International Journal of Information Security, Privacy and Digital Forensics
Vol. 3, No. 2 December 2019

A Metamorphic Malware Detection System

O.A Adesegun
Babcock University
Ilisan Remo
Ogun State
adeseguno@babcock.edu.ng

O. Onilede
Babcock University
Ilisan Remo
Ogun State
oniledeo@babcock.edu.ng

B.S. Adeniyi
Babcock University
Ilisan Remo
Ogun State
adeniyib@babcock.edu.ng

A. Udosen
Babcock University
Ilisan Remo
Ogun State
udosena@babcock.edu.ng

Abstract: *Malware is the major cause of data breaches, resulting in financial losses in excess of $400 billion in 2017. The key ones, eluding malware scanners, are metamorphic. Malware metamorphic engines use varying obfuscating techniques to evade virus scanners. Current detection solutions are not effective against metamorphic malware. This study investigated metamorphic malware intrusions and developed a detection mechanism that combats them. A taxonomy of current malware detection mechanisms was created through extensive review of extant literature. Cosine similarity index, used to compare two files was added to dynamic link library, a feature derived from the disassembly process of a portable executable, to effectively determine if a file is a malware or not. A prototype of the system was developed using the java programming language. The virustotal website, which contains about 66 antimalware engines and scanners, was used to scan benign and malicious files. Experiments were conducted to prove that certain concealing techniques could aid malware evade existing antivirus scanners. A prototype of the detection system was evaluated against malware obfuscated using register re-assignment and dead code insertion techniques. Dead code insertion, register reassignment and instruction substitution were the three beclouding techniques used by malware metamorphic engines. The use of cosine similarity index together with linked libraries approach to detect metamorphic malware prototype was developed. A portable executable file is classified as a malware when its similarity index is high, 0.6 – 1, and it uses suspicious dynamic linked libraries. It was discovered that the most difficult obfuscating technique to implement by malware metamorphic engine is instruction substitution because non-availability of a line of code that is syntactically synonymous is probable. It was also observed that the register re-assignment technique on a malware made it evade every antimalware scanner on the virustotal website. Results showed that the prototype was 100% accurate as long as the right threshold was used, and as long as the parent malware was known. It was concluded that financial losses through malware invasion would be avoided, by adding the developed detection system to complement existing detection systems, in order to capture metamorphic malware effectively. This will benefit the general public, as the adoption of the proposed detection system by antimalware companies such as Symantec and McAfee, would lead to more efficacious antimalware systems. This would contribute to a more secure computing environment.*

Keywords: *Code substitution, Cosine similarity index, Dead code insertion, Linked library, Malware, Metamorphic engine, Register reassignment.*

1.0 INTRODUCTION

Metamorphism comes from two Greek words, Meta which means to change and Morph which means form. When put together as one word, it is used to describe an entity that changes its form/structure. Hence, the word has an application spans from geology to human virology and then to computer virology. In geology, it refers to rocks that are formed as a result of the changing form of mineral deposit over a period of time in the presence of varying temperature and pressure. In medicine, it describes mutated human viruses that may have changed their structure due to several factors making cure from such diseases more difficult. In the area of computer science, metamorphic software is an application/ programme that is able to change its code while still maintaining the same behaviour/ characteristics as the initial software before the metamorphism. A Metamorphic engines is that section of an application that helps the application to change it form or structure (i.e. code) while still achieving the same behaviour as a parent application.

The word Malware comes from two words, malicious and software, put together would be malicious software. A malicious software can be defined as an application that installs itself in stealthy way without permission to steal data and make services unavailable to legitimate users among others in the computer Systems. (Elhadi, Maroof & Barry, 2013; see also Jain & Bajaj, 2014; Mohan & Hamlem, 2012; Sharma & Sahay, 2014).

The origin of malware can be traced back to Jon von Neumann's studies in the 1940's, when he was studying self-replicating mathematical model known as an automaton. (Kamarudin, Md Sharif, & Herawan, 2013). However, the first malware on the Windows' platform was first discovered in 1986. It was a virus developed by two brothers from Pakistan as a proof of concept that the windows platform was unsafe. (Kamarudin, Md Sharif & Herawan, 2013; Milošević, 2013).

Since 1986 when the two brother came up with their proof of concept, there has been an upsurge in the number of viruses to more than 1,000,000 different computer virus strains (Kumar, Kumar, & Kumar, 2014). Some researchers have observed that at least one computer is infected with a malware every 39 seconds as found in (Mohan & Hamlem, 2012).

Today, there are several reason other than proof of concept why malware continue to exist. Some of these reasons include but are not limited to the following:

Financial gain both on the side of the cyber criminals and the anti-virus companies (Mohan & Hamlem, 2012).

To spy on people's internet activities (Jain & Bajaj, 2014).

To deny legitimate user's from accessing legitimate web services (Kumar, Kumar, & Kumar, 2014).

Since the appearance of Brain.A in 1986, there have methods of detecting viruses as well as either deleting them from the computer system or neutralizing their effect on the computer system they affect. As a result, there have been techniques developed over the years to evade antivirus detection. This evasion techniques started with encrypting malware and then move on to oligomorphic malware and then to polymorphic malware and today we have metamorphic malware (Rad, Masrom, & Ibrahim, 2012; You & Yim, 2010).

1.1 Significance of the Study

In the 1990s the number of malware were only between 1000 and 2300 but today there are about 1,000,000 different types of malware (Kumar, Kumar, & Kumar, 2014) and the number of data breaches reported has escalated over the years. Table 1.1 details a list of current reported attacks/data breaches and the target of those malware. Embarking on this kind of research will reduce financial loss to people and corporations incur as a result of malware. For instance in 2001 there was an outbreak of a malware, Red worm that affected computers running the Windows NT and Windows 2000 operating systems. The malware caused an excess of $2billion in terms of financial loss to businesses and individuals (Kamarudin, Md Sharif, & Herawan, 2013). This kind of losses can and should be avoided. The output of this research will be to find an effective countermeasure to metamorphic malware which will reduce the amount of successful malware attacks, saving money for individuals and corporations.

Table 1.0 List of Reported Malware Attacks in 2016

SN	Date	Target	Description	Attack Type	Target Class
1	05/08/2016	Android Users	Security researchers from Kaspersky Lab reveals the details of a mobile trojan distributed via the AdSense Network.	Malware (Mobile)	Single Individuals
2	18/08/2016	Eddie Bauer	Eddie Bauer announces that unknown intruders broke into its network and planted malware for capturing payment card data from its POS network. Data belonging to customers who used payment cards at all 370 Eddie Bauer locations in the US, Canada was compromised.	PoS Malware	Industry: Retail
3	26/08/2016	Two unnamed petrochemical complexes in Iran	Bloomberg reveals that Iran has detected and removed malicious software from two of its petrochemical complexes. The malware was "inactive" and seems not to be related to recent petrochemical fires.	Targeted Attack	Industry: Oil
4	26/08/2016	Millennium Hotels And Resorts (MHR)	Millennium Hotels And Resorts (MHR) announce investigations into a suspected data breach at its properties following notifications received from the US Secret Service.	PoS Malware	Industry: Hotel and Hospitality
5	26/08/2016	Noble House Hotels and Resorts (NHHR)	Noble House Hotels and Resorts also announces an investigation following a data breach at its PoS System.	PoS Malware	Industry: Hotel and Hospitality
6	30/08/2016	SWIFT	SWIFT discloses new hacking attacks on its member banks as it pressured them to comply with security procedures instituted after February's high-profile $81 million heist at Bangladesh Bank. In a private letter to clients, SWIFT says that new cyber-theft attempts - some of them successful - have surfaced since June, when it last updated customers on a string of attacks discovered after the attack on the Bangladesh central bank	Malware	Finance

Source: Hackmageddon (2016).

Today, implementing attacks have now been made easy such that with one or two software downloaded from the internet it is possible to orchestrate an attack. (Hansman & Hunt, 2005) On the other hand these attacks have increased a great deal in sophistication. Figure 1 illustrate's this idea by showing a straight line increase in attack sophistication year after year against an asymptotic decrease of knowledge of an intruder on how the attack really operates. Figure 2.3 shows malware in second place in terms of the frequency for which it causes data breach on assets. Table 2.1 list some the various malware construction kit that the malicious users use in orchestrating these attacks.

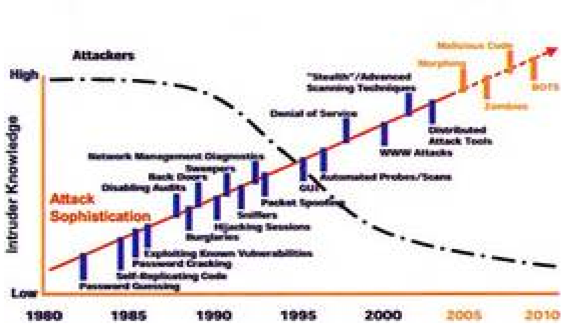


Figure 1: Attack Sophistication vs. Intruder Knowledge (Hahn, Guillen, & Anderson, 2006)

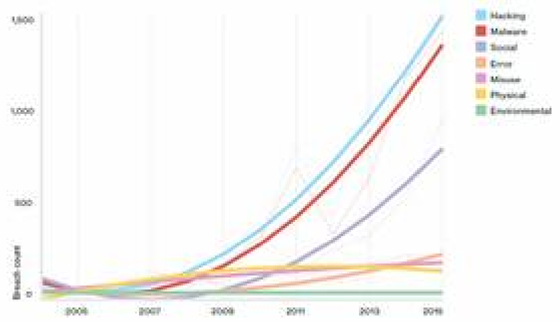


Figure 2: Graph indicating the Frequency of Various Actions in a Data Breach (Verizon Risk Team, 2016)

Table 2.0 Virus Construction Kits

Name	Version	Year
TeraBIT Virus Maker	TVM	2010
WAH C# worm generator	1.0	2009
NX Virus Creation Labs	NXVCL	2009
NEFI Virus Generator	v. 3.5	2008
Necrophilie	v.4.0	2007
Necronomikons M$ Auto CAD 2002 Virus Maker	NAVM	2004
Virus Creation Lab	VCL32	2004
Next Generation Virus Creation Kit	NGVCK 0.45	2002
NoMercy Excel Generator	NEG	1998
Phalcon-Skism Mass Produced Code Generator	PS-MPC 0.91	1992

Evolution of Malware

Malware analysis is like a cat and mouse game such that as new counter measures are developed to combat the existing malware, malware authors respond with new techniques to thwart such analysis (Sikorski & Honig, 2012). Therefore, over the years there has been changes in the methods used by malware authors. In this section, we examine these methods.

- Packing: This is a technique used by malware authors to compress/ shrink the size of their malware and to make analysis more complicated for antivirus applications. For packed malware unpacking has to be done before the file in question can be analysed.
- Polymorphism: In this technique, malware authors encrypt the malware code such that a static signature cannot be drawn for the malware. When the malware is triggered, it is decrypted and placed into memory for execution. Also, with each execution of the program, the malware is encrypted afresh which always result in different signatures for the malware.
- Metamorphism: This technique employs the use of dead code insertion, register reassignment, code substitution and so on

to becloud or obfuscate antimalware scanners, such that at each execution of the malware the syntax of the malware changes, but the malware remains the same semantically. This means that all functionality/behaviour of the malware remains the same as the root malware but the way it is written is different essentially making it almost impossible to detect such malware. Some other researchers such as Desai & Stamp (2010) developed other means of obfuscation that could be used by metamorphic malware.

2.1 Emergence of Antiviral Systems

The first antivirus company G DATA Software AG was founded in 1985 (G Data, 2016). Two years after, that is, in 1987 John McAfee designed his antivirus and distributed it as a shareware, and was credited as the inventor of the first antivirus application (Marshall Cavendish Corporation., 2008). Today the Mcafee antivirus company continues to exist and provide antiviral solutions and they do so as part of the intel security company. While John Mcafee created his antivirus and founded his company in the USA, two programing enthusiasts/friends Peter Paško and Miroslav Trnka in solvakia developed the NOD32 antivirus also in 1987, but started their company ESET only in 1992 (ESET, 2016; Vision Squared, 2016).

2.2 Components of Antiviral Systems

The components of an antivirus can be derived from the functions of an antivirus. The major functions of an antivirus software include prevention, detection, and removal of malware. As such the antivirus software have two major components. A detection component that helps in prevention and detection of malware during scanning of a computer and a component that removes these malware either by quarantining these malware or by deleting them. In some cases the antivirus software repairs the operating system damaged files or any other form of damage done by the malware. This is illustrated by Figure 4. The detection component of the antivirus software is the most important component. This is because if we are unable to detect a malware then we cannot be trying to reverse the malicious operations of these malware and altogether remove these malware.

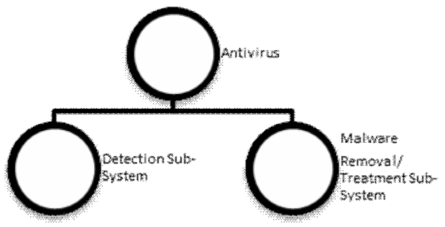


Figure 4 Components of an Antivirus.

2.3 Survey of Detection Mechanisms

Detection mechanisms are methods, schemes or algorithms used to identify whether a file is a malware or not. There are two major types of analysis used in detection systems namely; the static and the dynamic analysis.

2.2.1 Static Analysis

Static analysis essentially examines a portable executable without running or executing the portable executable by using tools referred to as disassemblers. The sole purpose of disassemblers is to convert a portable executable into assembly language code equivalent, while a de-compiler is used to convert an assembly language to a high level programing language such as C/C++ or Java equivalent, such that a programmer can then study/analyse the code generated to determine if the portable executable being examined is a malware or a normal file. The problem with using de-compilers is that the compilation process from high level programing language to machine language is lossy, therefore going back from machine language to a high level programing language would not generate a perfect representation.

The most common static analysis is the

signature based analysis. Other methods found in literature include the Opcode based analysis and the control flow based analysis.

a. Signature-based Analysis

In Signature based static analysis, signatures are created by the antiviral companies and sent as updates to the antivirus software to help these software identify malicious file. These signatures are usually created by malware analysts. These signatures are essentially gotten by either hashing a PE file or by extracting a string from the portable executable file.

Examples include String scanning, wildcard scanning, mismatch scanning and hashing.

Merits of using signature based analysis include Signature based analysis are very effective for known malware, producing less false positives and the time required classification of a file using this method is shorter.

Demerits of Signature based analysis include not effective against Zero(0) day malware, consuming a lot of hard drive space as each update is added to the signature database and not being effective against metamorphic malware.

b. Opcode Analysis

In opcode based analysis the portable executable is disassembled using a disassembler such as IDA pro in order to get the programs assembly language opcodes. Inferences are then drawn based on the opcode to determine if the portable executable file is malicious or not. Examples of detection mechanism in literature that use this scheme are Opcode HMM (Wong & Stamp, 2006), Opcode Graph Similarity (Runwal, Low, & Stamp, 2012), Opcode Historgram (Rad, Masrom, & Ibrahim, 2012), Chi Squared (Toderici & Stamp, 2013), and Simple Substitution Distance (Shanmugam, Low, & Stamp, 2013).

c. Control Flow Analysis

In control flow analysis a control flow graph is generated to represent all possible execution paths. The control flow graph is generated from analysis made from the disassembled portable executable, which may be in a high level programing language such as C/C++. In control flow analysis a database is created containing control flow graphs of malware which is then compared to the portable executable in question. Examples from literature include Code Graph (Lee, Jeong, & Lee, 2010), Model Checking (Song & Touili, 2012), API Call Grams (Faruki, Laxmi, Gaur, & Vinod, 2012), API CFG (Eskandari & Hashemi, 2012) Anotated Control Flow Graph (Alam, Horspool, & Traore, 2013) and Annotated Control Flow Graph (Alam, Horspool, & Traore, 2014).

2.2.2 Dynamic Analysis

Dynamic Analysis involves observing the behaviour of an executing application or portable executable to determine if it is malicious or not. As such safe environments are created so that the malware can be observed without damaging other files, or act in any other malicious way the malware writer intended. Though it is suggested that information flow based analysis exist as a subset of dynamic analysis (Alam, Traore, & Sogukpinar, 2014). There is stronger evidence that dynamic analysis is the same as information flow based analysis (Yin, Song, Egele, Kruegel, & Kirda, 2007). Examples from literature include Value Set Analysis (Leder, Steinbock, & Martini, 2009), Register Value Set Analysis (Ghiasi, Sami, & Salehi, 2012), and Behaviour Analysis (Yin, Song, Egele, Kruegel, & Kirda, 2007)

2.4 Model for Characterisation of Malware Metamorphic Engines

This section presents a graphic model and explains how this research characterizes malware metamorphic engines. In developing this model, the typical states of a malware were considered. Usually, when a software is written it is usually packed into a PE file which may be a .exe, .dll, .bat or any other PE format on the windows platform. At this level the instructions in the PE files are zeros (0s) and one (1s). Similarly when an attacker or persons with ill intentions write malware they would be packed it into one of these PE file formats. Disassembly is the process of converting these 0s and 1s to the closest form human readable form, the assembly language. The assembly language level being the level at which instructions are not lost due to the disassembly. A further de-compilation to a high level programing language would cause inaccuracy since the mapping of instruction from assembly language to a high level programming language has a one to many relationship. It is therefore at the level of assembly language a metamorphic engine is able to use any of the obfuscating techniques to generate new iterations of a malware. Figure 5 shows the model of how a new child malware can be generated from a parent malware.

Once the code has been disassembled the metamorphic engine uses at random and in different combinations any of the 3 obfuscating techniques graphically represented in the metamorphic engine section of Figure 5.

The metamorphic algorithm/obfuscating techniques used in the metamorphic engine include code substitution, dead code insertion and register reassignment.

a. **Instruction/Code Substitution:** This is implemented by having the code for metamorphic engine replace lines of code that can be re-written using a different syntax. For example, in the implementation of this work assembly lines with MOV R1, R2 for instance were replaced by PUSH R1 followed by POP R2 and line with XOR R1, R1 and SUB R1, R1 both zero the contents of register R1.

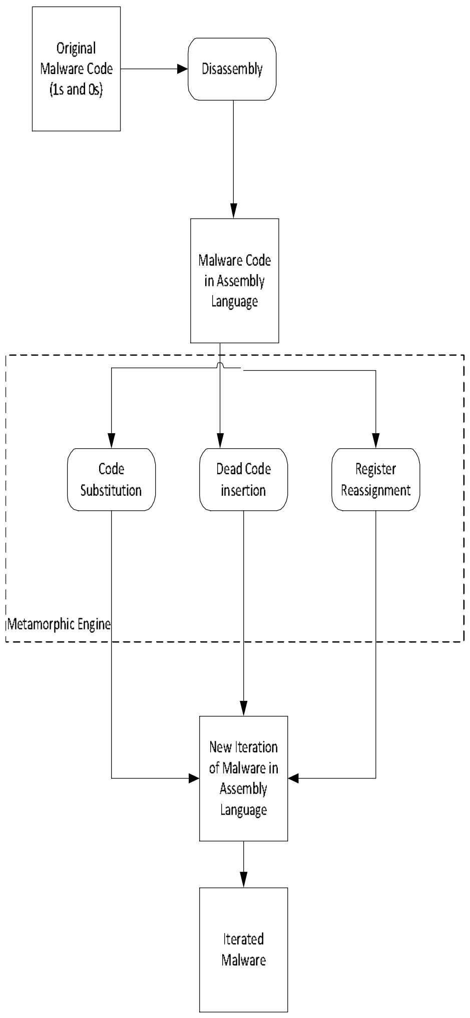

Original Malware Code (1s and 0s)
Disassembly
Malware Code in Assembly Language
Code Substitution
Dead Code insertion
Register Reassignment
Metamorphic Engine
New Iteration of Malware in Assembly Language
Iterated Malware

Figure 5: Characterisation of Metamorphic Engines

b. **Dead Code Insertion:** This is another method used by the metamorphic engine. It is simply implemented by inserting several NOP (which is a no operation in assembly language mnemonics) instructions in different lines of code of the program.

c. **Register Re-assignment**: This involves changing/swapping general purpose registers. That is swapping a register that holds a particular value without altering the code.

3.0 METAMORPHIC MALWARE DETECTION SYSTEM

The method proposed is classified as a static analysis since the Portable Executable (PE) file in question would not need to be executed before proper classification is done and this method produces faster results. The cosine similarity measure and the Linked Libraries call make up the solution proposed in this paper as a method that can be used to combat metamorphic malware. The use of similarity measures have been proposed by past works such as those found in Shanmugam, Low, & Stamp (2013), Runwal, Low, & Stamp (2012), and (Rad, Masrom, & Ibrahim (2012) but none of these works have a 100% accuracy in detecting metamorphic malware and will usually have to trade computational time for higher accuracies.

3.1 Database of Known Malware

The design of the proposed malware detection system is such that there is a database of known malware, metamorphic or otherwise. The major assumption this work takes is that the parent or a sibling malware must be in the database of known malware.

3.2 Input Sub-Section

The proposed detection system takes a PE file as its input. The PE file is then disassembled before any other operation can take place. Disassemblers such as IDA pro would be used to convert binary equivalent found in the PE files to assembly language so that analysis can be done by the detection stage.

3.3 Cosine Similarity Detection Sub-System

The cosine similarity measure used in this subsystem is a string based similarity measure that groups a collection of character as terms. The speed of the algorithm was the reason for its selection.

Each file in the database is cross checked with the file in question taking the following steps:

a. Collecting terms with distinct elements. This simply means each opcode, register and data are classified as elements but in cases where it occurs more than once in both files only one is taken at this stage.
b. A vector space is then created based on the frequency of terms in both files.
c. Computation is done based on the cosine similarity measure formula.

Once all this stages have been done for each of the malware in the database then the one that yields a similarity measure answer closest to 1 is the malware related to the file been scanned. Hence, classification can be made, determining if the file is a malware or not and also stating the malware to which the file is related. Figure 3 shows this diagrammatically.

This can be expressed mathematical as follows:

Let

P_1 (the known Malware file) be defined as a set $\{P_1\}$ &

P_2 (the suspicious unknown PE file) be define as a set $\{P_2\}$.

The collection of terms between P_1 and P_2 is defined as follows:

$T=\{t_1,t_2\ldots\ldots.,t_n\}$ $t_i \in P_1, P_2$ and each t_i is distinct.

The P.E file is represented as an n-dimensional vector $\overrightarrow{V_P}$.

Let tf(P,t) denote the frequency of term $t \in T$ in a P.E P. Then the vector representation of a P.E. is $\overrightarrow{V_P} = \left(\mathrm{tf}(P,t_1),\ldots\ldots,\left(\mathrm{tf}(P,t_n)\right)\right)$

After Building the vector space of the P.E., the similarity of the two files can be compared using the cosine similarity measure formula given as

$$\cos\theta = \frac{\overrightarrow{V_{P_1}} \cdot \overrightarrow{V_{P_2}}}{|\overrightarrow{V_{P_1}}| \times |\overrightarrow{V_{P_2}}|} = \frac{\sum_{i=1}^{n} P_1 \, P_2}{\sqrt{\sum_{i=1}^{n} {P_1}^2} \, \sqrt{\sum_{i=1}^{n} {P_2}^2}}$$
----- (1)

Where n= the number of assembly line code
For instance,

A known malware P.E file that has the following assembly language code stored in a database.

(named by line numbers in this instance):

Line 1
Line 2
Line 3
Line 4
.
.
.
Line n

A suspicious PE file has been disassembled and found to contain the following lines of assembly language code also, named by line numbers and lines that are different are labeled $diff_1,\ldots,diff_n$

Line 1
Line 2
Line 3
$diff_1$
Line 4
.
.
.
Line n

To compute the similarity of both files let use the mathematical model described above

if String $S_1 \in P_1$ = (Line1 Line2 Line3 line4 …..Line n)

and string $S_2 \in P_2$ =(Line1 Line2 Line3 $diff_1$ Line4….Line n)

Where each element $t_i \in S$ is equivalent to a line of assembly code

Then the collection of terms from S_1 and S i.e. $S_1 + S_2$ is

T = { Line1 Line2 Line3 $diff_1$ Line4….Line n}

Next, the frequency of string S_1 S_2 can be computed.

Finding the Vector of S_1

$$\overrightarrow{V_{S_1}} = (1, 1, 1, 0, 1 \ldots)$$

and the Vector of S_2

$$\overrightarrow{V_{S_2}} = (1, 1, 1, 1, 1 \ldots)$$

The similarity can therefore be computed using the cosine similarity measure formula as follows

$$\cos\theta = \frac{\overrightarrow{V_{S_1}} \cdot \overrightarrow{V_{S_2}}}{|\overrightarrow{V_{S_1}}| \times |\overrightarrow{V_{S_2}}|} = \frac{\sum_{i=1}^{n} S_1 \, S_2}{\sqrt{\sum_{i=1}^{n} {S_1}^2} \, \sqrt{\sum_{i=1}^{n} {S_2}^2}}$$
--------- (2)

The result then shows whether the two files are closely related or not because as mentioned earlier the closer the number tend to 1 the more similar the files are.

3.4 Linked Libraries and Final Evaluation

Linked Libraries simply refers to Libraries imported into a programme so that a programmer can leverage on pre-written functions. Therefore evaluating a programs linked libraries can give ample information about what the program would be doing. This information can be gotten by disassembling a PE file. When used alone as a method to detect malicious behaviour, there would be a high rate of false positives as normal programs could sometimes use the same libraries without intending/causing any harm. That is why we propose its use with the cosine similarity measure.

Table 3.0 Common Dynamic Linked Libraries Used by Malware

Dynamic Linked Library	Description
Kernel32	Contains core functionality of the Windows operating system such as access and manipulation of memory, files and hardware
Advapi32	Provides access to advanced core components such as service manager and the registry
User32	Contains all the user-interface components such as buttons, scroll bars and components for controlling and responding to user actions
Gdi32	Contains functions for displaying and manipulating graphics
Ntdll	Provides functionality like hiding or manipulating processes
WSock32 and Ws2_32	Provides networking functionality to programs
Wininet	Provides higher level networking functions that implement protocols such as FTP,HTTP, and NTP

Adopted from Sikorski & Honig (2012)

In this stage of our proposed solution the system checks through the Linked Libraries imported by the malware. Once Libraries that manipulate the operating system's registry or service manager such as Advapi32 or a Library such as Kernel32 which contain functions that help manipulate memory, files and the hardware are found. Then the system checks the result of the similarity index and makes a decision on whether the file in question is benign or if it is a malware. The linked libraries are gotten from the disassembly stage. Once a file is determined to be a malware the opcodes of such malware are stored in a file and updated to the database as an update for the system.

Implementation and Evaluation of the Proposed Metamorphic Malware Detection System

As a control for our experiment a simple word document (.docx file) containing 2 lines of text, 12kb in size was upload to the scanners available on the virustotal website and as expected it was classified as a clean file, as shown in Figure 7. In following ethical standards, a virus that only disables Wordpad, an application not used by most users of the windows operating system, was selected as a better malicious behaviour from the options given by the terabit virus maker as show in Figure 5. The virus generated was also not made to replicate itself either through a network or through a universal serial bus (USB) media. This was done to ensure that no other computer including the physical/ host machine was infected by the virus generated during the experiment.

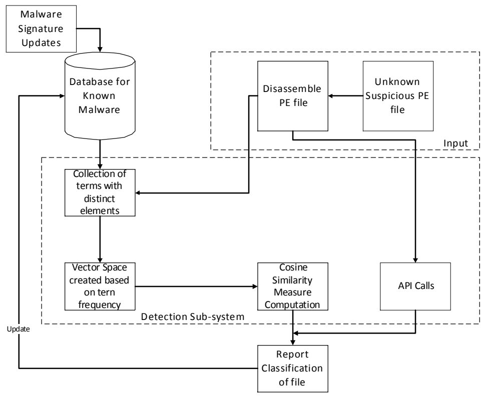

Malware Signature Updates
Database for Known Malware
Disassemble PE file
Unknown Suspicious PE file
Input
Collection of terms with distinct elements
Vector Space created based on tern frequency
Cosine Similarity Measure Computation
API Calls
Detection Sub-system
Update
Report Classification of file

Figure 6: Architecture of the proposed Malware Detection System

The file generated was then uploaded to the virustotal antivirus scanners and 57 of 66 scan engines classified the file scanned as a malware, while the other nine classified it as a normal file. This is shown in Figure 8. These initial tests show that virustotal scan engines are accurate to a large exetent. Since 86.3% of the malware scanners used in the industry today and present on the site conclude that the file is a malware. This is because the first file scanned is known to be a normal file and 60 of 60 scanners report that the file is normal. In the case where we generated a malware and scanned it 66 of 66 scanners also reported the file to be a malware.

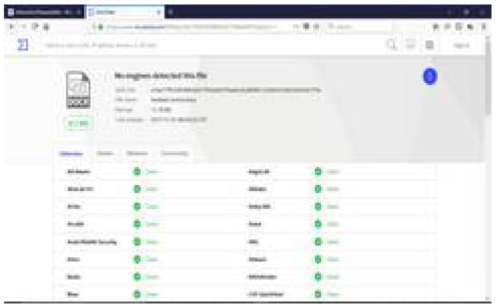
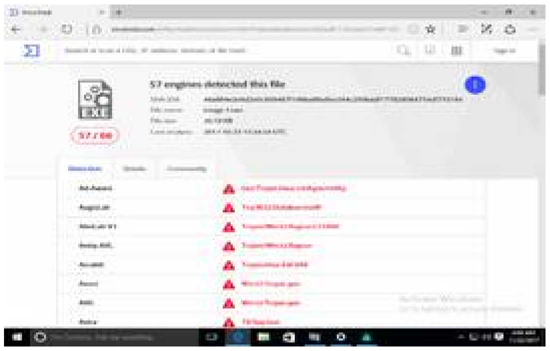

No engines detected this file

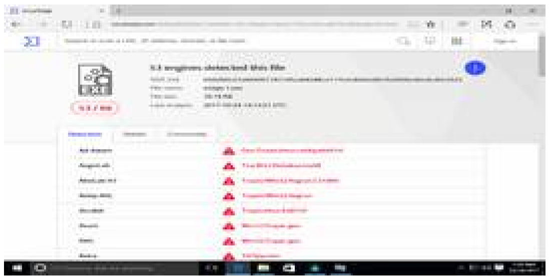
Figure 7: Result of a Scanned docx file

The malware generated was edited using Hiew text editor. Two NOP line were inserted in the P.E file as deadcode. The updated malware was then scanned at the virustotal website. As shown in Figure 8 53 of 66 anti-malware scanners detected the file as a malware. This was then taken even further by reassigning registers ecx for edx and vice versa still using the Hiew text editor. The results were staggering as no malware scanner was able to detect the malware as shown in Figure 10.


57 engines detected this file

57 / 66

Figure 8: Results of Scanned Malware without an Obfuscating Technique


53 engines detected this file

53 / 66

Figure 9: Results of Scanned malware with 2 deadcode Lines

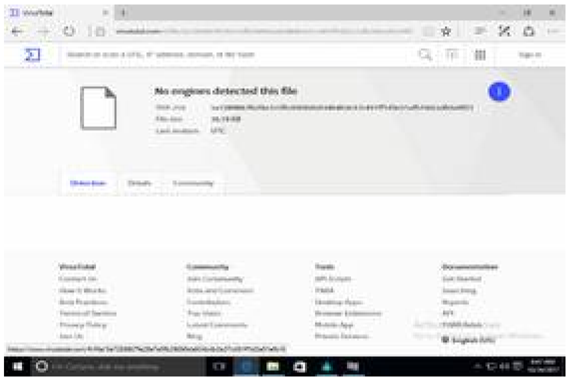
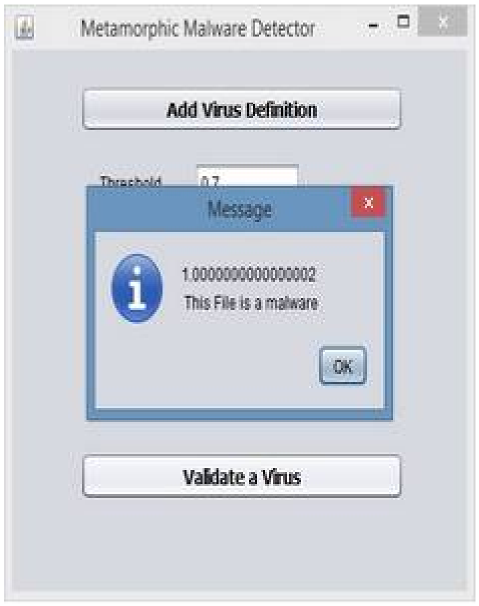

No engines detected this file

Figure 10: Results of Scanned malware with 2 deadcode Lines and register ecx and edx reassignments

Finally the malware to which no antimalware scanner was able to detect was disassembled using IDA pro and tested with our proposed metamorphic malware detections system. The test used the initial malware as the parent malware. Results of this test classified the result and child malware as malware with a very high similarity index of 1.0000000000000002. This is shown in Figure 11



0.7

1.0000000000000002

This File is a malware

Figure 11: Results from the metamorphic malware detector

Summary

In the area of computer science, metamorphic software is an application/programme that is able to change its code while still maintaining the same behaviour/ characteristics as the initial software before the metamorphism. A metamorphic engine is that section of an application that helps the application to change its form or structure while still achieving the same behaviour as a parent application. Some of the techniques used by the known engines highlighted and proven to help malware evade anti-malware scanners include register re-assignment, dead code insertion and instruction substitution.

The problem to which the research sought for a solution, is that, metamorphic malware have metamorphic engines which help these malware evade antimalware scanners using obfuscating techniques mentioned earlier. Hence a detection system that is specifically designed for metamorphic malware was designed.The following were milestones achieved to come up with the final metamorphic malware detection system: A taxonomy of current detection techniques was developed; A simulation of some characteristics of metamorphic engine was done and finally the design and implementation of the detection system was carried out.

Articles were gotten from google scholar search engine, deepdyve library, acm digital library, IEEE xplore digital library and direct downloads and requests to authors on the researchgate.net website. These articles were grouped and categorised according to method of detection which was represented diagrammatically as a taxonomy of malware detection systems. A simulation was also conducted to show the instruction substitution, register re-assignment and dead code insertion as some obfuscation techniques used by metamorphic engines. A design was developed for a metamorphic malware using the cosine similarity index and dynamic link libraries, this was represented diagrammatically and mathematically in the body of the work. The design was implemented using the java programming language. The virustotal website which contain about 66 antimalware engines and scanners was used to scan benign and malicious files during the evaluation experiment.

A prototype of our detection system was tested against a malware that was obfuscated using register re-assignment and dead code insertion obfuscating techniques. Results showed that our prototype was 100% accurate as long as the parent malware was known. The prototype however, could not detect zero-day malware, hence, the prototype is only effective against metamorphic malware if implemented as an antimalware solution should be implemented with other techniques already in existence.

4.0 CONCLUSION AND RECOMMENDATIONS

It can be concluded that the most difficult obfuscating technique to implement by malware metamorphic engine is instruction substitution since there may or may not be a line of code that is syntactically synonymous, that can be easily substituted.

In the evaluation experiment it was observed that as soon as register re-assignment obfuscating techniques was used no antimalware scanner was able to detect the malware. This is just one obfuscating technique out of the three simulated in this research. Also it is to be noted that the adversaries continue to look for ways to improve metamorphic engines by seeking for new ways to achieve obfuscation.

It is therefore the recommendation of this research that the prototype described, implemented and evaluated in this research, should be added to antimalware scanners to combat metamorphic malware specifically and other algorithms already in existence could be used against zero day malware. This would go a long way in reducing the financial losses recorded in threat reports year after year.

4.1 Limitations of the Study

Some of the limitation encountered during this study is that it was difficult to hunt down a metamorphic malware within the time frame allotted for the task to be completed. Time spent hunting for a metamorphic malware could easily run into years. Another limitation encountered is that websites filled with resources such as viruses, worms Trojans, books, virus creators which were helpful at the beginning of this research were also shut down by government agencies and "the powers that be" on the internet. Websites such as the vxhaven.org were shutdown. Malware will continue to remain a menace in the IT industry with metamorphic malware been one of the most difficult type of malware to detect due to its evasive nature.

REFERENCES

[1] Alam, S., Horspool, R., & Traore, I. (2013). MAIL: Malware Analysis Intermediate Language – A step Towards Automating and Optimizing Malware Detection. New York: ACM SIGSAC.

[2] Alam, S., Horspool, R., & Traore, I. (2014). MARD: A Framework for Metamorphic Malaware Analysis and Real-Time Detection. *Advanced Information Networking and Applications, Research Track – Security and Privacy.* Washington: IEEE .

[3] Alam, S., Traore, I., & Sogukpinar, I. (2014). Current Trends and the Future of Metamorphic Malware Detection. *SIN '14 Proceedings of the 7th International Conference on Security of Information and Networks* (pp. 9-11). Glasgow: ACM.

[4] Desai, P., & Stamp, m. (2010). A Highly Metamorphic Virus Generator. *Int. J. Multimedia Intelligence and Security*, 402-427.

[5] Eskandari, M., & Hashemi, S. (2012). ECFGM: Enriched Control Flow Graph Miner for Unknown Vicious Infected Code Detection. *Journal of Computer Virology, 8*(3), 99-108.

[6] Faruki, P., Laxmi, V., Gaur, M., & Vinod, P. (2012). Mining Control Flow Graph as API Call-Grams to Detect Portable Executable Malware. *Security of Information and Networks.* New York: ACM.

[7] G Data. (2016). *About G Data.* Retrieved September 7, 2016, from G Data Software: https://www.gdatasoftware.co.uk/about-g-data

[8] Ghiasi, M., Sami, A., & Salehi, Z. (2012). Dynamic Malware Detection Using Registers Valres Set Analysis. *Information Secuirty and Cryptology*, 54-59.

[9] Jain, M., & Bajaj, P. (2014). Techniques in Detection and Analyzing Malware Executables: A Review. *International Journal of Computer Science and Mobile Computing, 3*(5), 930-933.

[10] Kamarudin, I., Md Sharif, A., & Herawan, T. (2013). On Analysis and Effectiveness of Signature Based in Detecting Metamorphic Virus. *International Journal of Security and Its Applications, 7*(4), 375-384.

[11] Kumar, D., Kumar, N., & Kumar, A. (2014). Computer Viruses and Challenges for Anti-virus Industry. *International Journal of Engineering and Computer Science, 3*(2), 3869-3872.

[12] Leder, F., Steinbock, B., & Martini, P. (2009). Classification and Detection of Metamorphic Malware Using Value set Analysis. *Malware*, 39-46.

[13] Lee, J., Jeong, K., & Lee, H. (2010). Detecting Metamorphic Malwares Using Code Graphs. *SAC* (pp. 1970-1977). New York: ACM.

[14] Marshall Cavendish Corporation. (2008). *Inventors and Inentions* (Vol. 4). New York: Marshall Cavendish.

[15] Mohan, V., & Hamlem, K. (2012). Frankenstein: Stiching Malware from Benign Binaries. *In WOOT*, pp. 77-84.

[16] Rad, B., Masrom, M., & Ibrahim, S. (2012). Opcode Histogram for classifying Metamorphic Portable Executables Malware. *ICEEE*, 209-213.

[17] Runwal, N., Low, R., & Stamp, M. (2012). Opcode Graph Similarity and Metamorphic Detection. *Journal in Computer Virology, 8*(1), 37-52.

[18] Shanmugam, G., Low, R., & Stamp, M. (2013). Simple Substitution distance and metamorphic detection. *Journal of Computer Virology and Hacking Techniques*, *9*(3), 159-170.

[19] Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands on Guide to Dissecting Malicious Software*. San Francisco: No Starch Press Inc.

[20] Song, F., & Touili, T. (2012). Efficient Malware Detection Using Model-Checking. *FM: Formal Methods*. *7436*, pp. 418-433. Heidelberg: Springer.

[21] Toderici, A., & Stamp, M. (2013). Chi-squared Distance and Metamorphic Engines. *Journal in Computer Virology*, 1-14.

[22] Wong, W., & Stamp, M. (2006). Hunting for Metamophic Engines. *Journal in Computer Virology*, *2*, 211-229.

[23] Yin, H., Song, D., Egele, M., Kruegel, C., & Kirda, E. (2007). Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. *CCS '07 Proceedings of the 14th ACM conference on Computer and communications security* (pp. 116-127). New York: ACM.