Malware Behavior Extracting via Maximal Patterns

Jian Li * Ning Zheng * Ming Xu * YongQing Sun # JiouChuan Lin #

* 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 {nzheng,mxu}@hdu.edu.cn {yqsun,cjlin}@mail.trimps.ac.cn

***Abstract*—With the prevailing of malware, it is necessary to describe mal-program's behavior in an efficient way. In this paper, a malware behavior extracting system is addressed. It used Intel VT to trace malware's runtime system calls and extracted maximal patterns to describe malware behavior. A patterns extracting algorithm is proposed to extract maximal patterns from system calls sequence. Real-world malwares are used to validate this method. The results of experiments have shown that the method can describe the behavior of mal-program with strong resilience and high accuracy.**

Keywords— malware, behavior, Intel VT, maximal pattern

I. INTRODUCTION

A malware is a computer program which has malicious intend. It includes virus, Trojan, backdoors and so on. Some technologies are used by malware authors to evade signature detection from anti-malware software. Obfuscation is one of the common technologies, which includes Polymorphism and metamorphism. Anti-malware softwares which use signature-matching approaches are vulnerable to these obfuscation technologies [1].The reason is that the traditional description for malware behavior, such as signature based description can not achieve resilience to them. Besides, other software protection technologies, which include packing, anti-debug and anti-virtual machine technologies, have been widely used by malware authors. These technologies prevent malware to be analyzed, so behavior description always encounters low accuracy.

In order to against malware, an efficient behavior extracting system is required, which can be used to describe malwares' behavior with strong resilience and high accuracy. The output of the extracting system can be used by malware detectors to detect malware variants. Since this system provides a succinct description of malicious behavior present in a malware, it is also expected to help security analysis for understanding the malware.

In this paper, we use maximal patterns to describe the behavior of malware. A maximal pattern represents a subsequence and it's coverage in a runtime system calls sequence. This subsequence 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.

A malware behavior extracting system is addressed. In our malware behavior extracting system, we used Intel VT to trace the malware's runtime 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. An algorithm is proposed to extract the maximal patterns from system call sequence. Our experiments have shown that our method can describe the real-world malware's behavior with high accuracy and strong resilience.

II. RELATED WORK

Some malware behavior extracting frameworks have already been proposed. [1] present architecture for detecting malicious patterns in executables that is resilient to common obfuscation. Some semantic based research such as [2] created semantics templates to represent malicious traits. These templates were created based on instruction, variable and symbolic constants. A robust signature-based malware obfuscation detection technique is introduced in [3]. 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.

There are also some solutions on describing malware behavior using dynamic action. In [4], authors use Hidden Markov Models to construct the behavior of a process. In [5], authors define a new graph representation of program behavior and use a mining algorithm to constructs a malicious specification. In [6], author proposed a novel framework to automatically discover and analyze traffic generated by anomalous behaviors that interact with a non-solicited network traffic monitoring system. The IMDS [7] rest on the analysis of Windows API execution sequences called by PE files. [8] represents the malware behavior as a 35-Dimension feature vector. Each dimension stands for a malicious run-time behavior feature represented by corresponding Win32 API calls and their certain parameters. Besides, some sandbox environments and security analysis system [9] [10] [11] also use dynamic information of the mal-program. In this paper, we also extract maximal patterns from runtime system call of mal-program. The difference between our paper and these

978-0-7695-3887-7/09/$26.00 ©2009 IEEE

researches is that we address on the malware behavior extracting system but not the detection system.

Some similar researches have been proposed in the area of intrusion detection. [12] describes process behavior using fixed-length subsequence of system calls. [13] uses behavior patterns to detect sophisticated mimicry attacks in intrusion detection system. In our research, the major difference to them is that we not only divide the system calls sequence into subsequences, but also describe the program behavior with the consideration of the occurrence and length of behavior pattern. This consideration is based on the hypothesis that a subsequence with a longer length and higher frequency of occurrence should represent a more important task than these ones with shorter length and lower frequency of occurrence.

III. MALWARE BEHAVIOR PATTERNS EXTRACT

In this section, we'll describe our malware behavior extracting system. First we'll mention the system architecture. And then, we detail the method of system call tracing via Intel VT. At last, we'll propose our maximal patterns extract algorithm.

A. System architecture

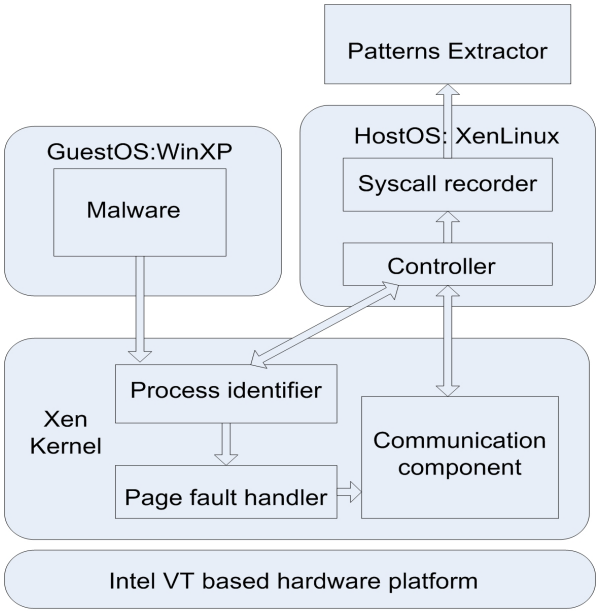

Figure 1: System architecture

The system architecture is shown in figure 1. We construct our system on Intel VT [14] based hardware virtual machine. Typically, we use Xen [15] as our virtual machine monitor. Our system can be divided into 4 major modules. First, we do some modification on Xen kernel. It used to trace the system calls and transfer this information into HostOS. Second, some user-space components have been built in HostOS module, which runs an operating system of XenLinux. Third, a patterns extractor is addressed to extract maximal patterns. The algorithm used by patterns extractor will be described in section 3.3. At last, we run the target malware in Windows XP, which is the GuestOS module and can be monitored by the Xen kernel.

The reason why we divide the target malware and behavior analysis modules into different operating system is that since Xen can ensure the transparency between HostOS and GuestOS, they cannot impact the execution of each other, and target malware cannot feel the existence of analysis environment. So some anti-debug technologies will lose their effectiveness. In [10], authors have discussed the transparency for this architecture.

B. System call tracing via Intel VT

We use the runtime sequence of system calls to construct 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 4.3.

Now, we detail the architecture and describe the flow of system call tracing. As shown in figure 1, the controller in HostOS specifies the target mal-process for tracing, and transfers it to the process identifier. Process identifier resides in Xen kernel module and uses CR3 register to identify the mal-process. In x86, the CR3 register contains the value of current process' page directory pointer, which can be used to uniquely represent a process.

When a system call is being invoked, an instruction called SYSENTER executes and transitions to protection ring 0. The CPU use a register called SYSENTER_EIP_MSR to contain the address which dedicates where to jump when SYSENTER executes. While the CR3 register point to the mal-process, the target will be identified, and then the process identifier modifies the value of SYSENTER_EIP_MSR. So if a system call is invoked by mal-process, a page fault will be raised. The page fault handler captured the page fault and use virtual machine introspection [16] to reach the information of the system call. This information will be transferred to the communication component. The communication component use event channel and shared memory to transfer these information to the controller. And the controller store the system calls into syscall recorder as a sequence.

At last, the sequence of system calls will be transfer into patterns extractor, which is used to extract maximal patterns. The algorithm used in patterns extractor will be described in section 3.3, and the resilience of it will be discussed in section 4.3.

C. Maximal Patterns Extract Algorithm Description

In figure 1, patterns extractor got a sequence of system calls from HostOS. The system calls sequence can be broken into subsequences. These subsequences frequently appear together in program execution, and thus might correspond to specific task on operating system or to a basic block in program's source code.

In the module of patterns extractor, we describe the behavior of a mal-program by maximal patterns. We use maximal patterns extracting algorithm to extract maximal patterns. Before describing the algorithm, we'll give a series of definitions:

Definition 1: A subsequence a in system calls sequence S is a maximal subsequence only if there is no subsequence 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.

For instance, figure 2 list a part of a system call sequence and figure 3 list a maximal subsequence. They are all expressed by system call names.

Definition 2: We suppose a is a maximal subsequence with length l. Then, $prefix(a)$ is the prefix of a with length $l-1$. And $suffix(a)$ is the suffix of a with length $l-1$.

For instance, if $a='108,25,137,78'$, then $prefix(a)='108,25,137'$ and $suffix(a)='25,137,78'$.

Definition 3: If $suffix(a)=prefix(b)$, then the function $PSconnect(a,b)$ returns a new subsequence ab' where b' is a subsequence that $b=prefix(b)b'$.

For instance, if $a='119,177,25', b='177,25,119'$, then $PSconnect(a,b)='119,177,25,119'$.

Definition 4: A Function $\mathrm{cov}(a,S)$ identifies the coverage of a subsequence a. In system calls sequence S, if the number of occurrences of subsequence a is n, the length of a is l_a and the length of S is l_S, then:

$$\mathrm{cov}(a,S)=n\times l_a / l_S$$

Definition 5: A maximal pattern p is a 2-tuple which can be expressed as $p=\{a,\mathrm{cov}(a,S)\}$, where a is a maximal subsequence and $\mathrm{cov}(a,S)$ identifies its coverage in S.

Definition 6: We define a mal-program $P=\bigcup_{1\le i\le I} p_i$, $p_i=\{a_i,\mathrm{cov}(a_i,S)\}$. Where p_i is a maximal pattern, I is the total number of maximal patterns and S is the runtime system calls sequence of P.

We use these maximal patterns to describe the behavior of a mal-program. It is based on the hypothesis that if a subsequence with a longer length or higher frequency of occurrence, it might be more important. The maximal patterns extracting algorithm is shown as below:

Maximal patterns extracting algorithm:

Input: S: sequence of system calls

L: the length of the shortest subsequence

K: the lowest frequency

Output: M :the set of maximal behavior patterns

1. Initial $E=\varnothing$ as the set of element maximal subsequence.
2. Scan S with a L-length window and locate all elementary subsequence into E with support at least K occurrences.
3. Initial $M=\varnothing$ as the set of maximal patterns.
4. Push all element subsequences into stack H.
5. While stack H is not empty:
6. Define a is the top subsequence in H.
7. If: there is a subsequence b in stack that $suffix(a)=prefix(b)$
8. Let $c=PSconnect(a,b)$
9. If the number of occurrences of subsequence a is equal to c
10. Replace a with c
11. Else if: there is a subsequence b in stack that $suffix(b)=prefix(a)$
12. Let $c=PSconnect(b,a)$
13. If the number of occurrences of subsequence a is equal to c
14. Replace a with c
15. Else:
16. Pop a
17. Set $p=\{a,\mathrm{cov}(a,S)\}$ into M
18. Return M

Firstly, we use a L-length window to slide the sequence S, and generate a set of elementary subsequences. Then we push these elementary subsequences into a stack H. This is shown in lines 1-4. Second, we combine these elementary subsequences using an efficient way in order to recover the original maximal patterns. This is shown in lines 5-17. At last, the set of maximal patterns is returned as shown in line 18.

IV. EXPERIMENTS

In this section, we'll discuss some experiments to validate our method. First, a single malware is analyzed to validate the efficiency of our system. Second, 800 malware will be analyzed to validate the detectability. At last, we use our experiment results to detect malware variants, which can be used to validate the resilience of the behavior extracting algorithm.

A. *Efficiency of our system*

For the fulfillment of efficiency, we analyze a malware called backdoor.win32.bifrose.kt, which bypasses the normal authentication and provides remote access to computers. Armadillo, a popular software protection tool with strong anti-debugging and anti-analysis protection, is used to pack the malware sample. We analyze the packed sample comparing against CWSandbox, Norman Sandbox and Anubis, which are popular sandbox environments. Table 1 shows the comparison of detect ability on the packed sample. It confirms the strong analysis ability of our system.

TABLE I. COMPARISON OF THE DELECTABILITY.

	Our system	**CWSandbox**	**Norman Sandbox**	**Anu-bis**
detectable	√	×	×	×

In our experiment, we intercepted 3529 system calls generated by backdoor.win32.bifrose.kt. A part of the system calls sequence is shown in figure 2.

……
NtProtectVirtualMemory
NtFlushInstructionCache
NtQueryInformationProcess
NtQuerySystemInformation
NtQuerySystemInformation
NtCreateSection
NtSecureConnectPort
NtClose
……

Figure 2. A part of the system call sequence

NtOpenProcessTokenEx
NtQueryInformationToken
NtClose

Figure3: One of the maximal subsequence, which represents the mal-behavior of stealing other processes' information.

We use the malware behavior patterns extracting algorithm to extract the behavior patterns of the target program. The statistics of the behavior patterns generated by the extractor are summarized in table 2.

TABLE II. STATISTICS OF THE BEHAVIOR PATTERNS

L / K	**2**	**3**	**4**	**5**	**6**	**7**
2	153	167	152	136	122	111
3	100	111	94	84	81	75
4	74	79	70	68	69	62
5	59	60	49	56	58	50
6	51	52	46	53	54	44
7	42	46	45	49	47	39

In table 2, L represents the length of the shortest subsequence and K represents the lowest frequency. One of the maximal subsequence a is shown in figure 3. It represents the mal-behavior of stealing other processes' information in operating system. As it occurs 134 times in the sequence S of 3529 system calls and. So the $\mathrm{cov}(a,S) = 134\times 3 / 3529 = 0.114$.

So, it's corresponding behavior pattern: $p = \{(\mathrm{NtOpenProcessTokenEx}, \mathrm{NtQueryInformationToken}, \mathrm{NtClose}), 0.114\}$

B. *Detectability*

Another factor we considered is the detectability to the real-world malware. We randomly chose a set of 800 malwares to verify it. The malware set include Worms, Trojans, Backdoors, which are named by kaspersky. The distribution of these malwares is shown in figure 4:

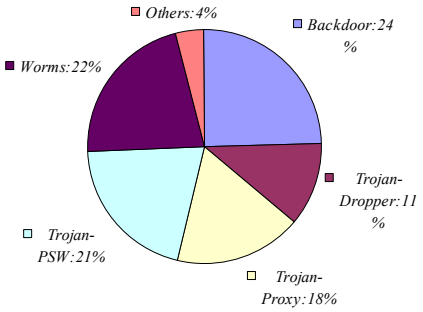

Figure 4: The distribution of malware testing set.

Because our purpose is extracting the maximal patterns as the description of malware, whether they are mal-patterns or benign-patterns, so we denoted a malware is successfully detected if there are maximal patterns being extracted. In the 800 samples, three malware's system call cannot be traced for unknown reason; two malware's maximal patterns cannot be extracted because their system calls are composed of consecutive occurrences of the same system call number. Table 3 lists the name of these malwares.

TABLE III. MALWARES CANNOT BE DETECTED.

Malwares cannot be traced	Backdoor.Win32.Bingle.a Backdoor.Win32.Bingle.b Backdoor.Win32.Bifrose.bgi
Malwares cannot be extracted patterns	Trojan-IM.Win32.VB.p Trojan-Dropper.Win32.Tiny.c

For all of malwares with successful detection, we used $L = 3$ and $K = 5$ to extract the behavior patterns with different sequence length, and the average number of patterns has been shown in figure 5. X-axis denotes different system calls sequence length, and Y-axis denotes the average number of patterns. We can see the average number is increasing with the sequence getting longer. And when it reached a certain length, the number of patterns will be stable.

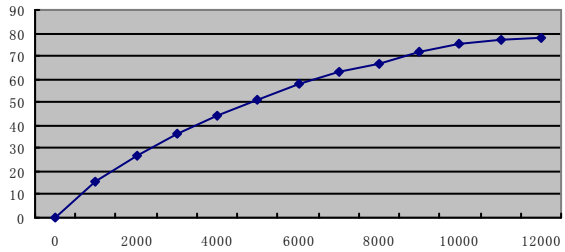

Figure 5: Average number of behavior patterns.

C. *Resilience to obfuscation*

In order to verify the resilience to obfuscation for our behavior extracting system, we measure the similarity between malware variants. 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 a pair of non-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.

Between two malwares A and B , we calculated the intersection rate of two set of the malware's maximal patterns, which was used to identify the similarity. So the similarity can be calculated like:

$$similarity(A,B)=\frac{\sum \mathrm{cov}(c,A)+\sum \mathrm{cov}(c,B)}{\sum \mathrm{cov}(c,A)+\sum \mathrm{cov}(c,B)+\sum \mathrm{cov}(a,A)+\sum \mathrm{cov}(b,B)}$$

Where $c\in A\cap c\in B$, $a\in A\cap a\notin B$, and $b\notin A\cap b\in B$. Because the denominator is approximately equal to 2, it can be sampled as:

$$similarity(A,B)=\frac{\sum \mathrm{cov}(c,A)+\sum \mathrm{cov}(c,B)}{2} \quad (1)$$

We randomly chose 20 pairs of malware variants and 20 pairs of non-variants, and calculated their similarities using equation (1). We intercepted system calls sequence with length of 1000 and use $L=3$ and $K=5$ in these experiments. The results have been shown in figure 6. X-axis denotes different pairs of malwares, and Y-axis denotes the similarity. High scores were reached between malware variants, and low scores were reached between non-variants. So the resilience to the obfuscation is demonstrated.

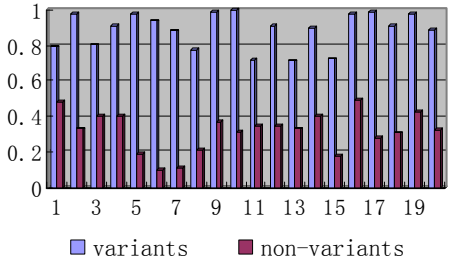

Figure 6: Similarity calculated by equation (1).

V. Conclusion and Future Work

We addressed a malware behavior extracting system in this paper. In this system, we used Intel VT to trace the malware's runtime sequence of system calls, and used patterns extracting algorithm to extract maximal patterns, which are represented as a set of 2-tuples. We validate our method with real-world malwares. The results of our experiment have shown that the proposed method can describe the behavior of mal-program with strong resilience and high accuracy.

Our system addressed on describing the malware's behavior using maximal patterns. However, these patterns include mal-patterns and benign-patterns. So how to identify whether a pattern is a mal-one or a benign-one is a challenging work.

Besides, the output of our system can be used by malware detectors and security analyst for understanding the malware. So using the maximal patterns to construct a malware detection system is our ongoing work. Besides, we'll address on more efficient algorithm to extract maximal patterns.

References

[1] M. Christodorescu and S. Jha, Static analysis of executables to detect malicious patterns, *In Proceedings of USENIX Security Symposium*, Washington, 2003, pp. 12-12.

[2] M.Christodorescu and S.Jha, Semantics-aware malware detection. *In Proceedings of the 2005 IEEE Symposium on Security and Privacy,* Oakland, CA, USA, 2005, pp. 32-46.

[3] A. H. Sung and J. Xu, Static analyzer of vicious executables (SAVE), *In Proceeding of the 20th Annual Computer Security Applications Conference,* Washington, 2004, pp. 326–334.

[4] D. Gao, M. K. Reiter, and D. Song. Behavioral distance measurement using Hidden Markov Models. *In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection,* 2006.

[5] M.Christodorescu, S.Jha, and C.Kruegel. Mining Specifications of Malicious Behavior. *In 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Croatia, 2007, pp. 5–14.

[6] U.Zurutuza, R.Uribeetxeberria, D.Zamboni, A data mining approach for analysis of worm activity through automatic signature generation. *In Proceedings of the 1st ACM workshop on Workshop on AISec*, Alexandria, Virginia, USA, 2008, pp. 61-70.

[7] Y.Ye, D.Wang, IMDS: Intelligent Malware Detection System, *In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining,* California, USA, pp. 1043-1047.

[8] Y.t.Hu, L.Chen, M.Xu, N.Zheng, Unknown Malicious Executables Detection Based on Run-time Behavior, *In Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, Volume 04, pp. 391-395 .

[9] C.Willems, T.Holz, and F.Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy,* 2007, 5(2), pp.32-39.

[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.

[11] X.Jiang, X.Wang, and D.Xu. Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction. *In Proceedings of the 14th ACM conference on Computer and Communications Security,* New York, USA, 2007, pp. 128–138.

[12] S.A.Hofmeyr, S.Forrest, A.Somayaji, Intrusion detection using sequences of system calls, *Journal of Computer Security* ,Volume 6 , Issue 3, 1998, pp. 151 – 180.

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

[14] G.Neiger, and A.Santoni. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal,* 2006, 10(3), pp. 167-177.

[15] I.Pratt, and K.Fraser. Xen 3.0 and the art of virtualization. *In Proceedings of the 2005 Ottawa Linux Symposium,* 2005, pp. 65-78.

[16] T.Garfinkel, and M.Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *In Proceedings of the 10th Network and Distributed Systems Security Symposium,* San Diego, USA, 2003, pp. 191-206.