2009 International Conference on Management of e-Commerce and e-Government

Unknown Malware Detection Based on the Full Virtualization and SVM

Hengli Zhao, Ning Zheng, Jian Li
Institute of Computer Application Technology
HangZhou DianZi University
Hangzhou,China
e-mail: zhenry86@163.com, nzheng@hdu.edu.cn

Jingjing Yao, Qiang Hou
The Third Research Institute of the Ministry of Public Security
ShangHai,China
e-mail: yjj@mail.trimps.ac.cn

***Abstract*—Malware has become the centerpiece of security threats on the e-commercial business. The focus of malware research is shifting from using signature patterns to identifying the malicious behavior patterns. Many researcher extract behavior pattern from system call sequences to identify malware from benign programs with data mining techniques. Most system call tracing tools must run alongside the malware in the same system environment and could be easily detected by malware. In this paper, we propose a new system calls tracing system based on the full virtualization via Intel-VT technology. Malicious samples are running in a GuestOS and they can not detect the existence of system call tracing tool running in the HostOS. We collect a system call trace data set from 1226 malicious and 587 benign executables. An experiment based on the SVM model shows that the proposed method can detect malware with strong resilience and high accuracy.**

Keywords-malware; full virtualization; native API sequence; SVM

I. INTRODUCTION

A Malware are computer programs which do damage or inconvenience to computer users. Security threat posed by malware such as viruses, worms, trojan horses, rootkit are becoming more serious. Traditional signature-based detection performs poorly especially malware author employ polymorphism and metamorphism methods to frustrate analysis. Other malware research turns to the analysis based on malware behavior, because no matter the disguise, malware will behave badly. One technique for behavior analysis is using data mining techniques to automatically extract behavior pattern from malware and benign programs. Researcher trace the sequence of system calls as the program behavioral characteristics, because system call is the only way for a program to interact with the operating system, it's natural place to intercept a program and perform monitoring [1].

Traditional tools for obtaining system calls include disassemblers, debuggers and hooking system calls in a dynamic black box. These tools are vulnerable to dynamic code translation and debugger fingerprinting [2]. Dynamic hooking tools rely on monitoring a program once it is running in a virtual machine (VM) to isolate and roll back the system when damage happens. Unfortunately, malware authors are using anti-virtualization [3] to evade detection. When the malware detects it is running inside a VM, it will often exit or behave differently to fool analyzers and conceal its intentions.

In this paper, we design a system call tracing system based on full virtualization via Intel VT. We place trace program running in HostOS and target program running in GusetOS, trace tool can get system call events of GuestOS through the virtual machine monitor (VMM) and virtual machine control structure (VMCS). We capture special VT-transition to identify the process and get the sequence of system calls at real-time.

In summary, this paper makes contributions as follows:

- We design a robust system call tracing system based on full virtualization via Intel VT. The trace system could be transparent to the malware application.
- We use SVM to extract the various features from the system call sequences of malware and benign programs, and then construct a model for malware detection.

The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 describes our system call trace system. Section 4 details the construction of SVM model and the experiment results. Section 5 concludes this paper and gives the future work.

II. RELATED WORK

Data mining has been the focus of malware researchers to detect unknown malwares. A number of classifiers have been built and shown this way have very high accuracy rates [4]. Most of these classifiers use system calls as their primary feature like [11]. According to the tracing of system call, some analysis tools and sandboxing environments such as CWSandbox [5] could be used to trace windows API. Anubis [6] could trace native API and windows API.

All approaches mentioned above use either in-guest environment or whole-system emulation, which are vulnerable to be attacked. A more efficient solution, which is called out-of-guest approach was proposed. Out-of-guest approach places analysis tool and target program into different operating systems. This approach has a range of advantages and has been used in a number of solutions. Livewire [7] uses this approach to pull the intrusion detection system out of the host for greater attack resistance. Ether [8] uses virtual machine technology to improve security and accountability of the analysis systems.

978-0-7695-3778-8/09 $26.00 © 2009 IEEE
DOI 10.1109/ICMeCG.2009.114

III. SYSTEM CALL TRACING SYSTEM

In this section, we propose our system on tracing malware behavior based on the full virtualization. Firstly, we introduce full virtualization with Intel VT, then we give the details of our system architecture and how it can monitor system call at real-time. At last we will introduce the advantage of our trace system.

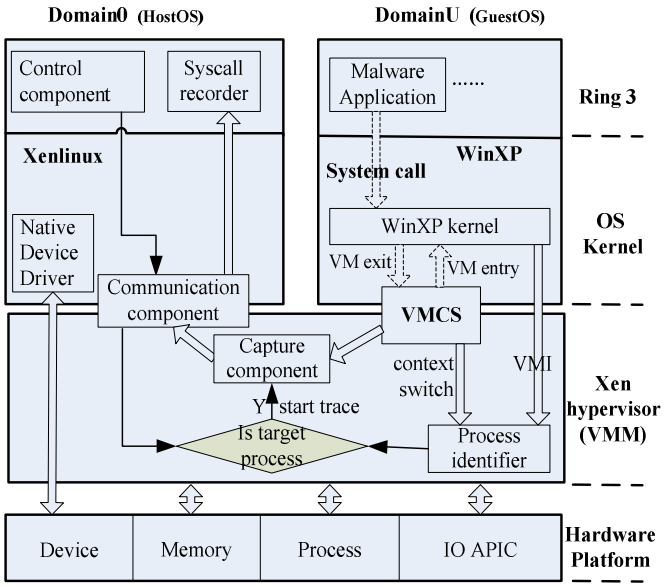

Figure 1. System architecture.

A. Full virtualization with Xen and Intel-VT

Full virtualization can provide a complete simulation of the underlying hardware. Unlike para-virtualized approach which has to modify the OS kernel to run in para-virtual machine. To implement full virtualization on x86 platform, Intel launched hardware virtualization extensions to augment the x86 instruction set (Intel-VT).

We extended the Xen as our virtual environment which consists of Xen hypervisor and domains. The Xen hypervisor is worked as VMM that allows the hardware resources to be virtualized and dynamically shared between various virtual machines(VMs). These VMs belong to different domains. One domain which has special privilege called Domain0, Dmain0 is a modified Linux kernel (also called HostOS). It has special rights to access physical I/O resources and interacts with other VMs which called DomainU (GuestOS). As shown in Figure 1 our system use Xenlinux as HostOS and let an unmodified WinXP as a GuestOS which is responsible for running the target malware. We divide the target program and system call trace modules into different system since Xen can ensure the transparency between HostOS and GuestOS, they cannot impact the execution of each other. And the most important is that target malware cannot feel the existence of analysis environment, so some anti-virtualization and anti-hooking technologies will lose their effectiveness.

Intel-VT has two forms of CPU operation: VMX root operation and VMX non- root operation. It also defines two transitions: a transition from VMX root operation to VMX non-root operation is called VM entry, and another transition from VMX non-root operation to VMX root operation is called VM exit. Xen Hypervisor runs in VMX root mode and GuestOS runs in non-root mode. Some important system events which include context switch, page-fault, and hardware interruption in GuestOS will cause VM exit. We can set VMCS to manage VM transition and capture the process status change and system call events. Next section we will details the monitoring of system call execution.

B. Monitoring system call execution

Our system call tracer consists of two components: the user-space component and the kernel-space monitor component. User-space control component assigns a process in HostOS application level. Kernel-space monitor component identify the process and realize the system call interception in Xen hypervisor.

In order to identify the process, we must set VMCS in the VMM layer. VMCS manages processor behavior in VMX non-root operations when VM entries and VM exits happen. The VMCS is logically divided into sections, two of which are the guest-state area and the host-state area. The guest-state area contains the state of the virtual CPU associated with the GuestOS. It includes fields corresponding to registers that manage processor operation, such as the segment registers, CR3. The CR3 register contains the value of page directory pointer of current process. The change of process running in the GuestOS will result in the value change of CR3. So we custom the VMCS to gain control on every context switch (mov cr3) which causes a VM exit when the page directory entry pointer is accessed in the GuestOS. We also use virtual machine introspection to get the process name and our process identify component will compare with an assigned process. If the process matches the target process specified by user-space control component, Tracing is started.

In order to achieve the goal of system call interception, our system exploits the x86 system call entry mechanism to inform the capture component that system call was invoked by the assigned process. When a system call is being invoked, SYSENTER instruction executes. The behavior of SYSENTER is configured with the value in the model specific register (MSR). When SYSENTER is executed, values in MSR are loaded in several registers, including the stack pointer to the kernel mode stack and instruction pointer to the value in SYSENTER_EIP_MSR register. SYSENTER_EIP_MSR contains the address which dedicates where to jump when the SYSENTER executed. We set the value of SYSENTER_EIP_MSR to a chosen value which is not present and stored the original value. When a SYSENTER executes, a page fault will be raised and VM exit will occur. The capture component capture the page fault and get the information of current system call which will then be transferred to the communication component. The communication component use event channel and shared memory to transfer the information to the system call

recorder. We must reset the stored value into SYSENTER_EIP_MSR. Execution of target process resumes as if SYSENTER jumped directly to the expected address, so interception can be transparent to the malware application and prevent malware from avoiding monitor.

Compared with other tools, our system fulfills the integrity and robust requirements of malware analysis system. By placing tracer and target malware in different system, malware cannot affect the environment which the analyzer resides in. Even if malware gains root access or completely damage guest operating system, the analysis software and the tracer date won't be affected. This will ensure that the analysis system has strong robustness. In VT-architecture, GuestOS's processor state will be saved in VMCS when VM exits occurs, And VM entry will load them back from VMCS. Each action of target process will cause a change of GuestOS CPU state. We can set the VMCS to manage which state will cause a VM exit. When expected events cause vm exit, all the CPU change information will be reflected in the value change of VMCS. We monitor VMCS so when the target process is about to run or it interact with the GuestOS, our tool does not miss any behavior. This will guarantee our analysis system have high integrity.

IV. SVM MODEL AND EXPERIMENT

A. Data preprocess and SVM construction

Through our system call tracing system, we will get a large data set of system call sequences. In this section, we'll use SVM to process this dataset and detect unknown executables. First we assign each system call an index value, for example '116' assign to system call "NtOpenFile" showing as Figure 2 (a). We save the numerical sequence correspond to each system call in a data file.

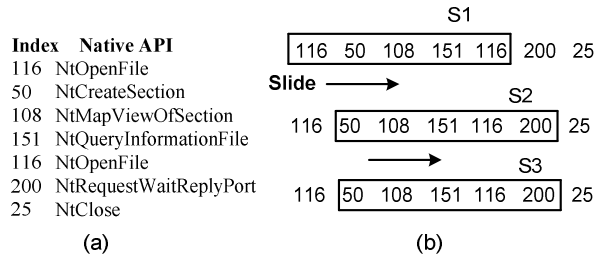

Figure 2. (a) A part of the system call sequence (b) short API call traces slide with k=5.

In order to use the SVM model, we slide a window of size *k* across the trace sequences, recording each unique sequence of length *k* that is encountered. We use the short k-length system call sequence to generate the feature vector for SVM. For example, if k=5, one get the unique sequences show detail as Figure 2 (b). Short sequence of system calls represents the order of system call by executing process. Wenke Lee [12] found that one cannot get useful message from system call sequence when window size larger than 30. Take conditional entropy and computational consumption into consideration, we adapt 8 as the value of short sequence length k in our paper. Short sequence can extracted from malware and benign samples by scanning the history of system call with k length slide window. We saved these short system call sequence in feature database. We adopt the zhang's distance algorithm [11] with threshold 4 to reduce number of Short sequences. Then we adopt the attributes reduction of Information-Gain [9] on the feature sets. We calculate the Information Gain of each feature:

$$IG(j) = -\sum_{v_j \in (0,1)} \sum_{c \in \{c_j\}} P(v_j, C) \log \frac{p(v_j, C)}{p(v_j)\, p(C)} . \quad (1)$$

Where $IG(j)$ denote the Information Gain value of feature j, C represents one class in $\{C_i\}$, $\{C_i\}$ have binary value which means malware or benign sample in our paper. $p(v_j, C)$ denotes the probability of feature j with the value of v_j in class C. $p(v_j)$ denotes the probability of feature j equal v_j in all training sets. $p(C)$ denotes the probability of class C in all training sets. At last we select 1600 features which have the lower IG value and save in the feature database D. After Data preprocess we get system call sequence X of each executable program, then we use 8-length windows to slide sequence X and generate short system call sequence t. We compare every short system call sequence t with the feature database D. If t exists in the feature database D, we set the number of occurrences of t in X as the value of corresponding feature vector. Through this way we can generate a feature vector for every sample. Then we can use the well-established method of Support Vector Machines (SVM) to train and classify the sample set.

B. Experiment result

During the experiment, we use the software LIBSVM. To evaluate our system we were interested in several quantities:

- False Negative (FN): the ratio of malicious executable examples classified as benign.
- False Positive (FP): the ratio of benign programs classified as malicious executables.

After comparison with other kernel Function, we choose Radial Basic Function in our SVM model:

$$K(x, y) = \exp(-\frac{\| x - y \|^2}{\sigma^2}) . \quad (2)$$

There are two parameters in equation (2) while using RBF kernel: $1/\sigma^2$ and C. It is not known beforehand which C and $1/\sigma^2$ are the best for one problem. Consequently some kind of parameter search must be done. So we try two variable group value of $(1/\sigma^2, C)$ to test the classification performance of SVM.

In our experiments, the data set consists of 1813 samples split into 1226 malicious and 587 benign executables. Malicious executables are provided by ANTIY laboratory. Benign executables are collected from the Internet and system directory, for benign executables, they are firstly

scanned by Kaspersky and Rising to eliminate hidden malicious ones. We use 30% of samples as testing data set and 70% as training data set. The detail experiment result shows in Table 1.

TABLE 1. EXPERIMENTAL RESULT OF DETECTION SYSTEM

C	$1/\sigma^2$	FN(%)	FP(%)
50	10	5.46	6.13
100	1	6.53	7.32
200	0.5	7.78	8.43

In order to verify the useful and robust of our system call tracing toward the different malware samples in the wild. We compare the system call trace report between Argus [10], CWSandbox [5], Anubis [6] which are popular system call tools over the world. We submit the same 1250 malware samples for these tools, and the return number of Useful tracing result show in Table 2. We define the "Useful" means that the trace result must contain at least one actual interaction with operator system. The result of this compare also show that the malware author keep on studying the new behavior analysis tool and try to find new way to anti-analysis. Our system traces 1226 malware samples correctly. It performs best and show that full virtualization is a effective technology on the tracing of malware behavior.

TABLE 2. COMPARE RESULT OF SYSTEM CALL TRACING SYSTEM

Trace platform	**Argus**	**CWSandbox**	**Anubis**	**Our system**
number	534	788	1125	1226
Useful rate	43 %	63 %	90 %	98%

V. CONCLUSION AND FUTURE WORK

In this paper we proposed our novel system call tracing system based on full virtualization. We also show the strong resilience of our system call tracing system through the comparison with other trace tools. The result of our experiment based SVM has shown that our method can detect malware with high accuracy.

In our trace system we only get system call sequence and ignore system call parameter such as which file is created with NTCreateFile. This issue must be met in the future work.

REFERENCES

[1] J. Xu, A.H. Sung, P. Chavez, S. Mukkamala, "Polymorphic malicious executable scanner by API sequence analysis" in 4th International Conference on Hybrid Intelligent Systems (HIS), 2004, pp. 378–383

[2] VALSMITH, AND QUIST, D. Hacking Malware: Offense is the new Defense. Defcon 14, August 2006.www.offensivecomputing.net/dc14.

[3] RUTKOWSKA, J. Red Pill: Detect VMM using (almost) One CPU Instruction, November 2004. http://invisiblethings.org/papers/redpill.html.

[4] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data mining methods for detection of new malicious executables. In Proceedings of the IEEE Symposium on Security and Privacy, pages 38{49, Los Alamitos, CA, 2001. IEEE Press.

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

[6] U.Bayer, C.Kruegel, and E.Kirda. TTanalyze: A Tool for Analyzing Malware. In 15th Annual Conference of the European Institute for Computer Antivirus Research, Hamburg, Germany, 2006: 180−192.

[7] 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: 191-206.

[8] 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.

[9] J. Han, M. Kamber, Data Mining : Concepts and Techniques, Morgan Kaufmann, August 2000

[10] Yongtao Hu Unknown Malicious Executables Detection Based on Run-Time Behavior In Fuzzy Systems and Knowledge Discovery, 2008 pp. 391-395.

[11] Zhang, B; Yin, J; Hao, J; Zhang, D; Wang, S. Using Support Vector Machine to Detect Unknown Computer Viruses, International Journal of Computational Intelligence Research, Vol. 2, No. 1, 2006, pp. 100 – 104

[12] Lee,W., Dong,X.: Information-Theoretic measures for anomaly detection. In: Needham,R., Abadi M, (eds):Proceedings of the 2001 IEEE Symposium on Security and Privacy. Oakland, CA: IEEE Computer Society Press (2001)130-143.