

Metamorphic Virus Detection using Uniform Characteristic

Nilambari G. Narkar
Student, Computer Engineering Dept
D. J. Sanghvi College of Engineering
Mumbai, India
narkar.nilam@yahoo.co.in

Dr. Narendra M. Shekokar
HOD, Computer Engineering Dept
D. J. Sanghvi College of Engineering
Mumbai, India
narendra.shekokar@djsce.ac.in

Abstract:-A metamorphic virus generates copies of itself using code obfuscating techniques. A new generation virus has the same functionality as the parent but it has a different pattern. The pattern change makes it difficult for signature-based virus detection technique to identify that different iterations are the same malicious program.

In earlier research, hidden Markov models (HMMs) have been used to identify some metamorphic viruses. However, recently metamorphic viruses evade HMM-based detection. In this paper, we propose uniform characteristic metamorphic virus detection techniques based on alter distance.

Keywords:-Metamorphic Virus.

1. INTRODUCTION

The internet is a worldwide network connecting millions of computers. A computer virus is a program that, when executed, replicates itself without the user's permission or knowledge [11]. On execution virus loads itself into the computer's memory and continues to run even after the host files shut down its execution. A typical virus comprises of three modules which are infect, trigger and payload [6]. Virus construction kits are available, which makes virus creation extremely simple [17]. As a result, users who have negligible knowledge can create potential viruses. There are several antivirus programs available that can be used to detect malware [14]. The most commonly used antivirus detection technique is signature detection, which consists of searching the content of the files in "signatures" stored in antivirus database. A signature consists of a string of bits found in a

presents our proposed techniques. Section VI presents our conclusions.

2. CODE OBFUSCATION TECHNIQUES

Metamorphic virus use code obfuscation technique to change new generation "shape" but not its behavior. Code obfuscation techniques are performed on both the data section and the control flow of an assembly program [13]. Control flow obfuscation technique involves unconditional jump instructions and instruction reordering. Data flow obfuscation is achieved by transposition, junk code insertion, equivalent instruction substitution, register renaming, and subroutine permutation. This makes it more resistant to code emulation detection technique. The virus body has different structure with same functional behavior.

1.1 Register Swap (Register Usage Exchange)

Register swapping is metamorphic techniques. This technique changes register operands in the virus body with different equivalent registers. Instructions remains constant for all virus generation, only register changes. For example, instruction

particular virus. To escape signature-based detection, virus writers sometimes use code obfuscation techniques which alter the configuration of the code. The techniques used to obfuscate code include reordering assembly instructions, dead code insertion, and equivalent instruction substitution [3]. The result is a morphed virus that has the same functionality as the original[1]. However, if the morphing is sufficient, no common signature will be present [1]. These metamorphic viruses generated dissimilar copies of it using code morphing techniques [1].

This paper is organized as follows. Section II contains various code obfuscation techniques that can be used to generate highly morphed viruses. In Section III, contain virus detection techniques. Then in Section IV, contain related work. Section V

"movedi, 0004h" can be substituted with "movebx, 0004h." The W95/RegSwap virus [26] is an example of metamorphic virus that uses the register swap technique. Wildcard string can usually detect such metamorphic viruses [15].

1.2 Junk Instruction Insertion

Junk code insertion is technique employed by metamorphic viruses to change the look of the virus body. Junk instructions do not have an effect on the program outcome and it may not even execute [11].

1.3 Equivalent Instruction Substitution

Equivalent instruction substitution is technique used to substitute an instruction or a block of instructions with an equivalent instruction or an equivalent block of instructions [13].

1.4 Instruction Transposition

Transposition is used to change the sequence of execution of the instructions. Instruction permutation between the instructions does not affect the program outcome and it can be

applied only if there is no mutual dependency between the instructions[1].

1.5 Subroutine Transposition

Subroutine transposition is technique that changes the look of a virus by reordering the subroutines. There can be $n!$ different generation of subroutines for n different subroutines. The W32/Ghost virus [15] implements the subroutine transposition technique.

3. VIRUS DETECTION TECHNIQUE

Some of the most commonly used virus detection techniques are:

3.1 Signature Detection

Signature detection technique is widely used to detect viruses. The groups of bits, which are found in a virus file are stored in the databases. [25] The virus scanner searches the complete file system for signatures. If the signature is found then the file is marked as infected.

3.2 Heuristic Analysis

Heuristic analysis is used to detect newly generated or unidentified viruses. There are two types of heuristic scanning techniques. The difference between the two approaches is whether the heuristic scanner makes use of CPU emulation to scan for virus like behavior or not. A heuristic scanner has two phases of operation when scanning files for viruses. In the first phase of the operation, the scanner observes the behavior of the program and looks for a specific area in the file where the virus would attach itself.

In the second phase, it determines the program logic which can be executed by computer instructions in the specific areas identified in the first phase [8]. The program is flagged as a virus, if it contains a certain percentage of the computer instructions similar to the virus instructions. The Heuristic analysis results in many false positives as it mostly operates on the basis of past experience [18]. This might not detect new viruses that contain code different from a previously known virus program.

4. RELATED WORK

W.Wong[2] has proposed detection of metamorphic virus using hidden Markov models (HMMs).HMM is a state machine where the transitions between states have fixed probabilities. Each state in an HMM was associated with a probability distribution for observing a set of observation symbols.

HMM was “trained” to represent a set of data, which is usually in the form of observation sequences. The states in the trained HMM then represent the features of the input data, while the transition and the observation probabilities represent the statistical properties of these features. Given any observation sequence, it was matched against a trained HMM to determine the probability of seeing such a sequence. The probability was high if the sequence was “similar” to the training sequences[2].

SrilathaAttaluri[27]has proposed detecting metamorphic virus using Profile Hidden Markov model where multiple sequences of genes are combined to form an alignment that contains the hidden relation between them. A model created from the resultant multiple sequence alignment (MSA) is used to measure the relativity of an unknown sequence to a family. Several of the metamorphic viruses were not detected by commercial virus scanners. The goal of the research presented here is to use uniform characteristic approach to see if we can detect the metamorphic viruses.

5. PROPOSED METHODOLOGY

Proposed system is working in two phases- Learning & Testing.

In learning phase, CPU emulator is used. It is the software that simulates the behavior of a CPU as it allows virus to execute in an environment from which it cannot escape. Virus code that runs in an emulator is then disassembled into individual instructions. Junk instructions inserted in code are removed. Then from disassembled virus instructions extract virus features& API call. Count frequency of occurrence of particular feature & API call. Store frequency of occurrence in database. Features & API calls are arranged in descending order depending on frequency of occurrence. Figure 1 shows functioning of learning phase.

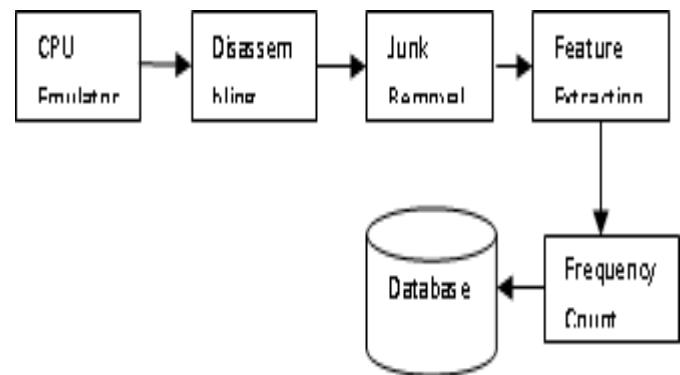


Figure 1.Learning Phase

Next in testing phase, CPU emulator is used so that virus detection is performed at entry point. Initially a code is disassembled into individual instruction. Junk instructions are removed.

Features are extracted & first similarity test is performed. If feature are similar then extracted feature is virus & virus database is updated else next Similarity Analysis is performed on extracted feature. If similarity score is equal to threshold then extracted feature is virus & virus database is updated else next Alter distance test is performed on extracted feature.

If Alter distance value greater than zero then extracted feature is virus & virus database is updated else next dummy file of known size is provided for execution.

If dummy file size increases after execution then extracted feature is virus & virus database is updated.

Periodically a database is refreshed so that frequently encountered features are on the top in database. Figure 2 shows working of testing phase.

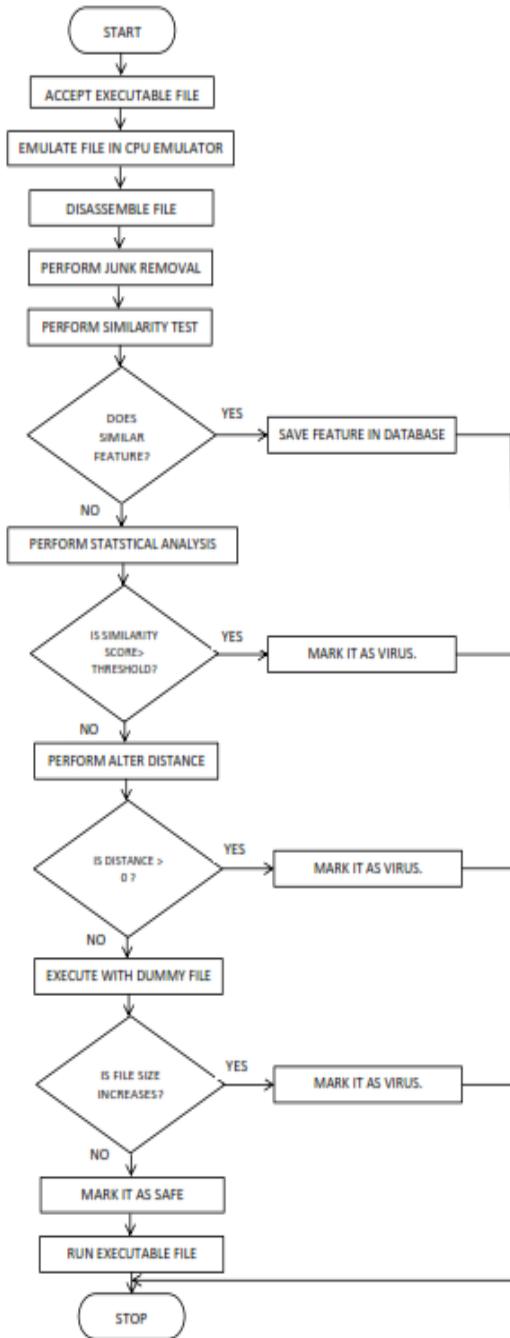


Figure 2. Flowchart of Testing Phase

5.1 Similarity Index Test

To measure the similarity between the virus copies, two assembly files were compared based on the op-code sequence presented in them. The following steps were followed to compute the similarity between two files[4]

- Given two assembly files, file1.asm and file2.asm, the sequence of op-codes were taken out from both the files,

excluding labels, comments, blank lines and other directives. Resulting op-code sequences were called F1 and F2 for file1.asm and file2.asm, respectively. m and n represented the number of op-codes in F1 and F2, respectively. A number was assigned to each of the op-code in the resulting op-code sequence: 1 for the first op-code, 2 for the second, and so on.

2. Op-code sequence was divided into subsequences of three consecutive op-codes. The op-code sequences, F1 and F2, were compared considering all the subsequences of three consecutive op-codes from each sequence. A match was considered, if three op-codes are the same in any order. For example F1 was (add, call, test, sub, mov) and F2 was (mov, add, call, sub, test). The sequence (call, test, sub) in F1 matched with (call, sub, test) of F2. The process was repeated for all the op-codes in F1 and F2.

For example:

Table 1: Similarity Index Test

Opcode	Op-code sequence	
Index	F1	F2
0	add	mov
1	call	add
2	test	call
3	sub	sub
4	mov	test

3. m and n represented total number of op-code in F1 and F2 respectively. To find the total number of matches in F1, all matches were computed and added together. The total number of match was divided by m to get the similarity score of F1. Similarly, similarity score for F2 was computed.

Similarity score for F1:

$$S1 = (\text{total number of matches in F1}) / m$$

Similarity score for F2:

$$S2 = (\text{total number of matches in F2}) / n$$

4. The similarity score between the files, file1.asm and file2.asm was obtained by taking the average of F1 and F2.

$$\text{Total Similarity Score: } (S1+S2)/2.$$

5.2 Statistical Analysis:

If feature extracted is API call then it is compared with existing database using similarity measure like Cosine, Extended Jaccard 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[21].

- Cosine Similarity :** Cosine similarity is a measure of similarity between two vectors of n dimensions by finding the angle between them.
- Extended Jaccard Measure:** The extended Jaccard coefficient 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'.

- c) **Pearson Correlation:** Correlation gives the linear relationship between two variables. For a series of n measurements of variables Vs' and Vu'.

5.3 Alter Distance

An alter distance is used to measure the number of alter operations needed to alter one string into another. For given string s1 and s2, the alter distance is calculated based on the amount of difference between the two sequences of the strings. The difference in the strings is based on the sequence of characters each string contains. Allowable alter operation to transform one into another is insertion, deletion and substitution. For example, the alter distance between "meeting" and "leading" is 3, as the following three alters are required to change one string into the other, and there is no another way to get the same result in less than three alters:

1. meeting → leeting (substitution of ,l for ,m)
2. leeting → leating (substitution of ,a for ,e)
3. leating → leading (substitution of ,d for ,t)

5.4 Computing Alter Distance

For a given two sequence s1 and s2 and three alter operations, the alter distance for the sequences is valued to transform sequence s1 to sequence s2. We use dynamic programming to find the alter distance from s1 to s2. If s1 has n characters and s2 has m characters, alter distance is computed using Longest Common Subsequence (LCS) algorithm.

LCS-LENGTH(X, Y)

```

1  m = X.length
2  n = Y.length
3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4  for i = 1 to m
5      c[i, 0] = 0
6  for j = 0 to n
7      c[0, j] = 0
8  for i = 1 to m
9      for j = 1 to n
10         if xi == yj
11             c[i, j] = c[i - 1, j - 1] + 1
12             b[i, j] = " $\nwarrow$ "
13         elseif c[i - 1, j] ≥ c[i, j - 1]
14             c[i, j] = c[i - 1, j]
15             b[i, j] = " $\uparrow$ "
16         else c[i, j] = c[i, j - 1]
17             b[i, j] = " $\leftarrow$ "
18 return c and b

```

Table 2:LCS Computation

	x →	m	e	e	t	i	n	g
↓	0	0	0	0	0	0	0	0
l	0	↑0	↑0	↑0	↑0	↑0	↑0	↑0
e	0	↑0	↖1	↖1	←1	←1	←1	←1
a	0	↑0	↑1	↑1	↑1	↑1	↑1	↑1
d	0	↑0	↑1	↑1	↑1	↑1	↑1	↑1
i	0	↑0	↑1	↑1	↑1	↖2	←2	←2
n	0	↑0	↑1	↑1	↑1	↑2	↖3	←3
g	0	↑0	↑1	↑1	↑1	↑2	↑3	↖4
	-	-	e	-	i	n	g	

Where X-Axis is feature in database & Y-Axis is New Extracted feature

Longest common subsequence between two sequences can be computed.

Alter distance = Length(sequence s2)- LCS(value).

6. CONCLUSION

Our proposed system shows that the morphed viruses having arbitrary percentages of dead code and subroutine insertions are still detectable within a certain error rate using alterdistance methods. Comparing API call a metamorphic virus can be detected.

7. REFERENCES

- [1] Mahim Patel, "Similarity Tests for Metamorphic Virus Detection," presented at San Jose State University, May 2011
- [2] W.Wong, "Analysis and Detection of Metamorphic Computer Viruses," presented at San Jose State University, May 2006.
- [3] Da Lin, "Hunting for Undetectable Metamorphic Viruses," presented at San Jose State University, December 2009.
- [4] P.Mishra, "A Taxonomy of Software Uniqueness Transformations," presented at San Jose State University, December 2003.
- [5] Evgenios Konstantinou, "Metamorphic Virus Analysis & Detection," presented at Royal Holloway University of London, January 2008.
- [6] J.Aycock, "Computer Viruses and Malware", Springer Science Business Media, 2006.
- [7] A Venkatesan, "Code Obfuscation and Metamorphic Virus Detection," presented at San Jose State University, December 2008.
- [8] Symantec, "Understanding Heuristics: Symantec's Bloodhound Technology".
- [9] "Understanding Computer Viruses".
- [10] Hossein Bidgoli, "Handbook of Information Security".
- [11] E. Daoud and I. Jebril, "Computer Virus Strategies and Detection Methods," Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008.
- [12] Jikku Kuriakose, "Metamorphic virus detection using feature selection technique", in IEEE ICCT, September 2014.
- [13] J. Borello and L. Me, "Code Obfuscation Techniques for Metamorphic Viruses", in Springer-Verlag France, February 2008.
- [14] Wikipedia, "Antivirus software", November 2010.
- [15] P. Szor, "The Art of Computer Virus Research and Defense," Addison-Wesley, 2005.

-
- [16] P. Szor, P. Ferrie, "Hunting for Metamorphic," *Symantec Security Response*, 2003.
 - [17] VX Heavens.<<http://vx.netlux.org/>>.
 - [18] Wikipedia, "Heuristic analysis", March 2009.
 - [19] Wikipedia, "Levenshtein Distance", March 2011.
 - [20] Cormen, Leiserson, Rivest, Stein. Introduction to algorithms (2ed, MIT, 2001).
 - [21] KevadiaKaushal, "Metamorphic Malware Detection Using Statistical Analysis", in *IJSCE*, July 2012.
 - [22] Wing Wong, "Analysis & Detection of Metamorphic Computer Viruses", presented at *San Jose State University*, May 2006.
 - [23] D.Bilar, "Statistical Structures: Fingerprinting Malware for Classification and Analysis", Proceedings of the Black Hat Convention, Las Vegas, 2006.
 - [24] Webopedia. "Internet".
 - [25] M. Stamp, "Information Security: Principles and Practice", August 2005.
 - [26] Orr, "The Molecular Virology of Lexotan32: Metamorphism Illustrated", July 2007.
 - [27] SrilathaAttaluri, "Detecting metamorphic virus using Profile Hidden Markov model", presented at *San Jose State University*, December, 2007.