Ransomware Analysis and Defense

WannaCry and the Win32 environment

Justin Jones, Narasimha Shashidhar

Department of Computer Science, Sam Houston State University, Huntsville, TX, USA.
e-mail: {jxj037, karpoor}@shsu.edu

Abstract—Ransomware is a specific type of malware that threatens the victim's access to her data unless a ransom is paid. It is also known as a cryptovirus due to its method of operation. Typically, ransomware encrypts the contents of the victim's hard drive thereby rendering it inaccessible to the victim. Upon payment of the ransom, the decryption key is released to the victim. This means of attack is therefore also sometimes aptly called cryptoviral extortion. The ransomware itself is delivered to the victim using several channels. The most common channel of delivery is by masquerading the malware as a Trojan horse via an email attachment. In this work, we study a high-profile example of a ransomware called the *WannaCry worm*. This ransomware is particularly malicious since it has the ability to traverse computing equipment on a network without any human intervention. Since this worm has had a large scale impact, we find it imperative and instructive to better understand the inner workings of this high-profile ransomware. To this end, we obtain a sample of WannaCry and dissect it completely using advanced static and dynamic malware analysis techniques. This effort, we hope, will shed light on the inner workings of the malware and will enable cyber security experts to better thwart similar attacks in the future by: a) generating appropriate signatures and b) developing stronger defense solutions. Our analysis is conducted in a Win32 environment and we present our detailed analysis so as to enable reproduction of our work by other malware analysts. This, we hope, will further advancement in generating appropriate signatures to detect the worm. Secondly, we present a prototype software that will enable a user to prevent this malware from unleashing its payload and protect the user on a Win32 environment in an effort to advance the development of efficient software defense mechanisms to protect users from such a worm attack in the future.

Keywords—Ransomware, cryptovirus, extortion, static and dynamic analysis, malware analysis, cyber security.

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 ransom for the safe return of system functionality while holding the user's data hostage. While initially not incredibly sophisticated, this type of software has evolved from simple scripts that change file extensions and making 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, we explore a specific piece of malware known as *WannaCry* that recently made headlines around the world and we perform a full static and dynamic analysis to explore its inner workings. Seeking a full understanding of this malware is

a fruitful exercise since this will enable malware analysts in developing appropriate signatures to thwart the spread of this malware. Furthermore, we believe that this deeper understanding of the inner workings of the malware will also enable one to develop fine-tuned software defense solutions that will protect the user. To this end, we will endeavor to write a functional piece of software to stop this malware from executing within a Win32 environment. Our primary contribution therefore is two-fold: to demonstrate the inner workings of WannaCry so as to facilitate further analysis by malware analysts and to uncover several hidden "features" of this cryptovirus and to further the software defense mechanisms against similar malware in the future by developing a functional piece of defense software.

2. Prior and Related Work

The academic body of work is rich with research articles studying cryptoviral extortion and the associated malware's effects on a computing environment. For instance, Young [7] presented the experimental results obtained by implementing the payload of a cryptovirus on the Microsoft Windows platform. Since most malware, including the WannaCry worm, infect the MS-Windows platform, this bias is reflected in the work done by researchers in the literature (and our present work) as well [9]. Kumar and Kumar [3] present an overview of the modus operandi and the general structure and working of a cryptovirus. A more detailed treatment of an instance of such malware is presented by Filiol and Raynal [2] including a discussion of Code Red, Slammer and the Blaster worms. An example of a large scale cryptoviral extortion is presented in the work done by McCormack [4] which outlines both the financial and data losses incurred due to this attack. A historical perspective on the birth, neglect and explosion of ransomware is presented by Young and Yung [8] and by Salvi and Kerkar [6]. Pascariu et al. [5] presented a process of reducing the attack surface in the case of ransomware attacks. A more recent study conducted by Chen and Bridges [1] introduced a method to identify and rank the most discriminating ransomware features from a set of system logs. In an effort to reveal the presence of the WannaCry malware, they ranked features that reveal a set of actions produced by malware on the system logs thereby automating the process and avoiding tedious manual analysis. Despite this rich body of work, a thorough treatment of a specific instance of this family of malware is non-existent in the literature. A search of research articles on most databases for WannaCry reveals that there is hardly any work done in the academic realm in understanding this malware. It is certain that industry (Norton, Symantec, McAfee and others) must invest substantial time and effort in understanding this malware so as to develop efficient defense mechanisms while the academic realm has not kept pace with the industry. It is this reason that offered us the motivation to delve deeper into understanding WannaCry and conduct advanced static and dynamic analysis of this ransomware. As a final note before concluding the related work survey, the field of malware analysis has made tremendous advances and the techniques and tools used by some of the authors of these above mentioned prior art have been superseded by newer tools. In this sense, our present work also serves to illustrate these newer trends in malware analysis. This survey is not meant to be comprehensive and does not explore all avenues of research in malware analysis (or WannaCry analysis). The interested reader is directed to the above mentioned research articles and the references therein.

3. Analysis

In this section, we begin by describing the vulnerability that WannaCry exploits, followed by its general file structure, cryptographic function calls before diving into static and dynamic analysis.

3.1. WannaCry/WCry

3.1.1 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, we instead shift our focus 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:

ed01ebfbc9eb5bbea545af4d01bf5f107
1661840480439c6e5babe8e080e41aa

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 our 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.

3.1.2 General File Data

Utilizing PEiD[6], we notice that the program was packed using Microsoft Visual Studio C++ 6.0 for Win32 and we should hence have no trouble unpacking it. Utilizing Dependency Walker[7], we see that the program uses *ADVAPI32.DLL* which is the source of many cryptographic security functions implemented in Windows as shown in Fig. 1. In fact, this is where the SMB exploit *EternalBlue* is found. It is interesting to note however that had we not known ahead of time that the program uses *ADVAPI32.DLL*, we wouldn't have thought to gather this piece of information during our examination until we go further through the decompilation and execution steps. As it stands, most if not all ransomware will want to utilize this dynamic linked library if it intends to perform any non-trivial cryptographic operations, such as encryption/decryption of files. The inclusion of this library is the first indication that this software may be ransomware.

Looking into the functionality imported from this library, we see that *CRYPT32.DLL*, and the imports immediately indicate this program is performing a large number of cryptographic function calls as shown in Table 1. Of particular interest are the functions *CryptGenKey*, and *CryptEnrcypt* which we believe (based on rudimentary static analysis) are responsible for generating a random encryption key, and the actual encryption operation. Here, we

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/

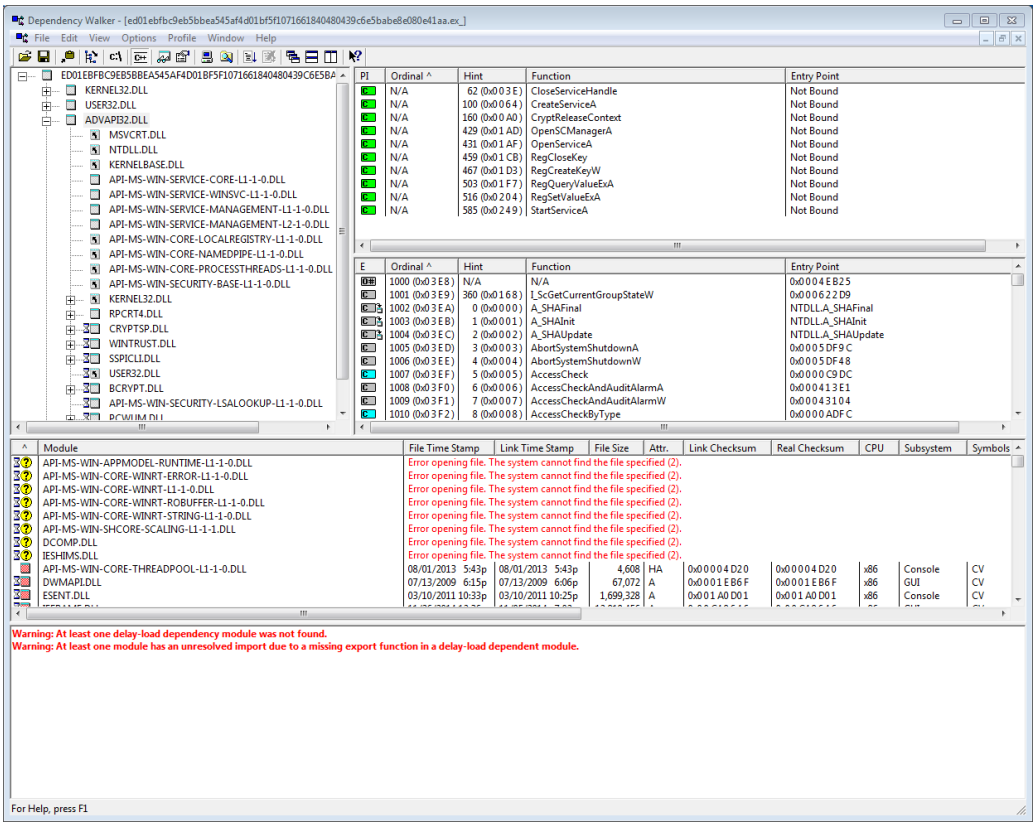


Fig. 1: Dependency Walker overview of WannaCry

would like to make a minor comment about the tools we have used in our work such as PEiD and Dependency Walker: many of these tools have gained prominence among malware analysts as robust and sound tools to conduct forensic/malware analysis. It is with this knowledge that we find confidence in employing them for our research as well.

We can also see that Wannacry 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 (Graphical User Interfaces). We also see *MSVCRT.DLL* being used, which again irrefutably indicates that it was packed with Microsoft Visual Studio.

Utilizing PEview[8] (Fig. 2) 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 sophisticated packer was not used and the binary was generated by a well-behaved compiler. Hence, there will likely not be dynamic unpacking within memory during execution. So, a full memory dump would not greatly benefit the analysis in this instance.

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

Examination with Resource Hacker[9] indicates that the software is trying to masquerade as *diskpart.exe* as shown in Fig. 3, but offers no other useful information otherwise because the WannaCry files are encrypted in the "WNcry@2017" archive and are password protected, as we will show in the IDA Pro analysis portion of our work.

3.1.3 Decompilation

Before delving into the binary code listing for WannaCry, a quick look at the "Strings" window confirms this is some variant of WannaCry or a program pretending to be one. Utilizing the IDA call flow graph[10], we see that in addition to the basic startup functionality, the program also calls _WinMain@16. From our analysis, we can infer that the rest of the function calls cascade from this call to _WinMain@16 since this is where the archive gets unpacked. The software utilizes try/catch/finally blocks and extensive memory manipulation such as allocating heap memory we believe in an effort to protect itself from being erased from RAM by another process, to accomplish its task. However in many cases, we found that the creator(s) seemingly failed to turn on optimization in their compiler as shown in Fig. 4.

Additionally, after examining the try/catch/finally blocks, we 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-coded for opening and manipulation (Fig. 5). In some cases, the functionality of these files isn't fully known until completion of advanced dynamic analysis and related procedures. Some work has been done by anti-virus vendors, namely Symantec[11] in an effort to identify the file names and their descriptions. We do not delve any deeper into this detail here since this analysis is still in its infancy and much more work needs to be done to comprehensively uncover the roles that each of these files play in enabling the ransomware. In Table 2, we outline some important files along with their descriptions.

Moving into the software work flow, the program enters the main method, _WinMain@16, and extracts a password protected archive named "WNcry@2017" 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.[12] 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*. Based on our analysis, we have found that it currently attacks 151 different file types, which can be found in the decompilation, but this count seems to vary among the different variants

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

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

11. https://www.symantec.com/connect/blogs/what-you-need-know-about-wannacry-ransomware

12. After searching online, we discovered that these processes were specifically related to database and mail servers.

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	

TABLE 1: Cryptographic Function Calls.

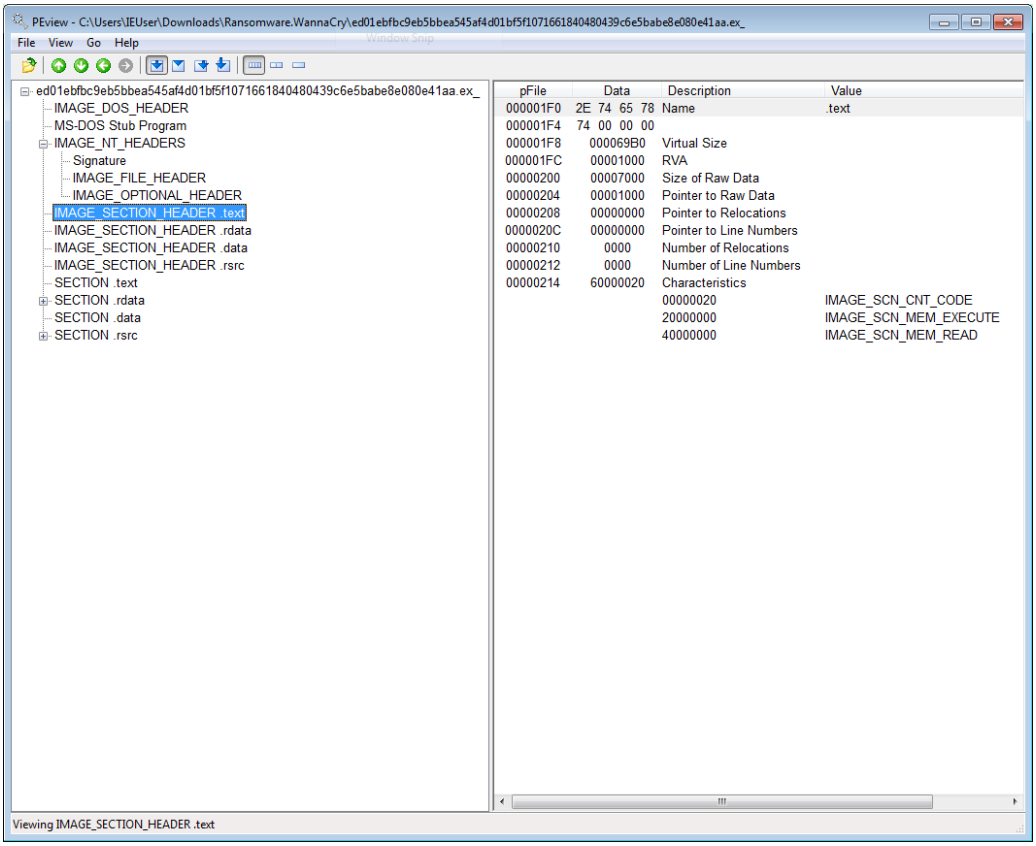


Fig. 2: PE headers for WannaCry

of the ransomware. 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 utilizes a TOR client for communication with the attackers to verify payment, which is retrieved 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 we have analyzed does not include this kill-switch. It seems to be the only difference between the 'original' and the version that we are examining.

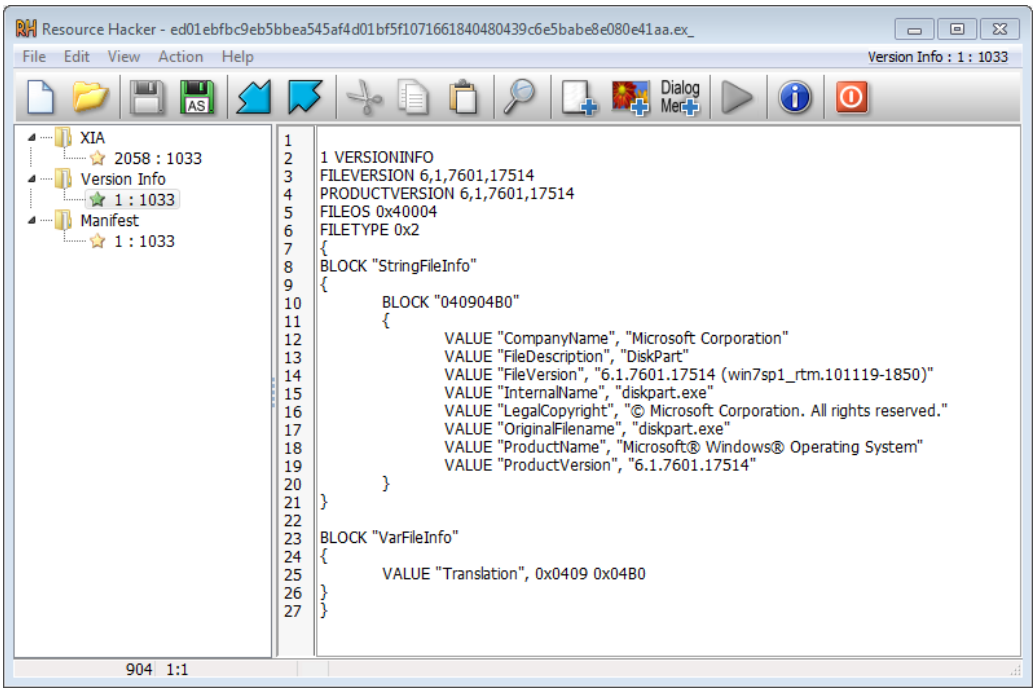


Fig. 3: Resource Hacker & WannaCry

Filenames and 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

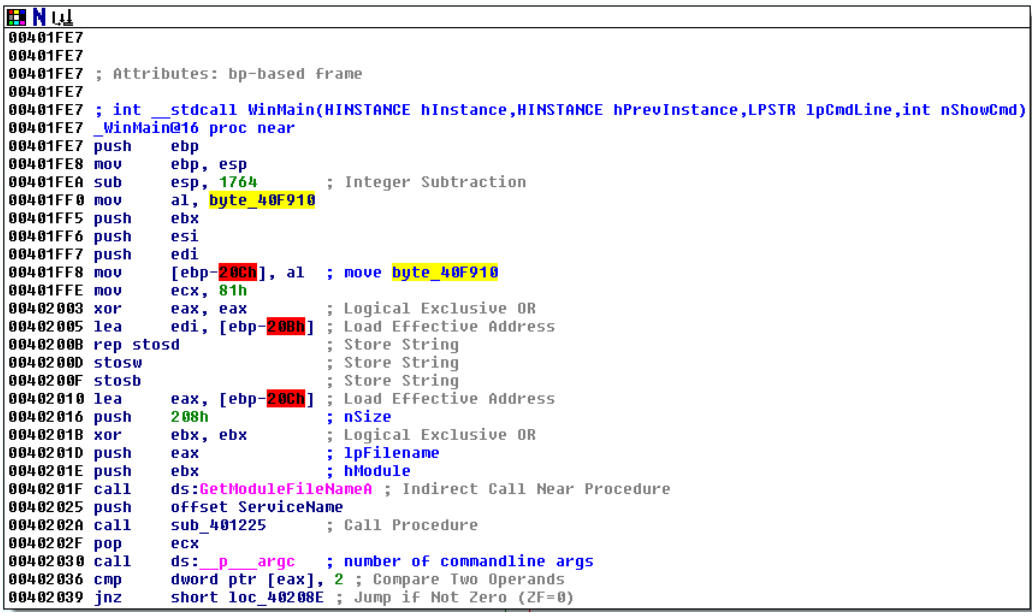
TABLE 2: Filenames and Descriptions.

3.1.4 Dynamic Analysis

First, we launch the malware directly from the location /∼/Downloads/Ransomware.WannaCry/. Upon initial launch (Fig. 6), the wallpaper is immediately changed, and we 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 event outbreak and most likely therefore accurate. Given that the last modified date is equivalent, some of the metadata was likely lost, obscured or otherwise modified at some point of the process.

After the payload is extracted and execution begins, a UAC[13] (User Account Control (UAC) is a technology and security infrastructure introduced with Microsoft's Windows Vista and Windows Server 2008 operating systems, with a more relaxed version also present in Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012 and Windows 10) 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.

13. http://bit.ly/2z3IqkE



```
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-208h] ; 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)
```

Fig. 4: Poor Optimization

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.

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 completely taking over the system. We were unable to find any saved record of these changes for the program. So, it doesn't seem plausible that the host is actually completely restorable upon payment, as the malware 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 prior to infection. Prior to infection, the network capture showed no traffic and no network applications active, as expected. 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.

3.1.5 Putting it all Together

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 esti-

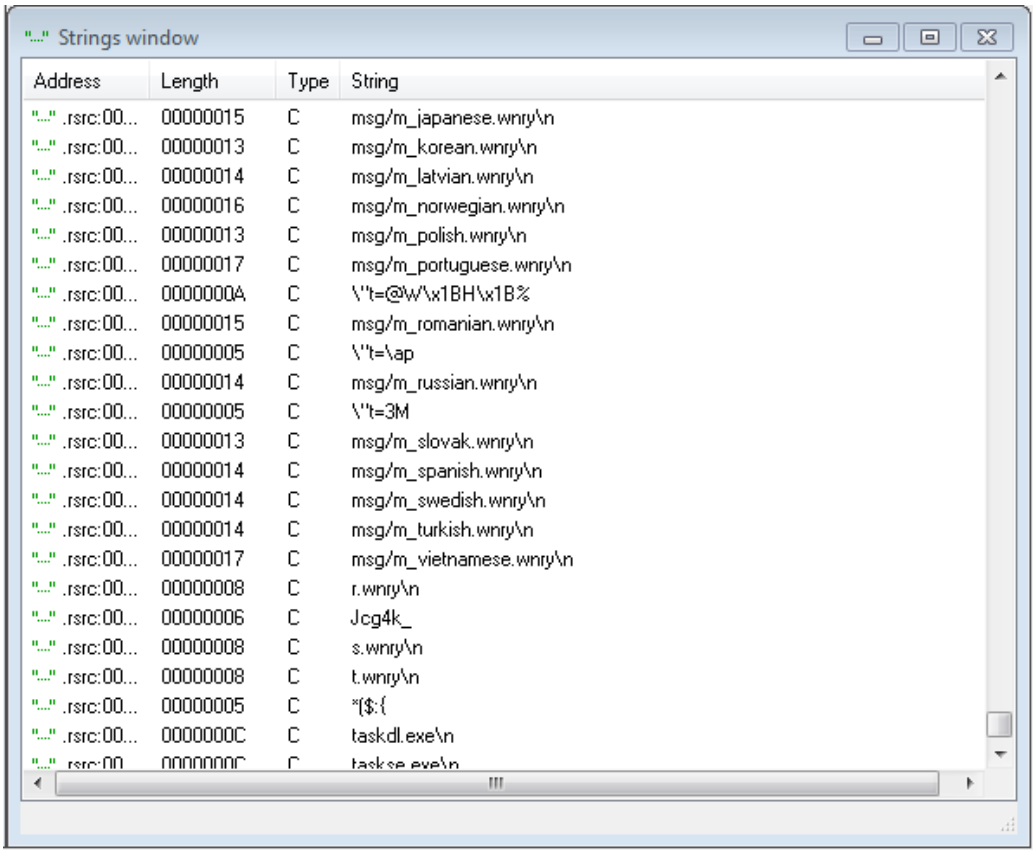


Fig. 5: *.wnry filenames

mated 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. We additionally tested a program called *Wanakiwi*[14] 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 operating system in any case and check all devices on the network, including network-attached storage.

14. https://github.com/gentilkiwi/wanakiwi

4. The Defense Software

In this section, we outline our ideas and detail our process of implementing our defense solution.

4.1. An Overview

Our goal behind developing a defense program was based around the desire to highlight the implementation details of the Windows NT environment (and vulnerabilities) and general anti-virus software development processes. The general operation of the software can be seen below in Algorithm 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

Fig. 6: Initial Infection

by any processes that chooses 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. We considered any process with more than 5 references to be unsafe, although realistically one could deny launch for even one Virus Total reference.

The first hurdle with developing this software was creating a kernel driver. Using the details provided in the book *Windows Internals, 6th ed.*[15] and also following a tutorial on CodeProject.com[16], and finally studying a Microsoft-supplied example driver found on their Windows Example drivers GitHub repository[17], we were able to develop a kernel driver. 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, we learned much of the Windows NT API to handle events and perform actions around those events.

We originally intended to fully write the kernel driver ourself; it soon became clear that without an expert knowledge of Windows API and programming experience, it would be incredibly challenging. The next best option would be to adapt a currently existing open-source callback driver for our purposes. To this end, we initially attempted to use a provided Microsoft example implementation,

15. https://docs.microsoft.com/en-us/sysinternals/learn/windows-internals

16. http://bit.ly/2lxXIv5

17. https://github.com/Microsoft/Windows-driver-samples

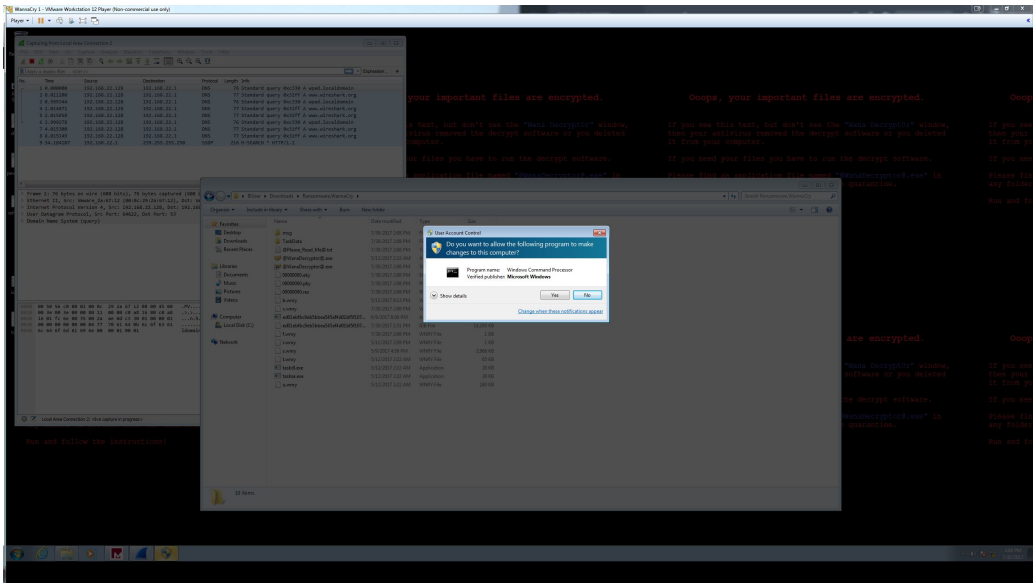


Fig. 7: UAC Prompt on first run

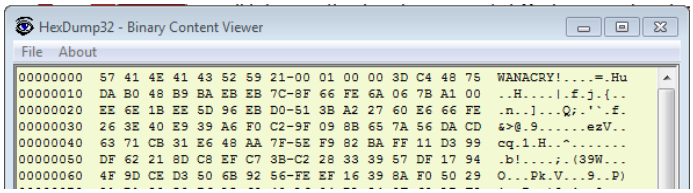


Fig. 8: Modified File Headers

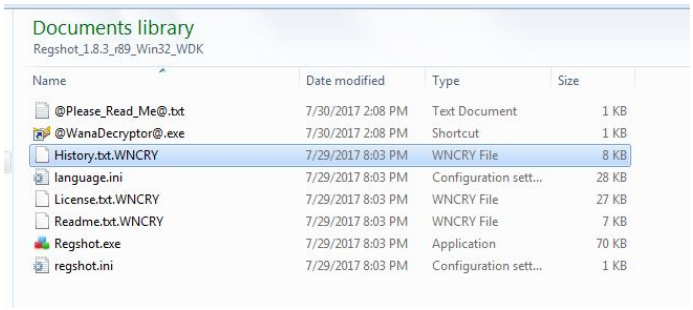


Fig. 9: An example directory after infection

however the original intentions for the program were very different from what we were intending to use it for and adaptation was difficult. We used this code base as our development foundation for a few weeks, however we eventually switched to the implementation provided by Ivo Ivanov on CodeProject.com, referenced previously in this section. Within this implementation we removed the test instantiations, and added sqlite3 functionality for the database and the VirusTotal API as our 'engine'. For actually sending off tests to VirusTotal, we elected to compute the SHA-256 digest of the executable to be launched using OpenSSL/libcrypto after retrieving the process path from the process

Algorithm 1: A generalized algorithm

```
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
```

handle. Section 7.3 in the Appendix contains a list of the 3rd party APIs that we used for this software. Unfortunately at the time of this writing, our software needs to be stress tested against these third party APIs and is therefore not as robust as we would like it to be. This work is therefore, currently a work in progress. The major portion of the challenges are centered around a few of the third party APIs that we have used in our implementation and their dependencies and we have not been able to test the program functionality completely.

5. Conclusion

Our work, to the best of our knowledge, is the first to demonstrate the inner workings of the infamous ransomware/cryptoviral extortion malware *WannaCry*. We have used advanced static and dynamic analysis techniques using state-of-the-art tools. Our hope in this research project was to illustrate the devious nature of WannaCry in an effort to prevent future attacks of this nature and to arm the user with sufficient knowledge and information to combat such an attack. Our tool, while still in its infancy, is a first approach to defend oneself against an attack by such malware.

6. Acknowledgments

The authors would like to thank the support offered by The Center for Enhancing Undergraduate Research Experiences and Creative Activities at Sam Houston State University. The authors are solely responsible for the views expressed in this paper, which do not necessarily reflect the position of the supporting organization.

References

[1] Chen, Q. and Bridges, R. A. (2017). Automated behavioral analysis of malware a case study of wannacry ransomware. *arXiv preprint arXiv:1709.08753*.

[2] Filiol, É. and Raynal, F. (2008). Malicious cryptography... reloaded. In *CanSecWest Conference, Vancouver, Canada.[online] http://cansecwest. com/csw08/csw08-raynal. pdf*.

[3] Kumar, S. M. and Kumar, M. R. (2013). Cryptoviral extortion: A virus based approach. *International Journal of Computer Trends and Technology (IJCTT)*, 4(5):1149–1153.

[4] McCormack, M. (1996). Europe hit by cryptoviral extortion. *Computer Fraud & Security*, 6(1996):3.

[5] Pascariu, C., BARBU, I.-D., and Bacivarov, I. C. (2017). Investigative analysis and technical overview of ransomware based attacks. case study: Wannacry. *Int'l J. Info. Sec. & Cybercrime*, 6:57.

[6] Salvi, M. H. U. and Kerkar, M. R. V. (2016). Ransomware: A cyber extortion. *Asian Journal of Convergence in Technology*, 2(2).

[7] Young, A. L. (2006). Cryptoviral extortion using microsoft's crypto api. *International Journal of Information Security*, 5(2):67–76.

[8] Young, A. L. and Yung, M. (2017). Cryptovirology: The birth, neglect, and explosion of ransomware. *Communications of the ACM*, 60(7):24–26.

[9] Young, A. L. and Yung, M. M. (2005). An implementation of cryptoviral extortion using microsoft's crypto api.

7. Appendix A

7.1. 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

7.2. Database Diagram

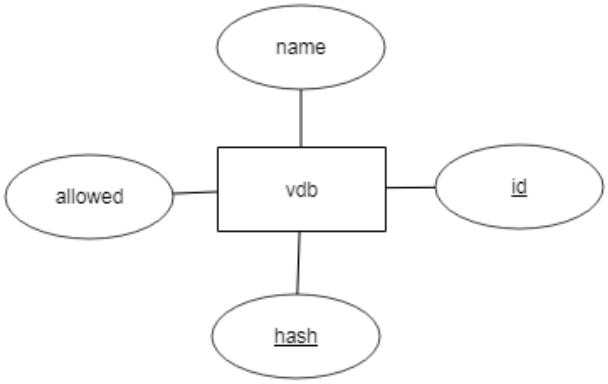


Fig. 10: VDB Database Diagram

Fig. 10 illustrate the database diagram we have employed.

7.3. 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/