Secure Log Transfer by Replacing a Library in a Virtual Machine

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. Ensuring the integrity of logs is essential to reliably detect and counteract attacks, because adversaries tamper with logs to hide their activities on a computer. Even though some research studies proposed different ways to protect log files, adversaries can tamper with logs in kernel space with kernel-level malicious software (malware). In an environment where Virtual Machines (VM) are utilized, VM Introspection (VMI) is capable of collecting logs from VMs. However, VMI is not optimized for log protection and unnecessary overhead is incurred, because VMI does not specialize in log collection. To transfer logs out of a VM securely, we propose a secure log transfer method of replacing a library. In our proposed method, a process on a VM requests a log transfer by using the modified library, which contains a trigger for a log transfer. When a VM Monitor (VMM) detects the trigger, it collects logs from the VM and sends them to another VM. The proposed method provides VM-level log isolation and security for the mechanism itself. This paper describes design, implementation, and evaluation of the proposed method.

Keywords: Log transfer, log protection, virtual machine, digital forensics.

1 Introduction

Logging information about activities and events in a computer is essential for troubleshooting and for computer security. Logs are important not only for detecting attacks, but also for understanding the state of the computer when it was attacked. The importance of logs for computer security is described in Special Publication [1]. Adversaries tamper with logs to hide their malicious activities and the installation of malwares on the target computer [2–4]. If logs related to those activities are tampered with, detection of problems might be delayed, and the delay could cause further damage to services. In addition, log tampering impedes the detection, prevention, and avoidance of attacks. With the growth of cloud computing in recent years, security in VMs has become more important [5, 6]. Especially, log forensics in cloud application has great importance [7]. However, existing logging methods are not designed for VMs or cloud applications.

As described in a paper [8], secure logging using VMs provides integrity and completeness for logging. Boeck et al. proposed a method to securely transfer logs utilizing a trusted boot and a late launch [9]. While this method can prevent attacks to logging daemons, adversaries can still tamper with logs in kernel space. Logs must go through an Operating System (OS) kernel when transferred out of the computer. If malware is installed on, logs could be tampered with in kernel space. SecVisor [10] is a method that prevents the execution of illegal codes in kernel space. However, these methods depend on the structure of the OS kernel, making it difficult to adapt to various OSes. In a situation where a single machine provides many VMs, different OSes could be running on each VM. VMI [11] can be considered as a logging method for VM. However, VMI has problems including performance degradation and granularity of information.

These researches are considered as a method of log protection. However, even though the importance of logging for cloud application is increased [7], there is no method specialized for logging in VM environment. VM is commonly used for providing cloud computing environment. Providing services like logging hurts performance of APs on VMs [8]. Thus, reducing performance overhead incurred by additional services is an important challenge.

To collect logs from outside the VM securely, we propose a secure log transfer method using library replacement. To trigger a log transfer to a VMM, we embed an instruction in a library function to cause a VM exit. On Linux and FreeBSD, we modified the standard C library, `libc`, which contains standard logging function. When the VMM detects a VM exit, the VMM collects the logs generated by APs in the source VM and transfers them to the logging VM, which stores the logs to a file. We assumed that the modified library is secured in the memory by the method [12] that protects a specific memory area from being modified by kernel-level malware.

With the proposed method, adversaries cannot tamper with logs in kernel space because the VMM collects logs before they reach in kernel space. Because the modification to a library is kept minimal, adapting different OSes requires less effort. Performance degradation is minimal because the overhead incurs only when an AP calls a logging function. The proposed method replaces only a library, which includes a function to send logs to a syslog daemon. Therefore, we can make the possibility of bug inclusion low. Additionally, bugs in a library give less effect than that in a kernel.

This paper also describes evaluations of the proposed system. We evaluate the system with the standpoint of security of logs, adaptability to various OSes, and performance overhead. To evaluate the system with the standpoint of security of logs, we analyze the security of a logging path. Experiments to tamper with logs in the logging path are also described. Adaptability of the proposed system is provided with case studies to adapt to various OSes. Performance evaluations with APs commonly used in servers are described. As described in a paper [13], VMI causes large overhead. For practical use, performance degradation should be kept as small. With these evaluations, this paper presents how the proposed system is practical for generally-used APs and multi-VM environment.

The contributions made in this paper are as follows:

- We propose a secure log transfer method by replacing a library in a VM. With the proposed system, a kernel-level malware cannot delete or tamper with logs. Moreover, by comparing collected logs and tampered logs, we can identify the area that is tampered with.
- We design a tamper-resistant system using VMM. We implemented all of our system inside the VMM because of its attack-resistance.
- The proposed system is implemented with minimal modification to `libc`. Although no modification is preferable, modifying the library gives two advantages: slight overhead and ease of adaptation to varied OSes. This also reduces the possibility of bug inclusion, and makes the system more secure.

2 Method of Log Transfer

2.1 Existing Log Transfer Methods

In Linux and FreeBSD, syslog is a protocol for system management and security monitoring. Syslog consists of a syslog library and a syslog daemon. New syslog daemons and protocols [14–17] have been developed to achieve greater security. New syslog daemons can transfer logs to out of a computer and can encrypt syslog traffic using transport layer security (TLS). However, during log transfer, adversaries can delete or tamper with the log with a kernel-level attack [3]. Other methods using inter-process communications can be attacked in the same manner. Other malware tamper with logs by replacing syslog daemons [2].

VMI [11] inspects VMs by retrieving hardware information about the target VM and constructing a semantic view from outside the VM. ReVirt [18] collects instructions-level information for VM logging and replay. CloudSec [19] performs a fine-grained inspection of the physical memory used by VMs and detects attacks that modify kernel-level objects. While these methods enable us to collect information inside VMs, they increase complexity of semantic view reconstruction and performance overhead. In addition, the reconstruction of a semantic view strongly depends on the structure of the OS.

To overcome this problem, in-VM monitoring method [13], which inserts an agent into a VM, is proposed. It protects the agent from attacks from inside the VM. Inserting an agent is a practical and efficient way to collect information, however, it is difficult to adapt to various OSes because the implementation of an agent depends on the structure of the OS. The VMM-based scheme [20] can collect logs inside VMs without modifying a kernel or inserting agents. However, it has a large overhead and strong dependency to architecture of OS.

2.2 Problems of Existing Methods

Existing methods have the following four problems:

(1) Transferring log via inter-process communications can be preempted by kernel-level attacks.

(2) Collecting logging information inside a VM by monitoring the behavior of APs or OSes cause unnecessary performance overhead.
(3) Collecting logging information from various OSes requires efforts to adapt the method to a variety of OSes.
(4) Additional code increases the likelihood of bugs in the system.

No suitable method is currently available to transfer logs out of the VM. For security management, a secure logging method is required. Monitoring from outside the VM is a new approach, because the monitor itself is secured by VM-level separation. On the other hand, the information obtained by the method is difficult to translate into a semantic view or is too fine-grained. While VMI and other introspection methods securely collect information inside a VM, constructing the semantic view of the VM is strongly depends on structure of the target OSes. Adapting those methods to various OSes is nontrivial work. Inserting an agent into a VM can cause undesirable effects and make the VM unstable.

3 Secure Log Transfer by Replacing a Library in a VM

3.1 Scope and Assumptions

This paper covers the prevention of log tampering via attacks to the kernel, to the logging daemon, and to files that contain logs. Attacking specific APs requires nontrivial work and it cannot tamper with logs completely; therefore, adversaries attack the point where all logs go through. If we focus our attention on attacks to APs, preventing log tampering in kernel space and in a logging daemon is a reasonable challenge.

We assume attacks for a VMM is difficult because the conditions that allow attacks are limited. Therefore, we assume that a VMM can prevent those attacks.

3.2 Objectives and Requirements

The objectives of this paper are as follows:

Objective 1 To propose a fast and tamper-resistant log transfer method.
Objective 2 To propose a log transfer method that is easy to adapt to various OSes.

The objective of our research is to address problems detailed in Section 2.2. To address those problems, providing a tamper-resistant log transfer method is necessary. Specifically, we aim to prevent log tampering from kernel-level malware like adore-ng [3]. Moreover, low overhead is desired to implement the method to APs in the real world. Further, an OS-independent method is preferable, because it is assumed that various OSes are running on each VM.

To achieve the objectives, the followings are required.

Requirement 1 Transfer logs as soon as possible.
Requirement 2 Isolate logs from a VM.

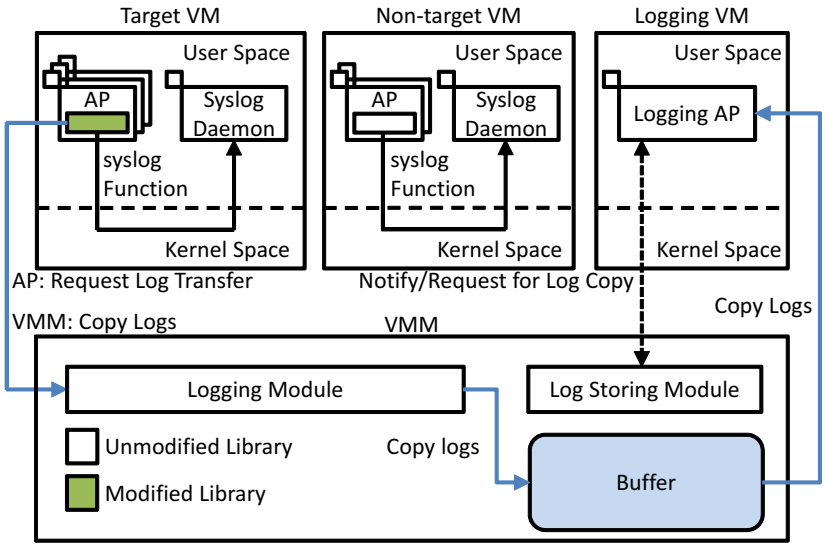

Fig. 1. Overview of the proposed system.

Requirement 3 Secure the log transfer mechanism itself.
Requirement 4 Make the log transfer method OS-independent and small.
Requirement 5 Reduce unnecessary overhead related to log transfer.

In a logging path, logs generated by a process are passed to a kernel because the kernel provides the ability to send messages to other processes. Therefore, to prevent log tampering in kernel space, it is necessary to collect logs from outside the VM before the logs reach kernel space. To prevent tampering of log files, they must be isolated from the VM. To ensure the security of the log transfer method itself, install the method outside the VM. With low dependency on the OS, migration to other OSes becomes easy. Moreover, a smaller program size helps to reduce the possibility of bugs. A VM exit, which is a CPU-mode transfer between a VM and a VMM, can cause additional overhead. To adapt the method to APs in the real world, unnecessary VM exits must be removed.

3.3 Overview of the Proposed Method

The overall design of the proposed system is shown in Figure 1. In the proposed system, the target VM works on a VMM and the VMM collects logs from the VM. We assume that all of the VMs fully virtualized by Intel VT-x. An AP on the target VM can transfer logs with the proposed system as follows:

(1) An AP requests a log transfer to a VMM.
(2) The logging module inside the VMM receives the request and copies logs from the AP to the buffer inside the VMM.

(3) The VMM sends a notification to a logging AP inside the logging VM. Then, the VMM sends the logs to the logging AP.
(4) The logging AP receives the logs and stores them to a file. The logging VM accepts logs only from the VMM.

We modified the VMM to transfer logs from the target VM to the logging VM. The logging module, the log storing module and the buffer VMM are additional part to the original VMM. We modified `libc` in the target VM to send a log transfer request to the VMM in each call of syslog function. The modified library executes an instruction that causes a VM exit, which triggers a log transfer to the logging VM before sending logs to the logging daemon in the current VM. Only a VM that contains the modified library can send the request. In Figure 1, the target VM requests a log transfer in every syslog function call; on the other hand, the non-target VM never makes the request.

Collecting logging information immediately after the invocation of the syslog function fulfills the requirement 1. With this feature, tampering logs in kernel space is impossible. Using the logging VM to store logs fulfills the requirement 2. Resources allocated to a VM, such as memory, network, disk space, and others are separated from resources allocated to another VM, therefore, it is difficult to tamper with logs outside the VM being attacked. It is also difficult to attack a VMM from inside a VM; therefore, using a VMM and modifying a library fulfill the requirement 3. Library modification also makes OS-adaptation easier and fulfills the requirement 4. Finally, VM exits occur only when a syslog library function is called; therefore, the requirement 5 is fulfilled.

3.4 Comparison between the Proposed Method and VMI

The proposed method and VMI are similar from the standpoint of collecting information inside VM. However, there are following differences between them:

- Security of logs.
- Dependency to a data structure in a VM.
- Overhead.

The proposed system can achieve greater security of logs than VMI. VMI collects information of VMs by monitoring hardware states and some events. However, it is difficult to detect log generation by monitoring hardware states or events. Even if VMI can detect log generation, when VMI detects it after a mode transition to kernel space, logs are tampered by kernel-level malware. By contrast, kernel-level malware cannot tamper with logs because the trigger of log transfer is given by a library in the user space of each VM.

To inspect a state of a VM, VMI collects some information strongly related to a data structure in a VM. Thus, VMI must have enough knowledge of layout of data structure in the VM. Additionally, to inspect a state of a VM, VMI must collect a lot of information (e.g. process list, process descriptor). This creates strong dependency to version of OSes in VMs.

As just described above, VMI can inspect a state of a VM with fine-grained information; however, it creates strong dependency of data structure in a VM and some overheads. On the other hand, however the proposed system cannot collect much information of a VM; it achieves weak dependency of data structure in a VM and low overheads. VMI has large overhead because it monitors a state of a VM with various and fine-grained information. A research [13] shows that VMI causes 690% overhead in monitoring of process creation. On the other hand, in-VM monitoring causes only 13.7% overhead in that monitoring. Thus, the approach of the proposed system is efficient because the system can be considered as one of an in-VM monitoring. Additionally, our proposed system only monitors invocation of syslog function. Therefore, overhead related to the proposed system arises only when an AP invokes syslog function.

4 Implementation

4.1 Flow of Log Transfer

Transferring logs from a VM to a VMM takes place in two phases: requesting the log transfer and copying the log. This section describes the implementation of each phase in Section 4.2 and Section 4.3. The modified code to `libc` library is shown as Figure 4 and explained at Section 5.5. The overall flow is as follows:

(1) An AP in the target VM requests a log transfer.
(2) A VM exit occurs and the VMM receives the request.
(3) The VMM copies logs from the AP to the VMM buffer.
(4) The VMM sends a log storing request to the logging VM.
(5) The logging VM receives the request and notifies the VMM that it is ready to receive the logs.
(6) The VMM copies the logs to the logging VM.
(7) The logging VM stores the logs to a file.

4.2 Request of Log Transfer

We embed a `cpuid` instruction in a library to request a log transfer to the VMM from an AP. The instruction does not affect the CPU state; however, if executed in a virtualized environment, the instruction causes a VM exit. Therefore, we embedded the instruction into a library to request the copying of logs to the external VMM before sending the logs to a logging daemon. The interface of log transfer request is shown in Table 1. The embedded codes set the appropriate values to the registers and execute the `cpuid` instruction. Additional codes are shown in Section 5.5. We utilize `cpuid` instruction to counteract detection of our approach that scans memory or a library file. One of a typical instruction to call a VMM is `vmcall`. If we use `vmcall` instruction as a trigger of log transfer, adversaries easily detect our approach by scanning a memory because the instruction is not used in regular APs. To make detection of our approach harder, we utilize `cpuid` instruction.

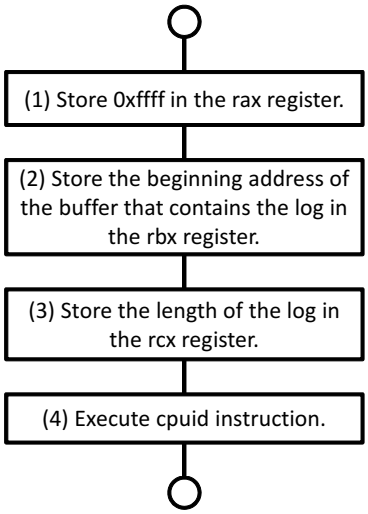

Fig. 2. Flow of a log transfer request.

Table 1. Interface of log transfer.

Register	Explanations
rax	0xffff: the value represents a log transfer request.
rbx	Address of the buffer that contains logs to transfer.
rcx	Length of logs to transfer.

Figure 2 depicts the flow of a log transfer request. At first, the AP on the target VM stores 0xffff in the rax register, the beginning address of the buffer in the rbx register, and the length of the buffer in the rcx register. Then, the AP executes `cpuid` instruction to request a log transfer.

4.3 Log Copying from a VM to a VMM

Figure 3 depicts the flow of log copying by a VMM. A `cpuid` instruction is a trigger for log transfer. After detected the instruction, the VMM copies logs from the AP and notifies to the logging VM if the value contained in the guest's rax register is 0xffff. If not, the VMM do not copy logs and only emulates the instruction. The buffer inside the VMM is implemented as a ring buffer to reduce the loss of logs in a high-load situation. Step (4) only sends notification. Log copying to the logging VM is made asynchronously. Thus, the time of log copying is kept as short as possible.

5 Evaluation

5.1 Purpose and Environment

We evaluated the proposed system at following standpoints:

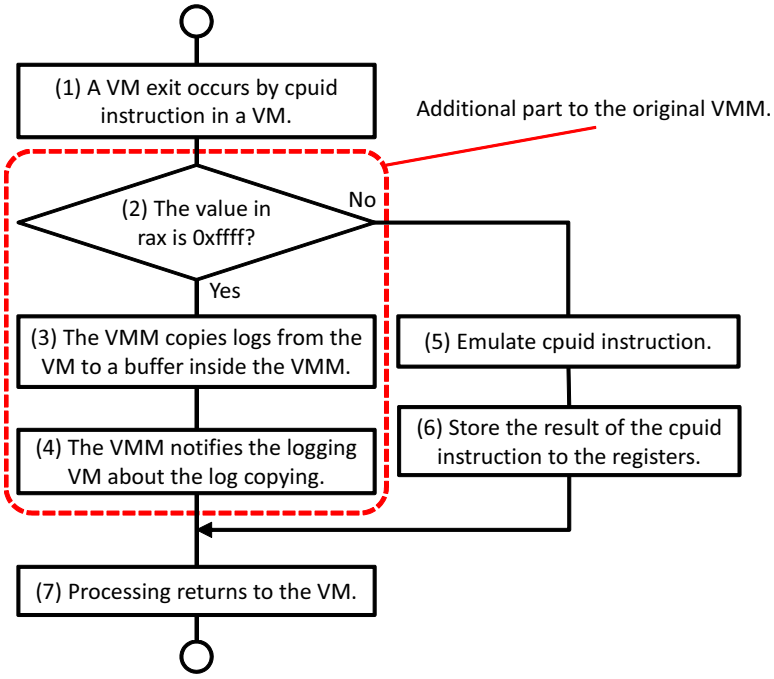

Fig. 3. Flow of log copying from the AP to the VMM.

- Security of logs in a logging path
  To evaluate the ability of prevention of log tampering, we inserted a malware into the kernel running on a VM.
- Prevention of log tampering and loss
  To check whether the proposed method can prevent log tampering and loss, we tried to prohibit log storing procedure with malware and some attacks.
- Completeness of log collection
  By sending a massive number of log transfer requests from an AP in the target VM, we tested the system in a high-load environment.
- Efforts for adapting various OSes
  Ease of adaptation to various OSes was also evaluated.
- Performance evaluation
  Performance overhead in Database Management System (DBMS) is also evaluated.
- Performance in multi-VM environment
  We measured performance of a web server with many VMs to clarify performance overhead incurred by the proposed system in multi-VM environment.

Software used for evaluation is described in Table 2. We implemented a prototype of the proposed system with Xen [21] hypervisor.

Table 2. Software used for evaluation.

VMM	Xen 4.2.0
OS (The logging VM)	Debian (Linux 3.5.0 64-bit)
OS (The target VM)	FreeBSD 9.0.0 64-bit, Debian (Linux 2.6.32 64-bit)
Web server	thttpd 2.25b
Database management system	PostgreSQL 9.2.4
Syslog daemon	rsyslogd 4.6.4
Benchmark	ApacheBench 2.3
	pgbench 9.2.4 (included with PostgreSQL 9.2.4)
	LMbench version 3

5.2 Security of Logs in a Logging Path

Logs can be tampered with at the following point: (1) The time when a process generates a log, (2) The time between the sending of a log and its receipt by a syslog daemon, (3) The time between the receipt of a log and storing it to a file, and (4) The time after the output of a log.

Kernel-level malware like *adore-ng* [3] can tamper with logs in time (2) and (3). Attacks for syslog daemon like *tuxkit* [2] can tamper with logs in time (3). Adversaries who have privileges to write to the log file can tamper with logs in time (4). Our proposed method can prevent attacks in time (2), (3), and (4) because logs are transferred to outside of the VM before it reaches in a kernel.

Without hypervisor-based software runtime memory protection mechanism [12], we cannot prevent log tampering in time (1). A kernel-level malware can manipulate memory of user processes; therefore, it tampers with logs before they are transferred to out of a VM. With the memory protection mechanism [12], a kernel-level malware cannot tamper with logs of user processes. Thus, to prevent log tampering in time (1), the memory protection mechanism [12] is necessary.

5.3 Prevention of Log Tampering and Loss

To check whether the proposed method can prevent log tampering or not, we tried to tamper with logs. First, we used *adore-ng* [3], which is a kernel-level malware that tamper with logs sent to the syslog daemon, to check if the proposed system can prevent log tampering in kernel space. The *adore-ng* patches runtime memory of kernel code to tamper with logs. The *adore-ng* monitors inter-process communication using socket function and deletes a message if it contains disadvantageous words for the adversary. This experiment proves that the proposed method can prevent log tampering by the kernel-level malware. Logs sent to the VMM with the proposed method were not tampered with while logs stored in the target VM is tampered with. Moreover, we can find log tampering by comparing logs between the target VM and the logging VM. With this comparison, we can estimate a purpose of the adversary.

Third, we tampered with a policy file of syslog daemon as no logs are written to files. The policy file is loaded by the syslog daemon at a start-up. By this

attack, no logs are written even if a syslog daemon is running. In this situation, we confirmed that the proposed system collects logs with no modification or loss. This result shows that the proposed system is resistant to attack for policy file of syslog daemon. The result also shows that log tampering by replacing a syslog daemon has no effect in a log collected by the proposed system. Thus, the proposed system is resistant to attacks like *tuxkit* [2].

Fourth, we stopped a syslog daemon on a target VM to prevent logging. Obviously, no logs are transferred to the syslog daemon. We also confirmed that the proposed system can collect logs completely. However, its completeness depends on a flow of the log transfer. In GNU `libc`, a syslog function aborts log transfer when the establishment of a connection is failed. Our prototype used for evaluation requests log transfer before establishing a connection to the syslog daemon; therefore we can collect logs completely. This implies that logs might be lost if the library requests log transfer after establishing connection.

Finally, we tampered with a log file. This type of attack is used in *LastDoor* backdoor [4]. It wipes specific entries in log files. Because the logs written to the file are already transferred to the logging VM, while logs in the target VM are tampered with, there is no effect to the log file in the logging VM.

These results show that the proposed system can collect almost all logs and collected logs are not affected by attacks on the target VM. Additionally, adversaries tend to install log tampering malware to a place where all logs go through. For example, *adore-ng* [3] is installed to a kernel function and *tuxkit* [2] is installed to a syslog daemon. All logs sent by syslog library function go through that kernel function and syslog daemon. From the reason, we can estimate that log tampering attacks to an AP, which is a source of logs, is rare.

5.4 Completeness of Log Collection

To ensure that the proposed system can collect all logs in the target VM with no loss, we tested the proposed system in a high-load environment. In an experiment, we sent a log transfer request 10,000 times within approximately 0.26 seconds. The length of the log in each request was approximately 30 bytes. All logs were successfully transferred to the logging VM. No logs were incomplete or lost. This result shows that our proposal is sufficient in terms of completeness of log collection in a high-load environment.

5.5 Efforts for Adapting Various OSes

In the prototype, we implemented the proposed method with FreeBSD and Linux as a target VM and Xen as a VMM. To adapt to various OSes, modification to the target VM must be minimal. We added 20 additional lines of codes to `libc` on FreeBSD and Linux. Figure 4 shows the result of `diff` command. As shown is Figure 4, we can adapt the proposed system to the `libc` library by inserting `cpuid_logxfer()` function before invocation of a send system call. The rest of the additional codes are definition of the `regs` structure and the `cpuid_logxfer`

```
void
  __vsyslog_chk(int pri, int flag, const char *fmt, va_list ap)
  {
            int saved_errno = errno;
            char failbuf[3 * sizeof (pid_t) + sizeof "out of memory []"];

+           reg_t regs;
+           regs.rax = 0xffff;
+
  #define   INTERNALLOG LOG_ERR|LOG_CONS|LOG_PERROR|LOG_PID
            /* Check for invalid bits. */
            if (pri & ~(LOG_PRIMASK|LOG_FACMASK)) {
***************
*** 278,283 ****
--- 297,308 ----
            if (LogType == SOCK_STREAM)
              ++bufsize;

+           regs.rbx = (unsigned long)buf;
+           regs.rcx = bufsize;
+
+           cpuid_logxfer(regs.rax, &regs);
+           regs.rax = regs.rbx = regs.rcx = 0;
+
            if (!connected || __send(LogFile, buf, bufsize, send_flags) < 0)
              {
                if (connected)
```

Fig. 4. The result of diff command between source codes of the unmodified library and the modified library.

function. These additional lines consist of (1) setting the registers with the appropriate values and (2) executing the `cpuid` instruction. Based on the size of the additional code, adapting the proposed system to various OSes would be a small effort.

5.6 Performance Evaluation

Measured Items and Environment We measured the performance of the syslog function, some system calls, and an AP. We also measured performance overhead in multi-VM environment. The performance measurements of both the syslog function and an AP show the additional overhead incurred by the proposed system. On the other hand, the performance measurement of some system calls shows that the proposed system causes additional overhead only when the syslog function is called.

We measured the performance with a computer, which has Core i7-2600 (3.40 GHz, 4-cores) and 16 GB memory. In each measurement, one virtual CPU (VCPU) is provided and 1 GB memory is allocated to each VM. Hyper-threading is disabled. Each VCPU is pinned to physical CPU core to avoid the instability of measurement. If many VMs work on one physical CPU, performance of APs on those VMs would be instable. Each VM has one VCPU and 1 GB memory.

Table 3. Performance comparison of the syslog function.

	Time (μs)	Overhead (μs (%))
Xen	31.47	–
Proposed system	33.38	1.91 (6.08%)

Table 4. Frequency of library function calls when providing a web page with thttpd web server.

Function name	Count	Rate (%)	Function name	Count	Rate (%)
strncasecmp	1600	17.77	strftime	200	2.22
strlen	1400	15.55	accept	200	2.22
strcpy	800	8.89	gmtime	200	2.22
vsnprintf	600	6.67	__errno_location	200	2.22
memmove	400	4.44	time	100	1.11
strchr	400	4.44	close	100	1.11
select	301	3.34	read	100	1.11
gettimeofday	301	3.34	getnameinfo	100	1.11
strstr	300	3.33	strcat	100	1.11
fcntl	300	3.33	readlink	100	1.11
strpbrk	300	3.33	strrchr	100	1.11
strcasecmp	200	2.22	syslog	100	1.11
__xstat	200	2.22	writev	100	1.11
strspn	200	2.22			

Syslog Function and System Call In the proposed system, the modified library requests log transfer when an AP called syslog function. To clarify the overhead incurred by the proposed system, we measured and compared the performance of syslog function with unmodified Xen and the proposed system. Table 3 compares the performance of the syslog function between Xen and the proposed system. In the proposed system, the additional overhead of the syslog function is 1.91 μs (6.08%), which is small enough, because the function is not called frequently.

Table 4 shows counts of function call in thttpd accessed by ApacheBench for 100 times. We measured the number of counts of library function call by `ltrace`. Table 4 shows the ratio of syslog function call in thttpd is about 1%. Additionally, we measured a performance impact of library functions in thttpd with the same workload. Table 5 shows the result of measurement. These results are measured in Ubuntu 13.04. The function named `__syslog_chk` is same as `syslog`. As shown in Table 5, performance impact of syslog function is only 0.18%, thereby it can be considered as 6.08% of overhead in syslog function has limited impact of the performance of APs.

Additionally, we measured the performance of some system calls by LMbench, which measures the performance of file creation and deletion, process creation, system call overhead, and other processes. In this measurement, the additional overhead is not significant.

Table 5. Performance impact of library functions in thttpd.

Function name	Rate (%)	Function name	Rate (%)
writev	76.90	memmove	0.13
poll	17.71	gmtime	0.10
strncasecmp	0.82	strcasecmp	0.10
strlen	0.81	strftime	0.10
strcpy	0.51	strspn	0.09
close	0.30	strcat	0.09
__vsnprintf_chk	0.30	read	0.08
__xstat	0.23	getnameinfo	0.05
strchr	0.22	memcpy	0.05
fcntl	0.22	time	0.05
__syslog_chk	0.18	strrchr	0.05
accept	0.17	__strcpy_chk	0.04
readlink	0.15	malloc	0.02
strpbrk	0.14	mmap	0.00
strstr	0.14	open	0.00
__errno_location	0.14	realloc	0.00
gettimeofday	0.14		

Table 6. Performance comparison of a PostgreSQL.

tmpfs	VMM	TPS	Relative performance
disabled	Xen	400.37	–
	Proposed system	395.76	0.99
enabled	Xen	1,448.80	–
	Proposed system	1,372.60	0.95

Performance of AP We measured performance overhead by the proposed system on a DBMS. To measure the performance overhead caused by the proposed system in DBMS, we used PostgreSQL as a DBMS. We configured PostgreSQL to call syslog function in each transaction. We used pgbench to measure performance of PostgreSQL. The workload with pgbench includes five commands per transaction. The benchmark measures transactions per second (TPS) of a DBMS. The concurrency of transactions is set to one.

Table 6 shows the comparison of a performance of the PostgreSQL DBMS. Higher TPS is better. Performance degradation with the proposed method is less than 1%. The proposed method degrades performance of a CPU intensive process. Because PostgreSQL accesses to disk heavily, the overhead incurred with the proposed method becomes small. To clarify that the proposed system is CPU intensive, we measure the performance with `tmpfs`, which provides a memory file system. Transactions do not require access to disk; therefore, performance overhead with the proposed method would be higher. Table 6 shows that the relative performance to unmodified Xen with tmpfs is about 5%. The performance degradation is higher than that in the case without tmpfs. If a processing is I/O intensive, performance degradation with the proposed method becomes

Table 7. Throughputs of a web server (request/s) in multi-VM environment.

File size	VMM	Number of VM						
		0	2	4	6	8	10	12
1 KB	Xen	1396.9	1329.27	1295.61	1225.22	1171.51	1231.72	1172.15
	Proposed system	1231.06	1150.54	1057.95	1017.53	987.24	1015.69	946.61
	Relative performance	0.88	0.87	0.81	0.83	0.84	0.82	0.8
10 KB	Xen	680.61	658.15	639.76	627.9	628.56	609.45	615.64
	Proposed system	664.48	626.12	612.93	559.02	582.24	578.89	589.58
	Relative performance	0.98	0.95	0.89	0.92	0.93	1.00	0.96
1,000 KB	Xen	11.41	11.41	11.4	11.39	11.38	11.39	11.39
	Proposed system	11.41	11.41	11.4	11.39	11.37	11.39	11.06
	Relative performance	1.00	1.00	1.00	1.00	1.00	1.00	0.98

less. Thus, the proposed method is suitable for I/O intensive APs. In this measurement, we configured PostgreSQL to call syslog in each transaction, however, logging frequency in general use of DBMS becomes less. Thus, the performance degradation can be assumed as almost negligible in normal use.

Performance in Multi-VM Environment To examine the ability of our proposal to scale to its target of many domains, we measured a performance of a web server in a VM with many other VMs. These VMs have a process that sends logs using syslog function every second. This evaluation is experimented with the machine that has four CPU cores; the logging VM is placed on the core 0, a VM that has a web server is placed on core 1, and other VMs are placed on core 2 and core 3 to measure the pure performance changes of the web server. We placed 2, 4, 6, 8, 10 and 12 VMs on core 2 and core 3. The number of VMs on core 2 and core 3 is same. Scheduling priority of each VM is configured as same. The performance is measured by ApacheBench on a remote machine with 1 Gbps network.

Table 7 shows performance in each environment. Figure 5 shows changes of performance in each environment. If the number of VM increases, the performance of the web server degrades. Performance degradation with the proposed system is less than about 10% when the file size is larger than 10 KB. Especially, when the file size is 1,000 KB, performance degradation is nearly 0. From the result, we can estimate that change of relative performance related to the number of VM is small enough. Despite the number of VM changes, change of relative performance is approximately same. For this reason, the proposed system is efficient in multi-VM environment.

6 Related Works

6.1 Secure Logging

Accorsi classified and analyzed secure logging protocols [22]. In that paper, extensions of syslog, including syslog-ng [15], syslog-sign [16], and reliable syslog

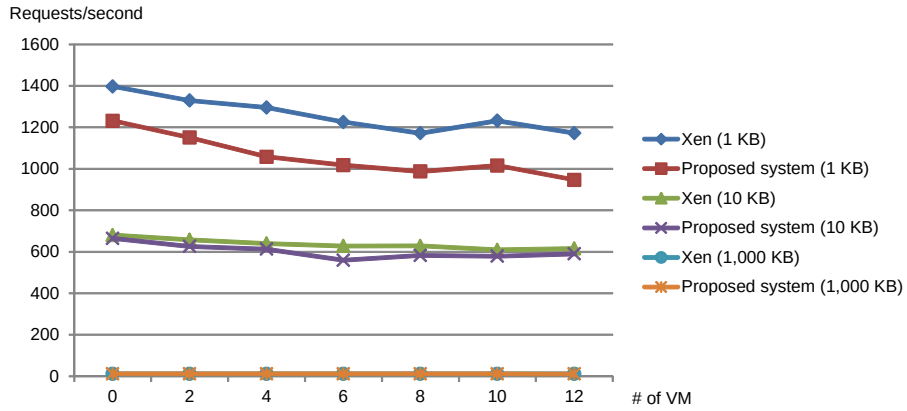

Fig. 5. Performance comparison in multi-VM environment. Horizontal axis shows the number of other VM. Vertical axis shows throughput of a web server in request/s; higher measurements are better.

[17] are distinguished as a protocol that provides security in transmission of log messages, not for storage phase. We focus on transmission phase because our proposal is highly related to that phase. Accorsi described that only reliable syslog fulfills security requirements that guarantee the authenticity of audit trails. Even if those protocols can detect and verify log message as not tampered, they cannot prevent deletion or tampering of logs. At this point, those protocols are different from our proposal. Therefore, this paper proposes a protection of log messages from a viewpoint of system security. By combining our proposal and existing secure logging protocol, we can increase security of logged data.

6.2 Logging with Virtual Machine

ReVirt [18] logs non-deterministic events on a VM for replay. Because it logs events for analysis of attacks, types of data are different from our proposal. While ReVirt logs instruction-level information, our proposal collects log messages for syslog. With our proposal, we can easily monitor the target VM without deep analysis of logged information because those logs are already formatted.

Virtual machine is also used to separate logged information [23]. While reference [23] separates information about file system logs, our proposal separates logs for syslog. They utilized split device driver model of Xen and it is provided for para-virtualization, thus, their proposal can be applied only for para-virtualized environment. Our prototype is implemented with fully virtualized environment; however, implementing in para-virtualized environment is easy.

VMI [11] and other introspection method [13] can be considered as a logging method with a VM. In that regard, these methods are similar to our proposal. However, information gathered by those methods are not formatted like syslog, therefore, to analyze these data, existing tools are unavailable. In contrast, with

our proposal, existing tools work well without modification because the format of information gathered by our proposal is same as messages produced by syslog. Our previous work [20] can gather information from a VM without modification to a library in that VM. However, to adapt to various OSes, it requires modification to a VMM. Modification to a VMM requires restart of all VMs on that VMM. Besides, it causes measurable overheads. By contrast, although modification to a library on a VM is required, our proposal in this paper requires no modification to a VMM to adapt to various OSes and has less overhead.

7 Conclusions

The secure log transfer method by replacing a library in a VM provides processes on a VM with an ability to transfer logs without involving the VM kernel. Thus, even though kernel-level malware tamper with logs on that VM, logs gathered by our proposal have no effect. In addition, we implemented the proposed system with VMM, therefore, attacking the proposed system from a target VM is difficult enough because of the property of a VMM. Further, adapting the method to various OSes is easy because of its implementation with library modifications. Evaluation of resistance for log tampering shows that tampering of logs from the target VM is difficult enough. From the experiment of adapting different OSes showed that an effort of adaptation is only 20 lines of additional code to `libc` library. Performance evaluation shows that performance degradation of syslog function is only about 6%. Performance degradation is negligible if a processing of an AP is I/O intensive. Performance evaluation in multi-VM environment shows that the proposed system has enough performance with many VMs.

References

1. Kent, K., Souppaya, M.: Guide to computer security log management, special publication 800-92 (September 2006)
2. spoonfork: Analysis of a rootkit: Tuxkit. http://www.ossec.net/doc/rootcheck/analysis-tuxkit.html
3. stealth: Announcing full functional adore-ng rootkit for 2.6 kernel. `http://lwn.net/Articles/75991/`
4. Symantec: Backdoor.lastdoor. `http://www.symantec.com/security_response/writeup.jsp?docid=2002-090517-3251-99`
5. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. Journal of Network and Computer Applications **34**(1) (2011) 1–11
6. Grobauer, B., Walloschek, T., Stocker, E.: Understanding cloud computing vulnerabilities. IEEE Security & Privacy **9**(2) (march-april 2011) 50–57
7. Marty, R.: Cloud application logging for forensics. In: Proceedings of the 2011 ACM Symposium on Applied Computing. SAC '11 (2011) 178–184
8. Chen, P.M., Noble, B.D.: When virtual is better than real. In: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems. HOTOS '01, Washington, DC, USA, IEEE Computer Society (2001) 133–138

9. Boeck, B., Huemer, D., Tjoa, A.M.: Towards more trustable log files for digital forensics by means of "trusted computing". International Conference on Advanced Information Networking and Applications (2010) 1020–1027
10. Seshadri, A., Luk, M., Qu, N., Perrig, A.: Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. SIGOPS Oper. Syst. Rev. **41**(6) (October 2007) 335–350
11. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: In Proc. Network and Distributed Systems Security Symposium. (2003) 191–206
12. Dewan, P., Durham, D., Khosravi, H., Long, M., Nagabhushan, G.: A hypervisor-based system for protecting software runtime memory and persistent storage. In: Proceedings of the 2008 Spring simulation multiconference. SpringSim '08 (2008) 828–835
13. Sharif, M.I., Lee, W., Cui, W., Lanzi, A.: Secure in-vm monitoring using hardware virtualization. In: Proceedings of the 16th ACM conference on Computer and communications security. CCS '09 (2009) 477–487
14. Adiscon: rsyslog. `http://www.rsyslog.com/`
15. Security, B.I.: Syslog server — syslog-ng logging system. `http://www.balabit.com/network-security/syslog-ng`
16. Kelsey, J., Callas, J., Clemm, A.: Signed syslog messages. `http://tools.ietf.org/html/rfc5848` (May 2010)
17. New, D., Rose, M.: Reliable delivery for syslog. `http://www.ietf.org/rfc/rfc3195.txt` (November 2001)
18. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: Revirt: enabling intrusion analysis through virtual-machine logging and replay. SIGOPS Oper. Syst. Rev. **36**(SI) (December 2002) 211–224
19. Ibrahim, A., Hamlyn-Harris, J., Grundy, J., Almorsy, M.: Cloudsec: A security monitoring appliance for virtual machines in the iaas cloud model. In: 2011 5th International Conference on Network and System Security. (Sep. 2011) 113–120
20. Sato, M., Yamauchi, T.: Vmm-based log-tampering and loss detection scheme. Journal of Internet Technology **13**(4) (July 2012) 655–666
21. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. **37**(5) (October 2003) 164–177
22. Accorsi, R.: Log data as digital evidence: What secure logging protocols have to offer? In: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 02. (2009) 398–403
23. Zhao, S., Chen, K., Zheng, W.: Secure logging for auditable file system using separate virtual machines. In: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications. (2009) 153–160