MALWARE ANALYSIS TOOLS & TECHNIQUE

Year 2 (2020/21) Semester 4

School of Infocomm Technology

Diploma in Cyber Security and Digital Forensics

ASSIGNMENT

	Malicious Executable	***Malicious Document***
Type	Win32 Executable	Microsoft XML Document
Filename	Win32.WannaPeace.zip	
MD5 Hash	eefa6f98681d78b63f15d7e58934c6cc	
URL Download	https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Win32.WannaPeace	

Student Name	Student ID
Ezra Ho Jincheng	S10194982A
Matthias Gan	S10197146D

Contents

1. Lab Setup

As the process of analyzing a malware is dangerous and might result in potential unwanted results, we have decided to setup a lab to be used to analyze the malwares in a safe environment. A virtual machine is used for the analysis of the malwares to ensure that any unwanted consequences can be reverted and that the malwares will have no interaction with the host machine.

1.1. Virtual MachineSetup

This section is used to describe the setup of out machines used to conduct the malware analysis. It includes the information regarding the specifications of the virtual machine systems, the host operating system as well as the guest operating system.

1.1.1. Virtual MachineSoftware

VMWare Workstation Pro by VMWare is chosen as the preferred virtual machine software since it is a very powerful software that allows users to change settings of the virtual machine.

1.1.1.1. VMware Workstation Pro

- Virtual Machine Software: **VMware® Workstation 16 Pro**
- Version: **16.1.0**
- Build Number: **17198959**
- Installation Link: https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html

1.1.1.2. Host Operating System

- Host System:
  - Operating System: **Microsoft Windows 10 Home**
  - Processor: **AMD Ryzen 5 2600 Six-Core Processor 3.40 GHz**
  - Memory: **32GB**
  - System Type: **64-bit Operating System, x64-based processor**
  - Version: **20H2**
  - OS Build: **19042.630**

The host machine is the machine that is used to download the virtual machines. It is also used to run the virtual machines. All malware analysis is conducted on the virtual machine, and the host machine will not have any contact with the malware. This ensures the safety of the host machine, preventing it from getting infected by the malware.

1.1.2. Guest Operating Systems

- Guest System:
  - Operating System: **Microsoft Windows 8.1**
  - Processor: **AMD Ryzen 5 2600 Six-Core Processor 3.39 GHz**
  - Memory: **2GB**
  - System Type: **32-bit Operating System, x64-based processor**
  - Version: **6.3**
  - OS Build: **9600**

This Windows 8 virtual machine is used for the basic static and dynamic analysis of the malware. All malware analysis will be done on this machine. It already has all the required static and dynamic analysis tools installed. Initial static analysis will be done on the malware. The malware will then be run on this machine; This machine has been setup and have no access to the network or the host machine. Multiple snapshots has been made to allow us to easily restore to a clean version of the operating system.

1.2. Network Diagram

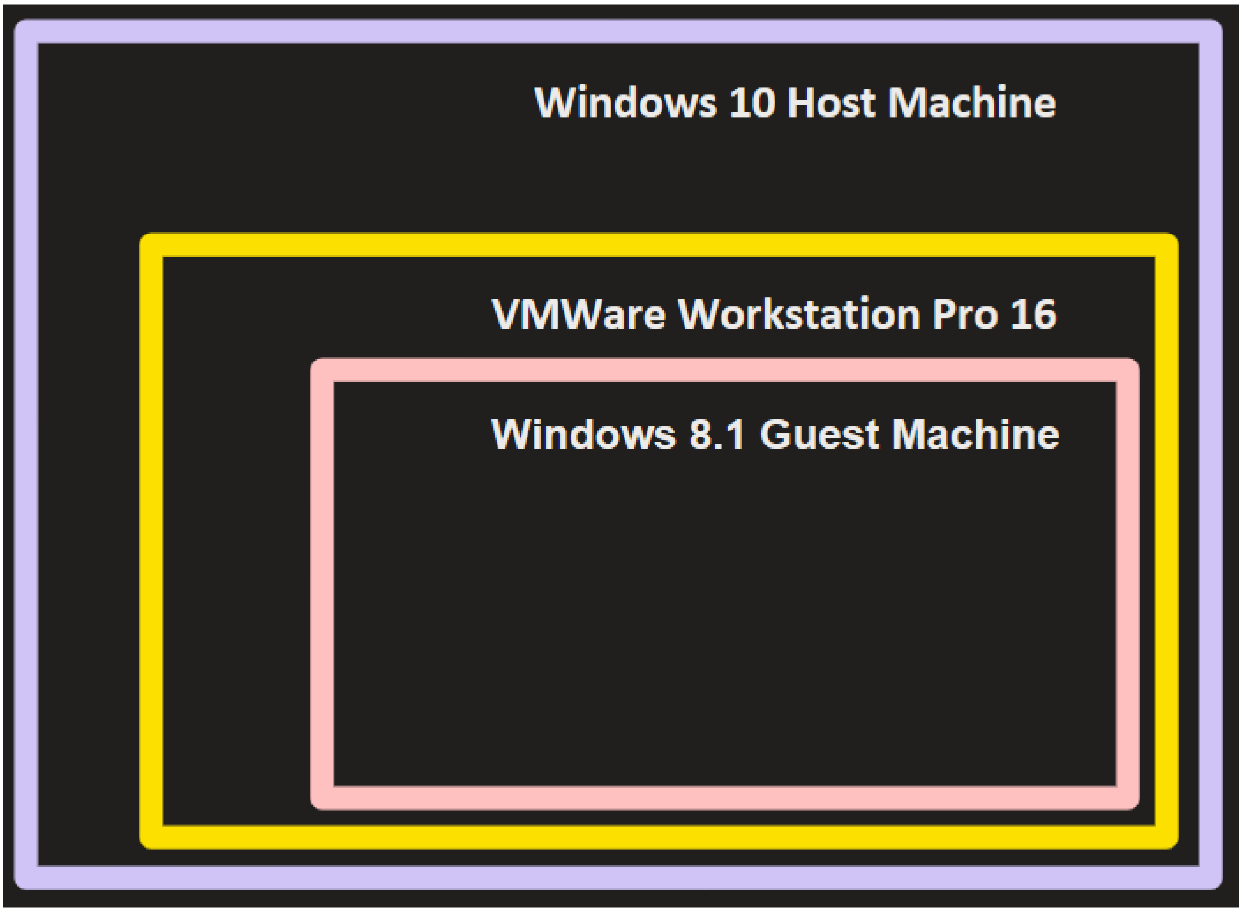


Figure 1.2 – Network Setup Diagram

The virtual machine is configured as shown by the image above. It has been configured to the Host-only mode in VMWare Workstation Pro 16. This would ensure that the virtual machines will not have access to the internet, any other networks or even the host machine. The act of isolating network access from the virtual machine would ensure that the malware is contained and would not have the capabilities of spreading.

1.3. Network Configuration

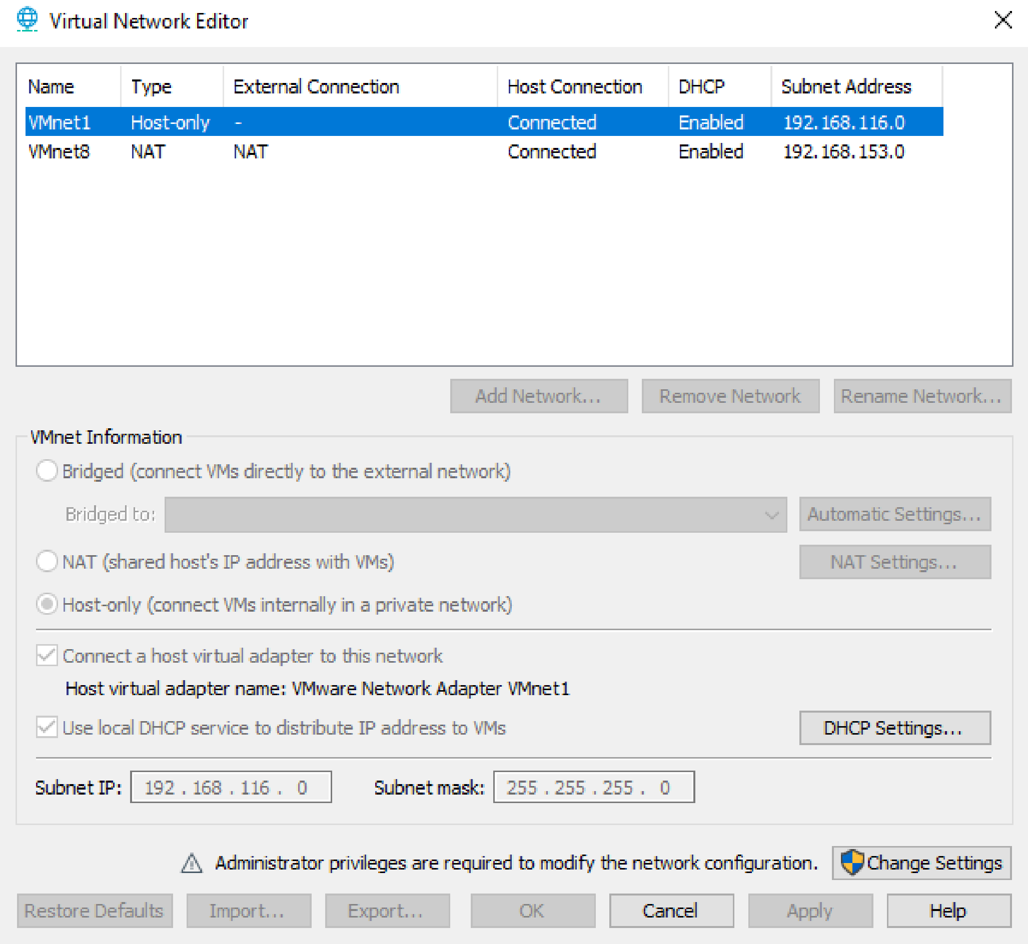


Figure 1.3.1 – VMware Workstation Pro 16 network settings

As stated, the guest virtual machine will be utilizing the Host-only mode. It will be connected to VMnet1, which is the Host-only network. The guest machine will receive a local IP address from VMware Workstation Pro's DHCP server. However, this local IP address will not allow it to connect to the internet or any foreign network. The configuration is as shown above.

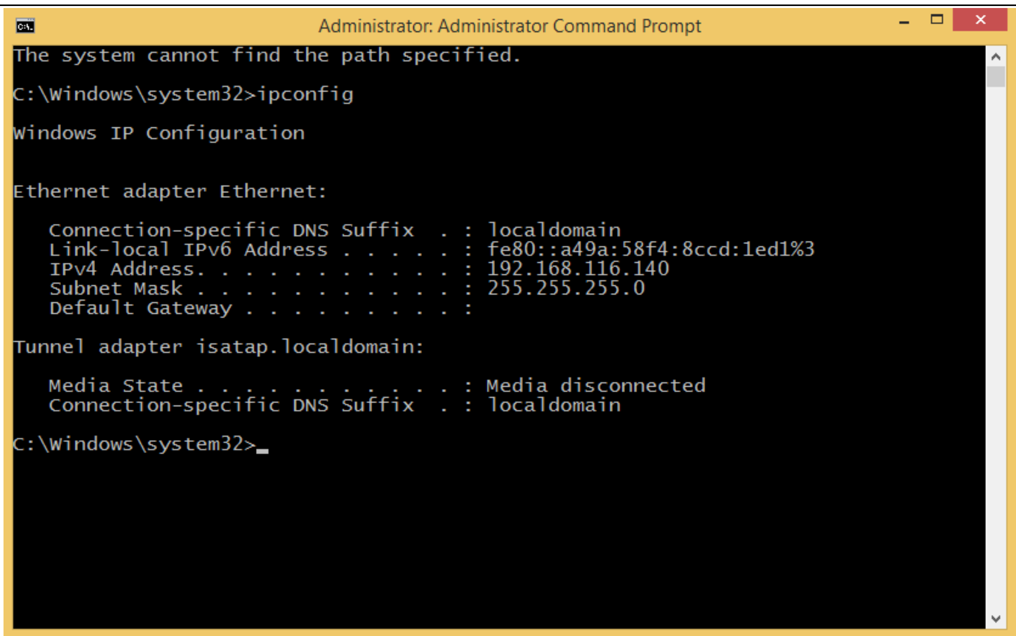


Figure 1.3.2 – Network Configuration of Virtual Machine

The figure above shows the network configuration of the guest virtual machine. It is assigned an IP address of 192.168.75.127 with the subnet mask of 255.255.255.0. The default gateway points to the IP address of VMware Workstation Pro's DHCP server.

2. Malware Analysis Tools

Various tools were utilized to analyze this malware. This includes tools for both static and dynamic analysis. This section will outline the different tools used for the analysis of the malware.

2.1. Basic StaticAnalysis

Basic Static is defined as the examination and analysis of the malware without the malware being executed. The information about the malware is gathered through tools that can extract and obtain the Portable Executable (PE) headers, the strings, and functions imported and exported by the malware. Indications of suspicious information such as functions or strings could be used by basic static analysis, which will give analyst the sense of the functionality of the malware. Basic static analysis is performed by examining the host-based indicators and network-based indicators. Host-base indicators such as imported functions, which are used to modify file and registry contents or to create new processes. Network-based indicators such as IP-addresses, external URLs or functions used to initiate connections and uncommon ports will help to identify if the malware is attempting to make a connection with an external network.

Though basic static analysis is a great way to perform initial analysis on the malware. It could often be insufficient, since more complex malwares may not reveal the full functionality. Thus, methods such as Basic Dynamic Analysis are required to reveal more information to the analyst.

2.1.1. PEiD

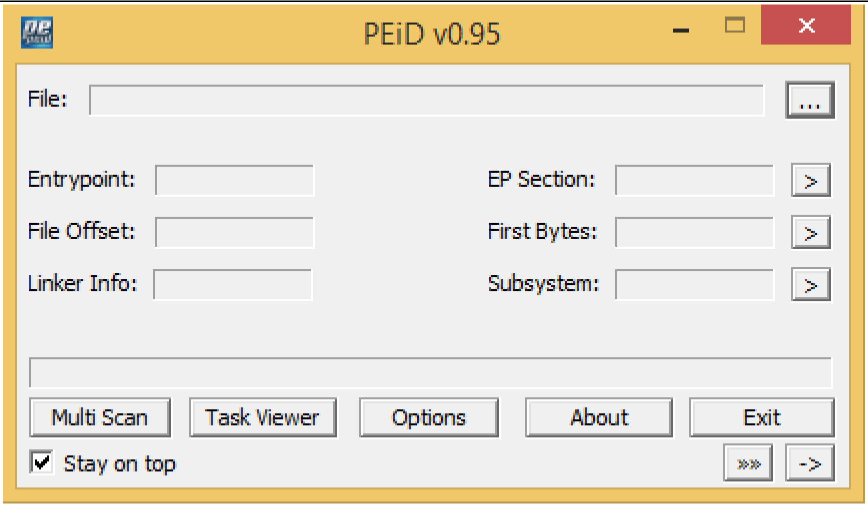


Figure 2.1.1.1 – PEiD user interface

This image above shows the user interface of PEiD. When a malware is being loaded into the tool, all fields will be filled and displays information to the analyst as shown in the figure below.

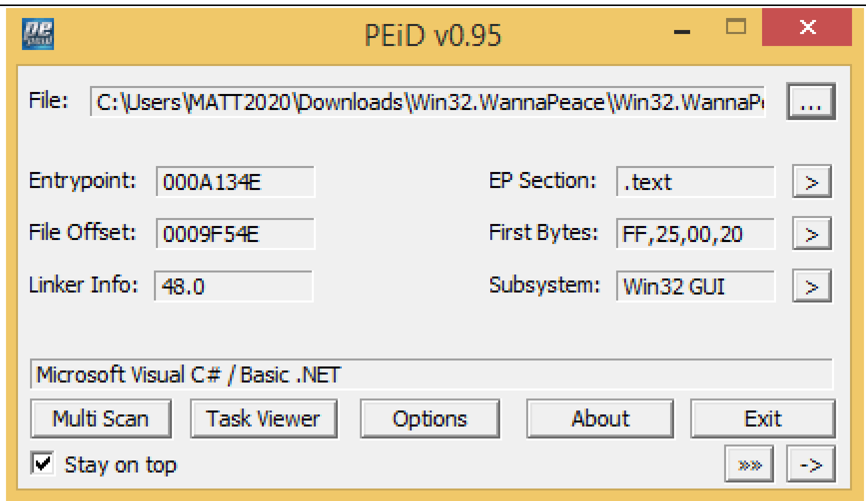


Figure 2.1.1.2 – PEiD user interface with WannaPeace

PEiD is a tool that detects the most common packers, cryptors, and compilers for the PE files. It can currently detect more than 470 different signatures in PE files. PEiD is used to identify and detect malware as some other tools are not able to extract any useful information from the malware while the malware is packed. PEiD can be used to identify if a malware Is packed and which packer was used to pack the malware.

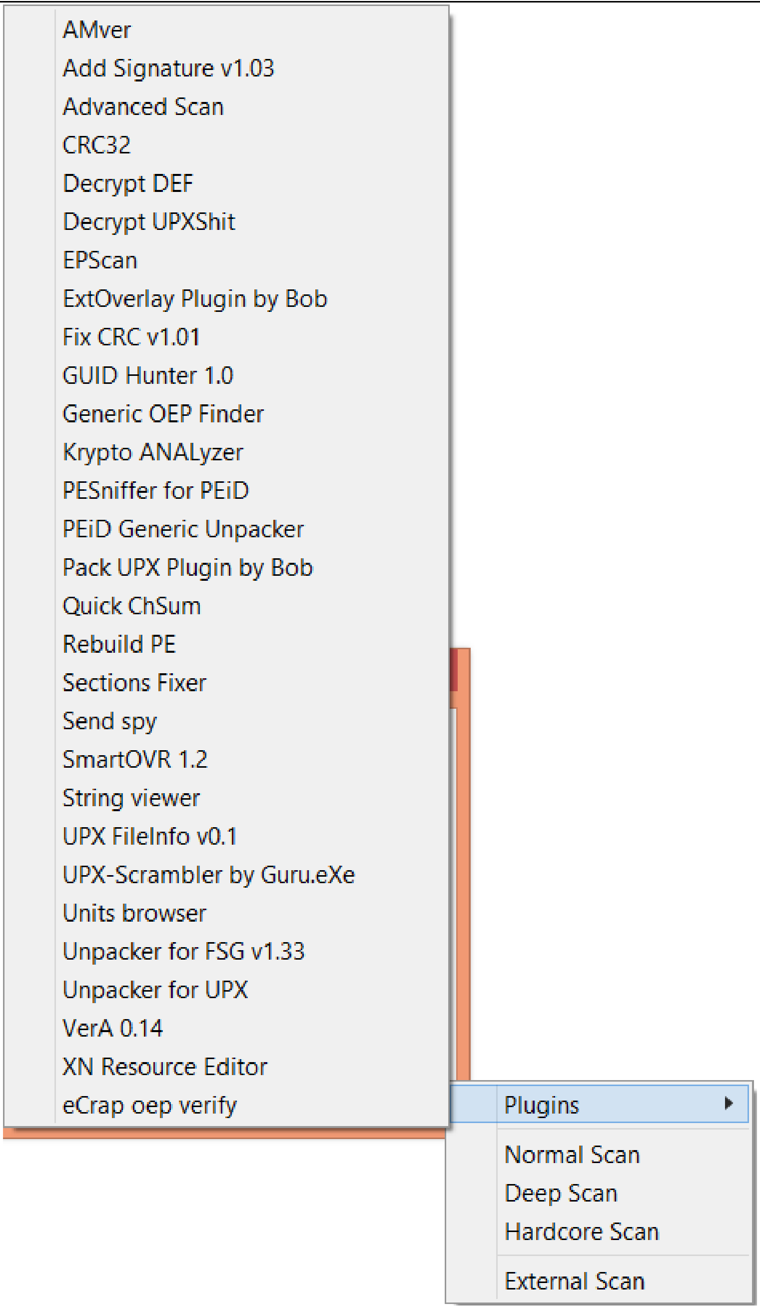


Figure 2.1.1.3 – PEiD's built in plugins

PEiD comes with many built-in plugins which can be used to unpack PE files that were packed with other common packers. Hence, if the malware is packed with a common packer such as UPX (Ultimate Packers for Executables), PEiD would easily pick up the hint that the file was packed and unpack the file.

The official website of PEiD, www.peid.info , has been discontinued for quite some time and is no longer used. Although links to the downloads were discontinued, it can still be used and hosted. There are other links available online which allows the tool to be downloaded from. One such website is: https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

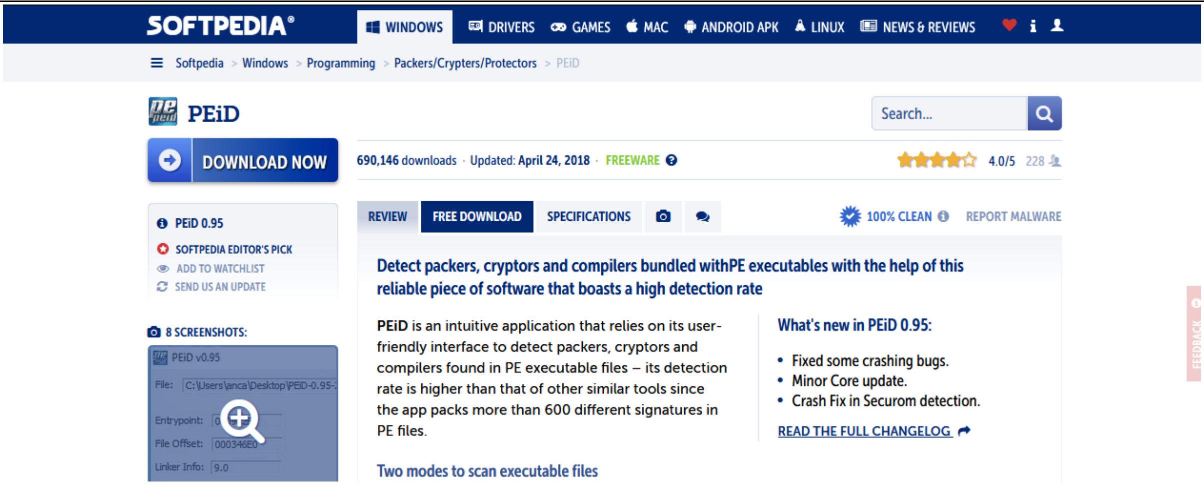


Figure 2.1.1.4 – PEiD's alternate download website

This shows the website where PEiD can be downloaded as the official website has been discontinued.

2.1.2. PEview

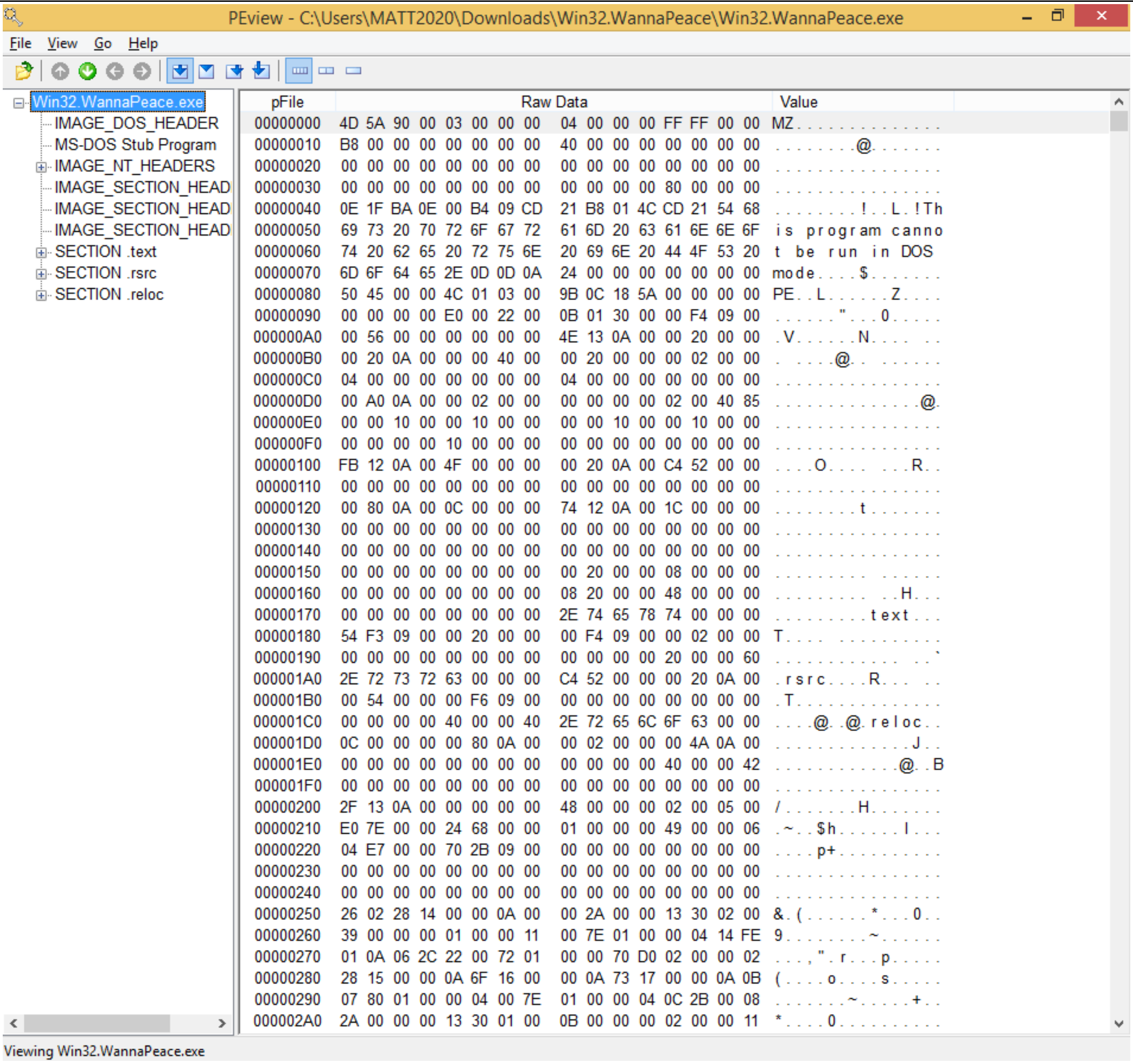


Figure 2.1.2 – PEview's user interface with WannaPeace.exe

PEView is a basic static analysis tool which provides a simple and efficient way to view the structure and content of 32-bit PE files and Component Object File Format (COFF) files. It displays the multiple sections of the files such as header, section, directory, import table, export table, and resource information. Analyst can browse through the structure of the malware and extract information from the file headers and different sections of the file as shown from the image above.

The website to download PEView is still up and running, and can be downloaded from: https://wjradburn.com/software/PEview.zip

2.1.3. Dependency Walker

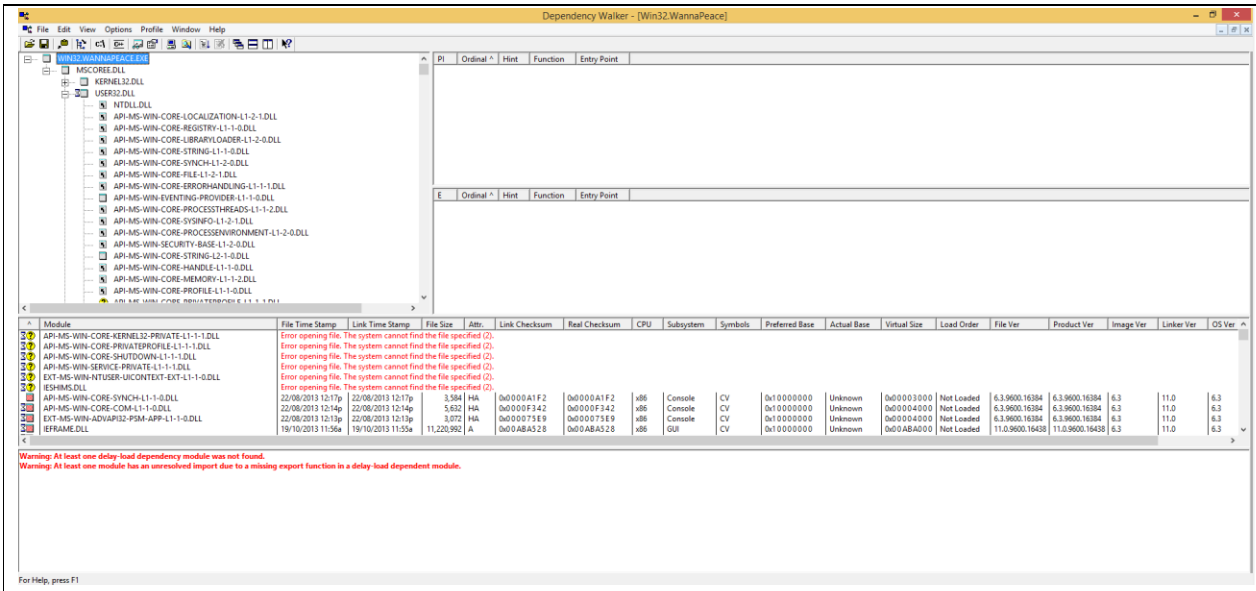


Figure 2.1.3 – Dependency Walker's user interface

Dependency Walker is a basic static analysis tool which is often used in malware analysis to scan 32-bit and 64-bit Windows module. It lists all imported and exported functions of the module. It displays the dependencies of the file along with detailed information about files, which includes file path, the version number, the machine type, and debug information. This hierarchical tree diagram of all dependent modules built by Dependency Walker is useful to identify functions imported by the malware, which then can be used maliciously.

Dependency Walker can still be downloaded from the official website listed: http://www.dependencywalker.com/

2.1.4. Bintext

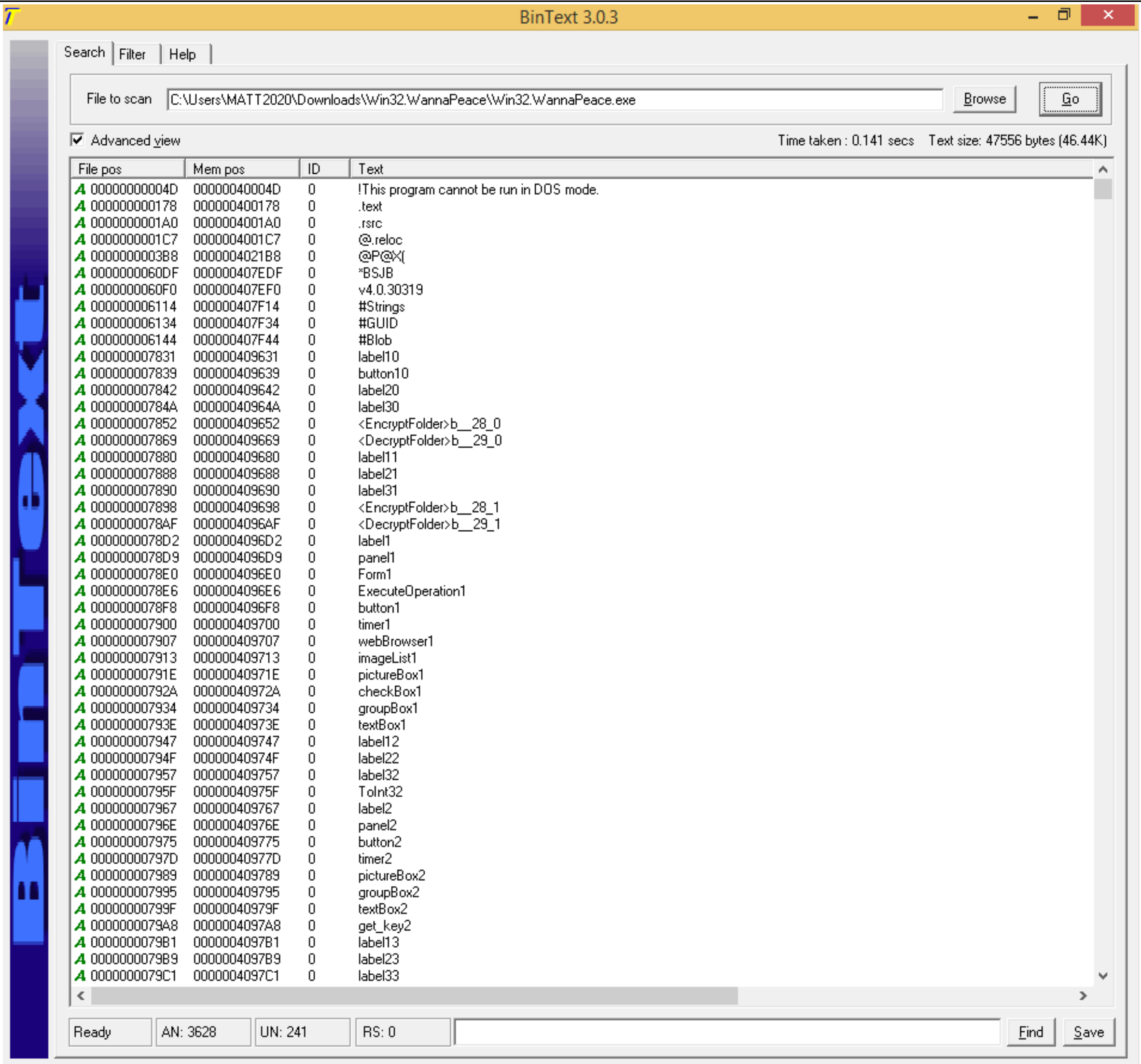


Figure 2.1.4 – Bintext's user interface

Bintext is a small, fast, and yet powerful basic static analysis tool that can find Ascii, Unicode and Resource strings in a file and extracts it from the malware. It is also vert useful for revealing the functionalities of the malware. Bintext can extract host-based and network-based indicators, which will provide the analyst with a basic overview of the malware. The strings that were extracted can be used for further analysis with other tools.

Bintext can be downloaded from the McAfee website as it is the most recently updated version:

[http://b2b- download.mcafee.com/products/tools/foundstone/bintext303.zip](http://b2b-download.mcafee.com/products/tools/foundstone/bintext303.zip)

2.1.5. WinMD5

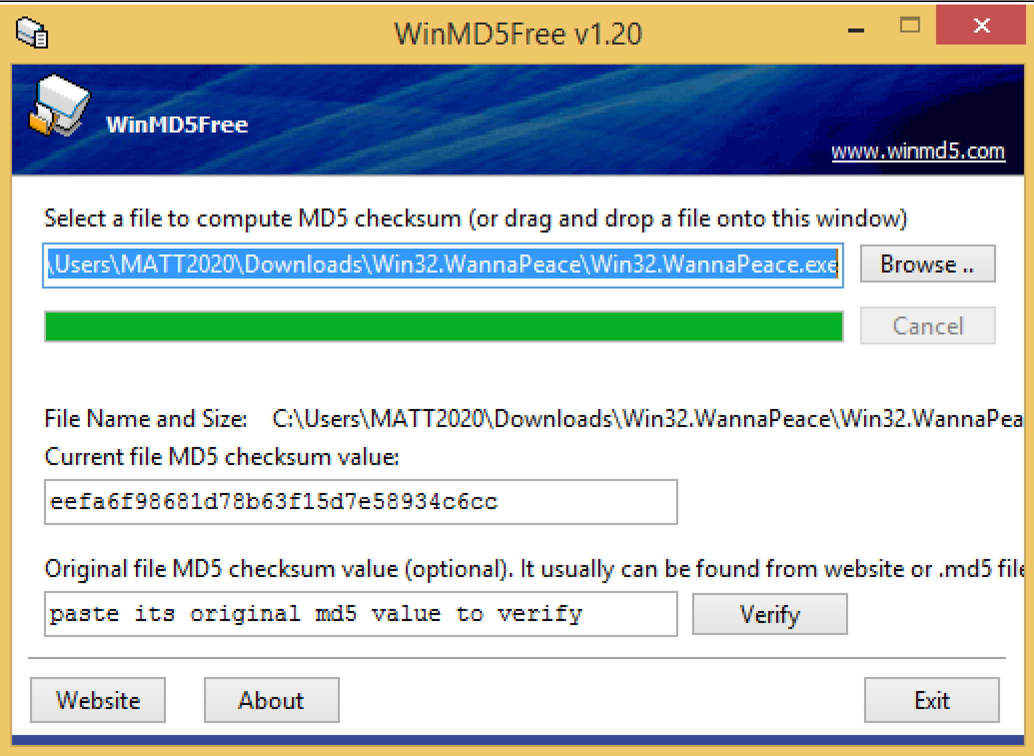


Figure 2.1.5 – WinMD5's user interface

WinMD5 is a small, yet fast utility which is used to compute MD5 has values for files. WinMD5 is often used to obtain the fingerprint of the malware and check the obtained information against anti-virus malware scanners such as VirusTotal. This would check if the malware has been captured on the databased before. If such, analyst may be able to refer to the steps to revert the software to a clean and uninfected state. WinMD5 also has the capability to verify is the malware renamed itself as a separate process by comparing the hash values of the original and modified name of the malware.

WinMD5 can be downloaded from its official website: http://www.winmd5.com/

2.1.6. PowerISO

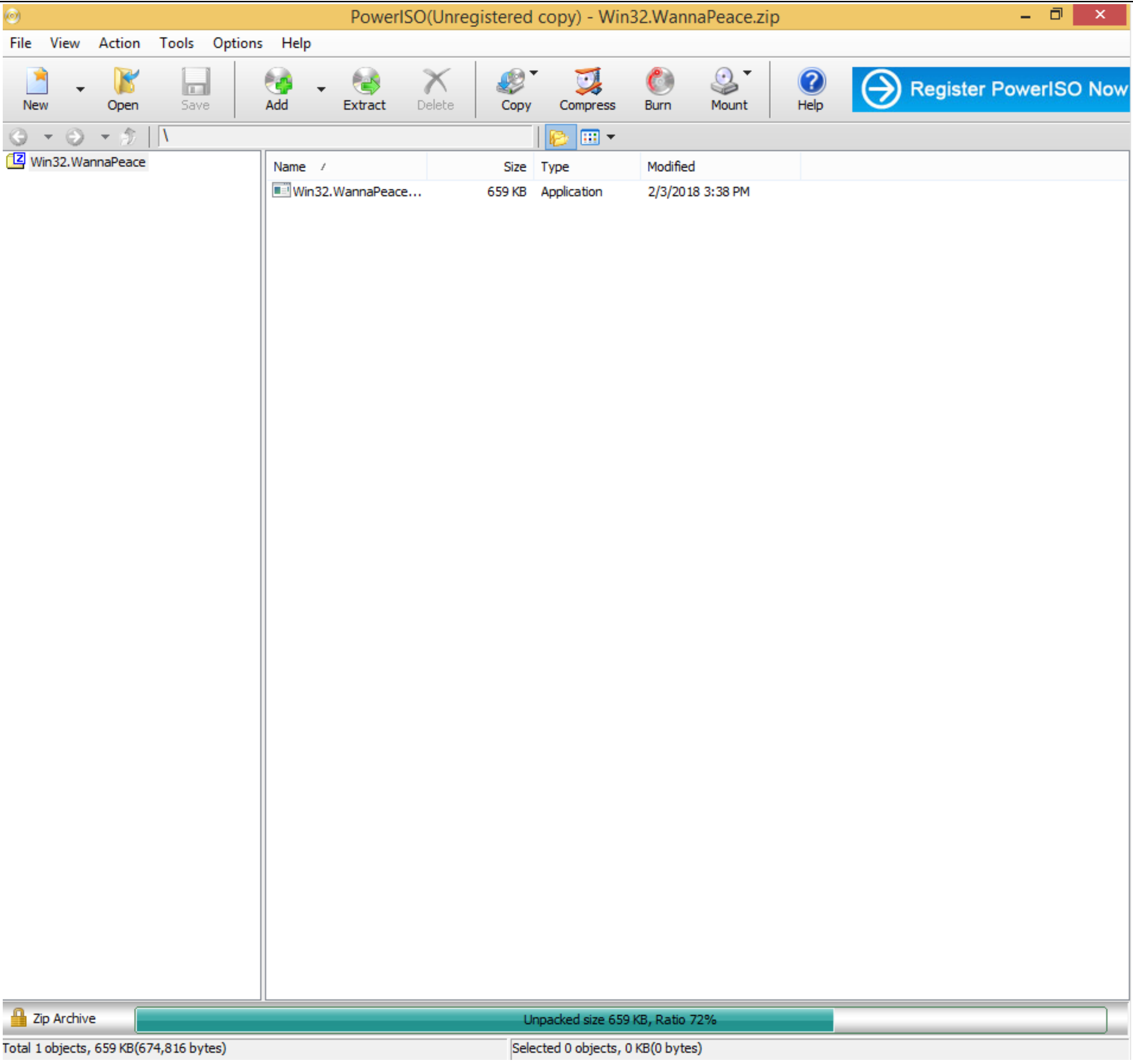


Figure 2.1.6 – PowerISO's User Interface opening WannaPeace.exe

PowerISO is a disk image utility that can be used to decompress or extract several file types such as BIN, DAA, UIF, DMG, MDF and IMG. It also has other features such as Create, Edit, Burn, Mount, and many other functions. But within the assignment scope, PowerISO will be used to help the analyst extract contents from the malicious document, which could append malicious contents to itself.

PowerISO could be downloaded from the official website: https://www.poweriso.com/download.php

2.1.7. PeStudio

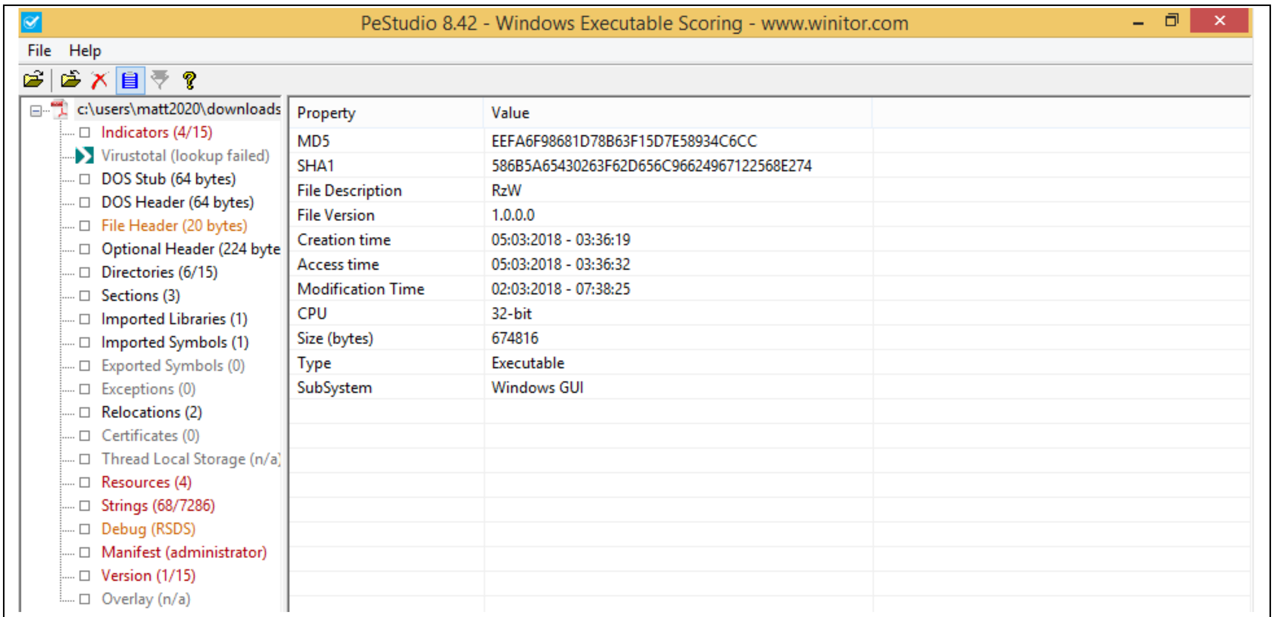


Figure 2.1.7 – PeStudio's user interface loaded with WannaPeace.exe

PeStudio is a tool used worldwide to perform initial malware analysis.

This tool can be used to determine if the file is malicious, based on certain indicators found within the file. Unsuspected metadata, suspicious patterns, and anomalies could be left in a malware to hide their intent, which would all be used as indicators to determine if the file is suspicious. This tool is rather simple to use since analyst are only required to drag the executable file into the PeStudio window within the virtual machine. No risk of infection as the file analyzed is not being run, making it safe to inspect malicious executable files.

PeStudio can be downloaded at TechSpot instead of the official website since the former has a newer version:

https://www.techspot.com/downloads/6350-pestudio.html

2.2. Basic DynamicAnalysis

Basic Dynamic Analysis would help to discover some of the functionalities of the malware. In addition, not all functions or strings extracted during the static analysis is malicious or can be used during run time. Therefore, basic dynamic analysis is also an important part of malware analysis.

Basic Dynamic Analysis refers to using tools to monitor the behavior of the malware, while it is being executed and run on an isolated machine. This allows analyst to discover the actual functionalities of the malware by using monitoring tools to monitor the modifications the malware does or what it does to the environment. It allows analyst to discover additional functionalities previously undiscovered by basic static analysis.

2.2.1. ApateDNS

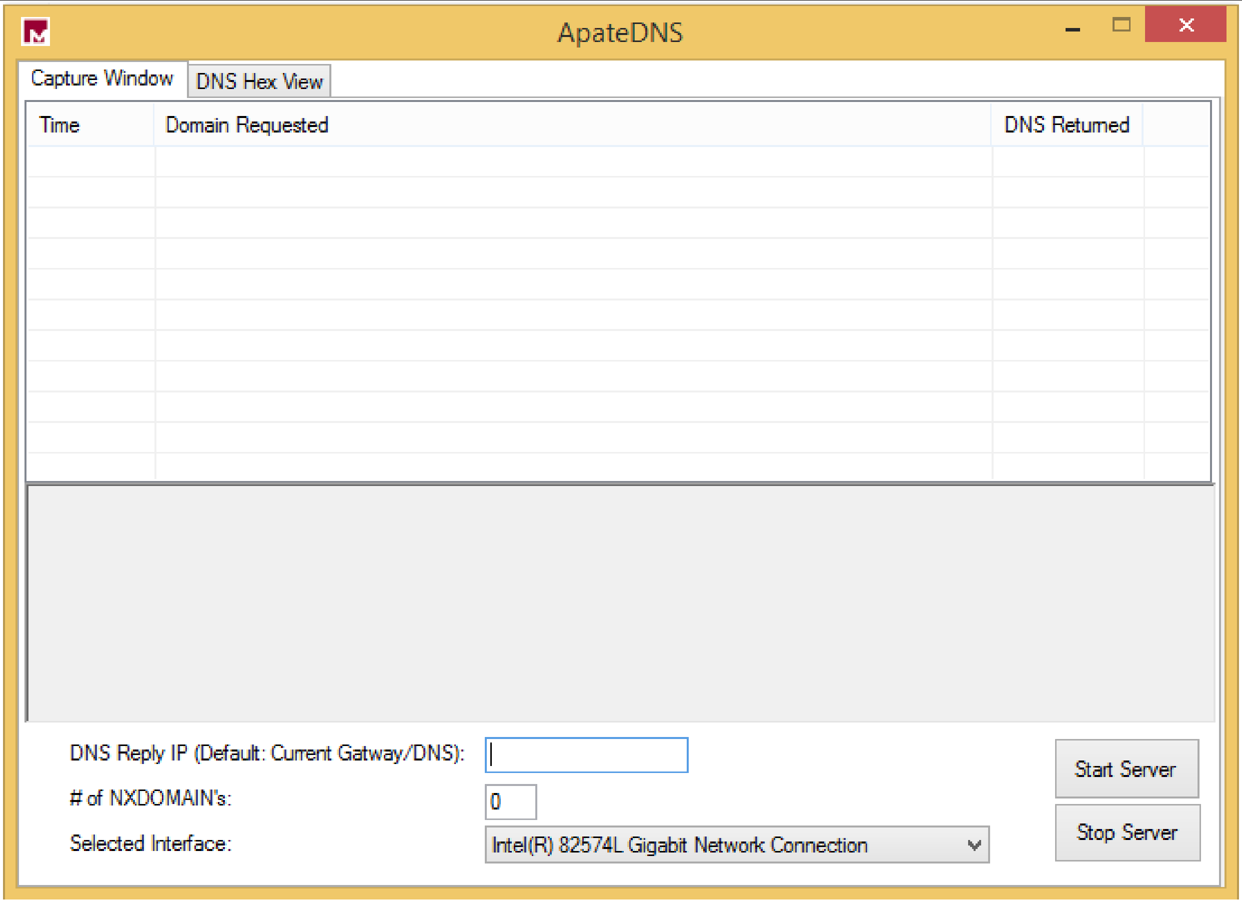


Figure 2.2.1 – ApateDNS's user interface

ApateDNS is a basic dynamic analysis tool for controlling DNS responses. It acts as a DNS server on the local system and could reply to any DNS queries made by the system. ApateDNS spoofs DNS responses to DNS requests generated by the malware to a specified domain name or IP address located at the bottom of the user interface using UDP port 53. It can also verify or discover additional IP addresses or hostnames that could not be found during basic static analysis. It is an extremely useful tool for analyzing whether the malware can connect to any network.

ApateDNS is free to download from FireEye through their website: https://www.fireeye.com/services/freeware/apatedns.html

2.2.2. Process Explorer

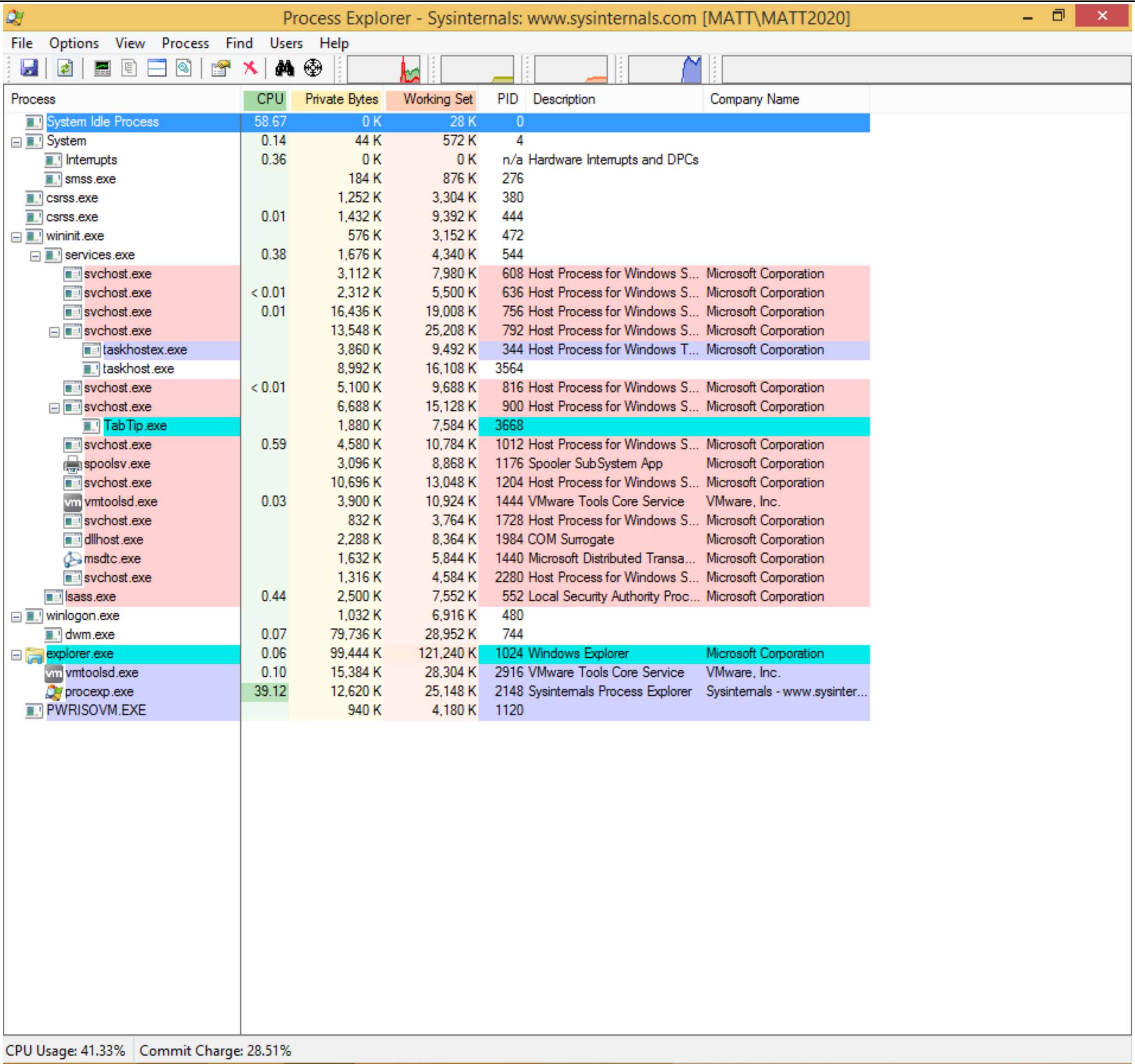


Figure 2.2.2 – Process Explorer's user interface

Process Explorer is a basic dynamic analysis tool developed by Microsoft that keeps track of processes running in the system during malware analysis. This is important to be kept open as it would allow the analyst to view the changes the malware does to the system through Process Explorer. It provides each process with a PID (Process ID) in the case that there are multiple processes with the same name. Process Monitor is especially useful when identifying the process name and malware running, alongside the details of all the handles and DLL loaded by the malware.

Process Explorer can be downloaded from the official website of Microsoft: [https://docs.microsoft.com/en- us/sysinternals/downloads/process-explorer](https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer)

2.2.3. Process Monitor

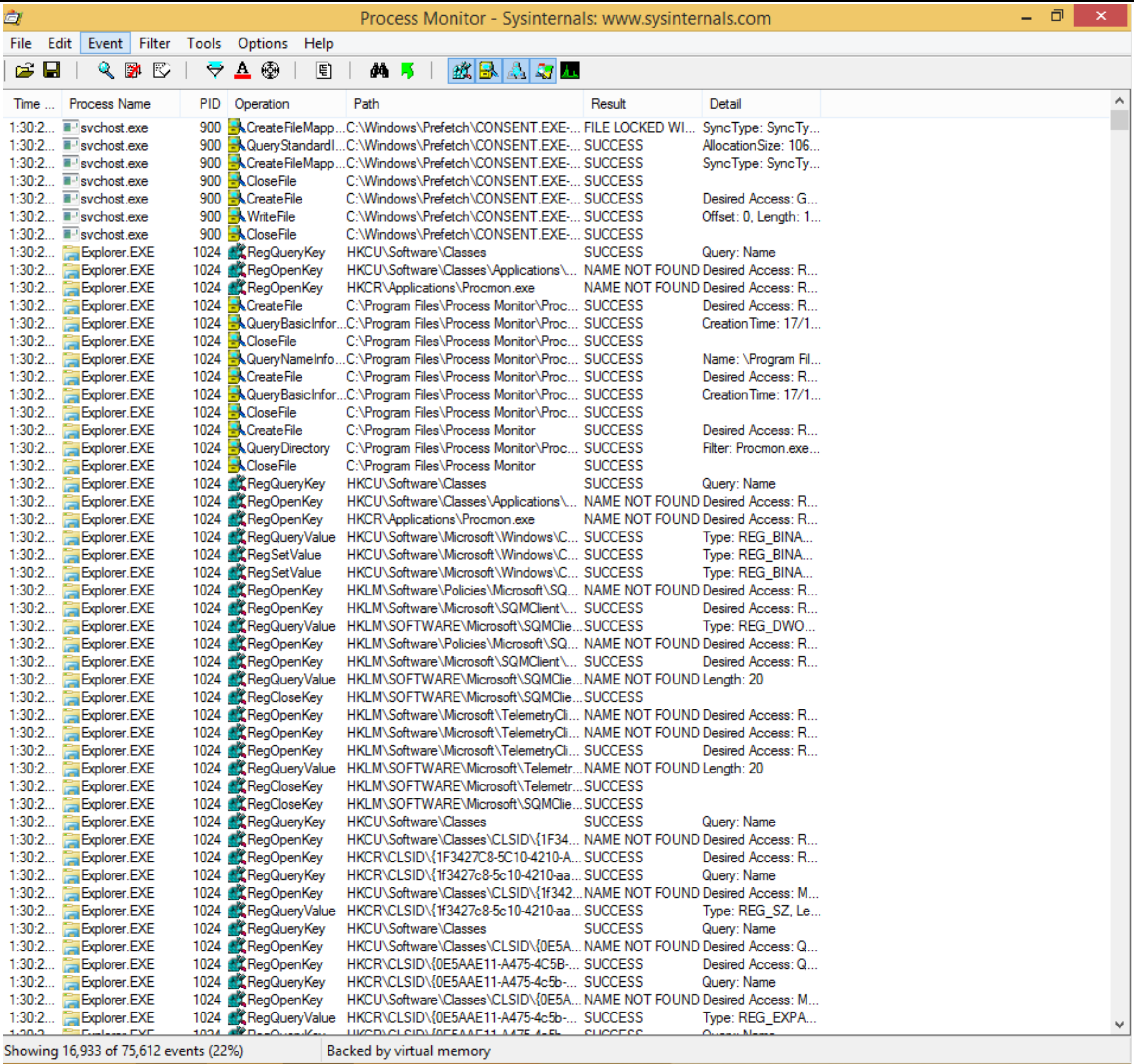


Figure 2.2.3 – Process Monitor's user interface

Process Monitor is a widely used basic dynamic analysis tool to monitor Windows filesystem, registry, and process activity in real-time. Process Monitor combines 2 tools, FileMon and RegMon. Additional features added were non-destructive filtering of data and boot time logging, just to name a few. This means that all data was captured by Process Monitor, but only a select few were displayed to the user. Process Monitor is useful when it comes to determining system changes made by the malware in real time.

Process Monitor can be downloaded from the official website of Microsoft: [https://docs.microsoft.com/en- us/sysinternals/downloads/procmon](https://docs.microsoft.com/en-us/sysinternals/downloads/procmon)

2.2.4. Regshot

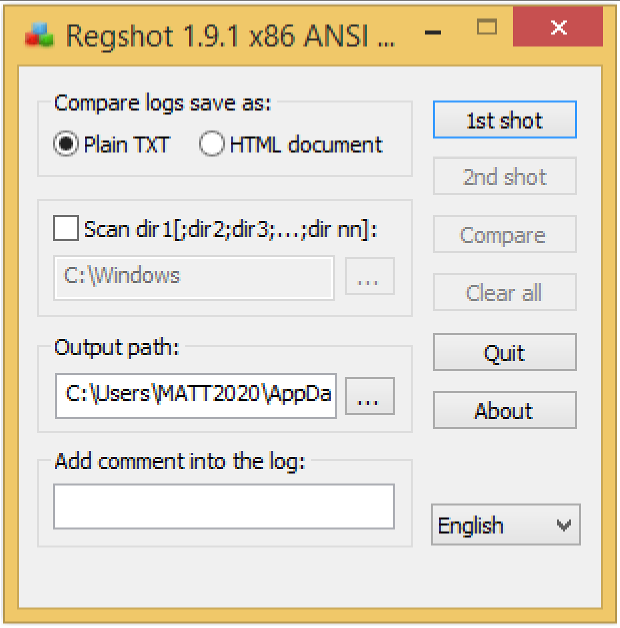


Figure 2.2.4 – Regshot's user interface

Regshot is an open-source dynamic analysis tool used to monitor system registry for changes by taking a snapshot before and after the malware is run. This is to check whether the malware made any changes to the system upon execution. This is especially useful for a basic dynamic analysis as most changes takes place in the system registry during a malware attack. Hence, with this tool, analyst would be able to easily observe the changes made in the registry, which will provide analyst with some clues and hints as to what the malware functionality is about.

We are using Regshot 1.9.1 as it is the more updated version. Although it is in the beta stages, it is helpful as Regshot 1.9.0 had an error when comparing and attempting to display the output of the changes of the system.

This tool can be downloaded from:

https://sourceforge.net/projects/regshot/files/regshot/1.9.1-beta/Regshot-1.9.1-beta_r321.7z/download

2.2.5. Netcat

Netcat is a network analysis tool used for reading from and writing to network connections either through TCP or UDP. It has features such as port scanning, transferring files and port listening, all of which are essential for a network analysis tool. It is capable of detecting the request sent out by the malware to external networks.

Netcat can be downloaded from http://netcat.sourceforge.net/download.php

3. Analysis of Executable Malware

Firstly, the analysis of executable malware will be conducted. The malware used is WannaPeace.exe and is a well-known ransomware that could potentially stop the user from accessing their PC or data within the PC. It might also ask the users to pay the hackers for them to release the ransomware grip on the user's device.
The malware used was downloaded from theZoo on GitHub and the password of the zip file is "infected".

Type	Win32 Executable
Filename	WannaPeace.exe
MD5 Hash	eefa6f98681d78b63f15d7e58934c6cc
URL Download	https://github.com/ytisf/theZoo/blob/master/malwares/Binaries/Win32.WannaPeace/Win32.WannaPeace.zip

3.1. Basic Static Analysis

Basic static analysis will be conducted on WannaPeace.exe using the tools shown in the section in the report above. This will provide us with a brief overview of the malware and the functionalities.

3.1.1. De-obfuscation and Unpacking

Before performing malware analysis, it is essential to verify that the malware is not obfuscated, meaning that it is concealed so it cannot be analyzed. The simplest method to perform obfuscation is to pack the malware using a packer, such as UPX. When the malware is packed or obfuscated, the code and functionality of the malware is concealed, and basic static analysis will not be extract anything useful from the malware in this state. By using PEiD, we can determine is the malware is packed, which tool the attacker packed the malware with if it is packed and unpack the malware if required.

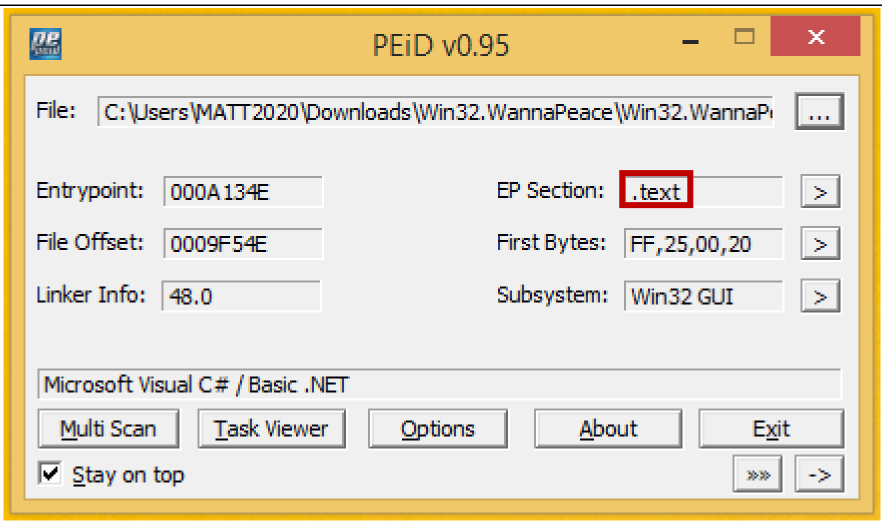


Figure 3.1.1 – PeiD analysis of WannaPeace.exe

The image below shows the analysis of WannaPeace.exe using PEiD. As seen from the image, the EP Section (Entry Point Section) indicated .text, meaning that the malware was not packed. A typical packed malware would indicate the packer's signature in the EP Section. Microsoft Visual C# was identified as the complier for WannaPeace.exe. Since the malware was not packed in the first place, it does not need to be unpacked manually by us.

3.1.2. Fingerprinting

Fingerprinting is crucial to the process of malware analysis as the analyst must identify the malware when the analysis is complete. The analyst would compare the hash values of the malware to ensure and check the name of the malware did not change when running on the isolated device. Fingerprinting is done by running the malware through a hashing algorithm to generate a unique hash for the malware that is irreversible. WinMD5 tool was chosen for this task as it is reliable and efficient, and more programs can be added to directly compare the has values.

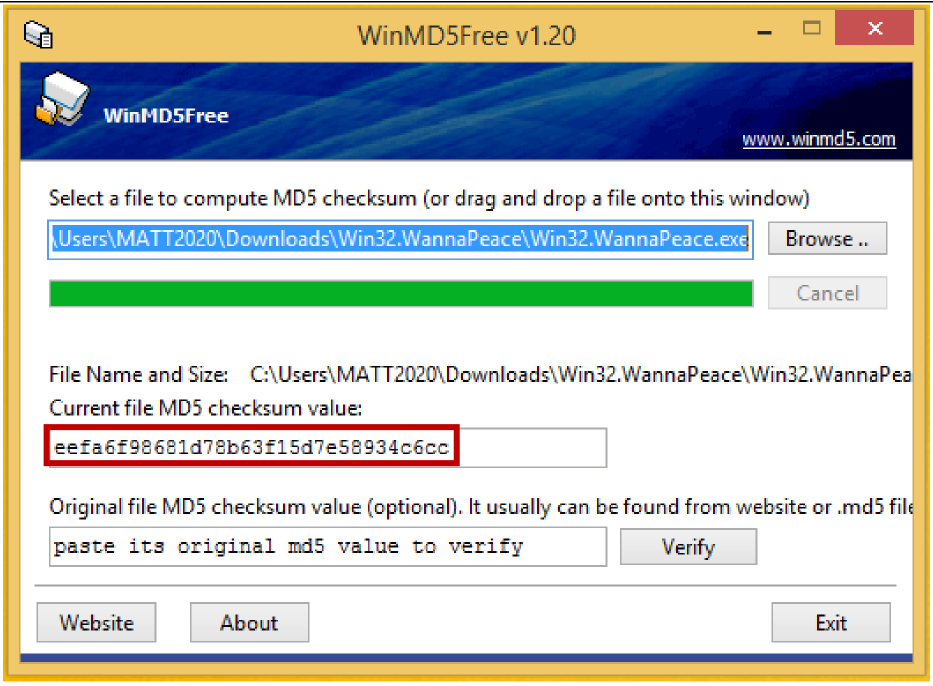


Figure 3.1.2.1 – WinMD5 checksum value of WannaPeace.exe

The image above shows the MD5 checksum value of WannaPeace.exe. This can be further verified by the checksum shown in theZoo on Github, indicating that the malware analyzed in this isolated system is the same as the one initially downloaded.

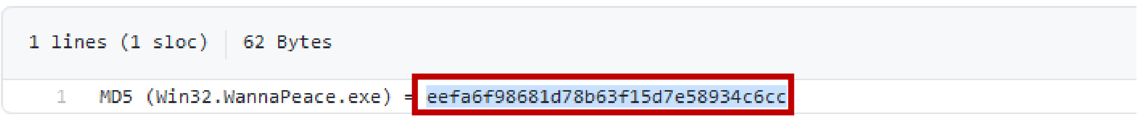


Figure 3.1.2.2 – theZoo MD5 checksum value of WannaPeace.exe

3.1.3. Malware Information

Obtaining information is an important part of malware analysis as it will tell us more about the malware. The information can be extracted from the PE file, which contains information that would allow analyst to extract the structure of the malware as it could display the import functions and DLLs of the malware.

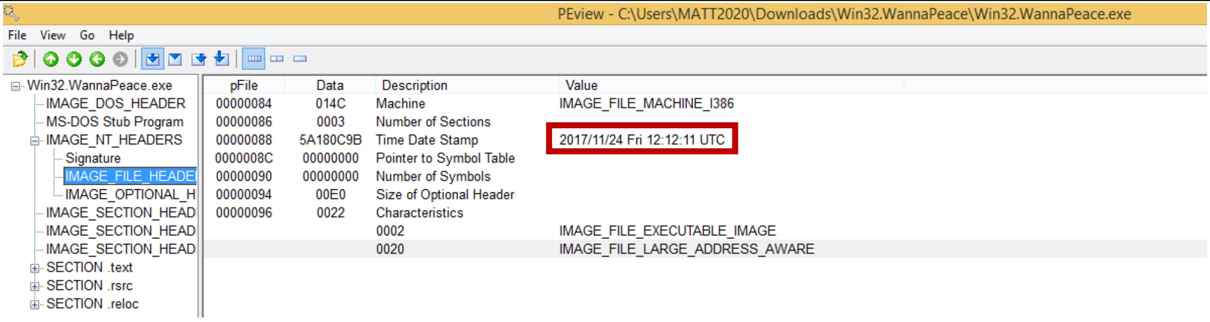


Figure 3.1.3.1 – File header of WannaPeace.exe using PEView

From the image shown above, it shows the potential date and time the malware was complied, which was on the 24th of November 2017, on Friday 12:12:11 UTC time.

Unfortunately, PEView did not display any functions of imported DLLs of the malware. Fortunately, the next tool used would be able to display all the imported DLLs and functions of the malware with speed and accuracy.

3.1.4. Dependent DLLs

DLLs are essential to every program as they depend on DLLs and import functions for the program to work correctly. The image below shows the DLLs of WannaPeace.exe using Dependency Walker on the isolated VM.

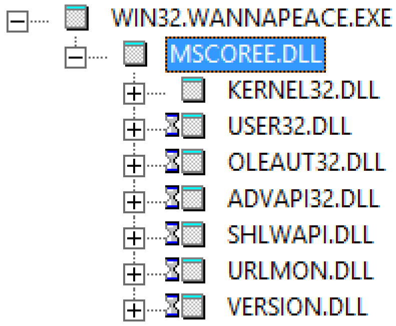


Figure 3.1.4.1 DLLs of WannaPeace.exe using Dependency Walker

There is a total of 7 different imported DLLs shown in the image above. Each of these DLL serves different function to allow the malware to function properly, containing host-based and network-based indicators. These DLLs potentially contains hints on how the malware would work.

3.1.4.1. KERNEL32.DLL

KERNEL32.DLL is the lowest-level DLL, providing basic functionalities such as memory management, interrupt handling and input / output handling. KERNEL32.DLL is most likely needed for the malware to properly execute, since it contains functions where the malware is required to execute processes and manage RAM.

The images below are the imported functions of KERNEL32.DLL. Some of them are boxed in a red box, indicating that it is an important function which could provide some information about the functions of WannaPeace.exe.

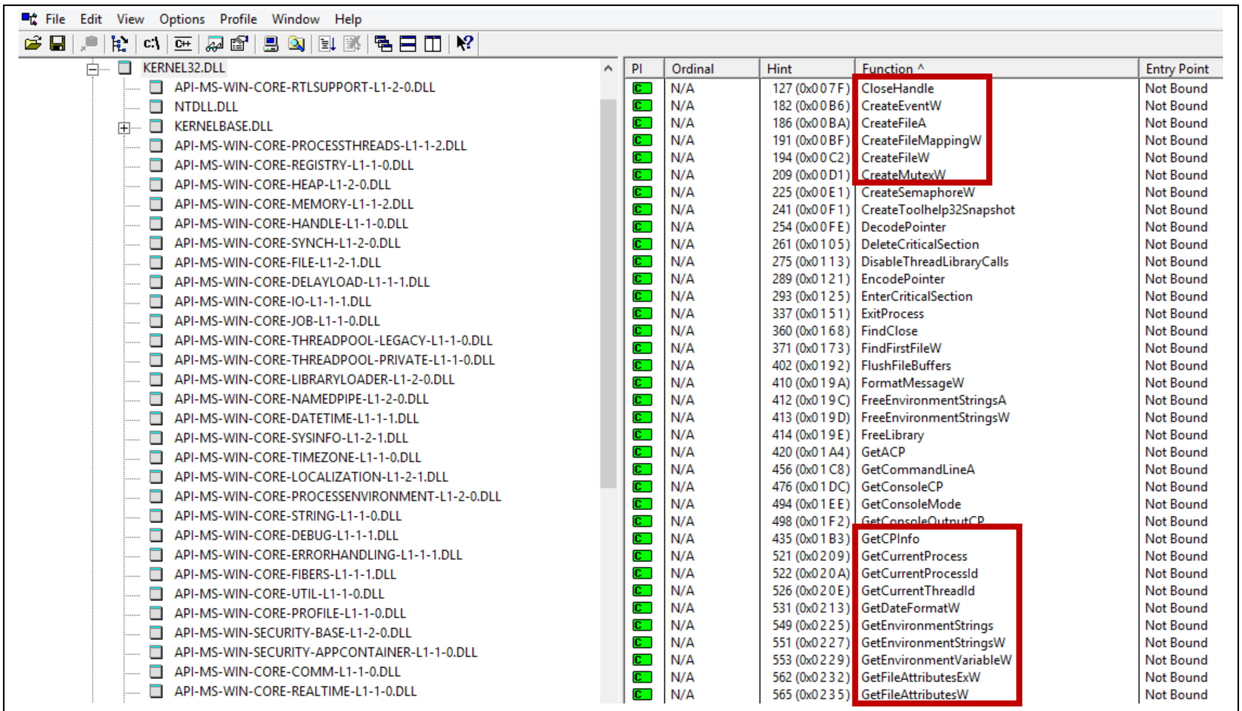


Figure 3.1.4.1.1 – DLLs of KERNEL32.DLL in WannaPeace.exe using Dependency Walker

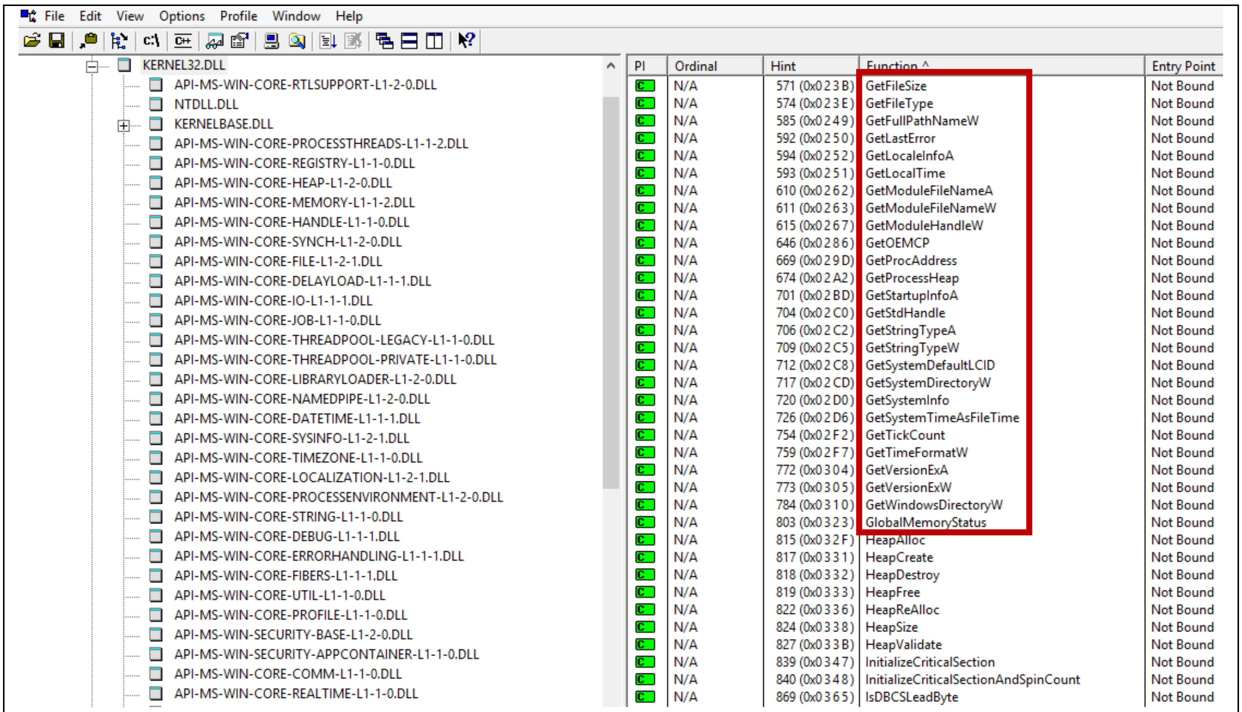


Figure 3.1.4.1.2 – DLLs of KERNEL32.DLL in WannaPeace.exe using Dependency Walker

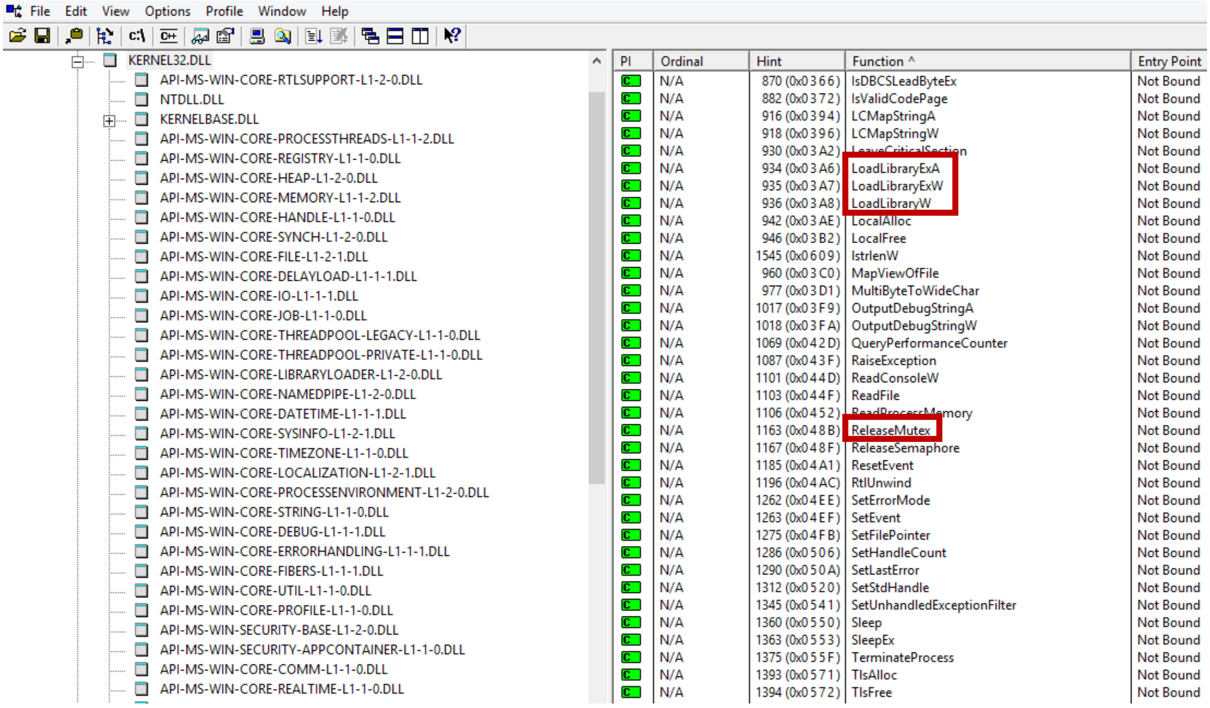


Figure 3.1.4.1.3 – DLLs of KERNEL32.DLL in WannaPeace.exe using Dependency Walker

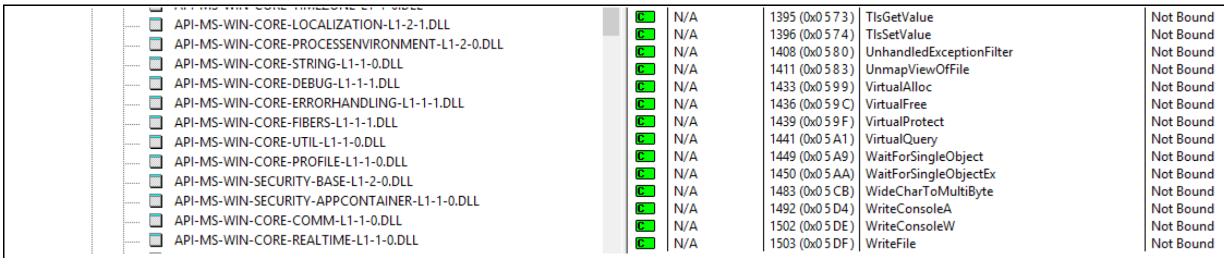


Figure 3.1.4.1.4 – DLLs of KERNEL32.DLL in WannaPeace.exe using Dependency Walker

From the images above, KERNEL32.DLL imported a lot of functions. However, the ones indicated in the red box were highlighted.

Figure Number	Function Name	What does it do?
3.1.4.1.1	CloseHandle	Closes an open object handle.
3.1.4.1.1	CreateEventW	Creates or opens a named or unnamed event object.
3.1.4.1.1	CreateFileA	Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified.
3.1.4.1.1	CreateFileMappingW	Creates or opens a named or unnamed file mapping object for a specified file.
3.1.4.1.1	CreateFileW	Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified.
3.1.4.1.1	CreateMutexW	Creates or opens a named or unnamed mutex object.
3.1.4.1.1	GetCPInfo	Retrieves information about any valid installed or available code page.
3.1.4.1.1	GetCurrentProcess	Retrieves a pseudo handle for the current process.
3.1.4.1.1	GetCurrentProcessId	Retrieves the process identifier of the calling process.
3.1.4.1.1	GetCurrentThreadId	Retrieves the thread identifier of the calling thread.
3.1.4.1.1	GetDateFormatW	Formats a date as a date string for a locale specified by the locale identifier. The function formats either a specified date or the local system date.
3.1.4.1.1	GetEnvironmentStrings	Retrieves the environment variables for the current process.
3.1.4.1.1	GetEnvironmentStringsW	Retrieves the environment variables for the current process.
3.1.4.1.1	GetEnvironmentVariableW	Retrieves the contents of the specified variable from the environment block of the calling process.
3.1.4.1.1	GetFileAttributesExW	Retrieves attributes for a specified file or directory.
3.1.4.1.1	GetFileAttributesW	Retrieves file system attributes for a specified file or directory.
3.1.4.1.2	GetFileSize	Retrieves the size of the specified file, in bytes.

3.1.4.1.2	GetFileType	Retrieves the file type of the specified file.
3.1.4.1.2	GetFullPathNameW	Retrieves the full path and file name of the specified file.
3.1.4.1.2	GetLastError	Retrieves the calling thread's last-error code value. The last-error code is maintained on a per-thread basis. Multiple threads do not overwrite each other's last-error code.
3.1.4.1.2	GetLocaleInfoA	Retrieves information about a locale specified by identifier.
3.1.4.1.2	GetLocalTime	Retrieves the current local date and time.
3.1.4.1.2	GetModuleFileNameA	Retrieves the fully qualified path for the file that contains the specified module. The module must have been loaded by the current process.
3.1.4.1.2	GetModuleFileNameW	Retrieves the fully qualified path for the file that contains the specified module. The module must have been loaded by the current process.
3.1.4.1.2	GetModuleHandleW	Retrieves a module handle for the specified module. The module must have been loaded by the calling process.
3.1.4.1.2	GetOEMCP	Returns the current original equipment manufacturer (OEM) code page identifier for the operating system.
3.1.4.1.2	GetProcAddress	Retrieves the address of an exported function or variable from the specified dynamic-link library (DLL).
3.1.4.1.2	GetProcessHeap	Retrieves a handle to the default heap of the calling process. This handle can then be used in subsequent calls to the heap functions.
3.1.4.1.2	GetStartupInfoA	Retrieves the contents of the STARTUPINFO structure that was specified when the calling process was created.
3.1.4.1.2	GetStdHandle	etrieves a handle to the specified standard device (standard input, standard output, or standard error).
3.1.4.1.2	GetStringTypeA	Deprecated. Retrieves character type information for the characters in the specified source string. For each character in the string, the function sets one or more bits in the corresponding 16-bit element of the output array. Each bit identifies a given character type, for example, letter, digit, or neither.
3.1.4.1.2	GetStringTypeW	Retrieves character type information for the characters in the specified Unicode source string. For each character in the string, the function sets one or more bits in the corresponding 16-bit element of the output array. Each bit identifies a given character type, for example, letter, digit, or neither.

3.1.4.1.2	GetSystemDefaultLCID	Returns the locale identifier for the system locale.
3.1.4.1.2	GetSystemDirectoryW	Retrieves the path of the system directory. The system directory contains system files such as dynamic-link libraries and drivers.
3.1.4.1.2	GetSystemInfo	Retrieves information about the current system.
3.1.4.1.2	GetSystemTimeAsFileTime	Retrieves the current system date and time. The information is in Coordinated Universal Time (UTC) format.
3.1.4.1.2	GetTickCount	Retrieves the number of milliseconds that have elapsed since the system was started, up to 49.7 days.
3.1.4.1.2	GetTimeFormatW	Formats time as a time string for a locale specified by identifier. The function formats either a specified time or the local system time.
3.1.4.1.2	GetVersionExA	Retrieves version of Operating System.
3.1.4.1.2	GetVersionExW	Retrieves version of Operating System.
3.1.4.1.2	GetWindowsDirectoryW	Retrieves the path of the Windows directory.
3.1.4.1.2	GlobalMemoryStatus	Retrieves information about the system's current usage of both physical and virtual memory.
3.1.4.1.3	LoadLibraryA	Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.
3.1.4.1.3	LoadLibraryExW	Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.
3.1.4.1.3	LoadLibraryW	Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.
3.1.4.1.3	ReleaseMutex	Releases ownership of the specified mutex object.

From these highlighted functions, there are a lot of “Get” functions, which are used to obtain information from the infected device such as command line, file attributes, and full path names. There is also “Load” function, where the malware will load a specified module into a particular address space, and the “Create” function is used to create events. A ReleaseMutex was spotted and it releases the ownership of a specified mutex object, this function would fail if the calling thread does not own the mutex object.

From these imported functions found in KERNEL32.DLL, it is suspected that WannaPeace.exe malware would obtain data and files of the users due to the types of get function stated in KERNEL32.DLL. However, these DLLs are needed for all programs that run on DOS and cannot be classified as malicious based on this evidence alone.

3.1.4.2. USER32.DLL

This DLL implements the Windows USER component that creates and manipulates the standard elements of the Windows user interface, such as desktop windows, menus and more. It allows programs to implement a graphical user interface (GUI) matching the Windows look and feel. Programs call functions from Windows USER to perform operations such as create and manage windows, receive windows messages which usually comes in a form of mouse and keyboard inputs. There are little to no functions within USER32.DLL which could be used maliciously.

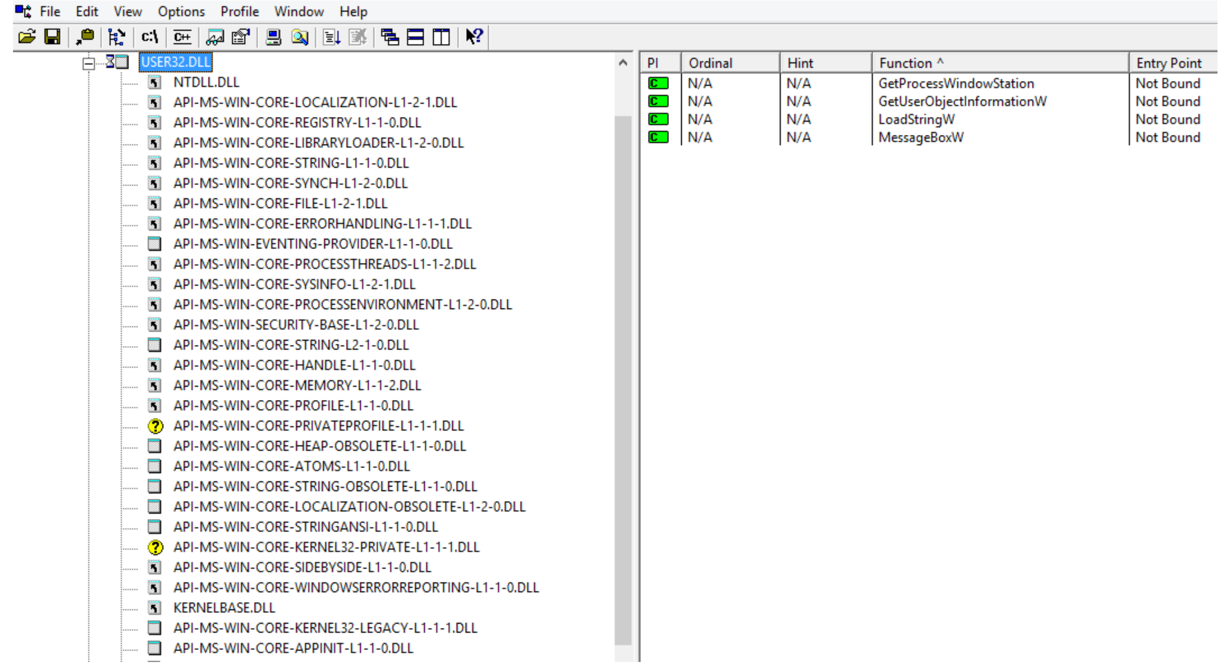


Figure 3.1.4.2.1 – DLLs of USER32.DLL in WannaPeace.exe using Dependency Walker

Figure Number	Function Name	What does it do?
3.1.4.2.1	GetProcessWindowStation	Retrieves a handle to the current window station for the calling process.
3.1.4.2.1	GetUserObjectInformationW	Retrieves information about the specified window station or desktop object.
3.1.4.2.1	LoadStringW	Loads a string resource from the executable file associated with a specified module and either copies the string into a buffer with a terminating null character or returns a read-only pointer to the string resource itself.
3.1.4.2.1	MessageBoxW	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

3.1.4.3. OLEAUT32.DLL

There are no imported functions within this DLL.

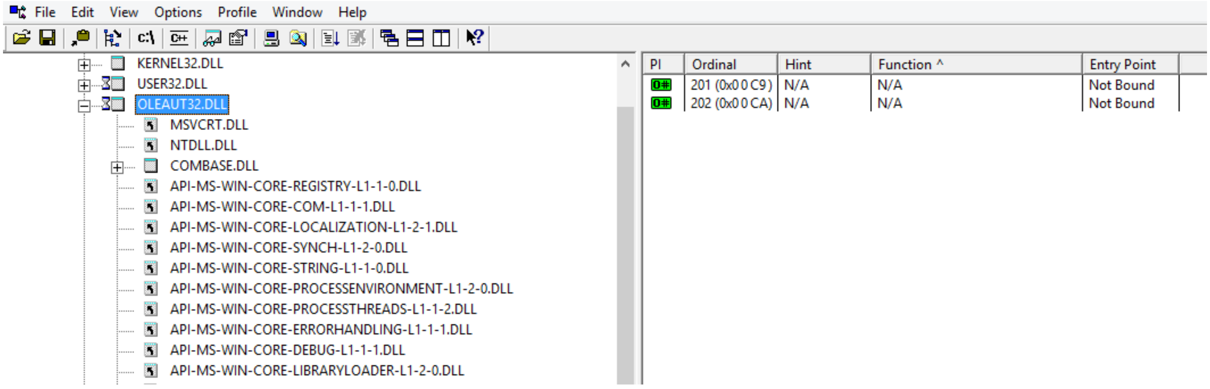


Figure 3.1.4.3.1 – DLLs of OLEAUT32.DLL in WannaPeace.exe using Dependency Walker

3.1.4.4. ADVAPI32.DLL

Advanced API Services Library (ADVAPI32.DLL) was made by Microsoft, designed to support numerous APIs such as registry and security calls. This also further provides functionality and is the one responsible for restarting and shutting down the system, Windows Registry, and managing user accounts.

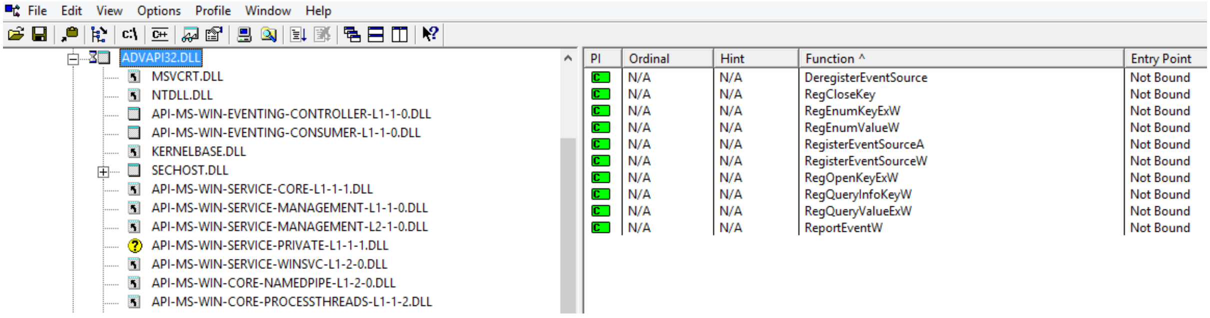


Figure 3.1.4.4.1 – DLLs of ADVAPI32.DLL in WannaPeace.exe using Dependency Walker

Figure Number	Function Name	What does it do?
3.1.4.4.1	DeregisterEventSource	Closes the specified event log.
3.1.4.4.1	RegCloseKey	Closes a handle to the specified registry key.
3.1.4.4.1	RegEnumKeyExW	Enumerates the subkeys of the specified open registry key. The function retrieves information about one subkey each time it is called.
3.1.4.4.1	RegEnumValueW	Enumerates the values for the specified open registry key. The function copies one indexed value name and data block for the key each time it is called.
3.1.4.4.1	RegisterEventSourceA	Retrieves a registered handle to the specified event log.
3.1.4.4.1	RegisterEventSourceW	Retrieves a registered handle to the specified event log.
3.1.4.4.1	RegOpenKeyExW	Opens the specified registry key. Note that key names are not case sensitive.
3.1.4.4.1	RegQueryInfoKeyW	Retrieves information about the specified registry key.
3.1.4.4.1	RegQueryValueExW	Retrieves the type and data for the specified value name associated with an open registry key.
3.1.4.4.1	ReportEventW	Writes an entry at the end of the specified event log.

From the analysis above, I think that there are no suspicious import functions within APIADV32.DLL as all the import functions do seem necessary and are required for the management of Windows.

3.1.4.5. SHLWAPI.DLL

SHLWAPI.DLL, also known as Shell Light-Weight Utility Library, contains functions for URL and UNC paths, registry entries and color settings. This is one of the DLLs which is crucial to the system process and should not be removed, otherwise, the device may fail to function properly. The file contains machine code, and it could be loaded into RAM and run as a task.

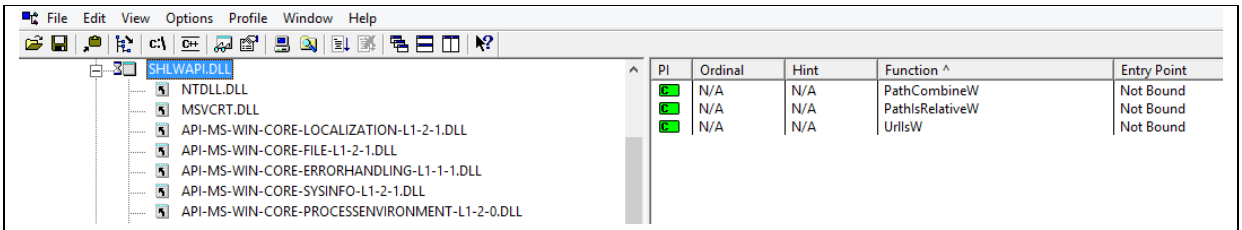


Figure 3.1.4.4.1 – DLLs of ADVAPI32.DLL in WannaPeace.exe using Dependency Walker

Figure Number	Function Name	What does it do?
3.1.4.5.1	PathCombineW	Concatenates two strings that represent properly formed paths into one path; also concatenates any relative path elements.
3.1.4.5.1	PathIsRelativeW	Searches a path and determines if it is relative.
3.1.4.5.1	UrlIsW	Tests whether a URL is a specified type.

Hence, SHLWAPI.DLL does not pose a threat or contain malicious codes which could compromise the security and safety of the isolated VM.

3.1.4.6. URLMON.DLL

It was developed by Microsoft and is a module that contains functions used by Microsoft OLE (Object Linking and Embedding). This is a process required for the device to properly function. This file contains machine code and would run on RAM when executed on the device in a form of a task. It is also rated to not pose any harm to the system.

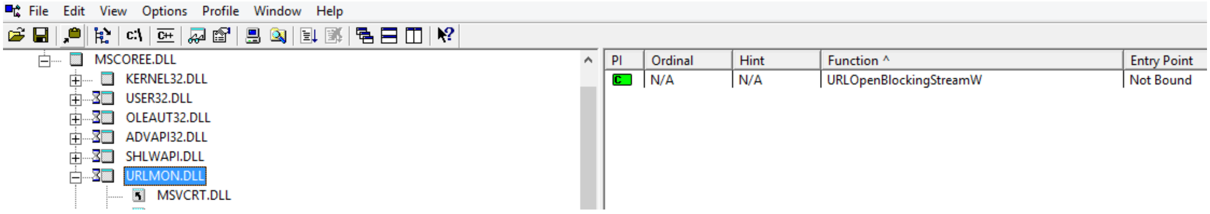


Figure 3.1.4.6.1 – DLLs of URLMON.DLL in WannaPeace.exe using Dependency Walker

Figure Number	Function Name	What does it do?
3.1.4.6.1	URLOpenBlockingStreamW	Creates a blocking type stream object from a URL and downloads the data from the Internet. When the data is downloaded, the client application or control can read it by using the IStream::Read method.

Thus, URLMON.DLL does not pose a threat or contain malicious codes which could compromise the security and safety of the isolated VM.

3.1.4.7. VERSION.DLL

This is a module containing application programming interface (API) functions used for Windows version by applications. It is a necessary system process and should not be removed. It contains machine code and when the software starts, the commands in VERSION.DLL will be executed on the system. It will then load into RAM and run as a version checking and file installation library process.

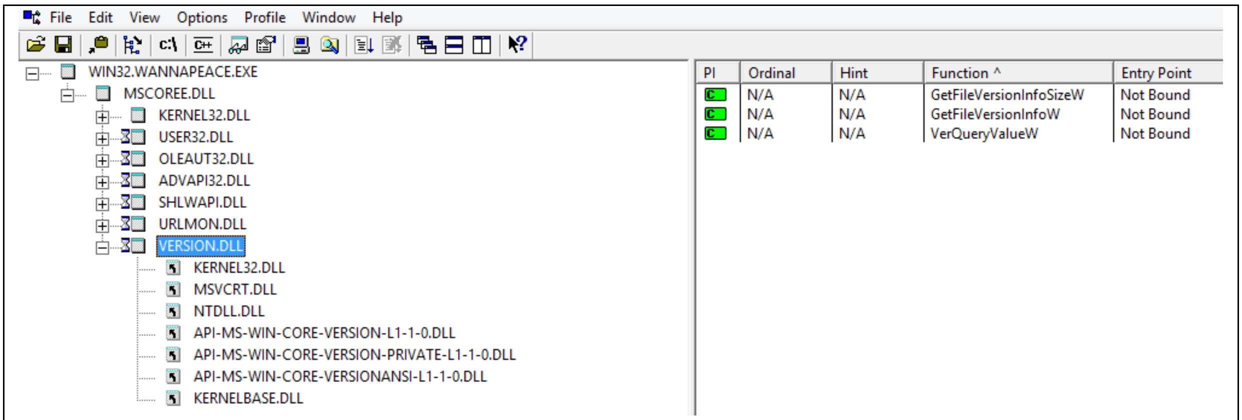


Figure 3.1.4.7.1 – DLLs of VERSION.DLL in WannaPeace.exe using Dependency Walker

Figure Number	Function Name	What does it do?
3.1.4.7.1	GetFileVersionInfoSizeW	Determines whether the operating system can retrieve version information for a specified file. If version information is available, GetFileVersionInfoSize returns the size, in bytes, of that information.
3.1.4.7.1	GetFileVersionInfoW	Retrieves version information for the specified file.
3.1.4.7.1	VerQueryValueW	Retrieves specified version information from the specified version-information resource. To retrieve the appropriate resource, before you call VerQueryValue, you must first call the GetFileVersionInfoSize function, and then the GetFileVersionInfo function.

Therefore, VERSION.DLL does not pose a threat or contain malicious codes which could compromise the security and safety of the isolated VM.

3.1.5. String Analysis

Malware files contain strings inside the code, which were placed by developers to aid in the programming process. String analysis refers to the extraction of strings from the malware, helping analyst identify the functionalities of the malware. Strings can include IP addresses, hostnames, or function names.

BinText is incredibly useful when it comes to displaying strings from the malware. BinText is a file text scanner / extractor that helps find character strings buried in binary files. It extracts text from any kind of files and displays plain ASCII text, Unicode text and Resource strings.

```
A 000000077928   000000479728   0   string path = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFiles) + @"\drivers.txt";
A 00000007798F   00000047978F   0       if (System.IO.File.Exists(@"" + path))
A 0000000779C3   0000004797C3   0       {
A 0000000779D6   0000004797D6   0           ProcessStartInfo si = new ProcessStartInfo();
A 000000077A15   000000479815   0           si.FileName = AppDomain.CurrentDomain.BaseDirectory + @"RzW.exe";
A 000000077A68   000000479868   0           si.UseShellExecute = true;
A 000000077A96   000000479896   0           Process.Start(si);
A 000000077ABC   0000004798BC   0       }
```

Figure 3.1.5.1 – Strings in WannaPeace.exe using BinText

When analyzing the code, it’s trying to find the folder path to a special folder to add driver.txt into it. It will then start a new process, using Shell Execute and start the process immediately.

```
A 000000089976  00000048B776  0    26/10/2017
A 00000008998B  00000048B78B  0    using System;
A 00000008999A  00000048B79A  0    using System.Collections;
A 0000000899B5  00000048B7B5  0    using System.Collections.Generic;
A 0000000899D8  00000048B7D8  0    using System.Globalization;
A 0000000899F5  00000048B7F5  0    using System.IO;
A 000000089A07  00000048B807  0    using System.Reflection;
A 000000089A21  00000048B821  0    using System.Resources;
A 000000089A3A  00000048B83A  0    using System.Security.Cryptography;
A 000000089A5F  00000048B85F  0    using System.Text;
A 000000089A73  00000048B873  0    using System.Threading;
A 000000089A8E  00000048B88E  0    namespace standalone
A 000000089AA7  00000048B8A7  0        class Program
A 000000089ABA  00000048B8BA  0        {
A 000000089AC1  00000048B8C1  0            static void Main(string[] args)
A 000000089AEA  00000048B8EA  0            {
A 000000089AF5  00000048B8F5  0                bool status = false;
A 000000089B17  00000048B917  0                string pw = "";
A 000000089B34  00000048B934  0                string dir = System.Environment.CurrentDirectory + @"\Unlocked";
A 000000089B82  00000048B982  0                Console.WriteLine("Locker Standalone Decrypter");
A 000000089BC1  00000048B9C1  0                Console.WriteLine("-------------------------");
A 000000089C00  00000048BA00  0                Console.WriteLine("Enter Password to decrypt:");
A 000000089C40  00000048BA40  0                pw = Console.ReadLine();
A 000000089C68  00000048BA68  0                if (pw != string.Empty)
A 000000089C8D  00000048BA8D  0                {
A 000000089C9C  00000048BA9C  0                    ExtractResources(dir);
A 000000089CC4  00000048BAC4  0                    status = EncryptionUtils.DecryptFolder(dir, pw,true);
A 000000089D0D  00000048BB0D  0                    if (status)
A 000000089D2A  00000048BB2A  0                    {
A 000000089D3D  00000048BB3D  0                        Console.WriteLine("Decryption Completed");
A 000000089D7D  00000048BB7D  0                    }
A 000000089D90  00000048BB90  0                    else
A 000000089DA6  00000048BBA6  0                    {
A 000000089DB9  00000048BBB9  0                        try
A 000000089DD2  00000048BBD2  0                        {
A 000000089DE9  00000048BBE9  0                            Thread.Sleep(2000);
A 000000089E16  00000048BC16  0                            Directory.Delete(dir, true);
A 000000089E4C  00000048BC4C  0                            Console.WriteLine("Decryption Failed");
A 000000089E8D  00000048BC8D  0                        }
A 000000089EA4  00000048BCA4  0                        catch (Exception ex)
A 000000089ECE  00000048BCCE  0                        {
A 000000089EE5  00000048BCE5  0                            Console.WriteLine(ex.Message);
A 000000089F1F  00000048BD1F  0                        }

A 000000089F36  00000048BD36  0                    }
A 000000089F49  00000048BD49  0                }
A 000000089F5A  00000048BD5A  0
A 000000089F6A  00000048BD6A  0            }
A 000000089F77  00000048BD77  0            private static void ExtractResources(string dir)
A 000000089FB1  00000048BDB1  0            {
A 000000089FBC  00000048BDBC  0                try
A 000000089FCD  00000048BDCD  0                {
A 000000089FDC  00000048BDDC  0                    string fInfo = "";
A 00000008A000  00000048BE00  0                    Assembly asm = Assembly.GetExecutingAssembly();
A 00000008A041  00000048BE41  0                    Stream fstr = null;
A 00000008A068  00000048BE68  0                    //Create The output Directory if it Doesn't Exist
A 00000008A0AB  00000048BEAB  0                    if (!Directory.Exists(dir))
A 00000008A0D8  00000048BED8  0                    {
A 00000008A0EB  00000048BEEB  0                        Directory.CreateDirectory(dir);
A 00000008A120  00000048BF20  0                    }
A 00000008A135  00000048BF35  0                    //Loop thru all the resources and Extract them
A 00000008A175  00000048BF75  0                    foreach (string resourceName in asm.GetManifestResourceNames())
A 00000008A1C6  00000048BFC6  0                    {
A 00000008A1D9  00000048BFD9  0                        fInfo = dir + @"\" + resourceName.Replace(asm.GetName().Name + ".Resources.", "");
A 00000008A241  00000048C041  0                        fstr = asm.GetManifestResourceStream(resourceName);
A 00000008A28C  00000048C08C  0                        if(fInfo.Contains("Lzma"))
A 00000008A2BC  00000048C0BC  0                        {
A 00000008A2D3  00000048C0D3  0                            fInfo = System.Environment.CurrentDirectory + "\\" + resourceName.Replace(asm.GetName().Name + ".Resources.", "");
A 00000008A35F  00000048C15F  0                        }
A 00000008A378  00000048C178  0                        if (fstr != null && !fInfo.Contains("key2.ico"))
A 00000008A3BE  00000048C1BE  0                        {
A 00000008A3D5  00000048C1D5  0                            SaveStreamToFile(fInfo, fstr);
A 00000008A40D  00000048C20D  0                        }
A 00000008A424  00000048C224  0                    }
A 00000008A439  00000048C239  0                }
A 00000008A448  00000048C248  0                catch (Exception ex)
A 00000008A46A  00000048C26A  0                {
A 00000008A479  00000048C279  0                    Console.WriteLine(ex.Message);
A 00000008A4A9  00000048C2A9  0                }
A 00000008A4B8  00000048C2B8  0            }
A 00000008A4C5  00000048C2C5  0            private static void SaveStreamToFile(string fileFullPath, Stream stream)
A 00000008A517  00000048C317  0            {
A 00000008A522  00000048C322  0                if (stream.Length == 0) return;
A 00000008A551  00000048C351  0                // Create a FileStream object to write a stream to a file
A 00000008A598  00000048C398  0                using (FileStream fileStream = System.IO.File.Create(fileFullPath, (int)stream.Length))
A 00000008A5FD  00000048C3FD  0                {
A 00000008A60C  00000048C40C  0                    // Fill the bytes[] array with the stream data
A 00000008A64C  00000048C44C  0                    byte[] bytesInStream = new byte[stream.Length];
A 00000008A68D  00000048C48D  0                    stream.Read(bytesInStream, 0, (int)bytesInStream.Length);
A 00000008A6DA  00000048C4DA  0                    // Use FileStream object to write to the specified file
A 00000008A723  00000048C523  0                    fileStream.Write(bytesInStream, 0, bytesInStream.Length);
```

```
A 00000008A76E  00000048C56E  0          }
A 00000008A77D  00000048C57D  0      }
A 00000008A78A  00000048C58A  0  }
A 00000008A793  00000048C593  0  public static class EncryptionUtils
A 00000008A7BC  00000048C5BC  0  {
A 00000008A7C3  00000048C5C3  0      #region Variables
A 00000008A7E0  00000048C5E0  0      static List<FileInfo> files = new List<FileInfo>();  // List that will hold the files and sub files in path
A 00000008A855  00000048C655  0      static List<DirectoryInfo> folders = new List<DirectoryInfo>(); // List that hold directories that cannot be accessed
A 00000008A8D6  00000048C6D6  0      #endregion
A 00000008A8EC  00000048C6EC  0      #region Public Methods
A 00000008A90E  00000048C70E  0      public static bool DecryptFolder(string folderDirectory, string pword, bool compressed)
A 00000008A96F  00000048C76F  0      {
A 00000008A97A  00000048C77A  0          bool status = false;
A 00000008A99C  00000048C79C  0          string fileLocation = "";
A 00000008A9C3  00000048C7C3  0          string salt = "";
A 00000008A9E2  00000048C7E2  0          byte[] encPW = null;
A 00000008AA06  00000048C806  0          try
A 00000008AA17  00000048C817  0          {
A 00000008AA26  00000048C826  0              status = Directory.Exists(folderDirectory);
A 00000008AA65  00000048C865  0              if (status)
A 00000008AA82  00000048C882  0              {
A 00000008AA95  00000048C895  0                  DirectoryInfo di = new DirectoryInfo(folderDirectory);
A 00000008AAE3  00000048C8E3  0                  //Clear Folder and File list
A 00000008AB15  00000048C915  0                  folders = new List<DirectoryInfo>();
A 00000008AB4F  00000048C94F  0                  files = new List<FileInfo>();
A 00000008AB82  00000048C982  0                  //Build new Folder and File list
A 00000008ABB8  00000048C9B8  0                  GetAllFilesInDir(di, "*");
A 00000008ABEA  00000048C9EA  0                  foreach (FileInfo fi in files)
A 00000008AC1E  00000048CA1E  0                  {
A 00000008AC35  00000048CA35  0                      fileLocation = fi.FullName;
A 00000008AC6C  00000048CA6C  0                      if (fi.Name != "key2.ico" && fi.Name != "Lzma.dll")
A 00000008ACB9  00000048CAB9  0                      {
A 00000008ACD4  00000048CAD4  0                          //Build the Encrypted Password with a unique salt based on the file's info
A 00000008AD3C  00000048CB3C  0                          string fileData = string.Format("{0}", fi.Name.Substring(0, fi.Name.IndexOf(".")));
A 00000008ADAD  00000048CBAD  0                          salt = Convert.ToBase64String(GetBytes(fileData));
A 00000008ADFD  00000048CBFD  0                          encPW = EncryptPassword(pword, "123", salt);
A 00000008AE47  00000048CC47  0                          string strPW = Convert.ToBase64String(encPW);
A 00000008AE94  00000048CC94  0                          DecryptFile(fileLocation, encPW);
A 00000008AED5  00000048CCD5  0                          if (compressed)
A 00000008AF02  00000048CD02  0                          {
A 00000008AF21  00000048CD21  0                              DecompressFileLZMA(fi.FullName, fi.FullName.Replace(".zip", ""));
A 00000008AF86  00000048CD86  0                              //Delete the original file
A 00000008AFC2  00000048CDC2  0                              if (File.Exists(fi.FullName))
A 00000008B001  00000048CE01  0                              {
A 00000008B024  00000048CE24  0                                  File.Delete(fi.FullName);
A 00000008B063  00000048CE63  0                              }
A 00000008B086  00000048CE86  0                          }

A 00000008B0A5  00000048CEA5  0                      }
A 00000008B0C2  00000048CEC2  0                  }
A 00000008B0D9  00000048CED9  0              }
A 00000008B0EC  00000048CEEC  0          }
A 00000008B0FB  00000048CEFB  0          catch (Exception ex)
A 00000008B11D  00000048CF1D  0          {
A 00000008B12C  00000048CF2C  0              Console.WriteLine(ex.Message);
A 00000008B15C  00000048CF5C  0              status = false;
A 00000008B17D  00000048CF7D  0          }
A 00000008B18E  00000048CF8E  0          return status;
A 00000008B1AA  00000048CFAA  0      }
A 00000008B1B7  00000048CFB7  0      /// <summary>
A 00000008B1CE  00000048CFCE  0      /// Encrypt String
A 00000008B1EA  00000048CFEA  0      /// </summary>
A 00000008B202  00000048D002  0      /// <param name="clearText">Clear Text to be Encrypted</param>
A 00000008B24A  00000048D04A  0      /// <param name="password">Password to use during encryption</param>
A 00000008B298  00000048D098  0      /// <param name="salt">Salt to use during Encryption</param>
A 00000008B2DE  00000048D0DE  0      /// <returns></returns>
A 00000008B2FF  00000048D0FF  0      public static byte[] EncryptPassword(string clearText, string password, string salt)
A 00000008B35D  00000048D15D  0      {
A 00000008B368  00000048D168  0          byte[] saltBytes = Encoding.Unicode.GetBytes(salt);
A 00000008B3A9  00000048D1A9  0          byte[] clearBytes = System.Text.Encoding.Unicode.GetBytes(clearText);
A 00000008B3FC  00000048D1FC  0          PasswordDeriveBytes pdb = new PasswordDeriveBytes(password, saltBytes);
A 00000008B453  00000048D253  0          byte[] encryptedData = EncryptPW(clearBytes, pdb.GetBytes(32), pdb.GetBytes(16));
A 00000008B4B2  00000048D2B2  0          //return Convert.ToBase64String(encryptedData); //For returning string instead
A 00000008B50E  00000048D30E  0          return encryptedData;
A 00000008B531  00000048D331  0      }
A 00000008B53C  00000048D33C  0      #endregion
A 00000008B552  00000048D352  0      #region Private Methods
A 00000008B575  00000048D375  0      private static void GetAllFilesInDir(DirectoryInfo dir, string searchPattern)
A 00000008B5CC  00000048D3CC  0      {
A 00000008B5D7  00000048D3D7  0          // list the files
A 00000008B5F6  00000048D3F6  0          try
A 00000008B607  00000048D407  0          {
A 00000008B616  00000048D416  0              foreach (FileInfo f in dir.GetFiles(searchPattern))
A 00000008B65B  00000048D45B  0              {
A 00000008B66E  00000048D46E  0                  //Console.WriteLine("File {0}", f.FullName);
A 00000008B6B0  00000048D4B0  0                  files.Add(f);
A 00000008B6D3  00000048D4D3  0              }
A 00000008B6E6  00000048D4E6  0          }
A 00000008B6F5  00000048D4F5  0          catch
A 00000008B708  00000048D508  0          {
A 00000008B717  00000048D517  0              Console.WriteLine("Directory {0}  \n could not be accessed!!!!", dir.FullName);
A 00000008B778  00000048D578  0              return;  // We already got an error trying to access dir so don't try to access it again
A 00000008B7E2  00000048D5E2  0          }
A 00000008B7F3  00000048D5F3  0          // process each directory
A 00000008B81A  00000048D61A  0          // If I have been able to see the files in the directory I should also be able
```

```
A 00000008B877  00000048D677  0          // to look at its directories so I don't think I should place this in a try catch block
A 00000008B8DC  00000048D6DC  0          foreach (DirectoryInfo d in dir.GetDirectories())
A 00000008B91B  00000048D71B  0          {
A 00000008B92A  00000048D72A  0              folders.Add(d);
A 00000008B94B  00000048D74B  0              GetAllFilesInDir(d, searchPattern);
A 00000008B980  00000048D780  0          }
A 00000008B98F  00000048D78F  0      }
A 00000008B99C  00000048D79C  0      private static byte[] EncryptPW(byte[] clearText, byte[] key, byte[] iv)
A 00000008B9EE  00000048D7EE  0      {
A 00000008B9F9  00000048D7F9  0          MemoryStream ms = new MemoryStream();
A 00000008BA2C  00000048D82C  0          Rijndael alg = Rijndael.Create();
A 00000008BA5B  00000048D85B  0          alg.Key = key;
A 00000008BA77  00000048D877  0          alg.IV = iv;
A 00000008BA91  00000048D891  0          CryptoStream cs = new CryptoStream(ms, alg.CreateEncryptor(), CryptoStreamMode.Write);
A 00000008BAF5  00000048D8F5  0          cs.Write(clearText, 0, clearText.Length);
A 00000008BB2C  00000048D92C  0          cs.Close();
A 00000008BB45  00000048D945  0          byte[] encryptedData = ms.ToArray();
A 00000008BB77  00000048D977  0          return encryptedData;
A 00000008BB9A  00000048D99A  0      }
A 00000008BBA7  00000048D9A7  0      private static void DecryptFile(string inputFile, byte[] key)
A 00000008BBEE  00000048D9EE  0      {
A 00000008BBF9  00000048D9F9  0          string ext = Path.GetExtension(inputFile);
A 00000008BC31  00000048DA31  0          string outputFile = inputFile.Replace(ext, "_enc" + ext);
A 00000008BC7A  00000048DA7A  0          //Prepare the file for decryption by getting it into a stream
A 00000008BCC5  00000048DAC5  0          FileStream fsCrypt = new FileStream(inputFile, FileMode.Open);
A 00000008BD13  00000048DB13  0          //Setup the Decryption Standard using Read mode
A 00000008BD50  00000048DB50  0          RijndaelManaged rijndaelCrypto = new RijndaelManaged();
A 00000008BD95  00000048DB95  0          CryptoStream cs = new CryptoStream(fsCrypt, rijndaelCrypto.CreateDecryptor(key, key), CryptoStreamMode.Read);
A 00000008BE12  00000048DC12  0          //Write the decrypted file stream
A 00000008BE41  00000048DC41  0          FileStream fsOut = new FileStream(outputFile, FileMode.Create);
A 00000008BE8E  00000048DC8E  0          try
A 00000008BE9F  00000048DC9F  0          {
A 00000008BEAE  00000048DCAE  0              int data;
A 00000008BEC9  00000048DCC9  0              while ((data = cs.ReadByte()) != -1)
A 00000008BEFF  00000048DCFF  0              { fsOut.WriteByte((byte)data); }
A 00000008BF33  00000048DD33  0              //Close all the Writers
A 00000008BF5C  00000048DD5C  0              fsOut.Close();
A 00000008BF7C  00000048DD7C  0              cs.Close();
A 00000008BF99  00000048DD99  0              fsCrypt.Close();
A 00000008BFBD  00000048DDBD  0              //Delete the original file
A 00000008BFE9  00000048DDE9  0              File.Delete(inputFile);
A 00000008C012  00000048DE12  0              //Rename the encrypted file to that of the original
A 00000008C057  00000048DE57  0              File.Copy(outputFile, inputFile);
A 00000008C08A  00000048DE8A  0              File.Delete(outputFile);
A 00000008C0B4  00000048DEB4  0          }
A 00000008C0C3  00000048DEC3  0          catch (Exception ex)
A 00000008C0E5  00000048DEE5  0          {

A 00000008C0F4  00000048DEF4  0              throw ex;
A 00000008C10F  00000048DF0F  0          }
A 00000008C11E  00000048DF1E  0          finally
A 00000008C133  00000048DF33  0          {
A 00000008C142  00000048DF42  0              fsOut = null;
A 00000008C161  00000048DF61  0              cs = null;
A 00000008C17D  00000048DF7D  0              fsCrypt = null;
A 00000008C19E  00000048DF9E  0          }
A 00000008C1AD  00000048DFAD  0      }
A 00000008C1BA  00000048DFBA  0      private static byte[] GetBytes(string str)
A 00000008C1EE  00000048DFEE  0      {
A 00000008C1F9  00000048DFF9  0          byte[] bytes = new byte[str.Length * sizeof(char)];
A 00000008C23A  00000048E03A  0          System.Buffer.BlockCopy(str.ToCharArray(), 0, bytes, 0, bytes.Length);
A 00000008C28E  00000048E08E  0          return bytes;
A 00000008C2A9  00000048E0A9  0      }
A 00000008C2B6  00000048E0B6  0      private static void DecompressFileLZMA(string inFile, string outFile)
A 00000008C305  00000048E105  0      {
A 00000008C310  00000048E110  0          SevenZip.Compression.LZMA.Decoder coder = new SevenZip.Compression.LZMA.Decoder();
A 00000008C370  00000048E170  0          FileStream input = new FileStream(inFile, FileMode.Open);
A 00000008C3B7  00000048E1B7  0          FileStream output = new FileStream(outFile, FileMode.Create);
A 00000008C404  00000048E204  0          // Read the decoder properties
A 00000008C430  00000048E230  0          byte[] properties = new byte[5];
A 00000008C45E  00000048E25E  0          input.Read(properties, 0, 5);
A 00000008C48B  00000048E28B  0          // Read in the decompress file size.
A 00000008C4BD  00000048E2BD  0          byte[] fileLengthBytes = new byte[8];
A 00000008C4F0  00000048E2F0  0          input.Read(fileLengthBytes, 0, 8);
A 00000008C520  00000048E320  0          long fileLength = BitConverter.ToInt64(fileLengthBytes, 0);
A 00000008C56B  00000048E36B  0          coder.SetDecoderProperties(properties);
A 00000008C5A0  00000048E3A0  0          coder.Code(input, output, input.Length, fileLength, null);
A 00000008C5EA  00000048E3EA  0          //Cleanup
A 00000008C601  00000048E401  0          input.Close();
A 00000008C61D  00000048E41D  0          output.Flush();
A 00000008C63A  00000048E43A  0          output.Close();
A 00000008C657  00000048E457  0          coder = null;
A 00000008C672  00000048E472  0      }
A 00000008C67F  00000048E47F  0      #endregion
A 00000008C693  00000048E493  0    }
A 00000008C6F5  00000048E4F5  0  !This program cannot be run in DOS mode.
```

Figure 3.1.5.2 – Strings in WannaPeace.exe using BinText

```
A 0000000A3C5B  0000004A665B  0  <?xml version="1.0" encoding="utf-8"?>
A 0000000A3C83  0000004A6683  0  <assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
A 0000000A3CCE  0000004A66CE  0    <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
A 0000000A3D10  0000004A6710  0    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
A 0000000A3D48  0000004A6748  0      <security>
A 0000000A3D58  0000004A6758  0        <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
A 0000000A3D9E  0000004A679E  0          <!-- UAC Manifest Options
A 0000000A3DC1  0000004A67C1  0              If you want to change the Windows User Account Control level replace the
A 0000000A3E19  0000004A6819  0              requestedExecutionLevel node with one of the following.
A 0000000A3E61  0000004A6861  0          <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
A 0000000A3EAA  0000004A68AA  0          <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
A 0000000A3EFE  0000004A68FE  0          <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />
A 0000000A3F50  0000004A6950  0             Specifying requestedExecutionLevel element will disable file and registry virtualization.
A 0000000A3FB8  0000004A69B8  0             Remove this element if your application requires this virtualization for backwards
A 0000000A4018  0000004A6A18  0             compatibility.
A 0000000A4034  0000004A6A34  0          -->
A 0000000A4041  0000004A6A41  0          <requestedExecutionLevel level="requireAdministrator" uiAccess = "false" />
A 0000000A4096  0000004A6A96  0        </requestedPrivileges>
A 0000000A40B4  0000004A6AB4  0      </security>
A 0000000A40C5  0000004A6AC5  0    </trustInfo>
A 0000000A40D7  0000004A6AD7  0    <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
A 0000000A411D  0000004A6B1D  0      <application>
A 0000000A4130  0000004A6B30  0        <!-- A list of the Windows versions that this application has been tested on and is
A 0000000A418B  0000004A6B8B  0            is designed to work with. Uncomment the appropriate elements and Windows will
A 0000000A41E6  0000004A6BE6  0            automatically selected the most compatible environment. -->
A 0000000A4230  0000004A6C30  0        <!-- Windows Vista -->
A 0000000A424E  0000004A6C4E  0        <!--<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}" />-->
A 0000000A429A  0000004A6C9A  0        <!-- Windows 7 -->
A 0000000A42B4  0000004A6CB4  0        <!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}" />-->
A 0000000A4300  0000004A6D00  0        <!-- Windows 8 -->
A 0000000A431A  0000004A6D1A  0        <!--<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}" />-->
A 0000000A4366  0000004A6D66  0        <!-- Windows 8.1 -->
A 0000000A4382  0000004A6D82  0        <!--<supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d0da78}" />-->
A 0000000A43CE  0000004A6DCE  0        <!-- Windows 10 -->
A 0000000A43E9  0000004A6DE9  0        <!--<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />-->
A 0000000A4435  0000004A6E35  0      </application>
A 0000000A4449  0000004A6E49  0    </compatibility>
A 0000000A445F  0000004A6E5F  0    <!-- Indicates that the application is DPI-aware and will not be automatically scaled by Windows at higher
A 0000000A44CD  0000004A6ECD  0         DPIs. Windows Presentation Foundation (WPF) applications are automatically DPI-aware and do not need
A 0000000A453B  0000004A6F3B  0         to opt in. Windows Forms applications targeting .NET Framework 4.6 that opt into this setting, should
A 0000000A45AA  0000004A6FAA  0         also set the 'EnableWindowsFormsHighDpiAutoResizing' setting to 'true' in their app.config. -->
A 0000000A4612  0000004A7012  0    <!--
A 0000000A461A  0000004A701A  0    <application xmlns="urn:schemas-microsoft-com:asm.v3">
A 0000000A4654  0000004A7054  0      <windowsSettings>
A 0000000A466B  0000004A706B  0        <dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
A 0000000A46CA  0000004A70CA  0      </windowsSettings>
A 0000000A46E2  0000004A70E2  0    </application>

A 0000000A46F4  0000004A70F4  0    -->
A 0000000A46FD  0000004A70FD  0    <!-- Enable themes for Windows common controls and dialogs (Windows XP and later) -->
A 0000000A4756  0000004A7156  0    <!--
A 0000000A475E  0000004A715E  0    <dependency>
A 0000000A476E  0000004A716E  0      <dependentAssembly>
A 0000000A4787  0000004A7187  0        <assemblyIdentity
A 0000000A47A0  0000004A71A0  0            type="win32"
A 0000000A47B8  0000004A71B8  0            name="Microsoft.Windows.Common-Controls"
A 0000000A47EC  0000004A71EC  0            version="6.0.0.0"
A 0000000A4809  0000004A7209  0            processorArchitecture="*"
A 0000000A482E  0000004A722E  0            publicKeyToken="6595b64144ccf1df"
A 0000000A485B  0000004A725B  0            language="*"
A 0000000A4873  0000004A7273  0          />
A 0000000A487F  0000004A727F  0      </dependentAssembly>
A 0000000A4899  0000004A7299  0    </dependency>
A 0000000A48AA  0000004A72AA  0    -->
A 0000000A48B3  0000004A72B3  0  </assembly>
```

Figure 3.1.5.3 – Strings in WannaPeace.exe using BinText

These contains codes and steps to execute the encryption and lockdown of the files of the infected systems, rendering users to not be able to access their files or even their systems.

```
A 0000000251C5   000000426FC5   0   WSystem.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
A 000000025222   000000427022   0   &System.Windows.Forms.ImageListStreamer
A 0000000264AB   0000004282AB   0   QSystem.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
A 000000026503   000000428303   0   System.Drawing.Bitmap
A 0000000266F4   0000004284F4   0   QSystem.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
A 00000002674C   00000042854C   0   System.Drawing.Bitmap
```

Figure 3.1.5.4 – Strings in WannaPeace.exe using BinText

These contains public key tokens which potentially might be the tokens required to encrypt or decrypt the security placed by the attackers.

```
A 0000000A3C5B   0000004A665B   0   <?xml version="1.0" encoding="utf-8"?>
A 0000000A3C83   0000004A6683   0   <assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
A 0000000A3CCE   0000004A66CE   0     <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
A 0000000A3D10   0000004A6710   0     <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
A 0000000A3D48   0000004A6748   0       <security>
A 0000000A3D58   0000004A6758   0         <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
A 0000000A3D9E   0000004A679E   0           <!-- UAC Manifest Options
A 0000000A3DC1   0000004A67C1   0                If you want to change the Windows User Account Control level replace the
A 0000000A3E19   0000004A6819   0                requestedExecutionLevel node with one of the following.
A 0000000A3E61   0000004A6861   0           <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
A 0000000A3EAA   0000004A68AA   0           <requestedExecutionLevel  level="requireAdministrator" uiAccess="false" />
A 0000000A3EFE   0000004A68FE   0           <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />
A 0000000A3F50   0000004A6950   0              Specifying requestedExecutionLevel element will disable file and registry virtualization.
A 0000000A3FB8   0000004A69B8   0              Remove this element if your application requires this virtualization for backwards
A 0000000A4018   0000004A6A18   0              compatibility.
A 0000000A4034   0000004A6A34   0           -->
A 0000000A4041   0000004A6A41   0           <requestedExecutionLevel level="requireAdministrator" uiAccess = "false" />
A 0000000A4096   0000004A6A96   0         </requestedPrivileges>
A 0000000A40B4   0000004A6AB4   0       </security>
A 0000000A40C5   0000004A6AC5   0     </trustInfo>
A 0000000A40D7   0000004A6AD7   0     <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
A 0000000A411D   0000004A6B1D   0       <application>
A 0000000A4130   0000004A6B30   0         <!-- A list of the Windows versions that this application has been tested on and is
A 0000000A418B   0000004A6B8B   0             is designed to work with. Uncomment the appropriate elements and Windows will
A 0000000A41E6   0000004A6BE6   0             automatically selected the most compatible environment. -->
A 0000000A4230   0000004A6C30   0         <!-- Windows Vista -->
A 0000000A424E   0000004A6C4E   0         <!--<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}" />-->
A 0000000A429A   0000004A6C9A   0         <!-- Windows 7 -->
A 0000000A42B4   0000004A6CB4   0         <!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}" />-->
A 0000000A4300   0000004A6D00   0         <!-- Windows 8 -->
A 0000000A431A   0000004A6D1A   0         <!--<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}" />-->
A 0000000A4366   0000004A6D66   0         <!-- Windows 8.1 -->
A 0000000A4382   0000004A6D82   0         <!--<supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d0da78}" />-->
A 0000000A43CE   0000004A6DCE   0         <!-- Windows 10 -->
A 0000000A43E9   0000004A6DE9   0         <!--<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />-->
A 0000000A4435   0000004A6E35   0       </application>
A 0000000A4449   0000004A6E49   0     </compatibility>
A 0000000A445F   0000004A6E5F   0     <!-- Indicates that the application is DPI-aware and will not be automatically scaled by Windows at higher
A 0000000A44CD   0000004A6ECD   0          DPIs. Windows Presentation Foundation (WPF) applications are automatically DPI-aware and do not need
A 0000000A453B   0000004A6F3B   0          to opt in. Windows Forms applications targeting .NET Framework 4.6 that opt into this setting, should
A 0000000A45AA   0000004A6FAA   0          also set the 'EnableWindowsFormsHighDpiAutoResizing' setting to 'true' in their app.config. -->

A 0000000A4612   0000004A7012   0     <!--
A 0000000A461A   0000004A701A   0     <application xmlns="urn:schemas-microsoft-com:asm.v3">
A 0000000A4654   0000004A7054   0       <windowsSettings>
A 0000000A466B   0000004A706B   0         <dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
A 0000000A46CA   0000004A70CA   0       </windowsSettings>
A 0000000A46E2   0000004A70E2   0     </application>
A 0000000A46F4   0000004A70F4   0     -->
A 0000000A46FD   0000004A70FD   0     <!-- Enable themes for Windows common controls and dialogs (Windows XP and later) -->
A 0000000A4756   0000004A7156   0     <!--
A 0000000A475E   0000004A715E   0     <dependency>
A 0000000A476E   0000004A716E   0       <dependentAssembly>
A 0000000A4787   0000004A7187   0         <assemblyIdentity
A 0000000A47A0   0000004A71A0   0             type="win32"
A 0000000A47B8   0000004A71B8   0             name="Microsoft.Windows.Common-Controls"
A 0000000A47EC   0000004A71EC   0             version="6.0.0.0"
A 0000000A4809   0000004A7209   0             processorArchitecture="*"
A 0000000A482E   0000004A722E   0             publicKeyToken="6595b64144ccf1df"
A 0000000A485B   0000004A725B   0             language="*"
A 0000000A4873   0000004A7273   0           />
A 0000000A487F   0000004A727F   0       </dependentAssembly>
A 0000000A4899   0000004A7299   0     </dependency>
A 0000000A48AA   0000004A72AA   0     -->
A 0000000A48B3   0000004A72B3   0   </assembly>
```

Figure 3.1.5.5 – Strings in WannaPeace.exe using BinText

These contain instructions and dependencies for the malware to apply for the different versions of operating systems WannaPeace.exe was downloaded in.

A 0000000955CE 0000004973CE 0 Lzma.dll

Figure 3.1.5.6 – Strings in WannaPeace.exe using BinText

A 00000009F4A8 0000004A12A8 0 E:\Users\SCORPION\Downloads\privatelocker_version2\PrivateLocker\obj\Debug\RzW.pdb

Figure 3.1.5.7 – Strings in WannaPeace.exe using BinText

U 00000000AFBE 00000040CDBE 0 Ris3lTInGS@WannaPeace

Figure 3.1.5.8 – Strings in WannaPeace.exe using BinText

U 00000000BEA8 00000040DCA8 0 {0}\Locker.exe

Figure 3.1.5.9 – Strings in WannaPeace.exe using BinText

U 00000000BEE2 00000040DCE2 0 Lzma.dll

Figure 3.1.5.10 – Strings in WannaPeace.exe using BinText

U 0000000A3BA4 0000004A65A4 0 RzW.exe

Figure 3.1.5.11 – Strings in WannaPeace.exe using BinText

A 0000000084CC 00000040A2CC 0 System.Runtime.Versioning

Figure 3.1.5.12 – Strings in WannaPeace.exe using BinText

From Figure 3.1.5.6 to Figure 3.1.5.12, All these contains something to do with the process of installing and enabling the malware to run successfully.

A	000000007DC7	000000409BC7	0	pword
A	000000007DCD	000000409BCD	0	DecryptPassword
A	000000007DDD	000000409BDD	0	EncryptPassword
A	000000007DED	000000409BED	0	password
A	000000007DF6	000000409BF6	0	Replace
A	000000008AC8	00000040A8C8	0	EncryptedFolder
A	000000008AD8	00000040A8D8	0	SpecialFolder
A	000000008AE6	00000040A8E6	0	DecryptFolder
A	000000008AF4	00000040A8F4	0	EncryptFolder
A	000000008B02	00000040A902	0	sender
A	000000008B09	00000040A909	0	Decoder
A	000000008B11	00000040A911	0	Encoder
A	000000008B19	00000040A919	0	Buffer
A	000000008B20	00000040A920	0	get_ResourceManager
A	000000008D55	00000040AB55	0	CompilerError
A	000000008D63	00000040AB63	0	IEnumerator
A	000000008D6F	00000040AB6F	0	ManagementObjectEnumerator
A	000000008D8A	00000040AB8A	0	GetEnumerator
A	000000008D98	00000040AB98	0	.ctor
A	000000008D9E	00000040AB9E	0	.cctor
A	000000008DA5	00000040ABA5	0	CreateDecryptor
A	000000008DB5	00000040ABB5	0	CreateEncryptor
A	000000008DC9	00000040ABC9	0	PrivateLocker.standalone.cs
A	000000008DE5	00000040ABE5	0	System.Diagnostics
A	000000008DF8	00000040ABF8	0	DisableClickSounds
A	000000008E0B	00000040AC0B	0	System.Runtime.InteropServices
A	000000008E2A	00000040AC2A	0	System.Runtime.CompilerServices
A	000000008E4A	00000040AC4A	0	System.Resources
A	000000008E5B	00000040AC5B	0	get_EmbeddedResources
A	000000008E71	00000040AC71	0	WannaLocker.Form1.resources
A	000000008E8D	00000040AC8D	0	PrivateLocker.Properties.Resources.resources
A	000000008EBA	00000040ACBA	0	DebuggingModes
A	000000008EC9	00000040ACC9	0	get_Images
A	000000008ED4	00000040ACD4	0	get_ReferencedAssemblies
A	000000008EED	00000040ACED	0	GetDirectories
A	000000008EFC	00000040ACFC	0	PrivateLocker.Properties
A	000000008F15	00000040AD15	0	WriteCoderProperties
A	000000008F2A	00000040AD2A	0	SetDecoderProperties
A	000000008F3F	00000040AD3F	0	resourceFiles
A	000000008F4D	00000040AD4D	0	GetFiles
A	000000008F56	00000040AD56	0	files
A	000000008F5C	00000040AD5C	0	EnableVisualStyles
A	000000008F6F	00000040AD6F	0	GetManifestResourceNames
A	000000008F88	00000040AD88	0	PasswordDeriveBytes
A	000000008F9C	00000040AD9C	0	GetBytes
A	000000008FA5	00000040ADA5	0	dwFlags
A	000000008FAD	00000040ADAD	0	Settings

Figure 3.1.5.13 – Strings in WannaPeace.exe using BinText

These strings extracted contains the process of the malware obtaining access and setting a password to encrypt and decrypt the files. There is a command, Replace, on the 5th line, suggesting that there is something to do with replacing the password the user has set. The malware might also set up encrypted folders as suggested by the strings extracted. The malware could also tamper with the resources and services as shown by strings such as PrivateLocker.Properties.Resources.resources and WannaLocker.Form1.resources.

U	000000097C3E	000000499A3E	0	Average
U	000000098506	00000049A306	0	VS_VERSION_INFO
U	000000098562	00000049A362	0	VarFileInfo
U	000000098582	00000049A382	0	Translation
U	0000000985A6	00000049A3A6	0	StringFileInfo
U	0000000985CA	00000049A3CA	0	000004b0
U	0000000985E2	00000049A3E2	0	CompanyName
U	0000000985FC	00000049A3FC	0	Igor Pavlov
U	00000009861A	00000049A41A	0	FileDescription
U	00000009863C	00000049A43C	0	LZMA#
U	00000009864E	00000049A44E	0	FileVersion
U	000000098668	00000049A468	0	4.12.4863.12691
U	00000009868E	00000049A48E	0	InternalName
U	0000000986A8	00000049A4A8	0	Lzma.dll
U	0000000986C2	00000049A4C2	0	LegalCopyright
U	0000000986E0	00000049A4E0	0	Copyright @ Igor Pavlov 1999-2004
U	00000009872A	00000049A52A	0	OriginalFilename
U	00000009874C	00000049A54C	0	Lzma.dll
U	000000098766	00000049A566	0	ProductName
U	000000098780	00000049A580	0	LZMA# SDK
U	00000009879A	00000049A59A	0	ProductVersion
U	0000000987B8	00000049A5B8	0	4.12.4863.12691
U	0000000987DE	00000049A5DE	0	Assembly Version
U	000000098800	00000049A600	0	4.12.4863.12691
U	0000000A3962	0000004A6362	0	VS_VERSION_INFO
U	0000000A39BE	0000004A63BE	0	VarFileInfo
U	0000000A39DE	0000004A63DE	0	Translation
U	0000000A3A02	0000004A6402	0	StringFileInfo
U	0000000A3A26	0000004A6426	0	000004b0
U	0000000A3A3E	0000004A643E	0	Comments
U	0000000A3A5A	0000004A645A	0	CompanyName
U	0000000A3A7E	0000004A647E	0	FileDescription
U	0000000A3AAE	0000004A64AE	0	FileVersion
U	0000000A3AC8	0000004A64C8	0	1.0.0.0
U	0000000A3ADE	0000004A64DE	0	InternalName
U	0000000A3AF8	0000004A64F8	0	RzW.exe
U	0000000A3B0E	0000004A650E	0	LegalCopyright
U	0000000A3B42	0000004A6542	0	2017
U	0000000A3B56	0000004A6556	0	LegalTrademarks
U	0000000A3B82	0000004A6582	0	OriginalFilename
U	0000000A3BA4	0000004A65A4	0	RzW.exe
U	0000000A3BBA	0000004A65BA	0	ProductName
U	0000000A3BE2	0000004A65E2	0	ProductVersion
U	0000000A3C00	0000004A6600	0	1.0.0.0
U	0000000A3C16	0000004A6616	0	Assembly Version
U	0000000A3C38	0000004A6638	0	1.0.0.0

Figure 3.1.5.14 – Strings in WannaPeace.exe using BinText

A large number of the string extracted seems to revolve around the installation process of the malware onto the system. These are some of them just for reference, things such as LegalCopyright and LegalTrademarks suggests that this malware is a dummy and malicious software and not the legitimate software users thought it was. This will have the users not second guess or doubt whether the software is real or fake.

3.1.6. Basic Static Analysis with PeStudio

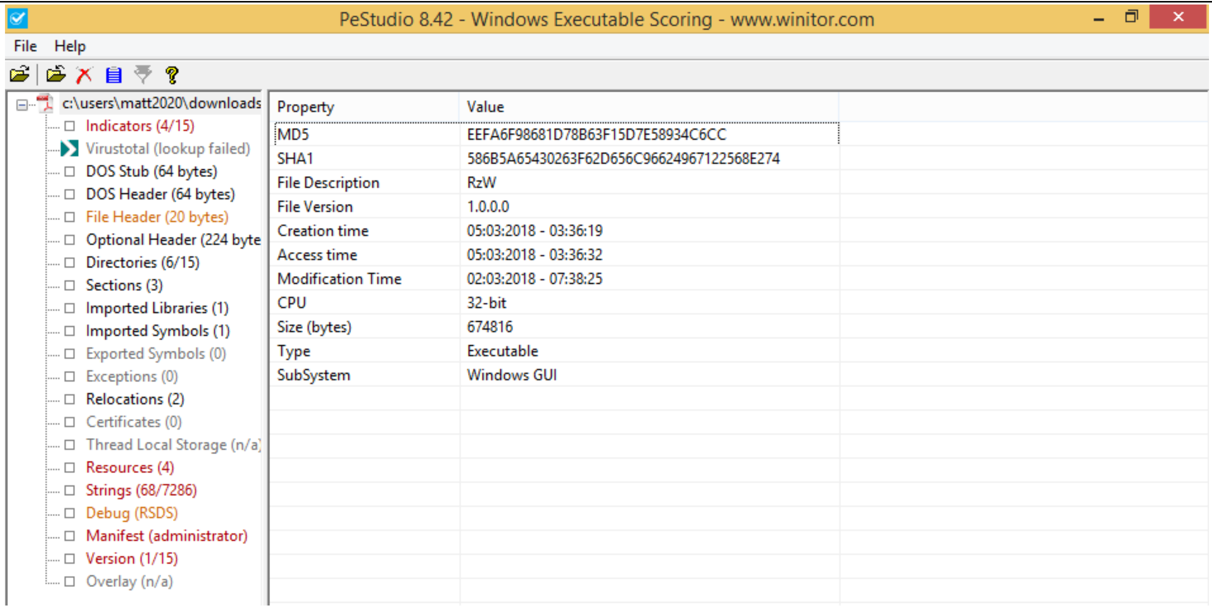


Figure 3.1.6.1 – General Information of WannaPeace.exe using Pestudio

PeStudio is used to perform a general overview of malware analysis. It provides a huge variety of information of malware including the strings extracted and resources imported by the software, just to name a few. But, as specialized tools were used in the sections above, there were no new information gained when using this tool as it was covered by all the other tools.

3.1.7. Basic Static Analysis Conclusion

After performing basic static analysis, I found that the malware did not have any obfuscation techniques used such as packing. I also found additional information such as the dates and time the malware was created and identified the DLLs and import functions of the malware, which are malicious and could be potentially very lethal to the system it infects. By analyzing the imported strings and DLLs, I thought that this could be a ransomware due to all the encryption and decryption of files and folders indicated on the strings. Therefore, with this knowledge in mind, more attention would be given to the malware when it comes to sections or suspicious codes which could confirm the true nature of WannaPeace.exe.

3.2. Basic Dynamic Analysis

Basic Dynamic Analysis would allow the malware to be executed in a safe and controlled environment where all the actions are monitored. It allows analyst to fully understand the functionalities of the malware in this manner as Basic Static Analysis only allowed us to gather general information about the malware and was not able to get a definitive conclusion of the malware.

3.2.1. Applications for Basic Dynamic Analysis

Before executing the malware there are a few software which should be opened beforehand to properly analyze the changes the malware will do to the isolated environment.

3.2.1.1. Regshot

Regshot is necessary because it temporarily saves the state of the Windows machine, which can then be compared with the state of the machine after the malware has been executed. This will allow us, the analyst, to view the changes in a clear and simple manner. Click on "1st shot' located on the top right corner as shown in the image below to capture a temporary image of the current state of the machine to be compared to the "2nd shot", which will only be clicked when the malware has already been executed.

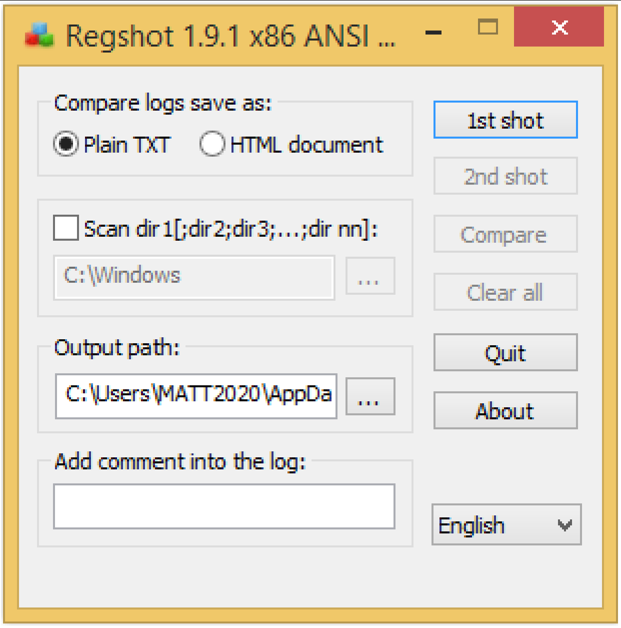


Figure 3.2.1.1 – User Interface of BinText

3.2.1.2. Process Monitor

Process Monitor, as the name implies, monitors the processes within the device. It monitors the applications and background processes, keeping track of information such as Process Name, Process ID (PID), Operation, Path, Result, Detail and Time. This tool can be useful as it will display new processes when the malware is executed, allowing the analyst to isolate and focus on the newly executed malware.

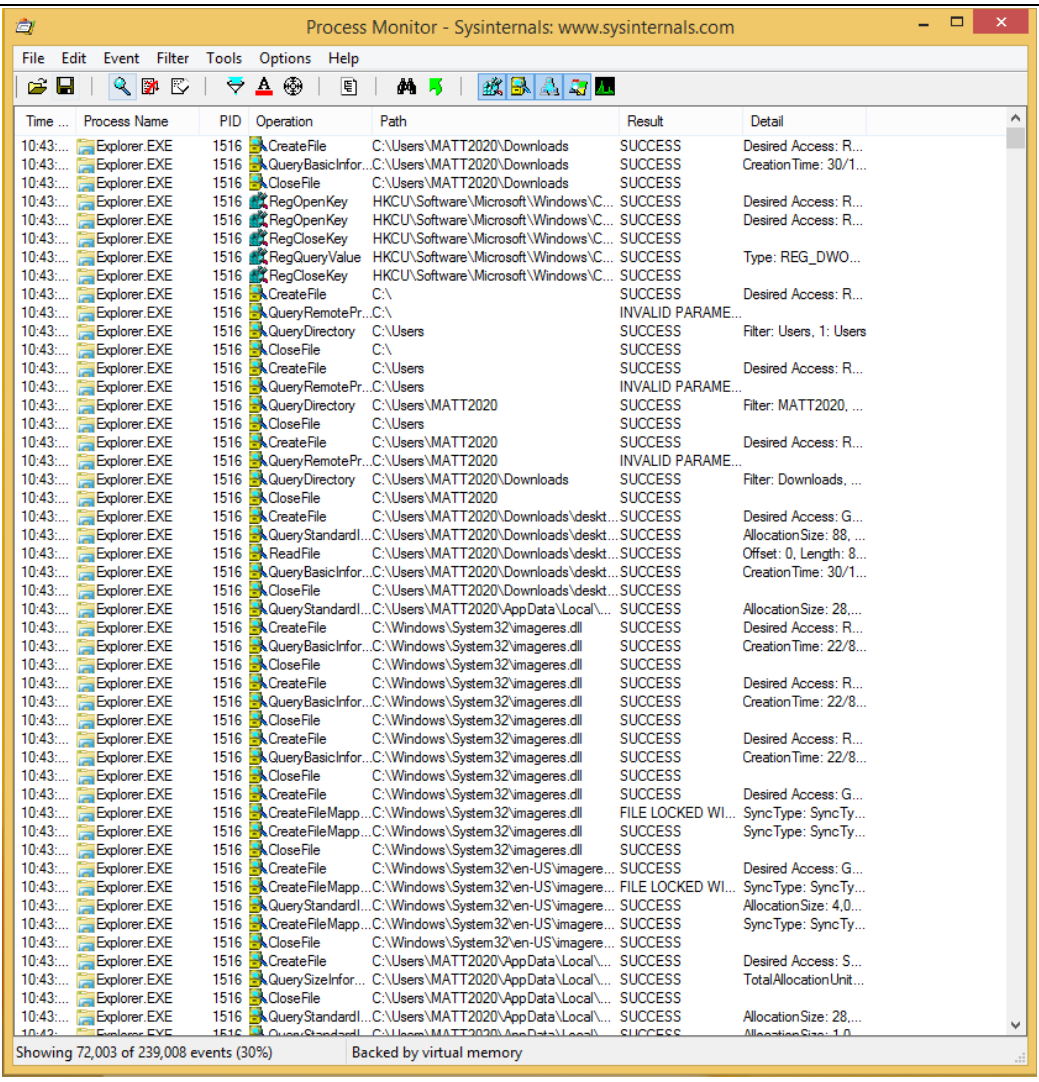
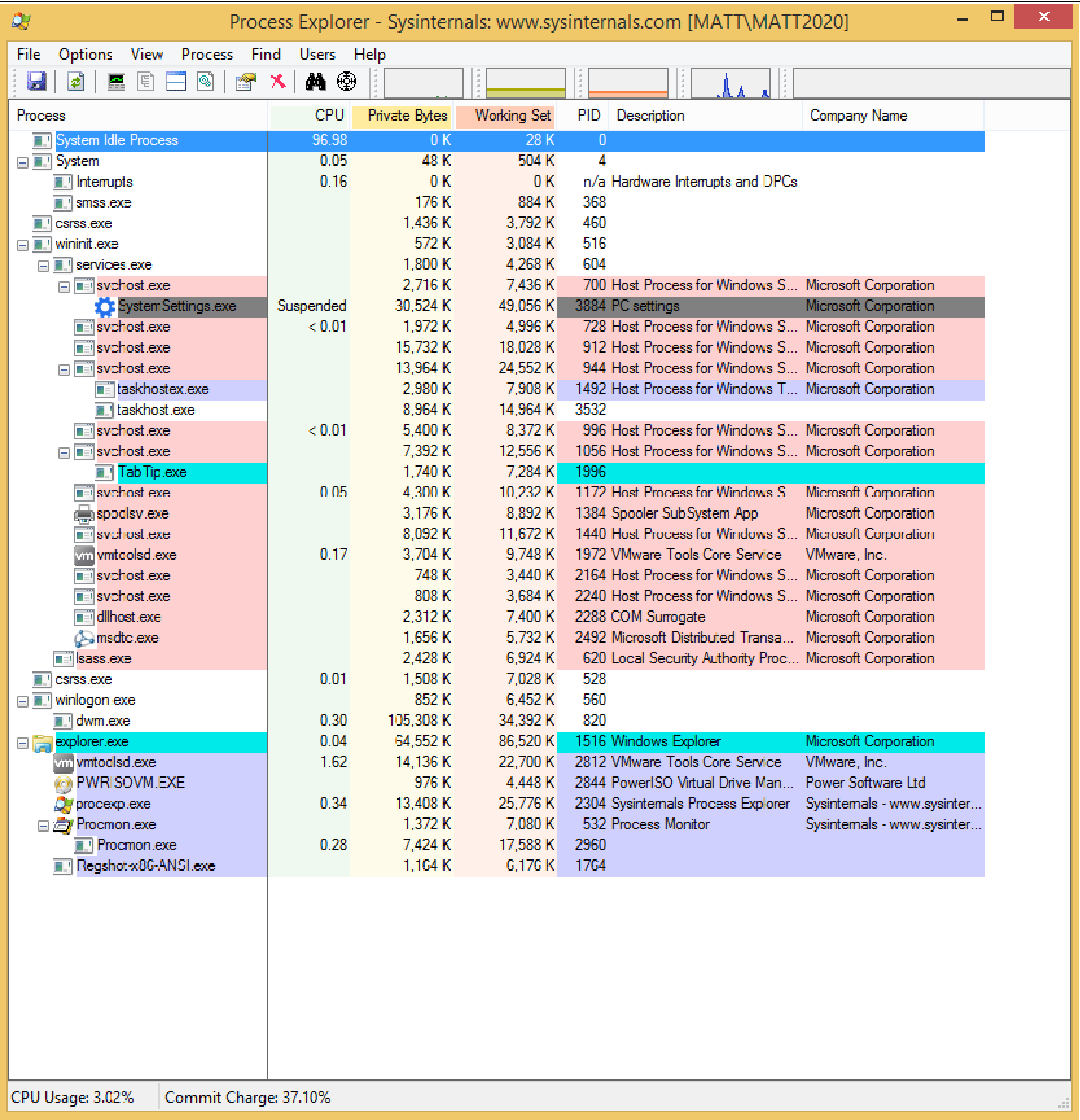


Figure 3.2.1.2 – User Interface of Process Monitor

3.2.1.3. Process Explorer

Process Explorer is another application similar to Process Monitor. It also helps the analyst view the newly executed process much easier as there are lesser number of process taking place in the user interface as compared to Process Monitor. It is able to keep track of details such as Process, CPU, Private Bytes, Working Set, PID, Description of Process, Company Name. Like Process Monitor, this tool is powerful and useful to the analyst, whereby the newly executed malware can be isolated and focused on.



Process Explorer - Sysinternals: www.sysinternals.com [MATT\MATT2020]

File Options View Process Find Users Help

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
System Idle Process	96.98	0 K	28 K	0		
System	0.05	48 K	504 K	4		
Interrupts	0.16	0 K	0 K	n/a	Hardware Interrupts and DPCs	
smss.exe		176 K	884 K	368		
csrss.exe		1,436 K	3,792 K	460		
wininit.exe		572 K	3,084 K	516		
services.exe		1,800 K	4,268 K	604		
svchost.exe		2,716 K	7,436 K	700	Host Process for Windows S...	Microsoft Corporation
SystemSettings.exe	Suspended	30,524 K	49,056 K	3884	PC settings	Microsoft Corporation
svchost.exe	< 0.01	1,972 K	4,996 K	728	Host Process for Windows S...	Microsoft Corporation
svchost.exe		15,732 K	18,028 K	912	Host Process for Windows S...	Microsoft Corporation
svchost.exe		13,964 K	24,552 K	944	Host Process for Windows S...	Microsoft Corporation
taskhostex.exe		2,980 K	7,908 K	1492	Host Process for Windows T...	Microsoft Corporation
taskhost.exe		8,964 K	14,964 K	3532		
svchost.exe	< 0.01	5,400 K	8,372 K	996	Host Process for Windows S...	Microsoft Corporation
svchost.exe		7,392 K	12,556 K	1056	Host Process for Windows S...	Microsoft Corporation
TabTip.exe		1,740 K	7,284 K	1996		
svchost.exe	0.05	4,300 K	10,232 K	1172	Host Process for Windows S...	Microsoft Corporation
spoolsv.exe		3,176 K	8,892 K	1384	Spooler SubSystem App	Microsoft Corporation
svchost.exe		8,092 K	11,672 K	1440	Host Process for Windows S...	Microsoft Corporation
vmtoolsd.exe	0.17	3,704 K	9,748 K	1972	VMware Tools Core Service	VMware, Inc.
svchost.exe		748 K	3,440 K	2164	Host Process for Windows S...	Microsoft Corporation
svchost.exe		808 K	3,684 K	2240	Host Process for Windows S...	Microsoft Corporation
dllhost.exe		2,312 K	7,400 K	2288	COM Surrogate	Microsoft Corporation
msdtc.exe		1,656 K	5,732 K	2492	Microsoft Distributed Transa...	Microsoft Corporation
lsass.exe		2,428 K	6,924 K	620	Local Security Authority Proc...	Microsoft Corporation
csrss.exe	0.01	1,508 K	7,028 K	528		
winlogon.exe		852 K	6,452 K	560		
dwm.exe	0.30	105,308 K	34,392 K	820		
explorer.exe	0.04	64,552 K	86,520 K	1516	Windows Explorer	Microsoft Corporation
vmtoolsd.exe	1.62	14,136 K	22,700 K	2812	VMware Tools Core Service	VMware, Inc.
PWRISOVM.EXE		976 K	4,448 K	2844	PowerISO Virtual Drive Man...	Power Software Ltd
procexp.exe	0.34	13,408 K	25,776 K	2304	Sysinternals Process Explorer	Sysinternals - www.sysinter...
Procmon.exe		1,372 K	7,080 K	532	Process Monitor	Sysinternals - www.sysinter...
Procmon.exe	0.28	7,424 K	17,588 K	2960		
Regshot-x86-ANSI.exe		1,164 K	6,176 K	1764		

CPU Usage: 3.02% Commit Charge: 37.10%

Figure 3.2.1.3 – User Interface of Process Explorer

3.2.1.4. ApateDNS

This is a tool for controlling DNS responses through a simple GUI. Since it is a phony DNS server, ApateDNS spoofs DNS responses to a user-specific IP address by listening on UDP port 53 on the local machine. This is used to check if the malicious malware executed requires the use of network to complete its execution. The target for ApateDNS is the loopback address (127.0.0.1) of the machine to obtain responses. Once the target is DNS in inserted in the "DNS Reply IP" portion of the user interface, click on "Start Server" to start the process.

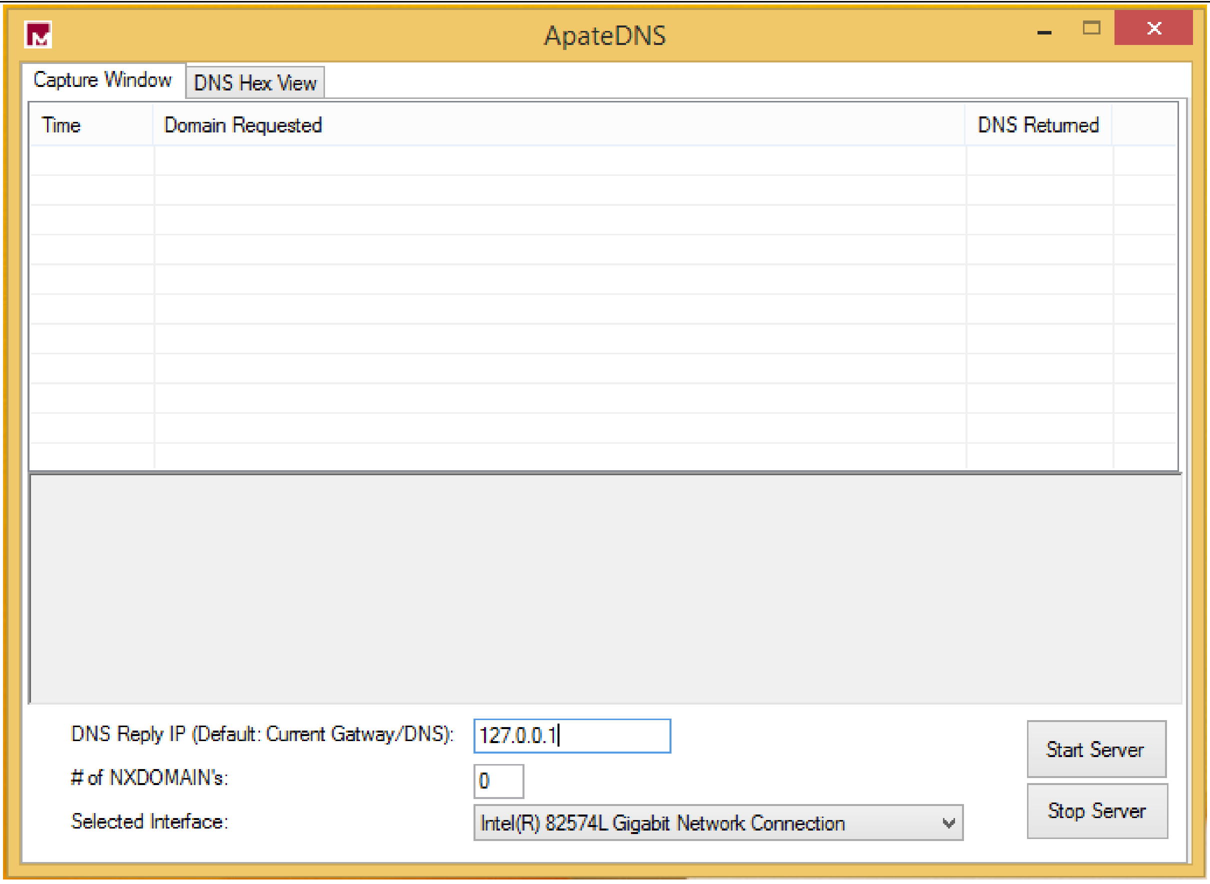


Figure 3.2.1.4 – User Interface of ApateDNS

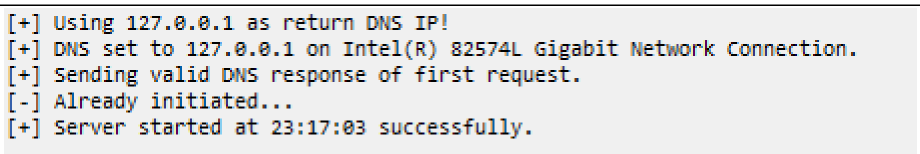


Figure 3.2.1.5 – User Interface of ApateDNS

There should be a response similar to this when the server has started.

3.2.2. WannaPeace.exe Execution

Upon the execution of WannaPeace.exe process, the malware will display this Adobe Reader popup window. It won't ask the user to click on any button. The text at the bottom of the window translates to: Opening the document failed!!! Try again.

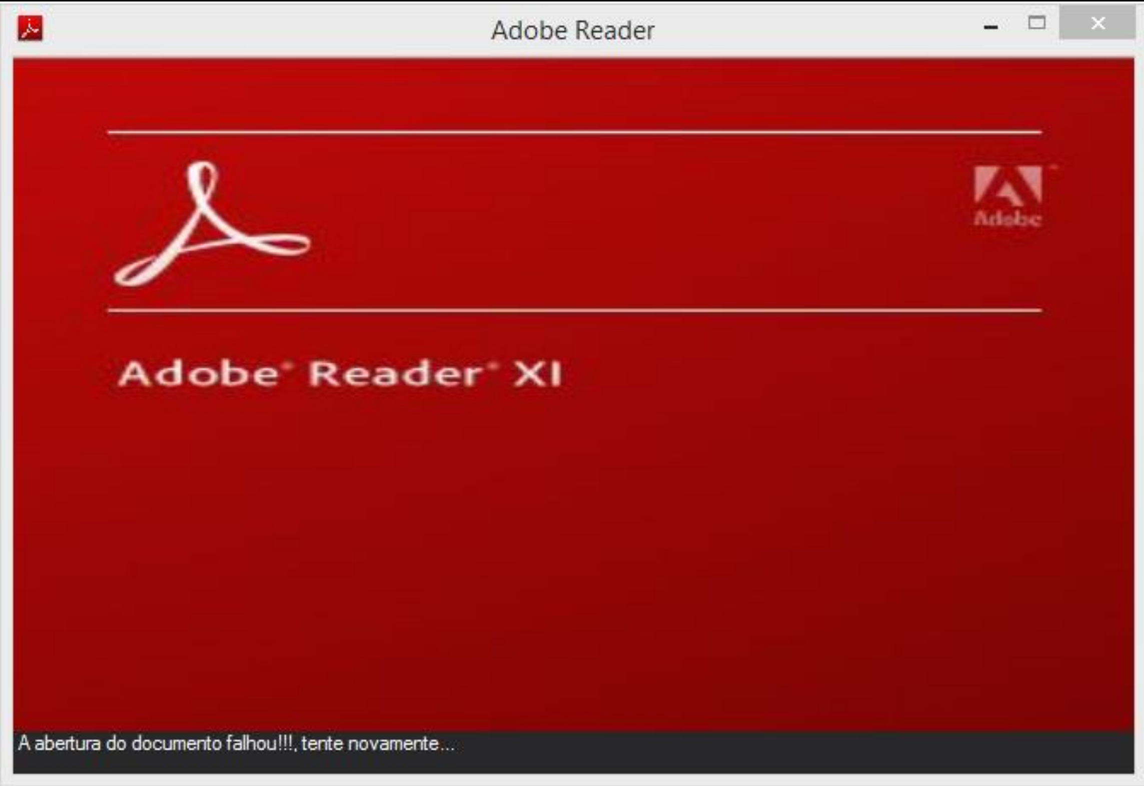


Figure 3.2.2.1 – Adobe Reader pop-up when WannaPeace.exe was executed

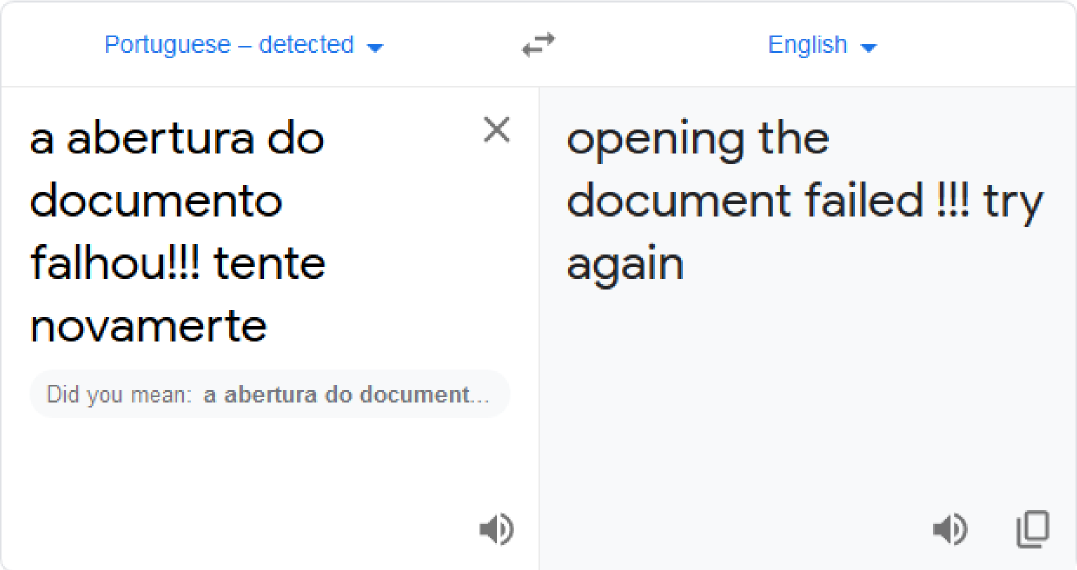


Figure 3.2.2.2 – Google Translate of the text of Adobe Reader

This error alert message will then pop-up after around 7 seconds of the initial execution of WannaPeace.exe. This alert indicates to the user about the error detected as it could not load file or assembly or one of the dependencies.

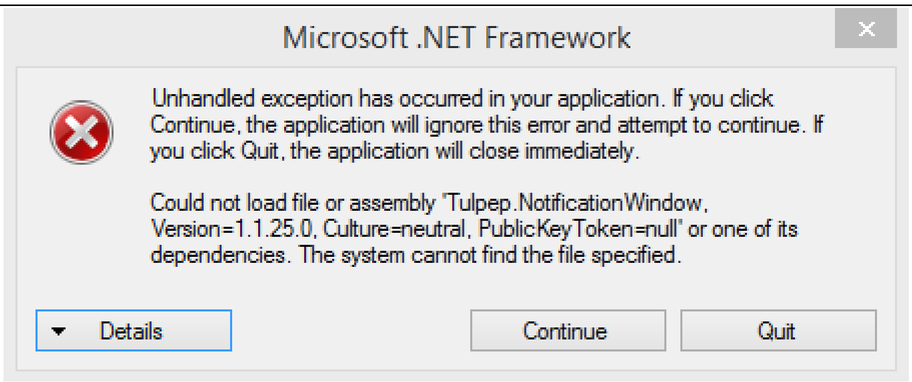


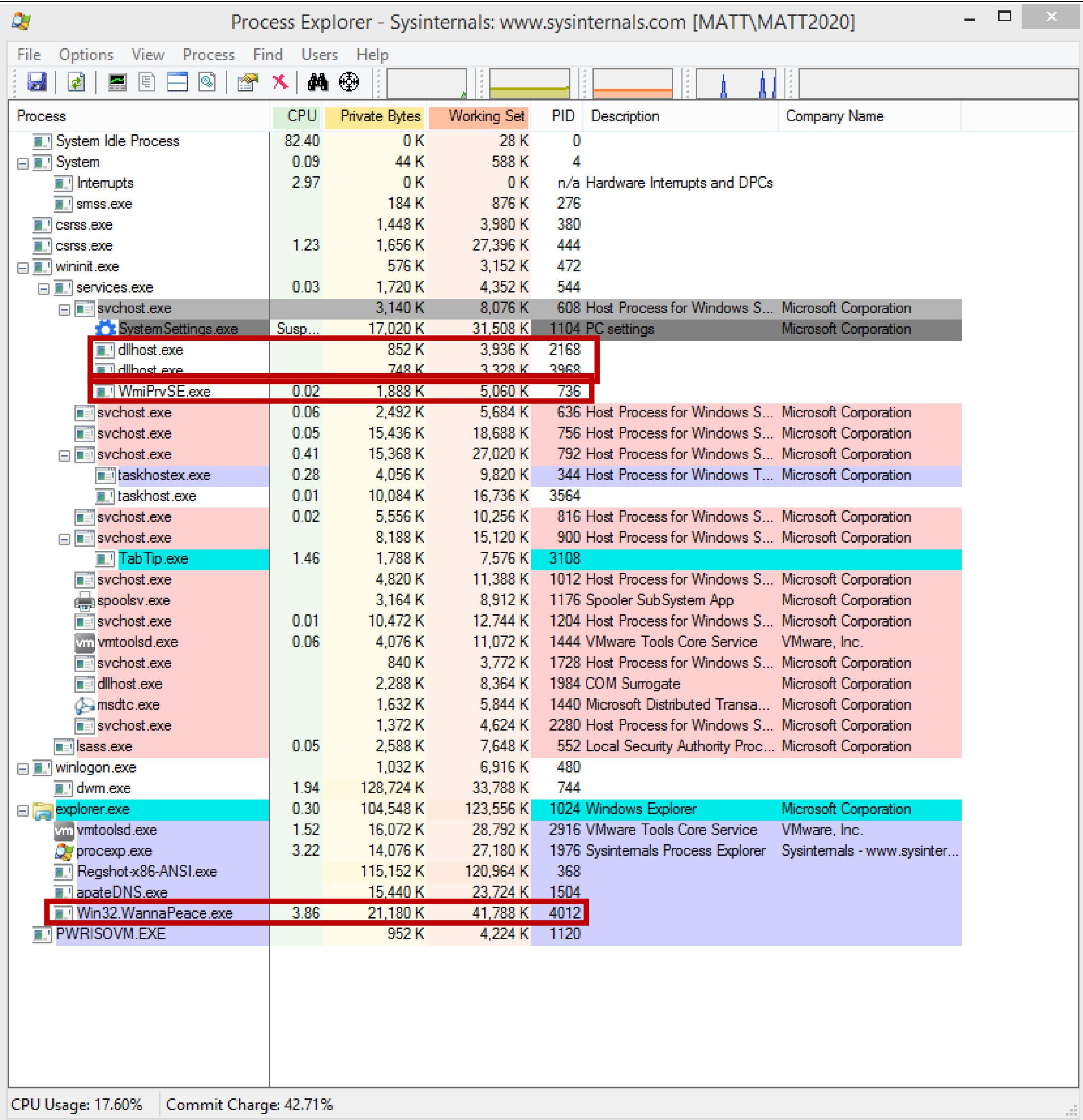
Figure 3.2.2.3 – Windows Pop-up error message

3.2.2.1. Process Explorer

Looking at Process Explorer when the malware is executed, here are the list of processes detected by Process Explorer.

Process ID (PID)	**Process Name**
2168	dllhost.exe
3968	dllhost.exe
4012	Win32.WannaPeace.exe

These are the images showing the processes executed as soon as WannaPeace.exe was executed. This is similar to the table as indicated above.



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
System Idle Process	82.40	0 K	28 K	0		
System	0.09	44 K	588 K	4		
Interrupts	2.97	0 K	0 K	n/a	Hardware Interrupts and DPCs	
smss.exe		184 K	876 K	276		
csrss.exe		1,448 K	3,980 K	380		
csrss.exe	1.23	1,656 K	27,396 K	444		
wininit.exe		576 K	3,152 K	472		
services.exe	0.03	1,720 K	4,352 K	544		
svchost.exe		3,140 K	8,076 K	608	Host Process for Windows S...	Microsoft Corporation
SystemSettings.exe	Susp...	17,020 K	31,508 K	1104	PC settings	Microsoft Corporation
dllhost.exe		852 K	3,936 K	2168		
dllhost.exe		748 K	3,328 K	3968		
WmiPrvSE.exe	0.02	1,888 K	5,060 K	736		
svchost.exe	0.06	2,492 K	5,684 K	636	Host Process for Windows S...	Microsoft Corporation
svchost.exe	0.05	15,436 K	18,688 K	756	Host Process for Windows S...	Microsoft Corporation
svchost.exe	0.41	15,368 K	27,020 K	792	Host Process for Windows S...	Microsoft Corporation
taskhostex.exe	0.28	4,056 K	9,820 K	344	Host Process for Windows T...	Microsoft Corporation
taskhost.exe	0.01	10,084 K	16,736 K	3564		
svchost.exe	0.02	5,556 K	10,256 K	816	Host Process for Windows S...	Microsoft Corporation
svchost.exe		8,188 K	15,120 K	900	Host Process for Windows S...	Microsoft Corporation
TabTip.exe	1.46	1,788 K	7,576 K	3108		
svchost.exe		4,820 K	11,388 K	1012	Host Process for Windows S...	Microsoft Corporation
spoolsv.exe		3,164 K	8,912 K	1176	Spooler SubSystem App	Microsoft Corporation
svchost.exe	0.01	10,472 K	12,744 K	1204	Host Process for Windows S...	Microsoft Corporation
vmtoolsd.exe	0.06	4,076 K	11,072 K	1444	VMware Tools Core Service	VMware, Inc.
svchost.exe		840 K	3,772 K	1728	Host Process for Windows S...	Microsoft Corporation
dllhost.exe		2,288 K	8,364 K	1984	COM Surrogate	Microsoft Corporation
msdtc.exe		1,632 K	5,844 K	1440	Microsoft Distributed Transa...	Microsoft Corporation
svchost.exe		1,372 K	4,624 K	2280	Host Process for Windows S...	Microsoft Corporation
lsass.exe	0.05	2,588 K	7,648 K	552	Local Security Authority Proc...	Microsoft Corporation
winlogon.exe		1,032 K	6,916 K	480		
dwm.exe	1.94	128,724 K	33,788 K	744		
explorer.exe	0.30	104,548 K	123,556 K	1024	Windows Explorer	Microsoft Corporation
vmtoolsd.exe	1.52	16,072 K	28,792 K	2916	VMware Tools Core Service	VMware, Inc.
procexp.exe	3.22	14,076 K	27,180 K	1976	Sysinternals Process Explorer	Sysinternals - www.sysinter...
Regshot-x86-ANSI.exe		115,152 K	120,964 K	368		
apateDNS.exe		15,440 K	23,724 K	1504		
Win32.WannaPeace.exe	3.86	21,180 K	41,788 K	4012		
PWRISOVM.EXE		952 K	4,224 K	1120		

Figure 3.2.2.1.1 – Highlight of executed process from WannaPeace.exe on Process Explorer

Even if the windows were closed by clicking the "x" button on the top right of the Adobe Reader window, the suspicious processes will still remain running in the background.

When the malicious processes indicated above was double-clicked, this pop-up will open. This immediately shows an anomaly as there is no path and no directory, whereas other processes have directories as shown in the images below in Figure 3.2.2.1.4. This issue is consistent with all the malicious processes.

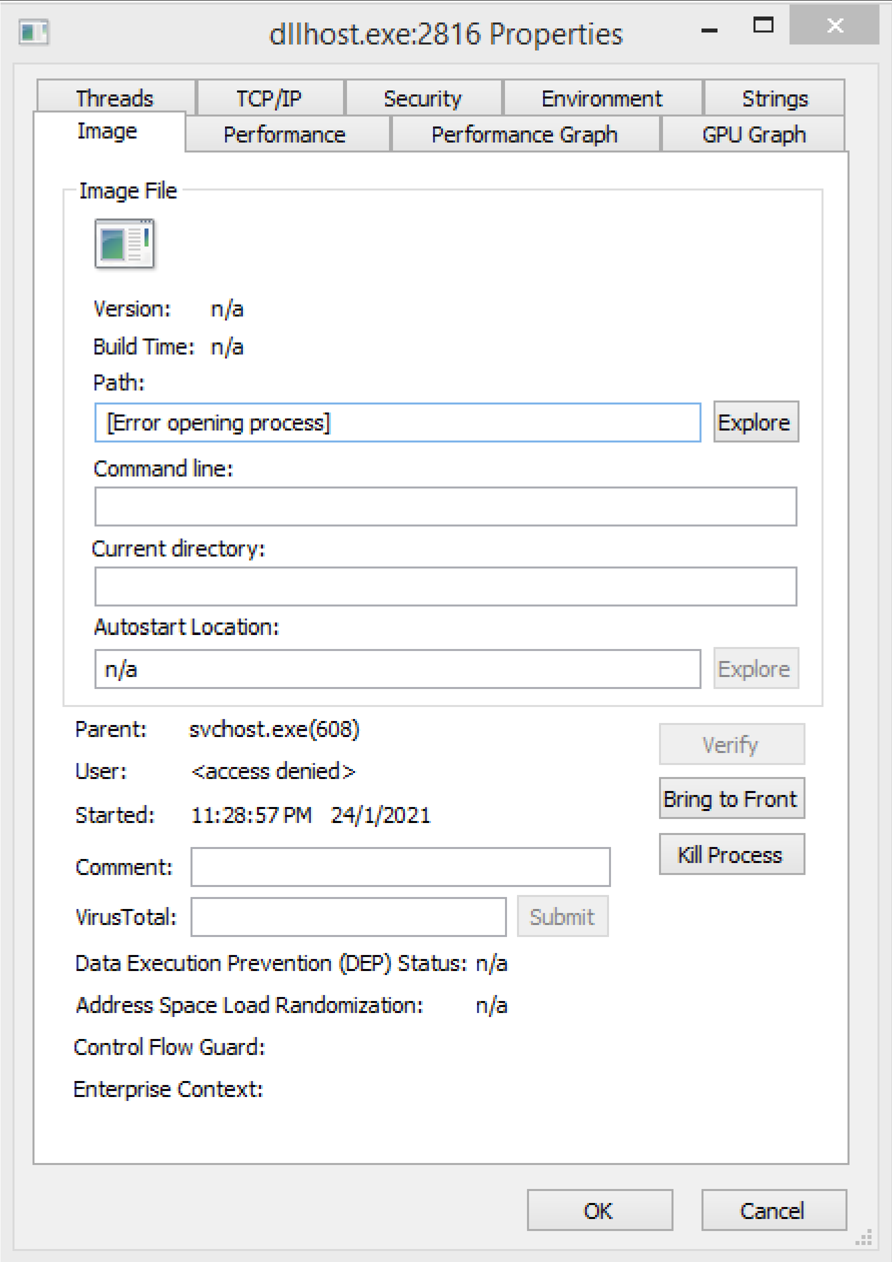


Figure 3.2.2.1.3 – dllhost.exe:2816 malicious process properties

This is the example of a typical and non-malicious process.

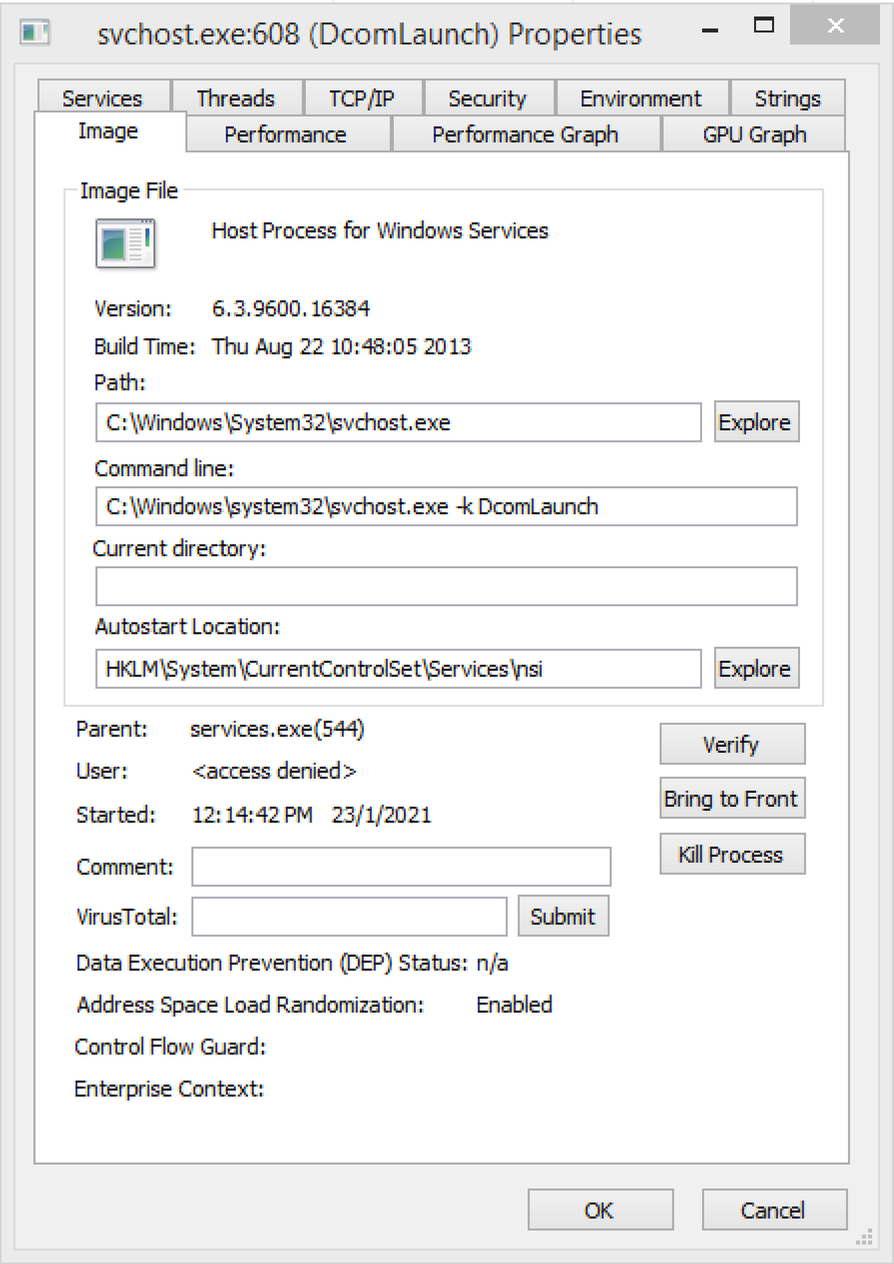


Figure 3.2.2.1.4 – svchost.exe:608 typical process properties

3.2.2.2. Regshot 1.9.1

Checking up again on Regshot, and after taking the second shot since the malware has been executed. When the "Compare" button was pressed and it would usually show the screen which shows the changes made to the system. Initially, Regshot 1.9.0, which was the default downloaded version of Regshot in the VM, was used to capture the shots and compare the results but did not display anything except an error message as shown in Figure 3.2.2.2.1.

When searching online for the solution, it was discovered that there is Regshot 1.9.1 beta and it should solve the issues, according to online forums. I attempted it and fortunately it worked. The results of the 1st shot, 2nd shot, and comparison can be seen in Figure 3.2.2.2.2, Figure 3.2.2.2.3 and Figure 3.2.2.2.4.

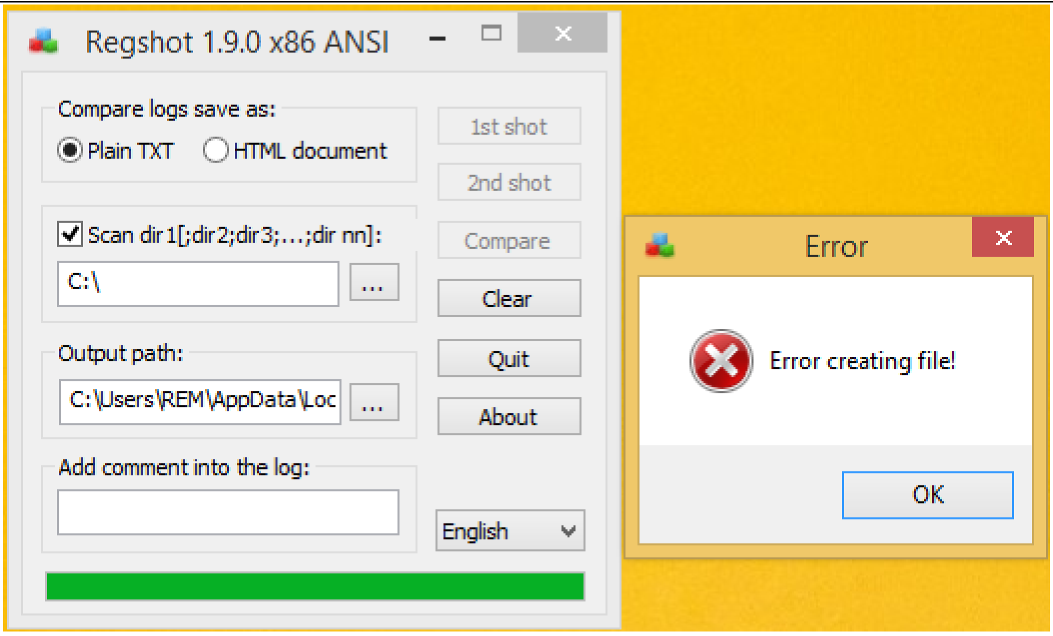


Figure 3.2.2.2.1 – Regshot 1.9.0 comparison error pop-up

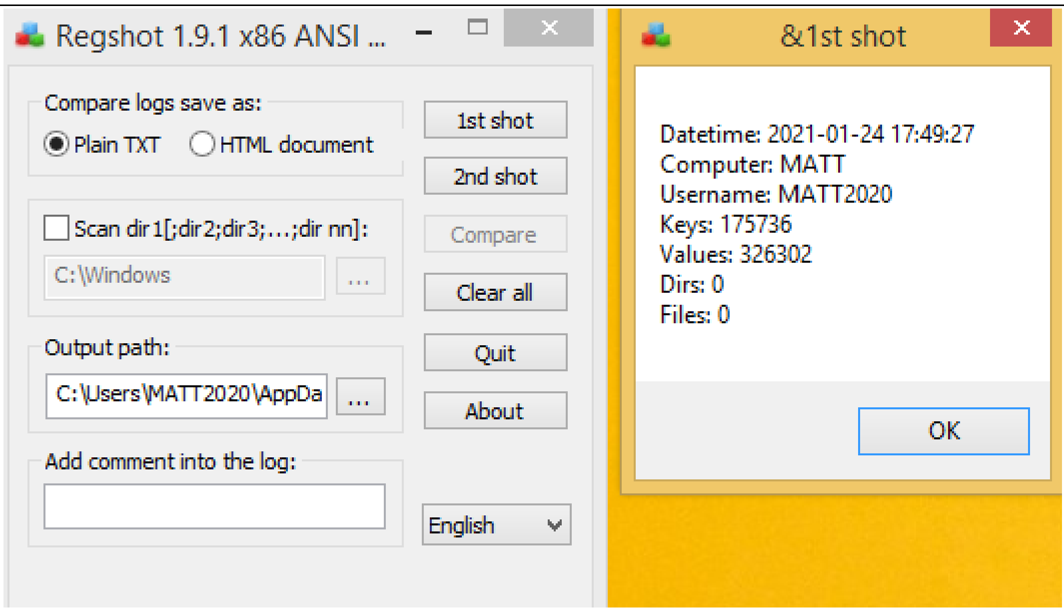


Figure 3.2.2.2.2 – Regshot 1.9.1 1st shot success pop-up

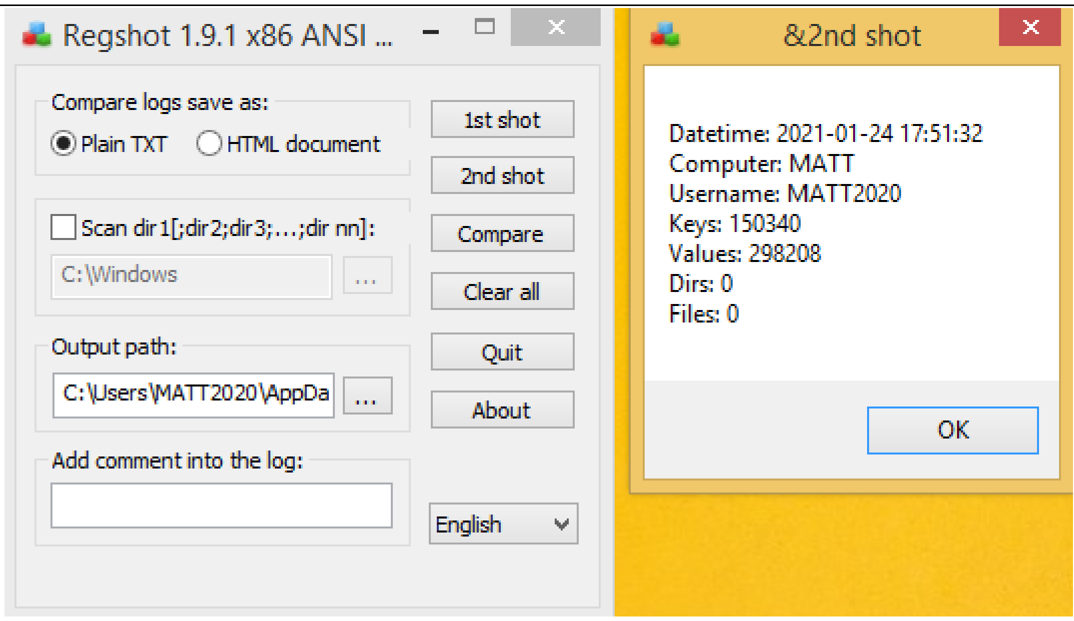


Figure 3.2.2.2.3 – Regshot 1.9.1 2nd shot success pop-up

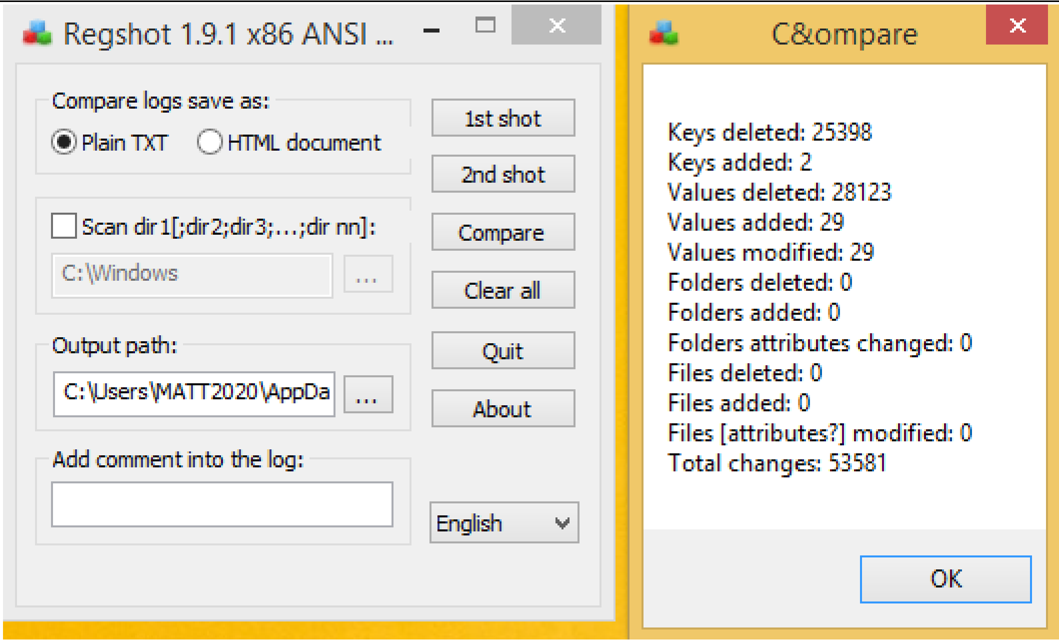


Figure 3.2.2.2.4 – Regshot 1.9.1 Comparison success pop-up

The Figures 3.2.2.2.5 and 3.2.2.2.6 shows the summary of the changes made by WannaPeace.exe on the isolated Windows system. A total of 25398 keys were deleted and a total of 53581 changes were made.

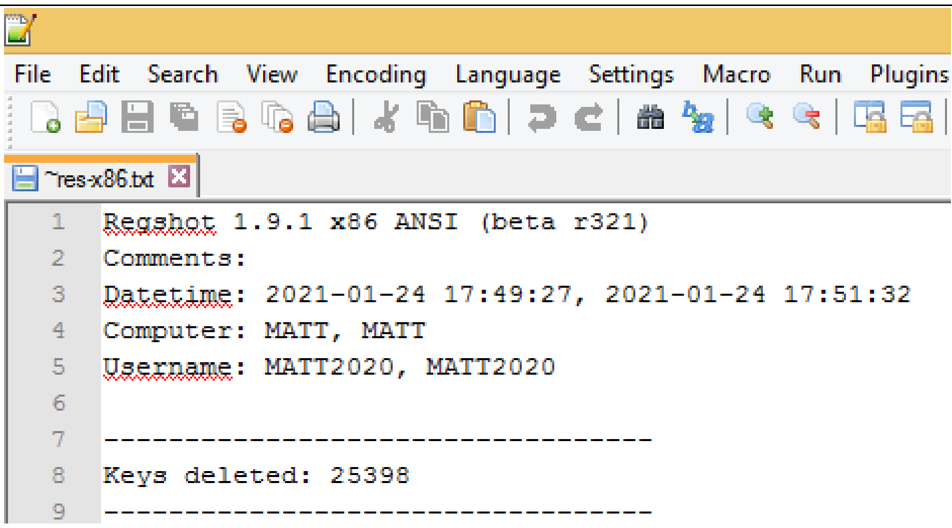


Figure 3.2.2.2.5 – Regshot Comparison: summary

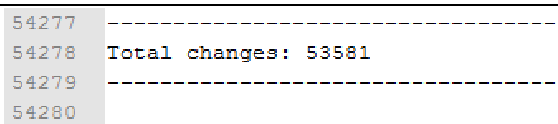


Figure 3.2.2.2.6 – Regshot Comparison: summary

3.2.2.3. Process Monitor

Now, looking at Process Monitor, the PID of Win32.WannaPeace.exe is different because I reset the VM to a clean snapshot to try again. Within Process Monitor, there are a lot of details, however, there are a few processes I'd like to highlight.

Since the suspicions after Basic Static Analysis is that this could be a ransomware, it should have some sort of network connections with the attacker. Hence, I checked the TCP Reconnect operation on PID 3760 and discovered 2 of such operations running on port 49217 using the http protocol.

Process

File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
2:19:5...	Win32.Wanna...	3760	TCP Reconnect	MATT:49217 -> MATT:http	SUCCESS	Length: 0, seqnum: 0, connid: 0
2:19:5...	Win32.Wanna...	3760	TCP Reconnect	MATT:49217 -> MATT:http	SUCCESS	Length: 0, seqnum: 0, connid: 0

Figure 3.2.2.3.1 – Process Monitor : TCP Reconnect operation on PID 3760

The malware also changes registry of Protocol Defaults. This image below shows the malware changing the details of the protocol defaults in order for it to properly execute.

2:19:5...	Win32.Wanna...	3760	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults\	SUCCESS	Desired Access: Read
2:19:5...	Win32.Wanna...	3760	RegQueryKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Query: Cached, SubKeys: 0, Values: 8
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 0, Name: , Type: REG_SZ, Le...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 1, Name: http, Type: REG_D...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 2, Name: https, Type: REG_D...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 3, Name: ftp, Type: REG_DW...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 4, Name: file, Type: REG_DW...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 5, Name: @ivt, Type: REG_D...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 6, Name: shell, Type: REG_D...
2:19:5...	Win32.Wanna...	3760	RegEnumValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	Index: 7, Name: knownfolder, Type: ...
2:19:5...	Win32.Wanna...	3760	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProtocolDefaults	SUCCESS	

Figure 3.2.2.3.2 – Process Monitor : Protocol Defaults on PID 3760

These operations shown below restricts users from downloading files. This is where WannaPeace.exe controls the registry settings and disables features such as Feature Control File Download to make this restriction possible.

2:19:5...	Win32.Wanna...	3760	RegOpenKey	HKLM\SOFTWARE\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_RESTRICT_FILEDOWNLOAD	SUCCESS	Desired Access: Query Value
2:19:5...	Win32.Wanna...	3760	RegQueryValue	HKLM\SOFTWARE\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_RESTRICT_FILEDOWNLOAD\Win32.WannaPeace.exe	NAME NOT FOUND	Length: 144
2:19:5...	Win32.Wanna...	3760	RegQueryValue	HKLM\SOFTWARE\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_RESTRICT_FILEDOWNLOAD*	NAME NOT FOUND	Length: 144
2:19:5...	Win32.Wanna...	3760	RegCloseKey	HKLM\SOFTWARE\Microsoft\Internet Explorer\Main\FeatureControl\FEATURE_RESTRICT_FILEDOWNLOAD	SUCCESS	

Figure 3.2.2.3.3 – Process Monitor : Feature Restrict File Download on PID 3760

Another suspicious piece of information obtained was the CRYPTBASE.dll file which was created and loaded successfully. This could be a .dll file which stores instructions to execute the encryption process of files and directories if this was a ransomware malware.

Figure 3.2.2.3.4 – Process Monitor : Create CRYPTBASE.dll on PID 3760

WannaPeace.exe was also found to change the registry settings of Microsoft Strong Cryptographic Provider, which could alter the default RSA Full cryptographic service provider.

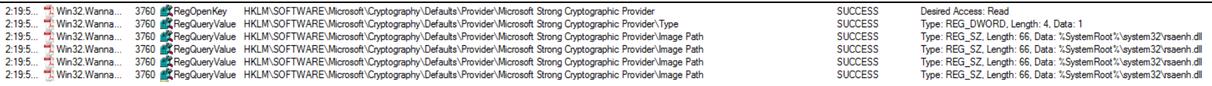


Figure 3.2.2.3.5 – Process Monitor : Microsoft Strong Cryptographic Provider on PID 3760

These details found within Process Monitor shows the attacker executing certain processes and information. It successfully changed the registry setting as highlighted.

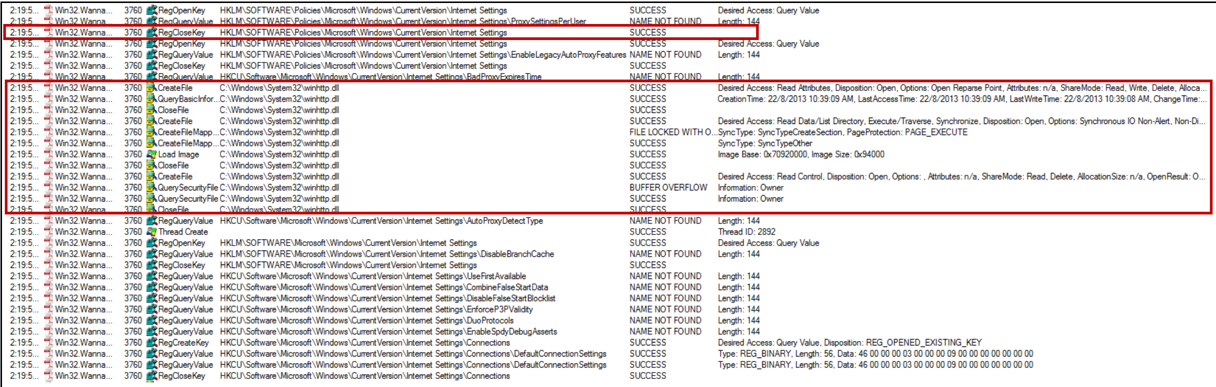


Figure 3.2.2.3.6 – Process Monitor : Internet Settings on PID 3760

Within the same image, there is a long list dedicated to the attacker creating winhttp.dll file and inserting the desired access such as Read Data / List Directory, Execute / Traverse and more. These could be used to send the information regarding the structure of the system and how the files and directories are branched to the attacker.

By filtering the processes to display “WriteFile” processes and only from PID 2356 (I reset the VM to a clean one to try again). From the results shown after the filter, these directories do not seem to contain any information.

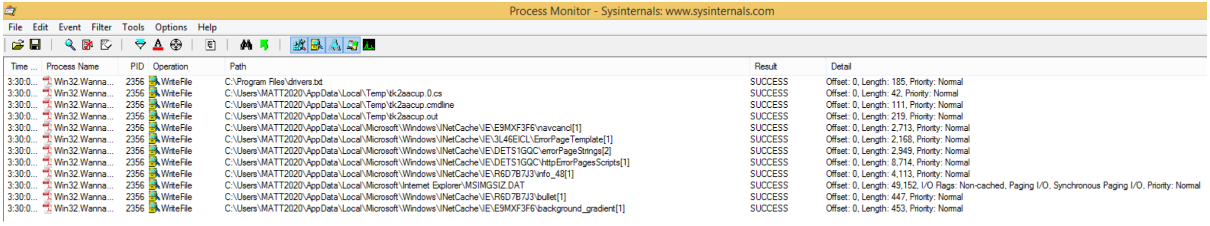


Figure 3.2.2.3.7 – Process Monitor : WriteFile operation on PID 2356

3.2.2.4. ApateDNS

Since most malware is designed to interact with the attacker, it often attempts to make connections to external networks. Thus, network analysis was performed using ApateDNS to check if the malware is making any connection requests to external networks.

Upon executing WannaPeace.exe, it was detected by ApateDNS that a domain request attempt made to www.horacerta.com.br was made. Although I wanted to open the site to check it out, it was dangerous and only used HTTP connection, hence I refrained from it. However, from some Google search, I guess that this website is covered as a hotel reservation website where users can book rooms in various cities such as Los Angeles, Chicago, and Buenos Aires.

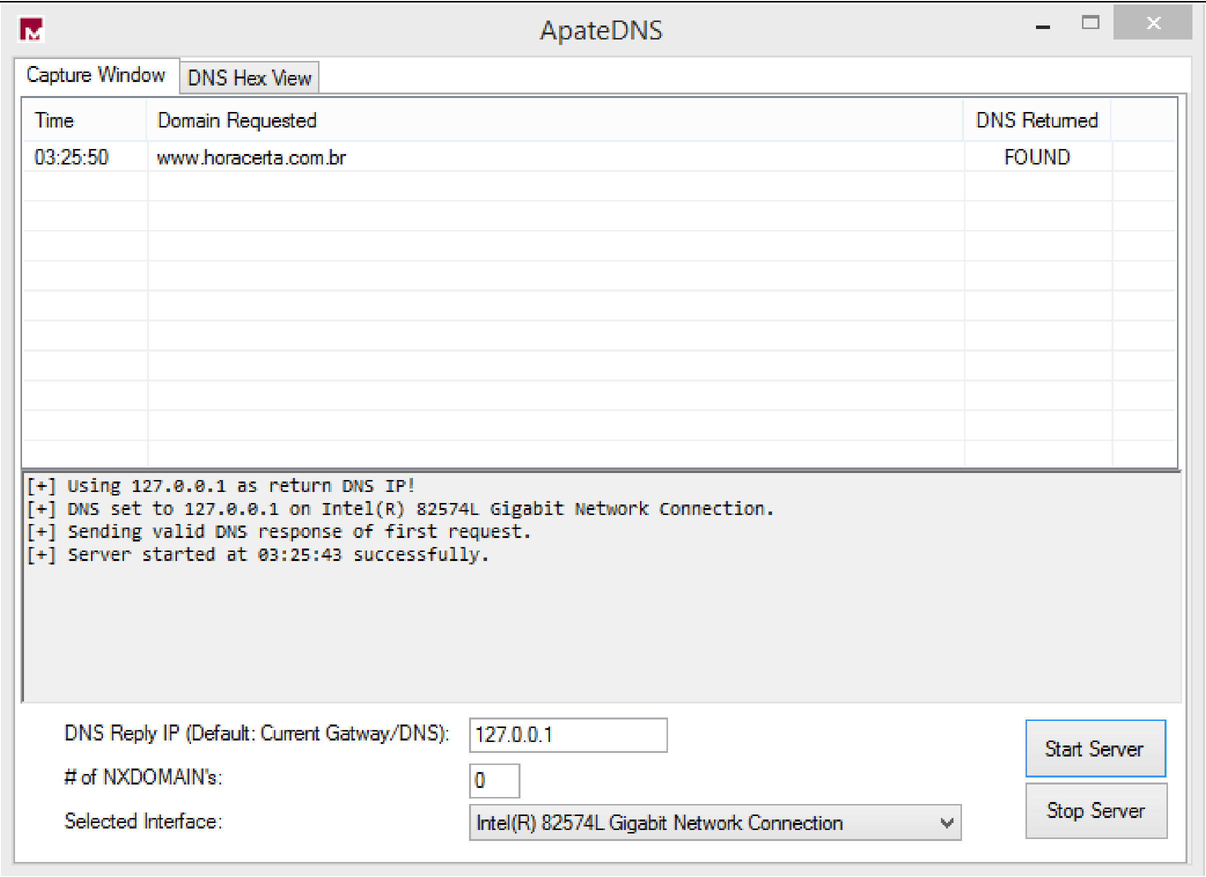


Figure 3.2.2.4.1 – ApateDNS response

However, after waiting 30 minutes, these are all the DNS returned.

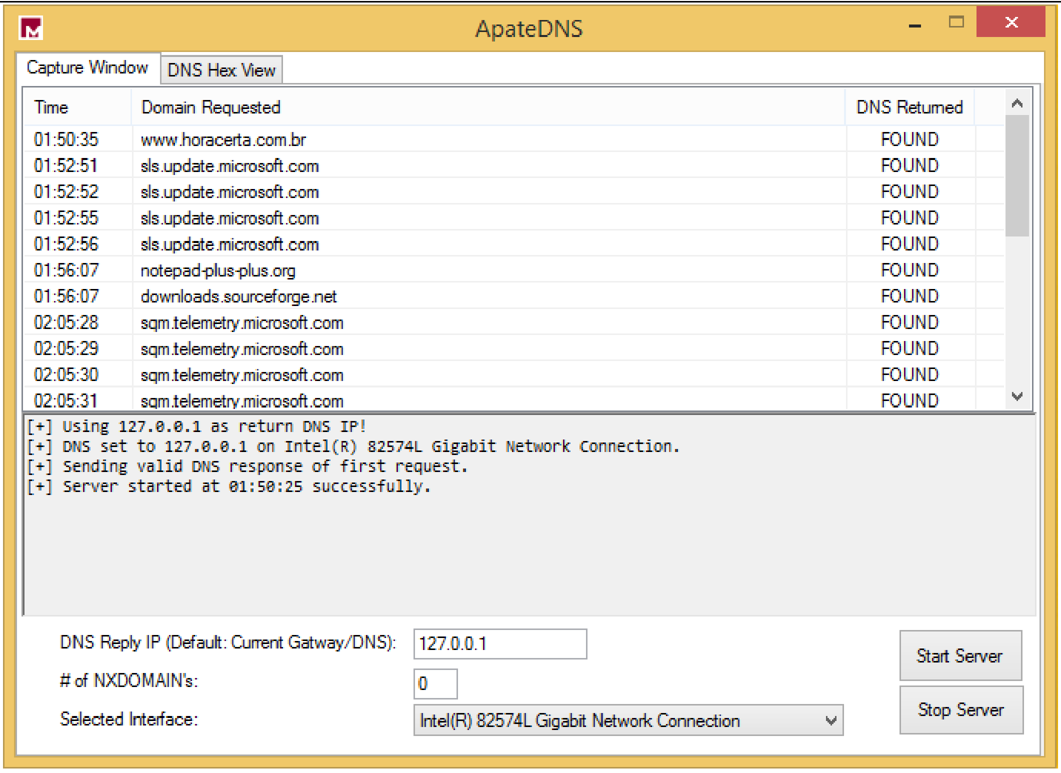


Figure 3.2.2.4.2 – ApateDNS response (EXTRA)

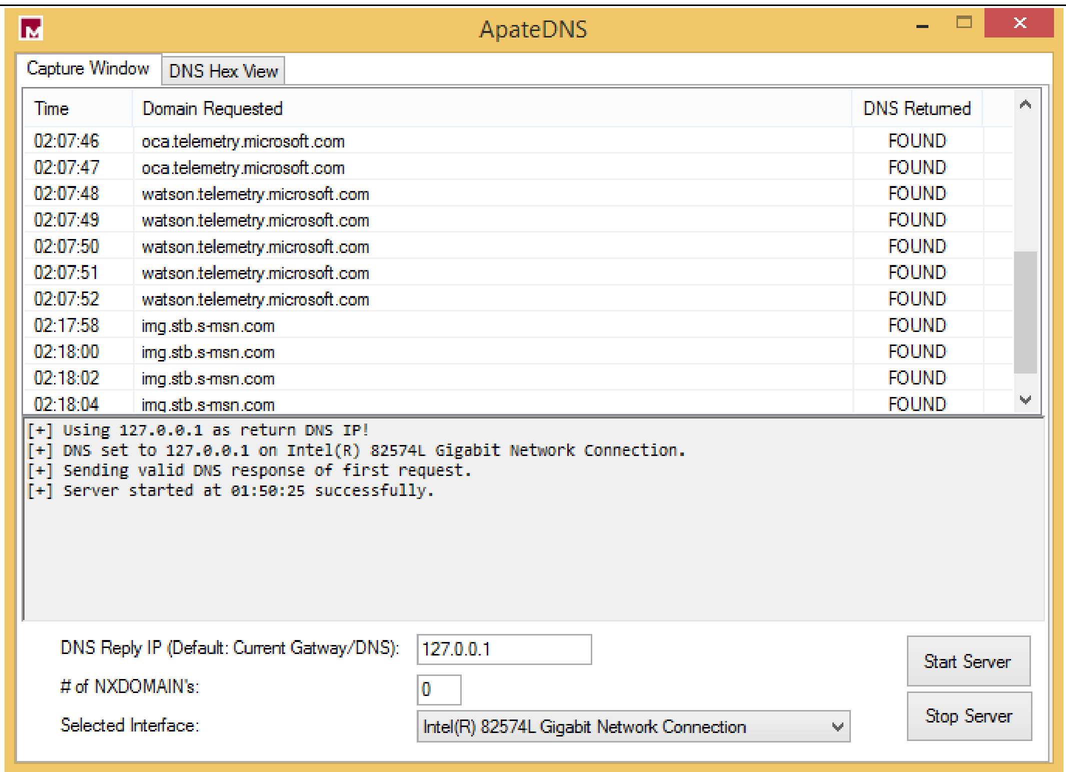


Figure 3.2.2.4.3 – ApateDNS response (EXTRA)

3.2.3. Basic Dynamic Analysis Conclusion

After performing basic dynamic analysis on the WannaPeace.exe, I am almost able to fully understand what the malware is capable of doing. Through general analysis, it is known that the attacker uses http protocol port 49217 to establish a connection with the malware. The attacker also uses the C# programming language based on the strings seen to create functions to such as create, encrypt, decrypt the files. Further analysis showed that the malware executed created much more .dll files such as winhttp.dll, where basic static analysis would not be able to pick this detail up, only basic dynamic analysis could do so. Lastly, it uses a legitimate program called Adobe Reader to bluff users into thinking that the software downloaded has issues or that there is an installation issue due to the number or errors popping up when launching the malware. This would lead the user to not think twice and assume there is nothing wrong with it and will not suspect that it is a malware in disguise.