Ransomware Analysis and Defense
WannaCry and the Win32 environment

Justin Jones

August 3, 2017

Contents

1 Introduction

A type of malware known as *ransomware* has recently become very prevalent in the cyber security world, taking over user systems and demanding compensation for the safe return of system functionality and data. While initially not incredibly sophisticated, this type of software has evolved from simple scripts that change file extensions and make empty threats to full-blown attacks affecting hundreds of thousands of systems worldwide that implement sophisticated NSA-developed exploits as their propagation vector. In this paper I will explore a specific piece of malware known as *WannaCry* that recently made headlines around the world, performing a full static and dynamic analysis. Additionally, I will endeavor to write a useable piece of software to stop malware from executing within a Win32 environment.

2 Analysis

2.1 WannaCry/WCry

2.1.1 Background

WannaCry (referring to the general family consisting of all named variations of WannaCrypt, WCry, WanaCrypt, WanaCrypt0r, etc) came into prevalence during a massive attack starting on May 12, 2017. This software utilizes an exploit called EternalBlue[1], a known vulnerability in the Server Message Block (SMB) protocol used by Microsoft Windows which was previously patched in a critical update outlined in KB4013389[2]. As this vulnerability has been explored and detailed very thoroughly already, focus will be shifted to WanaCry's implementation and software aspects while avoiding the inner workings of the exploit.

The working sample of WannaCry has been obtained from theZoo[3], with SHA256 hash:

ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

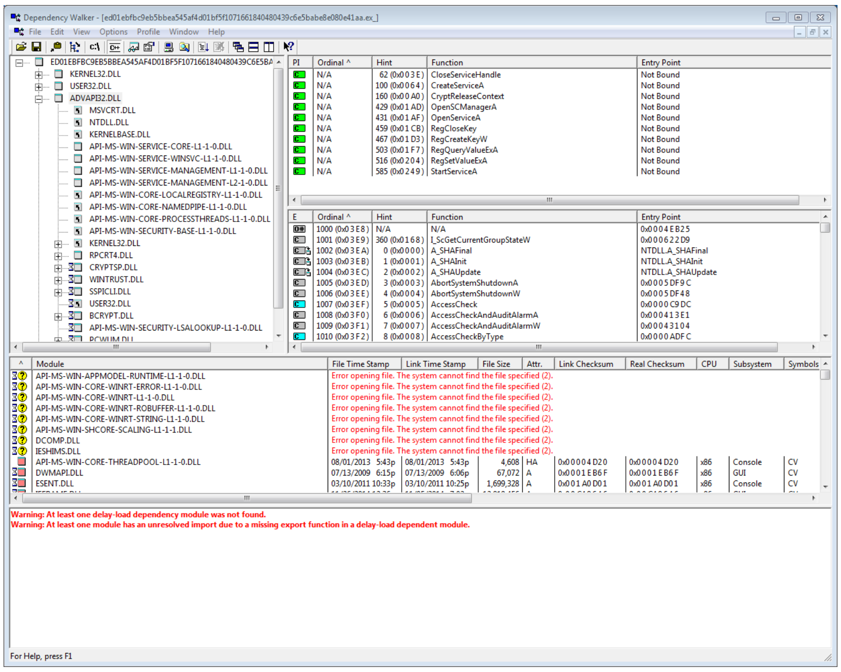
and is positively identified by VirusTotal as a member of the WanaCry family[4]. The software is being tested in a Microsoft-supplied 32-bit Windows 7 appliance as a VMWare Player virtual host. The appliance is given host-only network access and all outgoing traffic is recorded with Wireshark during analysis. All relevant analysis files are supplied in the GitHub repository[5] accompanying the paper, including the Wireshark pcap, IDA Pro idb, registry snapshots and a compressed file containing the extracted payload.

2.1.2 General File Data

Utilizing PEiD[6], it can be seen that the program was packed using Microsoft Visual Studio C++ 6.0 for Win32 and we should have no trouble unpacking it ourselves.

Utilizing Dependency Walker[7], we see that the program uses *ADVAPI32.DLL* which is the source of many cryptographic security functions implemented in Windows. In fact, this is where the SMB exploit *EternalBlue* is found, however had I not known ahead of time the program uses it I wouldn't gather that information from this examination until I go through the decompilation and execution. As it is, most if not all ransomware will want to utilize this dll if it intends to seriously encrypt any files. The inclusion of this library is the first indication that this software may be ransomware.

[1]http://bit.ly/2spdT15
[2]https://support.microsoft.com/en-us/help/4013389/title
[3]http://thezoo.morirt.com/
[4]http://bit.ly/2s93pCl
[5]https://github.com/NachoChef/Malware-Analysis-and-Defense
[6]https://www.aldeid.com/wiki/PEiD
[7]http://www.dependencywalker.com/



Dependency Walker - [ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.ex_]

```
ED01EBFBC9EB5BBEA545AF4D01BF5F1071661840480439C6E5BA
    KERNEL32.DLL
    USER32.DLL
    ADVAPI32.DLL
        MSVCRT.DLL
        NTDLL.DLL
        KERNELBASE.DLL
        API-MS-WIN-SERVICE-CORE-L1-1-0.DLL
        API-MS-WIN-SERVICE-WINSVC-L1-1-0.DLL
        API-MS-WIN-SERVICE-MANAGEMENT-L1-1-0.DLL
        API-MS-WIN-SERVICE-MANAGEMENT-L2-1-0.DLL
        API-MS-WIN-CORE-LOCALREGISTRY-L1-1-0.DLL
        API-MS-WIN-CORE-NAMEDPIPE-L1-1-0.DLL
        API-MS-WIN-CORE-PROCESSTHREADS-L1-1-0.DLL
        API-MS-WIN-SECURITY-BASE-L1-1-0.DLL
        KERNEL32.DLL
        RPCRT4.DLL
        CRYPTSP.DLL
        WINTRUST.DLL
        SSPICLI.DLL
        USER32.DLL
        BCRYPT.DLL
        API-MS-WIN-SECURITY-LSALOOKUP-L1-1-0.DLL
        PCWUM.DLL
```

PI	Ordinal ^	Hint	Function	Entry Point
C	N/A	62 (0x003E)	CloseServiceHandle	Not Bound
C	N/A	100 (0x0064)	CreateServiceA	Not Bound
C	N/A	160 (0x00A0)	CryptReleaseContext	Not Bound
C	N/A	429 (0x01AD)	OpenSCManagerA	Not Bound
C	N/A	431 (0x01AF)	OpenServiceA	Not Bound
C	N/A	459 (0x01CB)	RegCloseKey	Not Bound
C	N/A	467 (0x01D3)	RegCreateKeyW	Not Bound
C	N/A	503 (0x01F7)	RegQueryValueExA	Not Bound
C	N/A	516 (0x0204)	RegSetValueExA	Not Bound
C	N/A	585 (0x0249)	StartServiceA	Not Bound

E	Ordinal ^	Hint	Function	Entry Point
O#	1000 (0x03E8)	N/A	N/A	0x0004EB25
C	1001 (0x03E9)	360 (0x0168)	I_ScGetCurrentGroupStateW	0x000622D9
C	1002 (0x03EA)	0 (0x0000)	A_SHAFinal	NTDLL.A_SHAFinal
C	1003 (0x03EB)	1 (0x0001)	A_SHAInit	NTDLL.A_SHAInit
C	1004 (0x03EC)	2 (0x0002)	A_SHAUpdate	NTDLL.A_SHAUpdate
C	1005 (0x03ED)	3 (0x0003)	AbortSystemShutdownA	0x0005DF9C
C	1006 (0x03EE)	4 (0x0004)	AbortSystemShutdownW	0x0005DF48
C	1007 (0x03EF)	5 (0x0005)	AccessCheck	0x0000C9DC
C	1008 (0x03F0)	6 (0x0006)	AccessCheckAndAuditAlarmA	0x000413E1
C	1009 (0x03F1)	7 (0x0007)	AccessCheckAndAuditAlarmW	0x00043104
C	1010 (0x03F2)	8 (0x0008)	AccessCheckByType	0x0000ADFC

^	Module	File Time Stamp	Link Time Stamp	File Size	Attr.	Link Checksum	Real Checksum	CPU	Subsystem	Symbols
	API-MS-WIN-APPMODEL-RUNTIME-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-CORE-WINRT-ERROR-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-CORE-WINRT-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-CORE-WINRT-ROBUFFER-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-CORE-WINRT-STRING-L1-1-0.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-SHCORE-SCALING-L1-1-1.DLL	Error opening file. The system cannot find the file specified (2).								
	DCOMP.DLL	Error opening file. The system cannot find the file specified (2).								
	IESHIMS.DLL	Error opening file. The system cannot find the file specified (2).								
	API-MS-WIN-CORE-THREADPOOL-L1-1-0.DLL	08/01/2013 5:43p	08/01/2013 5:43p	4,608	HA	0x00004D20	0x00004D20	x86	Console	CV
	DWMAPI.DLL	07/13/2009 6:15p	07/13/2009 6:06p	67,072	A	0x0001EB6F	0x0001EB6F	x86	GUI	CV
	ESENT.DLL	03/10/2011 10:33p	03/10/2011 10:25p	1,699,328	A	0x001A0D01	0x001A0D01	x86	Console	CV

Warning: At least one delay-load dependency module was not found.
Warning: At least one module has an unresolved import due to a missing export function in a delay-load dependent module.

For Help, press F1

Figure 1: Dependency Walker overview of WannaCry

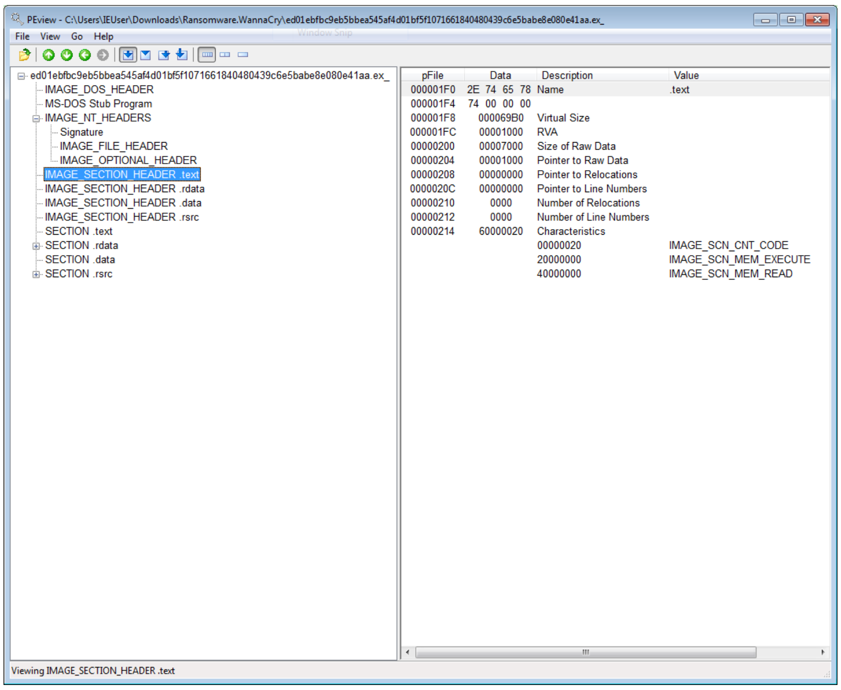
Looking into the functionality imported from this library, we see *CRYPT32.DLL*, and the imports immediately indicate this program is performing a large amount of cryptographic functions:

CryptDecrypt	CryptDeriveKey	CryptDestroyHash	CryptDestroyKey	CryptDuplicateHash
CryptDuplicateKey	CryptEncrypt	CryptEnumProviderTypesA	CryptEnumProviderTypesW	CryptEnumProvidersA
CryptEnumProvidersW	CryptExportKey	CryptGenKey	CryptGenRandom	CryptGetDefaultProviderA
CryptGetDefaultProviderW	CryptGetHashParam	CryptCreateHash	CryptGetProvParam	CryptGetUserKey
CryptHashData	CryptHashSessionKey	CryptImportKey	CryptReleaseContext	CryptSetHashParam
CryptSetKeyParam	CryptSetProvParam	CryptSetProviderA	CryptSetProviderExA	CryptSetProviderExW
CryptSetProviderW	CryptSignHashA	CryptSignHashW	CryptVerifySignatureA	CryptVerifySignatureW
CryptContextAddRef	CryptAcquireContextW	CryptGetKeyParam	CryptAcquireContextA	

We can also see that it uses the kernel library and the user library, specifically utilizing *GDI32.DLL* which gives access to local registry functions RegCloseKey, RegEnumValueW, and RegOpenKeyExW and is used for displaying graphics and creating GUIs. We also see *MSVCRT.DLL*, again indicating it was packed with Microsoft Visual Studio.

Utilizing PEview[8] to examine the PE header, we see a compile time of 11/20/2010. As this is a very recent exploit, this is not likely the true compile time for obvious reasons. More likely, the time was either faked or the system clock of the compiling machine was not accurately set. The subsystem indicated is *IMAGE_SUBSYSTEM_WINDOWS_GUI* which offers further evidence that the program utilizes a graphical interface. The virtual size and raw data size are about the same, indicating a more sophisticated packer was likely not used and the program was generated by a compiler. Accordingly there will likely not be dynamic unpacking within memory during execution, so a full memory dump would not greatly benefit the analysis although in many cases it would prove useful.

[8]http://wjradburn.com/software/



PEview - C:\Users\IEUser\Downloads\Ransomware.WannaCry\ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.ex_
File View Go Help

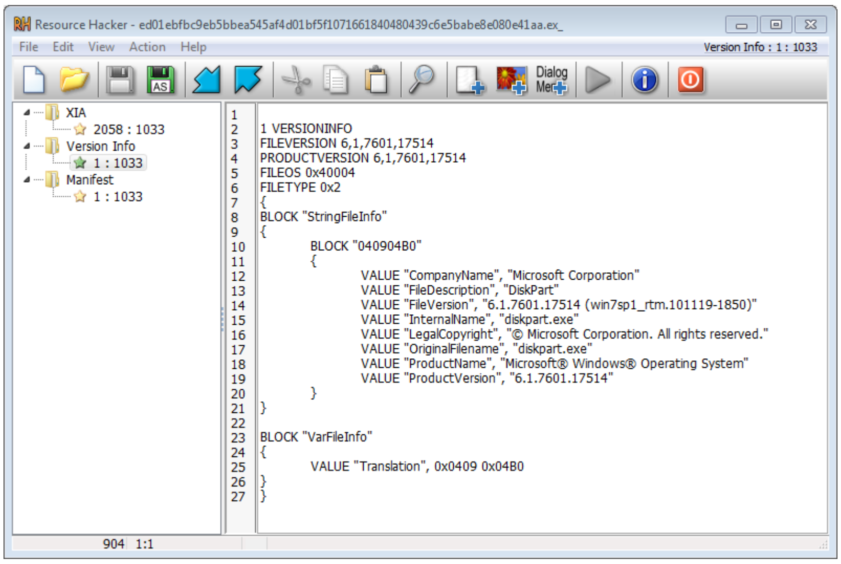
```
ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.ex_
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
    Signature
    IMAGE_FILE_HEADER
    IMAGE_OPTIONAL_HEADER
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .rdata
  IMAGE_SECTION_HEADER .data
  IMAGE_SECTION_HEADER .rsrc
  SECTION .text
  SECTION .rdata
  SECTION .data
  SECTION .rsrc
```

```
pFile      Data         Description               Value
000001F0   2E 74 65 78  Name                      .text
000001F4   74 00 00 00
000001F8   000069B0     Virtual Size
000001FC   00001000     RVA
00000200   00007000     Size of Raw Data
00000204   00001000     Pointer to Raw Data
00000208   00000000     Pointer to Relocations
0000020C   00000000     Pointer to Line Numbers
00000210   0000         Number of Relocations
00000212   0000         Number of Line Numbers
00000214   60000020     Characteristics
                        00000020                  IMAGE_SCN_CNT_CODE
                        20000000                  IMAGE_SCN_MEM_EXECUTE
                        40000000                  IMAGE_SCN_MEM_READ
```

Figure 2: PE headers for WannaCry

Examination with Resource Hacker[9] indicates that the software is trying to mascarade as diskpart.exe, and offers no useful information otherwise because the WannaCry files are encrypted in the "WNcry@2017" archive and are password protected, which we will see in the IDA Pro analysis.

[9]http://angusj.com/resourcehacker/


Resource Hacker - ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.ex_
File Edit View Action Help
Version Info : 1 : 1033
Dialog Mer
XIA
2058 : 1033
Version Info
1 : 1033
Manifest
1 : 1033
904 1:1

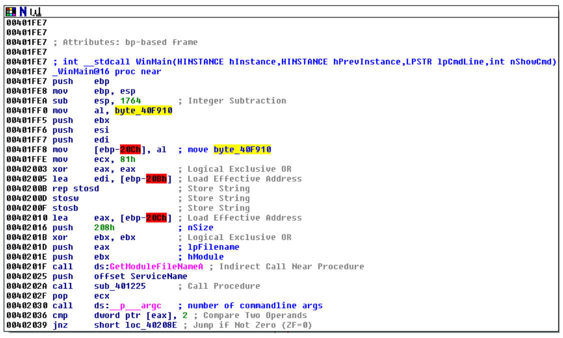
```
1 VERSIONINFO
FILEVERSION 6,1,7601,17514
PRODUCTVERSION 6,1,7601,17514
FILEOS 0x40004
FILETYPE 0x2
{
BLOCK "StringFileInfo"
{
        BLOCK "040904B0"
        {
                VALUE "CompanyName", "Microsoft Corporation"
                VALUE "FileDescription", "DiskPart"
                VALUE "FileVersion", "6.1.7601.17514 (win7sp1_rtm.101119-1850)"
                VALUE "InternalName", "diskpart.exe"
                VALUE "LegalCopyright", "© Microsoft Corporation. All rights reserved."
                VALUE "OriginalFilename", "diskpart.exe"
                VALUE "ProductName", "Microsoft® Windows® Operating System"
                VALUE "ProductVersion", "6.1.7601.17514"
        }
}

BLOCK "VarFileInfo"
{
        VALUE "Translation", 0x0409 0x04B0
}
}
```

Figure 3: Resource Hacker & WannaCry

2.1.3 Decompilation

Before moving into the code, a quick look at the Strings window confirms this is some variant of WannaCry or a program pretending to be such. Utilizing the IDA call flow graph[10], it can be seen that in addition to the basic startup functionality the program also calls _WinMain@16, from which the rest of the function calls will cascade from as this is where we unpack the archive. The software utilizes try/catch/finally blocks and extensive memory manipulation to accomplish its task, however in many cases I found that the creator(s) seemingly failed to turn on optimization in their compiler as shown in Fig 4.

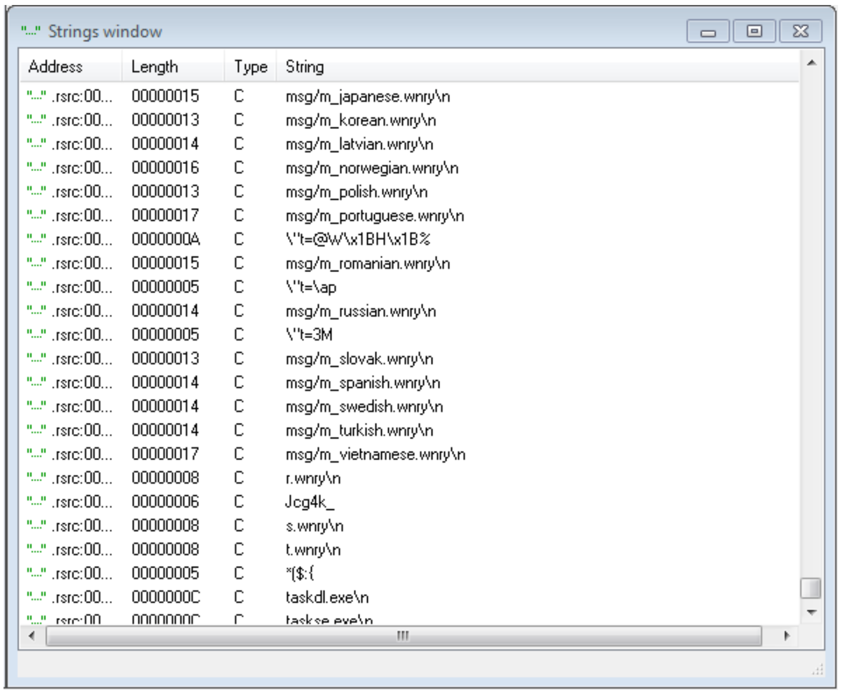


```
00401FE7
00401FE7
00401FE7 ; Attributes: bp-based frame
00401FE7
00401FE7 ; int __stdcall WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nShowCmd)
00401FE7 _WinMain@16 proc near
00401FE7 push    ebp
00401FE8 mov     ebp, esp
00401FEA sub     esp, 1764       ; Integer Subtraction
00401FF0 mov     al, byte_40F910
00401FF5 push    ebx
00401FF6 push    esi
00401FF7 push    edi
00401FF8 mov     [ebp-20Ch], al  ; move byte_40F910
00401FFE mov     ecx, 81h
00402003 xor     eax, eax        ; Logical Exclusive OR
00402005 lea     edi, [ebp-20Bh] ; Load Effective Address
0040200B rep stosd               ; Store String
0040200D stosw                   ; Store String
0040200F stosb                   ; Store String
00402010 lea     eax, [ebp-20Ch] ; Load Effective Address
00402016 push    208h            ; nSize
0040201B xor     ebx, ebx        ; Logical Exclusive OR
0040201D push    eax             ; lpFilename
0040201E push    ebx             ; hModule
0040201F call    ds:GetModuleFileNameA ; Indirect Call Near Procedure
00402025 push    offset ServiceName
0040202A call    sub_401225      ; Call Procedure
0040202F pop     ecx
00402030 call    ds:__p___argc   ; number of commandline args
00402036 cmp     dword ptr [eax], 2 ; Compare Two Operands
00402039 jnz     short loc_40208E ; Jump if Not Zero (ZF=0)
```

Figure 4: Poor Optimization

Additionally, after examining the try/catch/finally blocks, I believe that if any files were missing from the archive or otherwise corrupted upon extraction, the program would likely fail to fully take over the system and not execute. We can also see the filenames included in the archive, as the software has the names hard-oded for opening and manipulation, in Fig 5. In some cases, the functionality of these files isn't fully uncovered until the dynamic analysis.

[10]This image is very large, so it is provided in the repository [wcry.gdl] rather than inline.



```
"..." Strings window
Address          Length      Type   String
"..." .rsrc:00...  00000015    C      msg/m_japanese.wnry\n
"..." .rsrc:00...  00000013    C      msg/m_korean.wnry\n
"..." .rsrc:00...  00000014    C      msg/m_latvian.wnry\n
"..." .rsrc:00...  00000016    C      msg/m_norwegian.wnry\n
"..." .rsrc:00...  00000013    C      msg/m_polish.wnry\n
"..." .rsrc:00...  00000017    C      msg/m_portuguese.wnry\n
"..." .rsrc:00...  0000000A    C      \'t=@W\x1BH\x1B%
"..." .rsrc:00...  00000015    C      msg/m_romanian.wnry\n
"..." .rsrc:00...  00000005    C      \'t=\ap
"..." .rsrc:00...  00000014    C      msg/m_russian.wnry\n
"..." .rsrc:00...  00000005    C      \'t=3M
"..." .rsrc:00...  00000013    C      msg/m_slovak.wnry\n
"..." .rsrc:00...  00000014    C      msg/m_spanish.wnry\n
"..." .rsrc:00...  00000014    C      msg/m_swedish.wnry\n
"..." .rsrc:00...  00000014    C      msg/m_turkish.wnry\n
"..." .rsrc:00...  00000017    C      msg/m_vietnamese.wnry\n
"..." .rsrc:00...  00000008    C      r.wnry\n
"..." .rsrc:00...  00000006    C      Jcg4k_
"..." .rsrc:00...  00000008    C      s.wnry\n
"..." .rsrc:00...  00000008    C      t.wnry\n
"..." .rsrc:00...  00000005    C      *($:{
"..." .rsrc:00...  0000000C    C      taskdl.exe\n
"..." .rsrc:00     0000000C    C      taskse.exe\n
```

Figure 5: *.wnry filenames

Filenames & Descriptions	
b.wnry	Background image
c.wnry	Configuration
s.wnry	Tor communication (endpoints from c.wnry)
t.wnry	Default keys
u.wnry	User interface
taskdl.exe	Cleanup
taskse.exe	Support
msg	A folder of language packs

Moving into the software workflow, the program enters the main method, _WinMain@16, and extracts a password protected archive named "WNcry@2ol7" and creates the necessary services and procedures. The program retrieves host information, as well as the correct language file. It then loops through the extracted files in the current working directory, and elevates privileges. It then collects and terminates a number of processes that it has pushed into the system stack.[11] This is seemingly so that the files won't be in use which allows the processes to encrypt them without trouble. The software then continues to loop through directories and encrypts files as it goes, appending them with the .WNCRY extension, seemingly for 'shock value' with *tasksche.exe*. It currently attacks 151 different file types, which can be found in the decompilation, but this count seems to vary among the different variants. The program deletes Shadow Volume Copies and disables backup restoration. It finally displays the 'lock' screen along with the rest of the UI contained in *u.wnry* and implemented as *@WanaDecryptor@.exe*. Once the payment is verified, the program then goes through and decrypts all of the encrypted files, and exits. The program

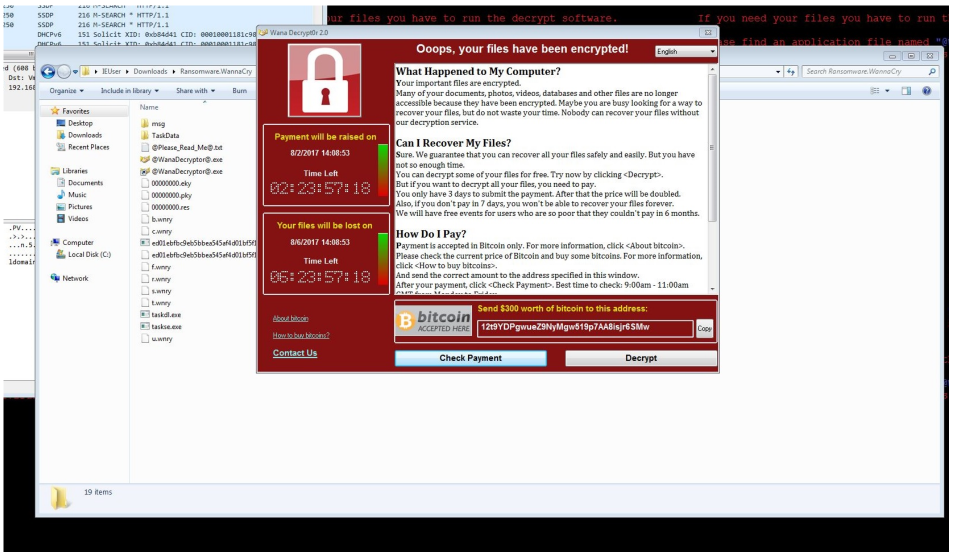
[11]After searching online, I discovered these processes were specifically related to database and mail servers.

utilizes a TOR client for communication with the attackers to verify payment, which is retrieves before the encryption begins.

One of the notable early discoveries was that there was a built in 'kill-switch' that would stop the software from fully taking over systems and encrypting files, however it should be noted that the version of software I have does not include this kill-switch. It seems to be the only difference between the 'original' and the version that I am examining.

2.1.4 Dynamic Analysis

The file is being launched directly from ˜/Downloads/Ransomware.WannaCry/. Upon initial launch, the wallpaper is immediately changed, and you can see the dropper extracts the files from the embedded archive into the root exe directory. These files have a creation date of 5/9/2017 to 5/12/2017, falling more in-line with the outbreak event and most likely accurate. Being that the last modified date is equivalent, some of the metadata was likely lost, obscured or otherwise modified at some point of the process.


Wana Decrypt0r 2.0

Ooops, your files have been encrypted!

What Happened to My Computer?
Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?
Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You can decrypt some of your files for free. Try now by clicking <Decrypt>.
But if you want to decrypt all your files, you need to pay.
You only have 3 days to submit the payment. After that the price will be doubled.
Also, if you don't pay in 7 days, you won't be able to recover your files forever.
We will have free events for users who are so poor that they couldn't pay in 6 months.

How Do I Pay?
Payment is accepted in Bitcoin only. For more information, click <About bitcoin>.
Please check the current price of Bitcoin and buy some bitcoins. For more information, click <How to buy bitcoins>.
And send the correct amount to the address specified in this window.
After your payment, click <Check Payment>. Best time to check: 9:00am - 11:00am

Payment will be raised on
8/2/2017 14:08:53
Time Left
02:23:57:18

Your files will be lost on
8/6/2017 14:08:53
Time Left
06:23:57:18

Send $300 worth of bitcoin to this address:
12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw

Figure 6: Initial Infection

After the payload is extracted and execution begins, a UAC prompt pops up requesting elevated privileges (Fig 7). If the victim denies the privileges, the software is still able to continue with encryption but cannot delete Shadow Volumes or backups.

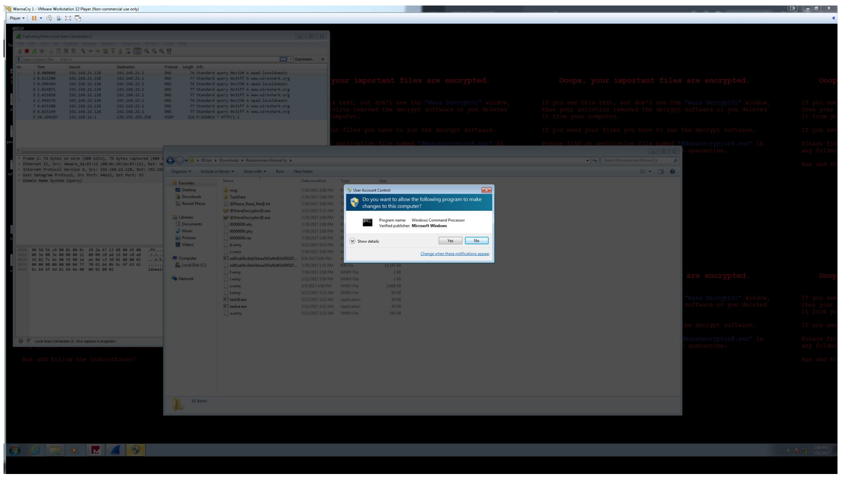
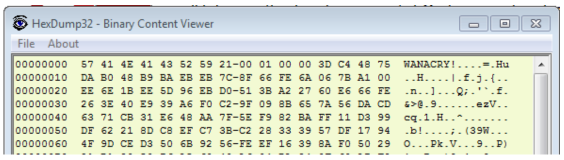


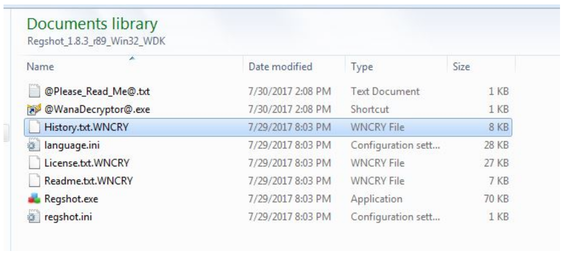
Figure 7: UAC Prompt on first run

The program creates additional files to accompany the files extracted with the archive. *f.wnry* stores a list of decrypted files, and *r.wnry* is a copy of @Please_Read_Me@.txt. *WanaDecryptor.exe* is the user GUI implementation, and the TOR client is retrieved into the *TaskData* folder. Additionally, RSA key files are created for the encryption/decryption processes. The actual encryption simply navigates down through directories encrypting files, modifying the file headers (Fig 8) and leaving a copy of @Please_Read_Me@.txt and a shortcut to @WannaDecryptor@.exe (Fig 9) in the directory.



```
00000000  57 41 4E 41 43 52 59 21-00 01 00 00 3D C4 48 75  WANACRY!....=.Hu
00000010  DA B0 48 B9 BA EB EB 7C-8F 66 FE 6A 06 7B A1 00  ..H....|.f.j.{..
00000020  EE 6E 1B EE 5D 96 EB D0-51 3B A2 27 60 E6 66 FE  .n..]...Q;.'`.f.
00000030  26 3E 40 E9 39 A6 F0 C2-9F 09 8B 65 7A 56 DA CD  &>@.9......ezV..
00000040  63 71 CB 31 E6 48 AA 7F-5E F9 82 BA FF 11 D3 99  cq.1.H..^.......
00000050  DF 62 21 8D C8 EF C7 3B-C2 28 33 39 57 DF 17 94  .b!....;.(39W...
00000060  4F 9D CE D3 50 6B 92 56-FE EF 16 39 8A F0 50 29  O...Pk.V...9..P)
```

Figure 8: Modified File Headers



Name	Date modified	Type	Size
@Please_Read_Me@.txt	7/30/2017 2:08 PM	Text Document	1 KB
@WanaDecryptor@.exe	7/30/2017 2:08 PM	Shortcut	1 KB
History.txt.WNCRY	7/29/2017 8:03 PM	WNCRY File	8 KB
language.ini	7/29/2017 8:03 PM	Configuration sett...	28 KB
License.txt.WNCRY	7/29/2017 8:03 PM	WNCRY File	27 KB
Readme.txt.WNCRY	7/29/2017 8:03 PM	WNCRY File	7 KB
Regshot.exe	7/29/2017 8:03 PM	Application	70 KB
regshot.ini	7/29/2017 8:03 PM	Configuration sett...	1 KB

Figure 9: An example directory after infection

The software completely destroys the registry, in the case of this test machine deleting 324 keys. The program then adds 1547 new keys and modifies the remaining registry keys, entirely focused on wholly taking over the system. I was unable to find any saved record of these changes for the program, so it doesn't seem that the host is actually completely restorable upon payment, as the software claims.

The IP address for the victim in this test is *192.168.22.254*, and the VMware Virtual Adapter IP address is *192.168.22.1*. The initial Wireshark capture was started on the victim network adapter previous to infection, and showed no traffic with no network applications active. Once the infection is initialized, we see regular SSDP transmissions followed by DHCP Solicit XID requests attempting to discover network devices and obtain new host information. Continuing with the capture, we see TCP requests and TLSv1 packets to retrieve the key whenever we try to verify payment, indicative of the TOR communication lines.

2.1.5 Conclusion

The WannaCry ransomware is reasonably sophisticated, and takes advantage of the EternalBlue vulnerability as a reliable propagation vector as evidenced by the infection count of an estimated 230,000 machines. The best option to prevent passive infection is to apply Microsoft update KB4013389, and to otherwise follow best practices regarding downloaded files, email attachments, etc. I additionally tested a program called *Wanakiwi*[12] that successfully retrieved a decryption key, however it only works if the system hasn't been restarted since infection and must be used as soon as possible to ensure the needed information is still in memory. The registry keys were not restored upon decryption, so once the needed information is retrieved it is best to perform a fresh install of the OS in any case and check all devices on the network, including network-attached storage.

3 The Defense Software

3.1 An Overview

The endeavor to write a defense program was based around a desire to become more familiar with the Windows NT environment and general antivirus development processes. The general operation of the software can be seen below in Alg. 1. A system callback, SetCreateProcessNotifyRoutineEx, is used to listen for process creation by using PsSetCreateProcessNotifyRoutineEx. This is implemented through a simple kernel-level driver that will launch a system *ProcessEvent* that can be caught by any processes that choose to listen. Specifically for this implementation, upon launch the process handle is passed into a queue of processes to be 'processed'. Whenever a new item enters this queue, the executable path is retrieved, and the file is hashed. If there is no known definition based on this hash, the hash is sent to VirusTotal utilizing the public C++ API, version 2.0. The result is then used to make a new entry into the database, and process creation either continues or is denied as deemed necessary by the results. I considered any process with more than 5 references to be unsafe, although realistically one could deny launch for even 1 VT reference.

The first hurdle with developing this software was creating a kernel driver. Having no previous experience myself, I read *Windows Internals, 6th ed.*[13] and also followed a tutorial on CodeProject.com[14]. Finally, I also referenced a Microsoft-supplied example driver found on their Windows Example drivers GitHub repository.[15] Because the kernel should not be made to wait for a response that is any longer than a few milliseconds (at most!), the execution is denied by default and the process will be relaunched if it has no known results from VirusTotal. Naturally in the process of working with a kernel driver, I learned much of the Windows NT API

[12]https://github.com/gentilkiwi/wanakiwi
[13]https://docs.microsoft.com/en-us/sysinternals/learn/windows-internals
[14]http://bit.ly/2lxXIv5
[15]https://github.com/Microsoft/Windows-driver-samples

to handle events and perform actions around those events.

```
Initialization:
Register PsSetCreateProcessNotifyRoutineEx callback;
Wait for process creation
Upon creation, run the following:
Data: Process Handle
Hash the Process Executable;
if hash in database then
    if {db_entry}.allow == True then
        allow execution;
        print allowance message;
    else
        deny execution;
        print denial message;
    end
else
    transmit hash to VirusTotal;
    {receive reply}
    if no VT results then
        allow execution;
        print allowance message;
    else
        deny execution;
        print denial message;
    end
    construct new database entry;
    set tuple launch permission;
    add to database;
end
```

Algorithm 1: A generalized algorithm

I originally intended to fully write the kernel driver myself; it soon became clear that without heavy background knowledge of C++ and the Windows API already it would be incredibly difficult. The next best option would be to adapt a currently existing open-source callback driver for my uses. I initially attempted to use a provided Microsoft example implementation, however the original intentions for the program were very different from what I was intending to use it for and adaptation was difficult. I used this as my development basis for about 4 or 5 weeks, however I eventually switched to the implementation provided by Ivo Ivanov on CodeProject.com, referenced previously in this section. Within this implementation I removed the test instantiations, and added sqlite3 functionality for the database and the VirusTotal API as my 'engine'. For actually sending off tests to VirusTotal, I elected to compute the SHA-256 digest of the executable to be launched using OpenSSL/libcrypto after retrieving the process path from the process handle. Section 4.3 contains a list of the 3rd party APIs that I used for this software.

Unfortunately at the time of this writing, the software does not compile and is not complete. Currently, the bulk of my issues are centered around a few of the 3rd party APIs and their dependencies and I have not been able to test program functionality. That being said, I'm sure there are other issues that I will encounter that are code-centric once I sort these problems out.

4 Appendix A

4.1 List of affected file extensions

.der .pfx .key .crt .csr .pem .odt .ott .sxw .stw .uot .max .ods .ots .sxc .stc .dif .slk .odp .otp .sxd .std .uop .odg .otg .sxm .mml .lay .lay6 .asc .sqlite3 .sqlitedb .sql .accdb .mdb .dbf .odb .frm .myd .myi .ibd .mdf .ldf .sln .suo .cpp .pas .asm .cmd .bat .vbs .dip .dch .sch .brd .jsp .php .asp .java .jar .class .wav .swf .fla .wmv .mpg .vob .mpeg .asf .avi .mov .mkv .flv .wma .mid .djvu .svg .psd .nef .tiff .tif .cgm .raw .gif .png .bmp .jpg .jpeg .vcd .iso .backup .zip .rar .tgz .tar .bak .tbk .paq .arc .aes .gpg .vmx .vmdk .vdi .sldm .sldx .sti .sxi .hwp .snt .onetoc2 .dwg .pdf .wks .rtf .csv .txt .vsdx .vsd .edb .eml .msg .ost .pst .potm .potx .ppam .ppsx .ppsm .pps .pot .pptm .pptx .ppt .xltm .xltx .xlc .xlm .xlt .xlw .xlsb .xlsm .xlsx .xls .dotx .dotm .dot .docm .docb .docx .doc

4.2 Database Diagram

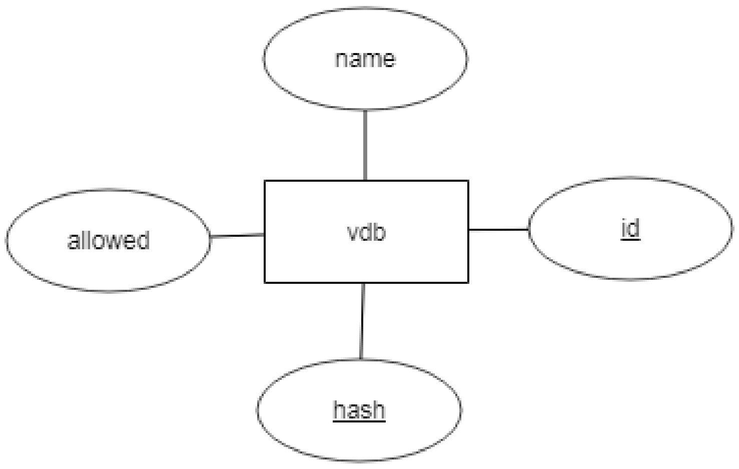


Figure 10: VDB Database Diagram

4.3 3rd Party Dependencies

sqlite3	https://sqlite.org/index.html
jansson	http://www.digip.org/jansson/
openssl	https://www.openssl.org/
vtapi	https://github.com/VirusTotal/c-vtapi
libcurl	https://curl.haxx.se/libcurl/