Metamorphic Malware Detection Using Statistical Analysis

Kevadia Kaushal, Prashant Swadas, Nilesh Prajapati

***Abstract*— Typically, computer viruses and other malware are detected by searching for a string of bits found in the virus or malware. Such a string can be viewed as a “fingerprint” of the virus identified as the signature of the virus. The technique of detecting viruses using signatures is known as signature based detection.**

Today, virus writers often camouflage their viruses by using code obfuscation techniques in an effort to defeat signature-based detection schemes. So-called metamorphic viruses transform their code as they propagate, thus evading detection by static signature-based virus scanners, while keeping their functionality but differing in internal structure. Many dynamic analysis based detection have been proposed to detect metamorphic viruses but dynamic analysis technique have limitations like difficult to learn normal behavior, high run time overhead and high false positive rate compare to static detection technique. A similarity measure method has been successfully applied in the field of document classification problem. We want to apply similarity measures methods on static feature, API calls of executable to classify it as malware or benign.

In this paper we present limitations of signature based detection for detecting metamorphic viruses. We focus on statically analyzing an executable to extract API calls and count the frequency this API calls to generate the feature set. These feature set is used to classify unknown executable as malware or benign by applying various similarity function.

***Index Terms*— Metamorphic Virus, Malware Detection, API calls, Similarity measures.**

I. INTRODUCTION

In today’s age, where a majority of the transactions involving sensitive information access happen on computers and over the internet, it is absolutely imperative to treat information security as a concern of paramount importance. Computer viruses and other malware have been in existence from the very early days of the personal computer and continue to pose a threat to home and enterprise users alike. A computer virus by definition is “A program that recursively and explicitly copies a possibly evolved version of itself” [1]. A virus copies itself to a host file or system area. Once it gets control, it multiplies itself to form newer generations. A virus may carry out damaging activities on the host machine such as corrupting or erasing files, overwriting the whole hard disk, or crashing the computer. These viruses remain harmless but keep reproducing themselves. In any case, viruses are undesirable for computer users. There are several types of computer virus namely Stealth viruses, Worms, Trojan horses, Rootkits, Spyware, Encrypted and polymorphic viruses, metamorphic viruses etc.

The most popular virus detection technique used today is signature detection, which looks for unique strings pertaining to known viruses. Once detected, a virus is no longer a threat if the signatures on the system are kept up to date. Hence, signature-based detection is very effective for known malware but the major drawback is the inability to detect new, unknown malicious code that result in zero day attacks. The signature of a virus is typically created by disassembling the virus into assembly code, analysing it, and then selecting those sections of code that seem to be unique to the virus.

Metamorphic viruses alter the virus entire code without changing its impact. Code obfuscation techniques like garbage code insertion, register renaming, code reordering using jumps or sub-routine permutations and equivalent code substitution are used to generate various variants that belong to a virus family. Metamorphic viruses are quite potent against this signature based detection technique since they can create variants of themselves by code-morphing and the morphed variants do not necessarily have a common signature.

Our research focuses on extracting the behaviour of the malware through API call sequence analysis rather than the typical "pattern matching" detection process that are evaded by obfuscations of the byte sequence through metamorphic and polymorphic techniques. We identify the features of the extracted API calls in the unpacked executable binary. Our goal is to use this API calls feature set to distinguish between malicious and benign executable files.

II. MALWARE TYPES

Malware writers are continually trying to invent new methods to defeat antivirus software. Their worst enemies are the most commercially popular antivirus products. Since the appearance of first virus on microcomputer the battle between virus writers and anti-virus researchers never comes to end. To challenge virus scanning products, virus writers constantly develop new obfuscation techniques to make virus code more difficult to detect. To escape generic scanning, a virus can modify its code and alters its appearance on each infection. The techniques that have been employed to achieve this end range from encryption to polymorphic techniques, to modern metamorphic techniques.

A. *Polymorphic Malware*

Polymorphic malware like any other malware is a computer program that reproduces and causes harm to the computer. However, the variant produced by polymorphic malware

Manuscript received on July, 2012

Kevadia Kaushal, Department of Computer Engineering, BVM Engineering College, Vallabh Vidyanagar, Gujarat, INDIA.

Prof. Prashat Swadas, Head of Computer Engineering Department, BVM Engineering College, Vallabh Vidyanagar, Gujarat, INDIA.

Prof. Nilesh Prajapati, Associate Professor of Information Technology Department, BVM Engineering College, Vallabh Vidyanagar, Gujarat, INDIA.

constantly changes. This is done by filename changes, compression, encrypting with variable keys etc. The resulting variant has the same functionality as the parent malware. The decryptor (D) changes shape from generation to generation, but behind the encryption there is still a constant virus body.

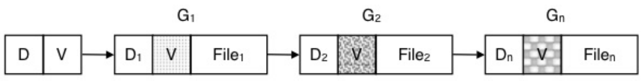
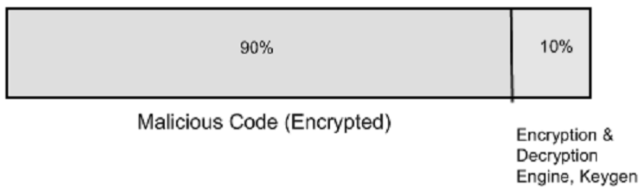


Figure 1: Polymorphic Malware Replication

Polymorphic malware produce different variants of itself while keeping the inherent functionality as same. This is achieved through polymorphic code. Concept of polymorphic code is core to a polymorphic malware. It is a style of code that mutates keeping the original algorithm the same.

The small section of polymorphic malware code containing the key generator and encryption-decryption module is responsible for morphing the malware and creating variants that do not have the same fingerprint. The problem of polymorphic malware is that the decryption block remained mostly the same in all the variants. The 10% of the can be used for as signature/fingerprint of the malware. The main body of a polymorphic malware consists of Malicious code and Encryption-Decryption code as shown in Figure 2.


90%
10%
Malicious Code (Encrypted)
Encryption & Decryption Engine, Keygen
Figure 2: Anatomy of a Polymorphic Malware

B. *Metamorphic Malware*

Creating a polymorphic virus is a very complex and challenging task for virus writers. They often waste months on creating a new polymorphic virus, a virus that can take an antivirus vendor just a few hours to detect. The problem with polymorphic viruses is that they eventually have to decrypt themselves and present their constant body in memory in order to function. Advanced detection techniques can wait for the virus to decrypt its self and then detect it reliably.

Unlike, polymorphic malware, metamorphic malware contain a morphing engine. The morphing engine is responsible for obfuscating the whole malware. The body of a metamorphic malware can be broadly divided into two parts namely Morphing engine and Malicious code as shown in Figure 3.

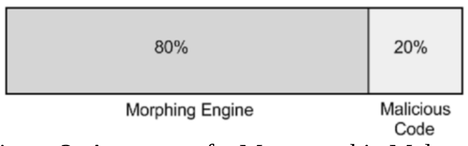


Figure 3: Anatomy of a Metamorphic Malware

The metamorphic virus uses no encryption - with some exceptions - to hide its code. In fact, an advanced metamorphic virus has no constant data anywhere between generations; new generations look completely different. Simply speaking, this virus changes its shape every time it infects a new file or a new system, while preserving its functionality. No hexadecimal search strings can be extracted from it, thus detection using strings is virtually impossible. Figure 4, illustrates the replication of a metamorphic virus. It is obvious that no constant data exists between different generations.

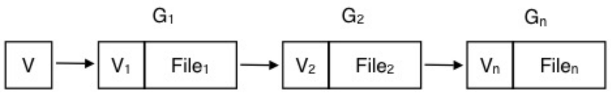


Figure 4: Metamorphic Virus Replication

Metamorphic malware represent the next class of virus that can create an entirely new variant after reproduction. The new variant produced is in no-way similar to the original variant. Metamorphic malwares do not use encryption as most polymorphic malware. Instead metamorphic malwares reply on code obfuscation techniques. Since the metamorphic malwares have do not produce variants having same body, they easily evade signature based detection. Since, most current anti-virus software primarily use signature based detection, metamorphic malware currently are greatest threat.

III. OBFUSCATION TECHNIQUES

To avoid detection, metamorphic viruses use several different techniques to evolve their code into new generations that look completely different, but have exactly the same functionality. This section describes in detail many of these techniques.

A. *Garbage Code Insertion*

Garbage or do-nothing codes are programming instructions that are a part of the program physically but not logically. They are not related to the program's outcome. Do-nothing instructions such as register exchanging (XCHG) slow down code emulation. Other instructions such as "NOP","MOV ax, ax", "SUB ax, 0", etc make the virus look different and thus possibly escape heuristic analysis. Garbage instructions may also be branches of code that are never executed or which have some calculations done on the variables declared in other garbage blocks. The main idea of this code obfuscation technique is to confuse and exhaust the virtual machine or person traversing the virus code.

TABLE I
GARBAGE CODE INSERTION

Original Code	**After Garbage Insertion**
push ecx push eax pop ebx push eax pop ebx mov ebp, [ebx] pop ebx	push ecx **nop** push eax pop ebx push eax **nop** pop ebx **nop** mov ebp, [ebx] pop ebx

B. *Register Usage Exchange*

This technique replaces the use of a register in an instruction with another unused register. Though this method has no impact on program behaviour, it does serve to evade virus signature based scanners.

TABLE II
REGISTER USAGE EXCHANGE

Original Code	After Register Exchange
pop edx	pop eax
mov edi,0004h	mov ebx,0004h
mov esi,ebp	mov edx,ebp
mov eax,000Ch	mov edi,000Ch
add edx,0088h	add eax,0088h
mov ebx,[edx]	mov esi,[eax]
mov [esi+eax*4],ebx	mov [edx+edi*4],esi

C. Subroutine Permutation

In this type of code obfuscation the order in which the subroutines appear in the code is changed. This order is irrelevant and does not impact the functionality of the malware as the order in which a subroutine appears in the program is totally irrelevant and does not affect the execution of the program. As shown in Table III, the modules are re-ordered.

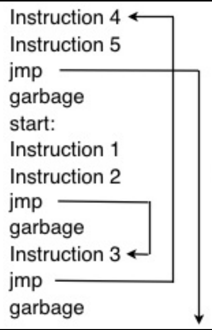
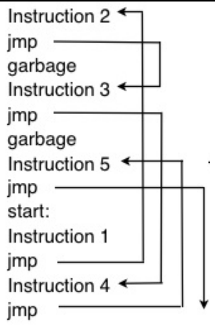
TABLE III
SUBROUTINE PERMUTATION

Original Code	After Permutation
Function1:	Function2:
MOV EAX, [X]	MOV EBX, [Y]
Function2:	Function1:
MOV EBX, [Y]	MOV EAX, [X]
Function3:	Function3:
ADD EAX, EBX	ADD EAX, EBX
MOV [X], EAX	MOV [X], EAX

D. Code Reordering through Jump Instructions

Similar to Subroutine permutation, code reordering is a kind of code permutation. This technique uses the JMP instruction as the backbone for obfuscation. This helps in basically creating different permutations of the code while keeping the functionality constant. The number of additional JMP instructions added will be proportional to the number of lines that are re-ordered.

TABLE IV
CODE REORDERING

Original Code	After Code Reordering
 Instruction 4 Instruction 5 jmp garbage start: Instruction 1 Instruction 2 jmp garbage Instruction 3 jmp garbage	 Instruction 2 jmp garbage Instruction 3 jmp garbage Instruction 5 jmp start: Instruction 1 jmp Instruction 4 jmp

E. Equivalent Instruction Replacement

Some metamorphic viruses are able to replace some of their instructions with other equivalent instructions. For example, the virus could replace the instruction “xor eax, eax” with the instruction “sub eax, eax.” Both instructions perform the same function - zeroing the content of the eax register - but have a different opcode.

TABLE V
EQUIVALENT INSTRUCTION REPLACEMENT

Original Code	After Replacement
push ebp	push ebp
mov ebp, esp	**push esp**
mov esi, ptr [ebp + 08]	**pop ebp**
test esi, esi	mov esi, ptr [ebp + 08]
mov edi, ptr [ebp + 0c]	**or esi, esi**
or edi, edi	mov edi, ptr [ebp + 0c]
xor edx, edx	**test edi, edi**
	sub edx, edx

IV. PROPOSED METHODOLOGY

Malware detectors are used to scan a computer system to identify malware, with the main purpose of preventing it from adversely affecting the system. The current malware detection methods usually rely on existing malware signatures with limited heuristics and are unable to detect those malware that can hide itself during the scanning process and those metamorphic malware that apply sophisticated obfuscation techniques. An anti-virus (AV) engine must perform three main tasks to protect computers: Scanning, Detection, Removal. A Malware detector D is defined as a function whose determine the executable program (p) which program is malicious or benign D: → P {malicious, benign}. Modern and traditional anti-malwares scan the programs (p) in a system for a byte sequence or malware signature (s) which it stored in the database engine. If the signature is found in the program (p), it will be identified as a malware, otherwise it is declared as benign, and this is represented in the equation below.

$$D(P) = \begin{cases} malware & if \quad s \in p \\ benign & otherwise \end{cases}$$

In this section we proposed our methodology to detect metamorphic malware based on behavioural patterns using statistical features of application programming interface (API) calls from executables using similarity measurement to detect and classify even unknown malware. Malware detection using API feature was the first method started by the New Maxico Tech's malware group. This method is called SAVE (Static Analyzer for Vicious Executables). In this method signature of malware is determined by its sequence of API calls. Each sequence is denoted as a vector [2]. In our proposed approach we use similarity measurement functions on extracted API which gives us feature set for classifying executables as malware or benign file.

Our proposed approach extracts the API call frequency from binary files or executable files and applies similarity measure for identification of malware. We proposed the approach in form of four major steps as shown in Figure 5.

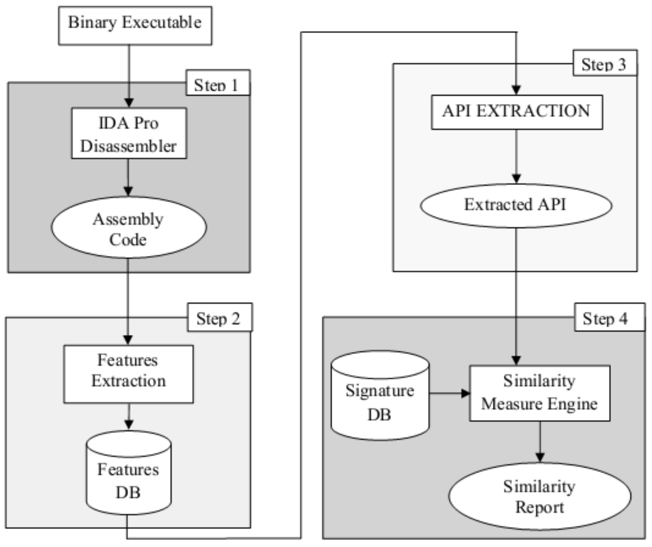


Figure 5: Overview of Proposed Methodology

First step, disassemble binary executables to retrieve the assembly program: We disassemble binary executable files using freeware IDA Pro Disassemble. IDA Pro disassemble binary file into assembly language and it also automatically recognize API calls for various compilers.

Second step, extract features from assembly program: In this step system processes the output from step 1 with the aid of IDA Pro plug-in to extract features from assembly code. Plug -in also stores this extracted features from binary executable into Slate database for effective and batch analysis.

Third step, extract frequency of important features: We first count the frequency of each important feature. This procedure gives vector of frequency distribution of each important feature for a given malware and it is stored in database. This database forms signature database for known malware.

Fourth step, applying similarity measurement for unknown binary executable: For identification of unknown binary executable above three steps are similar. After step 3 outputs is compare with existing signature database using similarity measure like Cosine, Extended Jacquard measure and Pearson correlation. Mean value of three measure is used to generate similarity score report. This similarity score report is used for identification of malware by setting appropriate threshold value. Value of threshold need to be investigated by empirical testing.

V. SIMILARITY ANALYSIS

A signature is a frequency distribution of API calls of a known virus that has been previously identified. Let's denote it Vs (vector of signature). The frequency distribution of API calls of a suspicious PE binary file is denoted Vu (vector of unknown). To identify whether the new executable with signature Vu is an obfuscated version of the virus represented by Vs, we measure the similarity between Vs and Vu.

We apply the traditional similarity functions on Vs' and Vu'. Cosine measure, extended Jaccard measure, and the Pearson correlation measure are the popular measures of similarity for vector. The cosine measure is given below and captures a scale-invariant understanding of similarity.

A. *Cosine Similarity*

Cosine similarity is a measure of similarity between two vectors of n dimensions by finding the angle between them.

$$\text{Cosine Similarity} = \cos\text{-}1 \left[\frac{Vs' \bullet Vu'}{\|Vs'\| * \|Vu'\|} \right]$$

B. *Extended Jaccard Measure*

The extended Jaccard coeffi¬cient measures the degree of overlap between two sets and is computed as the ratio of the number of shared attributes of Vs' AND Vu' to the number possessed by Vs' OR Vu'.

$$\text{Extended Jaccard Measure} = \frac{Vs' \bullet Vu'}{\|Vs'\|^2 + \|Vu'\|^2 - Vs'Vu'}$$

C. *Pearson Correlation*

Correlation gives the linear relationship between two variables. For a series of n measurements of variables Vs' and Vu', Pearson correlation is given by the formula below.

$$\text{Pearson Correlation} = \frac{\sum_{i=1}^{n}\left(Vs'_i * Vu'_i\right) - \left(n * \overline{Vs'} * \overline{Vu'}\right)}{(n-1) * S_{V_{s}'} * S_{V'_{u}}}$$

where Vsi' and Vui are values of variable Vs' and Vu ', respectively, at position i, n is the number of measurements, Svs' and Svu' are standard deviations of Vs' and Vu', respectively, and $\overline{Vs'}$ and $\overline{Vu'}$ are means of Vs' and Vu', respectively.

We calculate the mean value of the three measures. For a particular measure between a virus signature and a suspicious binary file, S(m) (Vsi', Vu'), which stands for the similarity between virus signature i and a suspicious binary file. Our similarity report is generated by calculating the S(m) (Vsi', Vu') value for each virus signature in the signature database. The index of the largest entry in the similarity report indicates the most possible virus the suspicious file is (a variant of). Let us denote the index as i_{max}. By comparing this largest value with a threshold, we make a decision whether the binary file is a piece of malware and identify which malware it is.

VI. CONCLUSION

Current signature based detection technique used by Anti Virus scanners can be easily defeated by applying polymorphic and metamorphic technique which generates variants of existing virus. Metamorphic virus uses mutation engine to evolve during propagation phase. Each evolved version viruses are functionally similar but their code structure changes considerably. So, signature based detection technique fails to detect such variant if signature is not present in its database. Mutation engine of metamorphic virus uses various code obfuscation technique to produce new variant each time virus propagate so it is believed that metamorphic viruses are hard to detect.

Proposed methodology aimed at detecting metamorphic virus, the key assumption is that to preserve functionality a metamorphic virus should contain sufficiently similar API calling sequence. So, we propose a statistical analysis of the API calls from binary executable using similarity measurement function. The similarity measure for finding distance of unknown malware with known behavior so that obfuscated malware could be detected efficiently.

Our aim is to develop a system that reverse engineers the unknown binary executable code without any need for manual inspection of assembly code & classify it as malware or benign.

ACKNOWLEDGMENT

It is honour and pleasure to express my heartfelt gratitude to those who helped me and also contributed towards the preparation on this research. I am indebted to my guide Prof.P.B.Swdas and Prof.N.B.Prajapati whose invaluable guidance and timely suggestion and constructive encouragement inspired me to complete the dissertation work in the present form. I would like to thank to the entire team of M.E. staff whose direct and indirect suggestion helped me creating this work. I would like to pay a special thanks to my parents for the sparing their invaluable time and inspiring me. Although there remain some names but none are remain un-thanked.

REFERENCES

[1] P. Szor, *The Art of Computer Virus Research and Defense Professional*, 1st ed., Addison-Wesley, 2005.
[2] Sung A H, Xu J, Chavez P, Mukkamala S, "Static Analyzer For Vicious Executables (SAVE)," Proceedings of 20th Annual Computer Security Applications Conference (ACSAC), pp. 326–334, IEEE Computer Society Press, ISBN 0-7695-2252-1 , 2004.
[3] Karnik Abhishek, Goswami Suchandra, Guha Ratan, "Detecting Obfuscated Viruses Using Cosine Similarity Analysis," First Asia International Conference on Modelling & Simulation, IEEE, pp. 165 -170,2007.
[4] Madhu K Shankarapani, Subbu Ramamoorthy, Ram S Movva, Srinivas Mukkamala, "Malware detection using assembly and API call sequences", journal in Computer Virology ,Springer, 2010.
[5] S. Momina Tabish, M. Zubair Shafiq and Muddassar Farooq, "Malware Detection using Statistical Analysis of Byte-Level File Content", ACM, 978-1-60558-669-4, 2009.
[6] V. Sai Sathyanarayan, Pankaj Kohli, and Bezawada Bruhadeshwar, "Signature Generation and Detection of Malware Families", Springer-Verlag Berlin Heidelberg 2008, pp . 336–349, 2008.
[7] Wen Fu, Jianmin Pang, Rongcai Zhao, Yichi Zhang, Bo Wei, "Static Detection of API-calling Behavior from Malicious Binary Executables", International Conference on Computer and Electrical Engineering, IEEE, 2008.
[8] Karnik Abhishek, Goswami Suchandra, Guha Ratan, "Detecting Obfuscated Viruses Using Cosine Similarity Analysis," First Asia International Conference on Modelling & Simulation, IEEE, pp. 165-170,2007.
[9] S. Momina Tabish, M. Zubair Shafiq and Muddassar Farooq, "Malware Detection using Statistical Analysis of Byte-Level File Content", ACM, 978-1-60558-669-4, 2009.
[10] Babak Bashari Rad, Maslin Masrom, "Metamorphic Virus Detection in Portable Executables Using Opcodes Statistic", Proceeding of the International Conference on Advance Science, Engineering and Information Technology 2011, ISBN 978-983-42366-4-9, 2011.
[11] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie and N. Tawb, "Static Detection of Malicious Code in Executable Programs".
[12] Mamoun Alazab, Sitalakshmi Venkataraman, Paul Watters, "Towards Understanding Malware Behavi our by the Extraction of API calls", Second Cybercrime and Trustworthy Computing Workshop, IEEE, 2010.
[13] P Vinod, R Laxmi & Gaur M, “Survey on Malware Detection Methods”, Hack vol. 74, 2009.
[14] IDA Pro, Online, www.hex-rays.com/
[15] MySQL, Online, www.Wikipedia.org/
[16] Python Programming, Online, www.python.org/