VMM-Based Log-Tampering and Loss Detection Scheme

Masaya Sato and Toshihiro Yamauchi
Graduate School of Natural Science and Technology
Okayama University
Japan
m-sato@swlab.cs.okayama-u.ac.jp, yamauchi@cs.okayama-u.ac.jp

Abstract

Logging information about the activities that placed in a computer is essential for understanding its behavior. In Homeland Security, the reliability of the computers used in their activities is of paramount importance. However, attackers can delete logs to hide evidence of their activities. Additionally, various problems may result in logs being lost. These problems decrease the dependability of Homeland Security. To address these problems, we previously proposed a secure logging scheme using a virtual machine monitor (VMM). The scheme collects logs and isolates them from the monitored OS. However, the scheme cannot store them automatically. Thus, logs in memory are lost when the computer is shutdown. Further, if the logs are not stored, it is impossible to detect incidents of tampering by comparing the logs of the monitored OS with those of the logging OS. To address these additional problems, this paper proposes a log-storing module and a tamper detection scheme. The log-storing module automatically stores logs collected by the logging module, and tamper detection is realized by comparing these stored log files with those of the monitored OS. We implemented the log-storing module and realized the tamper detection scheme. Evaluations reveal the effectiveness of the tamper detection scheme.

Keywords: *Log protection, detecting log tampering, syslog, digital forensics, virtualization technology*

1 Introduction

The countermeasure for terrorism is one important topic in Homeland Security. In the field of counter-terrorism, enormous quantity of data is gathered and analyzed for the planning of countermeasures. Computers and networks are used to gather and analyze data, computer science is deeply committed to homeland defense and security. In the field of computer science, countermeasures are considered for cyber terrorism as an activity in Homeland Security. Recently, information technology is used as a tool to control infrastructures. Cyber terrorism is able to cause critical damage on infrastructures in low cost. Thus, the countermeasure for cyber terrorism has been discussed.

However, the countermeasures might be weakened by attacking on the data gathered for Homeland Security. Therefore, the protection of the data is important. The protection of the logs of the APs is also necessary to ensure the validity of gathered information.

The computer terrorism has two characters: anonymity and the lack of evidences of attacks. In computer terrorism, it is difficult to acquire the information that specifies the attacker. Because there are no evidences left on attacks using network, the logs that records the behavior of the systems are important. For this reason, the protection of the information is necessary for the prevention and investigation of computer terrorism.

Insider threat study is important issue in the field of Homeland Security [1]. The purpose of insider threat study is to help understand, detect, and prevent bad insider activities. Log protection is one of the most important techniques of insider threat study because logs that contain records of system are necessary for forensics to specify attacker's activities [2, 3].

Digital forensics is a method or technology for addressing these problems. This is a scientific method or research technology for court actions, which allows us to explain the validity of the electronic records. Many researchers are working in this area of the protection of logging information [4-8].

Furthermore, in the United States, the federal chief information officer announced that the government starts using cloud computing in the federal government in September 2009 [9]. The privacy office of the department of Homeland Security is deeply involved in the initiative from the beginning. From these reasons, in the field of Homeland Security, it is strongly required for security mechanisms to adapt to the cloud computing environment. A firewall is efficient solution for security of the cloud computing environment because the users need to connect to it via network. However, the importance of logs is remaining because server and network logs are used to validate or confirm firewall rules [10].

Syslog is commonly used as a logging program in Linux. In this case, the logging information generated by the AP (user log) and kernel (kernel log) is collected by syslog. Syslog writes logs to file according to the policy, so attackers can tamper with logs by modifying the policy. Moreover, if the syslog program itself is attacked, the log files written are not reliable. In addition, the kernel log is stored in a ring buffer, and therefore, since the kernel log is collected on a regular schedule, if many logs are generated and stored in the ring buffer before the next collecting time, old logs may be overwritten by new logs. As described above, the user log and kernel log can be tampered with or lost.

To address these problems, we proposed a logging system to prevent tampering and loss of logs with the virtual machine monitor (VMM) [11]. In this system, the OS that should be monitored (the monitored OS) works on the virtual machine (VM). Logs in the monitored OS are collected by the VMM without any modification of the monitored OS's kernel source codes. Because the system collects logs just after the output of logs, any possibilities for tampering are excluded. In addition, no kernel logs are lost through the buffer being overwritten by new kernel logs.

Because the proposed system uses virtualization technology, it is compatible with cloud computing environment. Thus, the proposed system is suitable for providing the federal cloud computing environment with higher security.

Our previously proposed system has two problems.

(1) Loss of logged information when the machine is powered-off or restarted.
(2) Difficulty in detecting incidents of log tampering by comparing logs.

The system that we proposed earlier keeps logs of the monitored OS in the memory region of the VMM. As a result, logs in the memory are lost when the machine is powered-off or rebooted. Thus, if loss does occur, it is unable to detect loss and tampering of logging information. To solve these problems, this paper proposes a log-storing module that stores logs collected by the logging module to files. The log-storing module copies logs to the logging OS in as soon as they are collected from the monitored OS. The logging OS receives the logs and stores them in files via the syslog daemon.

The logging module in our previously proposed system also cannot compare logs directly. Consequently, it is unable to detect log tampering immediately. This paper also proposes a log-tampering detection function. The function compares the logs of the monitored OS with those of the logging OS. By comparing these log files, we can detect incidents of log tampering in the monitored OS. Moreover, if the logs in the monitored OS are tampered with, our proposed function can detect exactly where and how the logs were tampered with.

The contributions made in this paper are as follows:

(1) A log-storing module that enables the VMM to store logs in files in a separate VM is proposed. The logging module previously proposed in [11] is not able to store logs to the VM automatically. Consequently, accidental shutting down of a computer before the log-storing command execution may result in the logs currently in the memory being lost. The log-storing module enhances the logging mechanism by ensuring that log files are preserved.
(2) A scheme that detects incidents of log tampering and loss by comparing logs is proposed. The scheme enables us to detect incidents of changed or deleted logs. In addition, the scheme can identify the area where the change occurred. The results of experiments confirm that this scheme enables the detection of incidents of log tampering carried out by real malware.

The remainder of this paper is organized as follows: Section 2 describes the problems of logging with syslog and gives an overview of the previously proposed log collecting scheme, the problems it addresses, and the problems it did not resolve. Section 3 describes our proposed log-storing scheme. Section 4 describes our proposed method for detecting log tampering. Section 5 discusses the evaluations carried out on our proposed schemes. Section 6 discusses related work, and section 7 concludes this paper.

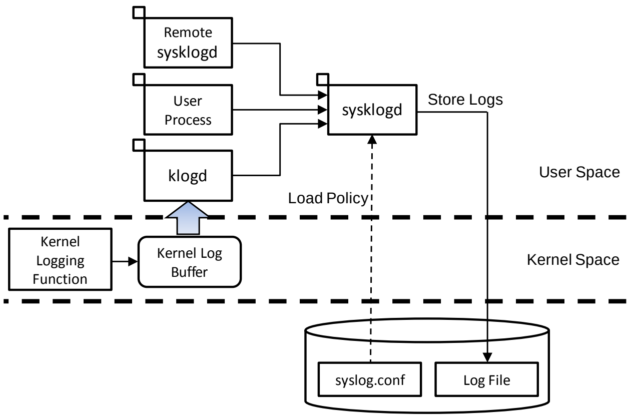

Figure 1 Architecture of syslog.

2 The Logging Module and Its Problems

In this section, we describe the architecture of syslog. We also look at the previously proposed logging scheme based on the VMM, along with its problems.

2.1 Syslog's Problems and Requirements for Addressing Problems

Syslog is a protocol for system management and security monitoring. Syslog consists of a library and a daemon. Figure 1 shows the architecture of syslog. User and kernel logs are collected as follows.

The syslog library provides functions for user program to send log messages to the syslog daemon. The syslog function sends messages to /dev/log with the send or write system call, and the syslog daemon collects logs from /dev/log with the read system call.

The kernel accumulates logs in internal buffer (kernel log buffer). The kernel logging daemon (klogd) gathers logs from the kernel log buffer, and afterwards, klogd similarly sends logs to the syslog daemon.

Syslog also has a filtering function. Its policies are described in the configuration file (syslog.conf).

Syslog has the following problems:

(1) The behavior of the syslog daemon can be modified by tampering with the configuration file. In addition, if the syslog daemon itself is tampered with, its output can be unreliable.
(2) Users who have permission to access logs can tamper with them intentionally.
(3) Kernel logs in Linux are accumulated in the ring buffer and are collected at fixed intervals. Thus, if the logs are not collected for a long time, old logs can be overwritten by new ones. Old logs will also be

overwritten if many logs are accumulated in a time that is shorter than the collecting interval.

New syslog daemons have been developed with the aim of achieving greater security [12, 13]. A number of current research projects are also geared towards the protection of the logs. These include protection of log files [4-6], protection of syslog programs [7], and other original logging method that independent of syslog [8]. However, no method has yet addressed all of the above problems.

To address the problems outlined above, we proposed and implemented a logging mechanism with virtualization technology that fulfills the following requirements:

(1) Detection of all outputs of log (user and kernel log).
(2) Isolation of log.
(3) Security of logging mechanism.

Our implemented system is OS independent, adaptable to various environments, and easy to adapt to newer OS kernel versions. Although it is necessary for protection of the log, OS (and version) independence had until this point proven to be an insurmountable obstacle in the implementation of this kind of system.

2.2 The Logging Module

2.2.1 Overview of the Logging Module

Figure 2 depicts the architecture of the logging module. The monitored OS runs in the VM, while the logging module operates in the VMM (the details are described below). Here, we use Xen as the VMM [14]. The logging module collects logs generated by a user process works in the monitored OS. After that, the collected logs are copied by the xend daemon, which operates in Domain0. Domain0 has privileged controls of the VMM. The xend daemon controls the VMM. It copies the accumulated logs from the VMM to Domain0 and stores them in files. Our previous paper [11] details the implementation of this mechanism.

2.2.2 The User Log Collector

The collector acquires logs when the requirement for sending user logs occurs. As shown in Figure 2, the logging module in the VMM hooks the system call that was invoked for sending logs from the user process to the syslog daemon.

To hook system calls in the VM, it is necessary that the mechanism enable the VMM to detect invocation of system calls in the VM. Therefore, in the logging module, we applied a mechanism that causes a page fault when a system call is invoked [15]. In a fully virtualized environment, if a page fault occurs on the VM, then the VMM is raised (VM exit) [16]. After the VMM has been raised, the logging module acquires the user logs and hides the occurrence of the page fault. Finally, the VMM raises the guest OS, which works as if no event has occurred.

In this method, to cause a page fault, we modified some registers of the monitored OS. A system call using the sysenter (fast system call) refers the value in sysenter_eip_msr and jumps to its address to execute the

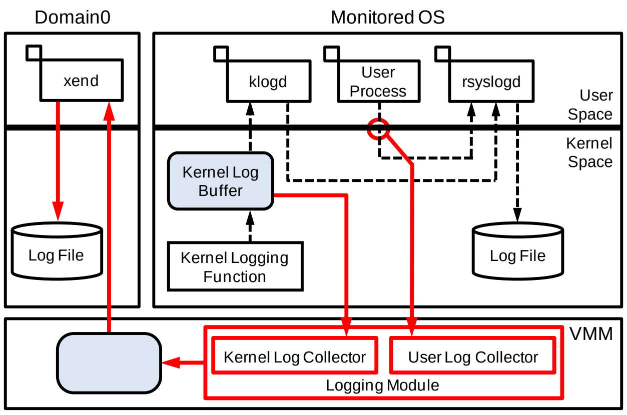

Figure 2 Architecture of the logging module.

system call function (sysenter_eip_msr is one of the machine-specific register (MSR)). Through modification of this value to another address to which access is not permitted from the monitored OS, a page fault is made to occur when a system call is invoked.

2.2.3 The Kernel Log Collector

The collector acquires logs when the kernel logging function is called in a guest OS. Normally, the VMM cannot detect a function call in a guest OS. To solve this problem, the system sets a breakpoint in the guest OS, a breakpoint exception occurs when some process reaches this breakpoint. In our previously proposed system, since the guest OSs are fully virtualized, breakpoint exception is handled by the VMM. Using the exception as an opportunity to acquire logs, the VMM can collect kernel logs.

When the processing is brought to the VMM, the logging module checks the state of the kernel log buffer of the monitored OS. If new logs have accumulated in the buffer, the logging module collects them. After that, the VMM returns the processing to the guest OS. Since these processes have no effect on the state of the guest OS, the guest OS can continue to write to the kernel log.

In this method, since kernel logs are collected when a kernel logging function is called, newer ones never overwrite old logs.

2.3 Logging Module Problems

The logs collected by the logging module are stored in the memory region managed by the VMM, logs are never copied without a request from Domain0. In this situation, logs in the memory will be lost when the machine is powered off or rebooted.

In addition, if the machine is powered off without the logs being saved to a file, the system loses the resources needed for log comparison and the detection of tampering.

3 The Log-Storing Module

3.1 Requirements

To address the problems outlined at Section 2.3, this paper proposes a log-storing module that automatically stores

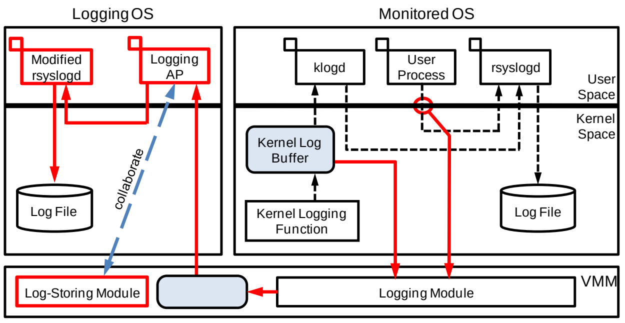

Figure 3 Architecture of the log-storing Module.

logs to disks. The module automatically transfers logs from the VMM to the logging OS, and the logging AP operating in the logging OS saves the logs to disks. This module enhances the logging mechanism by ensuring that log files are preserved.

There are two requirements that need to be met in order for the log-storing module to address the problems outlined in Section 2.3. These requirements as follows:

(1) Assured reception and storing of the logs from the VMM to files in the logging OS.
(2) Keeping the overheads that arise in the log-storing module at a minimum.

Requirement (1) is necessary for ensuring that the logs are indeed preserved, while requirement (2) is necessary because the overheads that arise in the log-storing module affect the performance of the monitored OS.

3.2 Overview of the Log-Storing Module

Figure 3 depicts the architecture of the log-storing module. In the Figure 3, arrows indicate the path of the logs as they are collected by the proposed system. The arrows with the closely spaced broken lines (black) indicate the conventional logging path used by the syslog daemon. The arrow with the more widely spaced broken line (light blue) indicates the collaboration between the logging AP and the log-storing module.

Figure 4 depicts the flow of the log storing. The flow divided into two parts: the accumulating part and the storing part. In the accumulating part, when the logging module detects system call invocation in the monitored OS, the logging module copies logs from the monitored OS to the VMM. The log-storing module then notifies the logging AP that logs has been collected. The Detail of the notification is described in Section 3.3. After the notification, the logging module returns processing to the monitored OS. In the storing part, the logging AP requests a copy of the logs. To respond to the request, the VMM copies logs to the logging AP and the AP transfers the logs to the modified syslog daemon, which stores them to files.

Here, the syslog daemon in the logging OS uses the same logging policy as that used by the syslog daemon in the monitored OS. In storing the logs to files, the logs are compressed and the messages sorted based on this syslog daemon policy. If the logging AP had directly stored the logs to files, it would have been difficult to compare them to the logs in the monitored OS. Using the same policy in

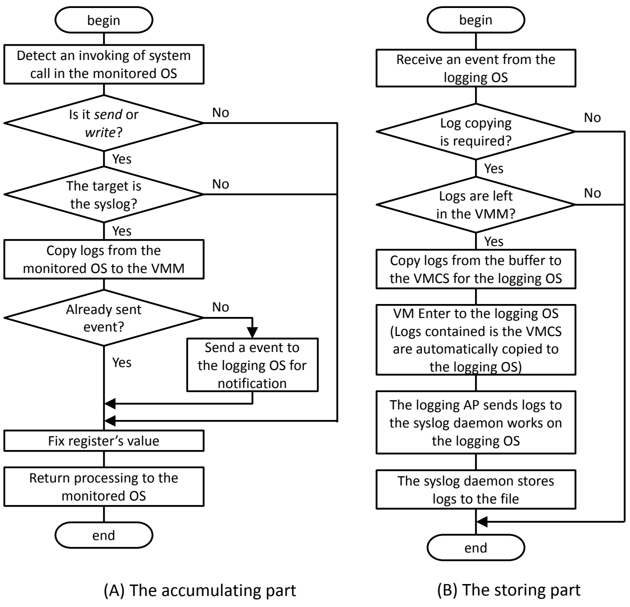
begin
Detect an invoking of system call in the monitored OS
Is it *send* or *write*?
No
Yes
The target is the syslog?
No
Yes
Copy logs from the monitored OS to the VMM
Already sent event?
No
Send a event to the logging OS for notification
Yes
Fix register's value
Return processing to the monitored OS
end
(A) The accumulating part
begin
Receive an event from the logging OS
Log copying is required?
No
Yes
Logs are left in the VMM?
No
Yes
Copy logs from the buffer to the VMCS for the logging OS
VM Enter to the logging OS (Logs contained is the VMCS are automatically copied to the logging OS)
The logging AP sends logs to the syslog daemon works on the logging OS
The syslog daemon stores logs to the file
end
(B) The storing part

Figure 4 The flow of log storing.

the logging OS as the monitored OS enables us to compare logs easily.

3.3 Communication between the VMM and the Logging AP

An event channel is used for communication between the VMM and the logging AP. Events are the standard mechanism for delivering notifications from the hypervisor to guests, or between guests. Events fall into three categories; inter-VM events, physical IRQ, and virtual IRQs. We use the inter-VM events in communication between the logging AP and the VMM.

Communications between the VMM and the logging OS consists of the VMM notifying the logging OS that logs have been collected by the logging module. In this case, there are two types of events that the logging AP can be made aware of. In the first type of event, a notification is given for each logs collected, while the other type of event is triggered when the size of the accumulated logs exceeds a specified limit. The latter has little overheads than the former because the number of copy is less than that of the former. However, the latter has a risk for losing large amounts of logs. If the machine is powered off without the logs being stored to the files in the logging OS, all of the logs in the VMM are lost. This situation largely affects the latter more than the former. Thus, from the viewpoint of log preservation assurance, the former is better than the latter.

Further, the latter requires that a large amount of memory be reserved in the VMM region. This over-reservation of memory in the VMM reduces the space available for VMs.

For these reasons, we selected the former notification technique for use with the logging AP.

3.4 Copying Logs to the Logging AP

3.4.1 Timing of Event Delivery

The event is not delivered instantaneously. An event is first queued to the target OS, after which the target OS is scheduled and queued events delivered. Thus, some amounts of delay in event delivery should be taken into consideration in log collection notification.

To compensate the delay of event delivering, the log-storing module should buffers logs in the memory. In contrast to buffering, our module sends an event immediately as each log is collected from the monitored OS because we need to store the logs as quickly as possible. Therefore, as soon as a notification reaches to the logging OS, the logging AP copies the logs from the VMM to its own memory region.

3.4.2 Reducing Copy Overheads

To fulfill the requirement (2), the overheads that arise during the copying of the logs must be kept to a minimum. Until the request hypercall reaches to the VMM, logs are buffered in the VMM. When the hypercall is invoked by the logging OS, all buffered logs are copied to the logging OS.

Currently, our proposed method buffers logs only when many events are queued to the logging OS. The logging module sends an event when the module detects log sending on the monitored OS. An event is asynchronously reaches to a guest OS. Thus, logs are copied from the VMM to the logging OS when the logging OS invokes hypercall to require log copying. In this case, if the logging module sends an event in every time of detection of log sending on the monitored OS, the logging OS may invoke hypercalls in every event. However, the log-storing module copies all logs accumulated in the VMM at one time. Thus, the logging OS invokes unnecessary hypercalls. The transition between a guest OS and the VMM takes about two microseconds. Thus, unnecessary hypercall invocation degrades performance.

To address this problem, the logging module reduces sending of unnecessary event. The logging module does not send an event if logs are accumulated in the VMM's memory region and the logging module already sent an event. With this mechanism, the VMM can copy logs to the logging OS with an event and a hypercall. Thus, there are only necessary event and hypercall exist. Figure 5 shows the reduction of unnecessary events. Incidentally, the reduction of unnecessary events does not degrade the log preservation assurance referred in Section 3.3.

To reduce the overheads, it is effective to minimize the number of copy. To minimize the copy overheads, log-buffering mechanism referred in Section 3.3 is effective. However, it is a challenging problem and not implemented in current our proposed system.

3.5 Log-Storing in the Logging OS

Figure 6 depicts the overview of the log storing procedure in the logging OS. To store logs to files, the logging AP and the modified syslog daemon are works on the logging OS. The logging AP receives logs of the monitored OS via the storing module works in the VMM. The modified

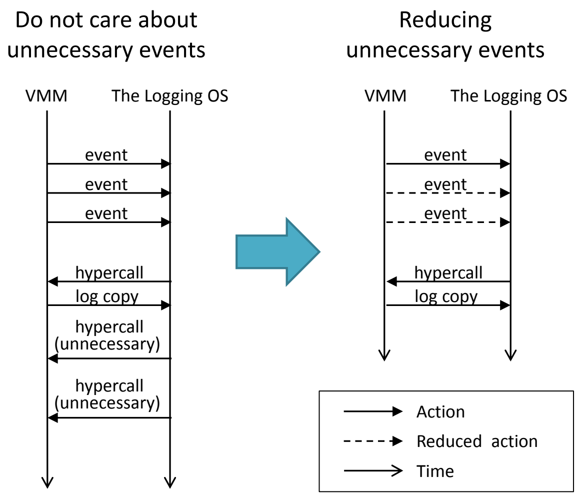

Figure 5 Reducing unnecessary events.

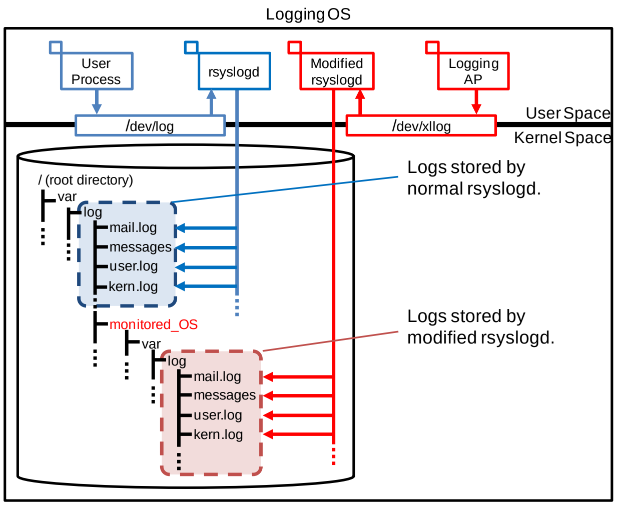

Figure 6 Log-storing with modified syslog daemon.

syslog daemon stores logs to files. The reason why we modified the syslog daemon in the logging OS is to avoid mixing of logs between the monitored OS and the logging OS. Normal syslog daemon receives logs via the /dev/log socket file. Meanwhile, the modified syslog daemon receives logs via the /dev/xllog socket file. The logging AP sends the collected logs to the /dev/xllog socket file and the modified syslog daemon receives logs via the socket file. In this method, the collected logs from the VMM are stored separately from the log files stored by the normal syslog daemon. In this situation, two syslog daemons (normal one and modified one) are running in the logging OS.

Furthermore, for ease of comparison, the modified daemon loads the policy of the syslog daemon operates in the monitored OS. At this time, if the policy loads by the modified syslog daemon is completely same as the policy loads by the normal one, the log files stored in the logging OS contains both logs of the monitored OS and those of the logging OS. If both logs are stored together, it is difficult to find out the logs collected by the logging AP.

For these reasons, we change the directory used for storing the logs collected by the logging AP. For example, we change the policy as stores the logs that are originally stored to the /var/log/messages to the /var/log/monitored_OS/var/log/messages. In this case, the modified syslog daemon in the logging OS assumes the root directory as the /var/log/monitored_OS. With this change, we can compare log files in the monitored OS with those in the logging OS easily.

4 Detection of Log Tampering

To detect incidents of log tampering, we compares the logs in the monitored OS with those of the logging OS. For fine-grained analysis, a file comparing method is useful. In this section, we describe the method to detect incidents of log tampering and loss by comparing log files.

Figure 7 depicts the flow of tampering detection. First, we replace hostname column of the log file in the logging OS as the hostname of the monitored OS because the file contains the hostname of the logging OS. Second, we mount the disk image of the monitored OS. Then, we extract log entries that we want to compare, and finally, compare log files between the monitored OS and the logging OS. In this process, if a difference is detected, that is the part tampered with.

4.1 Requirements for Detection of Log Tampering

To detect tampering by means of log file comparison, the following entries are required.

(1) Hostname.
(2) Timestamp.
(3) Username.
(4) Log message.

The reason is as follows. A hostname is needed in order to determine the source of log. A timestamp, a username, and a log message are necessary to ascertain the veracity of the environment containing the log. Attackers tamper with logs to hide the time of command execution and the identity of the who executed the command. A log message is needed in order to determine what changes were caused by attacks.

By comparing these entries between the monitored OS and the logging OS, we can detect what and how those entries were tampered with.

4.2 Comparing Logs

Acquire log files in the monitored OS and compare them with log files those are stored by logging AP. To get the logs in the monitored OS, we mounted the disk image that the monitored OS currently using. Thus, the comparison can be done even if the monitored OS is working on.

In comparing log files, diff is useful tool. However, it is difficult to compare logs in untouched format because the difference between logs in the monitored OS and the logging OS. If there are many differences in each log file, diff cannot give us efficient information. Thus we modified the syslog daemon in the logging OS to use the policy in the monitored OS. If the syslog daemon loads and uses the policy in the monitored OS, we can easily detect

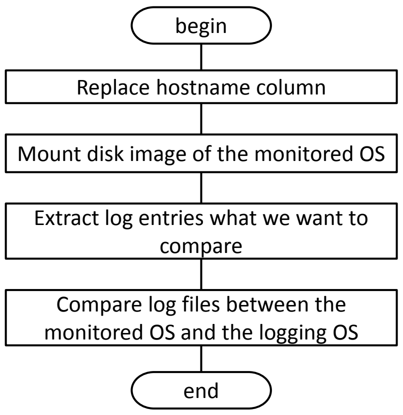
begin
Replace hostname column
Mount disk image of the monitored OS
Extract log entries what we want to compare
Compare log files between the monitored OS and the logging OS
end

Figure 7 The flow of tampering detection.

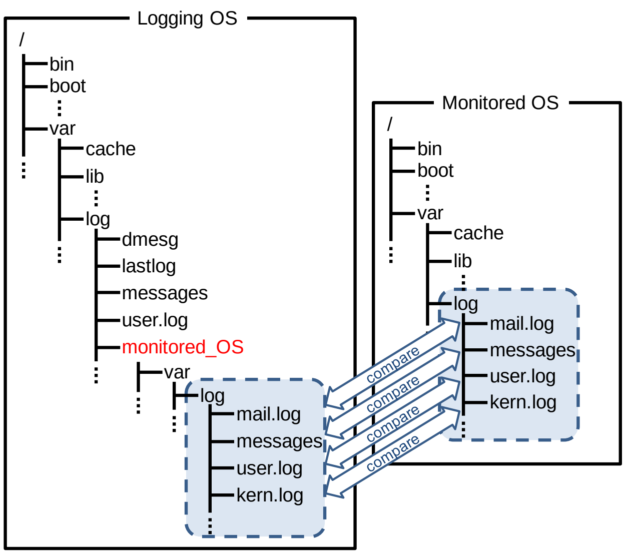

Figure 8 Directory tree in each OS.

tampering by comparing log files in each OS.

Figure 8 depicts the directory trees in each OS. As shown in Figure 8, to detect tampering, we just compare log files in each OS. The /var/log/monitored_OS/var/log/ directory in the logging OS corresponds to the /var/log/ directory in the monitored OS.

4.3 Log Formatting

The logs stored by the modified syslog daemon contain the hostname of the logging OS (not the hostname of the monitored OS), while other required entries are correctly stored. To address this problem, we format the logs in the logging OS collected by the logging AP to replace the hostname of the logging OS to that of the monitored OS.

4.4 Advantage

Incidents of log tampering are detected by comparing log files. If a log in the monitored OS is tampered with, the comparison enables us to detect this occurrence. The comparison also enables us to detect exactly where and how the log was tampered with.

Tripwire [17] can detect any changes in a designated set of files and directories. However, Tripwire’s detection mechanism uses signature comparison. As a result, it is unable to indicate what part of the file has changed. In

```
5,9d4
< Nov 27 22:56:22 debian  sudo:   user : TTY=pts/0 ; PWD=/home/user ; USER=root ; COMMAND=/bin/login thief
< Nov 27 22:56:24 debian login[2626]:   pam_unix(login:session): session opened for user thief by user( uid=0)
< Nov 27 22:56:51 debian  sudo:    thief : TTY=pts/0 ; PWD=/home/thief/vanish ; USER=root ; COMMAND=/bin/bash
< Nov 27 22:58:08 debian  sudo:    thief : TTY=pts/0 ; PWD=/home/thief/vanish ; USER=root ; COMMAND=./vanish
thief thief-host thief-addr
```

Figure 9 Result of diff command applied to auth.log in the monitored OS and the logging OS.

Table 1 Software used for evaluation.

OS (Debian 5.0.3)	Domain0	Linux 2.6.18-xen
	HVM domain	Linux 2.6.26
	Not virtualized	
VMM	Xen 3.4.1	
Syslog daemon	rsyslogd 3.18.6	

order to achieve this, a method such as content comparison is needed. Our proposed detection method incorporates content comparison to detect file changes.

Our proposed method, however, is not suitable for instantaneous tamper detection. Thus, we recommend that our method be combined with a method that instantaneously detects file changes to provide comprehensive, efficient means of detecting and analyzing properties changes.

5 Evaluation

5.1 Purpose

We evaluated the system from three points of view: completeness of the collected logs, ability to detect tampering, and the overheads that arise in our proposed system. Table 1 indicates software used for evaluation.

5.2 Completeness of Collected Logs

To confirm the ability of the log-storing module to store logs completely, we used ApacheBench benchmark tool. With the tool, we requested a file 500 times in a short time. Here, the concurrency of the connection by the tool is 1 and the length of each log entry is 104, total length of these lines is 52,000. The web server outputs logs in each access and the log-storing module attempts to store logs in each output of log.

After the experiment, we checked the log file stored in the logging OS. The file contains all of log entries and it indicates the total time consumed by the experiment is less than 1 second. With this result, it is considered that the proposed log-storing module have enough ability to store logs with no loss of logs even in high-load situation.

5.3 Detection of Log Tampering

To detect log tampering by our proposed logging module and the log-storing module, we tamper with logs by a log wiper program. After that, comparing each log to detect tampering and file changes.

We use Vanish as a log wiper written for educational purpose. Vanish remove a log entry that contains a designated username, hostname and IP address.

For detection of log tampering, we delete log entries in the auth.log log file in the monitored OS with Vanish. We add a new user "thief" in the monitored OS. Here, we assume that the user "thief" can use sudo command. After logged in as "thief", we use sudo. The execution of sudo is logged to auth.log. Finally, we tampered with logs by Vanish to delete log entries that contain "thief". To detect this file change, we compared the auth.log in the monitored OS and that in the logging OS.

Figure 9 shows the output of diff command applied to the auth.log in the monitored OS and the logging OS. Log entries shown in Figure 9 are not appeared in the auth.log in the monitored OS because those are deleted by Vanish.

From the result, our proposed logging scheme prevented tampering of log files by isolating logs from the monitored OS. Moreover, the scheme can detect the file changes in the log files in the monitored OS.

5.4 Security of the Logging Path

To guarantee the integrity of a log, it is necessary to ensure the security of the logging path. Here, we compare the security of the logging paths of the existing and the proposed system.

First, we analyze the logging path of a user log. A user log might be attacked at the following points:

(1) The time when a user process generates a log.
(2) The time between the sending of a log and its receipt by the syslog daemon.
(3) The time between the reception by the syslog daemon and storing it to a file.
(4) After the output of a log.

The existing system cannot detect and prevent tampering or the loss of logs at any time. In contrast, in the proposed system, time (1) is the only possible time when attacks might be suffered. To protect the logs in the time (1), it is necessary to ensure the integrity of all programs that generate logs. In this case, DigSig [18] is suitable but this method does not satisfy our demand because it modifies the kernel codes. Moreover, this method cases a large overhead.

Second, we analyze the logging path of a kernel log. A kernel log might be attacked at the following moments:

(1) The time to generate a kernel log in a kernel.
(2) The time to output the log to a kernel log buffer.
(3) While stored in the kernel log buffer.
(4) The time during which a kernel logging daemon gathers a log.
(5) While the kernel logging daemon sends the log to syslog.
(6) While syslog stores the log to a file.
(7) After the output of a log.

Table 3 Performance comparison in each environment.

	1 KB File			100 KB File		
	(A) Linux	(B) Xen	(C) The logging module	(A) Linux	(B) Xen	(C) The logging module
(1) Linux	1	1.88	1.97	1	1.24	1.27
(2) Xen	0.53	1	1.05	0.80	1	1.02
(3) The logging module	0.51	0.96	1	0.79	0.98	1
(4) The logging module and the log-storing module	0.37	0.69	0.72	0.70	0.86	0.88

Table 2 Environment used for measurement.

	Server Machine			Client Machine	
CPU	Core 2 Duo (2.40 GHz)			Pentium 4 (3.00 GHz)	
Memory	The Logging OS	1 GB	2 GB	1 GB	
	The Monitored OS	1 GB			
Bandwidth	100 Mbps			100 Mbps	
Software	thttpd 2.25b			ApacheBench 2.3	
				Requests	100
				Concurrency	1

In the existing system, it is impossible to protect a log from an attack by a rootkit at any time. Furthermore, there is a possibility of attack similar to the logging of a user log if the kernel is safe. The proposed system gathers a log at time (2). We can consider tampering with the kernel logging function as an example of an attack at time (2). However, the log gathered by the proposed system is the previous one. Therefore, the logs might be attacked at time (3). Thus, the proposed system can address attacks on and after time (4). The improvement of the proposed system for gathering a log immediately after its output enables it to address time (3) as well.

5.5 Overheads

We measured the overheads that arise in our proposed system. To evaluate the effects for APs, we used thttpd web server and ApacheBench benchmark tool for measurement of throughputs of the web server. Since thttpd uses syslog function for logging, it is suite for evaluation of our proposed system.

Table 2 shows the environment used for measurement. In the environment, thttpd operates on the monitored OS and ApacheBench is executed on the client machine. The concurrency of the connection is 1 and the number of total requests is 100. We requested 1 KB and 100 KB files and measured throughputs in both experiments.

Figure 10 shows the results of the measurement. Table 3 compares the performance in each environment. From the comparison, the log-storing module causes large overheads. The throughput of 1 KB transfer in the environment (4) is about 37% to that in the environment (1). The throughput of 100 KB file transfer of the environment (4) is about 70% to that in the environment (1). The relative performance in 100 KB file transfer is better than that in 1 KB file transfer. It is considered that the ratio of file transfer becomes greater in total workload in the case of 100 KB file transfer. From these measurements, it is found that reducing the overheads that arise in our proposed system is a challenge for the future.

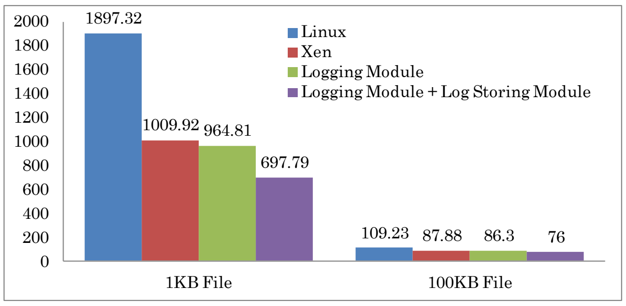

Figure 10 Throughputs of web server in each environment (file size: 1 KB).

5.6 Case Study

Assuming the situation that an attacker intruded into the virtual machine, which contains large amount of information related to Homeland Security. In ordinary circumstances, it is difficult to intrude into those virtual machines. At this time, we assume insider threats. If the insider intruded into the virtual machine, he firstly try to terminate or fake the logging daemon to hide his activities. In here, we assume the attacker alter the logging daemon that are modified to do not output logs related to his malicious activities. After successfully altered the daemon, he operates some malicious work and delete their activities related to the intrusion. Finally, he collects some important information and restores the logging daemon.

In this case, if the administrator of the virtual machine installed the tripwire, he can detect the modification of log files but he cannot prevent modification of files.

If the proposed system is installed in the virtual machine monitor lying under the virtual machine, he can prevent tampering of the log files because the VMM collects logs to the other virtual machine. Thus, the administrator can also detect tampering with log files.

6 Related Work

In this section, we discuss work related our research outlined in this paper. Section 6.1 describes a file integrity checking method, while Sections 6.2 to 6.4 describe log protection methods.

6.1 File Integrity Check

Tripwire [17] can detect changes in a designated set of files and directories. However, because the detection is carried out by comparing file signatures, it is unable to detect where in a file the change occurred, or how the file was changed. Kim and Spafford [17] state that the file comparison method is better than signature comparison and has fine detection capabilities. However, they also allege that the file comparison method is resource and time intensive. Our proposed tamper detection method detects and copies only those logs used by syslog and kernel logging function. Thus, there is no need to copy an entire file. Moreover, the method detects and copies only new logs-which require minimal overheads. For these reasons, we believe that our proposed method includes none of problems outlined in [17].

Our proposed method, however, is not suitable for instantaneous tamper detection. Thus, we recommend that our method be combined with a method that instantaneously detects file changes to provide a comprehensive, efficient means of detecting and analyzing property changes.

6.2 Protection of Log File

Some research has been carried out on the protection of files by the file system. The system NIGELOG has been proposed for protecting log files [4]. This method has a tolerance for file deletion. It produces multiple backups of a log file, keeps them in the file system, and periodically moves them to other directories. By comparing the original file and the backups, any tampering with the log file can be detected. Moreover, if any tampering is detected, the information that has been tampered with can be restored from these backups.

The protection of files with the file system is still vulnerable to attacks that analyze the file system. Therefore, a log-protection method using virtualization has been proposed [5]. This method protects logs by saving them to another VM, so it is impossible to tamper with the logs from other VM. However, this method aims to protect the log of a journaling file system, so the scope of the protection target is different from that in our research.

The hysteresis signature is used to achieve the integrity of files. However, it is known that the algorithm of the hysteresis signature has a critical weak point. Although the hysteresis signature can detect the tampering and deletion of files, it cannot prevent tampering and deletion. Moreover, the manager of the signature generation histories can tamper with the histories and files. Therefore, a mechanism to solve this problem using a security device has been proposed [6]. Because this method constructs a trust chain from the data in the tamper-tolerant area of the security device, the source of the trust chain is protected from attackers. Nevertheless, this method is not versatile because it uses the special device..

6.3 Protection of Syslog

The methods mentioned above are protecting log files. However, they cannot protect logs before storing of them. Thus, a method to guarantee syslog's integrity has been proposed [7], which uses a Trusted Platform Module (TPM) and a late launch by a Secure Virtual Machine (SVM) to ensure the validity of syslog. The validated syslog receives logs and sends them to a remote syslog.

Table 4 Security comparison between the proposed system and related works.

	Prevention of log-tampering	Prevention of log loss	Detection of log-tampering and loss
The proposed system	X	X	X
Tripwire [17]			X
NIGELOG [4]	X	(X)	X
Security device and hysteresis signature [5]	(X)		X
Protect the log of a journaling file system using virtualization [6]	(X)	(X)	X
Protection of syslog with TPM and SVM [7]	(X)	(X)	
LSM-based secure system monitoring [8]	X	X	

6.4 Other Logging Method

An original logging method, independent of syslog, has been proposed for audit [8]. This method uses Linux Security Modules (LSM) to collect the logs, and Mandatory Access Control (MAC) to ensure their validity. The system also uses SecVisor [19], and DigSig [18]. SecVisor ensures the security of the logging framework, and DigSig prevents rootkit from making modifications to access permissions. DigSig adds a signature to a program, and prevents the execution of an unknown program by verifying its signature. This method collects logs in its own way, but the method modifies the kernel source codes. In general, kernel modification is difficult and complex, so the method lacks versatility. In addition, the method uses variety of mechanisms, the overheads arising from them have large effect on daily operations on computers.

6.5 Comparison between the proposed system and related works

Table 4 shows the comparison between our proposed system and related works. The comparison noticed on prevention of log tampering, prevention of log loss, and detection of log tampering and loss.

The proposed system can prevent log tampering and loss and detect them. Protection of the logs of journaling file system [6] is secure than other methods. However, the protection method only protects the log of journaling file system. Other methods only prevent tampering or loss, and many other methods aims to detect tampering and loss.

7 Conclusion

In this paper, we proposed and described a log-storing module that stores logs collected by the logging module in a separate VM. We also described our log tampering detection technique, which is based on log comparison.

Evaluations of our proposed method's ability to detect tampering by real malware were also described with the results of the evaluations confirming that our log-tampering detection function has enough ability to detect this kind of tampering.

An evaluation of the impact of our proposed method on the performance of the monitored OS was also concluded. The evaluation shows that the proposed method decreases the performance of thttpd web server to 37% of that operates in not-virtualized environment in a worst case. From the evaluation, it is found that reducing the overheads that arise in our proposed system is a challenge for the future.

Acknowledgment

This research was partially supported by Grant-in-Aid for Scientific Research 21700034 and a grant from the Telecommunications Advancement Foundation (TAF).

References

[1] *Marisa Reddy Randazzo, Michelle Keeney, Eileen Kowalski, Dawn M. Cappelli and Andrew P. Moore, Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector, Carnegie Mellon University Technical Report CMU/SEI-2004-TR-021, 2005.*

[2] *Karen Kent and Murugiah Souppaya, Guide to Computer Security Log Management, NIST Special Publication 800-92, 2006.*

[3] *Jeffrey Hunker and Christian W. Probst, Insiders and Insider Threats - An Overview of Definitions and Mitigation Techniques, Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 2011, pp.4-27.*

[4] *Tetsuji Takada and Hideki Koike, NIGELOG: Protecting Logging Information by Hiding Multiple Backups in Directories, International Workshop on Database and Expert Systems Applications, 1999, pp.874–878.*

[5] *Siqin Zhao, Kang Chen and Weimin Zheng, Secure Logging for Auditable File System using Separate Virtual Machines, Proc. IEEE International Symposium on Parallel and Distributed Processing with Applications, 2009, pp.153 –160.*

[6] *Yuki Ashino and Ryoichi Sasaki, Proposal of Digital Forensic System using Security Device and Hysteresis Signature, Proc. Third Inter-national Conference on International Information Hiding and Multimedia Signal Processing (IIH-MSP 2007) – Vol. 02, 2007, pp.3–7.*

[7] *Benjamin Boeck, David Huemer, A Min Tjoa, Towards More Trustable Log Files for Digital Forensics by Means of "Trusted Computing", Proc. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), 2010, pp.1020–1027.*

[8] *Takamasa Isohara, Keisuke Takemori, Yutaka Miyake, Ning Qu and Adrian Perrig, LSM-Based Secure System Monitoring using Kernel Protection Schemes, Proc. International Conference on Availability, Reliability, and Security, 2010, pp.591–596.*

[9] *Vivek Kundra, Streaming at 1:00: In the Cloud, http://www.whitehouse.gov/blog/Streaming-at-100-In-the-Cloud/, 2009.*

[10] *Muhammad Abedin, Syeda Nessa, Latifur Khan, Ehab Al-Shaer and Mamoun Awad, Analysis of firewall policy rules using traffic mining techniques, International Journal of Internet Protocol Technology (IJIPT), Vol.5, No.1/2, 2010, pp.3–22.*

[11] *Masaya Sato and Toshihiro Yamauchi, VMBLS: Virtual Machine Based Logging Scheme for Prevention of Tampering and Loss, 2011 International Workshop on Security and Cognitive Informatics for Homeland Defense (SeCIHD'11), Lecture Notes in Computer Science, Vol.6908, 2011, pp.176–190.*

[12] *Adiscon's Rsyslog, The enhanced syslogd for Linux and Unix rsyslog, http://www.rsyslog.com/*

[13] *The free software company BalaBit, Syslog Server | syslog-ng Logging System, http://www.balabit.com/network-security/syslog-ng/*

[14] *Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield, Xen and the Art of Virtualization, Proc. 19th ACM Symposium on Operating Systems Principles, 2003, pp.164–177.*

[15] *Artem Dinaburg, Paul Royal, Monirul Sharif and Wenke Lee, Ether: Malware Analysis via Hardware Virtualization Extensions, Proc. 15th ACM conference on Computer and Communications Security, 2008, pp.51–62.*

[16] *Intel, Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2, http://www.intel.com/Assets/PDF/manual/253669.pdf (2009).*

[17] *Gene H. Kim and Eugene Howard Spafford, The Design and Implementation of Tripwire: A File System Integrity Checker, Proc. 2nd ACM Conference on Computer and Communications Security, 1994, pp.18–29.*

[18] *Axelle Apvrille, David Gordon, Serge Hallyn, Makan Pourzandi and Vincent Roy, DigSig: Run-time Authentication of Binaries at Kernel Level. Proc. 18th USENIX Conference on System Administration, 2004, pp.59–66.*

[19] *Arvind Seshadri, Mark Luk, Ning Qu and Adrian Perrig, SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes, Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles, 2007, pp.335–350.*