A Secure and Efficient Kernel Log Transfer Mechanism for Virtualization Environments*

DONGHAI TIAN1,2, JUNHUA CHEN3,+, CHANGZHEN HU1 AND JINGFENG XUE1

1Beijing Key Laboratory of Software Security Engineering Technique
Beijing Institute of Technology
Beijing, 100081 P.R. China
2Shanghai Key Laboratory of Integrated Administration Technologies for Information Security
3Key Laboratory of IOT Application Technology of Universities in Yunnan Province
Yunnan Minzu University
Kunming, 650500 P.R. China
E-mail: 1{dhai; chzhoo; xuejf}@bit.edu.cn; 3chenjunhuabj@163.com

Kernel logs are very important source of information for administrators to reconstruct security events. Once a sophisticated attacker intrudes a computer system, he (or she) may manipulate the kernel log to clear up the intrusion evidence. Previous solutions suffer from limitations in that: 1) Some methods do not provide adequate protection; 2) Some methods are not compatible with the existing systems or hardware; 3) Some methods incur considerable performance overhead. In this paper, we present SEKEL, a secure and efficient kernel log transfer mechanism based on virtualization technology. The basic idea of our approach is to decouple the kernel log collection and transfer procedures into two concurrent components. On one hand, the log collection component protected by the SIM framework is deployed in the target VM. On the other hand, the log transfer component is placed into a trusted execution environment for performance isolation. To deal with the synchronization problem introduced by our concurrent components, we extend Lamport’s ring buffer algorithm. The evaluation shows that SEKEL can protect kernel logs effectively with little performance degradation.

Keywords: kernel log transfer, virtualization, concurrent, synchronization, protect

1. INTRODUCTION

Logging information about sensitive activities occurred in a computer is very important for administrators to understand the computer security situation. Once an attacker intrudes a computer system, he (or she) will try to tamper with the logs, which could mislead administrators to identify the real intrusion. For example, the attacker may delete the related log items to hide the evidence of his (or her) malicious activities on a computer.

Unauthorized deleting and modifying the log items will compromise the log integrity. To address this problem, the access control mechanism can be applied to limit the attackers’ abilities. For instance, SELinux can ensure that only the specific process with high privilege can access the user log. However, this method is limited to protect the kernel log when advanced attackers exploit the OS kernel vulnerabilities to hijack the

Received September 24, 2015; revised December 1, 2015 & January 6, 2016; accepted January 14, 2016.
Communicated by Meng Chang Chen.
* This work was supported in part by the Opening Project of Shanghai Key Laboratory of Integrated Administration Technologies for Information Security and Open Found of Key Laboratory of IOT Application Technology of Universities in Yunnan Province.
+ The corresponding author.
To prevent illegal modification and detect tampering of logs, some methods [2] rely on the special hardware feature to encrypt kernel logs. To protect the logging information, some methods leverage the virtualization technology. Compared with the hardware-based methods, the VM-based methods [7, 8] can prevent log tampering even the kernel execution is hijacked by attackers. Since most of the VM-based methods rely on the underlying hypervisor for log collection and transfer, the frequent hypervisor involvement will incur considerable performance overhead. Moreover, some VM-based methods also require some modification to the OS kernel (or libraries), so these methods may not be practical for real deployment.

To address the above problems, in this paper, we present the design and implementation of SEKEL, a secure kernel log based on the virtualization technology. Compared with previous methods, SEKEL can protect the kernel log transparently with minimal performance cost. The basic idea of SEKEL is similar with the previous VM-based approaches. We utilize the virtualization technology to collect the kernel log in the target VM and then transfer it to a trust VM. Different from conventional VM-based approaches, we decouple the log collection and log transfer procedure into two different components, which can work in parallel. To protect the log collection component from being compromised by attackers, we make use of the SIM framework [9], which can achieve In-VM protection. To safeguard the log transfer component located in the target VM, we leverage the virtualization technology to deploy it in a trust execution environment. Furthermore, we introduce the direct memory mapping technique so that the log transfer component can access the kernel log of the target VM efficiently without the hypervisor involvement. To ensure the log collection and transfer components work concurrently, we extend Lamport’s ring buffer algorithm [6].

We have implemented a prototype of SEKEL based on the Xen hypervisor. To facilitate easy deployment of our system, we apply the stealth kernel module injection technique [3] to inject a kernel module into the target VM so that the kernel logging operation can be intercepted transparently. The experiments show that our system can protect the kernel log effectively with little performance cost.

In summary, our approach makes the following contributions:

- We apply the SIM, stealth kernel module injection, and inline hook technique to protect kernel logs transparently.
- We extend Lamport’s ring buffer algorithm to address the synchronization problem when the log collection and transfer components work in parallel.
- We design and implement SEKEL based on Xen. The evaluations show that our system can protect kernel logs effectively with small performance cost.

### 2. SECURITY ASSUMPTION AND THREAT MODEL

Our method to protect kernel logs is based on four security assumptions. First, we assume that the target operating system runs on the virtualization environment. Moreover, that our hypervisor is trusted thanks to its relatively small Trusted Computing Base (TCB). Second, we assume the kernel message to be logged is trusted before invoking
the kernel logging function. Third, we assume the trust VM is trusted. Finally, we assume the target guest kernel is trusted before our protection is deployed.

Our threat model allows attackers to hijack the kernel execution to compromise the kernel buffer that stores the kernel message to be logged. Furthermore, the attackers can disable the logging daemon, which is used to store the kernel log to a file. In addition, the attackers are allowed to manipulate the kernel log file directly. Under our threat model, the attackers are very powerful to temper with kernel logs.

3. OVERVIEW OF OUR APPROACH

Generally, the high-level idea of our approach is to utilize the virtualization technology to achieve kernel log protection. By securely transferring the kernel log from the target VM to trust VM, our system can well protect the kernel log from being tampered by kernel-level attackers. Unlike traditional VM-based methods that require frequent hypervisor involvement (e.g., trapping the kernel log event), our method does not need any hypervisor involvement during the kernel log protection. Therefore, our approach can greatly improve the system performance.

The rationale of our method is to decouple the kernel log collection and kernel log transfer procedures into two different modules. By utilizing the multi-core technology, these modules can work concurrently. Specifically, when a kernel log is generated, the collection module does not immediately transfer this information to the local kernel logging daemon for storing, but it just stores this information into the kernel buffer of the target VM. In the meanwhile, the transfer module is responsible for obtaining the log information from the kernel buffer and sending it to the Trust VM for being stored to a file. Since the transfer module is located in the trust VM, it is very difficult for attackers to compromise this module. To ensure the transfer module can efficiently access the log information of the target VM, we make use of the direct memory mapping mechanism to share the kernel buffer. Moreover, we extend the Lamport’s ring buffer algorithm [6] so that the collection module and transfer module can access the shared kernel buffer concurrently without unnecessary contention. On the other hand, to protect the collection module from being tampered, we make use of the SIM framework to construct a separate protection domain for placing this module in the target VM.

4. SYSTEM DESIGN AND IMPLEMENTATION

We have developed SEKEL, a prototype based on top of Xen hypervisor [1] to demonstrate our approach. The general architecture of our system is shown in Fig. 1. This architecture can be divided into three parts: Trust VM, Guest VM, and VMM (also called hypervisor). The Collector resides in the user space of the trust VM. Its main function is to collect the kernel log generated by the Guest VM. For this purpose, the Interceptor is injected into the kernel space of the Guest VM as the front driver to hook the kernel logging function so that the logging information can be transferred to the Collector. The Memory Mapper located in the VMM is responsible for mapping the kernel memory of the Guest VM to the Collector’s address space. By doing so, the Collector can access the kernel memory of the Guest VM directly.
4.1 Kernel Log Collection

To collect kernel log, a traditional VM-based method utilizes the debug breakpoint technique [13, 14] to trap a logging function call in the kernel space. However, doing so may incur considerable performance overhead in that the interception will result in the context switch between the guest OS and hypervisor. To address this issue, we make use of the SIM framework [9], which enables an In-VM agent to collect system events efficiently without sacrificing the security. Specifically, we utilize the Shadow Page Table (SPT) management subsystem in the hypervisor to divide the original kernel space into two separate address spaces, which are mutually exclusive with the same memory regions. In other words, SEKEL maintains an additional SPT to isolate the interceptor from the kernel space.

As shown in Fig. 2, the kernel log interceptor cannot access the kernel code regions in the interceptor address space, while the OS kernel cannot access the memory region of the interceptor in the kernel address space. However, the kernel log interceptor can ac-
cess its own and kernel data regions. To facilitate the normal execution transfer between the interceptor and OS kernel, a mechanism called transition gate is introduced for switching address spaces. The transition gate is the only memory region that has executable privilege in both address spaces, and its function is to set the CR3 register that contains the physical address of the target SPT. By default, any modification to the CR3 register by the guest OS will be intercepted by the hypervisor. As a result, these operations will also impose significant performance cost. To address this problem, we utilize the recent hardware feature (i.e., CR3 TARGET LIST), which enables the transition gate to modify the CR3 register without being trapped into the hypervisor.

After the interceptor address space is constructed, the interceptor can be placed into this address space. For this purpose, a straightforward method is to load a kernel module into the kernel address space and then notify the hypervisor to manipulate the memory permissions. However, doing so may require some human efforts to install additional kernel driver into the guest OS. Instead, we employ the stealth kernel module injection technique [3] for the interceptor deployment. By doing so, the guest OS will not be aware of the additional kernel component. The basic idea of this technique is to utilize the hypervisor’s higher privilege to hijack the guest OS’s execution and then inject the kernel module into the kernel memory.

To apply the SIM framework to the SMP environment, our system constructs multiple transition gates, each of which corresponds to a different CPU. Particularly, the interceptor needs to query the processor id to select the associated transition gate before the execution is transferred back from the interceptor address space to the kernel address space.

Before injecting the kernel code, we need to first select an unused memory region in the kernel space for placing the code. To do so, we can leverage the existing kernel memory allocation mechanism. Specifically, we first analyze the function body of kmalloc to locate its return instruction. Then, we exploit the online patching provided by the hypervisor to replace the return instruction with a trap instruction (i.e., INT3). After that, we register the INT3 handler in the hypervisor. By doing so, the hypervisor can trap the return operation of the kmalloc function. At that time, the new kernel memory region is allocated but not used yet. Thus, this memory region can be used to hold the kernel code temporarily. To inject the kernel code permanently, we construct an invocation gate and then put it into the temporary memory region. Next, this invocation gate will invoke a memory allocation function. To execute the invocation gate, we need to modify the EIP of the guest OS to the start address of the invocation gate when the execution is transferred back to the guest OS. After the invocation gate is executed, the newly allocated memory region can be used for placing the injected code. Thanks to the trap instruction, the hypervisor will re-trap the return operation of the allocation function. Then, the hypervisor should restore the EIP and emulate the return operation to continue the normal execution. Moreover, we should recover the return instruction in the end of the kmalloc function to avoid the unnecessary side effect when the allocation function is invoked again.

After the interceptor is injected into kernel memory, we exploit the in-line hook technique to intercept the kernel logging function. As shown in Fig. 3, the function prologue of the logging function will be rewritten by a jump instruction that transfers the execution control to the interceptor function. Then the interceptor can get the string
buffer to be logged from the kernel stack. After the interceptor finishes its task (i.e., log collection), it will transfer the kernel execution back to the original logging function. Since rewriting the logging function prologue will overwrite the subsequent instructions, the interceptor needs to resume the overwritten instructions in the end. It is worth noting that the overwritten instructions should be aligned to the instruction boundary so some NOP instructions are needed for padding the boundary gap.

4.2 Kernel Log Transfer

With the collected kernel log inside the interceptor, the next step is to transfer the log information to the trust VM. To this end, previous VM-based methods [7, 8] utilize the underlying hypervisor as the relay for log transfer. However, these methods suffer from limitations in that: (1) the inevitable VMM involvement (e.g., copying log from Guest VM to VMM) during the log collection procedure will introduce significant performance overhead when the guest OS generates logs frequently; (2) Storing the log inside the VMM may introduce unnecessary memory overhead. To address these problems, we propose a fast and secure kernel log transfer mechanism by utilizing the direct memory mapping and concurrent algorithm.

4.2.1 Direct memory mapping

The basic idea of this technique is to exploit the hypervisor to manipulate the shadow page tables of the VMs so that the Collector inside the Trust VM can directly access the kernel buffer of the Guest VM. To this end, the Collector should first allocate a chunk of memory. Due to the on-demand paging mechanism, the page table entries (PTEs) are not created initially. Therefore, the Collector needs to read the newly allocated memory space to trigger the PTE creation. Then, the hypervisor updates the page frame address of the PTE to the real physical address of the target buffer, which can be obtained from the guest VM’s physical-to-machine (P2M) table. In this way, the Collector can access the guest VM’s kernel buffer in its own address space.
4.2.2 Concurrent algorithm

The basic log transfer mechanism is shown in Fig. 4. When the kernel log is generated, the Interceptor will first capture this event and then store the log string into the express data structure. In the meanwhile, the Collector is responsible for collecting the log information from the express data structure. In order to ensure the Interceptor and Collector can work in parallel, we make use of Lamport’s ring buffer as the express data structure for efficient log transfer.

![Fig. 4. Log transfer mechanism.](image)

The specific algorithm is illustrated in Fig. 5. This algorithm mainly includes two functions. On one hand, the kernel log Interceptor has to invoke the Enqueue function (Line 17-33) to insert the log string into the shared ring buffer. On the other hand, the Collector needs to invoke the Dequeue function (Line 34-49) to fetch the data from the ring buffer. Different from conventional Lamport’s algorithm, we extend the basic ring buffer to a linked list of ring buffers. By doing so, the potential failure of Enqueue operation can be avoided when the ring buffer gets full. To ensure the algorithm work stably, the size of the ring buffer will grow exponentially (Line 23).

Initially, only one ring buffer is shared between the Interceptor and Collector. When a new ring buffer is allocated by the Interceptor, the Collector will not know the physical address of this new buffer because these two components reside in different VMs. Therefore, the Memory Mapper inside the hypervisor should be notified to map the physical address of the new buffer to the Collector’s address space so that it can be directly accessed by the Collector. To access the new buffer, the Collector should translate the address of the target VM to its own address (Line 42).

Since our log transfer mechanism relies on the ring buffers located in the kernel space of the guest VM, advanced attackers may exploit kernel rootkits to temper with the ring buffers. As a result, the compromised kernel log may mislead administrators to analyze the potential intrusion. To address this issue, we propose a secure method for verifying log integrity. More precisely, we employ a MAC (Message Authentication Code) function to generate a secure tag for each kernel log item. The tag is actually a MAC value, and it is attached to the end of each log item. Since the MAC function requires a secret key to generate MACs, the key should be protected from being stolen by attackers. Thanks to the SIM protection mechanism [9], the secret key can be kept inside the Interceptor’s address space so that any code outside of the Interceptor cannot access the secret key. To verify whether the log item is compromised, we just re-calculate the MAC value and then compare it with the secure tag. If they are not equal, it indicates the log item is tempered. Then, the hypervisor will respond accordingly (e.g., suspending the guest VM).
4.3 Kernel Log Storing

With the kernel log collected inside the Collector, the next step is to store the kernel log to the file. To this end, we can make use of the syslog function. Since a writing file operation is relatively slow, we need to minimize the number of writing operations. For this purpose, we can also apply Lamport’s ring buffer algorithm. As shown in Fig. 6, the Collector temporarily stores the kernel log string into the ring buffer. Then, the logging module fetches the log string from the ring buffer and writes it to a file. In order to address the performance issue, the logging module only carries out real writing operations when it consumes a half of the ring buffer.
In this section, we evaluate both the detection effectiveness and the performance of SEKEL. All the experiments are carried out on a Dell PowerEdge T410 workstation with two 2.13G Intel Xeon E5606 CPUs and 16 GB memory. Xen 3.4 is used as the VMM. For the guest VM, four virtual CPU (VPU) and 4 GB memory are allocated to it. We use the Fedora 12 (linux-2.6.31) as the Trust VM, and Ubuntu 8.04 (linux-2.6.24) as the guest VM.

5.1 Effectiveness

To evaluate the effectiveness of SEKEL to protect the kernel log, we carry out several kernel-level attacks. First, we utilize a kernel-level malware (i.e., adore-ng [11]) to tamper with the kernel log buffer that will be transferred to a logging daemon for being stored to a local storage. Second, we exploit a similar kernel-level malware (i.e., tuxkit [10]) to attack the logging daemon (i.e., klogd) so that the target kernel log will not be stored to a file. Third, we make use of a kernel-level rootkit to escalate privilege for a normal process. Then, we exploit a buffer overflow vulnerability of the process to hijack its execution to overwrite the kernel log file (i.e., /var/log/kern.log). With our protection mechanism, all the kernel logs are effectively saved another copy in the Trust VM. Moreover, we can identify the compromised kernel log by comparing the collected kernel log in the Trust VM with the native one in the target VM. Fourth, we implement a custom rootkit to modify the interceptor’s code and data by utilizing the /dev/kmem device. Thanks to the Secure-In-VM protection, our system successfully prevents the unauthorized modification. Fifth, we implement a malicious kernel module to tamper with the ring buffer that temporarily stores the log string for concurrent log transfer. By verifying the secure tag, our system can identify that the log string integrity is compromised. Finally, we test a custom kernel module that invokes a large number of kernel logging functions. The experiment shows that our system can collect all the kernel logs generated by this kernel module in the Trust VM.

5.2 Performance Overhead

To evaluate the performance cost of our protection mechanism, we conduct a set of performance experiments. First, we execute the micro-benchmark to evaluate the performance cost of invoking a kernel logging function in a kernel module. To this end, we utilize the rdtsc instruction to record the hardware timestamp right before and after the kernel function invocation. In our test, the native system takes 5073 CPU ticks (the max
and min values are 5386 and 4825) for the kernel function to finish its task. Due to our added protection mechanism, SEKEL needs to spend 7128 CPU ticks (the max and min values are 7572 and 6837) for the protected kernel function to complete the operation.

Table 1. Running overhead of SEKEL.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Native performance</th>
<th>SEKEL approach</th>
<th>Out-of-VM approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lighttpd request throughput</td>
<td>7835.72#/s</td>
<td>8.73%</td>
<td>35.39%</td>
</tr>
<tr>
<td>Nginx request throughput</td>
<td>8729.41#/s</td>
<td>9.51%</td>
<td>37.12%</td>
</tr>
<tr>
<td>Apache transfer rate</td>
<td>6382Kb/s</td>
<td>6.43%</td>
<td>29.57%</td>
</tr>
<tr>
<td>Apache response time</td>
<td>76.27ms</td>
<td>7.25%</td>
<td>32.68%</td>
</tr>
</tbody>
</table>

To evaluate the application level performance cost, we also carry out some application benchmarks in the target VM with our protection. Since the traditional applications cannot generate kernel logs, we need to modify the applications to trigger kernel log generation. For this purpose, we have to install a custom kernel module into the kernel space. By utilizing the ioctl mechanism, the application can notify the kernel module to invoke the kernel logging function. In our test, the Apache web server is modified for triggering logging 256 bytes string into a kernel log when it invokes a library function. Similarly, we also modify the Nginx and Lighttpd servers to notify the kernel module for generating kernel logs when they invoke the library functions. Then, we utilize the ApacheBench program to measure the average transfer rate and response time of these web servers, all of which are configured to serve a 72KB php webpage. In addition to running the above benchmarks under SEKEL’s protection, we also conduct these experiments in the native Guest VM without protection and the protected VM that relies on the Out-of-VM approach [7, 8], which requires the hypervisor to monitor the kernel logging function invocations. Specifically, the Out-of-VM approach first utilizes the online patching technique provided by the hypervisor to patch the prologue of the kernel logging function with the vmcall instruction. By doing so, the kernel execution will be transferred to the hypervisor when a kernel logging function is invoked. Then, the hypervisor can retrieve the kernel logging information and transfer it to the trust VM.

Table 1 shows the result of these application benchmarks. We can see that the performance overhead introduced by SEKEL is much smaller than the Out-of-VM approach. In general, the performance cost relies on the frequency of the kernel logging function invocation. If the application triggers a lot of kernel logging operations, the add-on overhead would be a little higher. Otherwise, the add-on cost could be ignored. The add-on overhead is mainly due to the cost of the log interception, which needs to switch address spaces when the interceptor is invoked.

5.3 Kernel Log Transfer Delay

In order to evaluate the kernel log transfer delay, we develop a kernel module to invoke a great number of kernel logging functions. Then, we leverage the rdtsc instruction to measure the average transfer time from the interceptor (inside the guest VM) to
collector (inside the trust VM) for a single kernel log item that contains 256 bytes. The evaluation shows the average transfer time is about 629 CPU ticks (the max and min values are 708 and 576). The transfer time mainly consists of two parts: the access time to the ring buffer and generation time for the secure tag.

6. RELATED WORK

File integrity check methods [5, 12] have been well studied for many years. They are very effective in detecting log file changes. However, they are limited to defend against attacks that manipulate the kernel log buffer before the inside information is stored into a file. Isohara et al. [4] present a LSM-based monitoring scheme to defend against attacks that attempt to manipulate the monitoring module and its result. Unfortunately, this scheme could be bypassed if advanced attackers exploit the kernel vulnerabilities. Zhao et al. [15] propose a log protection based on the virtualization technology. This method makes use of shard memory and a separate VM to save logs. Since this method uses a split device driver model of Xen to achieve high performance, it can be only applied in a para-virtualization environment. Boeck et al. [2] propose a TPM-based method to protect the log-producing application from being tampered. Since this approach depends on the special hardware feature (e.g., Secure Virtual Machine), it may not be practical for wide deployment. Sato et al. [7] propose a log-storing module and a tamper detection scheme in the virtualization environment. This approach relies on the hypervisor to store logs in a separate VM. As a result, the hypervisor needs to trap each log event. When the log is generated frequently, it incurs significant performance overhead. Recently, Sato et al. [8] present a secure and fast log transfer mechanism. This method requires some modifications to a library to trigger a log transfer to a hypervisor. Thus, it may not be applied to protect the recent commodity system, whose libraries cannot be changed. Moreover, this method also introduces considerable performance cost for the log transfer.

7. CONCLUSION

In this paper, we present SEKEL, a novel kernel log transfer mechanism by utilizing the virtualization technology. We combine the Out-of-VM and In-VM approaches to achieve a secure and efficient kernel log protection. To ensure the kernel log collection and transfer components work in parallel, we make use of Lamport’s ring buffer algorithm. By applying the kernel module injection technique, our protection system is compatible with the existing OS. The evaluations show that SEKEL can protect kernel logs effectively with good performance.

REFERENCES


Donghai Tian (田东海) received his Ph.D. degree in Computer Science from Beijing Institute of Technology in 2012. He is currently an Assistant Professor in School of Software, Beijing Institute of Technology, China. His research interests include system security and virtualization technologies.

Junhua Chen (陈君华) received his Ph.D. degree in Computer Science from Beijing Institute of Technology in 2010. He is currently an Associate Professor in Key Laboratory of IOT Application Technology of Universities in Yunnan Province, Yunnan Minzu University, China. His research interests include information security and Internet of things.

Changzhen Hu (胡昌振) received his Ph.D. degree in Mechatronics Engineering from Beijing Institute of Technology in 1996. He is currently a Professor at Beijing Institute of Technology. His research interests include information security, network, and pattern recognition.

Jingfeng Xue (薛静锋) received his Ph.D. degree in Computer Science from Beijing Institute of Technology in 2003. He is currently a Professor at Beijing Institute of Technology. His research interests include information security and software engineering.