Reversing Project -

WannaCry Ransomware

Analysis by: Enzo DeStephano

RIT - CSEC 202, Section 01

Table of Contents

[**Background**](#_eopnxn8hfkho) **3**

[**Environment/System**](#_ey863m2e34in) **3**

[**Basic Static Analysis**](#_w6xxula48cy) **4**

[VirusTotal](#_l2lsplujwnti) 4

[Strings/FLOSS](#_pl4rbysnpil8) 5

[PEiD](#_mtgzqpf88kcm) 6

[Resource Hacker](#_sft0mz3w6f21) 6

[Dependency Walker](#_j7skc0lhls0h) 7

[PEView](#_ychlajmqd86n) 7

[**Basic Dynamic Analysis**](#_4grp3pusu363) **8**

[Regshot](#_zetk4w8dnb8a) 8

[Process Monitor](#_503q9t4dle8g) 8

[Process Explorer](#_t50fye5b68s6) 10

[Network Related Tools](#_rghhjw1ke7yx) 10

[**Advanced Static Analysis**](#_bmsnivcp8ggi) **11**

[IDA Pro](#_v1gt2xq91cso) 11

[**Advanced Dynamic Analysis**](#_rbg2u9fatlll) **18**

[X32dbg](#_jmh51lebxjyc) 18

[**Potential Danger**](#_7gw13ga9uxqv) **21**

[**Malware Removal**](#_ncnl4rbgofz4) **21**

[**Final Thoughts**](#_v7xxmqciy487) **21**

[**Resources**](#_dpp7e7klrqns) **21**

# Background

In May of 2017, the ransomware WannaCry spread around the world, encrypting over 200,000 machines and costing hundreds of millions of dollars in damages. WannaCry used a Windows exploit, known as EternalBlue, as a way to affect as many systems as possible. The EternalBlue vulnerability was patched by Microsoft, but many machines had not updated their systems and were still open at the time of the WannaCry attack.

# Environment/System

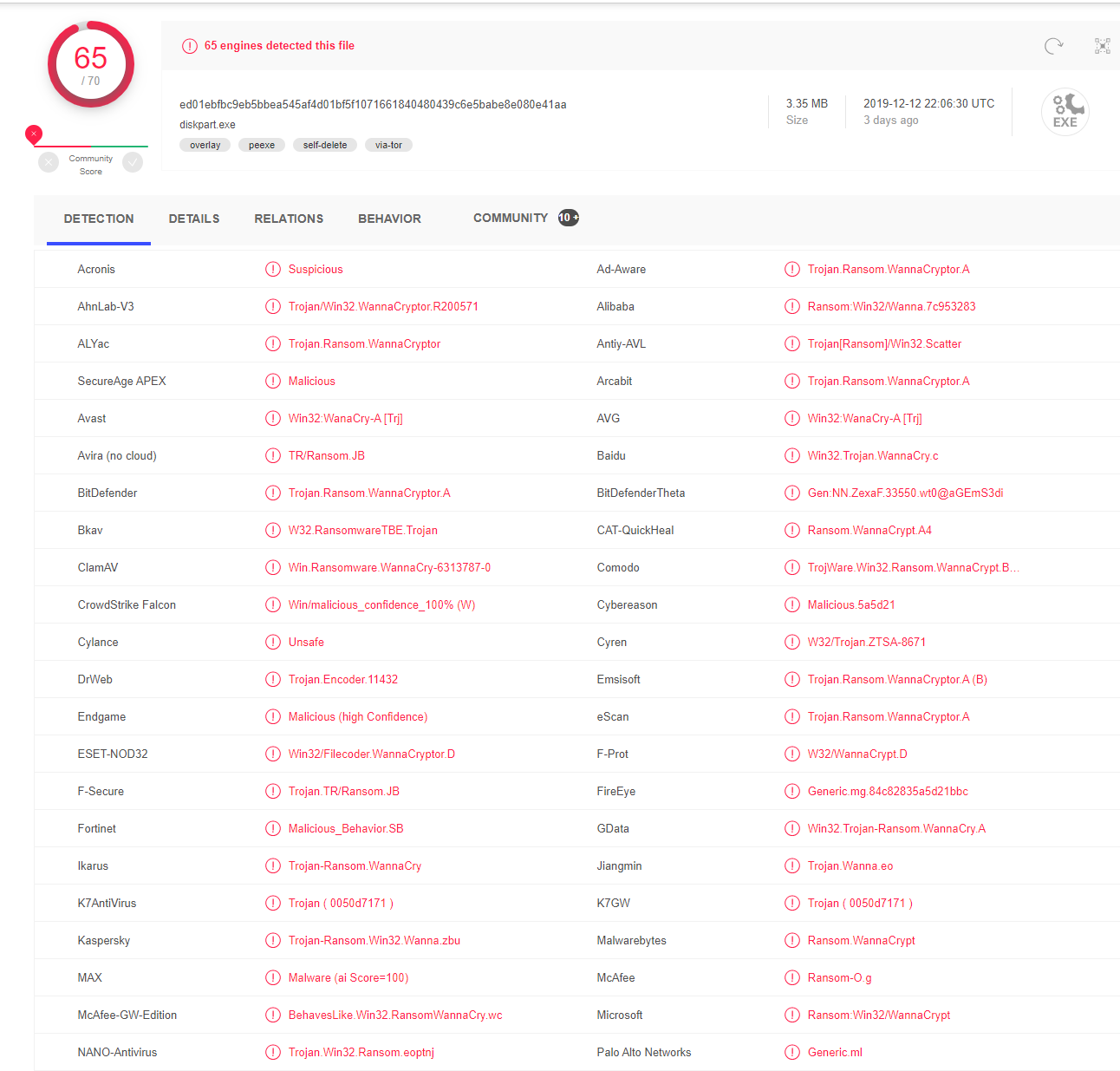
To analyze this WannaCry binary, I used a Windows 10 Virtual Machine being hosted within an air gapped lab. This was to prevent any chance of spreading if the malware was able to escape the virtual machine.

The WannaCry binary was downloaded from a malware Github called the Zoo. When looking at the screenshots, the executable file will be named either **ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe** or **wannacry.exe.**

# Basic Static Analysis

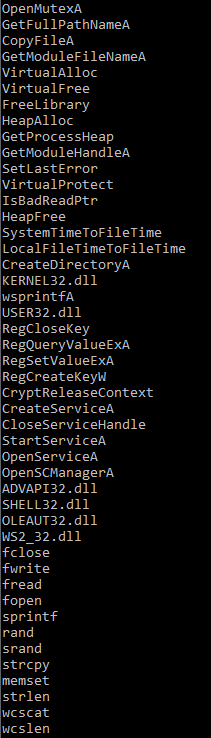
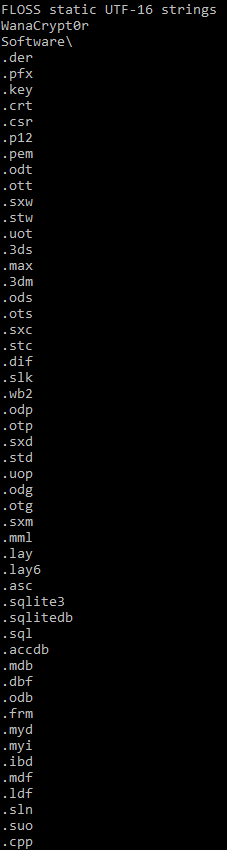
## VirusTotal

Starting off, VirusTotal required an Internet connection, so this was the only tool where the malware was placed on a system connected to the Internet. Since WannaCry was such an impactful piece of malware, it was easily detected by multiple different detection tools. If just given this binary with the random executable name shown below, it would be easy to find out both what type of malware it was and which malware it specifically was.



## Strings/FLOSS

Running FLOSS to find strings contained within the binary was very helpful in determining what might be going on. One of the first big sections of readable strings are calls to common Windows functions, not only helping display what it might do, but also showing that this malware may be meant to run on a Windows environment specifically.



In the first screenshot, some of the important values to check out are the calls to fwrite/fopen/fopen, StartServiceA, RegSetValueExA, and CreateFileA. There are many important others called, but looking at these specifically shows that files can be opened and changed, registry keys could be edited, and new files can be created.

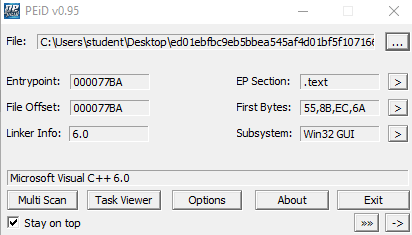
In the second screenshot, FLOSS found a bunch of file extensions, and a dead giveaway for the malware: the string WanaCrypt0r. These extensions are the types of files that will be encrypted by WannaCry.

Further investigation showed that WannaCry presented text options in multiple different languages, and some strings were decrypted by FLOSS but still not yet understandable.

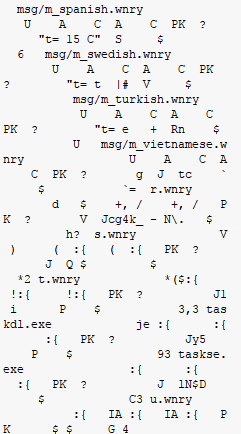
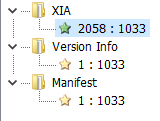
## 

## PEiD

PEiD is a program that is used to find out compiler/packer systems that might have been used when creating the executable. Placing the binary within PEiD shown it could have been compiled C++ code, and no packer was detected.

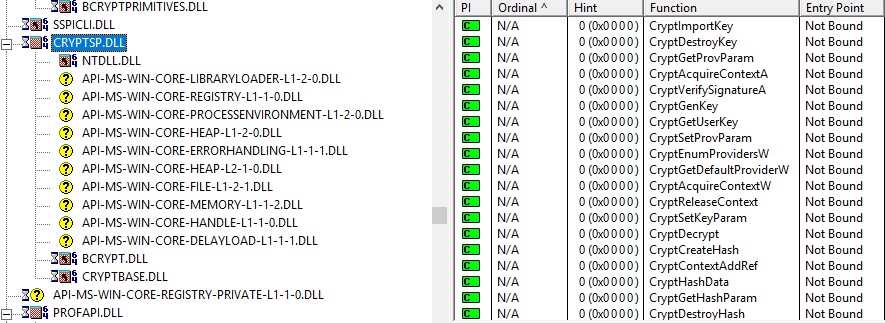


## Resource Hacker

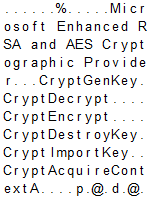
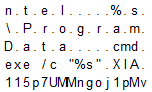
Taking a look at resource hacker, most of the important information can be gleaned from the XIA folder. It is not clear what XIA stands for, but it shows up multiple times later on. Within this XIA folder, the languages offered can be found, but with a custom extension: .wnry. There are multiple files that show up below with this extension, including r.wnry, s.wnry, t.wnry, and u.wnry. There are also some executable files that show up (taskdl and taskse).

## Dependency Walker

WannaCry uses a lot of different functions contained within a lot of DLLs, so dependency walker outlines these DLLs and their functions that are being used. There was a bunch of important data showing all of the functions being used, but some of the more interesting DLLs were Cryptsp.dll and bcrypt.dll. These contain many different cryptography functions that further prove that this binary is ransomware.



## PEView

PEView takes a look at the different types of headers within the PE executable file. Looking through the different headers brought to light some peculiar info, like shown in screenshot 1. “cmd.exe /c %s” appears, showing that the binary will execute something using the CMD shell. XIA shows up again, along with a random string that ends up being a bitcoin wallet address. In the next screenshot, more encryption information is shown, showing that RSA and AES encryption are being used. 

## 

# Basic Dynamic Analysis

## Regshot

Regshot shows edits that occur to the Windows Registry. Some instances that should be pointed out are the addition of the WanaCrypt0r key, the addition of certain values like the malware executable (possibly for persistence), and the modification of the background image. (This screenshot is very hard to read, so I recommend zooming in to see its details.)

## Process Monitor

Process Monitor was extremely helpful in showing the changes the specifically wannacry.exe made to the system, as I had a Process Name filter in place. I found that running process monitor while using x32 was also very helpful, as I could see exactly what subroutines were changing things on the system.

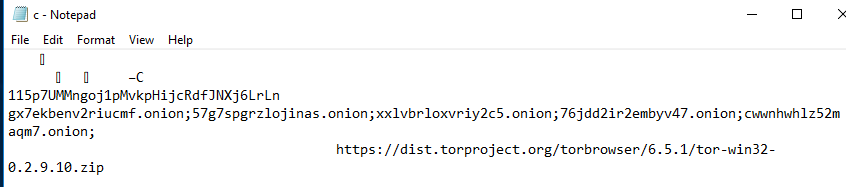
In the first screenshot, wannacry.exe creates multiple different .wnry files and writes important data to them. I opened the c.wnry file after it was written, and it contains a bitcoin wallet address and tor browser links, shown in the second screenshot.

## 

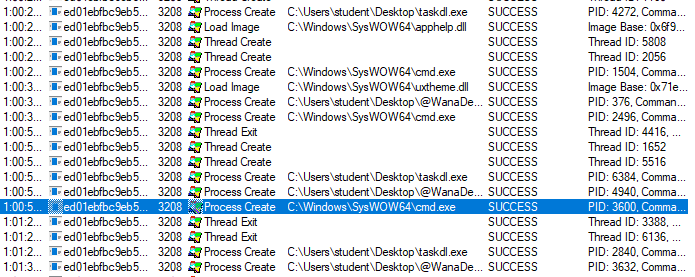
## 

## 

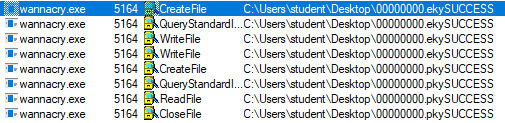
## 



Looking further into process monitor, some of the other process/threads that it creates are calls to multiple different executables, including the cmd shell and its own created .exe files.

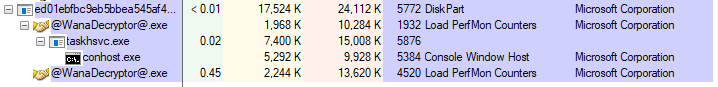


Lastly, there are two more very important files were shown to be created: 00000000.eky and 00000000.pky. The .pky file holds the public RSA encryption key, and the .eky file holds the AES encrypted RSA private key.



## Process Explorer

Process Explorer was used in this part of the analysis to show the process tree that running the wannacry malware had produced. Also, after about thirty seconds of killing the malware process within this program, it would show up again, proving that there was some sort of persistence in place to make sure the malware is always running and encrypting new files.



## Network Related Tools

Some of the networking tools that were used in this analysis were apateDNS, netcat, and Wireshark. Although these tools still work on a system without Internet access, WannaCry was not shown reaching out to any domains shown on these tools. ApateDNS did not detect show any irregular callouts to different domains, and netcat listening on port 80 did not pick anything up. Wireshark was also started before running the malware, but nothing but normal traffic showed. I found some tor domains within the binary, and the server must be communicated with in order for payment and decryption to occur, so I am slightly unsure of what happened here.

# 

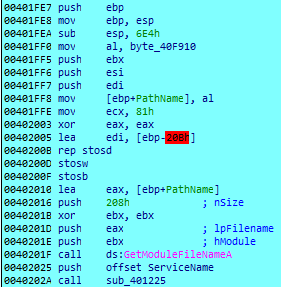
# Advanced Static Analysis

## IDA Pro

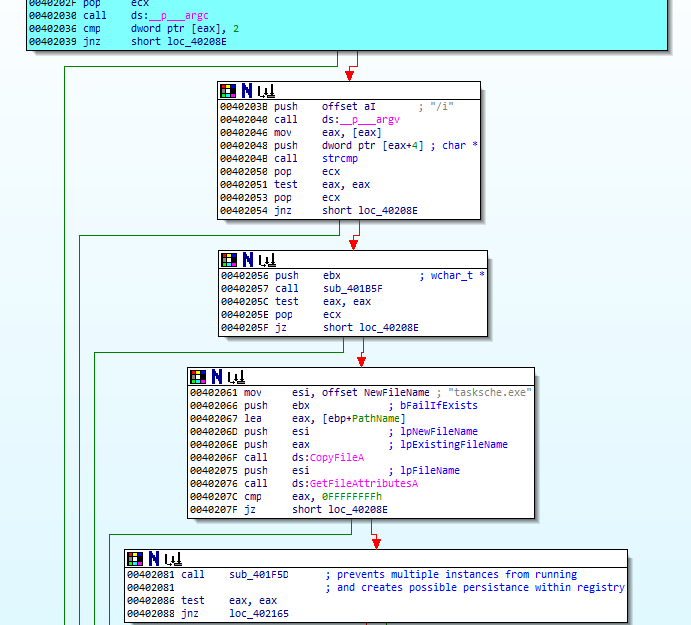
IDA starts out in the WinMain method of the binary.



There is a lot of prologue material, but the most important part is the obtaining of the PathName, which holds the path to the wannacry.exe file. This pathName variable is used many times further on.



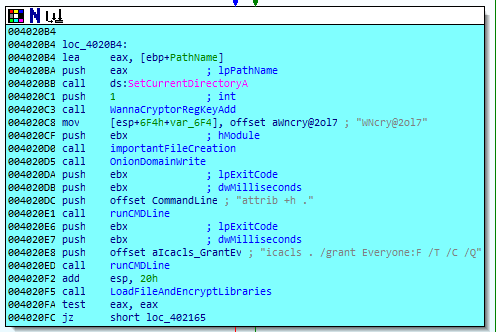
Right at the end of this WinMain prologue block is a check for the number of command line arguments. As can be seen, if “/i” is the command line argument, the code blocks in white in the screenshot below execute. The most important part of this is the creation of the tasksche.exe file, which is a copy of the wannacry.exe file. This executable is then placed as a service that starts consistently to enable persistence on the machine. It is also executed via the command line. This is shown in the next two screenshots, after the one below.





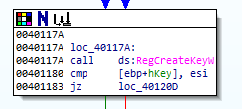


Looking into some of the subroutine functions called without the command line argument, the important ones that I found were the labelled: WannaCryptorRegKeyAdd, ImportantFileCreation, OnionDomainWrite, runCMDLine, and LoadFileandEncryptLibraries. I will be going each of these in order.

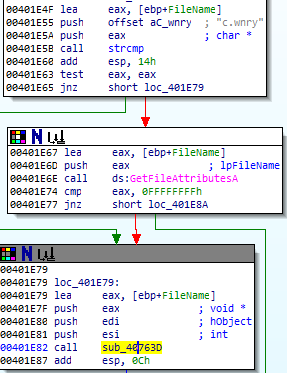


The first of these is the creation and addition of the WanaCrypt0r registry key. I was unable to determine its functionality, but process monitor obtained data from the key during execution.

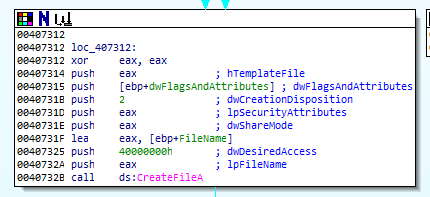




The next subroutine, labelled importantFileCreation, creates a bunch of the important files that are to be referenced during execution. IDA only shows the c.wnry file being created, but during x32dbg execution, there are multiple different .wnry and .exe files being created.



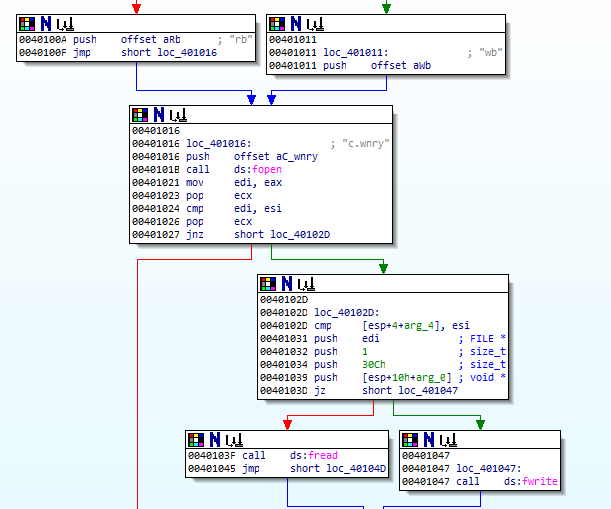
Diving far into the sub\_40763d subroutine, the CreateFileA function is found.



The next function is where the onion domains are written to the c.wnry file. These three random strings seem purposeless at first, but they are hardcoded bitcoin wallet addresses. (Check the screenshot on the title page, it’s the same!)



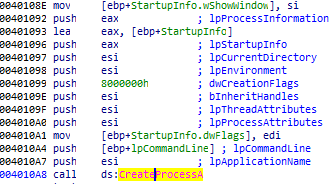
Going into the wncryUse subroutine, the c.wnry file is being opened and written/read to. The onion domains is not in plaintext, but show up in the file after this method is called.



The function labelled runCMDLine is executing the command being pushed as an argument above. This was



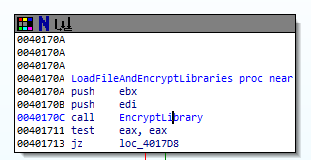
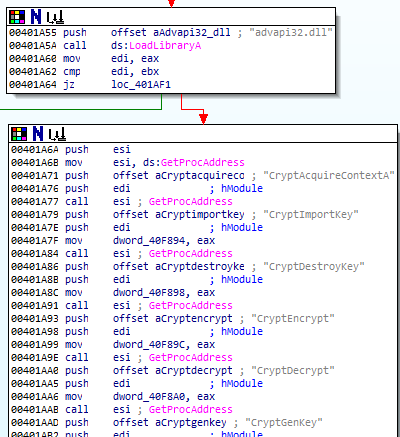
It is done through CreateProcessA, but the functionality was unclear to me.



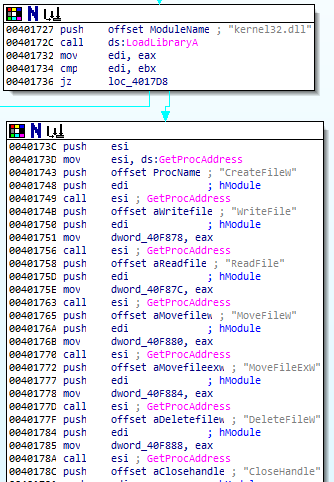
The next runCMDLine call has a more clear purpose, giving everyone (including the malware) permission to access all files on the machine.This is also done using CreateProcessA.



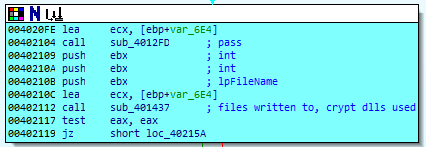
The last important subroutine in the code block above is the loading of the File and Encryption libraries. Within the prologue, the encryption library is loaded as shown in the two screenshots below.



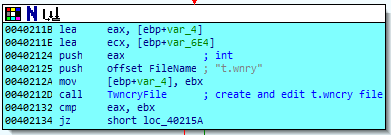
After loading the encryption functions and returning, the many file-related functions are loaded in.

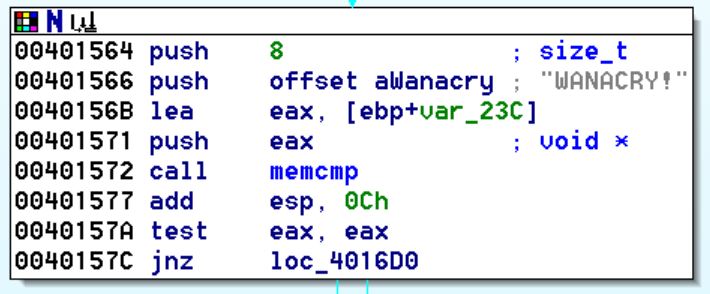


Some of the other interesting pieces shown in WinMain are sub\_401437, which wrote important data to a lot of the .wnry files and used some of the encryption dlls. The string “Microsoft AES and RSA” can also be found many subroutines deeper.

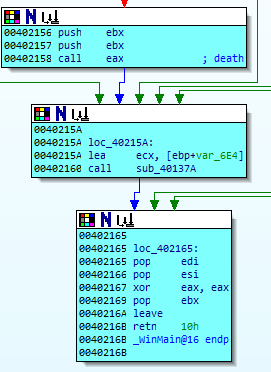


The next subroutine creates the t.wnry file, which holds a ton of encrypted data and the string “WANACRY!” at the beginning.





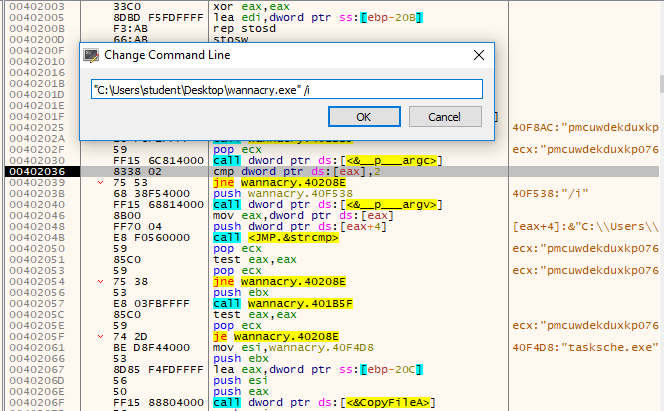
The last subroutine worth looking at in WinMain is this call to an address loaded in eax. IDA cannot look into this call because the address begins with 1000XXXX. Instead of the 0040XXXX like usual. This could only be analyzed while running in a debugger. After this call to eax, all of the files on the system become encrypted.



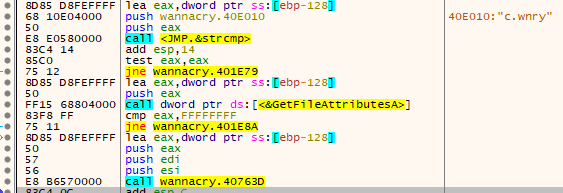
# Advanced Dynamic Analysis

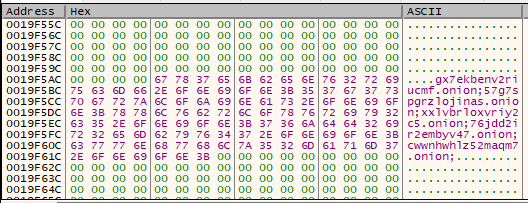
## X32dbg

The first thing that I looked at after checking through IDA was running the malware with and without the command line argument “/i”. After checking that the number of arguments was correct, strcmp was used to see if the given argument was equal to “/i”. After this, the addition and start of the service tasksche.exe began and the files were encrypted right after the service was started. If no argument was given, the program will set up all of the other files first before calling the eax location and encrypting all of the system files.



Next, going through the c.wnry subroutine where the bitcoin wallet addresses show up, tor addresses are found. After each wallet address was pushed to the stack, following their addresses in dump showed the following data below.





The next important part that x32 found was when diving into the call eax. (Note the 1000XXXX addresses) There are many calls to encryption DLLs, and two important files are created below with extensions .eky and .pky. As mentioned above, the .pky file holds the public RSA encryption key, and the .eky file holds the AES encrypted RSA private key.



# Potential Danger

WannaCry is a form of ransomware, and uses a strong method of encrypting files. Anything that is physically connected to a machine will have its data encrypted. If there aren’t any recent backups (or no backups), then the data loss that can occur will exact a heavy price. WannaCry is often also used in conjunction with multiple different types of malware such as worms that create backdoors, become persistent, and laterally move across as many devices as it can.

# Malware Removal

Unfortunately, there is no way tool made for decrypting data affected by Wannacry. Paying the bitcoin fine is also not recommended either, since there is no confirmation that you will actually get your data back. The best that could be done is to revert to an older backup, or just wipe the disk clean, since any new files that are created will also become encrypted due to the processes continually running and checking on the machine.

# Final Thoughts

Looking at this project as a whole, I feel as if I learned a lot of new things about analyzing a malicious binary. The main takeaway for me was using multiple tools at the same time. At first, I took a step by step approach, using each tool and then moving to the next one. Seeing that this was a much more complicated task at hand, using multiple resources at once helped simplify this greatly. As mentioned earlier, running process monitor and going step by step in x32dbg helped find important subroutines. Running IDA and x32 concurrently also was helpful as I could take a look inside a function in IDA before deciding if it was worth jumping into or not. Overall, my analysis was not perfect and there were definitely certain aspects that escaped me, but it was definitely doable with skills and tools at hand.

# Resources

Malware download link (Github:ytisf/theZoo): <https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Ransomware.WannaCry>

VirusTotal Scan: <https://www.virustotal.com/gui/home/upload>

Background: <https://en.wikipedia.org/wiki/WannaCry_ransomware_attack>

Windows Function Documentation: <https://docs.microsoft.com/en-us/windows/>