Morphing Engines Classification by Code Histogram

Babak Bashari Rad[1], Maslin Masrom[2], [3]Suhaimi Ibrahim, [4]Zalina Mohd Daud
[1]Faculty of Computer Science and Information Technology, [2,4] Razak School of Engineering and Advanced Technology, [3]Advanced Informatics School
University Technology of Malaysia
Kuala Lumpur, Malaysia
[1]babak.basharirad@hotmail.com, [2]maslin@ic.utm.my, [3]suhaimiibrahim@utm.my, [4]zalina@ic.utm.my

***Abstract*— Morphing engines or mutation engines are exploited by metamorphic virus to change the code appearance in every new generation. The purpose of these engines is to escape from the signature-based scanner, which employs a unique string signature to detect the virus. Although the obfuscation techniques try to convert the binary sequence of the code, in some techniques, the statistical feature of the code binaries will be still remain unchanged, relatively. Accordingly, this feature can be utilized to classify the engine and detect the morphed virus code. In this article, we are going to introduce a new idea to classify the obfuscation engines based on their code statistical feature using the histogram comparison.**

Keywords-component:* *Computer Virus, Malware Morphing Engines, Obfuscation Engines, Mutation Engine, Metamorphic Virus, Code Histogram, Histogram Comparison

I. Introduction

The purpose of code obfuscation techniques is to make program codes more complicated to be comprehensible by a static analysis [1-2]. To achieve this purpose, the obfuscation engine transforms the program code to another dissimilar edition, but keeps the behavior of the different versions equivalent [3]. This skill can be used to protect the software from tampering or being visible for hackers, however it is widely being utilized by virus writers to make their virus armored against the antivirus experts [4]. Metamorphic viruses try to convert their code into new versions with dissimilar byte sequences, by means of obfuscation techniques. Therefore, traditional string signature-based scanners are not efficiently able to detect and classify the new instances.

In this study, we aim to propose a relatively novel idea to deal with metamorphic engine, which emphasizes on a statistical feature of the code, the histogram of the code bytes. Here, we are going to show that this new approach is applicable for classification of the obfuscation engines. This solution can be developed and improved in the future to be more reliable and effectual and be used in the antivirus scanners.

In next section, we review the recent attempts on this problem and most related methodologies and experiments. Then we present our proposed solution and explain its novelty. In the next part, we illustrate the methodology and implementation process, and in the final section, we give the summary and some recommendations for the future developments.

II. Previous Related Works

Several studies and experiments have been conducted on the metamorphic virus and obfuscation engines detection and classification problems. Wong and Stamp in [5] utilized the Hidden Markov Model for detection of metamorphic engines. They extracted assembly opcode sequence from manually disassembled assembly source files of a virus family collection and trained a HMM to present the virus family characteristics. Later, they used the trained model to classify the metamorphic virus. The drawback with their approach was the disassembling preprocess. It was a time-consuming, incompetent and impossible, especially when it deals with considerable quantity of virus files. Furthermore, they implement experiments to compare the similarity between some pairs of morphed variants of the obfuscation engines by employing the method developed by Mishra in [6]. His method compares two opcode sequences by considering all subsequences of three consecutive opcodes from each sequence.

Govindaraj in [7] also used Hidden Markov Model for the metamorphic virus detection problem, but he focused on a practical solution to deal with executable files directly. His method extracts the code section of the infected portable executable containing the virus code. Then it detects the 14 MFO (Most Frequently Occurred) instruction opcodes and builds a opcode sequence to train the HMM. He eliminated the manual preprocess disassembly phase completely.

Karnik et al. in [8] used the cosine similarity function to compare two suspicious infected files upon the static analysis of the portable executable files. They proved that for some given metamorphic variants of a virus, it is possible to classify them using the cosine similarity function.

Lin and Stamp in [9] show that it is possible to make the metamorphic viruses not detectable via Hidden Markov Model detectors, by morphing viruses such a way that they looks similar to benign programs. They showed that the HMM-based classifiers could not success when they inserted some sub-procedures copied from normal files, but inserting the copy of small code pieces had not produced the same result.

Rad et al. in [10] used the histogram of the machine instruction opcodes as a feature to present a virus variant. They utilized the Euclidean histogram distance metric to compare a pair of portable executable files. They showed that for some particular obfuscation techniques, their simple introduced approach could detect morphed variants of a file. In [11], Rad et al. used another histogram metric, Manhattan metric, and compare the result obtained by two Minkowski-form distances with r = 1 and r = 2. In both two latter ones, Rad et al. applied a preprocess phase to manually disassemble each portable executable file, break it down into its building function blocks and extract the instruction opcodes and convert them to a histogram as a feature of the file.

III. Obfuscation Engines

A. Morphing Techniques

Metamorphic viruses can obfuscate their body by several techniques. The most applicable techniques are [3, 10, 12-13]:

1) *Register or variable exchange* - using different registers or variables in code instructions for each new produced instance,

2) *instruction substitution* - replacing instructions with their possible equivalents instructions or code blocks or subroutines,

3) *Code transposition* - code and subroutines reordering or changing the flow control by conditional or unconditional jumps,

4) *Instruction permutation* - reordering instructions that are independent,

5) *Garbage code insertion* – inserting NOP instruction and other instruction that has no effects on the operation of the code among the original program instructions.

B. Metamorphic Virus Creation Kits

Virus creation kits, which are widely employed recent years, are serious matters of the antivirus vendors. These kits make the generation process very simple, such that no professional programming or computer knowledge is required to produce new viruses. Many virus construction tools are accessible and can be downloaded from internet [14]. Some of them employ stealth and anti-antivirus techniques to make the task of virus scanner harder. In addition, some kits are equipped with obfuscation engines to produce metamorphic virus variants.

One of the most famous and powerful tools is NGVCK(Next Generation Virus Creation Kit), which is introduced in 2001 [12]. It has an advanced mutation engine and can automatically produce high quality morphed variants of the virus. Wong and Stamp in [5] calculated the similarity between each pair of virus variants created by four virus creation tools, NGVCK, VCL32, MPCGEN, and G2, downloaded from the VxHeaven website [14]. Their findings demonstrate that the efficiency of these kits differs very much. Among the mentioned tested tools, the best generator kit, NGVCK, is capable to generate virus variants that contain only a small amount of similarity, the other tools creates variants that are very similar, more than 60 percent, on average. Randomly chosen benign programs have about 35 percent likeness, which show that, except the NGVCK, the other virus creation kits cannot efficiently mutate the code [5]. Therefore, based on their experiments, NGVCK obviously perform better than the other kits in terms of producing dissimilar virus variants, means that the virus variants generated by NGVCK are significantly dissimilar from their other family members, while the virus variants created by the other tools are more similar.

IV. Proposed Solution

The proposed solution is a development of the methodology presented by Rad et al. in [10]. It is based on the concept of using the statistical feature of the virus variants, but in new solution, we suggest some improvements and novel ideas:

A. Eliminate the Disassembly Process

The first development of the presented approach by Rad et al. in [10] is to remove the manual disassembly phase. We suggest implementing the algorithm directly on the binary executable files. With this idea, it would be more practical to be employed by antivirus software. In addition, manual preprocess section include of binary code disassembling, breaking down and feature extraction is very time-consuming. In new proposed method, it will be contain of feature extraction part, meaning that the algorithm extract the bytes of the virus and produce a histogram of the code, which is the representative of the virus.

B. Histogram as the Feature for a Virus Family

After feature extraction process for each member of a metamorphic family, we can make an average of the created histograms and produce a more general histogram as the representative of the virus family. It can be later used to classify whether a new virus belongs to a family or not. To achieve this purpose, we have to specify an appropriate threshold value that is able to classify a new virus and decide that it matches with which known virus family.

C. Code and Data Analysis

In [10], the authors experimented the algorithm on the code section of the virus file and the histograms are created only from the instructions opcodes. In this paper, we propose to make the histogram from the both instructions code and data bytes, including the instructions opcodes and operands. We believe that because all family member of a metamorphic engine use a same constant algorithm to generate the variants, an average of their histograms can represent the family statistical characteristics in terms of employing similar instructions. However, this can work properly if the representative histogram of a family constructed from enough number of the family members, otherwise it cannot be sufficiently trustable to classify the members correctly.

V. Methodology

A. Feature Extraction

Each metamorphic virus engine has a histogram as the representative of the family. It includes of the average statistical characteristic of the engine. To find such a histogram as the feature, we can extract the code statistics and present it as a histogram for each member of the family. Then we produce an average histogram to show the statistical characteristic of the engine. Fig. 1 demonstrates this process, briefly.

B. Classification

For each virus family, a threshold value must be selected such that be able to classify the family members correctly. It means that it must make the false negatives and false positives as less as possible. Distance between the histogram of the input file and representative histogram of the engine can be calculated via several distance metrics. The Minkowski-form distance measurement, with *r = 2,* or Euclidean distance is one of the most common metrics to measure the dissimilarity of two histograms [15].

$$d^2_{X,Y} = \sum_{i=1}^{n} (x_i - y_i)$$

Choosing the proper threshold value is the crucial issue in this classification problem. Fig. 2 shows the classification procedure.

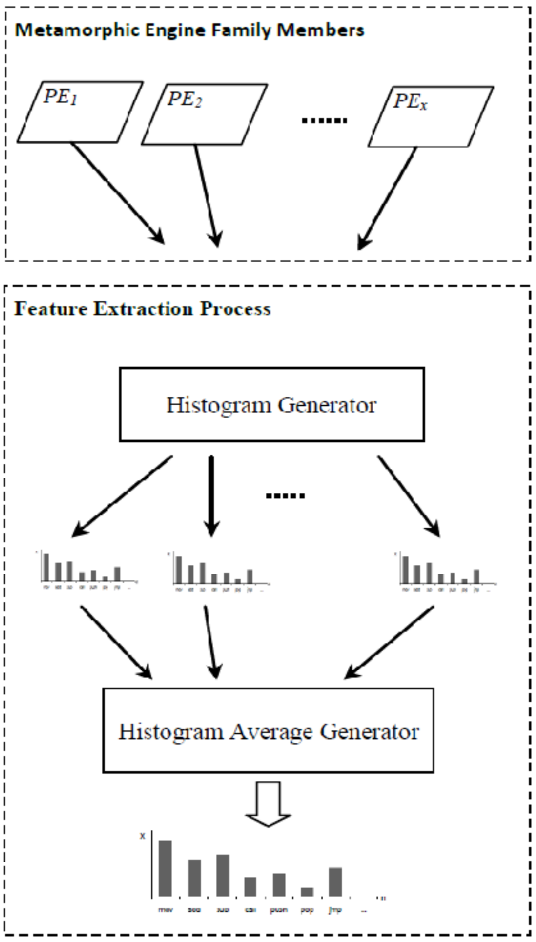
Metamorphic Engine Family Members
PE1
PE2
......
PEx
Feature Extraction Process
Histogram Generator
.....
Histogram Average Generator

Figure 1. Feature extraction process.

VI. Summary and Future Works

In this paper, we present a simple novel idea for classifying the metamorphic engines based on their code statistical property. This statistical feature is presented by a histogram of the code bytes for each obfuscation engine or virus family, distinctly. We believe for a large scale of data set include of virus family members, the representative histogram can reveal the statistical feature of the family. Then, histogram distance metrics can be used to compare the histogram of a suspected file with the histogram of a virus family to find whether it is a member of that family or not.

In this paper, we have not examined the proposed solution, so the efficiency of the improvements is not evaluated and compared to previous method. We plan to implement the proposed improvements and evaluate the efficiency of new ideas in the future researches.

The most advantage of this method is the low complexity of implementation. This is very simple approach to classify the morphing engines.

For the future works, this simple method can be developed for other statistical properties or it may be limited and filtered for some special byte values that are more specific for a particular virus family. In addition, the histogram comparison can be weighted to emphasize on certain family properties.

ACKNOWLEDGMENT

This research was supported by UTM Razak School Grant through the project 4B010.

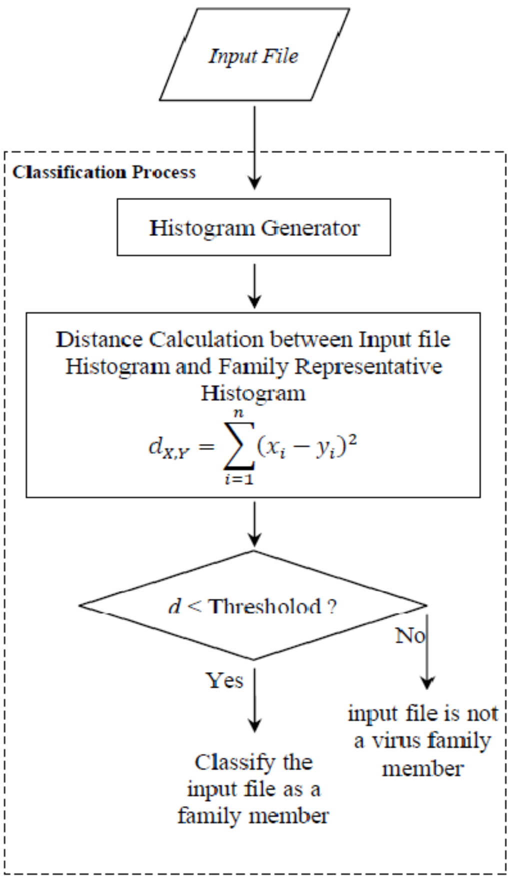
Input File
Classification Process
Histogram Generator
Distance Calculation between Input file Histogram and Family Representative Histogram
$d_{X,Y} = \sum_{i=1}^{n} (x_i - y_i)^2$
d < Thresholod ?
Yes
No
Classify the input file as a family member
input file is not a virus family member

Figure 2. Feature classification process.

REFERENCES

[1] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," Proc. Fifth International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA 2010), IEEE, 2010, pp. 297-300.

[2] K. Murad, et al., *Evading Virus Detection Using Code Obfuscation*, in *Future Generation Information Technology*, T.-h. Kim, et al., Editors. 2010, Springer Berlin / Heidelberg. p. 394-401.

[3] J.M. Borello and L. Me, "Code obfuscation techniques for metamorphic viruses," Journal in Computer Virology, vol. 4 (no. 3), 2008, pp. 211-220.

[4] B.B. Rad, M. Masrom, and S. Ibrahim, "Evolution of Computer Virus Concealment and Anti-Virus Techniques: A Short Survey," International Journal of Computer Science Issues (IJCSI), vol. 8 (no. 1), 2011, pp. 113-121.

[5] W. Wong and M. Stamp, "Hunting for metamorphic engines," Journal in Computer Virology, vol. 2 (no. 3), 2006, pp. 211-229, doi: 10.1007/s11416-006-0028-7.

[6] P. Mishra, *Taxonomy of Uniqueness Transformations*, in *The Faculty of the Department of Computer Science*. 2003, San Jose State University: San Jose, CA. p. 110.

[7] S. Govindaraj, *Practical Detection of Metamorphic Computer Viruses*, in *Faculty of the Department of Computer Science*. 2008, San Jose State University: San Jose, CA.

[8] A. Karnik, S. Goswami, and R. Guha, "Detecting obfuscated viruses using cosine similarity analysis," AMS 2007: First Asia International Conference on Modelling & Simulation Asia Modelling Symposium, Proceedings, 2007, pp. 165-170.

[9] D. Lin and M. Stamp, "Hunting for undetectable metamorphic viruses," Journal in Computer Virology, 2010, pp. 1-14, doi: 10.1007/s11416-010-0148-y.

[10] B.B. Rad and M. Masrom, "Metamorphic Virus Variants Classification Using Opcode Frequency Histogram," Proc. 14th WSEAS International Conference on COMPUTERS, WSEAS Press, 2010, pp. 147-155.

[11] B.B. Rad and M. Masrom, "Metamorphic Virus Detection in Portable Executables Using Opcodes Statistical Feature," Proc. International Conference on Advanced Science, Engineering and Information Technology 2011 (ICASEIT 2011), 2011, pp. 403-408.

[12] P. Szor, The Art of Computer Virus Research and Defense. Addison-Wesley Professional, 2005.

[13] D. Bruschi, L. Martignoni, and M. Monga, "Code normalization for self-mutating malware," IEEE Security & Privacy, vol. 5, 2007, pp. 46-54.

[14] VXHeavens. *VX Heavens - Computer Virus Information, Library, Collection, and Sources.* 2009; Available from: http://vx.netlux.org/vl.php.

[15] G. Shakhnarovich, T. Darrell, and P. Indyk, Nearest-Neighbor Methods in Learning and Vision: Theory and Practice (Neural Information Processing). The MIT Press, 2006.