, GetFullPathNameA, GetModuleFileNameA.

When observing the strings in the executable, most of them are random strings that are not relevant, with the exception of the imports and a couple of other strings that were near them.

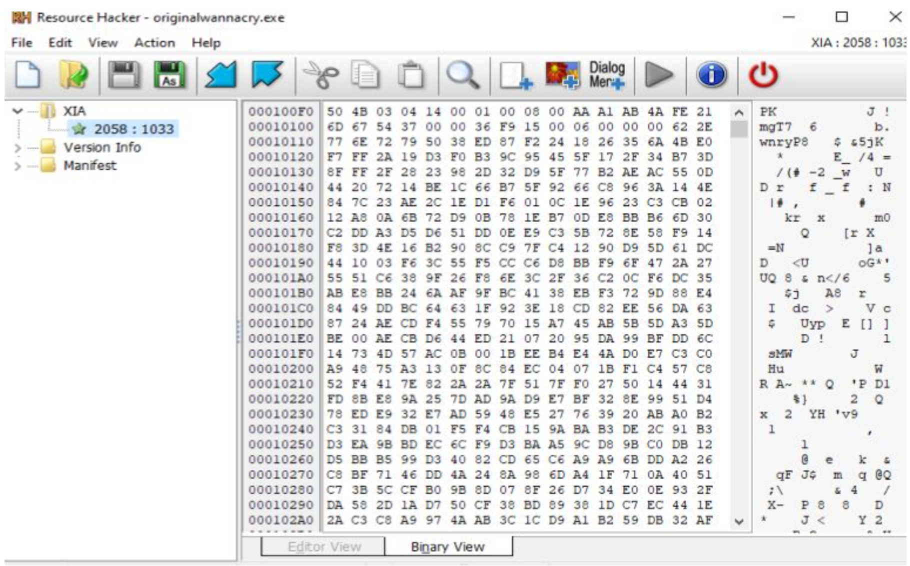

WANACRY!
%s\%s
CloseHandle
DeleteFileW
MoveFileExW
MoveFileW
ReadFile
WriteFile
CreateFileW
kernel32.dll
RSA2


VirtualProtect
IsBadReadPtr
HeapFree
SystemTimeToFileTime
LocalFileTimeToFileTime
CreateDirectoryA
KERNEL32.dll
wsprintfA

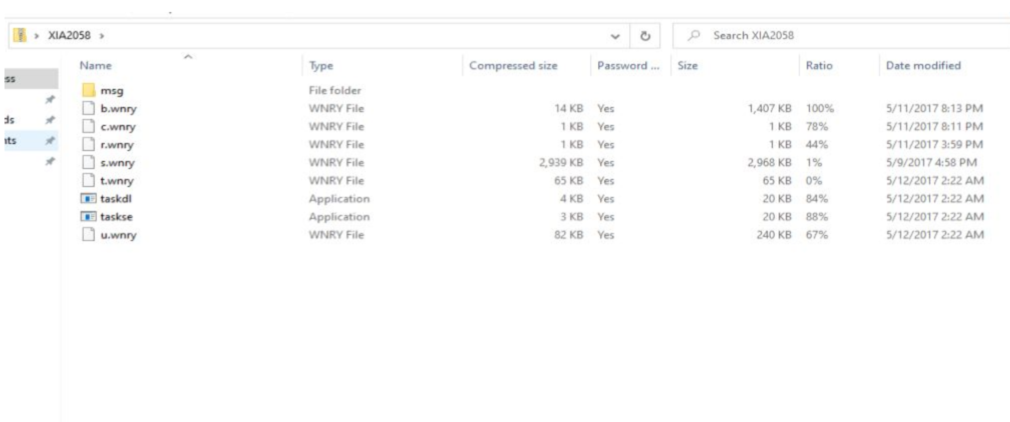

data error
stream error
file error
stream end
need dictionary
invalid distance code
invalid literal/length code
invalid bit length repeat

The first red flag that I saw in strings, besides the imports, was the string “WANACRY!” which insinuates that the executable is indeed WannaCry and is ransomware. What really caught my eye besides this obvious string was CloseHandle,DeleteFileW, MoveFileExW, MoveFileW, ReadFile, WriteFile, CreateFileW, and RSA2. These are all used in ransomware as files are being copied, encrypted and moved as suggested by these strings. The DeleteFileW is also a function of ransomware as it will delete the encrypted files(RSA2) after a certain time limit which could be determined from SystemTimeToFileTime and LocalFileTimeToFileTime being used to determine when to delete the files. A network connection is also a possibility based on the string CloseHandle, and some of the errors found in strings such as data error, stream error, file error, and stream end suggesting that the data is being transmitted to an outside source.

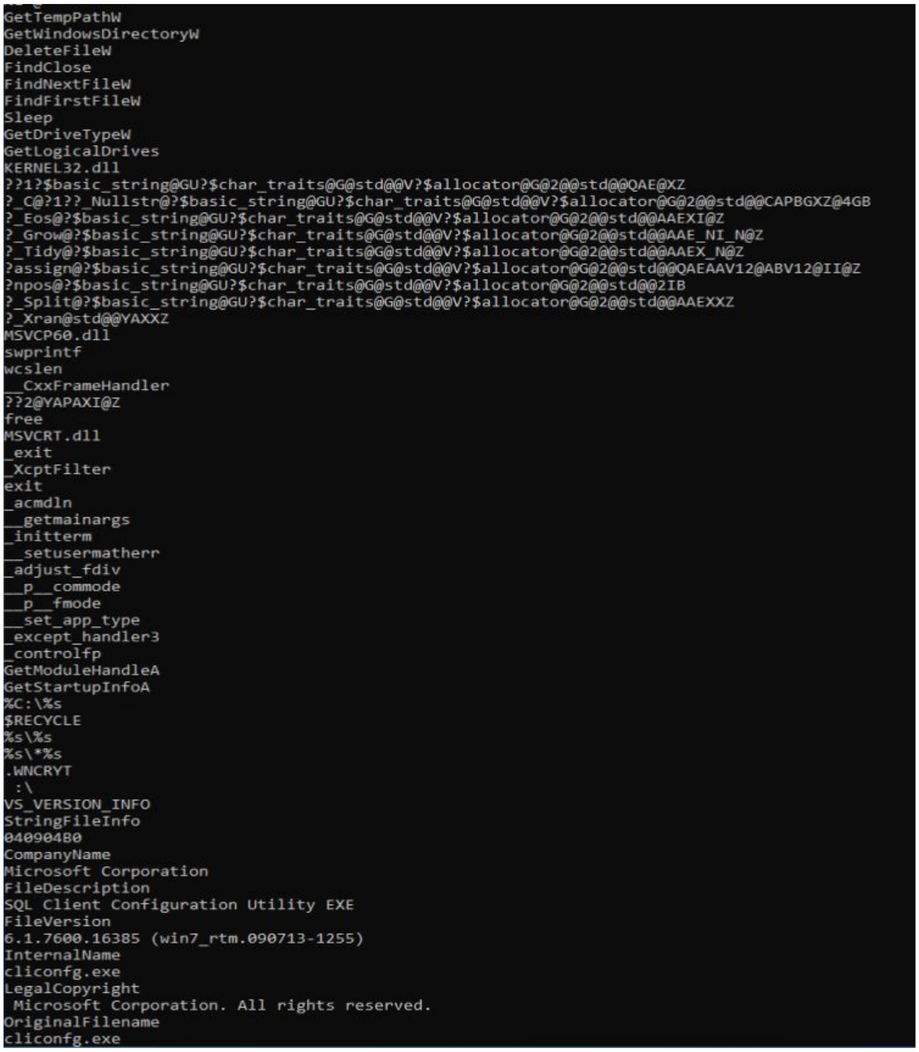
After performing advanced static analysis, I noticed a file that was being extracted from the executable called XIA. I opened the executable in resource hacker and noticed that XIA was there so I saved it as a bin file.



Then I changed the file extension of XIA.bin to XIA.zip and it contained files inside, but the zip file was encrypted. I found the password in IDA and was able to access it.



Taskdl is most likely performing file manipulation. It has strings such as GetWindowsDirectory, GetTempPathW, DeleteFileW, FindNextFileW, FindFirstFileW, GetLogicalDrives, and GetDriveTypeW.



```
GetTempPathW
GetWindowsDirectoryW
DeleteFileW
FindClose
FindNextFileW
FindFirstFileW
Sleep
GetDriveTypeW
GetLogicalDrives
KERNEL32.dll
??1?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@QAE@XZ
?_C@?1??_Nullstr@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@CAPBGXZ@4GB
?_Eos@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@AAEXI@Z
?_Grow@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@AAE_NI_N@Z
?_Tidy@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@AAEX_N@Z
?assign@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@QAEAAV12@ABV12@II@Z
?npos@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@2IB
?_Split@?$basic_string@GU?$char_traits@G@std@@V?$allocator@G@2@@std@@AAEXXZ
?_Xran@std@@YAXXZ
MSVCP60.dll
swprintf
wcslen
__CxxFrameHandler
??2@YAPAXI@Z
free
MSVCRT.dll
_exit
_XcptFilter
exit
_acmdln
__getmainargs
_initterm
__setusermatherr
_adjust_fdiv
__p__commode
__p__fmode
__set_app_type
_except_handler3
_controlfp
GetModuleHandleA
GetStartupInfoA
%C:\%s
$RECYCLE
%s\%s
%s\*%s
.WNCRYT
 :\
VS_VERSION_INFO
StringFileInfo
040904B0
CompanyName
Microsoft Corporation
FileDescription
SQL Client Configuration Utility EXE
FileVersion
6.1.7600.16385 (win7_rtm.090713-1255)
InternalName
cliconfg.exe
LegalCopyright
 Microsoft Corporation. All rights reserved.
OriginalFilename
cliconfg.exe
```

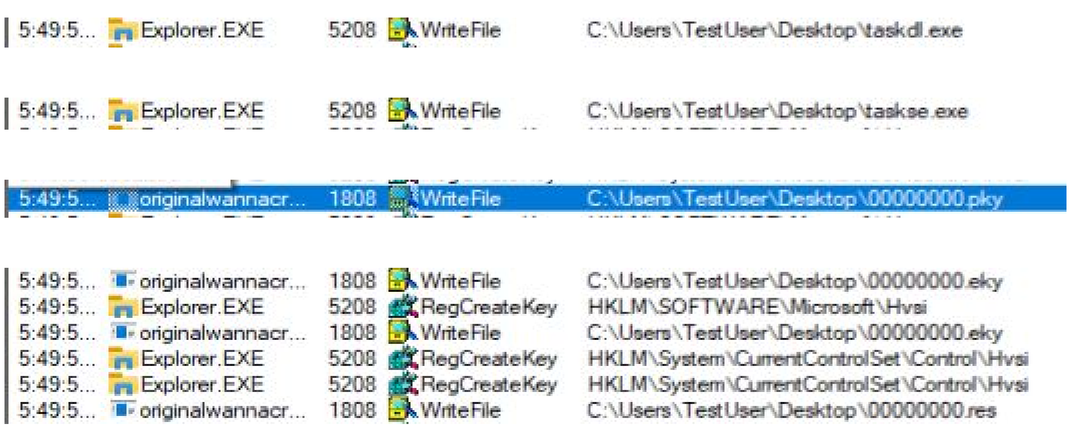
Taskse is most likely performing process handling as suggested by the strings WaitForSingleObject, GetProcAddress, LoadLibraryA, GetModuleHandleA, CloseHandle, GetStartupInfo, CreateProcessAsUserA, and GetCurrentProcess.

```
WaitForSingleObject
GetProcAddress
LoadLibraryA
GetModuleHandleA
Sleep
KERNEL32.dll
_except_handler3
_local_unwind2
__p___argv
__p___argc
MSVCRT.dll
_exit
_XcptFilter
exit
_acmdln
__getmainargs
_initterm
__setusermatherr
_adjust_fdiv
__p__commode
__p__fmode
__set_app_type
_controlfp
GetStartupInfoA
winsta0\default
SeTcbPrivilege
WTSQueryUserToken
wtsapi32.dll
DestroyEnvironmentBlock
CreateEnvironmentBlock
userenv.dll
CloseHandle
GetCurrentProcess
WTSGetActiveConsoleSessionId
kernel32.dll
CreateProcessAsUserA
DuplicateTokenEx
AdjustTokenPrivileges
LookupPrivilegeValueA
OpenProcessToken
advapi32.dll
WTSFreeMemory
WTSEnumerateSessionsA
Wtsapi32.dll
VS_VERSION_INFO
StringFileInfo
040904B0
CompanyName
Microsoft Corporation
FileDescription
waitfor - wait/send a signal over a network
FileVersion
6.1.7600.16385 (win7_rtm.090713-1255)
InternalName
waitfor.exe
LegalCopyright
 Microsoft Corporation. All rights reserved.
OriginalFilename
waitfor.exe
ProductName
Microsoft
```

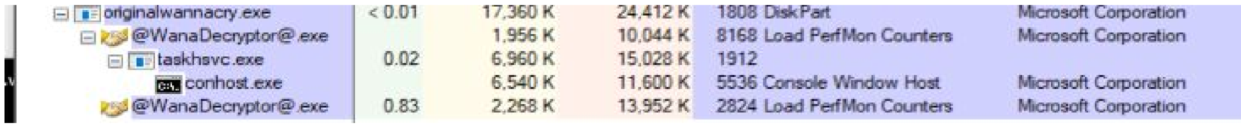
Basic Dynamic

Upon conducting my first round of basic dynamic analysis, I used ApateDNS to see if any network activity was occurring as well as netcat, process monitor, and regshot. ApateDNS and netcat did not show any network activity on the machine from the malware. Upon examining the regshot comparison from before and after the malware ran, I first noticed a registry value was created in Software\Microsoft\Windows\CurrentVersion\Run. Its purpose was to run the executable tasksche.exe on startup. The malware also creates @WanaDecryptor@.exe at the location of the original executable file, in this case the desktop.

When looking at procmon, another file that was created was in AppData\Local\Temp\ called hibsys.WNCRYT. Two other executables were created in the folder where WannaCry was executed named taskdl and taskse. Besides this, multiple files were created named 00000000 but with different extensions. A registry key was created called WanaCrypt0r as well.

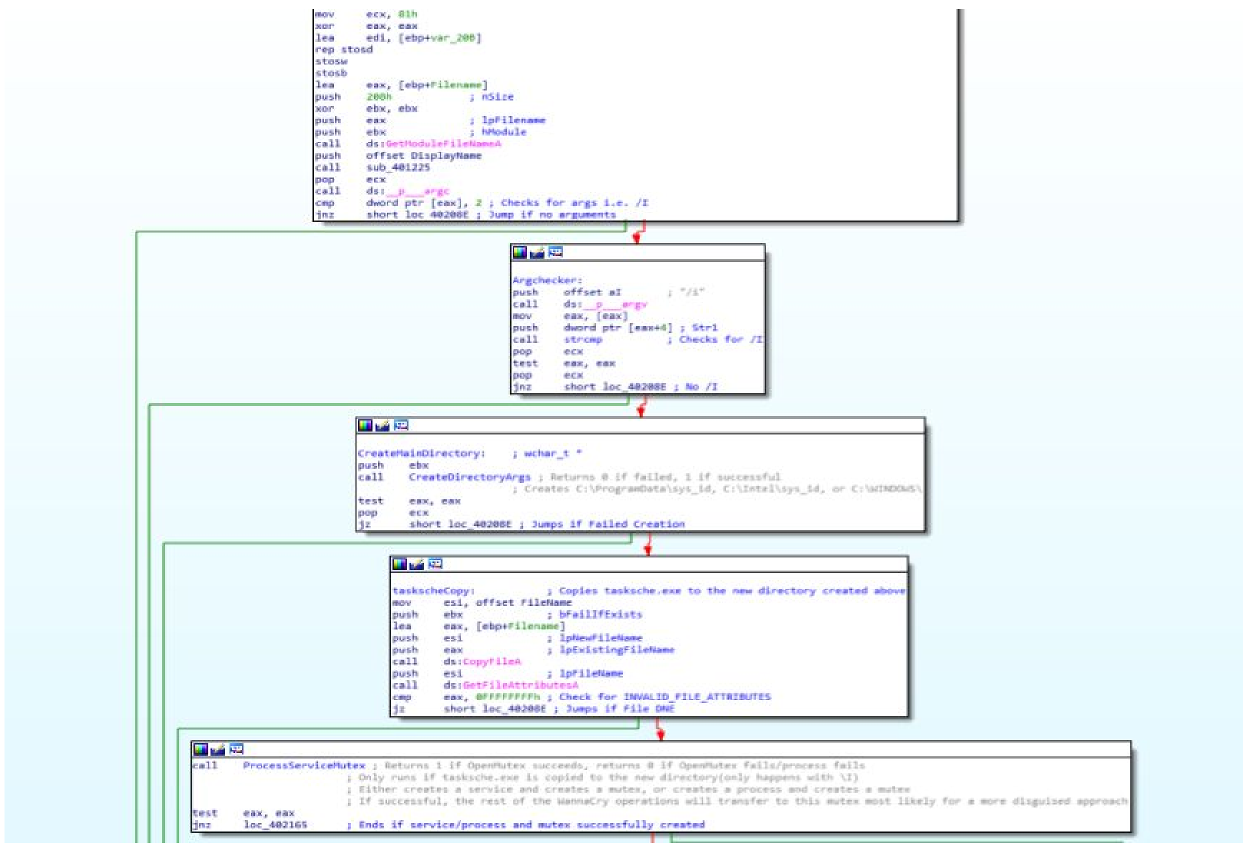


In procexp taskhsvc.exe was running as a child process under @WanaDecryptor@.exe. Upon further examination of taskhsvc.exe, the strings for this executable seem to be related to internet connections using a tor connection, showing that an internet connection is in fact occurring despite ApateDNS and netcat failing to catch any traffic.



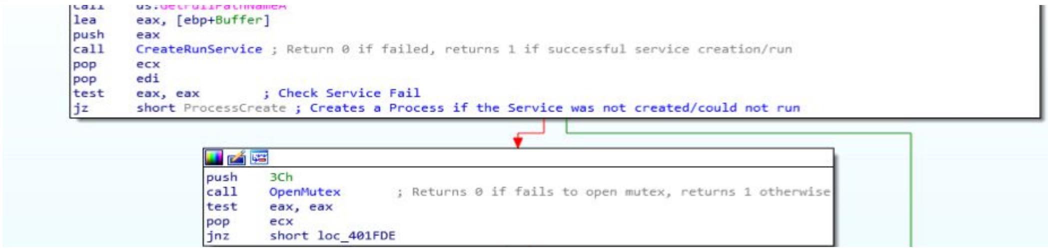
Advanced Static

I performed advanced static analysis using IDA Free 7. First I had to find the main function and rename it to WinMain. From there I started to dissect the local functions and calls to sub functions in order to determine the functionality of the executable.

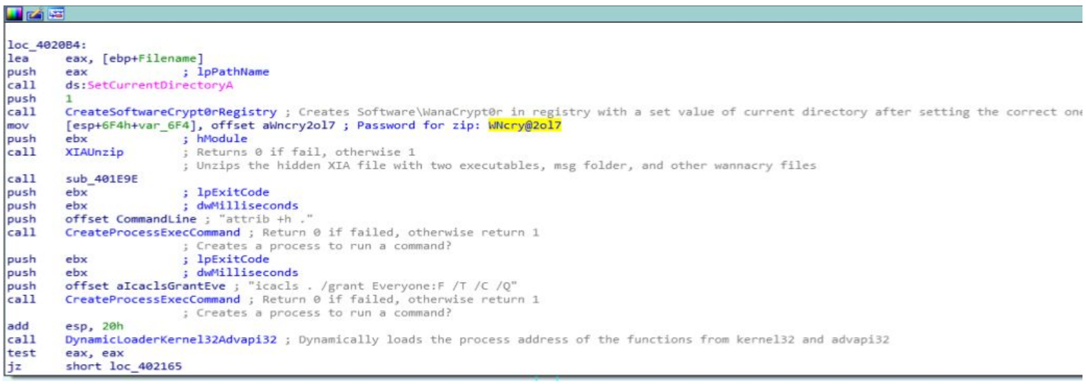


When I located the main function, I noticed that it checks for a command line argument of /i. This option seems to install a copy of tasksche.exe to a directory it creates at either

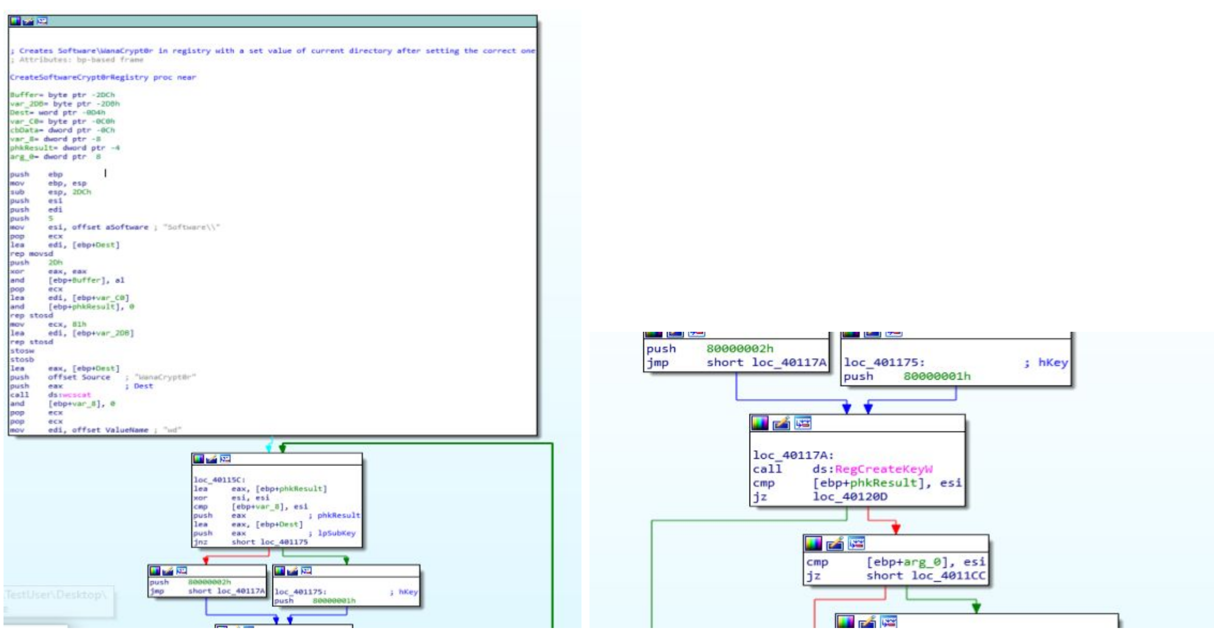
C:\ProgramData\Random16Characters\, C:\Intel\Random16Characters\ or C:\Windows\Random16Characters\. It then tries to open a mutex of the same file to make sure that the malware is not already running this new executable. If it successfully opens the mutex, then WannaCry will close since another instance has already infected this system. This is definitely a form of persistence being installed.

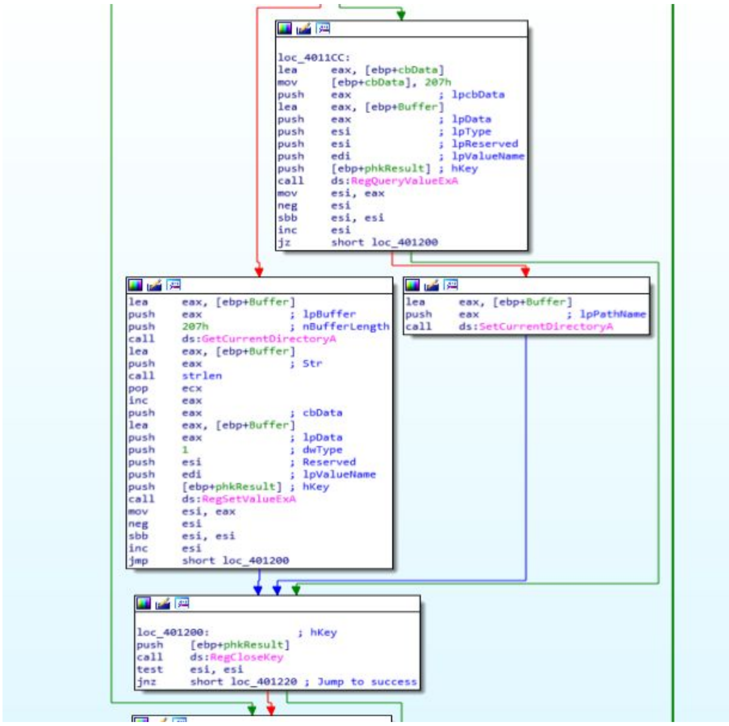


If no option is selected or the machine was not previously infected, the malware then continues to the next function, which creates a registry key, runs two commands, and dynamically loads certain functions.

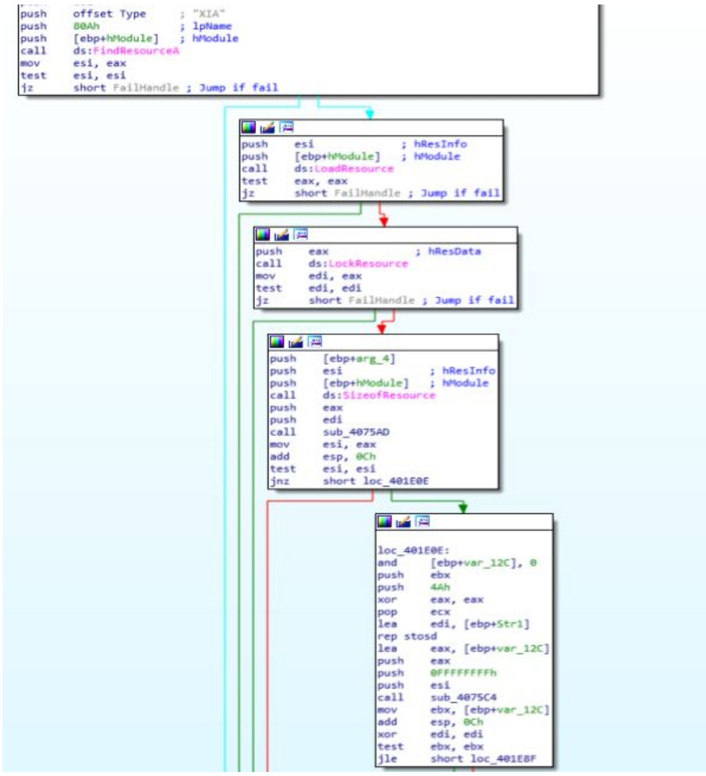


The malware creates a registry key SOFTWARE\WanaCrypt0r. It uses RegQueryValueExA, RegSetValueExA, RegCreateKeyW, and RegCloseKey to create the key above.

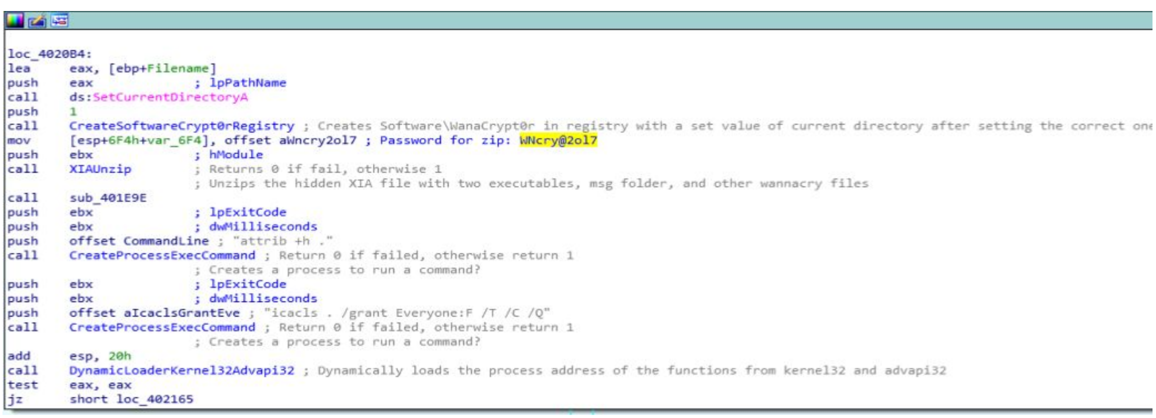




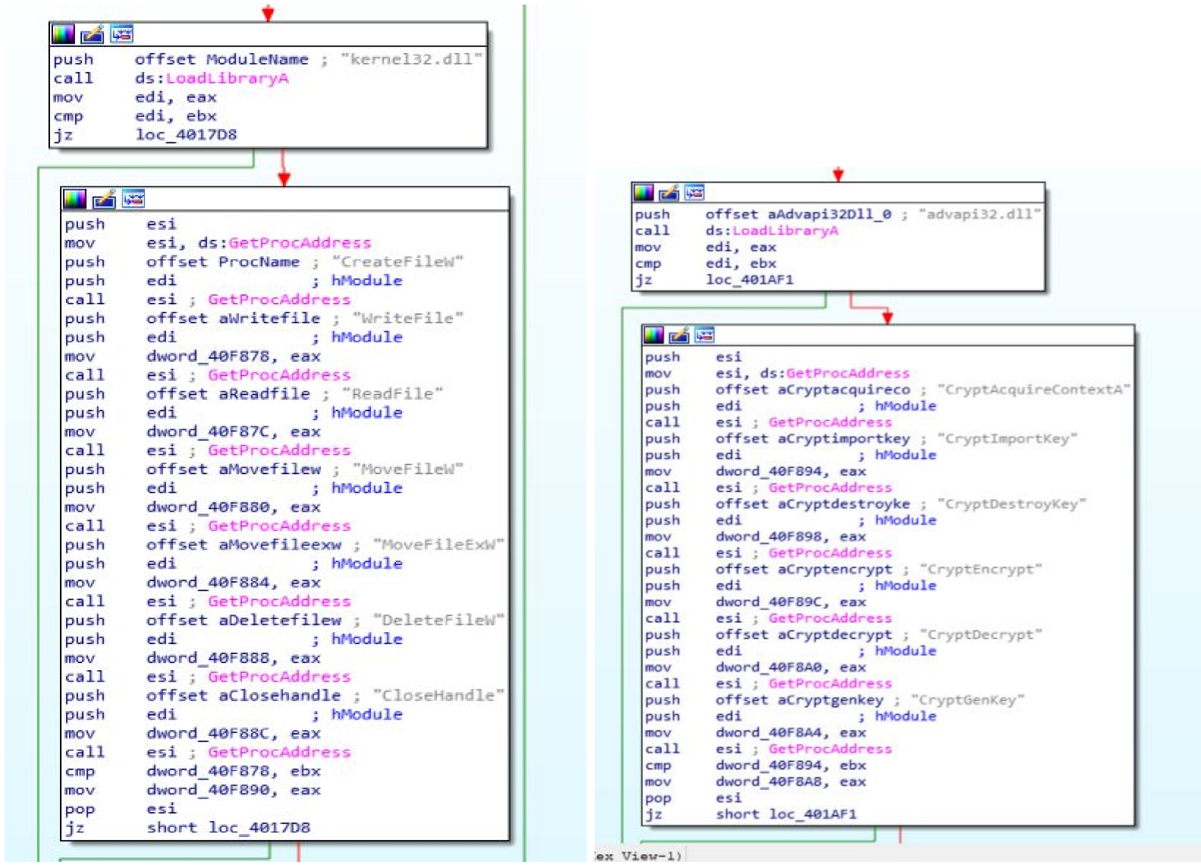
It then unzips a hidden file (with the password “WNcry@2ol7”) XIA that contains .wnry files and two executables, taskdl.exe and taskse.exe. Taskdl is mostly related to file manipulation while taskse is related to process handling.



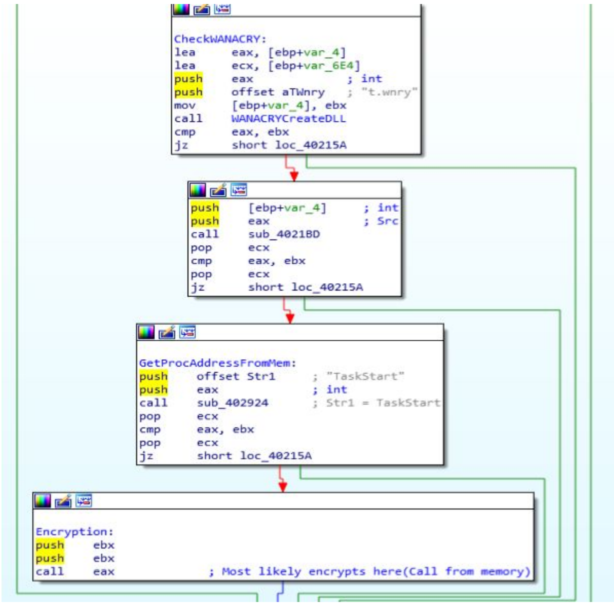
After unzipping these files, two commands are executed: “attrib +h” and “icacls ./grant Everyone:F /T /C /Q”.



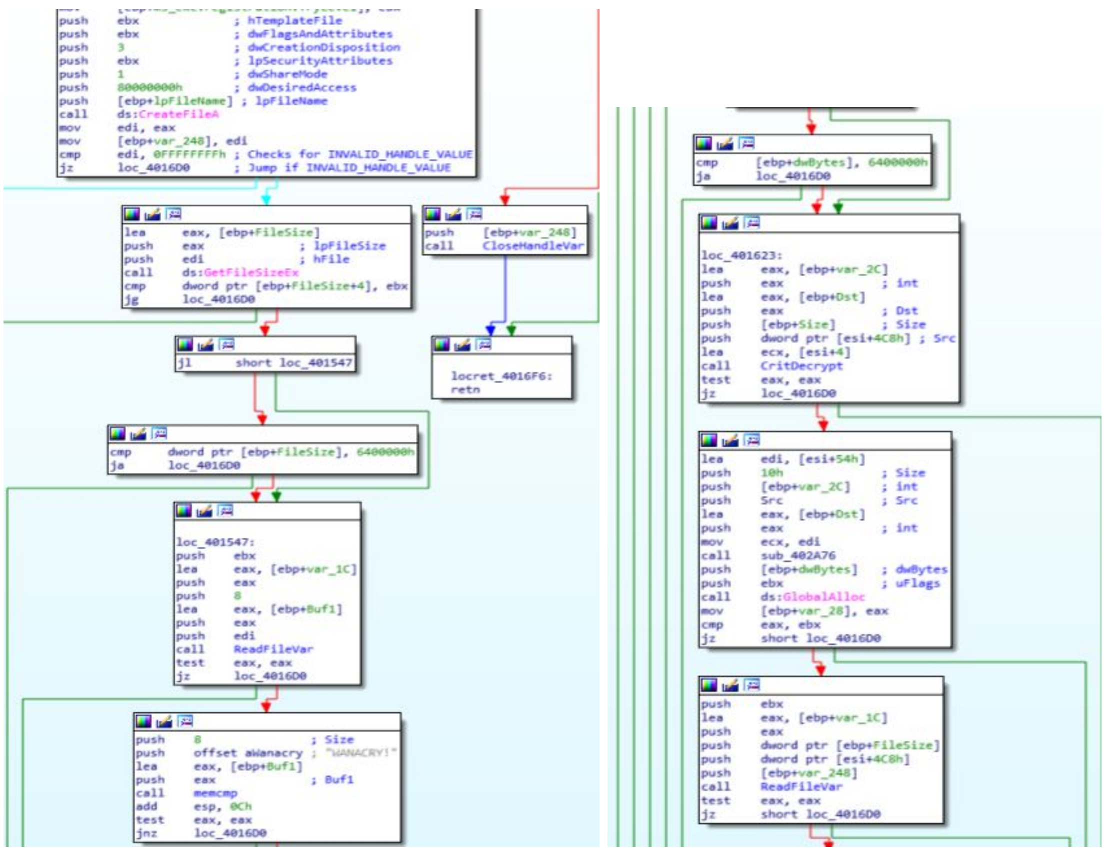
The malware then dynamically loads the process addresses of functions from kernel32.dll and advapi32.dll most likely as a form of obfuscation as they were stored in dword variables.



One of the final actions it does is get the process address from memory to load a hidden file related to encryption and then the hidden file most likely encrypts the computers files after it is called from memory. NOTE: The comments on this last picture below are from after advanced dynamic analysis with x32 Debug, leading to more discoveries in static analysis.

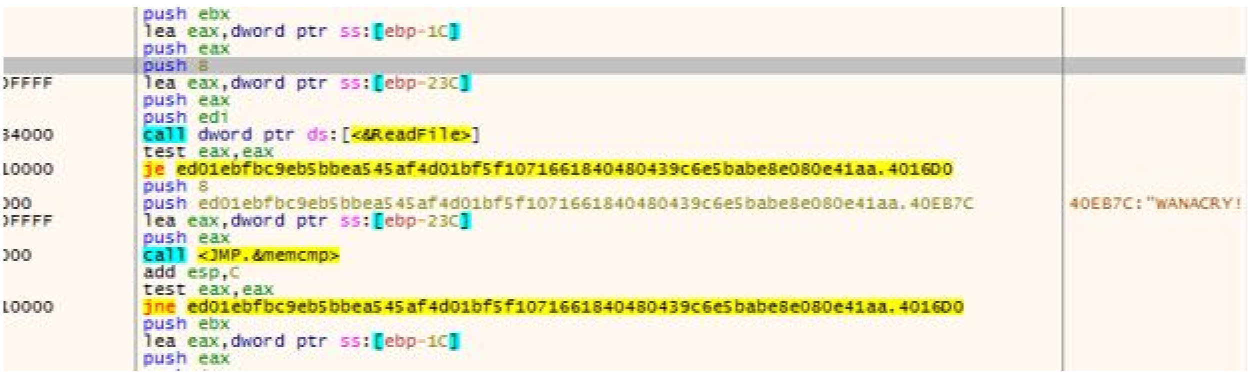


Below are screenshots showing how t.wnry would be read for WANACRY! and how the decryption of t.wnry would occur to create the new dll.

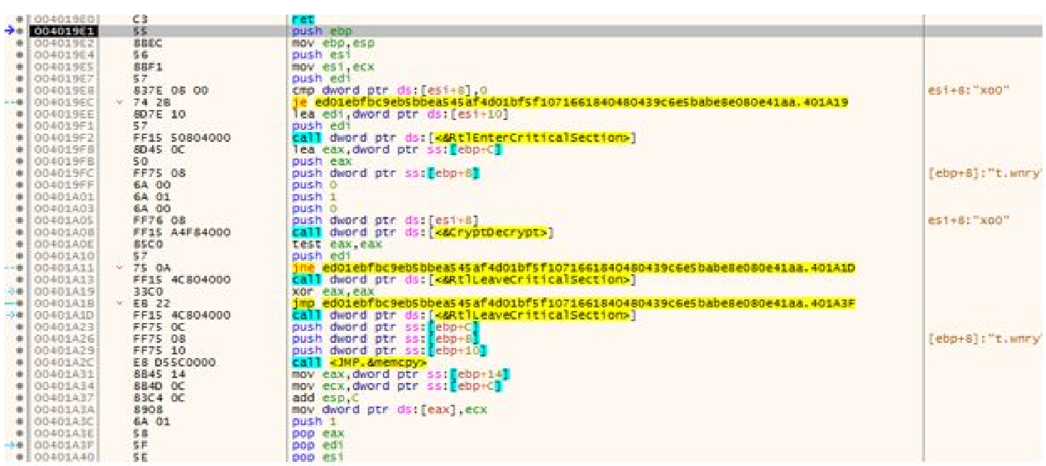


Advanced Dynamic

For advanced dynamic analysis, I used x32 Debug. I mainly focused on what happened after persistence was created by the original file since most of that was discovered through IDA, and the actual encryption part was called through memory instead, requiring dynamic analysis. After the files from XIA are extracted, the malware checks the file t.wnry for the value "WANACRY!".



T.wnry is then decrypted and saved as a dll. Next, it is called from memory using TaskStart. As the main executable does not perform the file encryption, I am assuming that this dll performs the actual encryption of the files.



Challenges

Challenges that I faced were mostly related to hidden encrypted files such as the executables in XIA and the encrypted dll. I was unsure of how to extract the XIA files at first, but eventually realized that it was a zip file. I opened up the executable in Resource Hacker and extracted XIA as a bin file which I then converted into a zip file. All I had to do then was enter the password “WNcry@2ol7” which I found in IDA in order to decrypt it. Inside were two executables related to file management and process handling as well as multiple .wnry files such as t.wnry. In regards to extracting the hidden dll, due to my lack of extended exposure to x32 Debug, I was unable to get access to the dll as it was created on the spot and loaded in from memory.

Summary

Overall, the malware can be executed in one of two ways, with the /i option, or without it. When installed with the /i option, it creates extra persistence by creating a new executable taskshe.exe if it doesn’t already exist, then running the service if it isn’t already running, and placing it in a new directory that it created at either C:\ProgramData\Random16characters\, C:\Intel\Random16characters\ or C:\Windows\Random16characters\. The malware then continues, or begins if the option /i was not selected, by creating a registry key Software/WanaCrypt0r for persistence to run the new executable or the original if a new one was not created. Next the hidden zip file XIA is unzipped into the current working directory of the malware(either the original location or the new one that was created). It then runs the commands “attrib +h .” and “icacls . /grant Everyone:F /T /C /Q”. After that it loads the functions CreateFileW, WriteFile, ReadFile, MoveFileW, MoveFileExW, DeleteFileW, and CloseHandle, CryptAcquireContextA, CryptImportKey, CryptDestroyKey, CryptEncrypt, CryptDecrypt, and

CryptGenKey as global variables as an attempt to obfuscate their usage in the executable. After these are loaded, one of the files that was extracted, t.wnry, is searched for the string "WANACRY!" and is then decrypted and saved as a dll. This dll is then started using TaskStart being called through memory to avoid being caught by standard malware detection. It is then assumed that the dll does the actual encryption of the files on the system, as there is no sign of encryption when examining the original malware in IDA.

In order to remove the malware, you will first want to remove the service registry entries Software\WanaCrypt0r and Software\Microsoft\Windows\CurrentVersion\Run\random16 characters value. You then need to remove any .wnry files and shortcuts to the readmes and wannadecryptor. This can be done manually, but I would recommend writing a script to complete this step. Next, delete the executable where it was originally run and delete the files that were originally created there. After that, go to the task manager and kill processes related to these executables in this order: Diskpart(original name of the original executable), tasksche, wannadecryptor, taskdl, taskse. Finally, go to the folder that is made up of a random 16 character string in ProgramData, Intel, or Windows and delete the entire folder. Now, WannaCry should be removed from the system, however, the files will not be recoverable.