Complicating Process Identification by Replacing Process Information for Attack Avoidance

Masaya Sato and Toshihiro Yamauchi

Graduate School of Natural Science and Technology, Okayama University,
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
m-sato@swlab.cs.okayama-u.ac.jp, yamauchi@cs.okayama-u.ac.jp

Abstract. Security-critical software is open to attacks by adversaries that disable its functionality. To decrease the risk, we propose an attack avoidance method for complicating process identification. 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 and application programs are unnecessary.

Keywords: Attack avoidance, process information, virtual machine.

1 Introduction

Attacks exploiting vulnerabilities in programs to illegally control computers are increasing. Therefore, software is developed to prevent such attacks and mitigate their effects. However, attacks still succeed when they are able to deactivate such software. For instance, Agobot [1] has the functionality to stop anti-virus software. T0rnkit [2] and Dica [3] stop log collectors in order to hide the installation process of malware from the system administrator of a target computer. The risk of damages to target computers increases when protective software (essential services) is deactivated. Therefore, it is an important challenge to detect and prevent attacks on programs such as anti-virus software and log collector to reduce damage to a computer, and to avoid the attack.

To prevent attacks on essential services, methods using a virtual machine monitor (VMM) have been proposed [4], [5]. These methods prevent the essential services from being affected by isolating them from the target computer using virtualization technology. Research [4] reveals a method for offloading the intrusion detection system (IDS) from one virtual machine (VM) to another. Moreover, Jiang et al. proposed a method for malware detection using a VMM [5]. However, these methods do not utilize existing essential services and software already installed and operational. Research [6] has proposed a method to prevent anti-virus software from being terminated without the consciousness of the anti-virus software users. This method monitors Windows APIs by SSDT hooking and filter out hazardous API calls that will terminate anti-virus software. This method is effective for termination of anti-virus software using API calls. However, this method is vulnerable to SSDT (System Service Descriptor Table) patching commonly used by rootkits because this method replaces some SSDT entries to their handlers. Protecting the system from kernel-level malware is a challenging problem.

To address these problems, this paper proposes an attack avoidance method to complicate process identification for 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 a process is 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 information, replacing the process information complicates the identification of an attack target.
- 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.
- An evaluation using a prototype of the proposed system shows the effectiveness of the method for attack avoidance.

2 Background

2.1 Attacks for Anti-virus Software

Agobot 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 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 an essential services are required.

2.2 Existing Countermeasures for Attacks

Research into an offloading host-based intrusion detection system (IDS) with a VMM is proposed in VMwatcher [4]. Implementing an IDS by modifying a VMM makes it difficult to attack the IDS. In a same manner, NICKLE [5], 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.

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 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 application 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 kernel is different from the information a VMM collects. This semantic gap makes it difficult to port functions from existing software to a VMM.

3 Attack Avoidance Method for Complication of Process Identification

3.1 Purpose

The following explains the purposes of our research:

Purpose 1) Avoidance of attacks to essential services
Purpose 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. Thus, it is preferable to avoid attacks without modifying existing software.

3.2 Basic Idea

To achieve the purposes outlined in Section 3.1, we propose complicating process identification to avoid attacks by replacing the process information for essential processes providing the security 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 codes 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.

3.3 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.

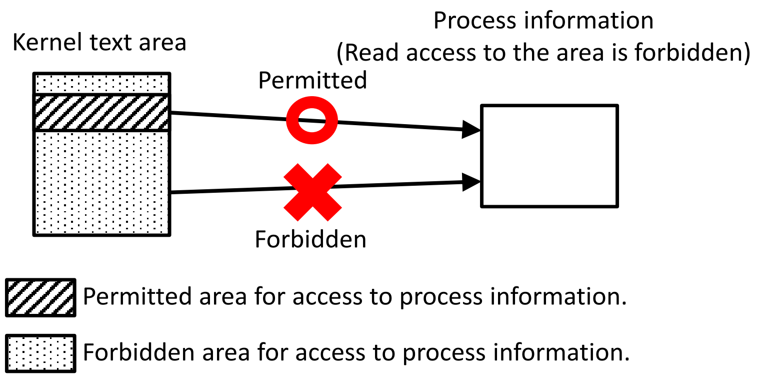

Kernel text area
Process information
(Read access to the area is forbidden)
Permitted
Forbidden
Permitted area for access to process information.
Forbidden area for access to process information.

Fig. 1. Access control to process information.

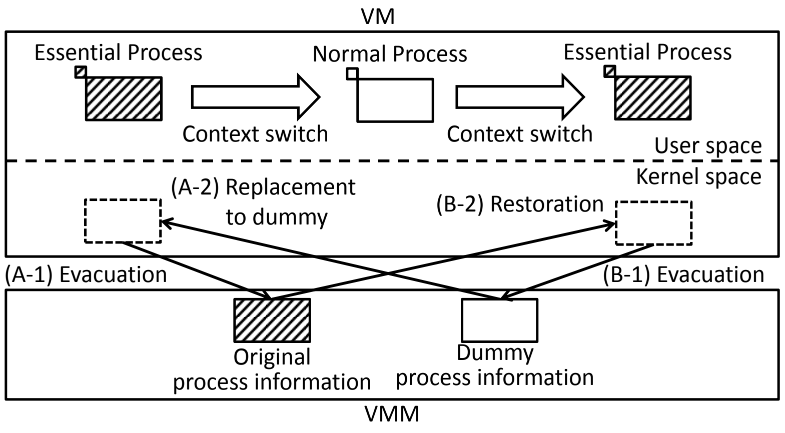


Fig. 2. Replacement of process information between essential process and normal process

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 Figure 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 4.

3.4 Method for Identifying Essential Processes and Countermeasures

Identification method. 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.

Adversaries can identify an essential process by comparing the processing time of the context switch between an essential process and a normal process or by continuously monitoring the process information. 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.

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, 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.

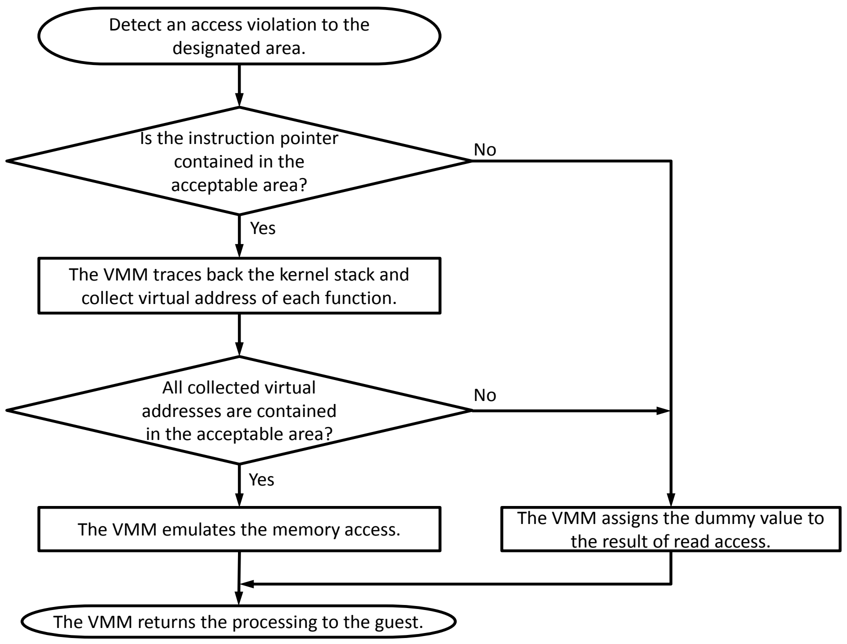
Countermeasures. To conceal the difference in the processing times of context switches, it might suffice to apply the same processing time to normal processes. This done, the difference in the processing times of context switches between the essential processes and the others becomes meaningless. However,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

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 Figure 3. 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 emulates the read access and returns an original value. If not, the VMM returns the dummy value.

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 [7]. 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.


Detect an access violation to the designated area.
Is the instruction pointer contained in the acceptable area?
No
Yes
The VMM traces back the kernel stack and collect virtual address of each function.
All collected virtual addresses are contained in the acceptable area?
No
Yes
The VMM emulates the memory access.
The VMM assigns the dummy value to the result of read access.
The VMM returns the processing to the guest.

.

Fig. 3. Determination procedure for access to process information.

3.5 Structure of the Proposed System

The overview of the proposed system is shown in Figure 4. 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 is evacuated from the VM and stored in 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.

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.

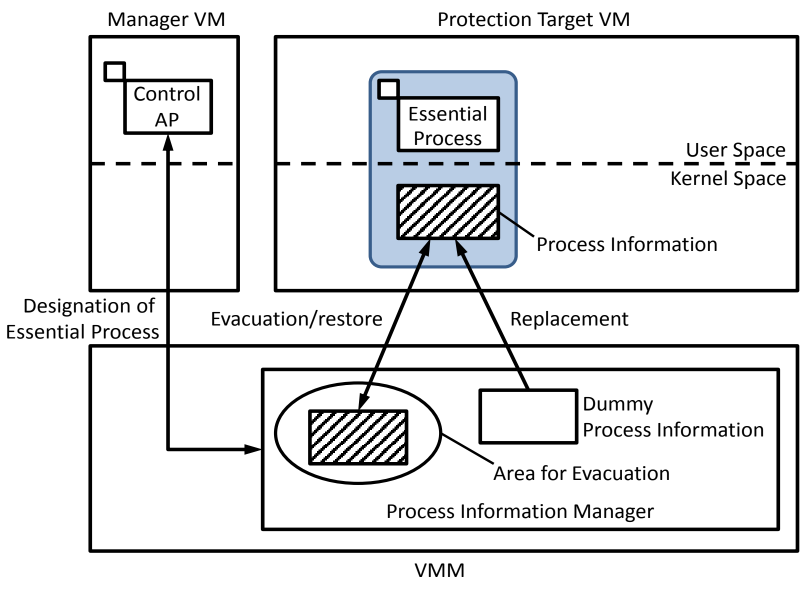


Fig. 4. Overview of proposed system.

3.6 Limitations

Because the proposed method only makes process identification difficult, the essential processes are visible from adversaries. Therefore, if an adversary stops a process at random, it is possible that an essential process will be stopped. To prevent attacks on essential processes, regulating access control to process information is effective. However, because attack prevention diverges from our stated purpose, we do not discuss strategies in attack prevention.

Because the proposed methods replace 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.

4 Replacement Method of Process Information

4.1 Replacement Target

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.

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.

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 *Terminate-Process()* 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 un-common 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.

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 a 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 were the dummy information always the same.

4.2 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.

4.3 Acquisition of Process Information in Guest OS

Acquisition of process information of current process. As shown in Figure 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 the process information in a guest OS from the RSP register. In this regard, the VMM must hold the definitions for each structure beforehand.

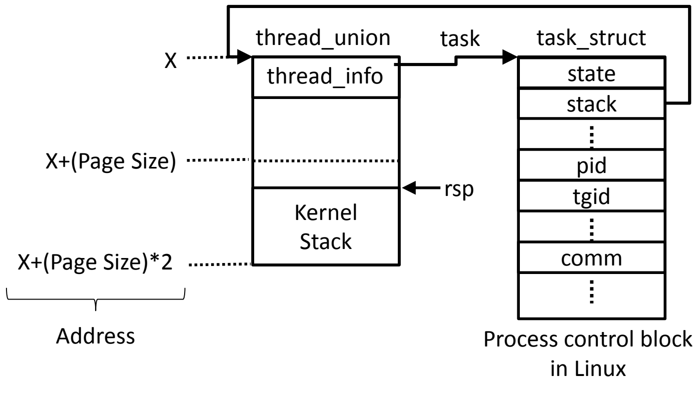


Fig. 5. Relation between *thread_union* and *task_struct*.

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 method 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 look-up 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 [8] 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.

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

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.

4.4 Designation of Essential Process

The proposed method requires the Control AP to designate the essential processes prior to replace their process information. Because of the structure of the proposed method, 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 method 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.

4.5 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.

5 Evaluation

5.1 Environment for Evaluation

The environment for evaluation is shown in Table 2. We evaluated the proposed system with an Intel Core i7-2600. The protection target VM is fully virtualized by Intel VT-x.

Table 2. Environment for evaluation.

VMM	Xen 4.2.0
OS (Manager VM)	Debian 7.3 (Linux 3.2.0 64-bit)
OS (Protection target VM)	Debian 7.3 (Linux 3.2.0 64-bit)

5.2 Purpose and Evaluation Method

The purpose of the evaluation is to confirm effectiveness of the proposed method to attacks. In our experiment, we examined whether the name of an essential process was replaced when the proposed method was applied to the essential processes. 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 method, we assumed *syslogd* as an essential process and changed its name to *apache2*.

5.3 Evaluation Results

On the protection target VM, we listed the names for all processes. The list did not contain *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 method.

6 Related Work

Some researchers have proposed methods for preventing the illegal alteration of memory contents [9], [10]. While these researches 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.

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. Our system is similar to the approach found in Sentry [12]. 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.

ANSS [6] 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.

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 method 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. This result indicates that attacks based on the process name are avoidable with the proposed method.

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

References

1. F-Secure: Agobot, http://www.f-secure.com/v-descs/agobot.shtml
2. F-Secure: Tornkit, http://www.f-secure.com/v-descs/torn.shtml
3. Packetstorm: dica.tgz, http://packetstormsecurity.com/files/26243/dica.tgz.html
4. Jiang, X., Wang, X. and Xu, D.: Stealthy Malware Detection Through VMM-Based "Out-of-

the-Box" Semantic View Reconstruction, Proc. 14th ACM Conference on Computer and Communications Security (CCS '07), pp.128–138 (2007)

5. Riley, R., Jiang, X. and Xu, D.: Guest-Transparent Prevention of Kernel Rootkits with VMM-Based Memory Shadowing, Lecture Notes in Computer Science, Vol.5230, pp.1–20 (2008)
6. Fu-Hau, H., Min-Hao, W., Chang-Kuo, T., Chi-Hsien, H. and Chieh-Wen, C.: Antivirus Software Shield Against Antivirus Terminators, IEEE Transactions on Information Forensics and Security, vol.7, no.5, pp.1439–1447 (2012)
7. Bahram, S.; Xuxian Jiang; Zhi Wang; Grace, M.; Jinku Li; Srinivasan, D.; Junghwan Rhee; Dongyan Xu: DKSM: Subverting Virtual Machine Introspection for Fun and Profit, 29th IEEE Symposium on Reliable Distributed Systems, pp.82–91 (2010)
8. 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 Opr. Syst. Rev., Vol.37, No.5, pp.164–177 (2003)
9. Dewan, P., Durham, D., Khosravi, H., Long, M., Nagabhushan, G.: A Hypervisor-Based System for Protecting Software Runtime Memory and Persistent Storage, Proc. 2008 Spring Simulation Multiconference (SpringSim'08), pp.828–835 (2008)
10. McCune, J.M., Yanlin, L., Nung Q., Zongwei, Z., Datta, A., Gligor, V., 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, Proc. 21st ACM SIGOPS Symposium on Operating System Principles, pp.335–350 (2007)
12. Srivastava, A. and Giffin, J.: Efficient Protection of Kernel Data Structures via Object Partitioning, Proc. 28th Annual Computer Security Application Conference (ACSAC'12), pp.429–438 (2012)