A Feature Engineering Approach for Classification and Detection of Polymorphic Malware using Machine Learning

By

Emmanuel Masabo
BSc (UR-CST), M.Eng (CSU)
Email: masabem@gmail.com
Reg. No. 2014/HD05/18682X

A Thesis Submitted to the Directorate of Research and Graduate Training in Fulfilment of the Requirements for the Award of the Degree of Doctor of Philosophy in Software Engineering of Makerere University

Department of Computer Networks
School of Computing and Informatics Technology
Makerere University

December 2019

Author's declaration

I hereby declare that this research thesis is my original work and has not previously, in part or in its entirely been presented to any other university towards the award of any degree.

Signed: ... . Date: December 3, 2019

Emmanuel MASABO, RegNo.: 2014/HD05/18682X

This research thesis has been presented for examination with my approval as the university supervisor.

Signed: ..... Date: 04/Dec/2019

Dr. Swaib Kyanda Kaawaase (PhD)

Signed: ..... Date: 4/Dec/2019

Dr. Julianne Sansa Otim (PhD)

Abstract

Malicious software have evolved as a major threat in Computer Systems. These threats aim at compromising systems from low level to high level, from individuals to large corporations or even government institutions. Their creation technology is constantly evolving by using sophisticated tactics to create multiple instances of the existing ones, by encrypting the malicious payload as well as changing the code structure at each infection, while retaining the same functionality. Due to polymorphism in today's malware, current solutions are not yet able to sufficiently address this problem. Such solutions are mostly signature-based and a changing malware means a changing signature. Generally, the detection rate should be 100% in an ideal setup. However, there is a high rate of False Positives and False Negatives. The frequent updating of signature-based systems requires more resources. Previous research shows that current malware detection rate by anti-malware products is between 25% and 50%. They easily evade detection and classification in their respective families is also hard, making it more difficult to eliminate them.

This study addresses this problem by providing a structured deeper analysis of key features or parameters that enable polymorphism in malware. This study proposed a Novel Feature Engineering (NFE) approach for a better classification and detection of polymorphic malware based on structural and behavioural features. Key features were highly engineered to help in building powerful classification and detection models. This research employs NFE approach to create an improved malware classification model. Three classifiers namely KNN, Linear Discriminant Analysis and Gradient Boosting Machine (GBM) were used for this task. The best model achieved an accuracy of 94% on malware classification. This research also developed an early-stage Deep learning based detection model. The experiments achieved an earlier detection accuracy of 99.66% within the first five seconds of file execution. The solutions provided by this research will revolutionize anti-malware industry in creating better protection mechanisms.

Acknowledgements

I would like to thank Dr. Swaib Kyanda Kaawaase and Dr. Juliane Sansa Otim for guiding me as my supervisors. Their commitment, feedback, and expertise shaped my research to a better level. I would also like to thank Dr. John Ngubiri for his advanced technical guidance, which helped me to successfully achieve my objectives. I am so grateful for his wisdom and instructions all along this PhD journey. I particularly thank RUFORUM and METEGA for financially supporting my studies. The same level of gratitude goes to SEiA 2018 organisers, Sci-GaiA organisers and Black in AI organisers for supporting me with travel grants to attend conferences. I am so grateful to my dear wife Irene Mushimiyimana for her encouragement and consistent support during my academic journey. I extend my gratitude to my children Eddrick Masabo, Elliot Masabo, Elvin Masabo and Eliora Masabo for their love, care and prayers. Finally, I would like to extend my gratitude to God for his love, life, guidance, protection and providence, by which this journey has been possible.

Dedication

I am so appreciative to my late parents Nshunguyinka Mathias and Mukakibibi Immaculee, whose love, childhood education and moral support enabled me to achieve this level of success. I would like to especially dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Outline: In this chapter, we provide a general discussion on malware in computer systems. We mainly focus on polymorphic executable malware that affect Windows Operating Systems. In Section 1.1, we give the background on malware and discuss the practice and challenges in addressing malware. In Section 1.2, we provide a detailed problem statement. Research objectives and questions are provided in Section 1.3 and Section 1.4, respectively. In Section 1.5, we discuss the significance of the study. Research contributions are highlighted in Section 1.6. Finally, the thesis outline is presented in Section 1.7.

1.1 Background of the Study

Malicious software, commonly known as malware represent hostile programs developed with the deliberate intent of compromising the security of computer systems, cause damage or intentionally violate the pertinent information privacy [1]. Malware is a general term which comprises all types of hostiles programs like rootkits, adware, bots, bugs, spyware, trojan horses, viruses and worms[2].

These malware undoubtedly cause an enormous security concern in today's computing environment, where people's lives are largely intertwined with digital devices [3]. Multiple devices such as personal computers, portable laptops, mobile tablets, etc., have undoubtedly gained incredible popularity when used for instantly accessing IT services. However, security in the use of such devices is of great concern because of many frequent

cyber attacks from malware [4].

According to the Symantec Internet Security Threat Report [5], nearly one million new threats created were injected daily into the wild in 2014. They also reported that more than 430 million malicious software had been injected into the Internet in 2015 [6]. In the third quarter of 2017, Kaspersky detected 277,646,376 malicious attacks from the Internet in 185 countries around the world [7]. A considerable number of these threats are usually new variants of existing malware families, although unknown types could as well be instantly detected [7]. Kaspersky Report of the third quarter of 2017 also provides ten top countries where Internet resources contain malware [7], as shown in Figure 1.1.

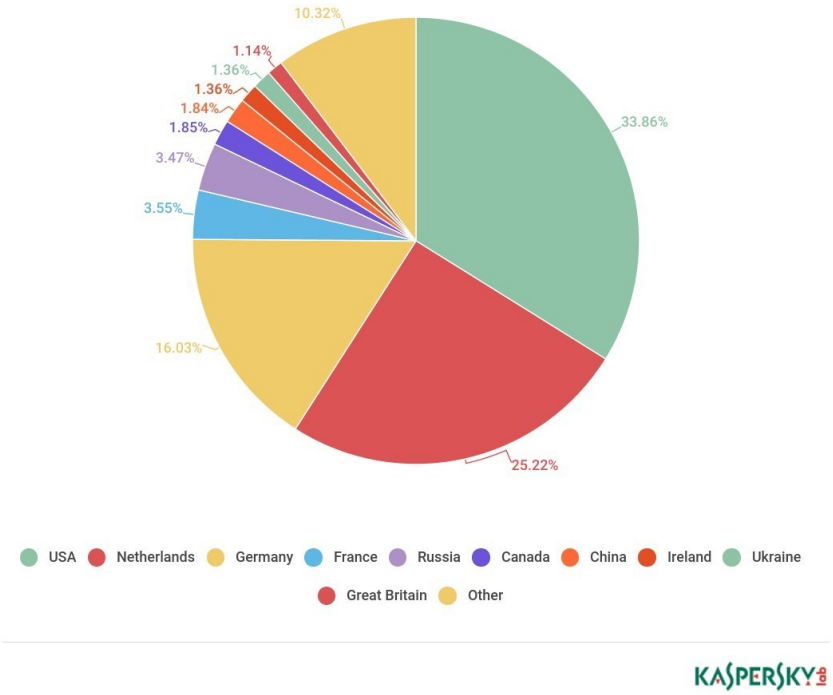

Figure 1.1: Distribution of web malware seeded sources by country in the third quarter of 2017 [7]

Most malicious codes are Windows compatible executable files, as described in Virus Total submission statistics [8] (See Figure 1.2). Virus Total is an online service that facilitates the analysis of files and URLs with the ultimate goal of identifying malware. Virus Total contacts many antivirus engines that, in turn, indicate whether the file has been identified as a malware or not. The cyber threat is constantly evolving, where

massive Distributed Denial of Service (DDoS) attacks are launched from less secure Internet of Things (IoT) devices [8]. This affects all key areas, including governments, private and public organizations.

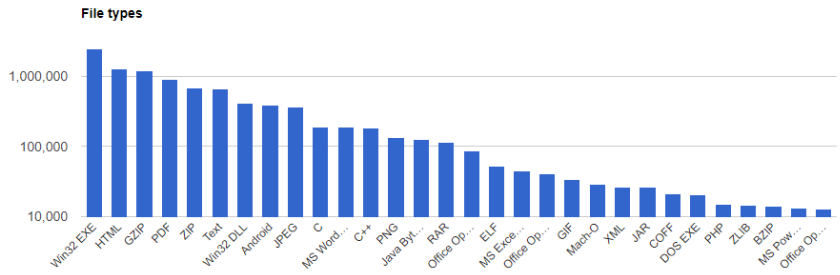

Figure 1.2: Distribution of web malware seeded sources by file type in the third quarter of 2017

Early malware were spread through file transfers using floppy and removable disks. Once a user opened an infected file on a clean computer, the system could be infected [9]. Later malware were much more complex, mainly spread from the Internet, and can be embedded in various types of files such as Portable Document Files (pdf), Portable Executable (PE) files, web formats, etc. [10], [11], [12], [13], [14]. Currently, malware are even more complex and highly sophisticated in terms of ferocious attacks and avoidance of detection [15], [16], [17], [18]. They are able to alter their identity with each new attack without changing the main functionality of the malware. They can encrypt and decrypt their payload using different keys [19], [20],[21], thus creating an infinite number of new variants [22], for soundly defeating the detection systems as well as hardening the analysis process [23], [24], [25], [26]. They can also manipulate their key codes by renaming variables, replacing controls, inserting junk codes, and so on...[27].

In order to adequately deal with the problem of malware, researchers create innovative techniques to prominently identify and accurately detect them. This process requires two crucial steps. In the first place, malware is carefully analysed to extract useful information about its specific functionality. Malware analysis is typically intended to answer ultimate questions such as what happened, how did it take place, who was the attacker, which facilities did the malware use, what vulnerabilities were exploited and what was their intent. Two analysis techniques are frequently used, namely, static and dynamic analysis [28]. With static analysis, a selected sample is analysed without executing it, while dynamic analysis requires the execution of the sample in order to scrupulously observe its behaviours [28]. This is performed in an emulated/virtual environment to prevent

any spread of the infection to the physical system. The goal is to gather unquestionable information and evidence that can undoubtedly help to get an acute response to an attack. After a sample is confirmed as malware during analysis, the analyst establishes a pattern that makes it possible to identify the malware (i.e. the unique signature). A new signature is then generated and used to quickly identify and counterfeit subsequent malware attacks in the future [17], [29], [30]. This signature should be reliable enough to be applied to only those malware it is designed for, without affecting genuine files. Otherwise, this could lead to false detections [31]. Secondly, classification or detection systems are developed based on the information provided during analysis [32], [33], [17], [34], [35], [26], [29]. They store signature patterns in their databases and steadfastly keep them updated regularly. However, this procedure restricts such systems from detecting polymorphic malware since they continuously alter their identities [36].

Many researchers have tentatively proposed a considerable number of utilitarian approaches to deal with polymorphic malware as described in articles [37], [38], [24]. Despite all the concerted efforts undertaken, this remains a complex problem [39], [40], [41] and entails more research endeavours. Anti-malware programs have quite limited effectiveness in unambiguously identifying and eliminating threats [35], [33]. Yuan et al.[42] reported that the detection rate of sophisticated zero-day and polymorphic malware by anti-virus products is between 25% and 50%.

In order to tremendously improve existing classification and detection techniques, further research is required and should deeply focus on the dissection and noble extraction of all key features, highly significant to sufficiently represent inner attributes that best characterize the malware [43]. Prior to the development of classification and detection models, features must be properly processed and well-engineered to maximize the efficiency of machine learning algorithms [44]. Feature engineering is one of the core components of a successful machine learning project [45], [46]. Feature engineering consists of exploiting data in terms of messy data cleanup and transforming data from the raw format to another, more reliable form, thus enabling the selection and creation of features that make machine learning algorithms work better [45], [47].

1.2 Problem Statement

Most malware detection solutions use signature-based techniques [32], [33], [17], [34] to scan the malware and look for matching patterns in their database in order to identify them. The signature is typically fixed and can merely identify one instance of malware.

This procedure is unable to detect polymorphic malware due to their self-modifying nature. Earlier researchers attempted to address polymorphism by especially using API calls features [48], [49] and machine learning [50], [31]. API calls represent particular operations carried out by malware to access and utilise system resources [51]. However, this approach also caused many False Positives. The main gap relies on features extracted and the way in which they are used. This problem is still a challenge because an overwhelming number of newly advanced malware occur daily.

1.3 Research Objectives

1.3.1 Main objective

The main objective is to design a novel feature engineering approach for improving the performance of machine learning models associated with classification and detection of polymorphic malware.

1.3.2 Specific objectives

1. To design a novel feature engineering (NFE) approach based on static and dynamic features.
2. To develop a supervised malware classification model using NFE features.
3. To develop an-early stage malware detection model, based on Deep Learning, using NFE features.

1.4 Research Questions

1. Can features extracted using advanced static and dynamic analysis techniques help to better manage polymorphism?
2. How can polymorphic malware be effectively classified in their respective classes?
3. How can malware be detected early enough before harm is done?

1.5 Significance of the Research

Many security threats appear every day. These threats are malware that mainly come from the Internet. They affect the physical security of people and their valuable assets, including data and devices. This research aims to explore problems related to polymorphic malware and to propose effective solutions to classify and detect them. The proposed solutions will undoubtedly contribute to the antivirus sector in adequately developing powerful security applications. This work will also attract more interest for a continuously advanced research among the cybersecurity research community.

1.6 Contributions

Specific contributions are:

1. The research designed a novel feature engineering (NFE) approach useful for creating input features that will help to improve remarkably the performance of classification and detection models. NFE additionally provides a unique approach for computing feature importances, manipulating features, and representing the dataset. The main advantages of NFE are as follows:

    a) It provides a unique way of computing feature importances

    b) It selects the most discriminating features

    c) It generates new features whenever necessary to increase the predictive power of the model

    d) It reduces overfitting on both training and testing sets

    e) It improves modelling accuracy.

    f) It minimises the training time

2. The research proposed a supervised learning model for improved classification performance of polymorphic malware.

3. The research proposed a deep learning model for earlier detection of polymorphic malware.

4. The research provided a thorough discussion on security breach techniques, attack mechanisms and advanced obfuscations techniques.

5. The research provided a comprehensive analysis of polymorphic malware and their evolution, as well as an extensive comparison of gaps,and strengths and weaknesses of existing techniques.

6. The research provided a comprehensive analysis and discussion on the performance metrics and the bias that exists in some research conclusions.

1.7 Thesis Outline

This thesis is structured as follows.

- **Chapter 2:** In this chapter, we provide a comprehensive review of the literature on polymorphic malware analysis and detection techniques. We provide an extensive discussion on the structure of malware and their polymorphic techniques. We present an overview of the existing techniques that ostensibly addressed this problem and identified relevant gaps from the literature.

- **Chapter 3:** In this chapter, we present the methodology used in this work. Details of the datasets and approaches about proposed solutions are introduced.

- **Chapter 4:** In this chapter, we propose a novel feature engineering (NFE) approach for improving the performance of malware classification and detection models.We examine the key features that allow malware to perform their malicious actions. We present a feature-learning algorithm that efficiently selects the most significant features and is capable of generating new features whenever necessary, in order to produce a final feature set with strong predictive signals.

- **Chapter 5:** In this chapter, we propose a malware classification model. We demonstrate the impact of NFE features in improving the classification performance of the model.

- **Chapter 6:** In this chapter, we propose an early-stage detection model of malware which is capable of detecting malware as early as possible. Deep learning is used to implement this model. We demonstrate the tangible impact of NFE features in improving Deep learning predictive capabilities.

- **Chapter 7:** Finally, in this chapter, we succinctly summarize the results of this work and discuss our key findings. We present concluding remarks and make recommendations for further research.

Publications from this Thesis

1. E. Masabo, K. S. Kaawaase, J. Sansa-otim, and J. Ngubiri, "Improvement of Malware Classification using Hybrid feature Engineering," SN Computer Science 1 (1), 2019.

2. E. Masabo, K. S. Kaawaase, J. Sansa-otim, and J. Ngubiri, "A State of the Art Survey On Polymorphic Malware Analysis and Detection Techniques," ICTACT J. Soft Comput., vol. 8, no. 4, 2018.

3. E. Masabo and J. Sansa-otim, "Big Data: Deep Learning for detecting Malware," 2018 ACM/IEEE Symp. Softw. Eng. Africa, vol. i, pp. 20–26, 2018.

4. E. Masabo, K. S. Kaawaase, J. Sansa-Otim, and D. Hanyurwimfura, "Structural Feature Engineering Approach for Detecting Polymorphic Malware," 2017 IEEE 15th Intl Conf Dependable, Auton. Secur. Comput. 15th Intl Conf Pervasive Intell. Comput. 3rd Intl Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr., pp. 716–721, 2017.

5. E. Masabo, K. S. Kaawaase, J. Sansa-Otim, and D. Hanyurwimfura, "Integrated Feature Extraction Approach Towards Detection of Polymorphic Malware In Executable Files," Int. J. Comput. Sci. Secur., vol. 11, no. 112, pp. 2017–25, 2017.

Chapter 2

Literature Review

Outline: This chapter provides a comprehensive review of the literature on available techniques and tools that have been used to address the malware problem, especially polymorphic malware. Several techniques exist and each has its own strengths and weaknesses. Section 2.1 presents a general overview of malware in computer systems. Section 2.2 presents a discussion on different malware types and their operating mechanisms. Section 2.3 presents and discusses the common malware evasion techniques. Section 2.4 discusses in details the techniques used in malware analysis and tools used to achieve this are presented in Section 2.5. Section 2.6 presents some techniques that were recently developed by researchers for detecting polymorphic malware. A general discussion and comparison is also provided in this section. Finally, in Section 2.8, the research gives an overview of the findings of the literature and presents the gaps that need further consideration. This chapter is based on the work in [52]

The primary contributions of this chapter are: 1) To provide an analytical review of the most successful later works on polymorphic malware detection. 2) To retrieve the gaps that require further research. 3) To assess the adequacy of the most popular tools used to analyze polymorphic malware. 4) To identify the most promising studies that can serve as the point of reference for future research. 5) To provide recommendations and directions for future research.

2.1 Malware in Computer Systems

Malware is a specific kind of malicious code, which is destructive. It is just like any other regular software. However, it has a harmful intent such as denial of service, attempt to confidentiality, information stealing or potential abuse of data integrity [27]. To reasonably achieve its key objectives, malware can perform various operations such as file activity (open, read, delete, modify, move), Registry activity (open, create, delete, modify, move, query, close), Service activity(open, start, create, delete, modify), Mutex (create, delete), process activity(start, terminate), Runtime DLLs, Network activity(TCP, UDP, DNS, HTTP), etc. Malware types range from simple to more complex ones such as polymorphic malware. Malware can perform a self-execution locally or can be controlled remotely via the Internet.

Malware families include but not limited to spyware, adware, trapdoor, trojan horse, sniffers, spam, botnet, logic bomb, Worm, Virus, key-loggers, ransomware, backdoors, adware, spyware, etc. [2]. Figure 2.1 below shows the example of a simple virus infection.

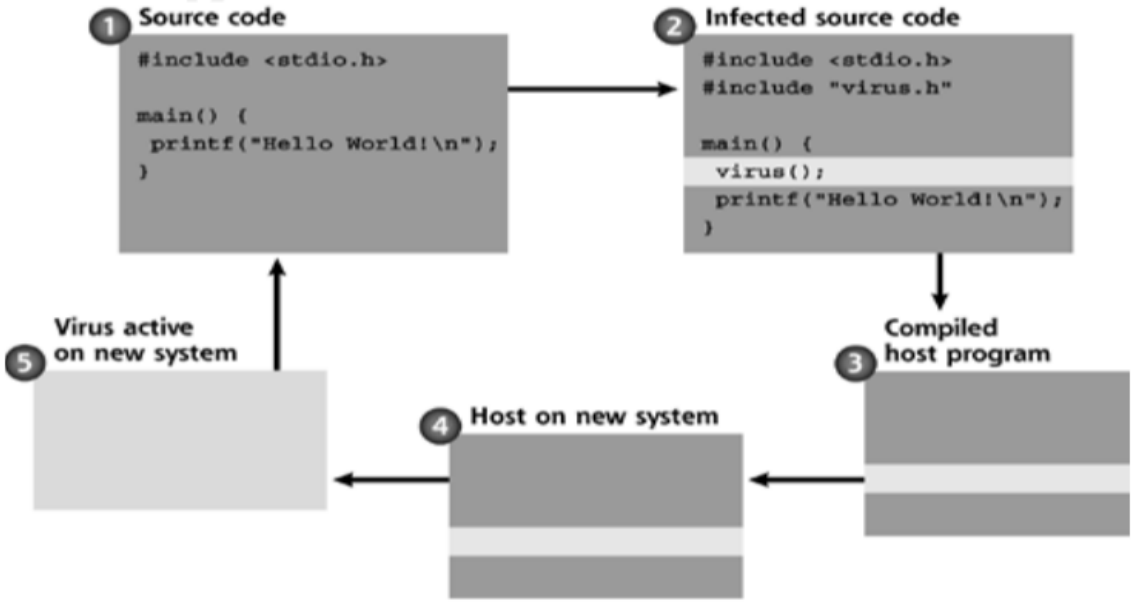

Figure 2.1: Infection by a code virus [53]

2.2 Malware Types and Operation

.

- *Virus*: This is a malware capable of hiding itself inside a benign program and can produce multiple copies of itself. It can spread these copies into files or other programs.
- *Ransomware*: This is a specific type of malware that locks PC screen with misleading information and scare the victim by strenuously urging him/ her to pay a ransom fee as a condition to regain access of the system.
- *Trojan horse*: This malware disguises itself as a benign program and reluctantly persuades the alleged victim to install it. Once installed, it carries out some destructive actions already hidden in its payload.
- *Rootkit*: This malware possesses the capability to elude detection by staying concealed on the system where it is installed. It hides its services from being visible in the list of system running processes.
- *Backdoor*: This malware is capable of bypassing normal authentication mechanisms by compromising the network or Internet connection. It creates hidden processes to allow any access whenever needed in the future.
- *Adware*: This malware places an unwanted advertisement to the user during the software installation process, with the purpose of generating income on the behalf of their author.
- *Exploit*: This type of malware exploits a particular vulnerability on the target system.
- *Key loggers*: These malware record keyboard strokes, thus stealing sensitive information regarding passwords, credit cards or other various credentials. The stolen information is forwarded to their authors.
- *Spyware*: These malware steal user's information without their knowledge and utilize it to conduct further malicious actions.
- *Potentially Unwanted Program*: This is about all programs that the user may find unwanted. They can be a threat to the system's security.
- *Worm*: a worm is like a virus by design structure. It is indeed considered as a subclass of a virus. It can spread from device to device by itself without any help of a particular person.

- *Advanced persistence Threats (APT)*: these malware employ sophisticated techniques to exploit vulnerabilities in systems. They can be managed remotely and have the capability to persistently strike a specific target.
- *Crypto malware*: also known as crypto ransomware or data locker, they prevent data access from the targeted system. They use encryption to restrict users from accessing their data and urge the user to pay for the encryption key.
- *Locker malware*: they are primarily designed to allegedly lock and forcibly prevent access to the device interface, largely leaving data untouched.
- *Zeus*: these malware are meant to steal banking information by collecting keystrokes logging and stilling information filled in forms.
- *Shadowbrokers*: these malware are meant to exploit system vulnerabilities and can be monitored remotely. They use plugins to capture webcam and microphone output, log keystrokes as well as accessing other drives for surveillance purposes.

Malware can be distributed through numerous channels such as:

- Drive-by download: unintentional downloading of malware from the Internet.
- Unsolicited email: embedded links or unknown attachments in emails.
- Physical media: malware spread using devices such as USB or other removable media.
- Self-propagation: The malware can spread itself through networks or from one computer to another.

2.3 Malware Evasion Techniques

2.3.1 Polymorphism

The first polymorphic malware was developed by Mark Washbrun in 1990 and was called 1260 virus [54]. Polymorphism is a stealth technique used by malware to create an unlimited number of new different malware variants [54], [27], [23] in order to harden analysis and detection. Their code is constantly changing at any infection [29], [55]. In general, polymorphic malware contain invariant bytes that are constant across all their created instances as well as variant bytes that change values at every infection [38], [56].

In order to obstruct the analysis process, polymorphic malware use code obfuscation techniques [48] such as packing, encryption and junk code insertion or substitution [9], [57], [24], [58], [54]. Packed files have different hash values. Packing is principally used by malware in order: 1) to harden the reverse engineering, 2) to harden a dynamic debugging process, 3) to reduce execution file size, 4) to harden static malware disassembly, 5) to bypass malware detection and 6) to defeat malware researchers [9]. The structure of a packed file is elucidated in Figure 2.2. Figure 2.3 shows the infection process of a polymorphic malware by which a mutation engine is used by a polymorphic generator to produce new instances.

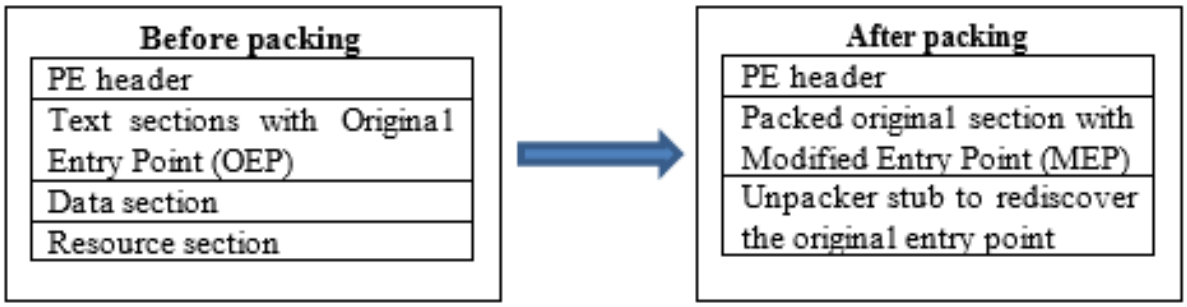

Figure 2.2: Packing and unpacking process

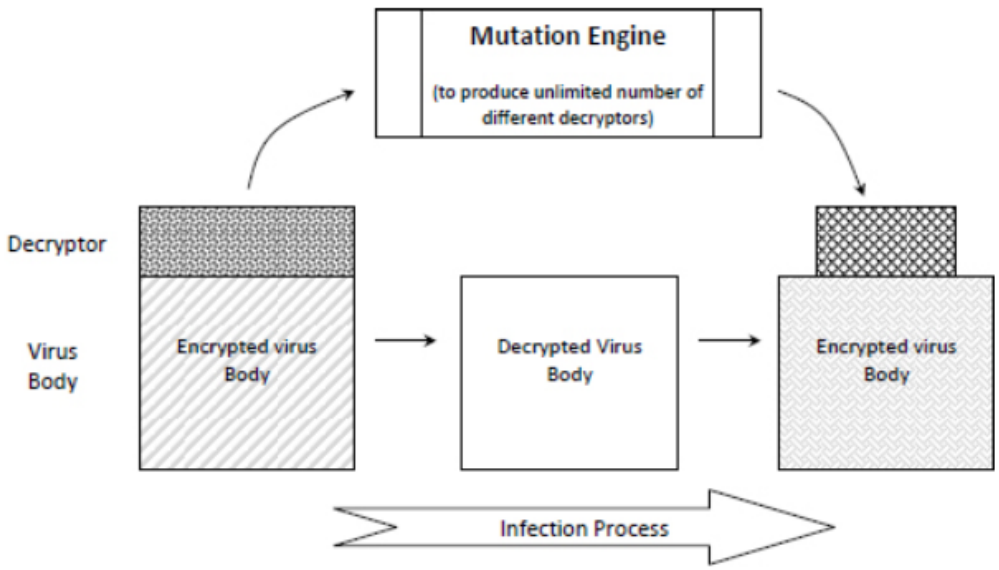

Figure 2.3: Polymorphic malware infection process [54]

Two categories predominantly characterize the segments of polymorphic malware, namely the invariant and variant bytes [38], [19], [59].

Invariant bytes

- *Protocol framing* is in charge of branching down the execution path of the code with a string that remains unchanged across all instances of polymorphic malware.
- *Return address* or *function pointers* are values that are used in overwriting a jump target to redirect execution.
- *Exploit code* is a set of invariant bytes in charge of malicious activities. It ensures that malicious behaviours are identical in all newly created malware variants [60]

Variant bytes

- *Encrypted code* or payload contains malicious codes that keep changing at every infection.
- *Decryption module* is in charge of decrypting the encrypted payload and control is passed to malicious code to start execution. These modules are obfuscated in numerous variants of polymorphic malware.
- *Decryption key* is required to decrypt the payload because multiple keys are generated by a polymorphic engine in order to allow the creation of multiple malware instances.

2.3.2 Obfuscation

Obfuscation makes malware harder to detect [61], [28]. It transforms a malicious code to a new different version while keeping the same functionality [54], [61], [59], [62]. Initially, this technique was used for protecting software developer's intellectual property. Later, it has been utilized by malware creators to neutralize detection and analysis efforts[61], [54], [63], [64], [34]. The obfuscated code is different from the original one and is hard to understand while trying to conceal malware internal purposes [28]. Obfuscation adds fewer important instructions or garbage to existing code to change its structure or appearance and nevertheless maintain the same behaviour [61], [65]. The binary sequence of malicious code is modified without affecting the original functionality [50], [66]. Obfuscation techniques include instruction replacement, instruction permutation, variable/register substitution, junk/dead code insertion and code transposition [61].

Dead code insertion

Figure 2.4 and 2.5 respectively show the part of the original code and dead code insertion examples.

To inject a dead code, a Non-operation (NOP) code is injected into the original code. A NOP sled is a lengthy sequence of instructions often included in the shellcode[1] as part of an exploit in order to increase the probability of an exploit to be successful [67]. Code analysis task becomes incredibly complicated when the code is nested as shown in Table 2.1.

Table 2.1: Non-operation code injection [54]

Instruction	**Operation**
ADD Reg, 0	Reg <– Reg + 0
MOV Reg, Reg	Reg <– Reg
OR Reg, 0	Reg <– Reg | 0
AND Reg, −1	Reg <– Reg &−1

Instructions in Table 2.1 can change the status flag registers, but no change is made on the value of the operand register [54]. It is noticeable that adding a zero to a register or assigning a register value to itself doesn't have any effect on the execution results [54].

[1]Shellcode is a payload of raw executable code. It is a code that attracts attackers to have interactive shell access on the compromised system. Shellcode also describes any piece of self-contained executable code.

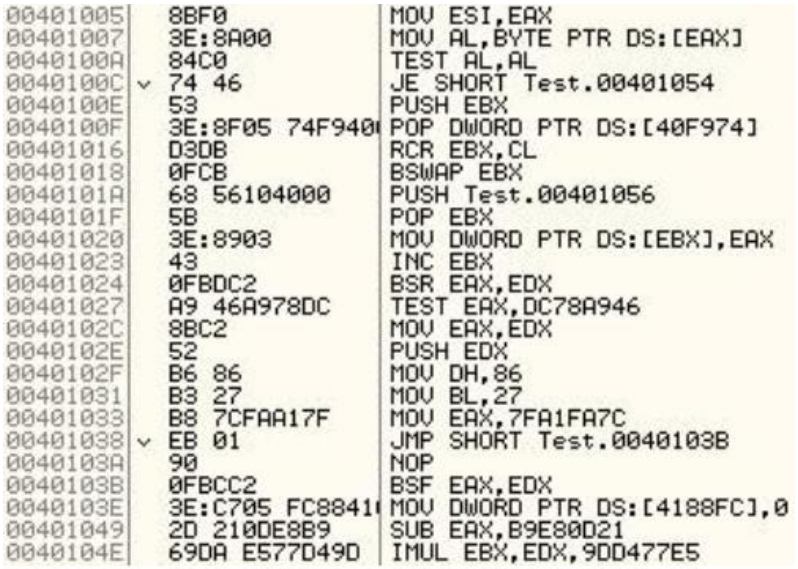

Figure 2.4: Original code [61]

Figure 2.5: Dead code insertion [61]

Register exchanging

This technique involves switching registers or memory variables of different malware variants [65], [68], [61] while keeping the identical behaviour [66], [34]. Two forms of *W95.Regswap* malware are shown in Figure 2.6 and 2.7 respectively. The functionality remains the same, although registers have been interchanged [69], [70]. This can comfortably defeat signature-based detection systems. The implementation of these techniques is shown in Table 2.2.

Table 2.2: Polymorphic techniques

Original code

```
#include
int main() {
int a,b,c;
c=1;
a=0;
b=6;
while (c){
a= a+b;
c++;
}
}
```

Variable renaming

```
#include
int main() {
int m,n,p;
p=1;
m=0;
n=6;
while(p){
m=m+n;
p = p+1;
}
}
```

Statement reordering

```
#include
int main() {
int a,b,c;
b=6;
a=0;
c=1;
while(c){
a= a+b;
c++;
}
}
```

Statement replacing

```
#include
int main() {
int a,b,c;
c=1*1;
a=0*c;
b=6/c;
while(c){
a= a+b;
c++;
}
}
```

Junk code insertion

```
#include
int main() {

int a,b,c;
c=1;
a=0;
b=6;
bool h=true;
if(h)
{
while(c){
a= a+b;
c++;
}
}
}
```

Spaghetti code

```
#include
int main() {
int a,b,c;
goto T;
N:
while(c){
a= a+b;
c++;
}
goto F;
T:
c=1; a=0; b=6;
goto N;
F:
}
```

Binary Code Sequence	Assembly Code	
5A	pop	edx
BF04000000	mov	edi,0004h
8BF5	mov	esi,ebp
B80C000000	mov	eax,000Ch
81C288000000	add	edx,0088h
8B1A	add	ebx,[edx]
899C8618110000	mov	[esi+eax*4+00001118],ebx

Figure 2.6: Variant 1 of "W95.Regswap" malware [54]

Binary Code Sequence	Assembly Code	
58	pop	eax
BB04000000	mov	ebx, 0004h
8BD5	mov	edx, ebp
BF0C000000	mov	edi, 000ch
81C088000000	add	eax, 0088h
8B30	mov	esi, [eax]
89B4BA18110000	mov	[edx+edi*4+00001118],esi

Figure 2.7: Variant 2 of "W95.Regswap" malware [54]

Instruction replacement/substitution

Instructions are replaced by equivalent ones as illustrated in the following example, where all given instructions are equal as they set the register eax to 0.

```
MOV eax, 0
XOR eax, eax
AND eax, 0
SUB eax, eax
```

Instruction permutation/reordering

The sequence of instructions is reordered randomly with the purpose of making the code binary sequence look different in multiple instances of the same malware [54], [61]. This reordering does not affect the functionality of the malware. This technique is able to generate n! different instances, where n is the number of subroutines. An example of instructions permutation is given in Table 2.3.

Table 2.3: Instruction reordering

Order 1	Order 2
mov eax, 0F	add esi, ebx
push eax	mov eax, 0F
add esi, ebx	push eax

Code transposition

This technique reorders or shifts the binary code sequences and uses unconditional or conditional branches to recover the original program execution flow [34], [54], [61] as shown in Figure 2.8.

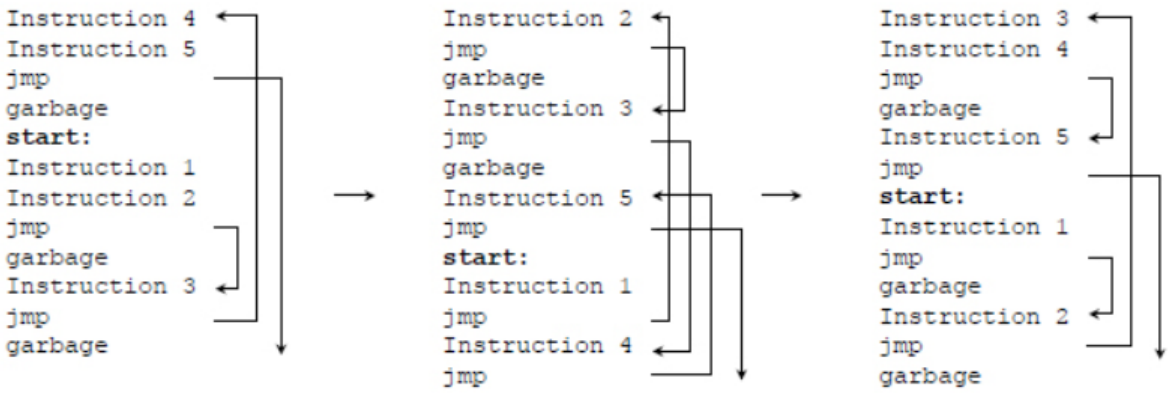

Figure 2.8: Binary code shifting [54]

Code integration

This technique is very sophisticated as the malware first decompiles the target program into smaller objects and secondly adds itself to them, then finally reassembles the integrated code to create a new file [61], [28].

Virtualization obfuscation

Instructions are virtualized to hide from the analysis. The malware includes a virtual machine module that is used to interpret the virtualized code [54].

2.3.3 Metamorphism

Metamorphic malware do not necessarily need to have encryption part [54]. They use the mutation engine to change the body of the code, thus creating new copies with

different structures, while maintaining the same behavior [71]. Their mutation engine has a disassembler, a code analyser, code transformer and an assembler. The disassembler converts the code into assembly instructions. After the code is analysed by the code analyser, the code binary sequence is changed and reorganized by the code transformer in order to make it look different. Finally, a new malicious mutated assembly code is produced by the assembler [72].

2.4 Malware Analysis Techniques

An analysis is a process that allows an analyst to get a clear picture of the malware structure and functionality [73], [74]. During analysis, different key features are extracted [75] and provide knowledge about the functionality of the malware. Informal categories of analysis include vulnerability analysis [76], source code analysis [37], [76], and behavioural analysis [37], [77]. Similarity analysis [28], [78], [79], [80] is undertaken to ensure that the malware is a variant of an existing one. Analysis task is very important and is the first thing to be done prior to developing reliable detection systems. There exist two predominant analysis techniques, namely Static analysis and Dynamic analysis [37], [32].

2.4.1 Static analysis

Static analysis is a process whereby information about the malicious program is extracted without executing it [34], [26], [10]. This makes static analysis safer than dynamic analysis as malware is not run [31]. It is classified in two categories such as basic static analysis and advanced static analysis. The basic static analysis is straightforward and quick. It gives basic information about the malicious program such as its version, file format, any suspicious imports, etc. It is not very effective because it can easily miss important details [81]. Advanced static analysis deals with code/structure analysis where the knowledge of assembly language, compiler code, and operating system concepts is required [81]. Malware functionality is analyzed by inspecting the internal code of the malware [25]. This analysis is able to provide information regarding malware identity, passwords, libraries, URL, programming language, etc. [25],[82]. Function routines and mutants can be identified [35]. With advanced static analysis, the code can be disassembled and decompiled [26]. However, static analysis is unable to handle packing and obfuscation. It reveals the presence of packing, but the binary needs to be unpacked for the success of static analysis [13], [29], [83], [82]. In addition to what has been

explained above, the following information can also be revealed by static analysis [84], [73], [37], [85], [86], [55]:

File fingerprinting: the cryptographic hash code (e.g. md5) is computed to know if a binary has not been modified.

Hash coded string extraction: this process helps to extract human-readable strings embedded in the compiled binary. With this information, a conclusion is drawn about some functionality of the binary. Imported and exported functions are revealed.

File format: The metadata of a file format is investigated to extract pertinent information such as file type, file format and compilation time.

Antivirus scanning: if a binary is hitherto known, it will be detected by one or more anti-malware programs.

Packer detection: due to obfuscation structure (e.g. encryption and compression) of that malware, suitable tools such as PEID [81] and Detect it Easy (DIE) can identify packer and compiler information.

Disassembly: This process consists of reverse engineering machine code to assembly language which is understood by a human. Tools like IDA pro [81] can be used. The generated assembly code helps the analyst inspect further the logic of the code and gather more information about the intent of the malware [87]. OllyBdg [81] tool is used in this process.

Static analysis provides two fundamental advantages. Firstly, it is safe during the analysis process, because malware doesn't have to be executed. Secondarily, it provides deeper information about execution paths of the malware. In addition to that, the static analysis moreover presents some disadvantages such as requiring much experience, consuming a lot of time, and not being able to handle packed files. It also gives ambiguous results when analysing malware that use self-modifying techniques [84].

2.4.2 Dynamic analysis

Dynamic analysis is the process of analyzing a malicious program by first executing and monitoring its runtime functionalities [34], [13]. Malicious behaviours are monitored and logged [84], [73], [37], [77], [40], [78]. During this process, the malware unpacks itself, and the changes it induces on the system are also observed. Dynamic analysis is undertaken in a virtual environment in order to ensure maximum protection of the host machine [26], [48], [10], [88], [89]. The basic dynamic analysis consists of observing the basic behaviours of the malware such as process creation, file activities or registry activities [81]. On the other hand, the advanced dynamic analysis undertakes a profound examination of the

internal state of a running malicious program. It employs advanced debugging techniques to single step through the malicious code and caries out a deep internal inspection to get more comprehensive information on the malicious behaviours [81]. The code is analyzed at runtime and any hidden code through packing is revealed [29], [55]. The identity of the malware is dynamically identified. Function calls, parameter analysis, and information flow are all visualized [25]. Dynamic link libraries (dll), processes and file activities are revealed [26], [90]. Dynamic analysis helps detect polymorphic mutants as well as packing and obfuscation attributes [13], [83], [56]. Memory analysis can be performed [10]. The interaction between malware, file system, processes, and the network is inspected [10]. However, dynamic analysis is computationally expensive and requires a lot of system resources [48], [86].

Advantages of dynamic analysis include:

- The investigation of live malware behaviours at runtime
- The handling packed files, and
- Automated analysis for large malware corpus

,

The disadvantages of Dynamic analysis are:

- The possibility of missing out some execution paths when the malware being monitored becomes dormant.
- Risk of spreading the infection by a network capable malware from the virtual environment to the host system, and
- The impossibility to monitor the malware when it incorporates the capabilities of refusing to execute when it is run in a virtual environment.

In addition to observing runtime behaviours of the malware, dynamic analysis can allow an analyst to capture the difference between two system states. An initial state is recorded before running the malware and another state is recorded after malware execution. A comparison will be undertaken and differences highlighted to assess the impact of the malware on the system [91].

Both static and dynamic analyses are essential techniques needed for a successful malware investigation task towards developing detection approaches. Table 2.4 highlights the comparison between both techniques.

Table 2.4: Comparison between both Static and Dynamic analyzes.

Static analysis	**Dynamic analysis**
A binary is not executed prior to analysis.	A binary must be executed prior to analysis.
Does not necessarily need a virtual environment.	It needs a virtual environment set up before analysis.
Can't handle packed binaries before manually being unpacked.	Packed binaries are automatically unpacked [54].
It is hard but can reveal almost the full code coverage or global view of a binary.	Reveals the only path execution of modules that are running [92].
Low false positive rate	High false positive rate.

2.5 Malware Analysis Tools

2.5.1 Static analysis tools

PEiD: This tool is used to detect the type of the packer and the compiler used to build an application. This can help the analyst identify a suitable program to unpack the binary. Entropy and checksum can also be calculated.

UPX: This is an ultimate packer for executables that helps in compressing/packing and decompressing/unpacking executable files. Packing process makes malware hard to analyse and detect. It shrinks the size of malware binaries. The packed program allegedly hides the original data and creates an unpacking stub that is called by the operating system when the malware attacks. The unpacking process helps in rediscovering the original executable and transfers execution to the original entry point. This tool can unpack malware packed by UPX or other packing tools as well.

IDA Pro: This is an interactive disassembler used to reverse engineer malware binaries and construct execution maps while discovering and analysing vulnerabilities. IDA is the choice of many malware analysts. Tasks undertaken by IDA include code disassembly, obfuscation traces discovery, function parameters analysis, function calls analysis, operation codes analysis, strings analysis, persistence tracking, privilege escalation analysis, DLLs injection analysis, Runtime DLLs analysis, process replacement analysis, code noise patterns analysis, rootkit functionalities analysis, Anti-debug analysis , Kernel mode activity analysis, User mode activity analysis, etc.

2.5.2 Dynamic analysis tools

Process Monitor: This tool provides real-time monitoring of the system, and visualizes all the events as they are happening. Running processes, system calls, file system, Network activity, registry activity, API calls, Mutex and Self-modifying code traces can all be monitored.

Dependency Walker: This tool helps explore Dynamic Link Libraries (DLLs), imported functions, exported functions and monitor interactivity between running processes and DLLs.

Regshot: This tool takes system snapshots before and after the malware is executed and produces a log of registry manipulations. Both snapshots are then compared to observe registry modifications. Registry modifications and system changes give a clue of malware behaviours.

ApateDNS: This tool helps the analyst to capture DNS requests made by malware without necessarily being connected to the Internet. It imitates DNS responses to a given IP address. It can display all received results in a hexadecimal format.

Wireshark: This is a packet sniffing program that helps to grasp how malware performs network communications. It can intercept and log the traffic that passes over the network when the malware is running. Wireshark provides packet details which enable the analyst to do the in-depth packet streams analysis.

INetSim: This is a tool suitable for simulating ordinary Internet services. It can provide fake services that allow the analyst to analyze the network behaviours of the malware. Services that are emulated by InetSim are but not limited to HTTP, HTTPS, FTP, IRC, DNS, SMTP. InetSim can also record all inbound connections and requests made by the malware.

2.5.3 Sandboxes

Sandboxes are tools used to analyze the malware using both static and dynamic analysis techniques. They provide a detailed report at the end of the analysis. Malware are executed within the sandbox and all side effects remain within the sandbox system without affecting the host system[89].

Anubis[89]: This is an automated malware analysis tool that helps monitor API functions, system service calls, function parameters, etc.

CwSandbox[89]: This is an automated malware analysis tool that helps monitor API calls, system calls, Registry manipulations tracking, Network traces, Operating

System interactions.

GFI sandbox: Automated malware analysis tool to monitor API calls.

JoeBox sandbox [93]: This is an automated malware analysis tool. Its key features include file system analysis, registry activity analysis; system calls analysis, rules generation, memory dump analysis, packing analysis and strings analysis.

Cuckoo sandbox: This is an automated malware analysis tool. It helps in analyzing file activity, registry, system calls, API, memory dumps, packing, network activity, etc. This tool also has advanced memory analysis capabilities that help in discovering deeper hidden malware functionalities.

2.6 Malware Detection Techniques

Malware detection is simply a technique valuable for identifying malware in all targets[94], either computers, applications or cyber-physical devices [95], [96], [1]. Polymorphic malware has many variants of the same functionality as discussed in previous sections. In order to detect them, similarity analysis is undertaken.

Drew et al.[97] proposed a detection approach based on sequence classification methods. Based on biological gene sequence mutations. Authors employ a similar concept to assess similarities in mutations of malware. They utilized a tool developed for gene classification called "The Super Threaded Reference-Free Alignment-Free N-sequence Decoder" (STRAND) for classifying polymorphic malware. This method was evaluated on a 500 GB malware dataset provided by Kaggle- Microsoft Malware Classification Challenge (BIG 2015) [98]. Features were developed for STRAND classifier from given sample bytes, where hex values were considered and missing values removed. This produced a single string/strand sequence containing only all hex contents extracted from malware file. Sequences of words of length k or k-mers were generated by strand. Word length of 10 characters and 2400 minihash values were used. Jaccard similarities were computed. Training time was less than 7 hours and strand classification accuracy was 95%, of which 9 classes of polymorphic malware in the given set were detected.

In their expanded version of work [97], Drew et al. in [99] developed an approach that detects polymorphic malware using sequence classification methods and ensembles. STRAND algorithm was used for the classification task. In this work, they also used a malware classification dataset from Microsoft 2015 challenge. They used bytes (.bytes) data and assembly data (.asm) features as provided by Microsoft. Ensembles were created by using both the asm and bytes models from Strand. The minihash technique was used

to detect similarity. Minihash scores of created models were computed, and the overall detection accuracy became 98%.

Selamat et al. [58] proposed an easy detection technique for detecting polymorphic malware based on dropped files. To evaluate their technique, authors in [58] executed malware in a virtual environment. They used process explorer to observe malware behaviours. Three different types of malware pythium2.exe, kax.exe, ieg.exe were used in the experiment. Each was executed twice. For the first and second executions, each malware could create a child process. It was observed that those created subprocesses during first and second executions were different. The conclusion was drawn that this is also a good indicator of polymorphic behaviour.

Fraley et al. [39] proposed a detection approach that utilizes topological feature extraction using static and dynamic analysis techniques. The proposed method relies on data mining techniques as a means to achieve classification scalability. This method uses IDA pro and cuckoo sandbox for feature extraction. Experiments were conducted on data set of 3637 samples of which 2400 were clean, 800 were malicious and 437 were unlabelled. They used function call graphs to trace instruction patterns. Belief propagation was used to uncover the properties of malicious files. Twenty features (12 structural and 8 behavioural) were collected and used to make a complete feature set. Few examples of extracted features include MOV, ADD, LEA, file size, and compiler type. The created feature set was converted into the ARFF data format compatible with the Weka data mining tool. Ensemble bagging algorithms were used in Weka for classification and cross-validation with 10 folds was used for validation. The overall accuracy was 99.9 % where low false negative is 0.001.

Naidu et al. [22] proposed a detection method that involved the extraction of syntactic structures of the malware from its hex code. It helped in identifying the piece of malicious code and the variant type of malware. The Smith-Waterman algorithm was used to identify variants. Experiments were carried out using two variants of JS.cassandra malware. Hex dumps were extracted using sigtool from ClamAV. Hex bytes were then converted to binary code and from binary to DNA sequences. The DNA sequences were input into JAligner tool where Smith-Waterman algorithm (SWA) was implemented. Common substrings or patterns were extracted Pairwise local alignment (PLA) was performed using SWA with different substitution matrices. Only substrings with the highest percentage of identities and similarities were extracted after PLA. 161 substrings were retained through six substitution matrices. These 161 substrings have been considered as meta-signatures and used to detect all known polymorphic variants of

JS.Cassandra. T-Coffee tool was used to perform multiple sequence alignment on the generated meta-signatures and generated consensuses. Rules were generated using a data mining classification algorithm called PRISM and this allowed the generation of 47 super signature substrings. Therefore, those 47 super signatures and 161 meta signatures were converted back to hexadecimal and tested against JS.Cassandra. This method was able to detect 100% of JS.Cassandra malware and all its known variants.

Harmonen in [9] proposed a client-server architecture for identifying polymorphic malware. This method has two advantages: storing remotely virus information databases without allowing individual client updates and protecting the client application as well. The client application is deployed on the target system and keeps monitoring the system of interest. Suspicious files are investigated. File hash and file metadata information are sent to the server. The server compares the received data with the one in its database. When multiple hash values are received from different clients and the corresponding metadata is the same, the file is identified as a polymorphic malware. The server is able to determine if the file is benign, known polymorphic malware or a new variant.

Naidu et al. [100] proposed a polymorphic malware detection approach that is based on automatic signature extraction. Needleman-Wunsch and Smith-Waterman algorithms are used for developing a detection mechanism. This method is built on top of same author's previous work [22] which is discussed above. Needleman-Wunsch algorithm was used together with Smith-Waterman algorithm. JS.Cassandra and W32.Kitti polymorphic malware were used in experiments. Hex dumps were extracted and converted to DNA. Pairwise sequence alignments were performed, meta and super signatures were generated. Finally, this method was able to detect 100% of JS.Cassandra and W32.Kitti known polymorphic variants.

Sharma et al. [101] proposed an approach that uses signatures and pattern matching to detect polymorphic malware. Experiments were done using tools such as SciPy, Numpy and Python. A python module called pydasm is used to extract features and its report recorded. Opcodes are extracted from the given instructions and are used for further processing. KNN algorithm is implemented and the method was able to detect polymorphic malware. The detection accuracy was unprovided.

Eskandari et al. [60] proposed a technique that combines two approaches, namely token extraction and multiple sequence alignment for achieving a high accuracy and noise tolerance flexibility. To evaluate their method, the authors used DARPA 1999 intrusion detection system dataset. They extracted tokens from suspicious flows and eliminated irrelevant parts of suspicious flow. They generated Simple regular expression (SRE)

signatures and applied multiple sequence alignment. Tests were done on DARPA 1999 intrusion detection system dataset, polymorphic versions of CodeRed II, Apache-Knacker, ATPhttpd and BIND-TSIG exploit malware. This method was able to analyze inherent similarities of samples, thus detecting successfully polymorphic malware.

Ahmadi et. Al [34] proposed a malicious files detection approach that is based on behavioural sequential patterns. API calls behaviour features were extracted using dynamic analysis and API calls log was generated. Initial dataset was then created from log data by only considering the repetitive patterns. Feature selection was conducted using Fisher score algorithm. Support Vector Machines (SVM) and decision tree algorithms were used for malware classification. This method was evaluated on a sample of 806 malware and 306 benign files. A detection accuracy of 95% was successfully achieved.

Kaur et al. [17] developed a hybrid anomaly and signature detection system that was used to detect zero-day polymorphic worms in an active network flow. The system architecture has three components such as Suspected Traffic Filter(STF), Zero attack evaluation (ZAE) and Signature generator(SG). Suspicious traffic is collected using STF where known malware are blocked and logged. Zero-day attacks are redirected to the honeypot for further analysis. STF module is composed of two components: Honeynet and IDS/IPS to capture and compare the network traffic flow. The Longest Common Prefix (LCP) algorithm is used in the comparison process. When IDS/IPS is found to have ignored an attack that was logged by honeynet, it means it's a new unknown attack. Analyses are done by ZAE component by which malicious strings are extracted such as NOP sleds, decryptor, shellcodes and return addresses. The SG module generates content based signatures by using invariant bytes found in a polymorphic malware. The method has been tested on a sample of 15435 packets of which 734 were polymorphic. The detection rate was 96% with almost zero positive rates.

Arshi et al. [48] proposed a behavioural detection approach based on machine learning. They focused on malware classification and clustering. 1270 malware samples of different format — pdf, executables, HTML, zipped and jpeg were analyzed. Logic Model Tree and K-Means algorithms have been used for the task of classification and clustering respectively. The results show that 18% of malware analyzed had network capabilities to connect to the outside world, while 82% aimed to corrupt the system locally. Malware have also been grouped successfully according to their file format types.

Meng at al. in [102] proposed a malware classification model called MCSMGS which combines deep learning with malware static genes based on API call sequences. Features were extracted as gene sequences. A convolutional network was constructed for analysis.

Their experiments achieved an accuracy of 98% on a dataset composed of PE files downloaded from VX heaven repository [103].

Yuan at al. in [42] proposed a multi-stage analysis technique that uses deep learning to classify or detect malware. Their experiment gave an accuracy of 94.83% and 94.66% F1 score. They stipulated that Deep learning can perform well even though it runs slowly.

Bojan at al. in [104] proposed a deep learning method for classifying malware through system call sequences. They constructed a convolutional neural network as a means to extract best features for classification. n-gram features were extracted. Their model achieved 85.6% on precision and 89.4% on recall.

Kolosnjaji at al. in [105] developed an adaptive semi-supervised malware classification technique that combines advantages of static and dynamic analyses to get better results. Samples used were collected from Virus Share, maltrieve [106] and other private repositories. Execution sequences were collected using Cuckoo sandbox and other features were obtained using PEInfo and Yara. Their developed semi-supervised model achieved an accuracy of 97.5%. It outsmarted supervised learning model which achieved 96.9%.

Rhode at al. in [107] created a recurrent neural network approach. This was used to detect if a file was malicious or benign by utilizing a small snapshot of behavioural data, instead of considering the post-execution log report. Their dataset was composed of 2345 malicious files from Virus Total, 2286 benign files from windows and 2876 ransomware from Virus Share. Their approach could achieve an accuracy of 94% within the first five seconds of execution.

Shibahara at al. in [108] proposed a dynamic malware analysis approach based on deep learning. The purpose of this approach was to increase the efficiency of dynamic analysis by determining when to stop it depending on the inconsistent change of malware communication purpose. Their experiments reduced the analysis time by 67.1% while keeping the overall coverage of collected malware to 97.9%. The dataset was composed of 29.562 malware samples.

Huang at al. in [109] proposed a Multi-Task Neural Network for Dynamic Malware Classification approach named MtNet. They used deep learning to classify benign from malware files. They trained the model on 4.5 million samples and achieved a classification error rate of 0.358% and low false positives under 0.07%.

2.7 Findings in the literature

As presented in previous sections,the problem of polymorphic malware has been widely addressed using different techniques. Some studies used machine learning while others also tried to address it using bio-informatics DNA techniques such as Multiple Sequence Alignment, Needleman-Wunsh, Smith-waterman, Strand gene sequence classifier, etc. Some studies use more than one algorithm to build models and select the most reliable one. The comparison of selected techniques is given in Table 2.5. Figure 2.9 highlights the algorithms used by selected detection techniques.

To evaluate the performance of their approaches, most researchers used accuracy as the main performance metric. Accuracies achieved were in the range of 90% to 100% as shown by Figure 2.10. Some other studies used also other performance metrics such as log loss, TPR, FPR, F-score, etc. as shown in Figure 2.10 and in Table 2.7.

Detection rates of 100% are subject of further exploration because it can simply be an over-fitting situation. This can happen when the dataset used is very small or the quality of the dataset itself. Table 2.6 and Figure 2.11 give detailed information about the datasets used in the presented techniques. Another observation is that during analysis, some studies extracted either static features or dynamic features, while others considered a hybrid of both types of features as shown in Table 2.8. Either can lead to good results in terms of performance metrics. However, an optimal detection system could use hybrid features and achieve better results [110] by minimizing the disadvantages of static and dynamic analysis while maximizing their advantages.

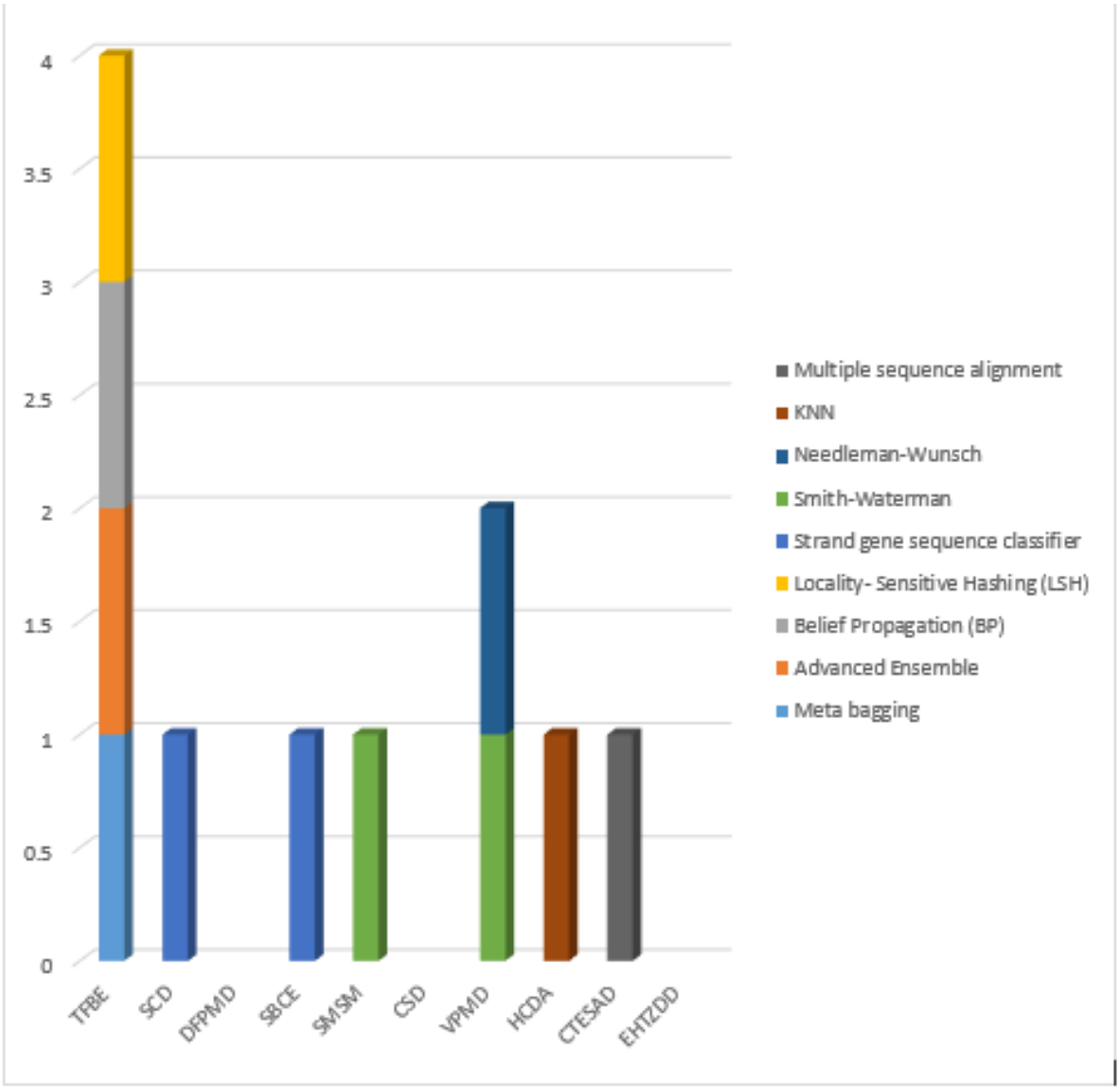

Figure 2.9: Algorithms used by selected techniques

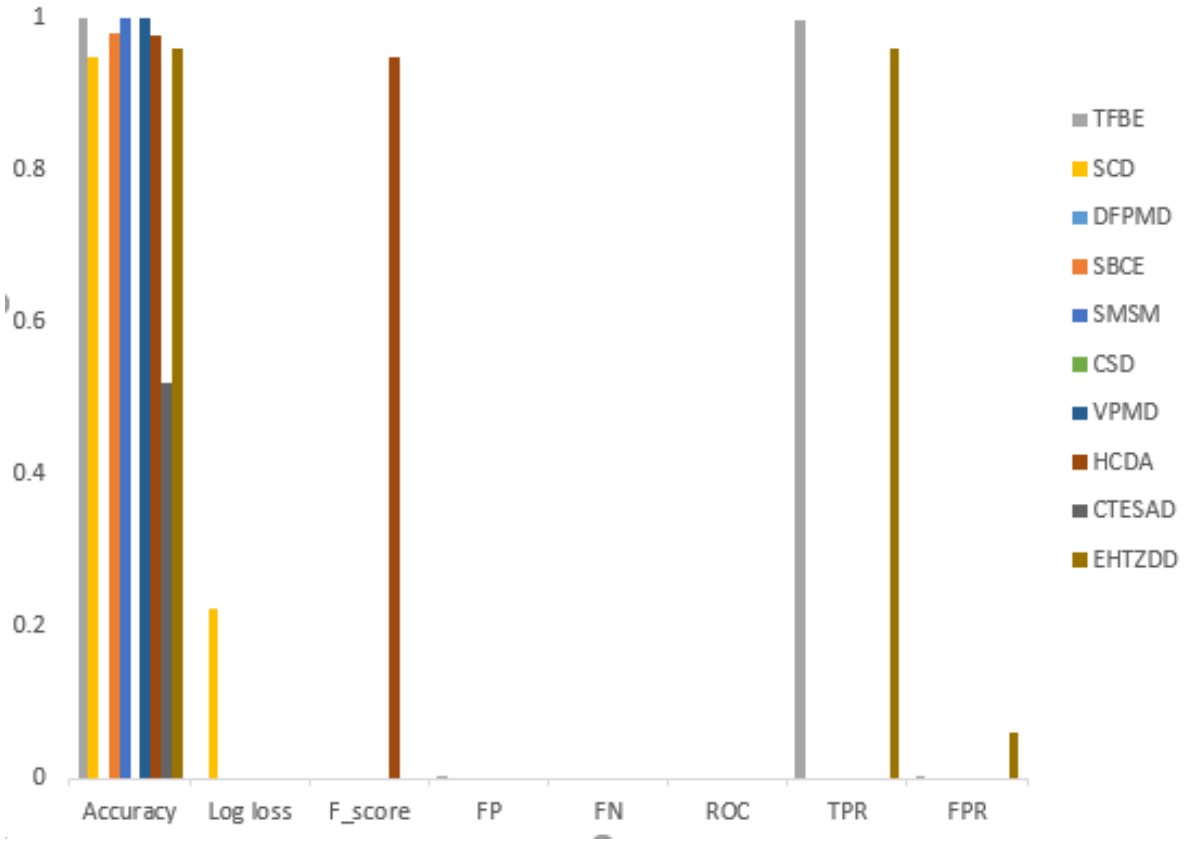

Figure 2.10: Performances of selected techniques

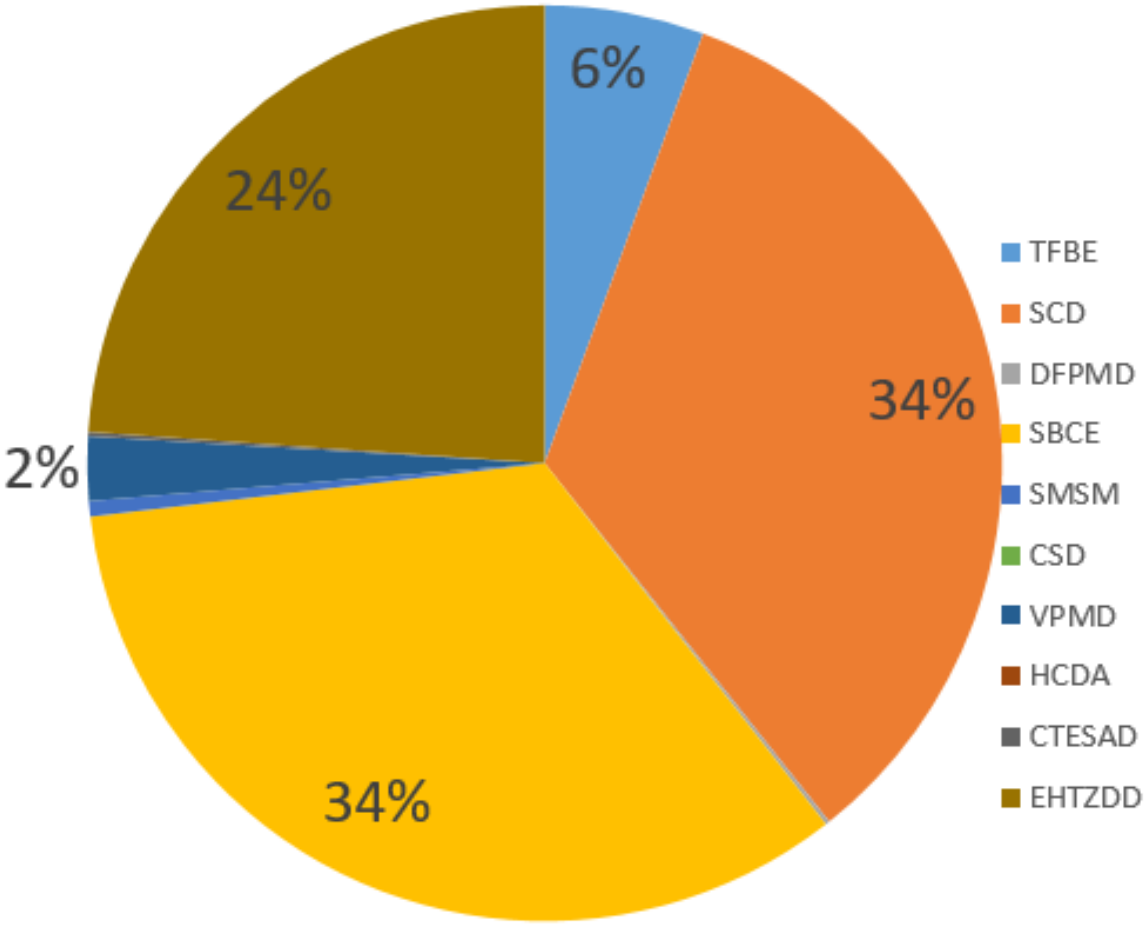

Figure 2.11: Number of dataset samples of the selected techniques

Table 2.5: Comparison of selected detection techniques

Technique	**Code**	**Contributions**	**Limitations**
Topological features based extraction [39]	TFBE	A novel approach that extracts, analyzes and combines multiple high factors to detect polymorphic malware. Faster and accurate for small samples	Need to be evaluated on a larger dataset.
Sequence classification detection [97]	SCD	A novel approach to detect polymorphic malware by using biological gene sequences and STRAND classifier. can work on a larger sample	Considers statically extracted features only. Behavioural features were not considered.
Dropped files based Polymorphic Malware Detection [58]	DFPMD	Proposed a simple but fast technique of identifying polymorphic malware using dropped files.	False positives may incur, just in case where the genuine file has the same features.
Sequence-based classification and ensembles detection [99]	SBCE	A novel approach that extends [72] to detect polymorphic malware by using biological gene sequences and STRAND classifier. It uses (.bytes) and .asm files and can work on a larger sample.	Considers statically extracted features only. Behavioural features not considered. False positives may incur just in case where the genuine file has the same features

Table 2.5 – continued from previous page

Technique	Code	Contributions	Limitations
String matching and substitution matrices [22]	SMSM	Proposed a novel detection method that extracts of syntactic structures from hex code and implements Smith-Waterman algorithm to identify variants. Accurate on small samples.	Need to be evaluated on a larger dataset. Considers statically extracted features only. Behavioural features not considered
Client-server based detection [9]	CSD	Novel client-server approach where virus information is stored on a remote database and does not allow individual client updates.	Serious problems can occur when the Internet is not available.
Viral polymorphic malware detection [100]	VPMD	Novel polymorphic malware detection approach that is based on automatic signature extraction and uses Needleman-Wunsch and Smith-Waterman algorithms for detection mechanism.	Needs to be evaluated on a larger dataset. Considers statically extracted features only. Behavioural features not considered. Considers only known polymorphic variants of an existing malware
Hybrid clustering detection approach [101]	HCDA	Detection method that uses signatures and pattern matching to detect polymorphic malware.	Need to provide detection accuracy. Behavioural analysis needs to be considered.

Table 2.5 – continued from previous page

Technique	Code	Contributions	Limitations
Combined Token Extraction and Sequence Alignment detection [60]	CTESAD	Novel signature-based technique that combines token extraction and multiple sequences alignment for achieving a high accuracy and noise tolerance flexibility. It is fast and noise tolerant	Low detection accuracy. Behavioural analysis needs to be considered.
Efficient hybrid technique for detecting zero-day polymorphic worms [17]	EHTZDD	Early detection and containment of zero day polymorphic malware. Automatic signature generation	Behavioural features not considered.

Table 2.6: Data sources of selected techniques

Techniques	**Dataset source**	**Number of samples**	**Number of families**	**Malware only**	**Malware and benign**
TFBE	ClamAV, VirusTotal, VirusShare and Contagio	3637			Yes
SCD	Microsoft kaggle malware dataset 500GB	21741	9	Yes	
DFPMD		100		Yes	
SBCE	Microsoft kaggle malware dataset 500GB	21741	9	Yes	
SMSM	Second Part to Hell	352	1	yes	
CSD	-	-	-	-	
VPMD	Second Part to Hell	1105+352	2	yes	
HCDA	-	-	-	-	
CTESAD	DARPA 1999 Intrusion Detection Evaluation Data Sets	100	4		
EHTZDD	honeypots	15435			yes

Table 2.7: Evaluation methods and performance metrics used by the selected techniques

Techniques	**Val_CV**	**Test_split**	**Test_Sample**	**Accuracy**	**Log loss**	**F_score**	**FP**	**FN**	**ROC**	**TPR**	**FPR**
TFBE	yes			0.9999			0.0001			0.997	0.003
SCD	yes			95%	0.222864						
DFPMD											
SBCE	yes			98%							
SMSM				100%							
CSD											
VPMD				100%							
HCDA				97.83%		95%					
CTESAD				52%							
EHTZDD				96%						0.961	0.06

Table 2.8: Algorithms and analysis methods used by the selected techniques

Techniques	**Static_Analysis**	**Dynamic_Analysis**	**Algorithms**
TFBE	Yes	Yes	Meta bagging Advanced Ensemble Belief Propagation (BP), Locality- Sensitive Hashing (LSH)
SCD	Yes		Strand gene sequence classifier Jaccard Similarity
DFPMD		Yes	
SBCE	Yes		Strand gene sequence classifier
SMSM	yes		Smith-Waterman algorithm
CSD			
VPMD	Yes		Needleman-Wunsch Smith-Waterman
HCDA	yes		KNN
CTESAD			multiple sequence alignment
EHTZDD	Yes		

2.8 Gaps in the literature

This chapter explored the literature about polymorphic malware analysis as well as their detection techniques and tools. Most of the previous works used accuracy as their main performance metric without considering the cases where the dataset is unbalanced, thus causing a bias towards the class with the majority of instances. Another observation is that due to the size of the dataset, the type and number of features used, the reported accuracy might be a result of over-fitting. Some extracted either static features or dynamic features, while others used both types of features. The techniques that use structural features only have a high rate of false negatives, whereas those that employ behavioural features are likely to have a high rate of false positives. On the other hand, the hybrid features might not provide the best-expected accuracy [111]. However, they offer significant benefits as they provide a wide knowledge to the model useful for online and offline detection of the malware.

Despite the available tools and strong defensive techniques, there are much more sophisticated attacking mechanisms built in polymorphic malware. Therefore, research must be done continuously as a major way to find more capable solutions to this big

problem. However, based on this review, we realized that each technique and tool has its weaknesses and strengths. The following gaps still have to be taken into consideration in order to develop efficient detection solutions:

- There is a need for an improved feature engineering mechanism that could address efficient detection at a larger scale.
- There is a need to investigate the impact of a combination of analysis and detection techniques for the improvement of detection approaches.
- Detection mechanisms need to be improved to better manage malware as early as possible before the system is infected.
- There is a need for interactive coordination in terms of information sharing and automatic detection responsibilities among detection systems. This could be implemented in a multi-agent based detection approach.
- There is a need for dynamic vulnerability analysis capabilities implemented in detection systems. This will enable them to scan installed applications and report problems in time to the software owners. Therefore, this will limit the spreading of zero-day polymorphic malware if owners can quickly fix the bugs.

CHAPTER 3

Methodology

Outline: In this chapter, the researcher discusses the overview of the research philosophy in general and puts a particular focus on the adopted philosophy and the underlining research process. Methods and techniques used for data collection, processing and analysis are also presented. The overview of the research philosophy is given in Section 3.1, whereas the adopted philosophy for this study is discussed in Section 3.2. In Section 3.3, we present the research process. Details about data acquisition are provided in Section 3.4. In Section 3.5, we provide a discussion on the requirements for malware analysis. We discuss about static and dynamic analyses. We also discuss the Exploratory Data Analysis (EDA). In Section 3.6, we present an overview of our novel feature engineering approach. Malware classification and detection models are discussed in Section 3.7 and Section 3.8, respectively. Finally, in Section 3.9, we discuss the evaluation metrics used in this study.

3.1 Research Philosophy

3.1.1 Overview of research philosophy

Dudovskiy in [112] defines the research philosophy as a belief of how data relating to a phenomenon is collected, analyzed and used. It focuses mainly on the nature, source and development of knowledge [113]. Saunders et al. [114] stated that the research philosophy

should be presented first in the methodology as described in the steps to be followed by the research Onion [113]. This philosophy should describe the strategies and methods chosen by the researcher in order to achieve the objectives of the study [114].

According to Dudovskiy [112], the research philosophy can be viewed in two different ways, namely ontology and epistemology. Ontology deals with the nature of reality. It encompasses the beliefs of individual interpretations of facts and their constituents [115]. Epistemology (also known as constructivism), on the other hand, considers reality as a product of human intelligence based on their experience in interacting with the real world [112]. It is based on cognitive psychology and includes people's mental activity in the process of discovering the reality [112].

In order to create knowledge, the research poses certain questions through formulations of beliefs and assumptions. The answers to these questions are therefore given by the results of the research. There are four types of research philosophies: pragmatism, positivism, realism and interpretivism [112].

Pragmatism asserts that concepts are relevant if and only if they support action [112], [116], [117]. According to this philosophy, the world can be interpreted in many different ways or there are multiple realities whose reality should be negotiated, debated and interpreted [117]. Pragmatism relies on the research question as the main deterministic aspect of the research philosophy [116]. Pragmatist researchers can use a combination of any necessary methods, approaches to answer the posed research questions [112]. They can combine both interpretivism and positivism or combine both quantitative and qualitative approaches [112].

Positivism says that there is only one single reality. It states that science is the only way to know the truth. This philosophy emphasizes that knowledge should only be acquired through observation and measurement [112] [112] [118]. The hypotheses are developed by inductive reasoning and tested during the research process. The main task of the positivist researcher is to collect data and interpret it using statistical tools. It provides predictive results of the phenomena and their relations between them [119].

Realism underpins that knowledge is developed through a critical approach [120]. There are two realism, namely direct realism and critical realism. Direct realism, also known as naïve realism is based on the concept of what you see is what you get. It actually portrays the world as what is perceived through human senses [121]. On the contrary, critical realism states that images and real-world sessions are misleading because they do not really represent what the world is.

Interpretivism underpins that the only access to reality should be through social con-

structs such as consciousness, language, instruments and shared meanings. It emphasizes that researchers have to integrate human interest into their study [119]. This philosophy places more emphasis on qualitative analysis than quantitative analysis. It encompasses perspectives on various approaches while recognizing the differences between people [112]. It may employ different methods to study different aspects of a problem.

3.1.2 Adoption of a research philosophy

This research adopted positivism research paradigm, based on the main objective of this study, which was to create an approach that improves the performance of classification and detection of polymorphic malware. Based on aspects of the positivism philosophy discussed in previous section, this work is consistent with positivism where a quantitative approach is employed in trying to know how many detections can be achieved to be considered good enough in addressing malware problem. Data collection was done, analysed, tested and predictions were made. Extensive experiments were performed to test the developed approach.

Based on the reasons outlined above which fall within the positivism research philosophy, the researcher has therefore adopted it as a guiding paradigm for the achievement of research objectives.

3.2 Adoption of a Research Approach

There are three types of research approaches, namely, deductive, inductive and abductive. The deductive approach is based on the fact that the research should develop theories/hypotheses and then devise a test strategy to validate the assumptions [112] [122] whereas, in inductive approach, the researcher develops a theory after having analyzed the data collected [112] [122]. In abductive approach, the researcher should identify incomplete observations and thereafter provide an explanation of this phenomenon using either quantitative/ qualitative methods or both methods [112] [122]. It addresses the weaknesses found in deductive and inductive approaches.

This research is built on the assumption/belief that with well-engineered features, machine-learning models can efficiently classify and detect polymorphic malware. The major focus was to create a feature engineering approach that improves classification and detection of model's performances. The efficiency of the model should prove the

assumptions already provided through testing and evaluation. Based on this background, the researcher adopted a deductive approach.

3.3 Research Process

The research process for this work comprises activities from the literature survey to model evaluation as described in Figure 3.1. After the literature review, the researcher created a conceptual framework illustrated in Figure 3.2. As the conceptual framework shows, there is a possibility of attack when three ingredients are combined, namely malware, the target and the lack of effective protection. The scope of this search was solely based on the malware component. The researcher focused on the exploitation, analysis and design of new mechanisms to represent malware and thus facilitate their classification and detection.

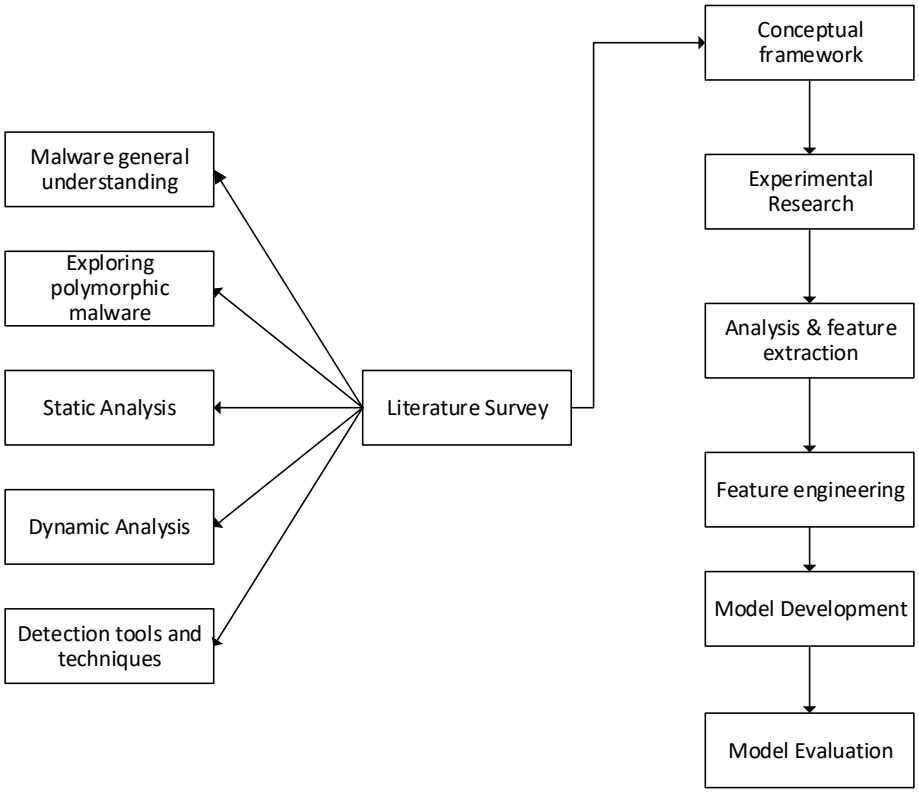

Figure 3.1: The research process

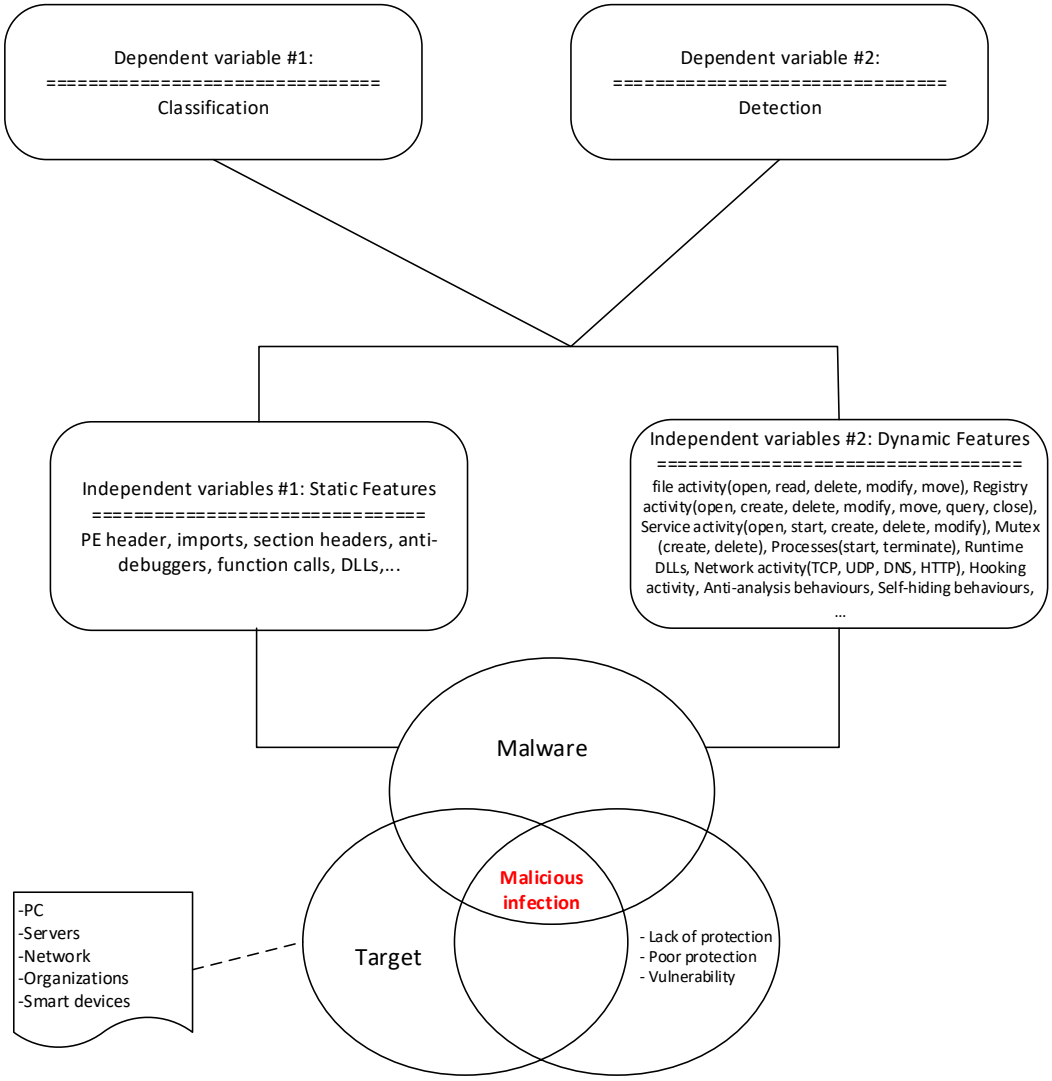

Figure 3.2: Conceptual framework

The researcher sought to understand the impact of both static and dynamic features in the overall process of classifying and detecting polymorphic malware. This study, therefore, employed a quantitative experimental approach, in which variables are manipulated whenever necessary. With this approach, the researcher could conduct a study on a small sample and generalize to a larger population.

In the context of this study, a small number of malware were used in the study and the resulting model should be generalized to work on a multitude of malware that appear frequently on a daily basis.

The strategies used to achieve the objectives of the study are shown in Table 3.1

Table 3.1: Strategies to achieve objectives

Objective	**Method/Strategy**	**Description**
To investigate the existing literature and identify the unaddressed gaps related to polymorphic malware.	-Literature review	The researcher explored journal papers, conference papers, as well as books.
To design a scalable novel feature engineering (NFE) approach based on static and dynamic features.	-Constructive research -Algorithm design paradigm	The user constructed a practical innovative solution using algorithms.
To develop an improved machine learning classification model based on (NFE) features using supervised classifiers.	-Empirical research -Experiments	Quantitative data were collected on which appropriate experiments were conducted.
To develop a Deep learning early-stage malware detection model based on (NFE) features.	-Empirical research -Experiments	Quantitative data were collected on which appropriate experiments were conducted.
To evaluate the developed models using standard evaluation metrics.	-Experiments	Standard performance metrics were employed to test the outcomes of the conducted experiments.

3.3.1 Experimental design

Experimental research is used when it is possible to manipulate variables. This is the case of this study because the researcher had to analyze the malware, extract and manipulate features in order to develop optimal techniques to achieve the objectives of the study.

3.3.2 Research population

The search used two types of programs (Benign and Malware) compatible with the Windows operating system. All the files were executables. Malware variants were grouped in their various respective families. The benign files were extracted from new Windows installation files or from various tested files.

3.3.3 Sampling process

3.3.3.1 Sample size analysis (Datasets)

The researcher needed an appropriate sample size to be able to make inferences about the population.

When the population is unknown, the minimum sample size is obtained using the following equation [123]:

$$n = \frac{z^2 * \widehat{p}(1-\widehat{p})}{\varepsilon^2}$$

,

Where n is the minimum size of the required sample, z is the z-score($\pm$). $z = 1.96$ for a 95% confidence level. $\widehat{p}$ is the population proportion and ε is the margin error.

Suppose that the total malware population is unknown due to the daily production of new and polymorphic malware. Let us have a population proportion of at least 4000 and a confidence level of 99%. Applying the above given formula, we get $n = 570$. This means that the researcher needed at least 570 malware samples to conduct an experiment. Good enough, the used datasets were bigger than the minimum sample size. Dataset 1 was composed of 6032 samples. Dataset 2 was composed of 343455 samples.

Based on the requirements of the study, which consisted mainly of using polymorphic malware, the researcher downloaded the samples grouped into their respective families and described as polymorphic by the majority of antivirus programs under the facility of virus total. Benign files were mainly chosen from fresh Windows installations.

This research uses datasets of different sizes. Datasets were big enough to allow the learning algorithms to acquire more knowledge adequate to avoid over-fitting or under-fitting.

3.3.4 Classification and detection architecture

The proposed classification and detection approach is illustrated in Figure 3.3. A malware sample is analysed using advanced dynamic analysis and advanced static analysis. Behavioral and structural features are extracted. All features were combined to make a big feature dataset. Features were further engineered using a novel feature engineering (NFE) approach. Appropriate models for polymorphic malware classification and detection were developed.

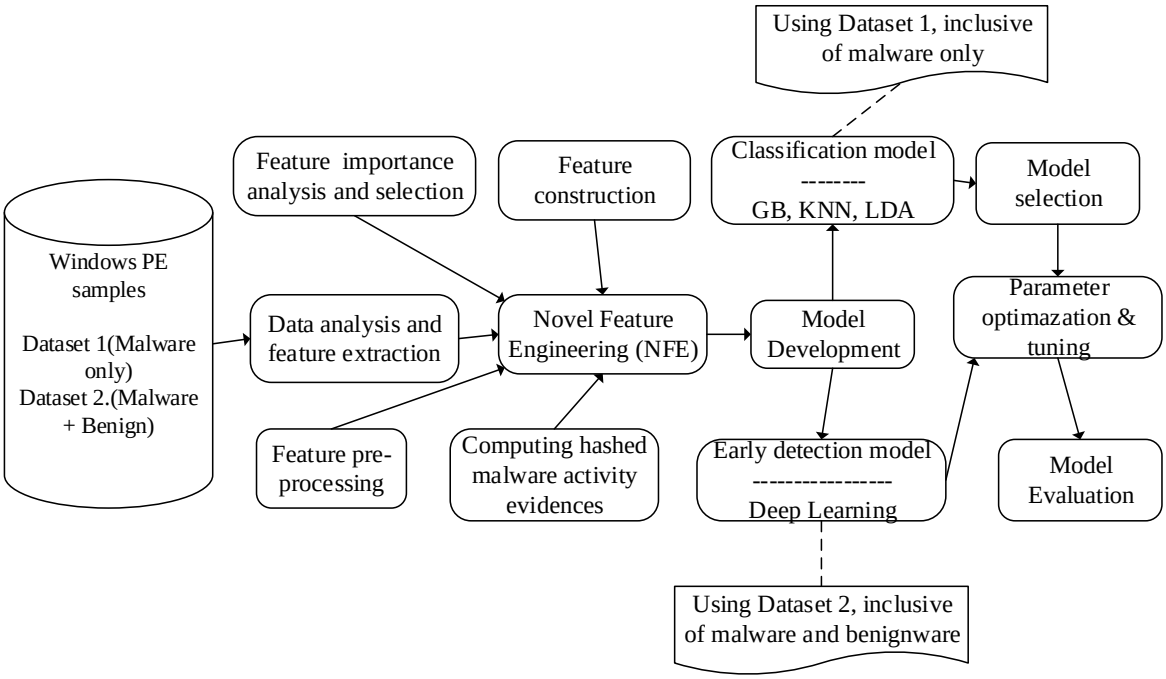

Figure 3.3: Classification and Detection architecture

3.3.5 Resource requirements

The main resources required were malware samples and benign samples. The researcher obtained different samples from online repositories. The labelling of samples was done by virus total. Duplicate samples were removed from analysis dataset. Malware samples were saved in a compressed .zip folder with credentials to prevent any type of accidental infection.

Other useful resources included tools such as IDA pro, Windows Sysinternals, and Cuckoo sandbox. Different sub-modules were developed such as feature extractor, feature pre-processor, feature engineer, malware classifier, early-stage malware detector. More experimental resources are shown in Table 3.2 below:

Table 3.2: Resource requirements

Resource	Description
Datasets	-Malware samples, -Benign samples
Analysis computer	i7, 1TB, 16GB RAM
Operating systems	Ubuntu 16.4, Windows 7
Oracle VBOX	For visualization
IDA Pro	Static analysis
Cuckoo sandbox	Static and Dynamic analyses
Python	For coding and implement all needed tasks
MongoDB	For storage
Additional Python modules(Scikit-lean,tensorflow, Pandas,Keras,...	For machine learning and data analysis related tasks

3.4 Data Acquisition and preparation

The appropriate samples for this research were obtained from online repositories. The researcher first obtained raw samples from malware zoo [124], which were then used for preliminary analyzes. Later, this research mainly considered two large public datasets: Dataset 1 (See Section 3.4.1) and Dataset 2 (See Section 3.4.2) composed of malware samples collected from multiple online repositories such as Virus Share [125], VX Heaven [126], Malware Tips [127], Malware repository [128], Malware Samples for researchers [129], and SecLab [130]. Thus, the samples used for this research were sufficiently comprehensive to cover a wide range of research objectives and can be used for further development of anti-malware tools.

3.4.1 Dataset 1

Dataset description

Dataset 1 has been provided by Marco Ramilli [131] and was named *"Malware Training Sets: A machine learning dataset for everyone"*. This dataset contains polymorphic malware classified in five different families. There were no benign files in this dataset. Different categories of malware contained in our main dataset are shown in Table 3.3. This dataset has been used to develop the classification model.

Table 3.3: Polymorphic malware families

Malware family	Number of samples
APT	292
Crypto	2020
Locker	431
Zeus	2019
shadowbrokers	1270

Feature extraction

Features were extracted using dynamic analysis and static analyzes. Sandboxes were used to perform static and dynamic analyzes. The concepts provided by P. Trinius at al. [132] in Malware Instruction Set for Behaviour-Based Analysis [MIST] were followed to extract features. MIST is essentially an optimized representation for an efficient investigation of malware behaviours using data mining and machine learning techniques. After feature extraction, every sample's information was organized into a JSON (JavaScript Object Notation) file format. JSON is a lightweight data interchange format that utilizes human-readable texts to represent data objects consisting of attribute-value pairs and array data types. The snapshot of the produced dataset is shown in Figure 3.4. Each JSON property shown in Figure 3.4 ([131]) is considered as a feature. Every feature represents the malicious actions at some level. In the dataset, pieces of evidence of each feature were identified for each malware sample and hashed using a hashing technique known as ELF-hash function [1]. The overall process of training set generation is illustrated in Figure 3.5 ([131]). Generated features are presented in Figure 3.6. The description of the extracted features is found in Appendix A.

[1]ELF is a mathematical function that converts a given input value into another cryptographic hash value of fixed length

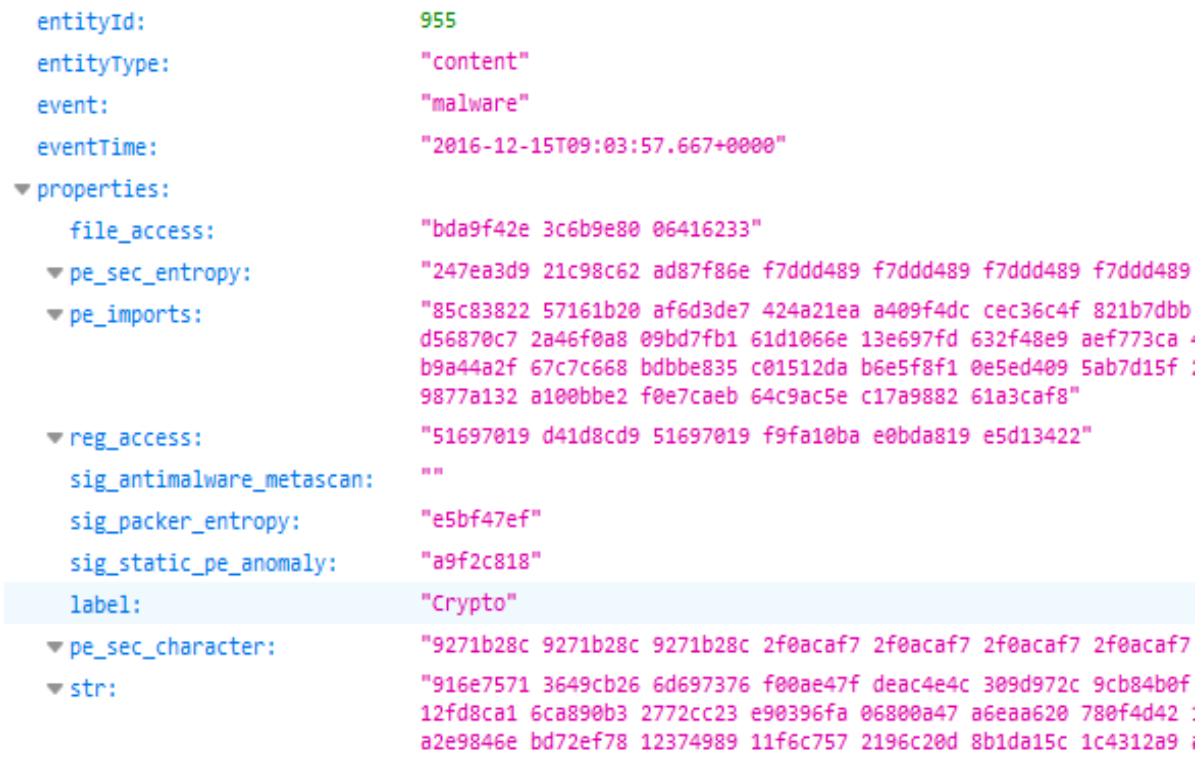

Figure 3.4: Extracted features in a JSON file

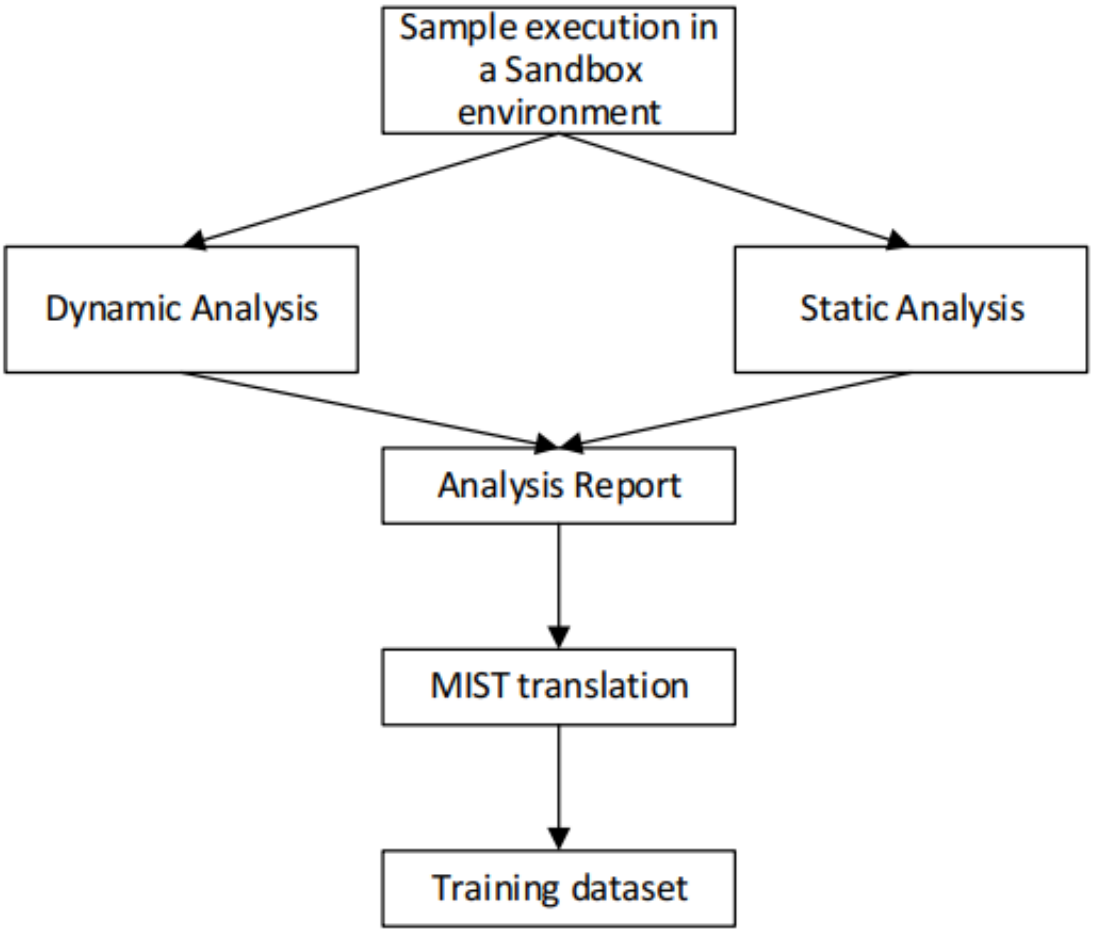

Figure 3.5: Training dataset generation flowchart

api_resolv	antisandbox_sunbelt_libs	disables_windowsupdate	persistence_ads	antimalware_metascan
cmd_exec	antisandbox_suspend	downloader_cabby	persistence_autorun	antisandbox_cuckoocrash
file_access	antisandbox_unhook	dridex_behavior	persistence_service	antisandbox_productid
file_delete	antivirus_virustotal	driver_load	polymorphic	antisandbox_restart
file_drop	antivm_generic_bios	dropper	pony_behavior	antisandbox_sboxie_libs
file_read	antivm_generic_disk	encrypted_ioc	process_interest	antisandbox_sleep
file_write	antivm_generic_disk_setupapi	exec_crash	process_needed	deletes_self
mutex_access	antivm_generic_diskreg	fraudguard_threat_intel_api	ransomware_extensions	deletes_shadow_copies
net_con	antivm_generic_system	infostealer_bitcoin	ransomware_file_modifications	disables_browser_warn
net_dns	antivm_vbox_files	infostealer_browser	ransomware_files	disables_system_restore
net_http	antivm_vbox_keys	infostealer_ftp	ransomware_from_StringFLOSS	disables_uac
pe_imports	antivm_vbox_libs	infostealer_keylog	ransomware_message	disables_windows_defender
pe_sec_character	antivm_vmware_devices	infostealer_mail	ransomware_radamant	office_security
pe_sec_entropy	antivm_vmware_files	injection_createremotethread	ransomware_recyclebin	origin_langid
pe_sec_name	antivm_vmware_keys	injection_runpe	rat_spynet	origin_resource_langid
reg_access	antivm_vpc_files	injection_rwx	reads_self	packer_entropy
reg_delete	antivm_vpc_keys	internet_dropper	recon_beacon	packer_upx
reg_read	antivm_xen_keys	ipc_namedpipe	recon_checkip	packer_vmprotect
reg_write	api_spamming	mimics_agent	recon_fingerprint	stealth_window
service_create	banker_zeus_mutex	mimics_filetime	recon_programs	virus
service_start	banker_zeus_url	mimics_icon	removes_zoneid_ads	yara_detection
FraudGuardThreatIntelAPI	bcdedit_command	modifies_certs	sniffer_winpcap	str
ShadowServer_White_list	bootkit	modifies_desktop_wallpaper	static_detection	antidbg_windows
ThreatCrowdResolutions	browser_security	modifies_hostfile	static_pe_anomaly	antiemu_wine_func
antianalysis_detectfile	bypass_firewall	modify_proxy	static_versioninfo_anomaly	antiemu_wine_reg
antiav_detectfile	clamav	modify_security_center_warnings	stealth_file	critical_process
antiav_detectreg	copies_self	modify_uac_prompt	stealth_hidden_extension	cryptam
antiav_servicestop	creates_largekey	multiple_useragents	stealth_hiddenreg	deepfreeze_mutex
antidbg_devices	creates_nullvalue	network_bind	stealth_hide_notifications	network_cnc_http
stealth_network	stealth_timeout	stealth_webhistory	network_torgateway	network_http

Figure 3.6: Dataset 1 features

3.4.2 Dataset 2

The second dataset is provided by Cardiff University [133]. This dataset contains a mixture of benign and polymorphic malware. Burnap et al.[107] used it to create early malware prediction approach. Features were collected by using Cuckoo Sandbox. The dataset has 594 benign samples and 595 malicious samples. Each file was run up to 305 seconds. The total number of entries in the CSV dataset file was 343455.

The dates when the files were first seen ranged from 2006 to 2017. The dataset was split into test and training set files according to the first seen date to mimic the arrival of completely new software. The training set included only samples seen for the first time by VirusTotal before October 10, 2017 at 11:15 am. The test set included samples seen after this date.

The dataset contained many types of malware namely Trojan, Virus, Adware, Backdoor, Bot, Worm, Rootkit as well as their variants.

Extracted features are shown in Table 3.4:

Table 3.4: Features for Dataset2

Feature	**Description**
sample_id	an identifier value for the samples (categorical)
vector	time in seconds since start of file execution (numeric)
malware	class label 0=benign, 1=malicious (categorical)
cpu_sysem	percentage of cpu being used to run programs in system kernel (numeric)
cpu_user	percentage of cpu being used to run programs in user space (numeric)
memory	bytes currently being used in memory (numeric)
swap	bytes currently being used in swap memory (numeric)
total_pro	total number of processes running (numeric)
max_pid	maximum process id held by a process (numeric)
rx_bytes	number of bytes being received (numeric)
tx_bytes	number of bytes being sent (numeric)
rx_packets	number of packets being received (numeric)
tx_packets	number of packets being sent (numeric)
test_set	True=sample belongs to test set, False=sample belongs to training set

3.4.3 Imbalanced Dataset Handling

In fact, most real-world datasets are imbalanced [134].The model learning process typically produces skewed results with superior predictive accuracy for the dominant class and poor accuracy for the minority classes.

The researcher employed a Synthetic Minority Over-sampling Technique (SMOTE), which is robust for learning from unbalanced data sets [135]. SMOTE uses an oversampling approach to rebalance the original training set by introducing synthetic data.

An instance x_i of an under-represented class is selected as the basis for creating new synthetic data points. Many of the nearest neighbors of the same class (points x_i1 to x_i4) are chosen from the training set based on a distance metric. A randomized interpolation is finally carried out in order to obtain new instances $r1$ to $r4$. This process is illustrated in Figure 3.7 ([135]).

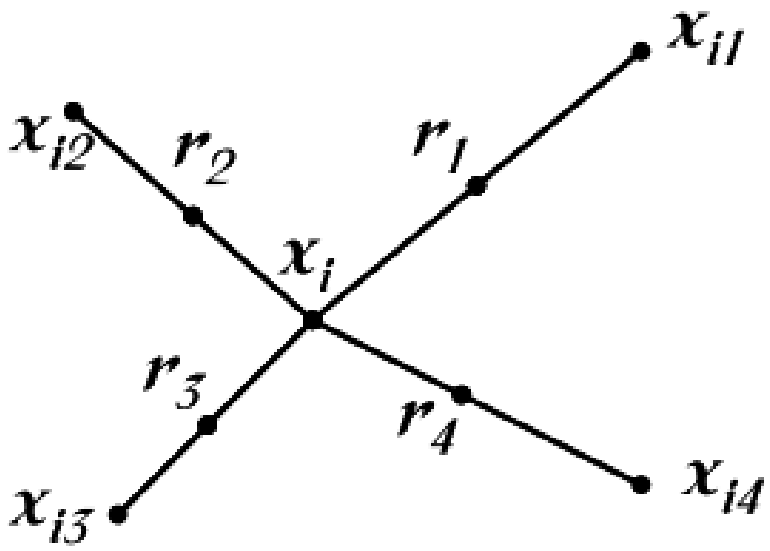

Figure 3.7: Synthetic data points creation process by SMOTE [135]

3.4.4 Feature Transformation

The purpose of this process was to have good training data, well optimized to meet the requirements of machine learning algorithms. Feature transformation facilitates the learning process and provides an additional background experience to input data, thereby allowing the learning algorithm to benefit from such experience [136].

The Datasets were composed of numeric variables with enormous differences among them. The transformation was needed to get good input features that would contribute to the success of learning algorithms. RobustScaler method was adopted for data transformation.

RobustScaler

This method is robust to outliers. It uses the interquartile ranges in the data normalization process. The following formula is applied on each feature.

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

, where x_i is a feature and Q_1 and Q_3 are first and third quantiles respectively.

3.5 Data Analysis

3.5.1 Static and Dynamic Analysis

Both static and dynamic analysis are used to obtain the characteristics and behaviors of the executable samples. The researcher analyzed the few raw samples obtained to extract features and understand the mechanism of malware polymorphism. Structural and behaviour characteristics were obtained through static analysis and dynamic analysis respectively. For system security, a virtualized environment has been established and used to analyze malware samples.

3.5.1.1 Malware analysis lab setup

The environment used consisted of the following components:

- Ubuntu 16.4 host machine. Cuckoo software for automatic static and dynamic analysis has been installed on this host machine. Malware samples were stored on this host machine. All of the samples were Windows executables not compatible with the Linux operating system. Through cuckoo web interface, samples were uploaded to the guest machine where they were inspected and run to extract malware structural and behavioural features.

- Oracle Virtual Box hypervisor. This is a virtual machine monitoring software that creates and runs guest machines.

- A windows 7 guest machine inside VBox hypervisor. Malware were executed on this host machine. The host machine was reset to initial status after executing each malware. A Python agent application was installed on the guest machines to ensure good communication between the host and guest machines during sample analysis.

All extracted features were saved in MongoDB as JSON reports. The tools used helped the researcher to perform the following tasks:

- Basic system monitoring
- Basic Application monitoring
- API monitoring
- Internal functions monitoring
- Suspiciousness hooking
- File activity monitoring
- Registry activity monitoring

3.5.2 Exploratory Data Analysis(EDA)

The purpose of EDA is to investigate data in order to understand its characteristics using statistical methods. EDA provides useful insights about the main characteristics of the data such as feature correlations, distributions, etc. It provides a solid understanding of data that is useful for preparing data for modeling, as well as generating hypotheses for later analysis [137]. Various visualization methods such as histograms, density plots, box plots, correlation and scatter matrices were used in this analysis.

The researcher used the EDA for quantitative analysis to understand, describe, and explain the relationships amongst features and the relationships between features and the predicted class. Using graphs, diagrams and tables, the researcher was able to draw conclusions to better understand the problem and create a new solution.

3.6 Novel Feature Engineering (NFE) Approach

It is all about features [47]. Informative, independent and discriminating functionalities are essential in the creation of classification and detection models.

In the context of improving existing approaches, this research proposed a novel feature engineering (NFE) approach based on structural and behavioural features. A good number of features have been taken into account to largely cover the most basic functionalities of malware. The impact of each feature on the outcome has been thoroughly evaluated and appropriate techniques have been applied to retain the most useful. By Using the

functionalities obtained with NFE, effective approaches to classification and detection of polymorphic malware have been proposed and evaluated.

NFE approach that uses the domain knowledge of the data to manipulate and create features that deliver superior performance gains. The proposed technique is robust to overfitting. It also prevents high rates of false positives or false negatives.

NFE provides a pre-processing approach to deal with any type of impurity in the data. This task consists of a series of subtasks to clean up the dataset and make it ready for use. NFE also provides the mechanism of selecting the most relevant features that will positively affect model performance. It is necessary to have good features that better describe the structures inherent to the data. A good model must be based on non-misleading, appropriately normalized, and informative data. Stuart Feffer in [47] states that without the proper features, no machine learning algorithm will work better. Good features can allow even a simple model to beat a complex one [138]. In general, the advantages of the NFE approach are as follows:

1. To provide a unique way of computing feature importances
2. To select the most discriminating features
3. To extract / generate new features whenever necessary to increase the predictive power of the model
4. To reduce overfitting on both training and testing sets
5. To improve modelling accuracy.
6. To minimize training time

3.7 Developing a Classification Model

In this Section, a number of experiments were conducted to develop and evaluate the classification model. Our problem falls in the multi-class category, where there are more than two classes to predict. There were five families of malware to be classified. We adopted supervised machine learning algorithms that have the capabilities of handling multi-class prediction problems such as Linear Discriminant Analysis (LDA), KNN and the Gradient Boosting Classifier [139].

LDA : LDA calculates statistical properties of data for each class. Mean and variance from data for each class are estimated. LDA makes predictions by estimating the

probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made. LDA uses Bayes theorem to estimate the probabilities [139].

KNN : K-NN uses a similarity test measure in prediction. Distances are calculated between the targeted instance and all other instances [139]. The shortest distance represents the strongest similarity. When there is a strong similarity, the predicted class is taken from the K-most similar instances.

GB : GB is a modern ensemble method for boosting or improving the performance of decision trees [139]. This is done by constructing a model from the training data and then creating a second model that attempts to correct the errors of the first model. Models are added until the perfect prediction performance is reached.

These algorithms were chosen because they support multi class problems. Another reason was that by using different algorithms in our experiments, we could test different options and adopt the one that gives the best results.

In order to estimate and evaluate the performance of our model, the test harness was conducted using a 10-fold cross-validation technique on the training set.

Experiments were done using raw features and new features generated by our novel NFE approach. Since the data set is unbalanced, ie the samples for each category of malware were not distributed proportionately, we also examined the two scenarios (unbalanced and balanced). In order to create a balanced version of the dataset, we adopted the Synthetic Minority Over-sampling Technique (SMOTE). This technique contributed to oversampling by generating synthetic data and increasing the number of minority classes.

3.8 Developing an early-stage Malware Detection Model

Signature-based systems use an already defined pattern to detect existing malware. They fail when a malware morphs itself. They have to wait for a new signature to be created. Another thing is that malware can change in an infinite number of instances. It is not possible to have an unlimited number of signatures and no database can store an unlimited amount of data.

As the malware corpus continues to grow, early detection is crucial for better protection. To meet early detection needs, the researcher adopted deep learning(DL) models as they are very useful in handling large datasets and provide high accuracy [42], [104].

Based on NFE features, the researcher created a deep learning model that perfectly learns what a malware looks like to identify unknown program as malicious or harmless with high accuracy as early as possible.

The model is able to learn and detect in a very short time, less than a few milliseconds. This could lead to further actions before damage is caused.

The researcher used Multi-Layer Perceptron Neural Network and implemented it using Python library called Keras with Tensor flow at the back end.

3.9 Evaluation

The researcher employed performance metrics for evaluation. Standard metrics were used such as Accuracy, Recall, F score, log loss, and confusion matrix. However, accuracy is not a good performance metric when the dataset is unbalanced. This study addressed the imbalance problem as well.

Confusion matrix

It is used to describe the performance of a classifier. Outputs are presented in a table layout shape. Information given by the matrix is as follows: True Positive (TP), which represents the number of malware correctly identified as being of a specific class 0. True Negative (TN) is the number of malware that are correctly identified as not being of class 0. False Positive (FP) represents the number of malware that are wrongly classified as being of class 0. False Negative (FN) shows the number of malware wrongly classified as not being of class 0.

Precision

It evaluates how often the model is correct when it correctly detects malware as being of class 0. It is defined as follows:

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

Recall

It evaluates how often the model can correctly detect malware as being of class 0. It is also called sensitivity. It is defined as follows:

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

F_measure

It is also known as the F1 score. This is a harmonic mean of recall and precision. It gives describes the robustness and precision of the model. It is defined as follows:

$$F\_measure = 2 * \left(\frac{Precision * Recall}{Precision + Recall} \right)$$

Logloss

: It is computed as follows

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Where M is number of classes

y - binary indicator (0 or 1) if class label c is the correct classification for observation o.

p - predicted probability observation o is of class c.

Accuracy

Accuracy is the fraction of predictions our model predicted correctly.

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

Chapter 4

Creation of a Novel Feature Engineering Approach

Outline:
In this chapter, we present the entire process of creating a novel feature engineering (NFE) approach. In Section 4.1, we discuss the experiments necessary to extract malware features. We give a case study analysis and demonstrate how malware achieve polymorphism. In Section 4.2, we provide details on how extracted features are converted from JSON format to CSV format. Since we used machine learning, we needed the dataset to be presented in .CSV format.. In Section 4.3, we present preliminary data pre-processing activities required for the successful creation of this technique. Exploratory analysis is discussed in Section 4.4. In Section 4.5, we provide final detailed steps for creating NFE. Finally, a summary is given in Section 4.6. This chapter is partially based on the works in [140][141]

4.1 Malware Features

Features represent specific activities of the malware. To understand such activities, the researcher performed preliminary static and dynamic analysis on a small dataset of polymorphic malware and benign samples. Cuckoo sandbox was used for this task. After analysis, reports were created in "json format". Finally, those ".json" were converted to ".csv" for further processing. In order to identify key functionalities, we monitored visual

behaviours of malware, while in execution. We followed the traces of API executions and explored static metadata.

Case study: Win32/Potao

Observation1: API calls

After running a variant of a polymorphic malware called Win32 / Potao twice, we found that this malware delivers the same payload in different ways.

Table 4.1 below shows API statistics during two executions/investigations of Win32/Potao in the same environment. It is obvious that the functionalities are exactly the same on both executions. Here, we recorded the statistics of API calls performed by the malware.

Table 4.1: API executions for win32/Potao malware

API	**Investigation1**	**Investigation2**
LdrGetProcedureAddress	19	19
NtClose	10	10
LdrGetDllHandle	4	4
NtTerminateProcess	3	3
NtOpenFile	3	3
NtCreateKey	2	2
NtMapViewOfSection	2	2
NtUnmapViewOfSection	2	2
NtQueryAttributesFile	2	2
NtCreateSection	2	2
LdrLoadDll	2	2
etc. ...	...	...

Observation2: Dropped files

Investigation 1

Table 4.2: Investigation1: Files dropped by Win32/Potao malware

Name	b029deb6da0a1a62_pgexhu.wm
Filepath	C:\Users\pc1\AppData\Local\Temp\pgexhu.wm
Size	18.4KB
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	c95916e402b83e84dddbdf9814c834ae
File content (partial)	user32.dlladvapi32.dll ws2_32.dllshell32.dlladvapi32 PlugNtDeviceIoControl File.exe. tmpcode=&md5=data=%d %d x:%d

Investigation 2

Table 4.3: Investigation2: File dropped by Win32/Potao malware

Name	b7464cf8dae62ee0_dbsdh.n
Filepath	C:\Users\pc1\AppData\Local\Temp\dbsdh.n
Size	17.2KB
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	13bb044121eb6c4281c273adbaf19abb
File content (partial)	user32.dlladvapi32.dll ws2_32.dllshell32.dlladvapi32 PlugNtDeviceIoControl File.exe. tmpcode=&md5=data=%d %d x:%d

As shown in Tables 4.2 and 4.3, this malware dropped files with randomized names and extensions as a means to disguise the detection system. However, the dropped files have the same content as highlighted in the tables above. By looking closely at the contents of both dropped files, we can see exactly that similar functions are embedded in them as a means to accomplish the same malicious payload. Another observation was that this malware injected and extracted two process dumps with different names from each other at each execution. These dumps are executable code fragments containing malicious content to be executed. The name of the dump process during the first run was: *"5c93a1c51d41c8097f00e04f.exe"*. The name of the dump process at the 2nd run was: *"5c93a38e1d41c8097f00e069.exe"*. Different names were randomly given to masquerade the identity of the malware.

From all observations shown above, it can be seen that polymorphic malware will be able to always disguise themselves.

We provide the behaviours of few other variants of Win32/Potao malware in Table 4.4. Although the API calls are not the same in most cases, the end functionality is the same for this particular malware.

Table 4.4: API calls for various variants of Win32/Potao malware

API	**Variant 2**	**Variant 4**	**Variant 5**	**Variant 7**	**Variant 8**
ReadProcessMemory	110708				
LoadStringW	78				
GetSystemTimeAsFileTime	120			1	3
LdrUnloadDll	106		3	3	5
NtQueryInformationFile	104	1		1	
RegQueryValueExW	1071				
NtClose	434	10	27	11	22
DrawTextExW	1004		1		
RegOpenKeyExW	631		1		
NtCreateFile	311	1	3	2	2
NtOpenProcess	680		1		
NtDelayExecution	696	3		2	1
NtOpenFile	157	3			
GetTimeZoneInformation	40				
GetKeyboardState	37				
GetKeyState	251				
LookupPrivilegeValueW	357				
RegCloseKey	453				
CoGetClassObject	303				
CoCreateInstance	146		3		
LoadResource	33	1	1		
GetForegroundWindow	309				
FindResourceExW	26				
SHGetFolderPathW	58				
GetSystemMetrics	2297		23		1
GetCursorPos	140				
NtCreateSection	47	2	1	1	1
GetFileAttributesW	130				
LookupAccountSidW	152				
RegOpenKeyExA	96				
NtQueryValueKey	30	1	3	5525	6
NtOpenKey	26	1	3	3	6
CoUninitialize	23		2		
timeGetTime	21		3		
UnhookWindowsHookEx	21				
LdrGetProcedureAddress	18	19	25	43	73
CoInitializeEx	18		2		
NtQueryDirectoryFile	17				
GetFileSize	17				
NtMapViewOfSection	16	2	1	4	4
SetErrorMode	14	1	1	1	

Table 4.4 continued from previous page

API	Variant 2	Variant 4	Variant 5	Variant 7	Variant 8
LdrLoadDll	13	2	3	8	13
NtFreeVirtualMemory	12			4	3
NtAllocateVirtualMemory	12	1	3	8	8
NtSetValueKey	12				
NtDuplicateObject	11				
NtCreateMutant	11		1	1	2
NtQuerySystemInformation	8			1	2
SizeofResource	7	1	1		
NtUnmapViewOfSection	7		2	1	2
GetSystemDirectoryW	7			1	2
NtQueryAttributesFile	7	2			
UuidCreate	7				
NetShareEnum	6				
EnumWindows	6				
LdrGetDllHandle	6	4	6	4	7
RegEnumKeyW	5				
FindResourceW	5				
GetComputerNameW	4				
GetFileSizeEx	4				
NtReadFile	4				
GetFileInformationByHandleEx	4				
CreateThread	2			1	2
CreateDirectoryW	2			1	
SetWindowsHookExW	2				
NtResumeThread	1				
SetWindowsHookExA		1	1	1	
__exception__			1		
SetUnhandledExceptionFilter			1		
GetTempPathW		1	1		
NtTerminateProcess		3	3	2	3
NtCreateKey					
NtOpenDirectoryObject		1			
SetFileAttributesW		1	1	1	
CreateActCtxW		1	1		
FindResourceA		1	1		
NtWriteFile		1	2	1	8
RtlDecompressBuffer		1	2	2	6
GetSystemInfo			1		1
NtOpenMutant			1		
GetSystemWindowsDirectoryW				83	1
NtOpenKeyEx				2	5
NtProtectVirtualMemory				3	4
SetFileTime				6131	

After performing this analysis on a few samples, the researcher identified generally the key features that enable polymorphism in malware. Details about these features are obtained in Table 4.5

Table 4.5: Key features

Feature	**Description**
Random file names drop	Writes encrypted scripts to files
Self avoidance by mutex	
Self destruct	
NetworkActivity	
Persistance through registry	It can register itself as a service.
Persistance through startup	It can register the run registry key value
Process injection	Create process dumps. It injects a snippet of code
Copy files into folders	
Register as part of a service group	
Environment awareness	Read computer name
Creation of many mutants	In order to synchronize copies of itself simultaneously injected into various core Windows processes(e.g., services.exe, iexplore.exe, winlogon.exe) that are already running.
Random mutex names	To avoid detection
File Delete	
Registry write	
Random process dumps	
Copy files into folders	
register itself as a service	
Checks if it is being debugged	Can check if the productId is a virtual machine. Can check a virtual hard drive. It Can check a virtual mouse.
Sleep or Delay Execution	
Packing Status	Most malware are packed for obfuscation purposes and this is the case of polymorphic malware. This feature reveals this obfuscation characteristic.
Number of sections	For most obfuscated malware, some sections are hidden to harden key functionalities. This feature gives hints about this characteristic.

Entropy	For obfuscated malware, entropy value is high, which means a high level of disorder in the file content.This is to complicate analysis and detection process.
Initialized Data Size	Size of initialized data section in bytes
Code Size	For most obfuscated malware, the code size is reduced
Entry Point	Starting address of an executable file.
Digital Signature Status	Most malware are not genuinely digitally signed as can be revealed by this feature.
OpCode at entry point	Most obfuscated malware don't start from the original entry point(OEP) of the real hidden malicious code. However, they start from their own entry point where there is an embedded code stub that is in charge of decrypting the malicious code and pass control to it to start infection process.

summary on observed behaviours

We realized that polymorphic malware can exhibit different behaviours as follows:

- Malware could drop different file names and types at every infection. The hash code of a file was changed, but the real content of malicious payload remained the same.
- Malware could inject different pieces of malicious executable codes in the memory at each infection.
- Some code sections were hidden due to encryption. This is achieved through obfuscation or packing.
- Malware could delete themselves and create new version of themselves with new names, either in the same folder or in another folder.
- Malware could hijack some legitimate processes and inject itself into them in order to run without notification and appear as legitimate processes.
- Some malware could be concealed with another legitimate file extension and icon other than the executable one. For example, a malware Potato malware appeared

as Microsoft word document, and while executed, it could load the executable payload and infect the system.

- Malware could create random mutex names to persist and at each time hide their identity.
- Malware could as well register themselves as services, which can run every time at start-up, or run from within other service groups. This ensured that they could exhibit persistence mechanisms.

Due to a few samples that were available during the analysis phase, and due to limited access to raw malware datasets, the researcher opted to use an online dataset titled "Malware Training Sets: A machine learning dataset for everyone" [131]. This dataset contained several polymorphic malware classified in their respective families. In addition to features of interest discussed in Section 4.1 above, this dataset had more important features to consider when developing our feature engineering approach. All features provided by this particular dataset are shown in Figure 3.6 and discussed in Appendix A.

4.2 Generation of a raw dataset from JSON files

The malware samples from dataset 1(See Section 3.4.1) were initially in JSON format. There were several JSON reports, and each report represented a malware.

The first task was to generate a raw data set by merging all individual JSON files. We used MongoDB to achieve this goal. MongoDB[142] is a fast NoSQL cross-platform document database program that supports and uses JSON based documents. It provides high availability, automatic scaling, and high performance. We created a simple batch script that was used to merge all the individual JSON sample files in one file as shown in Figure 4.1.

```
for /r . %%g in (*.json) do (
  set "var=%%g"
mongoimport --db test --collection test --file "%%g"
)
```

Figure 4.1: Batch script to merge JSON chunk files in one file

From JSON to CSV

We needed our dataset in CSV format for further processing. To do this, we followed the steps described in algorithm 1.

Algorithm 1 Dataset preparation

Require: F_{jsn} //json format file
Ensure: F_{csv} //csv format file
1: $list \leftarrow \emptyset$
2: $listDF \leftarrow \emptyset$ //empty list
3: **for all** $l \in size(F_{jsn})$ **do**
4: $\quad list \leftarrow append(l)$
5: **end for**
6: $list \leftarrow normalize(list)$
7: $listDF \leftarrow MakeDataframe(list)$
8: $F_{csv} \leftarrow transformToCSV(listDF)$

4.3 Data pre-processing

The dataset had a lot of null entries as well as impurities. This task consists of a series of subtasks aiming at cleaning our dataset and make it ready for use. Each feature represented a particular activity or a characteristic. Each feature value represented evidence that malware was engaged in a particular activity or had that specific characteristic.

In each feature, evidence were provided. There were features with empty evidence. This means that a particular sample is not affected by the specific behavior represented by this feature. Some other samples contained multiple evidence for one feature. For example, one sample had the following values for the feature **file_access (d41d8cd9 913f9c49 96da6d37)**, which means there were 3 pieces of evidence of file access activity recorded during the analysis of that particular sample. One of the data cleaning tasks was to transform the dataset by counting the number of pieces of evidence in each feature per sample. The outcome was $E_{num} \geq 0$, where E_{num} is the number of pieces of evidence of each feature per sample. All null values represented the absence of evidence in a particular situation. This led to a new version of the dataset composed of 4286 malware files and 89 features out of initial 150 features. The process is described by Algorithm 2.

Algorithm 2 Advanced Data preprocessing

Require: $DS_{unclean}$ //raw dataset
Ensure: DS_{clean} //cleaned dataset
PROCEDURE **cleanData**
$nrows \leftarrow size(DS_{unclean})$
$nfeatures \leftarrow length(DS_{unclean})$
$nrows2 \leftarrow 0$
$nfeatures2 \leftarrow 0$
$E_{num} \leftarrow 0$
if $nrows = 0$ **then**
  return NULL
else
  for $row = 1$ to $nrows$ STEP 1 **do**
    if $DS_{unclean}[row] = nil$ **then**
      $drop\ DS_{unclean}[row]$
      $nrows2 \leftarrow size(DS_{unclean})$
    end if
  end for
  for $col = 1$ to $nfeatures$ STEP 1 **do**
    if $DS_{unclean}[col] = nil$ **then**
      $drop\ DS_{unclean}[col]$
      $nfeatures2 \leftarrow length(DS_{unclean})$
    end if
  end for
  for $row = 1$ to $nrows2$ STEP 1 **do**
    for $col = 1$ to $nfeatures2$ STEP 1 **do**
      if $DS_{unclean}[row][col] = nil$ **then**
        $pass$
      else
        $w \leftarrow DS_{unclean}[row][col].split(" ")$ // splitting words/evidences by spaces
        $E_{num} \leftarrow length(w)$
        $DS_{clean}[row][col] \leftarrow E_{num}$
      end if
    end for
  end for
end if
RETURN DS_{clean}

4.4 Exploratory Data Analysis

In this work, the researcher conducted an Exploratory Data Analysis (EDA). The purpose of this was to critically perform initial data investigations in order to discover patterns,

anomalies, trends, and outliers in existing data by using visual and numerical techniques. The outcomes of this analysis helped in suggesting next steps of the research.

4.4.1 Understanding descriptive statistics on the dataset

This is a data profiling step meant to provide a quick understanding of data anomalies [137]. Figure 4.2 given below shows that the dataset values were not in the same range and had different mean values. Looking at the mean and the second quartile, it is clear that the data is skewed or asymmetrical. Therefore, standardization was required to build a highly performing model. Feature standardization involves converting them to the same scale [143]. This improves the performance of the model.

	api_resolv	cmd_exec	file_access	file_delete	file_drop	file_read	file_write	mutex_access	net_con	net_dns	...
count	3380.000	1014.000	3374.000	1069.000	1502.000	3319.000	1557.000	1407.000	188.000	277.000	...
mean	217.439	35.393	117.600	13.938	8.153	43.845	37.240	10.401	6.255	8.758	...
std	233.401	91.197	381.101	57.267	30.371	176.484	285.386	17.650	9.778	5.886	...
min	1.000	2.000	3.000	2.000	2.000	2.000	3.000	1.000	3.000	2.000	...
25%	27.000	9.000	3.000	3.000	2.000	3.000	3.000	1.000	3.000	4.000	...
50%	146.000	24.000	17.500	3.000	2.000	12.000	6.000	2.000	3.000	6.000	...
75%	358.000	27.000	61.750	9.000	6.000	24.000	15.000	5.000	6.000	12.000	...
max	1522.000	1159.000	8817.000	1293.000	902.000	4050.000	6999.000	147.000	120.000	29.000	...

Figure 4.2: Dataset descriptive statistics

4.4.2 Histogram analysis

From histogram analysis as shown in Figure 4.3, we see skewness and some extreme values. This implies that features required some standardization. Otherwise, the model built with such kind of features could give poor predictions.

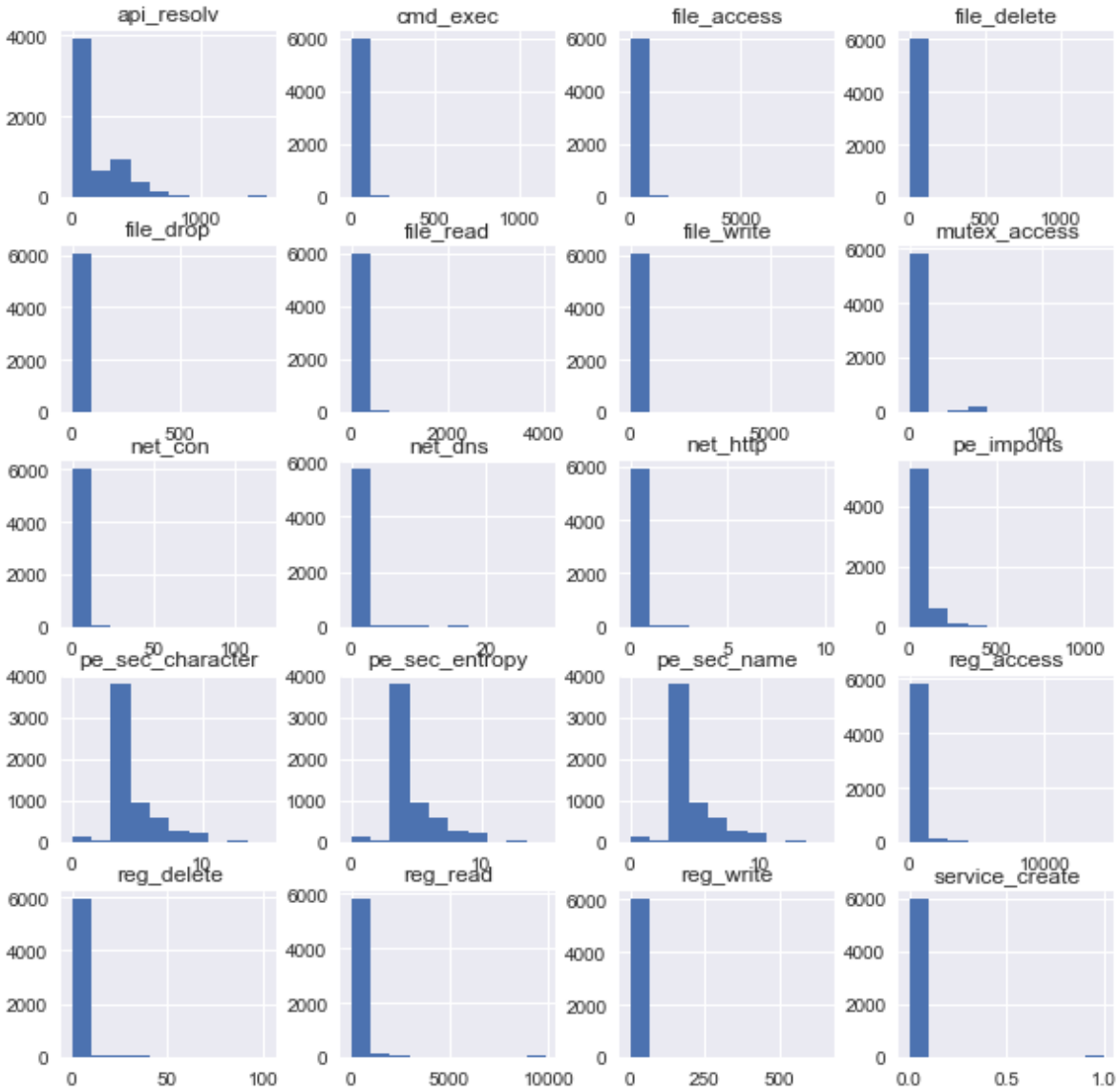

Figure 4.3: Histogram analysis of the first 20 features

4.4.3 Density plot analysis

Density plots also provide quick information on the distribution of the data. The given density plot is based on the first twenty features as shown in Figure 4.4 . It is clear that a considerable number of features have a skewed distribution. However, many statistical tests and machine learning algorithms require that features be normalized.

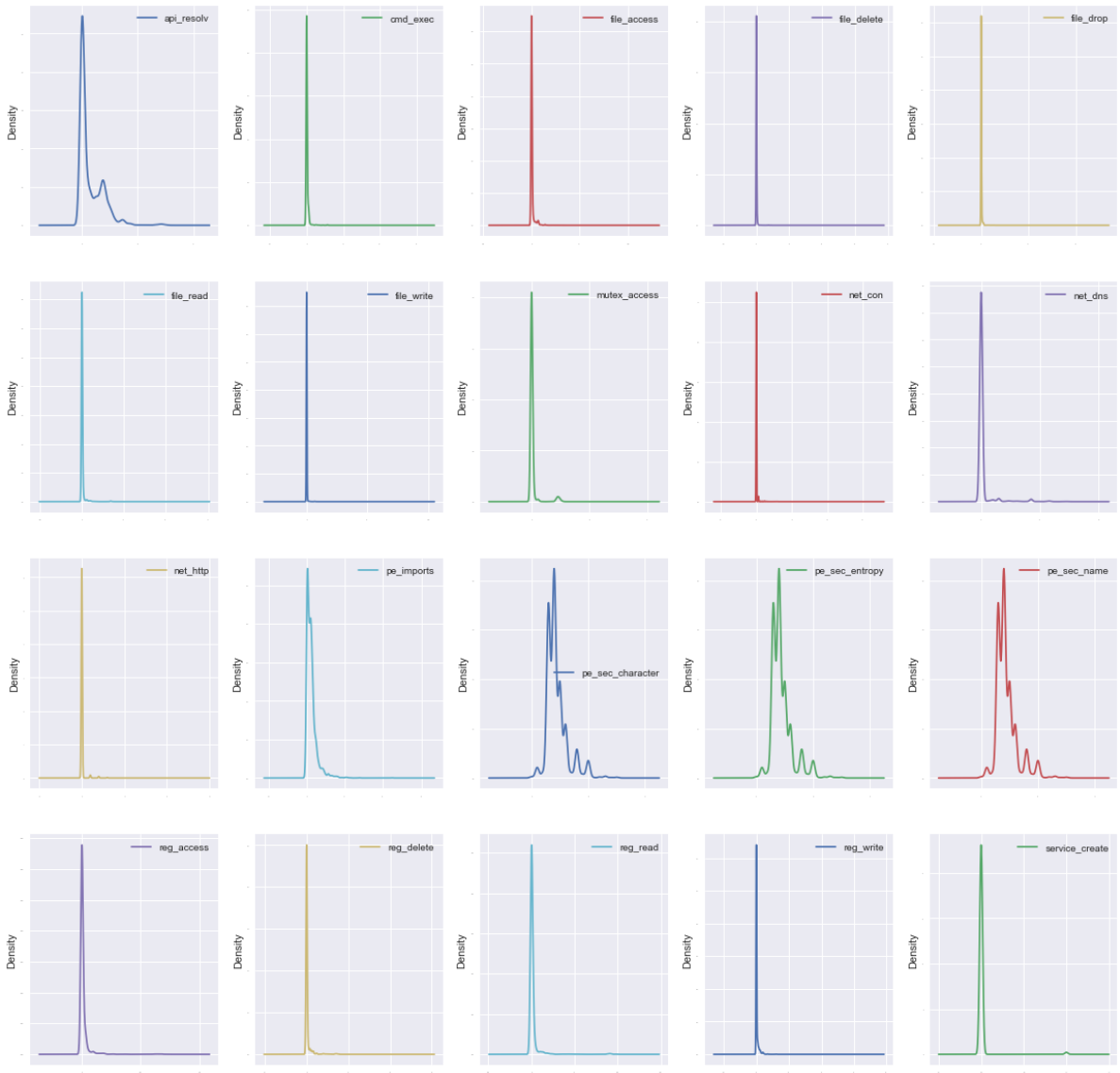

Figure 4.4: Scatter analysis of the first 20 features

4.4.4 Box plot analysis

Figure 4.5 describes a boxplot that shows how the values in the dataset were highly dispersed and that there were many outliers. Outliers are values that drastically deviate from others in the dataset. All observations below Q1-1.5(Q3-Q1) or above Q3+1.5(Q3-Q1) are outliers. This situation once again implies the need for standardizing our data before further processing.

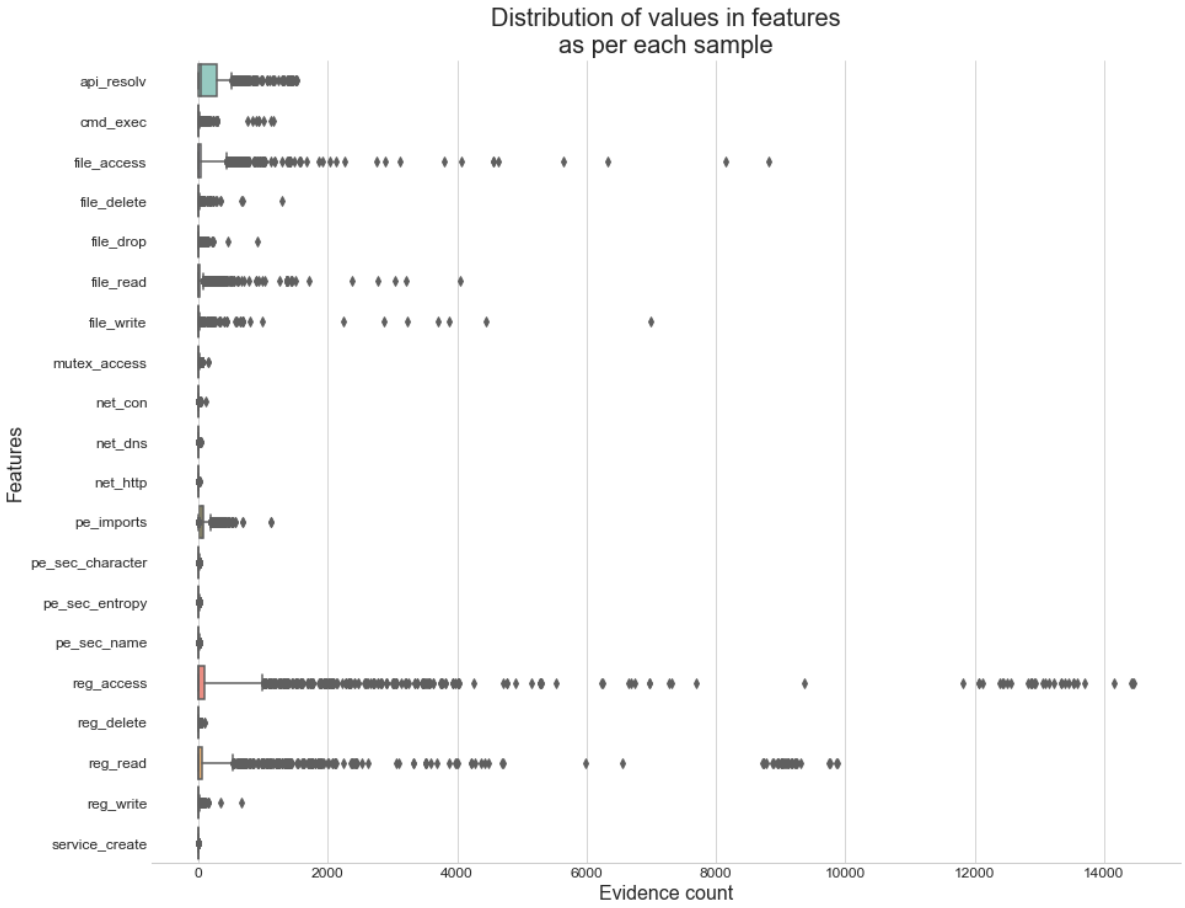

Figure 4.5: Box plot analysis of the first 20 features

4.4.5 Correlation matrix analysis

The snapshot in Figure 4.6 shows a matrix of correlation computed among features. This matrix is plotted in Figure 4.7 where different colour values and intensity help to better understand the correlation nature of features through visualization. The dark red colour shows a negative correlation whereas the grey colour shows a positive correlation. Such strongly related features are good candidates to be dropped in order to get good results. Strongly correlated features add no relevant information to other features. They just ruin the performance of the model. They can even increase the training time. They should, therefore, be removed. The correlation relationships [144] are highlighted in table 4.6 below:

Table 4.6: Correlation values

Value	Linear relationship
–1	Perfect negative
–0.70	Strong negative
–0.50	Moderate negative
–0.30	Weak negative
0	No linear relationship
+0.30	Weak positive
+0.50	Moderate positive
+0.70	Strong positive
+1	Perfect positive

	api_resolv	cmd_exec	file_access	file_delete	file_drop
api_resolv	1.000	0.105	0.145	0.084	0.120
cmd_exec	0.105	1.000	0.144	0.267	0.198
file_access	0.145	0.144	1.000	0.359	0.574
file_delete	0.084	0.267	0.359	1.000	0.379
file_drop	0.120	0.198	0.574	0.379	1.000
file_read	0.121	0.091	0.738	0.367	0.569
file_write	0.016	0.061	0.680	0.466	0.730
mutex_access	0.285	0.119	0.074	0.068	0.077

Figure 4.6: Computed correlations

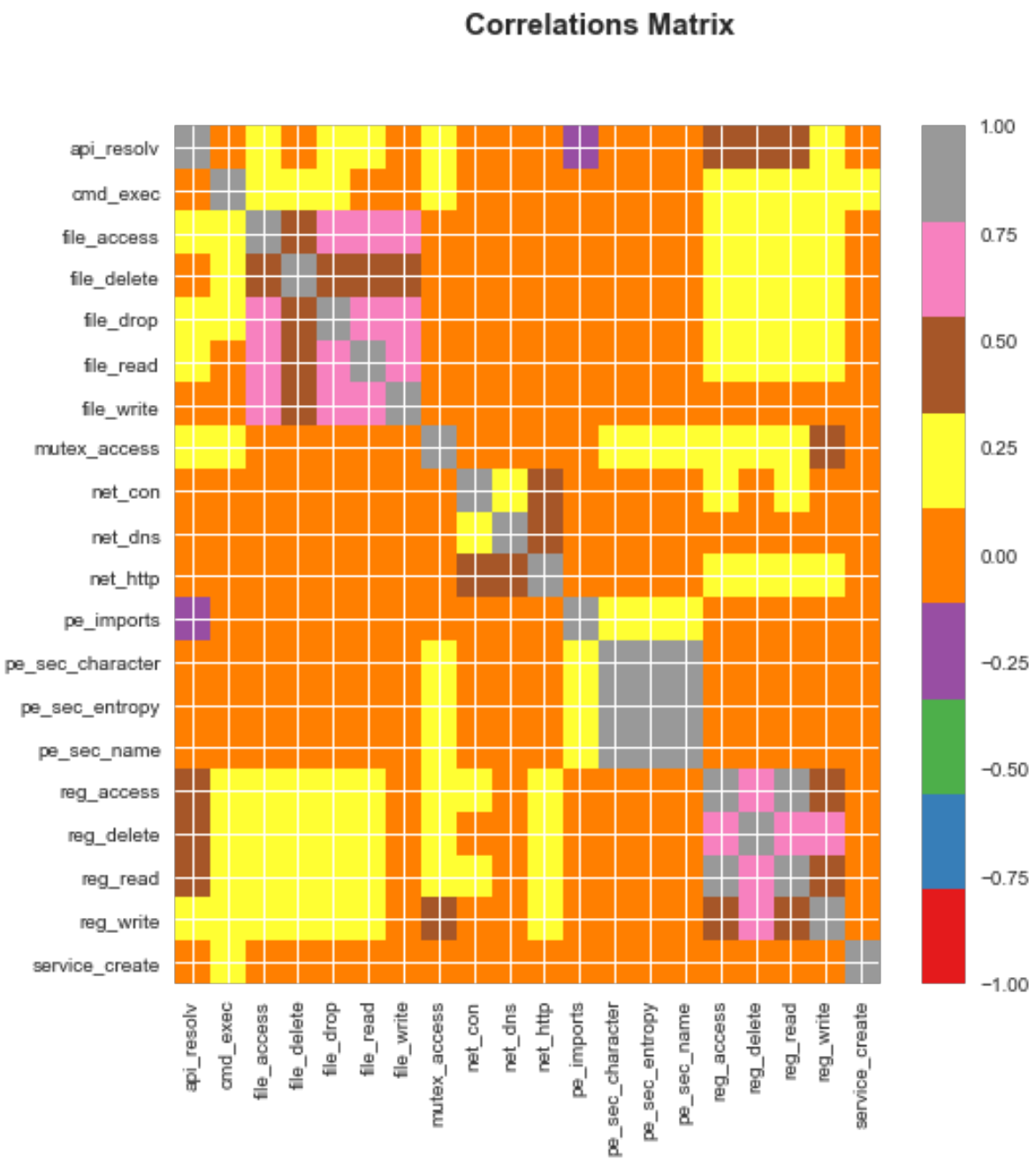

Figure 4.7: Features correlations matrix

4.4.5.1 Scatter matrix analysis

The scatter matrix provides a visual pairwise representation of a structured relationship between features. If they are features with a linear relationship, there will be highly correlated. Such features are candidates to be removed. In Figure 4.8 below, there are

many features in a strong correlation, which have to be dealt with properly.

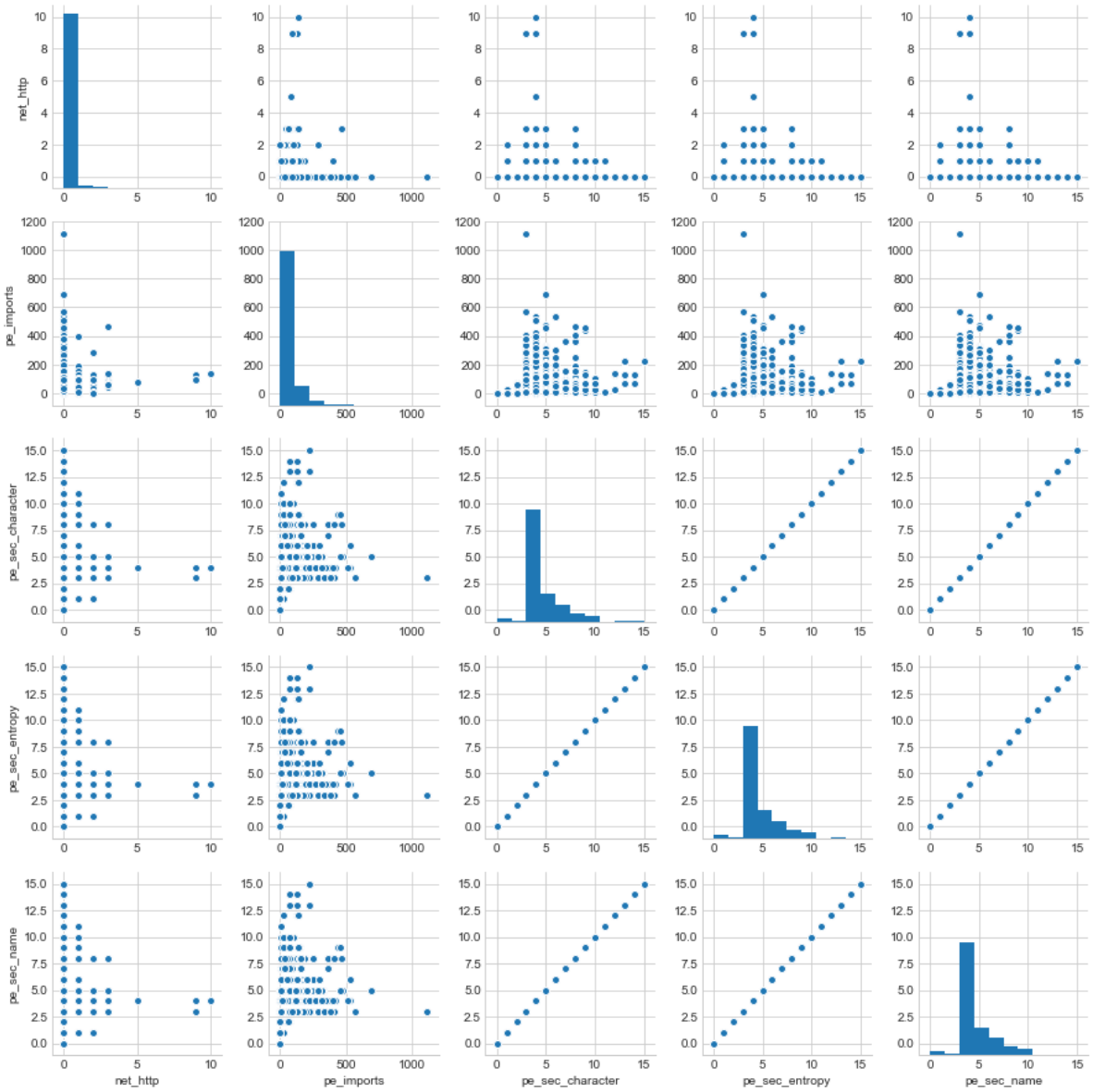

Figure 4.8: Feature pairwise scatter plot

From the observations done on the exploratory analysis of our dataset, we can deduct a hypothesis that there is no algorithm that can perform any better with poorly presented features. Although, features presented here were both static and dynamic, some anomalies were spotted. For instance, as shown in Figures 4.6, 4.7, and 4.8, it is clear that some features were highly correlated. Such features needed an advanced processing before any further steps in order to successfully answer the first research question and achieve

the first research objective. The next section will focus on experiments to prove this hypothesis as well as introduce a good feature engineering that will lead to the creation of good models.

4.5 NFE design

NFE should be able to select the best features based on their importances. It should as well be able to derive new features from existing ones in order to improve the prediction power of learning models.

Feature importance

In general, importance provides a score indicating the usefulness or value of each feature in the construction of optimized models. The more the weight of a feature, the greater its relative importance. Feature Importance addresses the issue of choosing which factors are the more important. It helps in creating a specific feature ranking. Removing less important features can increase both the accuracy and the speed of machine learning models [145].

There exist various techniques for choosing the best features and each technique might produce different values. Examples include Univariate Selection, Recursive Feature Elimination (RFE), Principal Component Analysis (PCA), Extra Tree Classifier, and Gradient Boost Machine (GBM).

This work's NFE approach is built on top of GBM feature importance computation. GB was chosen due to its efficiency in selecting best features that increase model prediction performance [139]. GB handles both numerical and categorical data.

Gradient Boost Machine

GB is an ensemble of decision trees. With GB, feature importance is calculated as the decrease of the impurity of the node weighted by the probability of reaching that node. The node probability can be calculated by the number of samples reaching the node, divided by the total number of samples [146].

Firstly, GB calculates feature importance for each decision tree as shown in Equation 4.1

$$ni_j = W_j C_j - W_{left(j)} C_{left(j)} - W_{right(j)} C_{right(j)} \tag{4.1}$$

Where,
ni_j is the importance of node j
W_j is weighted number of samples reaching node j
C_j is the impurity value of node j
$left(j)$ is a child node from left split on node j
$right(j)$ is a child node from right split on node j

The importance for each feature is then calculated as described in Equation 4.2 below.

$$fi_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{K all\ nodes} ni_k} \tag{4.2}$$

Where,
fi_i is the importance of feature i
ni_j is the importance of node j

A sample of one tree created by GB in our experiments is provided in Figure 4.9.

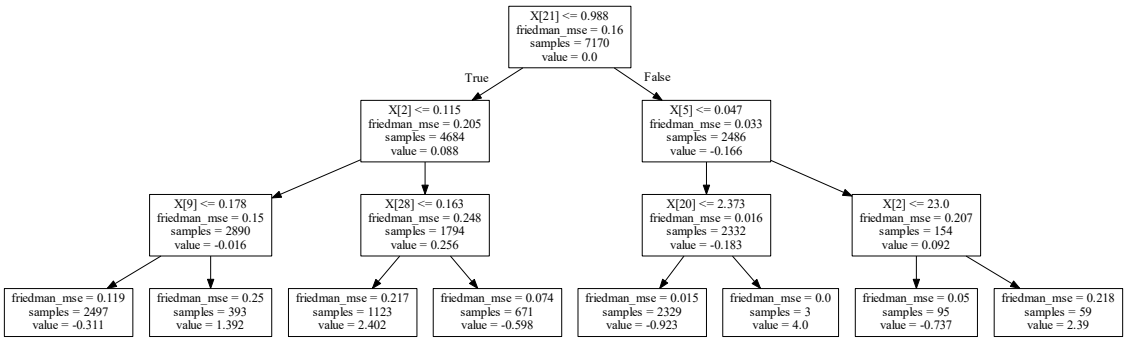

Figure 4.9: Example of a tree created by GB during the learning process

Overview of NFE approach

NFE initially builds on feature importances computed by GB, here denoted as $imp(f_i)$. It uses feature frequency $occ(f_i)$ and their ranking $rank(f_i)$ to deduce a mathematical relation between both frequency and ranking. This relation is here refereed as an estimator $e(f_i)$. A linear correlation $p(I, E)$ (See Equation 4.3) is computed to get the strength of the relationship between initial importances $imp(f_i)$ and the estimator $e(f_i)$.

The researcher assumes that there should be a positive relationship between feature occurrences and their initial ranking. The higher $p(I, E)$, the better new importances will be.

NFE is also able to drive new features from existing ones, when the selected features are not able to produce a satisfactory performance. NFE does this by applying mathematical transformation and aggregation functions such as mean, max, min, standard deviation, kurtosis, skew, sum, and count. For the success of this task, NFE normalises the dataset into parents and children subsets, if the dataset can work. NFE structure is described in Flowchart 4.10

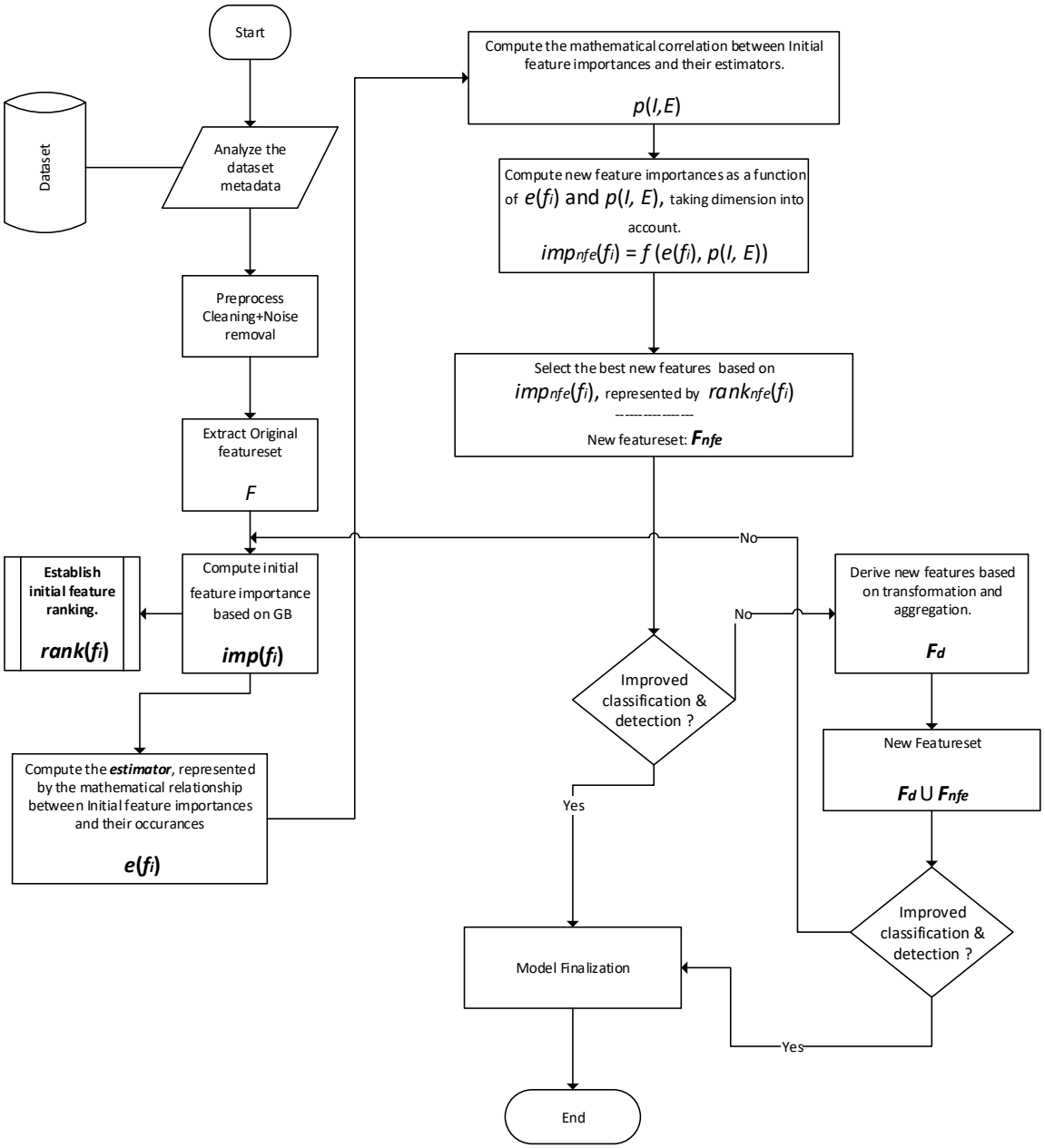

Figure 4.10: Overview of NFE Approach

The steps involved in designing a Novel Feature Engineering (NFE) approach are as follows:

1. Given a dataset DS of dimension d and size s, where d is the total number of features and s is the number of samples in the dataset.

We denote the initial featureset as:

$F = \{f_0, f_1, f_2, ..., f_i\}$, where $0 <= i < d$

Given the feature f_i, count the cumulative total number of occurrences in the dataset.

We denote total occurrence as $occ(f_i)$. Occurance is computed as follows:

$occ \leftarrow 0$
for $col = 1$ to d STEP 1 **do**
  for $row = 1$ to s STEP 1 **do**
    if $DS[col][row] =!nil$ **then**
      $occ \leftarrow occ + 1$
    else
      $pass$
    end if
  end for
end for

2. Compute initial feature importance for each feature f_i based on Gradient Boost algorithm.

   We denote initial importances as $imp(f_i)$.

3. Create initial feature ranking according to their importances.

   We denote initial ranks as $rank(f_i)$.

4. Compute the estimator value $e(f_i)$ for each feature. This value is computed based on feature occurrences and their ranks.

   $e(f_i) = \frac{occ(f_i)}{rank(f_i)}$

5. Find the strength of the association between initial feature importance $imp(f_i)$ and their estimators $e(f_i)$, represented by the correlation $p(I, E)$ as shown below:

$$p(I,E) = \frac{\sum_{i=1}^{d}(imp(f_i) - \overline{imp})(e(f_i) - \overline{e})}{(\sum_{i=1}^{d}(imp(f_i) - \overline{imp})^2 \sum_{i=1}^{d}(e(f_i) - \overline{e})^2)^{\frac{1}{2}}} \tag{4.3}$$

where $\overline{imp}$ is the mean of all initial feature importances. $\overline{e}$ is the mean of all feature estimators. d is the dimension of the dataset.

6. Compute the new NFE feature importances as follows:

$$imp_{nfe}(f_i) = f\left(e(f_i), p(I, E)\right) = \frac{e(f_i) * p(I, E)}{d} \tag{4.4}$$

where d is the dimension or total number of features.

7. Normalize individual features to the range between 0 and 1.

$$Norm(imp_{nfe}(f_i)) = \frac{imp_{nfe}(f_i)}{\sum_{i=0}^{d} imp_{nfe}(f_i)} \tag{4.5}$$

where $0 <= i < d$.

8. Create the new feature ranking $rank_{nfe}(f_i)$ based on newly computed importances $imp_{nfe}(f_i)$

   The higher $imp_{nfe}(f_i)$, the higher $rank_{nfe}(f_i)$.

9. If $c < d > 30$, where c is the number of classes and d is the total number of features:

   - Select first k top ranking NFE features f'_k based on $imp_{nfe}(f_i)$, where $k <= 30$
     New NFE feature set is:
     $F_{nfe} = \{f'_0, f'_1, f'_2, ..., f'_k\}$

   For a fast training process, if there is a large number of features, we propose that the new features to be selected should not be more than 30 if they provide a good performance. This assumption is based on observations done during various trial and error experiments.

10. Use the selected features in step 9 to create the model. If the performance of the model is poor, derive more features as shown in the next steps. The researcher assumes that the accuracy less than 90% can be considered as poor performance in malware classification and detection problems.

11. Let c be the number of classes, d the dimension of the dataset, and s the size of the dataset.

    If $c < d$ and $s >= 2^d$:

- partition DS into $DS_{1...j}$, where DS is the initial dataset, $DS_{1...j}$ are the normalised or related partitions extracted from DS, and j is the total number of sub-partitions.
- Generate aggregate features for DS_{pa} based on child DS_{ch} , where $pa <= j$ and $ch < j$.

  New derived feature set:
  $F_d = \left\{ f''_0, f''_1, f''_2, ..., f''_j \right\}$, where $j > 0$ is the total number of newly generated features.
- Combine derived features F_d with initial NFE features F_{nfe} to obtain the final feature-set F'_{nfe}.

  $F'_{nfe} = F_{nfe} \bigcup F_d$

At step 6 above, the new importances will produce a new ranking, from which the trial and error technique will be used to select the best candidate features for the model to achieve the optimal results. For example, in Figure 4.11, we present a comparison between initial feature importances based on GB and new feature importances produced by NFE technique on dataset 1. There is a slight change for which impact will be evaluated in the next chapter.

This NFE technique should be able to automatically select the best features based on assumptions in steps 10 and 11. Step 11 assumes that the dataset can be normalized in parent-children relationship.

Depending on the reported performance, features obtained at step 9 can be retained if a satisfactory level was achieved, otherwise, the approach should go to step 11 for further engineering.

After dataset normalization, new meaningful features will be generated by applying mathematical computations based on aggregation primitives such as mean, max, min, standard deviation, kurtosis, skew, sum, count.

Impact on features by NFE

Let us understand the changes in features brought in by NFE. Table 4.7 and Figure 4.11 show the comparison between initial top 30 feature ranks and importances according to

GB vis-a-vis their corresponding ranks and importances according to NFE. It can be seen that, NFE produced a number of modifications on the majority of features. The first four features remained the same(i.e ranks and importances are the same for both GB and NFE). However, from the fifth feature, changes started occurring.

Table 4.7: Top 30 important features and ranks according to GB and NFE

GB - Features	**NFE - Features**	**GB-Ranking**	**NFE-Rank**	**Importances(GB)**	**Importances(NFE)**
api_resolv	api_resolv	1	1	0.133474	0.290283
pe_imports	pe_imports	2	2	0.115194	0.187372
str	str	3	3	0.100998	0.128092
antivirus_virustotal	antivirus_virustotal	4	4	0.100268	0.071758
clamav	reg_access	5	13	0.052068	0.015838
packer_entropy	file_read	6	7	0.037033	0.031836
reg_access	packer_entropy	7	5	0.034343	0.04344
file_read	reg_read	8	6	0.031642	0.03557
reg_read	file_access	9	8	0.028477	0.031244
static_detection	pe_sec_character	10	22	0.0258	0.00295
file_access	pe_sec_name	11	9	0.022355	0.026298
persistence_autorun	pe_sec_entropy	12	17	0.020745	0.006206
packer_upx	clamav	13	19	0.018798	0.005988
static_pe_anomaly	static_pe_anomaly	14	14	0.018642	0.009334
reads_self	reads_self	15	15	0.018192	0.008584
injection_runpe	mutex_access	16	28	0.015063	0.001823
pe_sec_character	persistence_autorun	17	10	0.013553	0.022585
mutex_access	reg_write	18	16	0.011995	0.006705
reg_delete	packer_upx	19	23	0.011708	0.002775
reg_write	file_write	20	18	0.010708	0.006124
pe_sec_name	file_drop	21	11	0.010676	0.018283
pe_sec_entropy	static_detection	22	12	0.009438	0.017452
stealth_file	reg_delete	23	26	0.009261	0.002204
net_dns	cmd_exec	24	30	0.008856	0.001066
recon_programs	file_delete	25	33	0.0084	0.000646
copies_self	stealth_file	26	29	0.007785	0.001324
file_write	stealth_timeout	27	20	0.007773	0.004898
antiav_detectreg	injection_runpe	28	69	0.007305	0.000008
file_drop	copies_self	29	21	0.007138	0.004413
infostealer_mail	net_dns	30	39	0.007122	0.000355

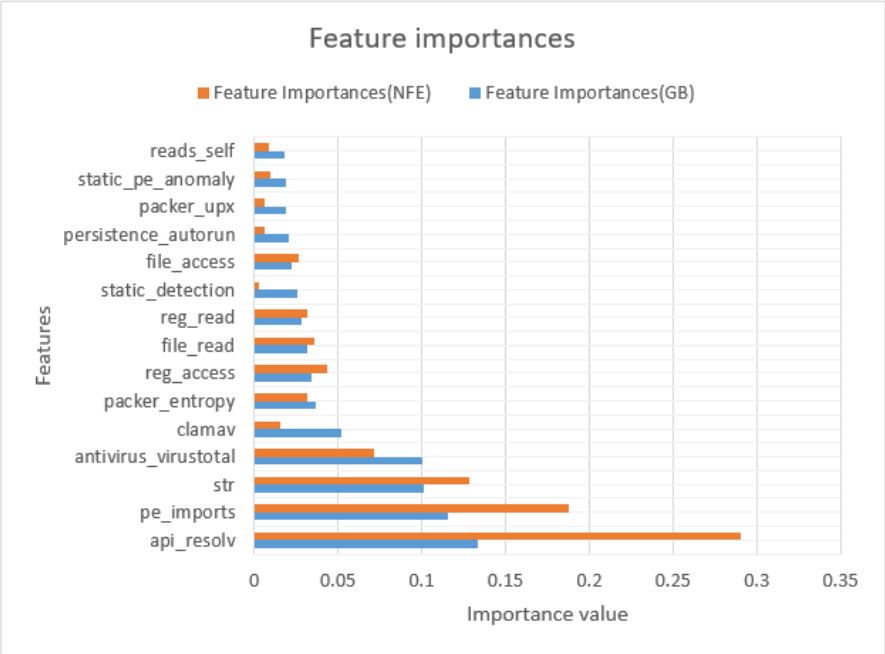

Figure 4.11: NFE - GB feature importances

Table 4.8 and Figure 4.12 show the changes in feature ranking and the total % of features affected by NFE. The statistics about the number of features replaced when applying NFE to initial features are shown in Figure 4.13. These replacements appear in the new NFE featureset, where not only initial GB features have been replaced, but also their importances and ranks have also been changed by NFE.

Table 4.8: % of features affected by NFE

Top initial features by GB	**Number of replaced features by NFE**	**Number of unaffected rankings**	**% of affected feature ranking by NFE**
5	1	4	20.0
10	2	4	60.0
15	3	6	60.0
20	3	6	70.0
25	4	6	76.0
30	3	6	80.0
35	4	6	82.9
60	8	6	90.0
70	6	6	91.4
80	2	6	92.5
89	0	6	93.3

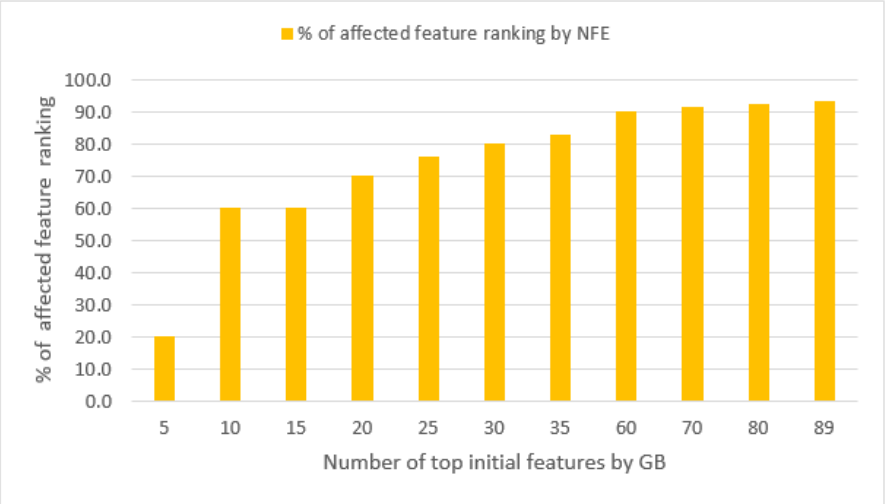

Figure 4.12: % of features affected by NFE

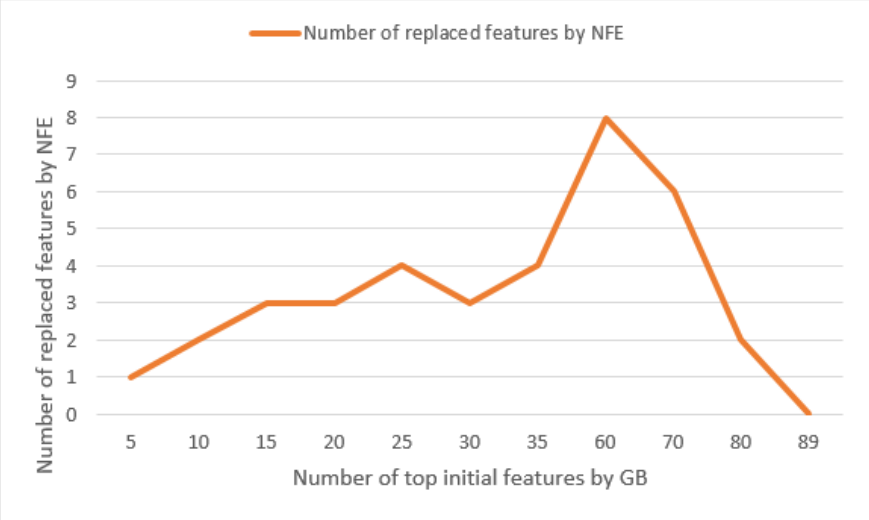

Figure 4.13: Number of GB features replaced in a new NFE featureset

Statistical Significance Analysis

In order to verify if the changes between GB and NFE differ significantly, Wilcoxon's signed-rank test and the Spearman's rank correlation coefficient have been calculated using data in Table 4.8. This table shows changes that occurred during the ranking process in given intervals. We tried to understand if there is a significance between initial GB features and the number of features unaffected by NFE. We use $\alpha = .05$ and $n = 11$.

We calculated the test static $Tstat = 1$. From the Wilcoxon Signed-Ranks Table, we find that $Tcrit = 10$ (two-tail test) at $n = 11$. Since $Tcrit = 10 > 1 = Tstat$, we reject the null hypothesis, and so conclude that there is a significant statistical difference between the changes produced by rankings of both GB and NFE. The computed Spearman's rank correlation coefficient ($r = 0.24$) shows a weak correlation, thus emhasizing a sigificant difference between GB and NFE features. Further experiments were conducted to verify if these changes make a difference in improving classification performance as shown in Chapter 5.

4.6 Summary

This chapter provided the details of experiments related to the creation of a Novel Feature Engineering (NFE) approach. A detailed discussion on malicious features and their extraction was provided using few variants of Win32/Potao malware. The researcher mainly adopted dataset 1 features to design NFE, as these features met the needed requirements for the study and the dataset itself had thousands of polymorphic samples.

The proposed NFE builds on top of Gradient Boost machine. NFE can compute new feature importances and produce a new ranking, thus enabling the selection of most relevant features that improve classification and detection performances of learning algorithms. Based on the structure of the dataset, NFE can also create or derive new features using statistical aggregation and transformation techniques. The effectiveness and efficiency of NFE are discussed in subsequent chapters on both classification (See Chapter 5) and detection (See Chapter 6). Also, the robustness of NFE on feature selection is presented in Chapter 5.

Chapter 5

Building a classification Model

Outline: In this chapter, we present the tasks related to the development of a classification model. A brief discussion of the dataset used is given in Section 5.1. In Section 5.2, we present feature processing process using NFE. In Section 5.3, we present the experiments conducted for the creation of a classification model. In Section 5.4, we discuss the impact of NFE features on different malware families. Section 5.5 provides a comparision between NFE and other feature selection techniques. A chapter summary is given in Section 5.6. This chapter is based on the work in [140].

5.1 Dataset 1

This dataset consisted of polymorphic malware grouped into five families. It initially had 150 features and 6032 malware samples. A full description is given in Appendix A. Table 5.1 shows a snapshot of this dataset after pre-processing.

The values in each column represent the number of proofs or the number of traces that a malware has exhibited a specific behavior (here represented by a feature). The missing values in some columns mean that there is no evidence that a malware has performed these activities.

After advanced preprocessing performed using Algorithm 2, all impurities were initially cleaned. In this case, the impurities represent all empty rows and all empty columns. The remaining dataset consisted of 4286 malware samples and 88 features. The distribution

of different familes available after preprocessing is shown in Figure 5.1. There were 1815 Crypto, 1142 Zeus, 629 shadowbrokers, 414 Locker, and 286 APT.

Table 5.1: Snapshot of Dataset 1. It contains malware samples only

api_resolv	cmd_exec	file_access	file_delete	file_drop	file_read	file_write	mutex_access	...	reg_read	reg_write	copies_self	str	label
37		12		2	9	3		...	300	8		221	APT
39		12		2	9	3	1	...	302	8		253	APT
								...				124	APT
14	32	12		2	6	3	1	...	18	2	3	186	APT
17		24			9			...	14	8		165	APT
22		3			3			...	22			303	APT
14		6			6			...	8	2		199	APT
15		3			3			...	16			161	APT
27		12		2	9	3	1	...	270			254	APT
18		12						...	22	6		279	APT

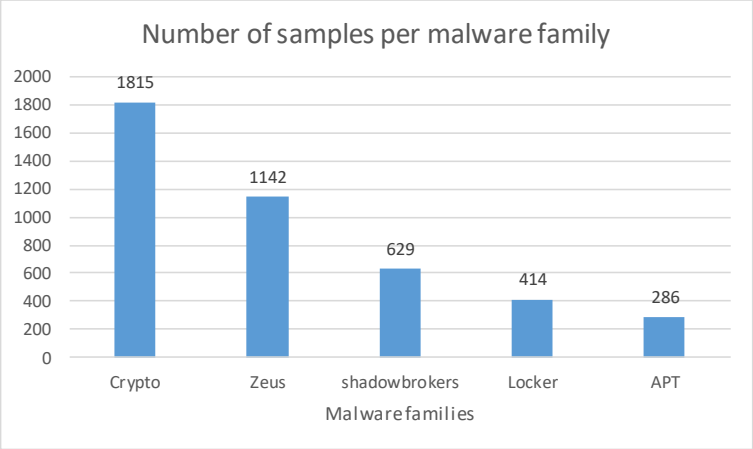

Figure 5.1: Frequency distribution of malware families after preprocessing

5.2 Feature processing with NFE

This feature engineering was based on NFE approach presented in Section 4.5. The results of this process are highlighted in Table 5.2.

- The dataset had 88 features. We computed their occurrences based on their frequency distribution.
- We computed initial feature importances using GB.
- Initial ranking is calculated based on GB importances.

- We computed the ratio between occurrences and initial ranks (here referred to as an estimator).
- We computed the correlation between the initial importances and estimators. This was equal to 0.913797827
- We then computed new feature importances as a function of estimator, correlation and dataset dimension. These are NFE feature importances.
- Finally new ranks based on NFE feature importances were generated. These are referred to as NFE ranks.

Table 5.2: NFE Feature engineering process on Dataset 1

Feature	**Occurances**	**Rank(GB)**	**Importances(GB)**	**Estimator**	**Importances(NFE)**	**Rank_(NFE)**
str	3426	3	0.100998	1142	11.72536	3
pe_imports	3341	2	0.115194	1670.5	17.15168	4
api_resolv	2588	1	0.133474	2588	0.26572	5
reg_access	2711	7	0.034343	387.2857143	3.976414	9
...	...	...	...	...	...	...
file_read	2537	8	0.031642	317.125	3.256046	10
reg_write	1092	20	0.010708	54.6	0.5606	15
file_drop	1141	29	0.007138	39.34482759	0.403969	16
packer_upx	694	13	0.018798	53.38461538	0.548121	20
mutex_access	1076	18	0.011995	59.77777778	0.613762	22
persistence_autorun	664	12	0.020745	55.33333333	0.568129	25
file_delete	800	38	0.004937	21.05263158	0.216156	27
net_con	159	51	0.001773	3.117647059	0.03201	29

On this dataset1, NFE produced features that were able to improve the classification models based on three supervised learning algorithm as discussed in the subsequent sections on experiments.

5.3 Classification Model

For this task, we splitted the dataset into a training set and a test set. The training set was used to train the proposed classifiers, and the test set was used to evaluate the performance of the classifier on unseen data. We used a 80 % / 20 % ratio when splitting data into training and test set respectively. To validate the the classifier during training, we used the standard cross validation technique with k folds = 10.

Three supervised classifiers were used for this task, namely KNN, LDA and GB. These classifiers can handle a multi-class problem as is the case for this dataset 1. Figure 5.2 describes the overview of classification model building process.

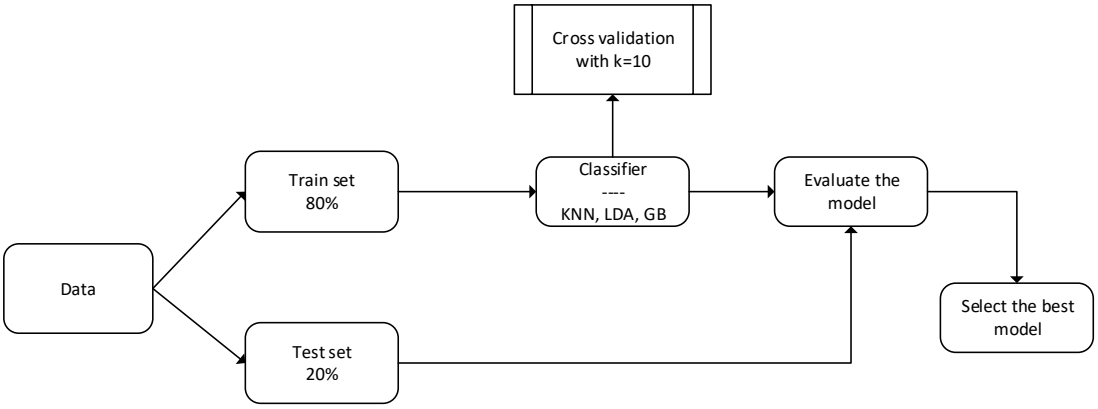

Figure 5.2: A schematic overview of the classification model building process

5.3.1 Experiment 1: Using Static features

This experiment is performed using only static features in the dataset. On the imbalanced dataset, the results obtained are shown in Table 5.3.

Table 5.3: Performance using static features only

Model	Accuracy	Recall	Precision	f1-score
GB	**0.87**	**0.89**	**0.89**	**0.88**
KNN	0.71	0.73	0.73	0.71
LDA	0.62	0.62	0.60	0.54

5.3.2 Experiment 2: Using Static features

This experiment is performed using only dynamic features in the dataset. The results obtained are shown in Table 5.4.

Table 5.4: Performance using dynamic features only

Model	Accuracy	Recall	Precision	f1-score
GB	**0.60**	**0.62**	**0.55**	**0.57**
KNN	0.53	0.55	0.54	0.54
LDA	0.55	0.57	0.62	0.55

5.3.3 Experiment 3: Using both static and dynamic features

This experiment is performed using all 88 raw features in the dataset. On the imbalanced dataset, the results obtained are shown in Table 5.5. Using all features on a balanced dataset slightly improved the performances as shown in Table 5.6. The accuracy of Gradient Boost Classifier increased by 5.4%.

Table 5.5: Performance using all raw features on the unbalanced dataset

Model	Accuracy	Recall	Precision	f1-score
GB	**0.82**	**0.82**	**0.84**	**0.81**
KNN	0.59	0.6	0.60	0.60
LDA	0.756	0.76	0.78	0.75

Table 5.6: Performance using all raw features on the balanced dataset

Model	Accuracy	Recall	Precision	f1-score
GB	**0.87**	**0.82**	**0.84**	**0.81**
KNN	0.74	0.6	0.59	0.59
LDA	0.67	0.67	0.76	0.67

5.3.4 Experiment 4: Using NFE features

Given that the performance attained in the previous experiment 1 are practically unacceptable, we adopted NFE approach for feature engineering to get improved performance. The performances achieved both on the imbalanced and balanced datasets are shown in Table 5.7 and Table 5.8, respectively.

Table 5.7: Performance using NFE features on the unbalanced dataset

Model	**Accuracy**	**Recall**	**Precision**	**f1-score**
GB	**0.93**	**0.93**	**0.93**	**0.93**
KNN	0.68	0.7	0.69	0.69
LDA	0.77	0.74	0.77	0.73

Table 5.8: Performance using NFE features on the balanced dataset

Model	**Accuracy**	**Recall**	**Precision**	**f1-score**
GB	**0.94**	**0.94**	**0.94**	**0.94**
KNN	0.79	0.66	0.66	0.66
LDA	0.68	0.7	0.81	0.74

The results are also shown in Figure 5.3. It can be seen that Gradient Boost has been improving in accuracy performances in all the experiments scenarios.

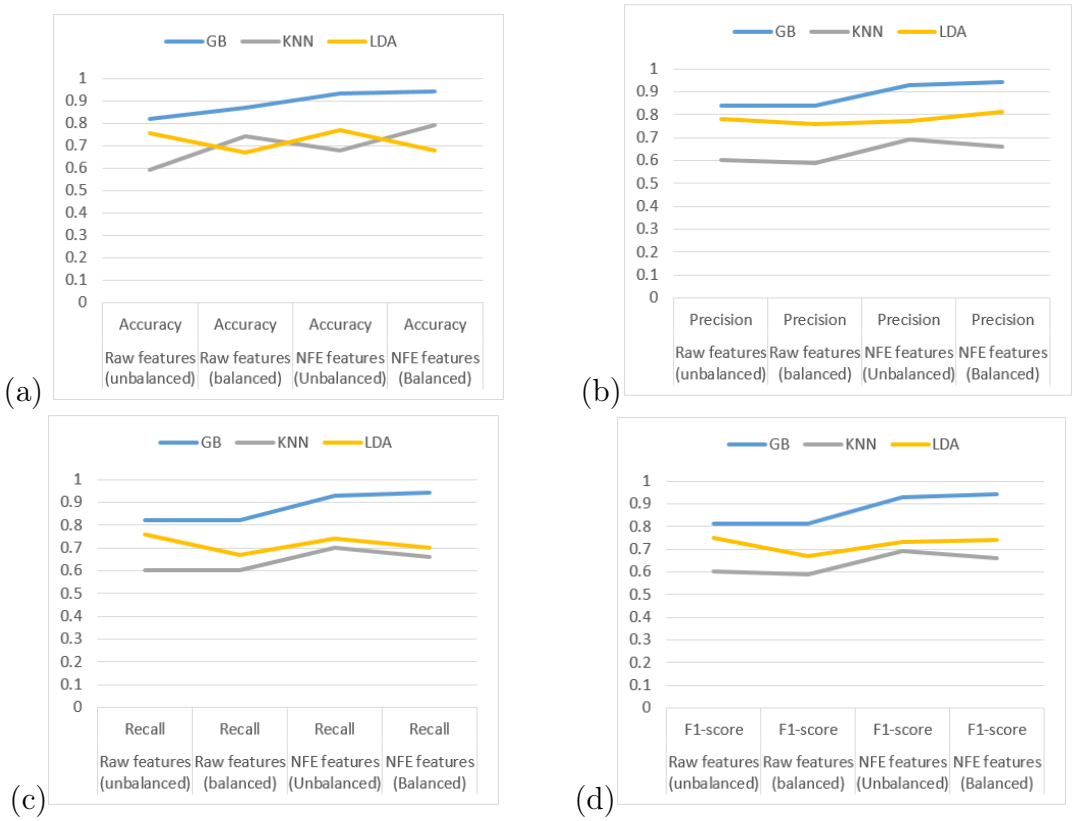

Figure 5.3: Changes in classification performances between various supervised classifiers. (a) shows variations in accuracy, (b) shows variations in precision, (c) shows variations in recall, (d) shows variations in F-score

5.4 Impact of NFE on malware

The number of correctly and incorrectly classified by GB model based on NFE is shown in Table 5.9. Based on results reported in Table 5.10, we see that shadowbroakers are the more affected by NFE, where among all samples classified as shadowbroakers, the model has the ability to specifically classify the real ones with a high precision of 99%. Again, based on the recall, we see that the model reached a level of 97%. This means that our model is very sensitive on classifying the real shadowbroakers, with the low False Positive rate of 0.03.

APT are the least affected by NFE, where among all samples classified as APT, the model has the ability to specifically classify the real APT with a high precision of 77%. Again, based on the recall, we see that the model reached a level of 87%. This means that our model is not very sensitive on classifying the real APT, with the False Positive

rate of 0.22.

Table 5.9: Confusion matrix showing correctly and incorrectly classified samples

	Predicted				
	APT	Crypto	Locker	shadow brokers	Zeus
APT	**41**	2	0	0	4
Crypto	3	**364**	2	1	11
Locker	6	9	**78**	0	3
shadow brokers	1	0	3	**120**	0
Zeus	2	3	4	0	**201**

Table 5.10: Various performance metric's scores on each malware family

	FN	FP	% RECALL	Precision	FN%	FP%
APT	6	12	0.872	0.774	0.128	0.226
Crypto	17	14	0.955	0.963	0.045	0.037
Locker	18	9	0.813	0.897	0.188	0.103
shadow brokers	4	1	0.968	0.992	0.032	0.008
Zeus	9	18	0.957	0.918	0.043	0.082

5.5 Comparing NFE with other feature selection techniques

One of NFE's strengths is the selection of features. This approach was compared to other three feature selection techniques to assess its robustness. The chosen techniques are Analysis of Variance (ANOVA), Recursive Feature Elimination (RFE) and Principal Component Analysis(PCA).

ANOVA

Analysis of Variance technique is used to analyse the statistical difference among group means in a sample. In the context of feature selection, ANOVA analyses the significance between each feature and the target vector. If features are categorical, ANOVA is done using Chi-square. If features are numerical, F-scores are computed for each feature,

where the higher the score, the more important the feature. To implement ANOVA, this research used F-scores because the processed dataset features were numeric.

RFE

Recursive Feature Elimination (RFE) technique uses the model accuracy to find the features that have more prediction power. It works by recursively removing features and building a model on the remaining ones.

PCA

Principal Component Analysis (PCA) is a data reduction technique that transforms a dataset into a more compressed form using linear algebra. The number of principal components chosen corresponds to the new number of features.

Results obtained are shown in Table 5.11. This table shows classification performance after selecting the top 30 relevant features as given by different techniques.

Table 5.11: Comparison of performances between NFE with ANOVA, RFE and PCA

Feature Selection Technique	**Accuracy**	**Precision**	**Recall**	**F-score**
ANOVA	91.1	93	93	93
RFE	73.7	78	76	75
PCA	62.9	56	55	54
NFE	**94**	**94**	**94**	**94**

Of the four techniques presented, NFE achieved better results for the 30 most discriminating features. NFE is slightly better than ANOVA in terms of precision, recall and F-score, but more better in accuracy than ANOVA. PCA provided poor performances in all measurements. All these differences are illustrated in Figure 5.4.

To further assess the effectiveness of NFE on feature selection, we conducted experiments using features selected by GB itself. The top 10, 20, 30, and 40 features were used. GB best classifier was used to build the model. Comparison results are given in Table 5.12. It can be seen that NFE has made a slight improvement in various feature selection settings. 94% has been the highest achievement for all performance metrics. NFE firstly reached this value achieved on Precision, Recall and F-score after only 20 top features. The same value has also been firstly reached by NFE after 30 to features. After 30 features, no much improvement has been noted on the highest performance.

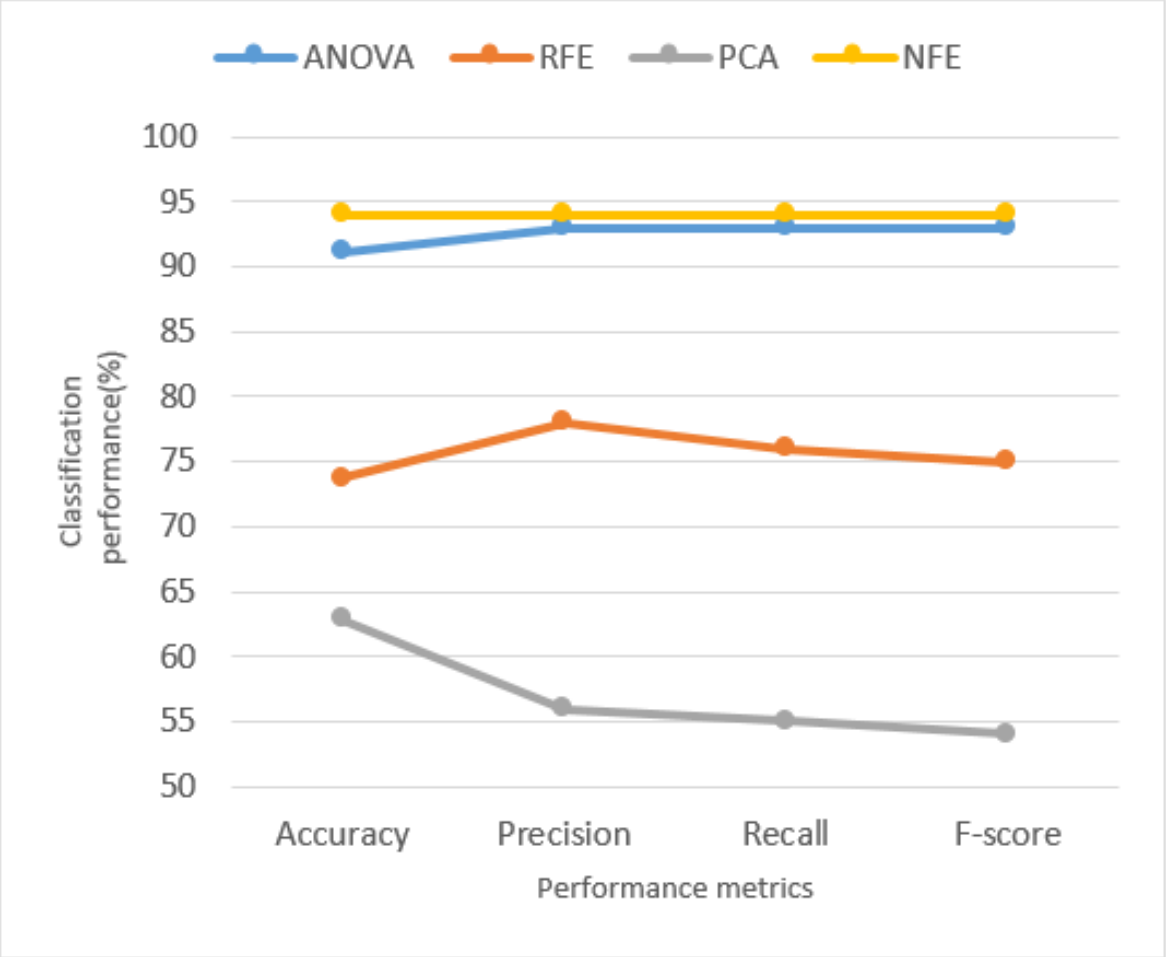

Figure 5.4: Comparison between NFE,ANOVA,PCA and RFE on feature selection performances

Table 5.12: Comparison of GB classifier performances between top selected GB and NFE features

		GB classifier performances			
Selected Features		**Accuracy**	**Precision**	**Recall**	**F-score**
GB features	Top 10	0.83	0.81	0.81	0.8
	Top 20	0.93	0.93	0.93	0.93
	Top 30	0.93	0.94	0.94	0.94
	Top 40	0.94	0.94	0.94	0.94
NFE features	Top 10	**0.91**	**0.92**	**0.92**	**0.92**
	Top 20	0.93	**0.94**	**0.94**	**0.94**
	Top 30	**0.94**	0.94	0.94	0.94
	Top 40	0.94	0.94	0.94	0.94

5.6 Summary

This chapter provided the detailed experiments related to the creation a malware classification model. The dataset used was discussed. Experiments conducted were also presented.

The first and second experiments were done using static and dynamic features separately. In this context, static features performed better comparing to dynamic ones. However, the general performance was not satisfying in terms of malware classification. Experiments three and four were done using a combination of raw static and dynamic features, as well as features produced by NFE(See Section 4.5). The results showed that with NFE, the best classier was able to achieve a classification performance of 94% on accuracy. This shows that a combination of both static and dynamic features can produce better and reasonable performances if a good feature engineering mechanism is applied on them. As the key charactristics of malware are captured in this context, the rate of false detections will also be very low. The impact of NFE on different malware families was also discussed. The dataset imbalance problem has also been taken into consideration during model creation process. The comparison made between NFE and other feature selection techniques showed that NFE is reliable as it produced better results for various performance metrics than other techniques.

Chapter 6

Building an Early-stage Malware Detection Model

Outline: In this chapter, we present the tasks related to the development of an early-stage detection model. A brief discussion of the dataset used is given in Section 6.1. In Section 6.2, we process features using NFE approach and generate new features. In Section 6.3, we introduce Multilayer perceptron neural network used to implement our Deep Learning model. We introduce the optimal hyper parameter settings that help optimize the output of a deep learning model. We develop the model and present the results. In Section 6.4, we discuss the early detection capabilities of our model. We demonstrate how our model can detect malware as early as possible before the damage can occur. A summary is given in Section 6.5. This chapter is based on the work in [147].

6.1 Dataset 2

This dataset contained a mix of benign and polymorphic malware. It had 10 input features. A full description of the features is given in Table 3.4 (See Section 3.4). The snapshot of this dataset is shown in Table 6.1. There were 594 benign samples and 595 malicious samples. Each file was executed up to 305 seconds. The total number of entries in the final dataset file was 343455, of which 173946 were benign and 169509 were malware. The dataset had a balanced distribution between malware and benign,

as shown in Figure 6.1. The train and test sets were originally provided by the dataset providers; we did not need to split them. The number of samples used for train set was 172323, and that for test set was 171132. The dataset had many outliers. For example, we can spot a lot of outliers in processor and memory activities for the first five seconds, as shown in Figure 6.2 and Figure 6.3, respectively.

Table 6.1: Snapshot of Dataset 2. It contains malware and benign files

vector	malware	total_pro	cpu_sys	cpu_user	max_pid	memory	rx_bytes	rx_packets	total_pro	sample_id	tx_bytes	tx_packets	swap	test_set
0	0	49	0.204887	0.088346	3036	650321920	3317908	2726	49	0	713908	1066	693821440	FALSE
1	0	44	0.044256	0.05838	3036	653524992	3319254	2736	44	0	715583	1079	698855424	FALSE
2	0	44	0.015052	0	3036	651018240	3320141	2744	44	0	716557	1087	698916864	FALSE
3	0	44	0	0.969	3036	651747328	3321028	2752	44	0	717516	1095	697835520	FALSE
11	0	43	0	0	3048	635617280	4987850	3603	43	304	2451770	765	682020864	TRUE
12	0	43	0.015519	0	3048	635641856	4988737	3611	43	304	2452713	773	682041344	TRUE
13	0	43	0	0	3048	635641856	4989310	3616	43	304	2453152	777	682041344	TRUE
163	1	42	0	0	3016	648421376	1994938	6221	42	658	2687169	5602	745324544	FALSE
164	1	42	0	0	3016	648155136	1996663	6244	42	658	2690329	5624	746049536	FALSE
165	1	42	0	0	3016	646934528	1998609	6269	42	658	2693978	5650	745943040	FALSE
104	1	45	0	0	3068	679383040	3121416	2991	45	873	163002	1353	767721472	FALSE
105	1	45	0	0	3068	678748160	3122227	2998	45	873	163717	1361	767721472	FALSE
106	1	45	0	0	3068	679079936	3122783	3003	45	873	164909	1372	767668224	FALSE

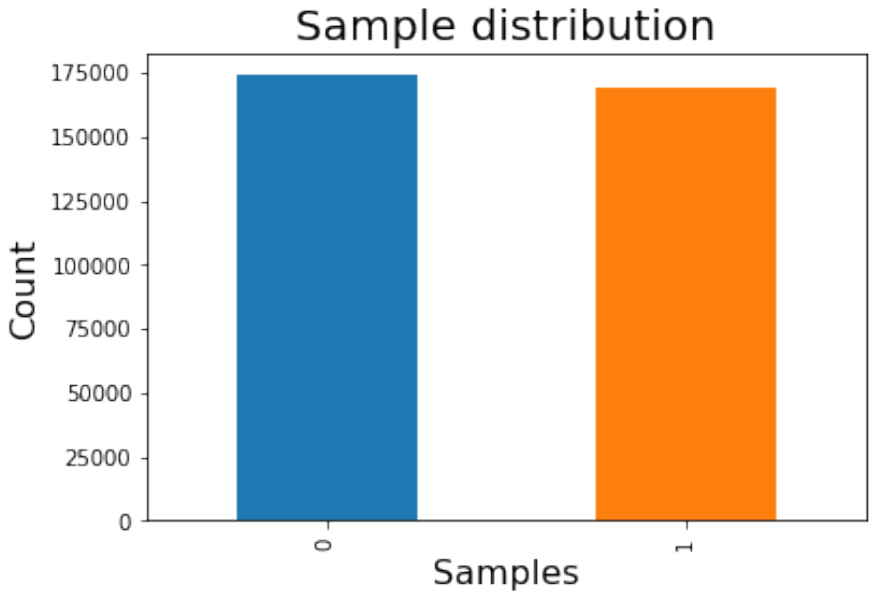

Figure 6.1: Distribution of malware and benign samples(1:Malware, 0:Benign)

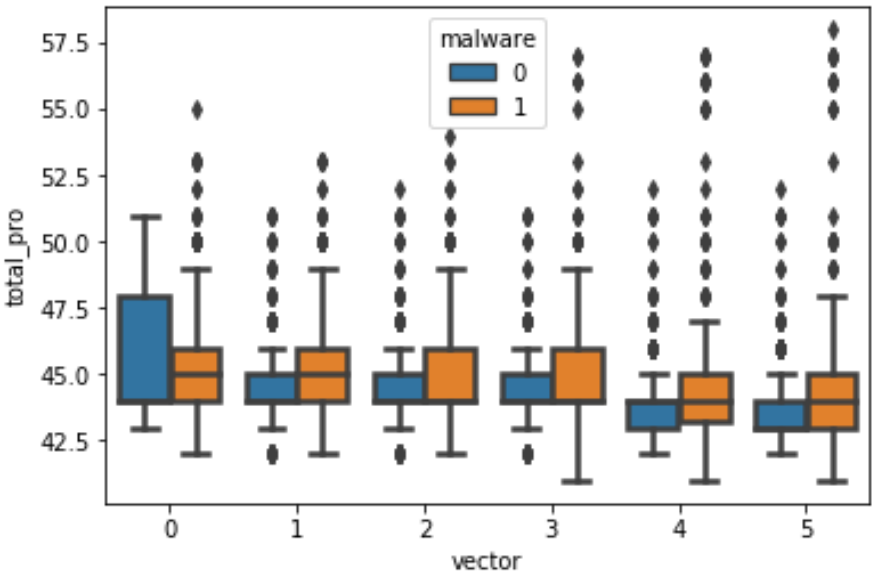

Figure 6.2: Processor usage in the first 5 secs

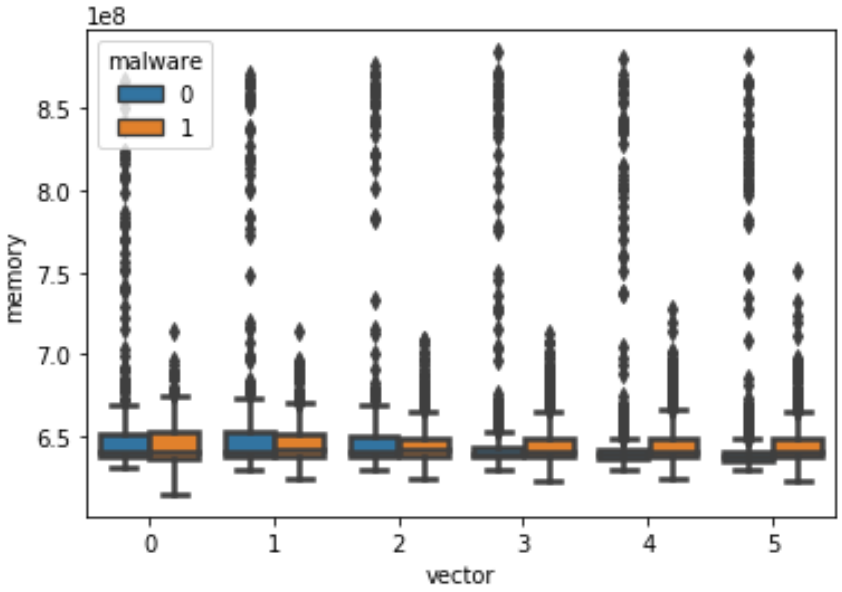

Figure 6.3: Memory usage in the first 5 secs

The values recorded during the analysis of malware and benign files are given in Figure 6.4.

	Benign		Malicious	
	Min.	Max.	Min.	Max.
Total Processes	43	57	44	137
Max. Process ID	3,020	26,924	3,020	5,084
CPU User (%)	0	100	0	100
CPU System (%)	0	100	0	100
Memory Use (MB)	941	8,387	939	1,957
Swap Use (MB)	655	2026	657	1,780
Packets Sent	326	1.1e−5	307	1.29e−5
Packets Received	4.07e−6	1.12e−9	4.07e−6	2.66e−8
Bytes Received (MB)	4.07	1,116	4.07	266
Bytes Sent (MB)	0.4	1,434	0.4	1,188

Figure 6.4: Minimum and maximum values of each input feature [107]

6.2 Feature processing with NFE

This feature engineering was based on NFE approach presented in Section 4.5. The results of this process are highlighted in Table 6.2.

- The dataset 2 had 10 features. We computed their occurrences based on their frequency distribution.
- We computed initial feature importances using GB.
- Initial ranking is calculated based on GB importances.
- We computed the ratio between occurrences and initial ranks (here referred to as an estimator).
- We computed the correlation between the initial importances and estimators. This was equal to 0.979767716
- We then computed new feature importances as a function of estimator, correlation and dataset dimension. These are NFE feature importances.

- Finally new ranks based on NFE feature importances were generated. These are referred to as NFE ranks.

Table 6.2: NFE Feature engineering process on Dataset 2

Feature	Ocuurances	Rank(GB)	Importances(GB)	Estimator	Importances(NFE)	Rank(NFE)
rx_bytes	172323	1	0.437545	172323	0.38384215	1
tx_bytes	172323	2	0.206025	86161.5	0.191921075	2
tx_packets	172323	4	0.086185	43080.75	0.095960537	3
max_pid	172323	5	0.065923	34464.6	0.07676843	4
total_pro	172323	6	0.037341	28720.5	0.063973692	5
swap	172323	8	0.017063	21540.375	0.047980269	6
cpu_sys	60703	3	0.092258	20234.33333	0.045071116	7
memory	172323	9	0.014602	19147	0.042649128	8
rx_packets	172323	10	0.008092	17232.3	0.038384215	9
cpu_user	42266	7	0.034967	6038	0.013449388	10

On this dataset 2, NFE generated additional features to amplify the predictive signals of the Deep Learning model. The initial highest ranked NFE features had provided a low accuracy of 63.18% on the test set. It was then crucial to generate new features from existing ones. New features were generated by applying mathematical aggregate functions ("sum", "max", "min", "mean", "count") on features. This process generated a total of 60 features, of which some of the most important were selected using NFE and used as input to the Deep learning algorithm.

6.3 Early-stage Detection Model

An early-stage detection model was developed based on early-stage activities of malware. The researcher used Multi-Layer Perceptron deep learning neural network. Neural networks are useful in modelling complex patterns in datasets. They use multiple hidden layers and non-linear activation functions. The input passes through several layers of hidden neurons and generates a prediction that represents the combined input of all neurons. Each layer should have a sufficient number of neurons and each architecture should have a sufficient number of layers to represent distinctions between the output classes [107]. A network architecture of the model used in this thesis is illustrated in Figure 6.5.

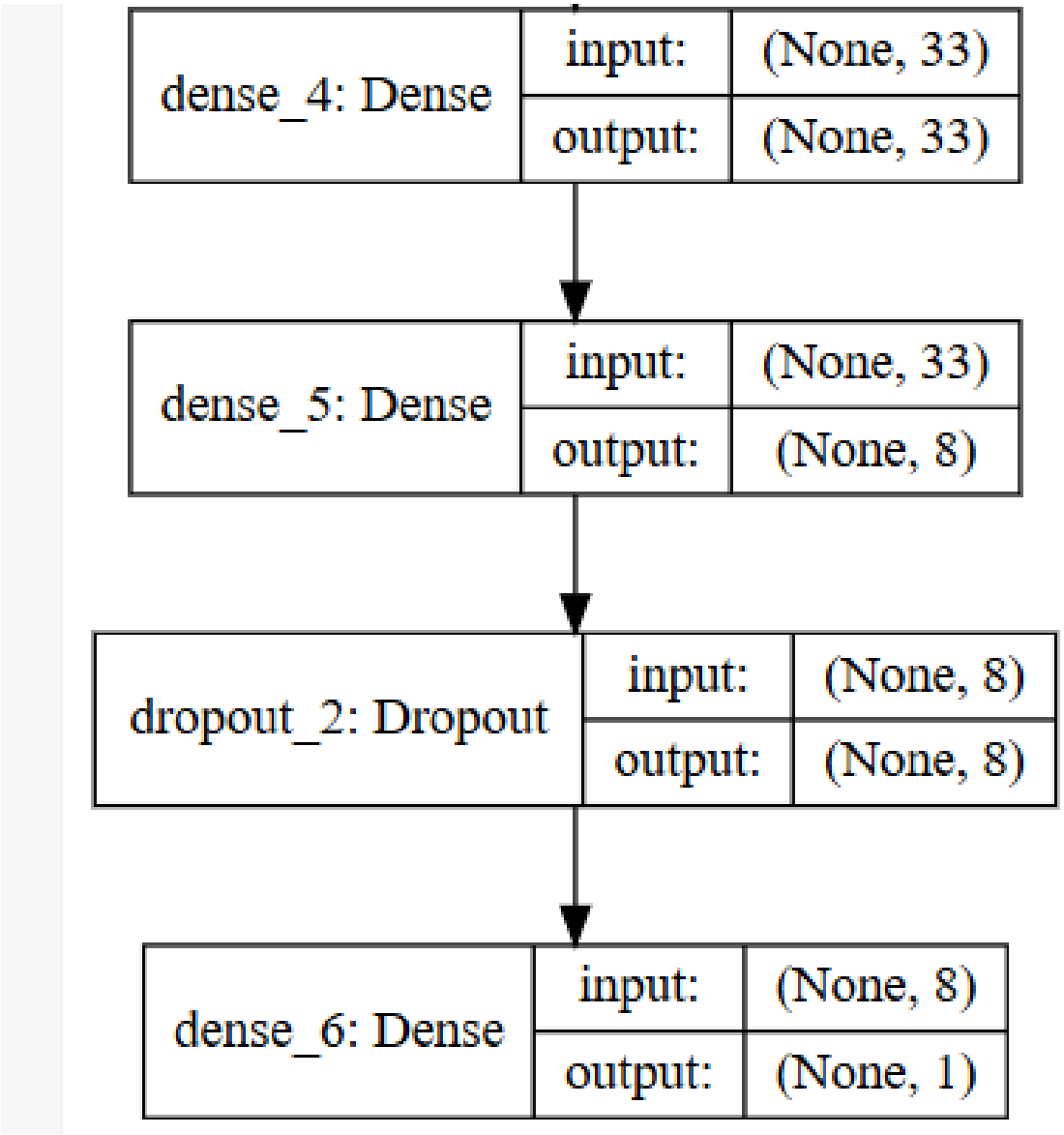

Figure 6.5: Model network architecture

As shown in Figure 6.5 above, the model has three fully connected layers(represented here by dense layers), two input and one output. There is only one dropout layer used to avoid overfitting.

6.3.1 Configuration of hyper-parameters

Each Neural Network has a set of parameters to configure for optimal results. These parameters allow the model to obtain a generalizable representation of the data. Although there are many parameters, the most commonly used include depth, hidden neurons,

epochs, dropout rate, weight regularisation through L1 and L2, and batch size. Depth refers to the number of layers used to learn the deeper parameters of the input. The output of each layer becomes an input for the next layer. The number of layers must be chosen in such a way to avoid overfitting. The size of a layer is represented by the number of hidden neurons responsible for the complexity of the model. Epochs represent the number of iterations performed by algorithm to work through the entire training dataset during the learning process.

Dropout is a regularization hyper-parameter necessary to avoid over-fitting. This works by turning off some layers during algorithm learning process across each iteration. It is used along with other regularization techniques such as L1/L2 on the weight and bias control. This improves the learning experience and reduces the classification error. L1 regularization penalises the weights that tend to grow to very large values, whilst L2 regularization limits the number of weights allowed to grow to large values [107]. The batch size defines the number of samples to work through before updating the internal model learned parameters [148].

During the model development process, our trial and error approach indicated that the dropout rate of 0.2, depth of 3 layers, 600 epochs, 8 hidden layers and the batch size of 100 samples yielded good results. Configurations are shown in Table 6.3. Nadam (Nesterov-accelerated Adaptive Moment Estimation) optimizer was used for adaptive learning setup. "Relu" activation function was used in the input and hidden layers. "Sigmoid" activation function was used in the output layer. Activation functions define the output of a neuron, given a set of input. Resulting values are mapped in ranges between 0 and 1 or -1 and 1, depending on the choice of the activation function. Nadam optimizer attempts to shape the model to be more accurate as possible by tying together the loss function and the model parameters, thus updating the model in response to the output of the loss function [149].

Table 6.3: Hyperparameter configuration

Hyperparameter	**Configuration value**
Depth	3
Epochs	600
Dropout rate	0.2
Weight regularisation	l1(0.01)
Batch size	100
Early stopping(Patience)	0

6.3.2 Model finalization

A python deep learning library called Keras was used to build and implement the Deep Learning model. Model performances during training iterations are shown in Figure 6.6 and Figure 6.7 respectively. The model achieved the highest accuracy of 99.66% on train set and 99.49% on the test set. The model also achieved the lowest log loss of 0.0378 and 0.0417 on both train and test sets respectively.

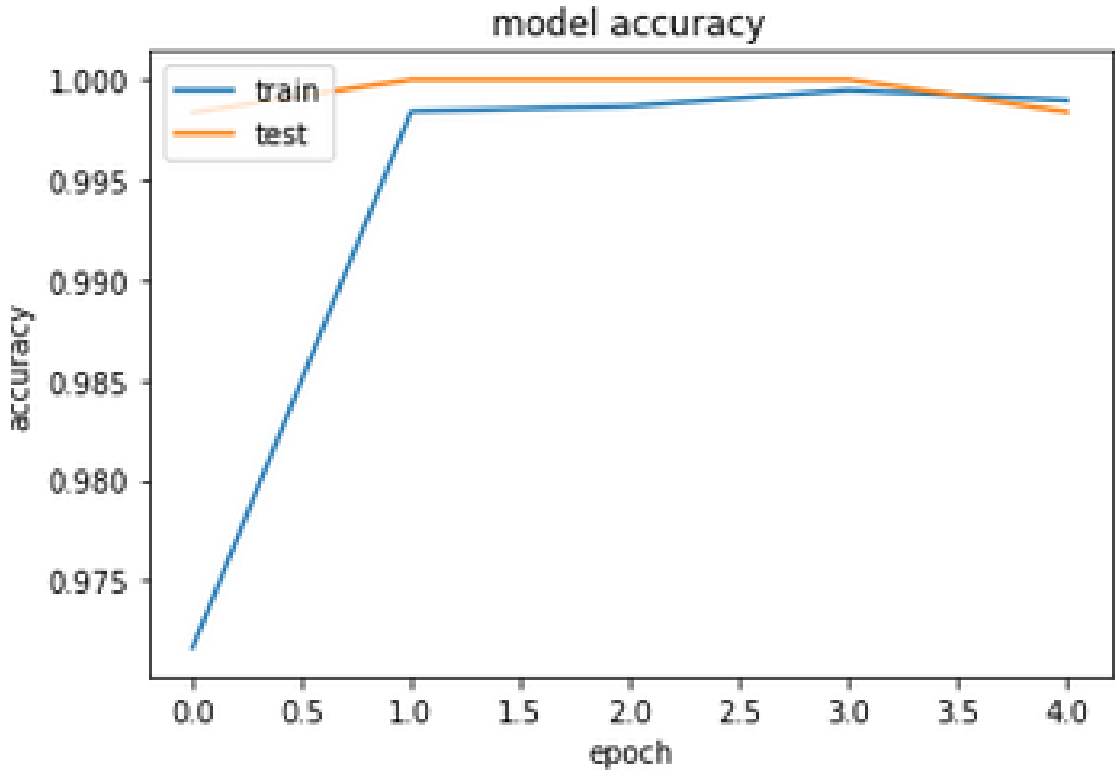

Figure 6.6: Model accuracy during training and testing.

This Figure 6.6 above shows the changes in accuracy when the model was learning during various iterations. Accuracy continued to increase until the 4th epoch. However, after this level, accuracy began to drop slightly. As performance began to deteriorate, the training process stopped. This happened at fifth epoch.

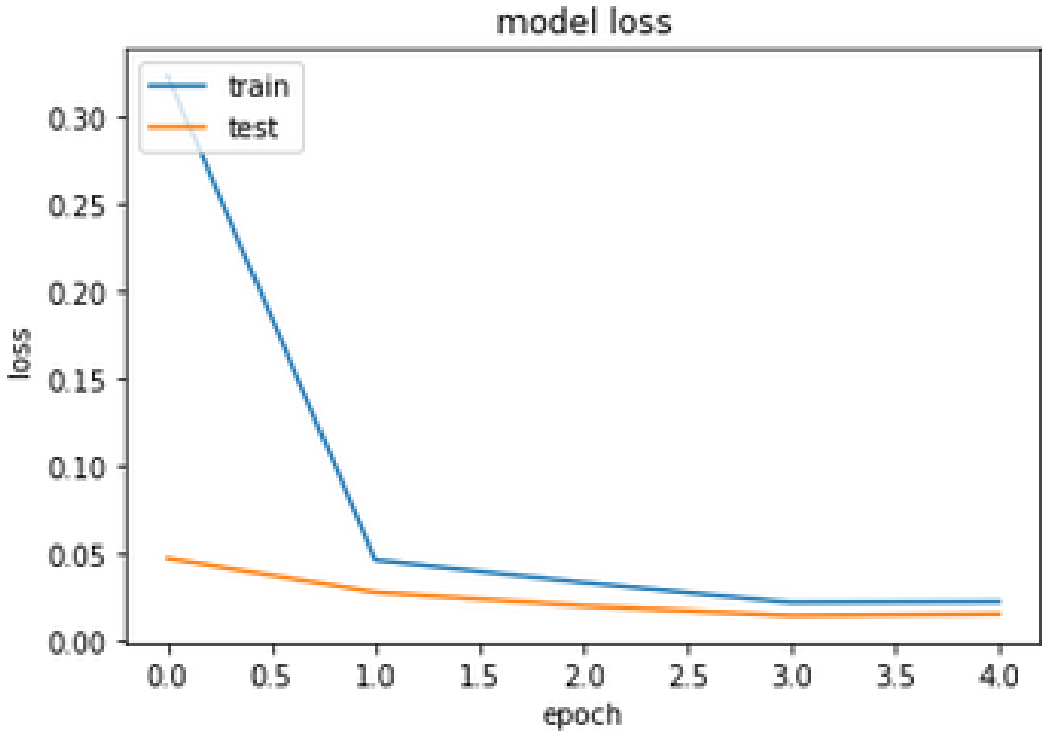

Figure 6.7: Model loss during training iterations.

This Figure 6.7 above shows the evolution of error losses during the learning process. A model works best if the error continues to decrease as during the training process. When the error begun to increase, the training process stopped. This happened at epoch five.

The detections made on test set are shown by the confusion matrix in the Figure 6.8. It is obvious that the model correctly detected all malware in the test set and missed out some few benign samples. This can be improved in further works.

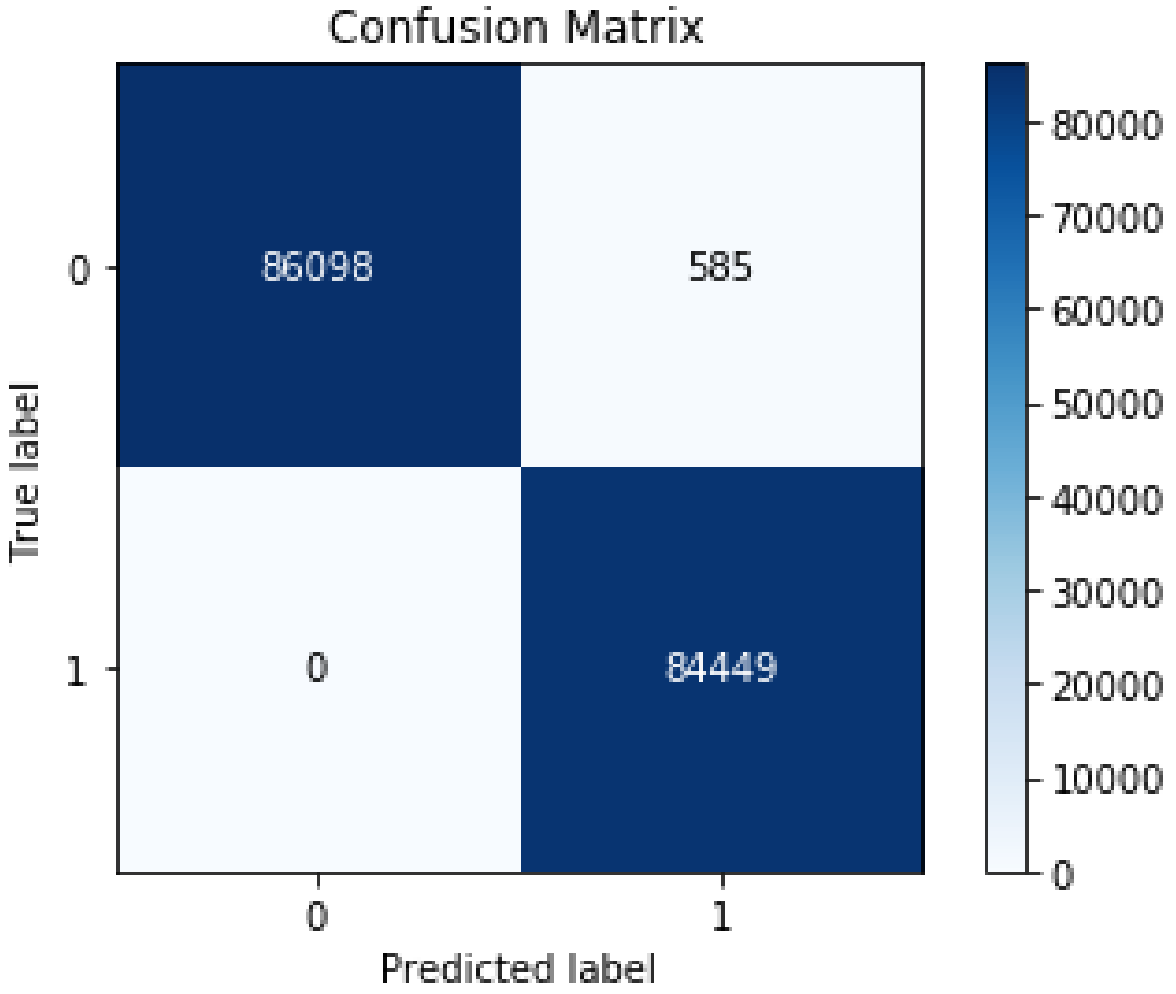

Figure 6.8: Detections made by the model during testing.(0: benign, 1: malware)

6.4 Simulating an early-stage detection scenario

The main goal is to identify the malware as early as possible so that adequate measures are taken immediately.

By selecting malware behaviours at random times, the model was able to achieve good results as shown in Figure 6.9. This is very important because the model is able to identify the malware at large scale whenever they appear in the targeted system.

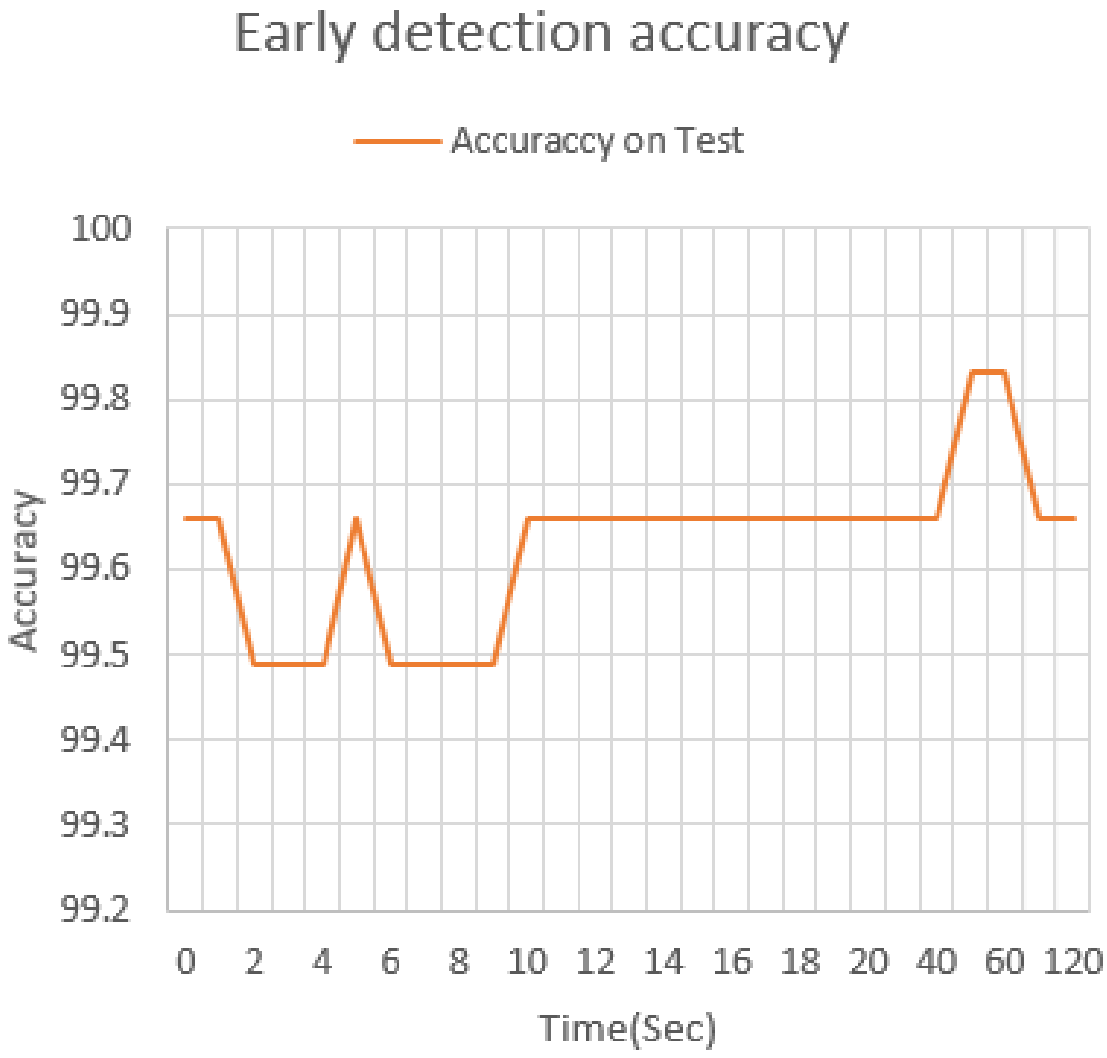

Figure 6.9: Detection accuracy at randomly selected seconds

This Figure 6.9 above highlights the model performance on different time stamps. The performance is not constant every second, but it remains high in general as it is greater than 99%. For an early detection of malicious programs, this performance is sufficient because it remains high even within a few seconds of file execution.

The results of this work were also compared with earlier work on the same dataset as given in Table 6.4 and Figure 6.10. The study used for comparison is the one done by Rhode et al. [107]. In their work, they used the same dataset. After 5 seconds of execution, their approach could predict with an accuracy of 94% if a sample is benign or malicious. The peak accuracy was 96% in less than 10 seconds. On the contrary,

our approach was able to detect whether an executable is malicious or benign, with an accuracy of 99.49% at the very beginning of file execution. Our model's peak accuracy was 96.66%. Our model's accuracy remained stable for at least 60 seconds into file execution. The fact that the model remains stable at different time scales is a good sign that it can predict the existence of a threat as early as possible.

Table 6.4: Early-stage detection performances at different time spans

Time(Seconds)	**Acc-Train(NFE)**	**Acc-Test(NFE)**	**Acc-Test by Burnapa**
0	99.66	99.66	80
1	99.83	99.66	85
2	99.83	99.49	87
3	99.66	99.49	91
4	99.5	99.49	93
5	99.33	99.66	94
6	99.66	99.49	93
7	99.66	99.49	95
8	99.66	99.49	95
9	99.66	99.49	94
10	99.66	99.66	95
11	99.66	99.66	94
12	99.66	99.66	95
13	99.66	99.66	95
14	99.66	99.66	95
15	99.66	99.66	95
16	99.66	99.66	94
17	99.66	99.66	95
18	99.66	99.66	96
19	99.66	99.66	93

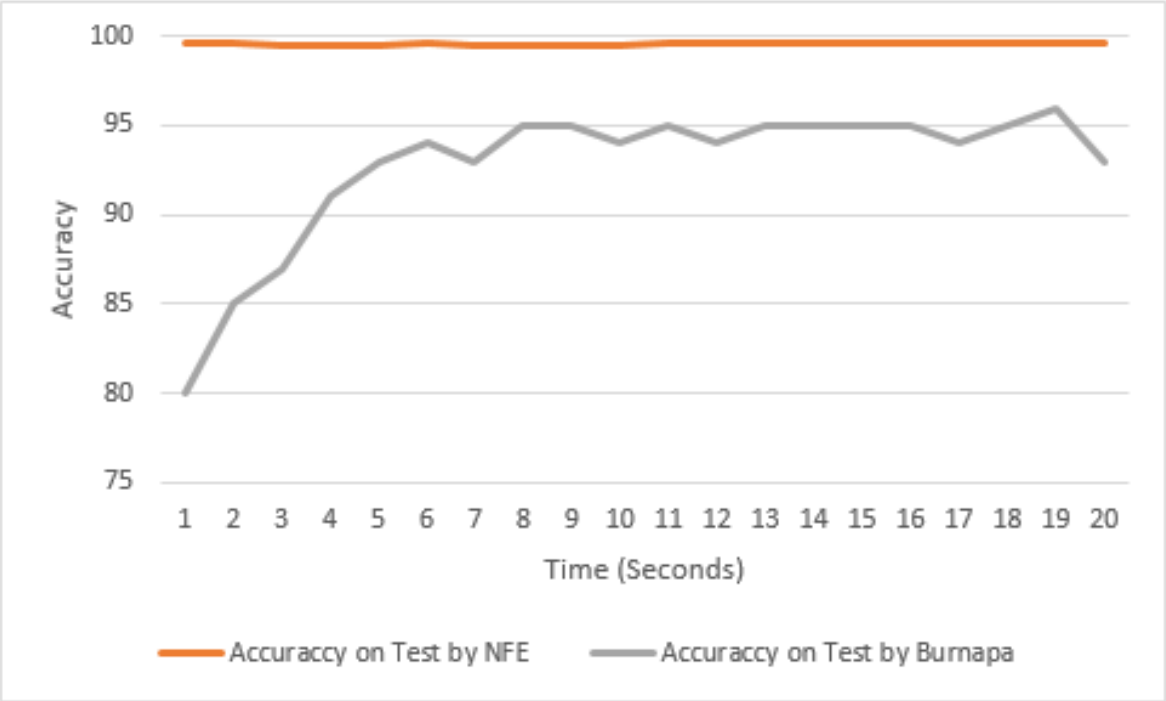

Figure 6.10: Comparative study between our approach and the previous study

6.5 Summary

This chapter provided the detailed experiments related to the creation an early-stage malware detection model. The dataset used was discussed. This dataset contained malware and benign files, of which behavioral activities were recorded during execution for a timestamp of 305 seconds. At each second of execution, an activity log was created.This log was then used to create this dataset. NFE (See Section 4.5) was used to process features. A deep learning model was created using a Multilayer perceptron neural network. The configuration of optimal parameters was discussed. The results showed that our Deep learning model was able to detect the malware in a short time and with a highest detection accuracy of 99% within the first 5 seconds of file execution.

CHAPTER 7

Discussion, Conclusion and Recommendations

7.1 Discussion

This research focussed on the problem of polymorphic malware. The first task was to explore, investigate, understand and identify gaps in the literature. The findings described below answer the research questions asked in Section 1.4.

The researcher found that many malware created daily are mostly variants of existing ones and they mainly target Windows Operating systems.

The researcher found that existing results in the literature had yet to be verified because it was difficult to compare for several reasons. For example, the datasets used are very different in most works. The features used are also different. The sizes of the datasets used are also not the same. It is therefore difficult to draw definitive conclusions about the weaknesses or strengths of existing approaches, especially those that use machine learning techniques.

The researcher found that accuracy was the main measure of performance used in many studies. Although this is a strong evaluation metric, it may not be sufficient because of the structure of the dataset. For example, if a dataset is unbalanced, there is a bias towards the class with the majority representation. To solve this problem, the researcher introduced a technique to deal with imbalance problems and also used other performance parameters such as recall, precision and F-score.

The study also found that most existing techniques are based on signature. As discussed in detail throughout this study, it is almost impossible for such techniques to

detect polymorphic malware.

This study also discovered that even the best known antivirus products may not detect some basic malicious samples. For instance, by submitting a sample to VirusTotal, major antivirus engines are contacted to scan, analyse and provide a report. It is rare to get a report where all the contacted engines have detected that the given sample was malicious. It is even possible to obtain an identification rate of less than 50%, where some renowned anti-malware products have failed to correctly identify the sample.

After noting the inconsistent malware classification and detection performances, the researcher found that this was mainly due to the features used in analysing and representing the sample.

In order to answer the first research question and accomplish the requirements of the first objective, the researcher sought that it was crucial to focus on feature engineering as the key to creating successful classification and detection models. The researcher sought that through feature engineering based on intrinsic features that characterize the key functionalities of malware, it was possible to model and represent the strategies, through which malware deliver their payloads or succeed in achieving their objectives.

To address the challenges discussed, this study proposed a Novel Feature Engineering (NFE) approach to improve the classification and detection of polymorphic malware. The study employed structural and behavioural features extracted from executables using static and dynamic analysis tools. The researcher also considered the effects of imbalanced datasets to improve performance. To deal with it, the researcher introduced an oversampling technique that allowed us to create a balanced dataset for better representation of all given classes. This has therefore contributed to more stable accuracy values. NFE produced a new feature set different from the initial GB featureset. To assess if GB and NFE features differ significantly, the researcher calculated Wilcoxon's signed-rank test and the Spearman's rank correlation coefficient. The results showed that there was a significant statistical difference between the changes produced by rankings of both GB and NFE.

To answer the second research question and address the second objective, The researcher created a classification model using a given multiclass dataset. Experiments were done using raw features and NFE features. The purpose was to show if NFE could improve the performances. Various evaluation metrics were used, namely Accuracy, Recall, Precision and F-score. Recall was used to measure the sensitivity of the built classifier in classifying relevant classes of malware. This helps evaluate the rate of false negatives. According to obtained results, GB classifier was able to classify malware with

an accuracy improvement of 12% from 82% to 94% (See results in Table 5.5 and Table 5.8). This has been possible by using features created by a novel technique referred to as NFE in this research.

To accomplish the third objective and answer the third question, the researcher created an early-stage malware detection model. Multilayer perceptron technique was employed based on NFE features. We demonstrated how a good detection model should detect malware as earlier as possible. The robustness of this model is explained by the good performances achieved at differing time scales. The accuracy was slightly greater than 99% from the first second of file monitoring during execution. Within the first five seconds of file execution, the model was able to achieve a detection accuracy above 99%. This accuracy remained in consistent ranges throughout the execution of the files in different time settings as shown in Table 6.4. This was a big improvement when compared to previous work as also shown in Table 6.4.

The findings have been compared to previous ones and it was proved that this work achieved better performance than those of existing researchers as discussed in Chapters 5 and 6, respectively. Other performance comparisons are shown in Table 7.1 below.

Table 7.1: Classification and detection results comparison

Technique	**Accuracy**	**Precision**	**F-score**	**Recall**	Logloss
Meng at al. [102]	98%				
Yuan at al. [42]	94.83%		94.66%		
Bojan at al. [104]		85.6%		89.4%	
Kolosnjaji at al. [105]	97.5%				
Rhode at al. [107]	96%				
Our detection model	99.66%				0.0378
Our classification model	94%	94%	94%	94%	

The overall contributions of this research are justified by the achievement of the research objectives. A feature engineering technique has been presented. The particularity of this technique is that it provides a unique representation of data, it improves feature selection and creates the additional input features necessary for the success of a learning model. With NFE, the researcher, therefore, developed classification and early-stage malware detection models.

7.2 Applicability of the results in Software Engineering domain

This research extensively used machine learning to address a variety of issues related to the classification and detection of malware. Various actions were performed to come up with effective Machine Learning models. The overall process from raw executable file analysis to the creation of the models is described in the sequence diagram shown in Figure 7.1.

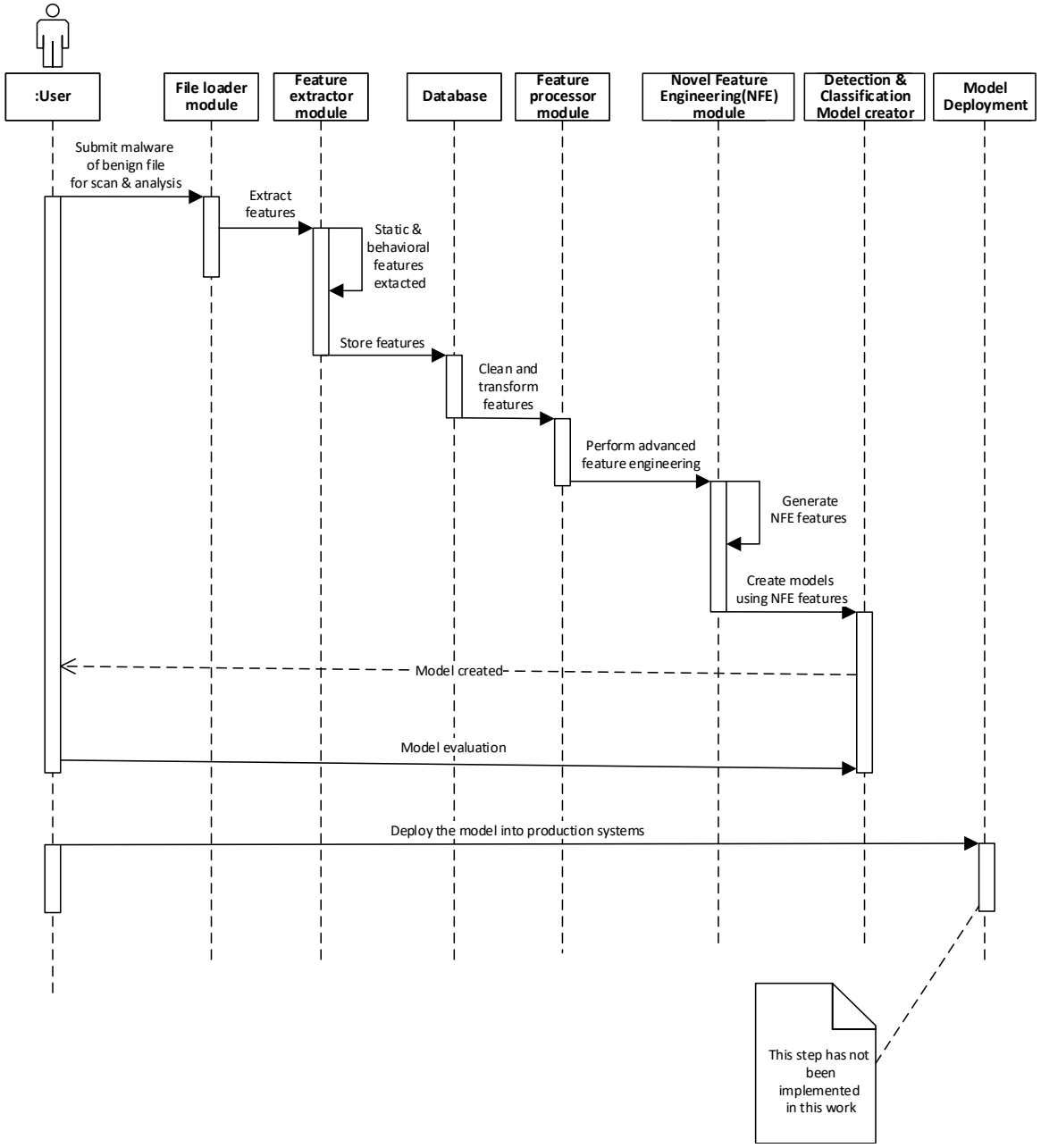

Figure 7.1: Sequence Diagram showing the process from raw file analysis to model creation and model deployment

Once a model is created, it is saved for later use. Before deployment, a model is used locally to classify and detect malware. As seen in Figure 7.1 above, model deployment activity was not implemented in this research. This is part of the future work.

By deploying a model, it is made accessible to be consumed. This means that the

model is integrated into other software products. This is mainly where the software engineering community will benefit from the results of this research. As this research is about the security, the results of this work target mainly anti-virus or anti-malware industry. In this context, by integrating or embedding these models into their products, the features provided by these models will add on significant value to their software in terms of improved security management capabilities, reliability, and scalability. This will ensure an optimal level for the protection of both applications and enterprises.

To efficiently and effectively achieve this integration on deployment, an appropriate API will be developed for software and intuitive interfaces created for people. Apart from API, the deployment could also be realized through other emerging techniques such as microservice architectures [154] or containerization [154], by which services provided in this research could be later queried or integrated into other production systems.

In general, users will be able to scan files for malware or benign through the use of integrated systems, and other systems will also have access to API services in order to scan for malware.

7.3 Limitations

However, the researcher also encountered some limitations. The main one was getting a malware dataset for analysis. It was not easy to find filtered samples and their variants. Samples provided by other researchers and online repository contain a mixture of malware with their variants and sometimes with benign samples included. Some samples were inconsistent and not be labelled. Some authors of interesting papers could not provide their samples for reproducibility.

Another limitation of this research in on model deployment. Due to time constraints and the initial objectives of the study, the researcher did not deploy the models as a piece of software. However, this will be done in future works as a means to apply the discoveries of this work at large scale.

7.4 Conclusion

This study proposed different novel techniques in terms of classifying and detecting malware. A good malware classification and detection system should be able to classify even the polymorphic variants that can appear in any form. Machine learning works well in building these models. However, for any algorithm to work at its best, it requires a

well-balanced, highly pre-processed and well-presented dataset. This can be accomplished through feature engineering. This work presented a novel feature engineering approach, which yielded good results on a complex dataset in classifying polymorphic malware. The achieved performances in multiple scenarios of the experiments by the Gradient Boosting Classifier showed an overall improvement of 12% in accuracy. Further, the study proposed an early-stage malware detection approach using deep learning. The findings showed a good detection performance on different time stamps, the lowest accuracy being 99.49% and the highest 99.66%. All models built were based on the new Novel Feature Engineering (NFE) approach.

We believe that if the techniques proposed in this study are implemented in end-to-end protection applications, the classification and detection of malware, especially polymorphic ones, should be considerably improved.

7.5 Recommendation

1. Feature Engineering is wide. Future research should continue to deepen it to improve understanding of the phenomenon, for better proactive protection.

2. Future research should consider how the next generation antiviruses (NGA) can be sandboxed and equipped with continuous analysis and learning mechanisms.

3. In the future, researchers should create a virtual execution layer (VEL) between Operating System and the applications. Sensitive applications should first be run from VEL, which must itself contain NGA. This will ensure a maximum level of protection.

Bibliography

[1] Z. A. Bhuiyan, T. Wang, T. Hayajneh, and G. M. Weiss, "Maintaining the Balance between Privacy and Data Integrity in Internet of Things," in *Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences*, 2017.

[2] R. Pasha, Y. Prathima, and L. Thirupati, "Malwise System for Packed and Polymorphic Malware," *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 3, no. 1, pp. 167–172, 2014.

[3] V. Teemu, L. Trinberg, and N. Pissanidis, "I accidentally malware - what should I do... is this dangerous? Overcoming inevitable risks of electronic communication," NATO CCD COE, Tallinn, Estonia, Tech. Rep., 2016.

[4] M. Kenney, "Cyber-Terrorism in a Post-Stuxnet World," *Orbis*, vol. 59, no. 1, pp. 111–128, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0030438714000787

[5] Symantec, "015 Internet Security Threat Report," p. 119, 2015.

[6] B. McKenna, "Symantec's Thompson pronounces old style IT security dead," *Network Security*, no. 2, pp. 1–3, 2016. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1353485805001947

[7] R. Unuchek, F. Sinitsyn, D. Parinov, and A. Liskin, "IT threat evolution Q3 2017. Statistics," 2017. [Online]. Available: https://securelist.com/it-threat-evolution-q3-2017-statistics/83131/

[8] VirusTotal, "File statistics during last 7 days (Nov 20 - Nov 27, 2017)," 2017. [Online]. Available: Filestatisticsduringlast7dayshttps://www.virustotal.com/en/statistics/

[9] T. Harmonen, "Identifying Polymorphic Malware," 2014.

[10] M. Vasilescu, L. Gheorghe, and N. Tapus, "Practical malware analysis based on sandboxing," *Proceedings - RoEduNet IEEE International Conference*, pp. 1–6, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6955304

[11] D. Maiorca, I. Corona, and G. Giacinto, "Looking at the bag is not enough to find the bomb," *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*, p. 119, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2484313.2484327

[12] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. P. Markatos, "Combining static and dynamic analysis for the detection of malicious documents," *Proceedings of the Fourth European Workshop on System Security - EUROSEC '11*, pp. 1–6, 2011. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1972551.1972555

[13] S. Kumar, C. Rama Krishna, N. Aggarwal, R. Sehgal, and S. Chamotra, "Malicious data classification using structural information and behavioral specifications in executables," *2014 Recent Advances in Engineering and Computational Sciences, RAECS 2014*, pp. 1–6, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6799525

[14] D. Liu, H. Wang, and A. Stavrou, "Detecting malicious javascript in PDF through document instrumentation," *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 100–111, 2014.

[15] M. Guri, G. Kedma, T. Sela, B. Carmeli, A. Rosner, and Y. Elovici, "Noninvasive detection of anti-forensic malware," *Proceedings of the 2013 8th International Conference on Malicious and Unwanted Software: "The Americas", MALWARE 2013*, pp. 1–10, 2013.

[16] A. Tamersoy, K. Roundy, and D. H. Chau, "Guilt by Association: Large Scale Malware Detection by Mining File-relation Graphs," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, pp. 1524–1533, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2623330.2623342

[17] R. Kaur and M. Singh, "Efficient hybrid technique for detecting zero-day polymorphic worms," *Souvenir of the 2014 IEEE International Advance Computing Conference, IACC 2014*, no. September 2011, pp. 95–100, 2014.

[18] M. Norouzi, A. Souri, and M. Samad Zamini, "A Data Mining Classification Approach for Behavioral Malware Detection," *Journal of Computer Networks and Communications*, vol. 2016, pp. 20–22, 2016.

[19] S. Paul and B. K. Mishra, "PolyS: Network-based Signature Generation for Zero-day Polymorphic Worms," *Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013*, vol. 6, no. 4, pp. 159–163, 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6514213

[20] L. S. Grini, "Feature Extraction and Static Analysis for Large-Scale Detection of Malware Types and Families," 2015.

[21] B. J. Kumar, H. Naveen, B. P. Kumar, S. S. Sharma, and J. Villegas, "Logistic regression for polymorphic malware detection using ANOVA F-Test," *Proceedings of 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017*, vol. 2018-Janua, pp. 1–5, 2018.

[22] V. Naidu, "Using Different Substitution Matrices in a String- Matching Technique for Identifying Viral Polymorphic Malware Variants," *Evolutionary Computation (CEC), 2016 IEEE Congress*, pp. 2903–2910, 2016.

[23] M. Chau, G. Alan Wang, and H. Chen, "A Syntactic Approach for Detecting Viral Polymorphic Malware Variants," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9650, no. April, 2016.

[24] A. Sharma and S. K. Sahay, "Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey," *International Journal of Computer Applications*, vol. 90, no. 2, pp. 7–11, 2014. [Online]. Available: http://research.ijcaonline.org/volume90/number2/pxc3894098.pdf

[25] S. K. Pandey and B. M. Mehtre, "A lifecycle based approach for malware analysis," *Proceedings - 2014 4th International Conference on Communication Systems and Network Technologies, CSNT 2014*, pp. 767–771, 2014.

[26] Y. Prayudi and S. Yusirwan, "the Recognize of Malware Characteristics Through Static and Dynamic Analysis Approach As an Effort To Prevent Cybercrime Activities," *Journal of Theoretical and Applied Information Technology (JATIT)*, vol. 77, no. xx, pp. 438–445, 2015.

[27] I. A. Saeed, J. B. Campus, M. A. Selamat, M. Ali, and M. A. Abuagoub, "A Survey on Malware and Malware Detection Systems," *International Journal of Computer Applications*, vol. 67, no. 16, pp. 975–8887, 2013.

[28] D. Uppal, V. Mehra, and V. Verma, "Basic survey on Malware Analysis, Tools and Techniques," *International Journal on Computational Science & Applications*, vol. 4, no. 1, pp. 103–112, 2014. [Online]. Available: http://www.airccse.org/journal/ijcsa/papers/4114ijcsa10.pdf

[29] S. Cesare, Y. Xiang, and W. Zhou, "Malwise-an effective and efficient classification system for packed and polymorphic malware," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1193–1206, 2013.

[30] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2089125.2089126

[31] U. Baldangombo, N. Jambaljav, and S.-J. Horng, "A Static Malware Detection System Using Data Mining Methods," *International Journal of Artificial Intelligence & Applications*, vol. 4, no. 4, p. 113, 2013. [Online]. Available: http://arxiv.org/abs/1308.2831

[32] A. K. Sahoo, K. S. Sahoo, and M. Tiwary, "Signature based Malware Detection for Unstructured Data in Hadoop," *Proceedings - 2014 Int. Conf. on Adv. in Electronics, Computers and Communications (ICAECC)*, pp. 1–6, 2014.

[33] S. Paul and B. K. Mishra, "Honeypot based signature generation for defense against polymorphic worm attacks in networks," *Proceedings of the 2013 3rd IEEE International Advance Computing Conference, IACC 2013*, pp. 159–163, 2013.

[34] M. Ahmadi, A. Sami, H. Rahimi, and B. Yadegari, "Malware detection by behavioural sequential patterns," *Computer Fraud and Security*, vol. 2013, no. 8, pp. 11–19, 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1361372313700721

[35] J. U. Joo, I. Shin, and M. Kim, "Efficient methods to trigger adversarial behaviors from malware during virtual execution in sandbox," *International Journal of Security and its Applications*, vol. 9, no. 1, pp. 369–376, 2015.

[36] A.-s. K. Pathan, "Automatic Defense against Zero-day Polymorphic Worms in Communication Networks," vol. 2, no. since 2006, pp. 9–10, 2013.

[37] R. Kaur and M. Singh, "A Survey on Zero-Day Polymorphic Worm Detection Techniques," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1520–1549, 2014.

[38] S. Paul and B. K. Mishra, "Survey of Polymorphic Worm Signatures," *International Journal of u- and e- Service, Science and Technology*, vol. 7, no. 3, pp. 129–150, 2014.

[39] J. B. Fraley and M. Figueroa, "Polymorphic malware detection using topological feature extraction with data mining," *SoutheastCon 2016*, pp. 1–7, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7506685/

[40] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," *IKT 2013 - 2013 5th Conference on Information and Knowledge Technology*, pp. 113–120, 2013.

[41] V. Naidu and A. Narayanan, "Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants," *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pp. 326–333, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7588867/

[42] X. Yuan, "PhD Forum: Deep Learning-Based Real-Time Malware Detection with Multi-Stage Analysis," *2017 IEEE International Conference on Smart Computing, SMARTCOMP 2017*, pp. 1–2, 2017.

[43] C.-T. LIN, "Feature Selection and Extraction for Malware Classification," *Journal of Information Science and Engineering*, vol. 31, pp. 965–992, 2015.

[44] Q. Jiang, "A Feature Selection Method for Malware Detection," *Proceeding of the IEEE International Conference on Information and Automation*, no. June, pp. 890–895, 2011.

[45] J. Brownlee, "Discover Feature Engineering, How to Engineer Features and How to Get Good at It - Machine Learning Mastery," 2014. [Online]. Available: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

[46] J. VanderPals, *Python Data Science Handbook | Python Data Science Handbook*. Sebastopol, CA: O'Reilly, 2016. [Online]. Available: https://jakevdp.github.io/PythonDataScienceHandbook/index.html

[47] S. Feffer, "It's all about the features," 2017. [Online]. Available: https://www.reality.ai/single-post/2017/09/01/It-is-all-about-the-features

[48] D. Arshi and M. Singh, "Behavior Analysis of Malware Using Machine Learning," in *Contemporary Computing (IC3), 2015 Eighth International Conference on*, 2015, pp. 481–486.

[49] P. M. Comar, L. Liu, S. Saha, P. N. Tan, and A. Nucci, "Combining supervised and unsupervised learning for zero-day malware detection," *Proceedings - IEEE INFOCOM*, pp. 2022–2030, 2013.

[50] A. Aleroud and G. Karabatis, "A contextual anomaly detection approach to discover zero-day attacks," *Proceedings of the 2012 ASE International Conference on Cyber Security, CyberSecurity 2012*, no. SocialInformatics, pp. 40–45, 2013.

[51] H. Dornhackl, K. Kadletz, R. Luh, and P. Tavolato, "Defining malicious behavior," *Proceedings - 9th International Conference on Availability, Reliability and Security, ARES 2014*, pp. 273–278, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6980292

[52] E. Masabo, K. S. Kaawaase, J. Sansa-otim, and J. Ngubiri, "A State of the Art Survey On Polymorphic Malware Analysis and Detection Techniques," *ICTACT Journal on Soft Computing*, vol. 8, no. 4, pp. 1762–1774, 2018.

[53] B. E. Skoudis and L. Zeltser, *Malware, Fighting Malicious Code*. Prentice Hall, 2003.

[54] B. Rad, M. Masrom, and S. Ibrahim, "Camouflage in Malware : from Encryption to Metamorphism," *International Journal of Computer Science and Network Security*, vol. 12, no. 8, pp. 74–83, 2012.

[55] J. Koret and E. Bachaalany, *The Antivirus Hacker's Handbook*. Wiley, Indianapolis, Indiana: John Wiley & Sons, 2015.

[56] G. Liang, J. Pang, and C. Dai, "A Behavior-Based Malware Variant Classification Technique," *International Journal of Information and Education Technology*, vol. 6, no. 4, pp. 291–295, 2016. [Online]. Available: http://www.ijiet.org/show-65-787-1.html

[57] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[58] N. S. Selamat, F. Hani, M. Ali, and M. Science, "Polymorphic Malware Detection," in *IT Convergence and Security (ICITCS), 2016 6th International Conference*, 2016.

[59] V. Mehra, V. Jain, and D. Uppal, "DaCoMM: Detection and classification of metamorphic malware," *Proceedings - 2015 5th International Conference on Communication Systems and Network Technologies, CSNT 2015*, pp. 668–673, 2015.

[60] Eskandari, M. S. Razieh, and A. Asadi, "Automatic Signature Generation for Polymorphic Worms by Combination ofToken Extraction and Sequence Alignment Approaches," in *Information and Knowledge Technology (IKT), 2015 7th Conference on. IEEE*, 2015.

[61] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proceedings - 2010 International Conference on Broadband, Wireless Computing Communication and Applications, BWCCA 2010*, 2010, pp. 297–300.

[62] C. Eagle, *The IDA pro book*. San Francisco: No Starch Press, Inc., 2011.

[63] X. Lu, J. Zhuge, R. Wang, Y. Cao, and Y. Chen, "De-obfuscation and detection of malicious PDF files with high accuracy," *Proceedings of the Annual Hawaii International Conference on System Sciences*, pp. 4890–4899, 2013.

[64] L. Wu, M. Xu, J. Xu, N. Zheng, and H. Zhang, "A novel malware variants detection method based on function-call graph," *2013 IEEE Conference Anthology, ANTHOLOGY 2013*, 2014.

[65] M. Alazab, S. Huda, J. Abawajy, R. Islam, J. Yearwood, S. Venkatraman, and R. Broadhurst, "A Hybrid Wrapper-Filter Approach for Malware Detection," *Journal of Networks*, vol. 9, no. 11, pp. 2878–2891, 2014. [Online]. Available: http://ojs.academypublisher.com/index.php/jnw/article/view/13565

[66] Y. Gao, Z. Lu, and Y. Luo, "Survey on malware anti-analysis," *5th International Conference on Intelligent Control and Information Processing, ICICIP 2014 - Proceedings*, pp. 270–275, 2015.

[67] A. Singh and B. Singh, *Identifying Malicious code through Reverse Engineering*, 2009, vol. 44.

[68] D. Cosovan, R. Benchea, and D. Gavrilut, "A practical guide for detecting the java script-based malware using hidden markov models and linear classifiers," *Proceedings - 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2014*, pp. 236–243, 2015.

[69] S. Singla, "A Novel Approach to Malware Detection using Static Classification," *(IJCSIS) International Journal of Computer Science and Information Security*, vol. 13, no. 3, pp. 1–5, 2015.

[70] S. Chaumette, O. Ly, and R. Tabary, "Automated extraction of polymorphic virus signatures using abstract interpretation," *Proceedings - 2011 5th International Conference on Network and System Security, NSS 2011*, pp. 41–48, 2011.

[71] Y. Qu and K. Hughes, "Detecting metamorphic malware by using behavior-based aggregated signature," *Internet Security (WorldCIS), 2013 World . . .*, pp. 13–18, 2013.

[72] S. Choudhary and M. D. Vidyarthi, "A Simple Method for Detection of Metamorphic Malware using Dynamic Analysis and Text Mining," *Procedia Computer Science*, vol. 54, pp. 265–270, 2015. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84944029156{&}partnerID=tZOtx3y1

[73] A. Verma, M. Rao, A. Gupta, W. Jeberson, and V. Singh, "a Literature Review on Malware and Its Analysis," *Int J Cur Res Rev*, vol. 05, no. 16, pp. 71–82, 2013.

[Online]. Available: http://www.ejmanager.com/mnstemps/45/45-1379402083.pdf?t=1395164049

[74] H. Qiu and F. C. C. Osorio, "Static malware detection with Segmented Sandboxing," *Proceedings of the 2013 8th International Conference on Malicious and Unwanted Software: "The Americas", MALWARE 2013*, pp. 132–141, 2013.

[75] S. Ranveer and S. Hiray, "Comparative Analysis of Feature Extraction Methods of Malware Detection," *International Journal of Computer Applications*, vol. 120, no. 5, pp. 1–7, 2015.

[76] L. Wang, Z. Li, Y. Chen, Z. J. Fu, and X. Li, "Thwarting zero-day polymorphic worms with network-level length-based signature generation," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, pp. 53–66, 2010.

[77] R. Tian, R. Islam, L. Batten, and S. Versteeg, "Differentiating malware from cleanware using behavioural analysis," *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010*, pp. 23–30, 2010.

[78] U. Bayer, E. Kirda, and C. Kruegel, "Improving the efficiency of dynamic malware analysis," *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, p. 1871, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1774088.1774484

[79] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of API calls," *Proceedings - 2nd Cybercrime and Trustworthy Computing Workshop, CTC 2010*, no. November 2009, pp. 52–59, 2010.

[80] M. A. I. Almarshad, M. M. Mohammed, and A. S. K. Pathan, "Detecting zero-day polymorphic worms with jaccard similarity algorithm," *International Journal of Communication Networks and Information Security*, vol. 8, no. 3, pp. 203–214, 2016.

[81] M. Sikorski and A. Honig, *Practical Malware analysis: The hands-on guide to dissecting malicious software*, 1st ed. San Francisco: No Starch Press San Francisco, CA, USA, dec 2012.

[82] A. Javed and M. Akhlaq, "On the approach of static feature extraction in trojans to combat against zero-day threats," *2014 International Conference on IT Convergence and Security, ICITCS 2014*, pp. 1,5, 2014. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp={&}arnumber=7021794{&}isnumber=7021698

[83] M. A. M. Ali and M. A. Maarof, "Dynamic innate immune system model for malware detection," *2013 International Conference on IT Convergence and Security, ICITCS 2013*, pp. 1–4, 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6717828

[84] S. Gadhiya and K. Bhavsar, "Techniques for Malware Analysis," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 4, pp. 2277–128, 2013.

[85] R. Islam, R. Tian, L. Batten, and S. Versteeg, "Classification of malware based on string and function feature selection," *Proceedings - 2nd Cybercrime and Trustworthy Computing Workshop, CTC 2010*, pp. 9–17, 2010.

[86] J. Landage and M. Wankhade, "Malware and Malware Detection Techniques: A Survey," *International Journal of Engineering Research ...*, vol. 2, no. 12, pp. 61–68, 2013. [Online]. Available: http://www.ijert.org/browse/volume-2-2013/december-2013-edition?download=6744:malware-and-malware-detection-techniques--a-survey{&}start=10

[87] H. Dornhackl, K. Kadletz, R. Luh, and P. Tavolato, "Malicious Behavior Patterns," *2014 IEEE 8th International Symposium*, pp. 384–389, 2014.

[88] S. K. Pandey and B. M. Mehtre, "Performance of malware detection tools: A comparison," in *Proceedings of 2014 IEEE International Conference on Advanced Communication, Control and Computing Technologies, ICACCCT 2014*, no. 978, 2015, pp. 1811–1817.

[89] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2089125.2089126

[90] S. Yusirwan, Y. Prayudi, and I. Riadi, "Implementation of Malware Analysis using Static and Dynamic Analysis Method," *International Journal Of Computer Applications*, vol. 117, no. 6, pp. 11–15, 2015.

[91] S. Hong and S. Lee, "New Malware Analysis Method on Digital Forensics," *Indian Journal of Science and Technology*, vol. 8, no. August, 2015.

[92] L. Apvrille and S. Antipolis, "Identifying Unknown Android Malware with Feature Extractions and Classification Techniques," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1, 2015.

[93] S. Neuner, V. V. D. Veen, and M. Lindorfer, "Enter Sandbox: Android Sandbox Comparison," *3rd IEEE Mobile Security Technologies Workshop*, no. October, 2014.

[94] A. Kumar, K. S. Kuppusamy, and G. Aghila, "A learning model to detect maliciousness of portable executable using integrated feature set," *Journal of King Saud University - Computer and Information Sciences*, vol. 31, no. 2, pp. 252–265, 2019. [Online]. Available: http://dx.doi.org/10.1016/j.jksuci.2017.01.003

[95] Z. A. Bhuiyan, M. Zaman, G. Wang, T. Wang, and J. Wu, "Privacy-Protected Data Collection in Wireless Medical Sensor Networks," in *Networking, Architecture, and Storage (NAS), 2017 International Conference*, 2017, pp. 1–2.

[96] M. Z. A. Bhuiyan, T. Wang, and G. Wang, "Event Detection through Differential Pattern Mining in Cyber-Physical Systems," *IEEE Transactions on Big Data*, 2017.

[97] J. Drew, T. Moore, and M. Hahsler, "Polymorphic Malware Detection Using Sequence Classification Methods," *2016 IEEE Security and Privacy Workshops (SPW)*, pp. 81–87, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7527757/

[98] Microsoft, "Microsoft Malware Classification Challenge (BIG 2015)," 2015. [Online]. Available: https://www.kaggle.com/c/malware-classification/

[99] J. Drew, M. Hahsler, and T. Moore, "Polymorphic malware detection using sequence classification methods and ensembles," *EURASIP Journal on*

Information Security, vol. 2017, no. 1, p. 2, 2017. [Online]. Available: http://jis.eurasipjournals.springeropen.com/articles/10.1186/s13635-017-0055-6

[100] V. Naidu and A. Narayanan, "Needleman-Wunsch and Smith-Waterman Algorithms for Identifying Viral Polymorphic Malware Variants," *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, no. August, pp. 326–333, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7588867/

[101] P. Sharma, S. Kaur, and J. Arora, "An Advanced Approach to Polymorphic / Metamorphic Malware Detection using Hybrid Clustering Approach," *International Research Journal of Engineering and Technology (IRJET)*, vol. 3, no. 6, pp. 2229–2232, 2016.

[102] X. Meng, Z. Shan, F. Liu, B. Zhao, J. Han, H. Wang, and J. Wang, "MCSMGS: Malware Classification Model Based on Deep Learning," *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pp. 272–275, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8250369/

[103] VX-Collection, "VX Heaven windows virus collection," 2016. [Online]. Available: https://archive.org/details/vxheaven-windows-virus-collection

[104] K. Bojan, Z. Apostolis, W. George, and E. Claudia, "Deep Learning for Classification of Malware System Call Sequences," *AI 2016: Advances in Artificial Intelligence*, 2016.

[105] B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert, "Adaptive Semantics-Aware Malware Classification," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* Springer, 2016, pp. 419—-439.

[106] Krmaxwell, "A tool to retrieve malware directly from the source for security researchers," 2015. [Online]. Available: https://github.com/krmaxwell/maltrieve

[107] M. Rhode, P. Burnap, and K. Jones, "Early-Stage Malware Prediction Using Recurrent Neural Networks," *Computers & Security*, no. December, pp. 1–28, 2018. [Online]. Available: https://doi.org/10.1016/j.cose.2018.05.010

[108] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, "Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning," *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7841778/

[109] W. Huang and J. W. Stokes, "MtNet: A multi-task neural network for dynamic malware classification," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9721, pp. 399–418, 2016.

[110] Y. H. Choi, B. J. Han, B. C. Bae, H. G. Oh, and K. W. Sohn, "Toward Extracting Malware Features for Classification using Static and Dynamic Analysis," *International Conference on Computing and Networking Technology*, pp. 126–129, 2012.

[111] 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," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, 2017.

[112] J. Dudovskiy, *The ultimate guide to writing a dissertation in business studies: a step-by-step assistance*. Pittsburgh, USA, 2016.

[113] N. Bajpai, *Business research methods*. Pearson Education, India, 2011.

[114] Saunders and M. NK, *Research methods for business students*, 5th ed. Pearson Education India,, 2011.

[115] P. Andrew, P. Pedersen, and C. McEvoy, *Research Methods and Designs in Sport Management*. Human Kinetics, 2011.

[116] J. Collis and R. Hussey, *Business Research: A Practical Guide for Undergraduate and Postgraduate Students*, 4, Ed. Palgrave Macmillan, 2014.

[117] J. Wilson, *Essentials of Business Research: A Guide to Doing Your Research Projec*. SAGE Publications, 2010.

[118] Easterby-Smith, R. M, Thorpe, and P. Jackson, *Management Research*, 3rd ed. SAGE Publications Ltd., London, 2008.

[119] H. Collins, *Creative Research: The Theory and Practice of Research for the Creative Industries*. AVA Publications, 2010.

[120] A. Novikov and D. Novikov, *Research Methodology: From Philosophy of Science to Research Design*. CRC Press, 2013.

[121] Saunders M, L. P, and T. A, *Research Methods for Business Students*, 6th ed. Pearson Education Limited, 2012.

[122] Bryman A. and E. Bell, *Business Research Methods*, 4th ed. Oxford University Press, 2015.

[123] G. D. Israel, "Determining Sample Size," *University of Florida Cooperative Extension Service*, no. November, pp. 1–5, 1992.

[124] Y. Nativ, "theZoo - A Live Malware Repository," 2017. [Online]. Available: https://github.com/ytisf/theZoo

[125] "VirusShare.com - Because Sharing is Caring." [Online]. Available: https://virusshare.com/

[126] "Viruses don't harm, ignorance does!" [Online]. Available: http://www.vxheaven.org

[127] "Malware tips." [Online]. Available: http://www.malwaretips.com

[128] "Malware repository." [Online]. Available: http://www.malware.lu

[129] L. Zeltser, "Malware sample sources for researchers." [Online]. Available: https://zeltser.com/malware-sample-sources

[130] "The computer security group at UC Santa Barbara." [Online]. Available: http://seclab.cs.ucsb.edu/

[131] M. Ramilli, "Malware Training Sets: a machine learning dataset for everyone," 2016. [Online]. Available: http://marcoramilli.blogspot.it/2016/12/malware-training-sets-machine-learning.html

[132] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A Malware Instruction Set for Behavior-Based Analysis," *Sicherheit Schutz und Zuverlässigkeit SICHERHEIT*, no. TR-2009-07, pp. 1–11, 2011. [Online]. Available: http://eprints.pascal-network.org/archive/00007694/

[133] P. Burnap and M. Rhode, "Behavioural machine activity for benign and malicious Win7 64-bit executables," 2018. [Online]. Available: http://doi.org/10.17035/d.2018.0050524986

[134] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost : Improving Prediction," in *European conference on principles of data mining and knowledge discovery*, 2003, pp. 107–119.

[135] A. Fern and S. Garc, "SMOTE for Learning from Imbalanced Data : Progress and Challenges , Marking the 15-year Anniversary," *Journal of Artificial Intelligence Research*, vol. 61, pp. 863–905, 2018.

[136] Amazon, "Amazon Machine Learning Developer Guide," Tech. Rep., 2015. [Online]. Available: https://docs.aws.amazon.com/machine-learning/latest/dg/machinelearning-dg.pdf

[137] K. Willems, "Python Exploratory Data Analysis Tutorial." [Online]. Available: https://www.datacamp.com/community/tutorials/exploratory-data-analysis-python

[138] Z. Markel and M. Bilzor, "Building a machine learning classifier for malware detection," in *2014 Second Workshop on Anti-malware Testing Research (WATeR)*, 2014, pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7015757/

[139] J. Brownlee, *Master Machine Learning Algorithms*, 1st ed., 2016. [Online]. Available: https://github.com/tim-hub/machine-learning-books/blob/master/MasterMachineLearningAlgorithms2016.pdf

[140] E. Masabo, K. S. Kaawaase, J. Sansa-Otim, and D. Hanyurwimfura, "Integrated Feature Extraction Approach Towards Detection of Polymorphic Malware In Executable Files," *International Journal of Computer Science and Security*, vol. 11, no. 112, pp. 2017–25, 2017. [Online]. Available: http://www.cscjournals.org/manuscript/Journals/IJCSS/Volume11/Issue2/IJCSS-1319.pdf

[141] M. Emmanuel, K. K. Swaib, S. O. Julianne, and H. Damien, "Structural Feature Engineering Approach for Detecting Polymorphic Malware," *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on*

Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), pp. 716–721, 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8328469/

[142] C. O. Truică, A. Boicea, and I. Trifan, "CRUD Operations in MongoDB," *International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013)*, no. Icacsei, pp. 347–350, 2013.

[143] I. Guyon and A. Elisseeff, "Feature Extraction, Foundations and Applications: An introduction to feature extraction," *Studies in Fuzziness and Soft Computing*, vol. 207, pp. 1–25, 2006. [Online]. Available: http://eprints.pascal-network.org/archive/00002475/

[144] D. J. Rumsey, *Statistics I & II For Dummies 2*, ser. –For dummies. Wiley, 2013. [Online]. Available: https://books.google.co.tz/books?id=2SniJODnS-UC

[145] B. J. Long, D. Mccurley, B. D. Snell, B. A. Larson, B. D. Snell, and B. E. S. Burns, "Predictive Analytics and Futurism Analytics," *Predictive Analytics and Futurism*, no. 17, 2018.

[146] Z. E. Xu, K. Q. Weinberger, and A. X. Zheng, "Gradient Boosted Feature Selection," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2014, pp. 522–531.

[147] E. Masabo and J. Sansa-otim, "Big Data : Deep Learning for detecting Malware," *2018 ACM/IEEE Symposium on Software Engineering in Africa*, vol. i, pp. 20–26, 2018.

[148] G. Sun and Q. Qian, "Deep Learning and Visualization for Identifying Malware Families," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. c, p. 1, 2018.

[149] Algorithmia, "Introduction to Optimizers," 2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-optimizers/

[150] D. A. Teich, "Machine Learning and Software Lifecycle Tools: Each Must Help The Other," 2018. [Online]. Available: https://www.forbes.com/sites/davidteich/2018/06/28/machine-learning-and-software-lifecycle-tools-each-must-help-the-other/{#}68a15bac1189

[151] K. Spirina, "How AI Changes SDLC and Improves Customer Experience," 2018. [Online]. Available: https://www.smallbizdaily.com/how-ai-changes-software-development/

[152] D. Singh and A. Sharma, "Software requirement prioritization using machine learning," *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2014-Janua, no. January, pp. 701–704, 2014.

[153] N. Chen, S. C. Hoi, and X. Xiao, "Software process evaluation: A machine learning approach," *2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings*, no. ii, pp. 333–342, 2011.

[154] Algorithmia, "Deploying Machine Learning Models at Scale," 2018. [Online]. Available: https://algorithmia.com/blog/deploying-machine-learning-at-scale

[155] Kunalgupta007, "Cuckoo Installation Guide: Malware Sandboxing," 2018. [Online]. Available: https://www.cybrary.it/0p3n/cuckoo-installation-guide-malware-sandboxing/

Appendix A

Description of extracted features

Feature	Description
api_resolv	Resolved API. Uses Windows APIs
cmd_exec	Command execution
file_access	Accessing the files
file_delete	Deleting files
file_drop	Dropping files
file_read	Reading files
file_write	Writing or creating a file
mutex_access	Create or access a mutex. The purpose of the Mutex is to prevent unwanted simultaneous access by multiple threads on a single
net_con	Network activity
net_dns	Dns requests
net_http	http requests
pe_imports	Importing functions
pe_sec_character	Section characters
pe_sec_entropy	Section entropy assessment
pe_sec_name	Section name
reg_access	Registry accessing
reg_delete	Deleting a registry key
reg_read	Reading a registry key
reg_write	Writing a registry key
service_create	Creating a service
service_start	Starting a service

Table 1 – continued from previous page

Feature	Description
FraudGuardThreatIntelAPI	FraudGuard is a service designed to provide an easy way to validate usage by continuously collecting and analyzing real-time internet traffic. This feature helps check malware internet traffic attemps for fraud activity.
ShadowServer_White_list	This server provides a lookup mechanism to test an executable file against a list of known software applications
ThreatCrowdResolutions	ThreatCrowd to quickly find and pivot on threats. Is the malware found by threatcrowd?
antianalysis_detectfile	Does it create anti analysis detection file?
antiav_detectfile	Attempts to identify installed AV products by installation directory
antiav_detectreg	Attempts to identify installed AV products by registry key
antiav_servicestop	Attempts to stop active services
antidbg_devices	Checks for the presence of known devices from debuggers and forensic tools
antidbg_windows	Checks for the presence of known windows from debuggers and forensic tools
antiemu_wine_func	Checks for the presence of wine emulator
antiemu_wine_reg	Detects the presence of Wine emulator
antimalware_metascan	Checks for the presence of a metascan of malware
antisandbox_cuckoocrash	Checks presence of a sandbox
antisandbox_productid	Checks presence of a sandbox
antisandbox_restart	Attempts to shutdown or restart the system, generally used for bypassing sandboxing
antisandbox_sboxie_libs	Checks presence of a sandbox

Table 1 – continued from previous page

Feature	Description
antisandbox_sleep	Sleeps when there is a sandbox. A process attempted to delay the analysis task
antisandbox_sunbelt_libs	Checks presence of a sandbox
antisandbox_suspend	Attempts to suspend its activity when there is a sandbox.
antisandbox_unhook	Tries to unhook Windows functions monitored by Cuckoo
antivirus_virustotal	File has been identified by at least one AntiVirus engine on VirusTotal as malicious
antivm_generic_bios	Checks the version of Bios, possibly for anti-virtualization
antivm_generic_disk	Queries information on disks, possibly for anti-virtualization
antivm_generic_disk_setupapi	Queries information on disks, possibly for anti-virtualization
antivm_generic_diskreg	Queries information on disks, possibly for anti-virtualization
antivm_generic_system	Queries system information, possibly for anti-virtualization
antivm_vbox_files	Detects VirtualBox through the presence of a file
antivm_vbox_keys	Detects VirtualBox through the presence of a registry key
antivm_vbox_libs	Detects VirtualBox through the presence of library functions
antivm_vmware_devices	Detects Vmware through the presence of a device
antivm_vmware_files	Detects Vmware through the presence of a file
antivm_vmware_keys	Detects Vmware through the presence of a registry key
antivm_vpc_files	Detects Virtual PC through the presence of a file

Table 1 – continued from previous page

Feature	Description
antivm_vpc_keys	Detects Virtual PC through the presence of a registry key
antivm_xen_keys	Detects Xen through the presence of a registry key
api_spamming	Making a spam request using API, disguising as genuine
banker_zeus_mutex	Creates Zeus (Banking Trojan) mutexes
banker_zeus_url	ontacts C&C server HTTP check-in (Banking Trojan)
bcdedit_command	Runs bcdedit (Boot Configuration Data) commands specific to ransomware
bootkit	Likely installs a bootkit via raw harddisk modifications. A bootkit is a variant of a rootkit, a type of malware with the ability to conceal itself from your operating system and antivirus software
browser_security	Attempts to modify browser security settings, mainly through the registry
bypass_firewall	Operates on local firewall's policies and settings
clamav	Detects Clam Antivirus
copies_self	Copies itself
creates_largekey	Creates or sets a registry key to a long series of bytes, possibly to store a binary or malware config
creates_nullvalue	Creates null values for misleading & hiding. Probably created in the registires.
critical_process	Attempts to dissuade users by prompting messages signalling a fake critical problem

Table 1 – continued from previous page

Feature	Description
cryptam	Detects the presence of Cryptam. Cryptam analyzes suspicious office documents to detect embedded executables or ... Find encrypted embedded executables common to APT malware attacks
deepfreeze_mutex	Checks for a known DeepFreeze Frozen State Mutex. This is done to find an alternative way of infecting the system. Deep Freeze works as a kernel-level driver and protects your system configuration by preserving the desired baseline settings as a restoration point.
deletes_self	Malware deletes itself
deletes_shadow_copies	shadow copies, which allows access to point-in-time copies of the currently locked files as created by Windows Restore. Shadow Copies can be created on local and external (removable or network) volumes by any Windows component that uses this technology, such as when creating a scheduled Windows Backup or automatic System Restore point. -à the malware can therefore delete shodow copies of itself.
disables_browser_warn	Attempts to disable browser security warnings
disables_system_restore	Attempts to disable System Restore
disables_uac	Attempts to modify or disable UAC prompt behavior
disables_windows_defender	Attempts to disable windows defender
disables_windowsupdate	Attempts to disable windows update
downloader_cabby	suspicious downloader (Cabby)

Table 1 – continued from previous page

Feature	Description
dridex_behavior	Exhibits behavior characteristic of Dridex malware. Dridex is a strain of banking malware that leverages macros in Microsoft Office to infect systems
driver_load	Loads a driver
dropper	Drops a binary and executes it
encrypted_ioc	At least one IP Address, Domain, or File Name was found in a crypto call. From https://github.com/ctxis/CAPE/tree/master/modules/si
exec_crash	At least one process apparently crashed during execution
fraudguard_threat_intel_api	FraudGuard is a service designed to provide an easy way to validate usage by continuously collecting and analyzing real-time internet traffic.
infostealer_bitcoin	Attempts to access Bitcoin/ALTCoin wallets
infostealer_browser	Steals private information from local Internet browsers
infostealer_ftp	Harvests credentials from local FTP client softwares
infostealer_keylog	Creates a windows hook that monitors keyboard input (keylogger)
infostealer_mail	Harvests credentials from local email clients
injection_createremotethread	Code injection with CreateRemoteThread in a remote process
injection_runpe	Executed a process and injected code into it, probably while unpacking
injection_rwx	Creates RWX memory
internet_dropper	Behavior consistent with a dropper attempting to download the next stage
ipc_namedpipe	A named pipe was used for inter-process communication

Table 1 – continued from previous page

Feature	Description
mimics_agent	Mimics the system's user agent string for its own requests
mimics_filetime	Mimics the file times of a Windows system file
mimics_icon	Mimics icon used for popular non-executable file format
modifies_certs	Attempts to create or modify system certificates
modifies_desktop_wallpaper	Attempts to modify desktop wallpaper
modifies_hostfile	The sample wrote data to the system hosts file
modify_proxy	Sets or modifies WPAD proxy auto configuration file for traffic interception
modify_security_center_warnings	Attempts to modify or disable Security Center warnings
modify_uac_prompt	Attempts to disable UAC
multiple_useragents	Network activity contains more than one unique useragent
network_bind	Starts servers listening on {0}
network_cnc_http	HTTP traffic contains suspicious features, which may be indicative of malware related traffic
network_http	Performs some HTTP requests
network_torgateway	Connects to Tor Hidden Services through a Tor gateway
office_security	Attempts to modify Microsoft Office security settings
origin_langid	Unconventionial binary language
origin_resource_langid	Unconventionial language used in binary resources
packer_entropy	The binary likely contains encrypted or compressed data
packer_upx	The executable is compressed using UPX

Table 1 – continued from previous page

Feature	Description
packer_vmprotect	The executable is likely packed with VMProtect
persistence_ads	Attempts to interact with an Alternate Data Stream (ADS). An alternate data stream (ADS) is a feature of Windows New Technology File System (NTFS) that contains metadata for locating a specific file by author or title
persistence_autorun	Installs itself for autorun at Windows startup
persistence_service	Created a service that was not started
polymorphic	Creates a slightly modified copy of itself. target–>ssdeep, sha1, size
pony_behavior	Exhibits behavior characteristic of Pony malware. -àPony may attempt to steal stored credentials, usernames and passwords, and other personal and confidential information. This information may be transmitted to a destination specified by the author. Spyware.Pony may allow an attacker to install additional software to the infected machine, or may direct the infected machine to participate in a malicious botnet for the purposes of sending spam or other malicious activities. Pony is malware that steals personal data. Research shows that cyber criminals spread this malware in various ways: spam emails, fake Adobe Flash Player updates, scams such as The HoeflerText Font Wasn't Found, etc. The presence of adware-type applications that generate pop-up ads can also lead to Pony infections.

Table 1 – continued from previous page

Feature	Description
process_interest	Expresses interest in specific running processes
process_needed	repeatedly searches for a not-found process, may want to run with startbrowser=1 option
ransomware_extensions	Appends known ransomware file extensions to files that have been encrypted
ransomware_file_modifications	Exhibits possible ransomware file modification behavior
ransomware_files	Creates a known ransomware decryption instruction / key file
ransomware_from_StringFLOSS	
ransomware_message	Writes a potential ransom message to disk
ransomware_radamant	Exhibits behavior characteristic of Radamant ransomware. à The Radamant Ransomware is designed to encrypt files and change their extension to RDM.
ransomware_recyclebin	Empties the Recycle Bin, indicative of ransomware
rat_spynet	Creates known SpyNet mutexes and/or registry changes
reads_self	Reads data out of its own binary image
recon_beacon	A process sent information about the computer to a remote location
recon_checkip	Looks up the external IP address
recon_fingerprint	Collects information to fingerprint the system. à indicators = DigitalProductId,MachineGuid,SystemBIOSDate
recon_programs	Collects information about installed applications
removes_zoneid_ads	Attempts to remove evidence of file being downloaded from the Internet
sniffer_winpcap	Installs WinPCAP
static_detection	Attempts to to avoid static detection

Table 1 – continued from previous page

Feature	Description
static_pe_anomaly	Anomalous binary characteristics
static_versioninfo_anomaly	unusual version info supplied for binary
stealth_file	Creates a hidden or system file
stealth_hidden_extension	Attempts to modify Explorer settings to prevent file extensions from being displayed
stealth_hiddenreg	Attempts to modify Explorer settings to prevent hidden files from being displayed
stealth_hide_notifications	Attempts to modify user notification settings
stealth_network	Network activity detected but not expressed in API logs. à Malware tries to hide its network activity
stealth_timeout	possible date expiration check, exits too soon after checking local time
stealth_webhistory	"Clears web history
stealth_window	a process created a hidden window
virus	Likely virus infection of existing system binary
yara_detection	Yara rule detections observed from a process memory dump. Did it match any yara rules?
str	Creates strings

Appendix B

Cuckoo Installation

Cuckoo is an automated malware analysis system. Its simplified setup is provided below according to cybrary [155] .

1) Install Ubuntu
2) sudo apt-get update
3) Reboot Ubuntu
4) sudo apt-get install git -y
5) sudo apt-get install python python-pip python-dev libffi-dev libssl-dev -y
6) sudo apt-get install python-virtualenv python-setuptools -y
7) sudo apt-get install libjpeg-dev zlib1g-dev swig -y
8) sudo apt-get install mongodb -y
9) sudo apt-get install postgresql libpq-dev -y
10) echo deb http://download.virtualbox.org/virtualbox/debian xenial contrib | sudo tee -a /etc/apt/sources.list.d/virtualbox.list
11) wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -O- | sudo apt-key add
12) sudo apt-get update
13) sudo apt-get install virtualbox-5.1 -y
14) sudo apt-get install tcpdump apparmor-utils -y
15) sudo aa-disable /usr/sbin/tcpdump
16) sudo git clone https://github.com/volatilityfoundation/volatility
17) cd volatility
18) sudo python ./setup.py install
19) cd ..
20) sudo apt-get install swig
21) sudo pip install m2crypto==0.24.0

22) sudo pip install -U pip setuptools
23) sudo pip install -U cuckoo
24) sudo pip install distorm3
25) sudo mkdir /opt/cuckoo
26) sudo chown cuckoo:cuckoo /opt/cuckoo
27) cuckoo –cwd /opt/cuckoo
28) sudo virtualbox
29) Install a virtual machine within VirtualBox. Name it 'cuckoo1'.
30) vboxmanage hostonlyif create
31) vboxmanage hostonlyif ipconfig vboxnet0 –ip 192.168.56.1
32) Use ipconfig to ensure the network adapter shows up.
33) Edit the network settings of the virtual machine as follows:
34) Static IP – 192.168.56.101 , Default Gateway – 192.168.56.1 , DNS – any DNS server (I used 8.8.8.8)
35) sudo iptables -A FORWARD -o ens32 -i vboxnet0 -s 192.168.56.0/24 -m conntrack –ctstate NEW -j ACCEPT
36) sudo iptables -A FORWARD -m conntrack –ctstate ESTABLISHED,RELATED -j ACCEPT
37) sudo iptables -A POSTROUTING -t nat -j MASQUERADE
38) sudo su
39) echo 1 >/proc/sys/net/ipv4/ip_forward
40) To verify that the virtual machine has an internet connection, open cmd and ping 8.8.8.8 and see if it replies.
41) Download and install both Python 2.7 and Pillow 5.2.0 on the Virtual machine
42) Install Guest Additions on the Windows machine
43) Copy cuckoo agent (found in /opt/cuckoo/agent on Ubuntu machine) to your OS
44) Disable firewall and UACon the virtual machine
45) Start the agent.py and close everything else on the virtual machine.
46) Take a snapshot of the virtual machine
47) cd /opt/cuckoo/conf
48) sudo nano cuckoo.conf
49) Change settings in here for the cuckoo configuration (if needed).
50) sudo nano virtualbox.conf

51) Settings need to be changed in the virtualbox.conf file. ‘interface = vboxnet0’, ‘machines = cuckoo1’, ‘label = cuckoo1’ and ‘ip = 192.168.56.101’.

52) sudo nano memory.conf

53) Change the guest_profile to the version of Windows you are using.

54) sudo nano reporting.conf

55) First, find [mongodb]. Then make sure ‘enabled = yes’.

56) cd /opt/cuckoo

57) cuckoo -d

58) Open a second terminal:

59) cuckoo web

60) Check for ‘http://localhost:8000/’. Open this link in Firefox or any browser and upload any file for analysis.

Appendix C

Polymorphic Malware snapshots

Investigation 1

Name	**b029deb6da0a1a62_pgexhu.wm**
Filepath	C:\Users\pc1\AppData\Local\Temp\pgexhu.wm
Size	18.4KB
Processes	1180 (**Potao_1stVersion_AC854A3C91D52BFC09605506E76975AE.exe**)
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	c95916e402b83e84dddbdf9814c834ae
SHA1	211332a57180306de8bd8e43513f9c51b7a453b0
SHA256	b029deb6da0a1a628d1f2a4345c5d703e93dcd24fe0f4b8482860a27cb2b3027

Figure C.1: Investigation 1 - Random file drop

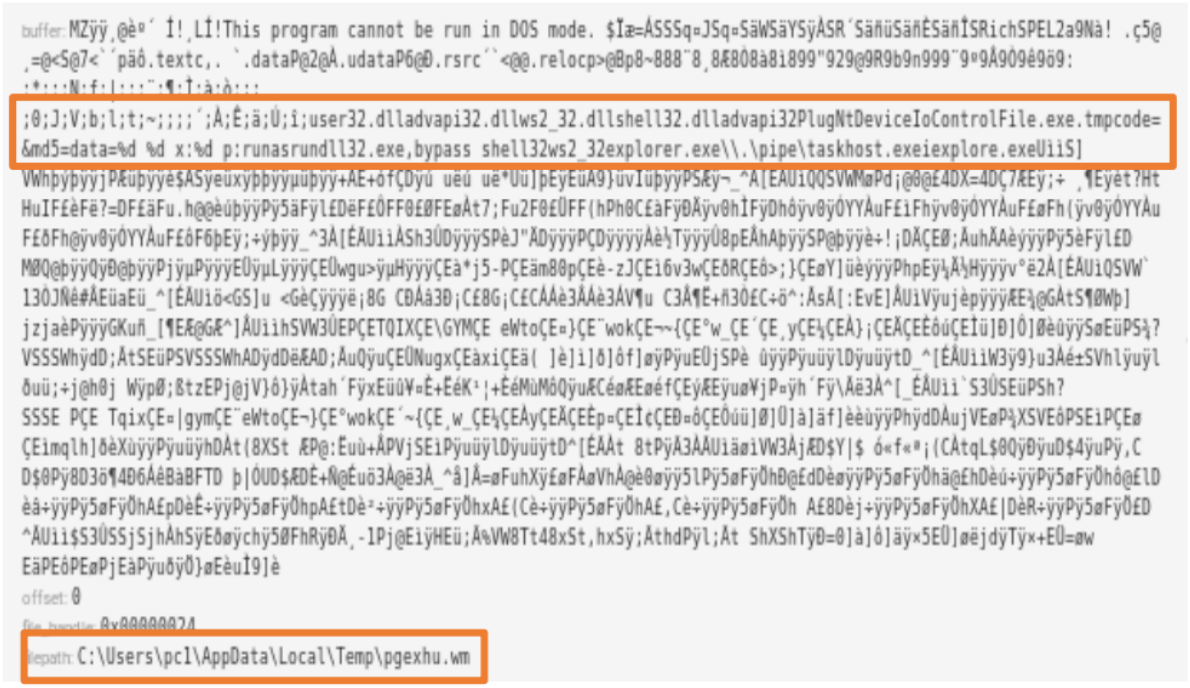

Figure C.2: Investigation 1 - Content of the dropped file

Investigation 2

Name	**b7464cf8dae62ee0_dbsdh.n**
Filepath	C:\Users\pc1\AppData\Local\Temp\dbsdh.n
Size	17.2KB
Processes	1180 (**Potao_1stVersion_AC854A3C91D52BFC09605506E76975AE.exe**)
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	13bb044121eb6c4281c273adbaf19abb
SHA1	f4633aa3f770a33adeefeff0ac0b8f00c35246dd
SHA256	b7464cf8dae62ee0b62430a2213383fb49393aff52736b198433beb7e4921e15
CRC32	95BEA931

Figure C.3: Investigation 2 - Random file drop

Figure C.4: Investigation 2 - Content of the dropped file

Investigation 3

Filepath	C:\Users\pc1\AppData\Local\Temp\xpnfke.og
Size	17.0KB
Processes	8432 (**Potao_1stVersion_AC854A3C91D52BFC09605506E76975AE.exe**)
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	b3f64626cdefb76c6d06d26e54870667
SHA1	1aa0217bcc05a49b229f7cb541288be7829cba95
SHA256	c8d62d99f3e9efd6839d1fe1c366d7e6e49514c610a6d435f1fcc7d3a0ea5586
CRC32	5E738332

Figure C.5: Investigation 3 - Random file drop

Figure C.6: Investigation 3 - Content of the dropped file

Investigation 4

Filepath	C:\Users\pc1\AppData\Local\Temp\yhxlsc.v
Size	18.4KB
Processes	1480 (**Potao_1stVersion_AC854A3C91D52BFC09605506E76975AE.exe**)
Type	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5	40e0e24c83d80b446b964899c4f193fc
SHA1	16a053ca47452ad76521b82cfa9b122ad57d5bfb
SHA256	9b59ba19e2641323d8feb6adfff69d86e1c860ea11fb7a0dc7230ce772b52ac4
CRC32	BE5A25CA

Figure C.7: Investigation 4 - Random file drop

buffer: MZÿÿ¸@è º´ Í!¸LÍ!This program cannot be run in DOS mode. $Ïæ=ÁSSSq¤JSq¤
SäWSäYSÿÀSR´SäñüSäñÈSäñÏSRichSPEL2a9Nà! .ç5@¸=@<S@7<`´pãõ.textc,. `.dataP@2@À.udataP6@Ð.rsrc´`
<@@.relocp>@Bp8~888¯8¸8Æ8Ô8à8i899"929@9R9b9n999¯9º9Å9Ò9ë9õ9::*:::N:f:|:::¯:¶:Ì:à:ò:;;
;0;J;V;b;l;t;~;;;;´;À;Ê;ã;Ú;î;user32.dlladvapi32.dllws2_32.dllshell32.dlladvapi32PlugNtDeviceIoCon
trolFile.exe.tmpcode=&md5=data=%d %d x:%d p:runasrundll32.exe,bypass shell32ws2_32explorer.exe\\.
\pipe\taskhost.exeiexplore.exeUìiS]VWhþýþÿÿjPÆüþÿÿè$ÃSÿeüxÿþþÿÿµüþÿÿ+ÃÊ+ófÇDÿú uëú uë*Uü]þÊ
ÿEüA9}üvÎüþÿÿPSÆÿ¬_^Ã[ÉÃUìQQSVWMøPd¡@0@£4DX=4DÇ7ÆEÿ;÷ ¸¶Eÿèt?Ht HuIF£èFë?=DF£ãFu.h@@èúþÿÿPÿ5ãFÿl£D
ëF£ÔFF0£ØFEøÀt7;Fu2F0£ÜFF(hPh0C£àFÿÐÃÿv0hÌFÿDhôÿv0ÿÔYYÀuF£iFhÿv0ÿÔYYÀuF£øFh(ÿv0ÿÔYYÀuF£ôFh@
ÿv0ÿÔYYÀuF£ôF6þEÿ;÷ýþÿÿ_^3Ã[ÉÃUìiÀSh3ÛDÿÿÿSPèJ"ÃDÿÿÿPÇDÿÿÿÿÀè½TÿÿÿÛ8pEÅhAþÿÿSP@þÿÿè÷!¡DÃÇEØ;ÃuhÃA
èýÿÿPÿ5èFÿl£DMØQ@þÿÿQÿÐ@þÿÿPjÿµPÿÿÿEÜÿµLÿÿÿÇEÜwgu>ÿµHÿÿÿÇEà*j5-PÇEãm80pÇEè-
zJÇEi6v3wÇEôRÇEõ>;}ÇEøY]üèýÿÿPhpEÿ¼Ã½Hÿÿÿvºë2À[ÉÃUìQSVW`13ÒJÑè#ÅEüaEü_^[ÉÃUìõ<GS]u <GèÇÿÿÿë¡8G C
ÐÁå3Ð¡C£8G¡C£CÁÀè3ÅÀè3ÁV¶u C3Å¶Ê+ñ3Ò£C÷õ^:ÀsÃ[:EvE]ÅUìVÿujèpÿÿÿÆE¾@GÀtS¶ØWÞ]jzjaèPÿÿÿGKuñ_[¶EÆ@G
Æ^]Å
UiihSVW3ÛEPÇETQIXÇE\GYMÇE eWtoÇE¤}ÇE¯wokÇE¬~{ÇE°w_ÇE´ÇE¸yÇE¼ÇEÀ}¡ÇEÃÇEÈõúÇEÌü]Ð]Ô]Øèùÿÿ SøEüPS¾?
VSSSWhÿdD;ÃtSEüPSVSSSWhADÿdDëÆAD;ÃuQÿuÇEÜNugxÇEàxiÇEã ´¯]è]i]õ]õf]øÿPÿuEÜjSPè ûÿÿPÿuüÿlDÿuüÿtD
_^[ÉÃUìiW3ÿ9}u3Àé±SVhlÿuÿlðuü;÷j@h0j WÿpØ;ßtzEPj@jV}õ}ÿÀtah´FÿxEüû¥¤Ë+ËéK¹¦+ÊéMùMõQÿuÆC
éøÆEøéfÇEýÆEÿuø¥jP¤ÿh´Fÿ\Ãë3À^[_ÉÃUìi`S3ÛSEüPSh?
SSSE PÇE TqixÇE¤|gymÇE¯eWtoÇE¬}ÇE°wokÇE´~{ÇE¸w_ÇE¼ÇEÀyÇEÃÇEÈp¤ÇEÌ¢ÇEÐ¤õÇEÔúü]Ø]Ü]à]ãf]èèùÿÿPhÿdD
ÅujVEøP¾XSVEõPSEiPÇEøÇEimqlh]ðèXùÿÿPÿuüÿhDÀt(8XSt ÆP@:Ëuù+ÅPVjSEìPÿuüÿlDÿuüÿtD^[ÉÃÀt 8tPÿ
Ã3ÀÃUìãøiVW3ÀjÆD$Y|$ ó«f«²¡(CÀtqL$0QÿÐÿuD$4ÿuPÿ,CD$0Pÿ8D3õ¶4Ð6ÁëBàBFTD þ|ÓUD$ÆDÈ+Ñ@Éuõ3À@ë3À_^å]Ã
=øFuhXÿ£øFÀøVhÀ@è0øÿÿ5lPÿ5øFÿÕhÐ@£dDèøÿÿPÿ5øFÿÕhã@£hDèú÷ÿÿPÿ5øFÿÕhó@£lDèâ÷ÿÿPÿ5øFÿÕhA£pDèÊ÷ÿÿPÿ5øF
ÿÕhpA£tDè²÷ÿÿPÿ5øFÿÕhxA£(Cè÷ÿÿPÿ5øFÿÕhA£,Cè÷ÿÿPÿ5øFÿÕh A£8Dèj÷ÿÿPÿ5øFÿÕhXA£|DèR÷ÿÿPÿ5øFÿÕ£D
^ÃUìi$S3ÛSSjShÀhSÿEðøÿchÿ5ØFhRÿÐÃ¸-1Pj@EìÿHEü;Á%VW8Tt48xSt,hxSÿ;ÃthdPÿl;ÃtShXShTÿÐ=0]à]ô]äÿ×5
EÜJøëjdÿTÿ×+EÜ=øw EäPEõPEøPjEàPÿuðÿÕ}øEèuÌ9]è
offset: 0
file_handle: 0x00000024
filepath: C:\Users\pc1\AppData\Local\Temp\yhxlsc.v

Figure C.8: Investigation 4 - Content of the dropped file