APPLICATION OF ARTIFICIAL INTELLIGENCE FOR DETECTING DERIVED VIRUSES

OMOTAYO FAUSAT ASIRU

(215082564)

A dissertation submitted in fulfilment of the requirements

for the degree of Master of Science in Computer Science

in the

School of Mathematics, Statistics and Computer Science

University of KwaZulu-Natal

Durban, South Africa

July 2017

UNIVERSITY OF KWAZULU-NATAL

COLLEGE OF AGRICULTURE, ENGINEERING AND SCIENCE

DECLARATION

The research was performed at the University of KwaZulu-Natal under the supervision of Professor JM Blackledge and co-supervision of Mr MT Dlamini. I hereby declare that all materials incorporated in this dissertation are my own original work except where acknowledgement is made by name or in the form of a reference. The work contained herein has not been submitted in part or whole for a degree at any other university.

[illegible]

OF Asiru

Date: 30 July, 2017

As the candidate's supervisor, I have approved the dissertation for submission

[illegible]

Professor JM Blackledge

Date: 30 July, 2017

As the candidate's co-supervisor, I have approved the dissertation for submission

[illegible]

Mr MT Dlamini

Date: 30 July, 2017

DECLARATION II – PLAGIARISM

I, Omotayo Fausat Asiru, declare that

1. The research reported in this dissertation, except where otherwise indicated, is my original research.
2. This dissertation has not been submitted for any degree or examination at any other university.
3. This dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
    a. Their words have been re-written but, the general information attributed to them has been referenced.
    b. Where their exact words have been used, their writing has been placed in italics and in quotation marks, and referenced.
5. This dissertation does not contain text, graphics or tables which are copied and pasted from the internet, unless specifically acknowledged, and the source being detailed in the dissertation and in References section.

[illegible]

Date: 30 July, 2017

DEDICATION

This research work is dedicated to God Almighty, my creator for seeing me through this research, to my loving husband, Olufikayo for his support and to my precious daughter, Ayooluwa.

ACKNOWLEDGEMENT

My appreciation goes to God first for the successful completion of my master's program; all glory and adoration be unto him. My profound and sincere gratitude goes to my husband, Mr Olufikayo Olowoyo for his love, prayer, physical and financial support. I could not have done this without you, thanks for being there.

Special thanks go to my supervisor, Professor JM Blackledge for his contribution and support. I am deeply grateful to my co-supervisor, Mr MT Dlamini for his counsel, guidance, contribution and his tolerance throughout this program.

I want to appreciate Pastor and Mrs Kayode Akindeji, Mr and Mrs Peter Olukanmi for their care, support and words of encouragement.

Finally, a big thank you goes to everyone that has contributed to the success of this project in one way or the other. Thank you all and God bless.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF INCLUDED PAPERS

Conference paper published in 16th ECCWS 2017

O.F. Asiru, M.T. Dlamini, J.M. Blackledge, “Application of Artificial Intelligence for Detecting Derived Viruses”. *Proceedings of the 16th European Conference on Cyber Warfare and Security ECCWS June 2017*. pp. 647-655.

ABSTRACT

A lot of new viruses are being created each and every day. However, some of these viruses are not completely new per se. Most of the supposedly ‘new’ viruses are not necessarily created from scratch with completely new (something novel that has never been seen before) mechanisms. For example, some of these viruses just change their forms and come up with new signatures to avoid detection. Hence, such viruses cannot be argued to be new. This research refers to such as derived viruses. Just like new viruses, we argue that derived viruses are hard to detect with current scanning-detection methods.

Many virus detection methods exist in the literature, but very few address the detection of derived viruses. Hence, the ultimate research question that this study aims to answer is; how might we improve the detection rate of derived computer viruses?

The proposed system integrates a mutation engine together with a neural network to detect derived viruses. Derived viruses come from existing viruses that change their forms. They do so by adding some irrelevant instructions that will not alter the intended purpose of the virus.

A mutation engine is used to group existing virus signatures based on their similarities. The engine then creates derivatives of groups of signatures. This is done up until the third generation (of mutations). The existing virus signatures and the created derivatives are both used to train the neural network.

The derived signatures that are not used for the training are used to determine the effectiveness of the neural network. Ten experiments were conducted on each of the three derived virus generations. The first generation showed the highest derived virus detection rate compared to the other two generations. The second generation also showed a slightly higher detection rate than the third generation which has the least detection rate.

Experimental results show that the proposed model can detect derived viruses with an average accuracy detection rate of 80% (This includes a 91% success rate on first generation, 83% success rate on second generation and 65% success rate on third generation). The results further show that the correlation between the original virus signature and its derivatives decreases with the generations. This means that after many generations of a virus changing form, its variants will no longer look like the original. Instead the variants look like a completely new virus even though the variants and the original virus will always have the same behaviour and operational characteristics with similar effects.

CHAPTER ONE

INTRODUCTION

1.1 Introduction

This chapter presents an overview of this dissertation and starts with a motivation of the study. This is followed by a brief discussion on the problem statement which continues with a statement on research question that the work aims to answer and a discussion on the aims and objectives of the study. A discussion on the scope and limitation of the study is then presented. The chapter also provides a definition of terms as defined for the purposes of this dissertation and concludes with an overview of the organization of the rest of this dissertation.

1.2 Motivation

Computer viruses come with a wide variety of nefarious activities that range from consuming an excess of memory to showing some funny and peculiar actions and performing serious temporary or permanent damage to computing systems (Filiol, 2005). Criminals are now using viruses to hijack and take over computers from unsuspecting users. Hijacked computers are then used to create botnets to spread spam, perform a Distributed Denial of Service (DDoS) and steal digital identities. Therefore, the effects of computer viruses have moved from just showing annoying messages on the screen, for example, to having an impact on company reputation and affecting business characteristics.

Several attempts have been made by a wide variety of anti-virus programs to solve the prevalent virus problem in computing systems. For example, McAfee, Symantec's Norton and Kaspersky are just a few of the market leaders in the anti-virus industry. In spite of all available anti-virus programs in the market today, virus attacks do not seem to slow down. Singhal (2014) has stated that it is possible for a computer that is connected to the internet to experience a virus attack every 39 seconds. This could mean that at least 2215 computers are being infected by virus on a daily basis.

Computer viruses are argued to cost businesses billions of dollars and the cost of the damage is increasing yearly. The increasing cost of damage from viruses can be attributed to the fact that new viruses are being created each day (Daoud, 2009). However, some of these supposedly 'new' viruses are not entirely 'new' per se. Most of the supposedly 'new' viruses are not necessarily created from scratch with completely 'new' characteristics and attack mechanisms. They just change their form and come up with different signatures to avoid detection (Feng & Gupta, 2009). Hence, such viruses cannot be argued to be 'new'. Examples of such viruses are polymorphic and metamorphic viruses. A polymorphic virus uses a mutation engine to change the virus code without changing the original algorithm (Khari & Bajaj, 2014). A metamorphic virus uses obfuscation techniques to generate a different virus body but with the same overall functionalities as with the original (Singla et al., 2015). This research refers to both of them as "derived viruses". A derived virus is a type of virus that modifies itself to produce another virus body from an existing virus while maintaining the same functionalities as the original. Just like new viruses, derived viruses are hard to detect with current scanning-detection methods. This is because virus scanning-detection methods rely mainly on known or existing signatures. Derived viruses on the other hand, and, at their first attempt in trying to infect a machine, do not have any known or existing signatures (Silverman, 2001).

1.3 Problem Statement

Since the first appearance of computer virus in the 1970s, there has been a lot of research targeted at solving the problem of virus attacks on computing devices. However, the protection of computing devices against viruses is still a major problem within the field of information security (Khorsand & Hamzeh, 2013). A recent incident of a virus attack was reported in Kumar (2016) when the National Health Service network in England was compromised by a computer virus. This led to cancellation of planned surgeries and medical appointments in many hospitals. Hence, computer viruses at the extreme level could be argued to result in loss of life. This could happen when patients are prevented by the acts of a computer virus from getting the medical attention they need.

Currently, computer viruses attack at least a million computers every year. This causes damage estimated to be billions of US dollars per year (Kamarudin et al., 2013). A virus named “Mydoom” was reported to have caused damages worth 38.5 billion dollars. The virus attacks window based computers and gets transmitted primarily through emails. It was reported to be one of the most damaging computer virus ever (Rajesh et al., 2015). Similarly, another virus named “I Love You” was reported to have caused damage worth 8.75 billion dollars. This virus sends a copy of itself to every address on the host’s address book and overwrites files while it renders the original file useless. Once this virus is on a computer system, it downloads a Trojan horse which ‘steals’ usernames and passwords from the computer and sends them back to the virus author (Rajesh et al., 2015). An incident of a virus attack which made headlines was reported in Otake (2015) on a compromised Japanese pension system. This case led to 1.25 million cases of personal details being leaked. The data were leaked by agency employees who opened a file containing a virus that was attached to an email. These are just a few of a number of examples indicating the massive damages which computer viruses have caused users and various organizations.

Over the years, the number of computer viruses such as the one responsible for breaching the Japanese pension system has increased. The number of computer viruses in the year 1992 was estimated to be between 1000 and 2300 (Daoud, 2009). It is also reported by Daoud (2009) that over ten years since then, (i.e. in the year 2002) the number of known viruses had increased to

60,000. By 2009, there were over 100,000 known computer viruses (Daoud, 2009). In year 2016, Kaspersky Lab (2016) in their antivirus solution reported a total of 69,277,289 unique malicious viruses. This is against 4,000,000 unique malicious viruses that were reported by the same product in the previous year (i.e. in 2015) (Ivanov et al., 2015). These statistics reflects a 1631.9% increase within a year, i.e. between 2015 and 2016. Furthermore, Symantec's internet security threat report shows a 125% increase in the number of zero-day vulnerabilities between 2014 and 2015. This is an indication that todays' threats are increasing at an exponential rate.

Many of today's threats are not new per se but just use some modern techniques to change forms thereby looking different each time while retaining their functionalities and basic characteristics. For example, nowadays, some virus writers are creating mutation engines. Mutation engines are used to create virus codes that have the ability to change their signatures at every infection to avoid detection. The self-modification ability that modern day viruses possess has put strain on the current scanning detection methods (Kumar, 2016).

An example of such is a tool kit called the "Dark Avenger's Mutation Engine" (Muhtadi, 2014). This tool has the self-modification capability which makes it hard for current anti-virus tools to detect viruses. On the other hand, the same tool makes it easier even for non-programmers to create a virus that mutates. This is done in such a manner that in taking any two files infected by the same virus, their signatures appears to be totally different and without any correlation. Hence, it can be argued that the reported exponential increase in the number of viruses is somehow inflated. The increase may be due to some viruses using different techniques to change their form and signatures. Therefore, it could happen that the same virus could be detected more than once but each time as something completely different from the original signature. Existing signature based anti-virus tools have failed to detect mutating viruses and those that have made successful attempts have a low detection rate (Hamza & Hussain, 2014).

In a bid to increase the detection rate of computer virus, many studies (as reported in the known literature) have focused on virus detection methods. However, only a few of these address the detection of derived viruses that change their form or signature. For example, Wang et al. (2009) proposes a method of detecting mutating virus by using a candidate virus gene library. The major drawback of their proposed method is the inability to increase the diversity of the genes in the virus library. The work of Nguyen et al. (2012) proposed a method based on memory abstraction for handling obfuscation in what they refer to as a polymorphic virus. The major characteristic of a polymorphic virus is its self-modification ability. However, the challenge of the proposed solution therein is that it requires an abstract form that captures the memory state of every executed instruction. Hence, Golovko & Bezobrazov (2015) argue that the best modern anti-virus tools can only detect 90% of known viruses and 70% of mutating or polymorphic viruses.

The detection of a derived virus remains the greatest threat, not because it is worse than a normal virus in terms of the damage it causes, but because it is hard to detect by current detection techniques as it changes its ‘look’ at every infection. The research reported in this dissertation therefore explores a novel approach to try and improve the detection rate of derived viruses. The following section discusses the research question that this study aims to answer.

1.4 Research Questions

The ultimate research question that this study aims to answer is; how might we improve the detection rate of derived computer viruses?

1.5 Research Aim

This research aims to derive a virus detection model to help improve the rate of detecting derived viruses from 70% as per Golovko & Bezobrazov (2015).

1.6 Research Objectives

The above aim is accomplished by fulfilling the following research objectives:

1. Investigate and review existing literature on computer virus detection in order to examine the reasons behind the failure of current detection methods in detecting derived viruses.
2. Design, develop and evaluate a model that correctly classifies virus signatures into existing or derived forms
3. Finally, make recommendations on how to further improve the detection rate of derived viruses.

1.7 Scope and Limitation

The scope of the research reported in this dissertation is limited in the sense that it only considers virus signatures of a fixed length i.e. 32 bytes. This is because our solution requires the input to be represented as a fixed-length feature vector (Le et al., 2014). Furthermore, this research scope does not consider virus signatures with special characters and those that are encrypted. It only considers virus signatures that are limited to the normal character set i.e. 0-9, a – z and A – Z.

1.8 Definition of Terms

A few terms are defined in this section, as used throughout this dissertation:

- Virus: Hamza & Hussain (2014) defines a virus as a program that can make copies of itself using a host computer and eventually causes temporary or permanent damage to the system. Nguyen et al. (2012) defines a virus as a block of binary code that can copy itself from one program into another without the knowledge of the program owner.

According to Dhruw et al. (2016), a computer virus is a piece of software that infects other programs within a computing device by modifying them. Based on these definitions and for the purposes of this dissertation, we define a computer virus as a piece of computer code that replicates itself by attaching itself to a host computer and performs unwanted operations within the computer system.

- Virus Signature: Mishra (2010) defines a virus signature as a sequence of bytes that can be found in a virus program code but is unlikely to be found in any other place. According to Zhong et al. (2012), a virus signature is a specific pattern of sequences of bytes that defines a virus. Khorsand & Hamzeh (2013) defines a virus signature as a unique byte sequence of the virus program code. Based on the definitions above and for the purposes of this dissertation, we define a virus signature as a unique string of characters or numbers that makes up a fingerprint that can be used to identify a virus. Our definition excludes special characters.

- Existing Virus: A virus whose signature is already known to the current virus detecting techniques.

- Metamorphic Virus: Dhruw et al. (2016) define a metamorphic virus as a virus that mutates by rewriting itself at every infection. Kakad et al. (2014) define a metamorphic virus as a type of virus that uses code obfuscation techniques to change its code at every infection. According to Rad et al. (2012), a metamorphic virus is a virus that mutates all its body with the help of a mutation engine.

- Polymorphic Virus: Dhruw et al. (2016) define a polymorphic virus as a virus that mutates with every infection in order to make detection by signature impossible. According to Kakad et al. (2014), a polymorphic virus is a virus that changes some instructions in new generations in order to overcome signature scanning. Joshi & Patil

(2012) define a polymorphic virus as a type of virus that uses encryption in order to appear differently at every infection thereby avoiding detection.

- Derived Virus: Based on the definitions of a metamorphic and polymorphic virus above, and, for the purposes of this dissertation, we define a derived virus as any virus that modifies itself using a mutation engine and evolves to have a signature that is different from its original or predecessor at every infection in order to avoid detection by signature scanning.
- Artificial Intelligence: Banerjee (2015) defines an artificial intelligence as the science of automating intelligent behaviours currently achievable by human beings. Pannu (2015) defines artificial intelligence as the development of intelligent machines and software that can reason, learn and communicate with objects. According to Borana (2016), artificial intelligence is the intelligence exhibited by an artificial entity to solve complex problems. Based on the definitions above, and, for the purposes of this dissertation, we define artificial intelligence as the development of intelligent systems capable of performing tasks that normally require human intelligence.
- Artificial Neural Network: Also referred to as neural network, Heaton (2011) defines a neural network as computerized simulation of an actual biological neural network. Sonali & Wankar (2014) define a neural network as an information processing paradigm that is inspired by biological nervous system such as brain processes information. Banerjee (2015) defines neural networks as biologically inspired systems that convert a set of inputs into a set of outputs by a network of neurons. Based on the definitions above, and, for the purposes of this dissertation, we define a neural network as an information processing paradigm inspired by the human brain which learns from sets of inputs to produce an output.

1.9 Overview of Chapters

This dissertation is structured as follows:

Chapter two discusses the background to computer viruses, i.e. evolution, anatomy and classification. Chapter three provides an overview of current virus detection techniques and a review of related literature. Chapter four presents and discusses the proposed model. Chapter five highlights and discusses the result of various experiments conducted with the proposed model. Finally, chapter six concludes this study and provides recommendations for future work.

CHAPTER TWO

BACKGROUND

2.1 Introduction

The previous chapter positioned our study and reflect on its growing importance and relevance to the field of cyber security. Chapter one has set the scene for this chapter to build on. This chapter briefly discusses the evolution of computer viruses and reflects on how they have evolved since their first appearance. This chapter goes on to discuss the anatomy of a computer virus and its classification. Lastly, it also discusses some virus mutation techniques in preparation for the next chapter.

2.2 Evolution of computer viruses

Over the years, virus writers have used different approaches to create viruses that can replicate easily and effectively. Using different approaches, viruses have evolved over the years and are becoming more and more difficult to detect. Furthermore, the damage caused by viruses is increasing year after year. Early viruses had little impact apart from annoying pop up messages. However today’s viruses have a reputation and financial impact.

The table below summarizes how computer viruses have evolved over the years with a description of each virus and the worth of damages caused by some of these viruses.

Table 2.1: Evolution of computer viruses (Rajesh et al. 2015) & (Joshi & Patil, 2012)

Year	Virus Name	Virus Description
1974	Wabbit	This is also known as Rabbit virus. This virus makes multiple copies of itself at a very high speed on a single computer until it clogs the computer, reducing performance and crashes the computer when a threshold is reached.

1975	Animal	This virus asked a number of questions from the user in an attempt to guess the type of animal the user was thinking of, while the program creates a copy of itself and animal in every directory to which the user had access.
1981	Apple Viruses 1,2,3	These are some of the first viruses in the public domain. They were found on Apple 11operating system and spread through Texas A & M via pirated computer games.
1988	Jerusalem	This virus is activated every Friday the 13th. It affects both .exe and .com files and deletes any program runs on that day.
1991	Tequila	This is the first widespread polymorphic virus found in the wild. Tequila changes its appearance at every infection.
1995	Word Concept	This virus spreads only through Microsoft word documents.
1996	Baza, Laroux and Staog	These are the first set of viruses to infect Windows95 files, Excel and Linux respectively.
1999	Happy99	This virus attaches itself to emails, displays fireworks to hide the changes being made and wishes the user a happy new year.
	Melissa	Melissa executes a macro virus in a document attached to an email. Thereafter, it forwards the document to 50 people in the user's outlook address book. This virus also infects other word documents and mails them out as attachments. The damage caused by this virus is estimated to be $1.1 billion.
2000	I Love You	This virus spread quickly across the globe by sending a copy of itself to every address in its host outlook address book. It overwrites important files with a copy of itself, marks all mp3 files as hidden and downloaded a Trojan horse that would steal user names and passwords and send them to the virus's author. The estimated damage caused by this virus is $8.75 billion.
	Stages	This virus disguised as a joke email about the stages of life and spreads fast across internet.
2001	Anna Kournikova	This virus which was written with a virus tool kit copies itself to the Windows directory and sends the file as an

		attachment to all addresses listed in Microsoft Outlook e-mail address book of its victim. The estimated damage is $166,827.
	Nimda	Nimda infects hundreds of thousands of computers in the world. The virus is one of the most sophisticated viruses with as many as five different methods of replicating and infecting systems. The estimated damage of this virus is $645 million.
	Klez	Klez and its variants are considered to be one of the most persistence viruses ever. It selects one email address from the host's address book to use as the "from" address, then sending the virus to all the other addresses. The estimated damage caused by this virus is $18.9 billion.
2003	SQL Slammer	This virus takes advantage of buffer overflow bugs in Microsoft's SQL Server database. It infected 75,000 computers in approximately 10 minutes and caused damages worth $1.2 billion.
2004	Mydoom	This virus spreads through email and file sharing software. It slowed overall Internet performance by about 10%, and average web page load times by about 50%. The worth of damages caused by mydoom virus is $38.5 billion.
	Sasser	The sasser virus affected one million computers running windows. The virus causes computers to reboot repeatedly and the damage caused by this virus is estimated to be $14.8 billion.
2008	Conficker	This virus infected somewhere between nine and fifteen million servers worldwide which include servers of large government organizations. It caused damage estimated to be $9.1 billion.
2010	Stuxnet	This virus targets Siemens industrial software through Microsoft window.
2012	Flame	Flame virus records skype conversations, audio, keyboard activities, network traffics and screenshots.

The table above summarizes how computer viruses have evolved between the year 1974 and 2012. Computer viruses are almost as old as computers themselves with the first computer virus being discovered in the 1970s and yet, it is still one of the major challenges of computing devices. Since then viruses have evolved from being spread through floppy disks to spreading through the Internet. Thus, it can be concluded that the evolution of the Internet and computer networks also brought about the evolution for computer viruses. This is because viruses spread faster and easier through the Internet as compared to when viruses were mostly spread by floppy disk.

The creation of signature scanning techniques for virus detection brought about the evolution of morphed viruses. These viruses can create variants with same functionalities but different body structure. This was done to prolong the process of analysis and also to make detection with signature scanning difficult. Operating systems and application software also brought about evolution in viruses as virus writers explore the vulnerabilities in newer versions of software to make viruses gain unauthorized access and cause damage on computers. Hence, viruses are likely to keep evolving in order to take advantage of the vulnerabilities in new software. The next section unveils the life cycle of a computer virus.

2.3 Computer virus life cycle

The major characteristic of a computer virus is its self-replication ability. Replication includes making a copy of the virus to the infected program. The infected program can then go on to infect other programs. In this way a virus can continue to spread to infect more programs. The most common computer viruses are parasitic in nature. This means that they work by attaching themselves to carrier objects. The carrier object may be a file or other entity that is likely to be transmitted to another computer such as an executable program. A virus is attached to its host in such a way that it activates when the host program is executed. Once activated, the virus looks for other suitable carrier objects and attaches itself to them. At this point, replication and spreading requires little or no human intervention.

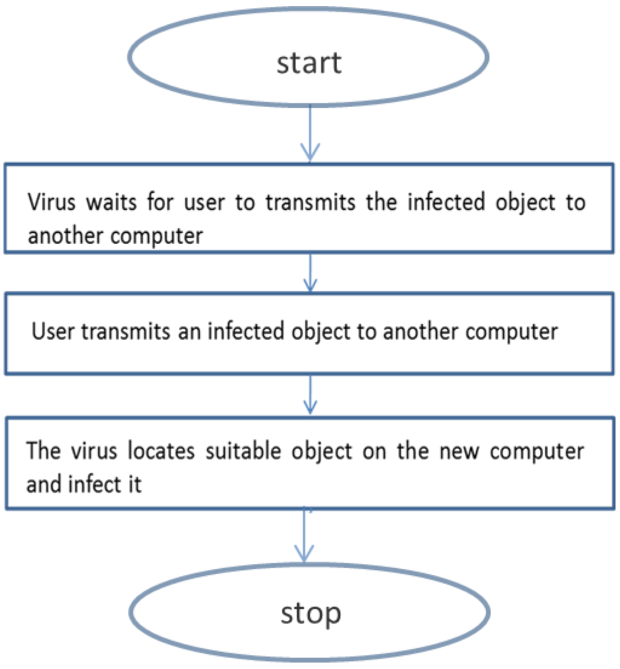

start
Virus waits for user to transmits the infected object to another computer
User transmits an infected object to another computer
The virus locates suitable object on the new computer and infect it
stop

Figure 2.1: The life cycle of a computer virus (Dhruw et al., 2016)

The figure above summarizes the life cycle on how a virus move from one carrier to another infecting its targets. Having discussed the life cycle of a virus, the next section explores the anatomy of a virus.

2.4 Anatomy of a computer virus

The virus code has two major parts, i.e. replication and payload routine. The next sub-section discusses each of these two parts.

2.4.1 Replication routine

The replication routine locates suitable objects and copies the virus to these objects (Dhruw et al., 2016). Computer virus sometimes remains in the memory and monitor a system's activity.

This enables a virus to infect files when they are used and spread easily. Instead of searching for the carrier files, the virus just scan files as they come through memory. There are several ways that viruses use for replication. However, it is common for the virus code to get executed when the carrier object is being used. Viruses that infect program files may attach the virus code to the beginning or the end of the program file, and patch the entry point so that when the program is run the virus code is executed first.

2.4.2 Payload routine

The payload of a virus is the part of the software that delivers the malicious code and causes the damage to computing devices (Joshi & Patil, 2012). In other words, the payload is the part of the code that contains the intended purpose of the virus and the conditions of execution. For example, a payload might be the part of the virus code that monitors keyboard to spy on passwords and email the password to an anonymous email address (Van der Made, 2003). A payload announces the presence of the virus by displaying a pop up message or displaying graphics, play music or videos in order to catch user's attention. The next section discusses the various ways in which viruses can be classified.

2.5 Classification of computer viruses

Computer viruses can be classified according to their different characteristics. The classification may be based on infection mechanism, targeted environment or concealment technique (Rad et al., 2011). A virus can either be a resident or non-resident virus when classified by infection mechanism. Based on virus classification by targeted environment, a virus can be classified as a file infector, boot sector infector, multipartite or macro infector (Dhruw et al., 2016). This dissertation is concerned more about derived virus and as such, it discusses more on virus classification by concealment technique as compared to the others. The next sub-section discusses some of these concealment techniques.

2.5.1 Classification by concealment technique

This is the technique that a virus uses to hide itself in order to avoid detection. The figure below shows how computer viruses are classified based on concealment techniques according to Singla et al. (2015).

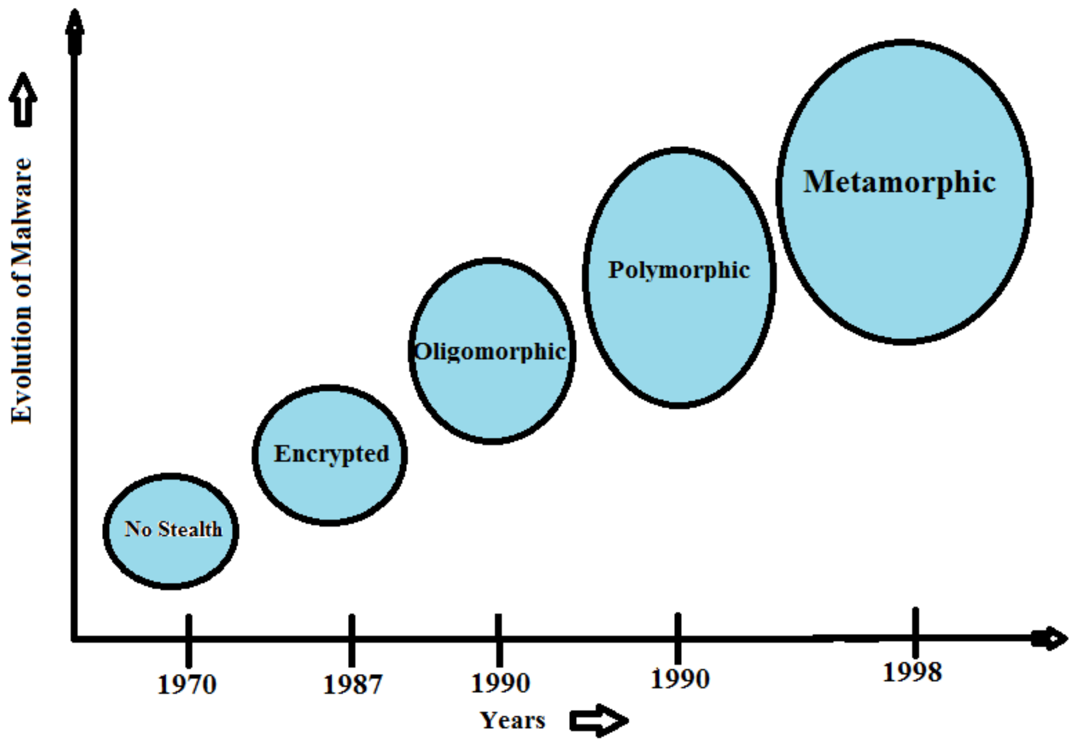


Figure 2.2: Evolution of concealment techniques (Singla et al., 2015)

The early viruses were developed without any encryption mechanism that could make their detection difficult. Therefore, it was easier for the anti-virus companies to get good detection results for the early viruses. Virus writers started to use different techniques to create more sophisticated viruses such as encrypted virus, oligomorphic virus, polymorphic and metamorphic viruses that would evade detection. The aim was to try and frustrate the effort of anti-virus companies. The next sub-sections discuss each of these concealment techniques in detail.

2.5.1.1 Encrypted virus

This is one of the first and easiest methods that virus writers used to hide the functionality of virus code. An encrypted virus usually consists of two parts. The first part is a small code that encrypts and decrypts. The second part is the encrypted part of the code (Rad et al., 2011). According to Edward & Slulason (1990), the major aim of virus writer that writes encrypted virus is to have a virus that cannot be easily detected by string matching signature-based virus detection mechanisms. The figure below shows the structure of an encrypted virus

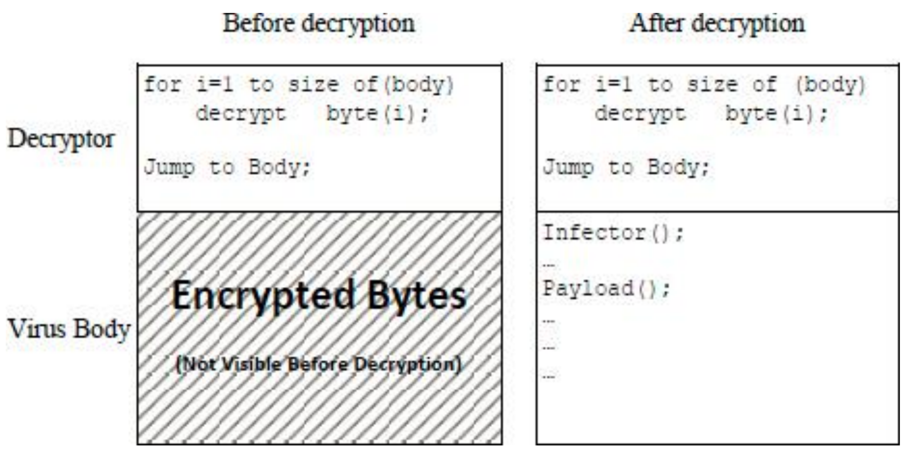


Figure 2.3: Structure of an encrypted virus (Rad et al., 2012)

The figure above shows the structure of an encrypted virus before and after decryption. The body of an encrypted virus which contains the payload is not visible before decryption. It only becomes visible after decryption when the virus is ready to execute its payload and infect a new target. The first part creates a random encryption key that is used to encrypt the remainder of the virus. When an infected program is invoked, the virus uses the stored random key for decryption. When the virus replicates, a different random key is selected. For every instance, the virus is encrypted with a different key. Hence, there is no constant bit that identifies an encrypted virus (Dhruw et al., 2016).

2.5.1.2 Oligomorphic virus

Unlike an encrypted virus that uses a decryptor, an Oligomorphic virus randomly picks one decryptor from a pool of decryptors to be used in its new variants (Rad et al., 2012). Therefore, detection of Oligomorphic viruses with signature scanning is more difficult.

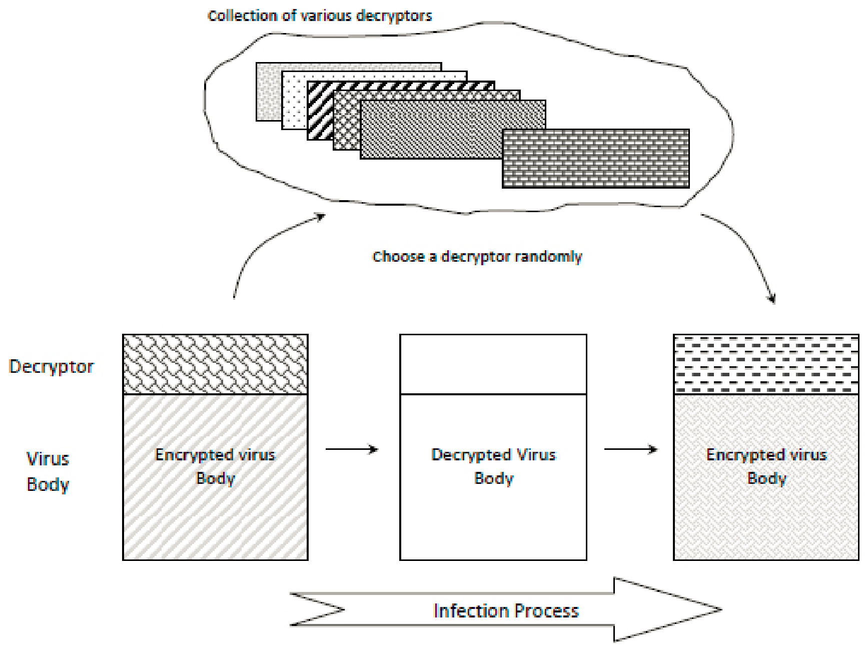


Figure 2.4 : Structure of an Oligomorphic virus (Rad et al., 2012)

The figure above shows the structure of an Oligomorphic virus with its multiple decryptors. Before decryption, the virus body is encrypted. When the virus discovers a new target to infect, it choses one of the decryptors and decrypts the virus body with the chosen decryptor after which infects. A different decryptor which is selected from the pool of decryptors is used at every instance of the virus.

2.5.1.3 Polymorphic virus

This type of a virus uses a mutation engine to change its form and develop a different signature at every infection. The mutation engine is used to modify some pieces of the virus body code (Rad et al., 2011). The modification is done with some mutation techniques such as insertion of junk codes or substitution of instructions. This is done to make detection by signature matching difficult. For example, an original signature of a particular polymorphic virus might be “881600808826000dcd13cd19". This virus can produce a derivative by adding a No Operation (NOP) instruction. A NOP instruction is an assembly language command that does nothing (Silva et al., 2015). The NOP instruction is used to allow future modification of code without rewriting or recompiling it (Clements, 2014). By adding a NOP instruction to the above polymorphic virus code, the signature becomes "88160080908826000d9090cd13909090cd19”. This same virus can further produce another supposedly “new" virus signature by adding a JUMP instruction. The JUMP instruction transfers program execution flow by jumping from one location to the other within the program (Kahanwal, 2013). By adding a JUMP instruction to the above polymorphic virus code, the signature becomes “eb10908826000deb0f9090cd13909090cd198816008088ebed" (Silverman, 2001). This now looks very different from the original signature and detection by signature matching becomes difficult.

2.5.1.4 Metamorphic virus

Unlike a polymorphic virus that mutates by changing some components of its body, a metamorphic virus mutates all of its body. A metamorphic virus has the ability to edit, rewrite and recompile its whole virus code to create another variant with a different body but still has the same functionality (Ling & Sani, 2017). For example, the signature of a particular metamorphic virus before an instruction replacement might be “558BEC8B760885F6743B8B7E0C09FF743431D2". The signature becomes “55545D8B760809F6743B8B7E0C85FF743428D2" after some instructions replacement. Just by making some changes to the instructions, another supposedly “new" virus signature is created. According to Ling & Sani (2017), a metamorphic virus contains 80% metamorphic engine and 20% malicious code.

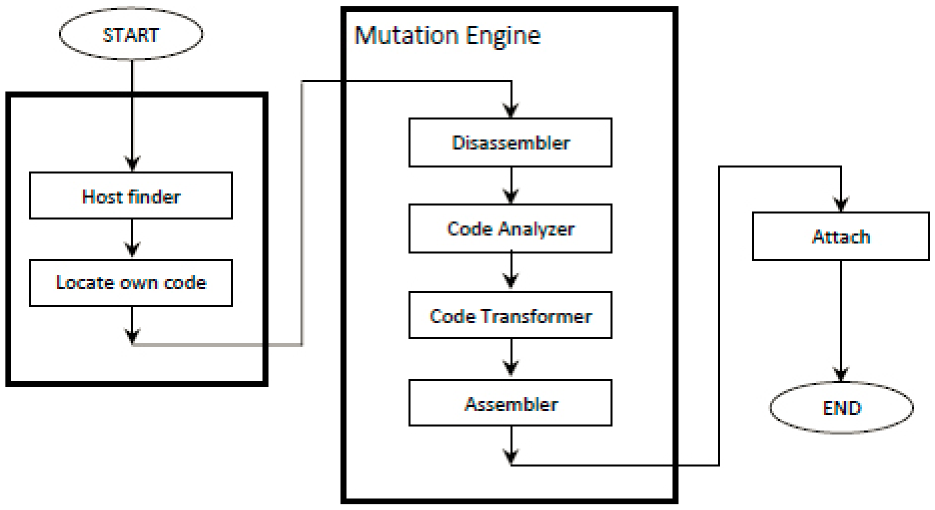


Figure 2.5: Structure of replicator and mutation engine in metamorphic virus (Rad et al., 2012).

As shown in the figure above, a metamorphic engine contains the following parts; a disassembler, a code analyser, a code transformer and an assembler. A metamorphic virus locates its code after discovering a new host, the code is passed to the mutation engine where it is disassembled, analysed, transformed and re-assembled again. Thereafter, the virus attaches itself to the new host but now with a different look. Hence, there is no constant signature for a metamorphic virus. Having classified computer viruses based on concealment techniques, the next section discusses some of the mutation techniques used by modern viruses.

2.6 Computer Virus Mutation Techniques

Modern computer viruses use different mutation techniques in order to make their code difficult to understand, analyse and in turn avoid detection by coming up with different signatures at every infection. The following are some of the mutation techniques used today by many virus writers.

2.6.1 Junk or Dead code insertion

The insertion of junk code is one of the easiest solutions to modify the binary sequence of a virus program. Junk insertion does not affect the functionality and behaviour of the code (Rad

& Masrom, 2010). These additional instructions do not change the content of CPU registers or memory. Hence, they are similar to a NOP instruction. Junk or dead code is also referred to as garbage instructions. For example, adding a zero to a variable or a register, or assigning a register value to itself, do not have any effect on the results of execution, yet this has effect on the virus detection. The tables below show how a particular virus produced two variants using junk insertion according to Rad & Masrom (2010).

Table 2.2 : Version 1(Rad & Masrom,

Binary Opcode	Assembly Code
C7060F000055	**mov** [**esi**], 5500000Fh
C746048BEC5151	**mov**[**esi**+0004],5151EC8Bh
Signature:	
C7060F000055C746048BEC5151	

Table 2.3: Version 2(With junk insertion)

Binary Opcode	Assembly Code
BF0F000055	**mov edi**,5500000Fh
893E	**mov** [**esi**],**edi**
5F	**pop edi**
52	**push edx**
B640	**mov dh**,40
BA8BEC5151	**mov edx**,5151EC8Bh
53	**push ebx**
8BDA	**mov ebx**,**edx**
895E04	**mov** [**esi**+0004],**ebx**
Signature:	
BF0F000055893E5F52B640BA8BEC5151538BDA895E04	

As seen in the tables above, the two variants have different signatures due to junk codes that are inserted in the second version of the virus but both variants perform same operation. Both transfer two double words into memory address specified by *esi*. Hence, there cannot be a signature that will identify the above virus and detection with signature scanning will be difficult.

2.6.2 Variable substitution

The register or memory variables are exchanged in different instances of a virus (Rad et al., 2012). This method does not change the behaviour of the code but it modifies its binary sequence. Hence, making each variant looks different from the former. The tables below show register substitution in variants of a particular virus according to Rad & Masrom (2010).

Table 2.4: Version 1(Rad & Masrom, 2010)

Binary Opcode	**Assembly Code**
5A	**pop edx**
BF04000000	**mov edi**,0004h
8BF5	**mov esi**,**ebp**
B80C000000	**mov eax**,000Ch
81C288000000	**add edx**,0088h
8B1A	**mov ebx**,[**edx**]
899C8618110000	**mov** [**esi**+**eax***4+00001118],**ebx**
Signature: 5ABF040000008BF5B80C00000081C2880000008 B1 A899C8618110000	

Table 2.5: Version 2 (With variable substitution)

Binary Opcode	**Assembly Code**
58	**pop eax**
BB04000000	**mov ebx**,0004h
8BD5	**mov edx**,**ebp**
BF0C000000	**mov edi**,000Ch
81C088000000	**add eax**,0088h
8B30	**mov esi**,[**eax**]
89B4BA1811000	**mov** [**edx**+**edi***4+00001118],**esi**
Signature: 58BB040000008BD5BF0C00000081C08800000 08B3089B4BA18110000	

Some register variables in the tables above are substituted which leads to the creation of another supposedly "new" virus because they have different signatures which is as a result of the variables substitution. For example, register *edx* on the first line of version 1 has been replaced with register *eax* in version 2. This causes some changes in the binary sequence and produced a new signature. Hence, detection with signature scanning will be difficult.

2.6.3 Instruction replacement

This method replaces some instructions with their equivalent instruction. This is like solving same programming task with different code instruction. This approach could be used to produce variants of a particular virus since the opcode pattern changes due to instruction replacement (Venkatachalam, 2010). Hence, the variants of the virus have different patterns of opcode but perform the same functionalities. Tables 2.6 and 2.7 show how a particular virus produced two variants as shown in Rad & Masrom (2010).

Table 2.6: Version 1(Rad & Masrom 2010)

Binary Opcode	**Assembly Code**
55	**push ebp**
8BEC	**mov ebp**, **esp**
8B7608	**mov esi**, **dword ptr** [**ebp** + 08]
85F6	**test esi**, **esi**
743B	**je** 401045
8B7E0C	**mov edi**, **dword ptr** [**ebp** + 0c]
09FF	**or edi**, **edi**
7434	**je** 401045
31D2	**xor edx**, **edx**
Signature: 558BEC8B760885F6743B8B7E0C09FF743431D2	

Table 2.7: Version 2(With instruction replacement)

Binary Opcode	**Assembly Code**
55	**push ebp**
54	**push esp**
5D	**pop ebp**
8B7608	**mov esi**, **dword ptr** [**ebp** + 08]
09F6	**or esi**, **esi**
743B	**je** 401045
8B7E0C	**mov edi**, **dword ptr** [**ebp** + 0c]
85FF	**test edi**, **edi**
7434	**je** 401045
28D2	**sub edx**, **edx**
Signature: 55545D8B760809F6743B8B7E0C85FF743428D2	

As shown in the two tables above, some instructions are replaced with their equivalent. For example, instruction *mov ebp, esp* is replaced by *push esp* and instruction *test esi, esi* is replaced with *or esi, esi*. The instructions replacement changed the opcode pattern. Hence, the variants have different signatures and detection with signature scanning becomes difficult.

2.6.4 Instruction transposition

In this method, the instructions in the virus code are reordered in a random fashion and control flow is adjusted to make it execute in the same order. This is accomplished by providing labels for each reorder and then using conditional jump instructions to jump the control to the labels (Rad et al., 2012). Thus, the instructions are reordered inside the code without altering the control flow. Figure 2.6 shows an example of instruction transposition in a particular virus.

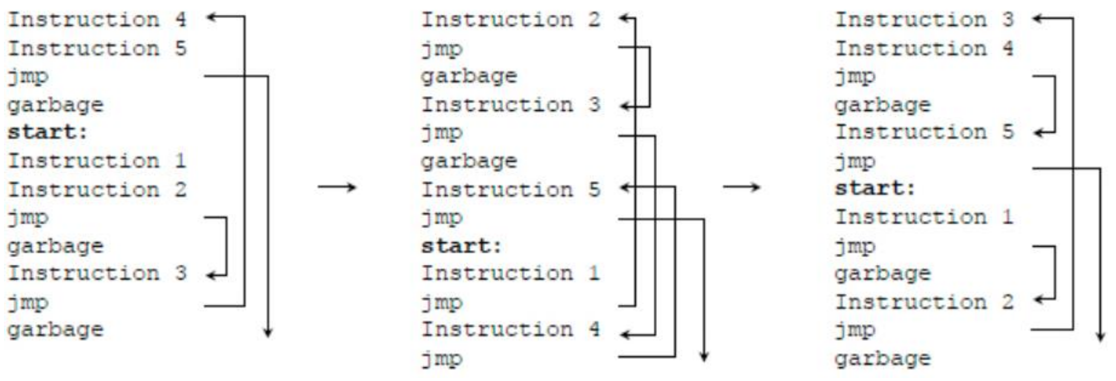


Figure 2.6 : Instruction transposition (Rad et al., 2012).

The first part of the figure shows the instructions in order 4, 5, 1, 2, 3. The second part of the figure shows the instructions in order 2, 3,5,1,4 and lastly the third part shows the instructions in order 3, 4, 5, 1, 2. Although the virus code is reordered, it is important to note that the control flow still remain the same. This is achieved by the several jump instructions introduced. A close look at the figure shows that control flow execution for the above three virus code is start, execute instruction 1, 2, 3, 4 and 5.

2.7 Conclusion

Over the years, computer viruses have changed from those that can be easily detected to those that replicate and cause unwanted changes within a computer system. Virus writers keep coming up with different techniques to make virus detection difficult. It started with encrypted virus and is currently on metamorphic virus. It is observed that as computer virus evolves, there has been an advancement in the concealment technique used and therefore making the latter always more difficult to analyse and detect than the previous. It is difficult for the current virus detection methods to detect today’s viruses because by the time they get detected, they would have already changed one or more of its components to look different from its predecessors. This has contributed to the huge number of viruses available today.

Amongst other things, the next chapter discusses virus detection techniques that are currently in use and also state their advantages and disadvantages.

CHAPTER THREE

LITERATURE REVIEW

3.1 Introduction

The previous chapter discussed the background of computer viruses, their anatomy, the evolution milestones and computer virus classification. Virus code obfuscation techniques were also discussed in the previous chapter.

This chapter provides an overview of the virus detection techniques that are currently in use and states their advantages and disadvantages. It then gives a review of research works done on virus detection. Ever since the first computer virus was found in the mid-1970s, several research works have been conducted to detect and remove virus from infected files and recover the files back to their original state. Anti-viruses are used for this purpose. The main functions of an anti-virus program are virus detection, file protection, removing virus from infected files and recovering damaged files (Mishra, 2010). Previous research works has resulted to the development of a variety of anti-virus programs that are currently in use today. The next section discusses some of the anti-virus software detection techniques.

3.2 Current anti-virus software detection techniques

The major methods used by anti-virus software for virus detection are as follow: signature scanning, integrity checker, heuristic scanner, anomaly based detection and CPU emulation (Mishra, 2010). In the following section, we discuss each of these techniques.

3.2.1 Signature Scanning

Signature scanning is regarded as the oldest and most common virus detection technique which is still being used today by many anti-virus software (Kamarudin et al., 2013). It involves a careful analysis of a particular virus with the aim of detecting a specific string of bytes called a virus signature. A virus signature is a unique string of bytes that correctly identifies a particular virus (Kamarudin et al., 2013). Virus signatures of the actual viruses are stored in the database of an anti-virus program which is then installed on the computer system. The anti-virus software works by comparing signatures of files in a computer against the signatures of viruses that are stored in the anti-virus software database. If a signature of any file matches a signature in the installed anti-virus database, such a file is declared infected and necessary action is taken whether to delete or quarantine it (Kakad et al., 2014). This method of virus detection is effective and gives accurate result but only to known viruses. The disadvantages of this method includes but not limited to its inability to detect new viruses and also its inability to detect derived viruses (i.e. viruses that use modern techniques to change form in order to avoid detection) (Hamza & Hussain, 2014). Another major drawback of signature scanning is the time lag between virus creation and detection. This is the amount of time it takes to get the signature that can be used to detect a virus after its creation. During this period, a new virus can easily spread and cause damage on billions of computing devices without being detected. The virus can continue to infect and cause damage for as long as it takes before the signature is found. Figure 3.1 below shows the steps of signature scanning as described by Mishra (2010).

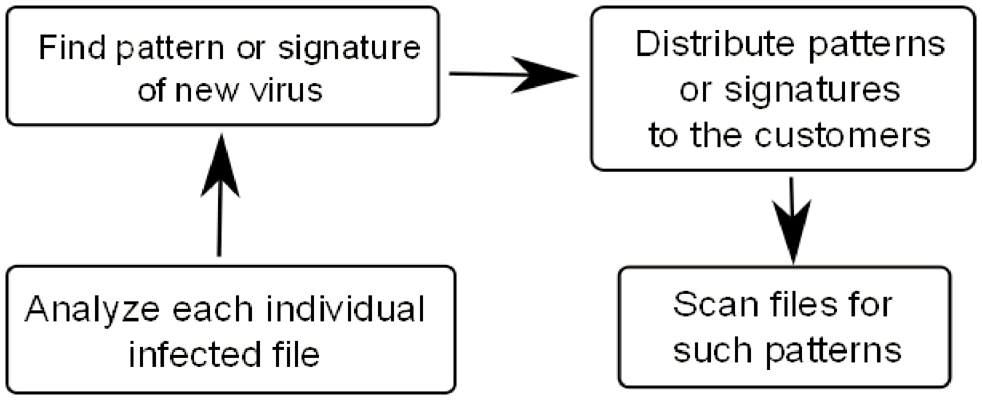

Find pattern or signature of new virus
Distribute patterns or signatures to the customers
Analyze each individual infected file
Scan files for such patterns

Figure 3.1: Steps of signature scanning (Mishra, 2010)

The figure above shows that to obtain signature for a new virus, the virus must have infected some files and that is where the steps to finding a virus signature begins. The infected file is analysed carefully in order to get the string of bytes (signature) that will correctly identifies the virus. When the signature is found, the anti-virus company distribute the newly found virus signature to their customers by uploading it to their central database. Each customer is expected to download the new virus signature by connecting to the anti-virus central database and download updates to the virus signature database on user computer. Thereafter, customers scan files on their computer for the newly found virus pattern.

3.2.2 Integrity Checker

An integrity checker keeps a small hash value or fingerprint of uninfected programs and files in a secured location at the beginning when a system is presumed to be clean from virus infections. It then detects the presence of a virus by comparing the hash value of a file with the hash value of its clean version. If there is no difference between the two hash values then the file is deemed to be clean. Otherwise, if the two hash values are different, then the file is considered to be infected. This method works on the basis that it is impossible for a virus not to make any change to its host program or file. Integrity checker can detect any type of virus. This could include new, unknown and even derived viruses. One of the major drawbacks of integrity checker is that when a change in a file or program is detected, it is often difficult to determine whether the change was induced by a virus or by a user changing the file. Hence, false positive is possible with this method. Another drawback of an integrity checker is its ineffectiveness if the computer has already been infected at the time when the program hash values are first calculated (Mishra, 2010).

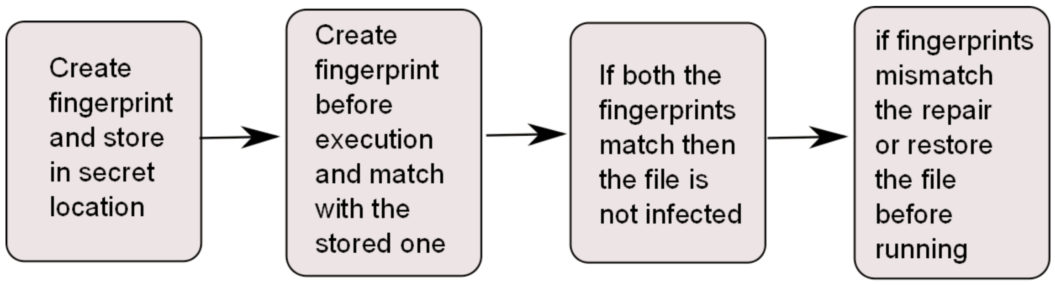

Create fingerprint and store in secret location
Create fingerprint before execution and match with the stored one
If both the fingerprints match then the file is not infected
if fingerprints mismatch the repair or restore the file before running

Figure 3.2 : Steps of an integrity checker (Mishra, 2010)

As shown in the figure above, the fingerprint or hash value of an application is created and stored in a secret location when the system is assumed to be uninfected. Before the application executes, its fingerprint is checked again and it is compared with the stored fingerprint. If the two fingerprints match then the application is not infected. Otherwise, if the two fingerprints do not match, the application is restored before it runs.

3.2.3 Heuristic Scanner

A heuristic scanner anti-virus program examines a target program and analyses its code to determine if it appears to be like a virus. A heuristic engine detects the commands within a program that are not typically found in normal application programs. This may include replication commands that are normally associated with viruses. This approach is relatively fast. However, heuristic scanners must look for a large number of different ways which virus-like operations may be implemented in order to reliably detect a malicious behaviour. This is because viruses may open files by using different codes and methods (Mishra, 2010).

3.2.4 Anomaly based detection

Anomaly is defined as something that is not normal. Anomaly based detection monitors the processes or activities on a host machine for any abnormal activity (Kakad et al., 2014). This could be activities that attempt to reformat a disk or move a file into one of the operating system folder which is generally not the activity of a normal program. If any such activity is found, the system flags for a potential threat. This method is capable of detecting new and derived viruses. However, false positive is possible with this approach (Kakad et al., 2014).

3.2.5 CPU / Code Emulation

This is an emulation based technology that loads a target program into a CPU emulator. The CPU emulator acts as a simulated virtual computer. As the target program is emulated, its malicious operations are identified and catalogued. An anti-virus software is used to determine if the target program is a virus or not from the catalogue of malicious operations. This method can detect new and derived viruses. However, this method is ineffective for viruses that activate only when a certain arbitrary conditions are met (Mishra, 2010). For example, a logic bomb that is programmed to execute on a particular date, if the condition is not met, a CPU emulator will not observe the infectious behaviour. Hence, a virus can go undetected.

In summary, the observation is that within the field of anti-virus, there is an increasing number of viruses released daily. These have really stretched the traditional virus detection techniques. This has increased the demand for techniques that identify new and derived viruses. The ineffectiveness of the signature based scanning method for detecting new and derived viruses call for an immediate solution. Some of the above detection methods listed above can handle this, but they are still far from perfection.

The next section reviews previous research works.

3.3 Review on previous research works

Several research works have already been conducted to find solutions to the problem of virus detection. These are discussed below. Some of these research findings are based on providing a solution to derived viruses while some are based on detecting new viruses. A new virus is a virus that is created from scratch with a new mechanism that has never been seen before. A derived virus is any virus that modifies itself using a mutation engine and come up with a signature different from its original or predecessor at every infection in order to avoid detection by signature scanning.

3.3.1 Research works for detecting new computing viruses

A method of virus detection based on immune theory and N-Gram analysis was proposed by Qin et al. (2009). Qin et al. (2009) uses N-Gram to extract the common features of known virus programs. This approach is used on the assumption that viruses have certain characteristics in common. The most frequent N-Gram is used as gene library. Immature cells are generated from the gene library and some are generated randomly. The immature and randomly generated detector experience a self-tolerance period. Any detector that matches any self antigen is eliminated (negative selection). The immature detectors that survive maturity are activated if viruses are detected. The activated detectors are transferred into memory so that they can quickly fight a virus if it re-surfaces. The detectors that are not activated are eliminated. This research work does not show how new detectors are generated. Hence, continuity of the immune system might be a problem.

An approach of virus detection that uses a high speed time delay neural networks was proposed by El-Bakry (2010). Neural networks are first trained in time domain to classify virus from non-virus. In the virus detection phase, each sub-matrix in the incoming data is tested for the presence or absence of a virus. At each position in the incoming input matrix, each sub-matrix is multiplied by a window of weights. The outputs of neurons in the hidden layer are multiplied by the weights of the output layer. If the final output is high, it means that the sub-matrix under test contains a virus. There is no experiment to show how the proposed method really works. Therefore, its efficiency and accuracy cannot be determined.

Zolkipli & Jantan (2010a) discusses a framework for malware detection using a combination of techniques i.e. signature based and genetic algorithm. The framework uses signature based technique as first defence for malware attack. The genetic algorithm is then used to select chromosomes from the signature based on selective value from the population. New chromosomes are produced according to the combination of the bit string from existing chromosomes in the existing population. Finally, the signature generator captures the malware behaviour and analyse the gene using genetic algorithm detection module. This is to generate the signature pattern and update it into malware. There is no experiment that shows how the framework works. Therefore, its efficiency cannot be guaranteed.

Yunlong et al. (2012) propose malicious code detection based on least square estimation. This method provides a pre-processing approach that sequences virus behaviour based on time correlation and subject-object orientation. Yunlong et al. (2012) proposes a threat judging method based on least-square estimation. An expert experience approach is used to introduce the code detection and then simulate experts to judge the threat values of malicious code. This approach can be used to complement the current antivirus software. However, training the experts might be a challenge. It was stated that this research is 87% accurate in detecting malware.

Han et al. (2012) propose a method of detecting malicious or illegal code by monitoring task behaviour of software applications. The top-down calling sequence of software applications within a computer is developed into a behaviour resource tree. When a user runs an application, if the calling sequence of the task deviates from the sequence in the behaviour resource tree, it is declared as illegal. Han et al. (2012) also came up with illegal code judgement rule based on known behaviours of malicious code. The challenge of this method is how to build behaviour resource tree for new application software. Moreover, legal activities such as a user updating software can cause false positive. Furthermore, one other disadvantage of rule based systems is that the system is as good as the defined rules. Anything that falls outside or deviates from the stated rules cannot be picked.

Multiple sequence alignment and an artificial neural network for malicious software detection were proposed by Chen et al. (2012). Chen et al. (2012) uses multiple sequence alignment techniques from bioinformatics to align variable length computer viral and worm code. An artificial neural network is then used to learn the critical features to determine the class of the aligned patterns. The algorithm for alignment involves some sort of gap insertion. However, there is no guarantee that the same number of gap insertions is made thereby making length of virus set alignment different from worms set alignment. An approach that does not require gap insertion is needed for the result to be trusted. This experiment was reported to be 88.89% accurate if the viral sequences are doubly aligned.

A multi-component feature for detecting malware by classifying portable executable file as benign or malware is proposed by Vinod et al. (2013). The multi-component feature comprises of portable executable metadata, principal instruction code, mnemonic bi-gram and prominent unigrams. Portable executable of malicious programs is first unpacked using generic unpackers. Unpacked samples are disassembled using a disassembler which generates assembly code. A parser is then developed to filter out mnemonics from assembly files. These processed files are used to extract prominent unigrams and bi-grams. This header information is filtered out from each malware and benign executable. The extracted features are used to generate multi-component feature. A multi-component feature is used to create a feature vector table. The table is used to construct a classification model for malware and benign family. The performance of classification model is evaluated by monitoring the evaluation metrics. The accuracy of this proposed model in percentile is not stated. However, Vinod et al. (2013) said that the proposed model performs better than some other model that exist in literature.

Khorsand & Hamzeh (2013) propose an idea of detecting malware by using portable executable (PE) header to train prediction by partial matching (PPE) model. PE file is a format of executable binaries that the windows operating system uses. PPE is a model that keeps frequent occurrence of symbols. Its encoder encodes each input symbol with probability provided by its map. PPE is trained by using the header of programs and strings in the body of the program. The experiment shown in this work uses small subset of binary content of files for modelling. Since, it is generally known that malware is made up of both malicious and benign code, using a small subset of binary content might decrease the effectiveness of the model.

A computer virus detection system based on an artificial immunity concept was proposed by Hamza & Hussain (2014). The system uses an algorithm that was developed based on clonal selection in an artificial immune system. The algorithm uses a set of downloaded virus signature to create virus clones and infects some already benign files with virus clones. This approach then uses various files to test the algorithm for virus identification and elimination. The algorithm shows how virus clones can be created and the created clones are used as prediction of future viruses. However, the algorithm does not show how virus can be detected. An average of 95% of detection rate was reported for this experiment. It must be mentioned that Hamza & Hussain (2014) did not specify if this include existing, derived or new viruses.

A method of detecting derivative malware samples using de-obfuscation-assisted similarity analysis is proposed by Wrench & Irwin (2015). The method uses a decoder developed to de-obfuscate and normalise code prior to analysis. The first part is achieved by using a decode function in php. This function scans the entire code for functions that obfuscate a segment of code. If any is found, it is processed and the normal code is extracted and replace what it represents. Thereafter, code normalisation is done. After de-obfuscation, both extracted and raw data are passed to a matrix module to produce matrices that represent the observed similarity between all given samples based on specified measure of similarity. The disadvantage of this system is that a virus written in any other language other than PHP cannot be detected. This is because the system scans for only php functions that obfuscate a segment of code.

A Neural Network Artificial Immune System for malicious code detection was proposed by Golovko & Bezobrazov (2015). The system uses a combination of Artificial Immune System and Artificial Neural Networks concepts. It works according to basic principles of the artificial immune system whereby a neural network is used as the immune detector to detect a malicious pattern by means of the analysis and structure of the executable code. The trained neural networks are then accepted as part of the Artificial Immune System component. The work of the neural network detector is to randomly scan the files and classify them as legitimate or malicious. If a detector is able to pick a malware, then cloning and mutation will occur on the detector that detects the malware. This is to be able to quickly fight the malware and have the detector registered in the memory in case the same malware shows up again. The idea of combining an artificial immune system and artificial neural network sounds great. However, it is stated that an increase in the number of malware leads to decrease detection rate. Also, the neural network used requires a unique node in the hidden layer for all inputs. This is often too large to work on real world problems. Even though the accuracy of the system was not stated, it was reported that the system performed better and more accurate than some of the current antivirus programs in use.

Andree & Nhien-An (2016) uses statistical methods, Naïve-Bayes algorithm and N-grams to find out if there is significant data that could be used to classify malware in control flow change. In order to find a discriminator between a malware and good software, Andree & Nhien-An (2016) first calculate and compare statistical values of the median, variance, variance coefficient and spread for both malware and good software. The second approach used by Andree & Nhien-An (2016) to find a discriminator is a Naïve-Bayes classifier. The classifier is first trained with sample datasets comprising of raw jump length data for both malware and good software. Thereafter, the classifier test unknown software against the data that has been learned by the classifier. The third approach used by the authors is based on the extraction on N-grams of words from a text. The N-grams of sample datasets are extracted and saved to a database for further comparison. Test samples are processed the same way the sample datasets are processed and saved into another database. The categorization test searches for occurrence of the to-test n-grams in the sample datasets and check for similarities. The data found in this work is not usable and its likelihood is too low to make a decision.

A combination of an artificial neural network and artificial immune system for virus detection was also proposed by Khang et al. (2016). This research describes three main components for the proposed model. The first component called the training data consists of a set of viral and benign files used for training and detecting stage. The second component is the population of detectors called antibodies. These are the ANN that are trained with the first component of the model and are used as detectors for recognizing malicious and benign files. The third component is the artificial immune network (Ainet) which is used for generating mature detectors. The proposed model results into set of ANN that can be used as computer virus detector. An average detection rate of 87.98% with an acceptable false positive rate was reported for this work. However, Khang et al. (2016) did not specify if the detection rate includes existing, derived or new viruses.

The next sub-section reviews research work that has been done for detecting derived viruses.

3.3.2 Research work for detecting derived computing viruses

A method of detecting virus mutations via dynamic matching was proposed by Feng & Gupta (2009). The research work is based on the fact that virus developers do not often write new codes in creating newer virus. Rather, the code of an existing virus is altered in such a way that the behaviour remains the same but the flow of the code differs. This makes it difficult for a signature to detect the supposedly ‘new’ variants. In their work, the behaviours of a virus are studied using CPU emulator and saved in a database during runtime. The database continues to grow based on the behaviours of viruses at run time. When a new virus is encountered, it is checked to see if it has behaviour similar to any in the database. If it does, then it will be confirmed as the mutated version of a virus that has been encountered before. The idea is good but since a CPU emulator is used, it will be difficult to get the behaviours of viruses that are to be executed based on some conditions. It was stated that the experiments performed showed that their work is effective in detecting the variants of six viruses out of seven viruses used for the experiment.

Daoud (2009) propose a model for detecting metamorphic viruses based on Artificial Immune System. The model uses three different layers of defence to detect a virus by imitating the human immune system. This model defines what self and non-self means at each layer. Moreover, an algorithm is developed for each layer based on the self and non-self idea. At the first layer, self is the hash of all trusted files and non-self is the hash strings of known viruses. The algorithm in this layer serves as the entry point to the computer files. The hash value of the file is calculated, if the value matches one of the self hash strings, then the file is clean. If the hash string matches one of the non-self then it is an infected file. Otherwise, the file is sent to the second layer to perform additional checks. At the second layer, self is the trained neural network on the flow graph of safe assembly codes. Non-self is the trained neural network on the flow graph of virus assembly codes. The algorithm on this layer disassembles the file, encodes the data and represents the flow graph by two dimensional matrix. If the result of the file sent from layer one is similar to self flow graph then the file is clean. If the result is similar to non-self flow graph, then the file is infected. Otherwise, the file is sent to the third layer to perform additional checks. At the third layer, self is the trained neural network on the receptor data of safe assembly codes. The non-self is a trained neural network on the receptors data of the viruses' assembly codes. If the encoded data is similar to three of the self receptors, then the file is clean. If the encoded data is similar to three of the non-self, then, the file is infected. The disadvantage of this system is that it can produce high false positive and false negative if the computer files are infected before the hash string is first computed. Furthermore, the system still requires self and non-self updates for new files and viruses respectively. This experiment reported 86% detection rate of metamorphic virus.

An approach was proposed by Wang et al. (2009) which involves characterization of the generation of virus detectors under supervision. This approach collects correlation of instructions within a virus program. This approach is based on an Artificial Immune System model which uses a training set as a guide to generate a candidate virus gene library. The gene library is then upgraded to a virus detection gene library. It does this by deleting all the non-viral information with negative selection in artificial immunity. This allows legal programs in the training set to be memorized. The model then compares genes on a gene level (DNA), analyse suspicious program on individual level and make classification decision by using the entire gene library. It is stated that the drawback of the model is its inability to maintain or increase the diversity of the genes in the virus library. An average recognition rate of 94% for derived virus was reported for this experiment.

Zolkipli & Jantan (2010b) proposes a framework for malware identification and classification. The proposed framework therein consists of two activities which are behaviour identification and malware classification. Samples of malware are run in a windows virtual machine environment. This is to understand the characteristics and behaviour of the malware and also to know its purpose and function. The defined malware behaviour for each sample is used for malware classification. Related classification techniques are applied for identifying the shared behaviour of each malware family. The classification process is also optimized by using an artificial intelligent technique. However, it should be stated that this framework has not been implemented and its accuracy is not stated.

A dynamic approach using signatures from an API for detecting metamorphic malware is proposed by Vinod et al. (2010). The API calls made by a family of malware is used to form matrix. A principal component analysis is used to extract the most significant API calls called critical API calls. The frequency of the critical API calls is used to create a family signature for each malware family. The signature is saved in the database. To test if a particular malware belongs to one of the known malware family; the API calls of the signature are extracted and compared to that in the database. It is reported that the results from the experiments performed are in accordance with some other result in literature.

A neural network ensembly method for detecting computer viruses was proposed by Liu et al. (2010). The research work discusses how machine learning could be used for virus detection. The characteristics of a virus are first extracted for classification learning through the system calling sequence of a program. Different machine learning approaches are then used to construct the ensemble. The results of the machine learning are combined according to Dempster Shafers evidence theory to form the final output of the system. This research work only discusses how an artificial neural network can be trained but does not show how the output of the trained artificial neural network detects a computer virus. However, an astounding 97% accuracy was reported for this work when compared with other traditional ways. It must be mentioned though that Liu et al. (2010) did not specify if this include derived or new viruses.

Vinod et al. (2012) proposes a method of detecting variants of metamorphic malware using bioinformatics techniques effectively used for Protein and DNA matching. In their proposed method, multiple sequence alignment is used to arrange opcode sequence of malware. This is to determine similarity among malware samples and also to determine frequent occurring pattern in a malware family. The frequent pattern depicts maliciousness. It is reported that the result of the experiments is comparable with some of the anti-virus software that are commonly in use today.

A memory-based abstraction approach to handle obfuscation in a polymorphic virus was proposed by Nguyen et al. (2012). The aim of this research work is to handle obfuscation in polymorphic virus. The methodology involves abstracting the binary code based on their memory states and analyse the abstracted states to detect useless instructions. This method is proposed because in spite of any obfuscation technique employed, the actual malicious code, once executed produces the same memory state patterns when properly abstracted. However, the major challenge of this approach is the need to develop an abstract form that captures the memory states affected by each instruction when executed. As such, it is stated that the approach is not practically possible because of the high complexity.

Zhong et al. (2012) proposes a malware classification method based on similarity of function structure to determine a program as a variant of a known malware program. In their approach, programs are characterized according to the functions they contain. Functions are characterized based on their features. A database is developed based on the extracted features of the known malware. Unknown programs are tested against the data in the database created as mentioned above to determine the family that they belong to. When given an unknown program, it is disassembled, divided into functions, and their binary, strings and lists of called APIs are obtained through the same process described above. This work is reported to have virus variants detection rate of 78%.

Kamarudin et al. (2013) conducted an analysis on the effectiveness of signature based anti-virus software in detecting metamorphic viruses. In their work, Kamarudin et al. (2013) created a seed virus using a virus generator. Thereafter, the seed virus is run on their morphine engine which contains some code obfuscation techniques. This process is done to generate a family of metamorphic variants for the seed virus and it was conducted 20 times which gives 20 metamorphic variants of the seed virus with each generated variant different from one another. This explains the reason why detection of metamorphic virus is difficult with signature based anti-virus software. Kamarudin et al. (2013) also performed a similarity test on 4 generations of metamorphic variants and the original seed. It was discovered that the similarity decreases with higher generations. The first generation virus is similar to the virus seed with about 60% and the fourth generation virus has an average similarity of 19.7% with the virus seed.

A model to classify files into malware or benign was developed by Kuriakose & Vinod (2014b). They use feature selection methods such as term frequency, inverse document frequency among others to develop the model. Malware and benign portable executables are unpacked and disassembled to get bi-gram data set. The bi-gram dataset is used for training and testing. The experiment shows that the model can detect more synthetic metamorphic viruses with average detection rate of 92%.

Gang & Zhongquan (2014) developed a malicious code detection solution that is based on fuzzy reasoning. The research work adopts a comprehensive scheme to get behaviours in such phases as file structure analysis and function call identification. It also analyses the most common obfuscation technique that insert data after call instruction and provide an algorithm that identifies information calls. A dynamic fuzzy neural network decision system is then used in order to determine mutating and unknown malicious executable viruses. The system does not show how viruses are detected. However, 95.2% detection accuracy was reported for this work's experiment.

Kuriakose & Vinod (2014a) proposes a non-signature based approach using feature selection technique for the detection of metamorphic viruses. A classification model for identifying viral and benign samples is developed from the proposed model. The metamorphic samples are unpacked and disassembled. Thereafter, bi-gram mnemonics are extracted from each file based on prior studies. Relevant bigram mnemonics are selected from the total bigram feature space. The obtained feature space is further pruned using the feature ranking method such as Categorical Proportional Distance (CPD), Weight of Evidence of Text (WET), Term Frequency – Inverse Document Frequency (TF-IDF) and Term Frequency-Inverse Document Frequency.

3.4 Research Gaps

The review presented in this chapter shows that a virus detection techniques based on signature characterisation is the oldest and most commonly used technique (Kamarudin et al., 2013) and has many disadvantages such as human intervention in signature extraction, the need for users to regularly update the signature database (Vinod et al., 2010), the time lag between virus creation and detection and most importantly, the inability to detect new and derived viruses (Mishra, 2010). Some other approaches that use behaviour to detect the presence of a virus have high false positives while some are still far from perfection. Some of the methods proposed in the literature involve some error prone pre-processing and some of these are too complex to implement. Another approach that sounds promising might be to use the concept of artificial immune systems to enhance computing devices with their own immune system. However, training on how to differentiate non-viral files from viral files seems to be a big challenge. This is important in a computing system where application software is installed and uninstalled on a regular basis. Some research explored the use of artificial intelligence in detecting computing viruses but more research still needs to be done in order to fully explore more of the available branches of artificial intelligence. Furthermore, very few research efforts focused on detection of derived viruses and the ones that do considered metamorphic viruses rather than considering detection of all variants of existing viruses irrespective of the type of variant it is. These are the challenges faced by the current virus detection techniques and also the techniques that are proposed in literature.

3.5 Conclusion

The review shows that more emphasis needs to be laid on the detection of derived viruses. This is because the major challenge of virus detection is the ability of the virus to change its form by using any of the modern techniques in order to avoid being detected by most of the current available anti-virus software. The review also shows that artificial intelligence in the form of artificial neural networks still needs to be explored further. Therefore, this research focuses on investigating how artificial intelligence in the form of artificial neural networks might be used to increase the detection rate of derived viruses.

The next chapter presents the proposed model for computing derived virus detection.

CHAPTER FOUR

PROPOSED MODEL

4.1 Introduction

In view of the current problems of detecting derived viruses as identified in the previous chapter, there is a need for a better solution. The solution should be dynamic in nature and not dependent on pre-defined virus signatures. The solution should also learn from existing virus signatures and be able to predict and detect derived viruses thereof. Hence, a model is proposed base on these requirements.

4.2 Model

The model is as shown in figure 4.1. The proposed model comprises of various building blocks depicting different processes. The next sub-section discusses the processes in each block.

4.2.1 Virus signature database

This is the first block of the model. This block generates the dataset using the following steps:

4.2.1.1 Existing/known virus signatures

The first step in building a virus signature database is to obtain existing virus signatures. The existing virus signatures can be obtained from any source such as VX Heaven and nlnetlabs website. However for this particular research, the signatures were obtained from nlnetlabs website. The size of a single signature string can vary (Wang, 2008). This experiment considers virus signatures of 32 bytes, this is because our solution requires the input to be represented as a fixed-length feature vector. The sample of existing virus signatures downloaded is as shown in figure 4.2.

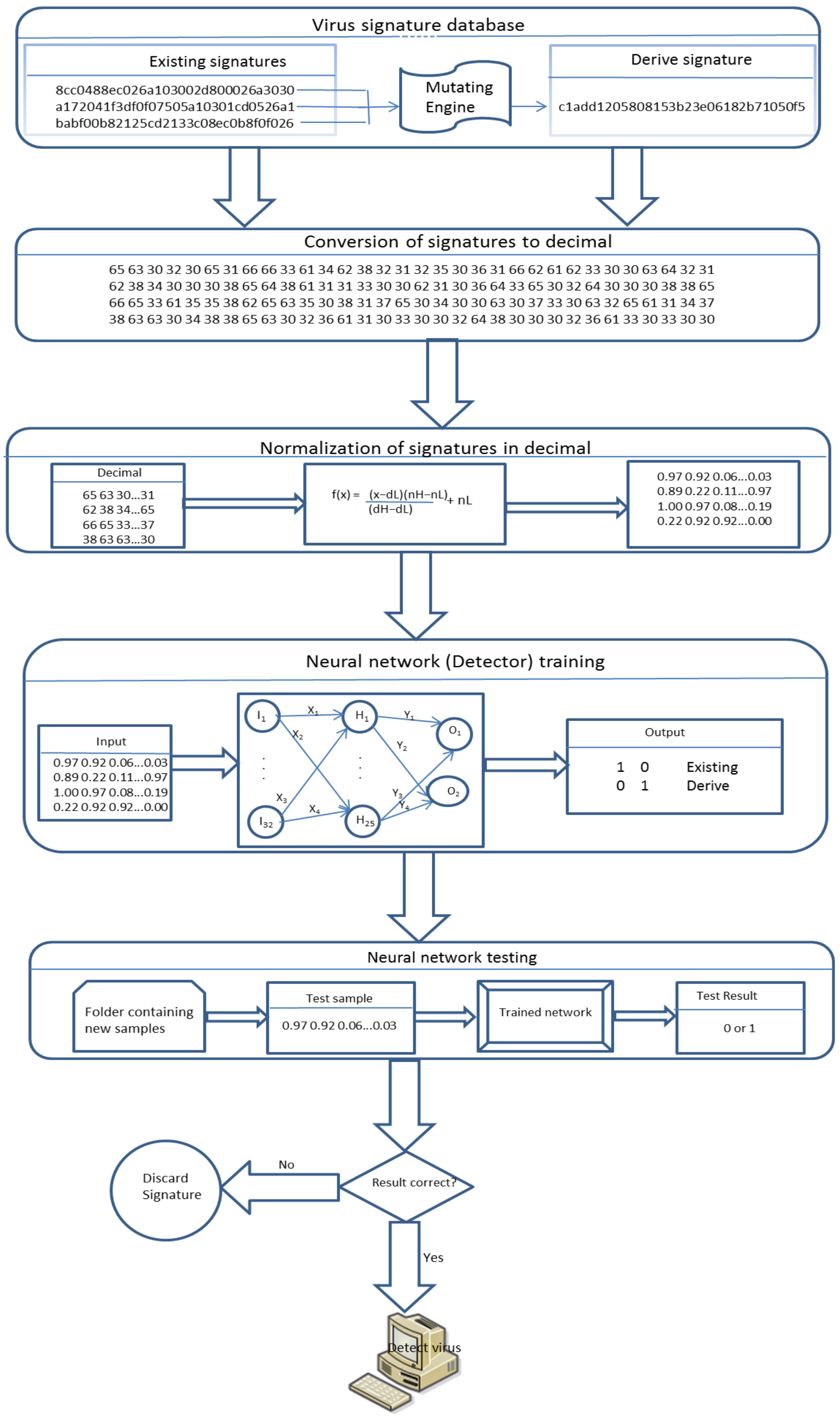
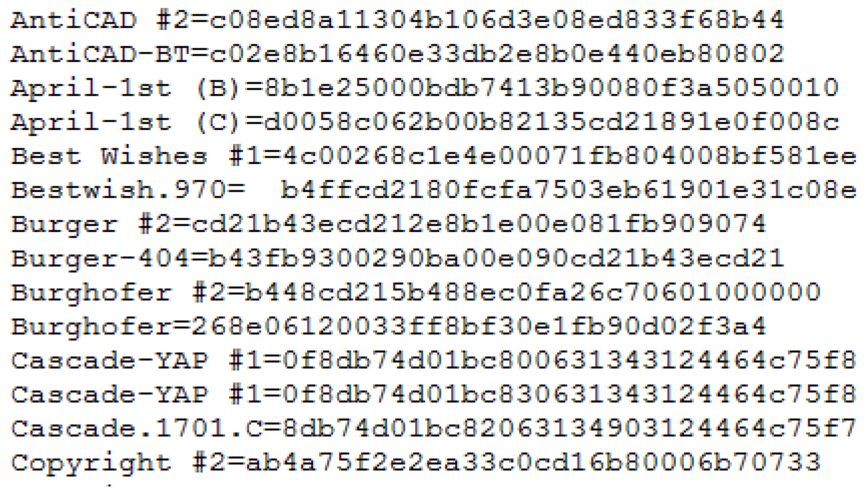


Figure 4.1: Virus detection Model



```
AntiCAD #2=c08ed8a11304b106d3e08ed833f68b44
AntiCAD-BT=c02e8b16460e33db2e8b0e440eb80802
April-1st (B)=8b1e25000bdb7413b90080f3a5050010
April-1st (C)=d0058c062b00b82135cd21891e0f008c
Best Wishes #1=4c00268c1e4e00071fb804008bf581ee
Bestwish.970=  b4ffcd2180fcfa7503eb61901e31c08e
Burger #2=cd21b43ecd212e8b1e00e081fb909074
Burger-404=b43fb9300290ba00e090cd21b43ecd21
Burghofer #2=b448cd215b488ec0fa26c70601000000
Burghofer=268e06120033ff8bf30e1fb90d02f3a4
Cascade-YAP #1=0f8db74d01bc800631343124464c75f8
Cascade-YAP #1=0f8db74d01bc830631343124464c75f8
Cascade.1701.C=8db74d01bc82063134903124464c75f7
Copyright #2=ab4a75f2e2ea33c0cd16b80006b70733
```

Figure 4.2: Existing virus signatures sample

4.2.1.2 Derive virus signatures

This is the second step of this block. A mutation engine is built and used here to generate derive virus signatures. This is achieved by combining sets of known virus signatures based on a pre-determined algorithm. The aim here is to simulate a derived virus signature using existing virus signatures and train the neural network with it. This is necessary since this research aims to train a neural network to be able to detect derived viruses based on the existing viruses. For example, given a group of existing virus signatures, we can generate a number of signatures. Derived virus signatures are generated up until the third generation using the built mutation engine. The process starts by arranging known virus signatures into groups based on their similarities. Each group can be argued to be a particular virus family consisting of related viruses. The virus signatures in a family are randomly combined to produce a derived virus signature for that family. The first derived signature is then added to the family. Thereafter, another randomization is conducted again on both the existing signatures and the newly generated derivative. This process is repeated until three new derivatives are generated. The same process is repeated for all the groups (families). For example, a family which originally has three virus signatures, the fourth signature for that family is generated by random combination of the three existing virus signatures. The newly created derive virus signature is added to the family and the family now consist four virus signatures. The fifth signature for that family is generated by random combination of the four virus signatures. The newly created derive virus signature is added to the family and the family now has five virus signatures. Lastly, the sixth signature for that family is generated by random combination of the five virus signatures. The newly created derive virus signature is added to the family and the family now has six virus signatures. All this is done by the mutation engine. Using this approach, the mutation engine was able to generate new virus derivatives of three generations grouped into a number of families.

4.2.2 Conversion of the virus signatures

The second block of the model is to convert the signatures obtained in stage 1 above from their strings representation to decimals forms. This is achieved with a pre-defined algorithm that converts string to decimal. This is necessary since neural networks are designed to accept numbers as their input (Heaton, 2011). It is also possible to represent the signatures in other numerical formats like binary and ASCII. However, this might introduce more complexity. For example, binary representation of each signature might generate lots of 0's and 1's which might not be easy to manage. For example, the first four characters of this signature "b840008ed8a11300b106d3e02d00088e" gives "0110001000111000 0011010000110000" in binary. Furthermore, ASCII representation might not be necessary since we have chosen to exclude special characters. An example of a particular virus signature in decimal is "30,64,38,34,30,36,33,65,32,30,38,38,38,65,30,61,34,30,30,61,65,64,38,32,30,38,66,38,34,61,38,32"

4.2.3 Normalization

The output from the block above is the signatures written in a decimal form. This is normalized to improve the efficiency of the model. There are two different categories of data entering the neural network for training. First is the numeric data and the other is the class each numeric input belongs. This two categories need to be normalized.

4.2.3.1 Normalizing signatures

Normalizing a numeric value is a process of mapping the existing numeric value to well-defined numeric range, such as -1 to +1 or 0 to +1. Normalization causes all of the attributes to be in the same range with no one attribute more powerful than the others (Heaton, 2011). Each virus signature used has 32 bytes and each of this is an attribute in which the neural network learns from. All virus signatures in decimal are normalized to values between +1 and -1 using the normalization formula below. This is because hyperbolic tangent activation function is used to scale the output.

$$F(x) = \frac{(x - dL)\,(nH - nL)}{(dH - dL) + nL}$$

The above equation normalizes a value *x*, where the variable *dH* represents the high value of the data, variable *dL* represents the low value of the data and the variable *n* represents the high and low normalization range desired i.e. -1 and +1.

4.2.3.2 Normalizing the classes

This is otherwise known as encoding because it is basically a way of representing nominal values. The nominal values here are existing signatures and derived signatures which are the two classes into which a signature can be grouped. One-of-*n* encoding is used here. This is because there are just two nominal classes to encode. Therefore, with one-of-*n* encoding, the neural network has two output neurons. Each of these two neurons represents each class (derived and existing virus signatures). A one (1) is assigned to a neuron that corresponds to the input signature and a zero (0) to the second neuron. So existing virus signature is encoded as one and zero (1, 0) and similarly a derived virus signature is encoded as zero and one (0, 1).

Thus an output value of (1, 0) assigned to a particular input means the input is an existing virus signature. Likewise, an output value of (0, 1) assigned to a particular input means the input is a derived virus signature. The output neuron with the highest activation is the class predicted by the neural network.

4.2.4 Neural network training

A neural network is trained and used as the virus detector. A supervised multilayer feed-forward neural network is used for this research. This is because a multilayer feed-forward neural network can learn a mapping of any complexity. The neural network being supervised means each input during the training is matched to its corresponding output which the neural network learns from. The network learning is based on repeated presentation of training samples. Multilayer perceptrons are feed-forward networks with one or more nodes between the input and output layer. The layer between the input and output layer is called hidden layer. Scaled conjugate gradient back-propagation is used because feed-forward neural network is often trained with the error back propagation training algorithm. The back propagation algorithm is an iterative process in which an output error signal is propagated back through the network and is used to modify weight values so that the current least mean square classification error is reduced. The figure below shows a multilayer feed-forward neural network.

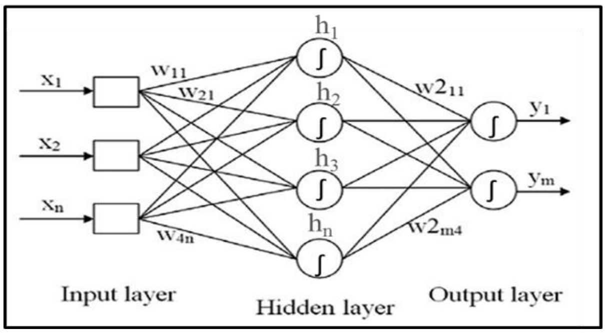


Figure 4.3: A multilayer neural network (Zainal-Mokhtar & Mohamad-Saleh, 2013)

The above figure shows how training is done in a multilayer neural network. The first layer is the input layer which accepts data ($x_1, x_2 ... x_n$) into the neural network and it is linked directly to the hidden layer. The hidden layer lies between the input layer and the output layer. The output layer is the last layer. This layer gives the neural network predicted output to the external party where it is needed for further processes or decision. Each node in the input layer has value for every input ($x_1, x_2 ... x_n$). The neural network initially assigns weights ($w_{11}, w_{21} ... w_{4n}$) to each connection. Each input is multiplied by its corresponding weight and the result of each connection is summed together and passed to the hidden layer nodes ($h_1, h_2, h_3 ... h_n$). A hyperbolic tangent activation (∫) function is used to scale the result and the final value becomes the value for the node the connections entered. This calculation is also performed on the nodes in the hidden layer using the values that entered the hidden layer nodes and the assigned weights ($w2_{11} ... w2_{m4}$) and result is sent to the output layer (y_1, y_m). The neural network checks its predicted output against the desired output; error is calculated by subtracting the desired output from the predicted output. The weights are adjusted based on the error and the iteration continues until the error becomes very small or insignificant.

4.2.5 Neural network testing

The performance of neural networks is determined by their generalization ability. Generalization is the property of trained neural networks to classify an input correctly even if it is not a member of the training set (Urolagin et al., 2011). This is what block five of this model does. New samples that are not part of the training dataset are given to the trained neural network for classification. A correct classification of the hidden samples means such a network has learned. Incorrect classification means otherwise.

Finally, the trained neural network is used as a virus detector especially for derived viruses.

The next section discusses the dataset used for this research.

4.3 Dataset

The training dataset consist of both existing and derived virus signatures. As earlier mentioned, the existing virus signatures were obtained from nlnetlabs website. The website contains virus signatures of various lengths. The obtained virus signatures were carefully examined and signatures with 32 bytes seem to be common more than any other bytes. Therefore, this research considers only virus signatures with 32 bytes. The derived virus signatures used were the output of the mutation engine. The mutation engine randomly combined the existing virus signatures as discussed earlier. The next section discusses the implementation of the model.

4.4 Model Implementation

The part of the model that was implemented first is the mutation engine. The mutation engine was developed with console C# in Microsoft visual studio 2013. The Microsoft visual studio runs on a 2.60GHz Intel core i5 PC with 4GB of memory and 64 bit OS running windows 8.1 professional. The mutation engine picks its data from a database that contains raw signatures of existing viruses and randomly combines the signatures to produce derived virus signatures. Furthermore, it converts both the existing and derived virus signatures to decimal and save them to another database accordingly.

The neural network training is the model part that was implemented secondly. The training is done with Matlab 2014 neural network pattern recognition tool (nprtool). Nprtool is a multilayer feed-forward neural network tool used to create and train a neural network for pattern recognition. The nprtool runs on a computer with same configuration as specified above. Training data was divided into three, 70% was for training, 15% for validation and 15% for testing the neural network. The neural network has 32 inputs, that is the bytes of virus signatures in use as each byte represents a feature and two outputs which are the classes. Parameter such as the number of hidden layers were varied a couple of times in order to get a neural network that best solves the problem of classifying any given virus signature into derived or existing signature. Training automatically stops when generalization stops improving, as indicated by an increase in the mean square error of the validation samples. This is done to avoid overfitting. Overfitting is a condition that arises when a neural network has mastered its training data. This is bad because it makes the neural network to have poor performance on new data. The program code for the neural network training is then written as a function and deployed using Matlab deploy tool. This created a class library (.dll) for the trained neural network. The created trained neural network library is used for the final part of the model.

The last part of the model is implemented with visual C# using Microsoft visual studio 2013. The Microsoft visual studio runs on a PC with same configuration as the mutation engine. The trained neural network library is used as the classifier in the C# program code. This last part reads a database that contains virus signatures; send the signatures to the trained neural network and the neural network based on the training predicts whether the signatures are existing or derived virus signatures.

4.5 Conclusion

This chapter gives a full detail of the model as proposed base on requirements. The model has learnt successfully from both existing and derived virus signatures presented to it during training. Hence, it can now classify any given virus signature into existing or derived.

The next chapter discusses the result of the experiments conducted with the model.

CHAPTER FIVE

RESULT AND DISCUSSION

5.1 Introduction

The previous chapter discussed the proposed model for computer virus detection. The model uses a mutation engine to create derivate of existing virus signatures and uses a trained neural network as detector within a computer system.

This chapter presents and discusses the result of the experiments conducted with the model. The training data are first presented to the model for classification. Thereafter, the mutation engine created new derived virus signatures, these new virus signatures are presented to the model to classify and the results are shown and discussed.

5.2 Mutation engine output

The mutation engine was given a number of existing virus signatures. These signatures were grouped by the mutation engine based on similarities. Thereafter, three variants each are created for all the groups as discussed in the previous chapter. The table below shows what the mutation engine output looks like.

Table 5.1: Mutation engine output

Name	Signature	Name	Signature
8-Tunes1	0fe0cd213d314c753d2e813e2b004d5a	Italian –D1	72225002000e2f20e00f70200ca0005c
8-Tunes2	33f6b9da03f3a550bb230353cb8ed0bc	Italian –D2	0d7260d5c08a2c26c0200bd27a3b67c6
8-Tunes -D1	53e53bda339503e55d21c3e069e5530d	Italian –D3	c0007000000580b20ebeee042fce7007
8-Tunes -D2	5e8f65e135def133313c93eee3083383	403 (B)	342e892603012e8c1605012ea307018d
8-Tunes -D3	e0030db38d503311d3a155bb33e0d505	403 (A)	2135cd21891e59018c065b018cc88ed8
Italian-#1	ec020e1ff3a4b82125061fbab300cd21	403 -D1	05025dde26885117001801d1e5771152
Italian-Gene	b106d3e02dc0078ec0be007c8bfeb900	403 –D2	00c130e05238c568cc1751c03101105c

The mutation engine created a number of families (groups) from the existing virus signatures given. Thereafter, derivation is done on each family up until the third generation. The first, second and third generations have suffix “D1”,”D2” and “D3” respectively as shown in table 5.1 above.

5.3 Model result

This section shows the result of the model using the trained neural network to classify any given virus signature into either existing or derived virus signature. As discussed in the previous chapter, existing virus signature is encoded with “1, 0” and derived virus signature is encoded with “0, 1”. Therefore, during testing, the output neuron with the highest activation (output) is the class predicted by the neural network (Model).

5.3.1 The result of the model on training data

The derived virus signatures used for training are presented to the model. The table below shows to which group the model classified each virus signature.

Table 5.2: Training data results

Derived virus signatures used for training								
First generation			Second generation			Third generation		
0,0112	0,9902	Derived	0,1679	0,8182	Derived	0,2529	0,8603	Derived
0,0971	0,878	Derived	0,2572	0,7532	Derived	0,1764	0,8246	Derived
0,0238	0,9704	Derived	0,1057	0,8758	Derived	0,3473	0,8538	Derived
0,1056	0,9331	Derived	0,5885	0,4675	Existing	0,1826	0,8004	Derived
0,0334	0,9769	Derived	0,0575	0,9153	Derived	0,0517	0,9699	Derived
0,0559	0,9708	Derived	0,2106	0,9103	Derived	0,9311	0,1538	Existing
Total Signatures (%)		100	Total Signatures (%)		100	Total Signatures (%)		100
Derived found (%)		93,5	Derived found (%)		84,4	Derived found (%)		77,9
Existing found (%)		6,5	Existing found (%)		15,6	Existing found (%)		22,1

The table above shows some of the result of the experiment conducted with the model. The first result (0,0112 and 0,9902) for the first generation has its second value greater than the first. This means the model categorized the input signature as a derived virus signature since we decoded derived virus signatures as “0, 1” during the training. Furthermore, the last result for the third generation (0,9311 and 0,1538) has its first value greater than the second. This means the model categorized the input signature as an existing virus signature since we decoded existing virus signatures as “1, 0” during the training. All experiment results are interpreted in the same way.

As shown in the table, the model was able to classify 93,5% virus signatures successfully as derived virus signatures and 6,5% were incorrectly classified as existing signatures in the first generation. Furthermore, in second generation, 84,4% virus signatures were successfully classified as derived virus signatures while 15,6% virus signatures were incorrectly classified as existing virus signatures. Lastly, for the third generation, 77,9% virus signatures were successfully classified as derived virus signatures and 22,1% signatures were incorrectly classified as existing virus signatures.

5.3.2 The result of the model on new data

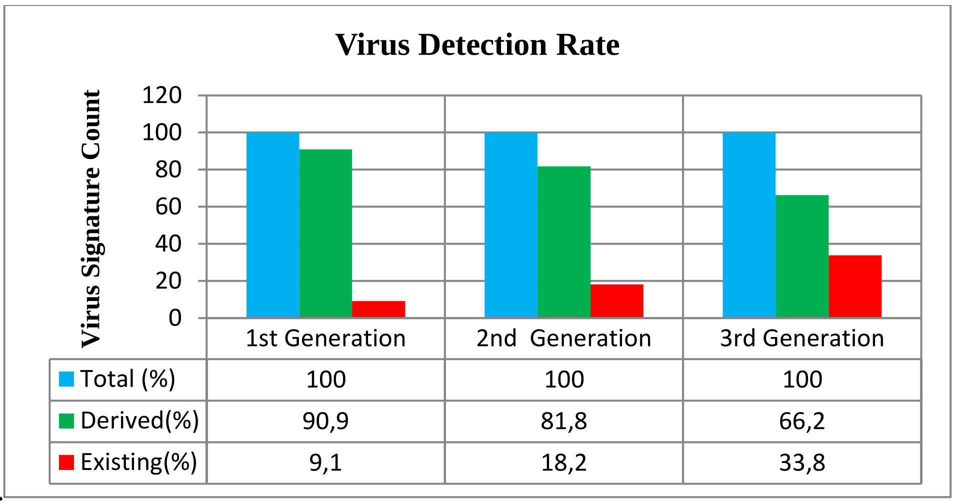
The mutation engine of the model created new variants of existing virus signatures for ten experiments. That is, it generated first, second and third generations for ten experiments. The new variants are presented to the model and the result is shown in the tables below. It is observed that the model performance on the ten experiments in terms of the derived and existing virus signatures classified is similar with a difference of two to five signatures classified or not classified for each generation in all the ten experiments. Therefore, we will only show results for experiment 1, 2, 3, 8, 9 and 10.

5.3.2.1 Experiment one

Table 5.3: Experiment one result

First generation			Second generation			Third generation		
0,0844	0,9638	Derived	0,3122	0,8133	Derived	0,016	0,98	Derived
0,2178	0,8955	Derived	0,227	0,7478	Derived	0,4707	0,6803	Derived
0,0394	0,9632	Derived	0,1119	0,9274	Derived	0,2625	0,8817	Derived
0,2533	0,8172	Derived	0,4435	0,679	Derived	0,7716	0,3993	Existing
0,1989	0,7582	Derived	0,2726	0,8031	Derived	0,0601	0,9293	Derived
0,0317	0,9595	Derived	0,1582	0,8777	Derived	0,1452	0,9406	Derived
0,5437	0,5096	Existing	0,7354	0,3186	Existing	0,413	0,7465	Derived
0,0366	0,9446	Derived	0,8389	0,2525	Existing	0,974	0,0679	Existing
0,3315	0,6154	Derived	0,1179	0,9066	Derived	0,1294	0,8627	Derived
0,2279	0,866	Derived	0,184	0,9006	Derived	0,9366	0,1676	Existing
Total virus signature (%)		100	Total virus signature (%)		100	Total virus signature (%)		100
Derived found (%)		90,9	Derived found (%)		81,8	Derived found (%)		66,2
Existing found (%)		9,1	Existing found (%)		18,2	Existing found (%)		33,8

In the first generation, the model was able to classify 90,9% signatures successfully as derived signatures out of the virus signatures given and 9,1% signatures were incorrectly classified as existing virus signatures. For the second generation, the model classified 81,8% signatures successfully as derived virus signatures and the remaining 18,2% signatures were incorrectly classified as existing virus signatures. Lastly, for the third generation, the model was able to classify 66,2% signatures successfully as derived signatures and the remaining 33,8% signatures were incorrectly classified as existing virus signatures. The result is shown graphically in the figure below



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived(%)	90,9	81,8	66,2
🟥 Existing(%)	9,1	18,2	33,8

Figure 5.1: Graphical representation of experiment one result

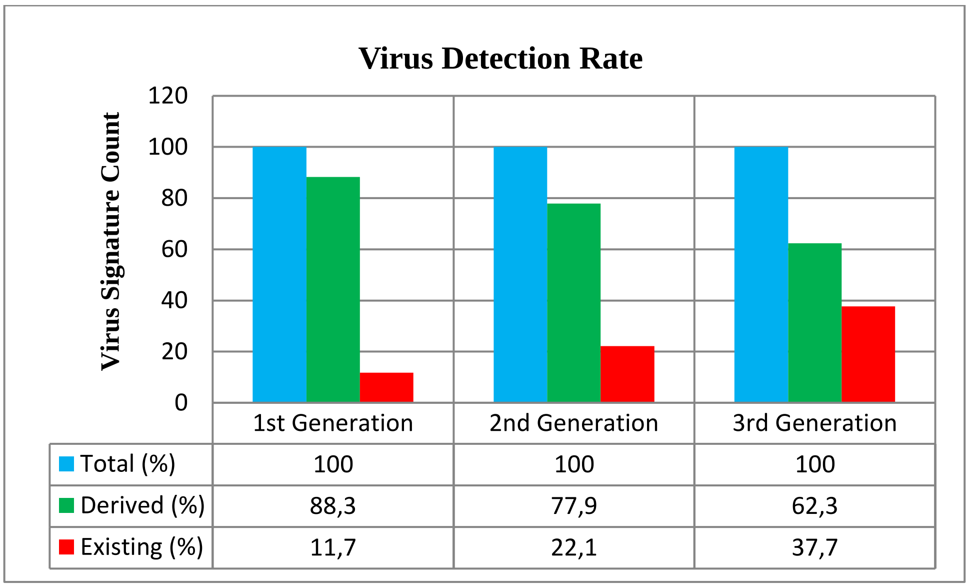
Figure 5.1 above shows the output of the model graphically for the first experiment. The percentage of derived and existing virus signatures found for each generation is shown.

5.3.2.2 Experiment two

Table 5.4: Experiment two result

First generation			Second generation			Third generation		
0,4164	0,636	Derived	0,1497	0,8725	Derived	0,841	0,347	Existing
0,2998	0,6644	Derived	0,04	0,9651	Derived	0,6937	0,4485	Existing
0,4208	0,6119	Derived	0,0479	0,9156	Derived	0,6841	0,45	Existing
0,4861	0,5689	Derived	0,3519	0,6855	Derived	0,2757	0,7225	Derived
0,1619	0,8758	Derived	0,5317	0,5808	Derived	0,0976	0,9433	Derived
0,145	0,9211	Derived	0,5325	0,7153	Derived	0,1128	0,9216	Derived
0,0033	0,9954	Derived	0,5371	0,659	Derived	0,7874	0,2685	Existing
0,0433	0,9291	Derived	0,0115	0,9857	Derived	0,0097	0,9889	Derived
0,6313	0,359	Existing	0,1557	0,8416	Derived	0,8012	0,2969	Existing
0,4478	0,5141	Derived	0,0667	0,9305	Derived	0,5261	0,5437	Derived
Total Signatures (%)		100	Total Signatures (%)		100	Total Signatures(%)		100
Derived found (%)		88,3	Derived found (%)		77,9	Derived found (%)		62,3
Existing found (%)		11,7	Existing found (%)		22,1	Existing found (%)		37,7

The table above shows experiment two result. In the first generation, the model was able to classify 88,3% signatures successfully as derived virus signatures and 11,7% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation, 77,9% signatures were successfully classified as derived virus signatures and the remaining 22,1% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation, the model successfully classified 62,3% signatures as derived virus signatures and 37,7% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived (%)	88,3	77,9	62,3
🟥 Existing (%)	11,7	22,1	37,7

Figure 5.2: Graphical representation of experiment two result

Figure 5.2 above shows the output of the model graphically for experiment two. The percentage of derived and existing virus signatures found for each generation is shown.

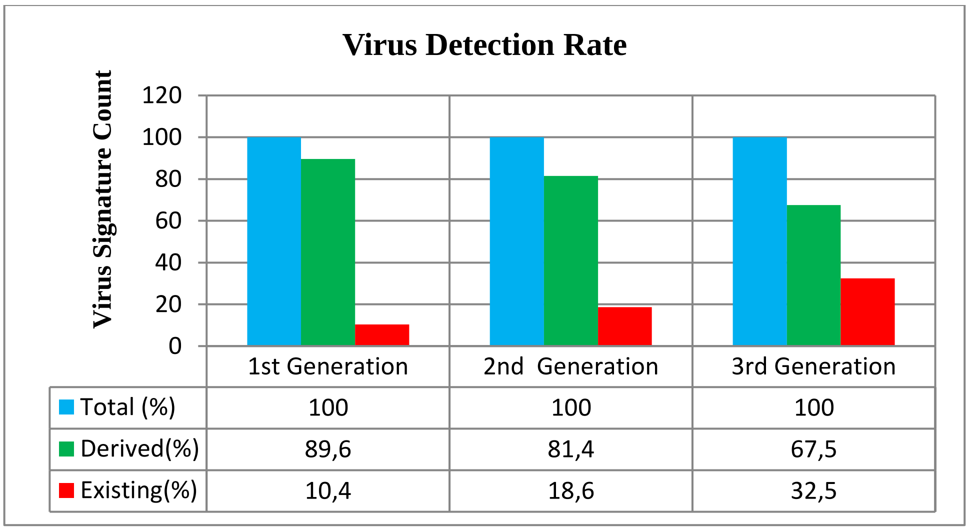
5.3.2.3 Experiment three

Table 5.5: Experiment three result

First generation			Second generation			Third generation		
0,0667	0,9164	Derived	0,9795	0,0607	Existing	0,4704	0,6508	Derived
0,0221	0,9663	Derived	0,1279	0,8741	Derived	0,6663	0,4095	Existing
0,7873	0,215	Existing	0,833	0,1313	Existing	0,3939	0,6194	Derived
0,4884	0,646	Derived	0,7332	0,4721	Existing	0,624	0,4892	Existing
0,1943	0,9068	Derived	0,0066	0,9886	Derived	0,4569	0,7153	Derived
0,6466	0,3631	Existing	0,4023	0,7275	Derived	0,1757	0,8966	Derived

0,1512	0,8905	Derived	0,1851	0,781	Derived	0,8623	0,2772	Existing
0,8919	0,2201	Existing	0,0246	0,9584	Derived	0,0296	0,9763	Derived
0,0715	0,8905	Derived	0,909	0,149	Existing	0,2413	0,8007	Derived
0,3493	0,7011	Derived	0,0474	0,933	Derived	0,0147	0,9903	Derived
Total signatures (%)		100	Total signatures (%)		100	Total signatures (%)		100
Derived found (%)		89,6	Derived found (%)		81,4	Derived found (%)		67,5
Existing found (%)		10,4	Existing found (%)		18,6	Existing found (%)		32,5

The table above presents experiment three result. In the first generation, the model was able to successfully classify 89,6% signatures as derived virus signatures and the remaining 10,4% signatures were incorrectly classified as existing virus signatures. Furthermore, for second generation, the model was able to classify 81,4% signatures as derived virus signatures while the remaining 18,6% signatures were incorrectly classified as existing virus signatures. Lastly, for the third generation, the model was able to classify 67,5% virus signatures as derived and the remaining 32,5% signatures were classified as existing. The figure below shows the graphical representation of the results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived(%)	89,6	81,4	67,5
🟥 Existing(%)	10,4	18,6	32,5

Figure 5.3: Graphical representation of experiment three result

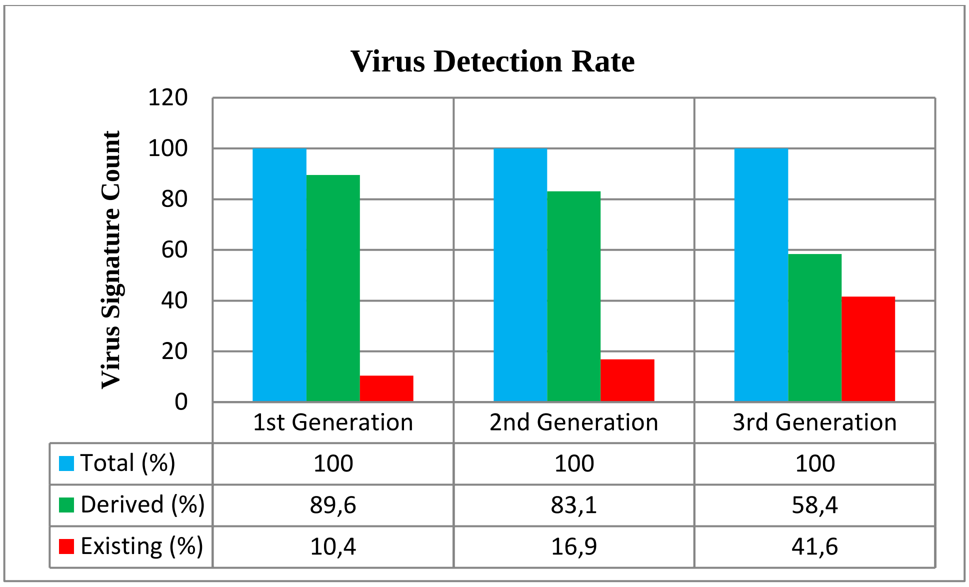
Figure 5.3 above shows the output of the model graphically for experiment three. The percentage of derived and existing virus signatures found for each generation is shown.

5.3.2.4 Experiment Eight

Table 5.6: Experiment eight result

First generation			Second generation			Third generation		
0,0991	0,8904	Derived	0,0718	0,9299	Derived	0,2443	0,7419	Derived
0,0445	0,9678	Derived	0,2033	0,8199	Derived	0,1266	0,924	Derived
0,1829	0,8369	Derived	0,1007	0,9336	Derived	0,0319	0,9588	Derived
0,2079	0,7856	Derived	0,0396	0,9652	Derived	0,6252	0,4111	Existing
0,17	0,8998	Derived	0,0097	0,9836	Derived	0,321	0,712	Derived
0,2269	0,8581	Derived	0,148	0,8647	Derived	0,1415	0,8433	Derived
0,5946	0,402	Existing	0,0389	0,9541	Derived	0,5859	0,3602	Existing
0,0223	0,9692	Derived	0,2605	0,7276	Derived	0,0895	0,9363	Derived
0,1451	0,9196	Derived	0,2634	0,7358	Derived	0,1212	0,9216	Derived
0,2192	0,7603	Derived	0,1527	0,8983	Derived	0,544	0,7212	Derived
Total signatures (%)		100	Total signatures (%)		100	Total signatures (%)		100
Derived found (%)		89,6	Derived found (%)		83,1	Derived found (%)		58,4
Existing found (%)		10,4	Existing found (%)		16,9	Existing found (%)		41,6

The table above shows experiment eight result. In the first generation, the model was able to classify 89,6% signatures successfully as derived virus signatures and 10,4% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation, 83,1% signatures were successfully classified as derived virus signatures and the remaining 16,9% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation, the model successfully classified 58,4% signatures as derived virus signatures and 41,6% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived (%)	89,6	83,1	58,4
🟥 Existing (%)	10,4	16,9	41,6

Figure 5.4: Graphical representation of experiment eight result

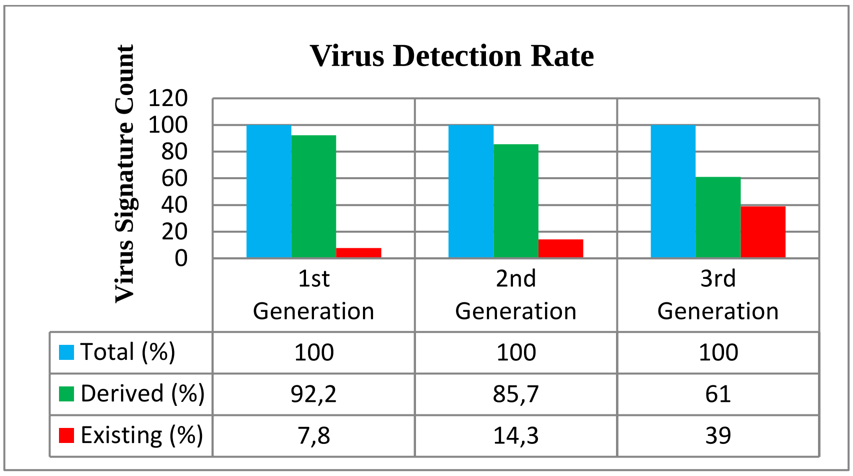
The figure above shows the output of the model graphically for experiment eight. The percentage of derived and existing virus signatures found for each generation is shown.

5.3.2.5 Experiment Nine

Table 5.7 : Experiment nine result

First generation			Second generation			Third generation		
0,2722	0,8287	Derived	0,1444	0,906	Derived	0,0634	0,9227	Derived
0,2824	0,7584	Derived	0,1611	0,8864	Derived	0,2157	0,893	Derived
0,9387	0,2627	Existing	0,4565	0,6876	Derived	0,2024	0,7508	Derived
0,3796	0,5423	Derived	0,1706	0,8696	Derived	0,0022	0,9961	Derived
0,1678	0,8315	Derived	0,5549	0,6878	Derived	0,1299	0,8336	Derived
0,0456	0,9691	Derived	0,5947	0,4937	Existing	0,0286	0,9565	Derived
0,4005	0,7387	Derived	0,2557	0,8671	Derived	0,0737	0,8717	Derived
0,363	0,7467	Derived	0,2226	0,8558	Derived	0,0444	0,9569	Derived
0,301	0,8319	Derived	0,0428	0,9588	Derived	0,352	0,7141	Derived
0,0607	0,9461	Derived	0,082	0,8875	Derived	0,2113	0,8303	Derived
Total signatures (%)		100	Total signatures (%)		100	Total signatures (%)		100
Derived found (%)		92,2	Derived found (%)		85,7	Derived found (%)		61,0
Existing found (%)		7,8	Existing found (%)		14,3	Existing found (%)		39,0

The table above presents experiment nine result. In the first generation, the model was able to classify 92,2% signatures successfully as derived virus signatures and 7,8% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation, 85,7% signatures were successfully classified as derived virus signatures and the remaining 14,3% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation, the model successfully classified 61% signatures as derived virus signatures and 39% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived (%)	92,2	85,7	61
🟥 Existing (%)	7,8	14,3	39

Figure 5.5: Graphical representation of experiment nine result

The figure above shows the output of the model graphically for experiment nine. The percentage of derived and existing virus signatures found for each generation is shown.

5.3.2.6 Experiment Ten

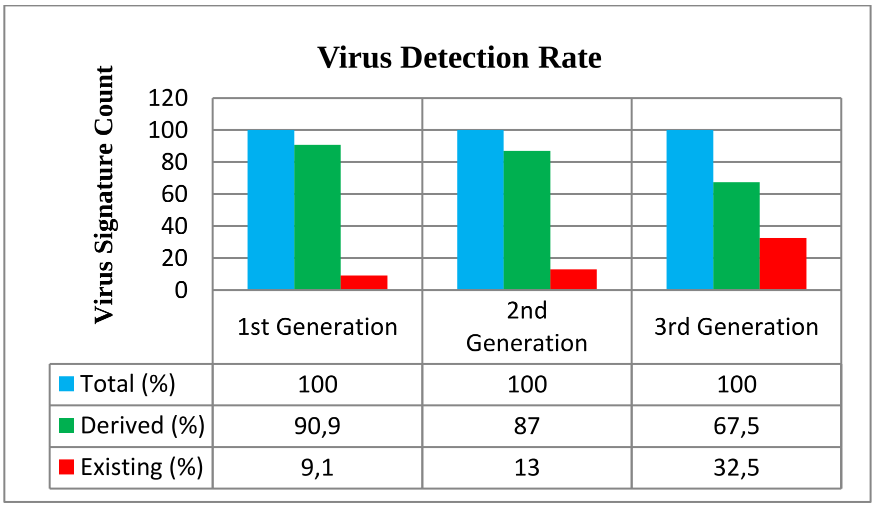
Table 5.8 : Experiment ten results

First generation			Second generation			Third generation		
0,3301	0,7172	Derived	0,0989	0,9382	Derived	0,0328	0,9539	Derived
0,0561	0,9468	Derived	0,5398	0,5812	Derived	0,0648	0,9434	Derived
0,2357	0,8563	Derived	0,3037	0,8131	Derived	0,2076	0,8535	Derived

0,2907	0,87	Derived	0,0179	0,9721	Derived	0,2528	0,7142	Derived
0,0119	0,9841	Derived	0,76	0,3854	Existing	0,3144	0,7639	Derived
0,3118	0,8458	Derived	0,1355	0,8211	Derived	0,7956	0,2975	Existing
0,2621	0,8241	Derived	0,22	0,8828	Derived	0,0808	0,9362	Derived
0,0164	0,9711	Derived	0,9518	0,064	Existing	0,8016	0,2788	Existing
0,2098	0,8558	Derived	0,7917	0,3095	Existing	0,03	0,9686	Derived
0,1972	0,8228	Derived	0,8034	0,2937	Existing	0,216	0,884	Derived

Total signatures (%)	100	Total signatures (%)	100	Total signatures (%)	100
Derived found (%)	90,9	Derived found (%)	87,0	Derived found (%)	67,5
Existing found (%)	9,1	Existing found (%)	13,0	Existing found (%)	32,5

The table above shows some experiment ten result. In the first generation, the model was able to classify 90,9% signatures successfully as derived virus signatures and 9,1% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation, 87% signatures were successfully classified as derived virus signatures and the remaining 13% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation, the model successfully classified 67,5% signatures as derived virus signatures and 32,5% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived (%)	90,9	87	67,5
🟥 Existing (%)	9,1	13	32,5

Figure 5.6: Graphical representation of experiment ten result

The figure above shows the output of the model graphically for experiment ten. The percentage of derived and existing virus signatures found for each generation is shown.

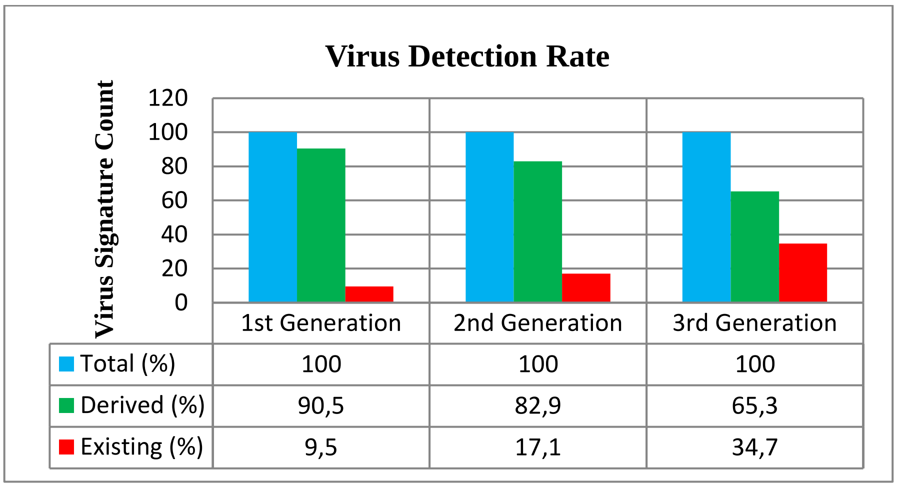
5.3.2.7 Overall result for all experiments

Table 5.9: Result for all experiments

Experiment	**1st Generation (%)**		**2nd Generation (%)**		**3rd Generation (%)**	
	Derived	**Existing**	**Derived**	**Existing**	**Derived**	**Existing**
1	90,9	9,1	81,8	18,2	66,2	33,8
2	88,3	11,7	77,9	22,1	62,3	37,7
3	89,6	10,4	81,4	18,6	67,5	32,5
4	90,9	9,1	80,5	19,5	70,1	29,9
5	89,6	10,4	83,1	16,9	64,9	35,1
6	92,2	7,8	83,1	16,9	62,3	37,7
7	90,9	9,1	84,4	15,6	72,7	27,3
8	89,6	10,4	83,1	16,9	58,4	41,6
9	92,2	7,8	85,7	14,3	61,0	39,0
10	90,9	9,1	87,0	13,0	67,5	32,5

The ten experiments conducted consist of a number of derived virus signatures. In the first generation of all the experiments, the model was able to classify an average of 90,5% signatures successfully as derived virus signatures and 9,5% signatures were incorrectly classified as existing virus signatures. Furthermore, in the second generation of all the experiments, an average of 82,9% signatures were successfully classified as derived virus signatures and the remaining 17,1% signatures were incorrectly classified as existing virus signatures. Lastly, in the third generation of all the experiments, the model successfully classified an average of

65,3% signatures as derived virus signatures and 34,7% signatures were incorrectly classified as existing virus signatures. The figure below shows the graphical representation of the overall results.



	1st Generation	2nd Generation	3rd Generation
🟦 Total (%)	100	100	100
🟩 Derived (%)	90,5	82,9	65,3
🟥 Existing (%)	9,5	17,1	34,7

Figure 5.7: Graphical representation of the result of all experiments

The figure above presents the overall results for all the ten experiments conducted with the model and also shows the percentage of derived and existing virus found in each generation.

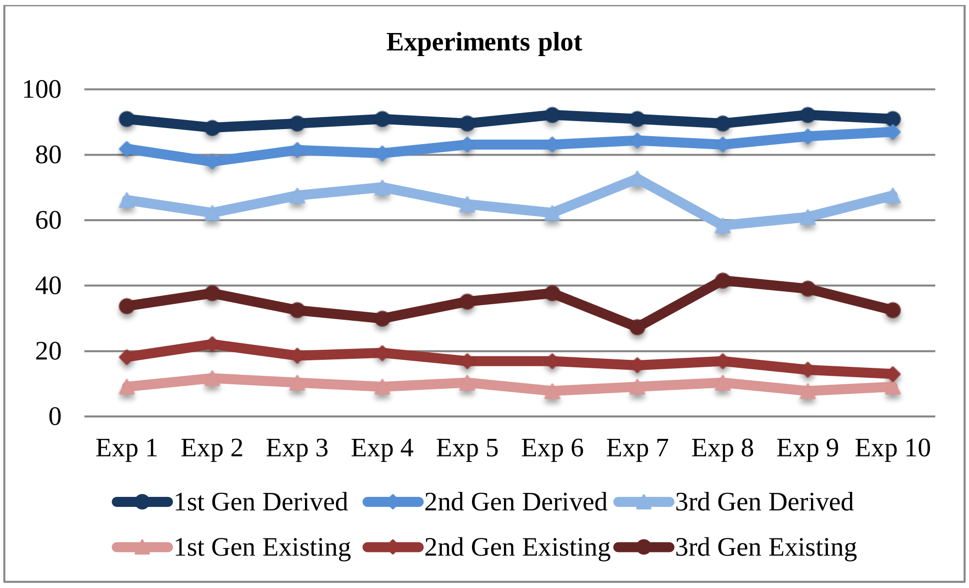


Figure 5.8 : Plot showing the results for all experiments with generations

The plot above shows the percentage of derived and existing virus signatures as classified by the model. As shown in the graph, the count of virus signatures that are classified as derived drops as the generation increases. On the contrary, the count of virus signatures classified as existing keeps increasing as the generation increases even though all signatures used are derived. This means that at a point after many generations, the derived virus signatures will be seen as a new or different virus signature. This explains the reason why signature scanning could not detect derived viruses.

5.4 Conclusion

This chapter presents the results of the experiments conducted with the model proposed in the previous chapter. It is observed in all the experiments conducted that the first generation has highest number of derived virus detection rate than the other two generations. The second generation also showed a higher detection rate than the third generation which has the least detection rate. Based on this, a conclusion can be drawn that the correlation between the actual virus signature and its derivatives decreases further down along the generations. This means that after many generations of a virus changing forms, its variants will no longer look like the original. The variants will look like a completely new virus even though the variants and the original virus will always have same behaviour and operation with similar effects.

The next chapter concludes this dissertation and highlights future work.

CHAPTER SIX

SUMMARY, CONCLUSION AND RECOMMENDATIONS

6.1 Introduction

This is the concluding chapter of this dissertation. The research work is summarized along with an evaluation of the achievement of research objectives. The direction for future work is highlighted.

6.2 Summary

The problem studied has a serious impact in information security. The protection of computing devices against computer viruses has been a major problem in the field of information security since its first appearance in the 1970s and the damage caused by viruses is increasing year after year. Early viruses had no serious impact asides displaying some annoying pop up messages. However, today’s viruses have a reputation and financial impact. Over the years, computer viruses have become and operate in stealth mode to avoid detection. This is usually achieved with many modern obfuscation techniques. Although, it is argued that new viruses are created daily. However, most of the supposedly ‘new’ viruses are not necessarily created from the scratch, rather the supposedly ‘new’ viruses are products of the existing viruses using different obfuscation techniques to change their look and form thereby looking like a new virus at every infection. The self-modification ability that the modern viruses’ exhibit has put strain on virus detection by signature scanning which is regarded as the oldest and the most commonly used detection technique. The problem of computing device against virus is not just about protection against known virus but also against unknown viruses. The unknown virus could be new or derived virus (Virus that comes into existence as a result of modification of existing viruses).

This research work therefore explores the use of neural networks together with a mutation engine to try and increase the detection rate of derived viruses. The mutation engine creates derivatives of existing virus signatures. The neural network is trained with both derived and existing virus signatures and is used as a detector to detect existing viruses and their variants in a computer system.

6.3 Conclusion

This research presents a novel model for detecting derived virus. The ability of the proposed model to accurately classify derived virus signatures means that it can be used as virus detector in a computer system. The model has successfully improve the rate of detecting derived viruses from 70% as per Golovko & Bezobrazov (2015). Furthermore, the model is better than signature scanning. This is mainly because it does not depend solely on a set of pre-defined virus signature; it also postulate what future signatures might look like and makes an attempt to detect them. Moreover, the model is able to classify viruses used for the training and also their new variants that were not used during the training. Currently, the proposed model has an average success rate of:

- 80.2% on detecting existing signatures,
- 85.2% on signatures used for training and
- 80% on the average for derived virus signatures.

In terms of the derived virus signatures; this includes success rate of 91% on first generation, 83% on second generation and 65% on third generation. Of note in these results is that the accuracy of detecting derived viruses decreases linearly as the number of generations increase. For example, the success rate would have been even lower on the fourth generation than it is on the third. This is to be expected because as more generations are added to the training set;

the difference between the original and the derivative virus signature gets even bigger. Furthermore, this explains the difficulty in current anti-virus systems to detect derived viruses. Although, the result seems to have a lesser accuracy as compared to some existing solutions; it is important to note that the proposed model was able to detect derived viruses which cannot be detected by existing systems.

6.4 Recommendations for Future Work

Our future work will try and improve the accuracy of the model. Furthermore, future work will experiment with variable virus signature sizes and those that contain special characters. Finally, the model is to be trained with encrypted virus signatures for robustness.

It is recommended that future research efforts continue along the direction of this study to explore this area in depth. It will be a great benefit to the field of information security to have a virus detection mechanism that does not detect only known viruses but also the new and derived viruses.

REFERENCES

Andree, L., Nhien-An, L., 2016. Control Flow Change in Assembly as a Classifier in Malware Analysis. *4th IEEE International Symposium on Digital Forensics and Security*, pp. 2-7.

Banerjee, R., 2015. Artificial Intelligence in Power Station. *International Journal Of Innovative Research In Electrical, Electronics, Instrumentation And Control Engineering*, 3(7), pp.86.

Borana, J., 2016. Applications of Artificial Intelligence and Associated Technologies. *Proceeding of International Conference on Emerging Technologies in Engineering, Biomedical, Management and Science ,2016* , pp.64–67.

Chen, Y., Narayanan, A., Pang, S., Tao, B., 2012. Multiple sequence alignment and artificial neural networks for malicious software detection. *8th International Conference on Natural Computation*, (Icnc), pp.261–265.

Clements, A., 2014. The No Operation Instruction. Available at: http://alanclements.org/nops.html [Accessed August 11, 2016].

Daoud, E. A., 2009. Metamorphic Viruses Detection Using Artificial Immune System. *International Conference on Communication Software and Networks Metamorphic,* pp.168–172.

Dhruw, M.K., Dewangan, Y., Patel, P., 2016. An Introduction of Computer Virus , History and its Evolution. *International Journal of Research*, 3(4), pp.275–282.

Edward, W., Slulason, F., 1990. Computer Virus Prevention, Recognition and Removal. , Virus Bulletin Ltd(November 1990).

El-Bakry, H.M., 2010. Fast virus detection by using high speed time delay neural networks.

Journal in Computer Virology, 6(2), pp.115–122.

Feng, M., Gupta, R., 2009. Detecting virus mutations via dynamic matching. *IEEE International Conference on Software Maintenance,* Canada. pp. 105–114.

Filiol, E., 2005. *Computer viruses : From theory to Applications* First Edition, Springer Berlin Heidelberg New York.

Gang, G., Zhongquan, C., 2014. A Kind of Malicious Code Detection Scheme Based on Fuzzy Reasoning. *7th International Conference on Intelligent Computation Technology and Automation*, pp.19–22.

Golovko, V., Bezobrazov, S., 2015. Neural Network Artificial Immune System for Malicious Code Detection. ResearchGate, pp. 1–7. Available at: https://www.researchgate.net/publication/268377401

Hamza, A., Hussain, D.J., 2014. Computer Virus Detection Based on Artificial Immunity Concept. *International Journal of Emerging Trends and Technology in Computer Science (IJETTCS)*, 3(2), pp.68–74.

Han, L., Fu, C., Zou, D., Lee, C., Jia, W., 2012. Task-based behavior detection of illegal codes. *Mathematical and Computer Modelling*, 55(1–2), pp.80–86.

Heaton, J., 2011. *Programming Neural Networks with Encog3 in Java* 1st Edition, Heaton Research, Inc.

Ivanov, A., Makrushin, D., van der Wiel, J., Garnaeva, M. Namestnikov, Y., 2015. Kaspersky Security Bulletin 2015. Overall statistics for 2015. Available at: https://securelist.com/analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/ [Accessed October 25, 2016].

Joshi, M.J., Patil, B.V., 2012. Computer Virus: Their Problems & Major attacks in Real Life. *Journal of Advanced Computer Science and Technology*, 1(4), pp. 316-324.

Kakad, A.R., Kamble, S.G., Bhuvad, S.S., Malavade, V.N., 2014. Study and Comparison of Virus Detection Techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(3), pp.251–253.

Kamarudin, I.E., Sharif, S.A.M and Herawan, T., 2013. On Analysis and Effectiveness of Signature Based in Detecting Metamorphic Virus. , 7(4), pp.375–386.

Kaspersky Lab, 2016. Kaspersky Security Bulletin: Overall statistics for 2016. Available at: https://securelist.com/statistics/ [Accessed January 16, 2017].

Khari, M., Bajaj, C., 2014. Detecting Computer Viruses. *International Journal of Advanced Research in Computer Engineering & Technology* 3(7), pp. 2359.

Khang, M.T., Nguyen, V.T., Le, T.D., 2016. A Combination of Artificial Neural Network and Artificial Immune System for Virus Detection. *Rev Jounal on Electronics and Communication* 5(3), pp.52–57.

Khorsand, Z., Hamzeh, A., 2013. A Novel Compression-Based Approach for Malware Detection Using PE Header. *5th Conference on Information and Knowledge Technology (IKT)*. Shiraz: IEEE, pp. 127–133.

Kahanwal, B., 2013. Abstraction Level Taxonomy of Programming Language Frameworks. *International Journal of Programming Languages and Applications,* 3(4) pp. 5.

Kumar, A., 2016. What is a Polymorphic Virus and how do you deal with it. Available at: http://www.thewindowsclub.com/polymorphic-virus [Accessed October 26, 2016].

Kumar, M. Hundreds Of Operations Canceled After Malware Hacks Hospitals Systems. *The*

Hacker News. N.p., 2016. Available at : http://thehackernews.com/2016/11/hospital-cyber-attack-virus.html [Accessed January 13, 2017].

Kuriakose, J., Vinod, P., 2014a. Discriminant Features for Metamorphic Malware Detection. Contemporary Computing (IC3), *Seventh International Conference*, IEEE, pp. 1-3.

Kuriakose, J., Vinod, P., 2014b. Metamorphic Virus Detection using Feature Selection Techniques. *5th International Conference on Computer and Communication Technology*. IEEE, pp. 141–146.

Le, Q., Mikolov, T., Com, T.G., 2014. Distributed Representations of Sentences and Documents. *31st International Conference on Machine Learning, Beijing, China, 2014* , 32, p.1.

Ling, Y.T., Sani, F.M., 2017. Review on Metamorphic Malware Detection in Hidden Markov Models. *International Journal of Advanced Research in Computer Science and Software Engineering*, 7(2), pp. 63.

Liu, G., Chen, W., Fen, H., 2010. A Neural Network Ensemble based Method for Detecting Computer Virus. *International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE),* 1, pp.391–393.

Mishra, U., 2010. *Methods of Virus Detection and Their Limitations*. SSRN Electronic Journal. Available at: http://ssrn.com/abstract=1916708 [Accessed January 1, 2017].

Muhtadi, M., 2014. Computer Viruses : Now and Then. *Al-Nasser University journal*, pp.10.

Nguyen, B.T., Ngo, B.T., Quan, T.T., 2012. A Memory-Based Abstraction Approach to Handle Obfuscation in Polymorphic Virus. *19th Asia-Pacific Software Engineering Conference,* pp.158–161.

Otake, T., 2015. Japan Pension Service hack used classic attack method. Available at: http://www.japantimes.co.jp/news/2015/06/02/national/social-issues/japan-pension-service-hack-used-classic-attack-method/#article_history, [Accessed January 10, 2017].

Pannu, A., 2015. Artificial Intelligence and its Application in Different Areas. *International Journal of Engineering and Innovative Technology*, 4(10), pp.79.

Qin, R., Li, T., Zhang, Y., 2009. An Immune Inspired Model for Obfuscated Virus Detection. *International Conference on Industrial Mechatronics and Automation*. China: IEEE, pp. 228–231.

Rad, B.B., Masrom, M., Ibrahim, S., 2012. Camouflage in Malware : from Encryption to Metamorphism. *International Journal of Computer Science and Network Security*. 12(8), pp.74–83.

Rad, B.B., Masrom, M., Ibrahim, S., 2011. Evolution of Computer Virus Concealment and Anti-Virus Techniques : *A Short Survey, International Journal of Computer Science Issues*, 8(1), pp.113–121.

Rad, B.B., Masrom, M., 2010. Metamorphic Virus Variants Classification Using Opcode Frequency Histogram. *Latest trends on computers* (1), pp.148-152.

Rajesh, B., Reddy, Y.R.J., Reddy, B.D.K., 2015. A Survey Paper on Malicious Computer Worms. *International Journal of Advanced Research in Computer Science & Technology,* 3(2), pp. 163, 165.

Singhal, D., 2014. Computer Viruses in India. *International Journal of Computer Applications, National Conference on Innovations and Recent Trends in Engineering and Technology,* pp.26.

Singla, S., Bansal, D., Gandotra, E., Sofat, S., 2015. Detecting and Classifying Morphed Malwares. *International Journal of Computer Applications*, 122(10), pp. 29.

Silverman, J., 2001. Understanding Polymorphic Viruses. *2001*. Available at: http://www.commercialventvac.com/UnderstandingPolymorphicViruses.html [Accessed January 12, 2017].

Silva, C.P., Dias, D.M., Bentes, C., Pacheco, M.A., CuperTino, L.F., 2015. Evolving GPU Machine Code. *Journal of Machine Learning Research*, (16), pp. 680.

Sonali, B.M., Wankar, P., 2014. Research Paper on Basic of Artificial Neural Network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1), pp.96.

Urolagin, S., Prema, K.V., Reddy, N.V.S, 2011. Generalization capability of artificial neural network incorporated with pruning method. *Proceedings of the 2011 international conference on Advanced Computing, Networking and Security*, pp. 171.

Van der made, P., 2003. Computer immune system and method for detecting unwanted code in a P-code or partially compiled native-code program executing within a virtual machine. Available at : https://www.google.com/patents/US20030212902.

Venkatachalam, S., 2010. Detecting Undetectable Computer. *Master's Theses and Graduate Research, San Jose State University.*

Vinod, P., Jain, H., Golecha, Y.K., Gaur, M.S. Laxmi, V., 2010. MEDUSA : MEtamorphic malware dynamic analysis using signature from API . MEDUSA . pp. 263-269. Available at: : http://www.researchgate.net/publication/221506943.

Vinod, P., Laxmi, V., Gaur, Naval, S., Faruki, P., 2013. MCF : MultiComponent Features for

Malware analysis. *27th International Conference on Advanced Information Networking and Applications Workshops,* pp. 1076-1079.

Vinod, P., Laxmi, V., Gaur, M.S., Chauhan, G., 2012. MOMENTUM : MetamOrphic Malware Exploration Techniques Using MSA signatures. *International Conference on Innovations in Information Technology* , 32, pp.232–237.

Wang, Q., 2008. Fast Signature Scan. , p.1. Available at: https://www.google.com/patents/US7454418 [Accessed January 25, 2017].

Wang, W., Zhang, P., Tan, Y., He, X., 2009. A Hierarchical Artificial Immune Model for Virus Detection. *International Conference on Computational Intelligence and Security*, 1, pp.1–5.

Wrench, P., Irwin, B., 2015. Towards a PHP Webshell Taxonomy using Deobfuscation-assisted Similarity Analysis. *Information Security for South Africa,* pp. 1–13.

Yunlong, W., Chen, C., Huiquan, W., Xinhai, X., Jie, Z., 2012. Research on Malicious Code Detection Based on Least-squares Estimation. *International Conference on Computer Science and Electronics Engineering*. China: IEEE, pp. 124–128.

Zainal-Mokhtar, K., Mohamad-Saleh, J., 2013. An Oil Fraction Neural Sensor Developed Using Electrical Capacitance Tomography Sensor Data. *OALib Journal,* 13(9), pp. 11392.

Zhong, Y., Yamaki, H., Takakura, H., 2012. A Malware Classification Method based on Similarity of Function Structure. *12th International Symposium on Applications and the Internet,* pp. 256-261.

Zolkipli, M.F., Jantan, A., 2010a. A Framework for Malware Detection Using Combination Technique and Signature Generation. *Second International Conference on Computer*

Research and Development, pp.196–198.

Zolkipli, M.F., Jantan, A., 2010b. An Approach for Malware Behavior Identification and Classification. *School of Computer Science ,Universiti Sains Malaysia.* pp.191-194.