Electronic Preprint for Journal of Information Processing Vol.23 No.5

Recommended Paper

Process Hiding by Virtual Machine Monitor for Attack Avoidance

Masaya Sato[1,a)] Toshihiro Yamauchi[1] Hideo Taniguchi[1]

Received: December 7, 2014, Accepted: June 5, 2015

Abstract: As attacks to computers increase, protective software is developed. However, that software is still open to attacks by adversaries that disable its functionality. If that software is stopped or disabled, the risk of damage to the computer increases. Protections of that software are proposed however existing approaches are insufficient or cannot use those software without modification. To decrease the risk and to address these problems, this paper presents an attack avoidance method that hides process from adversaries who intend to terminate essential services. The proposed method complicates identification based on process information by dynamically replacing the information held by a kernel with dummy information. Replacing process information makes identifying the attack target difficult because adversaries cannot find the attack target by seeking the process information. Implementation of the proposed method with a virtual machine monitor enhances the security of the mechanism itself. Further, by implementing the proposed method with a virtual machine monitor, modification to operating systems is unnecessary.

Keywords: attack avoidance, process information, virtual machine

1. Introduction

Attacks to programs increasing and its prevention become important research topics. To prevent and mitigate damages from those attacks, protective software is developed. In addition, programs that record or analyze such attacks become more important in current situation. We call these software as essential services. Recently, attacks for essential services to deactivate them are confirmed. For instance, Agobot has the functionality to stop anti-virus software. T0rnkit and dica stop log collectors in order to hide the malware installation process from the system administrator of a target computer. The risk of damage to target computers increases when essential services are deactivated. Therefore, it is an important challenge to protect essential services for reducing damage to a computer.

To prevent attacks on essential services, methods using a virtual machine monitor (VMM) have been proposed [1], [2]. These methods prevent the essential services from being affected by isolating them from the target computer using virtualization technology. Research [1] reveals a method for offloading the intrusion detection system (IDS) from one virtual machine (VM) to another. Moreover, Riley et al. proposed a method for malware detection using a VMM [2]. However, these methods do not utilize existing essential services and software already installed and operational. Hsu et al. proposed a method to prevent anti-virus software from being terminated without the consciousness of the anti-virus software users [3]. This method monitors Windows application program interfaces (APIs) by system service descriptor table (SSDT) hooking and filter out hazardous API calls that will terminate anti-virus software. This method is effective for preventing anti-virus software termination using API calls. However, this method is vulnerable to SSDT patching commonly used by rootkits because this method replaces some SSDT entries to their handlers. For this reason, protecting the system from kernel-level malware is a challenging problem.

To address these problems and to avoid attacks, this paper proposes an attack avoidance method hiding essential services from adversarial software. The proposed method complicates the identification of an essential service by replacing the process information with a dummy. Specifically, this method detects context switches and replaces the original process information with dummy process information when processes for the essential services are not running. Once the process is dispatched, the original information is restored. The process information of the essential service is replaced without disturbing its functionality. Adversaries cannot detect and identify a target for attack because the process information of the target is replaced. For security and adaptability, the proposed method is implemented using a VMM. Because of its design, a VMM is more difficult to attack than an operating system (OS). Furthermore, implementation with modification to a VMM can reduce the costs involved in modifying existing software.

The contributions made in this paper are as follows:

- We propose an attack avoidance method complicating process identification from adversaries. Because adversaries identify and attack a target process using process informa-

[1] Graduate School of Natural Science and Technology, Okayama University, Okayama 700–8530, Japan
[a)] sato@cs.okayama-u.ac.jp

The initial version of this paper was presented at Computer Security Symposium 2013 (CSS 2013) in October 2013. This paper was recommended to be submitted to Journal of Information Processing (JIP) by Program Chair of CSS 2013.

tion, replacing the process information complicates the identification of an attack target. Our proposal is a proactive approach to attacks contrary to existing reactive countermeasures.

- We design a system for replacing the process information of essential processes with a VMM. Because the proposed system is designed with modifications to the VMM along with an additional application program (AP) on a manager VM, the proposed system requires no modification to OSes and APs on a VM providing essential services.
- Evaluation results using a prototype of the proposed method show the effectiveness of the method for attack avoidance. Performance evaluation shows that the performance overhead in APs is negligible.

2. Background

2.1 Attacks for Anti-Virus Software

Agobot is malware that attacks anti-virus software. Agobot installs a 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 anti-virus software is disabled by malware such as Agobot, the risk of damage to the computer system increases.

T0rnkit and dica 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, moderation of damages, and avoidance of attacks for essential services are required.

2.2 Existing Countermeasures

Research into an offloading host-based intrusion detection system (IDS) with a VMM was proposed in VMwatcher [1]. Implementing an IDS by modifying a VMM makes it difficult to attack the IDS. In the same manner, NICKLE [2], which prevents the execution of a kernel-level rootkit, has been proposed. Because it monitors the execution of kernel code with a VMM, only authorized code can be executed. These methods help to prevent attacks that are difficult for existing methods without a VMM to detect and prevent.

Protection of anti-virus software from termination, which called ANSS, was proposed [3]. ANSS hooks system calls related to process termination and controls them if they would stop anti-virus software. ANSS is implemented as a Windows driver that operates in kernel mode.

2.3 Problems with Existing Methods

Existing methods cannot use essential services without modifying them. Furthermore, these methods are effective only when they are not themselves attacked. If these methods are themselves attacked by adversaries, a system administrator cannot utilize those services to avoid the attack. The methods described in Section 2.2 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 applications because the method involves the emulation of each system call. To completely offload the IDS, it is necessary to emulate all system calls. However, complete emulation is difficult to implement.

Even though effective VMM-based methods have been proposed, 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 kernels is different from the information a VMM collects. This semantic gap makes it difficult to port functions from existing software to a VMM.

3. Related Works

Out-of-the-VM monitoring approaches are proposed [1], [4], [5], [6], [7], [8]. They use virtualization technology to isolate the monitor for security of themselves. While some researches [1], [4] monitor inside the VM in a non-intrusive manner, other researches [5], [7], [8] modify the memory contents of the VM. Further, SIM approach [6] inserts an agent inside the VM. Our approach is similar to Ether and EagleEye however our purpose is different from them. While Ether and EagleEye provide monitoring of the VM, the proposed method monitors memory access and context switches, but these monitoring are used for providing proactive protection of essential services inside the VM.

Some researchers have proposed methods for preventing the illegal alteration of memory contents [9], [10]. While these methods are indeed useful in preventing attacks, the proposed method avoids them. Even if these methods succeed in preventing an attack, the existence of essential services is still detectable to adversaries. When adversaries detect essential services, they can nonetheless disable them to avoid detection. Because our method complicates process identification, adversaries will find it difficult to avoid detection by essential services hidden with the proposed method. For this reason, the proposed method is a proactive approach while existing methods are reactive. Because our approach and existing methods cover different stages of attack, it is possible to cooperate with each other.

Several researches [11], [12], [13] use VMM to provide access control of VM. SecVisor employs a similar approach for access control using a hypervisor [11]. While SecVisor protects kernel codes, the proposed method focuses on the data area of the guest kernel. HUKO [12] provides kernel integrity by access control of kernel extensions. HUKO is similar to SecVisor from the viewpoint that it uses policy based access control but employs hardware assisted paging. Our system is similar to the approach found in Sentry [13]. Sentry protects data inside the user VM by partitioning the data structure of the kernel. Our method, however, is advantageous in that it does not require any modification to

the data structure. Modifying kernel data structures requires high level knowledge of kernel development and thus degrades practicality. No modification to kernel data structure is preferable.

ANSS [3] is effective for protecting anti-virus software from termination. ANSS intercepts and monitors some API calls with parameters that will stop or suspend anti-virus software to filter out malicious calls. Even though ANSS is effective, it is vulnerable to malware patching SSDT entries. The paper proposing ANSS stated working with the anti-hooking mechanism is effective. Our method is tolerant to attacks patching tables in kernel space because the VMM restores the original process information when essential processes are running. Moreover, our method has possibilities to avoid unknown attacks as long as they rely on the process information. Boeck et al. proposed protection of log collectors [14]. This method is similar to the proposed method from the viewpoint of the protection target. It protects a logging daemon by providing hardware-based trust with Trusted Platform Module and a function of AMD's secure virtual machine (SVM) technology. In contrast, our proposal focuses on access to the process information. Because the process information is less dependent on hardware, our proposal is more general.

These approaches are effective for protecting software on a VM, however, some researches are weak for attacks or difficult to implement for various services. The out-of-the-VM approach is powerful but existing researches are reactive. The effect of reactive approaches is limited. By contrast, proactive approaches have a large impact on reduction of damage from adversaries. Some proactive approaches are proposed but they have some problems including problems stated in Section 2.3. For this reason, a proactive approach addressing the above problems is required.

4. Attack Avoidance Method for Complication of Process Identification

4.1 Purpose

The following explains the purposes of our research:

(1) Avoidance of attacks to essential services

(2) Use of existing software without modification.

It is difficult to handle various attacks with existing methods. Therefore, we aim not to protect but to avoid such attacks. Even if offloading the functionality of existing services is considerably effective, the cost of doing so is high because of modification to existing software. Thus, it is preferable to avoid attacks without modifying existing software.

4.2 Basic Idea

To achieve the purposes outlined in Section 4.1, we propose complicating process identification to avoid attacks by replacing the process information for essential services. Because adversaries identify a target to attack, we propose replacing the original process information of the target process with dummy process information. Moreover, by implementing our method in a VMM, the existence of our system is difficult to identify. Because of this, an attack on the proposed method itself is difficult and unlikely.

Because a VMM is developed only for providing VMs, interfaces for accessing it are limited and the total amount of source code involved is far less than in a normal OS. Thus, attacking the VMM is more difficult than attacking the OS. Moreover, implementing the proposed method does not necessitate modifications to the source code of the guest OS or its essential services. With this feature, existing software resources are utilized efficiently. For these reasons, we utilize a VMM with the proposed method.

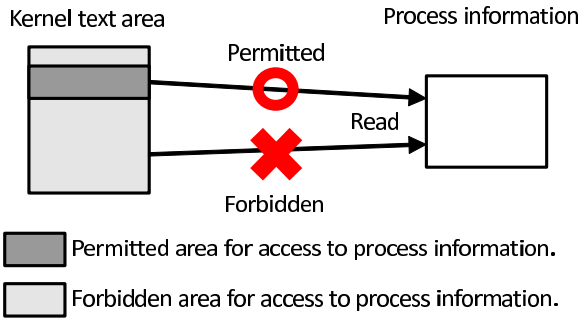

Fig. 1 Access control to process information.

The essential process provides essential services. The essential process is defined by the administrator based on importance of its functionality. For example, damages increase when a security software or an administration tool are stopped by adversaries. In such case, the administrator defines the security software or the administration tool as essential service and the process that provide the functionality of those services as essential process.

4.3 Hiding Process Information of Essential Processes

The complication of process identification consists of the following:

(1) Limiting access to the process information

(2) Replacing the process information

Figure 1 provides an overview of the procedure for limiting access to the process information. With this method, the kernel text area, which can access process information, is pre-defined. Access to the page that includes process information is set as forbidden. When an access violation to that page occurs, the method returns a dummy value when the subject is not included in the pre-defined area. If the subject is included in the pre-defined area, the method returns the original content. With this approach, the original process information is invisible from adversaries because only legitimate functions in the kernel text are permitted to access process information.

In replacing the process information, process information for the essential processes must first be replaced. When an essential process is running, the original process information is restored. The overall procedure for replacing the process information is shown in **Fig. 2**. Here, we define normal processes as all processes excluding essential processes. When a context switch from an essential process to a normal process occurs, the method exchanges the original process information with a dummy. Alternatively, when a context switch from a normal process to an essential process occurs, the method restores the original process information. With this approach, the original process information for the essential processes is invisible from other processes. This method does not disturb the execution of essential processes. The replacement of process information is described in detail in Section 5.

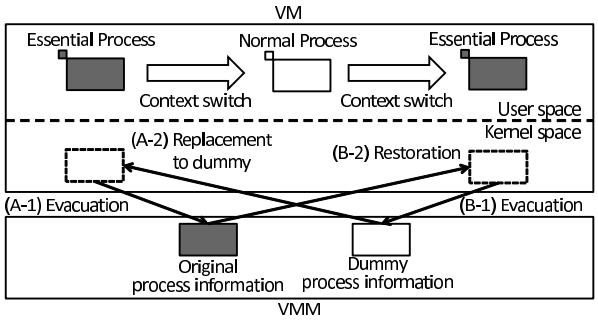

Fig. 2 Replacement of process information between essential process and normal process.

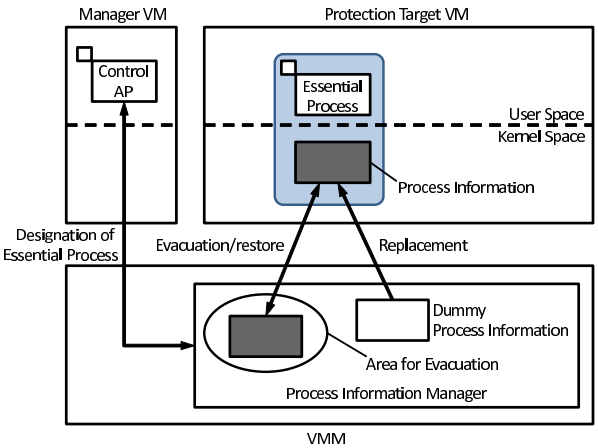

Fig. 3 Overview of proposed system.

The advantage of replacing process information of the essential process is manageability and performance. Replacement of the process information of essential process achieves the purpose, however, replacing the process information of a normal process will provide greater security. Replacement of the process information of a normal process makes it difficult to estimate the essential process from the information of a normal processes. However, it raises two problems: degradation of manageability and performance. If process information of all processes including essential processes and normal processes, the manager of the protection target VM cannot manage the VM. To manage the VM, additional interfaces or other management methods are required. Those additions have possibilities for additional vulnerabilities, so that additions to existing OSes should be kept small. Besides, replacement of the process information of a normal process causes degradation of performance of the VM. Because the proposed method replaces the process information when a context switch occurs, replacement of process information of essential process and normal process requires memory copy in each context switch. It results in large overhead in each context switch. For these reasons, we only replace the process information of essential process.

4.4 Structure of the Proposed System

The overview of the proposed system is shown in **Fig. 3**. The proposed system consists of a process information manager, replacing process information and controlling access to process information. The process information manager monitors context switches in the target VM. The process information manager exchanges essential process information with dummy information, and in legitimate case, restores the original. The original process information in the VM is copied to an area allocated in the VMM. The area is allocated and managed by the VMM for each VM. A variety of dummy process information is prepared in advance and the system determines what information is paired with each process when a context switch occurs.

With the proposed system, the security administrator designates the essential process. However, the administrator cannot distinguish which process is the essential process because the process information of that process is hidden from the administrator. Therefore, the administrator must determine which program should be treated as an essential process before it starts, and notify the information of that process to the administrator of the Manager VM. The administrator of the Manager VM designates to the process information manager which process is the essential process. In the proposed system, the Control AP designates which process is an essential process. Thus, a security administrator responsible for the protection of the target VM must communicate to the VMM manager in advance which processes are essential.

4.5 Potential Attacks

Adversaries can stop or disable essential processes if they detect the existence of the proposed system and identifying the essential process. Thus, it is necessary to make it difficult for adversaries to judge whether a process is an essential process or not.

4.5.1 Measurement of Context Switch Times

Adversaries can identify an essential process by comparing the processing time of the context switch between an essential process and a normal process. With the proposed system, the process information of the essential process is replaced. Thus, the time for essential process context switches is longer than with a normal process. Given this difference, adversaries can identify which process is an essential one.

To conceal the difference in the processing times of context switches, it might suffice to apply the same processing time to normal processes. This done, the difference in the processing times of context switches between the essential processes and the others becomes meaningless. However, the performance of the entire system degrades.

As an alternative, a time controlling function is effective. This function is used in malware analysis. Some malware detect the presence of debuggers by measuring the processing time and respond by changing their behavior to avoid analysis. To prevent this from happening, a time controlling function is proposed. This function stops the virtual CPUs allocated for malware. When the CPU is stopped, the debugger analyzes malware and resumes the CPUs when the analysis is complete. This function enables us to evade the detection of the proposed system by adversaries.

4.5.2 Continuous Monitoring of Process Information

Adversaries can identify the essential process by continuously monitoring the process information for each process to determine whether the process information has changed during a context switch. If a process is an essential process, its process information is replaced during a context switch whereas the process information for a normal process is remains unchanged. Therefore,

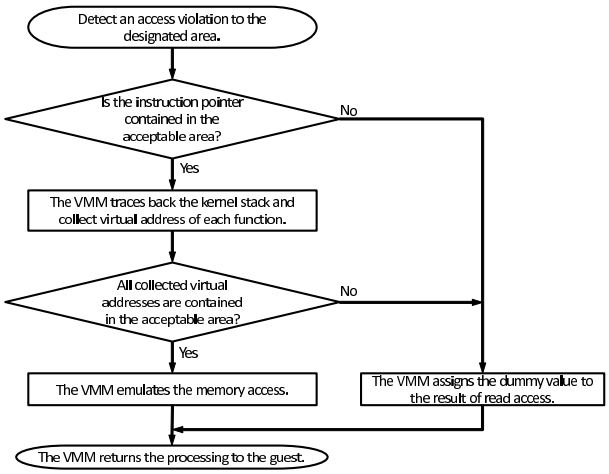

Fig. 4 Determination procedure for access to process information.

if part of the process information has changed, even though given a normal context switch it would not, adversaries can identify that process as an essential process.

To prevent detection by continuous monitoring of process information, a combination of access control and process information replacement is effective. Here, we assume an adversary who continuously monitors process information with a loadable kernel module in Linux. At first, the VMM forbids read access to areas containing process information from kernel codes. This is done to prepare for avoiding attacks. The area containing kernel code without kernel modules must be pre-defined. In this situation, if an access violation to the designated area occurs, the VMM determines whether the access is acceptable or not by following the procedure shown in **Fig. 4**. If an instruction pointer is out of range from the designated area, the VMM returns the dummy value to the guest. If not, the VMM traces back the kernel stack and collects the virtual addresses of each function. If all the addresses are contained within the designated area, the VMM permits the read access and returns an original value. If not, the VMM returns the dummy value.

The VMM permits read access by disabling read protection of the area. However, once the protection is disabled, adversaries also can read the area. Thus, the proposed system set the trap flag at same time to cause VM exit at the next instruction. If the trap flag is set, debug exception occurs in every instruction. After the debug exception is occurred at the next instruction, the proposed system enables the protection again and set the trap flag off. With this procedure, the proposed system can limit access to the area from instructions out of range from the designated area.

Because this access control model depends on an integrity of the guest kernel, an attack patches a kernel text must be considered. DKSM attack is one of an attack patching kernel text area [15]. To patch kernel text area, manipulation of CR0 is required. Because kernel text area is write protected ordinarily, adversaries manipulate CR0 to remove write protect of kernel text area. In fully virtualized environment with VT-x, access to control registers causes VM exit. Therefore, the VMM can detect patching of kernel text area by monitoring access to CR0. This monitoring ensures kernel code integrity and functionality of above access control model.

With this procedure, even if adversaries continuously monitoring process information, identifying an essential process is impossible.

4.5.3 Random Process Termination

If an adversary terminates processes at random, essential processes potentially stop. Because this attack do not require process identification, complication of process identification is ineffective.

Although this type of attack is diverge from our purpose, the proposed system can mitigate the damage caused by the attack by launching a decoy. If adversaries already know that the proposed system is installed on a computer, which is a target of them, and no essential process is found on that computer, they are likely to terminate processes randomly to terminate essential processes. However, if decoy process is exist in the target VM, adversaries found that the process as their attack target and terminate it. Though the process is terminated, its effect is small and the termination of essential processes is avoided.

There are two methods for launching decoy: making a process, which have less effect for the VM, as decoy or preparing special processes for decoy. Making a process for decoy requires choice which process is less important for the VM. Because it requires high-level knowledge for the task of each process, it is hard to choose the decoy process. In contrast, preparing a special process for decoy is easy to introduce. By swapping the process information of the essential process and the decoy, adversaries misunderstand the decoy as an essential process. Hence the decoy is supposed to be terminated, the effect by termination must be kept small. Making a process as decoy have effects for the VM but preparing special processes have less effect. One problem for preparing special processes for decoy is that it requires cooperation among the manager of the target VM and the manger of the VMM. However, the proposed system assumes that the manger of the VM designates the essential process by informing it to the manager of the VMM, the cooperation is not a critical issue. For this reason, preparing special processes for decoy is easy to introduce and have less effect for the VM than making a process as decoy.

4.6 Limitations

As stated in Section 4.5.3, it is possible that an essential process will be stopped by random process termination. To prevent attacks on essential processes, regulating access control to process information is effective. However, attack prevention diverges from our stated purpose. Thus, we do not discuss strategies in attack prevention. As stated in Section 4.5.3, the proposed system can be used for mitigation of damages caused by random process termination.

Because the proposed method replaces the legitimate process information of an essential process with dummy information, a security administrator tasked with protecting the target VM cannot control the essential process. This is inconvenient for the administrator. We assume that the essential process is a kind of resident programs. Thus, the scope for the application of the proposed method is restricted to resident programs. We do not assume this method will feasibly apply to other programs. To

do so, an additional interface would be needed for the security administrator to communicate with the process information manager. However, this addition would expose vulnerabilities. Thus, we do not consider implementing any additional communication interface to the VMM.

5. Replacement Method of Process Information

5.1 Replacement Target

5.1.1 Definition of Process Information

Assuming Linux for x86 or x64, we defined the following as process information:

(1) Process control block
(2) Kernel stack
(3) Hardware context
(4) Page tables
(5) Memory used by a process

Hiding all of the above information is necessary to make the process completely invisible. However, identifying a process from (3), (4), and (5) is considerably difficult. On the other hand, (1) and (2) include especially helpful information for process identification. For these reasons, we treat (1) and (2) as process information.

The following describes the process information in detail:

Process control block (task_struct)

The process control block contains information that is effective for process identification including the PID (Process ID), the TGID (Thread Group ID), the executable file name, and the PID of the parent process. In Linux, the process control block is given as task_struct structure, and it is generated for each process or thread.

Kernel stack and thread_info structure

Both the kernel stack and thread_info structure are allocated in a union, named thread_union. A thread_union is allocated for each task_struct. A kernel stack contains the address, arguments, and return value of functions called in the kernel space. The thread_union and task_struct are linked to one another.

5.1.2 Replacement Target

The process information defined above includes information used with a kernel when a process is not running. For example, a kernel schedules processes or delivers signals by reference to the process information for each process. For this reason, process scheduling and signal delivery would be obstructed were all process information replaced. Thus, two policies are considered for replacing process information.

Policy (1) Replace as much process information as possible, with the exception of information used by a kernel while the process is not dispatched.

Policy (2) Replace only information helpful to adversaries for identifying processes.

When replacing process information under Policy (1), processes are more difficult to identify than under Policy (2). However, replacement under Policy (1) requires many more replacement copies leading to overall performance degradation. Replacement under Policy (2) results in less overhead than Policy (1). However, the strategy suggested under Policy (2) requires that we survey what information is used by malware for identifying the attack target.

Understood merely as a countermeasure to adversarial attack, replacement under Policy (1) is preferable. However, practical utility requires the efficient suppression of any superfluous performance overhead. Therefore, in this paper, we employ Policy (2) for a replacement strategy.

5.1.3 Information Used for Process Identification

We turn now to a discussion of the information used by malware to identify an attack target process. Agobot, developed for Windows, searches the name of a program from a process list in a target computer. If a name matches an entry in the list, Agobot issues the TerminateProcess() function to stop the process and all threads within the process. Dica, developed for Linux, stops syslogd with the killall command. The killall command acquires the process PID to suspend processes by searching the name of the attack target from the proc filesystem. After acquisition, the command invokes a kill system call to stop the process.

Whereas it is not uncommon to find malware that stop processes, many of these programs discern the target process with the name of the program. Therefore, it is effective to replace the process name as well.

5.1.4 Adequate Dummy Information

To hide the existence of essential processes, dummy information should be chosen properly. For instance, to hide a process, the process name should be replaced with the name of a common program, running on common servers. If the name of an essential process is replaced with a common name, it will be more difficult for adversaries to detect the existence of the essential process. Additionally, the name should be chosen randomly. It would be easy to detect the existence of the proposed method when the dummy information always the same.

5.2 Trigger for Replacement of Process Information

To replace the process information, it is necessary to determine whether or not a process-switch from and a process-switch to are replacement targets. For this mechanism, the detection of context switches in a VM from a VMM is required.

In fully virtualized environments, a guest OS works in VMX non-root mode and a VMM works in VMX root mode. Some instructions in non-root mode cause a VM exit and the processing is switched to the software running in the VMX root mode. Instructions not permitted in VMX non-root mode contain write to CR3 register. In an OS supporting multiple virtual address spaces, write to CR3 occurs when context switching to change address space because CR3 contains a beginning address of a page directory. Therefore, a context switch in a VM can be detected by monitoring VM exits caused by writes to CR3.

5.3 Acquisition of Process Information in Guest OS

5.3.1 Acquisition of Process Information of Current Process

As shown in **Fig. 5**, thread_info and task_struct can be acquired by calculating the address from the RSP register with the VMM. Because the beginning address of a thread_info can be calculated from the RSP register and a task member of the thread_info indicates the beginning address of a task_struct, the VMM can acquire

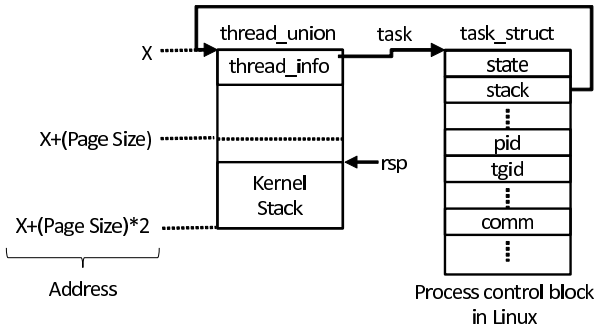

Fig. 5 Relation between thread_union and task_struct.

Table 1 Pros and cons of next-process identification methods.

Methods	Pros	Cons
Scanning method	Ease of implementation	Large performance overhead
List-based method	Performance overhead is small.	Total amount of memory usage increases.
Trigger-insertion method	Performance overhead is small.	Number of triggers is limited.

the process information in a guest OS from the RSP register. In this regard, the VMM must hold the definitions for each structure beforehand.

5.3.2 Acquisition of Next-Process Information

The method for acquiring the process information stated above is not effective for any processes set to run next (i.e., for the next-process). Therefore, another method is needed. What is about to be written to the CR3 register is usable information for the acquisition of process information concerning the next-process. Considering this, there are three methods for identifying the next-process to acquire its process information.

- Scanning method: this method scans the process list of the protection target VM to determine the next process.
- List-based method: With this method, the VMM holds a list, containing the CR3 value and the address of the task_struct for each essential process. This method searches the value for what is going to be written to the CR3 register to identify the next-process.
- Trigger-insertion method: This method inserts a trigger, switching execution from the protection target VM to the VMM, in the kernel of the protection target VM for the identification of the next-process. For example, inserting the INT3 instruction in the kernel memory area is effective. The trigger must be inserted in the place where the proposed system can acquire the process information concerning the next-process.

The advantages and limitations for each method are shown in **Table 1**. The scanning method is easy to implement because it requires only a lookup of the next process from the process list of the protection target VM. However, this method creates debilitating performance overhead because the method scans the process list after each context switch on the protection target VM. By contrast, the list-based method and the trigger insertion method do not require significant performance overhead. The list-based method uses a VM exit, which occurs unconditionally in a fully virtualized environment. Because unnecessary VM exits do not occur using the list-based method, performance overhead is minimal in above three methods. With the trigger-insertion method, unnecessary VM exits occur.

Thus, from a viewpoint of performance, the list-based method is considered best. Even though the list-based method is disadvantageous in terms of amount of memory usage, it can be estimated as sufficiently small. The amount of memory used by the list-based method can be estimated as under 100 bytes given that an entry in the list created by the method averages at nearly 10 bytes. Memory used by Xen [16] one of the more popular VMM is about 182 megabytes. The amount of memory used by the list-based method is therefore sufficiently small relative to Xen. One disadvantage to the trigger-insertion method is the limitation to the number of triggers. The use of debug registers and the insertion of the INT3 instruction are pertinent triggers with the method. Using debug registers is faster than inserting INT3 instruction. However, the number of debug registers is limited. For these reasons, we employ the list-based method.

5.4 Designation of Essential Process

The proposed system requires the Control AP to designate the essential processes prior to replace their process information. Due to the structure of the proposed system, it is necessary for the administrator of the protection target VM to provide the essential process information to the administrator of the manager VM via e-mail, or by other means. The proposed system does not provide a notification mechanism because additional interfaces to the VMM must be kept at a minimum. Such additions of the VMM interface risk exposing it to vulnerabilities. Before the program initiates, the administrator for the protection target VM must provide ether the full path of the essential process's executable file or a command name. The administrator of the manager VM takes that information and provides it in the process information manager. The exchange is implemented with an event channel a mechanism for VMs to communicate with a Xen hypervisor. When the process information manager receives the information, it can monitor the name of the process running on the protection target VM. If it detects that the process with the designated name is scheduled, the process information manager replaces the process information of that process on the protection target VM. Because the procedure for designating essential processes is conducted as described above, the information must be exchanged before the essential process initializes on the protection target VM.

5.5 Handling Multi-Core Processors

We shall here assume an environment with multi-core processors. When an essential process is running on one CPU core, other processes might be running simultaneously on the other CPU cores if multiple CPU cores are allocated to a VM. In this situation, a process running on one CPU core is able to refer to the process information of essential processes running on other CPU cores because the original process information is restored when the process is dispatched.

To address this problem, we prohibit running normal processes while essential processes are running. This is accomplished by

suspending all virtual CPUs except the virtual CPU used by an essential process. Therefore, any reference to essential process information by normal processes is restricted.

6. Evaluation

6.1 Purpose and Environment

The purpose of the evaluation is to confirm the achievement of the purposes stated in Section 4.1 and the performance overheads caused by the proposed system. To confirm the avoidance of attacks to essential services, we experimented that termination of essential processes in the following environment. With this experiment, we aim to check whether the proposed method can obstruct the activity of adversaries and avoid attacks or not. Additionally, we also checked that modification to existing software running on the VM is required or not. Performance evaluation with LMbench and Linux kernel compilation show the overhead incurred by the proposed system. With the performance evaluation, we demonstrate that how the proposed method affects the performance of a basic function of OS and a real world application.

The environment used for evaluation is shown in **Table 2**. All evaluations are performed on a machine with Intel Core i7-2600 (3.40 GHz, 4-cores) and 16 GB RAM. The protection target VM is fully virtualized by Intel VT-x. Hyper-threading and the hardware assisted paging functionality are disabled. One virtual CPU and 1 GB memory are allocated for the VM. The allocated virtual CPU is pinned to a physical CPU core to avoid measurement instability caused by physical CPU sharing. The virtual CPU for the manager VM and the protection target VM is pinned to other physical CPU core. The manager VM and the protection target VM use different physical CPU core.

6.2 Attack Avoidance

To confirm the effectiveness of the proposed system, we evaluated difficulty of detection and resistance for termination for essential services. In our experiment, we examined whether the name of an essential process was replaced when the proposed system was applied to the essential processes. Further, we tested that the essential process can be terminated by killall command or not. We assumed the killall command as a tool used by adversaries. The killall command searches the name of a program from the proc filesystem to determine the PID of the attack target.

In this experiment, we examined whether or not the original name of the essential process is listed under a ps command. These commands refer to the same information inside a kernel. To evaluate the proposed system, we assumed syslogd as an essential process and changed its name to apache2.

On the protection target VM, we listed the names for all processes but the list did not contain the string "syslogd." Instead, apache2 was listed in its place. This result shows that the name of the essential process was successfully changed—thus concealing it. The results also show that adversaries basing their attacks on the process name can be avoided using the proposed system. For instance, killall command terminates processes based on the process name. We also test that the killall command can terminate the essential process or not. The results of the experiment showed that the termination with killall command failed and the essential process continued running. As stated above, the killall command use the name of a program. Because the name of the program is replaced by the proposed system, killall command cannot find the target process. The kill command with PID of the attack target can terminate the target process. However, PID is not always the same and search of PID requires the information what the process it is. Because the name of a program is frequently used as key for searching PID, we believe the proposed system is still effective for the kill command.

This experiment showed that the protection target VM with the proposed method can avoid some attacks that terminates essential processes.

Table 2 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 3.2.0 64-bit)
Hardware	
CPU	Intel Core i7-2600 (3.40 GHz, 4-cores)
	Manager VM: 1 Virtual CPU
	Protection Target VM: 1 Virtual CPU
Memory	Total: 16 GB
	Manager VM: 1 GB
	Protection Target VM: 1 GB

6.3 Performance

6.3.1 Context Switching Time in Process Information Saving

We measured performance overheads in context switches caused by the process information saving functionality of the proposed system in the protection target VM. We use LMbench micro-benchmark version 3 for the measurement. LMbench includes measurement of contexts switching time in different numbers of processes with different working set sizes.

Table 3 shows the context switch times between different numbers of processes with different working set sizes. With the proposed system, context switching time degrade because copies of process information from the VM to the VMM is required in each context switch. After copying of process information, the proposed system replace the process information only when the process switching from or switching to is an essential process. Thus, context switching time also degrades when essential processes are running in the VM.

The measurement results show that the overhead with the proposed system is less than 15%. The context switching overhead caused by copies of process information is depend on the size of process information. Because the size of the process information is almost the same, the overhead does not significantly change in each case. From the results, performance degradation with increase of working set size is large compared with the degradation

Table 3 Context switching times in microseconds.

# of processes	2	2	2	8	8	16	16
Working set size	0K	16K	64K	16K	64K	16K	64K
Xen	25.9	40.8	44.9	39.6	42.2	38.9	39.4
Proposed system	30.0	42.6	41.0	43.5	50.4	42.5	41.7

Table 4 Linux kernel compilation times in seconds.

	Compilation time
Xen	133.0
Proposed system	131.2

with the proposed system.

Consequently, the overhead with the proposed system in context switch is less than 15%. Further, it has small effect when working set size is large.

6.3.2 Kernel Compilation

To evaluate performance overheads in APs, we measured times of Linux kernel compilation. The version of Linux kernel used for evaluation is Linux 3.2 and the kernel is configured with `make allnoconfig`. Compilation time is measured by `time` command.

Table 4 shows the measurement results. The difference of compilation time between Xen and the proposed system is less than 2 seconds and the compilation time with the proposed system is faster than the time with Xen. However, maximum time in experiments showed that the time with the proposed system is greater than the time with Xen.

From the observation of measurements, that difference is in an acceptable error range. Because the difference of compilation time is less than 2%, performance degradation incurred by the proposed system for APs is considered negligible.

6.4 Discussion

From the evaluation of attack avoidance, it is confirmed that attacks using process name are avoided by the proposed method. We believe that attacks using other process information are also avoidable when the proposed method hide it. Whatever attacks use process information for identification of the target process, the proposed method works well.

In addition, the proposed system is implemented in the VMM and no modification is introduced to software on the VM. Thus, existing software can be used without modification. Further, the proposed system replaces the process information of essential process when a context switch between the essential process and the essential process. This causes a little overhead to the performance of the VM, however, no additional monitoring is required.

Performance evaluation showed that the overhead in context switch time with the proposed method is less than 15%. Further, it is observed that the performance degradation for APs incurred by the proposed method is negligible. However, performance degradation is depend on the processes running on the VM. If many essential processes running on the VM, performance degradation of essential processes and normal processes on the VM increases because memory copy for replacement of process information is issued frequently. For this reason, the administrator must choose the essential process with consideration for the VM's total performance.

7. Conclusion

We proposed the replacement of process information for essential process with a VMM to complicate process identification by adversaries. Because adversaries identify an attack target process with available process information, a replacement of that process information by our system is effective in avoiding attacks of that kind. The proposed system is implemented by modifying the VMM and with a Control AP on the manager VM. Modification to guest OSes and APs on each VM is unnecessary.

An experiment using a prototype of the proposed system based on the Xen hypervisor showed that an essential process name was successfully replaced with a dummy name. In addition, an experiment showed that process termination with the killall command fails if the target is essential service and the proposed method is applied to that process. Performance evaluation results showed that the overhead incurred by the proposed system in context switch time is less than 15%. In addition, it is confirmed that the performance overheads in Linux kernel compilation is considered negligible.

Future work shall include the implementation of the access control function to the process information, evaluation with various types of real-world malware, and extensive performance analysis of the proposed system.

Acknowledgments This work was supported by JSPS KAKENHI Grant Number 13J08339.

References

[1] Jiang, X., Wang, X. and Xu, D.: 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).

[2] Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, *Proc. 11th International Symposium on Recent Advances in Intrusion Detection*, pp.1–20 (2008).

[3] Hsu, F.-H., Wu, M.-H., Tso, C.-K., Hsu, C.-H. and Chen, C.-W.: Antivirus Software Shield Against Antivirus Terminators, *IEEE Trans. Inf. Forensic Secur.*, Vol.7, No.5, pp.1439–1447 (2012).

[4] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proc. Network and Distributed Systems Security Symposium*, pp.191–206 (2003).

[5] Dinaburg, A., Royal, P., Sharif, M. and Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions, *Proc. 15th ACM Conference on Computer and Communications Security*, pp.51–62 (2008).

[6] Sharif, M.I., Lee, W., Cui, W. and Lanzi, A.: Secure in-VM Monitoring using Hardware Virtualization, *Proc. 16th ACM Conference on Computer and Communications Security*, pp.477–487 (2009).

[7] Srinivasan, D., Wang, Z., Jiang, X. and Xu, D.: Process Out-grafting: An Efficient "out-of-VM" Approach for Fine-grained Process Execution Monitoring, *Proc. 18th ACM Conference on Computer and Communications Security*, pp.363–374 (2011).

[8] Wu, Y.-S., Sun, P.-K., Huang, C.-C., Lu, S.-J., Lai, S.-F. and Chen, Y.-Y.: EagleEye: Towards mandatory security monitoring in virtualized datacenter environment, *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp.1–12 (2013).

[9] Dewan, P., Durham, D., Khosravi, H., Long, M. and Nagabhushan, G.: A hypervisor-based system for protecting software runtime memory and persistent storage, *Proc. 2008 Spring Simulation Multiconference*, pp.828–835 (2008).

[10] McCune, J., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V. and Perrig, A.: TrustVisor: Efficient TCB Reduction and Attestation, *Proc. 2010 IEEE Symposium on Security and Privacy*, pp.143–158 (2010).

[11] Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, *SIGOPS Oper. Syst. Rev.*, Vol.41, No.6, pp.335–350 (2007).

[12] Xiong, X., Tian, D. and Liu, P.: Practical Protection of Kernel Integrity for Commodity OS from Untrusted Extensions, *Proc. Network and Distributed Systems Security Symposium* (2011).

[13] Srivastava, A. and Giffin, J.: Efficient Protection of Kernel Data Structures via Object Partitioning, *Proc. 28th Annual Computer Security Applications Conference*, pp.429–438 (2012).

[14] Boeck, B., Huemer, D. and Tjoa, A.M.: Towards More Trustable Log Files for Digital Forensics by Means of "Trusted Computing", *International Conference on Advanced Information Networking and Appli-*

cations, pp.1020–1027 (2010).
[15] Bahram, S., Jiang, X., Wang, Z., Grace, M., Li, J., Srinivasan, D., Rhee, J. and Xu, D.: DKSM: Subverting Virtual Machine Introspection for Fun and Profit, *2010 29th IEEE Symposium on Reliable Distributed Systems*, pp.82–91 (2010).
[16] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *SIGOPS Oper. Syst. Rev.*, Vol.37, No.5, pp.164–177 (2003).

Editor's Recommendation

The authors propose method that obscures process identification based on process-related information without any modification to OSes and APs. We thus expect that this paper provides further research developments in software security.

(Program Chair of Computer Security Symposium 2013, Isao Echizen)

Masaya Sato received his B.E., M.E. and Ph.D. degrees from Okayama University, Japan in 2010, 2012 and 2014, respectively. In 2013 and 2014 he was a Research Fellow of the Japan Society for the Promotion of Science. He has been an Assistant Professor of Graduate School of Natural Science and Technology at Okayama University. His research interests include computer security and virtualization technology. He is a member of IPSJ and IEICE.

Toshihiro Yamauchi received his B.E., M.E. and Ph.D. degrees in computer science from Kyushu University, Japan in 1998, 2000 and 2002, respectively. In 2001 he was a Research Fellow of the Japan Society for the Promotion of Science. In 2002 he became a Research Associate in Faculty of Information Science and Electrical Engineering at Kyushu University. He has been serving as an Associate Professor of Graduate School of Natural Science and Technology at Okayama University since 2005. His research interests include operating systems and computer security. He is a member of IPSJ, IEICE, ACM, USENIX and IEEE.

Hideo Taniguchi received a B.E. degree in 1978, an M.E. degree in 1980 and a Ph.D. degree in 1991, all from Kyushu University, Fukuoka, Japan. In 1980, he joined NTT Electrical Communication Laboratories. In 1988, he moved to Research and Development Headquarters, NTT DATA Communications Systems Corporation. He has been an Associate Professor of Computer Science at Kyushu University since 1993 and a Professor of the Faculty of Engineering at Okayama University since 2003. His research interests include operating system, real-time processing and distributed processing. He is the author of Operating Systems (Shoko Publishing Co. Ltd), etc. He is a fellow of IPSJ. He is a member of IEICE and ACM.