Ensuring Kernel Integrity Using KIPBMFH

Zhifeng Chen(✉), Qingbao Li, Songhui Guo, and Ye Wang

State Key Laboratory of Mathematical Engineering and
Advanced Computing, Zhengzhou, China
xiaohouzi060123@gmail.com

Abstract. Kernel-level malwares are a serious threat to the integrity and security of the operating system. Current kernel integrity measurement methods have one-sidedness in selecting the measurement objects, and the characters of periodic measurement make TOC-TOU attacks unavoidable. The kernel integrity measurement methods based on hardware usually suffer high cost due to the additional hardware, while the kernel integrity measurement methods based on host are always likely to be passed. To address these problems, a kernel integrity protection approach based on memory forensics technique implemented in Hypervisor (KIPBMFH) is proposed in this paper. We first use memory forensics technology to extract the static and dynamic measurement objects, and then adopt time randomization algorithm to weaken TOC-TOU attacks. The experimental results show that KIPBMFH can measure the integrity of the operating system effectively, and has reasonable performance overhead.

Keywords: Kernel integrity · TOC-TOU · Memory forensics · Time randomization · Hypervisor

1 Introduction

The security of operating systems (OS) is one of the most important issues of computer security. As the kernel is the basic component of OS, its security impacts heavily on the security of OS.

Integrity measurement technique provides a comprehensive description of the system security through the measurement, verification and evaluation of the coincidence of the expectant and actual situation.

Current kernel integrity measurement methods can be divided into three categories according to their deployment levels, which are host-based, hardware-based and hypervisor-based, respectively.

The host-based methods [1–3] are most widely used. They mainly measure the static kernel objects such as system call table, which do not consider the dynamic object, cannot find the data-only attacks, and the use of periodic analysis techniques make them exist TOC-TOU (Time-of-Check to Time-of-Use) problem [4]. Besides, the objects of their methods are also limited.

The hardware-based methods [5, 6] solidify the detection tools at hardware level, which are isolated with the objects to be detected. They maybe tamper-resistant and

S. Qing et al. (Eds.): ICICS 2015, LNCS 9543, pp. 10–17, 2016.
DOI: 10.1007/978-3-319-29814-6_2

non-bypass, but due to the need for additional hardware support and high cost, they have not been widely used. In addition, they also have the TOC-TOU problem.

Given the isolation, insight and safety advantages of virtual machine technology, it has been used to measure and protect the integrity of kernel in recent years. The hypervisor-based methods [8–10] deploy the measurement tools in the hypervisor layer, which improve the safety of measurement tools to some extent. They bring additional load and performance loss, and only focus on measurement objects related to the process, do not consider other objects, such as modules, network connections, etc., resulting in inaccurate measurement results. The reasons are two-folded. One is the introduction of the intervention of virtual monitoring; the other is the lack of adequate upper semantic information.

Therefore, to solve these problems, a Kernel Integrity Protection Method Based on Memory Forensics in Hypervisor (KIPBMFH) is proposed, which is deployed in the Hypervisor layer. This method extracts the measurement objects in kernel memory space based on forensics technology. It uses lightweight hypervisor Bitvisor, which thus reduces the system cost and performance loss. This method also uses randomization time-triggered measurements, which makes extraction, analysis and measurement not follow the periodic rule any longer. So it can avoid the attacker to control the measurement rule, and weaken TOC-TOU attacks.

2 Problem Overview

The fundamental problem of the integrity measurement is obtaining measurement objects accurately, timely and comprehensively. However, existing kernel integrity measurement methods have one-sidedness in obtaining the objects. So to solve this key issue is important for improving the effectiveness of the integrity measurement.

Memory forensics technology is one of the currently popular digital forensics analysis techniques, which focuses on obtaining information from the computer memory, such as processes, files network connection status and other activities. It provides large amounts of information of system running status about network intrusion.

As integrity measurement objects exist in memory, the basic idea of memory forensics technology can be used to obtain the measurement objects. However, the memory forensics technology does not involve related discussions on the measurement objects. In addition, the memory forensics technology only simply provides a mechanism; it does not discuss how to set the measurement points and how to measure.

Therefore, applying the memory forensics technology on the kernel integrity measurement needs to address the following issues. Firstly, the memory forensics provides a method of extracting the measurement objects, but which measurement objects in memory to be obtained and how to get the measurement objects in the Hypervisor are the most important issue. Secondly, attackers can bypass the periodical integrity measurement to achieve attacks. So how to distribute the measurement points reasonably to reduce the possibility of escaping measurement without the use of real-time monitoring is another difficulty issue to be addressed.

3 Kernel Integrity Protection

3.1 Memory Forensics Based on EPT

Random Memory Extraction. Memory forensics methods to obtain physical memory contents can be divided into hardware-based and software-based methods [11]. These methods copy the entire contents of the physical memory to the storage devices. With the increase of physical memory, the time overhead grows fast. In fact, for a running operating system, the kernel space of physical memory is limited, and kernel memory analysis is also targeted to extract specific measurement objects, so dumping the entire contents of the memory to the external storage device is unnecessary.

Since the measurement objects are distributed in memory discretely, to overcome the deficiencies of the existing memory extraction method to realize on-demand extraction is to design a random extraction method. However, the address received by Hypervisor is the guest physical address, and access memory through host physical address. While the guest VM accesses the memory through guest virtual address. So to achieve memory access in Hypervisor has to design an algorithm which translates the guest virtual address into a host physical address. Therefore, we first need to do address translation, which translates the guest virtual address into the guest physical address, and then translates the guest physical address into the host physical address. Finally, we achieve access to the host physical memory in Hypervisor, thereby extracting arbitrary physical memory contents. In this paper, we take the Intel processor as the experimental environment. So we give the address conversion method based on the EPT as example.

Selectable PMCE (Physical Memory Contents Extraction) is designed and implemented according to the above ideal. Selectable extraction means that users can choose any address or address scope to get memory contents. This algorithm completes the memory randomly access and memory contents extraction, which solves the mapping problem of physical address and logical address. The algorithm is shown in Table 1, step 2–3 of the algorithm are corresponding to the address mapping process.

Kernel Memory Analysis. The memory contents extracted by PMCE are formatted with binary, so the next step is to identify the measurement objects from the contents. While the best solution is that determine the objects which influent the integrity of running kernel.

(1) determining the measurement objects. We determine the measurement objects according to the basic composition and running mechanism of kernel, and attacked objects. As we know, the objects like kernel code section, Interrupt Description Table(IDT), system call table(SCT), etc., have persistent values during the kernel running. Once they are destroyed, the expectation behavior of system will change, which means the integrity of kernel is destroyed. As they are persistent, we call them the static measurement objects. In addition, there are dynamically changed objects, such as processes, modules, network connection, and so on, we call them the dynamic measurement objects. Although they change dynamically, they have certain characteristics.

Table 1. Physical memory contents extraction algorithm

```
Algorithm 1. PMCE
Input: u32 beginaddress, u32 endaddress
Output: char[] MC//physical memory contents
len=endaddress-beginaddress;
gphy=gvtogp(beginaddress);//translate VM virtual address into VM physical address
hphy=gptohp(gphy);//translate VM physical address into host physical address based on EPT
while(len>0)
    MC[i]=read_hpys_b(hphy);//access physical memory contents after memory map
    len--;hpy++;i++;
return MC;
Function: u32 gvtogp(u32 va)
u32 cr3=get_guest_cr3();//guest VM CR3
u32 gpgde =gptohp(cr3) +8*va>>39; //pgd
u32 hpgde=gptohp(gpgde);
u32 gpude=readaddress(hpgde) &0xfffff000+8*(va>>30&0x1ff);//pud
u32 hpude=gptohp(gpude);
u32 gpmde=readaddress(hpude) &0xfffff000+8*(va>>21&0x1ff);//pmd
u32 hpmde=gptohp(gpmde);
u32 gpte=readaddress(hpmde) &0xfffff000+8*(va>>12&0x1ff);//pt
u32 hpte=gptohp(gpte);
u32 gpa=readaddress(hpte) &0xfffff000+8*(va&0x1ff);//pa
return gpa;
```

For example, the processes in Linux satisfy the relation: run_list⊆all-tasks or all-tasks==ps-tasks, where run_list represents the schedule processes, all_tasks represents the whole processes of system, ps-tasks represents the processes got by system tools.

Based on the above analysis, the measurement objects can be divided into static measurement objects and dynamic objects. They distribute in different memory positions, so next we will discuss how to reconstruct them from the memory.

(2) analyzing the measurement objects. The kernel symbol table of operating system is the entry point for memory analysis. The kernel symbol table stores the memory address of key data objects. Linux kernel symbol table names System.map, which stores thousands of symbol addresses, but we only concern with the contents related to integrity measurement. In addition, the definition of critical kernel data structures and offset of each field are also the important factors for memory analysis.

Now we take the analysis of the Linux system process analysis as an example to show the basic idea of memory analysis. Process information is stored in the task_struct structure, all of the processes are linked by double-linked list, shown in Fig. 1. Process #0 is the parent of all other processes, which stays in memory forever. So the analysis begins with Process #0. First, translate the logical address of Process #0 in System.map into host physical address, and then extract the contents of this process in memory by Hypervisor, and reconstruct PID, running status, process name and so on assisted by data structures definition of the operating system. Next, take Process #0 as the starting point, and then analyze the other processes through the double-link field of the process structure.

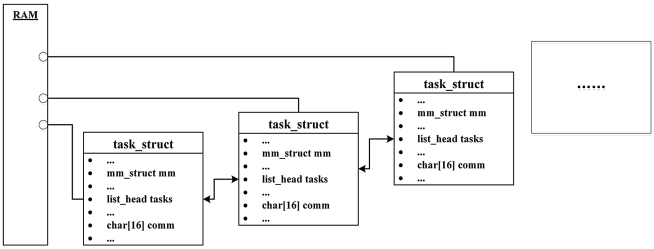

Fig. 1. Process structure and its organization on memory

In summary, when analyzing physical memory, we extract memory addresses of the measurement objects based on the kernel symbol table, and then reconstruct the measurement objects according the relation of addresses and critical kernel data structures.

3.2 Time Randomization Based Measurement Time Distribution Method

Due to the presence of TOC-TOU vulnerability of periodical measurement, attackers can bypass the integrity measurement by using this vulnerability. We set the value of the interval for T. The attackers can break the rule if they find out this vulnerability. Therefore, in order to avoid an attacker to grasp the rule of integrity measurement, we propose time randomization measurement method, which distributes the measure operations randomly into the time domain. However, some randomized algorithms may lead to measurement time points too dense or too sparse. The former will bring the system load increasing, while the latter will lead to measurement accuracy decline, so the distribution of measurement points cannot be arbitrary randomization.

According to the impact on system load caused by measurement time sequences of different time ranges and the measurement accuracy, we define the random seed in the [T/3, 6*T/5] range. This practice ensures that the distributions of measurement points meet the requirements, but also ensures that the measurement time is random. The specific process is as follows.

(1) Generate the seed s_seed at a given time range [T/3, 6*T/5];
(2) Select a largest prime number between 1 and n and an additional number of multiplier adder. And then calculate the value of s_time by formula (((multiplier*s_seed+adder)≫16)% (6*T/5)). If s_time is less than T/3, then the value of s_time should plus T/3 to obtain a time point RS. otherwise s_time is the final time point.
(3) Repeat step (2) until the number of time points is equal to n, and then finish the generation of time sequence.
(4) Synchronize the system time with the time sequences. Take the current system time as the first measure point, and then make each time point of the time sequence accumulate the system time in turn.

4 Experimental Results and Analysis

In this section, we provide the effectiveness and performance evaluation of KIPBMFH. Experimental environment is as follows.

Guest operating system is Ubuntu 12.04-x86, kernel version is 3.2.43, CPU is Intel (R) Core(TM) i5-750 @ 2.67 GHz, memory size is 4 GB. Using Bitvisor as Hypervisor, whose version is 1.4. Select rootkit, such as kbeast, enyelkm and suckit etc. as a test suit.

4.1 Effectiveness Evaluation

we verify the effectiveness of KIPBMFH through a variety of different rootkits, which use different methods to perform attacks. Table 2 shows the integrity measurement results comparison of KIPBMFH and other measurement tools in a variety of rootkit attacks, where "√" indicates a successful measurement, "×" indicates failure to measure the integrity of the system, "—" indicates that the system does not test this rootkit.

Table 2 shows that KIPBMFH successfully finds out variety of rootkit attacks, while Copilot [5], Osck [8] and Gobraltar [6] only find out part of rootkit attacks. In conclusion, the measurement accuracy of KIPBMFH is higher than the existing measurement tools.

Table 2. KIPBMFH and other tools' measurement for kinds of rootkit attacks

Rootkit	Attack level	KIPBMFH	Copilot	Osck	Gibraltar
ddrk	User, Kernel	√	—	—	—
wnps v0.26	Kernel	√	√	√	√
enyelkm v1.2	Kernel	√	√	√	√
adore-ng 0.56	Kernel	√	√	√	√
allroot	Kernel	√	√	√	√
mood-nt	Kernel	√	—	—	—
Knark-2.4.3	Kernel	√	√	—	√
suckit v2.0	Kernel	√	√	—	—
lrk v5	Kernel	√	×	×	×

In addition, KIPBMFH not only can discover attacks, but also can get more fine-grain attack information, such as the number of system call, system call name, hidden processes information, the hidden network connection information, and so on. So KIPBMFH has a stronger analytical capability than other measurement tools.

4.2 Performance Evaluation

In this section, we take integrity measurement time as performance index to evaluate the efficiency of KIPBMFH. Measurement time refers to the time from the memory forensic analysis to obtain the integrity measurement results.

In addition, measurement time has a guiding role to determine the period time T. Integrity measurement time includes the time of physical memory analysis and integrity measurement. Table 3 shows the time and the total time required for each part. The measurement time is approximately 161.05544 ms.

Table 3. Time cost of part of measure process

	Physical memory analysis	Integrity measurement	All
Time (ms)	159.45004	1.6054	161.05544

In addition, as KIPBMFH needs to read memory to measure the guest, we use Stream to test the memory bandwidth of guest OS. The evaluation shows that the system load of KIPBMFH is only 0.18 % in the case of 15 s. Compared to 2 % – 6 % of Osck and 3.6 % – 48.3 % of SBCFI [7], the system load of KIPBMFH is negligible. Compared to 0.49 % of Gibraltar and 0.84 % of Copilot, KIPBMFH not only has small load, but also provide certain security. Therefore, KIPBMFH has certain advantages in time overhead and system load comparing with these measurement tools.

5 Conclusion

In this paper, KIPBMFH is proposed aiming at address the overhead, TOC-TOU problem and one-sidedness of measurement objects of the existing measurement methods. Compared with the periodical measurement tools, the measure ability and accuracy of KIPBMFH is higher.

References

1. Wang, Y.M., Beck, D., et al.: Detecting stealth software with strider ghostbuster. In: Proceedings of the International Conference on Dependable Systems and Networks (2005)
2. Joy, J., John, A.: A host based kernel level rootkit detection mechanism using clustering technique. In: Nagamalai, D., Renault, E., Dhanuskodi, M. (eds.) CCSEIT 2011. CCIS, vol. 204, pp. 564–570. Springer, Heidelberg (2011)
3. Liu, Z.W., Feng, D.G.: TPM-based dynamic integrity measurement architecture. J. Electron. Inf. Technol. **32**(4), 875–879 (2010)
4. Bratus, S., D'Cunha, N., Sparks, E., Smith, S.W.: TOCTOU, traps, and trusted computing. In: Lipp, P., Sadeghi, A.-R., Koch, K.-M. (eds.) Trust 2008. LNCS, vol. 4968, pp. 14–32. Springer, Heidelberg (2008)
5. Petroni, Jr. N.L., Fraser, et al.: Copilot - a coprocessor-based kernel runtime integrity monitor. In: Proceedings of the 13th Conference on USENIX Security Symposium (2004)
6. Baliga, A., Ganapathy, V., Iftode, L.: Automatic inference and enforcement of kernel data structure invariants. IEEE Trans. Dependable Secure Comput. **8**(5), 670–684 (2011)

7. Petroni Jr. N.L., Hicks, M.: Automated detection of persistent kernel control-flow attacks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security (2007)
8. Hofmann, O.S., Dunn, A.M., et al.: Ensuring operating system kernel integrity with OSck. In: Proceedings of the 6th International Conference on Architectural Support For Programming Languages and Operating Systems (2011)
9. Li, B., Wo, T.Y., et al.: Hidden OS objects correlated detection technology based on VMM. J. Softw. **24**(2), 405–420 (2013). (in Chinese)
10. Lin, J., Liu, C.Y., Fang, B.X.: IVirt runtime environment integrity measurement mechanism based on virtual machine introspection. Chin. J. Comput. **38**(1), 191–203 (2015). (in Chinese)
11. Carvey, H.: Windows Forensic Analysis and DVD Toolkit, pp. 59–63. Elsevier: Syngress, Burlington (2009)

http://www.springer.com/978-3-319-29813-9

Information and Communications Security
17th International Conference, ICICS 2015, Beijing, China, December 9-11, 2015, Revised Selected Papers
Qing, S.; Okamoto, E.; Kim, K.; Liu, D. (Eds.)
2016, XVIII, 486 p. 133 illus. in color., Softcover
ISBN: 978-3-319-29813-9