Malware Obfuscation Measuring via Evolutionary Similarity

Jian Li * Jun Xu # Ming Xu * HengLi Zhao * Ning Zheng *

* Institute of Computer Application Technology, Hangzhou Dianzi University, P. R. China.

# The Third Research Institute of The Ministry of Public Security, P. R. China.

lijiandm@163.com xujun@stars.org.cn huxhc2008@163.com {mxu,nzheng}@hdu.edu.cn

Abstract— **With prevailing of the malware, it is necessary to measure the malware obfuscation. We traced the system calls as the dynamic action of malware, and used evolutionary similarity to measure obfuscation. An algorithm, which uses sequence alignment as a way of arranging the sequences to identify similar regions, has been proposed to calculate the similarity. We used real-world malwares to test the resilience of our method. Our experiment has shown that our method has strong resilience against common obfuscation technologies.**

Keywords-malware; obfuscation; similarity; evolutionary similarity

I. INTRODUCTION

A malware is a program that has malicious intend. It includes viruses, Trojans, worms, backdoors and so on. Some obfuscation technologies are used by malware authors to evade signature detection from anti-malware software. Polymorphism and metamorphism are two common obfuscation technologies. A polymorphic malware obfuscates its decryption loops using several transformations, such as Dead-code Insertion, Code Transposition, Register Reassignment and Instruction Substitution, while metamorphic malware uses these transformations to obfuscate the entire program. Anti-malware software which uses pattern-matching approaches is vulnerable to these obfuscation technologies [1]. In order to improve the resilience of anti-malware software, it is necessary to measuring the obfuscation similarity in malwares.

Besides, due to the voluntary sharing of ideas and code in malware production community, there are a small numbers of authors who produce completely new malicious programs. Rather, most of the different malicious programs are modifications of some previous one [2]. In a given malware family, malware always evolves with a bit change by borrowing or copying code between families. So measuring the similarity between unknown mal-sample and given malware can improve the efficiency and effectiveness of malware analysis. And it is expected to help reconstructing malware phylogenies by using similarity techniques [3].

In this paper, we use evolutionary similarity to measure the obfuscation similarity between malware variants. In bioinformatics, evolutionary similarity is a measure of evolutionary divergence between two homologous DNA, RNA, or protein sequences. We analyzed two malwares' runtime sequence of system calls, which is the only way for a program to interact with the operating system and a natural place to intercept a program and perform monitoring, since system calls often require a context switch [4]. An algorithm was proposed to measure the evolutionary similarity between the two sequences of system calls. Our experimental results demonstrated our method is resilient to common obfuscation technologies.

II. RELATE WORK

Obfuscation is defined as program transformation. In [5], authors discussed some obfuscation technology from the view of malware detection. [1] present an architecture for detect malicious patterns in executables that is resilient to common obfuscation. Some semantic based research such as [6] created semantics templates to detect malicious traits. These templates were created based on instruction, variable and symbolic constants. A robust signature-based malware obfuscation detection technique is introduced in [7]. It was based on the hypothesis that all versions of the same malware share a common core signature which is a combination of several features of the code. The major difference between these researches and our method is that we use dynamic action as the description of mal-program while the others use static information.

Besides, Obfuscation techniques are not only relevant to detecting morphed code but also several issues. [8] present the relationship between obfuscation technology and protection of intellectual property rights present in proprietary software. [2] provided a brief introduction to the issue of measuring similarity between malicious programs, and how software evolution history refactoring is known to occur in the area. [4] used system calls, which is a dynamic action, to detect sophisticated mimicry attacks in intrusion detection system. In bioinformatics, some technologies have been used to detect evolutionary divergence between two homologous DNA, RNA, or protein sequences [9]. The method was based on aligning long sequences which can uniquely represent the organism. These issues have some common characters to malware obfuscation detection. In this paper, we use evolutionary similarity to measure the malware obfuscation.

III. Malware Obfuscation Measuring

A. *Describe malware behavior using system calls*

Due to some software protection technologies used by malware authors, measuring program behavior from the static information must meet some difficulties. First, encryption and pack technologies have been widely used to prevent the mal-program to be dissembled for malware analysis. Second, signature matching approach, which is used to detect malware, is vulnerable to obfuscation technology. And then, it is difficult to exactly reconstructing the semantic based malware behavior from static information.

We use the runtime sequence of system calls to represent the behavior of a malware. Using dynamic information of malware process can bypass the obstacle caused by encryption and pack technology, since program will decrypt and unpack itself when it executes in operating system. As the way for a program to interact with the operating system, system calls represent important events occurred when process executes. Besides, since system calls have higher system abstractness than instructions, some obfuscation transformations based on instruction level, which includes dead-code insertion, register reassignment and instruction substitution, have litter effect on malware's runtime behavior. So system calls are more resilient to code obfuscation technologies. Our experiment results, which show the resiliency of our method, will be described in section 4.

The number of the system calls is fixed for a special operating system version. So the behavior of mal-process can be expressed as a sequence of a finite set of states. For instance, Windows XP SP2 has 284 system calls whose names begin with "Nt". The system call number covers from 0 to 283. We trace the system calls at runtime, and describe the behavior of mal-process as a sequence of system call number. For example, figure 1 shows two sequences of system calls. Each of them represents the runtime behavior of a process.

A: 119, 280, 173, 17, 113, 127, 25

B: 119, 280, 17, 113, 127, 170, 25

Figure 1.

In our experiments, we use Ether [10] as our system call tracer. Ether traces system calls via hardware virtualization, it remains transparent to target process and can keep a high tracing accuracy.

B. *Similarity measuring algorithm description*

We use evolutionary similarity to measure the obfuscation similarity between malware variants. In bioinformatics, evolutionary similarity is a measure of evolutionary divergence between two homologous DNA, RNA, or protein sequence. It always use sequence alignment as a way of arranging the sequences to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between them. We use a similar way to align two sequences of system calls via inserting gaps between the system calls so that identical numbers are aligned in successive columns. Figure 2 shows the alignment of two sequences:

A: 119, 280, 173, 17, 113, 127, -- , 25

B: 119, 280, -- , 17, 113, 127, 170, 25

Figure 2.

It is based on the hypothesis that in order to align identical numbers, the more gaps are inserted, the lower the similarity is. Because the sequences are getting longer but the number of identical numbers have not changed. In figure 2, the number of identical system call numbers is 6, which can be express as $N_{identical} = 6$. And the length after alignment is 8, which is expressed as $L_{alignment} = 8$. So the similarity can be calculated as:

$$Similarity(A,B) = \frac{N_{identical}}{L_{alignment}} = 0.75$$

We note that given a pairs of sequences, there may be different alignments. And different similarity will be measured for them. We propose an algorithm to measure the maximal similarity for any alignments. First we define a similar matrix S to specify the scores for aligned system call numbers. Here, $S[i,j]$ is the similarity for system call i and j. In Windows XP SP2, the system call number covers from 0 to 283.We define matrix S as: for $\forall 0 \le i \le 283, 0 \le j \le 283$: $S[i,j]=1$ when $i = j$; $S[i,j]=0$ when $i \ne j$.The algorithm is shown as follows:

Algorithm 1: Calculate evolutionary similarity between two sequences:

Input: similar matrix S, gap penalty d, sequence A and B.

Output : similarity between sequence A and B.

```
1 $La$=length of $A$, $Lb$=length of $B$.
2 Initial matrix $F[La+1, Lb+1]$ as the two-dimensional matrix to find the alignment.
3 Initial matrix $P[La, Lb]$ to recode the choice during the dynamic programming.
4 for $i = 0 \rightarrow La$ : $F[i,0] = d \times i$
5 for $j = 0 \rightarrow Lb$ : $F[0,j] = d \times j$
6 for $i = 1 \rightarrow La$:
7     for $j = 1 \rightarrow Lb$:
8         $choice1 = F[i-1, j-1] + S[A[i-1], B[i-1]]$
9         $choice2 = F[i-1, j] + d$
```

```
10        $choice3 = F[i, j-1] + d$
11        $F[i,j] = \max\{choice1, choice2, choice3\}$
12        The max choice is recorded into $P[i-1, j-1]$.
13 $l1 = La, l2 = Lb$
14 while $i > 0$ and $j > 0$:
15        if $P[i,j] == choice1$ : $i--, j--$
16        else if $P[i,j] == choice2$ : $i--, Lb++$
17        else if $P[i,j] == choice3$ : $j--, La++$
18 while $i > 0$: $i--, Lb++$
19 while $j > 0$: $j--, La++$
20 return : $Similarity(A,B) = \frac{F[l1,l2]}{La}$
```

Our algorithm is an example of dynamic programming. First, we'll do some initialization work in lines 1-5. And then, we recursively calculate $F[i,j]$ based on the principle of optimality. It is assigned to be the optimal score for the alignment of the first i characters in A and the first j characters in B. And the bottom right hand corner of the matrix is the maximal score for any alignments. During the traversing of matrix A, the optimality choices have been recorded in matrix P. This is shown in lines 6-12.

Due to our goal is getting the maximal similarity score between a pair of sequence, there is no necessary to get two aligned sequences. $P[i,j]$ records the choices during the dynamic programming. We traverse matrix P from the bottom right hand corner and only calculate the length of aligned sequence. This is shown in lines 13-19. At line 20, the maximal score is divided by aligned length for normalization, which is returned as the similarity.

IV. Evaluation and Experiments

In order to verify the effectiveness of our method, we evaluated our implementation of our algorithm against real-world malware variants. It is based on the hypothesis that a high evolutionary similarity score for two malwares will be calculated if they are a pair of variants, and a low score will be calculated if they are not a pair of variants. For instance, similarity score calculated for Backdoor.Win32.Bitcon.a and Backdoor.Win32.Bitcon.b should be high, and the score calculated between Trojan-PSW.Win32.Deathmin.g and Trojan-PSW.Win32.Dumbnod.c should be low. Two major factors we measured are the true positive rate and false positive rate:

True positive rate (TPR): The rate that a pairs of real-world variants were classified correctly.

False positive rate (FPR): The rate that a pair of non-variants was classified to a pairs of variants.

The goal of our experiment was achieving a high TPR and a low FPR. The method on measuring them will be discussed in section 4.2. All of the malware samples run in Windows XP SP2, and we traced the system calls by Ether [10].

A. Preprocessing

After we got the sequences from system call tracer, we must preprocess them before calculate similarity. Because program may have different states when they are in different environments, firstly we aggregated consecutive occurrences of the same system call number and removed 2-length duplicate sub-sequence. The aggregation and reduction phase was an experiment choice, which can improve the TPR. The duplicate sub-sequences which were longer than 2 will not be removed because removing them can not affect the experiment results obviously.

Second, 54 system calls at the beginning of each sequence should be removed. It was also an experiment choice. We observed every sequence has the same 54 system calls at the beginning. We believed that they were generated by programs initialization. Experiment results have shown that remove them would improve the TPR and reduce the FPR.

B. Classification method and experiments results

Our experiment consists of training phase and testing phase. We prepared 521 pairs of variants and 515 pairs of non-variants as our training set, and used other 187 pairs of variants and 172 pairs of non-variants as our testing set.

TABLE I. THE THRESHOLD USING DIFFERENT GAP PENALTY

d	E1	E2	threshold
0	0.86	0.41	0.64
0.1	0.86	0.44	0.65
0.2	0.88	0.5	0.69
0.3	0.89	0.56	0.73
0.4	0.91	0.62	0.77
0.5	0.6	0.48	0.54
0.6	0.58	0.58	0.58
0.7	0.68	0.68	0.68
0.8	0.78	0.78	0.78
0.9	0.89	0.89	0.89
1	1	1	1

The goal of training phase was calculating the threshold which is used to identify the variants and non-variants. First, we computed the similarity of 521 pairs of variants by algorithm 1, and calculated the average similarity $E_1 = \frac{1}{n_1}\sum_{i=0}^{n_1} similarity_i$, $n_1 = 521$. Second, we calculated the similarity of non-variants. The average similarity of non-variants was calculated as $E_2 = \frac{1}{n_2}\sum_{i=0}^{n_2} similarity_i$, $n_2 = 515$. So the threshold can be expressed as: $threshold = \frac{E_1 + E_2}{2}$. We denoted the gap penalty d was the only variable in our algorithm, we compared the threshold with different d . We have shown the threshold in table 1.

TPR and FPR were measured in testing phase. First, the similarity of 187 pairs of variants and 172 pairs of non-variants was calculated by algorithm 1. So classify accuracy can be expressed as $TPR = \frac{n_3}{n_4}$, n_3 is the number of variants pairs whose similarity is above the threshold, and $n_4 = 187$. False positive rate can be expressed as $FPR = \frac{n_5}{n_6}$, n_5 is the number of non-variants pairs whose similarity is above the threshold, and $n_6 = 172$. The classify accuracy and false positive rate with different gap penalty d have been shown in figure 3:

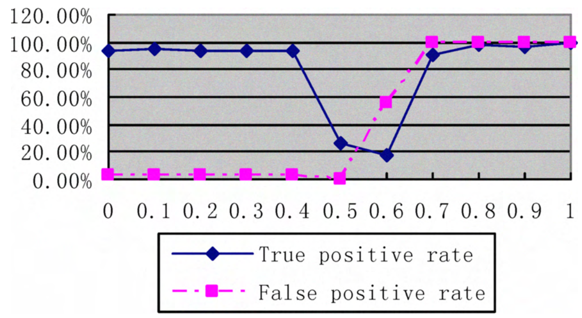

Figure 3: True positive rate and false positive rate

In figure 3, X-axis denotes different gap penalty d , and Y-axis denotes the classify accuracy and false positive rate. We can observe that bad performance was reached when the value of d is equal or higher than 0.5. We observed the reason is that the alignment of sequences is disorderly when d reached 0.5. High TPR and low FPR can be achieved when the value of d is below 0.5. And the best performance was achieved when $d = 0.1$, it got the TPR of 94.65% and the FPR of 3.49%. So resilience of our method can be demonstrated.

V. Conclusion and Future Work

In this paper, we used evolutionary similarity to measure malware obfuscation. An algorithm has been proposed to calculate the similarity. We used real-world malwares to test the resilience of our method. The resilience was weighed by true positive rate and false positive rate. Our experiment has shown that our method has strong resilience against real-world malware obfuscation.

The problem has been cognizant that the dynamical action of program is not unique, which depends on the runtime environment and the input value. In another aspect, sequence of system calls cannot completely depict the program dynamical action. So how to reduce the uncertainty of dynamical action description and use more complicated and effective module to depict the program action is our ongoing work.

Acknowledgment

This work is supported by the Natural Science Foundation of Zhejiang Province (No. Y1090114), and the Science and Technology Program of Zhejiang Province (No: 2008C21075).

References

[1] M.Christodorescu and S.Jha. Static Analysis of Executables to Detect Malicious Patterns. In 12th USENIX Security Symposium, 2003, pp. 169–186.

[2] A.Walenstein and A.Lakhotia. The Software Similarity Problem in Malware Analysis. In Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, Dagstuhl, Germany, July 2006.

[3] E.Karim, A.Walenstein and A.Lakhotia, Malware phylogeny using maximal π -patterns. In proceeding of the EICAR 2005 Conference, 2005. pp. 167–174.

[4] D.Gao, M.Reiter and D.Song. Behavioral Distance for Intrusion Detection. In 8th International Symposium on Recent Advances in Intrusion Detection. September 2005. pp. 63-81.

[5] C.Collberg, C.Thomborson and D.Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer science, University of Auckland, New Zealand, 1997.

[6] M.Christodorescu, S.Jha, S.A.Seshia, D.Song, and R.E. Bryant. Semantics-aware malware detection. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 2005: 32-46.

[7] Andrew H. Sung, Jianyun Xu, Patrick Chavez, and Srinivas Mukkamala. Static analyzer of vicious executables (save). In 20th Annual Computer Security Applications Conference, Tucson, AZ, USA, 2004, pp. 326-334,2004.

[8] C. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation – tools for software protection. Technical Report 170, Department of Computer Science, University of Auckland, 2000.

[9] M.Brudno, S.Malde, A.Poliakov, C.B.Do, O.Couronne, I.Dubchak and S.Batzoglou. Glocal Alignment:Finding Rearrangements During Alignment. Bioinformatics, 2003, 19(90001), pp. 54i-62.

[10] A.Dinaburg and P.Royal, Ether: Malware Analysis via Hardware Virtualization Extensions. In 15th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA, 2008.