Memory Access Monitoring and Disguising of Process Information to Avoid Attacks to Essential Services

Masaya Sato, Toshihiro Yamauchi, Hideo Taniguchi
Graduate School of Natural Science and Technology,
Okayama University, Okayama, Japan
Email: {sato,yamauchi,tani}@cs.okayama-u.ac.jp

***Abstract*—To prevent attacks on essential software and to mitigate damage, an attack avoiding method that complicates process identification from attackers is proposed. This method complicates the identification of essential services by replacing process information with dummy information. However, this method allows attackers to identify essential processes by detecting changes in process information. To address this problems and provide more complexity to process identification, this paper proposes a memory access monitoring by using a virtual machine monitor. By manipulating the page access permission, a virtual machine monitor detects page access, which includes process information, and replaces it with dummy information. This paper presents the design, implementation, and evaluation of the proposed method.**

***Keywords*-attack avoidance; process information; virtualization**

I. Introduction

The prevention of attacks to computers is an important research topic, for which protective software is developed. In addition to security software, logging and analyzing techniques are developed. We call these as essential services or software, which are necessary to computers in the current malware-spread situation; hence, the essential software becomes a target of attack from malware. Some malware have a function to terminate or deactivate essential software [1], [2], [3]. Min et al. proposed novel malware to subvert the self-protection function of antiviruses [4], whereas the current malware has advanced functionalities to terminate or disable antivirus software. If an attacker terminates or disables essential services, the computer is possibly taken control over by malware, or damage may increase. Thus, it is necessary to prevent attacks on essential services.

Some methods to prevent attacks to essential services were proposed in [5], [6], [7], [8] and [9]. Hsu et al. proposed antivirus software protection based on a kernel mode driver [5]. However, kernel mode countermeasures are vulnerable to kernel-level attacks. To address this problem, virtual machine monitor (VMM) based security mechanisms were proposed in [6], [7], [8], [9]. As a VMM is isolated from guest operating systems (OS), attacking a VMM is more difficult than attacking an OS. These approaches utilize the property of a VMM and prove that the VMM-based security mechanism have enough adequate security than an OS-based security mechanism. Virtualization technology is not limited to the abovementioned methods but is used for security monitoring [10], [11] because of its isolation property. However, the use of the existing approaches requires modification to essential services. From these observations, we determined that the security of essential services with no modifications to that software is an important challenge. However, the existing approaches do not satisfy these requirements.

To address this problem, we proposed a process-hiding method [12], which replaces process information of essential services with dummy information to avoid identification of the essential services. To protect the replacement mechanism from kernel-level attacks, a VMM replaces process information of each VM. However, the use of this replacement method may possibly allow attackers to identify essential services by continuously monitoring all process information. As the replacement method replaces process information of an essential service during context switching, an attacker can detect the essential process if he/she detects a change in process information at that time.

To address these problems, this paper proposes the memory access monitoring and disguising of process information to avoid attacks to essential services. To complicate identification of an essential service through continuous monitoring at kernel level, our method control accesses to process information of an essential service. In our method, only a permitted code can access process information. If another code accesses the process information of an essential service, the VMM returns dummy information to the source of access. We used a hardware-assisted paging mechanism to control and to monitor page access, which includes the process information of essential services. By removing read access to the pages, all access to that page causes transition from the VM to the VMM. Thus, we can monitor and control the access to those pages. The proposed method does not require modification to existing essential services. In addition, the proposed method is applicable to various application programs because it depends not on application programs but on the process information managed by an operating system.

II. Background

A. Attacks for Anti-Virus Software

Agobot [1] is malware that attacks anti-virus software. Agobot installs backdoor to Windows hosts. The malware seeks target processes by searching out the name from the process list in order to disable it. An investigation on August 8th, 2013, revealed that Agobot included 579 targeted process names. When malware disables anti-virus software, the risk of damage to the computer system increases.

T0rnkit [2] and dica [3] are malware for disabling a logging program. T0rnkit is a rootkit that aims to install a backdoor for concealing their location. Its target system is Linux. When installing programs used by t0rnkit, the malware stops the syslog daemon, thus hiding the installation process from a system administrator. Consequently, the system administrator cannot detect the installation or even the existence of other malware.

Some malware stops or disables software that prohibits their activity on the computer. If essential services are stopped or disabled, the risk of damage to the system increases. For this reason, detection, prevention, and attack avoidance for an essential service are required.

B. Existing Countermeasures

Hsu et al. proposed a protection method called ANSS for anti-virus software from termination [5]. ANSS hooks system calls related to process termination and control them if they would stop anti-virus software. ANSS works as a Windows driver that operates in kernel mode.

Virtual machine introspection [6] is the representative approach using virtual machines. Existing approaches monitors information of computers with user application programs and kernels. However, those methods are vulnerable to kernel-level attacks. Kernel-level attacks are emerging threats in current situation and those attacks are not negligible. However, VMI export the existing monitors to outside a VM. Because of strong isolation of VM, the monitors are secure than the monitor inside the kernel. Research into an offloading host-based intrusion detection system (IDS) with a VMM is proposed in VMwatcher [7] Implementing an IDS by modifying a VMM makes it difficult to attack the IDS. In the same manner, Riley et al. proposed NICKLE [8], which prevents the execution of a kernel-level rootkit. Because it monitors the execution of kernel code with a VMM, only the authorized code can run. These methods help to prevent attacks that are difficult for existing methods without a VMM to detect and prevent.

To address these problems, we proposed a process hiding method by replacing process information [12]. This method replaces process information of an essential process to dummy information to hide the essential process from attackers. This method addresses the above problems because it complicates identification of an essential process and it is implemented by a VMM. Thus, attacks to the essential service based on a process information becomes difficult and the mechanism itself is secure because isolation from attackers' codes on a VM. In addition, the process hiding method requires no modification to essential services.

C. Problems with Existing Methods

Existing methods cannot use essential services without modifying them. Furthermore, these methods are effective only when they are secure from attacks. If these methods are themselves attacked by adversaries, a system administrator cannot utilize those services to avoid attack. The methods described in Section II-B are advantageous given that attacks on a VMM are more difficult than attacks on an OS. However, porting the functions from existing software to a VMM is difficult and expensive. The IDS offload method without modification is an effective approach. However, it is difficult to apply to general application because the method involves the emulation of each system call. To offload the IDS completely, it is necessary to emulate all system calls. However, complete emulation is difficult to implement.

Even though the existing VMM-based methods are isolated from attackers on a VM, many of them cannot use existing software without modification. Moreover, exporting existing functions used by anti-virus software to a VMM is difficult. Further, the information collected by existing application programs (APs) and kernel is different from the information a VMM collects. This semantic gap makes it difficult to port functions from existing software to a VMM. The process hiding method stated in the Section II-B addresses these problems, however, it is vulnerable to attacks that monitors process information of all processes continuously. This attack can detect the essential service when the attacker detects changes to process information. This method does not prohibit termination of essential processes so that the attacker can terminate them. To avoid attacks to essential services, this problem must be addressed.

III. Memory Access Monitoring and Disguising of Process Information

A. Basic Idea

In the proposed method, process information is defined as information that can be used to identify a process. The proposed method monitors accesses to and disguises the process information depending on the source of access. Section III-B details the disguising flow. By using this method, only the essential service can view the original process information, whereas other processes and untrusted kernel modules view dummy process information.

We designed the proposed method without modifying an essential service's program. By utilizing a VMM, we can design the proposed method without modifying the APs. Because the VMM manages resources of each VM, the VMM can monitor their memory accesses. Moreover, a

design with a VMM makes it difficult to attack the security mechanism even if an attacker obtains kernel privilege in a VM. Because a VMM and VM are isolated, high privilege in a VM does not allow attacks to the proposed method in the VMM or another VM.

The proposed method addresses the limitations of our previous study [12] by monitoring accesses to process information. In our previous work, process information was replaced by dummy information. However, the original information was restored while the essential service was running, thus allowing attackers to detect it. In contrast, the proposed method can prevent detection by monitoring access to the process information. Even if the process information is restored, we can detect an access to it and handle it as attackers cannot view the original information.

B. Monitoring Accesses to Process Information of Essential Services

The proposed method prohibits reading access to process information. When some programs read the process information of an essential service on the VM, EPT violation occurs and the VMM manages it. The VMM checks whether the source of access is included in the permitted area; if yes, the VMM permits read access to the accessed area and returns to processing the VM. In other cases, the VMM saves the original process information to another VM, replaces the process information, and returns processing to the VM. By replacing the process information, the source in the not-permitted area views the dummy process information.

Table I shows who can view the process information. In the proposed method, a kernel and permitted kernel modules can view the original process information. However, other kernel modules cannot view the original information. We assume a kernel manages the process information but we do not consider the codes in user space. To check the source of access, we use an instruction pointer and a kernel stack. As a kernel stack consists of return addresses, we collect them and check whether they are in the permitted area.

We use an extended page table (EPT) to monitor accesses to process information. Each EPT entry has a permission vector including read, write, and execute. The proposed method disables the read bit of an EPT entry, which comprises process information of an essential service. In this situation, a read access to a page corresponding to the EPT entry causes an EPT violation. By detecting the EPT violation, the VMM can detect the read access to the process information of an essential services.

C. Process Information to Monitor

The followings are the process information in Linux:

1) Process control block
2) Kernel stack
3) Hardware context
4) Page tables

Table I
ACCESS PERMISSIONS FOR PROCESS INFORMATION OF ESSENTIAL SERVICES

	Source of Access
Permitted	• Operating System Kernel • Permitted Kernel Module
Prohibited	• Prohibited Kernel Module

5) Memory used by a user process

The process control block has many resources and pointers related to the process including process ID, thread ID, process name, and pointers to data structures related to files and sockets used by the process. It is easy to identify a process as essential by referring to its process control block.

A kernel stack has a call stack, which include information of function call and local variables. In multi-core environment, a process running on a core can estimate an essential process running on another core by analyzing the kernel stack. In addition, a data structure that controls a thread is accessible from the kernel stack because the data structure of the kernel stack is a union of the kernel stack and thread control data. An attacker can identify an essential process by referring to its thread control data.

A hardware context has register values, which may utilized to estimate the behavior of a process. Further, it is possible to estimate the behavior by analyzing the memory usage. Thus, page tables are also considered as process information.

The memory content used by a process is unique and contains much process-related information. Attackers are able to identify the behavior of a process by analyzing the memory content if they have sufficient knowledge of the application program.

Therefore, controlling accesses to the process control block, kernel stack, and memory content used by the process is necessary to prevent identification of an essential process. In this paper, we focus on the process control block. Even if access to a kernel stack must be controlled, it can be acquired by applying the same method used for the process control block. However, we did not handle the memory content used by a process because monitoring all accesses to the memory content requires a large performance overhead. We considered using some memory protection techniques; however, it is outside the scope of this study.

D. Allocation of Process Information

Figure 1 shows the allocation methods of process information. In the proposed method EPT restricts reads to the information of essential processes. Therefore, granularity of access control is the page size. As process control blocks are not allocated per-page, we modified the guest OS to allocate process control blocks per-page and the remaining area is filled by padding.

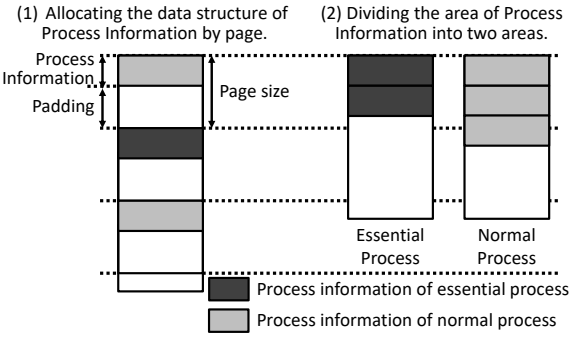

Figure 1. Allocation of process information.

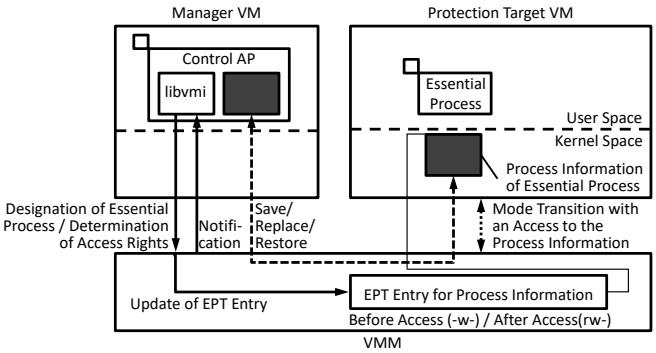

Figure 2. Overview of the proposed method.

E. Overview

Figure 2 shows the overview of the proposed method. The administrator of the Manager VM designates the essential process. The proposed method disables read access to an EPT entry corresponding to a page that includes information of the essential process. Figure 2 shows the case when the process control block of the essential process is designated as process information. Here, only write accesses to the page are permitted before a read access. After detecting the read access to the page, the Control AP requests the VMM to permit read access. When a prohibited code accesses the process information, the Control AP replaces the process information with a dummy information. To prevent attackers from distinguishing the dummy information, it must be chosen carefully. In most cases, the dummy information should be chosen randomly with values common to many programs. Adequate dummy information is written in our previous study [12].

F. Consideration

1) Identification of Essential Process from File or Communication Information: To identify an essential process, attackers can use information related to file or communication. For example, if an essential service accesses a specific file, attackers can identify the essential process by monitoring the access to the specific file.

Although that information is beneficial to attackers, the proposed method in this paper provides measurable complication for identification. Usually, information about file or communication is acquired through a process control block. Because the proposed method monitors accesses to a process control block, the VMM and the control AP can monitor accesses to file or communication information. When the VMM or the control AP detects accesses to them, we can replace or disguise them as if the process is not using a specific file or not communicating with a specific computer.

2) Management of Essential Processes on Protection Target VM: Information of an essential process is invisible to all the other processes except itself. This indicates that the manager of the protection target VM cannot manage the essential process. We do not consider this inconvenience a problem because we assume an essential process is a resident service, which is not controlled by the manager of a computer frequently. The manager of the protection target VM can manage the essential process by requesting the manager of the manager VM.

3) Application to Other Purposes: The proposed method can be used for other purposes. For example, the proposed method can be used as a countermeasure for the anti-analysis function of malware. Some malware stops their activities to avoid being analyzed if they find a malware analysis environment. By applying the proposed method to a malware analysis software may improve its success rate.

IV. Implementation

A. Save and restoration of process information

Figure 3 shows the flow when a code access to the prohibited area. When a read access to a page that is not permitted to read access, a VMM is called with EPT violation. In the prototype of the proposed method, the VMM calls the Control AP in the Manager VM. The Control AP checks the permitted area includes the instruction pointer or not. If the permitted area includes the instruction pointer, the Control AP scans a kernel stack of the Protection Target VM and collects return addresses. If the permitted area includes the instruction pointer and all return addresses in a kernel stack are comprised in the permitted area, the Control AP requests the VMM to enable read access to the page. If not, the Control AP saves the original process information and replaces it to a dummy information. After the replacement, the Control AP requests the VMM to enable read access to the page and enables single step mode.

Figure 4 shows the flow when an instruction is executed while the guest OS is in single step mode. After the VMM is called, the VMM transfer control to the Control AP. The control AP first disables single step mode. If the control AP replaced the process information with a dummy information, it restores the original process information. Finally, the Control AP requests the VMM to disable read access to

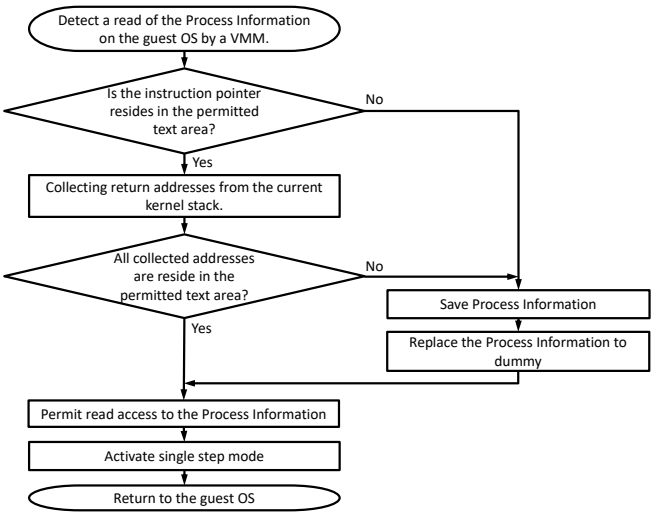

Figure 3. Flow when a code access to the prohibited area.

the page to detect the next read access to the process information.

B. Collecting Process Control Block

If the VMM detects a read access to the process information of the essential process from the area which is not permitted to access, it is required to collect process control block of the kernel running on the VM from the outside. In our method, the VMM collects the process control block from register values of the VM. Because a process control block and a kernel stack are linked each other, the VMM first gets the initial address of the kernel stack. Then the VMM gets the initial address of a process control block from a member in the kernel stack that points the process control block. The VMM gets the initial address of the kernel stack by masking lower 13 bits of the stack pointer because two pages are allocated to a kernel stack in Linux. Because the initial address of the kernel stack is also the initial address of `thread_info`, the `thread_info` has a member which point `task_struct` (process control block). With the above procedure, the VMM can collect and replace process control block. Thus, the VMM must know the definition of those data structures beforehand.

V. Evaluation

A. Purpose and Environment

To confirm that the proposed method can replace the process information and untrusted kernel modules cannot access the original process information, we listed processes from the user and kernel spaces inside the protection target VM. Further, to estimate the performance overhead in application programs, we measured the performance by using a micro-benchmark program.

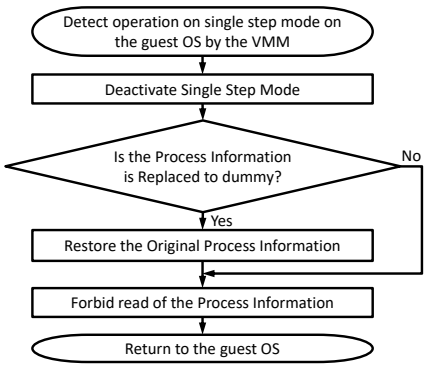

Figure 4. Flow when an instruction is executed while the guest OS is in single step mode.

Table II shows the environment used for evaluation. We implemented the proposed method with the Xen hypervisor [15] and Linux. We used LibVMI [13], which is a library based on XenAccess [14], to implement the proposed method. It is made for virtual machine introspection compatible with Xen. We did not modify Xen, and wrote a control AP on the manager VM. In the prototype, we used a hardware-based VM (HVM) domain as the protection target VM and dom0 as the manager VM. The computer used for evaluation comprises Intel Core i7-2600 (3.4 GHz) and 16 GB RAM. The protection target VM has one virtual CPU and 1 GB RAM. The manager VM comprises three virtual CPUs and 15 GB RAM. Hyper threading and turbo boost are disabled to suppress performance instability, and VT-x and EPT are enabled. Linux 3.2.65 and 3.2.0 run on the protection target and manager VMs.

B. Avoidance of Process Identification

To confirm that the proposed method can avoid attacks to an essential process according to its name of the process, we used malware *dica*, which is a rootkit for Linux and stops a logging daemon *syslogd* to hide the installation of another malware. *dica* uses the `killall` command to terminate syslogd. `killall` scans the program name of all the running processes. If a program name matches the target name, the `killall` command terminates the

Table II
Environment for evaluation.

Software	
VMM	Xen 4.2.3
OS	Manager VM: Debian 7.3 (Linux 3.2.0 64-bit)
	Protection Target VM: Debian 7.3 (Linux 3.2.65 64-bit)
Control AP	LibVMI 0.10.1
Hardware	
CPU	Intel Core i7-2600, 3.40 GHz, 4 cores
	Manager VM: 1 virtual CPU
	Protection Target VM: 1 virtual CPU
Memory	Manager VM: 15 GB
	Protection Target VM: 1 GB

process. Because the program name is a member (`comm`) of `task_struct`, we designated the `comm` member in `task_struct` as process information. The experimental results showed that if the proposed method is used, *syslogd* keeps running after *dica* is installed to the target VM. This is because the comm member of *syslogd* is replaced with a dummy name through the proposed method and `killall` is unnable to find *syslogd* from the process list of the VM. This experiment confirmed that the proposed method is effective for attacks based on process information.

C. Performance

We measured the processing time during illegal access to process information. We compared the processing time when the read access to the information of an essential process is granted or prohibited. We used a kernel module on the VM to collect a list of processes.

Figure 5 shows the measurement results when the process information is allowed to be read, prohibited to be read, and prohibited to be read and replaced with a dummy information. Table III shows the mean processing time in each case.

We measured the processing time for collecting a process list hundred times in three cases: no restriction for reading process information, restriction without replacement of process information, and restriction with replacement of process information. In the replacement case, we replaced the name of a program of the process. The kernel module reads a process list in a kernel from the beginning to the end. This causes EPT violation when a program reads the process information of an essential process. The control AP detects the EPT violation and determines whether the process information must be replaced. An idle process is a user process that repeats sleep for a second.

The measurement results showed that the performance overhead with read access restriction to process information is 0.024 (8%) s in the `init` and 0.034 s (11%) in the `idle`. The processing time during the replacement of process information increases by 0.154 s (45%) in the `init` and decreases by 0.003 s (1%) in the `idle`.

Compared to the method using replacement of process information [12], the proposed method further will degrades the performance. This is because the proposed method monitors read access to the process information in addition to replacing it. However, we believe the earlier measurement results show that its overheads are acceptable. The workload largely degrades the performance because it accesses process information frequently. In contrast, common workloads do not access process information much. Therefore, the performance overheads in the actual workload can be expected to be less than the aforementioned results.

D. Memory Consumption

As the proposed method requires changes to the size of process information, we calculated the memory overhead by using the proposed method. The increased memory size depends on the number of processes, size of the data structure treated as process information, variant of process information, and amount of process information. For instance, we assume the case in which the size of the process information ($S_{procinfo}$) is less than a page size S_{page}. We designate one process as an essential process. Then, the increased memory size is $((S_{page} - S_{procinfo}) \times V_{procinfo}) \times N_{proc}$ (N_{proc} is the number of processes, and $V_{procinfo}$ is the variants of data structure).

In the evaluation environment, we used Linux 3.2.65 as a guest OS. If we consider the process control block as process information, the increased memory of each essential process is 2,320 bytes because the size of `task_struct` is 1,776 bytes and the page size is 4 KB. In the evaluation environment, the increased memory is up to 209 KB because the number of processes in the environment is 70–90. Although the memory usage increased by 209 KB, its effect is small because the VM has 1 GB memory.

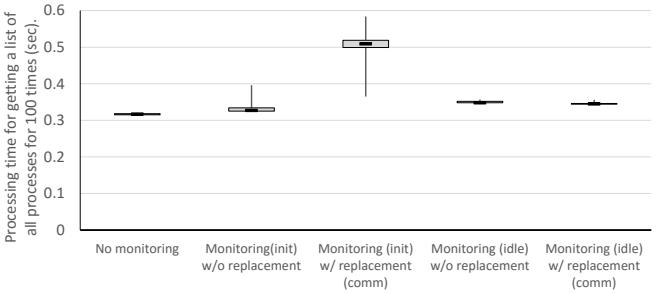

Figure 5. Performance of collecting a process list.

Table III
MEAN TIME FOR OBTAINING A PROCESS LIST 100 TIMES.

Read restriction of process information	Essential Process	Replacement of Process Information	Processing Time (sec)
Disabled			0.316
Enabled	init	Nothing	0.341
		comm	0.495
	idle	Nothing	0.350
		comm	0.347

VI. RELATED WORK

To prevent attacks to antivirus software, their vendors implemented various software protection mechanisms [4]. Some antivirus software prevents self-termination by inserting kernel mode or filter drivers to prevent users or malware from terminating antivirus services through APIs or commands. Other protection mechanisms prevent their executable files, libraries, and configuration files from being tampered by inserting an I/O filter driver. These self-protection mechanisms are effective for attacks; however advanced attacks can bypass them [4]. In addition, these mechanisms are dependent on the structure of the antivirus software. To protect antivirus software by using these

mechanisms, they must be modified and all files must be protected from being tampered. Our proposed method can prevent attackers from identifying antivirus software, thus decreasing further attacks because it is difficult to determine attack targets. By combining the proposed method and the existing antivirus self-protection mechanism, the security of the entire system will be enhanced.

Security monitoring studies often use virtualization technology. HyperTap [10] is a monitoring framework for reliability and security of VM. HyperTap monitors events inside a VM including context switches, system calls, and I/O accesses. It uses events natively occurring on virtualized environments; however, the proposed method uses additional memory access violation to monitor accesses to specific memory regions. As the proposed method can detect more events than HyperTap, additional overheads are large as well.

Similar to the proposed method, Xie et al. [11] focused on essential information managed by a kernel on a VM. The method in [11] compares information extracted from a VM and VMM. If the VMM detects a difference during the comparison, it analyzes hidden information and sends an alarm message to the end user. Compared with conventional VMI approaches [6], [7], Xie et al. reconstructed high-level information; including running processes, network connections, and opened files; from low-level information acquired by the VMM. The proposed method also acquired high-level information from low-level information. However, the purpose of information reconstruction is different from that in [11] because the proposed method reconstructs high-level information and replaces it to disguise process information from attackers.

VII. Conclusion

This paper proposed the design, implementation, and evaluation of memory access monitoring and disguising method of process information to avoid attacks to essential services. The proposed method monitors read accesses to the process information and disguises it when the prohibited area includes the source of the access. Because the proposed method disguises process information based on the source of read access, identifying an essential service from changes of process information becomes difficult.

Evaluations showed that the proposed method complicates the identification of an essential process from attackers. The modified source code of a guest OS is just one line and we did not modify the source code of essential processes. By using the proposed method, the measurement results showed performance degradation of less than 45% and a memory overhead of 209 KB when the number of processes is 90. These evaluations show that the proposed method can avoid attacks and its overheads are acceptable.

Acknowledgment

This work was partially supported by JSPS KAKENHI Grant Numbers 16K16067 and 16H02829.

References

[1] F-secure. Agobot. [Online]. Available: http://www.f-secure.com/v-descs/agobot.shtml
[2] F-secure, T0rnkit. [Online]. Available: http://www.f-secure.com/v-descs/torn.shtml
[3] Packetstorm. dica.tgz. [Online]. Available: http://packetstormsecurity.com/files/26243/dica.tgz.html
[4] B. Min and V. Varadharajan, "A novel malware for subversion of self-protection in anti-virus," *Software: Practice and Experience*, vol. 46, no. 3, pp. 361–379, 2016.
[5] F.-H. Hsu, M.-H. Wu, C.-K. Tso, C.-H. Hsu, and C.-W. Chen, "Antivirus software shield against antivirus terminators," *IEEE Trans. Inf. Forensic Secur.*, vol. 7, no. 5, pp. 1439–1447, 2012.
[6] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Network and Distributed Systems Security Symposium*, 2003, pp. 191–206.
[7] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection and monitoring through vmm-based "out-of-the-box" semantic view reconstruction," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 2, pp. 12:1–12:28, 2010.
[8] R. Riley, X. Jiang, and D. Xu, "Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing," in *Proc. 11th International Symposium on Recent Advances in Intrusion Detection*, 2008, pp. 1–20.
[9] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi, "Secure in-vm monitoring using hardware virtualization," in *Proc. 16th ACM conference on Computer and communications security*, 2009, pp. 477–487.
[10] C. Pham, Z. Estrada, P. Cao, Z. Kalbarczyk, and R. K. Iyer, "Reliability and security monitoring of virtual machines using hardware architectural invariants," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 13–24.
[11] X. Xie and W. Wang, "Rootkit detection on virtual machines through deep information extraction at hypervisor-level," in *2013 IEEE Conference on Communications and Network Security (CNS)*, 2013, pp. 498–503.
[12] M. Sato, T. Yamauchi, and H. Taniguchi, "Process hiding by virtual machine monitor for attack avoidance," *Journal of Information Processing*, vol. 23, no. 5, pp. 673–682, 2015.
[13] LibVMI Project. Libvmi. [Online]. Available: http://libvmi.com/
[14] B. D. Payne, D. D. A. Martim, and W. Lee, "Secure and flexible monitoring of virtual machines," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC)*, 2007, pp. 385–397.
[15] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, 2003.