Design of a Function for Tracing the Diffusion of Classified Information for File Operations with a KVM

Shota Fujii, Toshihiro Yamauchi and Hideo Taniguchi

Graduate School of Natural Science and Technology, Okayama University, Japan

fujii@swlab.cs.okayama-u.ac.jp {yamauchi, tani}@cs.okayama-u.ac.jp

Abstract. Cases of leaked classified information are increasingly common. To address this problem, we developed a function for tracing the diffusion of classified information within an operating system. However, this function suffers from the following two problems. First, in order to introduce the function, the operating system's source code must be modified. Second, there is a risk that the function will be disabled when the operating system is attacked. Thus, we designed a function for tracing the diffusion of classified information in a guest operating system using a virtual machine monitor. By using a virtual machine monitor, it is possible to introduce the proposed function in various environments, because the operating system's source code need not be modified. In addition, attacks aimed at the proposed function are made more difficult, because the virtual machine monitor is isolated from the operating system. This paper describes the implementation of the proposed function for file operations and child process invocation with the kernel-based virtual machine.

Keywords: Data Leak Prevention, Virtualization, Semantic Gap

1 Introduction

As personal information becomes increasingly valuable, so too is the need to prevent information leaks. According to an analysis [1] of incidents of leaked personal information, leaks often occur as a result of inadvertent handling and mismanagement, and this accounts for approximately 57% of all known cases of information leaks. To prevent information leaks, it is important that the user grasp the situation surrounding classified information. On the other hand, incidents that aim at stealing classified information occur with increasing frequency. In such a context, there is always the risk of increased damage when the victim cannot detect the information leak. To trace the status of classified information in a computer, and to manage the resources that contain classified information, we developed a function for tracing the diffusion of classified information [2] (specifically, an operating-system-based tracing function). This function manages any process that has the potential to diffuse classified information. In addition, the function represents the extent of the diffusion using a directed graph [3], and it traces the diffusion of classified information in multiple computers [4]. However, the function cannot be introduced in a closed-source

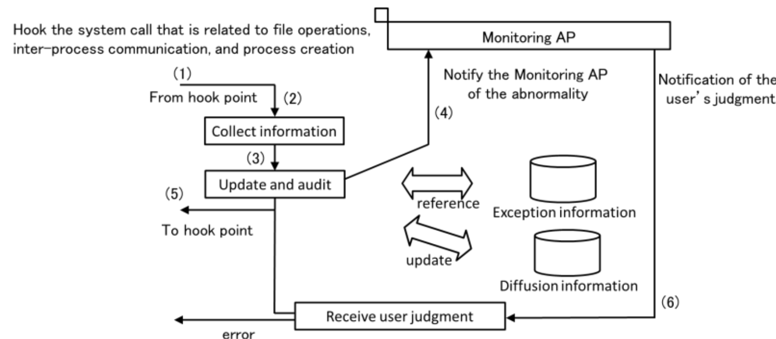

Fig. 1. Overview of the OS-based tracing function

operating system (OS) such as Windows, because implementing it requires modifying the source code. Further, when the kernel version is updated, the function must adapt to the newly updated kernel.

To resolve these problems, we designed a function for tracing the diffusion of classified information in a guest OS using a virtual machine monitor (VMM). This VMM-based tracing function is implemented by modifying the VMM, rather than the guest OS. Therefore, the VMM-based tracing function can be implemented without modifying the OS's source code. Further, it is expected that attacks specifically targeting this function will be rare, because the VMM is more robust than the OS.

This paper describes the implementation of the function for file operations and child process invocation with a kernel-based virtual machine (KVM).

2 OS-based Function for Tracing the Diffusion of Classified Information

2.1 Classified Information Diffusion Path

The OS-based tracing function [2] manages any file and process that has the potential to diffuse classified information. Classified information can be diffused by any process that involves opening the classified file, reading its content, or communicating with another process or file. Therefore, the diffusion of classified information is caused by the following operations:

(1) File operation
(2) Inter process communication
(3) Child process invocation

The OS-based tracing function traces the diffusion of classified information by monitoring these operations.

2.2 Overview of the OS-based Tracing Function

Figure 1 shows an overview of the OS-based tracing function. The OS-based tracing function traces the diffusion of classified information as follows:

(1) System calls that are related to the diffusion of classified information are hooked.
(2) The OS-based tracing function collects information for tracing the diffusion of classified information such as the file that is handled by the system call or the transmission-destination process.
(3) The OS-based tracing function updates the diffusion information using the information that is collected in (2) and audits its potential for leaking classified information.
    (A) When the audit discovers the possibility of a classified information leak, it notifies the monitoring application program (AP).
    (B) When the audit does not reveal any possibility of leaked classified information, control is returned to the system call.
(4) After receiving the results of the user's judgment from the monitoring AP, the OS-based tracing function controls the system call in accordance with the user's judgment.
    (A) When the user's judgment is affirmative, the system-call processing is continued.
    (B) When the user's judgment is negative, the system-call processing is terminated as an error.

In addition, the OS-based tracing function excludes files and processes that are unrelated to the diffusion of classified information. These files and processes are registered with exception information.

2.3 Problems with the OS-based Tracing Function

The tracing function has the following problems:

Problem 1 The source code must be modified before it can be introduced.

In order to introduce the OS-based tracing function, it is necessary to modify the OS's source code. Therefore, the OS-based tracing function cannot be introduced in closed-source OSs such as Windows. Furthermore, when the kernel version of the OS is updated, the OS-based tracing function must again modify the source code after the OS is updated.

Problem 2 Risk of an attack invalidating the tracing function

The OS-based tracing function is implemented in the OS. Therefore, an adversary or a malicious user can invalidate the function by attacking the OS. Should the function be invalidated, it becomes difficult to prevent information from being leaked and grasp the location of classified information.

In this paper, we propose a method that resolves both problems.

3 Function for Tracing the Diffusion of Classified Information in a Guest OS using a Virtual Machine Monitor

3.1 Requirements

To resolve the problems detailed above in Section 2.3, the following are required:

Requirement 1 The OS's source code must not be modified.

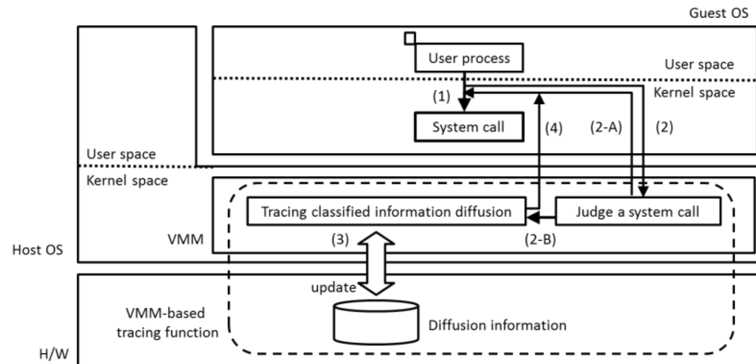

Fig. 2. Overview of the VMM-based tracing function

One solution to **Problem 1** is to avoid modifying the OS's source code altogether. This ensures that the function can be implemented in closed-source OSs such as Windows.

Requirement 2 The function should be isolated from the OS.

Isolating the function from the OS is a solution to **Problem 2**. Such a solution makes it difficult for an adversary or a malicious user to attack the function directly.

3.2 Overview of the VMM-based Tracing Function

The VMM-based tracing function is functionally equivalent to the OS-based tracing function. Figure 2 shows an overview of the VMM-based tracing function. The VMM-based tracing function traces the diffusion of classified information as follows:

(1) A user program in the guest OS requests a system call.

(2) The VMM-based tracing function hooks the system call in the guest OS from the VMM. After identifying the hooked system call, the following system-call processing is performed.

  (A) When the hooked system call is unrelated to the diffusion of classified information, control is returned to the guest OS and the system-call process is continued.

  (B) When the hooked system call is related to the diffusion of classified information, the VMM-based tracing function collects the information needed to trace its diffusion.

(3) The VMM-based tracing function updates the diffusion information using the information that is collected in (2-B).

(4) Control is returned to the guest OS and the system-call process is continued.

Given these steps, the VMM-based tracing function provides the guest OS with functions that are equivalent to the OS-based tracing function, without the need to modify the OS source code.

3.3 Tasks

To implement the VMM-based tracing function, the following tasks are required:

Task 1 Collecting the system-call information with the VMM.

Classified information is diffused by the system call. Therefore, it is necessary to hook the system call. In addition, the VMM-based tracing function collects the system call's information owing to judgment whether the system call is related to classified information diffusion.

Task 2 Collecting the OS information with the VMM.

The VMM-based tracing function manages any file or process that has the potential to diffuse classified information. Therefore, it is necessary to collect the information from the OS, such as the processes that are running, their transmission destination, and the files handled by the processes that are running.

In Sections 3.4 and 3.5, we describe the procedure by which the above tasks are accomplished. Further, this procedure is tailored for a 64-bit version of Linux in which the system call is executed by SYSCALL/SYSRET.

3.4 Collecting System Call Information with the Virtual Machine Monitor

Hooking a System Call Entry. The VMM-based tracing function hooks the system-call entry (viz., SYSCALL). By hooking SYSCALL, it is possible to detect system-call requests. In order to hook SYSCALL, the VMM-based tracing function sets the value of guest OS's MSR_LSTAR to the value of an unused page address. Executing SYSCALL changes the instruction pointer to the value in MSR_LSTAR, resulting in a page fault. Therefore, the VMM-based tracing function can hook the SYSCALL with the VMM by detecting page faults in the guest OS.

Hooking the System Call Exit. Each system call returns information concerning the success or failure of the system call, and details of the file handled by the system call as a return value. It is necessary to collect details about the file that is handled by the running process so that the VMM-based tracing function can trace the diffusion of classified information. Thus, the VMM-based tracing function hooks the system-call exit (viz., SYSRET). By hooking SYSRET, it is possible to obtain the system call's return value. In order to hook SYSRET, the VMM-based tracing function sets the breakpoint-address register to SYSRET's address. A breakpoint-address register specifies the breakpoint address and a debug exception is generated when a memory access is made to the breakpoint address. Thus, a debug exception occurs upon executing SYSRET. Therefore, the VMM-based tracing function can hook SYSRET with the VMM by detecting debug exceptions in the guest OS.

Collecting Information. It is necessary to judge whether the hooked system call is related to the diffusion of classified information. To identify the system call, the VMM-based tracing function uses a system-call number. In addition, it is necessary for the VMM-based tracing function to identify the transmission-destination file or process. A system call takes the file or process information to an argument. By obtaining the system call's argument, it is consequently possible to identify the transmission-destination file or process. Furthermore, as we have already described, the VMM-based tracing function obtains the system call's return value and utilizes the return value for identifying the transmission-destination file or process.

3.5 Collecting OS Information with the Virtual Machine Monitor

The VMM-based tracing function traces the diffusion of classified information using information from the OS, such as process information and file information. Then, the semantic gap must be bridged so that the VMM-based tracing function can obtain the OS information with the VMM. The semantic gap is the gap between the guest OS as it is viewed from the outside and the view of it from the inside. To bridge the semantic gap, the VMM-based tracing function constructs a semantic view by retrieving information about the guest OS beforehand.

4 Implementation

4.1 Environment

In this chapter, we describe the implementation of the VMM-based tracing function, using a KVM as the VMM and a 64-bit Linux OS with the 3.6.10 kernel as the guest OS. The VMM-based tracing function detects requests for system calls by hooking SYSCALL, and it obtains return values by hooking SYSRET. Therefore, the system call in the guest OS is executed by SYSCALL/SYSRET. In addition, the guest OS is fully virtualized with Intel VT.

4.2 Tracing Classified Information at Each Path

Current Status. The VMM-based tracing function traces the diffusion of classified information by hooking a system call related to file operations, inter process communication, and child process invocation. We have currently implemented the VMM-based tracing function exclusively for file operations and child process invocation. In the future, we will implement the VMM-based tracing function for inter process communications. The following describes the implementation of the VMM-based tracing function.

File Operation. The VMM-based tracing function hooks the open(), read(), write(), and close() system calls that are related to file operations. In addition, to trace the diffusion of classified information by file operations, the VMM-based tracing function collects the following information:
(1) Current-process identifier
(2) Identifier of the file that is handled by the system call
It is necessary for the VMM-based tracing function to collect the current-process identifier in order to judge whether the process requesting the system call is a management target when the VMM-based tracing function hooks each system call. To identify the current process, the VMM-based tracing function uses the process ID (PID). The VMM-based tracing function obtains the PID when the function hooks the SYSCALL. Moreover, it is necessary for the VMM-based tracing function to identify the file that is handled by the system call when the VMM-based tracing function judges whether file that is read is a management target, and to register the written file with diffusion information. To identify the file that is handled by the system call, the

VMM-based tracing function uses the inode number. The VMM-based tracing function obtains the inode number by following the data structure from process-control block to the file structure. Then, the VMM-based tracing function identifies the inode number using the file descriptor. The file descriptor is obtained with system call's return value in cases where open() is hooked. Likewise, the file descriptor is obtained by the system call's argument in cases where read(), write(), and close() are hooked.

Child Process Invocation. The VMM-based tracing function hooks the clone() system call, which is related to child process invocation. Moreover, in order to trace the diffusion of classified information by child process invocation, the VMM-based tracing function collects the following information:
(1) System call's product identifier
(2) Parent-process identifier
(3) Child-process identifier
The clone() system call creates not only a new process but also a new thread. The threads in the same process share resources such as information related to the files that are open. Therefore, classified information is not diffused from the process in the case of thread creation. On the other hand, resources are diffused from the parent process to the child process in the case of child process invocation. Thus, it is necessary to judge whether the product of clone() is a process or a thread. To determine this, the VMM-based tracing function uses the CLONE_THREAD flag. If the CLONE_THREAD is set, clone() creates the thread. Consequently, by using the CLONE_THREAD flag, it is possible to judge whether the product of clone() is a process or thread. The CLONE_THREAD flag is obtained from the clone() argument when the VMM-based tracing function hooks the clone() system-call entry.

Moreover, when the parent process is a management target, there is a risk that classified information will be diffused to the child process. Therefore, to judge whether the parent process is a management target, it is necessary to collect the parent-process identifier. To do so, the VMM-based tracing function uses the parent process' PID. The parent process' PID is obtained from the process-control block when the VMM-based tracing function hooks the clone() system-call entry.

Furthermore, the VMM-based tracing function registers the child process with the diffusion information when the VMM-based tracing function judges that there is a possibility that the classified information will be diffused to the child process. Thus, the child-process identifier must be obtained. To identify the child process, the VMM-based tracing function uses the child process' PID. The clone() system call returns the thread ID (TID) of the child process. When clone() creates a new process, the TID is identical to the PID. Thus, the child process' PID is obtained from the return value of clone() when the VMM-based tracing function hooks the clone() system-call exit.

5 Related Work

TightLip [5] is a privacy-management system that swaps an original process for a dummy process, called a "Doppelgangers," when a process that includes sensitive data attempts to write a different buffer to the network. This protects this data because the Doppelgangers does not itself contain sensitive data. Therefore, it is possible to

prevent sensitive data from being leaked. The VOFS [6] runs a Primary Guest VM that contains the user's main OS and a SVFS VM for preventing information leaks. When a user attempts to open a sensitive file, the Primary Guest OS contacts the SVFS VM and requests that it show the sensitive file that is stored in the content server. Then, the SVFS VM prevents sensitive files from being leaked by disabling device outputs. TaintEraser [7] is a method for tracing the diffusion of classified information using Dynamic Taint Analysis. Dynamic Taint Analysis tracks information that may have been tainted by other data. Subsequently, if the tainted data is written to another location in the memory, that destination is marked as tainted. Thus, it is possible to follow the classified information that is tainted.

6 Conclusion

This paper described the design and implementation of a VMM-based tracing function for file operations and child process invocation with a KVM. The VMM-based tracing function can be implemented without modifying the OS's source code. Thereby, we expect that the VMM-based tracing function can be introduced in various environments. Moreover, it is difficult to attack the function directly, owing to the isolation of the VMM from the OS. Furthermore, even if the kernel version is updated, the VMM-based tracing function will continue to be available, provided that the system-call specifications and the data structure remain unchanged.

In future work, we will implement the VMM-based tracing function for inter process communication, and we will reduce the overhead generated by the VMM-based tracing function, and evaluate its performance.

References

1. Japan Network Security Association, 2008 Information Security Incident Survey Report, http://www.jnsa.org/result/incident/data/2008incident_survey_e_v1.0.pdf
2. Tabata, T., Hakomori, S., Ohashi, K., Uemura, S., Yokoyama, K., Taniguchi, H.: Tracing Classified Information Diffusion for Protecting Information Leakage. IPSJ Journal. Vol.50, No.9, pp. 2088–2012 (2009) (in Japanese)
3. Nomura, Y., Hakomori, S., Yokoyama, K., Taniguchi, H.: Tracing the Diffusion of Classified Information Triggered by File Open System Call. In: Proc. 4th Int. Conf. on Computing, Communications and Control Technologies, pp. 312–317 (2006)
4. Otsubo, N., Uemura, S., Yamauchi, T., Taniguchi, H.: Design and Evaluation of a Diffusion Tracing Function for Classified Information Among Multiple Computers. In: 7th FTRA International Conference on Multimedia and Ubiquitous Engineering, pp. 235–242 (2013)
5. Yumerefendi, A. R., Mickle, B., Cox, L. P.: TightLip: Keeping Applications from Spilling the Beans. In: Proc. of the 4th USENIX conference on Networked systems design & implementation (2007)
6. Borders, K., Zhao, X., Prakash, A.: Securing Sensitive Content in a View-only File System. In: Proc. of the ACM Workshop on Digital Rights Management, pp. 27–36 (2006)
7. Zhu, D. Y., Jung, J., Song, D., Kohno, T., Wetherall, D.: TaintEraser: Protecting Sensitive Data Leaks Using Application-level Taint Tracking. In: ACM SIGOPS Operating Systems Review, pp. 142–154 (2011)