A HEURISTIC FEATURE BASED QUANTIFICATION FRAMEWORK FOR EFFICIENT MALWARE DETECTION

Measuring the Malicious intent of a file using anomaly probabilistic scoring and evidence combinational theory with fuzzy hashing for malware detection in Portable Executable files.

Anitta Patience Namanya

Submitted for the degree
of Doctor of Philosophy

School of Electrical Engineering and Computer Science
University of Bradford

2016

To my Mum and Dad, Alice and Nathan for their love, prayers and never ending encouragement.
To Mac, thank you for being you.
To Amara, very excited to meet you.
To my sisters Itah, Anolyne, Anglyne and Anabella, the best siblings anyone would ever ask for.

ABSTRACT

Anitta Patience Namanya "A HEURISTIC FEATURE BASED QUANTIFICATION FRAMEWORK FOR EFFICIENT MALWARE DETECTION"

Keywords: Malware detection, evidence combinational theory, hashes, portable executable, file features, anomalies, probabilistic scoring, automated static analysis, malware

Malware is still one of the most prominent vectors through which computer networks and systems are compromised. A compromised computer system or network provides data and or processing resources to the world of cybercrime. With cybercrime projected to cost the world $6 trillion by 2021, malware is expected to continue being a growing challenge. Statistics around malware growth over the last decade support this theory as malware numbers enjoy almost an exponential increase over the period. Recent reports on the complexity of the malware show that the fight against malware as a means of building more resilient cyberspace is an evolving challenge. Compounding the problem is the lack of cyber security expertise to handle the expected rise in incidents.

This thesis proposes advancing automation of the malware static analysis and detection to improve the decision-making confidence levels of a standard computer user in regards to a file's malicious status. Therefore, this work introduces a framework that relies on two novel approaches to score the malicious intent of a file. The first approach attaches a probabilistic score to heuristic anomalies to calculate an overall file malicious score while the second approach uses fuzzy hashes and evidence combination theory for more efficient malware detection. The approaches' resultant quantifiable scores measure the malicious intent of the file.

The designed schemes were validated using a dataset of "clean" and "malicious" files. The results obtained show that the framework achieves true positive – false positive detection rate "trade-offs" for efficient malware detection.

ACKNOWLEDGEMENTS

I would like to thank God for everything; His Grace, Guidance and the faith I have in Him to enable me finish this study. I would like to thank my parents for loving me unconditionally and their belief that allowed me to believe that I could accomplish all I set out to do. I am eternally grateful for Mac, baby Amara and my sisters. You are the inspirations that make me do my best each day.

I am very grateful to the University of Bradford for funding this research degree. This was an amazing opportunity that I shall forever be thankful for.

This work would not have been possible without the untiring support of my supervisors; Prof Irfan Awan, Dr Jules Pagna Disso and Dr Andrea Cullen who gave me so many chances to get this right that I am glad, I did not let you down. Thank you very much for your belief in me and your support. I also would like to extend both my examiners for the corrections suggested that enabled me to make this thesis a better write-up of my work.

My guardians in faith; Pastor Akpo. Pastor Andrea and Chapel of Grace friends, thank you for standing with me in prayer and being my tether to the Lord that makes all things amazing in His time.

A big thank you to Nettitude Ltd for all the support during this study. Access to data, technical support and the opportunity to work with such an amazing group of people shaped my thinking process every day.

I am indebted to all my siblings for being the much-needed inspiration, my colleagues in the cybersecurity research group; Adeeb, Rashid, Qublai and Hammad for being great sounding boards in these 4 years. To my new working family at Nettitude especially my boss Mark, my colleagues in Research at Bradford, thank you for all the laughter and jokes that kept me sane. To all my friends and family that kept me believing I could do this, Rashid, Mac, Bashir, Kabiru, Odili, Mwanja, Kemi, Pamela, Amanda, Aunt Joy, Aunt Jadress and many others, thank you and May God bless you.

PUBLICATIONS:

1. Namanya, A.P; Pagna-Disso, J; Awan, I (2015): Evaluation of automated static analysis tools for malware detection in Portable Executable files; *UK Performance Engineering Workshop(UKPEW), 2015 31st UKPEW, pp. 81-95, 17 Sept 2015*, University of Leeds, UK.
2. Namanya, A.P; Mirza, Q.K.A; Al-Mohannadi, H; Pagna-Disso, J; Awan, I (2016): Detection of Malicious Portable Executables using Evidence Combinational Theory with Fuzzy Hashing; *Future Internet of Things and Cloud (FiCloud2016), 2016 IEEE 4th International Conference, 22-24 August 2016*, Vienna, Austria.
3. Al-Mohannadi, H; Mirza, Q.K.A; Namanya, A.P; Pagna-Disso, J; Awan, I (2016): Cyber-Attack Modelling Analysis Techniques: An Overview; *Future Internet of Things and Cloud Workshops (W-FiCloud2016), 2016 IEEE 4th International Conference, 22-24 August 2016*, Vienna, Austria.
4. Namanya, A.P; Mirza, Q.K.A; Al-Mohannadi, H; Cullen, A; Awan, I (2016): Towards Building a Unified Threat Analysis and Management Framework; *UK Performance Engineering Workshop(UKPEW)& Cyber Security Workshop (CyberSecW), 2016 32nd UKPEW&CyberSecW, 7th – 8th Sept 2016*, University of Bradford, UK.
5. Munir, R; Ahmed, B; Al-Mohannadi, H; Rafiq, M; Namanya, A.P; (2016): Performance Security Trade-off of Network Intrusion Detection and Prevention Systems; *UK Performance Engineering Workshop(UKPEW)& Cyber Security Workshop (CyberSecW), 2016 32nd UKPEW&CyberSecW, 7th – 8th Sept 2016*, University of Bradford, UK.
6. Namanya, A.P; Pagna-Disso, J (2013): Performance modelling and analysis of the delay aware routing metric in Cognitive Radio Ad Hoc networks; *Wireless and Mobile Networking Conference (WMNC), 2013 6th Joint IFIP , vol., no., pp.1,8, 23-25 April 2013*, Dubai, UAE.

Submitted Journal Work:

1. Namanya, A.P; Mirza, Q.K.A; Al-Mohannadi, H; Pagna-Disso, J; Awan, I: Malicious Portable Executables Scoring Methodology using Evidence

Combinational Theory with Fuzzy Hashing.; **Second stage review** - *IEEE Transactions on Dependable and Secure Computing 2017.*

Posters:

1. Namanya,A.P; Pagna-Disso, J, Awan, I (2015): A framework for automated hybrid signature generation for Portable Executable malware detection; Poster presented at the ACM-W UK Inspire 2015 Celebration of Women in Computing. Imperial College, London, UK – Awarded 2nd Prize in Best Poster Award.
2. Namanya, A.P; Pagna-Disso, J, Awan, I (2016): Malicious PE Static Scoring method using Evidence Combinational Theory with Fuzzy Hashing; Poster Presented at ACM-W Europe WomENcourage 2016 Celebration of Women in Computing. Johannes Kepler University, Linz, Austria.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1. INTRODUCTION

1.1 Motivation

The internet has revolutionised the way the world manages information and data in academia, industry or individually. The evolution of the internet of things means that even the way of life is changing through the integration of technology use in everyday activities. With all the positives of having information at the fingertips, self-driven cars, smart homeware comes the downside of having compromised systems. The internet of things has already proved to be a very successful resource of a DDOS attack against KrebsonSecurity website [1] by “Mirai” [2] botnet malware. The DDoS attack set the highest record traffic known yet against a website on 620Gbps. The same botnet is also known to have been used to arguably take Liberia off the internet [3]. Cybercrime which is powered by criminals having access to compromised systems and information cost United Kingdom businesses over £ 1 billion and United States of America businesses over $3.8 billion in 2015 [4], [5]. Having already seen an increase of 200% in the last five years [6], cybercrime is expected to cost the world $ 6 trillion by 2021 [7] making it one of the most lucrative businesses. One of the biggest banking theft heists of 2016 was the $81 Bank of Bangladesh swift attack where malware was used to compromise a poorly configured network switch [8]

Malware is still one of the main vectors used to compromise networks and computer systems. With the expected growth in cybercrime, the fight against malware as a need in order to build a more resilient cyber space will continue to be challenging for both the industry and research community. The statistics where almost 600 million malware have been collected in 2016 [10] as of writing this report indicate that analysis and thereafter detection of these samples is bound to be a daunting task.

There is a scarcity of cyber security expertise [9], [11] which implies that there is not enough skilled people in the world to manage the ever growing cyber incidents, later on perform in-depth malware analysis. Automation of the malware analysis processes and detection of malware similar to already detected malware allows for the reduction of the ever widening gap created by the growing malware samples collected [12]

Many of the anti-malware solutions are fighting to keep up with the evolution of malware [13] and there is a continuous looming challenge of not having malware analysis and detection methods that are fast enough. Malware signatures that utilise a lot of resources and file scanning as a step in malware detection end up requiring a lot processing power [14]. Present day systems are high performance and resource-aware and therefore having an anti-malware scanning process like sandboxing which requires a lot of time is not optimal in these systems [12] especially when dealing with the huge number of samples discovered daily.

Malware; the ever-evolving threat to the cyberspace, provides an interesting challenge and therefore this study was undertaken to find a working solution that can strengthen the anti-malware community. For the rest of this chapter, we define proposed solutions in section 1.2, the aims and objects in section 1.3, our contributions in section 1.4 and scope of the work in section 1.5. We then provide an overview of the presentation structure of this thesis in section 1.6.

1.2 Proposed Solutions

The growing need for advanced secure systems implies that new, efficient and faster malware detection algorithms are required. Therefore, better alternatives to present day methods must be developed or existing methods need to be optimised with new approaches. Malware analysis and detection has been a topic of interest in the research community and a lot of advancements have been achieved in many of the malware analysis methods [15]. Unlike signature based detection methods, heuristics utilise different features in the malware and have proven to be better at unknown malware detection [16]. Heuristic based detection in the anti- malware research is used to describe effective algorithms used in malware detection which do not necessarily provide an optimum solution [17]. Combining multiple features to make a decision is seen in heuristic engines which use machine learning with malware attributes to provide improved detection [18]–[21]. This work proposes utilising the theory of multiple attribute based decision making, evidence combinational mathematical models and heuristic analysis to automate and improve malware analysis and detection. We propose measuring the malicious intent of a file by attaching a calculated score based on uncertainty models and probability scoring. The proposed solutions are:

a) We use probabilistic scoring and combine evidence from different heuristic anomaly features of PE files to measure the file's malicious intent. File format anomalies as some of the known heuristic indicators of compromise. Although the anomalies have been used in heuristic engines as a way of detecting malware [22] , there is no defined approach and scoring mechanism that can allow of a more universe file malicious intent score. We proposed using probability theory to make an intelligent algorithm that scores the file based on known anomaly chances in clean vs malicious files. Using a sample dataset, the anomaly scores are calculated and then used on another dataset for validation. CHAPTER 5 5 details the proposed method in achieving the scoring system. The developed algorithm automates the analysis and detection process while also attaching a quantifiable metric. The metric is used for better cognitive based decision making of the user thus improving their confidence in their decision on whether the file is clean or malicious.

b) We look at statically computed similarity hashes as heuristic feature representatives and investigate the effect of similarity hashes in relation to malware detection. Statistics from AV- test institute [10] show that only 12% of malware samples collected by the end of 2015 were new malware samples. This implies that similarity of the different sections and the files can be a measure of file maliciousness and therefore malware detection. The initial stage of triaging the malware to cluster the samples is a well-known step that normally uses hashing as a methodology. Although hashing faces the issue of high false negatives, a combinational approach could lead to better results. Our study in chapter 6 focuses on how to combine known similarity hashing methods in order to produce effective malware detection results. The similarity in the files detected by the hashing functions is used as the attribute similarity factor for a sample dataset and multiple attribute decision making and evidence combination mathematical models are applied to attach a score representing the malicious intent of the file. This automates the malware analysis and detection process and presents the system user with a quantifiable metric that improves the confidence in their decision making about the malicious state of the file.

1.3 Aims and Objectives

The main aim of this thesis is to design and implement two heuristic malware scoring approaches that make for a more efficient malware detection framework in Portable Executable files. Efforts towards achieving this aim led to the following objectives:

- Evaluate three prominent, open source automated malware static analysis tools focusing mainly on the analysis of Portable Executable files.
- Use the best automated static analysis tool to perform a study of PE file feature anomalies.
- Formulate approaches towards providing a malware scoring mechanism for improved confidence level in decision making for a standard computer user.
- Design, implement and test a heuristic scoring approach that uses the identified anomalies for detection of malicious portable executable files.
- Identify a fast and light heuristic based similarity hashing functions that can be combined for better malware detection rates in portable executable files.
- Carry out an investigation into the effect of the application of evidence combinational theory to the hashing analysis for more efficient malware detection and design, implement and test an approach that uses these techniques for more efficient malware detection.

1.4 Contributions

The main contribution of this work is the design of the approaches for a heuristic feature based quantification framework for efficient malware detection. The process of achieving this main contribution lead to:

1. An **evaluation study** of three prominent automated malware static analysis tools. This study provides an in-depth analysis of three prominent tools and leads to table that compares the various features of the different tools so that the reader can make an informed decision on which tool is better depending on their requirements. This study also identifies features

that are later incorporated into the design and implementation of the proposed approaches. Using a dataset of 2620 malicious samples, the study provides important statistics that are extracted using the tools and are detailed in chapter 4

2. **MalScore**: This is the first proposed approach that uses heuristic anomalies and attaching a probabilistic score so that the file's malicious intent can be quantified. This approach utilises a combination of Bayesian probability theory and the theory of total probabilities to generate the anomaly scores. Statically extracted features for Portable Executable files are compared malicious files vs the clean executables as a training phase in the design of the scoring method. The resulting file score can be used as a detection strategy or as a customised score to filter large datasets. Chapter 5 details the design, implementation, testing and analysis of this approach.
3. **MalHaScore**: This is the second approach and it introduces and investigates how different similarity hashing techniques can be combined to achieve better malware detection rates. Initially, this study explores 4 different hashing techniques that are currently used in malware analysis. Although each hashing technique produces interesting results independently, two evidence combination theory based methods are applied in order propose a novel way of combining the results achieved. The results achieved show that the detection rates are improved when evidence combination techniques are applied. Chapter 6 details the design, implementation, testing and analysis of this approach.

1.5 Thesis Scope

This work focuses on the use of automated static analysis and the features extracted to design and implement approaches that can provide efficient malware detection. This implies that dynamic analysis based approaches are not covered and are outside the scope of this work.

Using file heuristic features requires that file structure is uniform so the scope of this work covers only Portable Executable (.exe) files. We focus on designing approaches for improving the malware detection rates in PE files with the expectation that these approaches can be replicated for other file formats.

Replication of the approaches for other file types is outside scope and is not covered herein.

The implemented methods are tested on a dataset of known malicious files and clean files that are specified in section 2.5.2. Since our objective was to improve malware detection through scoring the malicious intent of the file, tests using a different dataset should achieve similar malware detection trends but different results.

1.6 Thesis Structure

The rest of this thesis is structured as follows:

- CHAPTER 2. BACKGROUND: This chapter covers the introduction to the world of malware by briefly discussing the various malware types, the evolution and the number growth trends seen in the last decade. It then also discusses the Portable File format structure, the evaluation methods employed in this study and the dataset preparation process.
- CHAPTER 3. LITERATURE REVIEW: This chapter discusses malware analysis techniques, the evasion techniques used by present day malware to thwart analysis and detection and the known malware detection techniques. It then delves into relevant recent literature related to the study that forms the foundation of the work presented in this thesis.
- CHAPTER 4. EVALUATION OF AUTOMATED STATIC ANALYSIS TOOLS FOR MALWARE DETECTION IN PORTABLE EXECUTABLE FILES: Three new automated static analysis tools are evaluated in this chapter. The intention is to identify the tool that extracts information needed for the study. The work presented in this chapter provides insight into how the evaluation was carried out, the results achieved, observations made and how the challenges faced were overcame.
- CHAPTER 5 MALSCORE: AN ANORMALY HEURISTIC FEATURE BASED PROBABILISTIC SCORING METHOD FOR DETECTION OF MALICIOUS PORTABLE EXECUTABLES: This chapter covers the design, implementation and analysis of the first proposed approach; MalScore. The details pertaining to all the steps taken to achieve and test the proposed scheme are discussed in detail and the results are presented and analysed. Arguments for this approach are discussed in the chapter.

- CHAPTER 6 MALHASCORE: MALICIOUS PORTABLE EXECUTABLE STATIC SCORING METHODOLOGY USING EVIDENCE COMBINATIONAL THEORY WITH HEURISTIC FEATURE CALCULATED HASHES.: This chapter discusses in the detail the design, modelling, implementation and testing of the second approach proposed. It further discusses the analysis results achieved for this approach and the reasons as to why this method works.
- CHAPTER 7 CONCLUSION This chapter presents the conclusion of the thesis and the known limitations of the proposed approaches. We also discuss some of the interesting areas that have been identified for future work in association to this study.

Malware, short for Malicious software is a program code that is hostile and often used to corrupt or misuse a system [12]. Introducing malware into a computer network environment has different effects depending on the design intent of the malware and the network layout. The internet and the worldwide web have given great advancement in how society communicates and the face of business. This has also given rise to the number of propagation avenues available to malware. Introduction of malicious code in one node can create a chain reaction across all the nodes accessible through the network, the node seats on. With organisations and countries heavily relying on network technologies, the commercial value of computer networks implies that exploitation of the vulnerability of the business network can cripple its operations and provide access to intellectual property and personal information to cyber criminals. This creates a commercial opportunity for malware and anti-malware ventures and therefore it is no surprise that it is estimated that cybercrime is expected to raise.

Malware has been a persistent problem across all computer networks and detection of malware is quite vital in securing a networking environment. The common used form of malware detection is signature based detection where a malware “signature” is constructed from unique patterns and characteristics derived from the malware code and the file contents. This is the technique used in most commercial anti-malware systems and the malware signature is relied upon because already discovered malware always tend to keep in circulation [22]. This implies that if an analysed malware is identified to have a form of an already known signature in its characteristics, it can be isolated or the program can be deleted from the system without knowing what other system changes it could have targeted.

With more than 91% [23] of computer users still using Windows Operating Systems based computers, more efficient detection of malicious files in the windows environment is of paramount importance. Moreover, the introduction of Windows 10, the cross-platform compatibility across different devices leads to the expectation that the analysis of the PE structure will play a big role in future endpoint security as the Internet of Things (IoT) evolution takes off.

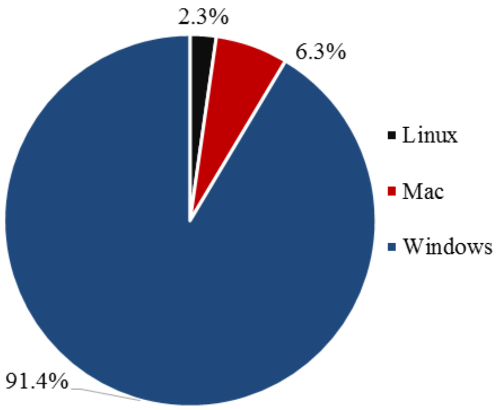


Figure 2.1 OS Market Share [23]

The fact that malware is software means that it follows a standard format required for it to be executed on the target. Portable Executable (PE) is the format for Microsoft binary executables and is very well document in [24]. Heuristic based detection uses distinct file features identified to be specific to malware for more efficient malware detection like the file compile time. Over the years, malware has evolved to add detection evasion methods also called obfuscation techniques. This has led to the introduction of semantic based detection where malware is detected based on the information obtained during the behavioural analysis of the malware sample. This chapter explores malware as an evolving topic and the structure of the Portable executable file with respect to heuristic based malware detection which is the focus of this study.

2.1 Malware Overview

Malware is defined as "*any code added, changed or removed from a software system in order to intentionally cause harm or subvert the intended function of the system*" [25]. The fact that malware can bring down a system which can cause loss of information and therefore money as well as life [26] represents a big threat to technology advancements. Recent developments in computer systems have seen a corresponding if not higher growth in variants of malware and their functionality. To understand what the malware is intended for in the system therefore identifying it, there is need to understand what class it falls under. The classification of malware depends on execution characteristics of the program. Malware is also classified depending on its payload, how it exploits or makes the system vulnerable and how it propagates [27]. This enables the malware to be subdivided into different types as further discussed below.

Virus

A virus is self-replicating malicious program. It exists as an executable and spreads by copying itself to other host systems. It is passive and needs to be transferred through files or media files or network files. Depending how the complex the code is, it can modify the replicated copies of itself [26], [28], [29]. Viruses can be used to harm host computers and networks, steal information, create botnets, render advertisements, and steal money among other malicious activities.

Worm

This is a self-replicating and active malicious program that can spread over the network by exploiting various system vulnerabilities. It uses targeted vulnerabilities in the operating system or installed software. It contains harmful routines but can be used to open communication channels which serve as active carriers. The Worm consumes a lot of bandwidth and processing resource through continuous scanning [28] and makes the host unstable which can sometimes cause the system to crash. It may also contain a payload that are pieces of code written to affect the computer by stealing data, deleting files or create a bot that can lead the infected system being part of a botnet. While viruses require human activity to spread, worms have the ability spread and replicate independently.

Trojan horse

Commonly referred to as Trojan, this is a program that presents as a legitimate software which when downloaded and executed embeds malicious routines or files on the host [29], [33]. In most cases, the Trojan horse when executed will install a virus or may have no payload. It cannot self-replicate and relies on the system operators to activate. It can however give remote access to an attacker who then can perform any malicious activity that is of interest to them. Trojan horse programs have different ways they affect the host depending on the payload attached to them and are usually spread through social engineering [30].

Spyware

This is a malicious program that uses functions in an operating system with the intention of spying on user activity. They sometimes have additional capabilities like interfering with network connections to modifying security settings on the system on the infected host. They spread by attaching themselves to legitimate software, Trojan horse or even taking advantage of known software vulnerabilities. Spyware can monitor user behaviour, collect keystrokes, internet usage habits and send the information to the program author [31].

Adware

Adware which is short for advertising supported software, automatically delivers advertisements seen especially in website pop-up ads and displayed by software. Most are designed to serve as revenue generating tools by advertisers. Some adware may come packaged with spyware which then makes this very dangerous as it can track user activity and steal user information [31], [32].

Root Kit

This is a program that employs a set of tools to avoid detection in a system. The tools are very advanced and complex programs written to hide within the legitimate processes on the computer infected therefore are very invasive and are difficult to remove. They are designed with the capability of taking full control of the system and gaining the highest privileges possible on the machine among other possible malicious activities [22], [28]. Because of the evasion techniques used by rootkits, most security vendor solutions are not effective in detecting and removing them and therefore, their detection and remove rely heavily on manual efforts. These may include but not limited to monitoring computer system behaviour for abnormal activities, storage dump analysis and system file signature scanning.

Bots

Bots are programs designed to perform specific operations. Bots are derived from 'robots' which were first developed to manage chat channels of IRC- Internet Relay Chat a text based communication protocol that appeared in 1989 [33]–[36]. Some bots are used for legitimate purposed like video programming and online contest among other functions. Malicious bots are designed to form botnets. A botnet is defined as a network of host computers (zombies/bot) that is controlled by an attacker or botmaster as seen in Figure 2.2. Bots infect and control other computer which in turn infect other connected computers thus formulating a network of compromised computers called a botnet. Bots are very commonly used as spambots, DDOS attacks, webspiders to scrape server data and distributing malware on download sites. CAPTCHA tests are used by websites to guard against bots by verifying users as humans.

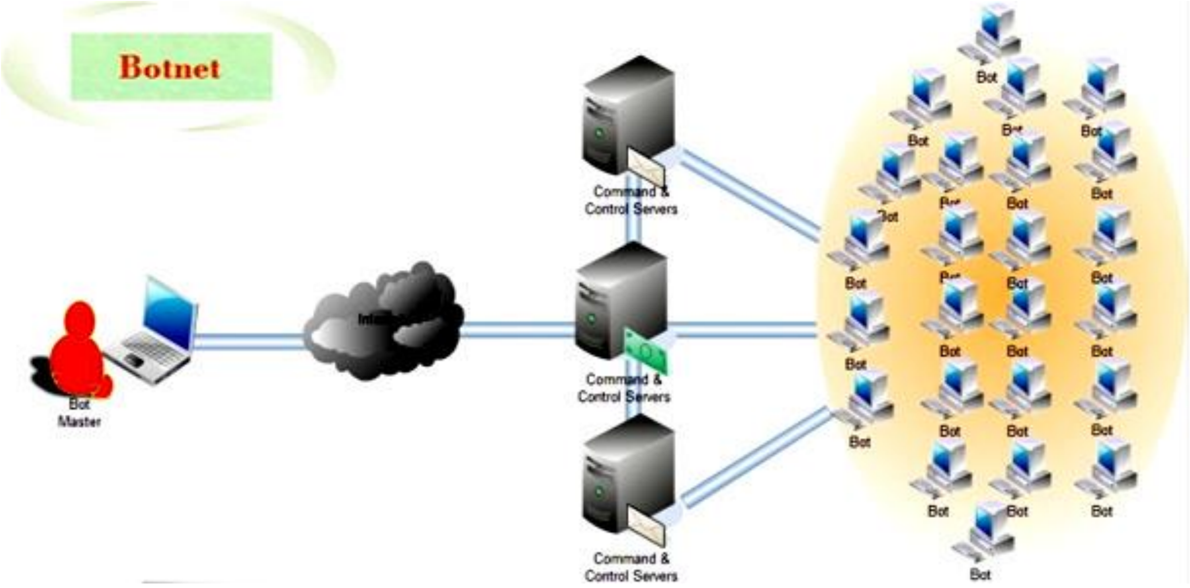
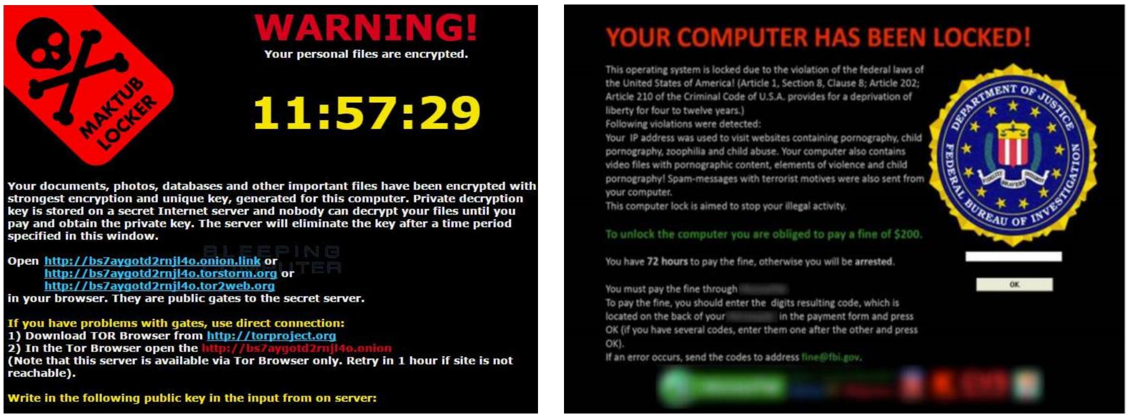

Botnet
Bot Master
Internet
Command & Control Servers
Command & Control Servers
Command & Control Servers
Bot

Figure 2.2 An Example of a Botnet layout [37]

Ransomware

Ransomware is a program that infects a host or network and holds the system captive while requesting a ransom from the system/network users. The program normally encrypts the files on the infected system or locks down the system so that the users have no access. It then displays messages as seen in Figure 2.3 that force the users to pay to have access to their systems again.



WARNING!
Your personal files are encrypted.

11:57:29

Your documents, photos, databases and other important files have been encrypted with strongest encryption and unique key, generated for this computer. Private decryption key is stored on a secret Internet server and nobody can decrypt your files until you pay and obtain the private key. The server will eliminate the key after a time period specified in this window.

Open http://bs7aygotd2rnjl4o.onion.link or
http://bs7aygotd2rnjl4o.torstorm.org or
http://bs7aygotd2rnjl4o.tor2web.org
in your browser. They are public gates to the secret server.

If you have problems with gates, use direct connection:
1) Download TOR Browser from http://torproject.org
2) In the Tor Browser open the http://bs7aygotd2rnjl4o.onion
(Note that this server is available via Tor Browser only. Retry in 1 hour if site is not reachable).

Write in the following public key in the input from on server:

YOUR COMPUTER HAS BEEN LOCKED!

This operating system is locked due to the violation of the federal laws of the United States of America! (Article 1, Section 8, Clause 8; Article 202; Article 210 of the Criminal Code of U.S.A. provides for a deprivation of liberty for four to twelve years.)
Following violations were detected:
Your IP address was used to visit websites containing pornography, child pornography, zoophilia and child abuse. Your computer also contains video files with pornographic content, elements of violence and child pornography! Spam-messages with terrorist motives were also sent from your computer.
This computer lock is aimed to stop your illegal activity.

To unlock the computer you are obliged to pay a fine of $200.

You have **72 hours** to pay the fine, otherwise you will be **arrested**.

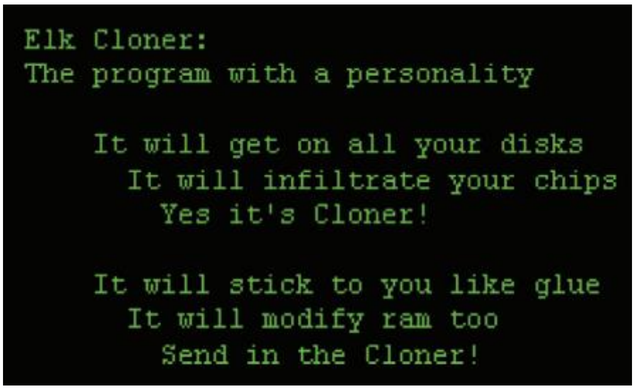
You must pay the fine through [redacted]
To pay the fine, you should enter the digits resulting code, which is located on the back of your [redacted] in the payment form and press OK (if you have several codes, enter them one after the other and press OK).
If an error occurs, send the codes to address fine@fbi.gov.

Figure 2.3 Ransomware display messages [38], [39]

Ransomware uses the same propagation means as a computer worm to spread and therefore user awareness and system updates are important mitigation measures.

2.1.1 Evolution of Malware

Malware has evolved from the days when it was an exciting prank/experiment gone wrong or uncontrolled to now when malware is used for commercial gain by exploiting and stealing user information. There are various documented instances of malware created within a laboratory setting like the 1962 Darwin game, 1971 Creeper, 1974 Rabbit Virus and 1975 Pervading Animal [28]. However, all the mentioned malware were kept within a laboratory environment and never escaped to the wild. The first virus known to have been able to escape its creation environment was the Elk Cloner introduced in 1981, six years after the first personal computers [40]. Elk Cloner infected Apple DOS 3.3, spread by attaching to disks introduced to the system and once triggered, it run a poem shown in Figure 2.4. After the success of this prank gone wild, Brain the first Microsoft PC virus was seen in the wild in 1986 [28] and like Elk Cloner, it was more annoying than harmful. However, it is the first virus known to conceal its existence on the disk thus evade detection. The next malware that would get out of hand and change the propagating properties of malware would be the Morris worm written in 1988 as an experimental, self-propagating, self-replicating program which was released on the internet [41].



```
Elk Cloner:
The program with a personality

    It will get on all your disks
      It will infiltrate your chips
        Yes it's Cloner!

    It will stick to you like glue
      It will modify ram too
        Send in the Cloner!
```

Figure 2.4 Elk Cloner Poem seen on infected Apple computers in 1981

In 1990, Yisreal Radai coined the term malware, short for malicious software that would thereafter be used to as a generic umbrella term for all software with undesired intent within a system [34]. The following decades saw an evolution in malware that is best defined as a two-dynamic evolution; the growth in complexity and malware sample numbers.

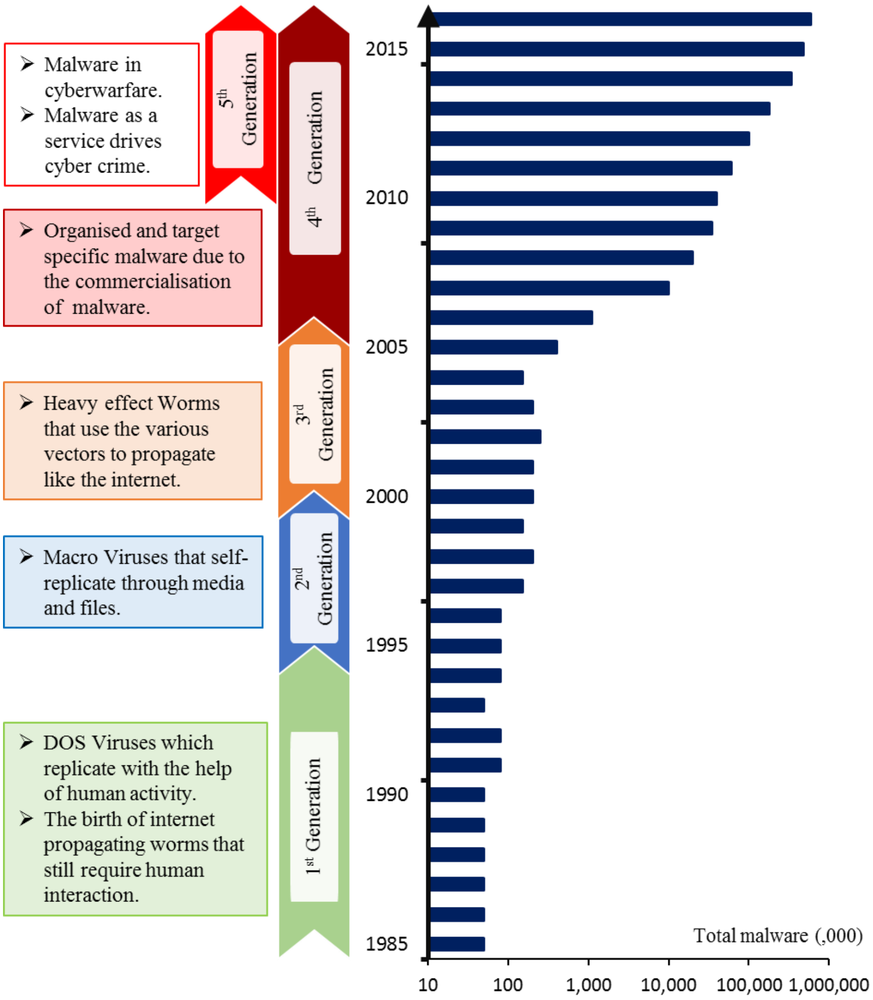


Figure 2.5 The evolution of generations of malware over the years

The growth in complexity is defined by the different generations of malware seen over the years [28], [30], [42] shown in Figure 2.5:

- The first generation (DOS Viruses) of malware mainly replicate with the assistance of human activity.
- Second generation malware self-replicate without help and share the functionality characteristics of the first generation. They propagate through files and media.
- Third Generation utilise the capabilities if the internet in their propagation vectors leading to big impact viruses.
- Fourth Generation are more organisation specific and use multiple vectors to attack mainly anti-virus software or systems due to the commercialisation of malware.

- Fifth Generation is characterised by the use of malware in cyberwarfare and the now popular malware as a service.

These generations group the various types of malware that were earlier identified and described. Each jump in generation is characterised by increase in complexity of the malware seen and more propagation vectors. The complexity in the malware discovered over the years always seem to follow the evolving trends in technology [28]. With the commercial value attached to having access to exploited systems or the ability to infiltrate a network, the malware writers do not want their malicious creations detected, analysed and rendered useless. This has led to the birth of malware samples which are very evasive when it comes to their analysis and detection. The known methods seen in malware to evade analysis and or detection are discussed in the next chapter.

The evolution of the malware sample numbers is best reflected by the number of malware samples collected by AV-test institute over the last decade as shown in Figure 2.6 [43]. From the trends seen in the graph, although the total malware samples collected are growing in number, the percentage of the total that is new malware shows a decreasing trend over the last 3 years. This percentage shift enables us to predict that if old malware can be detected and eliminated from the samples automatically, then the analysis time spent on the discovered samples can be greatly reduced.

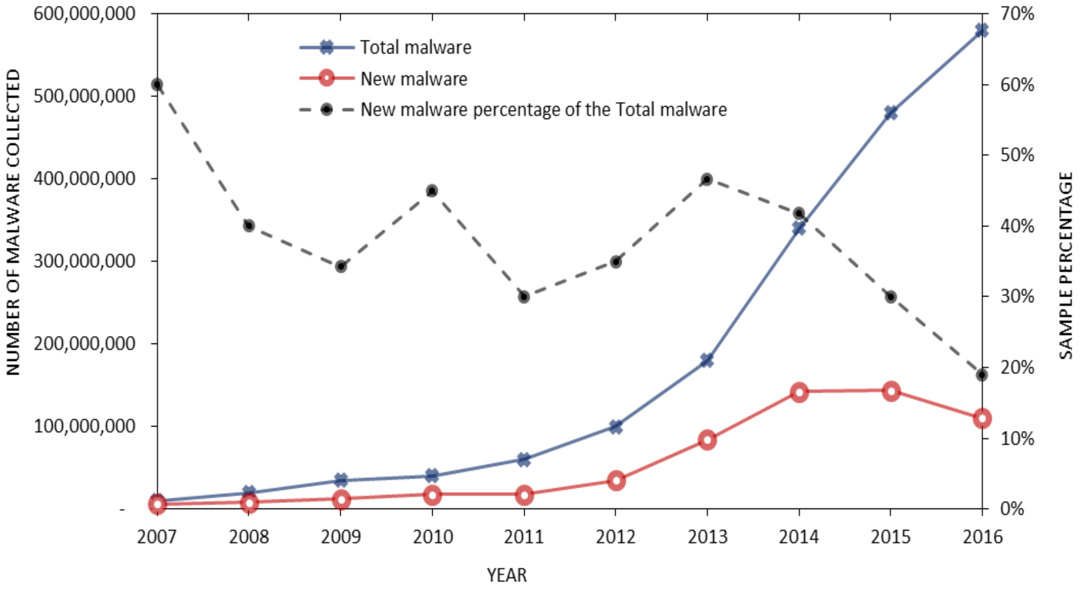


Figure 2.6 Malware sample statistics for the last decade

2.2 Portable Executable File Format (PE)

The Portable Executable (PE) format is an independent file structural format used for 16 bit and 32 bit windows systems. Pietrek [44] best describes the PE file format in a construction analogy "*a PE file is like a prefabricated home. It's essentially brought into place in one piece, followed by a small amount of work to wire it up to the rest of the world (that is, to connect it to its DLLs and so on).*" PE file format were introduced as part of the original specifications for Win32 with the release of Windows NT 3.1 in 1993 and was derived from the earlier Common Object File Format (COFF) [44]. It allows the Windows Operating system loader to manage the compiled executable code.

The PE file types are referred to as image files in [24] and the two types; DLL and EXE files are solely differentiated at the semantic level. DLL (Dynamic- Link Library) type files are used to export data or functions that other programs use. The functions are defined based on their intended use; internal or exported [45]. Internal functions are used with the DLL where they are defined while exported functions are used by other modules as well as with the DLL in which they are defined. A modular format provided by the design of the DLLs allows for easier modification and reuse in the ever evolving windows environment [24]. DLLs have various file extensions; .sys, .dll, ocx, .cpl, .fon, and .drv EXE type files (.exe) when launched run within their own process, not loaded into existing processes of other programs unlike DLL type files [46].

The creation and usage of EXE and DLL files is defined by the linker and loader as shown in Figure 2.7.

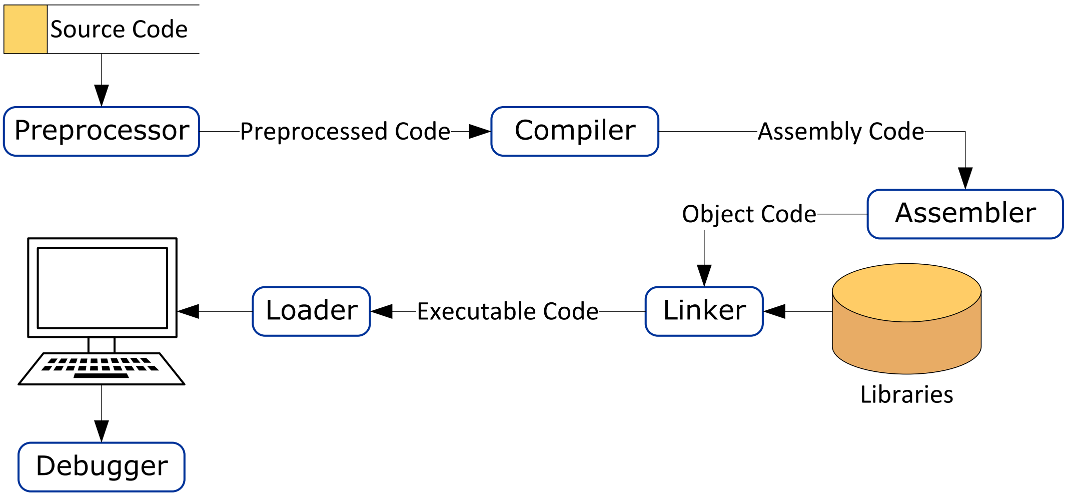


Figure 2.7 Relationship between the linker, executable(Image) and loader

A linker is a process that collects and combines different pieces of data and code into an executable file that can be loaded into memory. The linker combines libraries and object files into an executable code /image file. This processes leads to an image also called the portable executable file [24]. The loader is the program that loads the executable into main memory and an example is the windows installer.

Being Relocatable is one of the most important characteristics of the PE type files and this means that addressing during the loading process and therefore how internal address is handled in an PE file is important [46]. The addressing of the PE file is defined by the physical address, Base Address and the Virtual Addresses; relative or absolute. Physical addresses are used to access PE file parts that must be read from the storage disk. The base address stored in the field called ImageBase is the address of the first byte where the image file is loaded into memory [47]. The address specified in the ImageBase is always the preferred address although it is possible during the loading process of the image that this address might not be available in memory. This leads to the relocation of the module where another base address is chosen [46]. Relative virtual addresses (RVA) are offset addresses into the file that are used while the image file is loaded in memory. The loader calculates the required absolute address for the specific instruction by adding the base address to the RVAs. Using RVAs that are independent of the base address allows for the relocation of the image without having to re-calculate all the addresses. Although Virtual addresses are defined in [24] as absolute in-memory addresses , they are actually used relative to the BaseImage specified address. Addressing is important because malicious files tend to have anomalies in PE address field as intentional malformations, obfuscations mechanisms or malicious intent. These can be used as detected file vulnerabilities or indicators of compromise in malware prevention and detection.

The PE file Format has evolved over the years of the MS-DOS to the Win32 specifications to the now Win64 specification. Only a few changes have been introduced with each new specification because there is always need for compatibility across the operating systems and it makes sense for developers to use already existing and working file formats. The basic PE file format is best defined by Figure 2.8.

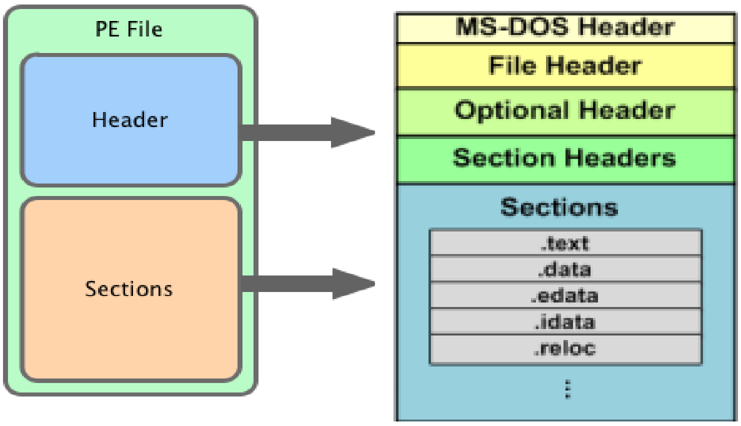
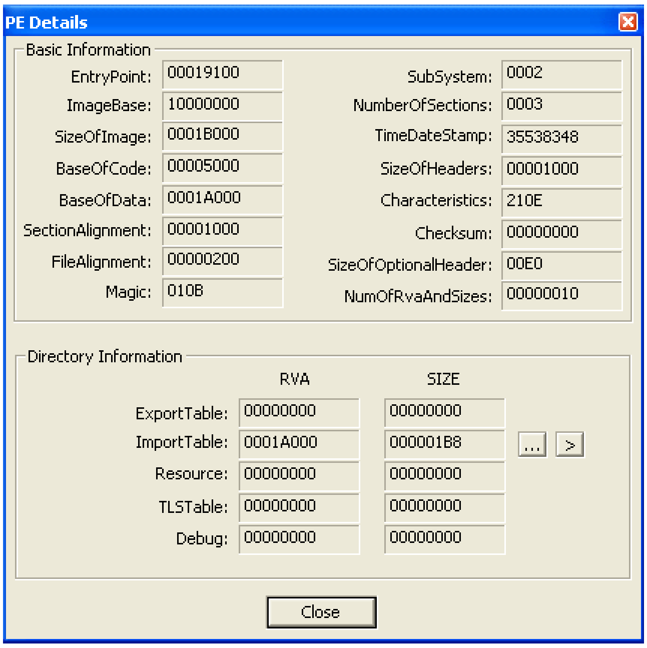


Figure 2.8 Generic PE File Structure [48]



Basic Information

EntryPoint:	00019100	SubSystem:	0002
ImageBase:	10000000	NumberOfSections:	0003
SizeOfImage:	0001B000	TimeDateStamp:	35538348
BaseOfCode:	00005000	SizeOfHeaders:	00001000
BaseOfData:	0001A000	Characteristics:	210E
SectionAlignment:	00001000	Checksum:	00000000
FileAlignment:	00000200	SizeOfOptionalHeader:	00E0
Magic:	010B	NumOfRvaAndSizes:	00000010

Directory Information

	RVA	SIZE
ExportTable:	00000000	00000000
ImportTable:	0001A000	000001B8
Resource:	00000000	00000000
TLSTable:	00000000	00000000
Debug:	00000000	00000000

Figure 2.9 The PE File details in PEiD

This known specification of the PE file allows for deeper file analysis based on its structure. For example, Figure 2.9 shows a file's structural details using PEiD; one of the many static parsers for the PE file format. However, the extract of the PE file details in PEiD in Figure 2.9 is quite limited based on the PE file detailed structure. There is more to explore in the PE file format and to do that, there is need to understand the standard structure as discussed in the next section.

2.2.1 PE Standard Structure

The PE Standard structure is a collection of different subdivisions which are necessary because the memory manager treats each differently when the file is loaded. Each subdivision contains different file content that are important for proper execution of the file. A normal PE standard file consists of the MSDOS Stub, PE File Header, and sections. Any other optional data is appended to the file also called an overlay [24] . Some applications use the overlay to store data without having to worry about the PE format or sometimes to prevent the data from being loaded into memory by the operating system.

As seen in [24], [44], [47], the PE file structure shown in Figure 2.10 starts with the MS-DOS Stub which is an application that has the ability to run in MS-DOS and is mainly supported for backward compatibility. The stub contains the file format signature; “MZ” in its magic feature and prints out the message “This program cannot be run in DOS mode“ if there is an attempt to run the image in DOS. This stub is ignored when the program is loaded in operations systems of higher versions. Next in the format is the PE File Header which consists of the PE signature (‘PE\0\0’), the COFF File Header, the Optional Header, and the Section Table. Compatibility of the PE file format to the operating system is specified by the certain signatures found within the file structure and always defined at the very beginning of the file. The PE signature is placed after the MS-DOS-Stub and its offset is defined in the e_lfanew field of the MS-DOS Stub.

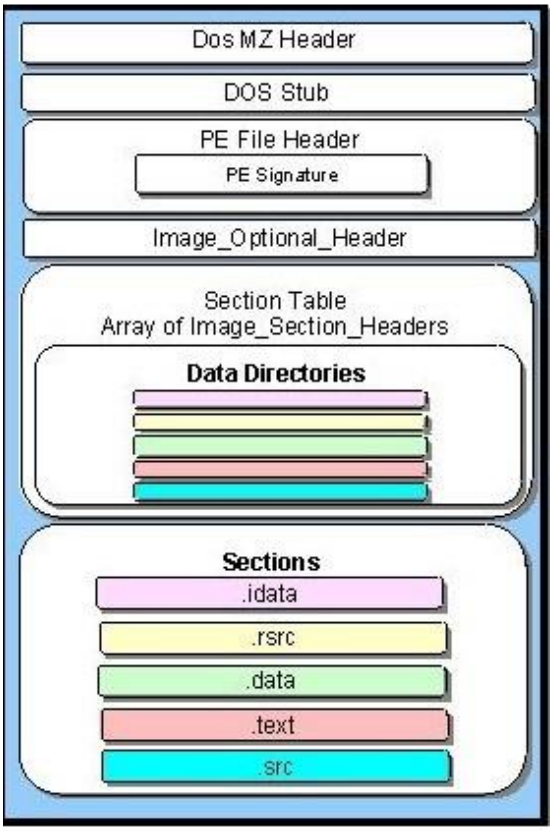


Figure 2.10 PE file stucture [49]

After the PE signature, the COFF file Header is located at the fixed offset relative to the start of the PE signature. Some of the features in this subheader are important in terms of anomaly analysis as a means for malware detection. Like all the substructure before, one of its features helps define the next chunk; SizeofOptionalHeader defines the size of the next subheader. The Optional Header which appears right after the COFF file Header is subdivided into 3 import chunks; Standard COFF Field, Windows Specific Fields and Data Directions. Some of the fields in these different subdivisions are later examined for the possibility to be heuristic indicators of compromise in case of detection of anomalies. After the optional Header, the Section table that contains the section headers; each defining the properties of related section found within the PE image file.

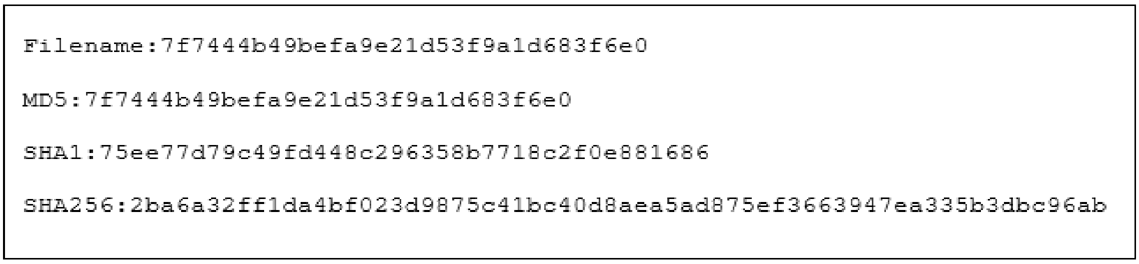
The sections as a chunk appear next and are subdivided based on the sections contained with the PE file. Each section defined as ‘basic unit of code or data within a PE or COFF file’ contains information specific for certain applications but not relevant to all. The naming convention of the sections allows for identification of the purpose of the section. However, since this naming convention is not strict, many malicious files and some legitimate packers and compile do not follow it. Misuse of section names is one of many anomalies is an interest in heuristic based detection despite the possibility of high false positives. Some of the section names that follow the standard convention are .text - a section that contains the executable or object code, .data- a section which consists of variable, uninitialized data, .bss – this section is typically used for program-wide initialized, global data, .rsrc – the resource section that consists of the embedded items.

2.3 Hashing algorithms

Hashing algorithms are computed digests of a file that are unique to the sequence of the file binary contents and structure [50].

2.3.1 Cryptographic Hashing algorithms

The cryptographic hashes like MD5, SHA1 and SHA256 shown in Figure 2.11 have been very popular in malware identification for integrity checking [51]. Since they are so unique, they are used when there is a need to have 100% match in the compared files as seen on most legitimate software distribution websites. An exact match of the file download shows that the file has not been compromised and is, therefore, safe before it is executed [22]. Whilst this is a safety measure that works for files where care is taken during file installation, sometimes, files are downloaded and installed in a hurry or without the knowledge of the computer user.



```
Filename:7f7444b49befa9e21d53f9a1d683f6e0

MD5:7f7444b49befa9e21d53f9a1d683f6e0

SHA1:75ee77d79c49fd448c296358b7718c2f0e881686

SHA256:2ba6a32ff1da4bf023d9875c41bc40d8aea5ad875ef3663947ea335b3dbc96ab
```

Figure 2.11 The Cryptographic Hashes of a malware Sample

These are the cases where detection of malicious or compromised files is very important. Although the different file hash types might not be very useful in such cases since a simple bit change in the file affects the hash digest computed, they have been repurposed and become useful in other methods. For example, section md5 and SHA hashes can be used to detect types of packing [52] and in some cases classification of malware families [53]. For this study, these specific hashes are not used, however, the work focuses on fuzzy hashing and feature specific file, section hashes that are further discussed in detail in the next subsections.

2.3.2 Fuzzy Hashing algorithms

Fuzzy hashing algorithms are designed to compare two different files and produce a percentage measure that represents the similarity between the files [54]. In this work, we review fuzzy hashes of interest that have been designed for malware detection and used in different studies focused on malware classification and detection.

2.3.2.1 Ssdeep Hash

Also known as Context Triggered Piecewise Hashing(CTPH) [54] or fuzzy hashing was first introduced in anti-spam research to detect similarity in files. Ssdeep one of the most famous methods of fuzzy hashing is a freeware, open source program that generates fuzzy hashes that when compared one against another, a similarity percentage score between the files is returned with a very high confidence of 99. The original idea of fuzzy hashing was developed by combining the piecewise hashing (Fowler/Noll/Vo –FNV hash) and the rolling hashing to produce a none cryptographic hash that is then used by a comparison algorithm that uses Levenshtein Distance to compare any 2 generated hashes for sequence similarity [50], [55]. The score is normalised to a range of [0-100] and the 50 is noted to be a reasonable threshold for a good match for spamsum algorithm.

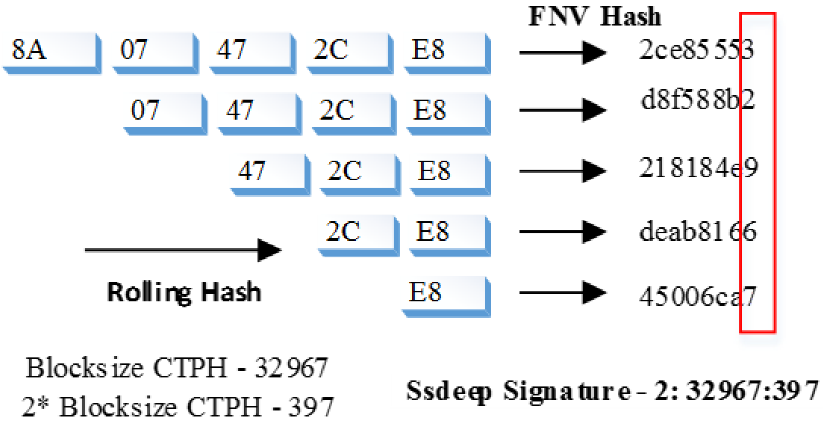


Figure 2.12 Calculating the Ssdeep Signature

```
Blocksize: Block_Signature:Double-Block_Signature
6144:tkDtqNp95Ltuj5K2gvuHqeYPYg31eaJq1DWBEU/e:utUpDtqKmw/LqJW
```

Figure 2.13 Ssdeep Signature Form

Kornblum [54] adopted it for Ssdeep for the purposes of forensic science and this application was extended to malware by the Mandiant cybersecurity firm with the purpose of providing the malware analysts [56] with information to guide their next step in the file analysis. An Ssdeep signature of a file takes the form shown in Figure 2.13 which also includes an extract of an Ssdeep hash of a file. When two hashes of two different files are compared, a similarity percentage score between the files is returned with a very high confidence of 99.

It is now a very crucial step in static analysis with many analysis tools attaching this hash next to the cryptographic hashes in any malware analysis report. This string is used to provide the similarity percentage [0-100] when compared with another hash from another file. The percentage of similarity attached to any two files can be sometimes the justification why the files are the same or in the same family of malware. In this work, the file Ssdeep hash and the resource section Ssdeep hash are considered.

2.3.2.2 Imphash

First proposed by Mandiant cybersecurity firm [57], Imphash is a hashing method that is calculated from the digest of the import section of the executable file. With many researchers focusing on the imported APIs as a way of understanding how the file would interact with the system based, the Import table which holds the APIs under traditional conditions provides added insight into the expected behaviour of the file. Its algorithm is implemented following these stages:

- Extract the structure of the PE file.
- Populate the imports in the order {API, Function (dll or sys or ocx)} for APIs found.
- Return the MD5 digest of the import strings populated.

Imphash matching allows the analyst to cluster malware based on the order and the contents of the import table. This means that a change of order in the imports in table compromises this hash value and packers can also be used to overcome this hash as a detection method since the import table is sometimes hidden. However, Imphash is still very useful considering that most malware share some common behaviour on how they interact with systems which allows for clustering and detection of similar structured malware. Imphash has been incorporated in many static analysis tools like VirusTotal.com, Peframe and Pefile among others.

2.3.2.3 PeHash

It is a function that generates the binary cryptographic hash value of the structural data found in the file header and executable’s sections [58]. Apart from the structure of the file, it also uses bzip2 compression ratio as an approximation for Kolmogorov complexity for obfuscated data in the sections. With the possibility that some malware repeat the use of specific encryption techniques, different instances of the malware sample can result in the same Kolmogorov complexity thus creating a clustering mechanism. The algorithm first creates 2 classes of hash buffers; global properties and Section hashes buffer. The structural features whose hash is included the calculation of the hash buffers in Table 2.1.

Hash Buffer	**Structural Features used**
Global properties	Image Charactersitics, Subsystem, Stack Commit Size and Heap commit size.
Section	Virtual Address, Raw Size, Section Characteristics

Table 2.1 PE file features included the PeHash Calculation

The bzip2 compression ratio for each section is included in the calculated section's hash buffer. The PeHash that is the result used is the SHA1 value of the overall hash buffer of the file. The analysis of this hash shows that this metric hash provides good clustering matches for instances of similar polymorphic malware samples. PeHash has not been fully extended into most static analysis tools since it is file type specific. However, since this work focuses on PE type files, this hash is considered.

2.4 Evidence Combinational Theory Methods.

These are methods used for decision support when there is uncertainty in the data being used to make the decision [59]. The introduction of uncertainty works well for malware detection as with each new sample analysed, there is always a degree of ignorance. This is mainly true because each file is deemed non-malicious until confirmed to be malicious. In malware analysis, analysts are more likely to obtain uncertain information from different analysis methods. Based on their expertise, they make decisions on the malicious status of the file. Evidence combinational theory provides a way to automate this process. If there are two different pieces of evidence with two different degree of belief (***X*** with Degree ***x*** and ***Y*** with degree ***y)*** that support the hypothesis (***M***) that the file is either malicious or not, the result heavily depends on the degree of belief placed on the different pieces of evidence. Methods of how to combine the different evidence is what is needed to be applied in the systems to make better informed decisions about how malicious the file is. However, with the need to keep the already designed systems working as they are modified to do better, there is need to look at building the frameworks using the existing and new tools to keep the malicious files out of the computing systems. In this study, likelihood of a file being malicious is based on different results of the hashing analysis. Using the fuzzy set union operators T-conorms introduces logical connectives to design the reasoning system [60] based on the degrees of belief. Strict Archimedean t-conorm are used because they can approximate every continuous t-conorm that takes the value in the range [0-1] [60]. This section discusses the two identified methods; fuzzy logic and the certainty-factor model that can be used to provide mathematically supported uncertain based decisions.

2.4.1 Fuzzy Logic

It is used in decision making where there is no deterministic data on which to rely the decision. The theory based on fuzzy sets was introduced in 1965 [61]and the resultant the fuzzy logic approach follows that the end result is only true if and only if either of the support evidence is true. Considering the initial hypothesis of Maliciousness (***M***), the degree in belief of this hypothesis using fuzzy logic approach defines the function:

$$x*y \text{ in } M \tag{2.1}$$

Using the important class of “strictly Archimedean” t-conorms of fuzzy logic [62] the algebraic sum is given by:

$$x * y = x + y - x \cdot y \tag{2.2}$$

This is the sum that used to assign a new percentage belief on the overall degree of belief in the hypothesis.

2.4.2 The Certainty Factor model

A reasoning method that manages uncertainty in rule based systems which was developed in 1975 for MYCIN expert system [63]. MYCIN was a rule based expert system that was designed to diagnosis infections due to bacteria [64]. To compute the overall belief in the hypothesis, an expert represents the uncertainty in the rule by using a single Common Factor (CF) for every rule. The CFs work as the degree of belief attached to each rule. Following the T-conorms, the degree of belief in the overall hypothesis using two supporting evidence is always higher or equal to the degree of belief in M obtained from one piece of evidence [65]. Using the notation defined, the overall belief is given by:

$$x * y = \frac{x + y}{1 + x \cdot y} \tag{2.3}$$

2.5 The Challenge of Evaluating Anti-Malware Solutions

Building an “anti-malware algorithm” that perfectly determines whether a PE file is malicious or not with 100% accuracy [66] is challenge that is yet to be overcome. To evaluate that the designed methods work, they should be trained and tested by a dataset. In this section, we discuss our dataset preparation and the evaluation metrics used to validate the designed systems.

2.5.1 Proposed solutions evaluation method

In this study, we use the binary classification matrix to test the proposed solutions. Evaluating the proposed algorithms follows the same approach where the achieved detection rates are calculated using the confusion matrix in Table 2.2. The objective is to achieve high true positive detection rates while keeping the false positive detections very low [43]. Therefore, the confusion matrix based metrics are computed for all the proposed algorithms to evaluate the effectiveness of the proposed methods. Based on the metrics achieved, the proposed method is analysed and discussed.

Analysis Results			
Actual Sample State		*Malicious*	*Clean*
	Malicious	True Positive(TP)	False Negative (FN)
	Clean	False Positive (FP)	True Negative (TN)

Table 2.2 Confusion Matrix

The options in the confusion matrix describe the different detection rates in the system and metrics are defined as [67]:

True Positive (TP): files that form part of the malicious dataset that are identified by the method as malicious.

False Negative (FN): files that form part of the malicious dataset that not flagged as malicious by the method.

True Negative (TN): files that form part of the clean files dataset that not flagged as malicious by the method.

False Positive (FP): files that form part of the clean files dataset that are flagged as malicious by the method wrongly.

False Positive Rate (FPR): a measure of the negative samples flagged as positive.

$$\text{FPR} = \frac{FP}{TN + FP} \quad (2.4)$$

Recall/ True Positive Rate: a measure of the actual positive samples detected.

$$\text{Recall} = \frac{TP}{TP + FN} \quad (2.5)$$

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

$$\text{Precision/ PPV} = \frac{TP}{TP + FP} \quad (2.6)$$

Accuracy: a measure of the true detections.

$$\text{Accuracy (ACC)} = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.7)$$

F$_1$ score is defined as the harmonic mean of precision and recall and calculated by:

$$F_1 \quad = \quad \frac{2 \cdot PPV \cdot Recall}{PPV + Recall} \quad (2.8)$$

2.5.2 Dataset preparation

In training and evaluating anti-malware solutions, there is need to test them against malicious files and clean files to achieve the best true positive vs false positive trade-off. The dataset of choice in our experiments a set of malicious files and a control dataset made up of clean files.

Dataset	Total Files
Malicious files	104528
Clean files	1638

Table 2.3 The Experiment Dataset

Malware dataset samples were downloaded from different online repositories, captured using our own honeypots over a period and downloaded from the malware repository at Nettitude Ltd, UK. All the collected malicious files (104528) were run through ClamAv engine version 0.99.2 [68] to ensure that they were all indeed malicious files. The malware family distribution datasets used in the study extracted from the ClamAV scan results are as shown in Table 2.4. The clean files were collect from fresh new installations of Windows XP, Vista, 7, 8 and Win 10 by running a customised batch script to collect all .exe files. The total dataset is shown in Table 2.4 Malware type distribution in the Malware Dataset and this dataset subdivided into various sub datasets per the requirement of the algorithm designed and under test.

Malware Type	Percentage	Malware Type	Percentage
Trojan	66.84%	Dropper	0.65%
Adware	22.30%	Virus	0.29%
Worm	9.03%	Spyware	0.11%
Downloader	0.71%	Exploit	0.08%

Table 2.4 Malware type distribution in the Malware Dataset

2.6 Chapter Summary

This chapter has introduced the world of malware by briefly discussing the various types, the evolution and the number trends of malware seen in the last decade. We then discuss the Portable File format structure, the hashing algorithms, evidence combinational methods, the evaluation methods employed in this study and the dataset preparation process. This chapter has provided the background information to the study carried out in this thesis. The next chapter will review the literature works that were explored during this study.

Present malware detection especially commercial anti-malware solutions struggle to keep up with the evolving landscape of malware [69]. Despite the various breakthroughs in malware detection research, it is always a game of "catch me if you can" with malware writers deploying new techniques to thwart the devised analysis and detection methods [70]. Malware use various techniques in order to survive as long as possible in the wild thus creating more revenue for the writers [71]. The anti-malware community both in the industry and academic continue to build systems that are targeted at fighting malware as a way of ensuring that our cyberspace is more resilient. In this chapter, we review the known malware analysis techniques, the various evasion techniques observed in malware samples and malware detection techniques. Applying selective reference to existing literature on malware detection, we present an extensive review of previous research work using heuristic based detection as defined by the scope of study.

3.1 Malware Analysis Techniques

Malware analysis describes how information about a malware sample is gathered. When a malicious PE sample is discovered in the wild or on a machine, it is usually an executable which has been compiled and therefore presented in machine language [30]. The main goal of malware analysis is to extract as much information from the discovered sample. This information is used to understand the malicious threat associated with the sample in order to contain the damage, reverse it where possible and build a method to guard systems against future infection by the same type of malware [11]. There are two types of analysis; static and dynamic which can be carried out at a basic level or advanced level based on the tools and methods used. These two can be combined in various stages of malware analysis for optimum result. We adopt and define malware analysis based a combination of the 4 stages of malware analysis proposed by Zeltser [72] and common analysis strategies used in software analysis as shown in Figure 3.1. The skill level required in the analysis stages increases with increase on the vertical axis.

This multistage analysis allows for the analyst to stop at any of the four stages as long as they can make a conclusive decision on the file malicious status; whether malicious or benign. After fully analysing the malware, its signature can be modelled so that it can be used in future to detect similar malware in other systems.

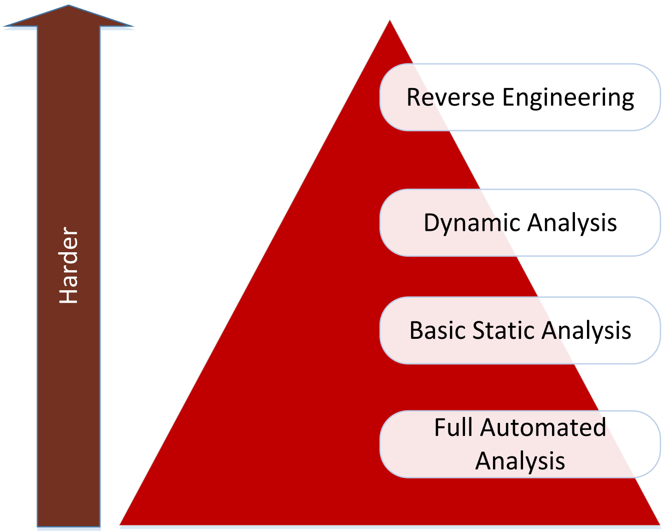
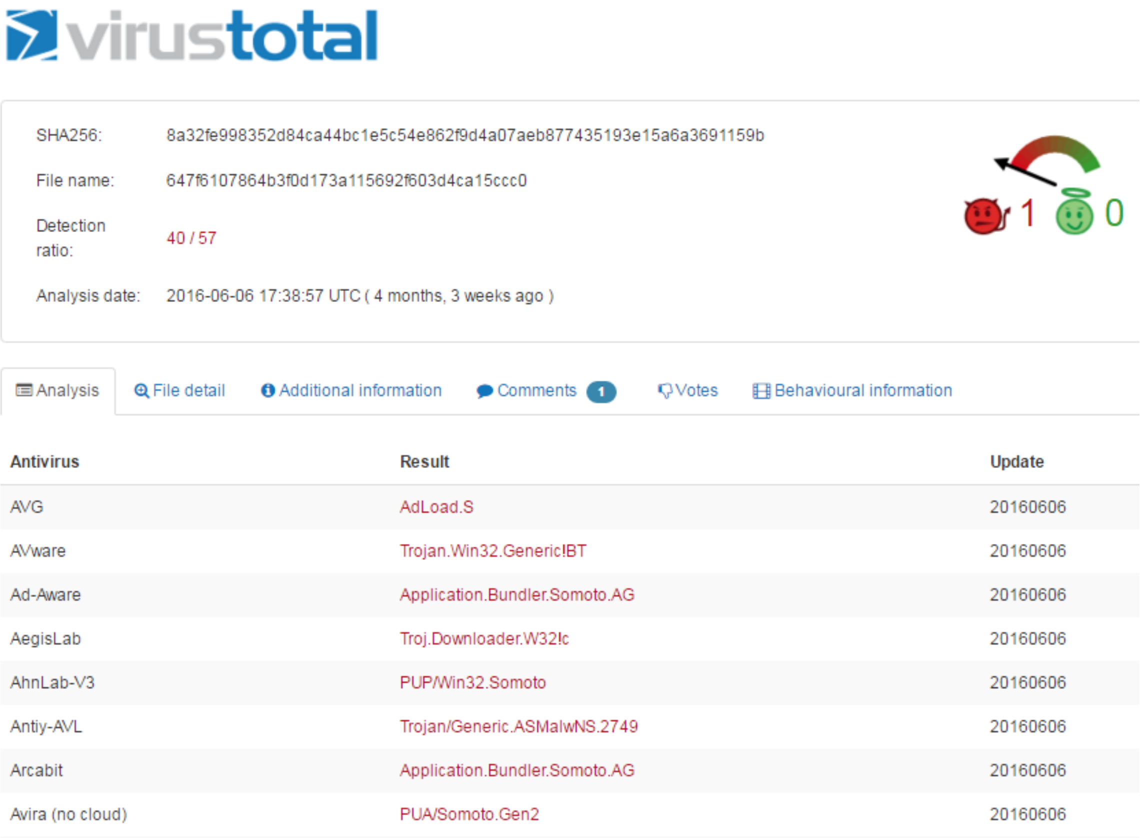


Figure 3.1 Stages of presentday Malware analysis

3.1.1 Fully Automated Analysis

Today, the landscape of malware analysis has evolved due to the development and release of open source, online and or readily available automated malware analysis tools. Automated static analysis tools like Peframe [73], Pyew [74] and Mastiff [75] provide sample analysis reports very easily that can help a malware analyst with the much needed initial results. They are limited by the requirement that the user should learn how they are used and the requirement to step up a simple analysis laboratory. Automated dynamic analysis tools like cuckoo [76], Anubis [77] and ThreatExpert [78] among other provide malware analysis reports on submission of the malware sample without the need for a laboratory. However, some of the reports require specialised skill and knowledge to full comprehend and therefore can be quite limiting to a user. A tool like VirusTotal [79] that uses over 50 antivirus engines to analyse uploaded malware samples provides a free report after analysis. Sometimes, the reports from these automated systems can lead to very conclusive decisions about the malicious status of the file.



SHA256: 8a32fe998352d84ca44bc1e5c54e862f9d4a07aeb877435193e15a6a3691159b

File name: 647f6107864b3f0d173a115692f603d4ca15ccc0

Detection ratio: 40 / 57

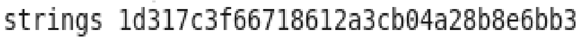
Analysis date: 2016-06-06 17:38:57 UTC (4 months, 3 weeks ago)

Antivirus	Result	Update
AVG	AdLoad.S	20160606
AVware	Trojan.Win32.Generic!BT	20160606
Ad-Aware	Application.Bundler.Somoto.AG	20160606
AegisLab	Troj.Downloader.W32!c	20160606
AhnLab-V3	PUP/Win32.Somoto	20160606
Antiy-AVL	Trojan/Generic.ASMalwNS.2749	20160606
Arcabit	Application.Bundler.Somoto.AG	20160606
Avira (no cloud)	PUA/Somoto.Gen2	20160606

Figure 3.2 Virus Total Report of a malicious sample

3.1.2 Basic Static Malware Analysis

Static Malware analysis is when the malware is examined without executing it [30]. Many automated static analysis tools are available and although we include them in the full automated analysis stage, their main purpose is to perform static analysis. The tools use known syntax or structural properties of the malware code to extract information from the file. Sometimes, an analyst uses the conventional command line based analysis to extract information. The information collected during this type of analysis is very simple and not always sufficient for a conclusive decision on the malicious intent of the file. It is however advised to start with this type as it can provide information to direct the next step in analysing that specific malware sample [11]. One of the basic static malware analysis methods is string analysis in Figure 3.3



```
strings 1d317c3f66718612a3cb04a28b8e6bb3
```

Figure 3.3 Basic String analysis command in a Linux Enviroment

The extracted information is analysed for any interesting information that might lead to understanding its intent like, APIs, URLs, IPs, Passwords and usernames to mention but a few. This information can be used for detection signatures. However, analysis evasion techniques like packing and encryption normally lead to this analysis step providing incorrect information or information that is not useful.

3.1.3 Dynamic Malware Analysis

Dynamic Malware analysis involves analysing the program while it is running on a system [80]–[82]. The malware is run in a “safe” and controlled environment also called ***sandboxing*** to avoid transference of the malware to other systems or networks.

Basic dynamic malware analysis involves observing how a collected sample interacts with system. Normally, virtualisation is used where a snapshot of the original state of the machine is taken, the malware is then introduced into the system and executed. The new state and the original state are compared for changes. The observed changes are then used to remove the infection from infected systems and/or modelling effective signatures. Like basic static malware analysis, it is an important initial step of malware analysis though it does not provide exhaustive information on the malware [16], [22], [30], [46]. This is mainly due to the fact that there is no investigative analysis done during the execution process of the malware to understand how it changes the system and the changes are made.

Advanced dynamic analysis involves using tools to examine the state of the executed malware as it is running. The internal state of the malicious code is examined to obtain more detailed information about the malware. This technique provides information that would normally be impossible to gather when using other techniques [30]. The analysis is always run in a controlled environment to ensure that all the inputs and output of the system are known and their effects can be accounted for. Various tools used at this stage will monitor the APIs, system function calls invoked, files created and/ or deleted, registry changes and the data processed by the program under analysis as it interacts with the system. Analysis of the parameters used during the API and function calls allows for the functions used to be grouped semantically while analysis of the data processed and propagated within the system gives an understanding of the files used and produced by the malware. These lead to the identification of the tasks that the malware undertakes to fulfil its functionality [83]. Advanced dynamic malware analysis is very helpful in identification of malware variants and obfuscated malware techniques as detailed analysis can provide behaviour profiles that can be correlated to existing malware families.

Automated dynamic malware analysis tools exist which give reports that can be used to group the malware according to behavioural properties [76], [83] and have been explored in the fully automated analysis section. However, most automated tools report require skill to understand and sometimes information gained from the manual advanced dynamic analysis and the analysis tools provide a clear understanding of the malware behaviour. The extracted information can then be used to develop counter measures and model the malware signatures to be used for future detection and system recovery.

3.1.4 Reverse Engineering

Reverse engineering, also called advanced static malware analysis involves loading the executable into a disassembler to loop through its execution process and then examining the instructions to understand what the program does [30], [42], [46]. In order to understand the instructions and flow the execution path of the program, one requires an understanding of the operating system that is being used or that was targeted and specialised knowledge of disassembly, the instruction codes/set and architecture of the system [46]. Debugging tools like IDA Pro [84], OllyDbg [85] and WinDbg [86] are few of the tools that are normally used at this stage to describe the execution process path followed by the malware. The Control Flow Graph which describes all the possible paths the program can take is normally extracted from these programs and can be used to detect malware variants. Information gathered during advanced static analysis/ Reverse Engineer can be used in building advanced protection mechanisms [83]. Since Reverse Engineering uses static analysis, analysis evasion techniques like instruction replacement can produce ambiguous results and malware that needs input information that cannot be statistically determined for example time and date can be hard to analyse using this technique [22], [30].

In almost all malware analysis instances, most of the techniques described above are combined to obtain as much information about the malware sample as possible. There is no defined specific order that an analyst follows to obtain the needed information [72]. It is normally up to experience and skill and known data that the decision about the malicious intent of the file is made.

3.2 Malware Evasion of Analysis and Detection Evasion.

The first malware to exhibit detection avoidance was the Brain virus, written in 1986 by the Farooq Alvi brothers [28]. Any attempt to read the virus infected disk section led to the computer displaying clean data [71] instead of the infected lot. This was the start of a fresh breed of malware written with the idea that they could work around the anti-virus systems employed in computer systems. The advancements seen in newly discovered piece of malware all point to the fact that the survival is the number one priority since the longer the malware is undetectable, the more profitable it is to the malware writer [71].

For the piece of malware to survive in any system, it must avoid being detected by the many security measures employed by the systems which include anti-virus software, firewall, and intrusion detection systems among others. Therefore, the evolution of these techniques adds another dimension to the challenge of fighting against malware [28]. Some of the known malware analysis and detection evasion techniques mainly target avoiding being fully analysed or being detected while other serve to evade both analysis and detection.

Originally, anti-reverse engineering was used in legitimate software as a way of protecting intellectual property by software companies and individuals [22], [30]. Malware writers then adopted the idea of anti-analysis, the tools and also created customised versions to evade detection by security solutions [30]. This evolution has led to the development of a new generation of malware and the birth of various variants of the same malware. In this section, we explore some of the known malware analysis and detection evasion techniques which we have classified into 2 categories; Anti-Analysis and Obfuscation techniques. Since the purpose of utilising the described techniques is to produce malware variants that are harder to detect and analyse, we later describe the various types of malware variants known.

3.2.1 Anti-Analysis Techniques:

No malware writer would like to have their program analysed, reverse engineered or detected. Therefore, modern malware use tricks to hinder the analysis process of the malware files [87]. If the anti-analysis tricks are undetected by the analyst, they can lead to dead ends, wrong information or being stuck in an infinite execution loop. In this section, we explore some of the known anti-analysis techniques seen in malware samples.

3.2.1.1 Anti- File Processing

File parsing is one of the methods used in malware analysis and detection where malware detectors use the known file structure to parse the file contents. This method is called file processing [88] and is shown in Figure 3.4. This allows for detection of malware using heuristic based analysis where extracted file features are used to determine if the file is malicious or not.

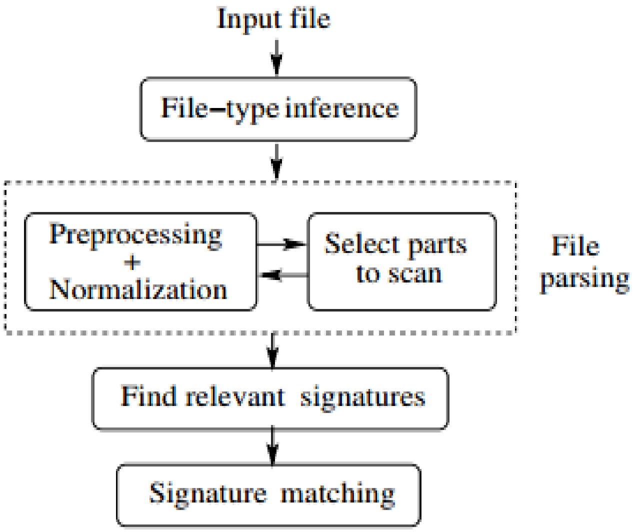


Figure 3.4 File Processing in Anti-virus systems [88]

Malware employ anti-file processing method to thwart scanners that rely on known file formats leading to the file parsers throwing errors instead of extracting the file features [89]. There are two identified exploits in this technique [88]:

- **Chameleon attacks**: This attacks the file interference stage of the file processing where it exploits the heuristic variations. It leads to the file appearing as one file type to the file parser while appearing as a different type to the target operating system.

- **Werewolf attacks**: This creates malformations in the file format that leads to the file parser being unable to extract information. The file under analysis appears to be of a different structure and this is common in executables and application specific formats.

3.2.1.2 Anti- Debugging

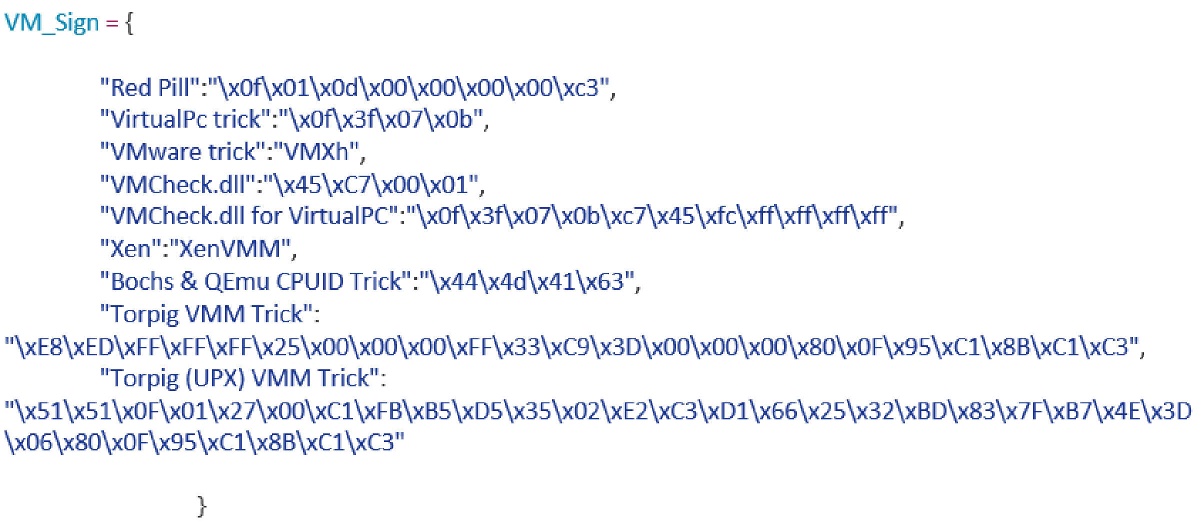
Anti-debugging techniques can be deployed by hooking to various interrupts, using interrupts to generate new decryption keys, through the use of runtime code checksums, checking debugging API routines loaded, checking various registry keys (according to a particular debugger software), using registers and stacks [30], [42], [46].

A debugger is a program that allows one to observe the rendered code as it runs and the most basic features include the ability to set breakpoints and trace through the executable code [46]. Some of the common debuggers are IDAPro [84], OllyDbg [90], Immunity Debugger [91] and WinDbg [86]. These tools are used because compiled programs are too complex for the human eye to be able to trace and predict all the possible execution paths. Malware writers try to frustrate such efforts by writing the malicious program to detect the presence of debuggers and then either give the wrong output or unexpected events [92].

Some of the APIs used in antidebug are CheckRemoteDebugger, DebugActiveProcess, FindWindow, GetLastError, GetWindowThreadProcessId, IsDebugged, IsDebuggerPresent, NtCreateThreadEx, NtGlobalFlags, NtSetInformationThread, OutputDebugString, pbIsPresent, Process32First, Process32Next, TerminateProcess, ThreadHideFromDebugged, UnhandledExceptionFilter and ZwQueryInformation [22], [28], [42], [46], [73], [87]

3.2.1.3 Anti- Virtual Machine

Running malware in a virtual machine is common and considered a safe method used to analyse the behaviour of the malware as it theoretically infects the virtual machine and never the host [42].



```
VM_Sign = {

            "Red Pill":"\x0f\x01\x0d\x00\x00\x00\x00\xc3",
            "VirtualPc trick":"\x0f\x3f\x07\x0b",
            "VMware trick":"VMXh",
            "VMCheck.dll":"\x45\xC7\x00\x01",
            "VMCheck.dll for VirtualPC":"\x0f\x3f\x07\x0b\xc7\x45\xfc\xff\xff\xff\xff",
            "Xen":"XenVMM",
            "Bochs & QEmu CPUID Trick":"\x44\x4d\x41\x63",
            "Torpig VMM Trick":
"\xE8\xED\xFF\xFF\xFF\x25\x00\x00\x00\xFF\x33\xC9\x3D\x00\x00\x00\x80\x0F\x95\xC1\x8B\xC1\xC3",
            "Torpig (UPX) VMM Trick":
"\x51\x51\x0F\x01\x27\x00\xC1\xFB\xB5\xD5\x35\x02\xE2\xC3\xD1\x66\x25\x32\xBD\x83\x7F\xB7\x4E\x3D
\x06\x80\x0F\x95\xC1\x8B\xC1\xC3"

                  }
```

Figure 3.5 Some of the Anti- Vm tricks seen in Peframe [73]

Virtualisation or malware sandboxing also enables faster analysis times than installing new hosts every time one needs to examine a new malware sample. To thwart the analysis of samples inside a virtual machine malware include anti-VM protection or they simply exit when malware is run in an isolated environment. Anti-VM techniques can be deployed by detecting whether they are running in a virtual machine or not. Some of the known anti- VM techniques identified by Peframe analysis tool is shown in Figure 3.5. This is achieved either by looking at VME artefacts in processes, file system, registry and memory or by looking for VME-specific virtual hardware, processor instructions and capabilities [22].

3.2.2 Obfuscation Techniques

Sometimes, once a program (P) is written, the program lines can be re-ordered or lines can be replaced without affecting the intended behaviour resulting into an equivalent but transformed program (P') [30] as shown in Figure 3.6. This transformation is called obfuscation. Obfuscated malware performs the same function as the original malware only that the signature is changed due to the applied changes without affecting the semantic functions.

P
Obfuscation Technique
P'

Figure 3.6 Program Obfuscation

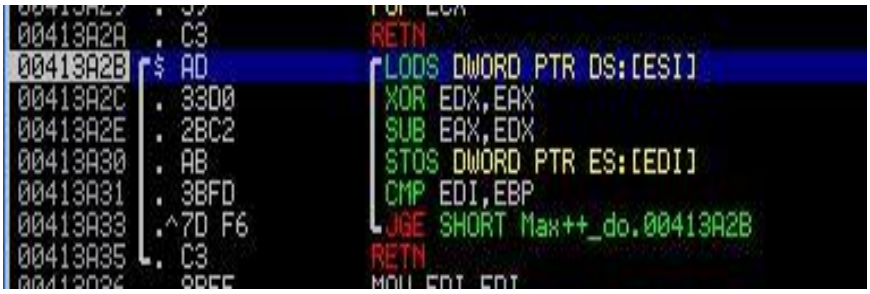
Obfuscation techniques server as anti-disassembly techniques since they try to thwart the reverse engineering process once the sample is loaded into a debugger [46]. These are some of the techniques identified under malware obfuscation.

3.2.2.1 Binary Obfuscation

Binary obfuscation techniques are methods of detection avoidance that are applied after the program has been compiled [93]. They enable the malware program to be packed or encrypted so that the malicious code cannot be accessed until it is in the system memory.

3.2.2.1.1 Encryption

Malware encrypt the original code or blocks of the original code transforming the code into blocks of code that do not make sense to the human eye. Malware writers are known to use existing encryption techniques for example the bitwise XOR base operator was used in ZeroAccess as shown in Figure 3.7. Since more encryption techniques like the base64 encoding and ROT operators can be decrypted once the pattern is realised, malware writers complicate their patterns by customising their patterns for example creating their own alphabets [94].



```
00413A2A  . C3          RETN
00413A2B  $ AD          LODS DWORD PTR DS:[ESI]
00413A2C  . 33D0        XOR EDX,EAX
00413A2E  . 2BC2        SUB EAX,EDX
00413A30  . AB          STOS DWORD PTR ES:[EDI]
00413A31  . 3BFD        CMP EDI,EBP
00413A33  .^7D F6       JGE SHORT Max++_do.00413A2B
00413A35  . C3          RETN
```

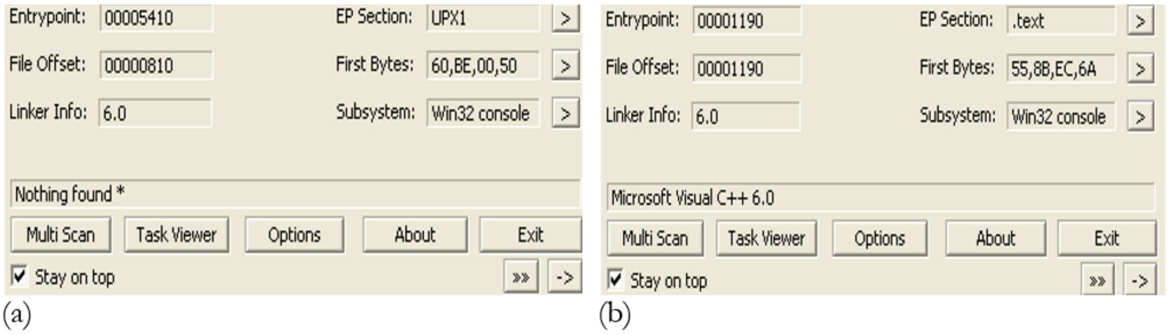
Figure 3.7 Extract from Zero Access Self decoding Subroutine

3.2.2.1.2 Entry point obfuscation

The Entry point obfuscation (EPO) techniques is implemented by the malware scanning the host file and then changing the pattern of certain API especially the ExitProcess to point to the beginning of the malicious execution code[95], [96]. This way, the program relies on a function call to get executed and not on the operating system loader. The malicious code is embedded safely inside the host file at a random location which this then called by the function call. Once it is executed, it passes the call routine control back to the actual subroutine. Examples of these types of malware are Rainsong, Zmit and Zhengxi [82].

3.2.2.1.3 Packing

The most common and default feature of packers is compressing the file into a smaller size. The packer works as an envelope that hides the program from any outside sources until it is run in the system. It also pre-pends the unpacker to the newly formed program which is the encrypted original program stored as data in the new executable [30]. This was initially developed by commercial software companies when disk size was of prime importance [52]. However, malware writers have adopted it and misused it. An example of a basic packer software is UPX as shown in Figure 3.8. In the analysis of malware especially when trying to reverse engineer the program, a packed program is easy to detect using analysis tools. With the introduction of custom built packers, reverse engineers must manually unpack the code using debuggers because of lack of the original packer information.



Entrypoint: 00005410 EP Section: UPX1
File Offset: 00000810 First Bytes: 60,BE,00,50
Linker Info: 6.0 Subsystem: Win32 console
Nothing found *
☑ Stay on top

Entrypoint: 00001190 EP Section: .text
File Offset: 00001190 First Bytes: 55,8B,EC,6A
Linker Info: 6.0 Subsystem: Win32 console
Microsoft Visual C++ 6.0
☑ Stay on top

Figure 3.8 Malware sample details packed with UPX (a) and after unpacking (b)

3.2.2.1.4 Stealth

A stealth malware is a type of malware that tries to remain undiscovered by hiding the infection events [28], instead of trying to obfuscate its code. It achieves this by restoring certain original properties of the host file for example, timestamps. It also intercepts system calls to hide any other resulting changes like the increase in the size of the host file. Other techniques used are creating alternate data streams (NTFS) [94] for infected files with malware in the alternate data streams. Alternate Data Streams is a feature developed by Microsoft NTFS to enable Windows support Macintosh Hierarchical File System (HFS). Files use multiple forks which allows the program to store the code and icons separately. Although the feature is intended for internal use, it can be used to hide files and therefore malware can be attached to a legitimate file and go undetected for a while.

3.2.2.1.5 Condition based Execution

Some Malware require specific inputs and unless these inputs are met, it does not execute. Some of the conditions are specific days or presence of specific features on the system. Unless these inputs are known previously and emulated, analysis of this malware normally leads to incorrect information [42] or the inability to perform a full analysis.

3.2.2.2 Code Obfuscation

Code Obfuscation techniques are applied to the program during the writing of the code itself [93]. These enable the code to have a confusing structure to the eye and the antivirus systems but the code will still perform the same function as an originally simple structured code. In other words, these methods are employed to change the syntax of the program without changing the semantics of the program. Some of the known methods are[93], [97]–[99]:

- Dead-Code Insertion; where "*trash-code"* lines are added in the program without changing the behaviour of the code.
- Code Transposition where the code is shuffled so that the binary order is different from the execution order.
- Register Reassignment which uses the replacement of registers with others within a specific code live range.
- Instruction Substitution where equivalent instructions are used to replace existing ones.
- Code Integration where the virus code is interweaved into the code of its target program.

3.2.3 Types of Obfuscated Malware

Obfuscated malware is defined by the structural and syntactic similarities and differences to already existing malware. These are grouped into 4 groups; packed, oligomorphic, metamorphic and Polymorphic malware depending on the detection evasion technique used.

3.2.3.1 Packed Malware

Most malware writers apply packers or even multiple packers to produce different variants of the same malware code. Perdisci, et al [52] states that more than 80%

of the new malware discovered are actually packed versions of already existing malware. Packers compress the file into a smaller size and sometimes encryption is applied to the compressed version of the file to make the unpacking process more difficult. "*The packer program automatically transforms an executable into a syntactically different but semantically equivalent representation*" [83] as seen in Figure 3.9. Some packers are custom built by malware writers and these can be used to actually detect that the file is malicious without the need for further analysis while commercial packers that are readily available online are seen in many malware variants [94].

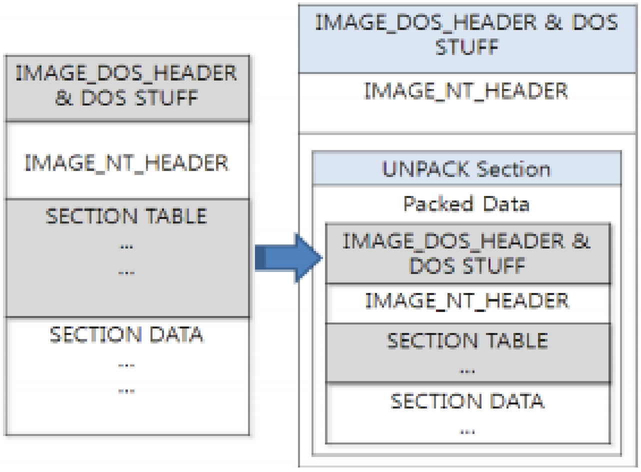


Figure 3.9 Structure of a Packed PE File[100]

3.2.3.2 Oligomorphic Malware

These malware also sometimes called ‘Semipolymorphic’ [31] employ multiple decryption routines which are chosen randomly at infection as a way of avoiding signature based detection. The Whale virus was the first malware to use this technique and it carried a few dozens of different descriptors and picked one randomly [82]. More malware were subsequently seen employing the same methods however the decryption engines grew in number. This led to the birth of a new type of malware; polymorphic malware.

3.2.3.3 Polymorphic Malware.

Polymorphic malware like oligomorphic malware, use decryption routines to change the look of the execution codes for every infection [101]. They have a wide range of decryption engines since they tend to use mutation engines. The mutation engines perform all the logic computations in rearranging the code to prevent detection by signature matching. The decryptor is run first once the malware is copied to the machine and it enables the execution of the malware. When the malware replicates itself, it encrypts the new malware with a different key and encloses the new decryption routine in the new code. It can however only generate up to a few hundred decryptors so it can be detected [98].

3.2.3.4 Metamorphic Malware

In this type of malware, the malicious code body is changed by using a combination of various obfuscation techniques. By using dead-code insertion, register reassignment and code transposition, the body of the code is changed into a new generation but it works the same way [98], [99]. This way, every generated variation of the malware looks different and therefore signature generation and signature based detection is very hard. Unlike most polymorphic malware which decrypt to a single constant code body in memory, metamorphic can have varying codes which makes their detection in memory rely on algorithmic scanning [28]. Metamorphic malware can also insert and interweave their code into the host program which makes the malware harder to detect [99].

3.3 Existing Malware Detection Techniques and Solutions

Malware detection is the process of identifying malicious code from benign code so that the system can be protected or recovered from any effects the specific malware. Malware detection requires the knowledge of the malware sample and therefore heavily relies on the analysis process retrieving correct and sufficient information. Detection techniques utilise a combination of the analysis techniques for a more conclusive answer about the malicious code [28], [30], [80], [82].

3.3.1 Integrity Checker

Compromising a computing system or network requires that some changes be made within the target environment. Integrity checkers are used in intrusion detection on the premise that a file which exists within the uncompromised operating environment is used as a measure to counter any future changes [28]. A hashing function like the MD5sum. SHA1or SHA256 is used to calculate the digest of the program or file and stored in the database [102]. When necessary, the digests of the program/ file are re-calculated and then compared against the originally calculated hash to check if the file has been modified. The challenges that this method presents are:

- The system initially used to calculate the stored hashes must be deemed clean and this is sometimes hard to guarantee.
- System updates and patches that are very prevalent in computer systems do modify system files and programs which means that the database of hashes needs to be updated for every update or there will very high false positives affecting the detection method.
- There is need to ensure that the reference database of hashes is stored offline and safely otherwise it presents a single point of failure in the detection mechanism. If the database is compromised, then all the resultant comparison checks are compromised as well. Hackers gaining access to the database implies that the database can be changed to include the hashes of the malicious programs.

Integrity checking is still considered quite important in malware detection and detection of any system modifications performed by a malicious program. It is however more of an incident recovery process method than malware infection prevention method [82].

3.3.2 Signature Based Detection

Signature based detection uses specific byte code sequences that are identified to be unique to a sample of malware in a specific family or variant and using them to detect the presence of similar coded files in the system [28]. This method is the most significant methods used by commercial anti-viruses. The unique byte of code sequence are saved in the anti-virus database as signatures and are developed by a group of malware experts after detailed analysis a significant number of malware [25]. Any file being scanned by the antivirus that is found to contain the signature of the unique byte code sequence is deemed to be malicious. This implies that a database of signatures must be maintained by the antivirus and updated every time new signatures are generated in order detect new malware. This creates one of the major challenges faced by system users as updating these signatures requires access to networking resources that might not be readily available all the time. Lack of optimisation of the signatures can also lead to the method being plagued by a high rate of false positives [82].

Research has proved that metamorphic can evade this type of detection using obfuscation techniques which implies that this method is susceptible to high false negative in present day environments [103]. However, the study in [104] shows that the mutation engine signatures can be used to detect metamorphic malware as a way of extending the signature based detection by successfully tracking variants of W32.Evol. Chouchane and Lakhotia argue that given known parameters of a mutation engine, the proposed technique can detect malware that has been generated using the engine and suggest that the method can be further extended to detect malware produced by specific toolkits. This work however fails to address the content growth of the signature database which implies more storage requirements and therefore resource utilisation. This suggested approach can also lead to false negatives if a syntactic change is applied to the mutation engine.

3.3.3 Semantic Based Detection.

Semantics-based malware detection is a detection approach which looks to identify the malware by deducing the logic of the code and matching it to already known malicious logic patterns. It follows the semantics of the code instructions within the file instead of looking at the syntactic properties unlike signature based detection [105]. This allows for the semantic based detection approaches to overcome obfuscation and can improve the detection of unknown malware since logic pattern is used. An overview of semantic based detection approaches is provided in [106].

3.3.4 Behavioural Based Detection

Behaviour-based detection techniques focus on using specific system/ application behaviour and activities that are observed during dynamic analysis of the sample to form patterns that can be used to identify software that invoke similar patterns. Although these techniques are largely immune to obfuscation, their applicability is limited by their performance as dynamic analysis requires time [107]. A survey of behavioural based malware detection is provided in [108]. Filiol present that one of the major challenges that is faced by behavioural based approaches is the difficulty associated with establishing the rule set of what is considered normal behaviour that a software invokes within a controlled environment. This implies that without the inference point of the expected normal behaviour, determining the unsafe activities and behaviour within the environment becomes an evolving challenge [108].

3.3.5 Heuristic Based Detection

Heuristic in computing is defined as "*Proceeding to a solution by trial and error or by rules that are only loosely defined"* [109]. The main idea behind heuristic based detection is that there is no need to know so much about the inner structure or logic of the program being scanned and the main aim to reach as close to a conclusive decision as possible using the best optimal path [110]. Therefore, heuristic based detection approaches use algorithms and or rules that scan for known patterns to identify malicious programs within an environment. The first heuristic approaches are known to have been built in1989 to detect DOS viruses [17]. Most antivirus programs today use a combination of heuristic engines and signature based scanners. Heuristic based detection have the following advantages [17], [28], [82]:

- They are fast because they are static analysis based which means that they scan the file without executing it unless emulation based analysis is included in the design of the engine.
- The heuristic rules and algorithms can be changed, customised and optimised based on the files being scanned, the operating environment and any new rules/ indicators identified by malware analysts.
- Since heuristic scanning is not predefined and evolves based on the requirements of the operating environment, targeted evasion is quite had to achieve.

Heuristic based approaches also have some limitations which are:

- They are plagued by high rates of false positives and false negatives if they are not properly fine-tuned, validated and optimised.
- Has access to limited information especially if relying on static based analysis and therefore there is need to use a combination of different features or applying integrated approaches to scanning for better results.

Heuristic based malware detection is achieved in 2 phases [82]:

Phase 1: Train the malware detector using a dataset so that important characteristics can be captured. In this training phase, sometimes, there is always need to use both the malicious dataset and the clean/benign dataset as a control measure. In other cases, the engine is trained with normal activity so that the engine can detect any abnormal activity.

Phase 2: Test and validate the heuristic engine to ensure that it is making decisions that are in line with what is expected.

Recent research in the use of datamining for malware detection [111] are considered heuristic based detection approaches. Since the study in this thesis uses heuristic detection based techniques, the next section explores related research works that were reviewed as foundation areas to this study.

3.4 Recent Heuristic Based Malware Detection Research Solutions

Our work examines how PE anomalies can be scored for more efficient malware detection and extends the functionality of file hash similarity functions. Therefore, using selective literature review, we focus on studies around; file hash similarity functions are used in malware analysis and detection, malware detection using PE anomalies and how multiple feature based decision making has been utilised to improve malware detection rates in various instances. We also explore some of the malware scoring methods that have been proposed.

Hashing functions are used for malware clustering and triaging in many malware analysis scenarios. Since there is no predefined meaning to the file similarity achieved between two malicious files, the interpretation of the hashing result is better left to the deduction skills of the analyst. Many studies have been conducted on the clustering sensitivity of hashing methods. A section of a technical report by DigitalNinjas [112] provides an initial study of hashing similarity. The work shows promise for the detection of the different families of malware that provides a level of confidence of 67% using only the Ssdeep hash technique. Although this work serves as a baseline, there is no comparison study against clean files as a control. This work is further extended by French and Casey [113] who carry out a study on the different fuzzy hashing methods available, Ssdeep and Sdhash. French and Casey provide insight into how and why fuzzy hashing works, validate the use of hashing in clustering families of malware and call for a cost benefit analysis of the hashing methods. Choi [114] proposes a methodology to use Imphash for malware clustering which was first introduced by Mandiant in [57]. New fuzzy hashing methods are introduced in [50] as new clustering algorithms that provide higher sensitivity rates for the clusters. Although the results show higher sensitivity matching, hashing is still restricted to clustering of the malware samples. Furthermore, [50], [53], [57], [112]–[115], [58] all use one hash to study each clustering experiment without looking at the potential increase in detection rates one could achieve combining different hashing methods.

Combining file features and file relations in order to improve malware detection results is introduced in [20] and thereafter the work develops a file verdict system called “Valkyrie”. The authors build a semi-parametric classifier model to perform the combination and test the model against a dataset of 39,138 malware samples. The work states that the system has been incorporated into the scanning tool of Comodo Anti-Malware software. Although this work shows high detection rates, the use of file relations introduces an interesting but ever evolving concept in malware detection. The comparison of their model against Kaspersky Anti- virus and McAfee Virus scan along with tools show their model outperforming the tools for both the detection and time efficiency. This is questionable as the latest tests on AV-test institute show that Kaspersky antivirus outperforms Comodo antivirus [10].

Kolter & Maloof in [18] examine the results of various classifiers on malware detection through a simple heuristic based technique of text classification, known as n-grams. The proposed approach tests the techniques including Naïve Bayes, decision trees, support vector machines and boosted variants. This approach not only uses multiple methodologies to train and test the algorithm, it also shows good detection rates, between 95-98%. However, the experiment used a limited dataset of 1971 malware which is a very small dataset compared to the malware samples collected nowadays. This method requires a lot of computational overhead which with our proposed method, we work to overcome by using hashes and implementing it using light weight methods.

The MaTR [15] approach combines static heuristic file features and decision tree machine learning algorithm to propose a method of improved malware detection. The work initially recreates the experiment environment of the approach in

[18], highlights it’s weaknesses which are then used to build a better and efficient detection algorithm. Experimentation using a dataset of 31193 clean malicious and 25195 clean files leads to 99.9 accuracy in the detection rates. Although this approach shows near ideal malware detection results, there is a possibility that malware obfuscation could introduce a limitation in the method as this is not addressed in the work.

Xinjian et al, [19] propose combining both statically and dynamic features to improve malware classification. The method proposes using classifiers and adopting the prediction if and only if the output is the same. They test the proposed method on 282 samples which achieves high detection rates. The small size of the test dataset creates a challenge for the method in addition to the method not being tested against other known techniques.

Combining features using evidence combination methods was introduced in the detection of android malware in [116] where the authors propose treating each feature statically extracted from an android applications as an information source and using Dempster-Shafer theory of evidence combination to combine the information sources. Using a dataset of 1580 malware samples the method achieves a detection accuracy of 97% and a false positive rate of 1.9%. The results show that combining different features does indeed improve malware detection rates and our work follows the same approach. We apply and extend this method to PE files and use basic static based hashes as representatives of heuristic features which reduces the used resources, cost and effort used in the proposed method in [116].

Studies towards attaching a malicious score to a file as a method of malware detection has been an evolving topic in security research. Taking the approach of the CVSS (Common Vulnerability Scoring System), MAEC project is introducing the concept of malware threat scoring system which uses predefined categories to attach a threat score to a file [117]. RSA, the security division of EMC has introduced the RSA Security Analytics Malware Analysis scoring categories [118]. The work is presented as a module but does not present a working prototype or a method on how to design the scoring mechanism. Both the MAEC and RSA categories look at Static analysis as a required category. Kumar et al [119] propose attaching a heuristic score to a PE file based on the features extracted. Using 10 static features and a dataset of 1360 malware and 1230 clean files, the proposed model achieves an accuracy detection rate of 85%. Although the detection rates are not high, the scoring approach proposes a method of allowing a malware analyst in classifying malware based on urgency. The initial results achieved by Kumar et al provide a foundation in file malicious scoring in a quest to build a more resilient cyber space. And the work presented in this thesis further explores and expands this initiative achieving an easier way of scoring and providing more detailed approaches on how to achieve the file malicious score.

3.5 Chapter summary

In chapter, we discuss malware analysis techniques, the evasion techniques malware use to thwart analysis and detection and the known malware detection techniques. We then review relevant literature of recent research approaches to malware analysis and detection. Using this foundational information as reflection point, the next chapter evaluates three known popular automated static analysis tools whose functionalities are analysed. This allows for development of the framework approaches that are the contributions of this thesis.

CHAPTER 4. EVALUATION OF AUTOMATED STATIC ANALYSIS TOOLS FOR MALWARE DETECTION IN PORTABLE EXECUTABLE FILES

4.1 Introduction:

Malware detection requires both static and dynamic analysis. However, static analysis is always considered the best first step when dealing with malicious files because it allows for malware to be analysed without the need to execute it [120]. The work by Egele et al [83] covers most of the present day dynamic analysis techniques and discusses the different analysis programs and tools that use these techniques. Although Ligh et al [42] provides insight into many static analysis tools, there is still a need for a detailed evaluation of some of the more recent prominent tools available today. The recent release of numerous automated static analysis tools which have given the cyber security community a much needed boast towards efficient static malware analysis requires additional exploring. This is especially true given that the rate at which malware samples are released is much higher than the rate at which malware analysts can fully investigate. The malware statistics from AV- test institute show that the number of malware samples collected each year keeps increasing as cybercrime takes advantage of increased dependency on technology due to the big drive towards the Internet of Things. There is a need for faster detection to reduce the impact of malware if people are to benefit from the evolution of the Internet of Things. This chapter evaluates three prominent, open source malware automated static analysis tools focusing mainly on the analysis of Portable Executable files. It provides an overview of the automated static analysis tools in the first section, the test environment in the second section and the 9 test scenarios with emphasis on PE features to perform a comparison of the tools and extract information from the samples. It also presents the findings of the scenarios and the tool feature comparisons and present a summary of the fingerprints found which can be used as indicators of compromise in executable files.

4.2 Overview of automated Static Analysis tools.

This section provides summarised overviews of the automated static analysis tools that are being evaluated in this study.

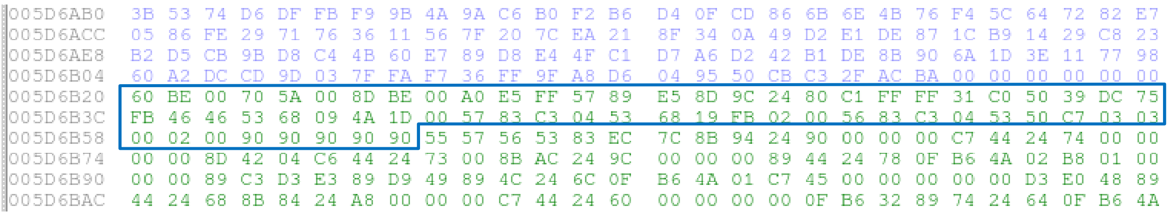
4.2.1 Peframe

Peframe is an open source, command line based static malware analysis tool written by Gianni Amato that extracts information from Portable Executable files [73]. It is written in python and uses the pefile module written by Ero Carrera [73] and the Anti-Virtual Machine Signatures written by Joxean Koret [74]. In its folder called …\peframe\signatures, there are 3 lists of signatures that are used to identify different characteristics present in the application being analysed.

alerts.txt: a list of APIs that peframe flags as suspicious in an application

antidg.txt: A list of APIs that peframe uses to identify the presence of debugger detection in the application

userdb.txt: A list of data block signatures used to flag the presence of packers detected, anti- VM and anti-debugging tricks. Peframe searches through the hex-dump of the file for the specified sequence of data block and if it is matched, the packer shown in the square blanket is identified as the packer used by the file under analysis.

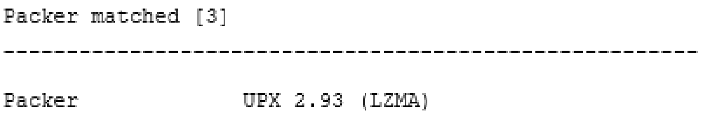


```
005D6AB0  3B 53 74 D6 DF FB F9 9B 4A 9A C6 B0 F2 B6  D4 0F CD 86 6B 6E 4B 76 F4 5C 64 72 82 E7
005D6ACC  05 86 FE 29 71 76 36 11 56 7F 20 7C EA 21  8F 34 0A 49 D2 E1 DE 87 1C B9 14 29 C8 23
005D6AE8  B2 D5 CB 9B D8 C4 4B 60 E7 89 D8 E4 4F C1  D7 A6 D2 42 B1 DE 8B 90 6A 1D 3E 11 77 98
005D6B04  60 A2 DC CD 9D 03 7F FA F7 36 FF 9F A8 D6  04 95 50 CB C3 2F AC BA 00 00 00 00 00 00
005D6B20  60 BE 00 70 5A 00 8D BE 00 A0 E5 FF 57 89  E5 8D 9C 24 80 C1 FF FF 31 C0 50 39 DC 75
005D6B3C  FB 46 46 53 68 09 4A 1D 00 57 83 C3 04 53  68 19 FB 02 00 56 83 C3 04 53 50 C7 03 03
005D6B58  00 02 00 90 90 90 90 90 55 57 56 53 83 EC  7C 8B 94 24 90 00 00 00 C7 44 24 74 00 00
005D6B74  00 00 8D 42 04 C6 44 24 73 00 8B AC 24 9C  00 00 00 89 44 24 78 0F B6 4A 02 B8 01 00
005D6B90  00 00 89 C3 D3 E3 89 D9 49 89 4C 24 6C 0F  B6 4A 01 C7 45 00 00 00 00 00 D3 E0 48 89
005D6BAC  44 24 68 8B 84 24 A8 00 00 00 C7 44 24 60  00 00 00 00 0F B6 32 89 74 24 64 0F B6 4A
```

Figure 4.1 Hex-Dump of entrypoint of file md5-a3c5e50c55c901767b0c3b7749a48c9b

The file is analysed as a hex-dump and the signature highlighted in

Figure 4.1 is identified to be identical to the signature in the userdb.txt. In the results, it returns that it has identified the presence of the packer UPX 2.93 as shown in the extract of the report below:



```
Packer matched [3]
--------------------------------------------------------

Packer            UPX 2.93 (LZMA)
```

Figure 4.2 Peframe Report Extract of the file

Although the results are described in the command line, it provides the option of printing the results to a text file which can be manipulated further for analysis.

4.2.2 Pyew

Pyew is a static analysis framework written by Joxean Koret that performs file and code analysis on PE, PDF, ELF and OLE2 file types [74]. It is mainly command line based although the tool *bokken* which has a GUI can use it as a backend to provide better user interaction. Pyew uses the standard *Pefile.py* module to read the file contents of the PE File. This enables it to read PE structure and display the contents depending on the command used. It also provides debugger properties without the need to install a debugger. The scripts of interest for a malware analyst are found in the folders:

…\pyew\plugins

Vmdetect.py contains some signatures of known anti-VM tricks which are used to detect the presence of these tricks in the file.

Virustotal.py is the script that searches the virustotal website for a report on the file being analysed using the MD5 and prints out the report it retrieves.

UserDB.txt: It is a copy of the PEiD packer detection signatures used by *Packer.py* to detect the presence of a packer.

…\pyew

gcluster.py; A new module that is not seen in any of the other tools which uses the graph vertices to compute the similarity between the call graphs.

graph.py: The script that retrieves the call graph of the file and an example of the extract can be seen in the Figure 4.3

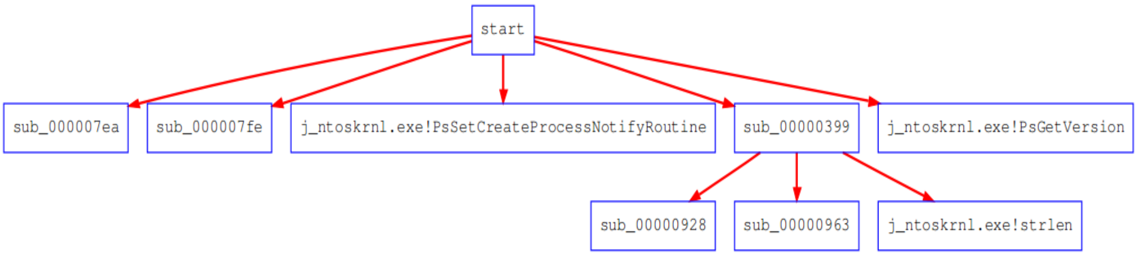


Figure 4.3 Call Graph of a sample malware

Using Pyew analysis is limited as it requires additional scripting to allow for automated multi analysis and requires for changes to be applied to the modules to get output logs that can be further manipulated as the information saved in the SQLite database is very limited and not of great value during the analysis.

4.2.3 Mastiff

Mastiff is a framework developed in python that performs static analysis of files and is command line based. It is designed to extract the characteristics of the file by automatically identifying the type of file being analysed and using the right techniques to analyse it. Mastiff relies on plugins which makes it easy to be extended for further use and can be used as a building block in the design of other frameworks.

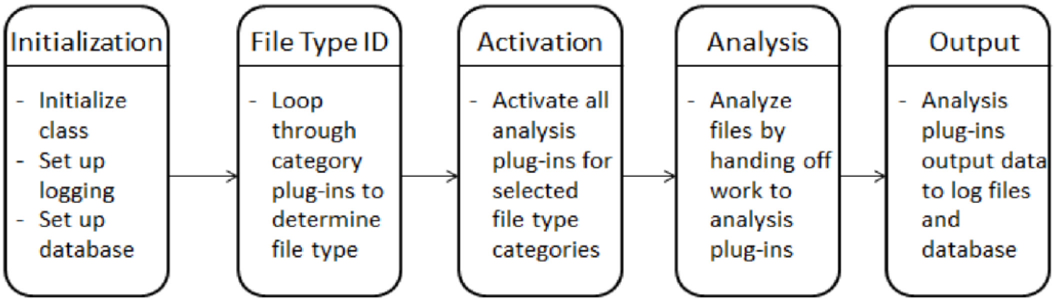

Initialization
- Initialize class
- Set up logging
- Set up database
File Type ID
- Loop through category plug-ins to determine file type
Activation
- Activate all analysis plug-ins for selected file type categories
Analysis
- Analyze files by handing off work to analysis plug-ins
Output
- Analysis plug-ins output data to log files and database

Figure 4.4 Mastiff Work Flow [75]

Mastiff functions and workflow as seen in Figure 4.4 are further documented by Hudak [75] with details on how the different modules work together to provide an in-depth analysis of the file. Mastiff supports the analysis of different file formats but the plugins of interest are found in the folders ***…\mastiff\plugins\analysis\EXE*** and ***…\mastiff\plugins\analysis\GEN***:

EXE-peinfo.py which is the script that extracts and dumps information the PE header and the structure of the executable.

EXE-resources.py extracts the information on the PE Resources directory

EXE-sig.py extracts the PE digital Signatures in the file.

EXE-Singlestring.py extracts single byte strings found in the file.

GEN-fuzzy.py which extracts the fuzzy hash of the file and files in the sample that match.

GEN-String.py extracts and decodes any of the embedded strings found in the code.

Mastiff provides an option of using the Virus Total API so that the files uploaded can be analysed by the virus total website and the report generated downloaded.

4.3 Test Environment

This section discusses the steps taken to setup the analysis environment to ensure validity and reliability of the results. It also describes the evaluation and analysis approach used in the study. Table 4.1 provides the details of the different machine specification, environment, files and tools used.

Tool	**Specifications/ Details**
Computer	Dell 745, CPU – Intel duo core @ 2.13GHz, RAM 2GB. Hard Disk – 150GB
Host Machine OS	Windows 7 Professional N Service Pack 1 (64 bit)
Virtual Machine Manager	Oracle VM VirtualBox Manager Version 4.3.20 r96997
Virtual Appliance	Honey Drive 3 – Royal Jelly installed with 80 GB vmdk space
Tools of analysis	Pyew version 3X Mastiff version 2.0 Peframe as seen on the GitHub accessed April 2015.
Dataset	2620 samples of malicious PE files were downloaded from http://www.nothink.org/honeypots/malware-archives/http://www.nothink.org/honeypots/malware-archives/
Data management tools	SQLite Studio version 3.0.6. Python IDLE version 2.7.9

Table 4.1 Experiment Setup Specifications for the evaluation of automated static analysis tools

4.3.1 Comparison, Analysis and Evaluation Approach

The dataset was downloaded and analysed to pick only PE type files. Since the dataset included many malware samples which could only be analysed one at a time by any of the tools it was necessary to write scripts to automate this process. With all the malware files saved into a single folder, each file as analyse and the results were dumped as a text file into another folder. By modifying the script written by tekdefense in [121] different scripts were written to enable auto multi-file analysis for Mastiff, Pyew and Peframe. The script used for Pyew also utilised its call graph module and call graph clustering module to obtain results necessary for this evaluation. Data analysis was performed on the information retrieved to obtain meaningful results. Figure 4.5 shows the study approach taken during this study.

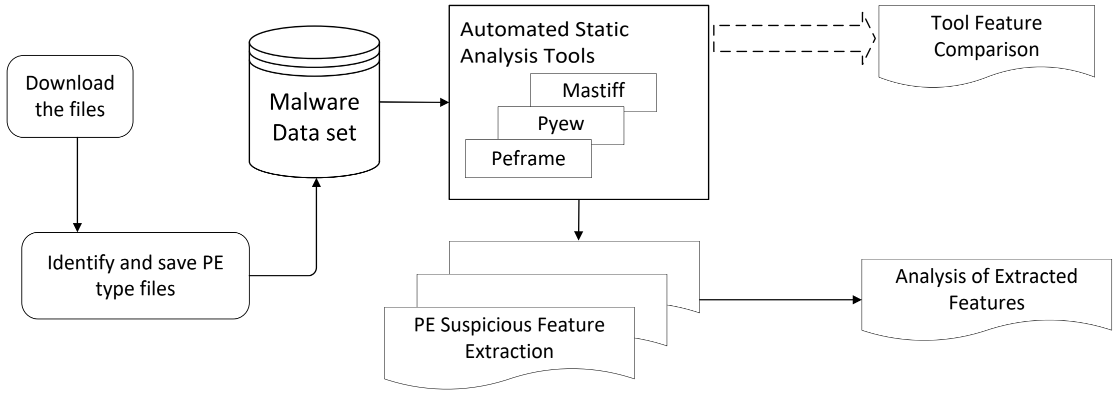

Download the files
Identify and save PE type files
Malware Data set
Automated Static Analysis Tools
Mastiff
Pyew
Peframe
Tool Feature Comparison
PE Suspicious Feature Extraction
Analysis of Extracted Features

Figure 4.5 Pictorial representation of evaluation study Approach

4.4 Test Scenarios

The test scenarios were formulated based on the information required to be extracted from PE files to detect any suspicious characteristics and the additional features that the tools provide that add value to the detection process. This section discusses the test scenarios chosen and the reasons to why they were deemed important.

4.4.1 File Identification:

Fields that identify the files such as the Filename, File size, MD5, SHA and SHA256 are important to extract because these can be used to check the integrity of the file downloaded. In addition to being used as identifiers when performing further analysis, they can be used to query existing databases for known malware.

4.4.2 Detection of Obfuscation Techniques

Obfuscation is a major characteristic of many malware as they try to evade detection or slow down analysis. Detection of some of the signatures that show that an application is using one or more of some of the obfuscation techniques may lead to detecting a malicious file. For this test scenario, three common methods applied in the binary obfuscation techniques as seen in malicious files are investigated. With the tricks discussed in chapter 3 known, analysts have developed signatures that detect specific packers, anti- VM and anti-debug obfuscation techniques. So, the results obtained from the tools will be analysed to determine the ability of the tool to detect; packers, Anti-VM and Anti- debug tricks in the file information. The tools are also expected to identify the specific packer type where possible which should assist in the next step where unpacking is required.

4.4.3 Analysis of APIs

The API calls extracted from a PE file can highlight the expected behaviour and characteristics of the file [46]. Some APIs may indicate that the file employs some obfuscation technique and how the file would interact with the system.

Some of the APIs used for anti-debug detection for example are [122]

> *IsDebuggerPresen*t: This function checks whether the application is being debugged by returning a non-zero.
>
> *NtQueryInformationProcess:* Returns internal Operating System structures related to the process passed. This function is no longer available for newer versions of Windows but it can still be seen in some malware samples
>
> *CheckRemoteDebuggerPresent:* Returns a non-zero value if the process passed to the function is being debugged.
>
> *OutputDebugString:* The function sends the debugger a string to display.

Some other notable characteristics are if the file expects to connect to the internet, how it will execute and/or how it accesses memory. Depending on the APIs clustered, files can be partially grouped into clusters of predicted behaviour so extraction of APIs is a very important feature of an analysis tool. For this field, two sub-definitions are considered; the extracted suspicious APIs per the tool and the general extracted APIs.

4.4.4 PE file feature analysis

Many of the PE file fields have specific standards set by Microsoft [24] so changes in these standards might indicate that the file is suspicious [42], [87], [100]. In this scenario, the test focuses on how much of the PE file field information is extracted by the tools or if used, whether the tool provides suspicious alerts for the fields that have invalid information in the fields or identification of any anomalies in relation to the PE structure formation.

4.4.5 Ssdeep hashing and Malware clustering

Cryptographic hashing computes a hash value on a data block and any changes in the data block produces are different hash value. If one file has the same cryptographic hashing value as a known malware sample, then it is concluded that the file is a copy of the known malware. Since malware variants tend to change some bit values of the original malware code, normal cryptographic hashing like the MD5 fails to detect the similarity of such files. So, to counter this limitation, fuzzy hashing uses a rolling window to produce a continuous stream of hash values. These hash values can be compared to produce a percentage score of similarity between the files compared which enables malware analysts to detect malware variants. Many hashing algorithms; Ssdeep, imphash and others have been investigated in different works [53], [115], [123] to show how they improve the accuracy of detection when used. This scenario looks at analysing what file hashes are computed and how they are used in file clustering.

4.4.6 Call Graph Extraction and Comparison

A call graph is a directional graph with nodes (N) that represent the functions that are interconnected with function calls represented by edges (E (i, j) where i to j define the execution path taken as shown in Figure 4.6. Extraction of a call graph represented by G = (N, E) provides a graphical representation of the execution process of the program.

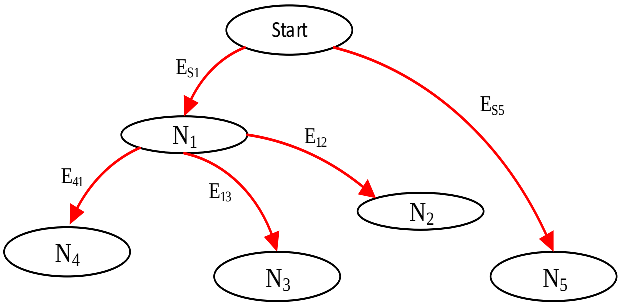


Figure 4.6 Call Graph (G) Structure

Call graphs have been used in different research works to show how they can predict the behaviour of the file [29], [30]. The work carried out by Kinable [124] shows that call graph matching and clustering can be used to detect malware variants. Availability of such information from a tool provides a way to increase the accuracy of the malware detection. The fields analysed in this scenario look at the extraction of code call graphs and comparison capabilities of code call graphs performed by the tool.

4.4.7 String Analysis

Strings obtained from a file during static analysis when not encrypted can provide a lot of information for a malware analyst, like URLs, executable files, registry key paths, command line options, passwords and IP addresses. Similarity in the information across different samples may be used to cluster them. However, it is important to note that sometimes, the malware obfuscates the strings and may also provide misleading information in the strings.

4.4.8 Third Party Plugin

The ability of the tools being evaluated to support the integration of third party tool as this enables the additional of new features during the analysis of malware is also analysed.

4.4.9 Usability

Although this is not a metric that would lead to determining how malicious a file might be, the ease of use for an analysis tool is an important feature to consider. This section is broken down into 2 subsections:

- User Interface: Command line, Graphical User interface and online analysis presence are what are considered as the measure for these tools.
- Output data management: During analysis, the most important factor is how easy it is to handle to information one gets from the analysis tool.

The observations made during the use of the tools in analysing the malware samples collected are further discussed; the technical skill level required and availability of tools that supplement the analysis of the output files.

4.5 Feature Comparison, Analysis and Evaluation.

Once the dataset of malware were analysed by the tools, the output data is analysed and results from the three tools are compared against each other including the tool features and usability as observed during the experiments.

4.5.1 Tool Feature Comparison

This section shows the comparison of the kind of information that the tools extracted and the additional features of the 3 tools. Table 4.2 provides the summary that would be important to know when deciding which tool would be best depending on the depth of static analysis required to be done on a file sample. This information was collected by observation during the analysis phase of this study.

4.5.2 Analysis and Evaluation

This section discusses the observations made during the study and provides a more detailed breakdown under each subsection.

Metric			Peframe	Pyew	Mastiff
General File Details		Filename	✓	✓	✓
		File size	✓	✓	
		MD5	✓	✓	
		SHA	✓	✓	
		SHA256	✓	✓	
Obfuscation Technique Detection		Packer	✓	✓	
		Anti- Vm	✓	✓	
		Anti-Debug	✓	✓	
APIs		General APIs Extraction	✓	✓	✓
		Suspicious API extraction	✓		
Calculated Hashes		SsDeep hash			✓
		imphash	✓		
		File Clustering based on hash			✓
PE File Details		Header	✓	✓	✓
		Sections	✓	✓	✓
		Section Entropy			✓
		Exports	✓	✓	✓
		Imports	✓	✓	✓
		Hex- Dump	✓	✓	✓
Call Graph		Generation		✓	
		Cluster Comparison		✓	
Usability	User interface	Command line	✓	✓	✓
		Graphical User Interface		✓	
		Online Analysis option			✓
	Output Data.	.txt files	✓	✓	✓
		.json files	✓	✓	
		.db output		✓	✓
Additional Features		String Extraction	✓	✓	✓
		Virus Total API utilisation		✓	✓
		Disassemble		✓	
		File metadata	✓		✓
		Extension by plugin	✓	✓	✓

Table 4.2 Static Analysis Tool Feature Comparison

4.5.2.1 File Identification:

The tools considered in this study provide different information that identifies the file. While mastiff logs the file results in a folder identified by the file name, it does not provide the MD5 or other hash values for the whole file. It instead computes the MD5 for each section and appends it at the beginning of each section. The Pyew module provides the option for the tool user to request for the filename, hash values and file name using a script called ***runme.py*** that can be edited to automate the request for each analysis. Peframe provides the most information for file identification. The comparison of the file identification information provided by the tool is shown in Table 4.2 in the general File details section. This shows that Peframe and Pyew are the stronger of the tools when there is need to immediately get file identification information upon analysis.

4.5.2.2 Detection of Obfuscation Techniques.

For this scenario, only Pyew and Peframe are considered. Pyew detects packers and anti-vm using known signatures while Peframe detects packers, anti-vm and anti-debug based on the signatures provided in the signature folder. Peframe allows for better extension options than Pyew especially for anti-vm and anti-debugging techniques as the text files used as comparison signatures can be edited. Pyew's signatures save for the packer signatures are hard-coded in the module. Analysis the results obtained from the tools based on the dataset and the comparison of the detection percentages are presented in the Figure 4.7.

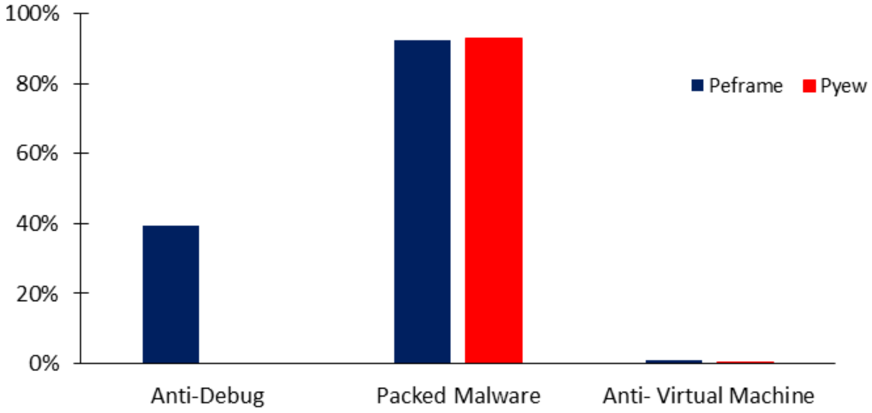


Figure 4.7 Comparison of Obfuscation Detection

4.5.2.3 Analysis of APIs

All the three tools extract the APIs that are identified in the file during analysis. While Peframe allows for only APIs deemed as suspicious based on the signature saved in its comparison file, Mastiff returns **string.txt** that contains all the strings in the analysed file which contains the APIs and Pyew returns a list of all the APIs when the command to return imports and exports are called. There is a variation in the top APIs detected by the different tools which can be explained by the fact that Peframe is heavily affected by packing where some packers led to the tool throwing an error and it returns only APIs it has matched to be suspicious in its signature database. Pyew and mastiff extract all the APIs and Pyew can perform deep code analysis to retrieve more APIs than Mastiff. For this scenario, in the future work, API calls will be extracted from a set of benign executables to perform a comparison on these functions.

4.5.2.4 PE header analysis

Here, the information given by the tools that can be used to detect if a file is suspicious or not is analysed. For example, Peframe extracts the compile time and an extract from a report of a malware sample analysed 6ec7e5c29b87c724735fea3c98b10288 shows that the file has an invalid date and it is also a good example of abnormal section names. Figure 4.8 shows that the compilation year analysis of the malware samples.

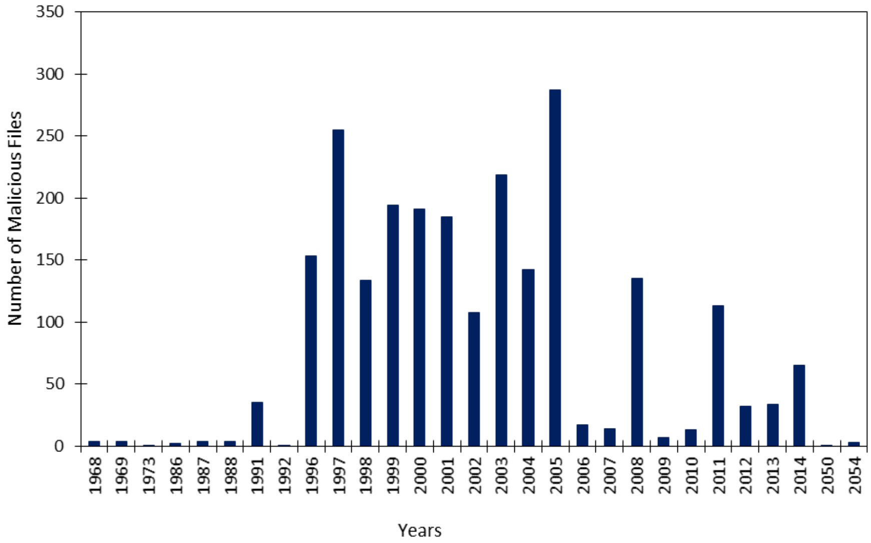
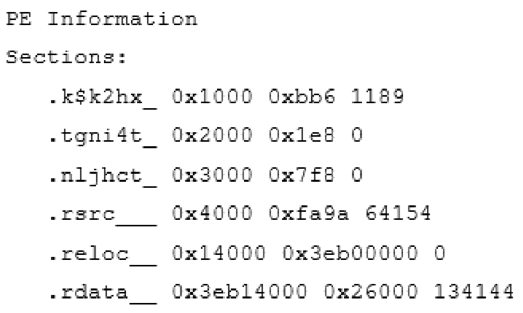
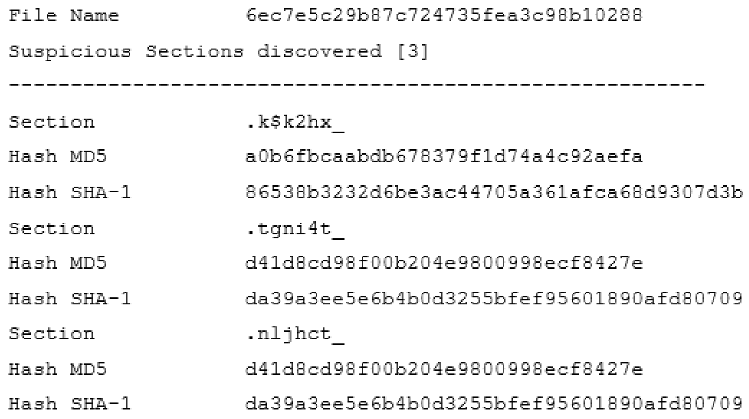


Figure 4.8 Compile year analysis of the files analysed



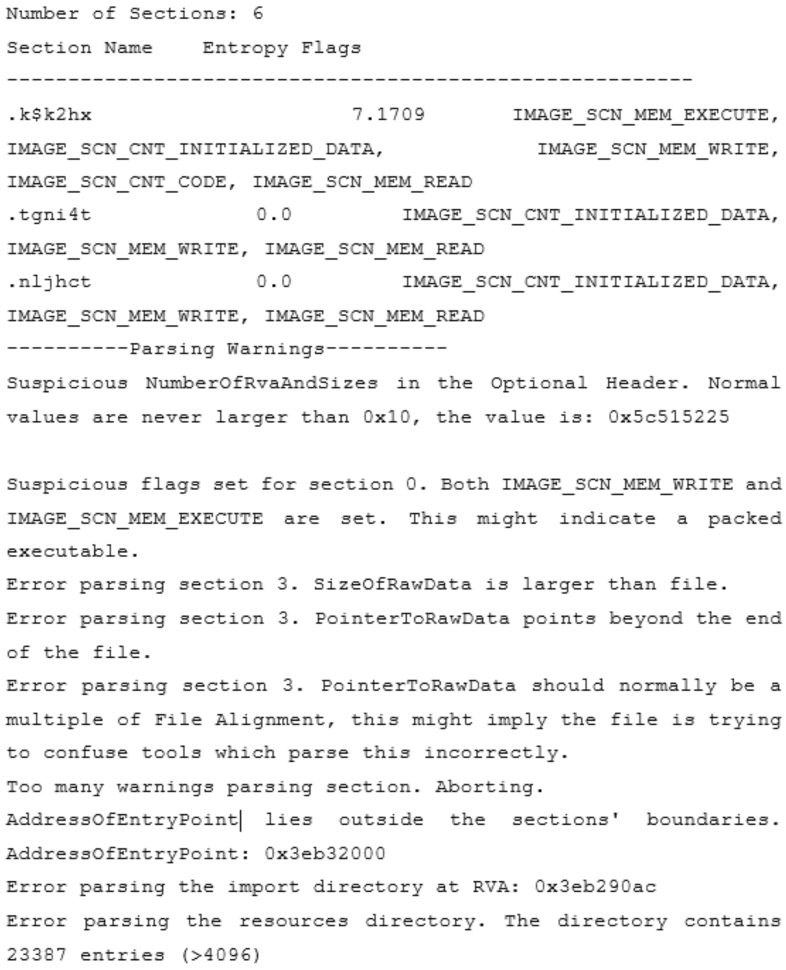
```
PE Information
Sections:
   .k$k2hx_ 0x1000 0xbb6 1189
   .tgni4t_ 0x2000 0x1e8 0
   .nljhct_ 0x3000 0x7f8 0
   .rsrc___ 0x4000 0xfa9a 64154
   .reloc__ 0x14000 0x3eb00000 0
   .rdata__ 0x3eb14000 0x26000 134144
```

Figure 4.9 Pyew Report



```
File Name          6ec7e5c29b87c724735fea3c98b10288
Suspicious Sections discovered [3]
-------------------------------------------------------
Section            .k$k2hx_
Hash MD5           a0b6fbcaabdb678379f1d74a4c92aefa
Hash SHA-1         86538b3232d6be3ac44705a361afca68d9307d3b
Section            .tgni4t_
Hash MD5           d41d8cd98f00b204e9800998ecf8427e
Hash SHA-1         da39a3ee5e6b4b0d3255bfef95601890afd80709
Section            .nljhct_
Hash MD5           d41d8cd98f00b204e9800998ecf8427e
Hash SHA-1         da39a3ee5e6b4b0d3255bfef95601890afd80709
```

Figure 4.10 Peframe report



```
Number of Sections: 6
Section Name    Entropy Flags
-------------------------------------------------------
.k$k2hx                    7.1709       IMAGE_SCN_MEM_EXECUTE,
IMAGE_SCN_CNT_INITIALIZED_DATA,            IMAGE_SCN_MEM_WRITE,
IMAGE_SCN_CNT_CODE, IMAGE_SCN_MEM_READ
.tgni4t            0.0         IMAGE_SCN_CNT_INITIALIZED_DATA,
IMAGE_SCN_MEM_WRITE, IMAGE_SCN_MEM_READ
.nljhct            0.0         IMAGE_SCN_CNT_INITIALIZED_DATA,
IMAGE_SCN_MEM_WRITE, IMAGE_SCN_MEM_READ
----------Parsing Warnings----------
Suspicious NumberOfRvaAndSizes in the Optional Header. Normal
values are never larger than 0x10, the value is: 0x5c515225

Suspicious flags set for section 0. Both IMAGE_SCN_MEM_WRITE and
IMAGE_SCN_MEM_EXECUTE are set. This might indicate a packed
executable.
Error parsing section 3. SizeOfRawData is larger than file.
Error parsing section 3. PointerToRawData points beyond the end
of the file.
Error parsing section 3. PointerToRawData should normally be a
multiple of File Alignment, this might imply the file is trying
to confuse tools which parse this incorrectly.
Too many warnings parsing section. Aborting.
AddressOfEntryPoint lies outside the sections' boundaries.
AddressOfEntryPoint: 0x3eb32000
Error parsing the import directory at RVA: 0x3eb290ac
Error parsing the resources directory. The directory contains
23387 entries (>4096)
```

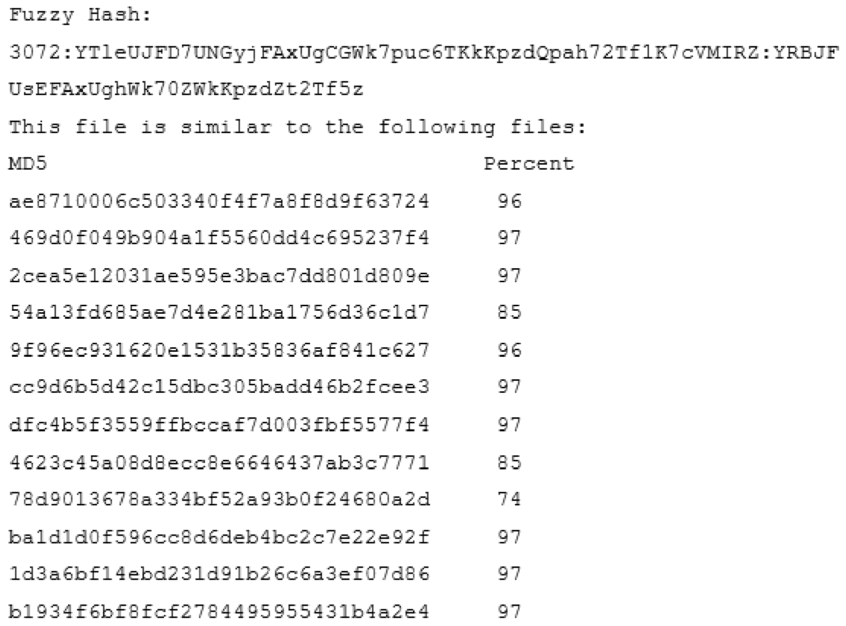
Figure 4.11 Mastiff Report

Figure 4.9, Figure 4.10, Figure 4.11 show the various report samples from the three different tools of the same sample- 6ec7e5c29b87c724735fea3c98b10288. By analysing the three reports, the information provided by the various tools defers in the level of detail. Pyew provides the shortest file report with the section names, the section addresses and sizes which are important when deeper analysis is required. Peframe provides detailed information about the header with the important field of the compile time and flags it because it detects that the time given is indeed invalid, it then also flags the 3 sections that have names that are unknown together with the hash values. These hashes can be used for hash matching during further detailed analysis.

The mastiff report provides findings in more detail even providing alerts based on the discrepancies it has detected. For example, like the file having an address of Entry Point that lies outside the section's boundaries which a known indicator that the file is malicious, a rawdata size that is larger than the actual file size. The mastiff report also provides more warning details like the section characteristic flag warnings, section field warnings, section entropy and directory warnings that can be used by analyst to deduce that a file is malicious based on PE header analysis than the other two tools. However the information from all the three tools is equally important to improve detection accuracy.

4.5.2.5 Ssdeep hashing and Malware clustering

Mastiff is the only tool of the three that calculates the Fuzzy Hash of the file and compares the hash against the hashes of the files already in the database to give similarity percentage in the files. Figure 4.12 shows an extract of one of the reports.



```
Fuzzy Hash:
3072:YTleUJFD7UNGyjFAxUgCGWk7puc6TKkKpzdQpah72Tf1K7cVMIRZ:YRBJF
UsEFAxUghWk70ZWkKpzdZt2Tf5z
This file is similar to the following files:
MD5                                 Percent
ae8710006c503340f4f7a8f8d9f63724      96
469d0f049b904a1f5560dd4c695237f4      97
2cea5e12031ae595e3bac7dd801d809e      97
54a13fd685ae7d4e281ba1756d36c1d7      85
9f96ec931620e1531b35836af841c627      96
cc9d6b5d42c15dbc305badd46b2fcee3      97
dfc4b5f3559ffbccaf7d003fbf5577f4      97
4623c45a08d8ecc8e6646437ab3c7771      85
78d9013678a334bf52a93b0f24680a2d      74
ba1d1d0f596cc8d6deb4bc2c7e22e92f      97
1d3a6bf14ebd231d91b26c6a3ef07d86      97
b1934f6bf8fcf2784495955431b4a2e4      97
```

Figure 4.12 Mastiff Fuzzy Hashing results

4.5.2.6 Call Graph Extraction and Comparison

Only Pyew has the modules that are responsible for generating call graphs and call graph comparison. However, the tool was unable to produce call graphs for packed malware and could only do so once the malware was unpacked.

To measure if the call graphs produced are good enough, call graphs generated by this tool and the ones generated by IDA Pro are analysed. The nodes identified by Pyew are fewer, however, it still produces a call flow that can be used to classify the malware based on the graph clustering module in the tool. This module was analysed next to measure the accuracy of its findings against the results obtained using Ssdeep hashing. Using the report shown in Figure 4.9, the graph clustering module was used to analyse the original file against 3 of the files in the extract. Analysis of the similarity detection was further performed using the file identified with *MD5: 5f232bc72932b846855cdddc8d86a01* and its' fuzzy hash matches from Mastiff. The files were uploaded in the graph clustering similarity module in pyew and Figure 4.13 shows the comparison of the results achieved. The results obtained from the module in Pyew gave 100% similarity across all the files for expert, Alist and Primes as shown in Table 4.3 and then in Figure 4.13 which argues against the module's accuracy in graph cluster matching.

File MD5	**Mastiff**	**Pyew**		
		Expert	**Alist**	**Primes**
dfc4b5f3559ffbccaf7d003fbf5577f4	97	100	100	100
4623c45a08d8ecc8e6646437ab3c7771	85	100	100	100
78d9013678a334bf52a93b0f24680a2d	74	100	100	100

Table 4.3 Comparison of Mastiff Similarity detection vs Pyew cluster graph similarity analysis

Mastiff gives more defined answers than pyew results although the difference can be seen in one file. While the graph clustering module in Pyew is a very good tool to be used in conjunction with other information extracted, it is not best to use it as a reliable tool but rather combine it with other tools to build a framework with better detection accuracy.

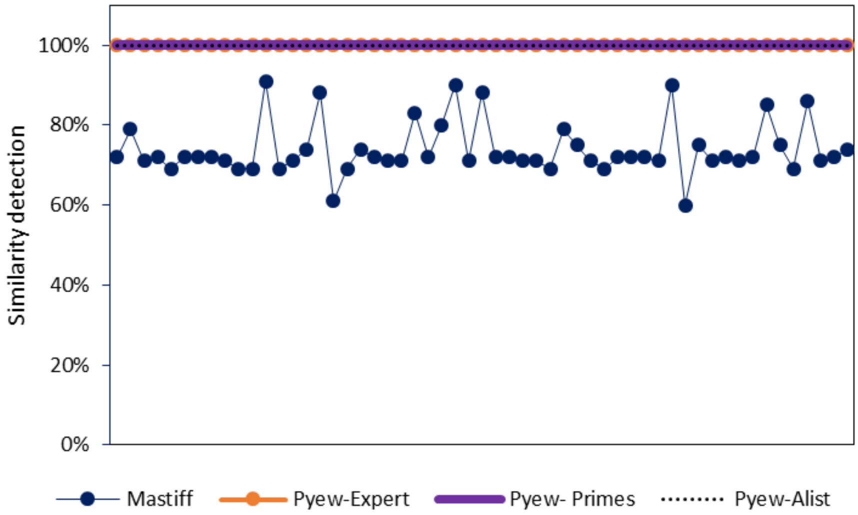


Figure 4.13 Similarity Detection Comparison

4.5.2.7 String Analysis

All the three tools extract strings. Using Peframe, URLs were extracted and a count was used to see how many times each URL appears in the dataset and the top 10 are listed in Table 4.4

Top 10 URLs	**No.**
http://ocsp.thawte.com0	43
http://nsis.sf.net/NSIS_Error	40
http://crl.thawte.com/ThawteTimestampingCA.crl0	39
http://ts-aia.ws.symantec.com/tss-ca-g2.cer0	39
http://ts-crl.ws.symantec.com/tss-ca-g2.crl0	39
http://ts-ocsp.ws.symantec.com07	39
http://crl.thawte.com/ThawtePCA.crl0	31
http://www.usertrust.com1	31
https://d.symcb.com/cps0%	28
https://d.symcb.com/rpa0	28

Table 4.4 Top URLs extracted

The URLs extracted can be used as fingerprints to detect if a file is malicious and they can also be used to group what kind of malware they are.

4.5.2.8 Third Party Plugins

Using the virus total plugin, the detection across the samples gives good results because the samples are older than one year. However, from the analysis, even given that the samples are old, Table 4.5 has a list of files where the Anti-Virus engines that Virus Total uses return a detection rate of 0 %.

Sample MD5	**%**
f2d69c64f6e98deb05243213e5561bf6	0%
6c55b3c4d59420b2f4198b2b2ea32d25	0%
7e7deb713a16c0ad00f3a7f7a9ae3eca	0%
77cfb9a441eb8516943da23dbd035cba	0%
e97143b1c63caf1db8e4a3ca086c3834	0%
049630bfdfa9f2d19aa9f9073352012d	22%
345004633174388211c2475cedb6de9a	54%

Table 4.5 VirusTotal analysis results giving 0% detection of known malware

This shows that even some malware are not detected by a collection of anti – virus engines and therefore there is a need to fill this gap. The overall detection analysis obtained from virus total is shown in

Figure 4.14.

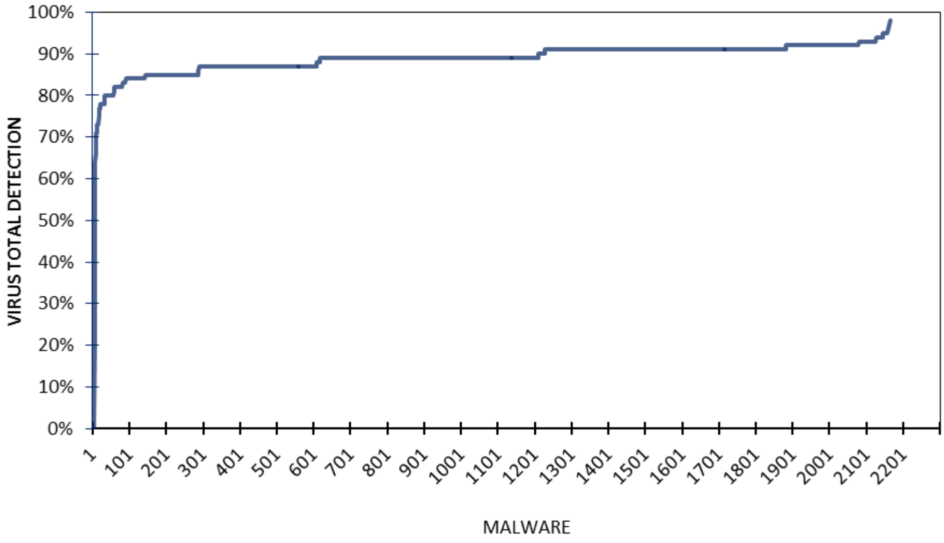


Figure 4.14 Virus Total Detection Analysis of the samples

Vigna [13] notes that relying on Anti-viruses to detect malware is not a good solution even a year after malware has been discovered. Vigna further argues that some engines would not detect the malicious files while detection of the malware on the day of discovery is limited to 51% of the engines sometimes and there are some cases where it takes up to 2 days before the anti-viruses can even detect a new malware sample. These statistics do not favour the reliance on the protection offered by anti-virus engines and particularly for virus total. There is always a time delay of 10 minutes between upload and retrieval of the report. However, the information offered by Virus total is a great addition for malware analysts when detecting malware in systems.

4.6 Chapter Summary

In this chapter, 3 new static analysis tools that provide advanced static analysis statistics are evaluated. Although most of the tools provide the same information, mastiff is more detailed than the others and Pyew introduces new modules which can improve the detection and clustering rates while Peframe provides a simple but straight forward report. During the experimentation, some of challenges faced that allowed us to develop scripts that allow for better automation of the analysis process and analysis output log manipulation and management. Some of the limitations of the tools that were identified were that not all indicators of compromise in a PE file are been fully explored in all the tool. This is a window that provides opportunity for newer automated static analysis tools.

The evaluation of the three tools provides a foundation for the next phase of study that looks at using information for the tools together with scoring methods to provide a quantifiable metric for the maliciousness of a file.

CHAPTER 5. MALSCORE: AN ANORMALY HEURISTIC FEATURE BASED PROBABILISTIC SCORING METHOD FOR DETECTION OF MALICIOUS PORTABLE EXECUTABLES

5.1 Introduction

In 2015, Forbes reported that it takes about 46 days for reported cyber incidents to be resolved [125] . One contributor to this big mean time to recover is that most incident responders are faced with the challenge of figuring out which files on the system could be malicious over those that are not. Given the ever-changing environment on a computer, such a determination is very challenging. Numerous static analysis methods provide the initial information required for an experienced analyst to deduce the intention of the file as long as the analyst has the expertise to use the tools. No matter how experienced the analyst is, analysis of high volumes of files will always require a lot of time. This challenge has led to the growth of automated static analysis tools whose results still require the use malware analysis expertise to deduce the intent of file under analysis. For most PE static analysis tools, file anomalies are some of the information used to decide if the file is malicious or not. Various studies show that anomalies in file features extracted can be used as indicators of compromise [52], [126]–[128]. Attaching a scoring method to anomalies as heuristic indicators of compromise (HIoC) could allow for a faster and more efficient automated static analysis while increasing the decision-making confidence level even to a standard system user. Following the evaluation of the 3 tools in chapter 4, this chapter proposes a method of measuring the malicious intent of file based on the identified heuristic anomalies by using probabilistic scoring. We initially present interesting heuristic anomalies observed from the analysis data of 1.6 million malware samples in section 5.2. Section 5.3 details the used test environment, the method design and the implementation process. 5.4 presents and analyses the achieved results and finally we conclude and summarise the chapter in section 5.5.

5.2 Heuristic observations from the static analysis data of 1.6 million malware samples and discussion.

During this study, we had access to data from Nettitude Research and Innovation Department's analysis of 1.6 million malware samples using a customised version of Peframe static analysis tool. For security reasons, the experiment setup of this analysis is not included in this work. We however, present the interesting observations made from this data.

Top 20 Malware Families in the Sample by ClamAv.	
Win.Adware.Imali-17	Win.Adware.Multiplug-60223
Win.Adware.Multiplug-5	Win.Trojan.Vilsel-4621
Win.Adware.Domaiq-1	Win.Trojan.Madangel-1
Win.Trojan.Antifw-171	Legacy.Trojan.Agent-1388596
Win.Trojan.Ramnit-1847	Win.Adware.Multiplug-53339
Win.Trojan.Agent-1388669	Win.Worm.Allaple-5
Win.Adware.MultiPlug-1	Win.Spyware.78636-2
Win.Adware.Multiplug-3	Heuristics.W32.Parite.B
Win.Trojan.Agent-1388655	Win.Trojan.Redir-13
Win.Trojan.Agent-1388676	Js.Malware.Autolike-1

Table 5.1 The top 20 malware types identified during analysis

In instances of file analysis using static analysis tools, it is possible for the analysis process to fail due to file malformations or anti- analysis techniques. Some of the errors obtained from the analysis data are shown in Figure 5.1.

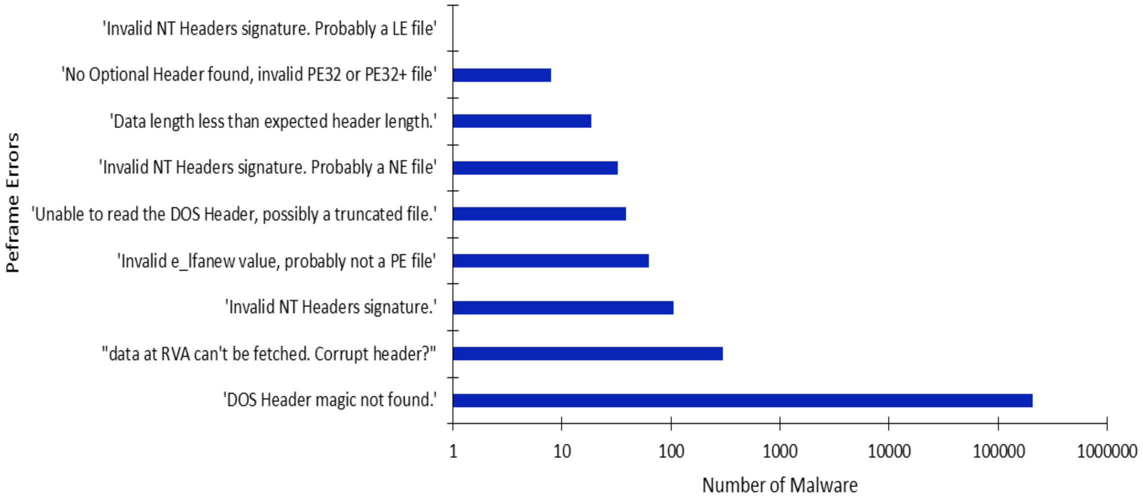


Figure 5.1 Peframe errors based on failed file analysis

The compile time analysis as shown in Figure 5.2 shows that there is some questionable compile times for example a file having been compiled in 2055 and seen in 2016. We see that about 19% of the files were compiled before the year

2000. Given that most legitimate software providers always work to update their software, it is reasonable to flag such a compilation timeline.

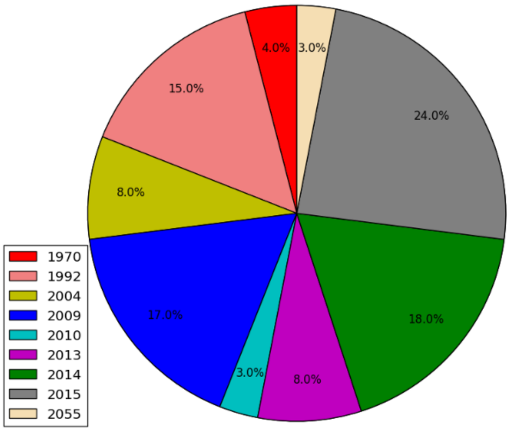


Figure 5.2 Malware compile time

Packer	Malware Samples	% of total sample
Microsoft Visual C++ 8	408013	25.5%
Nullsoft PiMP Stub -> SFX	73572	4.6%
Microsoft Visual C# / Basic .NET	72610	4.5%
Borland Delphi 3.0	64234	4.0%
Microsoft Visual C++ v6.0	52572	3.3%
PECompact 2.x -> Jeremy Collake	36672	2.3%
UPX 2.93 (LZMA)	16901	1.1%
Microsoft Visual Basic v5.0	16307	1.0%
Microsoft Visual C++ v7.0	15839	1.0%
AHTeam EP Protector 0.3 (fake PCGuard 4.03-4.15) -> FEUERRADER	11106	0.7%
MingWin32 GCC 3.x	9671	0.6%
Microsoft Visual Basic v5.0 - v6.0	7668	0.5%
MSLRH V0.31 -> emadicius	7347	0.5%
UPX v0.80 - v0.84	5981	0.4%
UPX -> www.upx.sourceforge.net	5521	0.3%
ASProtect V2.X DLL -> Alexey Solodovnikov	4091	0.3%
Microsoft Visual C++ v6.0 DLL	2486	0.2%
Microsoft Visual C++ 5.0	2294	0.1%
ACProtect 1.3x - 1.4x DLL -> Risco Software Inc.	1809	0.1%
Safeguard 1.03 -> Simonzh	1611	0.1%

Table 5.2 Top 20 identified Packers

Based on these results, compile time is a heuristic feature worth comparing when building a heuristic scanner. Another feature that was extracted from the samples was the top 20 packers identified as shown in Table 5.2. Next, we analyse data about the file sections and Figure 5.3 shows the distribution of the number of sections in the files while Figure 5.4 shows the 20 most popular section names extracted from the data and Figure 5.5 shows the distribution of the calculated section entropy.

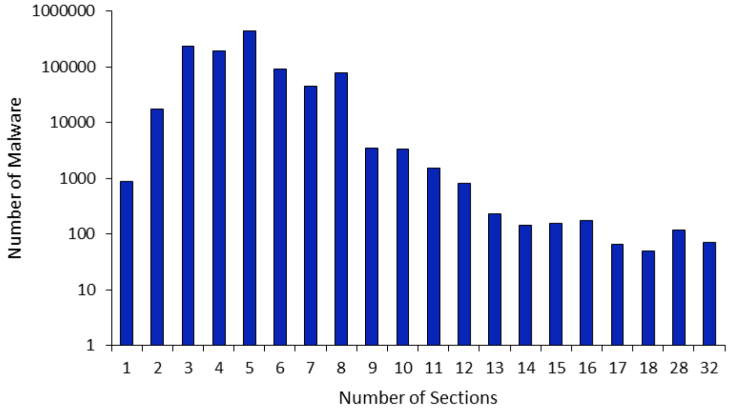


Figure 5.3 The Distribution of the number of sections in the analysis data

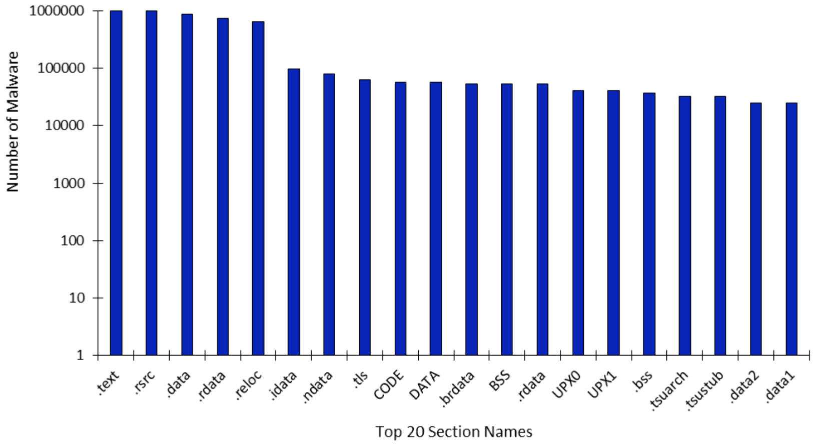


Figure 5.4 Top 20 Section Names in analysed sample

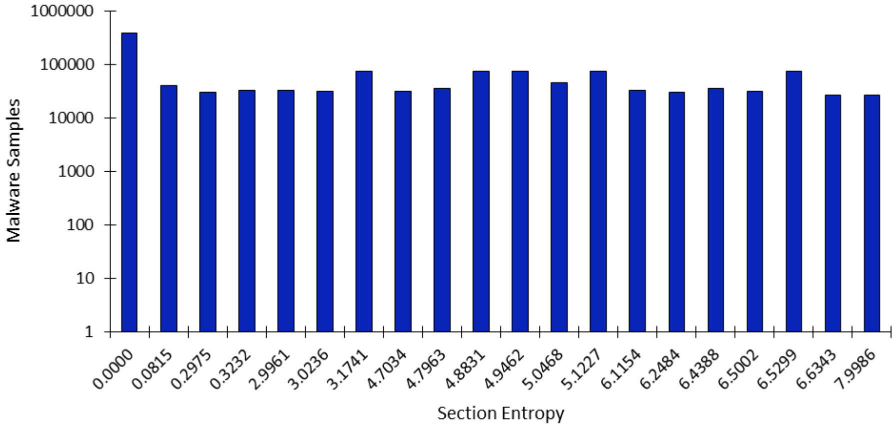


Figure 5.5 The Section Entropy distribution

The most common libraries seen in the malware samples were populated as shown in Figure 5.6. Peframe as a tool has API signatures that are deemed suspicious when seen in analysed files. The top 20 anti-debug APIs extracted are shown in Figure 5.7 are extracted and Figure 5.8 shows the distribution of the top 20 suspicious APIs seen.

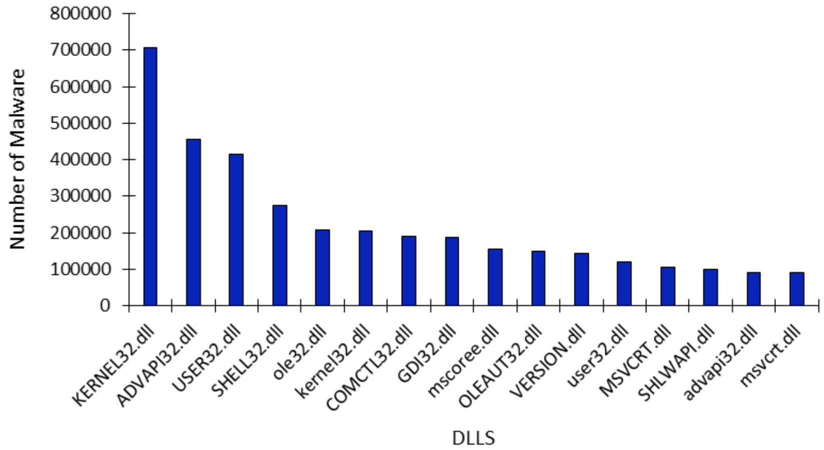


Figure 5.6 Top DLLs in the Analysed sample

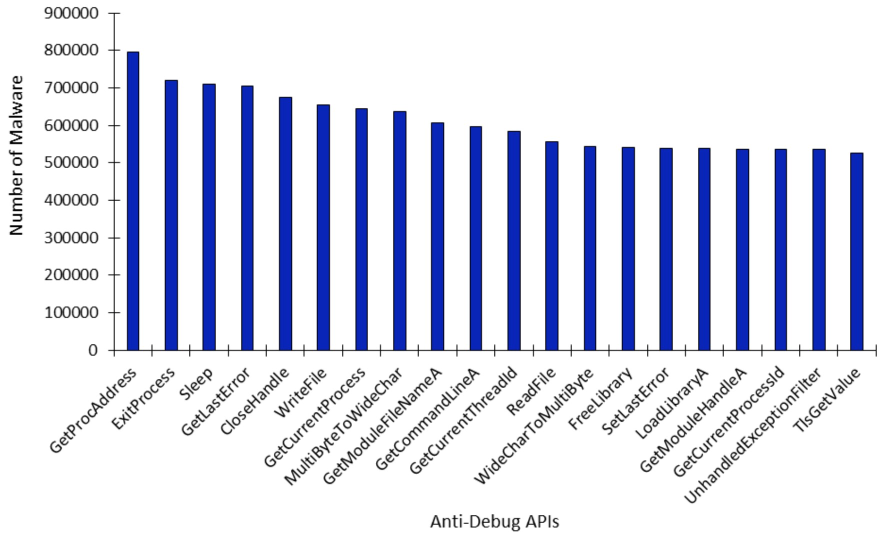


Figure 5.7 Top 20 Anti-debug APIs indentified

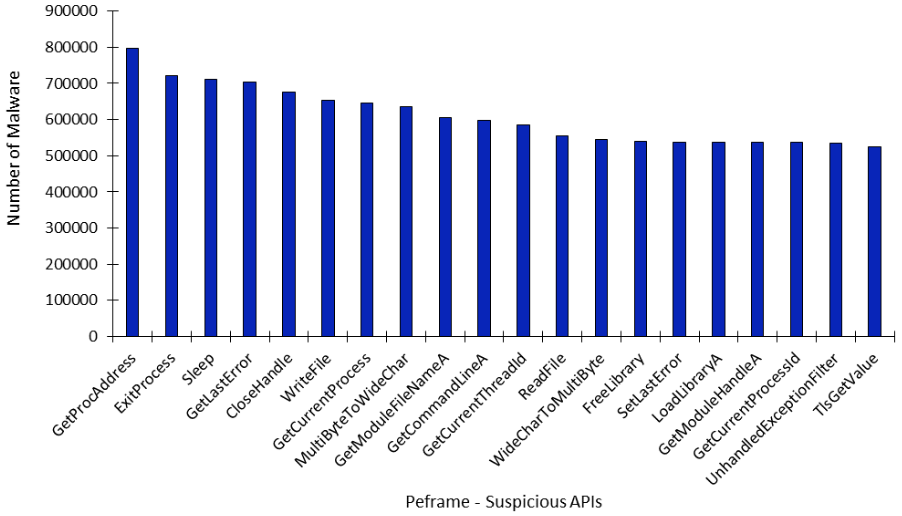


Figure 5.8 Top 20 Suspicious APIs indentified

One of the most important extracts from the file are the strings which can sometimes lead to initially understanding what the file functions are. Therefore, analysing the strings extracted from the analysis data was very relevant. The first step was to extract any data related to the filenames that could be extracted from the strings. The filenames shown Figure 5.9 are what were extracted.

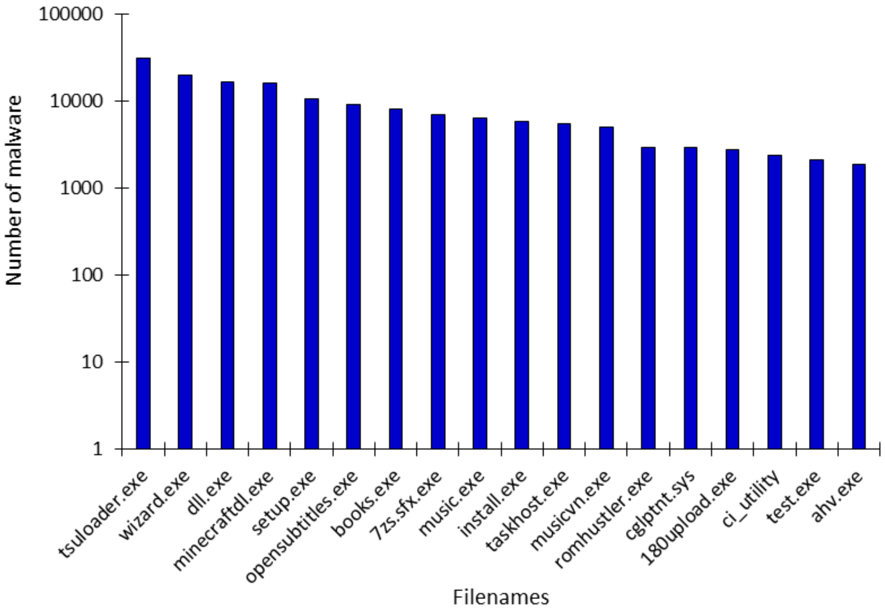


Figure 5.9 Filenames extracted from the data

Today, malware are known to try and connect to the outside world to either establish command and control, retrieve other files or even send data [80]. Therefore, any strings that look like URLs or emails can be very useful. Performing this analysis on the sample led to populating Table 5.3 and Table 5.4 which show the top 20 email addresses and top 20 URLs respectively.

Email Addresses	
support@getwebcake.com	info@mbsoft.gr
support@mitcsoftware.com	jdeb@autoscript.com
support@rjlsoftware.com	pop@harzing.com
support@yontoo.com	sales@totusoft.com
71174.2675@compuserve.com	sandy-cyf@163.com
Soft@leinao.com	supermca@yandex.ru
csli534@ctimail3.	support@buzzdock.com
huidawo@hotmail.com	support@bytessence.com
support@puffinwarellc.com	support@mypropertyprogram.com

Table 5.3 Email Addresses extracted from the data

Top 20 URLS
http://ocsp.thawte.com
http://crl.thawte.com/ThawteTimestampingCA.crl
http://ts-ocsp.ws.symantec.com
http://ts-aia.ws.symantec.com/tss-ca-g2.cer
http://ts-crl.ws.symantec.com/tss-ca-g2.crl
https://www.globalsign.com/repository/
http://crl.globalsign.net/root-r3.crl
https://www.globalsign.com/repository/
http://crl.globalsign.com/gs/gscodesignsha2g2.crl
http://ocsp2.globalsign.com
http://secure.globalsign.com
http://www.w3.org
http://ocsp.comodoca.com
http://bi.downthat.com
https://secure.comodo.net
http://crl.comodoca.com/COMODOCodeSigningCA2.crl
http://crt.comodoca.com/COMODOCodeSigningCA2.crt
http://nsis.sf.net
http://schemas.microsoft.com/SMI/2005/WindowsSettings
http://ocsp.usertrust.com

Table 5.4 Top 20 URLs extracted from the analysis data

5.3 Methodology Design and Implementation

The proposed method design and implementation are discussed in this section. The different steps and algorithms built and used are described herein. The approach is broken down into steps with the aim of obtaining a scoring method for PE files based on the anomalies identified during the study.

5.3.1 Test Environment and Dataset formulation.

The test bench environment is built to allow for fast analysis with output information being saved to a single file log for malicious files analysed and another for the clean files to allow for easier data analysis. The algorithm uses the prepared dataset that is subdivided as shown in Table 5.5 and the tools used are described in Table 5.6.

Dataset	Number of files			Use in the system
	Total	Clean	Malicious	
A ⟵ {Am, Ac}	50654	698	49956	Aggregate and score the Heuristic Indicators of Compromise
B ⟵{Bm, Bc}	55373	940	54433	Detection Scoring -method evaluation phase.

Table 5.5 MalScore datasets formation and their uses

Tool	Specifications/ Details
Computer	Dell T1700, CPU – Intel Xeon@ 3.1GHz, RAM 32GB. Hard Disk – 500GB
Machine OS	Linux Mint 17.1 (#64 – Ubuntu SMP)
Analysis and detection tools	Scripted in Python using some from Pefile [129], Peframe [73] and Pescanner [130] integrated with customised functions specific to this study.

Table 5.6 Malscore Test Bench Specifications

5.3.2 Aggregation of the Anomalies.

Creating a list of anomalies to consider in this study required extracting file features of the sample PE files both clean and malicious and aggregating them. For this we revisited work in Chapter 4. The anomalies are defined in this work as values with in the PE file that do not conform to the predefined values in the PE specification document [24]. Analysed malicious file extracts are used and the features of clean files are extracted as a control measure using dataset A.

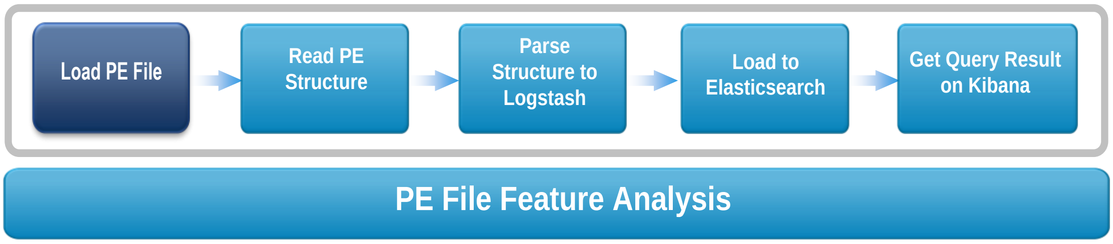


Figure 5.10 PE file Feature Analysis Component Layout

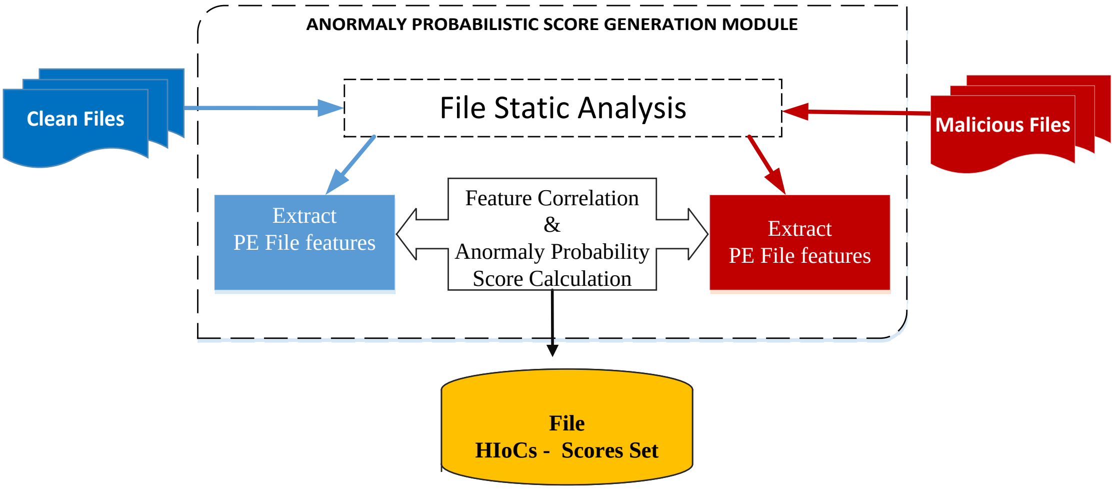


Figure 5.11 Pictorial representation of the Heuristic anomaly probabilistic score generation module

A customised PEparser based Python tool is used to parse PE files and the extracted information in Json format is passed to an ELK (Elastic Search, Logstash and Kibana) server [131] for better aggregation and visualisation of the data. The Clean PE file features are also analysed to extract corresponding information. The Heuristic features extracted of the Malicious and clean files are compared to obtain the Heuristic Indicators of Compromise Set (HIoCs) as shown in Figure 5.11. This comparison allows for the defining of the scoring rules that are used to define the trigger references.

5.3.3 Application of the Conditional Probability Theories.

Taking the hypothesis that a file is either malicious (M) or Clean (C) and A represents the likelihood of an Anomaly being detected in a file based on the defined scoring rule. Using Bayes' theorem, the hypothesis that the file is malicious given the anomaly is defined by [132]:

$$P(M \mid A) = \frac{P(A \mid M) \bullet P(M)}{P(A)} \tag{5.1}$$

However, P(A) is not known so applying the theorem of total probabilities, it can be broken down to known outcomes:

$$P(A) = P(A \mid M) \bullet P(M) + P(A \mid C) \bullet P(C) \tag{5.2}$$

Where P(A|M) and P(A|C) are determined from the training set. In designing this method, P(M) and P(C) are assigned a value of 0.5 each due to the principle of indifference. It assumed that the file is either clean or malicious and therefore:

$$P(C) = \neg P(M) \tag{5.3}$$

Since the hypotheses Clean and Malicious define the total probability space;
P(C) = P(M) = 0.5

5.3.4 Identification of Trigger features.

In this step, Dataset A in Table 5.5 is use to formulate the HIoC set using the features which are shown to be more dominant in malware files as compared to clean files are identified. A dominance rule is used to identify these trigger features and characteristics in the dataset. We define this dominance rule as:

$$\mathrm{S(X)} = (\mathrm{Dominant\ feature\ X}) \Leftrightarrow \forall_{\mathrm{i}} \, (\mathrm{P(M} \mid \mathrm{X_i}) > (\mathrm{P(C} \mid \mathrm{X_i})) \tag{5.4}$$

Where: S(X) is a set of all anomalies collected by the dominance rule.

X is a Heuristic Indicator of compromise/ anomaly, i= {1,2, 3,…}

We revisit work in [126] and perform analysis to create the scoring rules that are used as triggers to attach the different heuristic scores during the training phase. The top 25 dominant features that provided high dominant feature scores as shown in Table 5.7 form the HIoCs based score.

5.3.5 Formulation of the Individual Anomaly Score.

With the triggers of HIoCs set, scoring the anomaly means attaching a probabilistic score so that the overall file score is a quantifiable metric measuring file maliciousness.

Once the probability of the file being malicious given a specific anomaly is detected is known for each of the selected 25 HIoCs, there is need to normalise the probabilities so that the total anomalies score of the file adds up to 100. This requires defining the principle score attached to each anomaly detected individually S(A$_i$) as:

$$S(A_i) = \frac{P(M \mid A_i)}{\sum_n P(M \mid A_n)} \% \tag{5.5}$$

Where: S(A) is a score of the anomaly A, n is the total number of anomalies used in the study (n = 25), i= {1,2, 3,…}

The different anomaly scores are populated in a set S_Anom_set which are used by MalScore algorithm to score files under analysis.

5.3.6 MalScore: The Heuristic Scoring Algorithm.

In this algorithm, the file is analysed by Pefile and the file features are extracted.

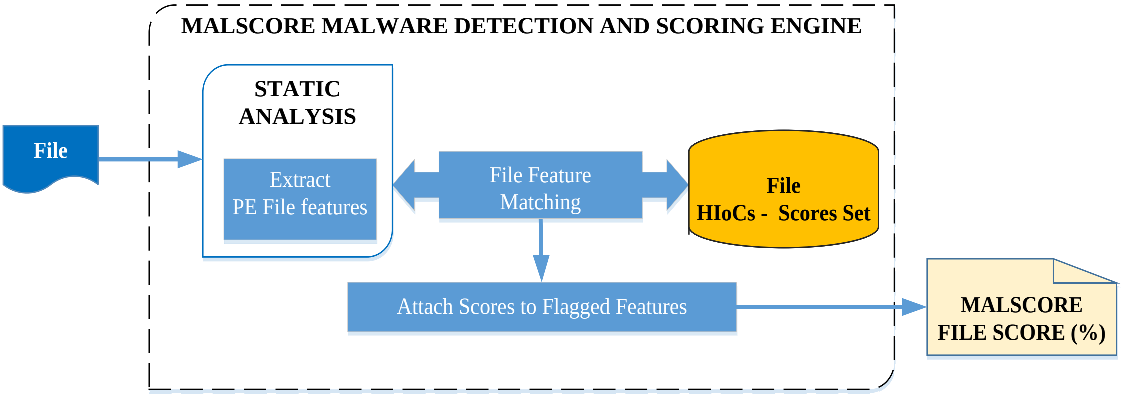


Figure 5.12 Pictorial representation of the MalScore approach

P(M) = P(C) = 0.5							
	Features (HIoCs)	**Scoring Rule**	**P(A/M)**	**P(A/C)**	**P(A)**	**P(M/A)**	**S(A)**
File header	Compile Time	Year < 1992 or Year > 2016	0.361	0.004	0.183	0.989	**4.24%**
	CheckSum	Value != Calculated Checksum	0.532	0.000	0.266	1.000	**4.28%**
	Number of Sections	Value < 1 or Value >9	0.637	0.009	0.323	0.986	**4.22%**
	Characteristics (RELOCS_STRIPPED)	Value = 1	0.752	0.100	0.426	0.883	**3.78%**
	Characteristics (LINE_NUM_STRIPPED)	Value = 1	0.820	0.351	0.586	0.700	**3.00%**
Optional Header	NumberOfRvaAndSizes	Value != 0	0.351	0.000	0.176	1.000	**4.28%**
	Size of Image	Value != SizeOfHeader + Sections SizeOfRawData	0.400	0.000	0.200	1.000	**4.28%**
	SizeOfOptionalHeader	Value != 224	0.310	0.003	0.157	0.990	**4.24%**
	Address of Entry Point/ File Size	Value >2	0.223	0.004	0.114	0.982	**4.21%**
	LoaderFlags	Value !=0	0.525	0.000	0.263	1.000	**4.28%**
Sections	Section Entropy	Value < 1 or Value > 7	0.470	0.022	0.246	0.955	**4.09%**
	Size of Raw Data	value = 0	0.930	0.005	0.468	0.995	**4.26%**
	Pointer to Raw Data	Value = 0	0.871	0.410	0.641	0.680	**2.91%**
	Section Virtual size and Raw size	Virtual size < Raw size	0.870	0.420	0.645	0.674	**2.89%**
	Section Names	Not in List	0.783	0.010	0.397	0.987	**4.23%**
Rsrc	Resource Section Sub-language	Value = 0	0.394	0.008	0.201	0.980	**4.20%**
	Resource Size/ file size	Value > 0.05	0.120	0.002	0.061	0.984	**4.21%**
Others	URLs	Present?	0.486	0.005	0.246	0.990	**4.24%**
	Anti-debug APIs	Number >2	0.983	0.060	0.522	0.942	**4.04%**
	Suspicious APIs	Number >5	0.990	0.050	0.520	0.952	**4.08%**
	File Entropy	Value > 7	0.792	0.034	0.413	0.959	**4.11%**
	Packer	File Packed?	0.970	0.362	0.666	0.728	**3.12%**
	Anti - VM	Present?	0.586	0.006	0.296	0.990	**4.24%**
	Embedded file	TRUE	1.000	0.000	0.500	1.000	**4.28%**
	File Signed	FALSE	0.651	0.001	0.326	0.998	**4.28%**

Table 5.7 The HIoC set Scoring formulation.

As shown in Figure 5.12. for each marked anomaly from the populated HIoCs detected in the extracted features, the score corresponding to that anomaly is attached to the total file MalScore (Fms). The Fms is returned after the algorithm has looped through the all the extracted features matching them against the set of HIoC as the quantifiable metric to measure the malicious intent of the file. The pseudo code for the algorithm is described in Algorithm 5.1.

Algorithm 5.1 MalScore- Heurisitic scoring

```
Input: PE file f , HIoCs, S_Anom_set
Output: Fms

Overall Heuristic Scoring Phase

procedure: Malscore
    Extract
            Ff ⟵ File Features (f)
    for a in Ff do
         if a ∈ HIoCs then
              $Fms_f$ =+ S_Anom_set (a)
          end if
      end for
return $Fms_f$
end procedure
```

5.3.7 Formation of the Heuristic Detection Score threshold.

Like any other malware detection method, the purpose of this technique is to capture all the malicious files while letting the clean file through. However, there is yet to be the perfect system with 100% true detection and 0% false detection. The best strategy is always to design a system that has very high true positive detection while keeping the false positive rates very low. The true positive – false positive trade-off creates a need for creating a threshold percentage above which the file is said to be malicious and below which the file is labelled clean.

5.4 Proposed Scoring and detection Method Results and Analysis

The validation of the designed method involved scoring both malicious and clean files. Using Dataset B in Table 5.5, the results achieved are as discussed here in. The MalScore file score area covers in Figure 5.13 show that most of clean files scored in the lower half of the scoring range while most malicious files score above it. Further analysis involved determining the MalScore threshold which determines at which point the file is deemed suspicious. We define this threshold as: File Score≤ 45%; File is clean and File Score > 45%; File is malicious. Using Figure 5.14, 45% provides a trade-off of 97.6% true positive detection vs 0.6% false positive detection. It is our recommendation based on the dataset tests that 45% is the threshold. To evaluate the fitness of the designed model, Figure 5.15 provides the ROC curve of the model which plots the true positive rate against the false positive rate. The trend of the ROC curve shows that this model is good fit and therefore, it can be used for efficient malware detection.

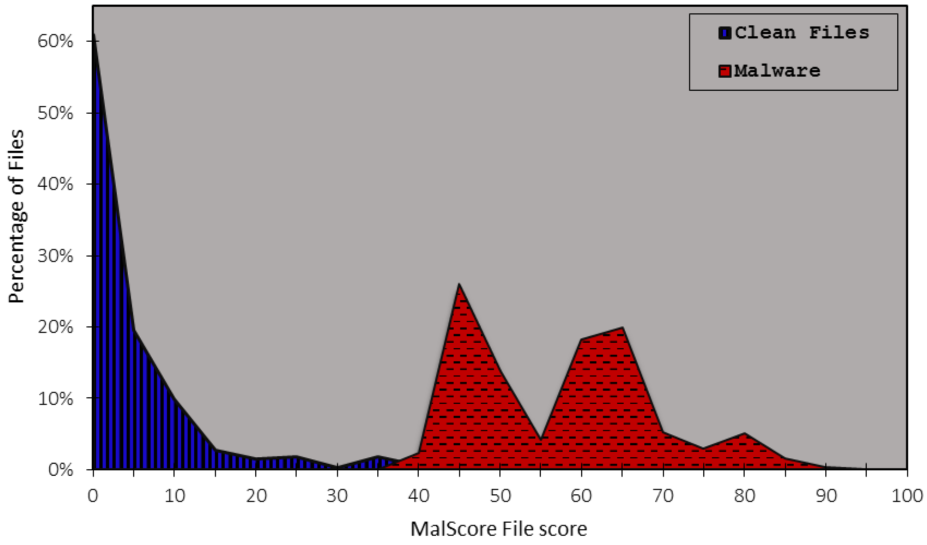


Figure 5.13 MalScore Malicious and Clean files Score Area curves

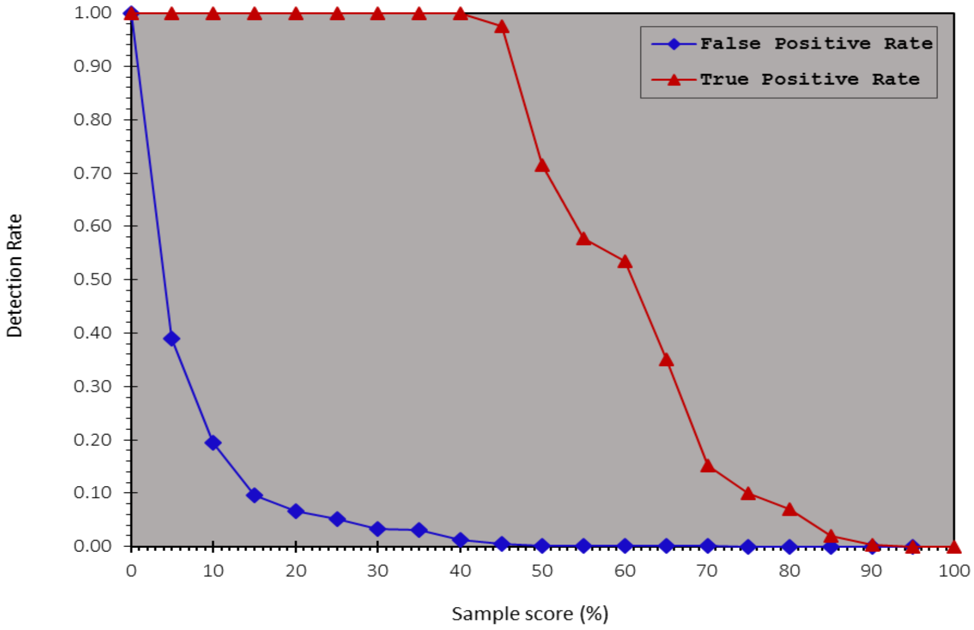


Figure 5.14 False and True Positive Detection Rates against the MalScore file scores

Further analysis of the method based on the malware family distributions in the dataset as seen in Figure 5.16 shows that the method performs better for some malware types; adware, Trojan and worm in comparison to other type. The average analysis time for each file for this method achieved in this study is 6 seconds which makes the method fast.

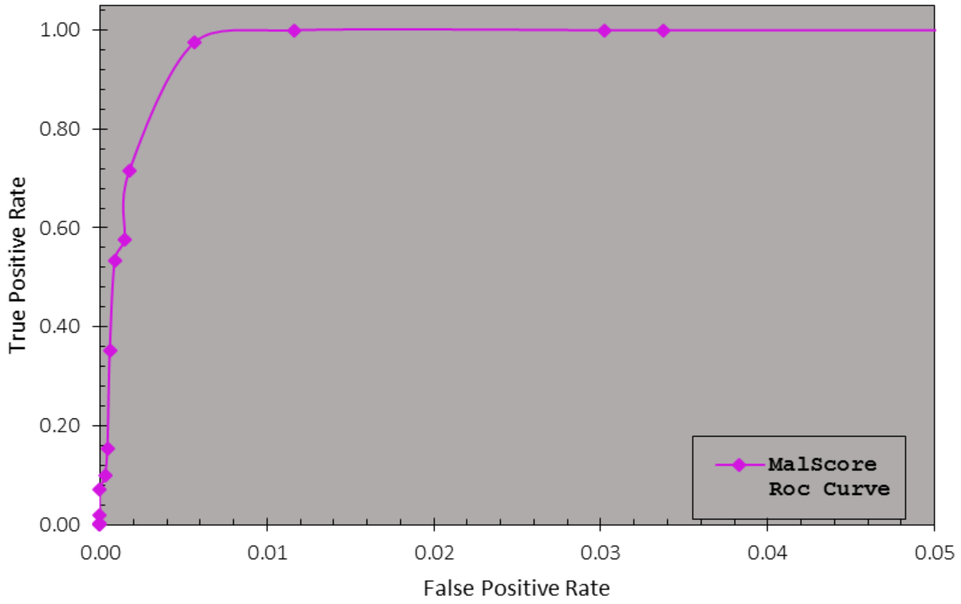


Figure 5.15 The Malscore ROC curve

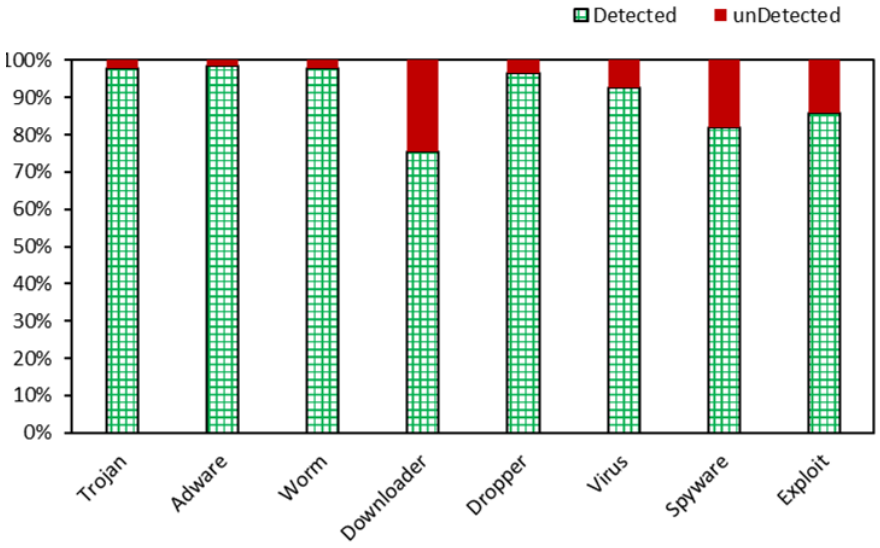


Figure 5.16 The detection ratio of the malware types in the test Dataset

5.5 Chapter Summary

This chapter presents the study strategy taken to design a heuristic anomaly based scoring approach MalScore that is one of the contributions of this thesis.

Initially, statistics and observations from the static analysis of 1.6 million malware samples using a customised Peframe are presented and discussed. Using these observations, we explain the design and implementation process of the heuristic malware detection approach proposed that uses anomaly probabilistic scoring. The results obtained from the testing process are presented and discussed and this approach allows for quantifying the file’s malicious status. The results achieved at the suggested threshold 45% of 97.2% true positive detection vs 0.6% false positive detection provide a basis for arguing for this approach. The method is built using python scripts that can be customised, are very light which makes for a light but effective solution.

CHAPTER 6. MALHASCORE: MALICIOUS PORTABLE EXECUTABLE STATIC SCORING METHODOLOGY USING EVIDENCE COMBINATIONAL THEORY WITH HEURISTIC FEATURE CALCULATED HASHES.

6.1 Introduction

Malware detection is an area with a lot of uncertainty especially in cases where we need to rely on the human factor for the better part of the analysis and therefore detection. Malware analysis is a challenge especially given the big number of malware that analysts must handle to build signatures. For instance, as of writing this report, nearly 600 million malware have been collected according to AV-Test Institute compared to the almost 500 million malware collected last year, 2015 [43] of which only 12% were new malware.

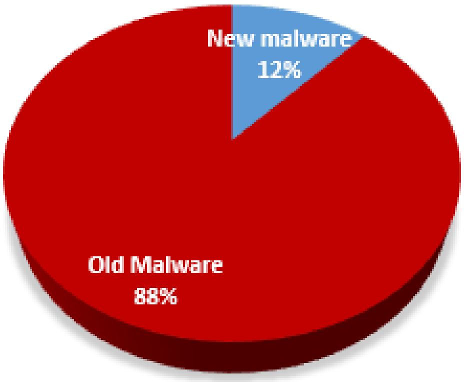


Figure 6.1 Old malware and New malware statistics

The evolution of networking technologies implies that malware delivery channels are readily available. The high-performance systems used today require similarly high performance security methods and therefore new optimised malware detection techniques that can perform real-time detection with very low effect on the performance of the system are needed. This automatically limits the use of dynamic analysis based detection methods. Dynamic analysis requires setting a time frame for the execution and observation of the file under analysis which can be quite time consuming. Fuzzy hashing is a known technique first developed for anti-spam research that is used to find the measure of similarity between two files [133]. It has been adopted in malware analysis and detection to speed up the processes given the exponential increase in malware samples discovered daily.

File similarity is used to cluster malware into families whose common signature can then be designed. However, Hashing has not been fully implemented to be used in malware detection because it can easily be evaded by applying a simple obfuscation technique such as packing. This challenge has limited the usage of hashing to triaging of the samples based on the percentage of similarity between the known and unknown. Various publications on the different fuzzy hashing methods show promising results mainly for the clustering of malware [50], [53], [56], [113], [134]. Although hashing faces the issue of high false negatives, a combinational approach could lead to better results. By focusing on files of the same type, structure as a discerning factor is eliminated. The introduction of various hashing functions that have been tested include some that overcome obfuscation allows this design and implementation of this approach as a means of better malware detection.

Most expert systems show low errors in decisions that are based on uncertainty because of the different mathematical theories developed and implemented [59]. Although reasoning under uncertainty introduces interesting concepts that can be applied to malware detection, there is need to better integrate these theories into the systems that are mainly dedicated to malware detection. Many reasoning models based on uncertainty have been developed to enable expert systems to make decisions based on unreliable data [59] and this theory is used in this study to propose a new approach to malware detection.

If the similarity in the files detected by the hashing functions is used as the heuristic attribute similarity factor for a sample dataset for the decision-making process of malware detection, multiple attribute decision making and evidence combination mathematical models are applicable in the automation of this process.

The proposed method is appealing because:

- Hash functions are easily calculated during the basic static analysis of a malware sample. This implies that the deployment cost and manual effort required for dynamic analysis and advanced static analysis are avoided.
- It is scalable and can be customised to needs of a malware analyst and the algorithms can be adopted to other file types using file similarity matching hashes.

- Considering the different hashes as heuristic file attributes reduces the storage capacity required by the system making it very light and therefore not impacting system resources heavily.
- We combined tried and tested similarity matching hashes that are popular in the field of malware detection that are provided in almost all automated static analysis tools like Peframe and Virustotal.

In this chapter, we explore the different ways fuzzy hashing can be used to detect similarities in a file by investigating hashes of interest. Each hashing method produces independent but related interesting results which are presented herein. The application of two different methods of combining the hash values to improve the detection rates is presented. The results show that the detection rates are improved when evidence combination techniques are used. File and section Ssdeep hashing, PEHash and Imphash techniques are used to calculate the similarity of the Portable Executable files. The similarity in the files detected by the hashing functions is used as the Heuristic content similarity factor for the sample of the dataset used.

6.2 Design choice of Hashes in scope

The choice of hashes used in the design of this method relied on their current use in similarity matching and the various file sections used to calculate the hashes. Table 6.1 shows the reasons as to why the various hashes were chosen. PeHash design purpose argument is that it works to overcome polymorphic malware. Imphash is proven to classify malware based on the import table. Ssdeep Hash is used to calculate both the overall file similarity and the resource section. File Ssdeep hash is a very common hash used for similarity matching in common automated static analysis tools like Mastiff.

Hash Type	Reason
PeHash (PeH)	Overcoming Malware Obfuscation
Imphash (ImpH)	Classification by API
File Ssdeep Hash (FuzH)	Overall File similarity
Resource section Ssdeep Hash (ResFH)	PE Resource section file similarity.

Table 6.1 Argument for in scope Hashes

The resource section (.rsrc) of a PE file is known to contain the information about any names and or types of embedded resources [80]. By combining the various aspects of the file sample using the 4 various hashes, the overall achieved score is intended to represent the file's similarity to already known malware samples.

6.3 Modelling and implementing the proposed method design.

Our approach is presented in this section and we describe the model design and implementation of the different algorithms that form the proposed method. Table 6.2 shows the notations used in the design of the different algorithm with the approach which is divided into 6 different Step. While designing the methodology for this study, the PE format [24] and the work in [135] are revisited.

Notation	**Meaning**
DB	Database
ImpH	Imphash
PeH	PeHash
FuzH	File Ssdeep Hash
ResFH	Resource Section Ssdeep Hash
Xi	Set of elements of i attribute
MD5	MD5 sum
CFH(a, b)	Ssdeep Hash Comparison Function to detect similarity percentage of a and b hashes.
HashFlag_set (H)	The Hash flag setting function for H type of hash
$\boldsymbol{\pi_i}$(DB)	All the tuples in DB of attribute i
$\sigma_{a=b}(DB)$	Generalised selection of all tuples in DB where a= b
Pop_MASHDB	Populate Malware Sample Hash Database Function
MALHACompare(*f*)	Malware Hash Compare Function for file *f*
Mal (*f*)	Malicious Measure of File *f*
i	Hash i where: i = {1, 2,3,4} ⟺ { ImpH, PeH, FuzH, ResF}
FST	Fuzzy Logic Combinational Metric
CFM	Certainity Factor model Combinational Metric
TDR	True Detection Rate
CFi	Common Factor of Attribute i
SFi	Evidence Support Factor of Attribute i
FuzzyLogicSum	The Fuzzy Logic Algebraic sum function
MYCINSum	The Common Factor (MYCIN) Algebraic sum function

Table 6.2 The Algorithm Notations

By focusing on only the hashing aspect of static analysis, we investigated ways in which the different hashes that define PE fields of interest can be used to detect the malicious samples in the dataset.

6.3.1 Single File Hashing Study

This was initial study done on one clean file (*arp.exe*) found in Windows systems. The file hashes including the cryptographic, peHash, Ssdeep hash for all the sections and for the file were calculated. Then the file was edited using Radare [136] to write “?a” characters into the file which is then saved with a different name. The hashes for the new file were also calculated. The hashes from the two files; the original and the edited version were compared as shown in Figure 6.2 so that the differences would provide a baseline for the next phase of the study.

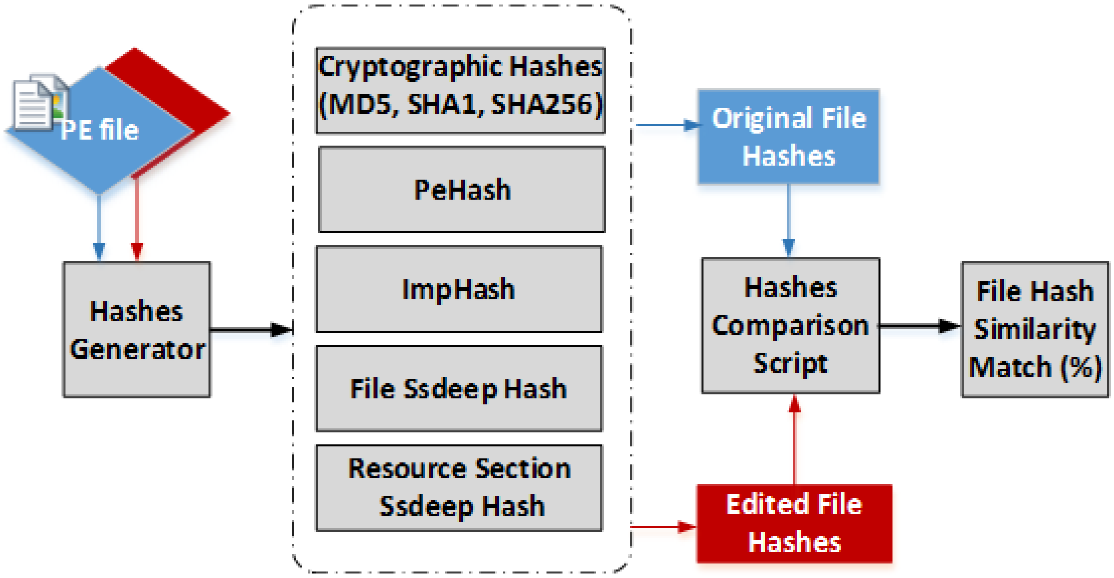


Figure 6.2 The single file Hashes Study

6.3.2 Collecting the Datasets

Using the files collected as discussed in section 2.5.2, each file was saved as its MD5 sum to ensure that there was no duplication of files in the dataset. The files were also clearly labelled clean or malicious and were kept in separate folders depending on the known status of the files. The malicious files dataset was split into 3 different subsets with the sets; A, Bm and Cm and the Clean files into 2 subsets; Bc and Cc which were used for different steps as shown in Table 6.3.

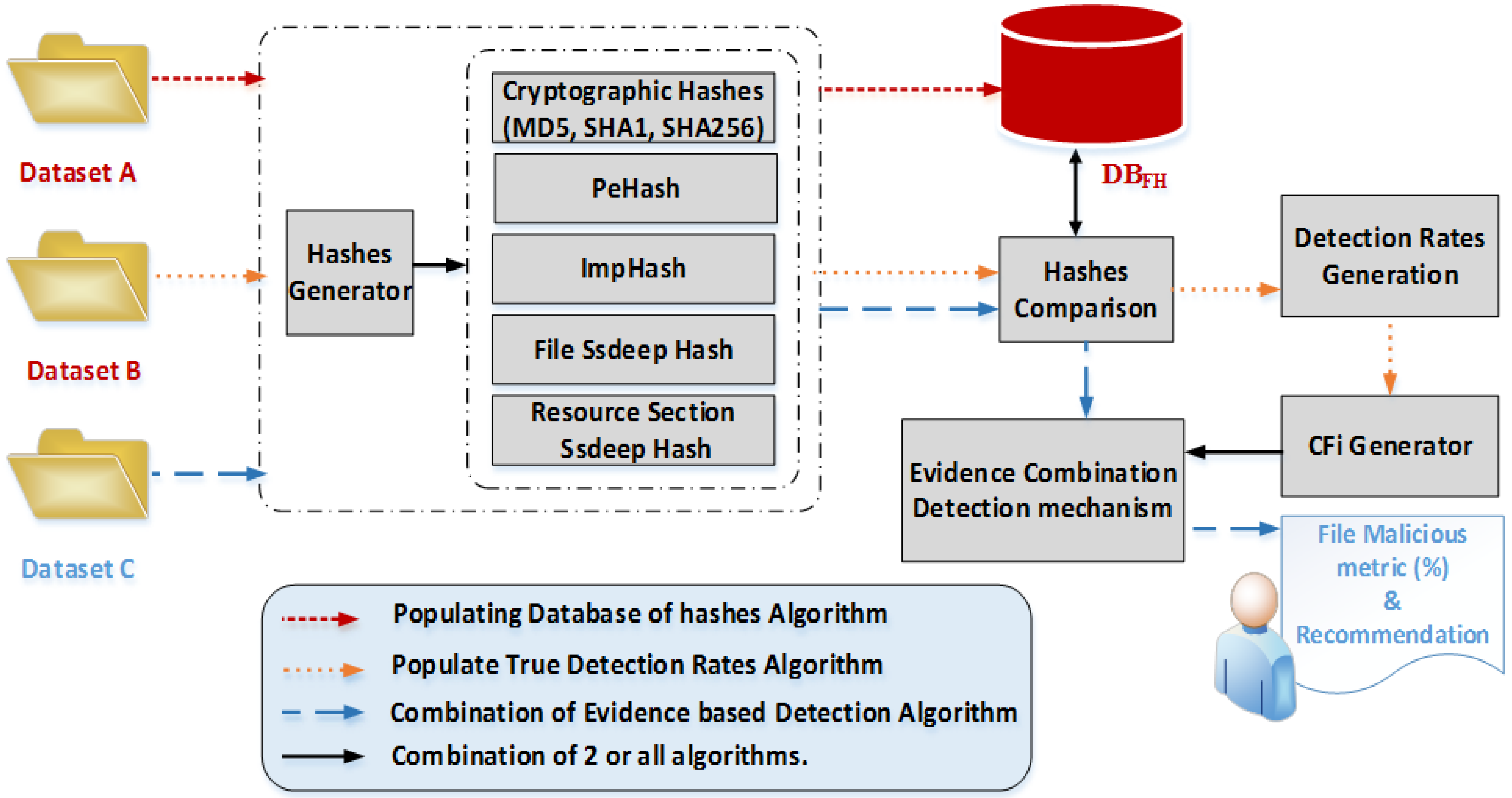

Dataset A
Dataset B
Dataset C
Hashes Generator
Cryptographic Hashes (MD5, SHA1, SHA256)
PeHash
ImpHash
File Ssdeep Hash
Resource Section Ssdeep Hash
DB$_{FH}$
Hashes Comparison
Detection Rates Generation
CFi Generator
Evidence Combination Detection mechanism
File Malicious metric (%) & Recommendation
Populating Database of hashes Algorithm
Populate True Detection Rates Algorithm
Combination of Evidence based Detection Algorithm
Combination of 2 or all algorithms.

Figure 6.3 The pictorial representation of the system

Dataset	Use in the system
A	Training phase.
B ⟵ {Bm, Bc}	Baseline-Creation phase and CFI generation.
C ⟵ {Cm, Cc}	Detection-method evaluation phase.

Table 6.3 Datasets Formation and their uses

6.3.3 Populating the Database of Hashes Signatures

The database of hashes (DBFH); the malicious hashes that are used as the initial signatures are populated with the randomly selected malware samples in dataset A using the process described by Algorithm 6.1.

Algorithm 6.1 Algorithm for Populating database of Hashes

```
Input: Malware Dataset $A$
Output: Flagged Hashes Database $DB_{FH}$
------------------------------------------------
 procedure: Pop_MSHDB
for malware $m$ in A do
        Extract the file hashes
                Hashes ($m$) ⟵ {MD5, ImpH, PeH, FuzH, ResF}
        If Hashes($m$) $\notin$ $DB_{FH}$ then
            add Hashes to $DB_{FH}$
end for
end procedure
```

The design of this framework also allows for this database to be updated should more signatures be identified that are not already saved in the database of hashes. The imphash, peHash, file Ssdeep hash and file resource section hash were computed and saved into an SQlite managed database with the file MD5 as the key identifier.

6.3.4 Hashes Similarity Based Criteria Factor Index (CFI) Formulation.

Once the training database was populated, the next phase involved determining how the individual hashes perform in terms of malware detection and how they can be combined to optimise the malware detection rate. Dataset B which comprises of both malicious files and clean files is used at this stage. The clean files are used as a control in this training stage. MD5 hash is ignored at this stage because the file MD5 gives us absolute certainty that the files are similar and thus the file would be malicious. Since we initially saved all the files in the datasets with their calculated MD5 as their identifiers, there was no possibility of duplication of a file in the dataset. This step was broken down into 2 sub-steps;

6.3.4.1 The individual performance of the hashes in relation to malware detection.

The initial stage of determining the individual performance of the hashes involved comparing the hashes calculated for files in dataset B against the DBFH. This was done by formulating the HashFlag_set where each of the 4 hashes had a specific position. The algorithm designed to set the HashFlag_Set fields is described by Algorithm 6.2 and Figure 6.4.

Algorithm 6.2 Algorithm for Hash Comparison

```
Input: PE file f and DBFH
Output: HashFlag_set

procedure: MALHACompare(f)
Extract
      Hashes(f) ⟵{ ImpH, PeH, FuzH, ResFH}
 return Hashes(f)

ImpH Comparison Phase

ImpHf ⟵ Hashes(f) ImpH
XImpH ⟵ $\pi_{MD5}\left(\sigma_{\text{"}ImpH=ImpH_f\text{"}}(DB_{FH})\right)$
if XImpH ≠ null then
    HashFlag_set (ImpH) = true
  else
    HashFlag_set (impH) = false
end if

PeH Comparison Phase

PeHf ⟵ Hashes(f) PeH
X PeH ⟵ $\pi_{MD5}\left(\sigma_{\text{"}PeH=PeH_f\text{"}}(DB_{FH})\right)$
if X PeH ≠ null then
     HashFlag_set (PeH) = true
  else
     HashFlag_set (PeH) = false
end if

FuzH Comparison Phase

FuzH f ⟵ Hashes(f) FuzH
XFuzH ⟵ $\pi_{md5,FuzH}(DB_{FH})$
if Max(CFH (FuzH,∀x1 ,({x0,x1}∈ XFuzH ))  > 0 then
    HashFlag_set (FuzH) = true
  else
    HashFlag_set (FuzH) = false
end if

ResFH Comparison Phase
ResFH f ⟵ Hashes(f) ResFH
XResFH ⟵ $\pi_{md5,ResFH}\ (DB_{FH})$
if Max(CFH (ResFH,∀x1 ,({x0,x1}∈ XResFH )) > 0 then
        HashFlag_set (ResFH) = true
  else
        HashFlag_set (ResFH) = false
end if
return HashFlag_set
end procedure
```

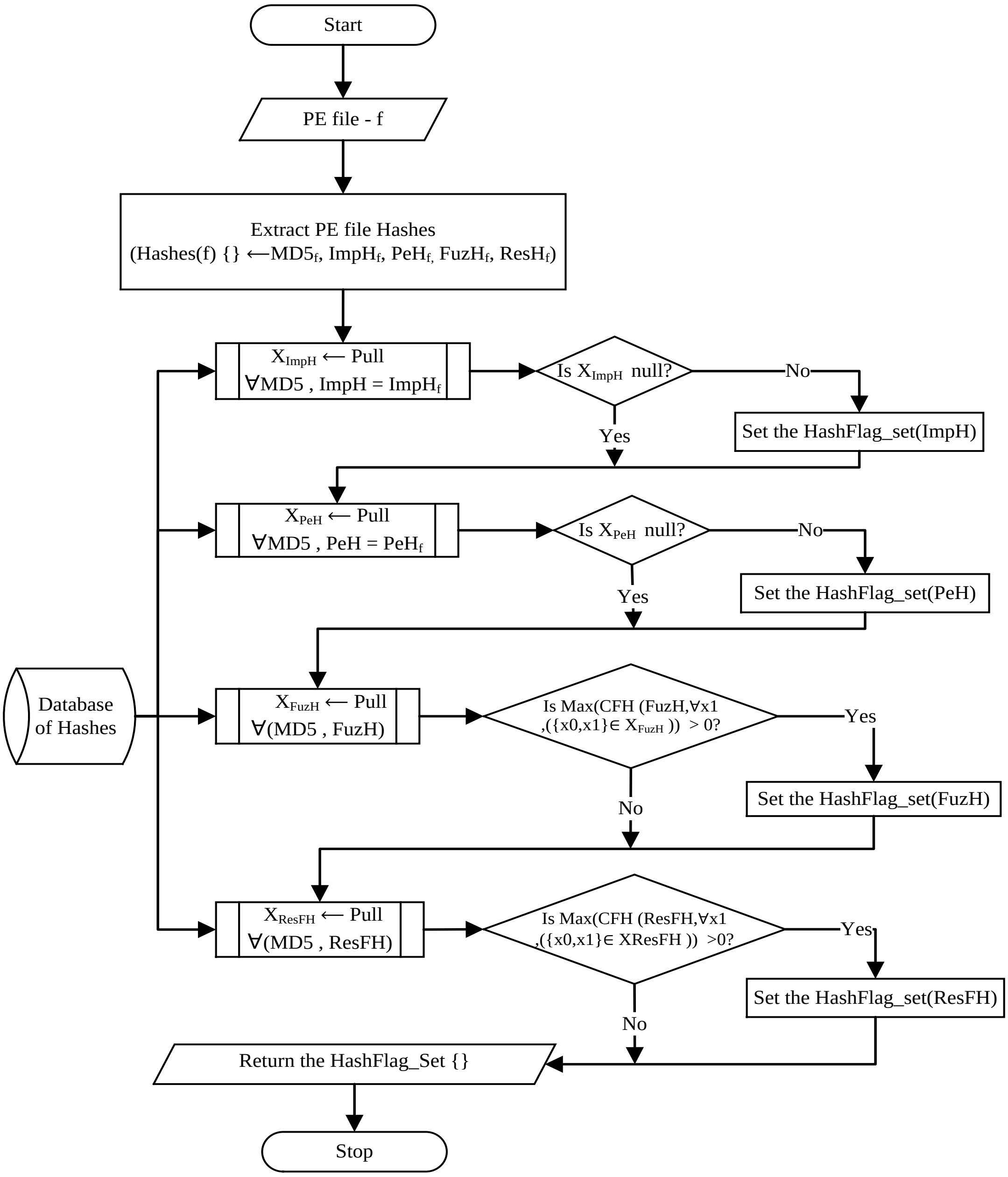

Start
PE file - f
Extract PE file Hashes
(Hashes(f) {} ⟵MD5f, ImpHf, PeHf, FuzHf, ResHf)
XImpH ⟵ Pull
∀MD5 , ImpH = ImpHf
Is XImpH null?
No
Yes
Set the HashFlag_set(ImpH)
XPeH ⟵ Pull
∀MD5 , PeH = PeHf
Is XPeH null?
No
Yes
Set the HashFlag_set(PeH)
Database of Hashes
XFuzH ⟵ Pull
∀(MD5 , FuzH)
Is Max(CFH (FuzH,∀x1 ,({x0,x1}∈ XFuzH)) > 0?
Yes
No
Set the HashFlag_set(FuzH)
XResFH ⟵ Pull
∀(MD5 , ResFH)
Is Max(CFH (ResFH,∀x1 ,({x0,x1}∈ XResFH)) >0?
Yes
No
Set the HashFlag_set(ResFH)
Return the HashFlag_Set {}
Stop

Figure 6.4 Flow chart for Algorithm used to set the HashFlags

For each file in Dataset B, the 5 respective hashes are computed and 4 different queries were run against the database. Each query returns a set of tuples;

$X_{Hi} \leftarrow \{md5, H_i\}$ (6.1)

Where; H_i = {PeH, ImpH, FuzH, ResFH}

During the comparison of PeHash and Imphash, only the hashes similar to the calculated hash were pulled from the database. If the set X_{hi} was Ø (null), the HashFlag_set position corresponding to the hash of type i was not set. It was set otherwise. While for resource Ssdeep hash and file Ssdeep hash, all the hashes were pulled from the database and an Ssdeep similarity match was done for the file hashes and the respective database populated hashes.

Algorithm 6.3 Algorithm for populating Detection Rates

```
Input: ds ⟵B , $DB_{FH}$
Output: DetectionRates
---
Overall Hash Based Detection Rate Phase
---
procedure: HaBaDR
for file (f) in B do
   $H\_flagset_f$
for i = 1 → 4                    ▻Loop through the hashes set ids
     if f ∈ B then
         if $H\_flagset_{fi}$ then
             $TP_i$ = +1
         else
             $FN_i$ = +1
        end if
      end if
      if f ∈ Clds then
         if $H\_flagset_{fi}$ then
             $FP_i$ = +1
         else
             $TN_i$ = +1
          end if
        end if
      Update $DetectionRates_i$ ⟵{← {$TP_i$, $FN_i$, $FP_i$, $TN_i$}
   end for
return DetectionRates
end procedure
```

If the maximum similarity percentage calculated was greater than zero, the HashFlag_set position corresponding to the hash of type i was set. It was not set otherwise. Each file corresponds to one set of HashFlag_set. The second phase entailed populating total count of instances where the flag position of the hash is set. For the malicious subset (True Positive –TP if set. False Negative –FN if not set), Clean subset (True Negative – TN if not set. False Positive – FP if set) as shown in Algorithm 6.3.

6.3.4.2 Calculate the CFI of all the individual hashes.

The detection rates obtained in the above sub-step are used to calculate the CFI of each hash which is used as a measure of belief for each hash detection factor. True detection rates are used as a measure of the hash methodology accuracy to minimise the error incorporated in the support factors. To make the detection rates compatible with the combinational theories, the positive detection rates are normalised to probabilities that add up to 1. The normalised detection rates take the form of the degree of belief in the uniform range [0, 1]. Simple Additive weighting [137] is applied to the detection rates so that the degree of belief/ Criteria Factor Index (CFI) for each Hash method is defined as:

$$\mathrm{CFI_a} = \left[\sum_{n=1}^{4} TDR_n\right]^{-1} * \mathrm{TDR_a} \qquad (6.2)$$

$$\text{Where } \mathrm{TDR_a} = \frac{TP_a + TN_a}{Number\_in\_DatasetB}$$

These CFI values are used as the expert factors for the respective hashing techniques to support the hypothesis that the file is malicious.

6.3.5 Application of Evidence Combination Theory Approach

The CFI values obtained in step 6.3.4 are inputs to Algorithm 6.4 for the combinational approach application to the analysis as further shown in Figure 6.5. The MD5 comparison phase is used as a known malware filter for the designed method. This phase is a redundancy step introduced to avoid replication of the malware samples in the evaluation experiment. The Hashes comparison phase uses the file calculated hashes and compares them against DBFH. The query in equation 6.1 is used in this phase too. The support factors for the hashes are computed from the results obtained from the respective queries. For Pehash and Imphash, if the resulted set is not null, then the corresponding Support Factor (SF) is equivalent to the CFI of the respective hash. Otherwise the hash's SF is set to zero. For Resource Section Ssdeep hash and file Ssdeep hash, the corresponding SF is equivalent to the CFI multiplied with the maximum similarity percentage achieved from comparing the file and the hashes in the database.

The Calculated SF values of the various hashes are combined using the evidence combinational models to get the algebraic sum for the overall hypothesis. The result is feed into the TLBSA (Traffic Light Based Scoring Assessor) which returns the calculated quantitative file status measure and a recommendation for the user.

6.3.6 Generation of the Traffic Light Based Scoring Assessor (TLBSA) Thresholds.

The calculated malware detection rate evaluation values provide a way of identifying how well the designed method works. However, to ensure that there is as high true positive detection vs very low false positives trade-off, we need to design thresholds. Therefore, the resultant percentages from the combined hashing technique are compared to add an overall “Traffic Light Based Scoring Assessor” (TLBSA) that evaluates the score attached to the file to give the user a recommendation based Table 6.4

TLBSA Colours	**Deduced Malicious intent of the file**	**System Recommendation**
Red	Definitely malicious	Do not Install
Amber	Medium Suspicion	Highly encouraged to submit it for further analysis
Green	Low Suspicion	May submit it for further analysis

Table 6.4 The TLBSA Colour definitions

The TLBSA threshold percentages are defined by best effort True positive- false positive trade-off. Since the system does not completely guarantee that the file is safe, the final decision is left to the user. The system however, ensures that the user is informed with the file malicious score and the system recommendation so that the human factor risk is controlled.

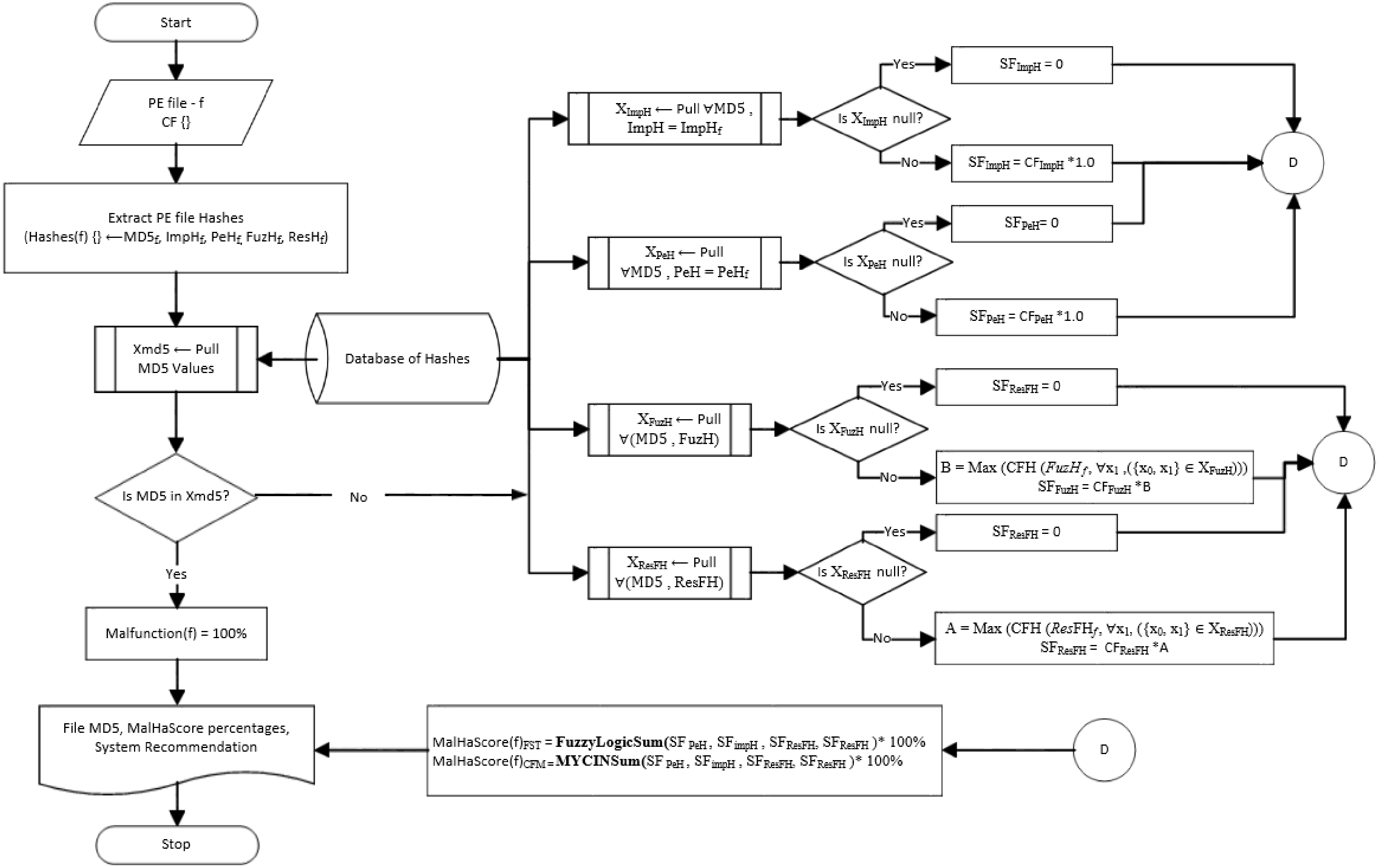

Start
PE file - f
CF {}
Extract PE file Hashes
(Hashes(f) {} ⟵MD5$_f$, ImpH$_f$, PeH$_f$, FuzH$_f$, ResH$_f$)
Xmd5 ⟵ Pull
MD5 Values
Database of Hashes
Is MD5 in Xmd5?
No
Yes
Malfunction(f) = 100%
File MD5, MalHaScore percentages,
System Recommendation
Stop
X_{ImpH} ⟵ Pull ∀MD5 ,
ImpH = ImpH$_f$
Is X_{ImpH} null?
Yes
No
SF_{ImpH} = 0
SF_{ImpH} = CF_{ImpH} *1.0
D
X_{PeH} ⟵ Pull
∀MD5 , PeH = PeH$_f$
Is X_{PeH} null?
Yes
No
SF_{PeH}= 0
SF_{PeH} = CF_{PeH} *1.0
X_{FuzH} ⟵ Pull
∀(MD5 , FuzH)
Is X_{FuzH} null?
Yes
No
SF_{ResFH} = 0
B = Max (CFH ($FuzH_f$, ∀x_1 ,({x_0, x_1} ∈ X_{FuzH})))
SF_{FuzH} = CF_{FuzH} *B
D
X_{ResFH} ⟵ Pull
∀(MD5 , ResFH)
Is X_{ResFH} null?
Yes
No
SF_{ResFH} = 0
A = Max (CFH ($ResFH_f$, ∀x_1, ({x_0, x_1} ∈ X_{ResFH})))
SF_{ResFH} = CF_{ResFH} *A
MalHaScore(f)$_{FST}$ = **FuzzyLogicSum**(SF_{PeH} , SF_{impH} , SF_{ResFH}, SF_{ResFH})* 100%
MalHaScore(f)$_{CFM}$ =**MYCINSum**(SF_{PeH} , SF_{impH} , SF_{ResFH}, SF_{ResFH})* 100%
D

Figure 6.5. Flow chart showing the calculation of the MalHaScore for file samples.

Algorithm 6.4 Combination of Hashes Based Detection Mechanism

```
Input: PE file f , CF and DBFH
Output:  Mal(f)
Extract file Hashes Phase
procedure:HaBCoMalD(f)
Extract
          Hashes(f) ⟵{MD5, ImpH, PeH, FuzH, ResFH}
 return Hashes(f)
MD5 Comparison Phase
MD5f ⟵ Hashes(f) MD5
 XMD5 ⟵ $\pi_{\mathrm{MD5}} \left(\sigma_{\mathrm{"MD5=MD5_f"}} (\mathrm{DB_{FH}})\right)$
 if XMD5 ≠ null then
        Mal(f) = 100%
        end procedure
 else
      goto HashComp
end if
return Mal(f)     ⊳provide a system recommended action for the user
end procedure
Hashes Comparison Phase
HashComp:
    ImpHf ⟵ Hashes(f) ImpH
    PeHf ⟵ Hashes(f) PeH
      FuzH f ⟵ Hashes(f) FuzH
      ResFH f ⟵ Hashes(f) ResFH
      XImpH ⟵ $\pi_{MD5} \left(\sigma_{"ImpH=ImpH_f"}(DB_{FH})\right)$
    if XImpH ≠ null then
        SFimpH = CF impH  * 1.0
    else
        SFimpH = 0
     end if

     X PeH ⟵ $\pi_{MD5} \left(\sigma_{"PeH=PeH_f"} (DB_{FH})\right)$
     if X PeH ≠ null then
        SF PeH = CF PeH  * 1.0
    else
        SF PeH = 0
     end if

XFuzH ⟵ $\pi_{md5,FuzH} (DB_{FH})$
    if X FuzH ≠ null then
        SFFuzH = CFFuzH  * (Max (CFH (FuzH f , ∀x1 ,({x0, x1} ∈ XFuzH))))
    else
        SFResFH = 0
     end if

XResFH ⟵ $\pi_{md5,ResFH} (DB_{FH})$
   if XResFH ≠ null then
```

```
        SFResFH =CFResFH * (Max (CFH (ResFHf , ∀x1, ({x0, x1} ∈ XResFH))))
    else
        SF ResFH = 0
    end if

MalHaScore(f)FST  = FuzzyLogicSum(SF PeH , SFimpH , SFResFH, SFResFH )* 100%
MalHaScore(f)CFM  = MYCINSum(SF PeH , SFimpH , SFResFH, SFResFH )* 100%
end procedure
```

6.4 Dataset preparation and test environment.

We initially test the method on smaller dataset containing a total of 22988 files and these were prepared as seen in Table 6.5.

Dataset	A	B	C	Total Files
Malicious files	7124	7355	7269	21748
Clean files		623	617	1240

Table 6.5 The Initial (St.1) Study Dataset

We then extended our study after collecting more files and the final total dataset was prepared as shown in Table 6.6 for the different required sub-datasets.

Dataset	A	B	C	Total Files
Malicious files	34224	32844	37460	104528
Clean files		698	940	1638

Table 6.6 The Final (St.2) Experiment Dataset

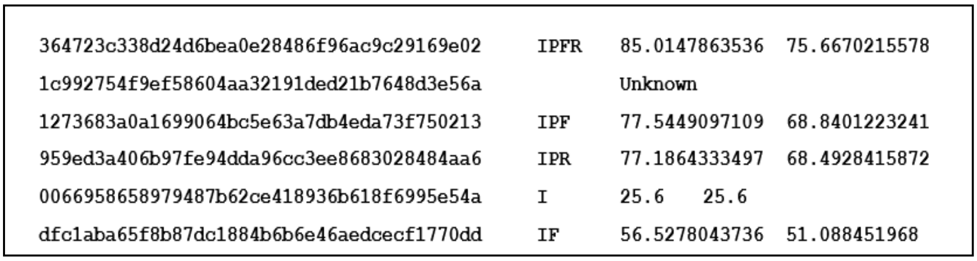
The test environment uses the tools specified in Table 6.7. We use a Linux-based operating system because the files under analysis are .exe files. The algorithms are scripted in python and some functions from Peframe and Pefile are extended to compute the file hashes under consideration; Ssdeep, PeHash, and Imphash. The algorithms are python scripts and the database of Hashes is managed using SQLite.

Tool	Specifications/ Details
Computer	Dell T1700, CPU – Intel Xeon@ 3.1GHz, RAM 32GB. Hard Disk – 500GB
Machine OS	Linux Mint 17.1 (#64 – Ubuntu SMP)
Static Analysis tool	Study specific Static Analysis Tool – calculates the Ssdeep, Resource Section Ssdeep hash, Pehash, and Imphash
Data management tools	SQLite Studio version 3.0.6. Python IDLE version 2.7.9

Table 6.7 Test Bench Specifications

6.5 Experimentation results and analysis

The result log is formulated to be interpreted as follows: The **first column** is the filename. The **second column** the calculated hashes flags of the file that found to be similar to those already existing in the database where I – Imphash, P – PeHash, F – File Fuzzy hash and R – Resource section fuzzy hash. The **Third column** gives either *one value*; "Unknown" if there is no database hash matched or *two values*; Fuzzy logic method score and Common factor method score.



```
364723c338d24d6bea0e28486f96ac9c29169e02    IPFR    85.0147863536   75.6670215578
1c992754f9ef58604aa32191ded21b7648d3e56a            Unknown
1273683a0a1699064bc5e63a7db4eda73f750213    IPF     77.5449097109   68.8401223241
959ed3a406b97fe94dda96cc3ee8683028484aa6    IPR     77.1864333497   68.4928415872
0066958658979487b62ce418936b618f6995e54a    I       25.6    25.6
dfc1aba65f8b87dc1884b6b6e46aedcecf1770dd    IF      56.5278043736   51.088451968
```

Figure 6.6 Result Log showing the Analysis results of 6 different files

6.5.1 Observations from the single file study test

The results from the single file test shown in Table 6.8 validate the interest in this study for similarity matching. Since obfuscated malware tend to reorganise their code to evade detection, these changes can be reflected in similarity matches. The results show that a small change in a file can greatly affect the results of similarity matching in some hashes while having very little to no effect in others. This justifies the investigations further carried out in this study on the hashes that return a similarity score greater than 0%.

6.5.2 Malware detection performance of the individual in-scope Hashes and calculation of the CFI.

In the second phase of the study, the main aim was to design a method to calculate the CFI metrics of the hashing techniques used. Dataset B is used in this step not only to set the CF metrics for the framework, the results obtained are also used to evaluate detection rates of the different hashing techniques as shown in. The counts of the hash fields in HashFlags_set for each achieved similarity match are tallied to obtain the confusion matrix performance metrics for all the hashes. The graphs in Figure 6.7 show how various hashes stack up against each other in detection rates.

Hash Type	**Original File Value**	**Edited File Value**	**Score(%)**
MD5	33f9b0e02d9d93f920605d02fb53f3fd	accd6591b8b8dad5f7f1470c90971e75	0
SHA1	4a22e401ad5adb7b3de8f819e86d8461d764d195	06b98e35c1f92f844b57376ee467ee977cc074bd	0
SHA256	1f4c090dfa389b3c6b16eb42299fb815f24efac7ca541bb60821e3da0131b8f6	bd4f056223439e83f2fffbe3c463e178da8465fabeb51243c04a3d2922de8fa2	0
Ssdeep-File	384:5u3Smmq6aYaBpYFAfjhXrToHWS4mW4sme9V:Avmq6affYFAfjhr8sgE	384:5u3Smmq6aYaBpYFmfjhXrToHWS4mW4sme9V:Avmq6affYFmfjhr8sgE	99
PeHash	5515f8e47661c7e170aee948cca7c8dc6198c08f	5515f8e47661c7e170aee948cca7c8dc6198c08f	100
Imphash	880bb6799a6e1a5ff7b4f022ff4003a9	880bb6799a6e1a5ff7b4f022ff4003a9	100
Ssdeep - Resources	96:8EWS1pEmWwOh/VsBgtAb88caS5Ur9I5fa9VWPBMXsmrC9V:NWS4mWNJXCu6Xsme9V	96:8EWS1pEmWwOh/VsBgtAb88caS5Ur9I5fa9VWPBMXsmrC9V:NWS4mWNJXCu6Xsme9V	100

Table 6.8 Comparison of Hashes from the Single File Study

Malware detection performance of the individual in-scope Hashes and calculation of the CFI													
	Recall (%)		**PPV (%)**		**ACC (%)**		**F-score (%)**	**Detection Rates**				**CFI (%)**	
								TRUE (%)		**FALSE (%)**			
	St. 1	*St. 2*	*St. 1*	*St. 2*	*St. 1*	*St. 2*	*St. 2*	*St. 1*	*St. 2*	*St. 1*	*St. 2*	*St. 1*	*St. 2*
ImpH	97.2	85.6	99.4	93.3	96.9	89.7	89.3	96.9	85.7	3.1	14.3	25.6	27.0
PeH	96	82.8	100	100	96.4	91.4	90.6	96.4	83.1	3.7	16.9	25.4	26.2
FuzH	95.21	76.2	100	100	95.6	88.1	86.5	95.6	76.7	4.4	23.3	25.3	24.1
ResFH	88.97	71.7	99.9	99	89.9	85.5	83.2	89.9	72.3	10.2	27.7	23.7	22.7

Comparative analysis of the individual hashes against the proposed Combined Hashing Methods.					**Comparative analysis of the performance of the proposed method after application of the TLBSA.**				
	Detection Rates					**PPV (%)**	**Recall (%)**	**Acc (%)**	**F-Score (%)**
	TRUE (%)		**FALSE (%)**						
	St. 1	*St. 2*	*St. 1*	*St. 2*					
ImpH	97.2	84.6	2.8	15.4	**ImpH**	74.2	84.9	77.7	79.2
PeH	96.3	82.7	3.7	17.3	**PeH**	99.7	82.3	91	90.2
FuzH	96.1	75.6	3.9	24.4	**FuzH**	79.9	75.5	78.2	77.6
ResFH	90.4	71	9.6	29	**ResFH**	60.5	71.5	62.4	65.5
Combined Hashing Methodology (CHM)	98.2	93.2	1.8	6.8	**FL_GTP (< 25%)**	99.2	92.2	91.6	95.5
					FL_ATP (≥ 75%)	99.9	70.5	71.2	82.7
					CF_GTP (< 25%)	99.2	92.1	91.6	95.5
					CF_ATP (≥ 70%)	100	69.8	70.4	82.1

Table 6.9 Experimentation Calculated Metrics

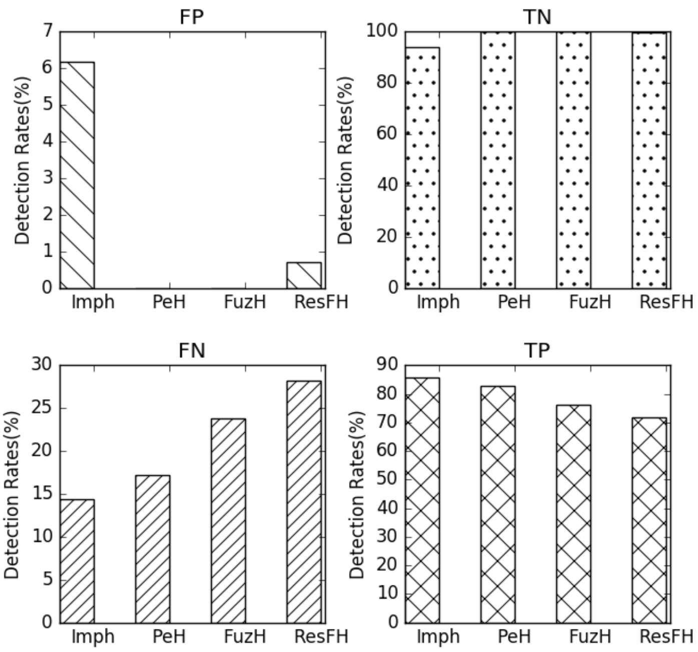


Figure 6.7 The Hashes Detection Rates using Dataset B for the final study

6.5.2.1 Comparative analysis of the individual hashes against the proposed Combined Hashing Methods.

Data C is used to calculate the overall file malicious percentage to validate the proposed framework. The output logs of the large-scale detection are analysed with the confusion. The results achieved for the proposed method are compared against the results achieved for each individual hash in Figure 6.8 and Figure 6.9 in the two studies. To further analyse how the methodology scores the clean and malicious files, there was need to investigate the different curves defined by the clean and malicious file scores for the two proposed methods. Figure 6.10 and Figure 6.11 show the file scoring area curves of each adopted method for the 2 studies which shows that most of the malicious files score higher than the clean files. The repeatability of the experiment using various datasets and observing similar trends in the end result file scores provides an argument for the proposed approach.

6.5.2.2 Performance of the proposed method after application of the TLBSA.

Since the aim of this study is to devise an optimum malware detection methodology, we investigate the true positive and false negative trade-off of the two proposed evidence combination methods. The following comparisons are made based on the results from the final study. The true positive rate against false positive rate curves of the two proposed methods are compared in Figure 6.12 and the precision against recall plots are compared in Figure 6.13. Since F-score is known to be a weighted measure of precision and recall, it is plotted against the file score threshold in Figure 6.14.

In Table 6.9, we compare the two proposed methods to determine the best TLBSA threshold percentages.

Where FL – Fuzzy Logic Method.
CF – Common Factor Model Method.
GTP – Green Threshold Percentage.
ATP – Amber Threshold Percentage.

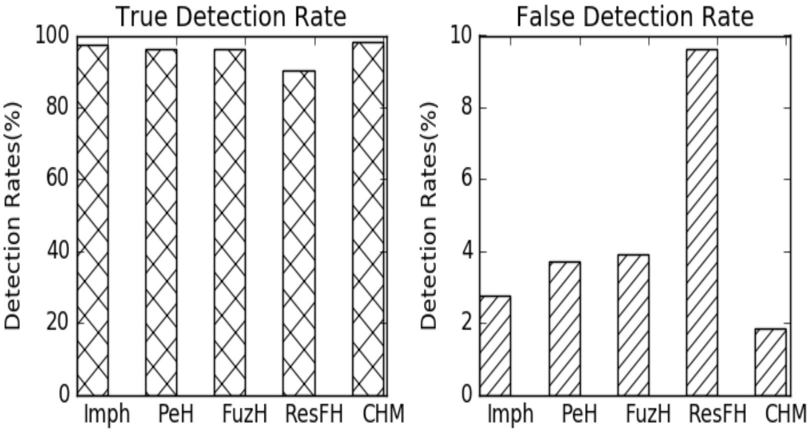


Figure 6.8 Comparing the individual hashes against the proposed combined method for the initial study

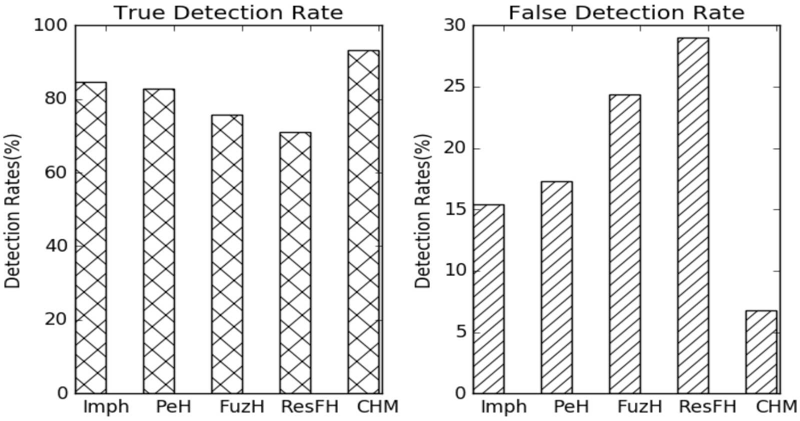


Figure 6.9 Comparing the individual hashes against the proposed combined method for the final study

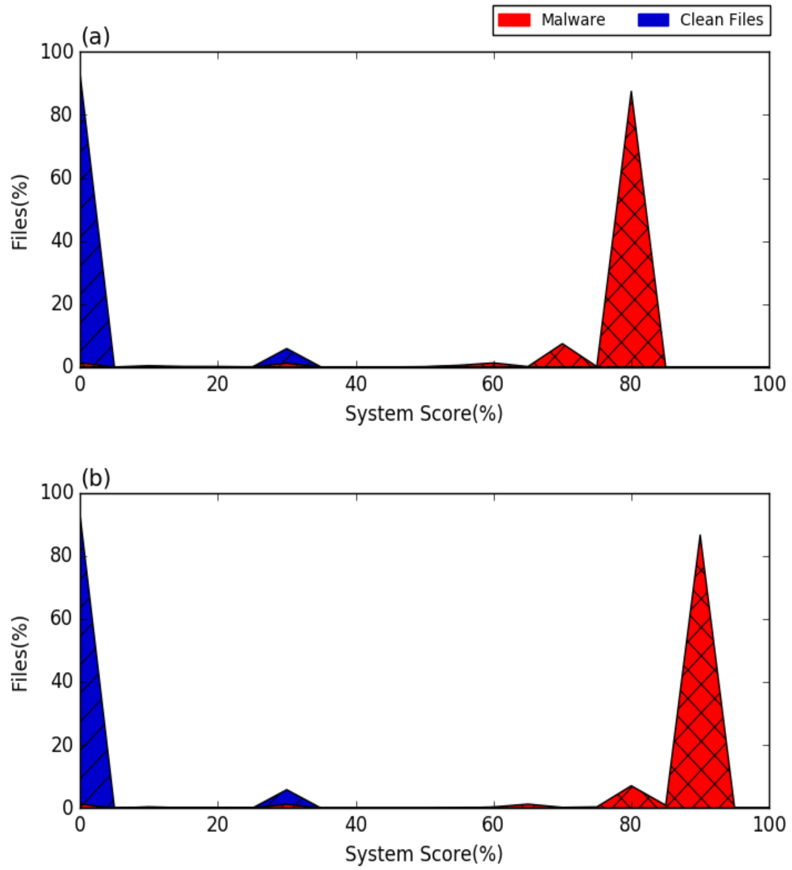


Figure 6.10 The Combined Hash Score Clean and Malware file Area curves (a) Common Factor method and (b) Fuzzy Logic Method from the initial study

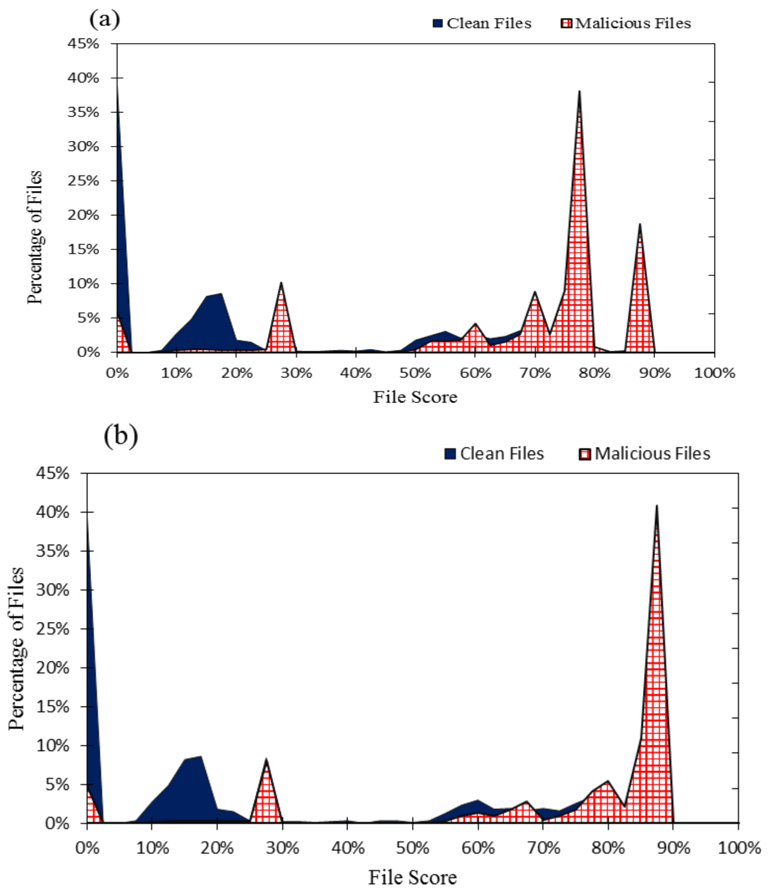


Figure 6.11 The Combined Hash Score Clean and Malware file Area curves (a) Common Factor method and (b) Fuzzy Logic Method from the final study

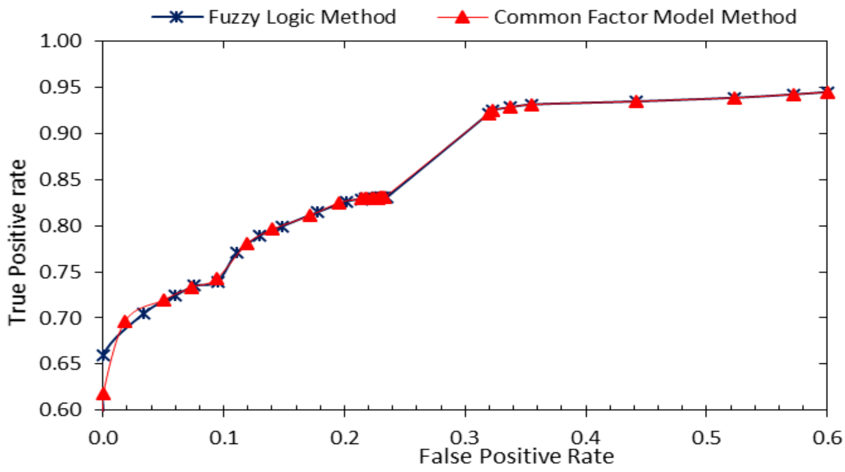


Figure 6.12 True positive rate vs False positive rate curves for Evidence Combination methods

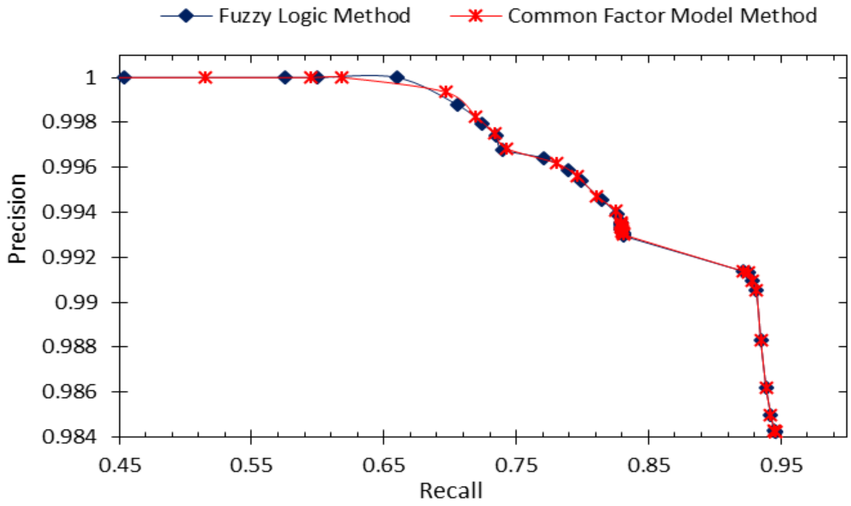


Figure 6.13 Precision- Recall curve of the proposed evidence combination methods

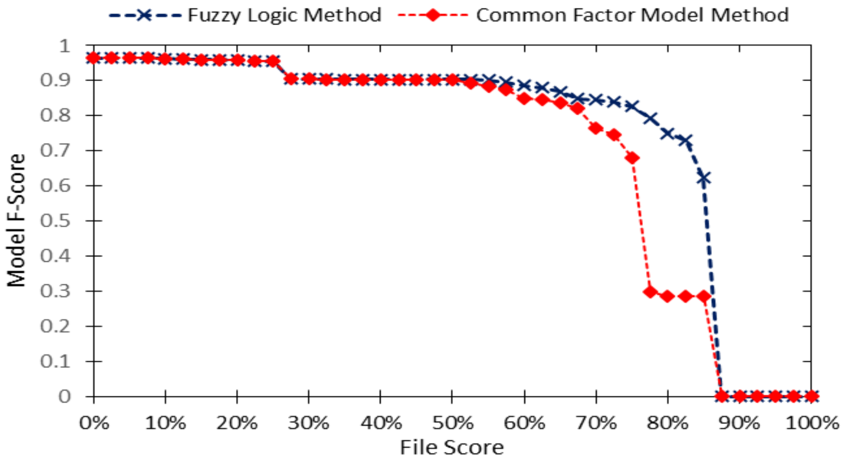


Figure 6.14 Model F-score for the different score percentage threshold

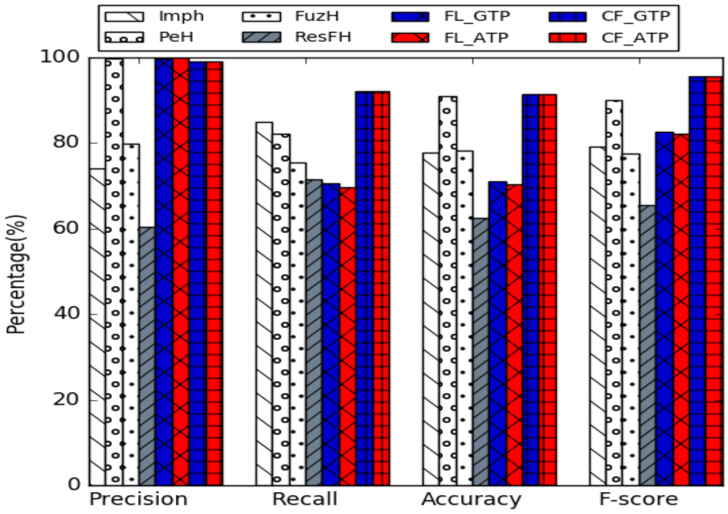


Figure 6.15 Recall, Precision, Accuracy and F-score Comparison for the proposed methodology percentage thresholds

Further analysis of the detection ratios of the malware types in the datasets is done in Figure 6.16 which shows that proposed method performs well for specific malware types. Given the deviation between the detection achieved in this method and the one in the earlier proposed method provides an argument for situations where they can be used in combination.

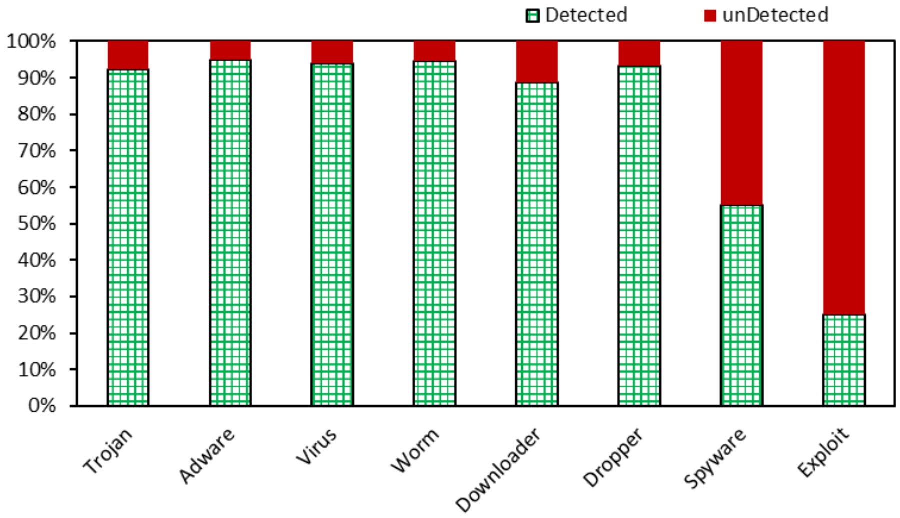


Figure 6.16 Malware type detection ratios for the dataset used.

6.6 Results Discussion of the proposed Evidence Combination of Hashes methods for malware detection.

This study proposes and evaluates two methods for combining the individual hashes results for malware detection. The initial study of the single file analysis provides the foundational argument for this investigation. Having created a small change in the contents of a clean PE file, the fact that the hashes return different similarity results substantiates the premise that similarity matching using hashing can overcome obfuscated malware. In the second phase of the study, the individual hash results achieved further support this argument. The introduced resource section hash matching gives the second precision value in the 4 algorithms. From the comparison in this phase of the study, PeHash is the best performing of the 4 hashes. In the third phase of the study, normalising the log so that the files which return a no similarity match score have a 0% score. This allows for easier analysis of the results as earlier mentioned. Analysis of the logs to validate the Combined hashing methodology results into achieving an overall false detection rate of 6.8% and a true detection rate of 93.2 %; the best performance values in comparison to the results achieved by the individual hashing algorithms. This shows that the proposed methodology out performs all the individual methods. However, reviewing the true positive to false positive detection trade-off for this proposed method shows that this technique is susceptible to very high false positive of 60% this required evaluation of the model to achieve a better trade off. We therefore introduce the TLBSA assessor at this stage so that the file status has 3 zones; Green zone where the file is less likely to be malicious, Amber zone where the file is likely to be malicious and the Red zone where the file is most likely to be malicious. Creating the percentage thresholds for these zones in this method required the analysis of the two proposed and evaluated methods. The initial analysis at this stage involved evaluating the analysed dataset clean file scores vs malicious file scores for the two evidence combination methods. Both curves in Figure 6.11 show that 83% of the malicious files obtain a malicious score above 50% while 78% of the clean files have a malicious score less than 50%. We then analyse the two proposed performance results against each other to gain an understanding of the better performing technique.

The main aim for this analysis is to increase the overall true positive detection rates while decreasing false positive detection rates. Therefore, Figure 6.12 compares how the two methods measure up against each other and the curves show that Fuzzy Logic provides a better performance of 65% True Positive Rate (TPR) to 0% False Positive Rate (FPR) vs 60% TPR to 0% FPR of the Common

Factor method initially and then both methods follow the same trend thereafter. For further evaluation, Figure 6.13 evaluates the methods' precision to recall and to pinpoint the thresholds that would work best for this method, Figure 6.14 compares the F-score achieved for both methods across the different malicious file scores of the tested dataset. With the thresholds obtained as discussed earlier and shown in Table 6.9, we evaluate how well our method works against the individual hashing algorithms in Figure 6.15. The Amber threshold percentage (ATP) which marks the percentage above which the file is said to be in the red zone out performs all the individual hash techniques. However, since this percentage creates a very low TPR of 70% for the Fuzzy logic method and 62% for the Common Factor Model method, there is need to analyse the needed Green threshold percentage (GTP). GTP marks the percentage below which the file is said to be in the green zone and above which it is said to be in the amber zone. It creates a much-needed raise in TPR of 92% for both the proposed techniques. With this integrated design to detecting malicious files based on the Traffic Light Based Scoring Assessor (TLBSA), the number of files detected is higher. The threshold percentages that allow optimum trade-offs and enable the system provide a user with information that helps protect their system with an accuracy of at least 92% achieved in this study.

6.7 Chapter Summary.

This chapter introduces a heuristic malware detection approach that successfully combines the hashing results to provide an overall best performing recall of 92%, a precision of 99%, an accuracy of 91% and an F-score of 96% which are higher than the detection rates for the independent hashes. The method can also be open sourced to allow for customised manipulation and extension of the algorithms by malware analysts. Since the technique uses static analysis, it is safe against malware that evade sandboxes and dynamic analysis environments. As a way of controlling the risk introduced by the human factor in security system, we present a quantitative measure for the malicious status of the file. The limitation of this system is that one needs a starting baseline of the database of malicious file hashes. However, the use of a simple database working with light weight analysis scripts reduces the impact of this limitation. In this study, the database of hashes created with 34224 rows of hash signatures only occupied

18MB. The methodology designed also allows for easy update of the signatures so the built model's performance increases with the number of hashes signatures. The results achieved in this study show that the proposed methodology provides a way of building an efficient integrated malware detection system. Our system was designed using light weight tools which makes it fast. In malware detection, the objective is to build a filter like system and with this work, we introduce a way of detecting malicious PE type files without the need for dynamic analysis unless the result is inconclusive.

7.1 Findings

This work proposes two approaches of building a heuristic feature based framework that quantifies the malicious intent of a file. Our approaches are designed to be light, fast, and efficient and at the end attach a numerical score to how malicious a file is deemed to be to the system. This numeric values is intended to ensure that even a standard computer user can decide with high confidence levels about the effect the file might have on the system. We limit our scope to Portable Executable files to design the schemes and use multiple attribute based decision making and evidence combination theory. We also limit the amount of storage resources required by the signatures by using similarity hashes that are calculated values and probabilistic scoring heuristic anomalies so that the approaches are not resource heavy.

Our MalScore study introduces a technique for scoring file feature anomalies by attaching a probability score to identified PE heuristic anomalies. The designed approach achieves a true positive detection rate of 97.6% to a false positive detection rate of 0.6% trade-off at a threshold score of 45%

The MalHaScore study successfully introduces a way of combining the similarity hashing results for a more efficient malware heuristic detection approach that provides an overall best performing true detection rate of 93.2%, a false detection rate of 6.8%. Recall of 92%, a precision of 99%, an accuracy of 91% and an F-score of 96% which are higher than the detection rates for the independent similarity hashes.

These approaches can be further customised based on the file type and extended based on the requirements of the malware analyst.

7.2 Comparison of the two proposed approaches

In this work, we designed two approaches that attach scores to PE files to measure the malicious intent of the file. We took a two-dimension approach where we designed methods that can be used by a standard computer user and a malware analyst by focusing on reducing the analysis time and providing a metric measure for file maliciousness.

The two approaches perform differently and have various strengths over each other as compared in Table 7.1 and Table 7.2 and can therefore, also be used in combination.

Feature	**MalScore**	**MalHaScore**
Average PE File detection time	6.1 Seconds	4.3 Seconds
Signature Storage Usage	15KB (This is for the 25 top features)	18MB (Varies based on the number of hashes stored)
Detection Rates	TPR– 97.6%, FPR – 0.6%	TPR- 93.2%, FPR - 6.8%
Recommended for	Standard Computer User	Malware Analyst

Table 7.1 Comparing the two proposed methods

The comparisons show that while MalHaScore is faster than MalScore, it requires more signature storage as MalHaScore relies on having a populated database of Hashes as its signature. MalScore out performs MalHaScore in terms of the detection rates. Therefore, we recommend that MalScore is more suited for the standard computer end user while MalHaScore can be used by an analyst since there is also an option for the analyst to revisit the similar files hashes that lead to the file score. The malware type detection rates show that MalScore outperforms MalHaScore in all the categories expect the Downloader malware type.

	MalScore		**MalHaScore**	
	Detected	**unDetected**	**Detected**	**unDetected**
Trojan	97.8%	2.2%	92.3%	7.7%
Adware	98.5%	1.5%	94.9%	5.1%
Worm	97.9%	2.1%	94.6%	5.4%
Downloader	75.5%	24.5%	88.7%	11.3%
Dropper	96.7%	3.3%	93.3%	6.7%
Virus	92.7%	7.3%	93.9%	6.1%
Spyware	81.8%	18.2%	55.0%	45.0%
Exploit	85.7%	14.3%	25.0%	75.0%

Table 7.2 Comparing the malware type detection ratios for the two proposed methods

7.3 Limitations and Challenges

Although the framework approaches achieve efficient malware detection rates, it has some limitations. During the design phase of this study, we limited our study to only Portable Executable files and therefore the designed heuristic methods were trained and tested on files of PE format only. This implies that it would not work for other file types although it is our belief that if the same processes are followed, the approaches can be adopted for other file types.

During the validation stage of our study, we intended to compare our work against already existing tools and approaches. However, after extensive research, we were unable to find a finished system that scored files and therefore these tests were not possible. Testing the methods against other anti-virus systems that do not achieve the same end output as our approaches would not yield results fair to our approaches.

Since the intent was to design a fast heuristic approach, we limit our malware analysis techniques to static based analysis to perform the feature extraction and hash function calculations. This allows the proposed methods to be very fast and not resource intensive. However, this also implies that heavily obfuscated malware might be able to evade detection by this framework.

One of the main challenges of this study was collecting the training and testing dataset. We actively searched and collected malware and clean files during the study in that the initial studies were fully explored using the large dataset that we had in the end. This led to the replication of various studies.

7.4 Future work

In this work, we present a quantification framework for efficient malware detection in portable executable files by proposing two approaches. These can be further extended to include some additional features. Some of the identified extensions are:

a) Combining the designed approaches: we propose using evidence combinational methods that would not be affected by the use similar theories in the component approaches. Investigating the achieved results could lead to an overall file score as one metric for measuring malicious status.

b) Adopting the methods for other file types: One of our hypotheses is that the designed approaches can be adopted for other file type for both mobile and Personal Computer malware. So, it would be worth exploring and testing the adoption of the approaches for android based malware and other file types like .pdf, macro malware among others.

c) Integrating n-gram code based analysis and detection approaches in the framework. Due to various recent research work that utilise n-gram code malware analysis, we believe that this is an approach that can be used in conjunction with the suggested approaches. So, it would be worth exploring to see how an extension of this work using the n-gram approach results would affect the detection rates achieved in this study.

d) Further explore the effectiveness of our detection method for different malware variants and unknown malware. Although we test the methods against known and identified malware types, it is our expectation that the designed methods can be used to detect variants of known and unknown malware.

7.5 Summary

As internet use evolves and the internet of things becomes a reality, there is need to be more vigilant about protecting networked systems from being compromised. This is especially important as Cybercrime is expected continue to raise up to costing the world $6 trillion by 2021 as earlier stated. Malware which is one of the ways through which computer systems are compromised and exploited is also expected to be a growing challenging with statistics showing that each year, the number of malware discovered keeps growing. Protection of the end systems is very important as they are the target due to the commercial value attached to information, processing resources or even the ability to disrupt their use as seen in the ransomware attacks. Anti-malware solutions available today allow for the protection of the system however there is still more improvement needed as systems become more high performance and computer usable becomes more individual controlled. Providing a standard user with more cognitive information for more efficient decision making on a file malicious intent can be the difference between a compromised system and a safe system.

REFERENCES

[1] Brian Krebs, “Akamai on the Record KrebsOnSecurity Attack — Krebs on Security,” \iIn-depth Security News and investigation, 22-Nov-2016.

[2] “Mirai ‘internet of things’ malware from Krebs DDoS attack goes open source,” \iNaked Security, 05-Oct-2016.

[3] Hacking *et al.*, “Mirai IoT botnet blamed for ‘smashing Liberia off the internet.’” [Online]. Available: http://www.theregister.co.uk/2016/11/04/liberia_ddos/. [Accessed: 10-Nov-2016].

[4] “Cyber crime cost UK business more than £1bn in the past year,” *ComputerWeekly*. [Online]. Available: http://www.computerweekly.com/news/450298242/Cyber-crime-cost-UK-business-more-than-1bn-in-the-past-year. [Accessed: 10-Nov-2016].

[5] “2015 Cost of Cyber Crime Study Global.” [Online]. Available: http://www.cnmeonline.com/myresources/hpe/docs/HPE_SIEM_Analyst_Report_-_2015_Cost_of_Cyber_Crime_Study_-_Global.pdf. [Accessed: 10-Nov-2016].

[6] “Cybercrime Costs More Than You Think | Hamilton Place Strategies.” [Online]. Available: http://www.hamiltonplacestrategies.com/news/cybercrime-costs-more-you-think#Paper. [Accessed: 10-Nov-2016].

[7] T. Seals, “Annual Cybercrime Costs to Double to $6 Trillion by 2021,” *Infosecurity Magazine*, 01-Sep-2016. [Online]. Available: http://www.infosecurity-magazine.com/news/annual-cybercrime-costs-double-6/. [Accessed: 27-Oct-2016].

[8] M. Corkery, “Hackers’ $81 Million Sneak Attack on World Banking,” *The New York Times*, 30-Apr-2016.

[9] “Lack of digital talent adds to cybersecurity problems,” *Washington Post*. [Online]. Available: https://www.washingtonpost.com/news/federal-eye/wp/2015/07/19/lack-of-digital-talent-adds-to-cybersecurity-problems/. [Accessed: 10-Nov-2016].

[10] “AV-TEST – The Independent IT-Security Institute.” [Online]. Available: https://www.av-test.org/en/statistics/malware/. [Accessed: 27-Oct-2016].

[11] “Kaspersky Lab - Demand for Security Talent Report.” [Online]. Available: https://business.kaspersky.com/files/2016/07/Kaspersky_Lab_Demand_for_Security_Talent_Report.pdf. [Accessed: 10-Nov-2016].

[12] M. Neugschwandtner, P. M. Comparetti, G. Jacob, and C. Kruegel, “FORECAST: Skimming off the Malware Cream,” in *Proceedings of the 27th Annual Computer Security Applications Conference*, New York, NY, USA, 2011, pp. 11–20.

[13] “Antivirus Isn’t Dead, It Just Can’t Keep Up.” [Online]. Available: http://labs.lastline.com/lastline-labs-av-isnt-dead-it-just-cant-keep-up. [Accessed: 03-Jul-2015].

[14] V. A. Batenin, “Computer resource optimization during malware detection using antivirus cache,” US7962959 B1, 14-Jun-2011.

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

[16] M. Alazab, “Profiling and classifying the behavior of malicious codes,” *J. Syst. Softw.*, vol. 100, pp. 91–102, Feb. 2015.

[17] “Heuristic Techniques in AV Solutions: An Overview | Symantec Connect.” [Online]. Available: https://www.symantec.com/connect/articles/heuristic-techniques-av-solutions-overview. [Accessed: 27-Oct-2016].

[18] J. Z. Kolter and M. A. Maloof, “Learning to Detect and Classify Malicious Executables in the Wild,” *J Mach Learn Res*, vol. 7, pp. 2721–2744, Dec. 2006.

[19] X. Ma, Q. Biao, W. Yang, and J. Jiang, “Using multi-features to reduce false positive in malware classification,” in *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, 2016, pp. 361–365.

[20] Y. Ye *et al.*, “Combining File Content and File Relations for Cloud Based Malware Detection,” in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2011, pp. 222–230.

[21] “A static Android malicious code detection method based on multi-source fusion - Semantic Scholar.” [Online]. Available: /paper/A-static-Android-malicious-code-detection-method-Du-

Wang/02a539ee4f78c968f873fd02f5841a2d3f55854c. [Accessed: 21-Nov-2016].

[22] J. Koret and E. Bachaalany, *The antivirus hacker's handbook*. Indianapolis, IN: John Wiley & Sons Inc, 2015.

[23] "Operating system market share." [Online]. Available: http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0. [Accessed: 25-Oct-2016].

[24] Microsoft Corporation, "Microsoft Portable Executable and Common Object File Format Specification." 2013.

[25] N. Idika and A. P. Mathur, "A survey of malware detection techniques," *Purdue Univ.*, p. 48, 2007.

[26] Bitdefender, "History of Malware," White Paper 2010.

[27] K. Mathur and S. Hiranwal, "A Survey on Techniques in Detection and Analyzing Malware Executables," vol. Volume 3, Issue 4, April 2013, pp. 423–428, 2013.

[28] E. Skoudis and L. Zeltser, *Malware: Fighting Malicious Code*. Prentice Hall Professional, 2004.

[29] "The Evolution Of Malware," *Dark Reading*. [Online]. Available: http://www.darkreading.com/risk/the-evolution-of-malware/a/d-id/1322461. [Accessed: 27-Oct-2016].

[30] M. Sikorski and A. Honig, "Practical Malware Analysis," *Netw. Secur.*, vol. 2012, no. 12, p. 4, 12.

[31] J. Aycock, *Spyware and Adware*, vol. 50. Boston, MA: Springer US, 2011.

[32] F. Thomas, *Adware: The Only Book You'll Ever Need*. Lulu Press, Inc, 2015.

[33] X. Wu and L. Cao, "Analysis and Design of Botnet Detection System," presented at the Computer Science & Service System (CSSS), 2012 International Conference on, 2012, pp. 947–950.

[34] C. C. Elisan, *Malware, Rootkits & Botnets A Beginner's Guide*. McGraw Hill Professional, 2012.

[35] R. Ferguson, "The Botnet Chronicles. A Journey to Infamy," Trend Micro, Incorporated, Nov. 2010.

[36] Z. Bu, P. Bueno, R. Kashyap, and A. Wosotowsky, “The New Era of Botnets,” McAfee Labs - An Itel Company, White Paper, 2010.

[37] “Estonia: To Black Out an Entire Country – part one.” [Online]. Available: http://resources.infosecinstitute.com/estonia-to-black-out-an-entire-country-part-one/. [Accessed: 27-Oct-2016].

[38] “The Art of the Maktub Locker Ransomware,” *BleepingComputer*. [Online]. Available: http://www.bleepingcomputer.com/news/security/the-art-of-the-maktub-locker-ransomware/. [Accessed: 27-Oct-2016].

[39] B. Sullivan, Columnist, and N. B. C. News, “‘Ransomware’ tricks victims into paying hefty fines,” *NBC News*, 26-Apr-2013. [Online]. Available: http://www.nbcnews.com/technology/ransomware-tricks-victims-paying-hefty-fines-6C9621426. [Accessed: 27-Oct-2016].

[40] F. Touchette, “The evolution of malware,” *Netw. Secur.*, vol. 2016, no. 1, pp. 11–14, Jan. 2016.

[41] Larry Boettger, “The Morris Worm: How it Affected Computer Security and Lessons Learned by it,” 24-Dec-2000. [Online]. Available: https://www.giac.org/paper/gsec/405/morris-worm-affected-computer-security-lessons-learned/100954. [Accessed: 27-Oct-2016].

[42] M. W. Ligh, *Malware analyst’s cookbook and dvd*, 1st ed. Indianapolis, IN: Wiley Pub., Inc, 2010.

[44] Matt Pietrek, “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format.” MSDN Library, Mar-1994.

[45] “Dynamic-Link Libraries (Windows).” [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms682589(v=vs.85).aspx. [Accessed: 04-Nov-2016].

[46] E. Eilam and E. J. Chikofsky, *Reversing: secrets of reverse engineering*. Indianapolis, IN: Wiley, 2005.

[48] Randy Kath, “The Portable Executable File Format from Top to Bottom.” 1997, Microsoft Corporation.

[48] T.-Y. Wang, C.-H. Wu, and C.-C. Hsieh, “Detecting Unknown Malicious Executables Using Portable Executable Headers,” in *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM ’09*, 2009, pp. 278–284.

[49] “Malware Researcher’s Handbook (Demystifying PE File).” [Online]. Available: http://resources.infosecinstitute.com/2-malware-researchers-handbook-demystifying-pe-file/. [Accessed: 09-Nov-2016].

[50] Y. Li *et al.*, “Experimental Study of Fuzzy Hashing in Malware Clustering Analysis,” presented at the 8th Workshop on Cyber Security Experimentation and Test (CSET 15), 2015.

[51] Robshaw, Matthew JB, “MD2, MD4, MD5, SHA and other hash functions,” Technical Report TR-101, RSA Laboratories, 1995. version 4.0, 1995.

[52] R. Perdisci, A. Lanzi, and W. Lee, “Classification of packed executables for accurate computer virus detection,” *Pattern Recognit. Lett.*, vol. 29, no. 14, pp. 1941–1946, Oct. 2008.

[53] C. Oprisa, M. Checiches, and A. Nandrean, “Locality-sensitive hashing optimizations for fast malware clustering,” in *2014 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2014, pp. 97–104.

[54] J. Kornblum, “Identifying almost identical files using context triggered piecewise hashing,” *Digit. Investig.*, vol. 3, Supplement, pp. 91–97, Sep. 2006.

[55] “ModSecurity Advanced Topic of the Week: Detecting Malware with Fuzzy Hashing,” *Trustwave*. [Online]. Available: https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-Advanced-Topic-of-the-Week--Detecting-Malware-with-Fuzzy-Hashing/. [Accessed: 22-Jan-2016].

[56] Dunham Ken, “A fuzzy future in malware research,” *The ISSA J.*, vol. 11, no. 8, pp. 17–18, 2003.

[57] “Tracking Malware with Import Hashing,” *M-unition*. [Online]. Available: https://www.mandiant.com/blog/tracking-malware-import-hashing/. [Accessed: 14-Jul-2015].

[58] Georg Wicherski, “peHash: a novel approach to fast malware clustering,” in *LEET’09 Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, 2009, vol. 1–1.

[59] J. Pearl, “Decision making under uncertainty,” *ACM Comput. Surv.*, vol. 28, no. 1, pp. 89–92, Mar. 1996.

[60] M. M. Gupta and J. Qi, “Fuzzy Logic and Uncertainty ModellingTheory of T-norms and fuzzy inference methods,” *Fuzzy Sets Syst.*, vol. 40, no. 3, pp. 431–450, Apr. 1991.

[61] L. A. Zadeh, “The role of fuzzy logic in the management of uncertainty in expert systems,” *Fuzzy Sets Syst.*, vol. 11, no. 1–3, pp. 199–227, 1983.

[62] B. Więckowski, “Review of Proof theory for fuzzy logics. Applied Logic Series, vol. 36,” *Bull. Symb. Log.*, vol. 16, no. 3, pp. 415–419, 2010.

[63] G. Shafer, *A mathematical theory of evidence*. Princeton, NJ: Princeton Univ. Press, 1976.

[64] B. G. Buchanan and E. H. Shortliffe, Eds., *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project*. Reading, Mass: Addison-Wesley, 1984.

[65] D. E. Heckerman and E. H. Shortliffe, “From certainty factors to belief networks,” *Artif. Intell. Med.*, vol. 4, no. 1, pp. 35–52, Feb. 1992.

[66] “How Symantec Antivirus system detects viruses | Symantec Connect.” [Online]. Available: https://www.symantec.com/connect/articles/how-symantec-antivirus-system-detects-viruses. [Accessed: 09-Nov-2016].

[67] J. Davis and M. Goadrich, “The relationship between Precision-Recall and ROC curves,” 2006, pp. 233–240.

[68] “ClamavNet.” [Online]. Available: http://www.clamav.net/. [Accessed: 04-Nov-2016].

[69] D. G. Vigna, “Antivirus Isn’t Dead, It Just Can’t Keep Up.” [Online]. Available: http://labs.lastline.com/lastline-labs-av-isnt-dead-it-just-cant-keep-up. [Accessed: 27-Oct-2016].

[70] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, “Proactive Detection of Computer Worms Using Model Checking,” *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 424–438, Oct. 2010.

[71] “A Brief History of Malware Obfuscation: Part 1 of 2,” *blogs@Cisco - Cisco Blogs*. [Online]. Available: http://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2. [Accessed: 27-Oct-2016].

[72] “Mastering 4 Stages of Malware Analysis.” [Online]. Available: https://zeltser.com/mastering-4-stages-of-malware-analysis/. [Accessed: 27-Oct-2016].

[73] “guelfoweb/peframe,” *GitHub*. [Online]. Available: https://github.com/guelfoweb/peframe. [Accessed: 03-Jul-2015].

[74] “joxeankoret/pyew,” *GitHub*. [Online]. Available: https://github.com/joxeankoret/pyew. [Accessed: 31-May-2015].

[75] Tyler Hudak, “Mastiff Documentation.” KoreLogic Security.

[76] “Malwr - Malware Analysis by Cuckoo Sandbox.” [Online]. Available: https://malwr.com/. [Accessed: 27-Oct-2016].

[77] “Anubis: Analyzing Unknown Binaries.” [Online]. Available: http://analysis.iseclab.org/. [Accessed: 27-Oct-2016].

[78] “ThreatExpert - Automated Threat Analysis.” [Online]. Available: http://www.threatexpert.com/. [Accessed: 27-Oct-2016].

[79] “VirusTotal - Free Online Virus, Malware and URL Scanner.” [Online]. Available: https://www.virustotal.com/. [Accessed: 27-Oct-2016].

[80] C. H. Malin, E. Casey, and J. M. Aquilina, *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress, 2008.

[81] Michael Sikorski and Andrew Honig. 2012. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software* (1st ed.). No Starch Press, San Francisco, CA, USA.

[82] Peter Szor. 2005. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional.

[83] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” *ACM Comput. Surv. CSUR*, vol. 44, no. 2, p. 6, 2012.

[84] “IDA: About.” [Online]. Available: https://www.hex-rays.com/products/ida/. [Accessed: 10-Nov-2016].

[85] “OllyDbg v1.10.” [Online]. Available: http://ollydbg.de/. [Accessed: 10-Nov-2016].

[86] “WinDbg.” [Online]. Available: http://www.windbg.org/. [Accessed: 10-Nov-2016].

[87] "Five Anti-Analysis Tricks That Sometimes Fool Analysts," *Malwarebytes Unpacked*. [Online]. Available: https://blog.malwarebytes.org/intelligence/2014/09/five-anti-debugging-tricks-that-sometimes-fool-analysts/. [Accessed: 03-Jun-2015].

[88] Jana Suman and Vitaly Shmatikov, "Abusing File Processing in Malware Detectors for Fun and Profit." [Online]. Available: https://www.cs.cornell.edu/~shmat/shmat_oak12av.pdf. [Accessed: 10-Nov-2016].

[89] Katja Hahn, "Robust Static Analysis of Portable Executable Malware," HTWK Leipzig Fakult¨at Informatik, Mathematik und Naturwissenschaften, Dec. 2014.

[90] "OllyDbg 2.0." [Online]. Available: http://www.ollydbg.de/version2.html. [Accessed: 10-Nov-2016].

[91] "Immunity Debugger." [Online]. Available: https://www.immunityinc.com/products/debugger/. [Accessed: 11-Nov-2016].

[92] "Anti-debugging and Anti-VM techniques and anti-emulation." [Online]. Available: http://resources.infosecinstitute.com/anti-debugging-and-anti-vm-techniques-and-anti-emulation/. [Accessed: 11-Nov-2016].

[93] C. K. Behera and D. L. Bhaskari, "Different Obfuscation Techniques for Code Protection," *Procedia Comput. Sci.*, vol. 70, pp. 757–763, Jan. 2015.

[94] "Obfuscation: Malware's best friend," *Malwarebytes Labs*. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2013/03/obfuscation-malwares-best-friend/. [Accessed: 24-Nov-2016].

[95] Joxean Koret and Elias Bachaalany. 2015. *The Antivirus Hacker's Handbook* (1st ed.). Wiley Publishing.

[96] M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, *Malware Detection*. Springer Science & Business Media, 2007.

[97] W. Rui, J. Xiaoqi, and N. Chujiang, "A Behavior Feature Generation Method for Obfuscated Malware Detection," presented at the Computer Science & Service System (CSSS), 2012 International Conference on, 2012, pp. 470–474.

[98] Y. Ilsun and Y. Kangbin, “Malware Obfuscation Techniques: A Brief Survey,” presented at the Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, 2010, pp. 297–300.

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

[100] S. Han, K. Lee, and S. Lee, “Packed PE File Detection for Malware Forensics,” in *2nd International Conference on Computer Science and its Applications, 2009. CSA ’09*, 2009, pp. 1–7.

[101] X. Li, P. K. K. Loh, and F. Tan, “Mechanisms of Polymorphic and Metamorphic Viruses,” in *Intelligence and Security Informatics Conference (EISIC), 2011 European*, 2011, pp. 149–154.

[102] “SANS - Information Security Resources.” [Online]. Available: https://www.sans.org/security-resources/idfaq/how-is-a-tool-like-an-integrity-checker-used-in-intrusion-detection/1/6. [Accessed: 28-Nov-2016].

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

[104] M. R. Chouchane and A. Lakhotia, “Using Engine Signature to Detect Metamorphic Malware,” in *Proceedings of the 4th ACM Workshop on Recurring Malcode*, New York, NY, USA, 2006, pp. 73–78.

[105] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, “Semantics-aware malware detection,” in *2005 IEEE Symposium on Security and Privacy (S P’05)*, 2005, pp. 32–46.

[106] M. D. Preda, M. Christodorescu, S. Jha, and S. Debray, “A Semantics-based Approach to Malware Detection,” in *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, NY, USA, 2007, pp. 377–388.

[107] R. Luh, S. Marschalek, M. Kaiser, H. Janicke, and S. Schrittwieser, “Semantics-aware detection of targeted attacks: a survey,” *J. Comput. Virol. Hacking Tech.*, pp. 1–39, May 2016.

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

[109] “heuristic - definition of heuristic in English | Oxford Dictionaries,” *Oxford Dictionaries | English*. [Online]. Available: https://en.oxforddictionaries.com/definition/heuristic. [Accessed: 28-Nov-2016].

[110] A.-S. K. Pathan, *The State of the Art in Intrusion Prevention and Detection*. CRC Press, 2014.

[111] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” in *2001 IEEE Symposium on Security and Privacy, 2001. S P 2001. Proceedings*, 2001, pp. 38–49.

[112] DigitalNinja., “Using Fuzzy Hashing Techniques to Identify Malicious Code,” Apr. 2007.

[113] David French, “Beyond Section Hashing,” 2010 CERT Research Report CMU/SEI-2012-TR-004, 2011.

[114] S. Arik, T. Huang, W. K. Lai, and Q. Liu, *Neural Information Processing: 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings*. Springer, 2015.

[115] A. Azab, R. Layton, M. Alazab, and J. Oliver, “Mining Malware to Detect Variants,” in *Cybercrime and Trustworthy Computing Conference (CTC), 2014 Fifth*, 2014, pp. 44–53.

[116] Y. Du, X. Wang, and J. Wang, “A static Android malicious code detection method based on multi-source fusion,” *Secur. Commun. Netw.*, vol. 8, no. 17, pp. 3238–3246, Nov. 2015.

[117] “Malware Threat Scoring System | MAEC Project Documentation.” [Online]. Available: http://maecproject.github.io/documentation/use_cases/cyber_threat_analysis/malware_threat_scoring_system/. [Accessed: 04-Nov-2016].

[118] “Malware Scoring Modules,” *RSA Security Analytics Documentation*, 05-Mar-2014. [Online]. Available: https://sadocs.emc.com/0_en-us/090_10.4_User_Guide/40_InvestigAnalysis/00_Investig_Flo/MaScorMod. [Accessed: 04-Nov-2016].

[119] A. Kumar and G. Aghila, “Portable executable scoring: What is your malicious score?,” in *2014 International Conference on Science Engineering and Management Research (ICSEMR)*, 2014, pp. 1–5.

[120] M. Christodorescu and S. Jha, “Static Analysis of Executables to Detect Malicious Patterns.,” Aug. 2003.

[121] “TekDefense - News - Tektip ep23 - MASTIFF with a splash of Maltrieve.” [Online]. Available: http://www.tekdefense.com/news/2013/2/22/tektip-ep23-mastiff-with-a-splash-of-maltrieve.html. [Accessed: 01-Jun-2015].

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

[123] S. Hou, L. Chen, E. Tas, I. Demihovskiy, and Y. Ye, “Cluster-oriented ensemble classifiers for intelligent malware detection,” in *2015 IEEE International Conference on Semantic Computing (ICSC)*, 2015, pp. 189–196.

[124] J. Kinable and O. Kostakis, “Malware classification based on call graph clustering,” *J. Comput. Virol.*, vol. 7, no. 4, pp. 233–245, Feb. 2011.

[125] “An ‘Average’ Cyber Crime Costs a U.S. Company $15.4 Million,” *Forbes*. [Online]. Available: http://www.forbes.com/sites/moneybuilder/2015/10/17/an-average-cyber-crime-costs-a-u-s-company-15-4-million/. [Accessed: 19-Oct-2015].

[126] Joel Yonts, “Attributes of Malware.” SANS Institute InfoSec Reading Room, 30-Jun-2012.

[127] H. Khan, F. Mirza, and S. A. Khayam, “Determining malicious executable distinguishing attributes and low-complexity detection,” *J. Comput. Virol.*, vol. 7, no. 2, pp. 95–105, Jan. 2010.

[128] R. Merkel, T. Hoppe, C. Kraetzer, and J. Dittmann, “Statistical Detection of Malicious PE-Executables for Fast Offline Analysis,” in *Communications and Multimedia Security*, B. D. Decker and I. Schaumüller-Bichl, Eds. Springer Berlin Heidelberg, 2010, pp. 93–105.

[129] “erocarrera/pefile,” *GitHub*. [Online]. Available: https://github.com/erocarrera/pefile. [Accessed: 01-Jun-2015].

[130] “hiddenillusion/AnalyzePE,” *GitHub*. [Online]. Available: https://github.com/hiddenillusion/AnalyzePE. [Accessed: 24-Nov-2016].

[131] “Elasticsearch: The Definitive Guide [master] | Elastic.” [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/guide/master/index.html. [Accessed: 28-Nov-2016].

[132] F. Taroni, A. Biedermann, and S. Bozza, *Statistics in Practice : Bayesian Networks for Probabilistic Inference and Decision Analysis in Forensic Science (2)*. Somerset, GB: Wiley, 2014.

[133] V. Roussev, “An evaluation of forensic similarity hashes,” *Digit. Investig.*, vol. 8, Supplement, pp. S34–S41, Aug. 2011.

[134] David French and William Casey, “Fuzzy Hashing Techniques in Applied Malware Analysis,” Results of SEI Line-Funded Exploratory New Starts Projects CMU/SEI-2012-TR-004, Aug. 2012.

[135] A. P. Namanya, J. P. Disso, and I. U. Awan, “Evaluation of automated static analysis tools for malware detection in Portable Executable files,” in *2015 31st UKPEW*, University of Leeds, 2015, pp. 81–95.

[136] “radare.” [Online]. Available: http://www.radare.org/r/. [Accessed: 23-Mar-2016].

[137] I. Kaliszewski and D. Podkopaev, “Simple additive weighting—A metamodel for multiple criteria decision analysis methods,” *Expert Syst. Appl.*, vol. 54, pp. 155–161, Jul. 2016.