Ransomware Analysis and Defense

WannaCry, SambaCry, and the Win32 environment

Justin Jones

July 30, 2017

Contents

1 Introduction 1

2 Analysis 1
2.1 WannaCry/WCry 1
2.1.1 Background 1
2.1.2 General File Data 2
2.1.3 Decompilation 4

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 all functionality and data. While initially not incredibly sophisticated, this genre, if you will, 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. I will further explore a piece of malware written for Linux operating systems that behaves in a similar fashion, known as *SambaCry*, and make comparisons. Finally, I will endeavor to write a useable piece of software to stop malware from launching 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

[1] http://bit.ly/2spdT15

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].

2.1.2 General File Data

Utilizing PEiD[5], 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[6], 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 had I not already known.

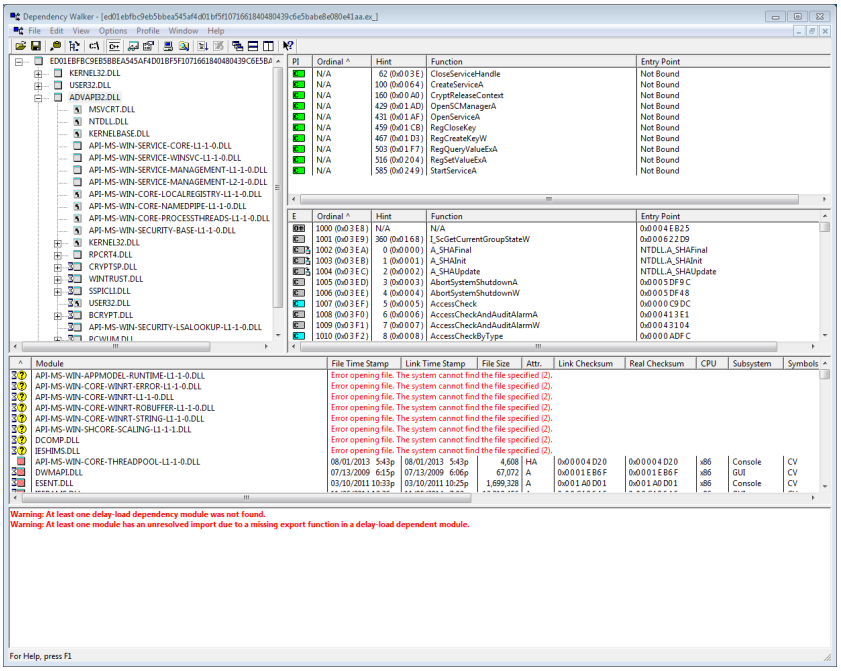

Figure 1: Dependency Walker overview of WannaCry

Looking into the functionality imported from this library, we see *CRYPT32.DLL*, and the

[2]https://support.microsoft.com/en-us/help/4013389/title
[3]http://thezoo.morirt.com/
[4]http://bit.ly/2s93pCl
[5]https://www.aldeid.com/wiki/PEiD
[6]http://www.dependencywalker.com/

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[7] 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 packer was likely not used and the program was likely generated directly by a compiler.

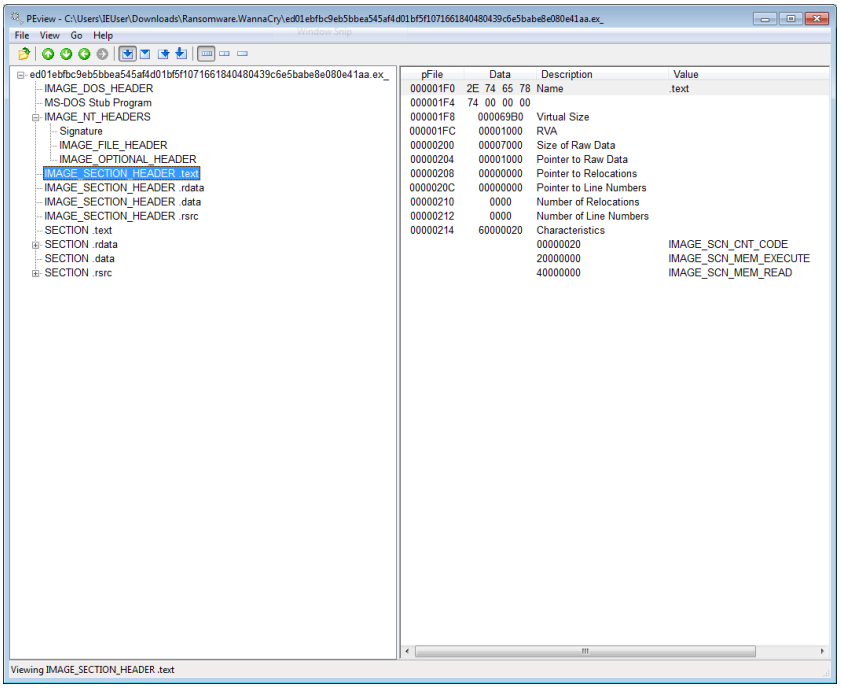

Figure 2: PE headers for WannaCry

Examination with Resource Hacker[8] indicates that the software is trying to mascarade as diskpart.exe, and offers no useful information otherwise.

[7] http://wjradburn.com/software/
[8] http://angusj.com/resourcehacker/

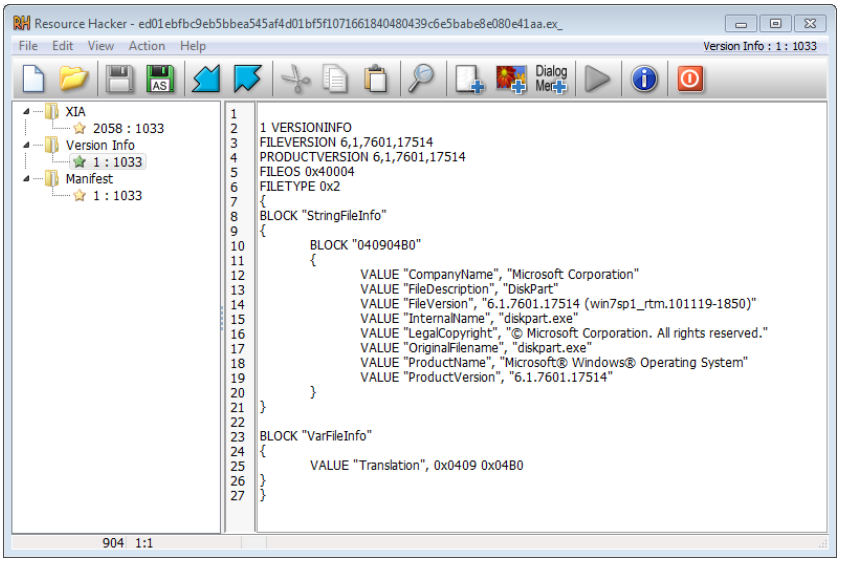

Figure 3: Resource Hacker & WannaCry

2.1.3 Decompilation

Utilizing the IDA call flow graph, it can be seen that in addition to the basic startup functionality the program also calls _WinMain@16, form which the rest of the function calls will cascade from.