2009 Third International Symposium on Intelligent Information Technology Application

Malware Obfuscation Detection via Maximal Patterns

Jian Li Ming Xu Ning Zheng Jian Xu
Institute of Computer Application Technology, Hangzhou Dianzi University, P. R. China
lijiandm@163.com mxu@hdu.edu.cn nzheng@hdu.edu.cn jian.xu@263.net

***Abstract*—Malware obfuscation is defined as a program transformation. It is always used in malware to evade detection from anti-malware software. In this paper, we propose a method to detect malware obfuscation using maximal patterns. Maximal pattern is a subsequence in malware's runtime system call sequence, which frequently appears in program execution, and can be used to describe the program specific behavior. The maximal pattern sequence is extracted from the malware's runtime system calls, and the similarity between two pattern sequences will be measured by evolutionary similarity. Based on the real-world malwares test data, the experiment results have shown that our method can efficiently detect malware obfuscation.**

Keywords-malware; obfuscation; maximal pattern; evolutionary similarity

I. INTRODUCTION

A malware is a program that has malicious intend. It includes viruses, Trojans, worms, backdoors and so on. Obfuscation is defined as program transformation, which includes metamorphism and polymorphism. A metamorphic malware obfuscates the entire program using several transformations, such as Dead-code Insertion, Register Reassignment, Instruction Substitution and Code Transposition, while polymorphic malware only uses these transformations to obfuscate its decryption loops [1]. They are used by malware authors for a long time. And anti-malware software which uses simple signature matching approaches is vulnerable to them. In order to improve the resilience of anti-malware software, it is necessary to detect the obfuscation 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 different malicious programs are modifications of some previous one [3]. In a given malware family, malware always evolves with a bit change by borrowing or copying code from previous edition. So measuring the similarity between unknown mal-sample and existing malware can improve the efficiency and effectiveness of malware analysis. And it is expected to help reconstructing malware phylogenies by using similarity techniques.

In this paper, we use maximal patterns to detect the obfuscation. A maximal pattern represents a specific subsequence in malware's runtime system calls sequence. It has the ability to describe the program specific behavior, and can be used to detect obfuscation accurately. We extract maximal pattern sequence from the sequence of system calls, and measure the similarity between two maximal pattern sequences by evolutionary similarity. Our experiments have shown that our method can detect the real-world malware obfuscation with high accuracy and strong resilience.

II. RELATE WORK

Obfuscations have been used for a long time to evade detection of malware. In [6], authors discussed some obfuscation technologies from the view of malware detection. Some static analysis approaches have been proposed for obfuscation detection. For instance, semantics templates were created in [1] to detect malicious traits, these templates were created based on instruction, variable and symbolic constants. Authors in [7] used software transformations to improve malware detection. In [2], authors present an architecture for detecting malicious patterns in executables that is resilient to common obfuscation. [8] detected obfuscation 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. While, some limits of static analysis for the detection of malicious code are explored in [9]. In this paper, we use dynamic information to detect malware obfuscation.

Besides, Obfuscation techniques are not only relevant to detecting morphed code but also several other issues. [11] present the relationship between obfuscation technology and protection of intellectual property rights present in proprietary software. [3] provide a brief introduction to the issue of measuring similarity between malicious programs, and how

978-0-7695-3859-4/09 $26.00 © 2009 IEEE
DOI 10.1109/IITA.2009.109

software evolution history refactoring is known to occur in the area.

Some relevant works to us have been done in the area of intrusion detection and bioinformatics. In [13], authors discover variable-length combinatorial pattern in biological sequences. In [10], authors used variable-length patterns to construct intrusion detection system. And [5] used system calls to detect sophisticated mimicry attacks in intrusion detection system.

III. MALWARE OBFUSCATION MEASURING

Maximal pattern is a subsequence in malware's runtime system call sequence, which can describe program specific behavior suitably. We firstly extract maximal pattern sequence from the runtime sequence of system calls, and measure the similarity between two maximal pattern sequences by evolutionary similarity. We use this similarity to detect obfuscation. Experiment results show that our method can efficiently detect the malware obfuscation.

A. Describe malware behavior using dynamic information

Runtime sequence of system calls is used to detect obfuscation. 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.

The number of the system calls is fixed for a special operating system version. For instance, Windows XP SP2 has 284 system calls named begin with "Nt". The system call number covers from 0 to 283. We trace the system calls at runtime. Figure 1 shows two sequences of system calls, S_1 denotes a part of system call sequence of Backdoor.Win32. AcidShiver.a, and S_2 denotes a part of Backdoor.Win32.AcidShiver.f:

$$S_1 = \langle 78,137,78,137,130,124,156,25,119,125,108,25,137,78 \rangle$$
$$S_2 = \langle 130,124,156,25,119,271,188,271,188,125,108,25,137,78 \rangle$$

In our experiments, we use Ether [12] 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. Extracting maximal pattern sequence from system call

We define a pattern represents a subsequence in malware's runtime system calls sequence. It frequently appears in program execution, and thus might correspond to specific task on operating system or to a basic block in program's source code [5]. A pattern a in system calls sequence S is a maximal pattern only if there is no pattern b for which holds that b contains a and the number of occurrences of b is equal to or lager than the number of occurrences of a.

System call sequence can be divided into a sequence of maximal patterns. For instance, M_1 is the maximal pattern sequence divided from S_1, and M_2 is the maximal pattern sequence divided from S_2:

$$M_1 = \left\langle \begin{array}{|c|}\hline 78,137,78,137 \\ \hline 130,124,156,25,119 \\ \hline 125,108,25,137,78 \\ \hline \end{array} \right\rangle \quad M_2 = \left\langle \begin{array}{|c|}\hline 130,124,156,25,119 \\ \hline 271,188,271,188 \\ \hline 125,108,25,137,78 \\ \hline \end{array} \right\rangle$$

We suppose M is the maximal pattern sequence divided from system call sequence S.We extract the maximal pattern sequence from system call sequence using algorithm 1:

Algorithm 1: Extract maximal pattern sequence from system call sequence.

Input: S : sequence of system calls
PL : the length of the shortest pattern
Fre : the lowest frequency

Output: M :the sequence of maximal patterns

1. Initial $E = \varnothing$ as the set of element maximal subsequence.
2. Scan S with a PL-length window and locate all elementary pattern into E with support at least Fre occurrences.
3. Initial $M = \varnothing$ as the sequence of maximal patterns.
4. Push all element subsequences into stack H.
5. While stack H is not empty and the length of S >0:
6. a is the top pattern in H.
7. If the frequency of a in S is larger than Fre :
8. If : there is a pattern b in H holds that b can prefix or suffix connect [13] a to c and the frequency of c in S is equal to or lager than a :
9. Replace a with c
10. Else:
11. Pop a
Delete all the pattern a in S and insert a into M
12. Return M

Firstly, we use a PL-length window to slide the system call sequence S, and generate a set of elementary patterns. Then we push them into a stack H. These are shown in lines 1-4. Second, we circularly combine these elementary patterns to recover the original maximal patterns, and output them. These are shown in lines 5-11. At last, the sequence of maximal pattern is returned as shown in line 12. We note that some uncommon system calls subsequence will be discarded while only some common ones will be inserted into M, maximal patterns sequence can describe program behavior more accurately than system call sequence. Two main parameters are the length of the shortest pattern PL and the lowest frequency Fre. Their influences on obfuscation detection will be discussed in section 4.

C. *Measuring obfuscation by evolutionary similarity*

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. 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 maximal patterns via inserting gaps between patterns so that identical patterns are aligned in successive line. We have researched in the efficiency of this method on aligning system call sequence [4].

M_1' and M_2' shows the maximal pattern sequences after alignment:

$$M_1' = \left\langle \begin{array}{|c|}\hline 78,137,78,137 \\ \hline 130,124,156,25,119 \\ \hline ---------- \\ \hline 125,108,25,137,78 \\ \hline \end{array} \right\rangle \quad M_2' = \left\langle \begin{array}{|c|}\hline ---------- \\ \hline 130,124,156,25,119 \\ \hline 271,188,271,188 \\ \hline 125,108,25,137,78 \\ \hline \end{array} \right\rangle$$

It is based on the hypothesis that in order to align identical patterns, the more gaps are inserted, the lower the similarity is. We note that given a pairs of sequences, there may be different alignments, and different similarity will be measured for them. We use the algorithm proposed by [4] to measure the maximal similarity for any alignments. We supposed two pattern sequences A and B are:

$$A = 'm,i,c,k,e,y' \qquad B = 'm,i,c,h,a,e,l'$$

Firstly, we insert a dump node in the front of two sequences:

$$A' = 'X,m,i,c,k,e,y' \quad B' = 'X,m,i,c,h,a,e,l'$$

We consider $F[L_A+1, L_B+1]$ is the similarity score matrix with A' placed horizontally at the top and B' placed vertically on the side, where L_A is the length of A and L_B is the length of B. We initialized it by:

$$F[i,0] = 0, i \in [0, L_A]; F[0,j] = 0, j \in [0, L_B]$$

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 patterns in A and the first j patterns in B. During the traversing, $F[i,j]$ is evaluated from the maximal score among $F[i-1,j]$, $F[i,j-1]$ and $F[i-1,j-1]+d$, where if $A[i-1] = B[i-1]$, $d=1$,otherwise $d=0$. The bottom right hand corner of the matrix is the maximal score for any alignments. Figure 1 shows the matrix during the traversing. Numbers in blue denote the identical pattern between two sequences. Number in red denotes the maximal score. And figure 2 shows the two sequences after alignment.

	X	**m**	**i**	**c**	**k**	**e**	**y**
X	0	0	0	0	0	0	0
m	0	1	1	1	1	1	1
i	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
h	0	1	2	3	3	3	3
a	0	1	2	3	3	3	3
e	0	1	2	3	3	4	4
l	0	1	2	3	3	4	4

Figure 1. Similarity score matrix

$$A_{Alignment} = 'm,i,c,-,k,-,e,-y'$$
$$B_{Alignment} = 'm,i,c,h,-,a,e,l,-'$$

Figure 2. Pattern sequences after alignment

IV. EVALUATION AND EXPERIMENTS

A. *Date set and Evaluation measures*

In order to evaluate the effectiveness of our method, we detect the obfuscation against real-world malware variants, we only consider whether a pair of malwares is variants or not. It is based on the hypothesis that a high 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.

Experiment includes training phase and testing phase. We randomly chose 505 pairs of variants and

528 pairs of non-variants as our training set, and other 197 pairs of variants and 210 pairs of non-variants as our testing set. These malwares include Worms, Trojans and Backdoors, which are named by Kaspersky. All of the malware samples run in Windows XP SP2, and we intercept 1000 system calls by Ether [12] for them. Then, we use algorithm 1 to extract maximal pattern sequences from system call sequences. So each malware sample is represented by a sequence of maximal pattern.

Two major factors we measured are the true positive rate (TPR) and false positive rate (FPR):

True Positive (TP): Number of variants correctly classified to be variants.

True Negative (TN): Number of non-variants correctly classified to be non-variants.

False Positive (FP): Number of non-variants incorrectly classified to be variants.

False Negative (FN): Number of variants incorrectly classified to be non-variants.

True Positive Rate (TPR): $TPR = TP/(TP+FN)$

False Positive Rate (FPR): $FPR = FP/(FP+TN)$

The goal of our experiment was achieving a high TPR and a low FPR.

B. Model Training and Classification

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 evolutionary similarity mentioned in section 3.3, and calculated the average similarity $E_1 = \frac{1}{n_1}\sum_{i=0}^{n_1} similarity_i$, $n_1 = 505$. Second, we calculated the similarity of non-variants. The average similarity of non-variants was calculated as $E_2 = \frac{1}{n_2}\sum_{i=0}^{n_2} similarity_i$, $n_2 = 528$. The threshold is expressed as:

$$threshold = E_2 + k*(E_1 - E_2) \qquad (1)$$

We observed three important parameters in experiments are the PL and Fre in algorithm 1, and the k in formula (1). The best performance can be reached when $PL = 4, Fre = 5$, and $k = 0.35$. Figure 3 shows the distribution of similarity of training set when $PL = 4, Fre = 5$, and $k = 0.35$, X-axis denotes the number of variants and non-variants, Y-axis denotes the similarity of them. The red circles represent variants and the blue crosses represent non-variants. We can intuitively observe the variants have higher similarity score than non-variants. But some specific pairs violated our assumption. We consider reason is the malware naming problem.

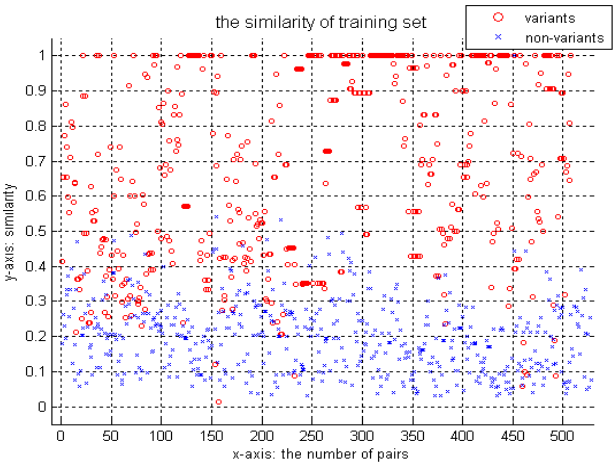

Figure 3. The similarity of training set with $PL = 4, Fre = 5$.

True positive rate and false positive rate were measured in testing phase. First, the similarity of 197 pairs of variants and 210 pairs of non-variants was calculated. So $TPR = TP/(TP+FN)$, where TP is the number of variants pairs whose similarity is above the threshold, and $TP+FN = 197$. False positive rate can be express as $FPR = FP/(FP+TN)$, FP is the number of non-variants pairs whose similarity is above the threshold, and $FP+TN = 210$.

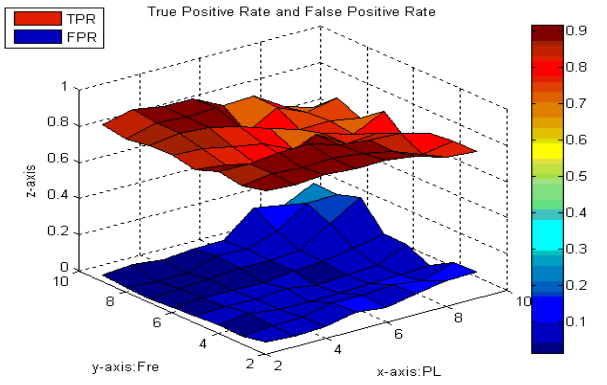

Figure 4. TPR and FPR with $k = 0.35$.

Figure 4 shows the TPR and the FPR using different PL and Fre when $k = 0.35$: X-axis denotes different PL, and Y-axis denotes Fre and Z-axis denotes the value of TPR and FPR. We can observe that bad performance was reached when the value of PL or Fre is getting lager. And the best performance was achieved when $PL = 4, Fre = 5$, it got the TPR of 91.5% and the FPR of 4.3%.

Figure 5 shows the TPR and FPR using different k when $PL = 4, Fre = 5$. In figure 5, X-axis denotes different k, and Y-axis denotes the value of TPR and FPR. We can see both the TPR and FPR are getting lower when k is getting larger. In the middle way, we

consider the best performance can be achieved when $k = 0.35$. And its corresponding ROC curve is shown in Figure 6, with the value of $AUC = 0.9678$. A good classify performance was achieved. It demonstrates that our method has strong resilience to detecting obfuscation.

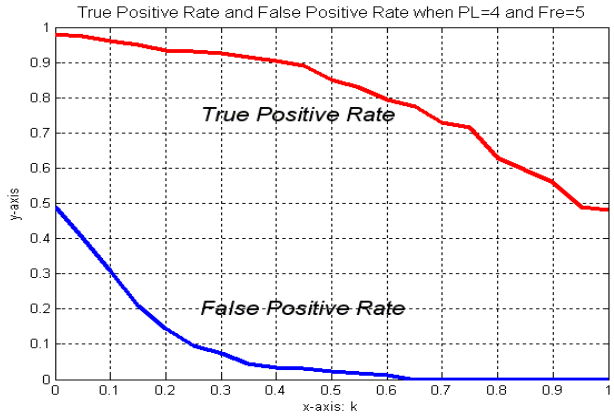

Figure 5. TPR and FPR with $PL = 4, Fre = 5$.

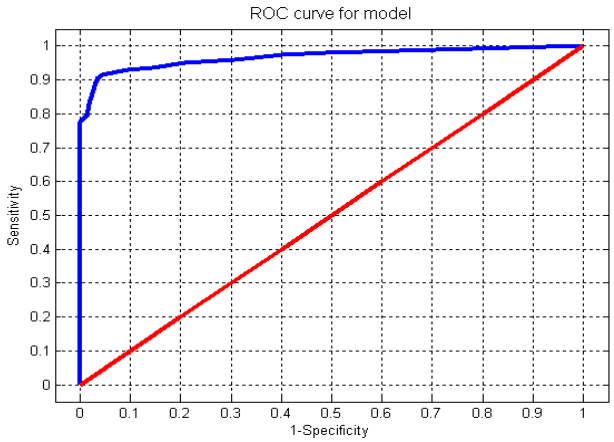

Figure 6. ROC curve with $PL = 4, Fre = 5$.

V. Conclusion and Future Work

In this paper, we propose a method to detect malware obfuscation using maximal patterns. We extract maximal pattern sequence from the malware's runtime system calls, and measure the similarity between two pattern sequences by evolutionary similarity. The experiment results have shown that maximal pattern have strong resilience to malware obfuscation. So constructing a malware detect system based maximal patterns matching is our on going 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, 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, 2005, pp. 32-46.

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

[3] 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.

[4] Jian Li, Jun Xu, Ming Xu and Ning Zheng. Malware obfuscation measuring via evolutionary similarity. In first International Conference on Future Information Networks, BeiJin,China,2009.

[5] 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.

[6] 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.

[7] M. Christodorescu, S. Jha, J. Kinder , S. Katzenbeisser, and H. Veith. Software transformations to improve malware detection. Journal in Computer Virology, 2007,3(4),pp.253-265.

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

[9] A.Moser, C.Kruegel, and E.Kirda. Limits of Static Analysis for Malware Detection. In Proceeding of Twenty-Third Annual Computer Security Applications Conference, 2007, pp. 421–430.

[10] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In Proceedings of the 2000 Recent Advances in Intrusion Detection, Toulouse, France, 2000, pp. 110-129.

[11] 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.

[12] 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.

[13] I.Rigoutsos and A.Floratos. Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. Bioinformatics, 1998, 14(1):55–67.