Design and Implementation of a Dynamic Metamorphic Engine

Mehran khosravi
Iran, Azad University of Broujerd
Email: mehran_khosravy@yahoo.com

Abstract

It is shown that the order of appearance of op-codes in malicious codes is different from normal codes. Based on this observation malicious codes are obfuscated by re-ordering their op-codes and making them close to benign code. Similarity test techniques can be applied to measure the similarity of suspicious code with its de-obfuscated or morphed copies. It is observed that the similarity test will decline if the percentage of the similarity of the suspicious code with benign codes is more than 20%. In this paper a new obfuscation algorithm to increase the similarity measure to above 20% is presented. The algorithm applies a supervised learning technique to determine the obfuscation method and to select the suitable benign code for assimilation. The effectiveness of our algorithm is demonstrated by applying two well known classes of antivirus, advanced and advanced+ [13], to detect the original and our obfuscated versions of four famous viruses known as Screen factor, Barfa,Waber and Aphaty.

Key Words:
Metamorphic malware, Malware signature, Statistical analysis code, Metamorphosis engine, Obfuscation, Cellular learning automata.

---

1. Introduction

Today, metamorphic malware are one of the most challenging issues in the context of computer security. Most of current antivirus detector engines are generally based on syntactic features of malware. They usually scan sequences of binaries in machine code level of files to find the signature of viral code. Syntactic characteristics make the antivirus scanner vulnerable against the smart malware writers who are increasingly employing mutation techniques that change the byte strings of malware syntactically, while the function and behavior of the malware will remain significantly unchanged. Metamorphic malware writers use different innovative morphing techniques to evade the signature-based analysis. Some common obfuscation skills used in this kind of viruses are 1) garbage code insertion, 2) instruction permutation, 3) make subroutine from code and call it 4) and Code Transposition.

Metamorphism is the ability of malware to transform its code. This ability was first introduced in viruses and was later used by worms, Trojans, and other malware. For the time being there are several metamorphic engines that implement only the logic for transforming code that can simply be linked to any program to make it metamorphic.

1.1 Evolution of Malware

1.1.1 Encrypted Malware

The simplest way to change the appearance of a malware is to use encryption. As shown in (Fig. 1). An encrypted malware consists of a small decrypting module (a decryptor) and an encrypted malware body. If a different encryption key is used for each infection, the encrypted malware body will look different[1].

Fig 1: Anatomy of a Encrypted malware[1]

1.1.2 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 constantly changes. This is done by filename changes, compression, encrypting with variable keys etc [2,3]. The resulting variant has the same functionality as the parent malware. As shown in (Fig. 2), the main body of a polymorphic malware consists of a malicious code and its related Encryption-Decryption function code.

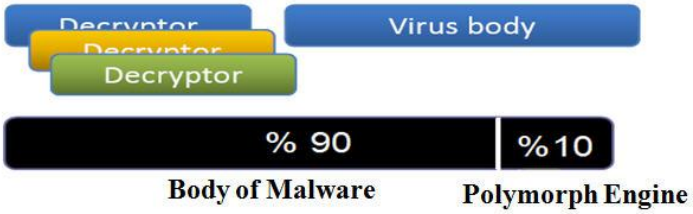

Fig 2: Anatomy of a polymorphic malware [1]

Polymorphic malware produces different variants of itself while keeping the inherent functionality as same. The concept of polymorphic code is core to polymorphic malware. It is a style of code that mutates while keeping the original algorithm the same [1].

1.1.3 Metamorphic Malware

Metamorphic malware represents the next class of virus that can create an entirely new variant after reproduction. Since the metamorphic malwares have do not produce variants having same body, they easily evade signature based detection. Since, most current antivirus software primarily use signature based detection, metamorphic malware currently are greatest threat [1].

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 (Fig. 3).

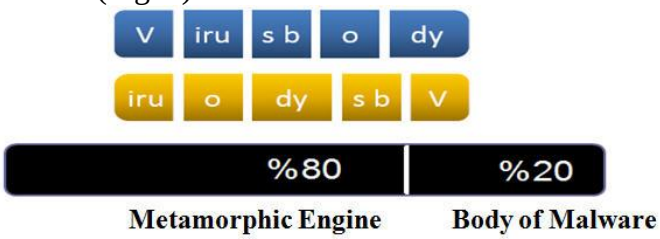

Fig 3: Anatomy of a metamorphic malware [1]

1.2 Detection Malware

As computer malwares evolve and become more complex, antivirus software must become more sophisticated to defend against malware attacks. This section discusses the malware detection techniques that have been deployed over the years.

1.2.1 Signature based detection

Signature based detection is one of the most popular and effective way of detecting virus. A signature as the name suggests is identification. It is like a person's fingerprint. It is used to identify a single virus or a family of virus. Signature based detectors need to update its

list of signatures frequently. If a new virus is found, then the signatures present in the database would be of no use since the signature of the new virus is not available in the database. Although antivirus detectors do not completely rely on signatures, they also look into heuristics sometimes.
However, signatures form a formidable percentage in the detection process. Signature detection is fast and simple. Signatures detection is not effective against some polymorphic and most metamorphic viruses [1,4].

1.2.2 Heuristics Based Detection

Recent virus detectors use signature detection along with heuristics. This help in reducing the number of false alarms. Heuristics is partially dependent on the behavior of the target malware. Each virus is born with a purpose. Not all viruses have the same purpose. The technique, methodology and the kinds of attacks used by each virus is unique of its kind. Heuristics sets certain base rules. These rules are mostly essential for proper functioning of the system and maintain its stability along with data integrity. Since a virus aims at tarnishing one or many of these, heuristics help in detecting and preventing new viruses to some extent.
Although Heuristics is promising, it cannot stand for itself. The number of false alarms raised is high[5]. Heuristics is good when combined with another reliable detection technique. Heuristics can be used to support or augment results of other detection techniques. A virus coder can still write a virus that does not obey the rules in a heuristics and still cause considerable damage to the system.

1.2.3 Hidden Markov Model based detection

In recent years, detection of metamorphic malware has been very much effective due to the application of Markov models to malware detection. Profile Hidden Markov Models (PHMM) is known for their success in determining relations between DNA and protein sequences. When applied for malware analysis it has been found that PHMM can effectively detect metamorphic malware. Although PHMM can detect malwares which are metamorphic, they still need a test data in order to train them. [1] Also, the process of filtering the data, disassembling them, training and scoring the whole dataset can be time consuming [5].

1.2.4 Similarity Analysis

Another method to detect malwares is through similarity analysis. A program is represented as some number of functions f, and each function contains some number of statements which are termed as vectors x and y. The total number of vectors for the same program P and for all functions f is kept same. Similarity analysis can be performed by using cosine similarity measure which is primarily used in text mining. In short, the maliciousness of a code is estimated [6].
The following steps are involved in typical similarity analysis based detection. The program executable is decompressed and disassembled. Each disassembled program represents a vector of functions. Each function is represented as an array of vector of functions. The similarity between the functions of a program P and P' is computed using cosine similarity measure or other methods. The value is them compared with the threshold value to determine if the given executable is malicious or not [6].

2. Related Work

This part of article is a review on recent work about metamorphic engine.

First one is an engine that proposed by Priti Desai from San Jose state university (2008) [7].This engine is inspired by malware Evol. Evol malware use inserting ineffective code; replace the name of registers and exchange uses the same commands for obfuscation itself. This metamorphic engine is also trying to generate malware with similarity to benign program. But this engine have some disadvantage: malware that produced by this engine

can be detected by HMM techniques and similarity test technique; obfuscation techniques embedded in this engine is too limited; structure of this engine is not dynamic; Some exchange commands embedded in this engine contains the logic and syntax errors that can impact the performance of malware mutations in addition binary volume of malware has increased from generation to generation [7].

Second is an engine base on advance obfuscation technique proposed by Jean-Marie Borello and other (2010) [8]. This engine use of advance permutation technique for obfuscation process. Main target of this engine is overcome to signature base detection technique. In this engine body of malware randomly separated in some block then with unconditional jump makes new control flow for code. This work was performed with a high rate and the resulting control flow graph is very vague. But this engine has some disadvantage: Unconditioned or conditioned jump commands with high rates and without managing inject into malware code that can identify areas for detection techniques; numbers of obfuscation techniques are so limited; malware that produced by this engine can be detected by HMM techniques and similarity test technique in addition there are no effort to generate malware with similarity to benign code [8].

Number three is an engine proposed by Da Lin from San Jose state university (2009) [9].This engine have five goals: 1- Generate morphed copies of a single input virus. These morphed copies should have a similarity of approximately 30% (match scores of benign code) with the base virus and among themselves.2- The morphed copies should have the same functionality as the base virus.3- The morphed copies should be "close" to the benign code. The assumption here is that the normal programs are the cygwin utility files of the same size as the base virus.4- The metamorphic engine should work on any assembly program. 5- Generated viral copies should be capable of escaping HMM detection.
For this goal they use of several obfuscation technique with an algorithm. To make a virus similar to a normal file, they developed an algorithm to calculate the score of similarity between the two files. The first one is a virus, and the second one is a normal file. But this engine has some disadvantage: This engine based on a specific type of malware and is no guarantee that there is being able to work on other malware; the garbage code that be injected to body of malware haven't any obfuscation techniques so normalization techniques can be detected and removed them in addition similarity scoring techniques delivered is locally and the only score of the modified section is calculated [9].

3. Proposed Engine

Our target is produce new metamorphic engine with ability to create new generation of basic malware code with high similarity to benign code and dynamic nature in process. For this target we use of cellular learning automata to be a regulatory factor in obfuscation process of new generation. Input of cellular learning automata are viral code and benign code and output is new generation of viral code with high similarity to input benign code. We use of cellular learning automata with global variable. These kind of cellular learning automata use global feedback of environment in learning process.
For every cell in cellular automata we use learning automata with variable structure, these kind of learning automata have 4 parameter $\{\alpha,\beta,P,T\}$:
α= actions of automata, β= the automata inputs, P= Probability vector of each of the actions, T= learning algorithms. Learning algorithm is liner:

- Favorable response

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)] \quad (1)$$
$$p_j(n+1) = (1-a)\,p_j(n) \qquad \forall j \quad j \neq i$$

- Poor response

$$p_i(n+1) = (1-b)\,p_i(n) \quad (2)$$
$$p_j(n+1) = (b/r-1) + (1-b)\,p_j(n) \quad \forall j \quad j \neq i$$

In relations (1) and (2), a and b are the rewards and penalties parameters.

We have four members of the learning automata with variable structure is organized. Actions of automata (α) equal to the set of different obfuscation techniques which are code injection unconnected, code injection connected, permutations in the blocks of code, creating a subroutine of the code and call it and changes in flow control are performed. Inputs of automata (β) corresponding to the selected block of code based malware have in mind. Select any of the actions is the vector (P) equal to P = {0.18, 0.18, 0.24, 0.22, 0.18} have in mind. And learning algorithms have the same mode as defined above in the comment.
To prevent normalization operations construction on the new generation of malware produced by techniques we use of inject cryptography function, which is similar to normal programs and also due to the complexity of the normalization operation.
Cellular learning automata inputs must be determined. For this purpose, the input is basic malware that have been pouring into the complex function, in the form of 10 blocks. We choice a random benign code and divided in 10 blocks. We must remove comment and space from booth code. Every two block (a viral block and a benign block) are input for a cell in cellular learning automata. Structure of proposed cellular learning automata showing in (Fig. 4).

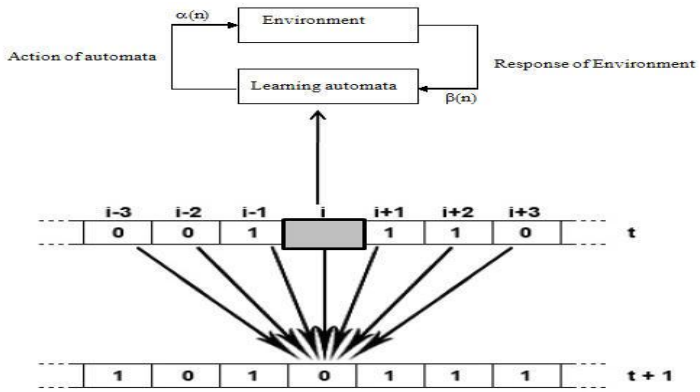

Fig 4: Structure of proposed cellular learning automata

3.1 Learning Process

In each step random number of block choice to interact with the environment and learning process. For each selected block we must look the set of parameter P and choice function with higher probability to interactive with environment. We apply the obfuscation function with this probability to selected block. So new block will be generating. For determine reward of selected action and update probability set of function we use basic similarity test for selected state in previous state and new state [9]. In this condition we have to state:

First- the similarity between two states be increase in new situation. We must update probability set of action in learning automata,we increase probability of function that make this change and other functions will be followed in order to maintain a probability of 1.

Second- the similarity between two states be decrease in new situation. We must update probability set of action in learning automata.We decrees probability of function that make this change and other functions will be followed in order to maintain a probability of 1.

This process is for local learning of automata now for global learning we must doing new similarity test between all block of pervious step and new step. In this condition we have to state:

First- full similarity between two states be increases in new step. Now we must do two works. A: update probability set of function in learning automata same local learning's, B: replace new generated block with previous block and start new learning step with new condition.

Second- full similarity between two states be decrees in new step. Now we must do two works. A: update probability set of function in learning automata same local learning's, B: start new learning step with new probability set and previous block.

But how long to continue the learning process? This question can be specified with three modes:

- The first case occurs when in learning process we have limited resources. In this mode after a certain number of learning process and ending resources, training stops and the last status block consider as outputs, and consider the possibilities for functions.

- The second case occurs when basic malware code essentially hasn't the ability and talent to be same as benign program .For this condition we define maximum number for learning process and the last status block was consider them as output.

- The third mode and the ideal state is complete learning process. Articles and resources on the most discussed that similarity of the benign code in minim nearly 14 percent, the maximum close to 93 percent and 35 percent are listed in moderate condition. Our goal is the same, building code with high similarity to benign code. Therefore, we ending learning process in step which new generation of code has at least 35 percent increase in similarity with benign code.

Proposed algorithm pseudo code showing in (Fig. 5).

4. Results And Analysis

An efficient mutation engine must have the following characteristics:

- Generate mutated malware with little similarity to the base malware and to each other.
- Generate mutated malware with high similarity to benign code.
- Support all kind of assembly code.
- Malware binary volume should not be greatly increased in the mutated malware.

- Obfuscations techniques that applied to the mutated malware code, should not be easily detectable.
- Function of the mutated malware does not change.

To evaluate the performance of the proposed transformation engine valid online tools available to detect malware in [13] has been used.
For testing, the assembly code for a number of known malware in [11,12] received and are used as the malware basis for evaluating the metamorphic engine. For benign code we use of binary executable files from Cygwin [11], they have low level operations such as the malware.

```
Algorithm CLAEn

Input    : Base Malware
Output: New Generation of Malware

Begin

Choice a Random Benign Code and divided in 10 Block
Divided Base Malware in 10 Block

Initialize CLA with 10 LA each with 5 actions

While (Number of Iteration is down of 100) Do

  For each Line in CLA Do in Parallel
   Each LA choice one of its action according to its action probability vector
       IF (New Similarity in Block > Old Similarity in Block)
          Old Block       =   New Block
          Old Similarity =   New Similarity
          Update Action Vector
       Else
          Update Action Vector
   IF (Full New Similarity > Full Old Similarity)
     Update all Block With New Block in CLA
     Run new Iteration with New Block and New Action Vector
  Else
   Replace all New Block with Old Block
   Run new Iteration with Old Block and New Action Vector

End For
End While
End
```

Fig 5: pseudo-code algorithm proposed

4.1 Commercial Malware Scanner

To test the proposed metamorphic engine power ; in overcoming the anti-malware products, 32 assembly code Malware , mutated base in the form of four 32-bit executable files to compile in a virtual machine environment and 13 famous anti-malware products with ranking award “Advance+” and “Advance” according to antivirus comparative march 2012 [13]. Results in (Table 1) are presented. Results shown in this table represent the engine performance in dealing with the techniques proposed existing anti-malware products.

Antivirus		Win32.ScrenF		Win32.Waber		Win32.Brof		Win32.Apathy	
		Original	Muted	Original	Muted	Original	Muted	Original	Muted
1	G Data	✓	✓	✓	×	✓	×	✓	×
2	Avira	✓	×	✓	✓	✓	×	✓	×
3	Kaspersky	✓	×	✓	×	✓	×	✓	×

4	Sophos	✓	×	✓	×	✓	×	✓	✓
5	F-Secure	✓	✓	✓	×	✓	×	✓	×
6	Panda	✓	×	✓	×	✓	✓	✓	×
7	Bitdefender	✓	×	✓	×	✓	×	✓	×
8	BullGuard	✓	×	✓	×	✓	×	✓	×
9	eScan	✓	×	✓	×	✓	×	✓	×
10	Avast	✓	×	✓	×	✓	×	✓	✓
11	ESET	✓	×	✓	✓	✓	×	✓	×
12	McAfee	✓	×	✓	×	✓	×	✓	×
13	Fortinet	✓	×	✓	×	✓	✓	✓	×

Table 1- Results of tests carried out against anti-malware products

5. Conclusions

In this article we proposed new metamorphic engine base on cellular learning automata, with ability to produce new generation of basic malware with special properties. New generation have low similarity to each other and high similarity to benign program. Dynamic nature of proposed engine is a special properties that in other same engine not be seeing until now. We inject complex function such as cryptographic function for prevent operation of normalization function. This property makes a power full ability in front of new technique of detection malware. Our new generations have high similarity to benign code so detection techniques base on similarity of two codes such as basic similarity test and HMM con not detect this new malware.

References

[1]. P. Szor, “The Art of Computer Virus Research and Defense”, Addison Wesley Professional, 2005.

[2]. You, I., Yim, K., “Malware Obfuscation Techniques: A Brief Survey”, Proceedings of International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010.

[3]. Schiffman, M., “A Brief History of Malware Obfuscation”, Cisco, security, http://blogs.cisco.com/security, Feb 2010.

[4]. Govindaraju A., “Exhaustive Statistical Analysis for Detection of Metamorphic Malware”, Master’s thesis, Jose State University, 2010.

[5]. Wong, W., Stamp, M., “Hunting for Metamorphic Engines”, Journal in Computer Virology(Springer), vol. 2, no. 3, pp. 211-229, 2006.

[6]. P. Mishra, “A taxonomy of software uniqueness transformations”, master’s thesis, San Jose State University, Dec. 2003.

[7]. Desai, P., “A highly metamorphic virus generator”, Int. J. Multimedia Intelligence and Security (JIMIS), Vol.1, No.4, p402-427, 2010.

[8]. Borello, J., Filiol, E., Mé, L., “From the design of a generic metamorphic engine to a black-box classification of antivirus detection techniques”, Journal in Computer Virology (Springer), p278-287, 2010.

[9]. Lin, D., Stamp, M., “Hunting for undetectable metamorphic viruses”, Journal in Computer Virology (Springer), 2011.

[10]. Borna Jafarpour, “Cellular Learning Automata, Machine Learning Report”, Amirkabir University of technology Computer Engineering and IT department, Spring 2006.

[11]. “Malware Data Base”, http://borax.poluxhosting.com/madchat/vxdevl/vxsrc.

[12]. “Malware Data Base”, http://www.vx.netlux.org.

[13]. Antivirus Comparatives, “On-demand Detection of Malicious Software”, March 2012, www.av-comparatives.org.

[14]. Gao, X., Stamp, M., “Metamorphic Software for Buffer Overflow Mitigation”, Master’s thesis, San Jose State University, 2008.

[15]. Chelangat Milgo, E., “Statistical Tools for Linking Engine- generated Malware to its Engine”, Master’s Thesis, Columbus State University, 2009.

[16] . Syaed Hamze,Javad ".A Metamorphic engine with ability to same with benign code ."MS thesis،Iran ،Shabestar ، 1390.

[17]. Mohamdreza,Maybodi ،Hamid,Baygi & Masod,Taherkhani " ،Cellular learning automata" ،Six Computer Conferences ،Page 163-153 ،1379.