

**Malware Analysis Tools and Techniques**

**Year 2 (2022/23), Semester 4**

**SCHOOL OF INFOCOMM TECHNOLOGY**

Diploma in Cybersecurity & Digital Forensics

**ASSIGNMENT**

Weightage: 40%

Deadline: 6th February 2022

| **Type** | Win32 Executable |
| --- | --- |
| **Malware Name** | Kovter Trojan |
| **Executable Filename** | PDFXCview.exe |
| **MD5 Hash** | 15af6227d39ca3f9d1dcd8566efb0057 |
| **URL Download** | <https://github.com/ytisf/theZoo/tree/master/malware/Binaries/Trojan.Kovter> |
| **Team Members** | Lim Kai Chong (S10226797)  Lim Jing Jie, Ian (S10222871) |

**Table of Contents**

[**1 Introduction**](#_8qn411vim8c1) **4**

[**2 Lab Setup**](#_45xbk4yp32eu) **5**

[2.1 Virtual Machine Setup](#_6g6dkmd4jzvu) 5

[2.2 Network Configuration](#_jk9i4a8fklpg) 7

[2.2.1 Host Operating System Network Configuration](#_929gydmuqh6w) 7

[2.2.2 Guest Operating System Network Configuration](#_s64spjrzoo41) 7

[2.2.3 Configured Lab Environment Diagram](#_vmkj2j2ytn09) 8

[**3 Malware Analysis Tools**](#_xpem8rxzto5y) **9**

[3.1 Basic Static Analysis Tools](#_pan152qrkp52) 11

[3.1.1 Tool 1 - PEiD](#_s15y22ibje9j) 11

[3.1.2 Tool 2 - PE-Bear](#_vvswct6rfp63) 12

[3.1.3 Tool 3 - Bintext](#_citxrgl208qi) 13

[3.1.4 Tool 4 - VirusTotal](#_6d8n5hori4rq) 14

[3.2 Advanced Static Analysis Tools](#_wwqwgn9bk9s) 16

[3.2.1 Tool 1 - IDAPro](#_nmz9rtba8v29) 16

[3.3 Basic Dynamic Analysis Tools](#_7f6iqv9l097d) 18

[3.3.1 Tool 1 - ApateDNS](#_at0ehpr6ivc) 18

[3.3.2 Tool 2 - Wireshark](#_aajb5r6o3t7l) 19

[3.3.3 Tool 3 - Regshot](#_yr7cbhbn8v2n) 20

[3.3.4 Tool 4 - Regedit](#_eymvbzai6stc) 21

[3.3.5 Tool 5 - Process Explorer](#_xum7bim16kkv) 22

[3.3.6 Tool 6 - Process Monitor](#_9hryjxcfmgs7) 23

[3.4 Advanced Dynamic Analysis Tools](#_7lkpbih3f8zn) 24

[3.4.1 Tool 1 - OllyDBG](#_nz8a4kvs874j) 24

[**4 Static Analysis**](#_n3znek7o7876) **26**

[4.1 Basic Static Analysis](#_sxuz9259l2e1) 27

[4.1.1 Initial Analysis via VirusTotal](#_w8goh2hm8fha) 27

[4.1.2 Preparing Malware for Static Analysis](#_66f7koy037v) 29

[4.1.3 String analysis](#_223nsabdxgvl) 34

[4.2 Advanced Static Analysis](#_41lj5jv5ni2h) 41

[4.2.1 Static Analysis Via IDAPro](#_ee51yccikrg) 41

[4.2.1.1 Imports](#_egrql43yhbg0) 41

[4.2.1.2 TLS & Misc](#_f66z9ffbpauf) 44

[4.2.1.3 Network Base Indicators](#_ednuhvfbe55q) 47

[4.2.1.4 Registry Key Manipulation](#_zbumv8aj5lfv) 50

[4.2.1.5 Persistence Via Mutex Processes](#_3tp5ccgxsyrq) 51

[4.2.1.6 Process Snapshot Tool](#_ceoldxqu7fxi) 53

[**5 Dynamic Analysis**](#_u35cou2rou0v) **55**

[5.1 Basic Dynamic Analysis](#_ry6tbwgkfryb) 55

[5.1.1 Initial Observations Upon Executing Malware](#_e4r068ppap6q) 55

[5.1.2 Registry Analysis](#_qk6dwxrmk3g8) 56

[5.1.3 Process Analysis](#_lgaj40mbufy5) 63

[5.1.4 Network Analysis](#_uvjeqvqj748q) 67

[5.2 Advanced Dynamic Analysis](#_g2eam8o6p27i) 70

[5.2.1 Import Address Table and Memory Map](#_f7yvqa9odz4q) 70

[5.2.2 Disassembler Window](#_rp2c2ojj9fwt) 72

[5.2.3 Debugging Function Calls using Breakpoints](#_4qm9byttp6f2) 73

[**6 Obfuscated Code Analysis of Malware**](#_ezw1289w5qur) **79**

[6.1 Obfuscated Code 1](#_g13d1p1g6nsb) 80

[6.1.1 Debugging the Code using Firebug](#_qyyfpl4sxoh2) 81

[6.1.2 Analyzing Interesting Variables](#_7h9zvvlyoudf) 83

[6.1.3 Decoding the Base64 String](#_z9b4pxd838bw) 84

[6.1.4 Functionality of Obfuscated Code 1](#_7afg0ppxf0va) 86

[6.2 Obfuscated Code 2](#_aar0yx6s2vzn) 87

[6.2.1 Functionality of Obfuscated Code 2](#_uovmfdgxtkxs) 87

[**7 General Analysis of Malware**](#_ovc5crglb9bh) **89**

[7.1 Malware Behavior](#_zg1f1nda08j6) 89

[7.2 Malware Persistence Mechanisms](#_b7kcqslhty3d) 90

[7.2.1 Fileless Persistence](#_ksu9wwp7znqb) 90

[7.2.2 Run Registry Key](#_9yxeeu3vioc1) 92

[7.2.3 Obfuscation](#_ygai6m8cr4op) 95

[7.2.4 Summary of Kovter’s Persistence Mechanisms](#_jx4s5utgy59b) 96

[7.3 Malware Type](#_2bm36cos5lo1) 97

[**8 Malware Defenses**](#_obrzg27u7rw) **100**

[8.1 Antivirus Solutions](#_bhbl8ia81vdf) 101

[8.2 Securing Powershell](#_eufqbys61kct) 101

[8.3 Software Updates](#_lstgbi9x9168) 102

[8.4 Regularly Monitoring Network Activity](#_8ebw3ydyfeff) 102

[**9 Malware Removal**](#_6doqljwf4ump) **104**

[9.1 Indications of presence of Kovter](#_afyltccqm29f) 104

[9.2 Removal of Kovter](#_hfthps8bgkjn) 106

[**10 Summary**](#_ry1mq8ylsmd4) **110**

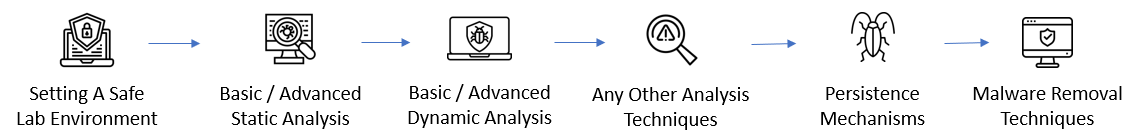
[**11 Video Submission Links**](#_57xj6tfg2aa9) **111**

# 

# 1 Introduction

The unprecedented advancements in technology, coupled with better understanding of operating systems, programming languages and other commercial tools have made certain tasks such as conducting cyberattacks much more complex than ever before. With the aid of malicious software, or what is commonly known as malware, that is only getting more and more sophisticated by the day, modern cyberattacks are able to pose a significantly more potent threat to organizations, public and private sectors, as well as individuals. As a result, there is an indispensable need for the knowledge, skills and tools required to analyze and reverse-engineer malware to be enforced. Fortunately, just as there exists a wide range of tools and techniques used to create malware, there also exists a similarly large variety of tools and analysis techniques that can be leveraged to reverse-engineer and analyze malware.

In this report, by conducting both static and dynamic analysis, and by leveraging the tools that exist in these analysis methods, an in-depth examination and documentation will be performed on a Windows 32-bit executable malware known as Kovter. In addition, further analysis of the malware will be conducted accordingly by observing the general behavior of the malware and determining whether any additional analysis techniques need or need not be used. Lastly, the report will also cover the persistence mechanisms employed by the malware, and the malware removal process, documenting the steps that need to be taken to remove Kovter from a Windows system. The figure below depicts the malware analysis process that this report will follow:



*Figure 1a - The Malware Analysis Process used in this report*

# 2 Lab Setup

Before the malware can be analyzed, a safe lab environment needs to be set up to ensure that the functionality of the executable can be examined and documented in a protected manner. As such, this section will detail the methodology used to set up a safe lab environment for the malware to be analyzed.

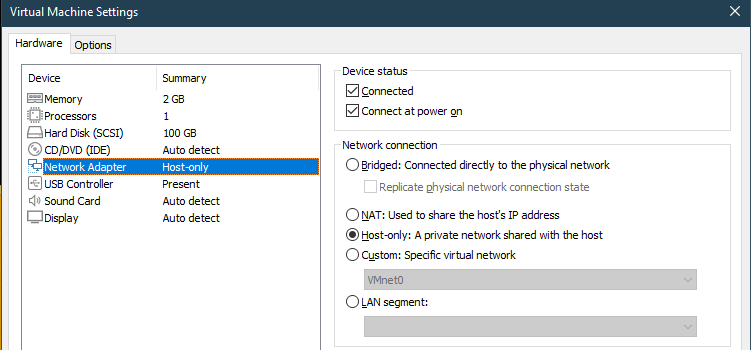
## 2.1 Virtual Machine Setup

The analysis of the Kovter malware will be conducted inside a Virtual Machine. This analysis methodology is a common technique applied in malware analysis, as a means to simulate an additional instance of an operating system on the same machine. This is to replicate and provide a real environment in a much-protected manner, to see how a malware sample interacts with everything on an operating system, from the file system to the registry.

| **Host Operating System** | Windows 11 |
| --- | --- |
| **Hypervisor Software to Host VMs** | VMware Workstation 16 Pro |
| **Guest Operating System** | Windows 8 |

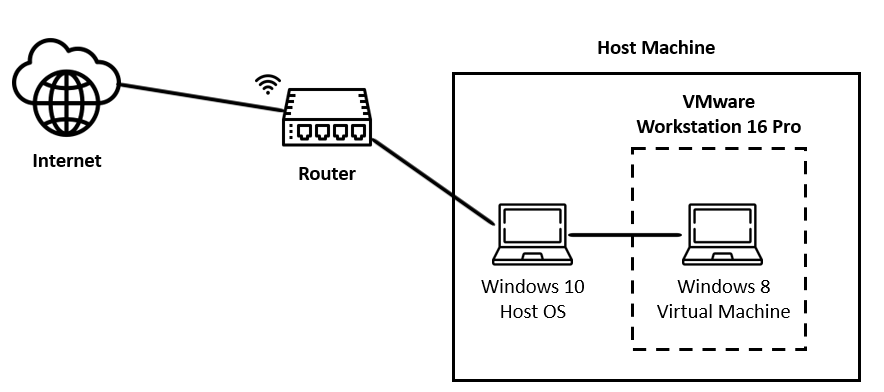
*Table 2.1a - The Host OS, Hypervisor software and Guest OS used to set up the Virtual Machine environment*

With reference to Table 2.1a, the entire lab environment will run on a **Windows 11** Host Operating System. The **VMware Workstation 16 Pro** application will be used to host the Virtual Machine in which the Kovter malware will be executed on. The Guest Operating System (i.e. Virtual Machine) that the malware will be executed on will be a **Windows 8** system.



*Figure 2.1a - The Virtual Machine configuration settings*

As seen from Figure 2.1a, 2 GB of memory space is allocated to the Virtual Machine, and the Virtual Machine is configured to a **host-only** network, which will be discussed in the later section. For now, the network diagram of the lab environment setup can be seen from the figure depicted below:



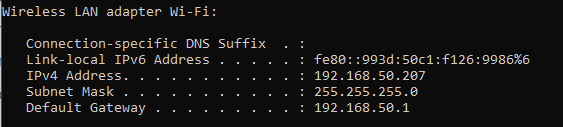
*Figure 2.1b - The lab environment setup*

## 2.2 Network Configuration

After setting the basis for the lab environment, proper configuration of the network will need to be performed, especially on the Windows 8 Virtual Machine, to ensure that any malicious payloads that the malware may execute will not affect the condition of the host machine. In that regard, proper checks will be conducted on the host operating system as well, to confirm that the host and guest operating systems are not operating within the same network.

### 2.2.1 Host Operating System Network Configuration

The quickest and most efficient way to determine the IP address of the Host operating system would be to simply run an ‘ipconfig’ command on the Command Prompt, which displays the TCP/IP network configuration values assigned to the host machine.

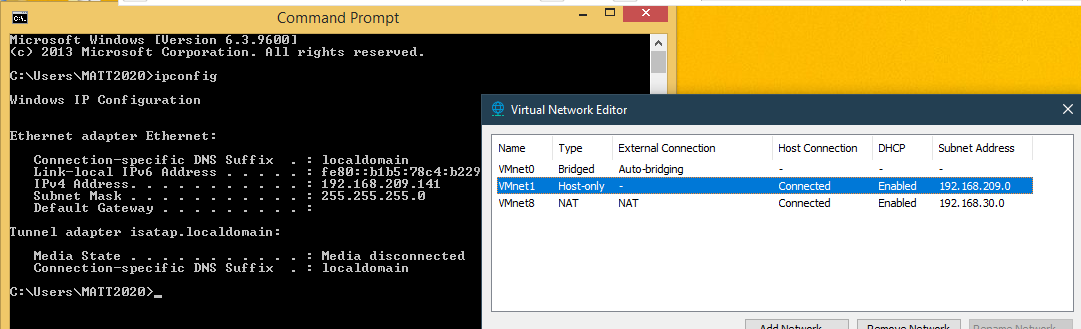


*Figure 2.2.1a - IP address of the host operating system*

Figure 2.2.1a shows that the IP address assigned to the host operating system is **192.168.50.207**.

### 2.2.2 Guest Operating System Network Configuration

As mentioned in the earlier sections, the Guest Operating System is configured to a **host-only** network, or more specifically, the host-only network provided by VMware Workstation 16 Pro. This means that the local IP address that is assigned to the Guest Operating System is obtained from the DHCP Server of the VMware Workstation 16 Pro application, and this local IP address cannot be used to access the Internet or any browsing agents.

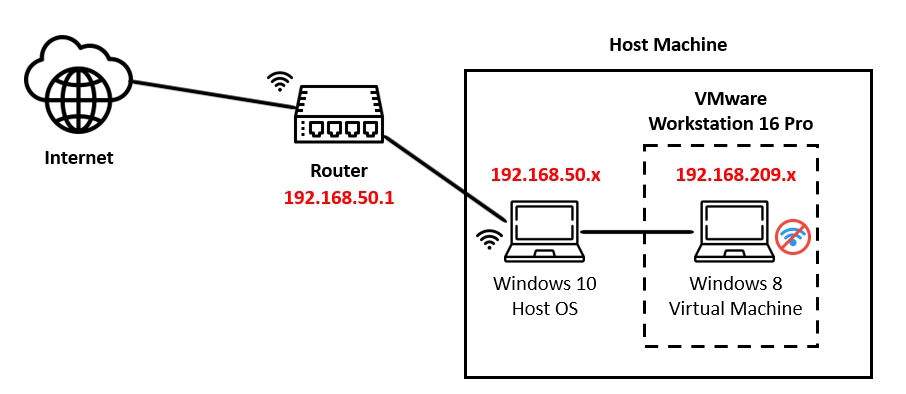


*Figure 2.2.2a - IP address of the guest operating system*

By looking at the Virtual Network Editor within VMware Workstation 16 Pro, and by also using the ‘ipconfig’ command in the Command Prompt of the Windows 8 Guest Operating System, we are able to deduce that the IP address assigned to the Guest Operating System is **192.168.209.141**. From this, we can also confirm that the Guest and Host Operating Systems are indeed connected to different networks.

### 2.2.3 Configured Lab Environment Diagram

After setting up a Virtual Machine and configuring the networks, we have now set up a proper lab environment, and can now proceed to analyze the malware. The figure below depicts the network diagram of the configured lab environment that we set up:

*****Figure 2.2.3a - Network diagram of the configured lab environment*

# 3 Malware Analysis Tools

In order to analyze and deduce the functionality of the malware, various tools that each provide a different function need to be leveraged accordingly for each analysis method. In this section, the tools that will be used in this report to analyze the malware will be covered and elaborated upon\*.

*\*Note: The tools used for Obfuscated Code Analysis will be explained in* [*Section 6*](#_ezw1289w5qur) *individually*

| **Analysis Technique** | **Tool** | **Functional Usage of Tool** |
| --- | --- | --- |
| Basic Static Analysis | PEiD | Identifying packer and unpacking |
| PE-bear | Compare executable composition of unpacked and packed version of Kovter |
| BinText | Analyze string text found in unpacked version of Kovter |
| VirusTotal | Gather extensive information about the executable file( Kovter) |
| Advanced Static Analysis | IDA Pro | Use to analyze assembly code of unpacked version of Kovter |
| Basic Dynamic Analysis | ApateDNS | Monitoring network activity |
| Wireshark | Log and sniff network traffic |
| Regshot | Comparing registry snapshots |
| Regedit | Observe the registry keys added to registry hives |
| Process Explorer | Monitors running processes |
| Process Monitor | Monitor registry, file system, network and process activity |
| Advanced Dynamic Analysis | OllyDBG | To debug and analyze assembly code |
| Obfuscated Code Analysis | PDF Stream Dumper | To format obfuscated Javascript code |
| Firebug | To debug HTML, CSS and Javascript code on a webpage |
| Base64 Decoder | To decode a Base64-encoded string |

*Table 3a - Table that lists and categorizes tool usage for malware analysis*

Table 3a categorizes and lists the various tools that will be leveraged in this report for the analysis of the Kovter malware.

## 

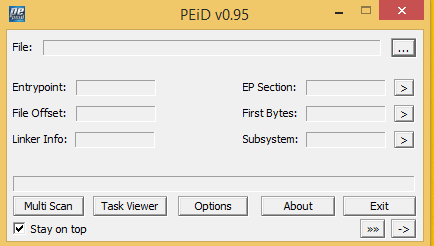
## 3.1 Basic Static Analysis Tools

With reference to Table 3a, in this section, the tools that will be used in this report to perform basic static analysis on the Kovter malware will be documented and elaborated upon.

### 3.1.1 Tool 1 - PEiD

PEiD, or Portable Executable Identifier, is a tool that can help us identify the type of packer or compiler used to build an executable file. This information can be used to help determine the level of obfuscation applied to the malware, which in turn can aid in the static analysis process.

When performing static analysis, it's important to know how the malware was built as it affects the way in which it behaves. For example, if a packer was used to compress the malware, we would need to unpack the file before performing any further analysis. PEiD makes this process easier by automatically identifying the packer and providing the necessary information to unpack the file.



*Figure 3.1.1.1 - PEiD’s user interface*

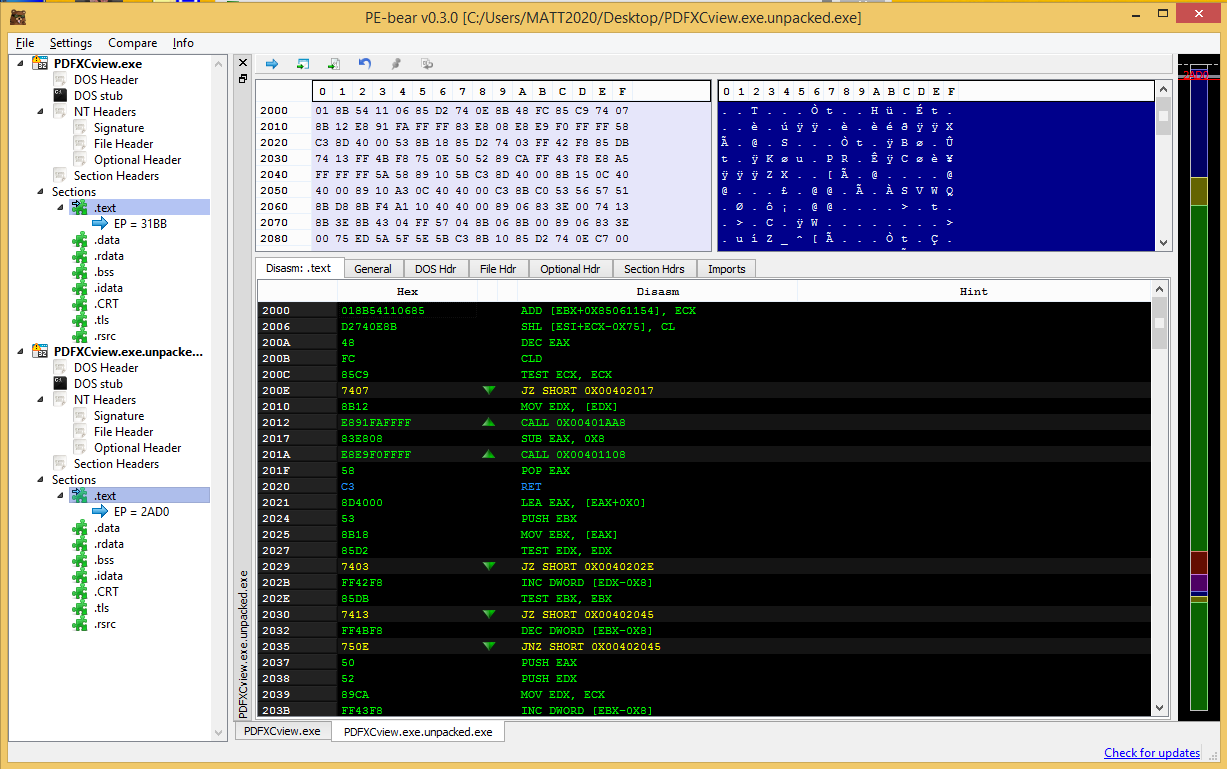
PEiD can be a valuable tool for us by aiding us to understand the inner workings of a malicious executable. By providing information about the packer and compiler used to build the file, PEiD helps to speed up the static analysis process and improve the accuracy of the results.

### 3.1.2 Tool 2 - PE-Bear

PE-Bear is a powerful tool for static malware analysis. It allows us to examine the inner workings of portable executable (PE) files, which are commonly used for Windows applications and services, without actually executing them. This provides a safe way to inspect the code and identify any malicious behavior, without the risk of infecting the system.

PE-Bear offers a wealth of information about the file being analyzed, including metadata, imports, exports, resources, and more. This information can be used to determine if a file is malicious or not, and what its capabilities and functions are. PE-Bear can also be used to detect obfuscation techniques that are commonly used by malware authors to conceal their code and evade detection.

Additionally, PE-Bear supports plugins and scripts, which can be used to automate and streamline the analysis process. These plugins and scripts can be used to perform tasks such as extracting embedded files, decrypting strings, or even executing code within the analyzed file.

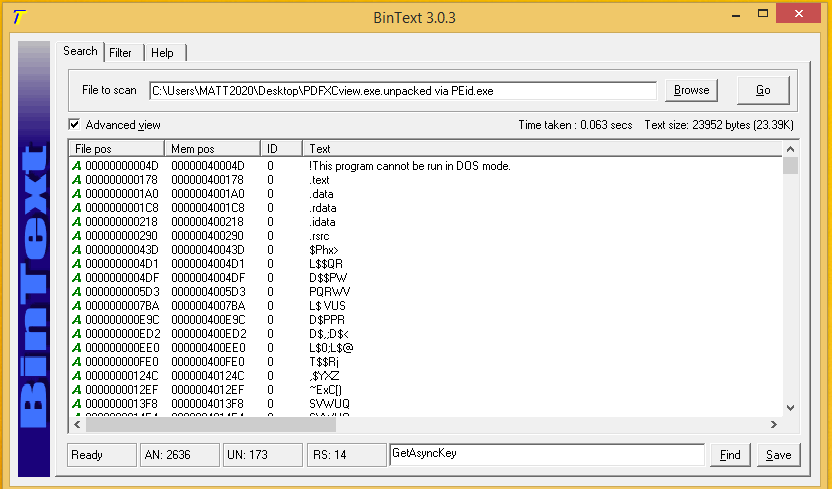


*Figure 3.1.2.1 - PE-bear interface with Kovter Malware shown*

### 3.1.3 Tool 3 - Bintext

BinText is a tool that is often utilized in the process of basic static analysis for malware analysis. This program is simple and straightforward, designed to extract and display the text strings within binary files such as executables and DLLs. In the field of malware analysis, BinText can be particularly useful in revealing hidden strings that are not visible within the code itself.

When conducting basic static analysis, BinText is used to extract and examine the strings within a malware sample. These strings can provide crucial insight into the malware's behavior and functionality, as well as any interactions it may have with other files, domains, or IP addresses. For instance, BinText can uncover URLs, IP addresses, and file paths that the malware may be communicating with or writing to, allowing analysts to determine the malware's origin and purpose.



*Figure 3.1.3.1 - Sample of Bintext analyzing an executable file*

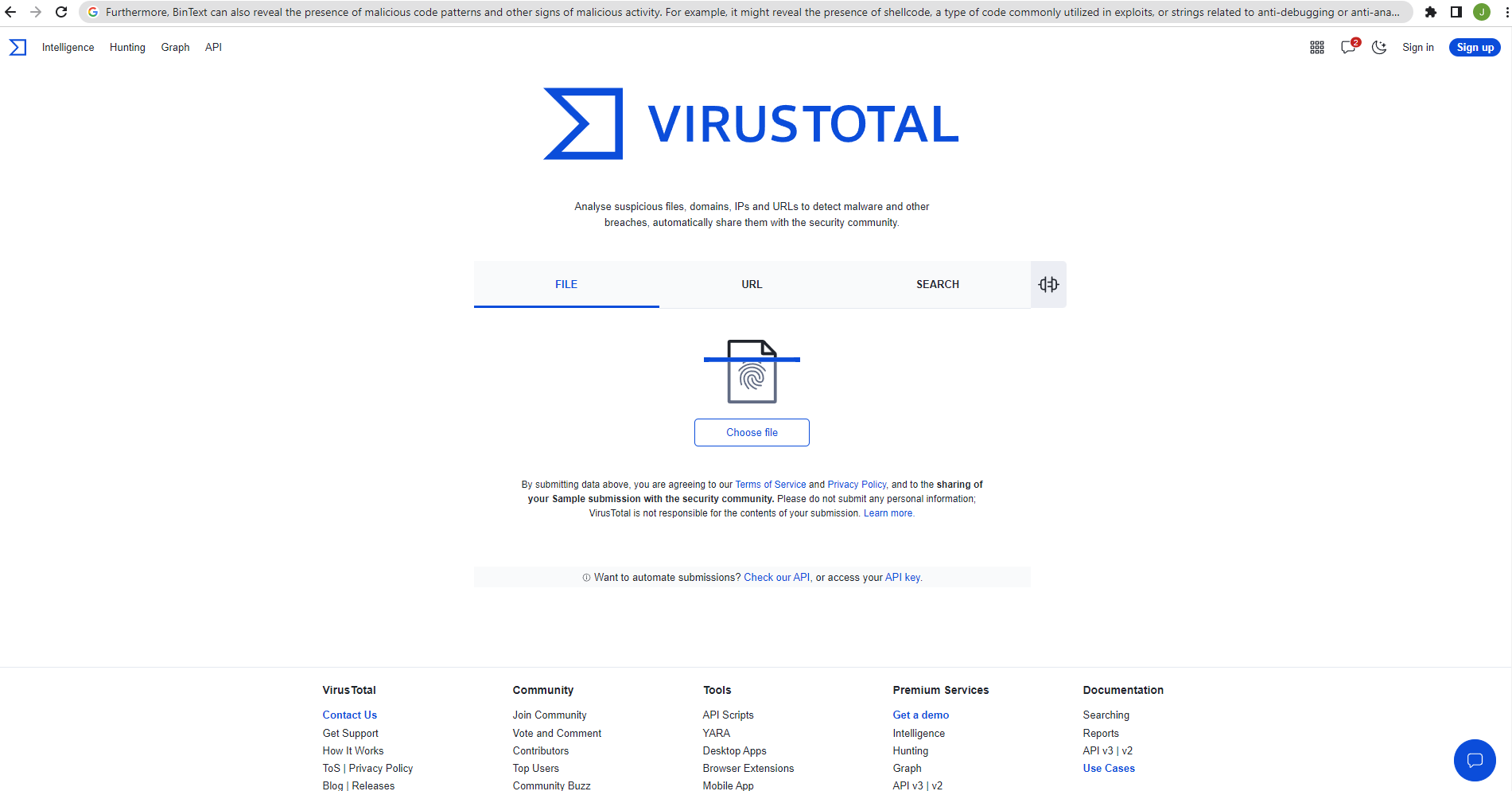
With reference to the above figure, BinText can also reveal the presence of malicious code patterns and other signs of malicious activity through readable string text such as the first line. Other strings might reveal the presence of shellcode, a type of code commonly utilized in exploits, or strings related to anti-debugging or anti-analysis techniques. This information can give us a better understanding of the malware's behavior and functionality, and help identify potential weaknesses that can be exploited for analysis or mitigation purposes.

### 3.1.4 Tool 4 - VirusTotal

VirusTotal is a free online tool that is widely used in the field of basic static analysis for malware analysis. It is a platform that aggregates more than 70 antivirus engines and allows users to upload and scan files for malicious content. VirusTotal analyzes the files and provides a report indicating which antivirus engines have flagged the file as malicious, and what kind of malicious behavior the file is known to exhibit.

In basic static analysis, VirusTotal is used to quickly and easily assess the presence of malware in a given file. By uploading the file to VirusTotal, we can quickly obtain a high-level overview of the file's behavior and determine whether it has been identified as malicious by any of the antivirus engines. This information can be used to make a preliminary assessment of the file's threat level and to determine whether further analysis is necessary.

In addition, VirusTotal can also provide valuable information about the file's behavior and functionality. For example, it can reveal the file's contacts with other domains, IP addresses, and URLs, as well as its interactions with other files. This information can be used to identify the malware's purpose and origin and to develop a mitigation strategy.



*Figure 3.1.4.1 - VirusTotal user interface*

## 3.2 Advanced Static Analysis Tools

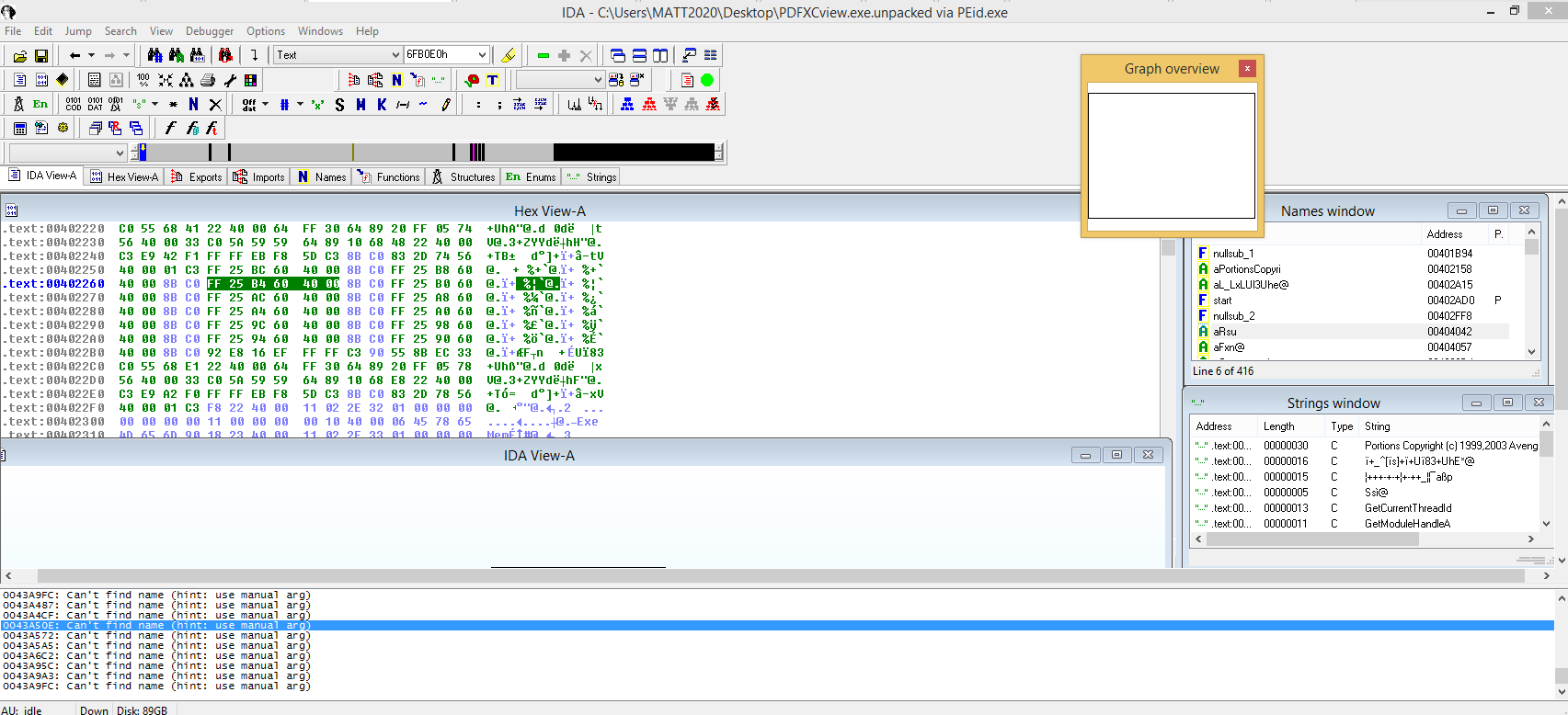
With reference to Table 3a, in this section, the tool that will be used in this report to perform advanced static analysis on the Kovter malware will be documented and elaborated upon.

### 3.2.1 Tool 1 - IDAPro

IDAPro is a powerful disassembler and debugger software and is particularly useful for reverse engineering, malware analysis, and vulnerability discovery.

One of the key functionalities of IDAPro is its ability to disassemble executable files into human-readable assembly code. This is achieved through the use of sophisticated algorithms that decompile the binary code into a form that can be understood by humans. This allows us to see how the code is executed and what it is trying to do, making it much easier to identify any malicious behavior.

Furthermore, IDAPro also provides a rich set of debugging tools, including the ability to set breakpoints, single-step through the code, and inspect the values of register and memory locations.It is also able to automatically identify and label important functions and data structures within the code, making it much easier to read the code, making it much easier to analyze the behavior of complex or obfuscated code, and to identify the root cause of any vulnerabilities or issues.



*Figure 3.2.1.1 - IDAPro Interface*

## 

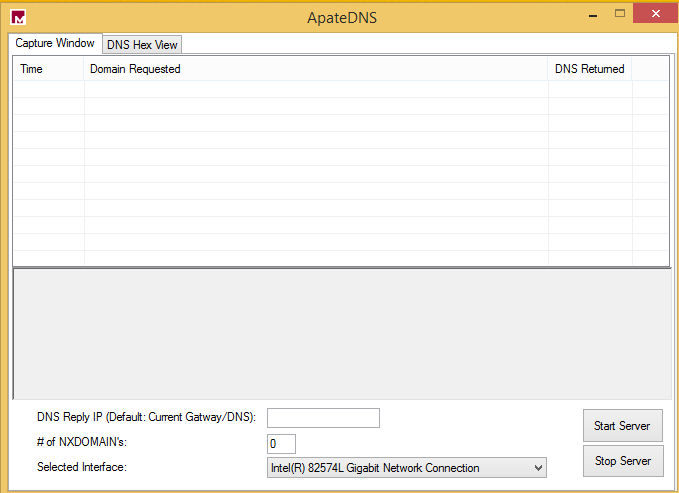
## 3.3 Basic Dynamic Analysis Tools

With reference to Table 3a, in this section, the tools that will be used in this report to perform basic dynamic analysis on the Kovter malware will be documented and elaborated upon. For reference, 6 tools will be used in this analysis stage, namely ApateDNS, Regshot, Regedit, Wireshark, Process Explorer and Process Monitor.

### 3.3.1 Tool 1 - ApateDNS

ApateDNS is a tool that can be used to analyze malware by intercepting and analyzing the DNS (Domain Name System) requests made by the malware. This is done by using a technique known as **DNS sinkholing.** DNS sinkholing is a technique used to intercept DNS requests by configuring a computer’s DNS settings to point to a local DNS server (usually **127.0.0.1**), rather than a remote server. The local DNS server, which runs ApateDNS will then be configured to respond to all DNS requests with fake IP addresses, effectively “sinkholing” the domain and preventing the malware from communicating with its Control and Command (C&C) server.

In addition to preventing the malware from communicating with its C&C server, ApateDNS also logs all DNS requests made by the malware and analyzes them to identify the domain and IP addresses used by the malware to track the malware’s network activities.

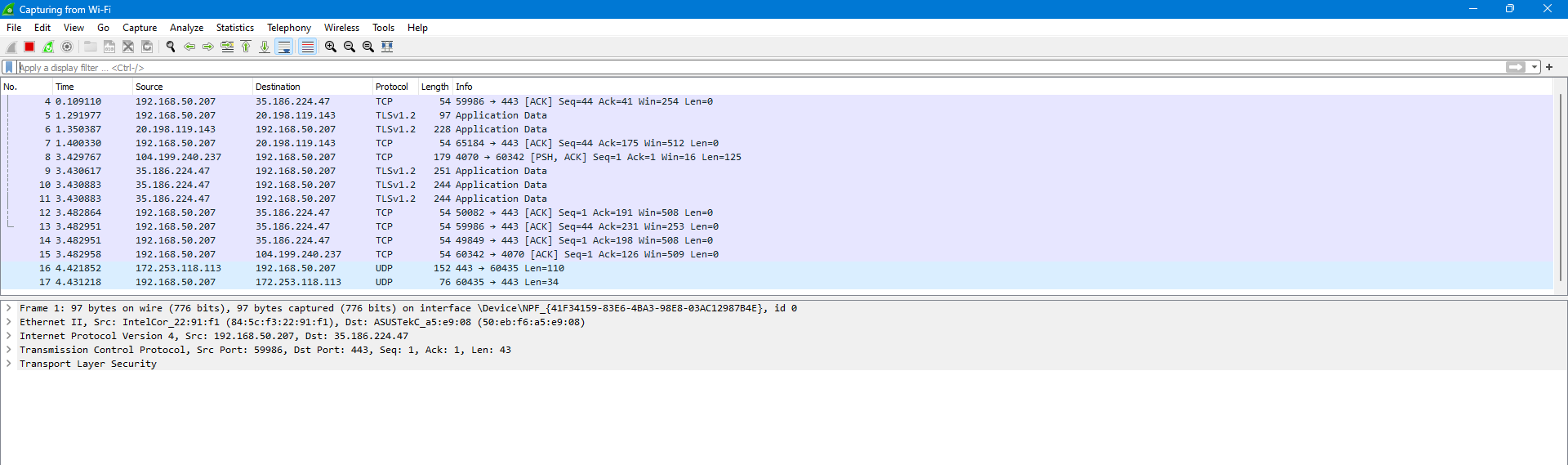


*Figure 3.3.1a - User interface of ApateDNS*

In the basic dynamic analysis section, ApateDNS will be used to monitor and analyze the network activity of the Kovter malware, to deduce what possible domains or IP addresses the malware may look to communicate or connect to. This will allow us to gain a better understanding of the behavior of the malware.

### 3.3.2 Tool 2 - Wireshark

Wireshark is a network protocol analyzer tool that allows a user to capture and inspect network traffic. It can be used to analyze malware by capturing and analyzing the network traffic of an infected computer to observe the network activity and network connections established by the malware. Wireshark can be used to capture and decode a wide variety of network protocols, including TCP, UDP, HTTPS, DNS and more, which can provide valuable information about the malware’s behavior and communications.



*Figure 3.3.2a - User interface of Wireshark*

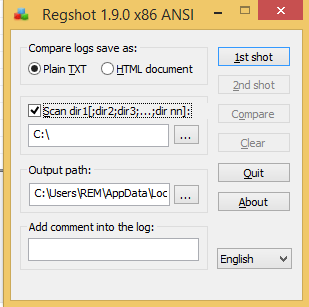
In the basic dynamic analysis section, Wireshark will be used in conjunction with ApateDNS to provide a more comprehensive analysis of the malware’s behavior by monitoring and analyzing the network activity of the Kovter malware.

### 

### 3.3.3 Tool 3 - Regshot

Regshot is a tool that is able to take snapshots of a system’s registry. In addition to this, Regshot is able to compare registry snapshots taken in a system taken between any given time or state. Regshot is typically used for malware analysis by taking snapshots of a system’s registry before and after the execution of a malware.

This is useful in analyzing malware because the registry is a database of system settings and configurations in a Windows operating system, therefore changes to the registry can give valuable information about the malware’s behavior such as what files it created, what registry keys it modified and what new processes it started.

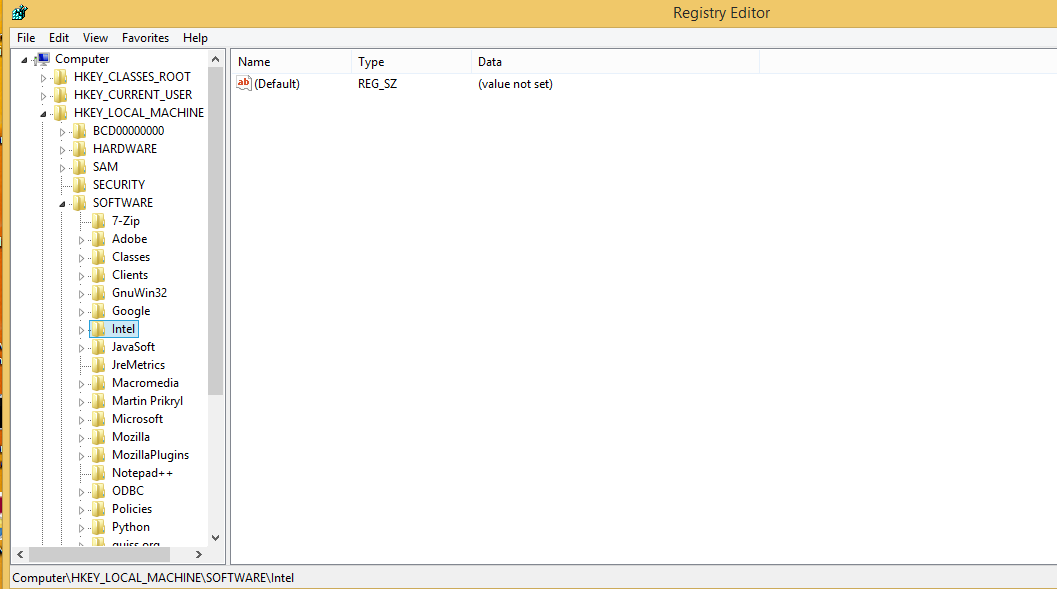


*Figure 3.3.3a - User interface of Regshot*

In the basic dynamic analysis section, this tool will be used to perform registry analysis by comparing the registry snapshots before and after the malware is executed, to observe any registry changes made by the malware that could provide useful insights in understanding the behavior of the malware.

### 

### 3.3.4 Tool 4 - Regedit

Regedit, or Registry Editor is a built-in tool in the Windows operating system that allows a user to view and edit the registry. It can be used to analyze malware by manually reviewing the registry changes made by the malware. This is a useful tool to analyze malware because malware generally modifies certain registry keys and values such as the Run keys, which are used to start programs automatically on system start up, /shell/open/command keys which are used to execute shellcode on a command-line interpreter, and many more. 

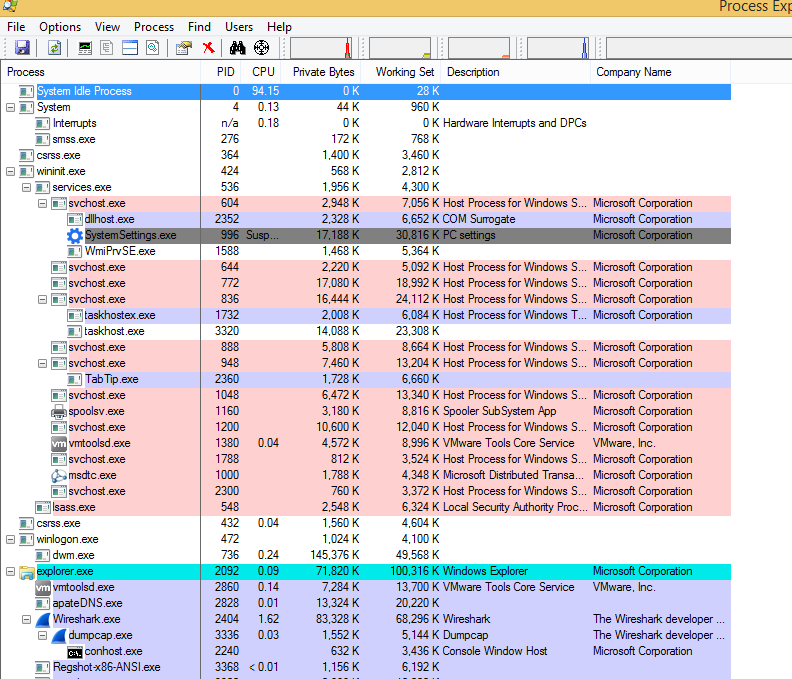
*Figure 3.3.4a - User interface of Regedit*

This tool is especially useful when used in conjunction with Regshot, as Regshot is able to list the registry keys modified by the malware so that if these registry keys need to be analyzed on a more granular level, Regedit can be used to manually navigate to these registry keys. As such, in the basic dynamic analysis section, Regedit will be used in conjunction with Regshot to perform registry analysis upon execution of the malware.

### 

### 3.3.5 Tool 5 - Process Explorer

Process Explorer is a tool that is used to view detailed information about the processes running on a Windows operating system. It can be used to analyze malware by identifying and studying the processes that the malware creates or modifies. Sometimes when the malware injects itself into a parent process, Process Explorer can also be used to deduce the parent process the malware has injected itself into. In addition to the above, Process Explorer can also be used to identify the DLLs loaded by a process, along with the strings the process uses, which are useful metrics in analyzing the intention and behavior of a process statically.



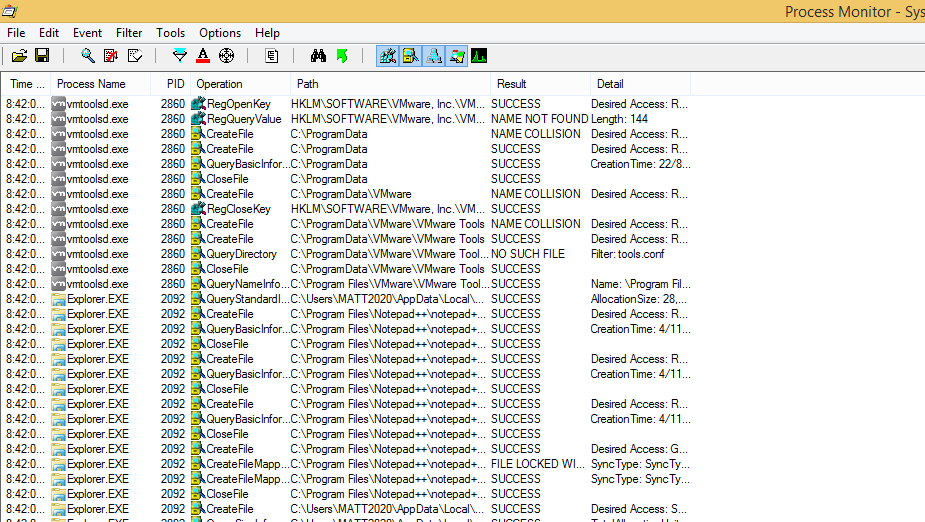
*Figure 3.3.5a - User interface of Process Explorer*

In the basic dynamic analysis section, Process Explorer will be used to perform process analysis and to view the processes that are created, modified or deleted by the malware.

### 

### 3.3.6 Tool 6 - Process Monitor

Process Monitor is a tool that monitors and logs various system events such as file system, registry, network and process activity in real-time. It can be used to analyze malware by identifying and studying the system events that the malware generates. Process Monitor can provide a lot of information about the malware’s behavior and its interactions with the system. It can reveal the files and registry keys that the malware created, modified or deleted, the processes it created or injected itself into and many more by manually filtering between different metrics within the application.



*Figure 3.3.6a - User interface of Process Monitor*

In the basic dynamic analysis section, Process Monitor will be used in conjunction with Process Explorer to perform the process analysis of the malware dynamically. By viewing the various operations made by the malware, these tools can also be used to gain a greater understanding of the behavior of the malware as well as its functionalities.

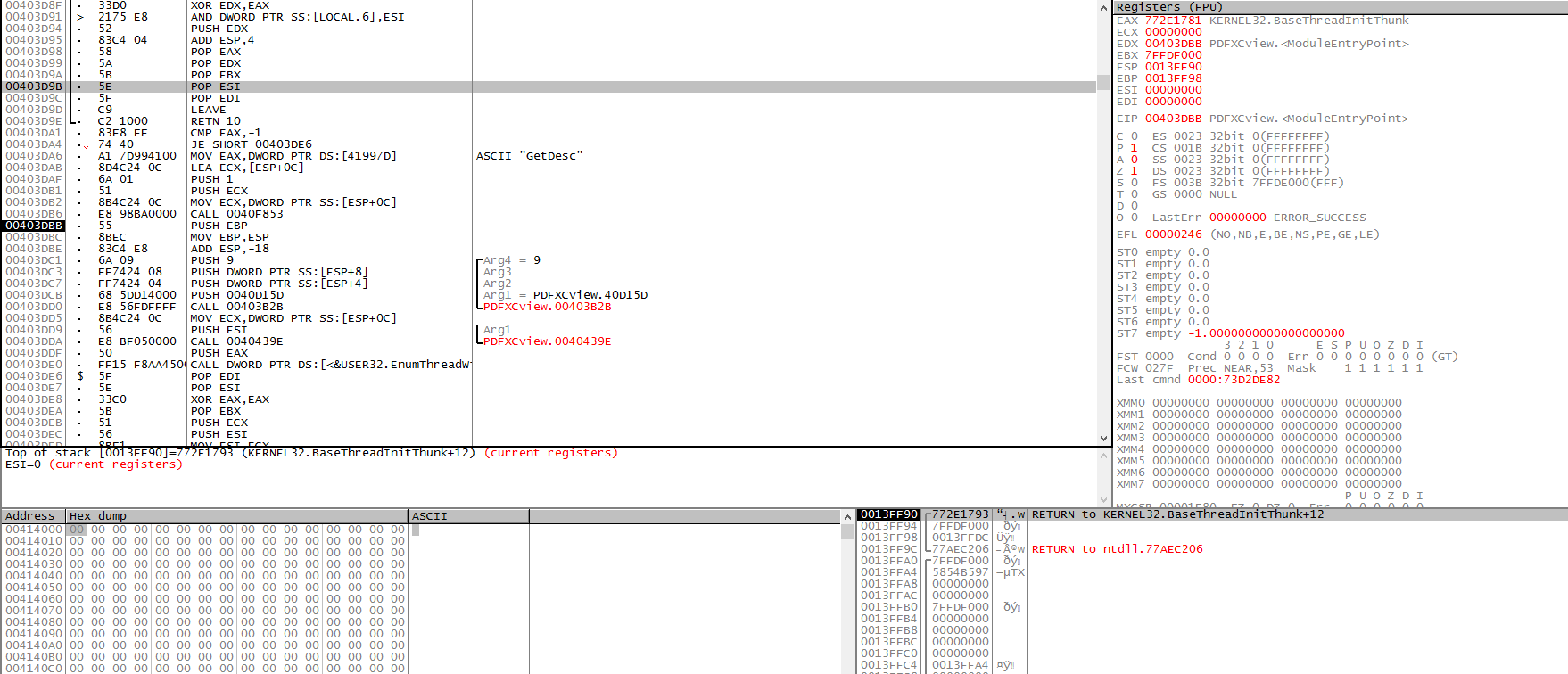
## 3.4 Advanced Dynamic Analysis Tools

With reference to Table 3a, in this section, the tools that will be used in this report to perform advanced dynamic analysis on the Kovter malware will be documented and elaborated upon.

### 3.4.1 Tool 1 - OllyDBG

OllyDBG is an x86 debugger for Microsoft Windows. It is designed to analyze binary files, primarily executables and dynamic link libraries (DLLs), in order to understand how they work, detect bugs, and perform reverse engineering tasks.

OllyDbg is widely used by software developers, security researchers, and reverse engineers for various purposes, including debugging and testing their own software, analyzing the behavior of malicious code, and reverse engineering proprietary software. It provides a user-friendly interface, an extensive set of features, and a powerful scripting engine, making it a popular choice among malware analysts.



*Figure 3.4.1a - User Interface of OllyDBG tool*

In this report, under the Advanced Dynamic Analysis section, OllyDBG will be used to analyze the defense mechanisms employed by the Kovter malware sample, as well as other behavioral patterns that the Kovter malware shows through debugging the different subroutines and observing the parameters called into the stack as the program is being executed.

# 4 Static Analysis

Static analysis is the process of examining the executable file without viewing the actual code used to compile and assemble the file. Static analysis can provide insight as to whether or not a file is potentially malicious. It is also an efficient way to provide more context regarding the functionality of the executable file, and it can also provide information that can help a malware analyst deduce whether an executable file contains certain host-based or network-based indicators. In this section, both basic and advanced static analysis techniques will be performed, and for each technique, the process and findings will be documented.

## 

## 4.1 Basic Static Analysis

### 4.1.1 Initial Analysis via VirusTotal

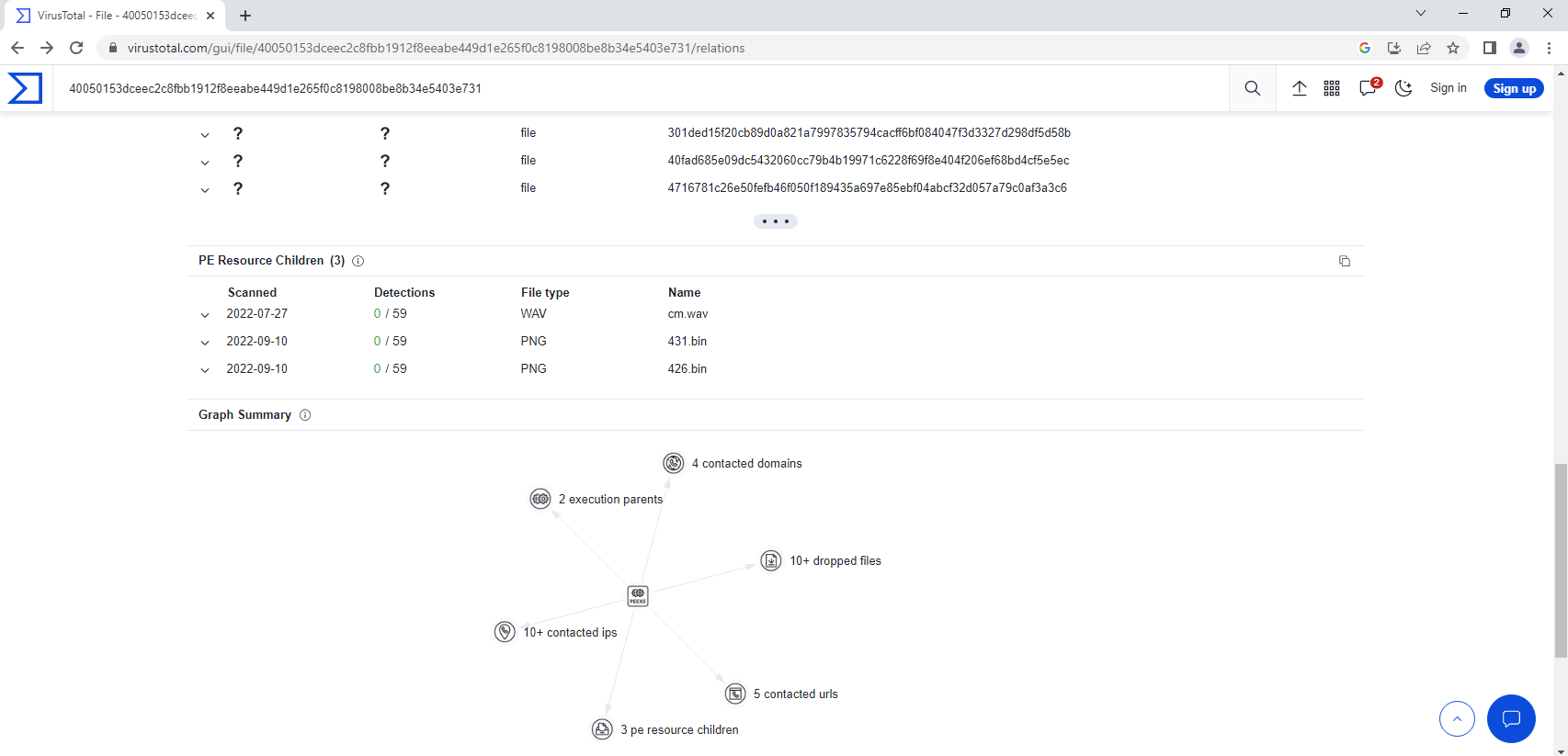
As mentioned in the tools section, VirusTotal is a website and a platform for performing basic static analysis of files and URLs. It is a popular tool that can be used for analyzing malware, as it provides a large amount of information about the file, including its behavior and characteristics. VirusTotal analyzes a file using a variety of antivirus engines. Furthermore, VirusTotal can provide a wealth of information about a file in a single report, making it an essential tool for basic static analysis of malware.

In this section we will Submit Kovter into VirusTotal and analyze the results.

### 

*Figure 4.1.1a - Virustotal analysis of Kovter*

With reference to figure 4.1.1a, the result "61 security vendors and 2 sandboxes flagged this file as malicious" indicates that the Kovter malware sample was detected as malicious.The number of security vendors that have flagged a file as malicious can indicate the severity of the threat posed by the malware and the extent of its prevalence in the wild. In the image above, the high number of security vendors that detected Kovter as malicious suggests that it is a widely spread and well-known threat. Additionally, the file was also analyzed by 2 different sandboxes, which are isolated environments used for testing and analyzing malicious software. Sandboxes can provide more in-depth analysis of the behavior and functionality of a malware, and the results obtained from sandboxes can complement the results obtained from security vendors.



*Figure 4.1.1b - PEid used to identify that the malware is packed*

With reference to figure 4.1.1b, the graph summary of Kovter in VirusTotal shows information about the behavior and activities of the malware, as well as its interactions with other files, domains, and IP addresses. The following is a brief description of the information contained in the graph summary:

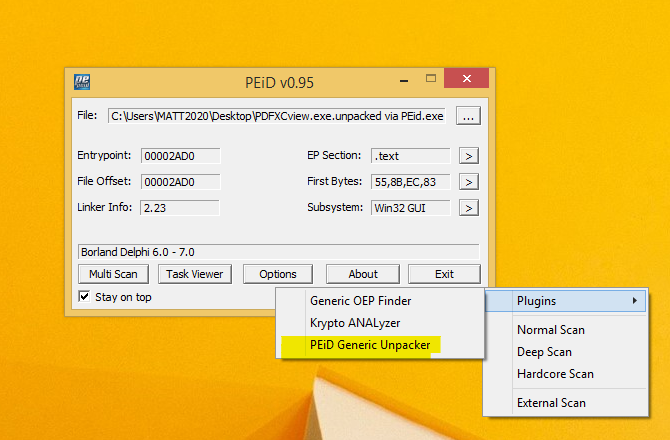
* 4 domains contacted: This refers to the number of domains that the Kovter malware sample contacted during its execution. These domains could be used for command-and-control (C&C) communication, exfiltration of data, or for other malicious purposes.
* 10+ drop files: This refers to the number of files that were dropped by the Kovter malware. Drop files are files that are created or written to disk by malware during its execution. These files could be used for persistence, as staging areas for further infections, or for other malicious purposes.
* 5 URLs contacted: This refers to the number of URLs that the Kovter malware contacted during its execution. These URLs could be used for C&C communication, downloading additional components, or for other malicious purposes.
* 3 PE resource children: This refers to the number of Portable Executable (PE) files that are resources of the Kovter malware. PE files are executables commonly used in Windows systems, and resources are data files that are embedded within the PE file.
* 10 IPs contacted: This refers to the number of IP addresses that the Kovter malware contacted during its execution. These IP addresses could be used for C&C communication, downloading additional components, or for other malicious purposes.
* 2 execution parents: This refers to the number of parent files that executed the Kovter malware. An execution parent is a file that is responsible for executing a malware, such as a dropper or a downloader.

### 4.1.2 Preparing Malware for Static Analysis

As indicated in the tools section, PEid can help identify the packer, compressor and or protector used in the malware, we can then use specialized tools to unpack or de-obfuscate the malware and reveal its true functionality. Additionally, PEiD can also provide information about the compiler and entropy of the malware, which can be used to determine if the malware is packed and potentially detect the presence of anti-debugging or anti-tampering mechanisms.

### 

*Figure 4.1.2a - PEid used to identify that the malware is packed*

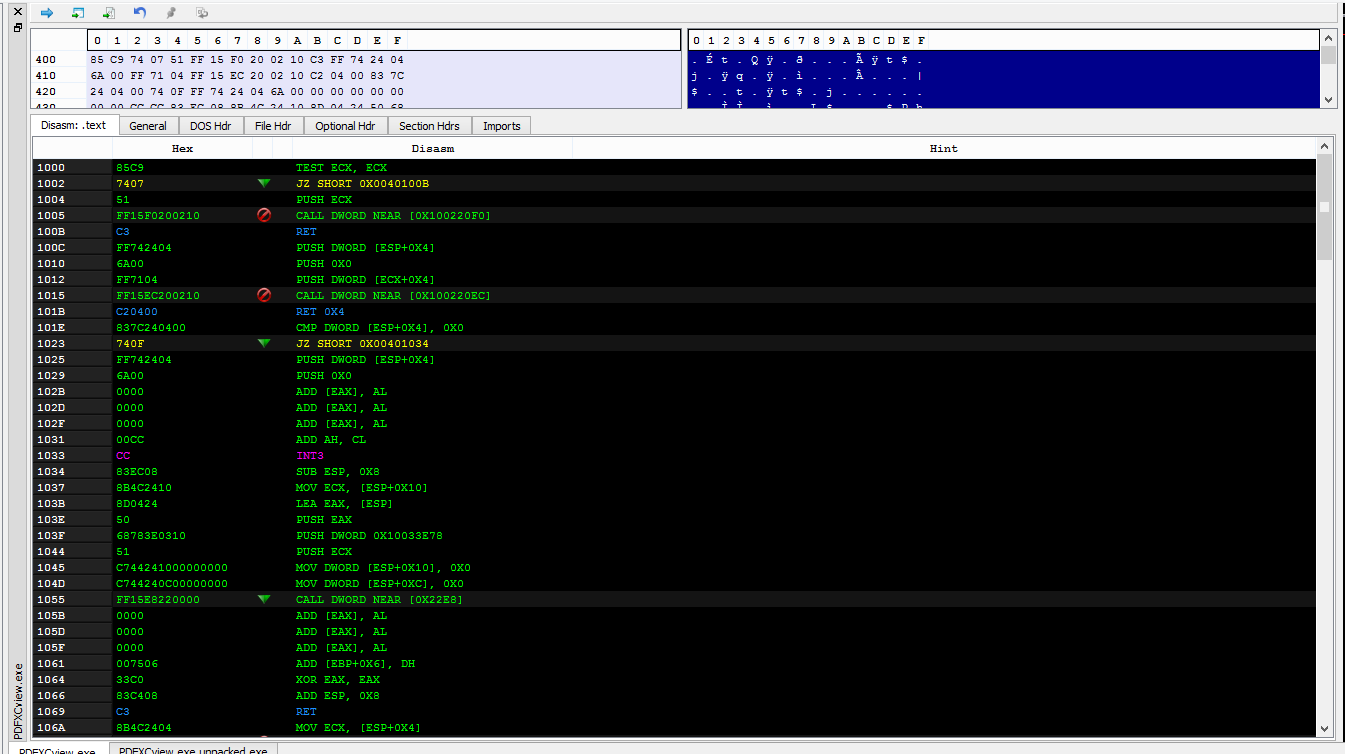


*Figure 4.1.2b - PEid used to unpacked malware*

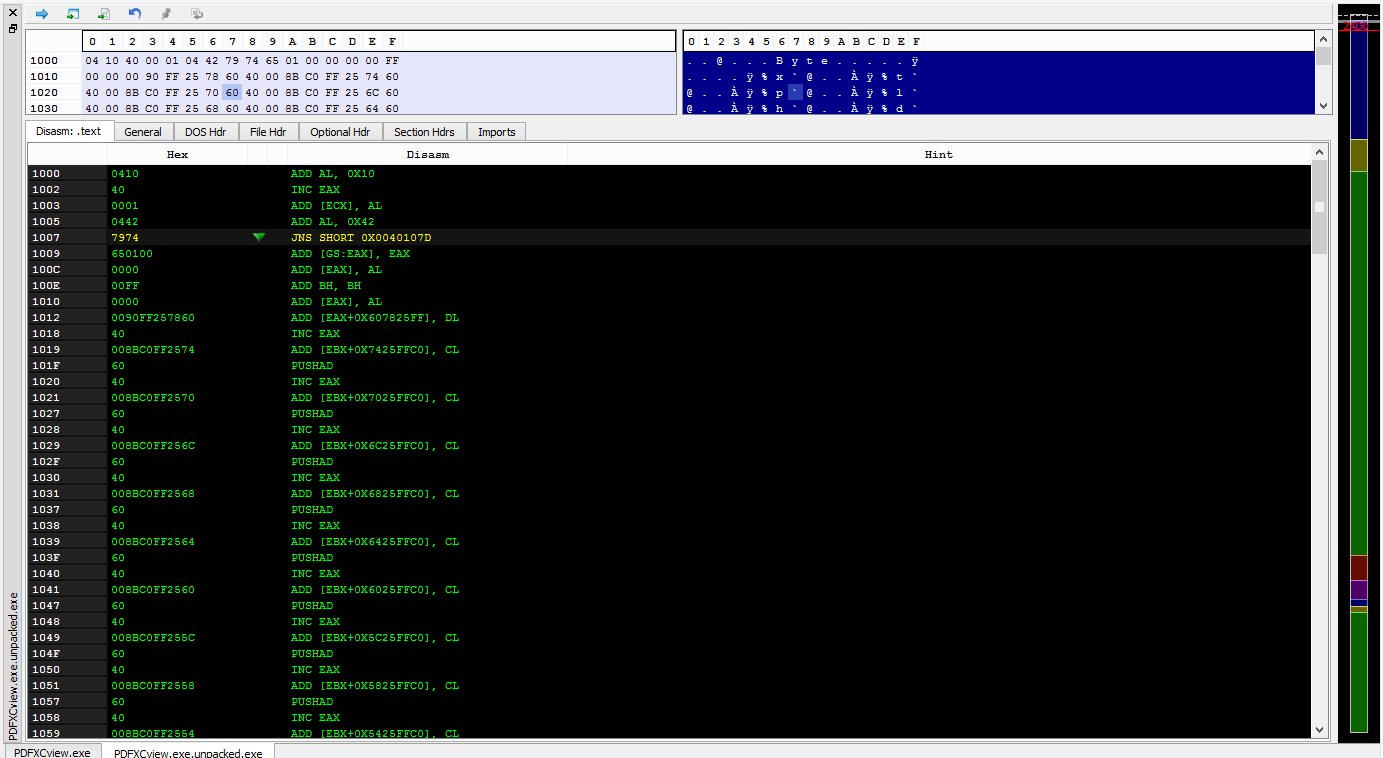
With reference to figure 4.1.2a, I utilized PEiD to unpack the Kovter malware, I load the Kovter malware file into PEiD, and let it automatically scan the file. PEiD then identifies the packer or protector used to compress and encrypt the file. In my case, PEiD showed me that the malware was compiled using the "Borland Delphi 6.0 - 7.0" compiler. I also noticed that the Entropy value of the file returned "7.66 (packed)" which is an indication that the file has been packed. Additionally, the FastCheck option in PEiD returns "Packed" as the result, confirming that the file is packed.

The Kovter malware is packed in order to evade detection and make it more difficult to reverse engineer. Packing is the process of compressing and encrypting the malware's code, which makes it more difficult for security software and researchers to detect and analyze. This technique is used by malware authors to evade detection by making it appear as a random set of bytes to anti-virus software and other security tools, which makes it difficult for these tools to identify the malware by its signature or behavior. Additionally, packing can also make it more difficult for security researchers to reverse engineer the malware and understand its capabilities and behavior. By packing the malware, the malware authors can make it more likely to evade detection and remain on the infected system for a longer period of time

**Comparison of packed and unpacked malware**

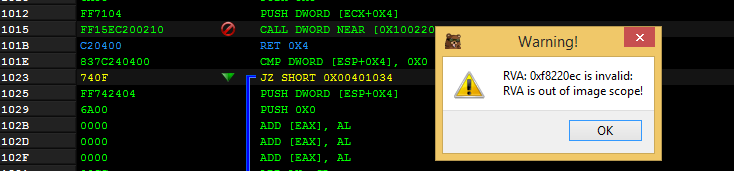


*Figure 4.1.2c - PE-Bear revealed a portion of the assembly code of the packed Kovter*

**

*Figure 4.1.2d - PE-Bear revealed a portion of the assembly code of the unpacked Kovter*

With reference to figure 4.1.2c & 4.1.2d, 4.1.2c depicts the assembly code of the packed Kovter malware, which is filled with invalid statements, making it difficult to read and understand the malware's behavior and capabilities. On the other hand, 4.1.2b shows the same portion of the Kovter assembly code but without the invalid statements, making it easier to read and understand. Comparing the two screenshots, it is clear that the packed version of the Kovter malware uses various techniques to evade detection.



*Figure 4.1.1e - PE-Bear showing details of an invalid statement in the packed Kovter malware*

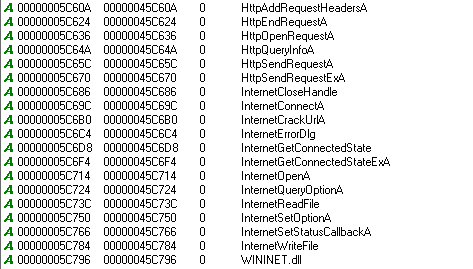
With reference to figure 4.1.2c, it shows a detailed view of the invalid statement in 1015 “call dword near [0x100220ec]” found in the packed version of Kovter. The invalid statement calls upon a value, attempting to transfer execution control to a different part of the code specifically, to the address stored at the memory location 0x100220ec. However, 0x100220ec is outside of the image scope therefore voiding it and making it invalid. Such techniques used by malware authors make the code difficult to read and understand, making it more challenging to analyze the malware.

### 

### 4.1.3 String analysis

As mentioned in section 3, using Bintext I am able to extract strings from the malware, which can provide valuable information about the malware's behavior and capabilities. The strings extracted can include plain text, as well as encoded or encrypted strings, and can reveal information such as file paths, registry keys, and other artifacts that can be used to track the malware and understand its behavior.

**Networking indicators**



*Figure 4.1.3a - Network indicators from a portion of string retrieved from Bintext*

With reference to figure 4.1.3a, the presence of these API’s in the Kovter malware suggests that the malware is attempting to establish a connection to the internet and communicate with a remote server.

The **InternetCrackUrlA** function, for example, is used to break down a URL into its component parts, such as the protocol, hostname, and port. This function can be used by the malware to parse URLs and extract information such as the server address and port that it needs to connect to.

The **InternetGetConnectedState** function, on the other hand, is used to determine the current state of the Internet connection on the infected machine, which can be used by the malware to determine if it is able to connect to the internet or not.

The **InternetOpenA** and **InternetOpenUrlA** functions are used to open a connection to a web server, and the **InternetReadFile** and **InternetWriteFile** functions are used to send and receive data over that connection. This suggests that the malware is able to send and receive data from a remote server and potentially receive commands or instructions from the attackers.

The **InternetSetStatusCallbackA** function allows the malware to register a callback function that will be called whenever the status of an internet operation changes, which can be used by the malware to monitor the state of the connection and detect any errors or changes.

The API’s indicated in figure 4.1.2a are most likely attempting to establish a connection to a remote server and communicate with it, potentially receiving commands or instructions from the attackers and sending data back to them. The information could contain sensitive information about infected machines or even downloads additional malware payloads.

**Registry manipulation**



*Figure 4.1.3b - Evidence of registry manipulation from a portion of string retrieved from Bintext*

With reference to figure 4.1.3b, the presence of these API’s in the Kovter malware suggests that the malware is attempting to access, read, and modify the Windows Registry on the infected machine.

The **RegCreateKeyExA**, **RegCreateKeyExW**, **RegOpenKeyExA**, and **RegOpenKeyExW** functions are used to create and open registry keys, which are the basic building blocks of the Windows Registry. These functions can be used by the malware to create new registry keys or access existing ones, potentially to store its own configuration settings or to hide its presence on the infected machine.

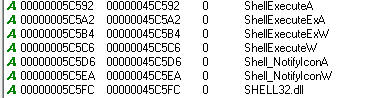
The **RegEnumKeyExA**, **RegEnumKeyExW**, **RegEnumValueA**, and **RegEnumValueW** functions are used to enumerate the subkeys and values of a registry key, which can be used by the malware to scan the registry for specific keys or values that it is interested in.

The **RegQueryValueExA**, **RegQueryValueExW**, **RegSetValueExA**, and **RegSetValueExW** functions are used to read and write values within a registry key, which can be used by the malware to modify or delete existing values or to add new values that it needs.

The **RegisterEventSourceA** and **ReportEventA** functions are used to create and write to event logs, which can be used by the malware to store information about its own activity on the infected machine. This can be used to hide its tracks and make it more difficult for security software or researchers to detect and analyze the malware.

The API strings in figure 4.1.3b indicate that the Kovter malware is likely attempting to access, read, and modify the Windows Registry on the infected machine. It can be used to store its own configuration settings, hide its presence, scan the registry for specific keys or values, and store information about its own activity on the infected machine. The malware is likely using the registry as a means of persistence, which allows it to survive across reboots and continue its malicious activity.

**Process Execution**



*Figure 4.1.2c - Evidence of process execution made by malware*



*Figure 4.1.2d - Evidence of process execution and mutex made by malware*

With reference to figure 4.1.3c and 4.1.3d, the presence of such API’s in the Kovter malware suggests that the malware is most likely attempting to create and execute new processes and mutexes and create new windows and icons.

The **ShellExecuteA** and **ShellExecuteW** functions are used to execute a file, such as a script, executable, or document and can be used by the malware to launch other files or programs as a means of spreading itself or downloading additional malware payloads.

The **ShellExecuteExA** and **ShellExecuteExW** functions are similar to the previously mentioned functions, but they provide extra information about the process that is executed and can be used to track it.

The **Shell\_NotifyIconA** and **Shell\_NotifyIconW** functions are used to create and manipulate icons in the system tray. This can be used by the malware to create a visible presence on the infected machine, which can be used to hide its presence or to provide the attackers with a method for interacting with the malware.

The **CreateMutexA** function creates a mutex object. This can be used by the malware to ensure that only one instance of it is running on the infected machine at a time.

**Miscellaneous**



*Figure 4.1.2e - Evidence of key logging*



*Figure 4.1.3f - Evidence of key logging*

With reference to figure 4.1.3f:

The **GetAsyncKeyState** function is used to determine the current state of a keyboard key or a mouse button; it returns whether the key is pressed or released. This function could have been used by the malware as a keylogger to monitor the state of the keyboard keys and mouse buttons, which can be used to detect user's input and steal sensitive information such as passwords or credit card numbers.

The **SetWindowsHookEx** function is used to install a hook procedure, which is a function that is called in response to specific events, such as keyboard and mouse input, window creation and destruction, and others. The malware could have used this function to install a hook procedure that can intercept and manipulate the input events, this can be used to steal sensitive information, execute arbitrary code, or even control the infected machine remotely.

Both of the functions above could allude to the malware’s ability to act as a keylogger and coupled with networking capabilities, sensitive information can be compromised.



*Figure 4.1.3g - Evidence of screenshotting*

With reference to figure 4.1.3g:

The **GetDC** function is used to retrieve a handle to a device context (DC) for the entire screen, this handle can be used to draw and manipulate the screen content. The malware can use this function to gain access to the entire screen and capture screenshots or record the users activity. This can be used to steal sensitive information and or record keystrokes.

The **BitBlt** function is used to perform a bit-block transfer of the color data corresponding to a rectangle of pixels from one device context to another. This function can be used to copy, stretch or rotate images, the malware can use this function to manipulate the screen content, cover up its tracks or change the information displayed on the screen.

Furthermore, with reference to figure 4.1.3a and the network section, the api’s mentioned could allude to the malwares intention of stealing sensitive data such as screenshots from infected computers and sending it to the attacker via the internet.



*Figure 4.1.3h - List running processes*

With reference to figure 4.1.3h:

The **CreateToolhelp32Snapshot** function is used to take a snapshot of the current processes, modules and heap on the system, it creates a handle to the snapshot which can be used to enumerate the processes and modules. The malware can use this function to enumerate all running processes, gather information about them and potentially identify and terminate specific processes or inject its code into a legitimate processes

This function can also be used to gather system information, like running processes and loaded modules, which can be used to identify the version of the operating system, running services and other useful information for the attackers..

Furthermore, with reference to the network section, 4.1.3e, 4.1.3g and 4.1.3h the api’s mentioned could allude to the malwares intention of stealing sensitive data such as screenshots and or login credentials(key logging) from infected computers and sending it to the attacker via the internet.

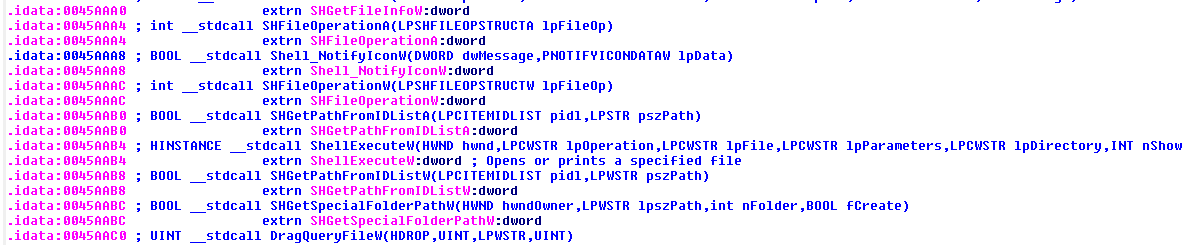
## 

## 4.2 Advanced Static Analysis

### 4.2.1 Static Analysis Via IDAPro

In this section we will be utilizing IDAPro to help us analyze the assembly code of the unpacked version of Kovter.

#### 4.2.1.1 Imports

Whilst analyzing the assembly code of the unpacked version of Kovter, I stumbled upon this chunk of code

*Figure 4.1.2a - List running processes*

With further research I found out that these imports are related to Windows shell functions, this alludes to Kovter utilizing these imports to carry out malicious activities on the infected system After extensive research I concluded that the imported functions might be malicious.

**SHFileOperationA** and **SHFileOperationW** are used to perform file operations such as copying, moving, and deleting files and directories. This can be used to spread the malware to other files and directories, hide its presence, or delete important files.

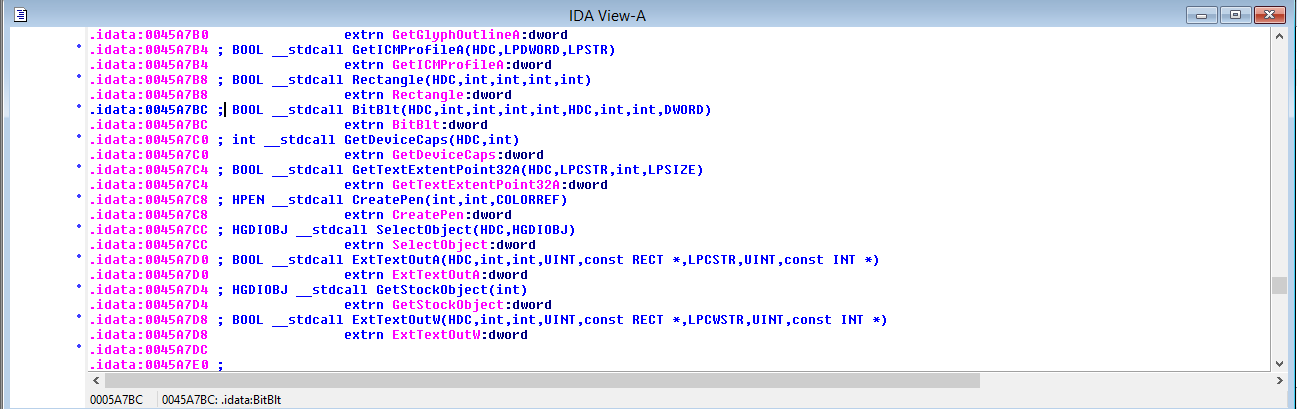
**SHGetFileInfoA** and **SHGetFileInfoW** are used to retrieve information about files and directories such as their type, size, and creation time. This information can be used to determine which files and directories to target for manipulation.

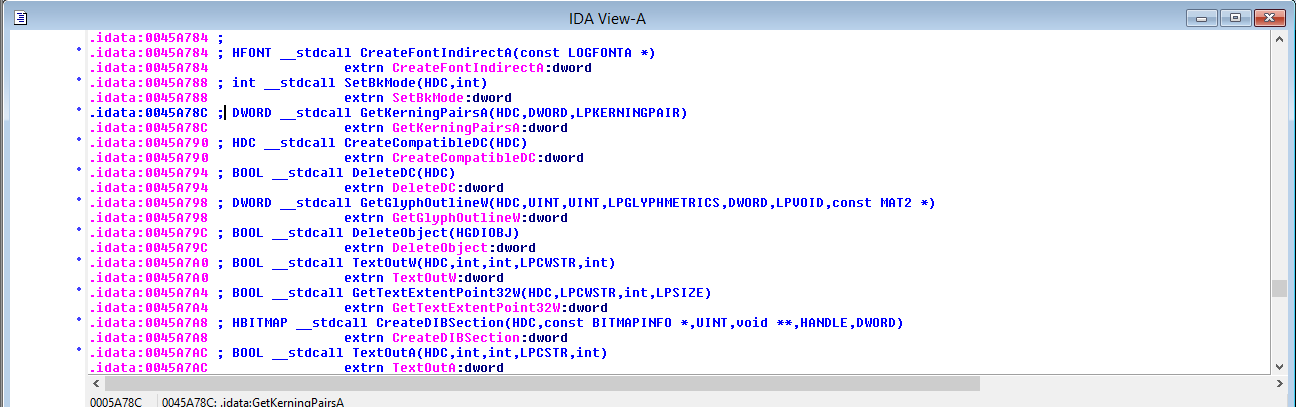
**SHGetMalloc** is used to allocate memory for the malware's operations, which allows it to carry out complex operations without being limited by the memory resources of the infected system.

**SHGetPathFromIDListA** and **SHGetPathFromIDListW** are used to retrieve the path of a file or directory from a list of system identifiers, which allows the malware to manipulate files and directories by their system paths.

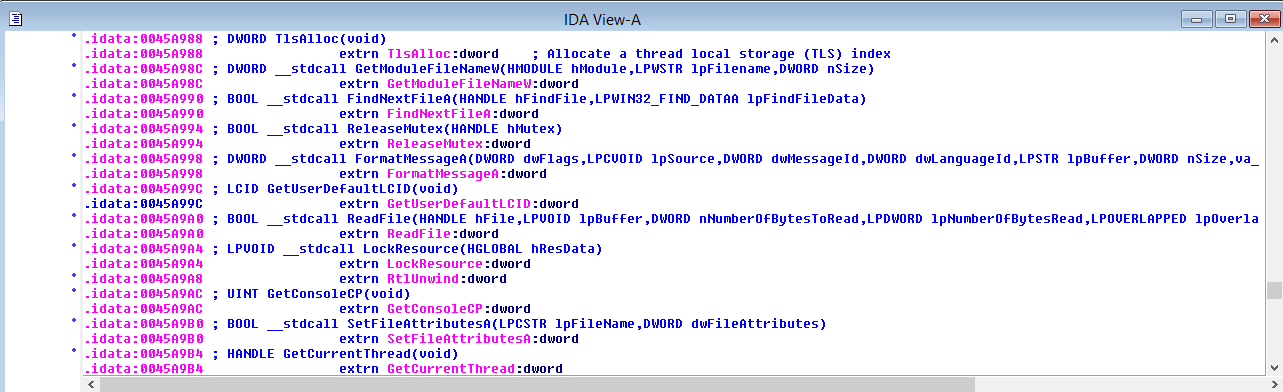
**SHGetSpecialFolderPathA** and **SHGetSpecialFolderPathW** are used to retrieve the path of a special folder such as the Desktop, Start menu, or Documents folder. This can be used to spread the malware to these commonly used locations or to hide its presence in these folders.

Furthermore, we know that Kovter claims to be a PDF viewer. However, A PDF viewer should not have these imports as they are not necessary for the functionality of a PDF viewer and are only used for malicious activities. Having these imports in a PDF viewer can indicate that the software is malicious or has been compromised.





*Figure 4.1.2b - Imported imports via COMTL32.DLL*



*Figure 4.1.2c - Imported imports via COMTL32.DLL*

With reference to figure 4.1.2b & 4.1.2c, these Windows GDI (Graphical Device Interface) functions found in Kovter's assembly code most likely indicate that the malware uses them to create and manipulate graphics and text on the infected system. This can be used to display fake error messages, create fake notifications, or hide the malware's presence.

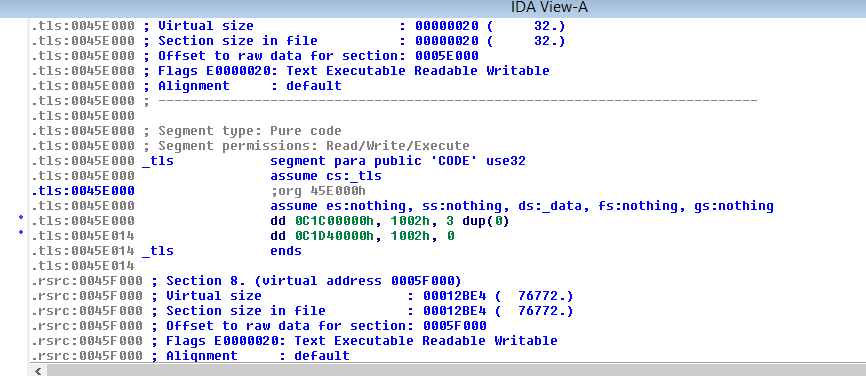
**CreateCompatibleDC**, **CreateDIBSection**, and **CreatePen** are used to create a device context and graphics objects such as pens, brushes, and bitmaps. This allows the malware to create images and graphics that appear to be legitimate and trick the user into taking certain actions.

**DeleteDC** and **DeleteObject** are used to delete device contexts and graphics objects, allowing the malware to clean up after itself and avoid detection.

**ExtTextOutA,**  **ExtTextOutW**, **GetDeviceCaps**, **GetGlyphOutlineA, GetGlyphOutlineW**, **GetICMProfileA**, **GetKerningPairsA**, **GetStockObject**, **GetTextExtentPoint32A, GetTextExtentPoint32W**, **Rectangle**, **SelectObject**, **SetBkMode**, **TextOutA** and **TextOutW** are used to manipulate and display text and graphics on the screen. This can be used to create fake error messages, notifications, or other forms of user interface that are designed to trick the user into taking certain actions.

In the case of Kovter, it was proven that the malware attempts to establish an Internet connection. This means that it could use these graphics functions to create fake error messages or notifications that appear to be from legitimate sources, tricking the user into taking certain actions that allow the malware to connect to the Internet and carry out additional malicious activities. These functions could also help log specific actions and capture screens of victims' computers and send them over the internet to the attackers.

#### 4.2.1.2 TLS & Misc

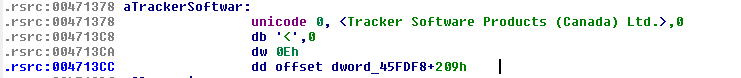


*Figure 4.1.2d - Chunk of suspicious assembly code*

With reference to the above figure( 4.1.2d), shows a portion of the thread local storage (TLS) and resource sections in Kovter's assembly code.The TLS is a special segment in the program that is used to store information that is local to each thread of the program. It is often used to store data that needs to be initialized for each new thread, such as thread-local variables, thread-local function pointers, or thread-local storage.

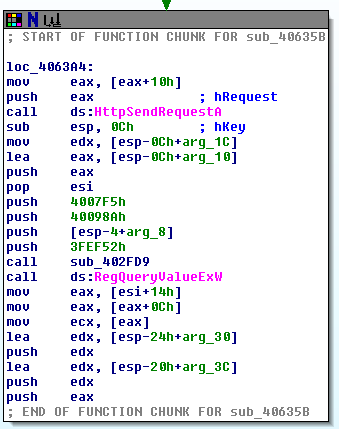
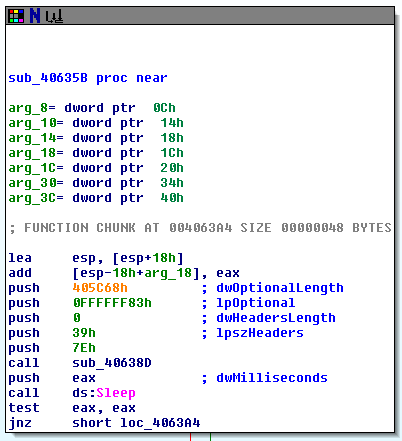
The section “.rsrc:0045F000” is a section in the program that stores resources, such as icons, images, and strings, that are used by the program. Notably, the presence of executable code in the TLS section and the resources section with flags set to "Text Executable Readable Writable" could possibly suggest that these sections may be used to store and execute malicious code.

Kovter may be using these sections to hide its presence and evade detection by anti-virus software. It could also be using the TLS section to store information about the state of each thread, allowing it to persist and continue its malicious activities even if some of its threads are terminated. This resources section may also be used to store encrypted or obfuscated payloads, or to store data used by the malware to carry out its attacks.



*Figure 4.1.2e - Chunk of assembly code*

With reference to the above figure( 4.1.2f), The assembly code references a company or software named "**Tracker Software Products (Canada) Ltd**." With further investigation, I was able to find that the company exists and that it is legitimate software company that specializes in PDF editor software, which is very similar to what Kovter was trying to claim to be.It is possible that Kovter is using this code to impersonate or hide its tracks as a legitimate software from this company.



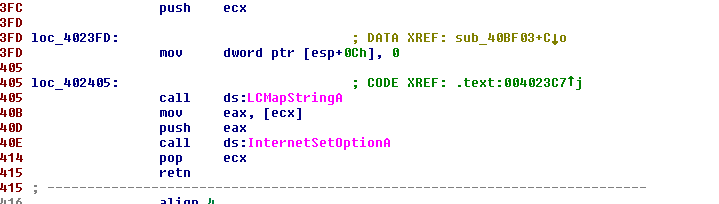
*Figure 4.1.2f - Chunk of assembly code*

The left image shows the conditions for the jump to occur which are determined by the value stored in the EAX register after the function ds:Sleep is called. If the value stored in EAX is equal to 0, the jump will occur. If the value stored in EAX is non-zero, the jump will not occur. The test instruction is used to perform a logical comparison of EAX with itself, which sets the status flags in the processor's flags register but does not modify the value of EAX.

The figure above shows a chunk of code, inside this one line calls upon the **HttpSendRequestA** function, which is used to send an HTTP request to a server. This could be used to communicate with a remote server, potentially for the purpose of sending or receiving sensitive information.

Additionally, the code calls the **RegQueryValueExW** function, which is used to query the value of a registry key. This could be used to access sensitive information stored in the registry, such as system configuration settings or user data.

#### 4.2.1.3 Network Base Indicators



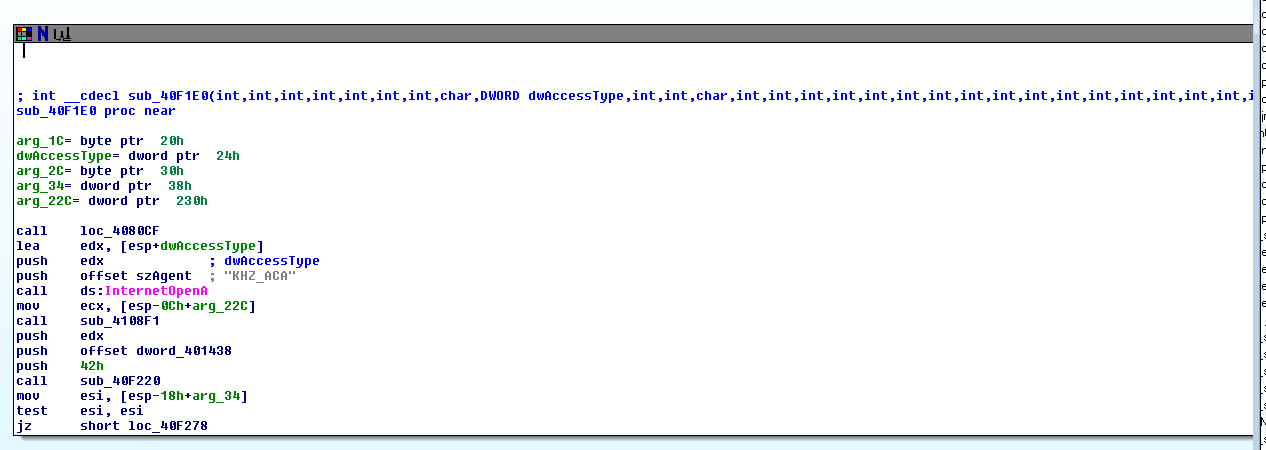
*Figure 4.1.2g - network indicators*

The above image shows a network indicator validating the claims made during basic static analysis about the Kovter malware trying to access the internet.



*Figure 4.1.2h - network indicators*

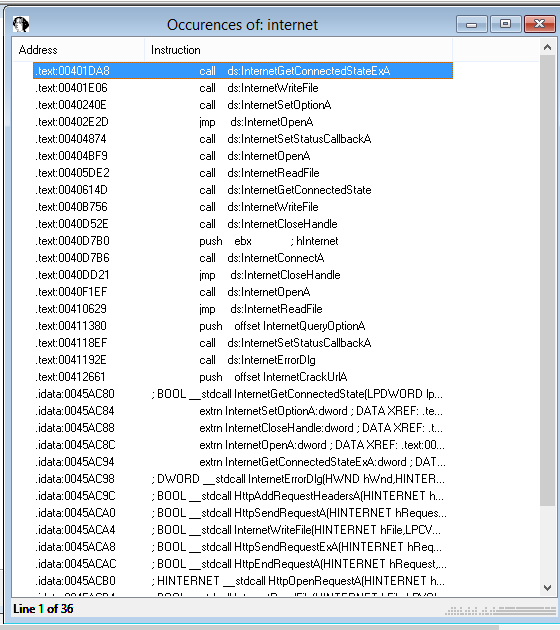
The function in the above figure appears to make use of the **InternetOpenA** function from the Wininet API. This function is used to initialize an application's use of the Wininet library, which provides access to the Internet. The fact that **InternetOpenA** is called in this code suggests that the Kovter malware is attempting to establish a connection to the Internet. The subsequent call to **sub\_408EA7** with eax (which would contain the handle returned by **InternetOpenA**) as an argument further supports the inference that the malware is trying to access the Internet



*Figure 4.1.2i - network indicators*

The above figure shows a function found in Kovter’s assembly language. This function in the code of Kovter that calls the InternetOpenA function from the Windows Internet API library indicates that the code is attempting to access the internet. This can be seen from the line "**call ds:InternetOpenA**". The **InternetOpenA** function is used to initialize a session with the internet and it can be used for various purposes, including sending and receiving data. The presence of this function in the code of Kovter suggests that the code is trying to establish a connection to the internet. Furthermore, the function is calling another function "**sub\_4108F1**" which could potentially be carrying out further actions related to the internet connection established. Additionally, the fact that the code is passing a hardcoded string "**KHZ\_ACA**" as the user agent argument in the call to **InternetOpenA** suggests that the code may be trying to mimic a specific browser or tool. Lastly, the jump "jz short loc\_40F278" will occur if the value stored in the register "esi" is equal to zero. The "jz" instruction stands for "jump if zero", and it will cause the program execution to jump to the location specified (in this case "loc\_40F278") if the zero flag in the processor's flags register is set. In other words, if the value stored in "esi" is equal to zero, the instruction will cause a jump to the specified location, otherwise the program execution will continue to the next instruction.

By utilizing IDAPro’s tool to search for a specific string we can attempt to search for string with “internet” in its name as it is a clear indication that Kovter is trying to connect to the internet

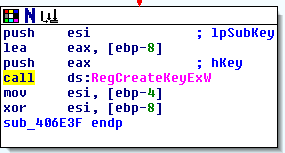


*Figure 4.1.2j - network indicators*

As seen from the above figure we can see the strings that have “internet” in its name and showing proof that Kovter calls upon such network based imports. This suggests that the malware has some connection to the internet or is using the internet for some purpose. This information can give insight into the malicious intentions of Kovter as it implies that it is trying to communicate with remote servers, access sensitive information, or perform other malicious activities that require internet access.

These network indicators help verify the claims that were made by VirusTotal, that Kovter was indeed attempting to connect to multiple addresses and URLs via the internet.

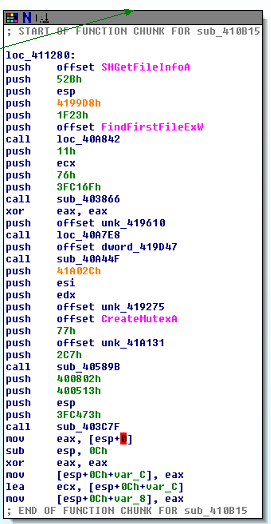
#### 4.2.1.4 Registry Key Manipulation



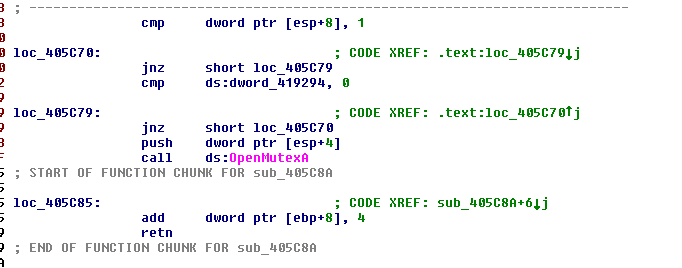
*Figure 4.1.2k - Indications of the creation of a reg key*

The above figure shows code used by malware (Kovter) and that it is using the **RegCreateKeyExW** function from the Windows API, which is used to create a new registry key or open an existing one. In the Windows registry, keys and values store configuration information for the operating system and other software. By creating a new registry key, Kovter can persist itself on the infected system and potentially manipulate or control certain aspects of the system's behavior. The use of registry keys is also a common technique used by malware to maintain persistence on a compromised system, therefore we can infer that Kovter might use this function to create registry keys that help it to stay installed on infected machines.

#### 4.2.1.5 Persistence Via Mutex Processes



*Figure 4.1.2l - Creation of Mutex*

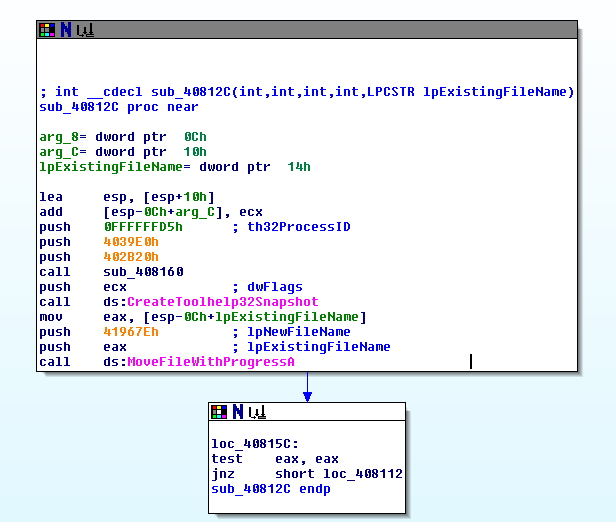


*Figure 4.1.2m - Opening Mutex*

With reference to figure 4.1.2l and 4.1.2m, we can see that there are several instances of creating and opening a Mutex (a synchronization object in the operating system) by the name "**OpenMutexA**". In the context of Kovter, it may be creating and opening a Mutex that can be used to coordinate the activities of different components or processes of the malware and ensure that only one instance of the malware is running at a given time. By creating a Mutex, Kovter can check if it's already running and avoid starting multiple instances, which could cause stability issues on the infected system.

Additionally, the use of Mutex can also be used to persist the presence of the malware on the system by locking certain resources or files, making it difficult to remove. This allows Kovter to survive across system reboots and ensures that it can continue its malicious activities even after the system restarts.

#### 4.2.1.6 Process Snapshot Tool



*Figure 4.1.2n - Usage of a process snapshot tool*

In the above figure we can see a function utilizing **CreateToolhelp32Snapshot** and a second function containing a jnz condition. The first function contains **CreateToolhelp32Snapshot,** a system API provided by the Windows operating system. It is part of the ToolHelp Library, which provides a set of functions that enable an application to gather information about system processes, heaps, modules, and threads. The second uses test to perform a bitwise operation on eax and eax, and if test returns not 0 then jnz will execute and move to 408112, otherwise it will not execute jnz and flow to the next command. The **CreateToolhelp32Snapshot** function creates a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes. The function returns a handle to the snapshot that can be used to enumerate the data in the snapshot. Coupled with information from earlier parts, we can infer that Kovter might be creating process snapshots and sending them over the internet to attackers to learn more about infected systems.

# 

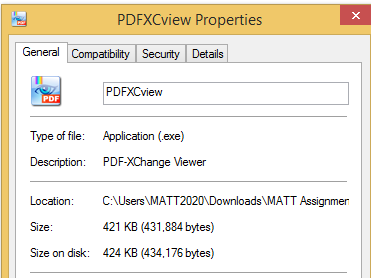
# 5 Dynamic Analysis

Dynamic analysis refers to the process of examining the executable file after file execution. Dynamic analysis is usually the second step in the analysis process, and is typically performed after static analysis has reached a dead end, either due to reasons such as obfuscation, packing or a conclusive exhaustion of available static analysis techniques. In this section, both basic and advanced dynamic analysis techniques will be performed, and for each technique, the process and findings will be documented.

## 5.1 Basic Dynamic Analysis

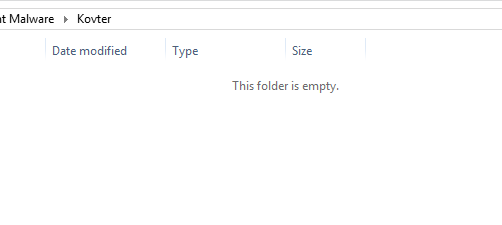
In this section, the malware will be analyzed using the basic dynamic analysis tools and techniques that were mentioned in the earlier sections.

### 5.1.1 Initial Observations Upon Executing Malware



*Figure 5.1.1a - Application properties of the malware executable*

With reference to Figure 5.1.1a, before executing the malware, we can see that the executable file is named “PDFXCview.exe”, with an application icon that resembles an eye and a PDF tag at the bottom of the icon.



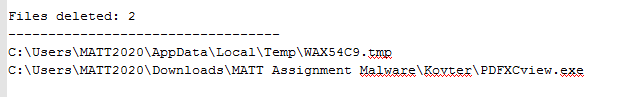
*Figure 5.1.1b - Executable is no longer in the folder upon execution*

With reference to Figure 5.1.1b, after executing the malware, we can see that the executable file immediately disappears, which suggests that the executable has found some other method to remain persistent on the system, either by renaming and installing itself onto another location, or via other unconventional methods. This observation suggests that we should start by analyzing the registry to see if any interesting registry changes were found.

### 5.1.2 Registry Analysis

As mentioned in the earlier sections, Regshot will be used to analyze registry changes upon malware execution. This is done by taking one registry snapshot before malware execution, and a second registry snapshot after executing the malware, followed by comparing both snapshots. This allows us to observe any changes to registry keys, files, processes and more that the malware performs, and allows us to gain a deeper understanding of the functionality of the malware.

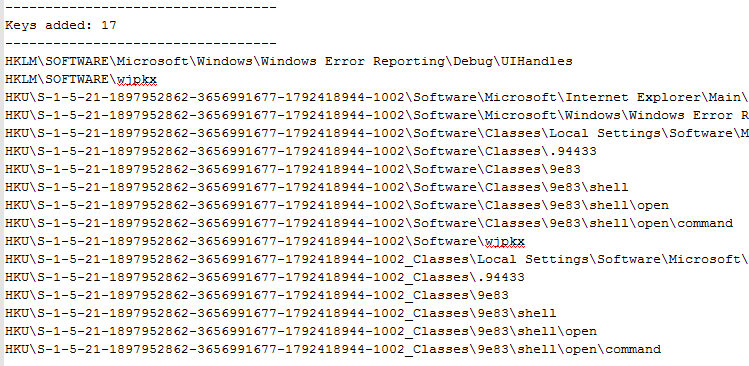
**Files Deleted**



*Figure 5.1.2a - Executable deletes itself upon execution, as shown by Regshot*

As seen from Figure 5.1.2a, the first thing we can note from comparing the registry snapshots is that the executable “PDFXCview.exe” indeed deleted itself upon execution.

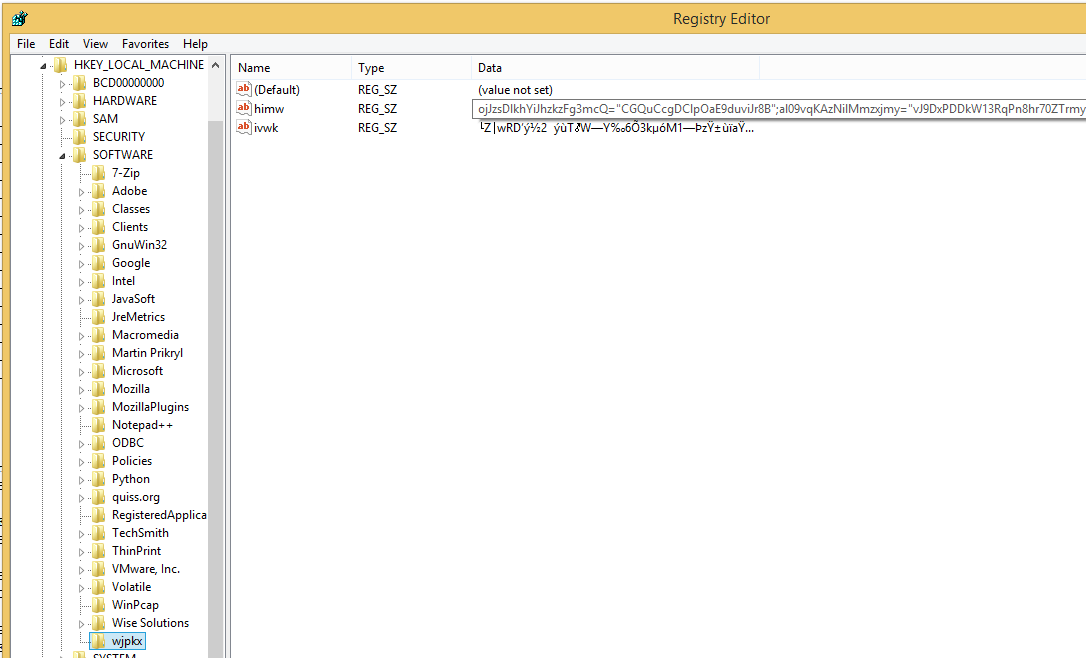
**Keys added**



*Figure 5.1.2b - Registry keys added by the executable*

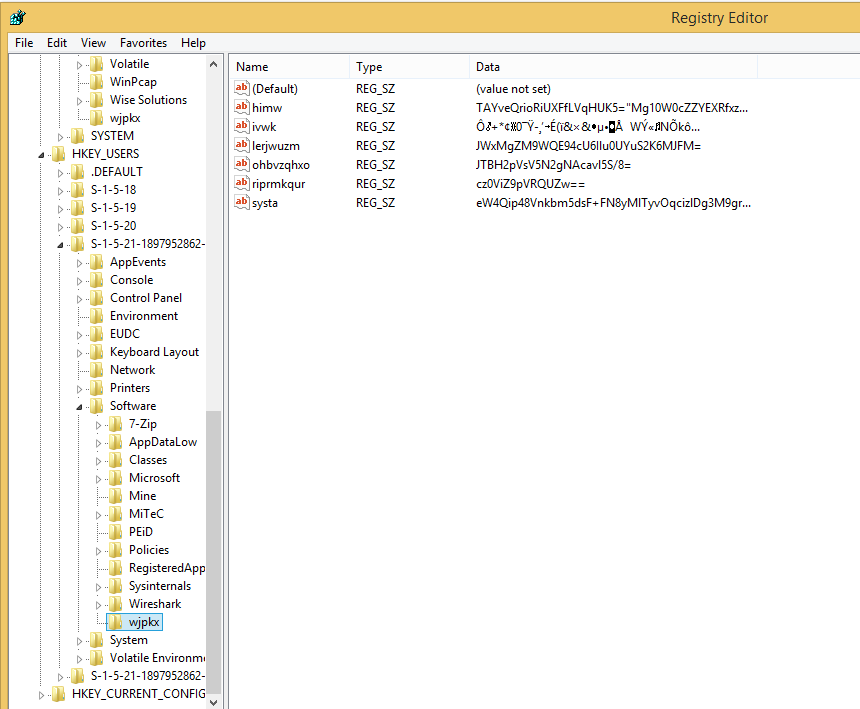
Next, Figure 5.1.2b shows the list of registry keys added by the executable. From the figure, the first thing to note is that the malware adds a considerable number of registry keys under the \...\shell registry hive, including **\shell\open\command** that suggest that the malware intends to access the command shell of the system, possibly to execute some form of shell code using a command line interpreter.

In addition, the malware also adds a registry key named **‘wjpkx’** to the HKLM registry tree, also known as HKEY\_LOCAL\_MACHINE, which is a Windows registry tree that contains configuration data used by all Windows users of that machine. This registry key name **‘wjpkx’** was also added to the HKU registry tree, otherwise known as HKEY\_USERS, that stores user-level configuration data.



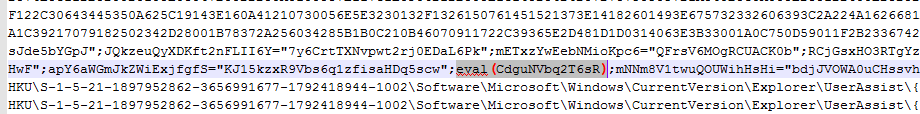
*Figure 5.1.2c - Using regedit to view the newly created registry key ‘wjpkx’ under HKLM*

With reference to Figure 5.1.2c, upon using regedit to view the newly created registry key **‘wjpkx’** under the HKLM registry tree, we can see that there are two values added **‘himw’** and **‘ivwk’** within that registry key.



*Figure 5.1.2d - Using regedit to view the newly created registry key ‘wjpkx’ under HKU*

Thereafter, with reference to Figure 5.1.2d, upon using regedit to view the newly created registry key **‘wjpkx’** under the HKU registry tree, we can see much more values added in that registry key, with naming conventions that seem equally gibberish and randomly generated such as ‘lerjwuzm’. We can also observe that the values “**himw**” and “**ivwk**” that exist in HKLM\Software\wjpkx also exist in HKU\Software\...\wjpkx, but the values if you compare closely between both figures, are different. In fact, while it may seem that the data stored in the ‘wjpkx’ registry keys are completely uninterpretable and meaningless, closer inspection in Regshot show indication that these values contain obfuscated Javascript code.



*Figure 5.1.2d - Indication of Javascript code found under HKLM\Software\wjpkx\himw*

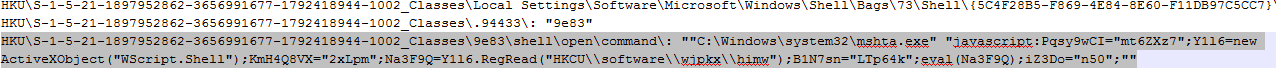
As seen from Figure 5.1.2e, the highlighted line of code observed in Regshot shows an **eval** function being called towards the end of the data added under **HKU\**

**Software\wjpkx\himw**, highly implying that the block of code is written in Javascript, and that the code attempts to execute a payload that may possibly be malicious.



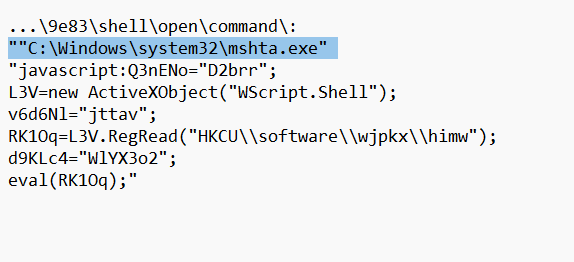
*Figure 5.1.2e - Formatting the obfuscated code*

After uncovering this, the entire chunk of obfuscated code was copied from Regshot to PDF Stream Dumper, which is a tool used to format Javascript code for better readability, as shown in Figure 5.1.2e. From the figure, we can see some code structures such as ‘for’ loops and functions such as String.fromCharCode(). Looking into the ‘for’ loop structures , we can even see a ‘^’ symbol, indicating that the Javascript code performs some kind of XOR operation, possibly as a means to encode and further obfuscate the code. To not deviate from the aim of conducting basic dynamic analysis in this section, analysis of the obfuscated Javascript code will be performed in a subsequent section, ([see Section 6.1](#_g13d1p1g6nsb)) but for now, we will continue with the basic dynamic analysis of the malware.



*Figure 5.1.2f - Shell command invoked upon executing malware*

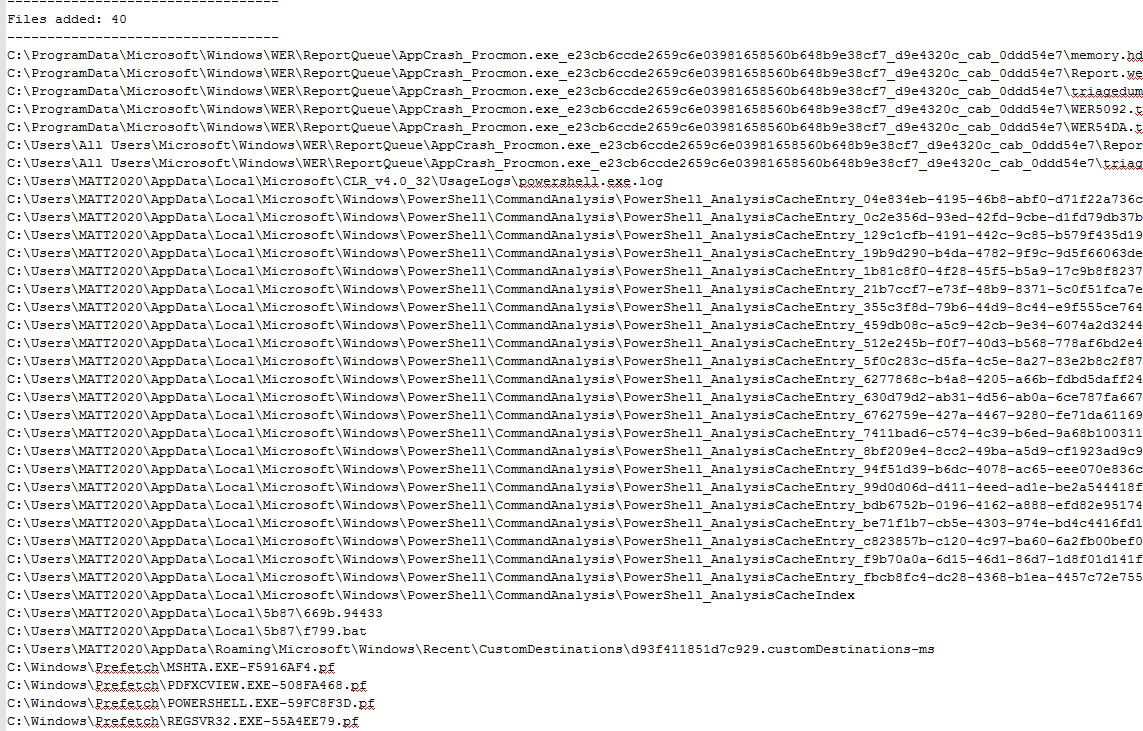
Looking back into the registry snapshot comparisons obtained from Regshot, as shown in Figure 5.1.2f, we can see that through adding another **\shell\open\command** registry key value, the malware also attempts to invoke another shell command:



*Figure 5.1.2g - Clearer image of the shell command invoked by the malware*

As seen from Figure 5.1.2g, the malware first attempts to run the **mstha.exe** binary, which is a utility that executes Microsoft HTML Application (HTA) files. In other words, this process allows scripting languages such as Javascript, VBScript or JScript to be executed through passing in an argument to the command line of a machine. After running the mstha.exe program, the malware attempts to execute Javascript payload using a command-line interpreter, as shown from “**javascript:**”. However, as seen from the figure, the following block of code also seems to be obfuscated. Since this block of code is also obfuscated Javascript code, a deeper analysis of this block of code will be performed later in the report as well ([see Section 6.2](#_dbzbntyidar)).

**Files Added**



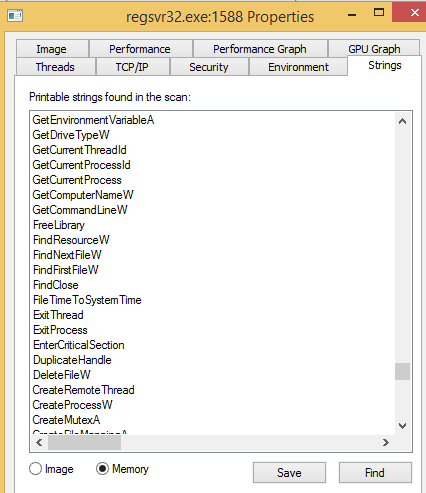
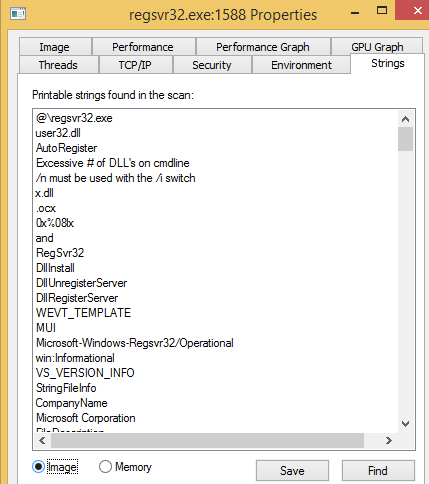
*Figure 5.1.2h - Files added by the executable*

Figure 5.1.2h shows the files added by the executable upon execution. From the figure, we can distinctly see that the executable is interested in the PowerShell application provided by Windows, which functions similar to the Command Prompt. From C:\Windows\Prefetch\POWERSHELL.EXE-508FA568**.pf,** we can even deduce that the Powershell application was executed, as the creation of a prefetch file in Windows indicates that the app was run for the first time, since malware execution. Apart from Powershell, the mstha.exe process was executed as well, followed by the executable PDFXCView.exe, and lastly **regsvr32.exe**. It is worth keeping an eye on regsvr32.exe, because while it is a legitimate process, the malware could have conducted some form of process injection, and injected itself onto the regsvr32.exe process for further persistence. With this in mind, we can now proceed to analyze the processes, to see if any more interesting information can be obtained.

### 5.1.3 Process Analysis

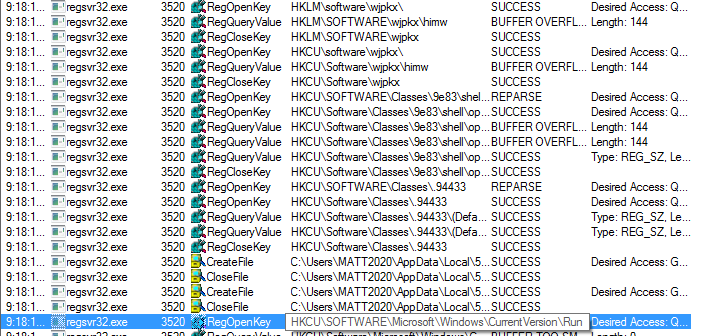
Process Analysis will be conducted using the Process Monitor and Process Explorer tools. These tools are essential in performing basic dynamic analysis, as they provide insight to running processes, and monitor activities in the registry, file system and more.

As mentioned in the previous section, the **regsvr32.exe** process should be something to look out for, therefore, process analysis will first be performed by viewing the regsvr32.exe process using the Process Explorer tool.



*Figure 5.1.3a - LEFT: Strings in Image of regsvr32.exe; RIGHT: Strings in Memory of regsvr32.exe*

One way to determine whether any process injection has occurred is to view the strings that the process uses. From comparing the slider bar of the left and right images in Figure 5.1.3a, we can clearly see that more strings were added to the process, as the slider bar shown in the left image is noticeably shorter than the slider bar on the right image. This indicates that any strings that did not exist in the image of the process but exist in the memory of the process belong to other applications, likely the malware executable, that has injected itself onto this process. In fact, from the left image in the above figure, we can see strings that show some host-based indicators, such as ExitProcess, DeleteFileW and more, all of which were APIs that were seen to be imported from a DLL while conducting static analysis on the executable file, highly suggesting that the executable did indeed inject itself into this process.

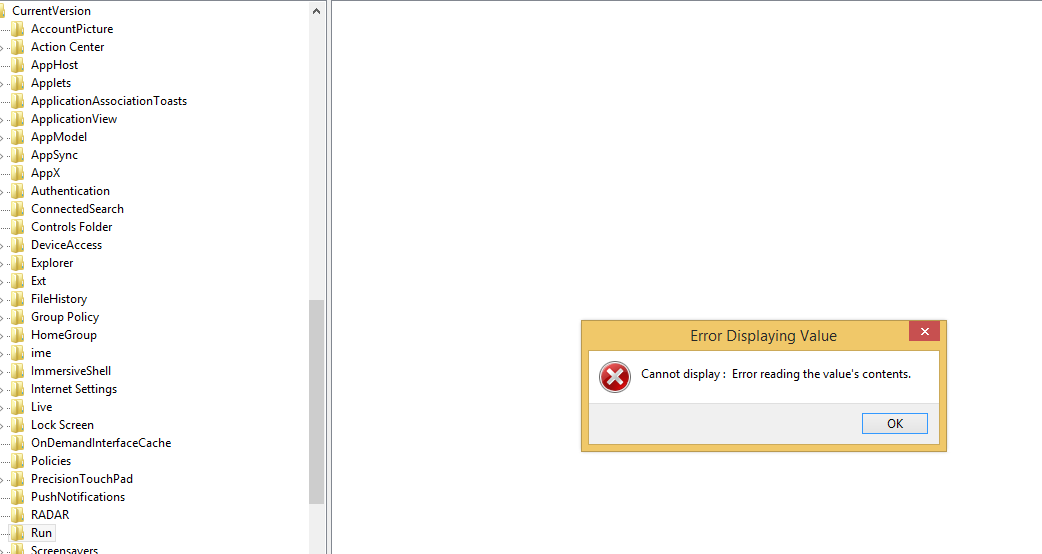


*Figure 5.1.3b - Filtering the process monitor tool by the ‘regsvr32.exe’ process name*

With reference to Figure 5.1.3b, navigating to Process Monitor further supports the prior hypothesis, as the regsvr32.exe process was clearly used to perform the functionalities of the Kovter malware. As seen from the figure, the process can be seen opening registry keys such as HKLM\SOFTWARE\wjpkx\himw that we saw in the earlier section, and also creating and closing some files in the \...\AppData\Local folder.

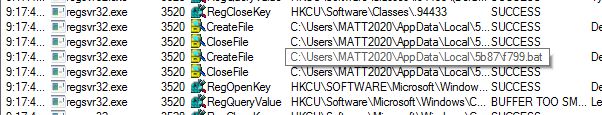
In addition, we can see that the HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\

Run registry path was opened as well, suggesting that the malware might have added values to this registry path as a persistence mechanism. This persistence mechanism is common among malware, since the \..\Run registry path automatically runs any programs stored in this path when the system starts up.



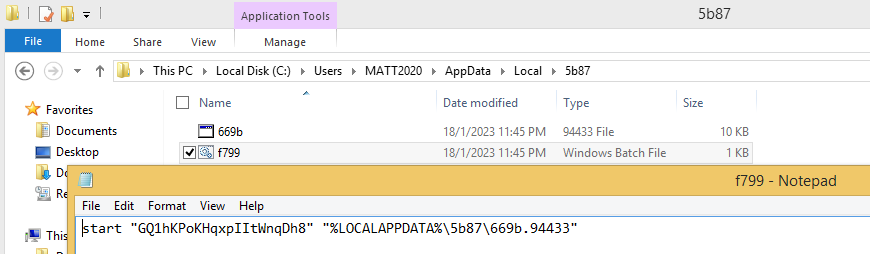
*Figure 5.1.3c - Malware prevents user from opening the Run registry key*

With reference to Figure 5.1.3c, upon navigating to this registry key using regedit, a pop up appears that says “Cannot display: Error reading the value’s contents”, suggesting that the malware prevents an infected user from viewing the Run key using the Registry Editor.



*Figure 5.1.3d - Malware creates and closes a batch file*

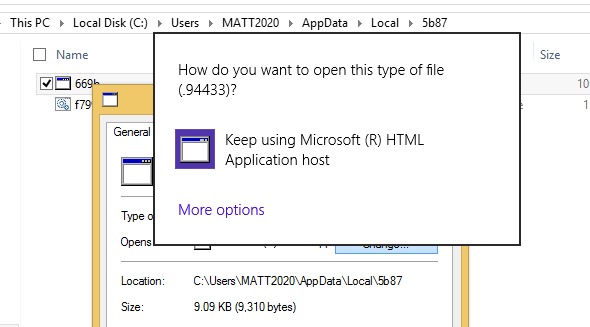
Looking deeper into the file operations performed by the executable, we can see that the malware creates a batch file **f799.bat**, as seen from Figure 5.1.3d. The creation of this batch file is suspicious, because batch files can be used to temporarily serve as a placeholder to store code that the malware wishes to execute. Therefore, to further explore this, the batch file was opened by navigating to the file path depicted in the above figure.



*Figure 5.1.3e - Contents of the batch file*

Figure 5.1.3e shows the content stored in the batch file created by the malware. The first striking thing that we can observe is the “start” command being invoked, which already suggests that the code stored in this batch file is attempting to start a service. Thereafter, we see a file path “\%LOCALAPPDATA%\5b87\669b.94433”, which is the file path that we are currently at. Looking at the folder in the above figure, we see another file above f799.bat called **669b.94433,** which is the file referenced in code.

Some obfuscation attempts were definitely made to conceal the file type of the 669b file, since 94433 is not a legitimate file extension. By viewing the properties of the 669b file, we see that the file can be opened using Microsoft (R) HTML Application Host, or in other words, **mshta.exe**:



*Figure 5.1.3f - Contents of the batch file*

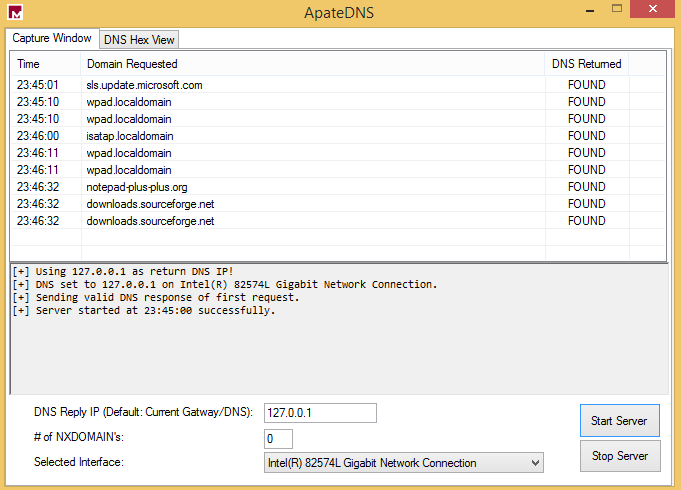
This attempt to open the mstha.exe binary indicates that the 669b.94433 file contains some form of script code that attempts to be executed via running the mstha.exe process, which as mentioned above, was the only way non-shellcode code could be executed on a command shell.

### 5.1.4 Network Analysis

As mentioned in the previous sections, network analysis of the malware will be performed using tools such as ApateDNS and Wireshark.

**Apate DNS**

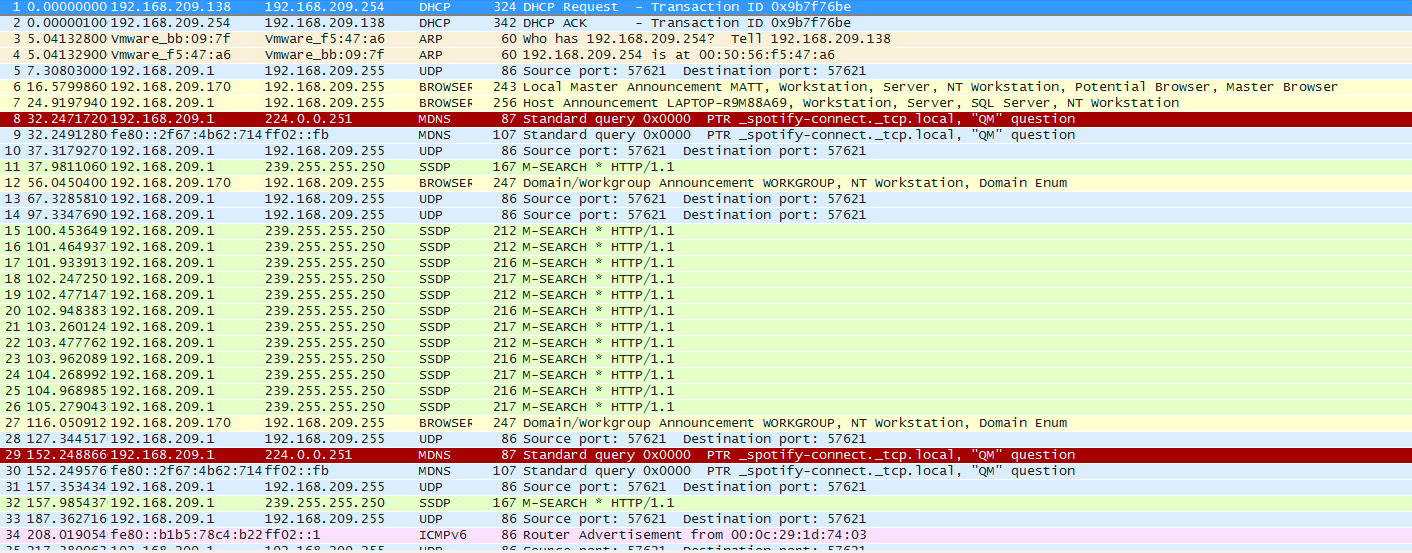
Upon setting the DNS server to 127.0.0.1 and, starting the server and executing the malware, no initial observations could be made:



*Figure 5.1.4a - ApateDNS results upon executing the malware*

As seen from Figure 5.1.4a, the results in ApateDNS are all noise, and not actual connections that were established between the malware and any domain addresses. Firstly, the wpad.localdomain domain is just the Windows machine trying to verify the existence of a proxy. Secondly, sls.update.microsoft.com is just a normal domain address Windows uses to perform software updates. Thirdly, notepad-plus-plus.org could have been invoked because the Notepad++ application was used to compare registry snapshots in Regshot, and lastly, downloads.sourceforge.net is nothing more than the domain address ApateDNS can be downloaded from.

**Wireshark**

****

*Figure 5.1.4b - Wireshark results upon executing the malware*

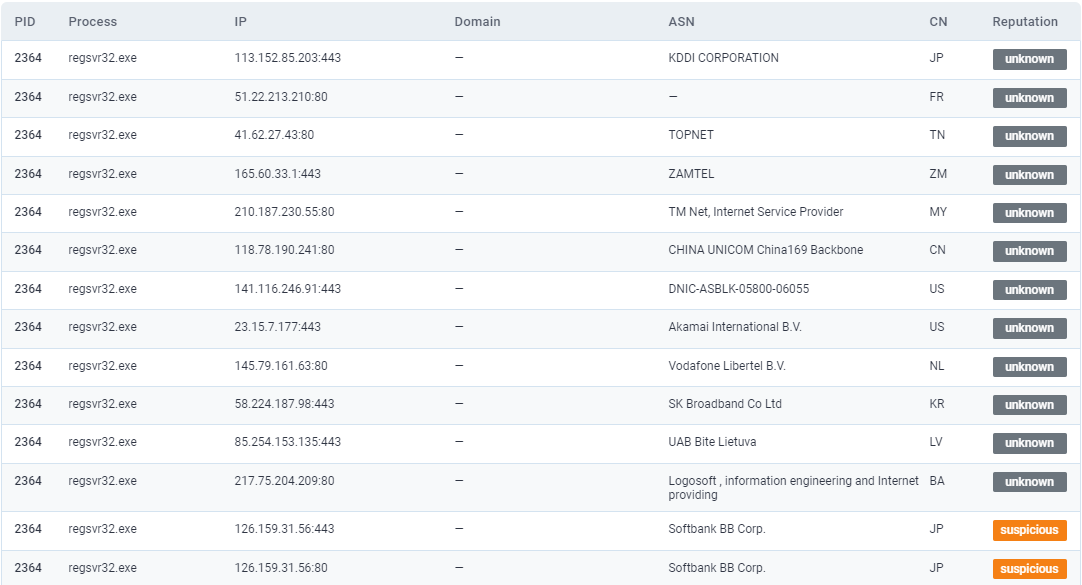
In addition, with reference to Figure 5.1.4b, Wireshark was not able to obtain any actual network-based indicators as well. As seen from the figure, there was no indication that any HTTP/HTTPS ports were being used, or any connections being established after the malware was executed.

In short, upon using ApateDNS and Wireshark, no unusual network behavior was found, which was strange considering the malware had imported several network-based APIs such as InternetReadFile and InternetOpenA. With such inconclusive and awry results, we could only deduce that either:

1. The malware was aware that it was being executed in a VM environment and therefore did not reveal some of its functionalities or
2. The malware was able to obscure and cleverly hide any connections that were being established to a remote server

As such, to test the first hypothesis, instead of running the malware in a VM environment, the malware was run in a sandbox environment instead, to see whether the malware would display any network-based indicators under a different environment.

Sure enough, upon running the malware in the [any.run](https://any.run/) interactive online malware sandbox, we were able to see that the malware did indeed establish some connections to different servers via ports 80 and 443:



*Figure 5.1.4c - Connections made by the malware*

As seen from Figure 5.1.4c, we can see that the malware attempts to establish multiple connections, each connection to a different remote server via an IP address. Upon looking at the organization names being assigned to these IP addresses, we can identify that the malware attempts to establish connections to multiple completely legitimate organizations. For example, Softbank BB Corp. is a multinational conglomerate holding company and Akamai International B.V. is a media and software services provider. This highly suggests that the malware could be a botnet, that aims to either conduct a Distributed Denial of Service (DDoS) attack on such organizations, or to steal sensitive information from such organizations, or simply to send spam.

## 

## 5.2 Advanced Dynamic Analysis

As mentioned, Advanced Dynamic Analysis of the Kovter malware will be conducted using the OllyDBG x86 debugger tool. This section will be split into three parts, namely observing the import address table and memory map, then observing interesting functions in the disassembler window and lastly debugging individual functions by setting breakpoints.

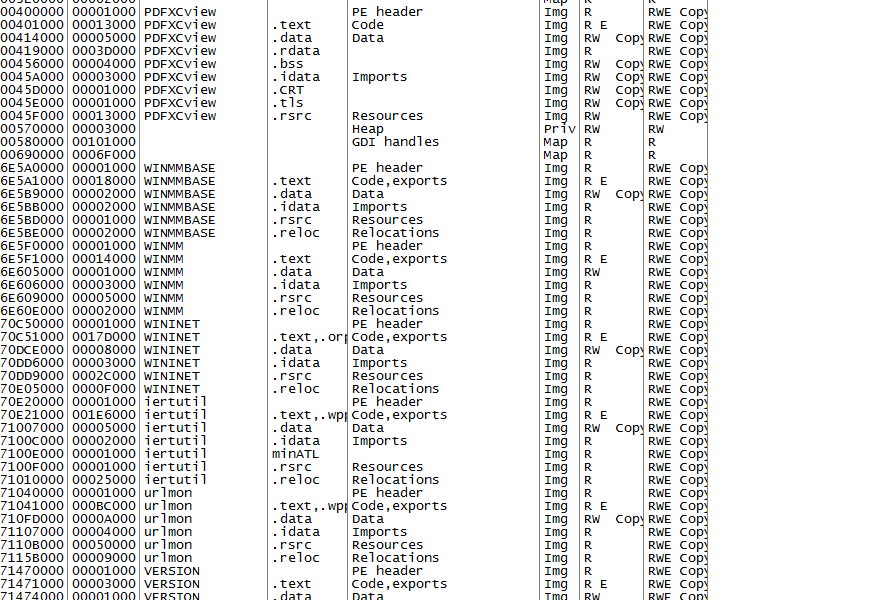
### 5.2.1 Import Address Table and Memory Map

Firstly, from importing the Kovter executable into OllyDBG and pressing Ctrl+N, we get introduced to the import address table used by the malware sample:



*Figure 5.2.1a - Import address table of the Kovter malware*

Observing the import address table is a typical technique employed by malware analysts when first using OllyDBG to analyze their malware sample. This is to look at the list of APIs imported by the malware, to ensure that the malware is unpacked before performing any debugging operations. As seen from the Figure, we can see that there is a long list of APIs imported by the malware sample, suggesting that it is already unpacked, so no unpacking operations need to be conducted before debugging the malware.

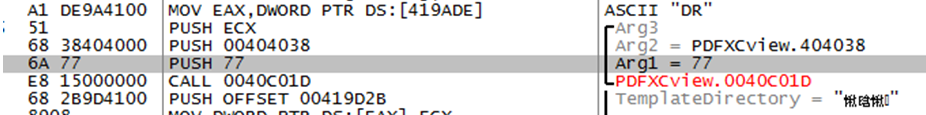
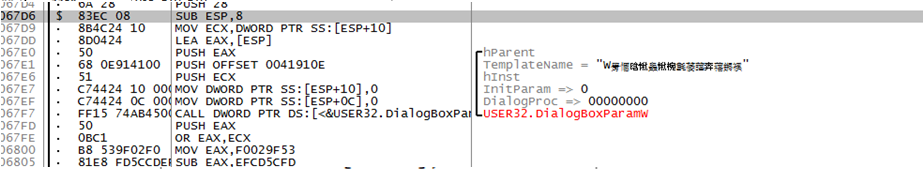
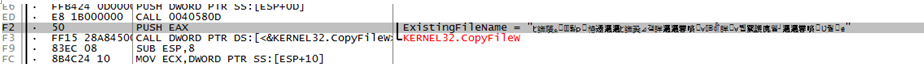


*Figure 5.2.1b - Memory map of malware*

Thereafter, as seen from the above figure, we look at the memory map of the malware sample. By viewing the memory map, we are able to observe a comprehensive overview of the executable’s memory usage, allowing us to easily identify areas of memory that are allocated, reserved, or unused. As seen from the above figure, the malware sample frequently rebases to different PE addresses, which could indicate that the malware attempts to evade detection from anti-virus software or reverse engineering tools. This is because frequent rebasing of a PE address makes a malware harder to be detected by anti-virus or reverse engineering tools, especially those that rely on specific memory addresses or offsets to analyze the behavior of the malware.

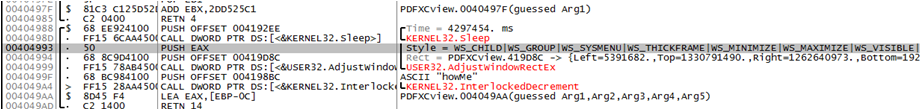
### 5.2.2 Disassembler Window

After looking through the import address tables and memory map, we can start to look into the malware sample. We start off by first looking in the Disassembler Window:



*Figures 5.2.1c - 5.2.1e - Obfuscated values stored in parameters pushed onto the stack*

From briefly looking through the Disassembler window, we see several references to parameter values under different memory addresses that are clearly either obfuscated or random characters. As seen from the above figures, we can see that the parameters **ExistingFileName**, **TemplateName** and **TemplateDirectory** all contain unintelligible values, which could either be an obfuscation attempt by the malware or just random noise meant to distract malware analysts from the functionality of the malware.



*Figure 5.2.1f - Kernel32.Sleep API being invoked, with a time of 4297454 ms*

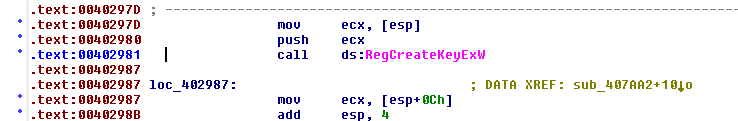
Next, with reference to Figure 5.2.1f, we can see at the memory address 00404988 the **Kernel32.Sleep** API being invoked. From the line above, we can see that the time parameter that is pushed onto the stack has a value of 4297454 ms, which indicates that the function attempts to pause the execution of the program for a time of 4297454 ms.

It can also be noted that the use of the sleep function could be a defense mechanism employed by the malware to evade dynamic analysis techniques. This is because when dynamic analysis is performed, malware analysts would usually look for immediate changes in the system, but if a sleep function is called, the malware would wait for a specified period of time before executing its functionalities, giving malware analysts the false impression that the executable that was ran dynamically has no malicious payloads. This could also explain how in the earlier section, when dynamic network analysis was performed, we were not able to see any network indicators from the malware despite knowing that the malware has imported several network APIs.

### 5.2.3 Debugging Function Calls using Breakpoints

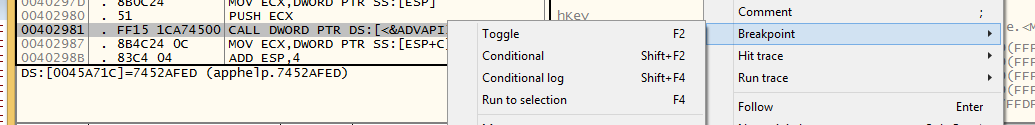
Since from the basic dynamic analysis section we were able to identify that the Kovter malware was able to perform a large amount of registry manipulation, a good starting point for debugging in OllyDBG would be to observe the APIs such as **RegCreateKeyExW**, **RegOpenKeyEx**, and/or any other APIs imported from the ADVAPI32 DLL. In order to observe and control the execution of these function calls at a granular level, we would need to set individual breakpoints and observe the values represented in the stack and registers.

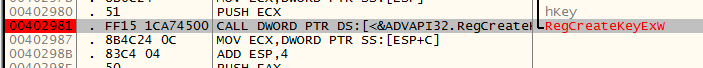
To do that, we first open up IDA pro and look for memory addresses in which these function calls were referenced.



*Figure 5.2.3a - RegCreateKeyExW referenced in memory address 00402981*

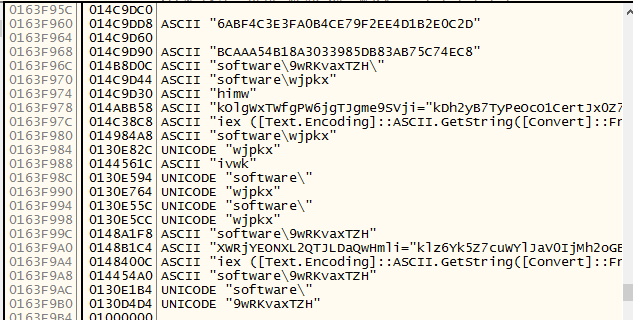
As seen from the above figure, we were able to deduce in IDApro that a function call referencing **RegCreateKeyExW** was made at memory address **00402981.** Therefore, to observe the functionalities of this function call, we can navigate to this memory address in OllyDBG and set a breakpoint:





*Figures 5.2.3b and 5.2.3c - Setting a breakpoint at memory address 00402981*

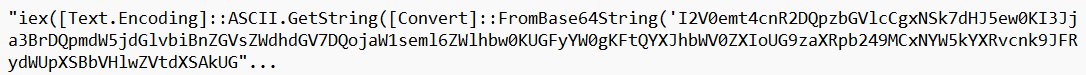
As seen from the above figures, we can set a breakpoint at memory address 00402981 by right-clicking the line, selecting “Breakpoint” -> “Toggle”. A successful breakpoint is denoted in OllyDBG by a red highlight over the memory address, as seen in the above figure, where 00402981 is highlighted in red. Thereafter, by pressing Fn+F9 to run the debugger and observing the registry and stack values, we get to see the functionalities performed by the **RegCreateKeyExW** function call:



*Figure 5.2.3d - Stack values added by RegCreateKeyExW*

Upon running the program, the debugger reaches the breakpoint we set at memory address 00402981 and stops. As expected, with reference to Figure 5.2.3d, we can see that the values in the stack window are registry keys the Kovter malware adds, and we can observe some familiar registry keys such as the **wjpkx** registry key that we saw appear earlier when we were comparing registry snapshots ([see Section 5.1.2](#_qk6dwxrmk3g8)).

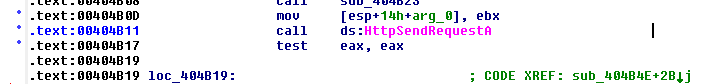
We also see some code that appears to be obfuscated. By copying the values and pasting it on a text editor, we roughly see that these values added to the registry are Base64-encoded:



*Figure 5.2.3e - Stack values added by RegCreateKeyExW*

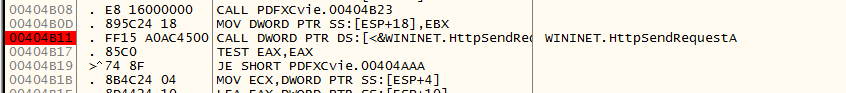
Therefore, from manually debugging the **RegCreateKeyExW** function, we get an idea that the malware author uses this function from the ADVAPI DLL to create registry keys, add scripts within the values of these registry keys and uses Base64 encoding to further obfuscate these scripts.

Next, we repeat this process for some WININET functions such as **HttpSendRequestA**. Again, through IDApro, we were able to determine the memory address where the **HttpSendRequestA** function is called to be **00404B11**:



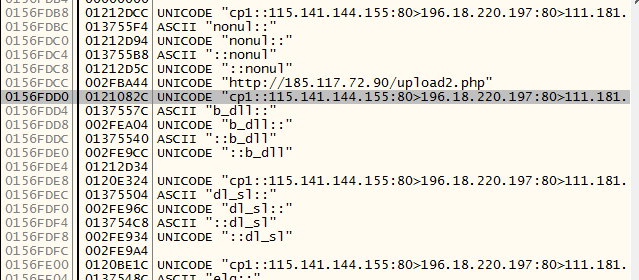
*Figure 5.2.3f - Memory address that references HttpSendRequestA*

Therefore, we navigate to this memory address in OllyDBG and set a breakpoint:



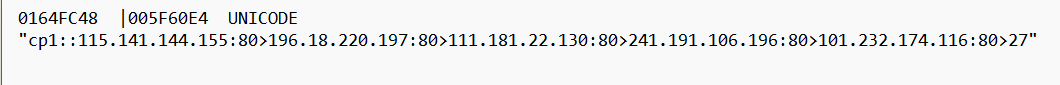
*Figure 5.2.3g - Setting a breakpoint at 00404B11*

With that, upon execution, we then view the data added to the stack window:



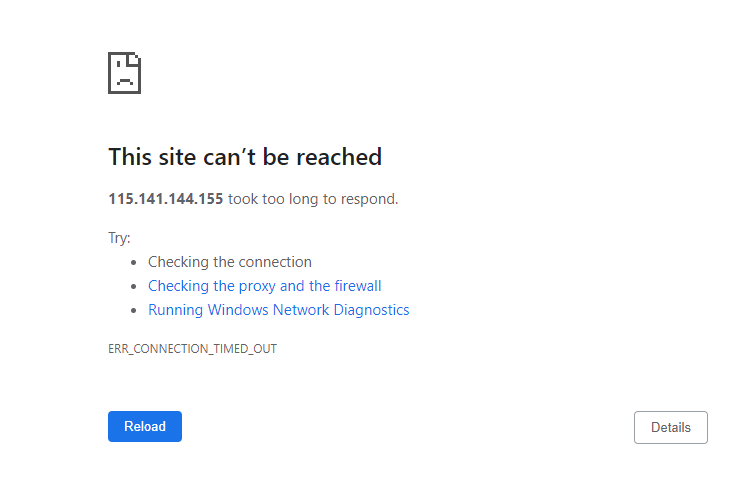
*Figure 5.2.3h - Stack values added upon execution of HttpSendRequestA*

With reference to Figure 5.2.3h, we see several network-based indicators.



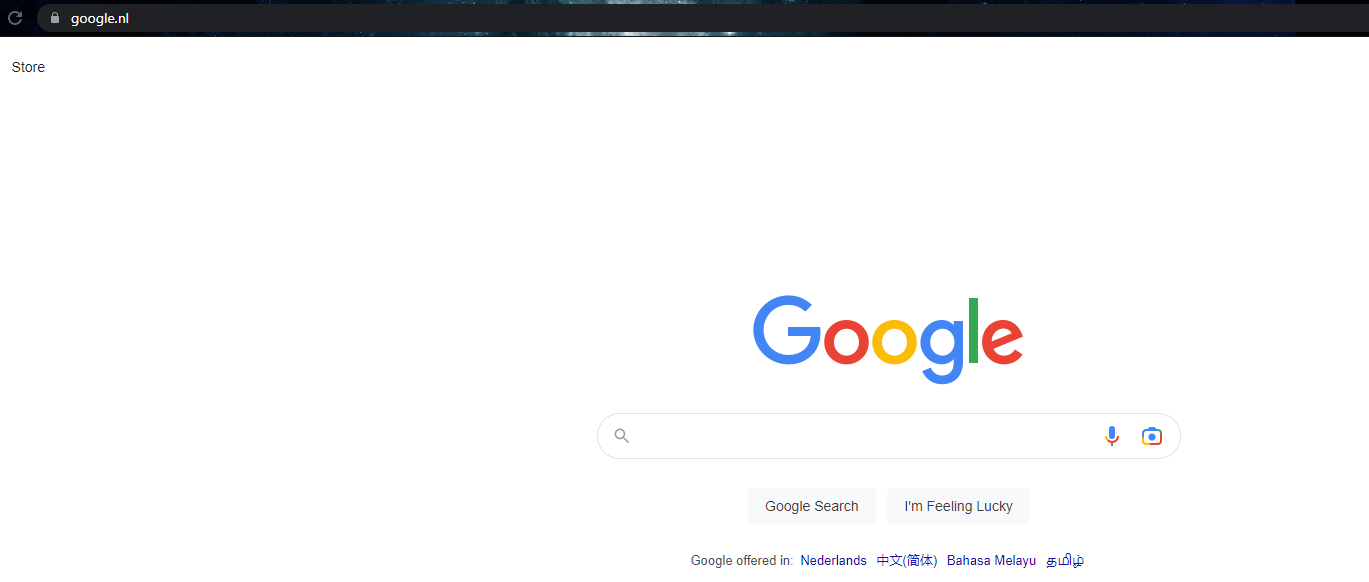
*Figure 5.2.3i - One string referenced several times in the stack window*

As seen from Figure 5.2.3i, there was one string in particular that was referenced several times within the stack window. By copying the contents of the string and pasting it onto a text editor, we see that the string contains 5 different remote IP addresses, and attempts to connect to each of these IP addresses through port 80 (HTTP). When we attempt to manually access these IP addresses through a browser, we are unable to establish any connection:



*Figure 5.2.3j - Unable to visit these domain addresses*

Next, we see another string of an IP address <http://185.117.72.90/upload>.php. Navigating to this domain address, we see that it is an IP address linked to Google:



*Figure 5.2.3k - Domain address of 185.117.72.90*

This is an indication that the malware may be a botnet, as it is sending random HTTP requests to different domain addresses that seem to have no malicious intent. It is interesting to note that the google address has a .nl tag, which indicates that this google platform is catered towards the Netherlands, which could suggest that the origin of the malware sample is from the Netherlands as well.

*Figure 5.2.3l - Further indication of botnet behavior*

Lastly, with reference to Figure 5.2.3l, we can see that after executing another WININET function, several domain addresses are listed and referenced, all of which are legitimate websites, that further allude to botnet behavior from this malware sample.

# 

# 6 Obfuscated Code Analysis of Malware

In this section, as a continuation from the dynamic analysis findings, we will attempt to analyze the obfuscated code that was discovered to be created by the malware under the **Basic Dynamic Analysis** section. By analyzing the obfuscated code, this section will aim to gain a better understanding of what the intention of the obfuscated code is, and how it plays a part in the functionality of the malware. Since both the static and dynamic analysis of the malware has already been performed, this section would be the last section where a thorough analysis of the malware’s functionalities will be conducted, before we have a general idea of what the malware does. This will allow us to link everything together and provide a full documentation of the functionalities, behavior and characteristics of the malware in the next section.

As seen in the Basic Dynamic Analysis section, two chunks of obfuscated code were found to be created by the malware. Therefore, this section will be split into two parts, where each part will be analyzing one chunk of obfuscated code.

## 

## 6.1 Obfuscated Code 1

As mentioned in the earlier sections, the first chunk of obfuscated code that was identified resided under the HKU\Software\wjpkx\himw registry key, which was a registry key created by the malware itself. The block of code was copied and pasted onto the PDF stream dumper application, where it was properly formatted for better readability. The figure below depicts a portion of the block of code:

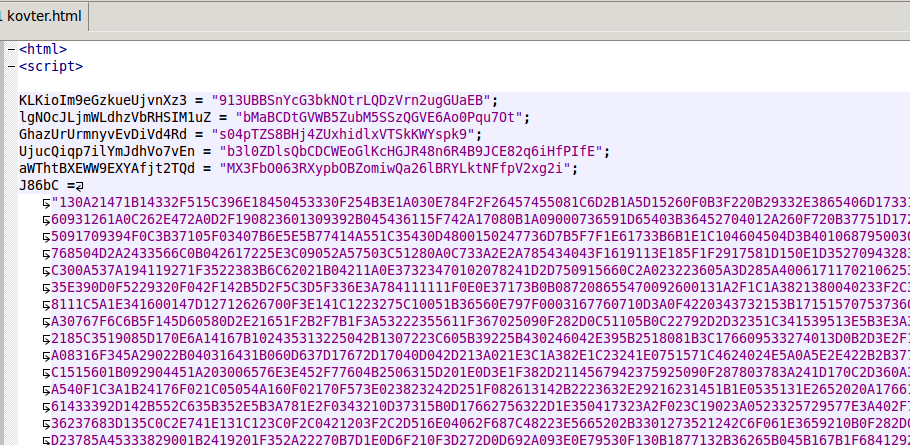


*Figure 6.1a - The obfuscated Javascript code*

Just to clarify, we can deduce upon initial triage that this block of code is Javascript due to the “eval()” function being called as seen in the above figure. Since we know that this block of code is obfuscated Javascript, we can use the Firebug tool to further analyze this code. To elaborate, Firebug is a Mozilla Firefox browser extension that can be used to inspect and debug HTML, CSS and Javascript code on a webpage. Since the code is obfuscated, we are unable to deduce the function of the code simply by reading the code. Therefore, by using Firebug, we are able to gain a better understanding of what the code does through debugging the code.

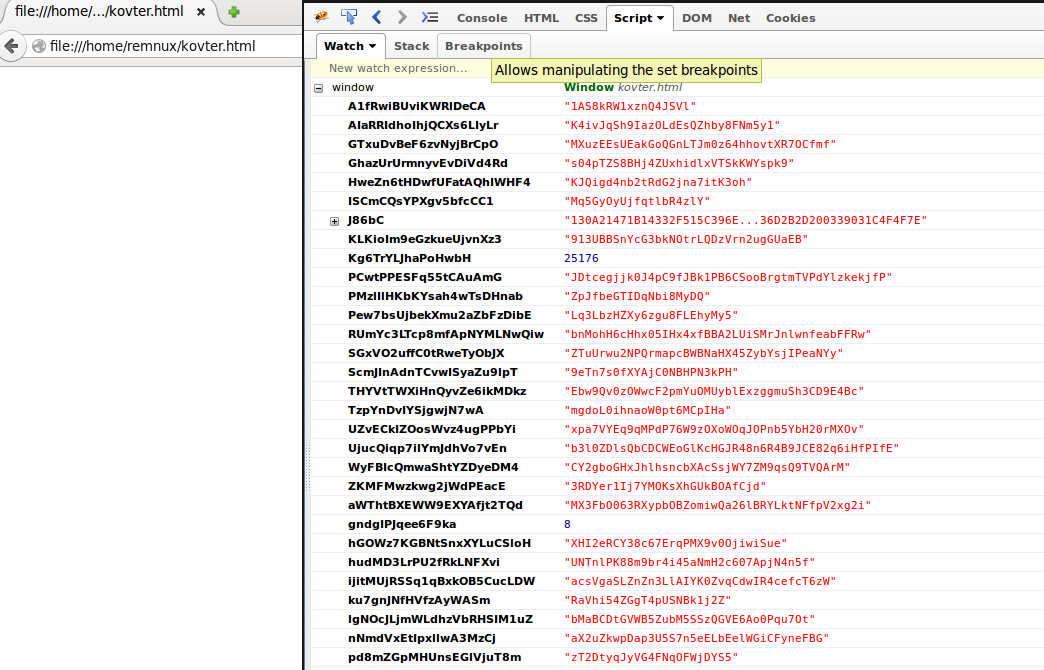
### 6.1.1 Debugging the Code using Firebug

To use Firebug, we first copy the obfuscated code that is Javascript-formatted and paste it into a text editor. Then we add <html> and <script> tags to the start and end of the block of code, and save it as a HTML file:



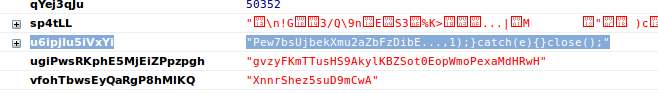
*Figure 6.1.1a - Saving the block of obfuscated Javascript code as a HTML file*

As shown in Figure 6.1.1a, after creating the HTML file, we can proceed to run the HTML file in the Mozilla Firefox browser, and open Firebug:



*Figure 6.1.1b - Saving the block of obfuscated Javascript code as a HTML file*

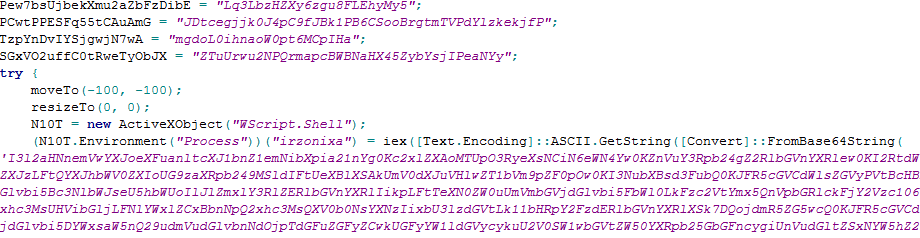
As shown in Figure 6.1.1b, after opening Firebug, we can proceed to analyze each variable individually to look for interesting results.



*Figure 6.1.1c - A variable that stores a “catch and close” structure*

With reference to Figure 6.1.1c, after looking through some variables, we are able to spot one variable in particular that has a catch and close code structure. This is worth looking into because a catch and close structure suggests that there is an attempt to validate the code such that in the event that the code encounters an error, the program does not crash, which indicates some effort was made to preserve the integrity of the code. Therefore, the value that was assigned into this variable was copied and pasted into PDF stream dumper, where it was formatted.

### 6.1.2 Analyzing Interesting Variables



*Figure 6.1.2a - A portion of the code assigned into u6lpjlu5iVxYi variable*

As seen from Figure 6.1.2a, after PDF Stream Dumper formats the results, we can clearly see that within the “try” block, the code first creates an ActiveX Object called WScript.Shell and stores it in **N10T**. To elaborate, an ActiveX Object is a software component used in Microsoft Windows operating systems, and is typically used to add functionality to a variety of applications or web browsers. Thereafter, the **N10T** variable creates an environment variable “**iraonixa**” and stores a string that was just decoded, using the Convert.FromBase64String() method into the environment variable.

Following this, navigating to the bottom of the code, we see that another variable, **K6Akk**, was created, and assigned to run **powershell.exe**:



*Figure 6.1.2b- The code used to run powershell.exe*

In Figure 6.1.2b, we can also see that the code specifies through the *iex* command, which stands forinvoke expression to read the environment variable **iraonixa**, which contains the string that was Base64-encoded. This suggests that the malware attempts to use the Base64-encoded string as executable payload in powershell.exe. This string of Base64-encoded code was also observed to be created upon execution of the **RegCreateKeyExW** function from the Advanced Dynamic Analysis section using OllyDBG.

To summarize, the table below depicts the likely function of each of the newly-introduced variables that exist within Obfuscated Code 1:

| **Variable** | **Function** |
| --- | --- |
| N10T | ActiveX Object that stores WScript.Shell |
| iraonixa | Environment variable that stores a Base64-decoded string |
| K6Akk | Variable used to run powershell.exe and read the environment variable ‘iraonixa’ |

*Table 6.1.2a - Table that lists the variables found to be within Obfuscated Code 1*

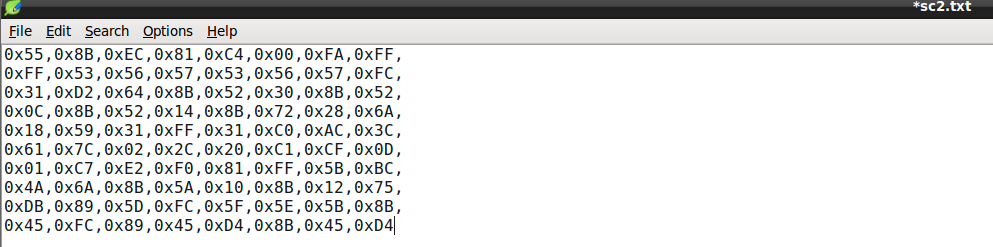
### 6.1.3 Decoding the Base64 String

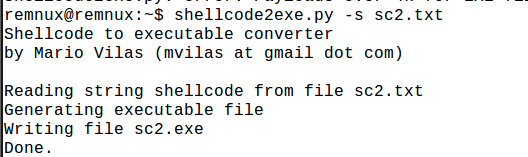
From the previous section, we found out that the malware attempts to execute a string that was Base64-encoded onto Windows powershell. Therefore, if we could decode the string ourselves, we would be able to interpret the code that the malware wants to execute in the command shell. By using a Base64-decoder from [base64decode.org](https://www.base64decode.org/), we were able to retrieve the code that the malware was attempting to execute in powershell:



*Figure 6.1.3a - Decoded Base 64 String*

From the above figure, we can see that the code first creates a function called **gdelegate,** which mainly performs some initialization operations. Thereafter, a new function **gproc** was created, and within that function something interesting was that the malware attempts to retrieve the address of a process via the API **GetProcAddress**. Since we deduced that the malware injects itself into the **regsvr32.exe** process ([see Section 5.1.3](#_lgaj40mbufy5)), we can infer that the malware performs its process injection within this block of code. Following this function, we can see that a byte array **sc32** was initialized, that seems to contain shellcode that is stored in hexadecimal format. Since this sequence of shellcode is expected to be executed in powershell, we can assume that the shellcode likely is used to import all the DLLs and APIs that the malware wishes to execute via the **regsvr32.exe** process.





*Figures 6.1.3b and 6.1.3c - Converting hexadecimal shellcode to an exe file*

With reference to Figures 6.1.3b and 6.1.3c, to confirm that the shellcode was indeed used to import DLLs and APIs, a portion of the shellcode was taken and converted from hexadecimal into an exe binary file using the shellcode2exe tool.

Thereafter, by analyzing the converted shellcode in IDA pro, we were able to see that this portion of the shellcode was indeed used to import the kernel32.dll DLL and some APIs from that DLL such as LoadLibraryA:



*Figures 6.1.3d - Proof that the shellcode is used to import DLLs and APIs*

### 6.1.4 Functionality of Obfuscated Code 1

With this, we can finally proceed to deduce the functionality of Obfuscated Code 1.

Firstly, from registry analysis ([see Section 5.1.2](#_qk6dwxrmk3g8)), we were able to identify that the Obfuscated Code was stored in a registry key created by the malware, HKU\

Software\wjpkx\himw.

Secondly, from debugging and further analysis, we were able to determine that the malware stores a Base64-encoded string in a variable, decodes it and executes it in Powershell.

Thirdly, from decoding the Base64-encoded string, we were able to infer that the function of the Base64-encoded code was to inject itself onto the **regsvr32.exe** process, and use the **sc32** byte array to execute all of the APIs it had imported from the DLLs through the process.

Therefore, when we link this together, we can deduce that the registry key HKU\

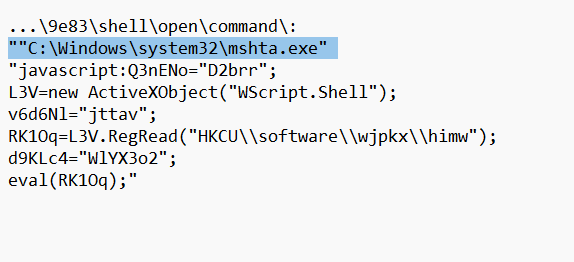
Software\wjpkx\himw stores obfuscated code that when executed via Windows powershell, will run all of the APIs that the malware imports through injecting itself onto the **regsvr32.exe** process.

# 

## 6.2 Obfuscated Code 2

With this, we can proceed to analyze Obfuscated Code 2, which was the second chunk of obfuscated code identified. Compared to Obfuscated Code 1, Obfuscated Code 2 is shorter, much less complex, and will likely require much less analysis.

### 6.2.1 Functionality of Obfuscated Code 2



*Figure 6.2.1a - Obfuscated Code 2*

Figure 6.2.1a shows Obfuscated Code 2. From the figure, we first identified that the malware uses the shell to run the **mshta.exe** binary, which as we described, was a utility that executes Microsoft HTML Application (HTA) files, and allows scripting languages such as Javascript, VBScript or JScript to be executed through passing in arguments in the shell. Thereafter, it executes a Javascript payload that creates a new ActiveX Object named WScript.Shell and stores it in a variable **L3V.** Thereafter, using this ActiveX Object variable and the RegRead operation, it reads from the HKCU\Software\wjpkx\himw registry key and stores the value in another variable **RK10q.** Lastly, using the eval() function, it executes the value stored in the **RK10q** variable.

As documented in the previous section, we know that the value stored in the HKCU\Software\wjpkx\himw registry key contains Obfuscated Code 1, which performs process injection and executes the APIs that the malware imports.

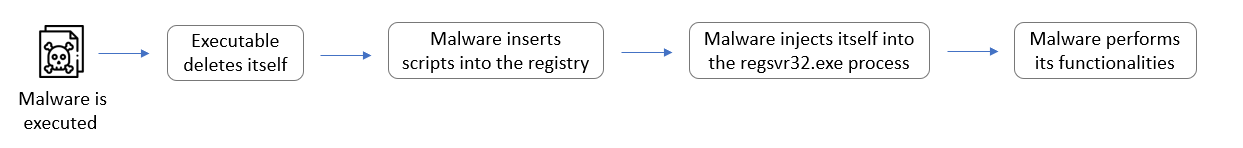
Therefore, we can deduce that the functionality of Obfuscated Code 2 is to first open the mshta.exe program so that malicious Javascript payload can be passed in to powershell, which will be used to read and execute Obfuscated Code 1 that is stored within the HKCU\Software\wjpkx\himw registry key.

# 7 General Analysis of Malware

After performing dynamic and static analysis, as well as analyzing the obfuscated code generated by the malware, we can now proceed to give a general analysis of the malware. In this section, the behavior, persistence mechanisms, functionalities as well as the malware type of Kovter will be discussed.

## 7.1 Malware Behavior

Kovter is a unique malware that is not only able to perform file and registry manipulation, but also able to establish connections between multiple different remote servers. To make a deeper analysis on the behavior of the malware, we would first need to understand how the malware works by analyzing its behavior from the moment the executable is executed to the moment the malware starts to perform its functionalities.



*Figure 7.1a - Behavior diagram of the Kovter malware upon execution*

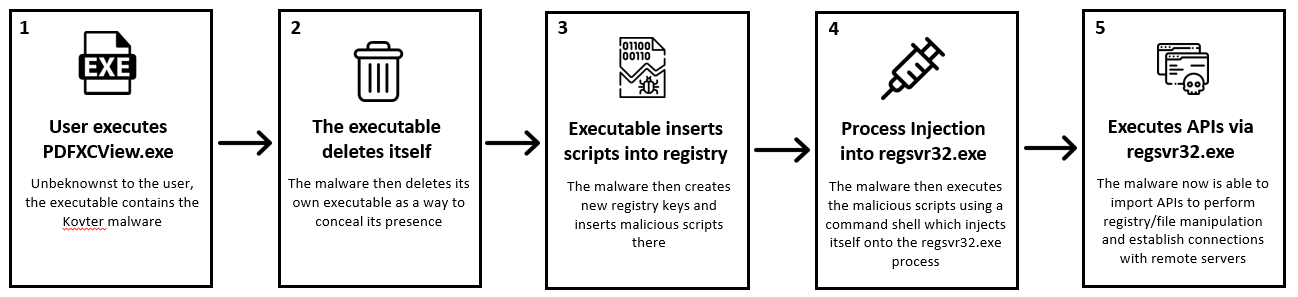
Figure 7.1a shows a surface-level depiction of the behavior of the Kovter malware from the moment the Kovter executable is executed, up to the point where it starts to perform its functionalities. Note that the figure above is only meant to serve as a mere illustration of how the Kovter malware generally behaves. As Kovter is a unique and rather complex malware sample, each individual segment would need to be broken down into a more granular level to truly uncover the behavioral characteristics, as well as the functionality of the malware. This will be done in the next section, where we uncover the persistence mechanisms employed by the Kovter malware. As a result, more of the behavioral traits as well as the functionalities of Kovter will only be discussed in a later section, where we will take into account these factors and all the information gathered about the malware to determine the malware type of Kovter.

## 7.2 Malware Persistence Mechanisms

Persistence mechanisms are techniques used by malware to ensure that they remain active on a compromised device, even after a device is rebooted or the malware is removed. The more sophisticated a malware is, the better it is at concealing its presence and remaining persistent in an infected system. In this section, we will explore the unique persistence methods employed by the Kovter malware, with the knowledge that we acquired about the malware from the previous analysis sections.

### 7.2.1 Fileless Persistence

Firstly, Kovter remains persistent in a compromised system by using a technique known as **fileless persistence**. Fileless persistence is a method used by malware to remain persistent in an infected system without writing any files or executables to the hard drive. This can be done by using built-in operating system tools, scripting languages and other non-malicious software to maintain access to a system. A malware that employs fileless persistence techniques tends to be harder to detect as such malware tends not to leave behind as many traces as traditional malware. In addition, fileless persistence makes it more difficult for traditional anti-virus and anti-malware software to detect and remove the malware, as there are no files on the hard drive to scan. As well be explained later, Kovter achieves fileless persistence by performing registry manipulation and process injection.

To document and explain how Kovter achieves fileless persistence, a more verbose diagram was illustrated that depicts how Kovter functions upon execution up to the point it starts to perform its functionality:

*Figure 7.2.1a - How Kovter functions upon execution*

**Step 1: User executes the Kovter executable**

First, an unsuspecting user will execute PDFXCView.exe, the executable containing the Kovter malware.

**Step 2: Deleting itself to conceal its presence**

Right after the Kovter executable is executed, Kovter first removes itself from the system. By doing so, it is able to increase its chances in remaining persistent in a system, by concealing its presence.

**Step 3: Inserts scripts into the registry via registry keys it creates**

At the same time, the malware sample creates a few registry keys and inserts code into these registry keys it creates. It obfuscates the code to further conceal the intent of the code, but after the obfuscated code analysis ([see Section 6.1](#_g13d1p1g6nsb)), we were able to deduce that the code stored in the registry keys were shellcode that was used to inject APIs that were imported by the malware into the regsvr32.exe process.

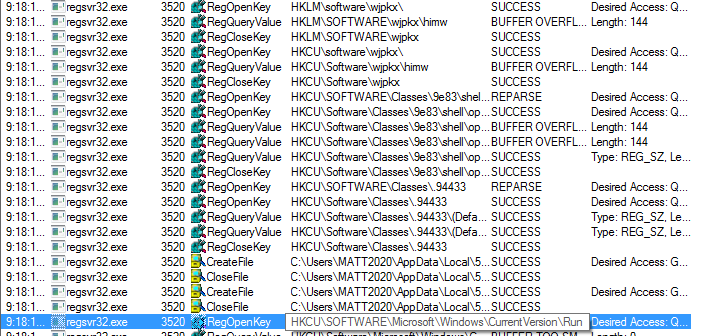
**Step 4: Injects itself into the regsvr32.exe process by reading and executing the contents stored in the registry keys it creates**

Thereafter, by using the \shell\open\command registry key and the mstha.exe program, Kovter executes Javascript payload via Windows Powershell that reads the values stored in the registry key it created and executes it. As we know, the values stored in the registry key it created is code that performs process injection, therefore, execution of this code injects the APIs imported by the Kovter malware into the regsvr32.exe process, thereby allowing the malware to perform its functionality without the executable remaining in the system as an application file.

### 

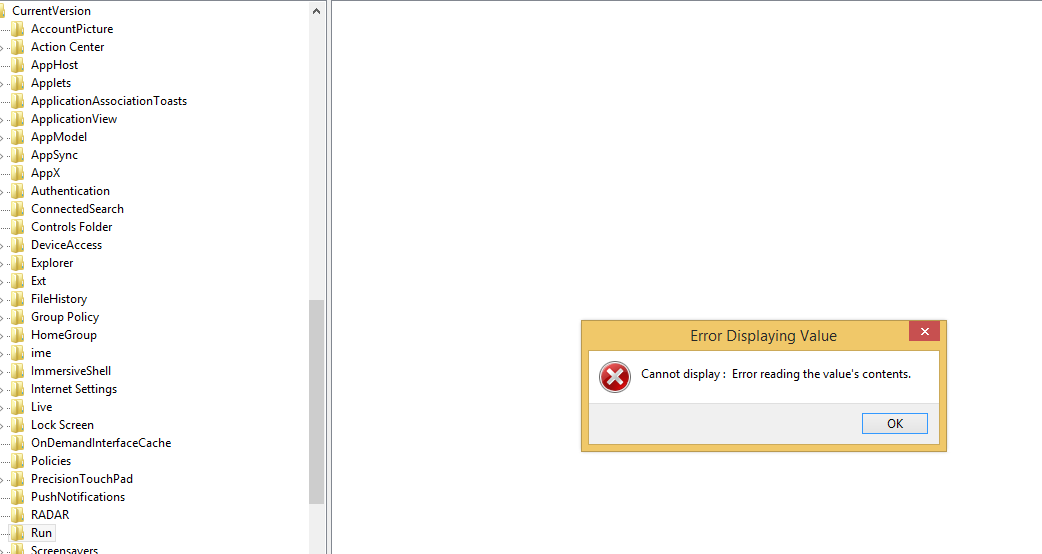
### 7.2.2 Run Registry Key

To ensure that the Kovter malware runs upon system startup, Kovter also adds a registry entry to the Run registry key under the HKCU registry hive:



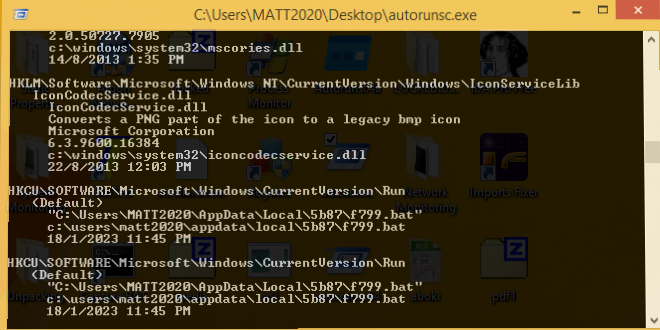
*Figure 7.2.2a - Adding an entry to the Run registry key*

However, as seen in the Basic Dynamic Analysis section ([see Section 5.1.3](#_lgaj40mbufy5)), Kovter prevents us from manually viewing the value added to the Run registry key via Regedit.



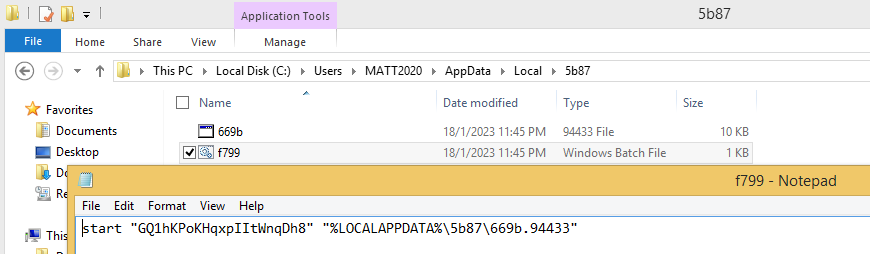
*Figure 7.2.2b - Kovter preventing the user from accessing the Run registry key*

Even though the Kovter malware prevents us from viewing the Run registry key in Registry Editor, using Autoruns, which is a utility provided by Sysinternals, we can still see which programs are automatically run upon system startup:



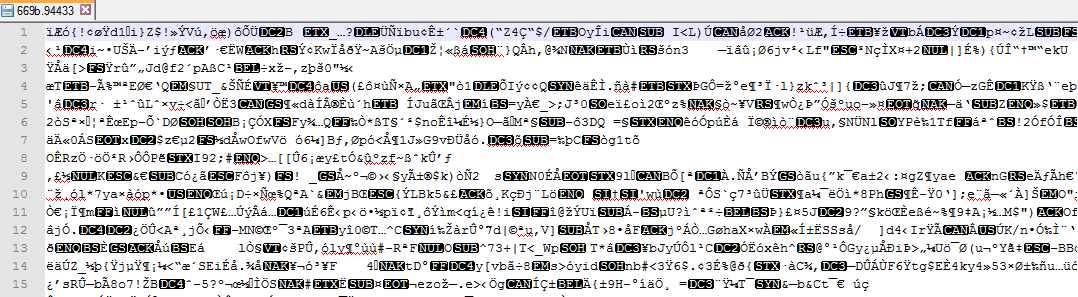
*Figure 7.2.2c - Identified that the f799.bat file was added to the HKCU Run Registry key*

With reference to Figure 7.2.2c, upon using the Autoruns tool, we can see that the value added by Kovter to the HKCU Run registry key was f799.bat.

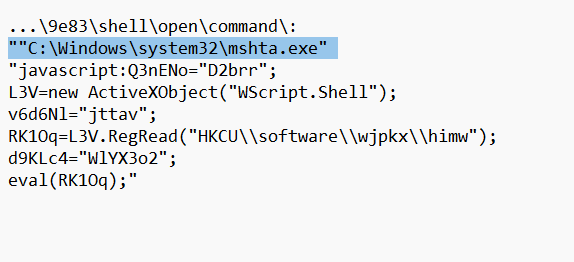


*Figure 7.2.2d - Code that is stored in f799.bat*

With reference to Figure 7.2.2d and through our earlier deductions ([see Section 5.1.3](#_lgaj40mbufy5)), we were able to determine that the f799.bat batch file was used to run the 669b.94433 file, that contained script code that had to be executed by starting the mshta.exe process.

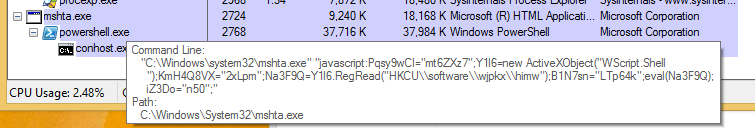
*Figure 7.2.2e - Code that is stored in 669b.94433*

Upon opening the 669b.94433 file in a text editor, we can see that the content is completely unreadable, but we can assume that this block of code is similar to the block of code as shown in the figure below that we previously analyzed, that also opens the mshta.exe process, and effectively executes the malware by injecting it into the regsvr32.exe process. This makes sense because upon system startup, when the contents of the 669b.94433 file is run, the Kovter malware is automatically executed by injecting itself into regsvr32.exe.



*Figure 7.2.2f - Obfuscated Code 1 that also uses mstha.exe to execute the malware*

Therefore, by adding a batch file in the Run registry key, that references another file that runs script code through the mstha.exe process, Kovter is able to remain persistent in a system by automatically injecting itself into the regsvr32.exe process whenever the infected system starts up, through the use of the mstha.exe process and Windows powershell, even without the existence of its own executable file



*Figure 7.2.2g - Mshta.exe process used to run powershell.exe upon system startup*

This can also be confirmed by restarting the system and viewing the results in Process Explorer. As seen from Figure 7.2.2g, upon restarting the system and opening Process Explorer, we are unable to see any regsvr32.exe processes but we are able to see the mstha.exe process used to run powershell.exe. Hovering over the mshta.exe process, we also see the javascript payload that it injects into powershell, which as we went over, is Obfuscated Code 2, that reads the value stored in the HKCU\Software\wjpkx\himw registry key.



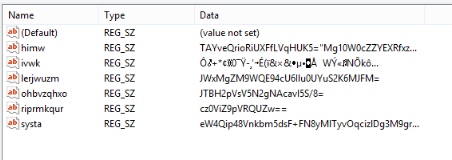
*Figure 7.2.2h - Mshta.exe process used to run powershell.exe upon system startup*

Thereafter, with reference to Figure 7.2.2h, we can see that shortly after the mshta.exe process invokes Windows powershell, the regsvr32.exe process starts.

### 7.2.3 Obfuscation

Lastly, the malware aims to remain persistent in a system through obfuscation. Obfuscation is the practice of making something difficult to understand or confusing, often to conceal the true meaning or intent of the original code or data. In the context of malware analysis, obfuscation is used by malware authors to make it harder for malware analysts to reverse-engineer or analyze a piece of malware sample. This allows the functionality of the malware to be better concealed, such that the malware is able to remain undetected to an infected user, who may not be aware that such code in fact contains malicious intent.

The Kovter malware uses obfuscation in almost every possible instance where it modifies or adds a value. Some examples include the registry keys it adds into different registry hives, the blocks of code it adds into the registry keys, the blocks of code it executes via Windows Powershell, the contents it stores in files such as 669b.94433 and many more. Some other techniques it also employs include XOR encryption and Base64 encoding.



*Figure 7.2.3a - Obfuscation attempts by the Kovter malware on the registry keys it creates*

## 

*Figure 7.2.3b - Obfuscation attempts on contents of files added by Kovter*

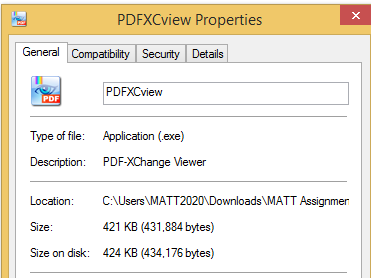
### 7.2.4 Summary of Kovter’s Persistence Mechanisms

In summary, by scattering persistence mechanisms into several layers (different registry keys, different blocks of code that each perform a different function), adding excessive obfuscation at every stage and going so far as to deleting its own executable and preventing an infected user from using Regedit to view the Run key, the Kovter malware achieves persistence by combining techniques such as fileless persistence, adding values to Run keys and adding obfuscation in unconventional and uniquely crafted methods.

## 7.3 Malware Type

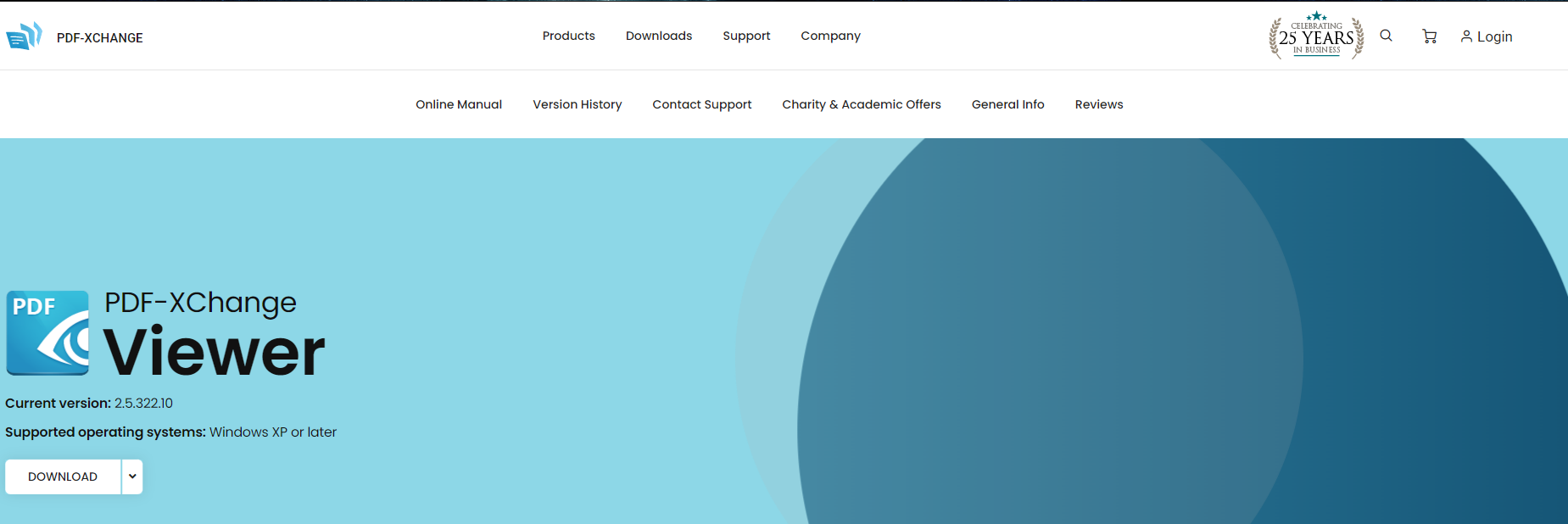
With this, we can start to determine the type of malware that Kovter possibly belongs to. This will be done by taking into consideration the functionalities and behavior of the Kovter malware, so that a more accurate deduction can be made.

Firstly, we can evidently say that Kovter is a **Trojan**, which is a type of malware that installs itself on a victim’s computer as a legitimate program. This is because the executable running the Kovter malware has an application name of **PDFXCView.exe**, which also stands for PDF-XChange Viewer.



*Figure 7.2a - Application name of that contained the Kovter malware*

Just by conducting simple research, we would be able to determine that PDF-XChange Viewer is indeed a legitimate application run by a legitimate company, PDF-XChange Co. Ltd.

*****Figure 7.2b - The legitimate PDF-XChange Viewer application*

With reference to Figure 7.2b, by performing comparison between both applications, we can observe striking similarities between the application icons of both executables, both of which contain an eye and the word “PDF”. This clearly suggests that the Kovter executable was attempting to disguise itself as the legitimate PDF-XChange Viewer Application, to trick its users into thinking that it functions as a PDF reader, when in fact, it contains much more malicious intent.

Secondly, there is also a likelihood that Kovter is a **Remote Access Trojan**. A remote access trojan, or RAT, is a type of malware that allows an attacker to gain unauthorized access and control over a victim’s computer.

One reason why Kovter may be a Remote Access Trojan is because the malware has the ability to perform file and registry manipulation, as well as the ability to access and utilize command shell tools in a machine, which are common traits that attackers leverage in RATs. In fact, Kovter’s control over the file, registry and shell operations in an infected machine is so vast that:

1. An infected user is not even able to view the contents of the Run registry key through Registry Editor
2. The malware is able to delete its own executable file but still remain persistent in the system through executing its functionalities (that are stored in registry keys) from the command shell
3. The malware is able to inject itself independently onto a legitimate process without the user having to start any service in the command shell

Lastly, there is also the likelihood that Kovter is a **botnet**. A botnet is a group of infected computers, or “bots” that an attacker can collectively control from a remote location. These bots can be instructed collectively to perform certain tasks such as conducting a Distributed Denial of Service Attack (DDoS attack), stealing sensitive information and many more.

Kovter may likely be a botnet malware because in the network analysis section ([see Section 5.1.4](#_uvjeqvqj748q)), running the malware in a sandbox-environment resulted in the malware showing that it establishes multiple connections with different remote servers, each remote server being legitimate organizations, that ranged from organizations providing media and software services, to even multinational conglomerates. In addition, we were able to see references to multiple different domain addresses made by the malware in the Advanced Dynamic Analysis section using OllyDBG. Therefore, it is very unlikely that this malware is a spyware, and it is much more reasonable to believe that this is simply a botnet malware that harbors malicious intent towards these organizations.

By sending a collective group of infected “bots” to these legitimate organizations, the malware author of Kovter could either be attempting to:

1. Launch a **Distributed Denial of Service Attack (DDoS)** on these organizations
2. **Spam** the email addresses of these organizations or
3. Launch a **click fraud attack** on the webpages of these organizations, to limit the effectiveness of any organizations launching ad campaigns, and to limit the effectiveness of analyzing customer engagement in the organization’s web page

# 

# 8 Malware Defenses

Malware defenses are important because they protect computer systems, networks, and data from malicious software that can cause harm and compromise sensitive information. Malware can infect computers and spread through networks, stealing sensitive data, disrupting operations, and compromising system security. Defenses against malware play a crucial role in maintaining the security and integrity of computer systems and networks.

Just from this report alone, we were able to see how destructive malware can be if left rampant on an infected machine. Therefore, this section will detail the several mechanisms users can employ to protect their systems against the Kovter malware.

As mentioned in the previous section, since there is a high likelihood that Kovter is a Trojan botnet, the remediation measures discussed in this section will be largely catered towards defending a system against a botnet and/or Trojan program. In addition, defense mechanisms against the exploitations employed by the Kovter malware will be detailed as well.

### 

### 8.1 Antivirus Solutions

Of course, to start off with the most obvious remediation technique, users can protect their systems from the threat of malware through installing Antivirus software. Antivirus software is designed to detect, prevent, and remove malware from a computer or network. By regularly scanning the system for malware, antivirus software can help prevent infections and minimize the impact of attacks. Additionally, antivirus software often includes real-time protection features that monitor for suspicious activity, such as incoming network connections or file modifications, and can alert the user or take action to block the activity. In today's connected world, malware threats are constantly evolving and becoming more sophisticated, to the point where not having any form of Antivirus software on your computer would simply be too big of a risk.

With so many reputable and convenient antivirus tools out there such as Norton, TotalAV, McAfee and more, users can significantly reduce the risk of getting infected by malware in their systems by simply choosing one of the many options available and installing it into their systems.

### 8.2 Securing Powershell

Securing PowerShell can also help defend against malware such as Kovter by limiting the power and capability of malicious actors who are trying to exploit vulnerabilities through executing shellcode. PowerShell is a powerful tool that provides administrators with the ability to automate various tasks, but it can also be used as an attack vector for malware attacks, as demonstrated by Kovter throughout this report. By properly securing PowerShell, organizations can reduce the risk of successful attacks and protect against malware like Kovter.

One way to secure PowerShell is by restricting the execution of scripts. By default, PowerShell allows scripts to be executed, but by disabling this feature, you can prevent malicious actors from running scripts that may contain malware like Kovter. Additionally, you can configure PowerShell to only run signed scripts, ensuring that only scripts from trusted sources are executed. This prevents the execution of binaries such as mstha.exe, which as seen over the course of this report, was the primary mechanism used by Kovter to achieve fileless persistence. Without mstha.exe, Kovter would not have been able to execute any form of its Javascript payload onto a command shell, and would have been unable to perform its process injection operations.

### 8.3 Software Updates

With regard to protecting systems against malware, software updates are also an effective and important factor for users to consider. Software updates often include security patches that fix vulnerabilities within a system that could otherwise be exploited by attackers. By keeping software up to date, users will be able to be equipped with the latest and most up-to-date security mechanisms provided by software against malware threats.

### 8.4 Regularly Monitoring Network Activity

Lastly, it is also worthwhile for users to know how to regularly monitor network activity. As mentioned, since there is a high likelihood that Kovter is a botnet, by regularly monitoring network activity, users would be able to determine whether any suspicious connections are being made, or whether any excessive amount of HTTP requests are being sent out by their computer systems. These are usually clear indications that show whether a computer system is infected by a botnet or not, and can serve as an additional layer of security in helping users detect the presence of a malware in their systems.

In summary, some techniques that were discussed in this section include antivirus solutions, securing powershell, performing software updates and regularly monitoring network activity. While this may seem a lot, there are still many other defense techniques out there such as proper firewall configurations, anti-spyware software and many more. It is worth noting that no single solution provides a 100% guarantee of protecting a system against malware, and users should adopt a combination of several defense techniques to provide the most comprehensive security strategy to protect their system from malware infections.

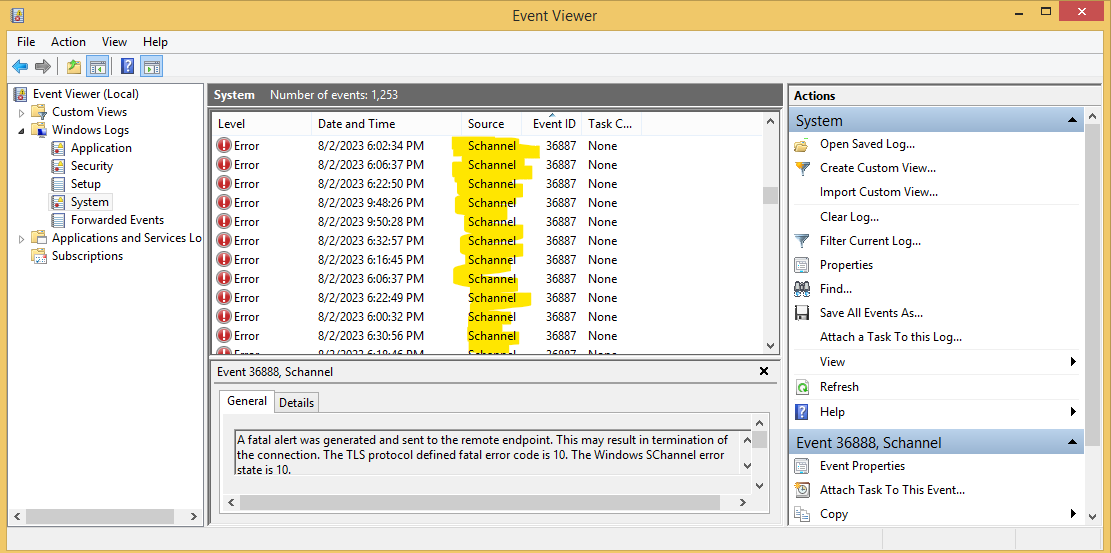
# 9 Malware Removal

In the event a user’s system is infected with malware, it is also important for users to be able to know how to remove such malware from their systems. As seen from this report, malware programs usually employ a variety of persistence mechanisms that make it much harder for users to remove the malware from their systems. Therefore, this section will document how users can detect whether their system is infected by the Kovter malware, and the steps that need to be taken in order to remove Kovter from their system

### 9.1 Indications of presence of Kovter

**Schannel Errors**

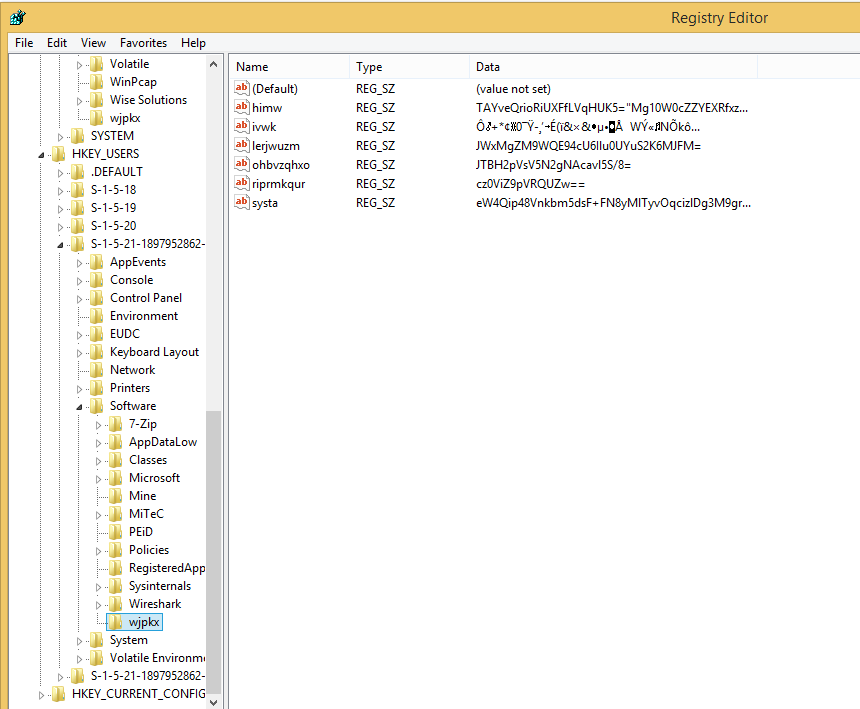
One indication of the presence of the Kovter malware is the presence of Schannel errors. Schannel errors in the Event Logs are a common indicator that Kovter malware is present in a computer. Schannel stands for Secure Channel, and it is a security protocol used by Microsoft Windows to establish encrypted connections. If a computer is infected with Kovter, the malware may cause Schannel errors to appear in the Event Logs, indicating a problem with the secure connection. These errors can be viewed by accessing the Windows Event Viewer and checking the System log. The Event Viewer can be opened by pressing Windows Key + R and typing "eventvwr.msc" in the Run dialog box. Once you have opened the Event Viewer, navigate to the System log and look for Schannel errors. If you find errors related to Schannel, it could be an indicator of Kovter infection. It's important to note that Schannel errors can also be caused by other factors, so additional investigation may be required to confirm the presence of Kovter malware.



*Figure 9.1.1 - Event Viewer showing presence of Schannel error which may Allude to a Infected system*

**Registry Analysis**

Another technique that users can employ to determine the presence of the Kovter malware in their systems is also to check their registry. Using a tool like Registry Editor, we can observe whether our system is infected with the Kovter malware by seeing whether or not there are any dubious registry keys or values that should not exist. One example of this would be the HKLM/Software/wjpkx registry key as shown in the dynamic analysis section of this report.

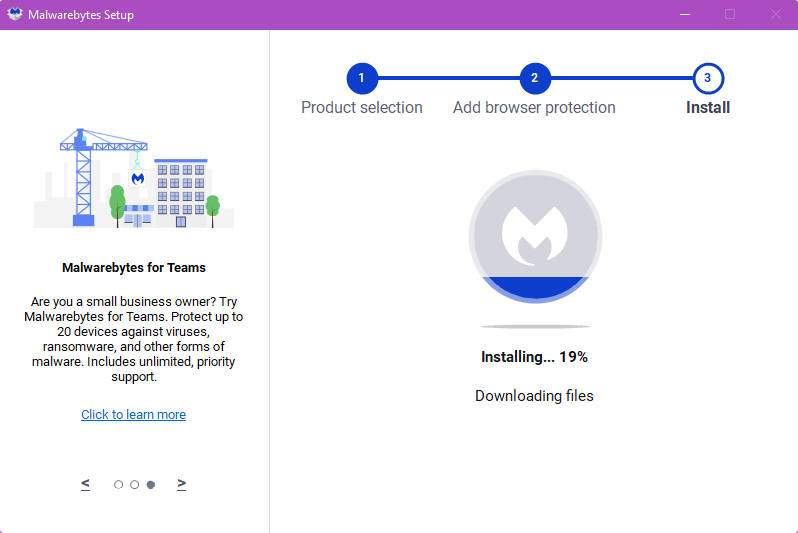


*Figure 9.1.2 - Analysing Regedit to determine presence of Kovter malware*

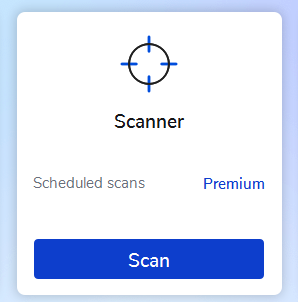
### 9.2 Removal of Kovter

The removal of the Kovter malware from an infected system is an important step to restore normal functionality and prevent further harm to the computer. In this section, we will detail the steps taken to remove the Kovter malware from the infected system using the Malwarebytes software. The use of Malwarebytes is based on its effectiveness in detecting and removing various types of malware, including the Kovter malware. The following steps were taken to effectively remove the Kovter malware and secure the system

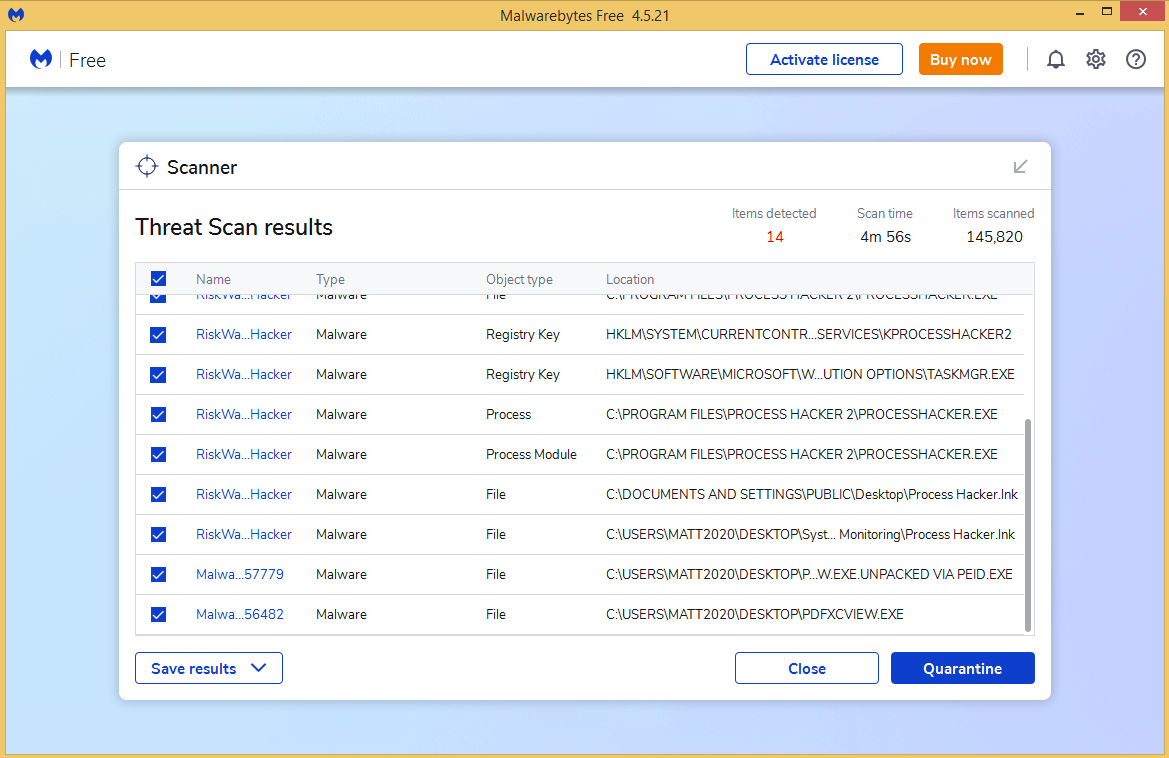
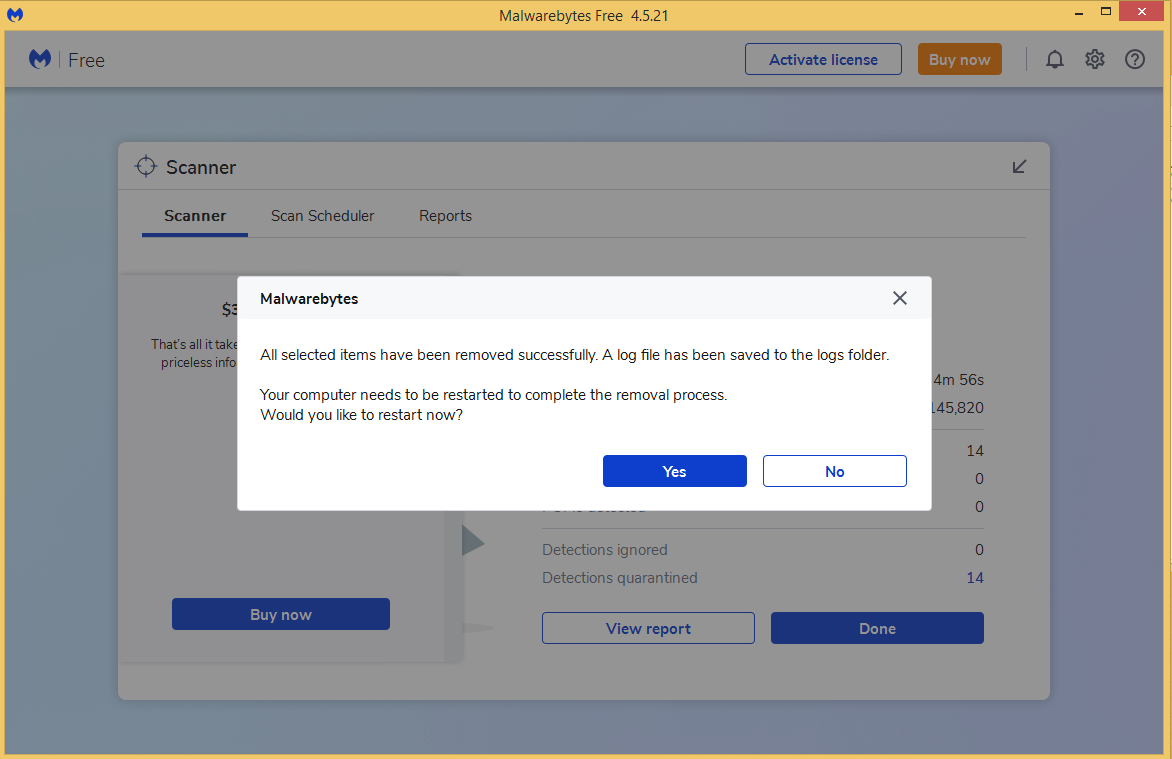
1. Download and install Malwarebytes on the infected computer.



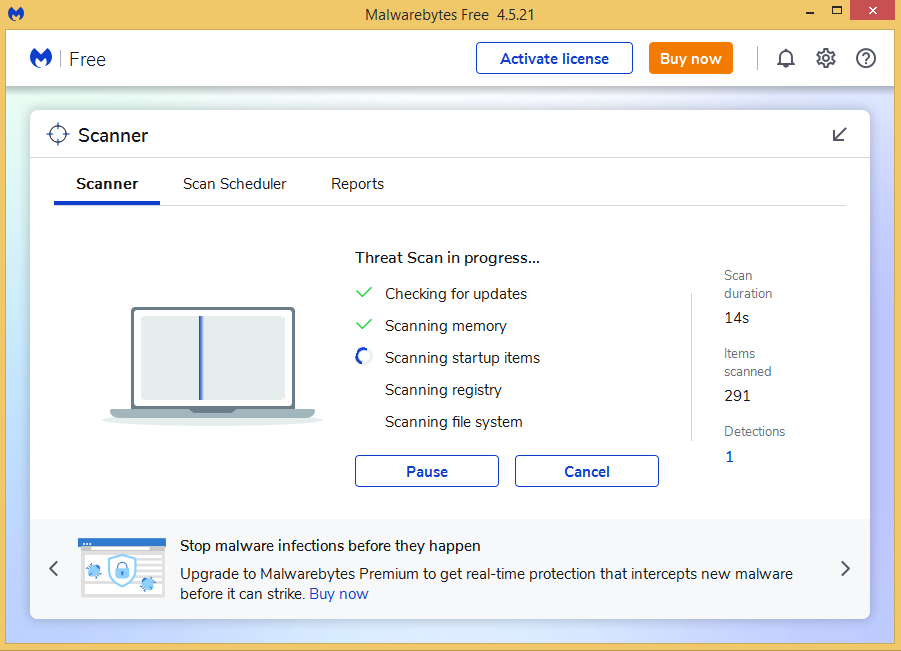
*Figure 9.2.1 - Installation of Malwarebytes on infected machine*

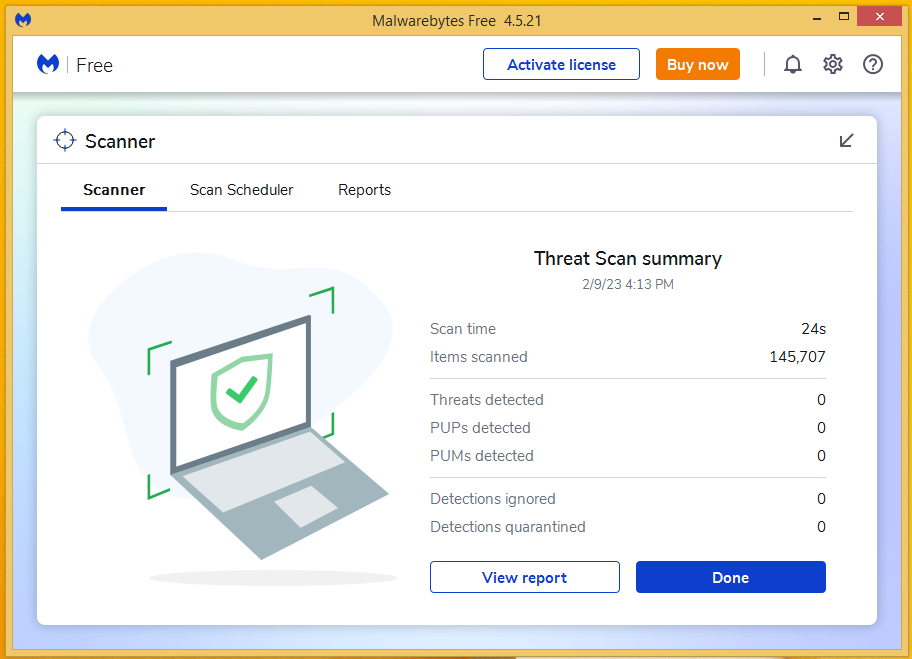
1. Run a full system scan using Malwarebytes

*Figure 9.2.2 - Scanning option via Malwarebytes*

1. Review the scan results, notice that MalwareBytes have successfully identified Kovter, and remove the detected Kovter malware by clicking the "Quarantine Selected" button. *Figure 9.2.3 - Quarantine Page*
2. Restart the computer to complete the removal process.

*Figure 9.2.4 - Restart Prompt*

1. Verify that the Kovter malware has been effectively removed by running another full system scan with Malwarebytes. From the bottom 2 image, it is evident that Kovter was successfully removed.

*Figure 9.2.5 - Second scan for verification*

*Figure 9.2.4 - Verification successful*

# 10 Summary

In summary, through untangling Kovter’s unique and complex persistence mechanisms, as well as its endless layers of obfuscation attempts and hidden functionalities, this report managed to comprehensively document the various traits and nature of the Kovter malware. By using both static and dynamic analysis techniques, as well as other analysis techniques throughout this report, we were able to extensively reverse engineer the functionalities of the Kovter malware, and produce a thorough examination that details the entire process of Kovter, from the moment the executable is ran on a system, to the moment the executable executes its functionalities and payloads.

# 11 Video Submission Links

| **Introduction Video** | <https://youtu.be/5zgb6wvpAlE> |
| --- | --- |
| **Static Analysis Video** | <https://youtu.be/WZfScRsKDcE> |
| **Dynamic Analysis Video** | <https://youtu.be/P1GNrics-hw> |
| **Obfuscated Code Analysis Video** | <https://youtu.be/hM_X3A0GCnM> |
| **Conclusion Video** | <https://youtu.be/gDXHToMEaXY> |

Note: Since we have an additional section where we analyze the obfuscated code created by the malware, we will produce an additional video talking about how obfuscated code analysis was performed.

Note: The table below shows the list of topics we will cover in each video:

| **Introduction Video** | * Malware sample download * Setup environment * Malware analysis process |
| --- | --- |
| **Static Analysis Video** | * Basic static analysis * Advanced static analysis |
| **Dynamic Analysis Video** | * Basic dynamic analysis * Advanced dynamic analysis |
| **Obfuscated Code Analysis Video** | * Obfuscated code 1 * Obfuscated code 2 * Persistence mechanisms |
| **Conclusion Video** | * Malware type * Conclusion * Reflection |