A Cloud-Based Intelligent and Energy Efficient Malware Detection Framework

A Framework for Cloud-Based, Energy Efficient, and Reliable Malware Detection in Real-Time Based on Training SVM, Decision Tree, and Boosting using Specified Heuristics Anomalies of Portable Executable Files

Qublai Khan Ali Mirza

Submitted for the Degree

Of Doctor of Philosophy

School of Electrical Engineering and Computer Science

University of Bradford

2017

To my little champ Mujtaba and my lovely wife Saba, thanks for lighting up my world, this is for you.

To my parents and their unconditional love and prayers

DECLARATION OF AUTHORSHIP

I, Qublai Khan Ali Mirza, confirm that this thesis contains my own work and has never been submitted for any other academic award. Any information derived from other material has been properly referenced.

Signed: Dated:

ABSTRACT

Qublai Khan Ali Mirza “A CLOUD-BASED INTELLIGENT AND ENERGY EFFICIENT MALWARE DETECTION FRAMEWORK”

Keywords: Malware detection, portable executables, file heuristics, SVM, decision tree, boosting, cloud computing, energy efficiency, real-time detection, automated static analysis

The continuity in the financial and other related losses due to cyber-attacks prove the substantial growth of malware and their lethal proliferation techniques. Every successful malware attack highlights the weaknesses in the defence mechanisms responsible for securing the targeted computer or a network. The recent cyber-attacks reveal the presence of sophistication and intelligence in malware behaviour having the ability to conceal their code and operate within the system autonomously. The conventional detection mechanisms not only possess the scarcity in malware detection capabilities, they consume a large amount of resources while scanning for malicious entities in the system. Many recent reports have highlighted this issue along with the challenges faced by the alternate solutions and studies conducted in the same area. There is an unprecedented need of a resilient and autonomous solution that takes proactive approach against modern malware with stealth behaviour.

This thesis proposes a multi-aspect solution comprising of an intelligent malware detection framework and an energy efficient hosting model. The malware detection framework is a combination of conventional and novel malware detection techniques. The proposed framework incorporates comprehensive feature heuristics of files generated by a bespoke static feature extraction tool. These comprehensive heuristics are used to train the machine

learning algorithms; Support Vector Machine, Decision Tree, and Boosting to differentiate between clean and malicious files. Both these techniques; feature heuristics and machine learning are combined to form a two-factor detection mechanism. This thesis also presents a cloud-based energy efficient and scalable hosting model, which combines multiple infrastructure components of Amazon Web Services to host the malware detection framework. This hosting model presents a client-server architecture, where client is a lightweight service running on the host machine and server is based on the cloud.

The proposed framework and the hosting model were evaluated individually and combined by specifically designed experiments using separate repositories of clean and malicious files. The experiments were designed to evaluate the malware detection capabilities and energy efficiency while operating within a system. The proposed malware detection framework and the hosting model showed significant improvement in malware detection while consuming quite low CPU resources during the operation.

ACKNOWLEDGEMENTS

I would like to thank Almighty Allah for blessing me with immense patience, strength, and knowledge to enable me finish this study. I would like to thank my parents for their unconditional love and prayers throughout this journey that allowed me to successfully reach this level. I am grateful to my brothers and most importantly my wife and son for making this journey much easier for me and inspire me to set my goals high.

I am extremely grateful to my supervisor Prof. Irfan Awan for his professional support and belief that gave me confidence to continue my research in the right direction. A big thank you to Dr. Jules Pagna Disso of Nettitude Ltd. for his expert technical insight in network security and critical analysis of my work during the entire period of this study and giving me access to data that made my experiments possible. I am very grateful to Dr. Anitta Namanya my colleague and friend for guiding me, sharing ideas with me, and most importantly providing me with technical support in the initial phase of my research.

I am indebted to all my friends and colleagues in the cyber security research group for all the fun and laughter that made this journey one of the most memorable time of my life. I would specially like to thank Bashir Muhammad, Adeeb Alhomoud, Hamad Al-Mohannadi, and all my friends from the research group for all the memories.

PUBLICATIONS:

Journal Paper:

1. **Q. K. A. Mirza**, I. Awan, and M.Younas, “CloudIntell: An intelligent malware detection system,” Future Generation Computer Systems, Jul. 2017, http://dx.doi.org/10.1016/j.future.2017.07.016

Conference Papers:

2. **Q. K. A. Mirza**, G. Mohi-Ud-Din, and I. Awan, “A Cloud-Based Energy Efficient System for Enhancing the Detection and Prevention of Modern Malware,” in 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA), 2016, pp. 754–761.

3. Namanya, A.P; **Mirza, Q.K.A**; Al-Mohannadi, H; Pagna-Disso, J; Awan,I (2016): Detection of Malicious Portable Executables using Evidence Combinational Theory with Fuzzy Hashing; *Future Internet of Things and Cloud (FiCloud2016), 2016 IEEE 4th International Conference , 22-24 August 2016*, Vienna, Austria.

4. Al-Mohannadi, H; **Mirza, Q.K.A**; Namanya, A.P; Pagna-Disso, J; Awan,I (2016): Cyber-Attack Modeling Analysis Techniques: An Overview; *Future Internet of Things and Cloud Workshops (W-FiCloud2016), 2016 IEEE 4th International Conference , 22-24 August 2016*, Vienna, Austria.

5. Namanya, A.P; **Mirza, Q.K.A**; Al-Mohannadi, H; Cullen, A; Awan,I (2016): Towards Building a Unified Threat Analysis and Management Framework; *(UKPEW) & Cyber Security Workshop (CyberSecW)*

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1. INTRODUCTION

1.1 Motivation

The current era of technology, which is also known as “the age of data” has changed the entire perception about technology. The amount of data generated everyday by devices with limited resources is unprecedented and the volume of world’s data doubles in every two years [1]. This means that the level of security required to protect the data generation and management entities is more than it ever was. The recent attacks by a ransomware known as “*WannaCry*”, which shook the infrastructure of many big organizations before it was stopped [2], [3] raises the question on the security mechanisms that are used to protect the computing infrastructure and sensitive data.

The samples of recent lethal malware; *WannaCry, Petya* [4]*,* [5] or *Mirai* [6] caught in the wild, are not only capable of damaging giant organization or causing financial damage to banks, they have the capability of bringing down the entire infrastructure of World Wide Web that could possibly trigger a catastrophic event [7] . The most disturbing aspect of this scenario is, that these malware target the existing vulnerabilities in individual computers without even triggering an alert in the security software installed [8]. Not only the individual computers, computers part of an enterprise network, or smart devices are attacked by such malware, the infected devices are frequently used to attack bigger targets [9], such as; internet service providers, government organizations and infrastructures, and email servers.

A little less than half a million malware are released every day, majority of them are variants of previously identified malware but still have the capabilities to execute a lethal attack [10]. This cyberwar and successful attacks reveal the multidimensional risks that are faced by every consumer of modern technology, causing a daunting damage of $1.7 billion only in the UK [11]. This number of financial damages caused by cyberattacks are expected to rise above $5.8 trillion by 2020 [12]. The damage a malware author can cause without even moving from their chair is not only staggering, it is also becoming an attractive form of business.

If there is a successful malware attack on an enterprise network, despite their security infrastructure, it takes around six months on average to detect an infection, eradicating that infection can take another month [13], [14]. The amount of damage caused by a malware infection is directly proportional to the amount of time taken to identify and eliminate that infection [15]. One of the most relevant example is of Zeus malware, which was initially identified in 2007 but couldn’t be stopped [16]. According to an estimation by some security companies, Zeus infected around 3.6 million PCs only in U.S. and millions more around the world [17].

This scenario raises a serious question on the presence of antiviruses and other security software along with the amount of resources they require to operate in an individual system or in a network. The current ecosystem of technology with an enormous amount of data generation capabilities not only requires a higher level of security mechanisms, it also require that mechanism to be extremely energy efficient giving it the ability to protect growing number

of heterogeneous devices. This study focusses on the limitations of current techniques and presents a framework, which consumes less resources and provides a higher level of malware detection.

1.2 Aims and Objectives

The aim of this study is to identify the current requirements of security against malware attack by investigating the anatomy of modern and sophisticated malware, which helps to evaluate the performance of current commercial antiviruses and identify their limitations against modern malware. This paves the way to design, develop, and evaluate a comprehensive and energy efficient malware detection framework targeting PE (Portable Executable) files, which amalgamates state-of-the-art malware detection techniques with the conventional techniques to enhance the detection of modern malware with obfuscation abilities. Following objectives had to be fulfilled to achieve this aim:

a) Understand the occurrence of important anomalies in a malware by statically analyzing a large set of malicious PE files.
b) Examine the malware analysis and detection techniques currently used commercially
c) Analyze the machine learning techniques introduced in malware detection by different studies
d) Design and implement an analysis module that can retrieve a comprehensive set of relevant feature anomalies from PE files with customized and decisive heuristics

e) Design, implement, and evaluate a module that incorporates the analysis module with conventional malware detection techniques to make accurate and reliable malware detection

f) Identify efficient and appropriate machine learning algorithms that can be trained to recognize anomalies and accurately detect a malicious file

g) Design a classification module that can learn from malware anomalies and differentiate between clean and malicious files with the help of machine learning algorithms

h) Develop a framework that amalgamate analysis module and classification module to work as a coherent unit

i) Evaluate the accuracy of the entire framework by testing it against a large set of clean and malicious files

j) Evaluate the commonly used commercial antiviruses for their resource consumption

k) Design and implement a hosting model for the malware detection framework that is energy efficient and does not rely on host systems' CPU resources

l) Deploy the framework on the hosting model and evaluate the energy efficiency and performance of the model along with the framework operations.

1.3 Proposed Solution

The results of current malware detection techniques and software are not quite effective [18] in terms of providing security to their consumers. There is a

diverse pool of techniques including the conventional malware detection techniques that can be used in a combination to enhance the malware detection rate on a commercial level for general users [19]. Unfortunately, majority of the novel techniques presented in the recent times are only limited to a certain aspect of detection or a specific type of files and do not provide an approach that targets the multidimensional problem. The problem faced by general and enterprise users is not just lack of ability to identify modern and previously unknown malware, it also involves the high resource consumption by the conventional detection software.

The signature based malware detection can detect known malware and when combined with malware heuristics it can possibly detect new variants of previously known malware. However, the level of success of this combination is highly dependent on the patterns and rules that are used to formulate the heuristics of feature anomalies. Implementation of machine learning techniques has also proved to be quite successful in many studies, which is also applied on anomaly heuristics, its success is also dependent on patterns and heuristics used to apply algorithms.

This research proposes the implementation of a combination of machine learning algorithms on a set of heuristics extracted from a large set of clean and malicious files. The proposed framework is based on a two-layer decision making process, which includes the first layer of decision making with the help of static heuristics analysis and the second layer of machine learning algorithms.

- An intelligent malware detection framework comprising of two-layered detection process was developed. The first layer is comprised of static heuristics analysis, which analyzes a file and decides about its legitimacy based on the anomalies detected in the feature heuristics. The decision made by this analysis is endorsed by external sources using the conventional detection techniques. In the second layer, machine learning algorithms; SVM (Support Vector Machine), Decision Tree, and Boosting on Decision Tree, were applied to make the detection process precise and highly reliable. The feature heuristics and anomalies extracted from sets of both clean and malicious files are used to train the machine learning algorithms, which makes the final decision about a file highly accurate. In the design, implementation, and evaluation of this part of the research, we trained and tested the machine learning algorithms against a large set of clean and malicious files.
- As mentioned above, this research targets two major problems of conventional security mechanisms; detection rate and resource consumption. In the second part of this study; a cloud-based hosting model that strategically combines different components of AWS (Amazon Web Services) was designed, implemented, and evaluated as a customized hosting model for the malware detection framework. The main idea behind this hosting model is to make the framework extremely energy efficient and at the same time have the capability of scaling the framework for continuous learning. The hosting model

allows the framework to work with a dual-aspect. The first aspect of the framework trains the algorithms with the feature heuristics, whereas, the second aspect allows the framework to work in real-time scenarios. The operational requirements of both aspects of the framework were evaluated and the resource consumption was compared with the resource consumption of commercial antiviruses' running in their scan mode.

1.4 Contributions

The key contribution of this research is the design and implementation of an intelligent and energy efficient framework for detection of Windows based modern malware. To achieve this, we had to divide the work in the following two directions:

1. ***An intelligent malware detection framework***, which initially examines the common and unique anomalies found in malware by statically analyzing a large set of malware. This helps to identify the use of such anomalies in identifying modern malware and the use of machine learning to make autonomous decision. This leads to the proposal of a framework, which incorporates conventional malware detection techniques with customized and comprehensive feature heuristics to train multiple machine learning algorithms. This study provides a detailed discussion on pivotal heuristics that differentiate a clean file from malicious file. This discussion is based on the analysis performed on large set of malicious files containing nearly one million files from different families of malware. The study presented in Chapter 3, implies

that how different important malware features can be combined to form patterns that will help machine learning algorithms to train for real-time detection. The role of conventional detection techniques in detecting known malware is also discussed leading to the integration of conventional detection techniques with the analysis module. Moreover, the benefits of machine learning algorithms and how they can be significant in lowering the false-positive rate and enhancing the accuracy if a good combination of heuristics is used to train them, are also discussed.

2. ***A cloud-based energy efficient hosting model,*** initially evaluates the conventional antiviruses to identify their CPU resource consumption while operating in scan mode. This helps to identify one of the main weaknesses of commercial antiviruses, which is then targeted to propose a hosting model. The hosting model for the framework discussed in Chapter 3, which has a client server architecture, strategically combines different components of AWS to design a bespoke hosting model for the intelligent malware detection framework. Each component of the hosting model is specifically designed to host each module of malware detection framework with energy efficiency, quick response, and scalability as primary goals. The study presented in Chapter 4, initially discusses the CPU resource consumption problem of commercial antiviruses and implications of their operations on the host machine. The specific requirements of individual modules in the malware detection framework are then focused, leading to the proposal

of a high-level architecture of the cloud-based hosting model. Subsequently, the selected components of AWS are discussed, with respect to the operational requirements of the individual modules. Moreover, the implementation of the hosting model, deployment of the framework, and finally the evaluation of both; framework and hosting model is presented. The client and server modules of the hosting model are separately monitored to evaluate their performance and compare it with the commercial antiviruses.

1.5 Research Scope

This work solely targets the features and heuristics of PE (Portable Executables), commonly known as .exe files. The discussions and contributions revolve around the analysis performed on PE files, use of proposed approach on other file types or in other environments is out of scope.

The proposed framework is specifically based on the features and heuristics generated through static analysis of PE files in conjunction with signature-based detection and machine learning algorithms. Dynamic analysis of files is out of scope.

Different datasets of clean and malicious files with known malware were used in this research, as discussed in Chapter 3. Different datasets might produce slight dissimilar results but they should produce similar level of accuracy and energy efficiency.

1.6 Thesis Structure

The remaining parts of the thesis are structured as follows:

- Chapter 2: Literature Review

  This chapter presents background of the research followed by a thorough discussion on the evolution of malware and the techniques used by modern malware to avoid getting detected. The discussion on different analysis techniques that can be used to analyse files along with their implications is then presented. Subsequently, relevant recent studies with their benefits and drawbacks are discussed, which lays the foundation for the presented research.

- Chapter 3: An Intelligent Malware Detection Framework

  This chapter presents the design, modelling, and implementation details of the intelligent malware detection framework. It starts by discussing the background of the proposed framework and why this specific approach of detecting malware was taken. The study finally presents the evaluation of the entire framework followed by the discussion on outcomes.

- Chapter 4: An Energy Efficient Hosting Model for the Malware Detection Framework

  Chapter 4 presents a cloud-based energy efficient hosting model for the malware detection framework proposed and discussed in Chapter 3. The chapter discusses each module and its hosting requirements

separately and how they are managed by the hosting model. It then evaluates both; framework and the hosting model, while running in real-time

- Chapter 5: Conclusion and Future Work

  The conclusion presents the discussion on identified problems solved by the proposed framework and the hosting model by highlighting the benefits of the proposed solutions. It then discusses the limitations of the solution and how they can be eliminated. Finally, it presents the future enhancements of the entire proposed framework and how it can be used for a broader domain.

CHAPTER 2. Literature Review

2.1 Introduction

The ever-evolving landscape of cyber-attacks requires to be tackled by an ever-evolving ecosystem of security tools, techniques, and mechanisms. Unfortunately, unlike the advancements seen in the malware proliferation in the recent past, the security mechanisms are still based on the conventional detection techniques used since many years [20]. Recently identified malware have used several different types of techniques for infection and propagation and their analysis show the innovative techniques they have used to bypass the security mechanisms of networks and individual computers [21]. However, such innovations are not employed by conventional antiviruses and other security mechanisms. Various new and unconventional approaches have been proposed in the recent past to stop a malware to infect and propagate but the question of their effectiveness persists and how general users can benefit from new techniques matters the most.

One of the better approach would be design techniques based on the modus operandi of malware. Different analysis techniques with thorough approaches provide deep understanding of malicious pieces of codes [22] but such tools and techniques require time and computational resources, which is also one of the significant drawbacks of antiviruses. In this chapter, recently proposed analysis and detection techniques along with the conventional malware detection

techniques are discussed. Techniques including static feature extraction to support malware detection process are specifically focused in this chapter.

2.2 Background

One piece of software that is legitimate in one computer might be considered illegitimate in another computer or network. This logic vaguely identifies what a malware is along with its literal meaning; malicious software. Malware target vulnerability in a computer or in a network and exploit it to infect the targeted machine. This is done to use the computer or the entire network for several malicious reasons and usually it takes months, in some cases, years to identify that the network is infected. The taxonomy in which the malware are divided is based on the techniques they employ to infect their target. The following table presents the differences between types of malware present in the wild.

Type	Description
Virus	Viruses are passive in nature, they bind themselves to an existing program and propagate by duplicating themselves but requires to be copied to spread. They target benign executable files and corrupt them by attaching themselves [23]
Trojan Horse	Trojans act as a legitimate program and trick users to run it. Once executed, trojans can create backdoors in the system for different malicious reasons [24]
Worms	A computer worm is a standalone and active piece of code, which does not need a host program. It has the capability to replicate itself

	across the network automatically by targeting vulnerabilities [25]. It continuously scans through the network for further propagation and consumes a lot of resources while doing so [26]
Spyware	Spyware do not necessarily harm the computer or network but they hide and monitor activities of individual users or the entire network [27]. They can be part of Trojans or worms and send the stolen information to their server [28]
Bots	It is derived from the word robot, with malicious intentions of forming a bot network otherwise known as botnet. Botnet is a large network of geographically dispersed computers working as bots or zombies, controlled by a C&C (command and control) server also known as botmaster [29]. Forming a botnet is just the foundation, which can be used for DDOS and other large scale attacks [30]
Rootkit	The rootkit is not a simple malware with replication capabilities, it is a quite sophisticated software with multiple tools packed inside [31]. Once they have infected a computer or network, their embedded tools play a vital role to not only hide its processes in legitimate processes. It can escalate privileges of its processes without alarming the security software [32]
Adware	These are advertising support software designed to autonomously deliver advertisements in the form of popups or within a webpage [33]. A majority modern adware are used for revenue generation and don't require popping up because they work as a background

	process [34]. Adware authors also use them to transport spyware, which can spy on browser behavior and online transactions [35]
Ransomware	In the past few years, this specific category of malware have caused a lot of damage to many businesses, government services, and individuals [36]. Once infected, the system or an entire network along with its data can be locked and it will demand a ransom to unlock the files [37]. Ransomware encrypt the files in a unique way, which are not possible to decrypt using usually available techniques [38]. It follows the replication techniques used by a worm to proliferate its copies [39].

2.3 Malware Evolution

Since the first malicious piece of code was written, malware anatomy has evolved significantly. This anatomy of malware is continuously evolving to avoid the latest eradication techniques used by security organizations [20] [40] [41]. Unfortunately, the mainstream security mechanisms generally used do not match the advanced evasion and infection techniques used by the modern and lethal malware [42]. Every successful malware infection proves that the malware authors and their techniques are at least one step ahead of the eradication techniques used by their victims.

The sophisticated detection evasion techniques used by malware signifies that the amount of time a malware stays undetected is directly proportional to the destruction it causes to the infected machine or network [41]. The anatomy of

latest malware reveals amalgamation of several evasion techniques, such as; polymorphism, oligomorphism, and metamorphism [43]. The implementation of such techniques benefits the malware authors in two different ways; it makes it virtually impossible for a conventional antivirus to detect it, malware use these techniques to generate their mutated copies for further propagation. Camouflage and mutational techniques have two basic objectives; armoring and proliferating the malware, these techniques are used by the malware authors for the past three decades with continuous and rapid enhancements that can be perceived in the analysis of recently discovered malware [44]. Such enhancements and lack of timely detection and prevention of modern malware depict the scarcity in the conventional defense techniques [45].

2.3.1 Malware Obfuscation

The conventional malware detection techniques identify a malicious piece of executable mainly by matching its signature and heuristics with a set of stored malicious signatures and heuristics. If a malicious piece of code is modified even without changing the primary behavior, apparently it becomes a new malware. The process of malware obfuscation doesn't change the functionality of the malware at all, it only changes the signature of that file. This type of change makes the file a completely new entity for antiviruses [46].

There are many techniques that are collectively or individually used by the malware authors to obfuscate their malicious pieces of code. The commonly used techniques are discussed in the following sections.

2.3.1.1 Encrypted Malware

Encrypting a file is the basic approach to change its physical appearance. Encryption is also one of the pioneering techniques used to evade the detection by avoiding signature matching and other similar techniques. An encrypted malware is comprised of two parts; the encrypted malware and its decryptor. The encrypted malware contains the main body of the malware and the decryptor is assigned the task of decrypting the main body once the infected programs executes. Usually, encrypted malware use simple XOR and the decryption is performed with the encrypted code's XOR [47]. Although, the simple encryption was quite effective for evasion in the early days of obfuscated malware because antiviruses only relied on pattern matching. Modern day malware authors implement much complicated patterns for this process, which makes the decryption for malware analysts nearly impossible. Techniques such as; multilayer encryption, customized key generation, embedded message encryption is quite significantly used in modern malware [40].

2.3.1.2 Oligomorphic Malware

The initial versions of encrypted malware were hard to detect but with advancements in the security mechanisms the basic approach used by encrypted malware was outdated. The oligomorphic malware were a newer generation of encrypted malware, which used mutated decryptors to encrypt and decrypt the main body of the malware [41]. A malware named *Whale* used one of the famous implementation of this technique, it carried many decryptors while propagating in a network and using a random decryptor for each instance of encryption and decryption. Other implementations of such techniques were more lethal and

employed techniques with dynamic generation of decryptors, which avoided the need of carrying a large amount of decryptors while propagating making the whole concept more efficient [48].

2.3.1.3 Polymorphic Malware

The weaknesses of slight consistency in the oligomorphic malware gave birth to the new generation of malware known as polymorphic malware. Polymorphic malware use different type of encryption each time while replicating itself across the infected machine or network. They use mutation engine while replicating their instances, which allows the code to be transformed without the logic being changed [48]. While propagating in a network, polymorphic malware replicate itself in an encrypted form with a key different than the previous one and the decryption technique is embedded in the body. The polymorphic malware can evade the detection to a certain extent using such techniques because only a certain amount of decryptors can be generated with this technique [49].

2.3.1.4 Metamorphic Malware

The use of polymorphism in malware allows them to encrypt/decrypt using different techniques but metamorphic malware do not decryption to unpack itself in a constant body. Avoiding signature based detection of metamorphic malware is much more convenient as compare to the previously discussed types of malware, as they can evolve their code dynamically while moving from one generation to another [49]. They also can embed their code into one or multiple host programs making the malware nearly impossible to detect. Metamorphic malware use a combination of different obfuscation techniques to evolve into a newer generation, which is considerably dissimilar from its predecessor but

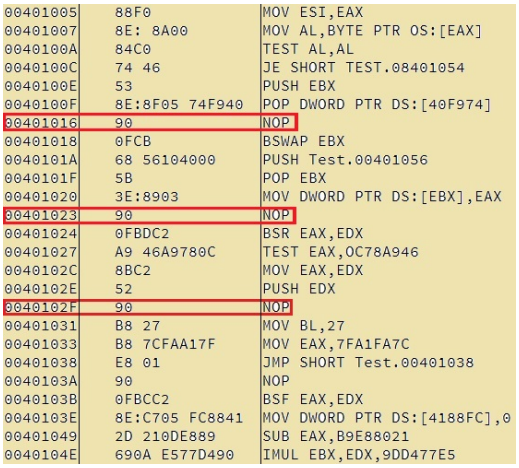
possess the same behavior. Following are the techniques that are used by the metamorphic malware [50].

A. Dead-Code Inclusion

```
00401005    88F0            MOV ESI,EAX
00401007    8E: 8A00        MOV AL,BYTE PTR OS:[EAX]
0040100A    84C0            TEST AL,AL
0040100C    74 46           JE SHORT TEST.08401054
0040100E    53              PUSH EBX
0040100F    8E:8F05 74F940  POP DWORD PTR DS:[40F974]
00401016    D8OB            RCR EBX,CL
00401018    0FCB            BSWAP EBX
0040101A    68 56104000     PUSH Test.00401056
0040101F    5B              POP EBX
00401020    3E:8903         MOV DWORD PTR DS:[EBX],EAX
00401023    43              INC EBX
00401024    0FBDC2          BSR EAX,EDX
00401027    A9 46A9780C     TEST EAX,OC78A946
0040102C    8BC2            MOV EAX,EDX
0040102E    52              PUSH EDX
0040102F    B6 86           MOV DH,86
00401031    B8 27           MOV BL,27
00401033    B8 7CFAA17F     MOV EAX,7FA1FA7C
00401038    E8 01           JMP SHORT Test.00401038
0040103A    90              NOP
0040103B    0FBCC2          BSF EAX,EDX
0040103E    8E:C705 FC8841  MOV DWORD PTR DS:[4188FC],0
00401049    2D 210DE889     SUB EAX,B9E88021
0040104E    690A E577D490   IMUL EBX,EDX,9DD477E5
```

Figure 2.1: Sample Code for Obfuscation

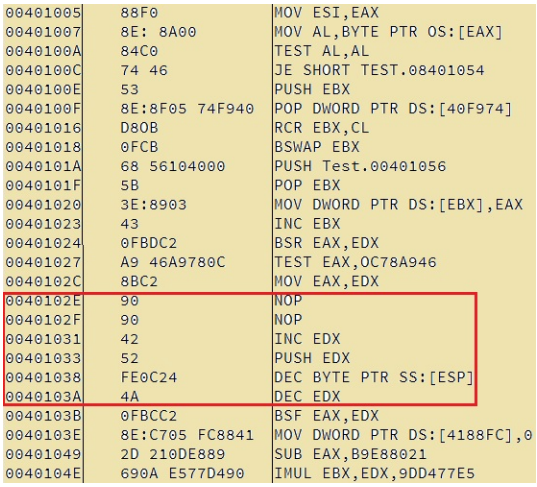
One of the simplest yet effective technique is including obfuscated or dead-code in the main body of the malware to evolve from one generation to another. The primary objective of this techniques is to make the evolved version of the code significantly different from the original code, which makes it extremely difficult to retrieve any operational hexadecimal search string [51]. These iterations in the code are identified as obfuscated because they do not change the behavior of the malware. The examples in Figure 2.1 and Figure 2.2 present the original and the obfuscated code respectively. Figure 2.2 presents the obfuscated version of the code, which uses the NOP command but the command doesn't make any difference to the code. This technique does obfuscate the code but it can be easily rectified by an antivirus only by eliminating the dead commands before performing the analysis.



```
00401005    88F0            MOV ESI,EAX
00401007    8E: 8A00        MOV AL,BYTE PTR OS:[EAX]
0040100A    84C0            TEST AL,AL
0040100C    74 46           JE SHORT TEST.08401054
0040100E    53              PUSH EBX
0040100F    8E:8F05 74F940  POP DWORD PTR DS:[40F974]
00401016    90              NOP
00401018    0FCB            BSWAP EBX
0040101A    68 56104000     PUSH Test.00401056
0040101F    5B              POP EBX
00401020    3E:8903         MOV DWORD PTR DS:[EBX],EAX
00401023    90              NOP
00401024    0FBDC2          BSR EAX,EDX
00401027    A9 46A9780C     TEST EAX,0C78A946
0040102C    8BC2            MOV EAX,EDX
0040102E    52              PUSH EDX
0040102F    90              NOP
00401031    B8 27           MOV BL,27
00401033    B8 7CFAA17F     MOV EAX,7FA1FA7C
00401038    E8 01           JMP SHORT Test.00401038
0040103A    90              NOP
0040103B    0FBCC2          BSF EAX,EDX
0040103E    8E:C705 FC8841  MOV DWORD PTR DS:[4188FC],0
00401049    2D 210DE889     SUB EAX,B9E88021
0040104E    690A E577D490   IMUL EBX,EDX,9DD477E5
```

Figure 2.2: Dead-Code Inclusion (Original Code in Figure 2.1)

To make dead-code inclusion more resilient against detection Figure 2.3 present an example, which obfuscates the code with impractical commands that are not exactly dead and do flow control for the compiler but doesn't necessarily make any difference to the functionality. This technique is hard to eliminate by conventional detection mechanisms because there are some practical differences in the both samples of the code.



```
00401005    88F0            MOV ESI,EAX
00401007    8E: 8A00        MOV AL,BYTE PTR OS:[EAX]
0040100A    84C0            TEST AL,AL
0040100C    74 46           JE SHORT TEST.08401054
0040100E    53              PUSH EBX
0040100F    8E:8F05 74F940  POP DWORD PTR DS:[40F974]
00401016    D8OB            RCR EBX,CL
00401018    0FCB            BSWAP EBX
0040101A    68 56104000     PUSH Test.00401056
0040101F    5B              POP EBX
00401020    3E:8903         MOV DWORD PTR DS:[EBX],EAX
00401023    43              INC EBX
00401024    0FBDC2          BSR EAX,EDX
00401027    A9 46A9780C     TEST EAX,0C78A946
0040102C    8BC2            MOV EAX,EDX
0040102E    90              NOP
0040102F    90              NOP
00401031    42              INC EDX
00401033    52              PUSH EDX
00401038    FE0C24          DEC BYTE PTR SS:[ESP]
0040103A    4A              DEC EDX
0040103B    0FBCC2          BSF EAX,EDX
0040103E    8E:C705 FC8841  MOV DWORD PTR DS:[4188FC],0
00401049    2D 210DE889     SUB EAX,B9E88021
0040104E    690A E577D490   IMUL EBX,EDX,9DD477E5
```

Figure 2.3: Inserting Impractical Commands

B. Registers Swapping

Another technique used by metamorphic malware is registers swapping, which was initially used by RegSwap malware in 1998. Malware using this technique will evolve from one generation by using the same code but by switching CPU registers [52]. This technique was initially useful for the malware authors but there was a weakness. In a conventional signature scan wildcard strings can be used to identify malware of newer generations with the signature of its predecessors.

C. Subroutine Permutation

Original code of a malware can be obfuscated with the help of this technique. By using subroutine permutation, a malware with *n* number of subroutines can generate *n!* number of unique variations of itself [53]. This technique was used by a malware *Ghost,* which had 10 subroutines and it had the ability to generate 3628800 unique variants of the original version but due to the persistent main content of individual subroutine it can be detected by using search strings.

D. Replacing Instructions

This technique uses equivalent instructions to substitute original instruction or a group of instruction. Instructions like XOR EAX, EA are equivalent to SUB EAX EA, if replaced, there will not be any change in the functionality of the code but they can generate a dissimilar hexadecimal instruction representation (opcode) [52]. Further details of this technique can be found in [54]. Figure 2.4 presents the sample code of Figure 2.1 with the application of substituting instructions.

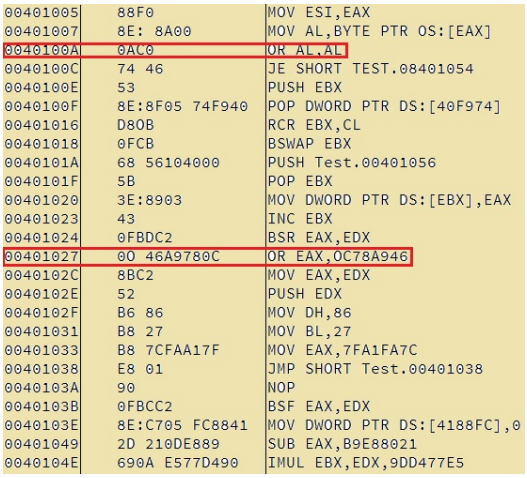


Figure 2.4: Substituting Instructions

E. Adding Jump Instructions

Another technique introduced to help the malicious code evolve dynamically from one generation to another was adding jump instructions in the code. The famous Windows 95 malware known as Zperm adopted this technique quite effectively. It dynamically adds and removes jump instructions in the main body, all these added instructions will point to a new instruction that will point to a new instruction [54]. This allows the malware to avoid generating a constant main body, which makes it extremely difficult for an antivirus to detect it. Figure 2.5 illustrates an example of how Zperm added jump instructions in its code. In each iteration of this malware, a new main body is generated that has no functionality difference but the control flow in the code is completely different.

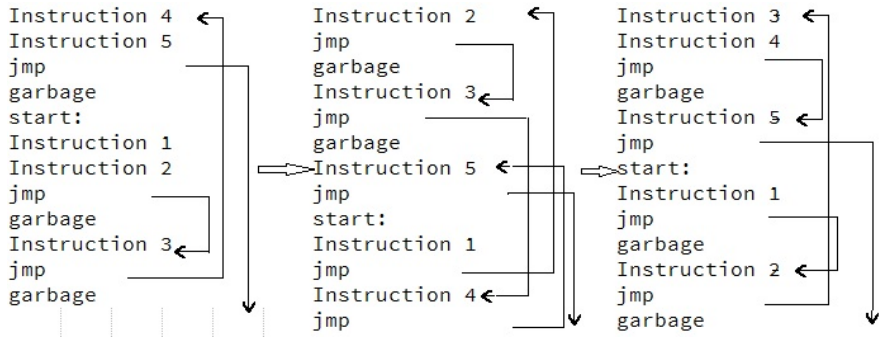


Figure 2.5: Sample Code of Zperm using Jump Instructions

F. Mutating Host Code

Mutating the host software code is another lethal technique used by metamorphic malware. This technique was pioneered by a malware known as Win95/Bistro that evolved rapidly into newer generations after infecting the host but while evolving and mutating its own code dynamically, it also evolved the host software by mutating its main body in every iteration [55]. This make things more complicated for security software to identify, random transformation of code was used by the mutation engine to generate new variants for this malware. Recovering the host software from this infection is nearly impossible as the malware not only mutates the main body of the host, it also obfuscates host's entry point, which doesn't allow the disinfection process to be completed [56]. Figure 2.6 presents a simple illustration of mutation and replication of a single malware sample.

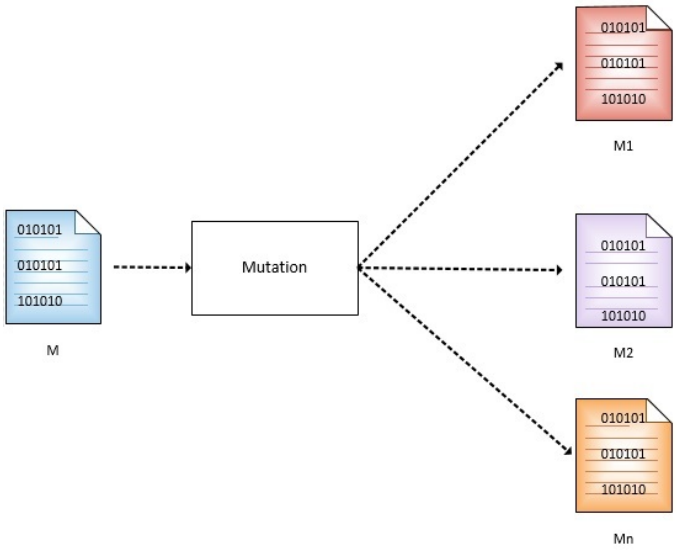


Figure 2.6: Mutation and Replication of a Single Malicious Sample

G. Code Integration

Infection of a malware using host code mutation is hard to detect but impossible to disinfect. Whereas, a more sophisticated and nearly impossible to detect techniques is code integration, which was pioneered by Win95/Zmist. The mutation engine of this malware has the ability to dissect an executable file into individual sections, it then substitutes itself with small code blocks in each section of the dissected executable and then rebuilds it [46]. This technique, if used properly, can allow a malware to flawlessly integrate its malicious code in the individual sections of the host executable, which is not only exceptionally hard to disinfect, it is impossible to even detect such infection by only using conventional detection techniques [56].

Apart from the obfuscation techniques used by modern malware to avoid detection, malware use some additional techniques to evade the efforts to understand their structure, characteristics and behaviour with the help of different

types of analysis. Anti-debugging is one of the techniques used by malware to avoid getting analysed. Figure 2.7 presents the anti-debugging APIs retrieved after statically analysing a malicious file. The advance analysis techniques discussed in the later section employ debugging tools to go through instructions contained in a file to operate in a system. The anti-debugging technique is implemented by using code checksums in runtime, decryption key generation with help of interrupts, monitoring API routines in debugging, monitoring registry keys. Many legitimate programs also use anti-debugging techniques in their code to avoid piracy but a legitimate and illegitimate file using anti-debugging techniques can be differentiated by comparing their implementations [57]. The anti-debugging APIs used mostly by malware are listed below.

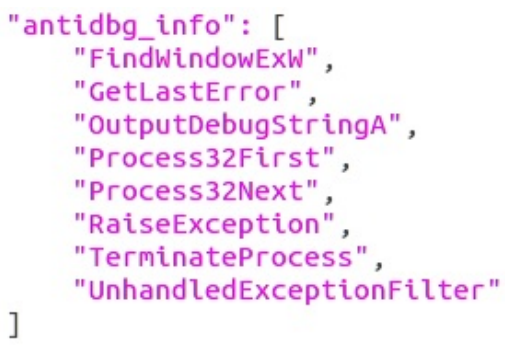


Figure 2.7: Anti-Debug APIs

Another most commonly used techniques by modern malware to avoid getting analysed is anti-virtual machine. Behavioural analysis techniques used against malware execute the malicious file in a virtualized environment to understand its objectives [58]. Anti-virtual machine technique is used by malware to avoid getting their objectives that could reveal their identity and variants identified. These techniques get activated as soon as malware identifies that it's been executed in a virtual environment, which stops the file to be completely unpacked.

2.4 Analysis of Malware

In the previous section, we discussed the evolution of malware over the period of decades and how many malware pioneered different obfuscation techniques to evade the detection mechanisms. In this section, we discuss different techniques that are used to understand the behaviours, characteristics, and objectives of a malware.

While analysing a malware it is pivotal to understand that one malware has a family of variants that could be in millions and it is practically impossible to capture and analyse each variant. The positive thing in this scenario is that the entire family of variants of one malware might have the same behaviour but the alarming thing is that each variant could have a separate objective while operating in an infected network or individual computer. Another thing that should be considered is that one malware can have various behaviours. Literature [59] claims that a single malware executing a set of malicious commands over the weekend can be replaced by a set of completely different commands that it executes on Mondays. This behaviour is usually observed in malware variants that specifically target enterprise networks and mainly perform their malicious activities during the weekend when continuous network monitoring is not possible. A similar approach is used by a malware, which goes in hibernation mode during the office hours and activates during night time to perform all the malicious tasks.

There are mainly two different types of analysis that can be performed on a malware; one to understand the behaviour and the other one for identifying the characteristics, they are known as dynamic and static analysis respectively. Both

types have their own benefits and play a significant role in comprehensively understanding a malware to detect it and prevent it from infecting and propagating.

2.4.1 Static Analysis

Statically analysing a malware is the most common technique that is used to understand its characteristics. Static analysis not only retrieves the basic characteristics of a file, it can give a comprehensive report that contains quite decisive information. As the name suggests, static analysis doesn't require the file to be executed and it only gathers the static information about the file [60]. There are several online and offline tools available that can be used to perform static analysis, many highly effective open-source tools can also be used for static analysis. Tools such as; VirusTotal [61], PEFrame [62], PEiD [63], PEStudio [64], Mastiff [65], and Pyew [66] straightforwardly generate analysis reports that can help to understand many simple yet decisive characteristics about a file. Many of these tools have graphical user interface that allows new analysts to grasp the idea of feature extraction. These malware analysis tools provide fully automated analysis with a limited requirement of setting up a simple laboratory, without the need of a high-performance computer. VirusTotal eliminates the need of setting up even a simple laboratory by providing a web platform for static analysis. This Google powered web platform uses a comprehensive engine comprised of fifty-nine antiviruses and provides thorough reports starting from basic string analysis to fully automated analysis. It also provides an API that can be integrated with any supporting tool or even through a command line. Static analysis is simple to perform and can provide a detailed

report of the static features of a malicious or clean file without the need of an isolated analysis environment, however, the behaviour of a file cannot be understood by a static analysis.

A much advance level of static analysis is performed in the form of reverse engineering, which uses disassembler to examine an executable's complete cycle of execution along with the embedded commands to understand the core objectives of the malware [67]. It is essential to know about the targeted operating system beforehand along with the system architecture, instruction sets, and assembly language. Reverse engineering is usually performed with the help of specialized tools, such as; OllyDbg [68], IDA [69], GDB [70], Immunity Debugger [71], and WinDbg. These programs can generate CFG (Control Flow Graph) that identifies the potential flow of the analysed executable. This not only helps to identify the possible behaviour of the executable, it can quite effectively identify the variants from one family of malware [55].

One of the obfuscation techniques known as instruction replacement, as discussed above, can cause obscurity in a CFG if it is implemented in the analysed executable. Additionally, malware that can dynamically change their code as they propagate within a single computer cannot produce a consistent CFG, which makes their overall behaviour hard to document.

2.4.2 Dynamic Analysis

Dynamic analysis is a much-detailed type of analysis and requires the file to be executed. It not only retrieves the physical characteristics, it can also identify the behaviour of a file. Unlike static analysis, dynamic analysis requires a sandbox,

which is a controlled environment and doesn't allow the malware to effect its surrounding with its infection while it's running [72]. Executing the malware in a sandbox allows the analyst to understand how a malware infects, how it propagates the infection, how it operates within a network or individual computer, and what its objectives are. This gives a detailed information about a malicious file and how it can be stopped.

Like static analysis, dynamic analysis also starts from a basic analysis and can go up to a quite comprehensive level. The basic level of dynamic analysis has the objective of identifying malware operations within a system [73]. This is performed in a virtualized environment, which replicates the original system and the original state of that environment is preserved. The malware is executed in that environment and once it is executed, the original state of the machine is compared with the new state to identify the changes made by the malware. This process doesn't give a detailed information about the malware as compared to the advanced dynamic analysis techniques but it is quite helpful to eradicate the infection of a malware from a system by identifying the changes it has made to a clean system [74]. This not only help to remove malware infections, it also doesn't require the resources usually required by a detailed analysis. This level of dynamic analysis is important like basic static analysis to gain the basic understanding of a malware, which allows to stop a malware and its further propagation.

Unlike the basic level of dynamic malware analysis, the advance level of dynamic analysis comprises of tools based on multiple techniques. In this level of analysis, each state of a malware while it's running in a controlled environment is

monitored closely, which includes the state of malware's code. The advance analysis is quite extensive, which is another reason that it runs in a controlled environment that allows the analysts to monitor each and every aspect of its functionalities and their implications on the system [75]. The detailed reports of such analysis contains the aspects of external API calls, function calls, internal and external network traffic, creation of new directories, alteration of existing directories, unauthorised ports access, changes in registry, dropped files, and state changes during the operational period [73]. This allows to understand the primary objectives of a malware based on its interaction with the system files, and entities within a network and outside the network. As discussed earlier, a single malware typically has a huge family of variants and analysing the entire family of variants that quite easily be in millions is practically impossible. This type of analysis gives a detailed understanding of malware behaviour, which not only allows to identify variants from the same family it also assists in formulating a solution to bring down the entire family of variants. Automated tools running as a web-service like Malwr [76] are quite useful and convenient for new and experienced analysts, as they don't require a sandboxed environment to be developed for dynamic analysis and they quite quickly provide detailed reports on many variants from a single malware family based on their behaviour [76].

2.5 Conventional Detection Techniques

In the modern era of computing malware infection is inevitable and so is the presence of at least one security software on individual computers. The discussion in the previous sections imply that avoiding a malware infection or even detecting an infection and removing it is merely impossible and the

detection techniques usually used cannot reach the level of stealth maliciousness of modern malware [20]. However, the conventional detection techniques used by antiviruses and other security software do protect the host systems against malicious attacks to a certain extent. These techniques are effective against previously known malware or the malware whose signatures and other apparent features are available in the database of security software. Major security software giants, such as; BitDefender, Symantec, Kaspersky, McAfee, etc. have a wide range of security software for both businesses and individual users. These software claim to provide a shield against modern malware and disinfect any previous infection by returning the affected software to its previous and legitimate state [77]. The techniques, which are used by these software are from a limited pool of techniques that is shared by all the security service providers. Although, many of these security service providers have some unique proprietary techniques and different implementations of conventional techniques, which makes them different from each other and distinguish their results but that doesn't raise the overall bar of malware detection rate.

Following section discusses the techniques that are most effectively used by antiviruses and other security software for malware detection and prevention.

2.5.1 Signature-Based Malware Detection

Signature detection is one of the commonly used techniques in antiviruses and other similar security software. It relies on sequences of specific byte codes that are unique to every file whether it's clean or malicious, these static footprints of malware samples are used to detect similar files in the host machine or network [78]. A small modification in the code can change the signature of the file,

however, it can still be detected based on the separate and accumulated signatures of individual sections of a file. The unique byte code sequences are used as a representation for each sample and stored in the database of antiviruses. If a file containing the similar signature is found in the host system and or network by an antivirus then it is classified as malicious [79]. Similar approach is used to identify clean files along with the authenticity of their publishers' certificate. Signature detection requires a comprehensive set of up to date signature that are regularly updated considering the massive number of malware captured every day. This gives rise to another problem that regularly updating and storing a large number of signatures requires access to similar amount network resources and storage space on the host machine. If the antivirus's signature database is not up-to-date with the latest signatures, which is usually the case, then it will not be able to detect majority of new threats faced by its users [77].

Lack of up-to-date signatures is not the only problem with this approach, as discussed earlier, the detection evasion techniques used by modern malware can dodge this technique by changing its source-code dynamically. The obfuscation techniques used by malware with metamorphic behaviour that can change their code dynamically as they propagate don't leave a static footprint as move laterally in a network or in a single machine. The signature of one of its sample is completely different from another sample and it can cause ambiguity for analysts and antiviruses. One of the approaches that can be taken is to target the mutation engine of such malware and detect them through their mutation engine [75]. However, many of these malware randomly choose their mutation engines from

a large pool of dynamically evolving engines, a mutation engine based detection can add on the existing problem of large resource consumption by these solutions.

2.5.2 Heuristics-Based Malware Detection

Heuristics detection is mainly used a supporting technique besides signature detection to make the detection process quick and accurate. The term heuristics-based detection doesn't accurately define the process because the main objective of this technique is to use defined algorithms to identify patterns of files that match the already identified patterns of malicious files [45]. Malware signatures are generated after a thorough static analysis, which also generate several patterns from each sample that are collectively called heuristics. These patterns are then incorporated with the signature database in antiviruses to support the process of detection. This type of detection doesn't necessarily use the collective patterns from one malware sample to detect similar malware, it also breaks down the patterns for detection [80].

Heuristics-based malware detection is based on static analysis, which makes it quite quick. It also can find variants from the same family based on pattern matching. As discussed above, the patterns generated after malware analysis, they are broken down and used to identify similar features present in a different file [81]. Unlike signature detection, heuristic detection is not static and predefined, it improvises based on the environment it is operating, which makes it hard for a malware to escape from it.

The techniques used by modern malware avoid signature detection by dynamically mutating themselves, which doesn't leave any static footprint on the system. With the help of heuristics generation a generic signature can be produced, which can be used against many, if not all, variants of a single family [82]. Although, there is a possibility of having several false positive with the implementation of this technique.

Based on the above discussion heuristic detection play a vital role in combination with signature detection to accurately detect malware but the reason why modern malware are still able to evade this combination is the limited amount of information that is used to generate the heuristics. A large majority of malicious files hook themselves with the legitimate files and if analysed the generated heuristics are a combination of patterns from clean and malicious files [80]. If such heuristics are used to detect malicious files, the malware that corrupt a small portion of legitimate files will be able to evade the detection. Consequently, if files are classified as malicious based on the small amount of alleged maliciousness then number of false-positive will significantly rise. Therefore, the combination of patterns used to generate heuristics need to be enhanced significantly to make more accurate decisions but this also means that more resources will be required to run such techniques.

2.5.3 Behavioural-Based Malware Detection

Behavioural detection is a technique that can successfully penetrate the detection evasion shield created by malware through obfuscation. It performs dynamic analysis of files to perceive their activities and behaviours in different operating environments, which are used to develop patterns to identify similar behavioural

patterns in other executables. As mentioned above, behavioural detection is based on dynamic analysis of executables that requires time and resources [59]. Although this technique can bypass the obfuscation techniques implemented by malware, it requires a detailed ruleset that explains the normal behaviours of executables in usual execution environments as compared to controlled or sandboxed environments. Without defining such parameters, it is significantly hard to identify a normal and an unsafe behaviour of an executable in a specific environment.

2.6 Recent Research Advancements in Malware Detection

In the previous sections, we have discussed different techniques used by modern malware to avoid getting detected by antiviruses and other security software, we also discussed analysis techniques that are used to understand the characteristics and behaviours of malicious executables. Additionally, we presented a discussion on different conventional detection techniques that are quite commonly used by security software to detect a malicious code, along with the foundations of these detection techniques. In this section, we are going to discuss different recent researches conducted that are relevant to our research along with their benefits and weaknesses. Our work focusses on different static analysis based heuristics extracted from a large sample of clean and malicious files to define rules to differentiate between both types. These heuristics are then used in conjunction with a combination of different machine learning algorithms. We specifically discuss recently conducted researches in the same area.

Use of different types of features extracted through static analysis or other methodologies has been proposed in several different studies [44], [24], [3], [83],

[48], [49], [74]. Many studies have proposed customised rules based on different features and heuristics from clean and malicious files [82], [81], [45], [86], [87], [88]. Machine learning has also been applied on a small set of heuristics by some studies to differentiate between clean and malicious files.

Using machine learning for the identification of malware has been proposed using several different techniques by many researchers [84], [85], [22], [89], [90], [52], [53]. Each of these studies have their own methodologies to approach the problem of malware identification, by increasing the true positive, and reducing the false positive rate. Majority of the research in malware detection is based on windows-based malware and only focus on the detection of one type of malicious code. However, more than 90% of the industrial environment is based on windows, therefore, the threat of windows-based malware is significantly higher.

The conventional techniques of malware detection, also known as signature-based detection used by antiviruses are still quite useful and it can flawlessly detect a known malware. These techniques are not very helpful when there is an attack from a new or unknown malware, which is why there is a huge gap in the industry, despite several studies in this area.

One of the most relevant studies in this area were conducted by Kolter and Maloof (hereon KM) [93]. They drew techniques from machine learning and data mining and applied them on their collection. In their study, they used a common text classification practice, n-grams, which tested the results of various classifiers on malware detection. The techniques included in their research were; SVM, decision trees, Naïve Bayes, and then applying boosting on each of the

techniques [84] [93]. The KM approach used the AUC (Area under Curve) of an ROC (Receiver Operating Characteristic) to evaluate the performance of their classifier, which they tested based on the highest information gains n-grams

They treated the presence or absence of the specified n-gram as Boolean on their classifier for boosted decision tree. As per their results, their model of boosted decision tree was able accomplish the finest accuracy rate out of all, achieving a 95% confidence interval AUC i.e. 0.9958 ± 0.0024. Boosting significantly enhances the performance of weak or unstable classifiers by decreasing their variance and bias but it can affect inversely on the stable classifiers, KM approach claims to improve the stable classifiers through boosting as well. The samples both benign and malicious used by them comprised of 1971 benign files and 1651 malicious files. The benign executables were retrieved from Windows OS (XP, 2000), and other online resources. Whereas, the malicious collection was obtained from MITRE Corporation and VX Heavens online repository. The KM research also used their approach of static heuristics technique for identifying payload functionality of malware. It identifies the functionality without dynamically analysing malware, which is an efficient way because it doesn't utilize resources required for sandboxing and eliminates the threats involved in dynamic analysis. They could identify payload functionality with the help of reverse engineering analysis reports of a subset of their complete collection. The KM approach showed promising results in two different directions; malware detection and payload identification. However, there are some weaknesses in this approach, considering the small sample size, missing 6 out of 291 malicious files is a real game changer in real life detection. This means if

the dataset is bigger, it can significantly increase the number of malicious files missed in a scan, which should be the main concern while detecting malware. Moreover, KM approach is not very effective for obfuscated malware and can easily omit such malicious files during the detection process.

MaTR approach is another noteworthy contribution in this domain in which they recreated the experimental environment of KM using same dataset and the same formula presented in equation 1 to highlight their weaknesses. MaTR approach used 31193 malicious and 25195 clean files in the initial experiment and compared their results with KM approach, which showed improvements over KM with the following mean and confidence intervals.

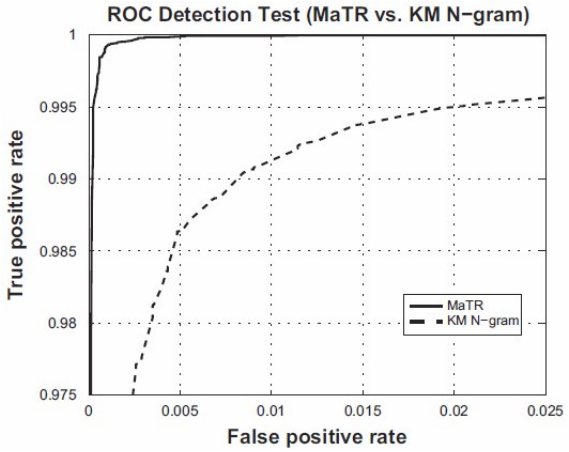


Figure 2.8: ROC Curves for KM n-gram Retest and MaTR [80]

The MaTR approach outperforms the KM approach and prove it by recreating the KM experiments, as presented in Figure 2.8 and Table 2.1. MaTR system design introduces an interesting approach by adding a human component in its system as illustrated in Figure 2.9. The reason behind introducing a human component is to give the system a capability of real-time detection [80]. This means that with the help of a human operator continuously monitoring the system logs, decisions

on legitimacy of the files can be made by looking at live anomalies occurring in the network. The human operator in this case provides appropriate responses for the type of malware rather than an automated and fixed response for all type of malware.

Table 2.1: Mean AUC and Confidence Interval of KM and MaTR, c.f. [80]

Method	Mean	95%CI
MaTR	0.999914	0.999840-0.999987
KM Retest	0.999173	0.998926-0.999421
KM Original	0.9958	0.9934-0.9982

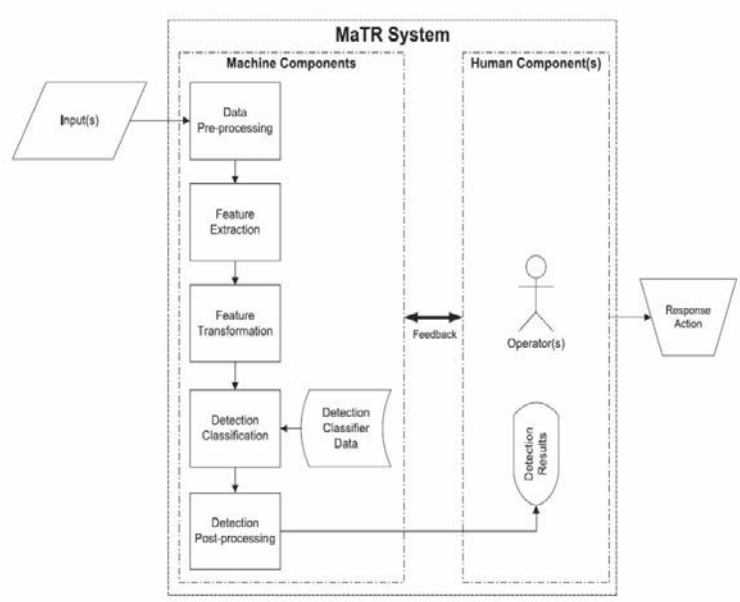


Figure 2.9: MaTR System Flow Diagram, c.f. [80]

For the classification of malware, MaTR approach uses bagged decision tree classifiers, which can enhance the performance of simple decision tree and make the results more accurate. The MaTR approach heavily relies on the human operator to take decision based on the detection results, which can be its main

weakness. In the live environment, when a malware attacks a system or an enterprise network an automated response is necessary because when it comes to malware detection time is a key component. A malware can propagate and replicate itself within minutes inside a network or in a single machine, which means that an automated response is necessary [85]. In MaTR approach, the parameter of response time and its affects are not mentioned. Additionally, the classification methodology of MaTR claims a very high detection rate, however, the performance on obfuscated malware is not present and it lacks the ability to do so.

There are several different studies that have used machine learning in malware detection by using different types of classifiers in their work. One of the studies have applied Decision Tree, Random Forest, Bagging, and Adaboost on the headers extracted from 32-bit PE files to differentiate between clean and malicious files. One of the main problem with this technique is that it compares the approach with antiviruses by highlighting its weaknesses but doesn't cover all the identified weaknesses. Additionally, the proposed approach in this study does produce promising results but as initial hypothesis of this study focussed on real-life implications, there is clear limitation of 32-bit PE files, new malware samples, and resource consumption.

Techniques such as [22], do have the ability to identify a vast range of previously unidentified malware by using a combination of static and dynamic malware analysis but one of the main limitations that are not covered in this approach is the amount of time needed to identify a malicious file in real-time, not to mention the amount of resources required for such a thorough and resource intensive

approach. This is a serious issue in malware detection, as the amount of time taken to detect an infection is directly proportional to the amount of damage caused by that infection.

Another relevant study [94] uses opcode generation through malware analysis and implements quite efficient machine learning algorithms. The results of this study are quite promising as well but one of the main weakness of this approach is the limitation of type of malware it's tested against. The collection of data used in this research is comprised of unpacked disabled malware and malware obfuscation is completely excluded, which raises the question about the benefits of this approach. Generating opcodes through statically analysing malware to train and test machine learning algorithms is an efficient approach but excluding a whole family of malware, which dominates the taxonomy of all malware families doesn't justify the application of this study.

As mentioned earlier, there are many studies that claim to effectively differentiate between clean and malicious files. Several of them use machine learning, fuzzy logic, and other techniques on statically or dynamically extracted features from both benign and malicious files. Nearly all of them target the vulnerabilities of conventional malware detection techniques that are commercially available. However, none of these studies present a solution that eliminates the generally highlighted weaknesses of conventional methodologies. One of the main weaknesses of antiviruses is the resource consumption of the host system while running in scan mode, which is an addition to the frequently discussed weakness, deficiency in detection rate. The studies claim to produce enhanced detection rate, quite often, lack the discussion about the real-time performance of their

approach, which includes the detection rate and more importantly the resources of host machine consumed while running in scan mode.

2.7 Chapter Summary

In this chapter, several aspects of malware infection, detection, and analysis relevant to our study have been discussed. Discussion starts with the evolution of malware and how the detection and infection techniques used by malware have enhanced in the past few decades. Numerous detection evasion techniques that have been used by malware over the years along with the combination of techniques that are currently used by modern malware to evade the detection process are also part of the discussion. Discussion also targets different analysis techniques that are used to analyse malware to extract their characteristics along with their behaviours and objectives, which led to the malware detection techniques that are generally used. Additionally, many conventional malware detection techniques generally used individually or in a combination by different antiviruses are discussed.

Later in the discussion, some recent research studies relevant to our work, which used features and heuristics extracted from clean and malicious files through static or dynamic analysis and later applied several different machine learning algorithms to enhance malware detection rate. With the help of detailed analysis of recent studies, we could identify the weaknesses still present in this domain, which helped us to design and implement a comprehensive solution comprising of an intelligent malware detection framework and its hosting model targeting multiple dimensions of the identified problem.

CHAPTER 3. AN INTELLIGENT MALWARE DETECTION FRAMEWORK

3.1 Introduction

A recent report claims that more than 7000 malware attacks are detected every hour and this number is for the attacks that are only targeting mobile devices [15]. This number is exponentially higher if the domain is broader, such as; personal computers, enterprise networks, web server, and other web enabled devices and infrastructure [10]. Out of millions of malware collected each year, majority of them are evolved versions of their predecessor [95], [96]. When a malware code is released in public, many of these malware are combined with a mutation engine, which allows other people with malicious intent to generate their version of that specific malware, such engines don't require a lot of programming or technical knowledge for doing so [97]. Majority of modern malware are equipped with automated mutation engines, allowing them to recurrently change their appearance, location, and other apparent features dynamically [98]. Obfuscation and replication techniques are used to change the apparent features of malware dynamically to avoid getting detected by antiviruses and even if a single instance of a malware is detected, multiple, yet very different, instances of the same malware are generated making it nearly impossible for the security software or the security analyst to detect it [99], [100].

Detecting a malware and preventing its infection or further propagation in a local network and in the wild requires an understanding of the infection and propagation techniques, which includes a comprehensive understanding of all the apparent features of malicious files along with how they behave in an

individual system or networked environment. The static and dynamic analysis of malware generate apparent and behavioral features of malicious files respectively, which allows the malware analysis and security experts to understand the dynamics of different types of vulnerabilities and the malware that exploit those vulnerabilities [91]. Both these analysis techniques are useful in different scenarios but if the main purpose is to accurately and rapidly detect a malware with minimum resource consumption, then static analysis is a better and reliable choice given that the tool used for analysis has a comprehensive and in-depth approach [101].

Analyzing a file statically doesn't guarantee that it is going to be perfectly identified as malicious or safe. Moreover, for a system to identify whether the file is safe or malicious, it must first learn how to distinguish between the two types by understanding the difference between their apparent features [101].

Integrating a combination of existing machine learning algorithms in the framework that will not only allow the framework to be rigorously trained to identify and differentiate between clean and malicious files, the comprehensive parameters used in the learning processes will help the framework to efficiently identify any unknown threats [92]. It is pivotal to use a rich set of features to train the algorithms, which could be produced with the help of a static analysis tool specifically customized to generate clean and comprehensive reports comprising of extremely relevant features [86].

Large enterprise networks and even individual computers generate a large amount of network and process logs, which are analyzed by security analysts

and administrators to detect any malicious behavior. These logs and similar data if analyzed properly can protect the system against many, if not every, type of attacks. The main drawback in this scenario is the dependency on analysts, which makes the whole process extremely slow and less reliable. Many researchers and corporate sector entities are incorporating machine learning for malware detection and to predict any future attacks with very high true positive rate [102], [103], [104], [105]. Proposed framework incorporates an optimum combination of machine learning algorithms that can efficiently detect a malicious activity without consuming a lot of system resources.

The approach taken in this research is a combination of conventional and novel techniques used for malware detection. This approach integrates the detection techniques generally used by antiviruses with state-of-the-art machine learning algorithms to develop a coherent framework that can be resilient and decisive against modern malware. The framework implements machine learning algorithms along with conventional detection techniques on a rich set of features extracted from clean and malicious files. To extract features from multiple files rapidly and accurately, an automated feature extraction tool was developed and later integrated with some open-source classes to make it more comprehensive. With the help of this static analysis based feature extraction tool a rich and diverse set of features were extracted from individual files from both classes; benign and malicious. The subsequent sections in this chapter present a thorough discussion of the overall framework comprising of classification methodology along with the analysis module that runs the feature extraction tool.

The proposed framework is more appealing as compared to many similar approaches for the following reasons:

- Although, dynamic analysis can retrieve a huge number of behavioral characteristics from a malicious file but the implications of this type of analysis include higher resource consumption along with analyst's involvement in the process. The comprehensiveness and automation in static analysis techniques can generate a set of features that can be used to identify a malware with much lesser resources.
- Real-time environment requires a detection mechanism with preemptive behavior that can detect a malware without any supervision. A framework that can learn from the heuristics of clean and malicious files and can differentiate between the two, can identify a malicious file without an in-depth analysis consuming time and other resources.
- The unique combination of multiple machine learning algorithm along with a rigorous validation technique ensures an unbiased and accurate prediction of threats.
- The comprehensive mechanism of classifying a file as malicious or safe, verifies the authenticity of the system along with the generation of detailed analysis reports, which are also used for real-time detection and prevention of known and unknown threats.
- Not many systems with such features generate output which can further be used for the enhancement of other systems or research objectives. The analysis report generation in an appropriate and easy to understand manner could facilitate the sharing of threat intelligence data on a larger

scale, which can also enhance the overall ability of the proposed framework.

In this chapter, we propose and evaluate an intelligent framework that can accurately detect both known and unknown malware threats. The framework is divided into two modules; first module uses an extensive tool which extracts the features from files that are later used to identify a clean or malicious file, second module use a unique combination of three different machine learning algorithms to identify a threat. The first module, which analyzes the files and generate a thorough report of their apparent features also can perform a basic classification that is useful in the long run, especially in the real-time detection. Whereas, the second module, which is defined as the classification module simultaneously apply machine learning algorithms; SVM, decision trees, and boosting on the extracted features to identify a malicious file. In the next section, we discuss the analysis results of around one million malicious files to understand the anomaly heuristics of such files.

3.2 Understanding the Anomalies

Before proposing and discussing the framework for detecting modern malware, it is essential to understand the anomalies that highlight the difference between the legitimate and malicious files. To develop a solution that accurately differentiates between the two file types, the fundamental step is to make the system learn about the features that make a file benign or malicious. As discussed previously, there are millions of malware captured every year and even though majority of them are just evolved versions of old and previously identified

malware, they do have some unique characteristics that allow them to stealthily penetrate a system or a network. Thoroughly analyzing a file statically can produce a rich set of characteristics for both benign and malicious file types and if both set of characteristics are compared, anomalies in the malicious set become evident given that, relevant and significant characteristics are compared. In this section, we discuss the features extracted through a thorough static analysis and their significance in the process of threat identification.

To understand and identify the characteristics of malicious files, we gathered many malware samples from various sources and analyzed them with PEframe, which is an open-source static analysis tool. We analyzed nearly one million PE files and generated a comprehensive set of quantifiable data. The details of test bench for this analysis are presented in Table 3.1.

Table 3.1: Static Analysis Test Bench Details

Tool/Machine	Details
Host Machine	Intel Core i7 4790 CPU @ 3.60 GHz RAM 16 GB, Hard Disk – 2 TB
Operating System	Ubuntu 14.04 LTS, 64 bit
Static Analysis Tool	PEframe with Python Scripts
Number of Samples	917705

The analysis performed on around 917705 malware samples produced a comprehensive set of data, which is pivotal for the methodology design. The main idea behind analyzing many malware samples is to retrieve features along with the conventional signatures, which can be used to differentiate between a benign and malicious file. The data produced after the analysis comprised of a good number of heuristics with some known and unknown anomalies. One of the major characteristic seen in nearly every modern malware and its variant is that they are packed and even if the basic behavioral characteristics are same, their appearance might be different because of different packers used for packing. With the help of this analysis, we retrieved the top 20 packers used by malware. Figure 3.1 presents the most popular packers used by modern malware, which is a very important attribute to consider. However, the most popular packers amongst malware are legitimate and belong to either Microsoft or other popular software providers that cannot be flagged as malicious just by identifying the name but majorly malware tend to use older versions of legitimate packers. This technique is specifically used to exploit a legitimate software, which is not supported by its publishing organization anymore, such software are not usually considered a threat by the antiviruses. Malware authors also use multiple packers to pack one malware to deceive antiviruses with legitimate packer on top of a packer originally used to pack the malware.

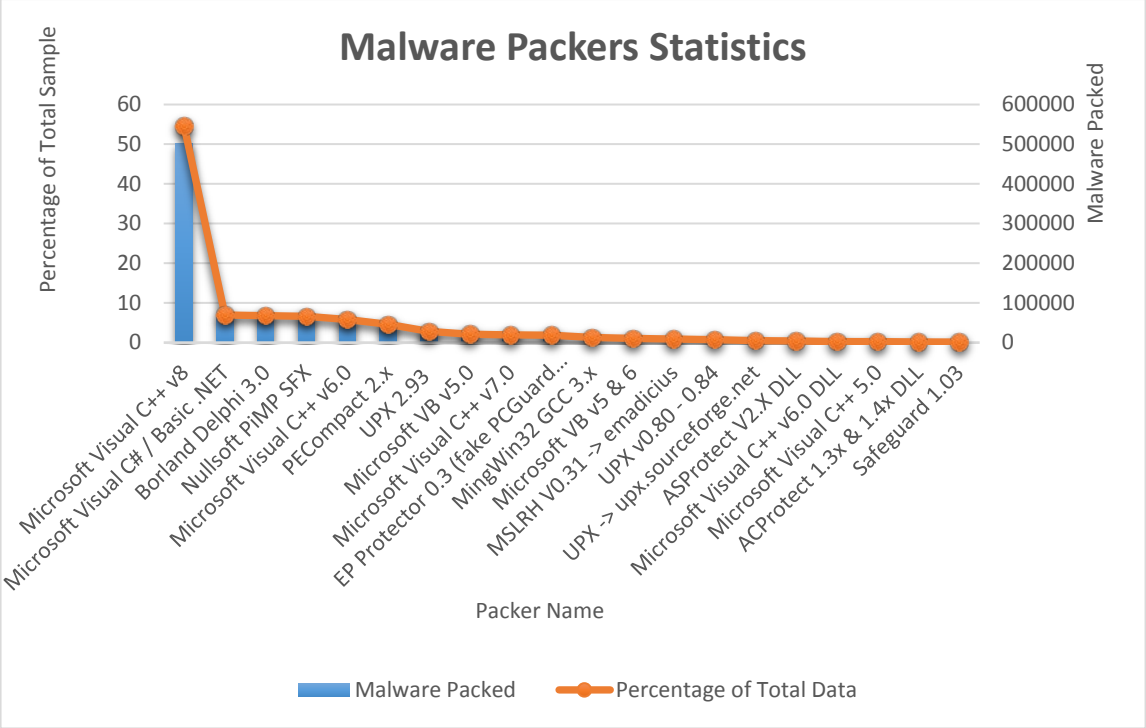


Figure 3.1: Statistics of Packers used by Malware

Many malware authors use techniques to avoid analysts understand their intentions by performing any type of analysis or reverse engineering on their packed code, which are known as anti-debug technique carried out with the help of APIs. Such techniques are also identified in the analysis, even though many analysis tools are not able to go past this point but they can retrieve if there is such technique used in an analyzed sample. The anti-debug technique is also used by many legitimate software publishers to avoid any attempt of piracy making it difficult to identify legitimate anti-debug and illegitimate anti-debug. However, a thorough static analysis can return the legitimacy of the APIs that are used by the analyzed file to implement anti-debug. Table 3.2 present the APIs and suspicious APIs retrieved from the malicious files, which denotes that a majority of these APIs are suspicious. This means that considering inclusion of APIs in the feature set can be quite useful.

Table 3.2: Anti-debug and Suspicious APIs

API Name	Number of Malware	Suspicious APIs
GetProcAddress	86000	GetProcAddress
Sleep	78000	Sleep
ExitProcess	76000	ExitProcess
CloseHandle	74500	CloseHandle
GetLastError	72301	GetLastError
WriteFile	69845	WriteFile
GetCurrentProcess	67458	GetCurrentProcess
GetModuleFileNameA	65472	GetModuleFileNameA
MultiByteToWideChar	63248	MultiByteToWideChar
GetCommandLineA	62147	GetCommandLineA
GetCurrentThreadid	61984	GetCurrentThreadid
WideCharToMultiByte	61547	WideCharToMultiByte
SetLastError	61471	SetLastError
FreeLibrary	61243	FreeLibrary
LoadLibraryA	61178	LoadLibraryA
GetCurrentProcessid	60521	GetCurrentProcessid
GetModuleHandleA	60341	GetModuleHandleA

UnhandledExceptionFilter	60314	UnhandledExceptionFilter
TlsGetValue	60158	TlsGetValue
ReadFile	60014	ReadFile

The tool used for analysis has a database of suspicious API signatures, which allows it to identify any API that falls under the category of being suspicious. All the files analyzed in this experiment were malicious, therefore, the analysis tool identified all the anti-debug APIs as suspicious. Even though legitimate files also use anti-debug feature but their APIs are not identified as suspicious and this specific feature can help identify a file as malicious.

Nearly all malware camouflage themselves to penetrate a network by using names that seem legitimate and important to the user. The analysis showed that many names used by modern malware to camouflage themselves recur quite frequently as presented in the Figure 3.2. These recurring names are quite relevant when trying to match and understand anomalies in a system or a network.

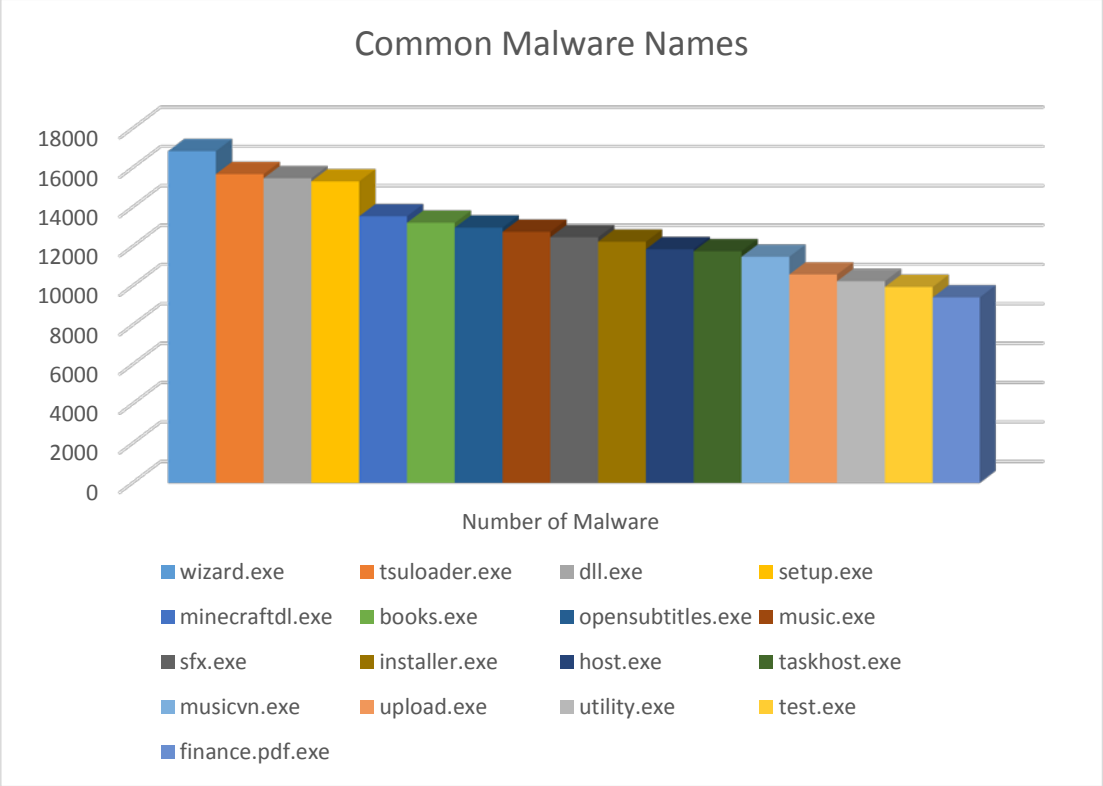


Figure 3.2: Commonly used Malware Names

Although, solely depending on these filenames is not a wise approach but it is important to flag an executable with a name such as; dll.exe, books.exe, music.exe, test.exe, etc. that are evidently suspicious. Another important aspect is the multiple sections in a malicious file and their names, as shown in Figure 3.3, these names denote the type of functionality each section holds that can be used to vaguely understand the motives of a malware.

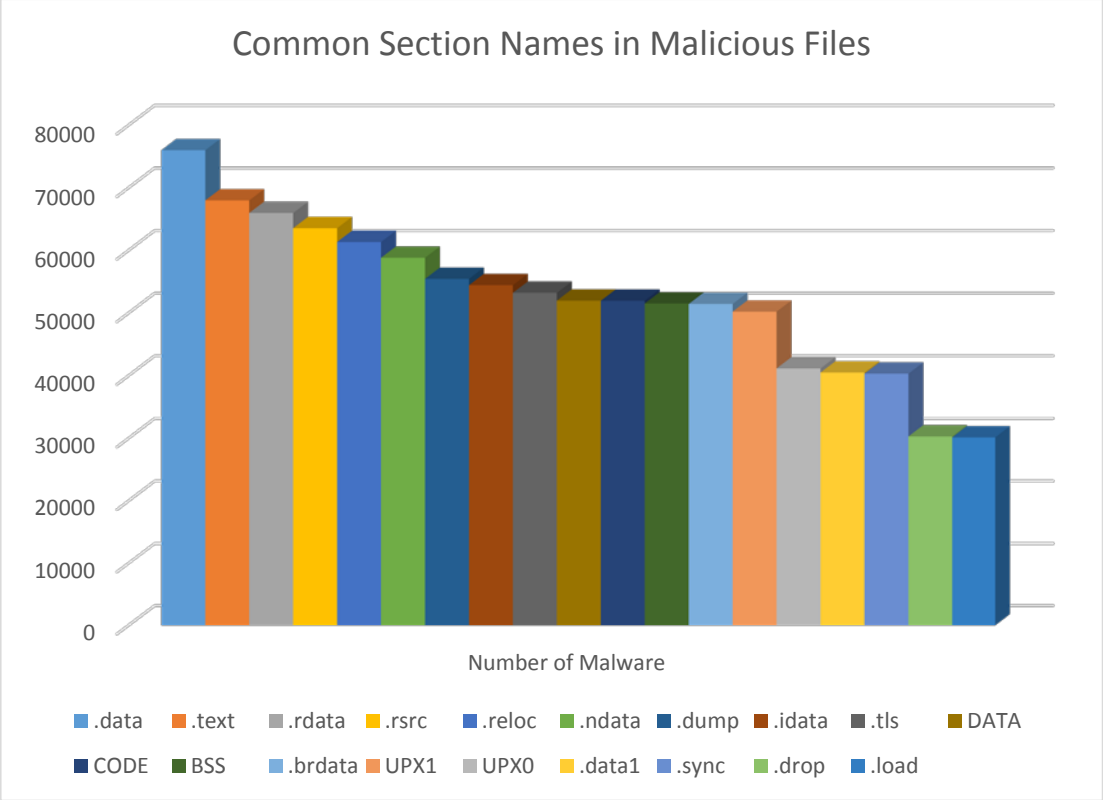


Figure 3.3: Mostly used Section Names in Malware

Malware these days are persistent in nature with long-term motives, after infecting a network their focus is to maintain the access by trying to connect to a command and control center, expand infection by importing more malicious data, exporting important data. Consequently, scrutinizing any IP address, URL, or email address retrieved from analysis can play a significant part in identifying a malicious file and can also be used to match with the similar data retrieved from newly analyzed files. Table 3.3 presents the top email addresses retrieved by performing the analysis.

Table 3.3: Top Email Addresses Retrieved during Analysis

Email Addresses
admin@mictosoft.com
info@microsott.com
support@gnail.com
claimnow@nationailottery.co.uk
admin@getwebcake.com
server@mitcsoftware.com
admin@rjlsoftware.com
accountrecovery@yontoo.com
sales@applee.com
iphone@aaple.com
account@yaah00.com
jdeb@autoscript.com
pop@harzing.com
sales@totusoft.com
sandy-cyf@163.com
sales@annazon.co.uk
returns@amazone.com
voucher@amazom.com
claim@iebay.com

The anomalies extracted through static analysis from a large set of malware are not significant when considered individually, however, if used collectively within a framework, the accurate classification decisions can pivot on such anomalies.

3.3 Building Blocks Overview

In the previous section, we discussed the features extracted after a thorough static analysis performed on a large set of malicious files. The discussion presented in the previous section shows the significance of each features or anomaly detected by the static analysis tool and how it can be used to differentiate between a malicious and a clean file. In this section, we present an overview of the building blocks used in the design and development of the framework proposed in this chapter. The framework proposed later in this chapter is based on two main modules; a) the analysis module and b) the classification module. Both these modules have their own significance, which is based on the uniqueness of the building blocks integrated to develop a comprehensive framework for malware detection.

3.3.1 Analysis and Features

The approach proposed later in this study primarily relies on the features extracted through static analysis. To extract the most relevant and pivotal features from the comprehensive set of benign and malicious files, a python-based automated analysis tool was developed that thoroughly analyzed the files statically and retrieve a rich set of decisive features, which are stored in separate files in a JSON format. These features are human readable and they are used for training, testing, and detection purposes by the classification module. To make

the static analysis and feature extraction tool more powerful and precise, some of the classes from an existing open-source tool PEFrame were integrated with the tool. After integration with open-source classes, the automated tool could work as an independent module in the overall framework. This feature extraction tool also integrates the private API of VirusTotal to endorse some of its many extracted results from a trusted third-party. Figure 3.4 [106] presents the features (without data) that are extracted after the analysis is performed on a single file.


```
{
    "hash": {   "sha256": "....",      "sha1": "....",        "md5": "...."    },
    "file_found": {        "Object": ["…." ], "Data": [ "..."], "XML": ["....",  "...."],
        "Linker File": ["...."],
        "Library": ["...."]    },
    "file_type": "....",
    "file_name": "....",
    "ip_found": [ "....",   "....",   "...."  ],
    "file_size": ".... " ,
        "virustotal": {   "total": "...", "positives": "...", "permalink": "....", "scan_date":"..."  },
    "pe_info": {  "compile_time": "....",  "packer_info": [...], "sections_number": "...",
    "resources_info": [ { "name": "....", "language": "....", "sublanguage": "....", "offset": "....",
                "data": "....", "size": "...." },
            {"name": "....", "language": "....", "sublanguage": "....", "offset": "....",   "data": "....",
                "size": "...." } ],
        "sections_info":[{"hash_md5":"....","malicious":"....","name":"....",        "size_raw_data": "....",
        "virtual_address": "....", "hash_sha1": "....","virtual_size": "...."            }, ],
        "import_function": {"….": [{"function": "....", "address": "...." } ],
            "KERNEL32.dll": [{"function": "....", "address": "...."},
                {".function": "....", "address": "...."} },
        "meta_info": {},
        "import_hash": "....",
        "export_function": [],
        "apialert_info": ["....", "....", "....", "...."],
        "sign_info": {}    },
    "url_found": ["....", ".....", "....", "...."],
    "mal_url": ["...."]
}
```


Figure 3.4: Sample JSON File without Extracted Features

Along with the conventional signatures used by the antiviruses for malware detection, this tool generates features that are not usually extracted by analysis tools and not as well used for generally detecting a malicious file. This not only makes this module unique, it also allows the classification module to use machine

learning algorithms on a unique and more relevant feature-set making the detection more accurate with much lesser probability of false positives.

We discussed the significance of packers and how malware authors use packers. Our analysis module not only identifies that the file under analysis is packed, it also identifies the name and version of the packer, along with using a third-party API to check the legitimacy of that specific packer. Additionally, our analysis module uses an updated list of packers generally used by malware [107] and the list is updated automatically with the help of the API. It is also used to identify if the analyzed malware is a variant of previously analyzed malware or belongs to a similar family of malware. The list of packers is stored in the database, which has a list of both malicious and legitimate packers updated frequently with every analysis supported by external API.

Some of the analysis tools extract the suspected API that the malware might try to access while executing. Similarly, our tool also extracts such APIs and to make the detection more accurate, we store the detected APIs in our database divided into clean APIs and malicious APIs. Therefore, when the analysis module extracts the APIs from a file it can be checked whether it's a malicious or non-malicious file and if the local database doesn't have any of the detected APIs the external sources are requested for legitimacy of the detected APIs.

Like APIs, our analysis module also extracts all the IPs the analyzed file is supposed to connect once it's executed. Such IP addresses may belong to a set of command and control servers controlling a botnet or something similarly malicious. These extracted IP addresses are stored in the database with two classes of IP addresses; clean and malicious, and then later matched whether

the detected IP address is malicious or clean. The database is continuously updated with every analysis.

The vital features from individual analysis are stored in the database and allows the system to identify a variant of an existing or previously analyzed malware. With the help of stored features, such as; hashes, packers, APIs, and IP addresses, the analysis module identifies if the currently analyzed file is a variant of previously analyzed malware.

As mentioned earlier, to endorse our results and initial classifications, we use external API powered by VirusTotal. With the help of this API we can implement the conventional detection techniques used by antiviruses by running the samples against an external engine comprising of 57 antiviruses. This API provides us with a verdict based on its own analysis, which plays a significant role in identifying the malicious file.

3.3.2 Machine Learning

The features extracted through static analysis play a significant part in the proposed framework. These features are then used to apply three different machine learning techniques used in this framework. The features extracted from PE files contain both malicious and clean features, which are divided into corresponding fields. We then use SVM (Support Vector Machine), decision tree, and boosting on decision tree to identify a malicious file.

3.3.2.1 SVM

Support vector machines, is a training algorithm which presents a decision boundary by maximizing the margin amongst training patters. The algorithm

presented by [108], has performed in an optimal fashion in many conventional scenarios, along with some studies similar to ours [93], [80], [109]. SVM creates a linear classifier, therefore, vector of weight $\vec{w}$ is its concept description and a threshold or an intercept, b. To make the problem linearly separable, a kernel function is used by the SVMs for mapping training data into a higher-dimensioned space. To set $\vec{w}$ and b that hyperplane's margin is ideal, quadratic programming is used, which means that distance to the closest examples of negative and positive classes is maximum from the hyperplane. While running, if $\langle \vec{w} . \vec{x} \rangle - b > 0$, positive class is predicted and if vice versa negative class is selected by the method. However, for larger set of problems, quadratic programming can be complex and expensive, whereas, to train SVM efficiently, SMO (Sequential Minimal Optimization) is a much better algorithm [110], it computes the probability of positive and negative class during execution [111]. For performance, we used implementation proposed in [111] for computing each class's probability and then we used positive class's probability as the rating. We used the following linear SVM formula to predict the positive classes:

$$t(x) = \sum_{n=1}^{N} \omega_n K(x, x_n) + \omega_0 \qquad (1)$$

Where $t(x)$ is the class label, which is either +1 (malicious) or -1 (benign), *n=1 to N* represent the sum of sample from 1 to N, $\omega_n K(x, x_n)$ is the weight of SVM and the kernel dot product and ω_0 is the bias.

3.3.2.2 Decision Tree

A decision tree is decision support mechanism with nodes that represent attributes and the leaf nodes that represent the class labels. Branches of the tree

that lead to children represent the values of the attribute. Values of the attributes and those attributes of an instance are used by the performance element to navigate in a tree starting from root and leading to leaves or an individual leaf. By choosing the attribute that perfectly separates the training samples into their appropriate classes, this is how a learning element generates a tree. Node, branches, and children are created for the attribute and the value of the attribute, the attribute is then eliminated from additional consideration, and the examples are distributed to the relevant child node [112]. This process runs in a loop until the same class examples are stored in a node and then class label is stored. Many implementation of decision trees remove subtrees which are expected to perform inaccurately on test samples, which avoids the overtraining of the whole algorithm. We have used MATLAB decision tree implementation for training and testing.

3.3.2.3 Boosting

Boosting is used for combining multiple classifiers to enhance the performance as compare to individual classifiers [113]. It uses ensemble methods, which significantly increase the overall performance, which has been tested and endorsed by many studies [114], [115], [116], [117]. By repetitively learning from a weighted dataset of a model, it creates a set of weighted models by assessing, and revising the dataset based on the performance of the model. During execution of the method, to predict the highest weight class, it uses a set of models and their weights. We only applied boosting on decision tree implementation, as our initial experiments didn't show any significance of

applying boosting on SVM. We used AdaBoost.M1 algorithm's [113] implementation in MATLAB to boost decision tree.

3.4 Proposed Framework Design

In the earlier section, we discussed the anomalies found when malware samples were statically analyzed. These anomalies play a vital role when combined to detect any previously known or unknown malware. However, relying on just these file anomalies is not enough to accurately detect a malicious file or an attack. As mentioned earlier, conventional detection techniques used by antiviruses are also important, if not sufficient, to differentiate between a legitimate and illegitimate file. If a framework is developed, which learns from the data retrieved through static analysis and conventional detection mechanisms then it will be able to accurately detect any malicious activity even if it was previously unknown. Therefore, we propose an intelligent malware detection framework, which integrates the mechanism of retrieving features and signatures through static analysis with conventional detection techniques used by multiple antiviruses along with three quite effective machine learning algorithms. This unique combination not only makes the whole process more reliable, it will make the detection mechanism more decisive and accurate.

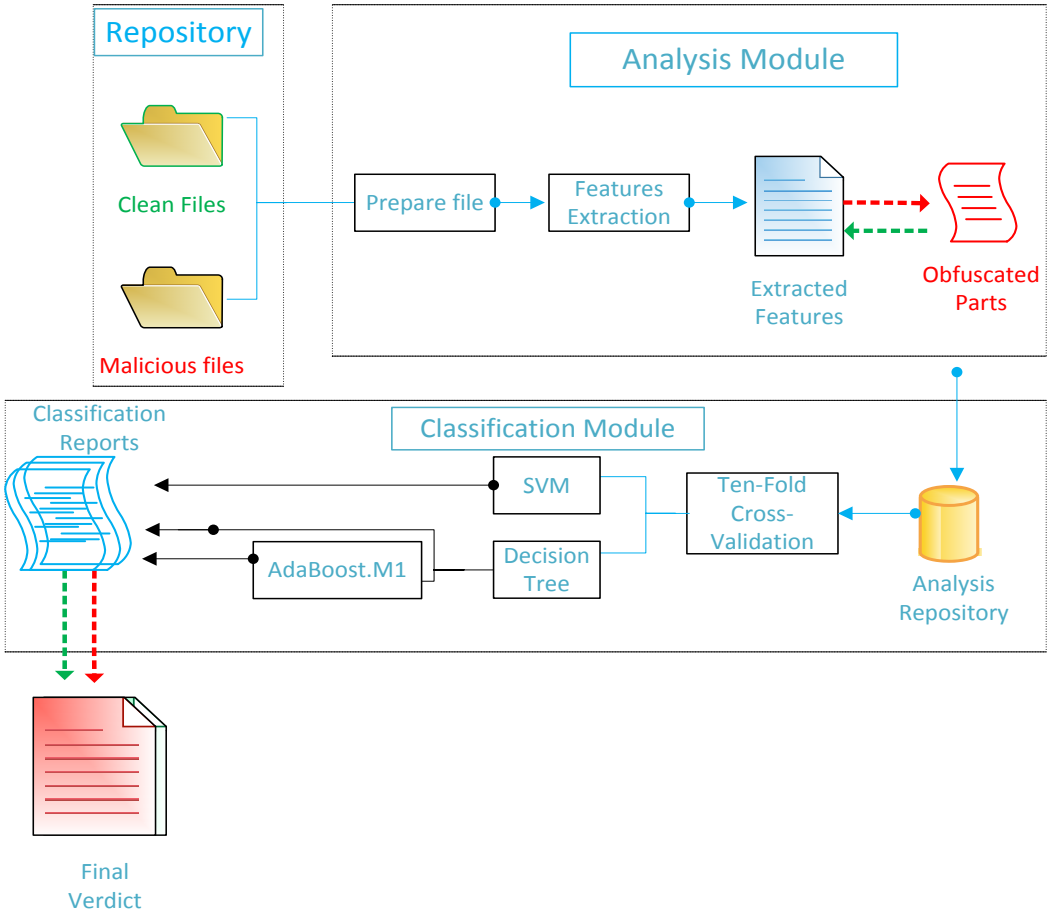


Figure 3.5: Proposed Framework Design [112]

Figure 3.5 presents the design of the proposed framework, which is comprised of two main components supported by the repository containing the clean malicious files. The analysis module performs static analysis of clean and malicious files generating comprehensive reports, which are then used by the classification module. The classification module uses machine learning algorithms to intelligently differentiate between clean and malicious files [106].

3.4.1 The Analysis Module

The analysis module comprises of the feature extraction tool, which statically analyzes portable executable files and generate a comprehensive set of heuristics based on the algorithms that are discussed in the later section.

3.4.1.1 Preparing a File

As illustrated in Figure 3.5 in the analysis module once the file is retrieved from the repository it is prepared for the analysis, which is extremely important and this technique is usually not present in other automated static analysis tools. The malicious files that are caught from the wild apparently use multiple obfuscation techniques, such techniques divert the analyst attention from the main file by placing multiple non-malicious files with garbage data stored in a zipped folder, which facilitates in generating ambiguous results if not removed. This, although, looks like a trivial step, but it makes the analysis more efficient, reliable and avoid any irrelevant analysis reports to be processed and stored in the database.

3.4.1.2 Extracting Features

The main function of this module is to perform a comprehensive static analysis of individual files by using a specifically designed static analysis tool and extract a rich set of features, which are stored in the respective JSON-based files. Although, this is the main function of the analysis module and does perform a comprehensive analysis but it doesn't require a lot of resources for doing so. The analysis is performed rapidly without the consumption of noticeable amount of CPU resources.

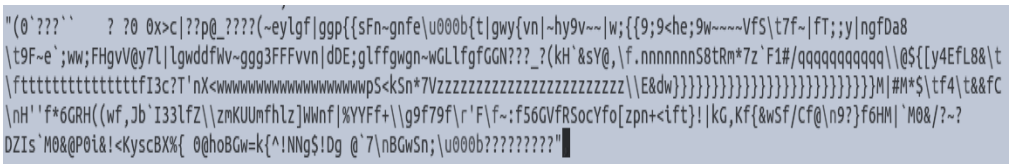


Figure 3.6: Obfuscated Part in Extracted Features

3.4.1.3 Removing Obfuscation

The features extracted after the comprehensive analysis contain a lot of parameters along with the obfuscated parts embedded in the malicious file, which makes it difficult for any analysis tool to identify the relevant features to classify a file as malicious or clean. Additionally, if the obfuscated parts are present in the feature set of a file then it will complicate the training of the entire classification module. The analysis module, after extracting the features from a file, identifies the obfuscation and removes it. Figure 3.6 presents the eradicated piece from the analysis report containing the obfuscated part. After removing the obfuscated parts, it reorganizes the contents of the JSON file to make it more comprehensible and in a proper sequence for later use.

3.4.2 The Classification Module

The classification module is a combination of machine learning algorithms applied on the large set of feature heuristics generated by the analysis module. Following is the sequence of operation of the classification module.

3.4.2.1 File Retrieval

The classification module works based on how well the machine learning algorithms are trained. The analysis reports of clean and malicious files containing their feature heuristics are stored in the analysis repository. Each file is retrieved and transferred to the classification part of the project.

3.4.2.2 Classification Techniques

This is the primary part of the classification module, which simultaneously runs the machine learning algorithms to differentiate between clean and malicious

files. The features presented earlier in Figure 3.4, are used to train support vector machine, decision tree, and boosting. SVM and decision tree run simultaneously, whereas, boosting is applied on decision tree to strengthen the weak classifiers. The implementation of these techniques is defined in the later section.

3.4.2.3 Classification Module Final Verdict

The final verdict of the classification module is based on the outcome of the machine learning algorithms. There are three algorithms using different techniques for classification in this module, the consensus from these algorithms generates the verdict, which is the final verdict of the entire framework. This verdict decides that whether a file is clean or malicious.

3.5 Modelling the Analysis Module

This section presents the design and implementation details of the analysis module. We will discuss the individual steps that are combined to form this module and the algorithms on which each step is based. Table 3.4 presents the notation used in the algorithms.

Table 3.4: Notations used in Algorithms

Notation	Meaning
F	Set of all files
f	A single file
F_M	Dataset containing all malware samples
F_C	Dataset containing all clean samples
EF_M	Dataset containing analysis reports of all malware samples
EF_C	Dataset containing analysis reports of all clean samples

EF_c	Extracted features of a single clean file
EF	Extract features
f_m	Single malicious file
f_c	Single clean file
EF_m	Extracted features of a single malicious file
ObF_m	Obfuscated elements in extracted features
f_{mal}	Identified malicious features
f_{clean}	Identified clean features
Mal_{DB}	Database of malicious features
$Clean_{DB}$	Database of clean features
$Hash(f)$	Hashes of a file $\{SHA1, SHA256, MD5\}$
$f(ExtAPI)$	Function to call external API
$ExHash_{mal}$	Hashes of malicious file pulled from external API $\{SHA1, SHA256, MD5\}$
$SecHash(f)$	Hashes of individual sections in a file $\{SHA1, MD5\}$
$IP(f)$	IP addresses present in a file
$ExIP_{mal}$	Malicious IP address pulled from external API
$Pack(f)$	Packer used by a file
$ExPack_{mal}$	Packer used by malware endorsed by external API
$ExAPI_{mal}$	Malicious API identified through external API
$API(f)$	APIs extracted from a file $\{Ant_{dbg}, Anti_{VM}, API_{call}\}$
$URL(f)$	URL extracted from a file
$Comp_t$	Compile time
Ant_{dbg}	Anti-debug API

$Anti_{vm}$	Anti-VM API
$ExURL_{mal}$	Malicious URL identified through external API
Ext_{verd}	External verdict
$Final_{verd}$	Final verdict

The analysis module not only extracts features of individual files, it also provides a verdict about the legitimacy of each file. The verdict provided by this module is based on a thorough process involving an external verdict supported by the private API of VirusTotal.com and an internal verdict based on a comprehensive decision-making matrix, which is discussed later in this section. There are series of different phases that complete this module and conclude the tasks that are required from it. Every individual file goes through these phases before it is classified as safe or malicious by this module.

Phase 1: File Retrieval and Feature Extraction

The whole process of analysis module starts with this phase where the module pulls an individual file from the repository. The module is connected to a repository that is divided into two separate sub-repositories; one for malicious and one for clean files. The module runs simultaneously in two different instances; one instance retrieves the malicious files f_m from the malicious repository F_M and the other one retrieves the clean files f_c from the clean repository F_C as described in Algorithm 1. After every individual file is retrieved, a through static analysis is performed and a rich set of features are extracted. The extracted features of an individual malicious or clean file EF_m and EF_c

respectively are stored in their respective repository EF_M andEF_C. The extracted features of an individual file from any of the two classes contain; $EF \leftarrow \{\text{Hash}, \text{lib}, \text{IP}, \text{packer}, \text{sec}_{\text{info}}, \text{anti}_{\text{dbg}}, \text{anti}_{\text{vm}}, \text{API}, \text{URL}\}$, which are stored as individual JSON reports. Many of modern malware samples have include obfuscated strings to avoid getting analyzed, if the analysis module finds any obfuscated code or strings during analysis, it eliminates it and rearranges the whole report for further phases.

Algorithm 1: Feature Extraction

Input: Malicious and Clean File f_m and f_c from F_M and F_C

Output: Extracted Features of Malware and Clean files EF_m and EF_c

Procedure: EF_m in EF_M && EF_c in EF_C

do

EF of f in F where $F \leftarrow \{F_M, F_C\}$

$EF \leftarrow \{\text{hash}, \text{lib}, \text{IP}, \text{packer}, \text{sec}_{\text{info}}, \text{anti}_{\text{dbg}}, \text{anti}_{\text{vm}}, \text{API}, \text{URL}\}$

while

F count > 0 && $EF_m \nexists EF_M$ || $EF_c \nexists EF_C$

// $EF_m \nexists EF_M$ means there is no repition of extracted features of a single file

if

$ObF_m \exists EF_m$

then

remove ObF_m

return EF_m

```
end procedure
```

Phase 2: Populating Database with Clean and Malicious Features

After the features are extracted and analysis reports are generated, the next step is to identify the clean and malicious features in both the repositories. The features are identified by two different techniques; through data already stored in Mal_{DB} containing a rich set of malicious features and with the help of external API. The data already stored in the $Mal_{DB}$was retrieved through a comprehensive analysis of around one million malware samples. The functionalities of this phase are also described in Algorithm 2.

As presented in Algorithm 2, after retrieving the analysis report of each file from its respective repository, which contains the extracted features. The algorithm matches the individual features by treating them as a separate entity. In this specific phase, the module extracts the hashes $Hash(f) \longleftarrow \{SHA1, SHA256, MD5\}$ from a single analysis report that is initially checked from the local database. If the local database of malicious features does not contain the $Hash(f)$ the request is sent to the external API, which returns the request by either classifying it as *true* (malicious) or *false* (non-malicious). If the request is returned with *true* then the $Hash(f)$ is stored in the Mal_{DB} and the response is sent to the next part of the module, which is responsible for the verdict. If the request is returned with a *false* then the $Hash(f)$ is stored in the $Clean_{DB}$. The next feature that is checked is $h(f) \longleftarrow \{SHA1, MD5\}$, which is similarly matched

as the previous one as presented in the Algorithm 2. This process is continued for three more features $Pack(f)$, $\boldsymbol{API(f)}$, where $\boldsymbol{API(f)} \leftarrow \{Ant_{dbg}, Anti_{VM}, API_{call}\}$, and $\boldsymbol{URL(f)}$. Each of these features are separately matched with the local and external sources and the results if retrieved from the external source are stored in the local database and forwarded to the next phase.

Algorithm 2: Populating Database of Clean and Malicious Features through External API

Input: $\boldsymbol{EF_m}$ in $\boldsymbol{EF_M}$ && $\boldsymbol{EF_c}$ in $\boldsymbol{EF_C}$

Output: $\boldsymbol{f_{mal}}$ in $\boldsymbol{Mal_{DB}}$ && $\boldsymbol{f_{clean}}$ in $\boldsymbol{Clean_{DB}}$

procedure: featureidentification(f)

pull $\boldsymbol{EF_m}$ in $\boldsymbol{EF_M}$ || $\boldsymbol{EF_c}$ in $\boldsymbol{EF_C}$

return $\boldsymbol{EF_m}$ || $\boldsymbol{EF_c}$

Hash Matching

$\boldsymbol{Hash(f)} \leftarrow \{\boldsymbol{SHA1}, \boldsymbol{SHA256}, \boldsymbol{MD5}\}$

$\boldsymbol{f(ExtAPI)}$

if

$\boldsymbol{Hash(f)} \nexists \boldsymbol{Mal_{DB}}$ && $\boldsymbol{Hash(f)} \exists \boldsymbol{ExHash_{mal}}$

then

$\boldsymbol{Hash(f)} \in \boldsymbol{Mal_{DB}}$

else

$\boldsymbol{Hash(f)} \in \boldsymbol{Clean_{DB}}$

end if

return $\boldsymbol{Hash(f)}$

Section Matching

$\boldsymbol{SecHash(f)}$

$\boldsymbol{f(ExtAPI)}$

if

$\boldsymbol{SecHash(f)} \nexists \boldsymbol{Mal_{DB}}$ && $\boldsymbol{SecHash(f)} \exists \boldsymbol{ExHash_{mal}}$

then

$\boldsymbol{SecHash(f)} \in \boldsymbol{Mal_{DB}}$

else

$\boldsymbol{SecHash(f)} \in \boldsymbol{Clean_{DB}}$

return $\boldsymbol{SecHash(f)}$

Packer Matching

$\boldsymbol{Pack(f)}$

$f(ExtAPI)$

if

$Pack(f) \nexists Mal_{DB}$ && $Pack(f) \exists ExPack_{mal}$

then

$Pack(f) \in Mal_{DB}$

else

$Pack(f) \in Clean_{DB}$

end if

return $Pack(f)$

API Matching

$API(f) \leftarrow \{Ant_{dbg}, Anti_{VM}, API_{call}\}$

$f(ExtAPI)$

if

$API(f) \nexists Mal_{DB}$ && $API(f) \exists ExAPI_{mal}$

then

$API(f) \in Mal_{DB}$

else

$API(f) \in Clean_{DB}$

end if

return $API(f)$

URL Matching

$URL(f)$

$f(ExtAPI)$

if

$URL(f) \nexists Mal_{DB}$ && $URL(f) \exists ExURL_{mal}$

then

$URL(f) \in Mal_{DB}$

else

$URL(f) \in Clean_{DB}$

end if

return $URL(f)$

IP Matching

$IP(f)$

$f(ExtAPI)$

if

$IP(f) \nexists Mal_{DB}$ && $IP(f) \exists ExIP_{mal}$

then

$IP(f) \in Mal_{DB}$

else

$IP(f) \in Clean_{DB}$

end if

return $\boldsymbol{IP}(\boldsymbol{f})$

Phase 3: The Verdict

The verdict is the last phase of this module, which basically presents a decision on a file that whether it is malicious or clean. The decision taken on the legitimacy of a file goes through a rigorous mechanism before it is finalized. This final verdict is based on two main conclusions; a) the external and b) the internal or final verdict, both decisions follow some defined principles.

Phase 3.1: The External Verdict

The external conclusion is based on the report retrieved through the private API of VirusTotal.com, which is the accumulated decision of 57 antiviruses but the decision that is returned as a response from this external source isn't always 100% positive or negative. Therefore, we further added constraints on the response from the external source before adding it as a decision in this module. According to our constraints, if the response coming back is positive more than 40% then the module considers it as a positive response, where positive means malicious. If the response is less than 40% then the module considers it as a negative response. The reason behind setting the threshold to 40% is that on many instances antiviruses suffer with high false-positive rates because they identify legitimate applications from unknown publishers as malicious and block them causing inconvenience for the users. Consequently, not many antiviruses make this mistake on similar type of files therefore there is a disagreement

between antiviruses over such file types and to avoid high false-positive rate 40% threshold level was decided.

Phase 3.2: The Internal Verdict

The internal verdict is more rigorous and deals with a larger number of parameters based on which the legitimacy of each file is decided. The matrix presented in Table 3.5 defines the idea behind the final decision-making process of this module that is based on many elements present in the local $\boldsymbol{Mal_{DB}}$ and $\boldsymbol{Clean_{DB}}$. These elements that are primarily part of the analysis reports of different files from both the categories are individually considered and accumulated in different combinations to finalize the verdict. In 21 out of 33 cases final verdict is contradicting with the external verdict, which means that the matrix presented in Table 3.5 does not only rely on the outcome of 57 antiviruses and consider the rest of parameters used by this decision-making mechanism equally important. In the decision-making matrix, "*T" (true)* means that the file is malicious and "*F" (false)* means that the file is safe. The matrix presented in Table 3.5 uses a novel approach of detecting malware with rigorous heuristic matching.

Table 3.5: Decision Making Matrix for Analysis Module

	Hash	Lib	IP	Packer	Section	Anti_dbg	Anti_vm	API	URL	Ext_ver	Inter_ver
Case 1	T	T	T	T	T	T	T	T	T	T	T
Case 2	F	T	T	T	T	T	T	T	T	T	T
Case 3	F	F	T	T	T	T	T	T	T	T	T

Case 4	F	F	F	T	T	T	T	T	T	T	T
Case 5	F	F	F	F	T	T	T	T	T	T	T
Case 6	F	F	F	F	F	T	T	T	T	T	T
Case 7	F	F	F	F	F	F	T	T	T	T	T
Case 8	F	F	F	F	F	F	F	T	T	T	T
Case 9	F	F	F	F	F	F	F	F	T	T	T
Case 10	F	F	F	F	F	F	F	F	F	T	F
Case 11	F	F	F	F	F	F	F	F	F	F	F
Case 12	T	F	F	F	F	F	F	F	F	F	T
Case 13	T	T	F	F	F	F	F	F	F	F	T
Case 14	T	T	T	F	F	F	F	F	F	F	T
Case 15	T	T	T	T	F	F	F	F	F	F	T
Case 16	T	T	T	T	T	F	F	F	F	F	T
Case 17	T	T	T	T	T	T	F	F	F	F	T
Case 18	T	T	T	T	T	T	T	F	F	F	T

Case 19	T	T	T	T	T	T	T	T	F	F	T
Case 20	T	T	T	T	T	T	T	T	T	F	T
Case 21	F	F	F	F	F	F	F	T	T	F	T
Case 22	F	F	F	F	F	F	T	T	T	F	T
Case 23	F	F	F	F	F	T	T	T	T	F	T
Case 24	F	F	F	F	T	T	T	T	T	F	T
Case 25	F	F	F	T	T	T	T	T	T	F	T
Case 26	F	F	T	T	T	T	T	T	T	F	T
Case 27	F	T	T	T	T	T	T	T	T	F	T
Case 28	F	F	F	T	F	F	F	F	F	T	F
Case 29	F	F	F	T	F	T	T	F	F	T	F
Case 30	F	T	F	T	T	F	F	F	F	T	F
Case 31	F	T	F	T	T	T	T	F	F	F	F
Case 32	F	F	T	F	F	F	F	T	T	F	T
Case 33	F	F	T	F	F	F	F	F	F	F	T

3.6 Evaluating the Analysis Module

We initially evaluated the analysis module to separately identify its malware detection capabilities. In this evaluation, we also compared the results of the analysis module with the 57 antiviruses used by Virustotal.com by using the same dataset for both. Separately evaluating the analysis module and comparing it with conventional malware detection techniques will not only endorse the level of competence of the proposed framework on the modular level, it will also highlight the possible weaknesses that can be eliminated in the classification module.

3.6.1 Data Collection and Experiment Environment

The data for this research consisted of 150000 malicious files and 87000 benign executables of Windows PE format. The benign executables were retrieved from fresh installation of Windows 7, Windows 8, Windows 10, Windows Server 2008, and Windows Server 2012. The malicious files present in the malware repository were obtained from our industrial partner Nettitude, which was a combination of different malware types. The distribution of both type of files is presented in Table 3.6.

Table 3.6: Distribution of Benign and Malicious Files

1	Benign	87000
2	Malicious	150000

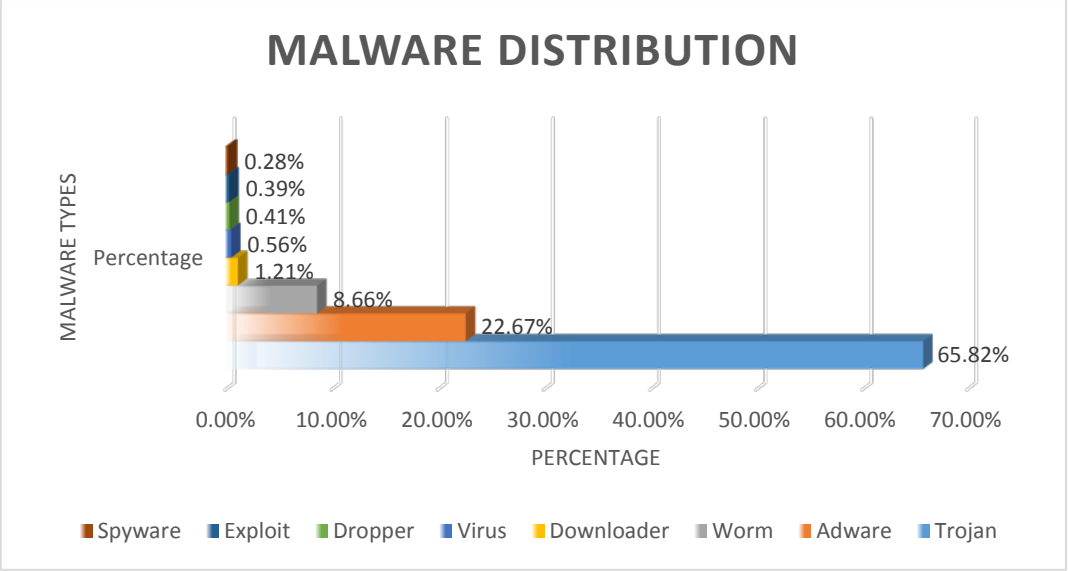


Figure 3.7: Malware Distribution in the Repository

The experiment environment was like the one discussed in the earlier part of this chapter, also presented in Table 3.7, but this time instead of using a standard static analysis tool, we tested the entire analysis module equipped with a customized and fully automated analysis tool with decision-making mechanism. The dataset was smaller than the one used in the initial experiment but it was comprised of unique and more recent samples. The dataset used in this set of experiments comprised of both malicious and clean files stored in their respective repositories. Figure 3.7 presents types of malware used in the experiments and their weightage in the dataset. The main idea behind this experiment was to evaluate the level of accuracy of the analysis module while it differentiates between clean and malicious files.

Table 3.7: Test Bench Details

Tool/Machine	Details
Host Machine	Dell PowerEdge T630 Xeon E5-2609V4 1.7GHz 32GB RAM 1TB HDD 5U Tower Server
Operating System	Ubuntu 14.04 LTS, 64 bit
Static Analysis Tool	Automated Analysis Module with feature identification and decision-making mechanism
Total Number of Samples	237000 (combined; benign + malicious)

3.6.2 Experiment Results and Analysis

The experiments performed by testing 237000 files against the analysis module returned some remarkable results. The results achieved form this experiment helped to evaluate two techniques by using a dataset of clean and malicious files. The results show the effectiveness of the analysis module for detection of malicious files, it also evaluates the effectiveness of conventional detection techniques used by antiviruses. As discussed earlier, the results received from the detection module are a combination of conventional detection techniques, thorough static analysis, and a decision-making matrix. The experiments performed, help to compare the results of both the techniques and identify the difference between their overall performances. The results discussed in the subsections are divided into two main categories; 1) the analysis module and 2)

antiviruses, the results shown under the label of antiviruses present the average reports of 57 antiviruses that are implemented by virustotal.com.

3.6.2.1 Understanding the Test on a Single File

This section discusses the analysis report of single files from both the categories. We performed the analysis on both malicious and clean files simultaneously and analysed the initial results to understand and evaluate the outcome and performance of our approach. Table 3.8 and Table 3.9 present the analysis report for individual malicious and clean files respectively, which explains the parameters used and their significance in the decision-making process. Both these tables show the Ext_{verd} and the rest of the parameters that are integrated based on the rules defined in the decision-making matrix to formulate the $Inter_{verd}$. The report of a single malicious file has a combination of outcomes present in green and red but the $Inter_{verd}$ is malicious, which means that the $URL(f)$ classified as malicious by the internal analysis based on the database of malicious URLs is not present as a malicious URL in the database of external source or not present in its database at all. This show the combination of both the approaches and the difference it makes while deciding on the legitimacy of each analysed file.

Table 3.8: Analysis Report of a Single Malicious File

	$Hash(f)$	$SecHash(f)$	$IP(f)$	$API(f)$	$URL(f)$	$Comp_t$	$Pack(f)$	$Final_{verd}$
SHA1	0a04af39f4a963b1577f23888d5f033974c6f265	1. "da39a3ee5e6b4b0d3255bfef95601890afd80709" 2. "d6ff4f8e61f4041 9d4680cf85dc62b5b47ed838a" 3. "7556471d7d55837c0ce397a5e127cfb5c6318a"						
SHA256	20d02be6fc4efad4577ba4c959d257e3adef67c9385febfb59053fdac8d68d41							
MD5	331eb511aa078ca6c1318d8bc5839abe	1. "d41d8cd98f00b204e9800998ecf8427e" 2. "f7aa81250ba378699001 38a6b59a55ec" 3. "acb8e418c691779c7059b0a8588cc9c4"						
			1.0.0.155	1. "CreateDirectoryA"; "CreateProcessA"; 2. "GetModuleFileNameA"; "GetModuleHandleA"; "GetStartupInfoA"; "ShellExecuteA"; 3. "Sleep"	1. http://bi.downthat.com/?script_error4=1& 2. http://ts-ocsp.ws.symantec.com07 3. https://secure-dine.top/jones/winstat2008.exe	2010-04-29 03:54:59	"UPX -> www.upx.sourceforge.net"	
Ext_{verd}	True	True	True	False	False	True	True	True
Int_{verd}	True	True	True	False	True	True	True	

Table 3.9: Analysis Report of a Clean File

	$Hash(f)$	$SecHash(f)$	$IP(f)$	$API(f)$	$URL(f)$	$Comp_t$	$Pack(f)$	$Final_{verd}$
SHA1	1a5f5cb368e2981281fc4ee38d9b8a82facfaac1	1. "08bc14c9f2dc95415955ce4fe237525560bebc18" 2. "3548399ecb4a607eaf7985d5bdad897ae7af0984" 3. "ca58ca064e98bb053732115b2df025773218493c" 4. "249eedbe46a06f8c69ded37c72fdc36a54967c6a"						
SHA256	1dd3056d3778585 1fee1b5503635700 86e0e9abbe4fb2078e8e0dc116f5b8a33	N/A						
MD5	b2fdf2b8ea10f18d8f001fd57d0743b8	1. "93a2bb181f244838f51fa8879ee8a21b" 2. "e5e35887f17208b86e2711d6831410b5" 3. "f9c2a9e214a48a675fa9fd2fc748d9be" 4. "829b4e4a508dd72f8532f54e5636ca91"						
			None	1. "CopyFileA", 2. "CreateDirectoryA", 3. "CreateFileA", 4. "CreateFileMappingA", 5. "CreateFileW", 6. "CreateProcessA", 7. "CreateProcessW", 8. "CreateThread",	1. "http://crl.microsoft.com/pki/crl/products/CSPCA.crl0H" 2. "http://crl.microsoft.com/pki/crl/products/tspca.crl0H" 3. "http://www.microsoft.com/pki/certs/tspca.crt0", 4. "http://www.microsoft.com/msi/patch_applicability.xsd"	2010-01-10 04:21:30	VC8 -> Microsoft Corporation
Ext_{verd}	False	False	False	False	False	False	False	False
Int_{verd}	False	False	False	False	False	False	False	False

3.6.2.2 Comparing Malware Detection Performance of the Analysis Module and Antiviruses

In this section, we thoroughly discuss and compare the performance of the analysis module proposed and implemented in this study with the conventional detection techniques from different aspects. The main objective was to design a method that uses the conventional detection techniques and introduce additional techniques that could enhance the overall detection rate. The comparison presented in Figure 3.8 illustrate the significant difference between the detection rate of the analysis module and the detection rate of antiviruses. The difference of 23.7% between the two approaches highlights the proof of performance enhancements in the analysis module, which detected 87.3% of the 150000 unique malware samples. This not just proves that the analysis module has a higher detection rate as compared to the conventional techniques, it also makes it more precise.

The evaluations are performed based on the following equations:

False Positive Rate (FPR): negative samples classified as positive.

$$\text{FPR} = \frac{FP}{TN + FP}$$

Recall/ True Positive Rate: actual positive samples detected.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision/ Positive Predictive Value (PPV): actual positive samples for all the positive detections.

Precision/ PPV = $\frac{TP}{TP + FP}$

Accuracy: a measure of the true detections.

Accuracy (ACC) = $\frac{TP + TN}{TP + TN + FP + FN}$

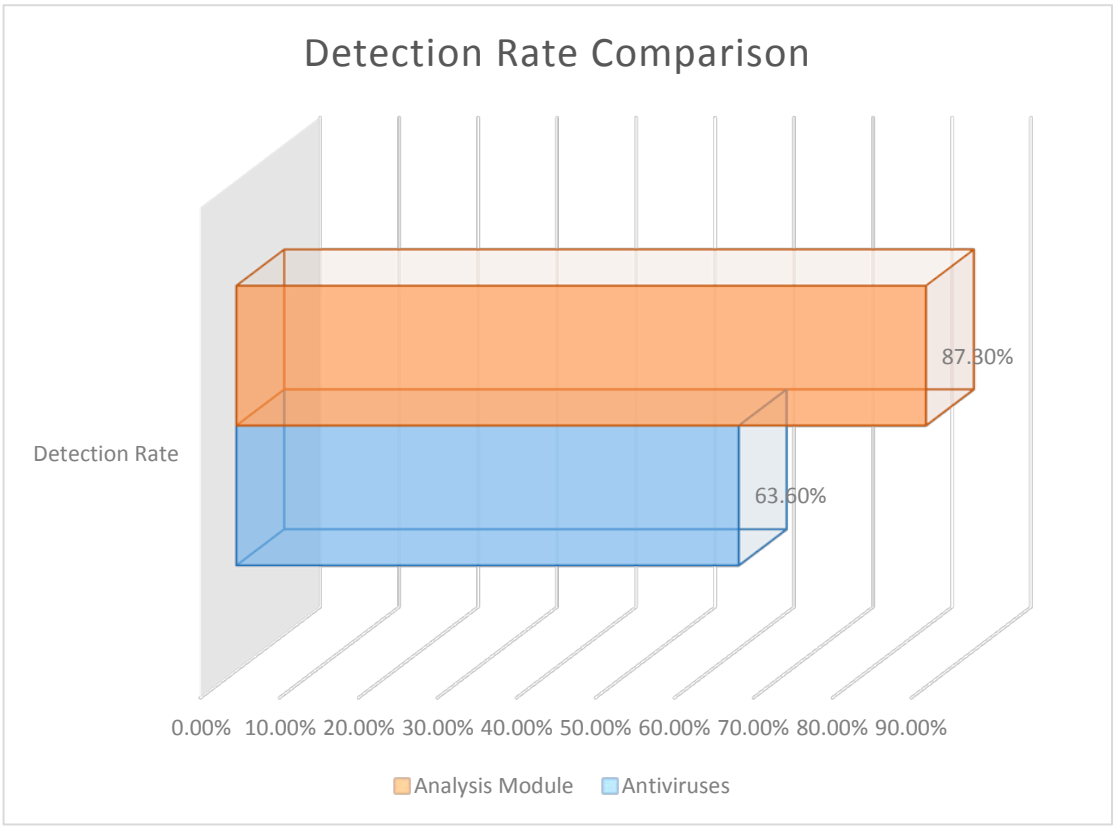


Figure 3.8: Detection Rate Comparison between Analysis Module and Antiviruses

Results presented in Figure 3.9 compare the difference between the two approaches in terms of TP, TN, FP, and FN. The figure identifies that apart from the true positive rate of both the approaches, the analysis module has higher accuracy in identifying the benign files as non-malicious with a 6% higher rate. The FP and FN comparison also shows a higher level of accuracy by the proposed approach.

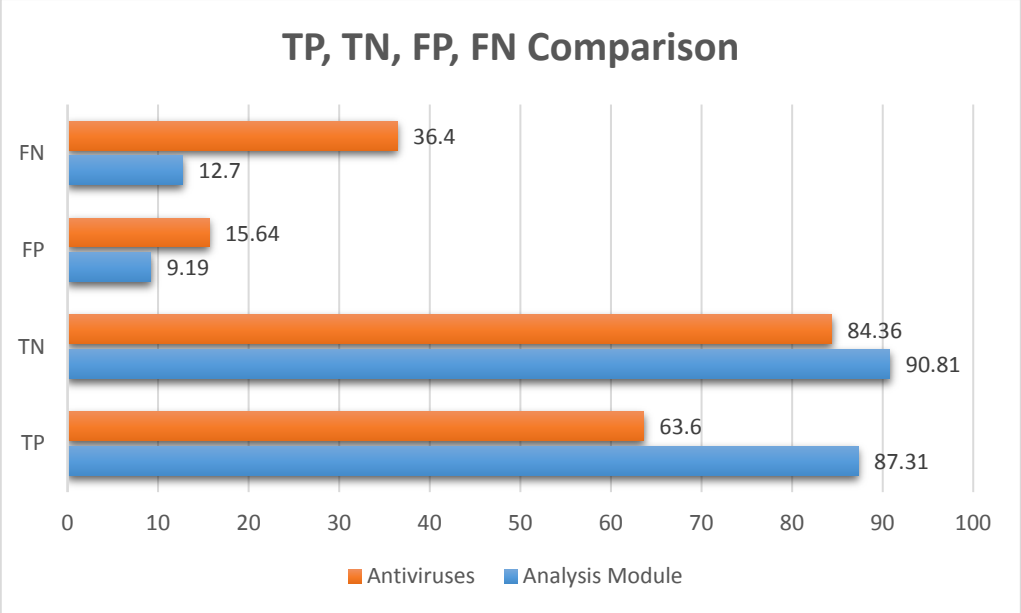


Figure 3.9: TP, TN, FP, FN Comparison

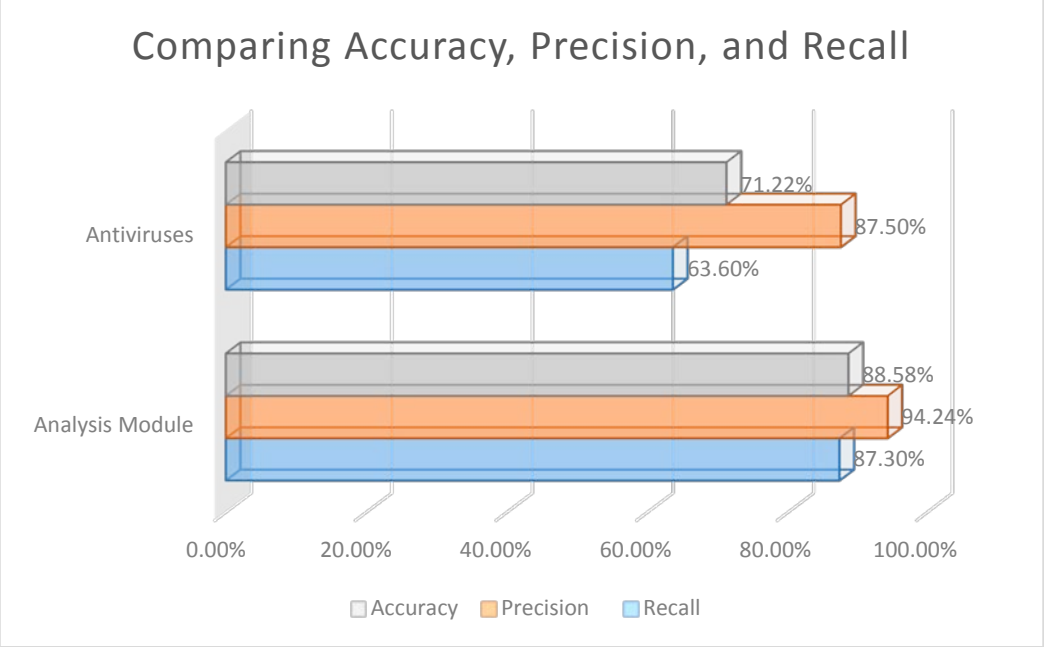


Figure 3.10: Accuracy, Precision, and Recall Comparison of both Approaches

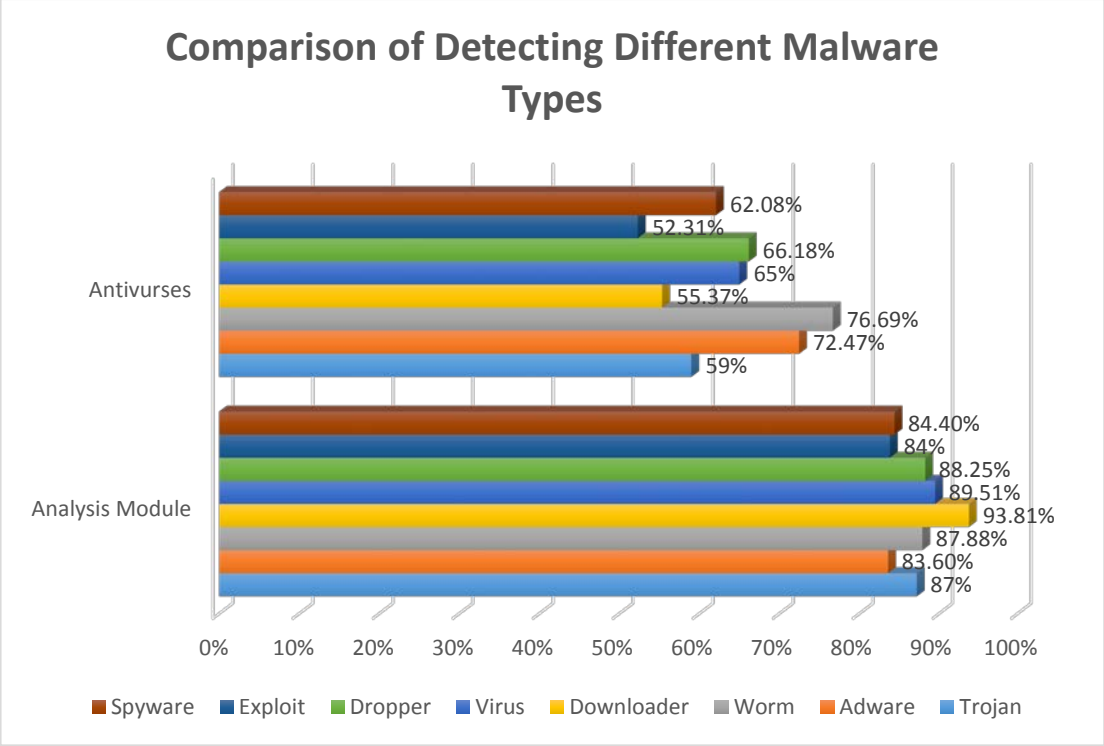


Figure 3.11: Comparison of Detection Rate with Respect to Malware Types

The comparison of results presented in Figure 3.8, Figure 3.9, Figure 3.10, and Figure 3.11 illustrate the outcomes of both the approaches and identifies efficacy of the analysis module. In Figure 3.11, we have presented detection rate with respect to different malware types that were present in the repository. This comparison presents an interesting set of numbers, which shows the weaknesses of conventional techniques and strengths of our approach in specific areas. The antiviruses result show that they perform comparatively well while detecting some specific types of malware such as; spyware and worm. However, the conventional approach didn't perform well while detecting viruses, downloader, and trojan. Trojans are in majority in our repository and in the wild, however, antiviruses combined were only able to detect 59% of these files, whereas, the analysis module detected 87% of trojans. This difference of 22% in

detecting trojan is significant and demonstrate the strengths of the proposed module.

As discussed earlier, the analysis module is combination of conventional detection techniques and a decision-making matrix. The decision-making matrix calculates the final verdict based on the parameters generated through comprehensive static analysis and external API, which gives the verdict of 57 antiviruses. The difference in performance of commercial antiviruses and the analysis module shows the effectiveness of the decision-making matrix. The decision-making matrix has not only enhanced the overall detection rate, it has also enhanced the level of accuracy in detecting different types of malware.

However, if the numbers in Figure 3.8, Figure 3.9, and Figure 3.10 are compared and analysed, it is understandable that although the performance is much better as compared to the conventional technique, the analysis module doesn't have the optimal output. The 9% and 12% false positive and negative are still quite high if real world scenarios such as enterprise networks are considered. To lower the numbers of false positives and negatives and further enhance the positive detection, it is required to add a layer that could use a different type of scrutinizing procedure.

3.7 Evaluating the Framework

Previous sections present the details of architecture and methodology of the approach we have proposed in this research. In this section, we present the observations of multiple experiments performed to evaluate the methodology proposed. We performed four different experiments on a standard experimental

design to help us evaluate the methodology from different aspects. In our first experiments, we used a smaller dataset of both benign and malicious executables and applied all the classification techniques. In the second experiments, we used a larger dataset to apply the classification techniques to monitor the enhancement in the detection rate. In the third experiment, we introduced obfuscated malicious files that are previously unknown and make the overall dataset much larger, which allowed us to observe the performance of the classification techniques against a much difficult dataset. The fourth experiment was performed on real-time data where we left the system running for more than two months.

3.7.1 Experimental Design

To validate the classification techniques used in this framework and generate unbiased classification reports, we implemented stratified ten-fold cross-validation. The cross-validation technique assesses the predictive models by distributing the dataset of samples into two sets; one for training and one for testing. This technique is executed based on K-folds and the original sample is divided into K size subsamples. Out of these K size subsets, one of the subset is kept for testing and K-1 subsets are used for training purpose. The whole process of cross-validation is reiterated K times to ensure that each of the subset is used exactly once as a testing set. The results after the process are then accumulated and averaged to calculate a final estimation. The main benefit of this technique is that all the samples are used for both training and testing processes and each of the sample is used for testing exactly once, which removes any chances of biased calculation and validates the predictive model rigorously.

To further evaluate the model, we used a formal analysis technique known as ROC (Receiver Operating Characteristic) analysis [118], which presented the true-positive rate of the model against the false positive rate.

3.7.1.1 Small Dataset

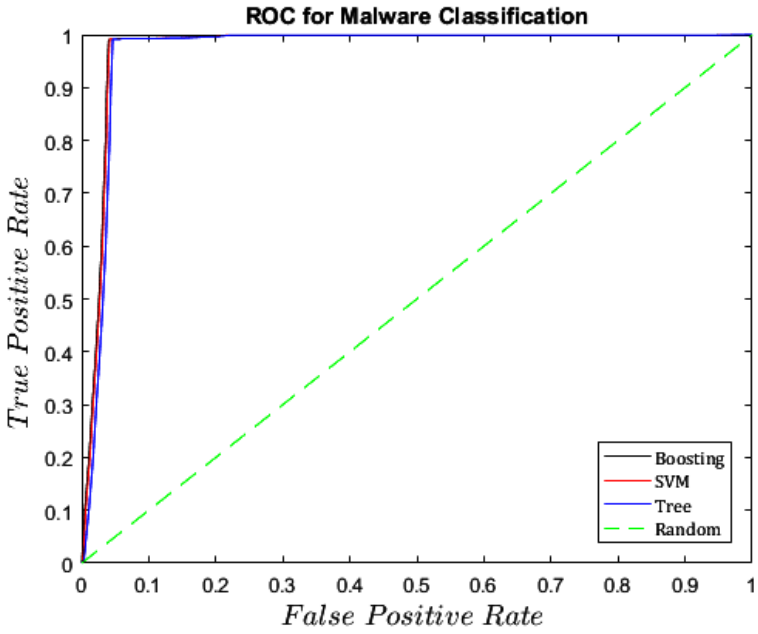


Figure 3.12: ROC curves for malware classification from a small dataset

Table 3.10: Result of applying classification techniques on extracted features of a smaller dataset

Method	Area Under Curve (AUC)
Decision Tree	0.9708
SVM	0.9727
Boosting	0.9747

3.7.1.2 Large Dataset

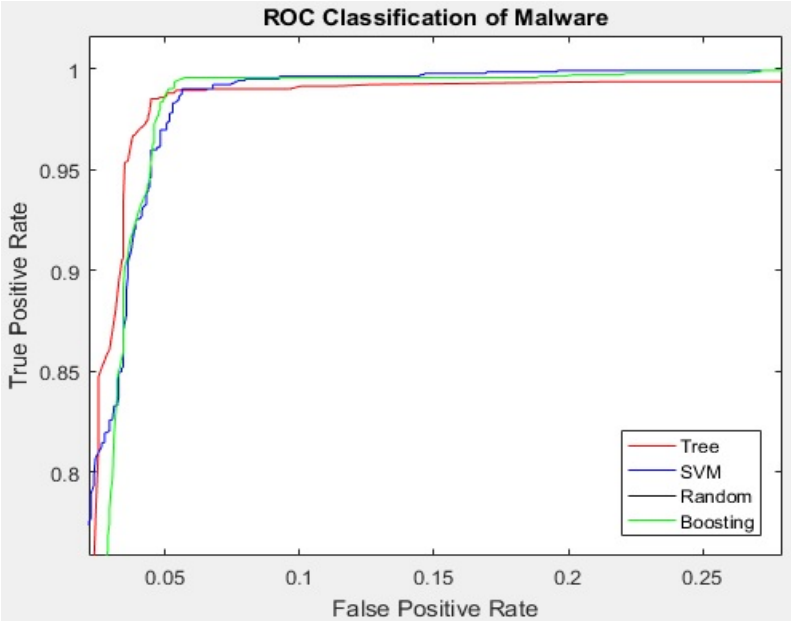


Figure 3.13: ROC Curves ROC curves for malware classification from a Large Dataset

Table 3.11: Result of applying classification techniques on extracted features of a large dataset

Method	Area Under Curve (AUC)
Decision Tree	0.9775
SVM	0.9896
Boosting	0.9969

3.7.1.3 Obfuscated Dataset

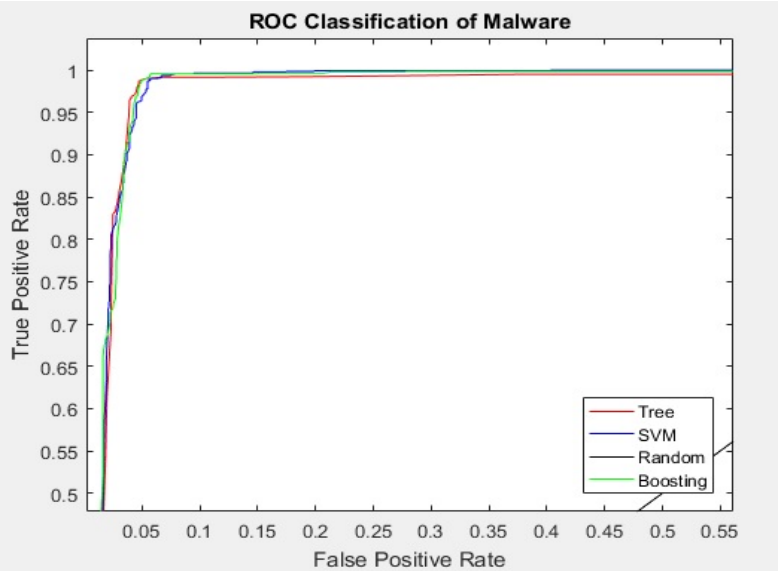


Figure 3.14: ROC curves for malware classification from Obfuscated Dataset

Table 3.12: Result of applying classification techniques on extracted features of a obfuscated dataset

Method	Area Under Curve (AUC)
Decision Tree	0.9740
SVM	0.9823
Boosting	0.9910

3.7.1.4 Real-Time Detection

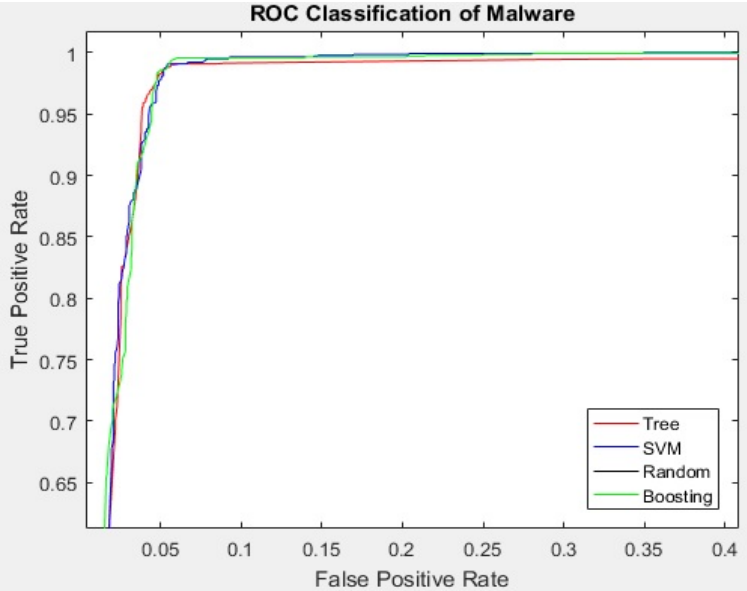


Figure 3.15: ROC curves for malware classification from Real-Time Detection

Table 3.13: Result of applying classification techniques on extracted features from real-time detection

Method	Area Under Curve (AUC)
Decision Tree AUC1	0.9765
SVM AUC	0.9892
Boosting AUC2	0.9963

3.7.2 Discussion

We performed four different experiments to evaluate the performance of our framework. The results presented in the previous section prove that the classification methodology proposed in this chapter satisfy the initial hypothesis of accurate malware detection. The first experiment tested the framework with a smaller dataset to understand the effectiveness of the overall approach.

After extracting the features, all the parameters required to perform the experiments were achieved. We separated a set of 500 files from each category to conduct the initial experiment by starting with ten-fold cross-validation and then applying classification techniques. The ROC curves of this experiment are presented in Figure 3.12 and the areas under curves are presented in Table 3.10. It can be observed from the table that applying boosting on Decision Tree enhance the outcome. Decision tree achieved an AUC of 0.9708 while SVM and boosting achieved 0.9727 and 0.9747 respectively. The experiment performed on the small sample of clean and malicious files gave a satisfactory output considering the learning samples were just 500, which shows that the classification techniques proposed in this research have the potential to perform much better if they are well trained with higher number of samples.

The results achieved from the experiments performed on the smaller dataset were satisfactory, but not better than the similar experiments performed by [85], [93], which achieved a better AUC as compare to our approach. However, the reason our initial experiment lack better AUC was the scarcity in training and testing dataset, which required a higher number of features for training. This was proved in the second experiment performed on a large dataset.

The results of second experiment presented in Figure 3.13 and Table 3.11 show remarkable improvement over the previous experiment and comparing to [93], [85]. SVM achieved a better rate than decision tree but the implementation of boosting on decision tree significantly enhanced the performance by producing a much higher detection rate. As discussed earlier, boosting can enhance the performance of unstable classifiers by decreasing their variance and bias but it can work inversely on stable classifiers [113], which is why we only applied boosting on decision tree and not on SVM.

Our experimental results prove that the proposed methodology can scale in performance on a larger set of files. The training and testing performed on larger dataset was extremely rigorous because of the feature set and techniques used, which also proved that modern obfuscated malware can also be identified with accuracy, as illustrated in Figure 3.14 and Table 3.12, presenting the result of applying techniques on obfuscated dataset. Evaluating classification methodologies based on machine learning against obfuscated and mutated dataset has not been performed by [93], [85] and many studies [109], [91], [90], [119], [120], [102], [121]. This also shows the versatility of our approach and its application on dealing with multiple security threats and can identify not just known but it can also predict unknown threats accurately.

To evaluate the methodology against live threats, we left the entire system running for more than four months. The main objective behind this experiment was to evaluate the framework against live and unknown threats. The proposed methodology showed extremely good results outperforming [93], in their similar experiments of real-time detection of malware. The proposed framework

achieved a highest of 0.9963 AUC from boosting, followed by 0.9892 and 0.9765 from SVM and decision tree respectively, as presented in Figure 3.15 and Table 3.13.

CHAPTER 4. AN ENERGY EFFICIENT HOSTING MODEL FOR THE MALWARE DETECTION FRAMEWORK

4.1 Introduction

In the current era of technology, securing an enterprise network or even an individual computer against different types of advance malware attacks is becoming extremely resource intensive [122]. If an enterprise is willing to spend a lot of resources to acquire tools and licenses for sophisticated network monitoring and protection then surely, they can enhance the level of security of their organizational network but at the same time such tools require a serious amount of resources, such as; dedicated servers, network bandwidth, continuous log management, trained human resources, etc [122], [123]. Similarly, a common user faces the similar type of threats on a smaller scale that are much difficult to identify. Such users can only acquire a license or subscription for an antivirus to protect personal data and other digital valuables against sophisticated attacks [124].

When it comes to higher security performance versus resource efficiency, there is always a tradeoff but the most important aspect of such scenario is how the impact of that tradeoff can be minimized. In the previous chapter, we discussed the malware detection capabilities of antiviruses and their effectiveness in the case of a malware attack. Another thing that is pivotal in this scenario is the impact of antiviruses on the host computers. Lack of malware detection capabilities is not the only problem in antiviruses, while scanning the host computers for malicious software these security software consume a significant

amount of CPU resources of the host machines. Operating systems give priority to antiviruses to perform their tasks and while doing so the OS is left with fewer resources, which makes the system vulnerable against several different threats. This means that not only the conventional security mechanism has a weak scanning outcome, it also implies that while performing a scan the antivirus is making the host system more vulnerable against several different threats.

The framework proposed in the previous chapter proves the initial hypothesis of enhanced accuracy in malware detection but the framework only solves half of the problem. To make the framework a complete solution that can replace conventional detection tools and techniques a hosting model is required that can cater the operational needs of the proposed framework. The primary objective while designing the hosting model was to use an approach which is less resource intensive and highly responsive, especially in a real-time scenario. The hosting model should be able to distribute the resource intensive tasks in an efficient way that would avoid burdening the host computer.

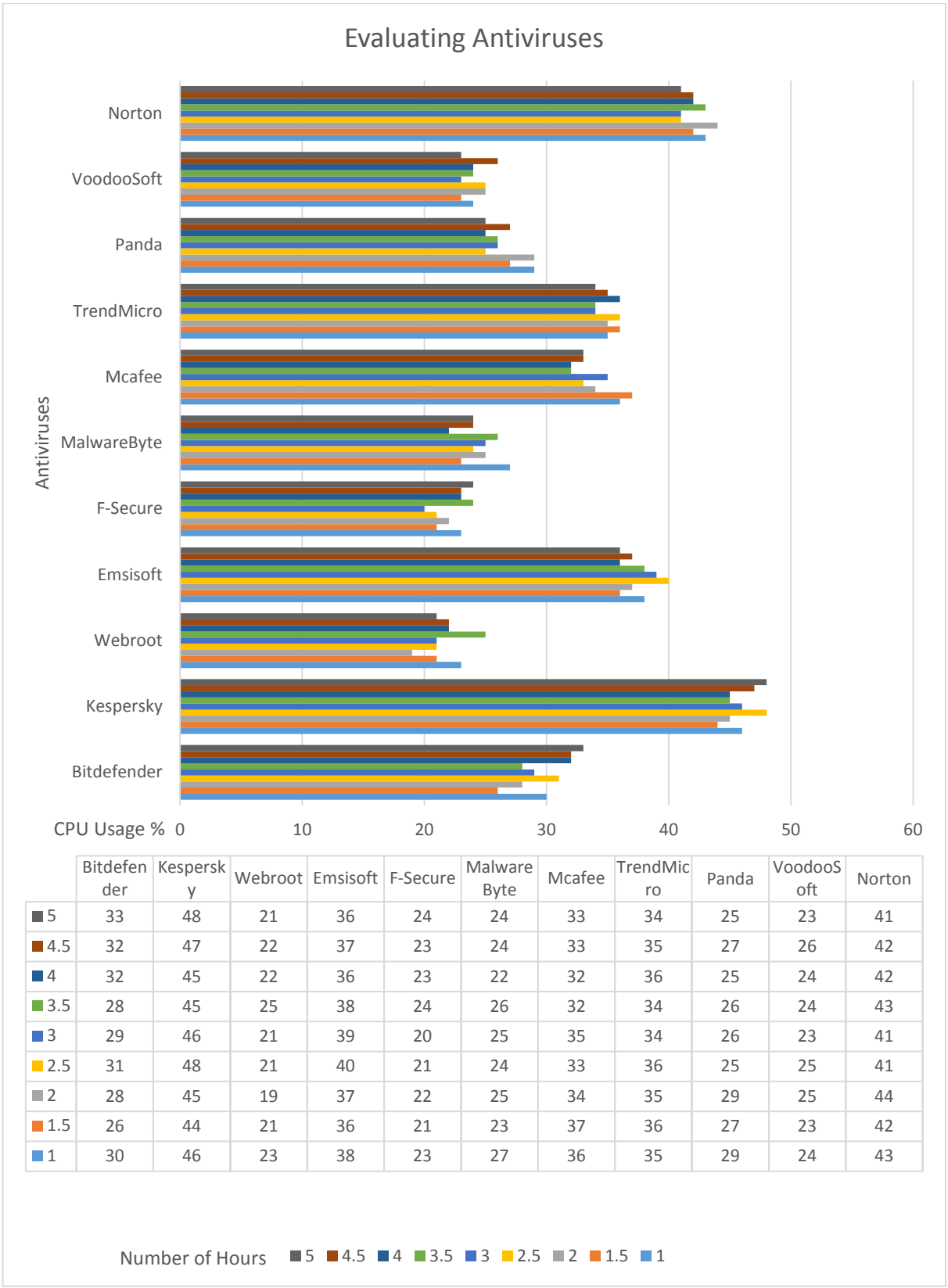
The approach discussed in this chapter is an amalgamation of different cloud-based services. In this approach, we present a comprehensive cloud-based architecture to host the intelligent malware detection framework discussed in the previous chapter along with a lightweight client powered by a rich engine running the malware detection framework. The client agent works as service, which replicates some of the main services of the framework for independent malware detection.

In the following sections of this chapter, we have evaluated conventional antiviruses followed by the description of the building blocks used to design and

implement the hosting model. After building blocks, design and implementation of the hosting model is presented, which is followed by the deployment of the malware detection framework on the hosting model. The next section presents a thorough discussion on the performance evaluation of both aspects of the hosting model, which is then compared by similar evaluation performed on conventional antiviruses.

4.2 Evaluation of Conventional Antiviruses CPU Utilization

It is important to evaluate the performance of current security mechanisms, specifically antiviruses, before presenting a solution. Majority of antiviruses currently leading the industry are host based, which means that they perform their signature generation, comparison, storage, and other resource intensive tasks on the host machine. Even the antiviruses that claim to be cloud-based perform some of the resource intensive tasks on the clients' computers. We selected eleven mostly used antiviruses and evaluated them against the repository of clean and malicious files to identify their CPU utilization. The experiment lasted for five hours, same dataset was used for all antiviruses evaluated, and antiviruses were running on 11 separate computers [18].



	Bitdefender	Kespersky	Webroot	Emsisoft	F-Secure	Malware Byte	Mcafee	TrendMicro	Panda	VoodooSoft	Norton
5	33	48	21	36	24	24	33	34	25	23	41
4.5	32	47	22	37	23	24	33	35	27	26	42
4	32	45	22	36	23	22	32	36	25	24	42
3.5	28	45	25	38	24	26	32	34	26	24	43
3	29	46	21	39	20	25	35	34	26	23	41
2.5	31	48	21	40	21	24	33	36	25	25	41
2	28	45	19	37	22	25	34	35	29	25	44
1.5	26	44	21	36	21	23	37	36	27	23	42
1	30	46	23	38	23	27	36	35	29	24	43

Figure 4.1: Evaluation Graph of 11 Antiviruses

The graph presented in Figure 4.1 [18] illustrates the comparison of antiviruses while running on scan mode continuously for five hours. The graph represents the averages of CPU consumption recorded from all eleven machines every thirty minutes. It is quite clear from the comparison that the antiviruses most commonly used in industry and personal computers have one of the highest CPU consumption in scan modes regardless of their malware detection statistics, which are not impressive as well. Antiviruses such as McAfee, Norton, Kaspersky, and Bitdefender have one of the highest CPU utilization average, which means that while the scan is running the host computer is only left with half the resources it originally has. The reason why the evaluation experiments lasted for five hours was because we wanted to check whether the CPU utilization decreases after the scan is continuously running but there was no noticeable change recorded. The antiviruses that claim to rely on their cloud-based engines, such as; Panda and Webroot, utilized more than 20% of the CPU in the scan mode.

The evaluation of antiviruses based on the amount of CPU resources they utilize reveals that low malware detection rate of antiviruses is not the only issue. The framework proposed and discussed in the previous chapter requires a hosting model that can overcome the issue of significantly high CPU utilization while satisfying all the operational requirements of the framework including the real-time operational requirements of the framework.

4.3 Building Blocks Overview

As discussed in the earlier section, the approach proposed later in this chapter is an amalgamation of different services of cloud infrastructure. We have used different cloud services offered by Amazon to design a thorough and scalable architecture to host the intelligent malware detection framework. Before discussing the design and implementation details of the said architecture, it is important to discuss the building blocks used in the proposed architecture. This section presents an overview of the building blocks used in this architecture.

4.3.1 Amazon Web Services

In this section, we are going to discuss the cloud-based web services we have used from Amazon in the proposed architecture. We have used multiple instances of three different cloud-based web services and connected them to design a cloud-based scalable network capable of hosting and executing resource intensive tasks. Following is the description of services we have used:

4.3.1.1 SQS (Simple Queuing Service)

Amazon SQS is a purpose-built service for message queues, fully managed by Amazon. It works flawlessly between different distributed applications and micro servers. Amazon's SQS has elastic capabilities that allow the queues to dynamically scale up or down based on system's overall requirements [125]. SQS allows client application or software components to send, receive, and store messages between multiple components without losing any messages or needing other connected services to be consistently available throughout [126]. This queuing service transports messages with embedded jobs, allowing

software components to trigger different functionalities through messages, making a distributed system work as a single, well synchronized software component. One of the main concerns in queue-based task processing is the execution of duplicated messages [127]. SQS FIFO queues are specifically designed and configured to ensure at-most-once message processing, with very limited throughput and in the same order as delivered. We have used SQS to implement the queuing mechanism in the architecture, explained in the later section.

4.3.1.2 EC2 (Elastic Compute Cloud)

Amazon EC2 is a cloud-based service that offers dynamically resizable computing space in the cloud. EC2 is a platform providing virtual dedicated-server hosting that runs instances of virtual machines also known as AMIs (Amazon Machine Images). Amazon offers a rich group of virtual machines preconfigured for several different tasks, such as; Ubuntu desktop, Ubuntu server, Windows server, etc. [127], [125]. Apart from the virtual machines, the primary function of EC2 is the computing platform for the VMs. Amazon offers hosting services with different sizes of computing and storage. The size of storage and computing can be selected based on the requirements of the hosted applications. The most beneficial aspect of EC2 is its elasticity, which allows the specific compute plan to dynamically scale if required allowing the hosted application to expand in size, network bandwidth, and computing power without any hindrance [128]. It uses pay-as-you-go approach and cost the user only for the time the service was up and running. Apart from the elasticity benefit, EC2 is

extremely quick to setup and launch, it also offers facilities to implement fault tolerant mechanism and applications that are resilient to failure [129], [127].

4.3.1.3 EFS (Elastic File System)

As the name suggests, EFS is a file system service with elastic features, which offers simple and scalable file storage service. EFS is specifically designed to be integrated with EC2 instances to work as a single cloud application [130]. EFS is one of the most convenient file system in the cloud and can easily be mounted with multiple cloud-based applications. If there is an application hosted on EC2 and EFS is mounted on that instance, it'll offer a standard interface for file system and access semantics for file system. This allows seamless integration of this file system with existing tools and applications. Moreover, a single EFS can be linked with multiple applications on EC2 instances or single application on multiple EC2 instances, allowing a common data source to cater the needs of distributed applications [131]. The elastic file system can also be linked with the local datacenter that are not linked with the cloud, which can also be used to conveniently migrate large data sets to the cloud. The versatility of this service allows it to be used for a broader domain range, such as; web applications, media processing, enterprise applications and services, big data and analytics, data storage, etc. [132].

4.4 The Hosting Model

The idea of developing a framework that implements multiple techniques to detect malware and integrates their result to enhance the accuracy in detection and consumes significantly less CPU and network resources while doing so, is incomplete without a hosting model. The concept of energy efficiency is

dependent on how the framework is hosted and the level of its scalability. This section presents a cloud-based hosting model, which is not only able to host the framework discussed in the previous chapter, it possesses a dynamic behavior allowing it to scale itself in runtime without affecting the performance of hosted framework. To design and configure the hosting model proposed in this section, we have used Amazon's cloud infrastructure along with its web services. Before discussing the model, it is important to understand hosting requirements of the malware detection framework.

The malware detection framework is comprised of three main modules that are further divided into submodules and have their separate requirements when it comes to hosting them. Following subsections define the individual requirements of each module:

4.4.1 Repository

The first module in the framework is the repository, the repository is further divided into three submodules; clean files repository, malicious files repository, and the analysis repository. As the name suggest, all three of these submodules store different type of files that are later used by other modules. As mentioned in the previous chapter, the clean and malicious files repositories contain clean and malicious files respectively, which are later analyzed. After regular intervals, these sub-repositories are populated by a new batch of hundreds of thousands of new clean and malicious files stored in their respective repositories. Moreover, the analysis repository contains the analysis reports of every clean and malicious file separately, with each batch of clean and malicious files stored in the repository the same amount of analysis reports is stored in the analysis

repository, once the analysis is performed. Therefore, to host such a mechanism, a large and secure filesystem is required, however, these repositories not always require a huge storage space. The storage is only required when the files are present in the repository and once the analysis is performed on all stored files, they are not required to be stored. Consequently, having a dedicated large storage is not feasible for such mechanism and requires something that is available on-demand with high reliability.

4.4.2 Analysis Module

Analysis module is the first line of defense while identifying a malware and runs a customized tool, which retrieve files from repositories, prepares them to be analyzed, extract rich set of features by performing analysis, and removes any obfuscation present in the extracted features. Analysis module extract features from both clean and malicious files simultaneously, therefore, two instances of this module need to be operational concurrently. As discussed in the previous chapter, the main detection process starts from this module, which means the next module is dependent on the outcome of analysis module. Moreover, the customized analysis tool running in this module also incorporates external APIs to get endorsement on some of the results from external sources, which means that there is a requirement of internet connectivity. Based on these requirements, the analysis module requires dual instances of a similar server along with reliable internet connection. It also requires flexible but reliable system resources, which means that the CPU power and memory should be readily available for the module but only when required. These resources are only required when the analysis module is running and it only runs when there is a new batch of clean

and malicious files in the repository or a single file identification is required. Therefore, when the analysis module is running the requirements should be fulfilled but when it's not running the hosting platform should be intelligent enough to manage the resources.

4.4.3 The Classification Module

The classification module is the final phase of the framework and it's a combination of different machine learning algorithms that are applied on the analysis reports produced by the analysis module to enhance the level of accuracy while identifying clean and malicious files. This module is linked with the analysis module through the analysis repository, which is the submodule of repository module. Moreover, the machine learning algorithms in the classification module run simultaneously to produce the accurate malware and benign file identification, which means that dedicated resources are required for this module. As discussed in the previous chapter, this module as two aspects while operational; first it uses the reports generated from the analysis of large number of clean and malicious files to train and test the accuracy of algorithms, and secondly it identifies the individual files analyzed separately. This whole process with both the aspects is recursive and continuously repeats itself in cycles. Therefore, like the analysis module, the classification module also requires a hosting platform that is dynamically scalable, cost effective, fault tolerant, easily coupled with the other modules.

The hosting requirements of three main modules of the intelligent malware detection framework discussed above clearly highlight the primary needs, which will make the framework seamlessly coherent and efficient in terms of

performance. There are certain unique and some common requirements for each of these modules, the most important things in these set of requirements is that all these modules are required to work together as a single unit, two out of the three main modules require internet connectivity, and the scalability of resources. These main requirements, along with the other requirements discussed above are specifically considered when designing the architecture of the hosting model.

4.4.4 System Architecture

This section presents the architecture of the hosting mechanism, specifically design for the intelligent malware detection framework. It caters for the needs of every individual module of the framework, identified and discussed in the previous chapter and provide the most relevant and reliable mechanism. We have used AWS (Amazon Web Services), the cloud platform of Amazon to design the hosting mechanism. As discussed in the previous section, the framework might require flexible storage and computing resources that can dynamically scale up and down along with internet connectivity to connect with external APIs used in the framework. Amazon provides cloud services that are extremely relevant in this scenario and are convenient when it comes to scalability. Earlier in this chapter, we discussed the building blocks that we have used to design the hosting model, we now discuss how we have used those building blocks.

The three building blocks discussed earlier are combined to build this hosting model. There are two phases in proposing this architecture; we first discuss the design and implementation of this model, and in the second phase we deployed the framework on the model designed. The model is based on a client/server

architecture and in this scenario server is the primary component because only this component hosts the framework.

4.4.4.1 Server

Figure 4.2 presents the high level of architecture of the hosting model, which illustrate the components of Amazon's cloud computing platform that are linked together to host the complete framework. Each unique component of the cloud platform is expected to host one or more components of the malware detection framework. The three components of AWS; SQS, EC2, EFS host queues, detection engine, and repositories respectively. The framework presented in the previous chapter in Figure 3.5, supposedly doesn't require any queues but to implement the framework to be used and evaluated in real-time, it is required that the framework should be able to receive requests and send responds to either local or remote clients. To make the hosting model efficient in terms of performance, we introduced queues to manage the large number of requests coming from multiple client, allowing the hosting model to be scalable dynamically. There are multiple queues presented in Figure 4.2 [106], which illustrate the dynamic behavior of the hosting model. The first three queues from Q1 to Qn are the request queues and R is the response Q, which makes the overall queues four. However, if there is only one remote client connected to this cloud-based framework then there are only two queues one for request and one for response. The architecture is designed to be dynamically scalable and can cater many clients without manually changing anything in the hosting model. Therefore, if the number of clients trying to couple with the framework hosted on

the cloud hosting model, the number of queues will dynamically increase based on the number of requests sent by the clients.

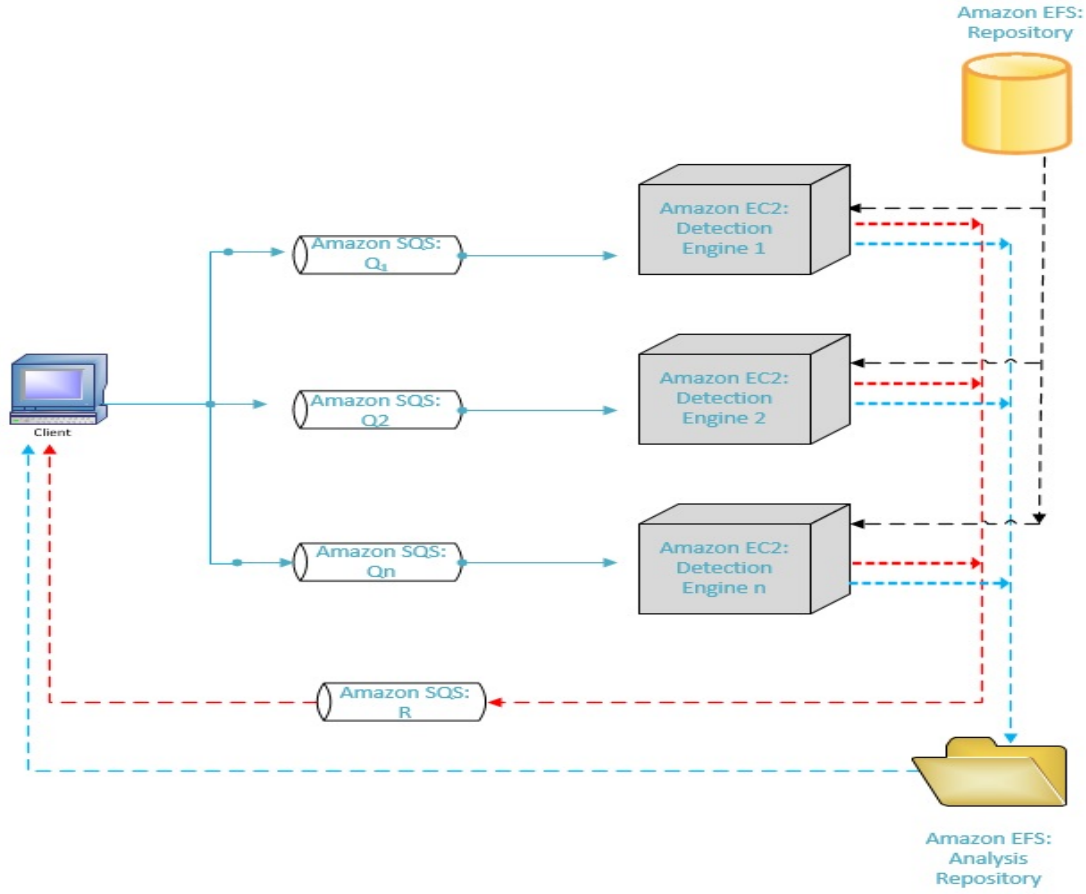


Figure 4.2: High Level Architecture

The analysis and classification modules are hosted on Amazon's EC2, which is mentioned as the detection engine in Figure 4.2. The high-level architecture in Figure 4.2, also presents multiple detection engines. Similar to the queues, for a single client, one detection engine is enough and even if the single client is continuously sending a large amount of requests that might exceed the bandwidth quota initially allotted to the SQS and EC2, the initial bandwidth quota will dynamically increase without any latency in the service and will subsequently be reduced to the initial level when the number of requests will reduce. Similarly,

if the number of clients increase significantly, the number of detection engines will also increase dynamically.

There are three repositories in the framework; clean files repository, malicious files repository, and analysis repository. All these repositories are held separately in a single repository hosted on Amazon's EFS. The elastic file system, as the name suggests, can expand and contract as required based on the number of files stored. The requirements of these repositories vary based on the number of files stored and don't always require huge storage capacity on the cloud-based storage that why EFS is a perfect choice. Unlike, detection engine and queues, repositories don't require multiple instances if number of clients, or requests increase even significantly. Repositories are used just to store and retrieve clean/malicious files and their analysis reports and don't require computation.

There are two different operational aspects of the intelligent malware detection framework; in the first one, it uses a large sum clean and malicious files to generate analysis reports and that are used to train and test the machine learning algorithms, in the second one, a single file is sent to the framework to get identified as clean or malicious. The hosting model accommodates both these aspects by using a combination of different building blocks.

In the first aspect of the framework, as depicted in Figure 4.3, the analysis module retrieves the clean and malicious files from the respective repositories and generate analysis reports for every file analyzed. In this phase, there is a requirement for the analysis module to be running on two active instances simultaneously, allowing the module to process clean and malicious files in a segregated environment with a much rapid pace. In this aspect, there are

hundreds of thousands of files in each repository and to make the process efficient, two active instances of this module are required. As discussed earlier, Amazon's EC2 is used to host the detection engine, which contain both analysis and classification modules. However, the lower level architecture of the first aspect in Figure 4.3 illustrate both the module separately to elaborate their individual functionalities.

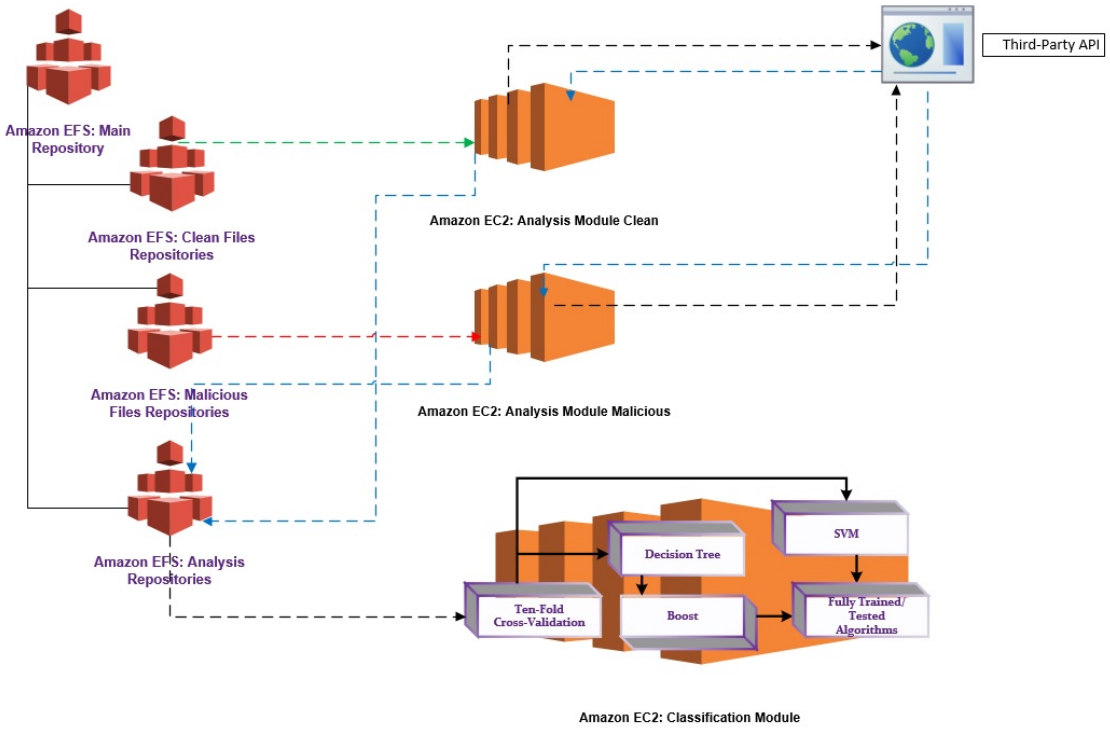


Figure 4.3: Low Level Architecture of First Aspect

EC2-based analysis modules for both clean and malicious files retrieve the files from the respective repositories simultaneously to perform the thorough analysis. Each of these instances of the analysis modules also seek help from a third-party private API of Virustotal.com to get endorsement on some of its results. Once the analysis is complete, the analysis report generated by the clean or malicious analysis module is stored in the respective analysis repository. After all files in

both repositories are analyzed, the classification module is triggered and uses the extracted features in the analysis reports to train and test the machine learning algorithms. Once the cycle of training and testing is complete, the classification module becomes fully trained to be used in real-time malware detection, details of which are discussed in the previous chapter.

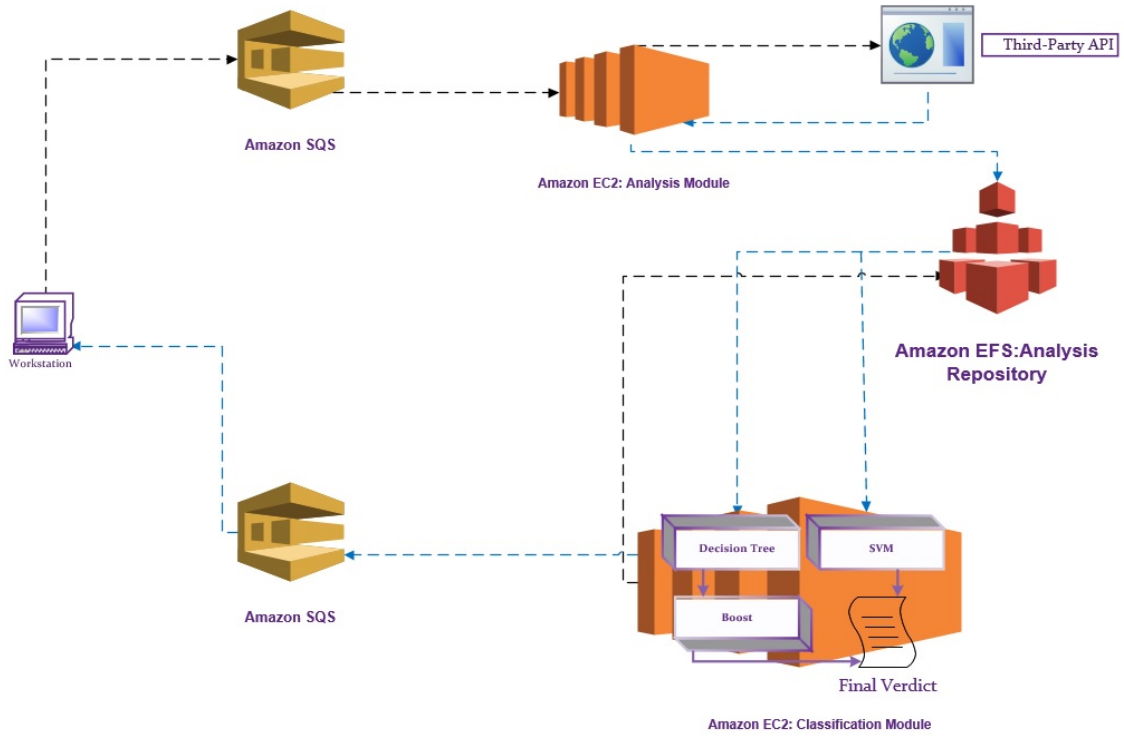


Figure 4.4: Low Level Architecture of Second Aspect

In the second aspect of the framework, the main objective is to identify the malicious software in the client. The lightweight agent running on the client's computer sends the suspicious file to the cloud-based server running the framework. The request is received by the queue system hosted on SQS, which organizes the messages from the clients on FIFO bases and send it to the available detection engine. If there is only one detection engine running, then the queuing system doesn't need to prioritize. Once the message, which contains the

suspicious file, is received by the detection engine. The file is first treated by the analysis module as shown in Figure 4.4 where a thorough analysis is performed and assisted by the external API and a preliminary decision is made and the analysis report is submitted to the analysis repository, as discussed in the previous chapter. If required, the control is then transferred to the classification module that uses the machine learning algorithms fully trained in the first aspect and classify the file as malicious or benign. Once the decision is made the response is sent to the response queue, which sends it to the client. The request message sent from the client contains unique client ID along with a unique message ID, which is used by the response queues to identify the corresponding client. These IDs are also used when the analysis report is submitted and retrieved to and from the analysis repository.

Both these aspects discussed above are part of the same architectural setting, the difference illustrated in Figure 4.3 and Figure 4.4 shows how the same cloud-based components are used differently to fulfil the tasks required by two aspects of the intelligent malware detection framework.

4.4.4.2 Client

Another module in this architecture is the client module that triggers the second aspect of the framework. The client module in this architecture is a simple and lightweight agent that works as a service in the client's system. Unlike conventional antiviruses, this lightweight agent is only comprised of four main components; a browser extension, process monitor, local cache, and file scanner. All four of these components work as a coherent unit and identify suspicious files with the help of local cache, requiring quite small amount of CPU resources of

the host system. The three components of this module; browser extension, process monitor, and file scanner search for files. Local cache is populated with the brief information of files, such as; signatures and basic heuristics, which are already analyzed by the server in the first aspect. This brief information of clean and malicious files helps the lightweight to decide about file's legitimacy locally, without sending the file to the server. Browser extension monitors any file user or software is trying to download and checks in the local cache if it is a malicious or benign file. If the file is present in the cache as malicious it is straightaway blocked, if the file is not present in the local cache, it is sent to the server module for further analysis and classification. Process monitor checks all the processes currently active and match their IDs against the local cache of malicious files and simultaneously sends it to the server for further verification. File scanner scan the existing files and follow the same procedure of local identification and then server identification. The main reason the client module is lightweight and doesn't require a lot of host machine CPU and storage resources is because it doesn't decompress or emulate the files locally and the signatures and heuristics are not stored locally in a descriptive format.

4.5 Framework Deployment

In the previous section, we discussed the proposed architecture to host the intelligent malware detection framework. In this section, we deploy the complete framework on the proposed cloud architecture as discussed above and evaluate

the overall performance.

Figure 4.5: Amazon Linux AMI

We chose Amazon Linux OS for the EC2 instance to host the analysis module and classification module. Both clean and malicious analysis module along with the classification module were hosted on different EC2 instances running the same AMI as shown in the Figure 4.5.

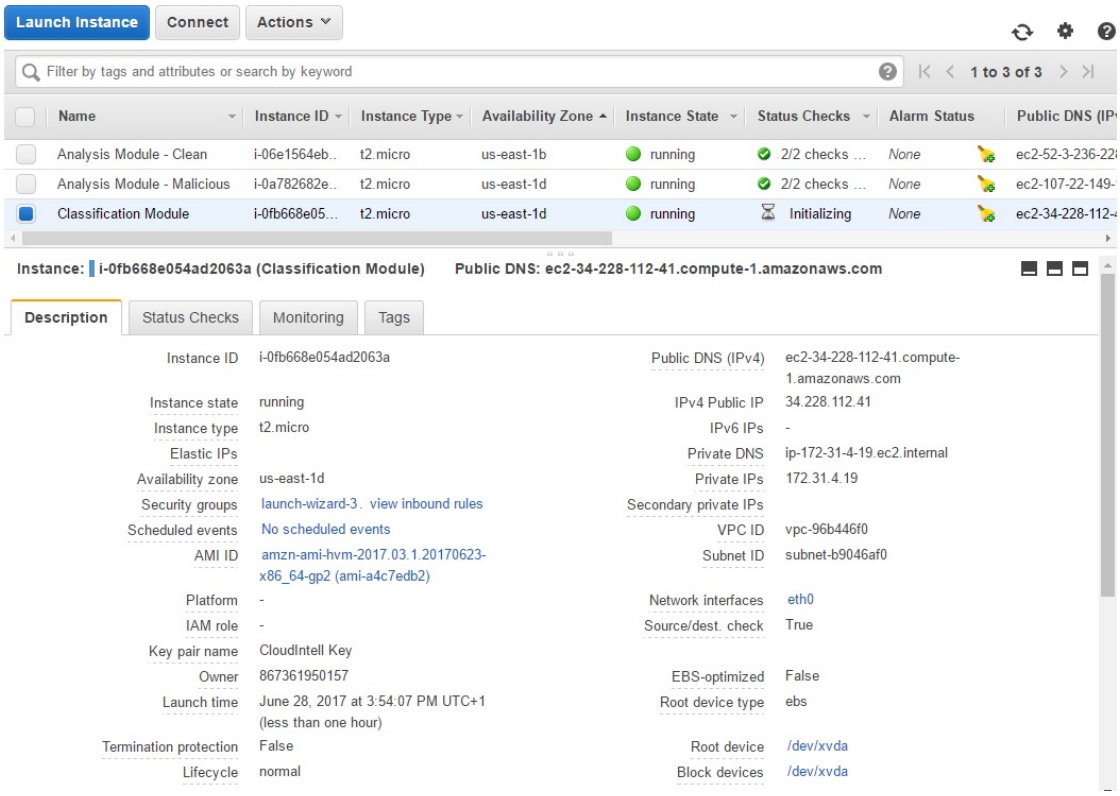


Figure 4.6: Analysis and Classification Modules

As shown in Figure 4.6, both analysis modules and the classification module are running on the EC2 AMIs. We launched the EFS and named it main repository,

which is further divided into two three sub-repositories; clean file repository, malicious file repository, and analysis repository. We then migrated our collection of clean and malicious files to the EFS-based repositories with an accumulated size of 623GB, as shown in Figure 4.7. In the next step, we set up the request and response FIFO queues using SQS FIFO, both these queues are configured to dynamically replicate if the overall requirement exceeds. The replication of the queues is triggered by the number of requests increasing the threshold level. Figure 4.8 depicts the deployment of the queues on AWS console.

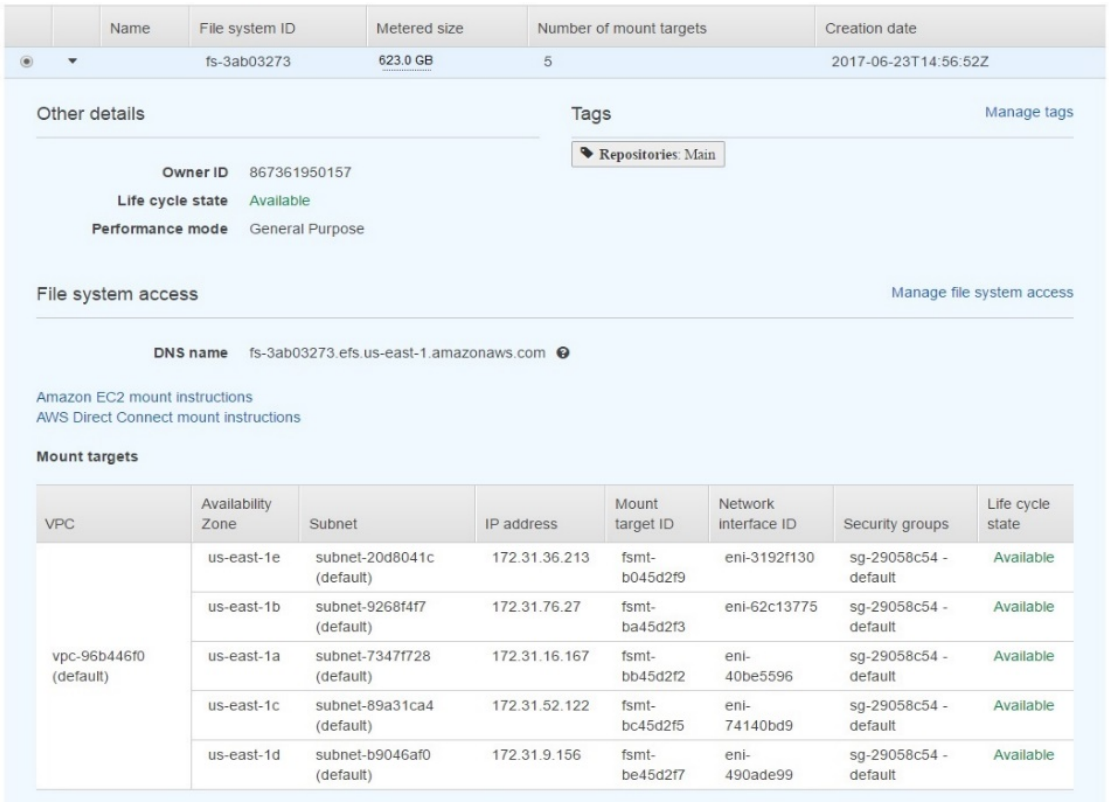


Figure 4.7: EFS Repository

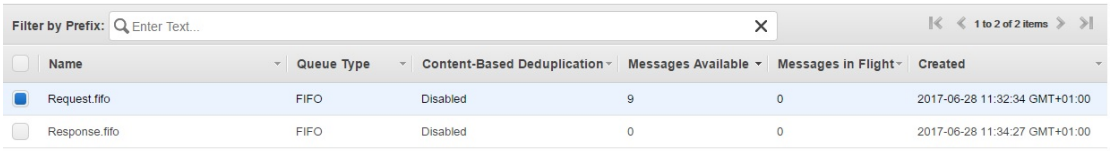


Figure 4.8: Request Response Queues

Once all the components of the framework were deployed, we then mounted the repositories with EC2 instances hosting the analysis and classification modules. The repositories hosted on EFS integrate and operate with EC2 seamlessly and work as a component of EC2, which means that after the mounting process is completed there is no extra command or process needed to store or retrieve a file to or from the repository. The elastic file system allows thousands of EC2 instances to be connected to a single EFS concurrently with file locking mechanism. Therefore, if the EC2 instances are dynamically increasing, they will be connected to the EFS based repositories, even if the number of EC2 instances is in thousands.

```
<queue quid="fdssgrell921hhgbzmurtaaqp">
<job id="1" task="analysis">

    <file type="source" bucket="rep_clean">AccessDatabaseEngine_x64.exe</file>
    <file type="destination" bucket="rep_analysis">AccessDatabaseEngine_x64.JSON</file>

    <cli>analyze -exe -JSON _SOURCE_ _DESTINATION_</cli>
    <retry_remain>3</retry_remain>

        <job id="2" task="classificatoin">

            <file type="source" bucket="rep_analysis">AccessDatabaseEngine_x64.JSON</file>
            <file type="destination" queue="quid">message.xml</file>

            <cli>classif -JSON -XML _SOURCE_ _DESTINATION_</cli>


</job>
</queue>
```

Figure 4.9: XML Request Message

After mounting the EFS with the EC2, we connected the request and response queues with the analysis and classification modules. The architecture uses FIFO

queues for request and response, which plays a vital role to prioritize the queries sent by the clients and also guarantees that the tasks of identifying a clean or malicious file sent by the client is only processed exactly-once. As one of the main objectives of this hosting architecture is energy efficiency, it is pivotal to avoid processing duplicated messages. SQS FIFO queues have a built-in feature, which ensures that all messages are delivered to the destination at least once but once delivered, the duplicates of every message are removed. This completes the server side connections complete and fully mounted, we now need to connect the client(s) with the cloud-based server.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- This document is a sample demonstrating result queue messages
content. -->
<!-- job_result is the root node for the document -->
<!-- The processed date time uses the date format yyyy-MM-dd HH:mm:ss. -->
<job_result processed_date_time="2017-06-28 01:01:01Z">
<!-- queue_stage_id represents at which stage the job is. -->
<queue_stage_id>1</queue_stage_id>
<!-- result_status indicates the result when an EC2
analysis & classification module attempted to process the queue task.
Values are: FAIL | ERROR | SUCCESS -->
<result_status>SUCCESS</result_status>
<!-- This is the unique hash identifier for this job. -->
<quid>fdssgrel1921hhgbzmurtaaqp</quid>
<!-- An important tag to hold verdict of the analysis.
 Will be either CLEAN FILE or MALICIOUS FILE.
 The body is wrapped in a CDATA section -->
<message>CLEAN FILE</message>
</job_result>
```

Figure 4.10: XML Response Message

Although the whole architecture is based on client-server approach, the main idea is to make both the components coupled together in a way that all the distributed modules work as a single component. We designed lightweight XML messages which are sent by the client through the queues and because the queues are connected to the analysis and classification modules, they automatically add a header to each message defining which EC2 instance is going to receive the

message, based on the availability of each running instance. The message presented in Figure 4.9 is the request message sent by the queue to the analysis engine. The message is in a hierarchical form showing nested child XML elements, which contain another job. The jobs are processed as per the hierarchy and the task type, which means that the first task in Figure 4.9 is for the analysis module which takes the file AccessDatabaseEngine_x64.exe and analyzes it. Once the analysis is performed the analysis report in JSON format is stored in the analysis repository. Consequently, the same message is then forwarded to the classification module because the job ID 2 has the classification task attached triggering the control transfer from analysis to the classification module. When the message is received by the classification module, the parent elements of the XML message are ignored and only the child elements are processed. The classification modules retrieve the analysis report of AccessDatabaseEngine_x64.exe from the analysis repository, which is stored with the name AccessDatabaseEngine_x64.JSON and runs it against the fully trained classification algorithms. When the classification module has completed the task, it sends the verdict in an XML based response message presented in Figure 4.10, it contains three main things from sever; the status of the job, ID of the message, and the verdict of both the modules. The status of the job shows whether it was a success or a failure, the message ID is for the queue to identify the sender and the order in which the message was processed, and the verdict is either CLEAN or MALICIOUS based on the classification. Once the message is received by the sending client, it looks for the message tag which contains the

verdict and if the verdict is CLEAN the file can be used by the user and if it's malicious, the client module blocks and removes it from the system.

4.6 Performance Evaluation

The hosting model proposed in this chapter was successfully able to host the intelligent malware detection framework and initial messages were sent and received. This section evaluates the operational performance of the framework while hosted on the cloud-based hosting model. As discussed earlier in this chapter, there are two aspects of the hosting model, therefore, this section will separately evaluate the first and the second aspect based on their operations.

4.6.1 The First Aspect

The first aspect of this hosting model is designed to support the training and testing of the classification module which relies on the analysis reports generated by the analysis module. Each cycle of this aspect retrieves the clean and malicious files from the respective repositories and reports are stored in the

analysis repositories, which are used by the classification module for training and testing.

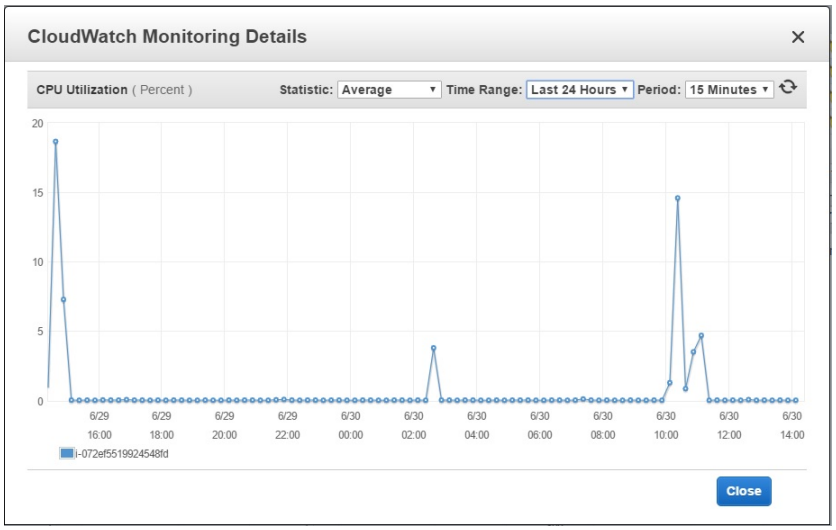


Figure 4.11: Analysis Module CPU Utilization - Clean

The start of this aspect of the hosting model revolves around analysis module along with clean and malicious files repositories. In this evaluation, we performed analysis of 100000 malicious and 75000 benign windows executables stored in their respective repositories hosted on EFS. There were two cloned EC2 instances of analysis module running simultaneously, one for each class of files. Analysis is thorough process involving feedback of external APIs and some of the files that are bigger in size take more time to get analyzed. Figure 4.11 and Figure 4.12 present the CPU utilization graph generated by Amazon EC2 monitoring tool, which represents the performance of clean and malicious instances of analysis module. Cycles for both these modules lasted for about 24 hours and as presented in both these figures, the highest CPU consumption during the clean and malicious file analysis was 19% and 23% respectively and that only for a very short period. This clearly shows that the analysis module

doesn't require a lot of CPU resources while performing the most resource intensive task. The multiple instances used for clean and malicious file analysis manage the load, save time, and consume less CPU power simultaneously. Few spikes shown in both Figure 4.11 and Figure 4.12 are caused by some exceptionally large files present in both the repositories, which is in fact not a usual occurrence in such scenarios. This shows that the hosting model and the hosted framework have the potential to manage resource intensive tasks.

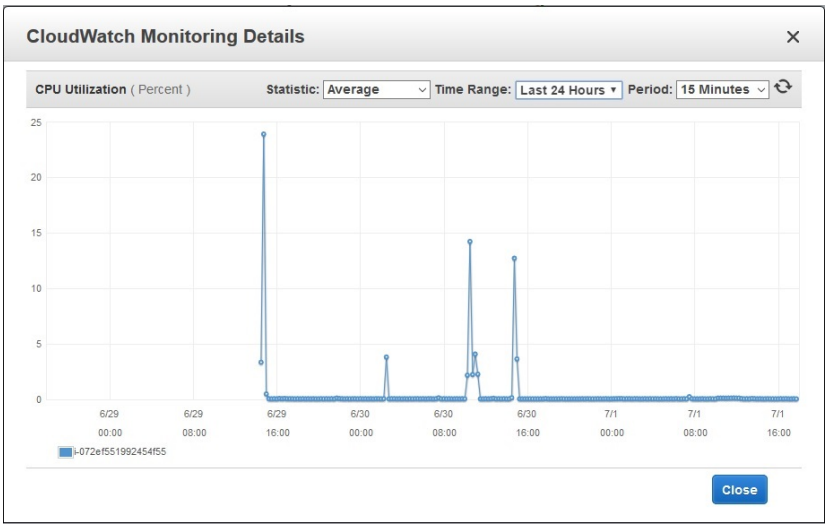


Figure 4.12: Analysis Module CPU Utilization – Malicious

In the first aspect, after the analysis is performed, the analysis repository is populated with a large sum of analysis reports that used by the classification module to train and test the classification algorithms. Figure 4.13 presents the CPU utilization of the classification module during the process of training and testing, it can be seen that the graph is not consistent and there are a few spikes reaching up to 43.7%. It can be seen in Figure 4.13, that at the start of this process the CPU consumption reaches its highest point and after some time it drops. The highest point of CPU consumption show that the training and testing

process is at its peak and the sudden drop show the interruption, which is caused by the large amount of data extracted from a single file during analysis. The classification module further cleans the files and prepare them for training and testing.

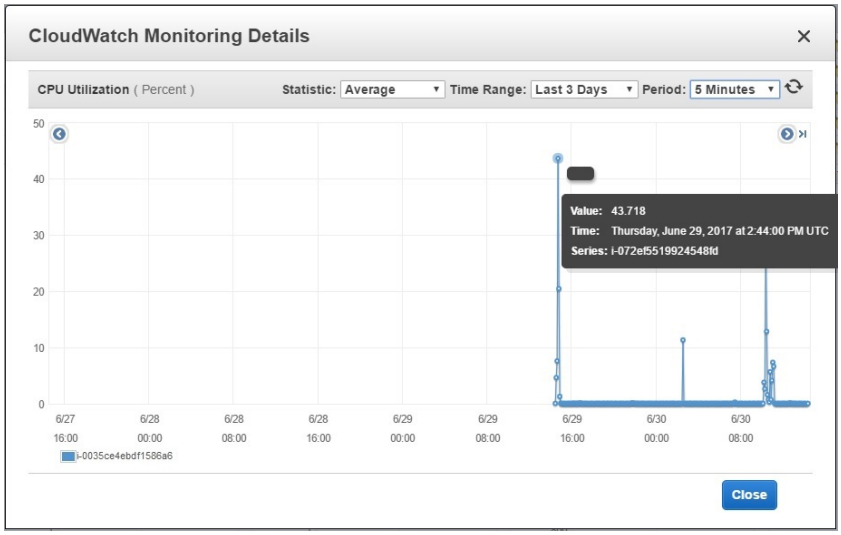


Figure 4.13: Classification Module CPU Utilization

After individually evaluating both the modules against the operations of first aspect, Figure 4.14 presents the comparison of both these modules in real-time. The orange line in the graph is the analysis module and the blue line is the classification module. The highest spike in the starting point of the orange line suggests that the analysis has started randomly with the heaviest file utilizing 45% of the CPU and then it dropped to the regular files. The occasional spike in this analysis module graph suggest the analysis of heavier files. The blue line starts after the analysis module has finished analyzing all the files in the repository and takes around 50% of the CPU power to start the training and testing process and immediately comes down to 25% right before the classification module is fully trained to identify clean and malicious files.

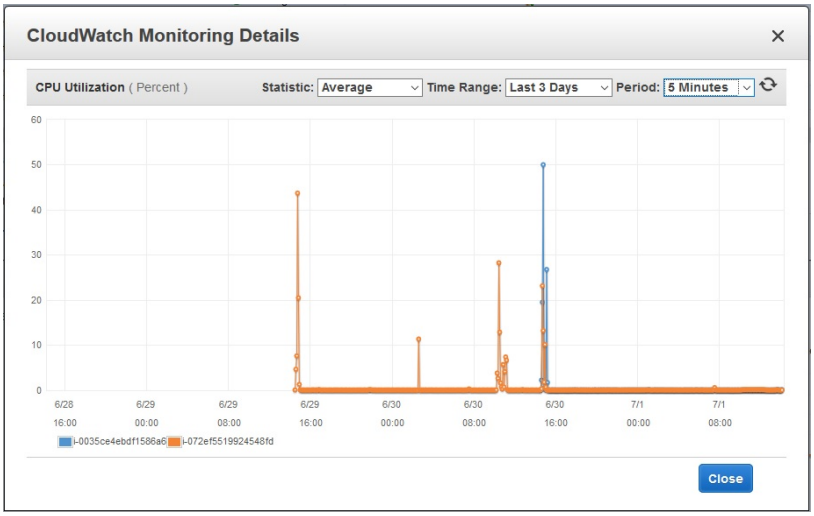


Figure 4.14: Comparing Analysis & Classification Module CPU Utilization in First Aspect

The first aspect of the hosting model is more resource intensive as compare to the second aspect, which deals with the real-time detection. Even though the first aspect hypothetically requires more CPU resources, it reached above 40% very few time, based on the type of file it was analyzing. This shows that the first aspect of this model, which is required to perform operations in real-time environments is energy efficient. Additionally, the on-demand elasticity of EC2 provides an ideal hosting mechanism for this model.

4.6.2 The Second Aspect

The second aspect primarily performs real-time malware detection based on the accomplished tasks of the first aspect. The main differences in the evaluation environment between the two aspects are; presence of request and response queues, requests from clients, single instance of analysis module, and classification module with an additional task of providing verdict to the client through the response queue.

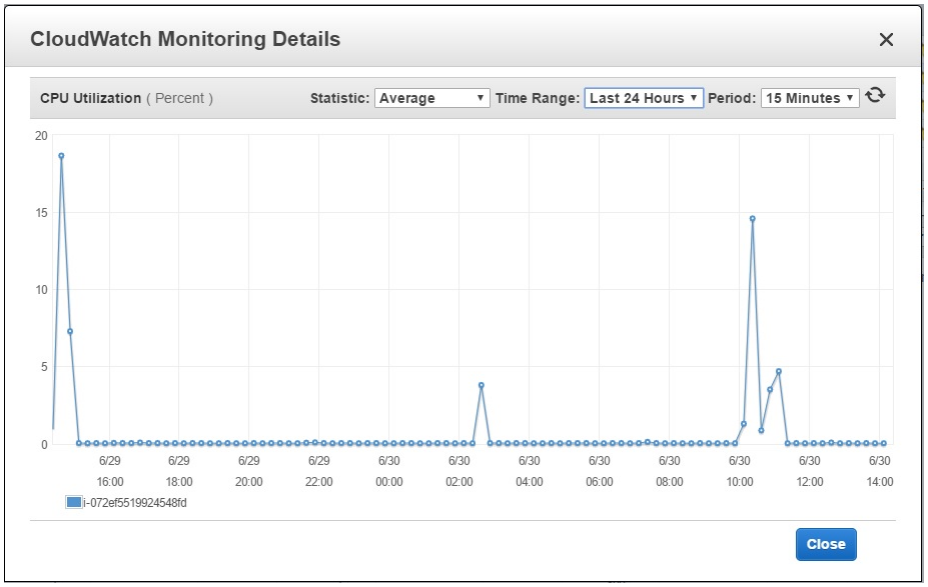


Figure 4.15: Analysis Module CPU Utilization

To evaluate the performance of the framework and the hosting model for second aspect, a large amount of request messages was sent simultaneously to the cloud-based framework. The request messages sent from multiple clients contained the suspicious files required to be identified as clean or malicious by the framework. We used 35 virtual machines running on physical machines, as clients to continuously and simultaneously send request messages. These virtual machines stored a combination of 93200 clean and malicious files. The main idea behind sending these simultaneous request messages was not to evaluate the malware identification accuracy of the framework, it was to evaluate how the framework and the hosting model perform in a real-time environment while fully trained and tested in the first aspect. Figure 4.15 presents the CPU utilization graph of the analysis module while it processes the request messages. As illustrated in Figure 4.15, the analysis module initially consumes around 19% of the CPU but after a while drops to under 1% and then in the middle and at the end it hikes up to 4% and 15% respectively. The reason it starts with a

comparatively higher CPU utilization is that the files under analysis are newer and don't have any previous analysis history stored in the system, therefore, the analysis module is thoroughly analyzing the files. However, as discussed earlier in this thesis, majority of the modern malware are variants of old malware and that's why similar type of files don't require a lot of processing while analyzing such files, it consumes much lesser CPU resources. This is also the reason why there are hikes in the middle and at the end of the graph.

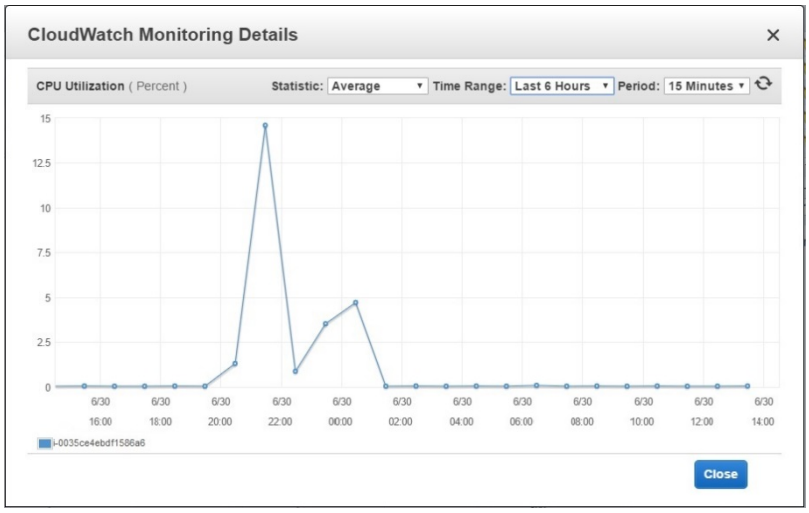


Figure 4.16: Classification Module CPU Utilization

Unlike first aspect, in the second aspect the classification module doesn't have to wait for the analysis module to finish analyzing a large sum of files. The whole framework in the second aspect works on the basis on individual requests, which makes this framework even more energy efficient. Figure 4.16 presents the CPU utilization percentage of the classification module while operating in the second aspect. As illustrated in Figure 4.16, the classification module started with a relatively higher CPU consumption but after a while drops to 1%. Over the period of 6 hours it didn't go above 5%, which happened because the classification

module is already fully trained and the tasks needed to be accomplished by the classification module in second aspect are not resource intensive.

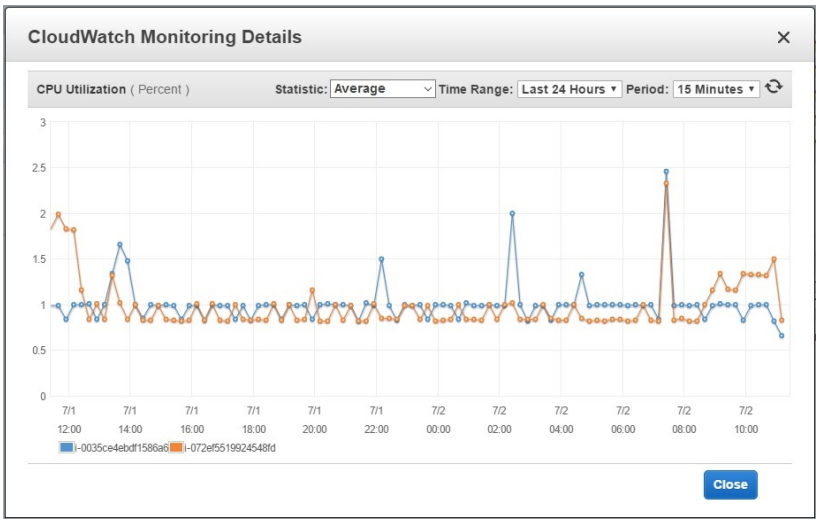


Figure 4.17: Analysis & Classification Module CPU Utilization Second Aspect

For a thorough performance evaluation of the framework and the hosting model in the second aspect we used the same approach of multiple clients with continuous request but this time we left the system running for more than 24 hours. The objectives behind letting the system run for a longer span were to identify overall CPU utilization from both the modules, how synchronously both modules work, and how the system behaves while operating completely unsupervised. The graph presented in Figure 4.17 illustrate the CPU utilization comparison of both the modules, the blue and orange line represent classification and analysis module respectively. It can be observed from the graph that the analysis module is active in the start and after each request the control is transferred to the classification module, therefore, classification module graph fluctuate. Throughout the graph, there is a continuous fluctuation in both the lines which shows how synchronously both modules are operating. Moreover, throughout this runtime, neither of the modules utilized more than 2.5% of the

CPU resources, which shows that the framework and the hosting model perform significantly as the number of cycles per module increase. Additionally, another important thing worth mentioning here is that the framework hosted on a third-part cloud hosting model was running unsupervised for more than 24 hours, throughout this period thousands of messages containing a wide variety of clean and malicious files were sent from the clients. Many of the files sent from the client were quite complex with embedded subdirectories that require separate analysis, the sudden hikes in the graph represent the amount of additional resources consumed while analyzing and classifying such files but despite such scenarios the overall system remained stable and performed well.

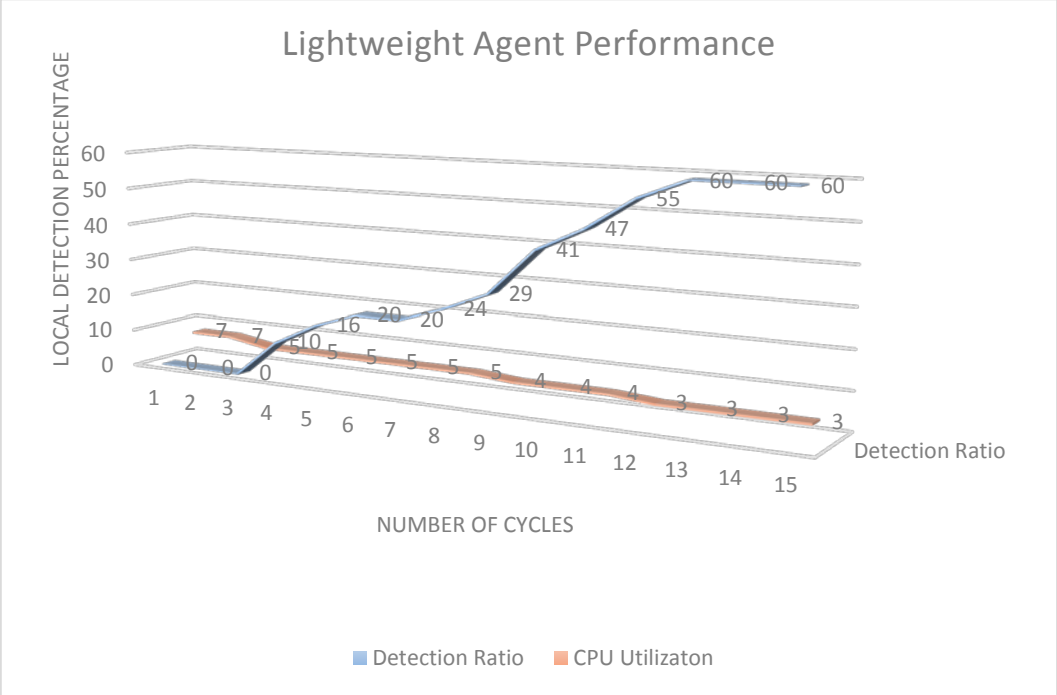


Figure 4.18: Lightweight Agent Performance

The performance of all 35 clients running the lightweight local agent was quite similar, Figure 4.18 presents the evaluation results of lightweight agents by illustrating their CPU consumption and local detection rate. As discussed earlier,

it stores a brief cache of clean and malicious files, which facilitates local detection without sending requests to the cloud-based framework. The lightweight agent is evaluated based on the number of cycles, in each cycle the lightweight agent sent random number of local files as individual messages to the server for identification. It can be seen in the graph that the local detection rate in the first three cycles is zero, which means that the local agent is sending all the files to the server for identification and at the same time it utilizes 7% of the CPU. After first three cycles, there is a sudden increase in local detection and the lightweight local agent is able to identify 10% of the files locally reducing the CPU utilization to 5%. As the framework starts to run in real-time the local cache of the lightweight agent gets populated with the brief analysis features making it possible for the local agent to differentiate between clean and malicious files independently. Consequently, local detection percentage increase with each cycle reaching up to 60% in the thirteenth cycle and dropping the CPU utilization to 3% while continuously operational for 24 hours.

These evaluations not only show that the framework and the hosting model are consistently energy efficient, it also shows that although the hosting model is based on a client server architecture, however, the client can work independently to some extent. This makes the system extremely resilient against targeted attacks or scenarios where the client module gets disconnected with the cloud-based framework. Additionally, the accumulated CPU utilization of all the modules of second aspect of the hosting model, which runs the framework in real-time, is much lesser than the CPU utilization of a majority of host base antiviruses that are discussed earlier in this chapter. Such features make the proposed

framework and its hosting model highly instrumental in both industrial and research environments.

4.7 Discussion

The intelligent malware detection framework required a flexible hosting model in terms of computational and storage resources. In the previous chapter, the framework discussed, proved to be extremely accurate in identifying clean and malicious files, it focused on the weaknesses initially identified in the similar solutions available, specifically; antiviruses. The cloud-based hosted model discussed in this chapter has the capability of hosting the intelligent malware detection framework proposed, implemented, and discussed in the previous chapter. At the same time, this hosting model was required to be energy efficient in terms of consuming the CPU resources, unlike the antiviruses evaluated earlier in this chapter. The hosting model proposed is based on the Amazon cloud platform consuming messaging, compute, and storage services that were combined to design this hosting model. The hosting model designed successfully hosted the malware detection framework and produced quick requests and responses.

One of the main objectives was to design and implement a hosting model that can host the intelligent malware detection framework and at the same time be energy efficient. The energy efficiency in this context means that wherever the framework is hosted it should utilize minimum CPU resources possible. The client-server based architecture, where the server is based on the Amazon's cloud and the client is a lightweight agent running on the host machine, did prove to be energy efficient based on the evaluations performed. The first aspect of the

hosting model which can be called the initial configuration of the whole system to support the second aspect, consumed comparatively higher CPU resources as compared to the second aspect. However, the high CPU consumption by both analysis and classification module, which marked between 40 to 50%, was only for a short period of time and only once in the twenty-four-hour span. Whereas, the second aspect, which operates in a real-time environment showed concrete evidence of energy efficiency on both, client and server sides of the model. When left running for twenty-four hours, the analysis and classification module consumed a maximum of 2.5% of CPU resources while stably operating in an unsupervised environment. The client module also showed promising results by starting with 7% CPU consumption and 0% local detection, which later reached 3% CPU consumption and 60% local detection in fifteen cycles of running in a twenty-four-hour span.

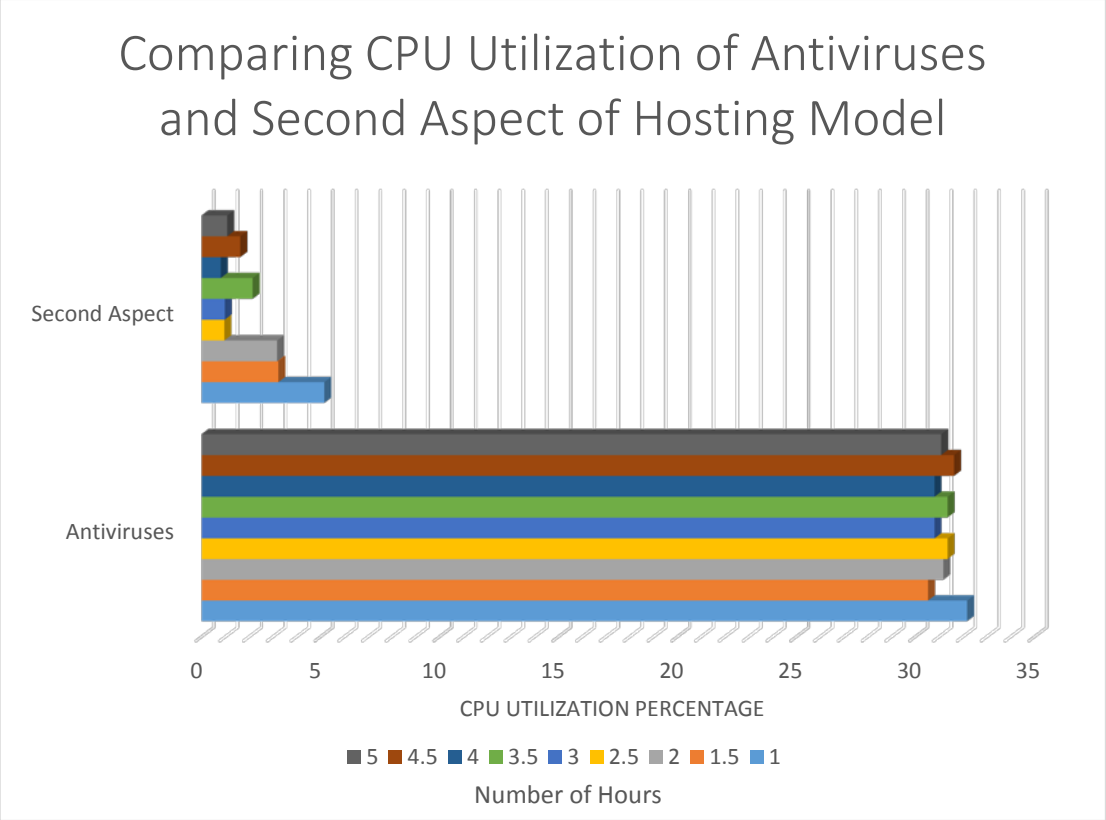


Figure 4.19: Comparing Hosted Framework with Antiviruses

In the final test, we compared the hosting model with antiviruses evaluated earlier in this chapter by running them simultaneously against the same repository of clean and malicious files, both the compared entities were left running for five hours and an average was taken every thirty minutes. The purpose of this evaluation was only to compare the CPU utilization of the proposed hosting model against the major antiviruses in the market. We only compared the second aspect of the hosting model because of its real-time application. The graph presented in Figure 4.19 shows the average CPU utilization of eleven antiviruses and the average of accumulated CPU utilization of client and server modules of second aspect. It can be observed from the values illustrated in the Figure 4.19 that the average CPU utilization of antiviruses is between 30 and 35% continuous for five hours. Whereas, the maximum combine CPU utilization of the second

aspect is 5% and that too in the first hour. The proposed hosting model along with the malware detection framework has shown significant efficiency in managing the CPU consumption. In the first hour of evaluation, the proposed system did reach a maximum of 5% CPU utilization but in the later hours consumption dropped to 3%. The first two hours of evaluation the average consumption was 5 and 3% respectively but the later three hours were significantly more energy efficient and the overall average was 1.26%. However, like the previous evaluation, there was no significant change in the CPU utilization of antiviruses during the five hours of evaluation and the utilization fluctuated between 30 and 32%.

The proposed hosting model proved to be extremely energy efficient, especially when compared with the major antiviruses. Along with being energy efficient, this hosting model is categorically quick in responding to the request messages from the clients, if clients send the request and couldn't differentiate between a clean and malicious file locally.

Although, the hosting model has fulfilled the primary objectives behind its design, there are some weaknesses that could be resolved to make the overall hosting model industrial scale. The client module currently has limited functionality and cannot scan the complete filesystem of a computer, we were able to identify the files in a limited filesystem to evaluate the framework and the hosting model but in an actual real-time environment, this lightweight client module wouldn't be able to identify the malicious files hidden deep in the filesystem of a computer. Additionally, the current structure of the client module doesn't allow it to be used in a networked environment and only supports individual computers, we were

able to run it on 35 separate machines but running on separate machines in a network is not quite efficient. A better approach would be to make the client module a lightweight network service that is supported by the cloud-based framework, which has the capability of scanning ports and other exposed vulnerabilities of a network or a single machine in a network.

CHAPTER 5. CONCLUSION AND FUTURE WORK

The growing number of malware attacks on enterprise and general users, raises concern over the presence of several security software to protect one system. The financial damages caused by such attacks are continuously rising, recent catastrophic attacks by *Mirai, WannaCry*, and *Petya* are few of many instances where security software and organizations were penetrated by the techniques used by these modern malware [7], [3], [5]. The present commercial antiviruses are good in detecting known malware but when it comes to newly released malware or a completely evolved version of previously known malware, the conventional detection techniques used by antiviruses become obsolete. As the information flow is increasing significantly, there is a need for better security mechanisms that can accurately detect known and unknown malware and their infection.

Additionally, another drawback of conventional antiviruses is the CPU resources they consume while running in scan mode. The percentage of CPU resources commercial antiviruses consume is significantly high, most of these antiviruses consume 35 to 50% of CPU resources while scanning the host system for infections [18]. Along with the scarcity in detection capability, this high resource consumption makes the host system more vulnerable against advanced threats by leaving the system with less resources for other high priority services.

Using machine learning techniques to identify malicious activity in a system or in a network have proved to be quite effective [121], [120], [105], [93], [109]. The framework proposed in this study approaches both the problems; a) accurately

identifying malware, and b) energy efficiency. The intelligent malware detection framework used machine learning techniques to enhance the malware detection rate and a cloud-based hosting model to support the operational requirements of the framework.

The framework is built around the heuristics generated by the analysis module through a bespoke feature extracting tool, which extracts a comprehensive set of features from a file through statically analysing it [106]. These heuristics are used to train the machine learning algorithms for accurately differentiating between clean and malicious files. We used decision trees, SVM, and then applied boosting on decision trees to improve the performance of weak classifiers. The analysis module can eliminate the obfuscated parts found in a malware to avoid any inaccurate information in the generated heuristics [106].

We designed multiple experiments to test our proposed framework from different perspectives. We tested our techniques against a dataset of malicious and clean files and applied ten-fold cross-validation followed by above mentioned machine learning algorithms for an unbiased set of experimental results. We used 150000 malicious and 87000 benign executables for training and testing.

SVM performed better than decision tree but applying boosting on decision tree improved the performance by generating the best result of 0.9969 area under the ROC curve. To evaluate the framework against much difficult dataset, we used a dataset of obfuscated malware, using the training of previous experiment. In the obfuscated experiment, boosting on decision tree generated 0.9910 area under the ROC curve. This not only proves the better performance against a difficult dataset of advanced malware, it also suggests that previous training was enough

to detect a different set of malware making it close to real-time detection. The real-time detection generated 0.9963 area under the ROC curve.

The performance of the framework was tested after deploying it on the hosting model, which evaluated both; framework and the hosting model. In this experiment, we initially evaluated the resource consumption of the framework while doing a thorough analysis along with training and testing of the algorithms. In this aspect, which was tested for 24 hours, there were two instances that the resource consumption went 40% to 50% while performing the analysis and training/testing of algorithms. Apart from this instance, the CPU utilization was under 10%.

We evaluated the real-time performance of the framework and the hosting model. In this evaluation, we tested the performance of the lightweight client agent along with the server side of the hosting model and how it caters the requirements of the framework. The fully trained and tested framework was left running for more than 24 hours and 35 separate clients were recursively sending clean and malicious files to be tested. The maximum combine CPU utilization of both lightweight agent and server side of hosting model was 5% in the first hour which later dropped to 3% in the next 3 hours and 1.26% in the rest of 20 hours of evaluation, while running in scan mode. These results show significant improvement as compared to the commercial antiviruses that on average consume 32% while running in scan mode.

Finally, we evaluated the individual performance of the lightweight client agent to identify local detection rate and CPU resource consumption of the host machine. In the initial cycles, the lightweight client agent consumed 7% of CPU resources

while sending all the files to the server for detection. After three cycles, the local agent started detecting 10% of the malware locally while consuming 5% of the CPU resources. After running it for fifteen cycles, this lightweight agent was detecting 60% of the files locally while consuming merely 3% of the CPU resources.

5.1 Limitation and Challenges

The proposed framework along with its hosting model not only presents promising results with enhanced malware detection abilities, it has the potential to provide an alternate platform for personal and enterprise level computer security. However, there are certain limitations and challenges that are required to be eradicated to make this framework ready to be adopted.

One of the fundamental things in this framework is the type of files it can analyse, which was mentioned in the start of this research that the framework only considers PE or .exe file format. This makes the framework limited to work for Windows based environments only and unable to analyse or identify another format of file. The intelligent malware detection framework relies on the heuristics generated by the analysis module to train the machine learning algorithms. Therefore, by making the analysis module to also identify and analyse non-PE based files along with adding some heuristics in algorithms, will make the entire framework capable of operating in broader domains.

Although, the framework can operate independently in real-time, which is also discussed in the evaluation of the framework and hosting model, the entire framework is not completely autonomous. The first aspect of the framework

requires the algorithms to be trained and tested to make real-time detection possible, however, this step is performed by manually storing large sets of clean and malicious files in their respected repositories, which are later used to generate heuristics. To make the system perform independently in long term against modern malware, it requires constant update of heuristics, which can be done by adding automated heuristics update from third-party APIs or honeypots.

The evaluation of the framework presents effective results in malware detection, which can be enhanced by constantly updating several heuristics and patterns. However, if previously unknown malware are successful in proliferating their variants then understanding their behaviour is the key to identify and block their entire network. This can be done by dynamically analysing malware, which is a resource intensive tasks and currently replaced by static analysis in our framework. A better approach will be to add a module in the hosting model that can cater the needs of a sandbox environment for dynamic analysis. This approach will be much energy efficient in terms of accommodating dynamic analysis processes.

Another limitation specifically in the hosting model is that it can only cater individual machines. This allow each client to be directly connected to the server but at the same time in a networked environment it can be time consuming. Lightweight client agent requires to be enhanced to work in a networked environment autonomously.

5.2 Future Work

The main objectives behind proposed framework were to overcome the problems currently faced by the users of commercial antiviruses by enhancing the malware detection rate and making the entire process less resource intensive. The proposed framework satisfies all the requirements initially defined, however, there are certain aspects that can be enhanced to make this framework suitable for different environments.

The current framework is hosted on a client/server architecture where client is a lightweight service and the server is hosted on a cloud-based hosting model. Currently the cloud-based framework has the ability to be scaled to support a large number of clients but the client module only support individual computers separately. This can be enhanced in future to support large enterprise networks with heterogeneous devices. This enhancement should only be successful if the initial idea of energy efficiency is followed, which is possible with the help of service replication [133]. The lightweight host agent can be replaced by a network service that replicates the framework hosted on the cloud. The replicated services will hypothetically consume similar resources as the current lightweight agent but to make it more efficient, dynamic server allocation can be used [134]. This idea is adopted from P2P botnets that change their domain dynamically after regular intervals and each of these domains are not malicious servers [135], in fact, the new domain is a legitimate network node making it extremely hard to stop [136]. The services of the entire framework can be replicated on dynamically allocated nodes with the help of open-source service replication tool, such as; Zookeeper that can replicate the services with limited resources [137] [138].

As mentioned, the idea is to make the framework appropriate for the modern enterprise networks with heterogeneous devices, such as; smart phones, laptops, and other smart devices with a diverse set of operating environments. By making the framework recognize and analyse different file formats, we will be able to make it support a heterogeneous environment. The current hosting model is capable of providing services for multiple large enterprise networks and it can scale if required.

The current framework performs very well against obfuscated malware but as discussed earlier malware are rapidly evolving, therefore, it is required to rapidly evolve the malware detection mechanisms. One of the best solution is to make the anomaly heuristics more elaborative by adding behavioural patterns of malicious software, which can be achieved by dynamically analysing malware. Third-party APIs can be used to perform the dynamic analysis on new and more obfuscated malware samples to generate a much thorough set of heuristics [76]. This will make the framework more resilient against modern and more complicated malware.

The future directions mentioned in this section are not only the aspects in which the proposed framework can be enhanced. Different enhancements discussed can open a new paradigm in security making it more open, resilient, and cross-platform. One of the primary reasons malware are successful against security systems is that they rely on open-source rather than proprietary. The diverse set of heuristics generated through static and dynamic analysis, as suggested in this section, can support security research community along with making malware identification more efficient.

REFERENCES

[1] M. Coren, “Data is expected to double every two years for the next decade,” *Quartz*, 30-Jul-2017. .

[2] S. R. Symantec, “What you need to know about the WannaCry Ransomware,” *Symantec Security Response*, 23-May-2017. [Online]. Available: http://www.symantec.com/connect/blogs/what-you-need-know-about-wannacry-ransomware. [Accessed: 31-Jul-2017].

[3] I. Sherr, “WannaCry ransomware: Everything you need to know,” *CNET*, 19-May-2017. [Online]. Available: https://www.cnet.com/uk/news/wannacry-wannacrypt-uiwix-ransomware-everything-you-need-to-know/. [Accessed: 31-Jul-2017].

[4] A. Ng, “Petya ransomware slams Windows PCs shut in massive attack,” *CNET*, 27-Jun-2017. [Online]. Available: https://www.cnet.com/uk/news/unprecedented-cyberattack-hits-businesses-across-europe/. [Accessed: 31-Jul-2017].

[5] S. R. Symantec, “Petya ransomware outbreak: Here’s what you need to know,” *Symantec Security Response*, 27-Jun-2017. [Online]. Available: http://www.symantec.com/connect/blogs/petya-ransomware-outbreak-here-s-what-you-need-know. [Accessed: 31-Jul-2017].

[6] J. Leyden, “Sh... IoT just got real: Mirai botnet attacks targeting multiple ISPs,” 12-Feb-2016. [Online]. Available: http://www.theregister.co.uk/2016/12/02/broadband_mirai_takedown_analysis/. [Accessed: 10-Dec-2016].

[7] P. Ducklin, “Deutsche Telekom outage: Mirai botnet goes double-rogue,” *Naked Security*, 29-Nov-2016. .

[8] A. Ng, “Ransomware’s global epidemic is just getting started,” *CNET*, 28-Jun-2017. [Online]. Available: https://www.cnet.com/uk/news/petya-goldeneye-wannacry-ransomware-global-epidemic-just-started/. [Accessed: 31-Jul-2017].

[9] L. H. Newman, “The Botnet That Broke the Internet Isn’t Going Away,” *WIRED*, 12-Sep-2016. [Online]. Available: https://www.wired.com/2016/12/botnet-broke-internet-isnt-going-away/. [Accessed: 31-Jul-2017].

[10] V. Harrison and J. Pagliery, “Nearly 1 million new malware threats released every day,” *CNNMoney*, 14-Apr-2015. [Online]. Available: http://money.cnn.com/2015/04/14/technology/security/cyber-attack-hacks-security/index.html. [Accessed: 10-Dec-2016].

[11] I. Ponemon, “2015 Cost of Cyber Crime Study: Global,” Oct-2015. [Online]. Available: http://www.cnmeonline.com/myresources/hpe/docs/HPE_SIEM_Analyst

Report-_2015_Cost_of_Cyber_Crime_Study_-_Global.pdf. [Accessed: 31-Jul-2017].

[12] T. Warren, J. Favole, S. Haber, and E. Hamilton, “Cybercrime Costs More Than You Think,” *Hamilton Place Strategies*, 02-Feb-2016. .

[13] L. Zeltser, “4 Steps To Combat Malware Enterprise-Wide,” Jan-2011. [Online]. Available: https://zeltser.com/malware-in-the-enterprise/. [Accessed: 31-Jul-2017].

[14] C. Osborne, “Most companies take over six months to detect data breaches,” *ZDNet*, 19-May-2015. [Online]. Available: http://www.zdnet.com/article/businesses-take-over-six-months-to-detect-data-breaches/. [Accessed: 31-Jul-2017].

[15] O. Ralph, “Malicious attacks account for bulk of data loss,” *Financial Times*, 08-Mar-2016. [Online]. Available: https://www.ft.com/content/7dec0636-e541-11e5-bc31-138df2ae9ee6. [Accessed: 31-Jul-2017].

[16] H. Binsalleeh *et al.*, “On the analysis of the Zeus botnet crimeware toolkit,” in *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)*, 2010, pp. 31–38.

[17] D. S. Liles, *ICCWS2014- 9th International Conference on Cyber Warfare & Security: ICCWS 2014*. Academic Conferences Limited, 2014.

[18] Q. K. A. Mirza, G. Mohi-Ud-Din, and I. Awan, “A Cloud-Based Energy Efficient System for Enhancing the Detection and Prevention of Modern Malware,” in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 2016, pp. 754–761.

[19] H. Al-Mohannadi, Q. Mirza, A. Namanya, I. Awan, A. Cullen, and J. Disso, “Cyber-Attack Modeling Analysis Techniques: An Overview,” in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, 2016, pp. 69–76.

[20] M. Christodorescu and S. Jha, “Testing Malware Detectors,” in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA, 2004, pp. 34–44.

[21] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification,” in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, New York, NY, USA, 2016, pp. 183–194.

[22] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, “OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection,” in *International Joint Conference CISIS’12-ICEUTE´12-SOCO´12 Special Sessions*, Springer, Berlin, Heidelberg, 2013, pp. 271–280.

[23] P. Szor, *The Art of Computer Virus Research and Defense*. Pearson Education, 2005.

[24] F.-G. Deng, X.-H. Li, H.-Y. Zhou, and Z. Zhang, “Improving the security of multiparty quantum secret sharing against Trojan horse attack,” *Phys. Rev. A*, vol. 72, no. 4, p. 044302, Oct. 2005.

[25] P. K. Kerr, J. Rollins, and C. A. Theohary, *The Stuxnet Computer Worm: Harbinger of an Emerging Warfare Capability*. Congressional Research Service, 2010.

[26] A. Aziz, “Computer worm defense system and method,” US8006305 B2, 23-Aug-2011.

[27] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy, “A Crawler-based Study of Spyware in the Web.,” in *NDSS*, 2006, vol. 1, p. 2.

[28] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer, “Behavior-based Spyware Detection,” 2006. [Online]. Available: http://static.usenix.org/legacy/events/sec06/tech/full_papers/kirda/kirda_html/. [Accessed: 31-Jul-2017].

[29] CISCO, “What Is the Difference: Viruses, Worms, Trojans, and Bots?,” *Cisco*, 2010. [Online]. Available: http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html. [Accessed: 02-Oct-2014].

[30] G. Gu, J. Zhang, and W. Lee, “BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic.,” in *NDSS*, 2008, vol. 8, pp. 1–18.

[31] R. Riley, X. Jiang, and D. Xu, “Multi-aspect Profiling of Kernel Rootkit Behavior,” in *Proceedings of the 4th ACM European Conference on Computer Systems*, New York, NY, USA, 2009, pp. 47–60.

[32] S. Embleton, S. Sparks, and C. C. Zou, “SMM rootkit: a new breed of OS independent malware,” *Secur. Commun. Netw.*, vol. 6, no. 12, pp. 1590–1605, Dec. 2013.

[33] E. Erturk, “A case study in open source software security and privacy: Android adware,” in *World Congress on Internet Security (WorldCIS-2012)*, 2012, pp. 189–191.

[34] K. Kancherla, J. Donahue, and S. Mukkamala, “Packer identification using Byte plot and Markov plot,” *J. Comput. Virol. Hacking Tech.*, vol. 12, no. 2, pp. 101–111, May 2016.

[35] G. Mezzour, L. R. Carley, and K. M. Carley, “Longitudinal analysis of a large corpus of cyber threat descriptions,” *J. Comput. Virol. Hacking Tech.*, vol. 12, no. 1, pp. 11–22, Feb. 2016.

[36] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, “Cutting the Gordian Knot: A Look Under the Hood of Ransomware Attacks,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2015, pp. 3–24.

[37] F. Mercaldo, V. Nardone, A. Santone, and C. A. Visaggio, “Ransomware Steals Your Phone. Formal Methods Rescue It,” in *Formal Techniques for Distributed Objects, Components, and Systems*, 2016, pp. 212–221.

[38] C. Everett, “Ransomware: to pay or not to pay?,” *Comput. Fraud Secur.*, vol. 2016, no. 4, pp. 8–12, Apr. 2016.

[39] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, “CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data,” in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 303–312.

[40] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, “Automated Classification and Analysis of Internet Malware,” in *Recent Advances in Intrusion Detection*, vol. 4637, C. Kruegel, R. Lippmann, and A. Clark, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 178–197.

[41] M. Damshenas, A. Dehghantanha, and R. Mahmoud, “A survey on malware propagation, analysis, and detection,” *Int. J. Cyber-Secur. Digit. Forensics IJCSDF*, vol. 2, no. 4, pp. 10–29, 2013.

[42] R. McMillan, “Is Antivirus Software a Waste of Money?,” *WIRED*, 02-Mar-2012. [Online]. Available: http://www.wired.com/2012/03/antivirus/. [Accessed: 07-Jul-2015].

[43] E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification: A Survey,” *J. Inf. Secur.*, vol. 05, no. 02, pp. 56–64, 2014.

[44] G. Jacob, H. Debar, and E. Filiol, “Behavioral detection of malware: from a survey towards an established taxonomy,” *J. Comput. Virol.*, vol. 4, no. 3, pp. 251–266, Aug. 2008.

[45] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, “A survey on heuristic malware detection techniques,” in *The 5th Conference on Information and Knowledge Technology*, 2013, pp. 113–120.

[46] M. Schiffman, “A Brief History of Malware Obfuscation: Part 1 of 2,” *blogs@Cisco - Cisco Blogs*, 15-Feb-2010. [Online]. Available: http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2/. [Accessed: 08-Sep-2014].

[47] I. You and K. Yim, “Malware Obfuscation Techniques: A Brief Survey,” 2010, pp. 297–300.

[48] A. Sharma and S. K. Sahay, “Evolution and detection of polymorphic and metamorphic malwares: A survey,” *ArXiv Prepr. ArXiv14067061*, 2014.

[49] S. M. Sridhara and M. Stamp, “Metamorphic worm that carries its own morphing engine,” *J. Comput. Virol. Hacking Tech.*, vol. 9, no. 2, pp. 49–58, May 2013.

[50] P. Szor, *Metamorphic Virus: Analysis and Detection*. Pearson Education, 2005.

[51] A. H. Toderici and M. Stamp, “Chi-squared distance and metamorphic virus detection,” *J. Comput. Virol. Hacking Tech.*, vol. 9, no. 1, pp. 1–14, Feb. 2013.

[52] G. Shanmugam, R. M. Low, and M. Stamp, “Simple substitution distance and metamorphic detection,” *J. Comput. Virol. Hacking Tech.*, vol. 9, no. 3, pp. 159–170, Aug. 2013.

[53] P. Desai and M. Stamp, “A highly metamorphic virus generator,” *Int. J. Multimed. Intell. Secur.*, vol. 1, no. 4, pp. 402–427, Jan. 2010.

[54] S. Ibrahim and B. B. Rad, “Camouflage in Malware: from Encryption to Metamorphism,” Aug. 2012.

[55] D. Bruschi, M. Lorenzo, and M. Monga, “Detecting self-mutating malware using control-flow graph matching,” in *Detection of Intrusions and Malware & Vulnerability Assessment*, vol. 4064, 2006, pp. 129–143.

[56] P. OKane, S. Sezer, and K. McLaughlin, “Obfuscation: The Hidden Malware,” *IEEE Secur. Priv.*, vol. 9, no. 5, pp. 41–47, Sep. 2011.

[57] M. N. Gagnon, S. Taylor, and A. K. Ghosh, “Software Protection through Anti-Debugging,” *IEEE Secur. Priv.*, vol. 5, no. 3, pp. 82–84, May 2007.

[58] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, “Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware,” in *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008*, 2008, pp. 177–186.

[59] G. Wagener, R. State, and A. Dulaunoy, “Malware behaviour analysis,” *J. Comput. Virol.*, vol. 4, no. 4, pp. 279–287, Nov. 2008.

[60] A. D. Schmidt *et al.*, “Static Analysis of Executables for Collaborative Malware Detection on Android,” in *2009 IEEE International Conference on Communications*, 2009, pp. 1–5.

[61] VirusTotal, “VirusTotal - Free Online Virus, Malware and URL Scanner,” 2016. [Online]. Available: https://www.virustotal.com/. [Accessed: 14-Jul-2016].

[62] G. Amato, “guelfoweb/peframe,” *GitHub*, 2016. [Online]. Available: https://github.com/guelfoweb/peframe. [Accessed: 29-Dec-2016].

[63] PEiD, “PEiD - aldeid,” 2007. [Online]. Available: https://www.aldeid.com/wiki/PEiD. [Accessed: 13-Jul-2017].

[64] M. Williams, “Identifying malware with PEStudio,” 2015. [Online]. Available: https://betanews.com/2015/01/19/identifying-malware-with-pestudio/. [Accessed: 13-Jul-2017].

[65] L. Zeltser, “SANS Digital Forensics and Incident Response Blog | Automating Static Malware Analysis With MASTIFF | SANS Institute,” 2013. [Online]. Available: https://digital-forensics.sans.org/blog/2013/05/07/mastiff-for-auto-static-malware-analysis. [Accessed: 13-Jul-2017].

[66] Joxean, *pyew: Official repository for Pyew*. 2017.

[67] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, “Automatic Reverse Engineering of Malware Emulators,” in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 94–109.

[68] A. Yadav, “Reverse Engineering with OllyDbg,” *InfoSec Resources*, 01-Nov-2013. [Online]. Available: http://resources.infosecinstitute.com/reverse-engineering-ollydbg/. [Accessed: 13-Jul-2017].

[69] A. Malik, “Reversing Basics - A Practical Approach Using IDA Pro | www.SecurityXploded.com,” 2009. [Online]. Available: http://securityxploded.com/reversing-basics-ida-pro.php. [Accessed: 13-Jul-2017].

[70] A. Gilpin, “gdb Tutorial,” 2004. [Online]. Available: https://www.cs.cmu.edu/~gilpin/tutorial/. [Accessed: 13-Jul-2017].

[71] Hydrasky, “Immunity Debugger – All things in moderation,” 2017. [Online]. Available: https://hydrasky.com/malware-analysis/immunity-debugger/. [Accessed: 13-Jul-2017].

[72] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, “Dynamic Analysis of Malicious Code,” *J. Comput. Virol.*, vol. 2, no. 1, pp. 67–77, Aug. 2006.

[73] M. EGELE, THEODOOR SCHOLTE, ENGIN KIRDA, and CHRISTOPHER KRUEGEL, “A Survey on Dynamic Malware Analysis Techniques and Tools,” *ACM Comput. Surv.*, vol. V, 2012.

[74] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, “Graph-based malware detection using dynamic analysis,” *J. Comput. Virol.*, vol. 7, no. 4, pp. 247–258, Nov. 2011.

[75] B. B. H. Kang and A. Srivastava, “Dynamic Malware Analysis,” in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds. Springer US, 2011, pp. 367–368.

[76] honeynet, “Malwr.com: powered by Cuckoo | The Honeynet Project,” 2012. [Online]. Available: https://www.honeynet.org/node/808. [Accessed: 13-Jul-2017].

[77] R. D. Cambridge, “Method and system for bi-directional updating of antivirus database,” US7080000 B1, 18-Jul-2006.

[78] D. Venugopal and G. Hu, “Efficient Signature Based Malware Detection on Mobile Devices,” *Mobile Information Systems*, 2008. [Online]. Available: https://www.hindawi.com/journals/misy/2008/712353/abs/. [Accessed: 13-Jul-2017].

[79] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, “An intelligent PE-malware detection system based on association mining,” *J. Comput. Virol.*, vol. 4, no. 4, pp. 323–334, Nov. 2008.

[80] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimaila, and S. Rogers, “Malware target recognition via static heuristics,” *Comput. Secur.*, vol. 31, no. 1, pp. 137–147, Feb. 2012.

[81] X. Wang and H. Xie, “Heuristic botnet detection,” US8555388 B1, 08-Oct-2013.

[82] M. Spiegel, B. McCorkendale, and W. Sobel, “Heuristic detection and termination of fast spreading network worm attacks,” US7159149 B2, 02-Jan-2007.

[83] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, “Classification of malware based on integrated static and dynamic features,” *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 646–656, Mar. 2013.

[84] J. Z. Kolter and M. A. Maloof, “Learning to Detect Malicious Executables in the Wild,” in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2004, pp. 470–478.

[85] T. E. Dube, “A Novel Malware Target Recognition Architecture for Enhanced Cyberspace Situation Awareness,” Air Force Institute of Technology, Wright Patterson AFB, OH, USA, 2011.

[86] A. P. Namanya, Q. K. A. Mirza, H. Al-Mohannadi, I. U. Awan, and J. F. P. Disso, “Detection of Malicious Portable Executables Using Evidence Combinational Theory with Fuzzy Hashing,” in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2016, pp. 91–98.

[87] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, “Behavior-based features model for malware detection,” *J. Comput. Virol. Hacking Tech.*, vol. 12, no. 2, pp. 59–67, May 2016.

[88] A. D. Keromytis and R. Di Pietro, Eds., *Security and Privacy in Communication Networks*, vol. 106. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[89] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho, “Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection,” in *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, 2010, pp. 201–203.

[90] D. Gavriluţ, M. Cimpoeşu, D. Anton, and L. Ciortuz, “Malware detection using machine learning,” in *2009 International Multiconference on Computer Science and Information Technology*, 2009, pp. 735–741.

[91] R. J. Mangialardo and J. C. Duarte, “Integrating Static and Dynamic Malware Analysis Using Machine Learning,” *IEEE Lat. Am. Trans.*, vol. 13, no. 9, pp. 3080–3087, Sep. 2015.

[92] Z. A. Markel, “Machine Learning Based Malware Detection,” NAVAL ACADEMY ANNAPOLIS MD, NAVAL ACADEMY ANNAPOLIS MD, USNA-TSPR-440, May 2015.

[93] J. Z. Kolter and M. A. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” *J. Mach. Learn. Res.*, vol. 7, no. Dec, pp. 2721–2744, 2006.

[94] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, “Opcode sequences as representation of executables for data-mining-based unknown malware detection,” *Inf. Sci.*, vol. 231, pp. 64–82, May 2013.

[95] T. H. Titonis, N. R. Manohar-Alers, and C. J. Wysopal, “Automated behavioral and static analysis using an instrumented sandbox and machine learning classification for mobile security,” US9672355 B2, 06-Jun-2017.

[96] Y. Li, X. D. Tan, and K. Xiao, “Systems and methods for detecting malware variants,” US8806641 B1, 12-Aug-2014.

[97] P. Natani and D. Vidyarthi, “An Overview of Detection Techniques for Metamorphic Malware,” in *Intelligent Computing, Networking, and Informatics*, Springer, New Delhi, 2014, pp. 637–643.

[98] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, “A Survey of Stealth Malware Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions,” *IEEE Commun. Surv. Tutor.*, vol. 19, no. 2, pp. 1145–1172, Secondquarter 2017.

[99] J. Ming, Z. Xin, P. Lan, D. Wu, P. Liu, and B. Mao, “Replacement Attacks: Automatically Impeding Behavior-Based Malware Specifications,” in *Applied Cryptography and Network Security*, 2015, pp. 497–517.

[100] J. Fraley, “Improved Detection for Advanced Polymorphic Malware,” *CEC Theses Diss.*, Jan. 2017.

[101] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, “A comparison of static, dynamic, and hybrid analysis for malware detection,” *J. Comput. Virol. Hacking Tech.*, vol. 13, no. 1, pp. 1–12, Feb. 2017.

[102] S. Ranjan, “Machine learning based botnet detection using real-time extracted traffic features,” US8682812 B1, 25-Mar-2014.

[103] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. CRC Press, 2016.

[104] Y.-T. Hou, Y. Chang, T. Chen, C.-S. Laih, and C.-M. Chen, “Malicious web content detection by machine learning,” *Expert Syst. Appl.*, vol. 37, no. 1, pp. 55–60, Jan. 2010.

[105] J. Sahs and L. Khan, “A Machine Learning Approach to Android Malware Detection,” in *2012 European Intelligence and Security Informatics Conference*, 2012, pp. 141–147.

[106] Q. K. A. Mirza, I. Awan, and M. Younas, “CloudIntell: An intelligent malware detection system,” *Future Gener. Comput. Syst.*, Jul. 2017.

[107] L. Sun, S. Versteeg, S. Boztaş, and T. Yann, “Pattern Recognition Techniques for the Classification of Malware Packers,” in *Information Security and Privacy*, 2010, pp. 370–390.

[108] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, 1992, pp. 144–152.

[109] M. Kruczkowski and E. N. Szynkiewicz, “Support Vector Machine for Malware Analysis and Classification,” in *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2014, vol. 2, pp. 415–420.

[110] J. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines,” *Microsoft Res.*, Apr. 1998.

[111] J. C. Platt, “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods,” in *Advances in Large Margin Classifiers*, 1999, pp. 61–74.

[112] S. R. Safavian and D. Landgrebe, “A survey of decision tree classifier methodology,” *IEEE Trans. Syst. Man Cybern.*, vol. 21, no. 3, pp. 660–674, May 1991.

[113] Y. Freund and R. E. Schapire, *Experiments with a New Boosting Algorithm*. 1996.

[114] M. V. Joshi, V. Kumar, and R. C. Agarwal, “Evaluating boosting algorithms to classify rare classes: comparison and improvements,” in *Proceedings 2001 IEEE International Conference on Data Mining*, 2001, pp. 257–264.

[115] X. Carreras and L. Marquez, “Boosting Trees for Anti-Spam Email Filtering,” *arXiv:cs/0109015*, Sep. 2001.

[116] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, vol. 1, p. I-511-I-518 vol.1.

[117] T. G. Dietterich, “Ensemble Methods in Machine Learning,” in *Multiple Classifier Systems*, 2000, pp. 1–15.

[118] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (ROC) curve.,” *Radiology*, vol. 143, no. 1, pp. 29–36, Apr. 1982.

[119] H. V. Nath and B. M. Mehtre, “Static Malware Analysis Using Machine Learning Methods,” in *Recent Trends in Computer Networks and Distributed Systems Security*, 2014, pp. 440–450.

[120] K. Rieck, P. Trinius, C. Willems, and T. Holz, “Automatic analysis of malware behavior using machine learning,” *J. Comput. Secur.*, vol. 19, no. 4, pp. 639–668, Jan. 2011.

[121] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. L. Traon, “Empirical assessment of machine learning-based malware detectors for Android,” *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 183–211, Feb. 2016.

[122] P. Faruki, V. Kumar, A. B, M. S. Gaur, V. Laxmi, and M. Conti, “Platform Neutral Sandbox for Analyzing Malware and Resource Hogger Apps,” in *International Conference on Security and Privacy in Communication Networks*, 2014, pp. 556–560.

[123] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, “Evolution, Detection and Analysis of Malware for Smart Devices,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 2, pp. 961–987, Second 2014.

[124] X. Chen, B. Mu, and Z. Chen, “NetSecu: A Collaborative Network Security Platform for In-network Security,” in *2011 Third International Conference on Communications and Mobile Computing*, 2011, pp. 59–64.

[125] D. Robinson, *Amazon Web Services Made Simple: Learn How Amazon EC2, S3, SimpleDB and SQS Web Services Enables You to Reach Business Goals Faster*. London, UK, UK: Emereo Pty Ltd, 2008.

[126] H. Yoon, A. Gavrilovska, K. Schwan, and J. Donahue, “Interactive Use of Cloud Services: Amazon SQS and S3,” in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 523–530.

[127] J. Barr, A. Narin, and J. Varia, “Building fault-tolerant applications on AWS,” *Amaz. Web Serv.*, pp. 1–15, 2011.

[128] J. Varia, “Architecting for the cloud: Best practices,” *Amaz. Web Serv.*, vol. 1, pp. 1–21, 2010.

[129] J. Varia and S. Mathew, “Overview of amazon web services,” *Amaz. Web Serv.*, 2014.

[130] S. Obrutsky, *Cloud Storage: Advantages, Disadvantages and Enterprise Solutions for Business*. 2016.

[131] Y.-R. Chen *et al.*, “Developing a Common Repository for Exchangeable Learning Objects,” in *PROCEEDINGS OF THE 9TH IEEE INTERNATIONAL CONFERENCE ON UBI-MEDIA COMPUTING“ UMEDIA-2016,”* 2016, pp. 1–6.

[132] V. Nagaveni and D. V. Pandya, “CLOUD COMPUTING STRATERGY: CLOUD STORAGE ANDSPECIFICATION REQUIREMENT,” *Int. Educ. Res. J.*, vol. 2, no. 12, Jan. 2017.

[133] W. T. Tsai, P. Zhong, J. Elston, X. Bai, and Y. Chen, “Service Replication Strategies with MapReduce in Clouds,” in *2011 Tenth International Symposium on Autonomous Decentralized Systems*, 2011, pp. 381–388.

[134] V. Bortnikov, G. Chockler, D. Perelman, A. Roytman, S. Shachor, and lya Shnayderman, “FRAPPE: Fast Replication Platform for Elastic Services,” *ArXiv160405959 Cs*, Apr. 2016.

[135] C. Yin, “Towards Accurate Node-Based Detection of P2P Botnets,” *The Scientific World Journal*, 2014. [Online]. Available: https://www.hindawi.com/journals/tswj/2014/425491/abs/. [Accessed: 31-Jul-2017].

[136] H. Hang, X. Wei, M. Faloutsos, and T. Eliassi-Rad, “Entelecheia: Detecting P2P botnets in their waiting stage,” in *2013 IFIP Networking Conference*, 2013, pp. 1–9.

[137] S. Skeirik, R. B. Bobba, and J. Meseguer, “Formal Analysis of Fault-tolerant Group Key Management Using ZooKeeper,” in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013, pp. 636–641.

[138] F. Junqueira and B. Reed, *ZooKeeper: Distributed Process Coordination*. O’Reilly Media, Inc., 2013.