A Secure Framework for Monitoring Operating Systems Using SPEs in Cell/B.E.

Kenichi Kourai
Kyushu Institute of Technology
kourai@ci.kyutech.ac.jp

Takuya Nagata
Kyushu Institute of Technology
takuya@ksl.ci.kyutech.ac.jp

***Abstract*—Recently, even operating systems are often compromised by the attackers. Since a compromised operating system affects all the applications including security software on top of it, the integrity of the operating system should be guaranteed. However, it is difficult to monitor the operating system securely. In this paper, we propose *SPE Observer*, which is a framework for securely monitoring operating systems using SPEs in Cell/B.E. SPE Observer guarantees the integrity and confidentiality of monitoring systems by the *isolation mode* of SPEs. To complement the isolation mode, SPE Observer monitors the running status of monitoring systems from an external *security proxy*. In addition, it schedules monitoring systems to mitigate the performance degradation of applications due to occupying SPEs. We have implemented SPE Observer in PlayStation 3 and developed the integrity monitor of the operating system. According to our experiments, it was shown that the integrity monitor on an SPE could detect a compromised operating system and that the application performance was dramatically improved by scheduling the integrity monitor.**

I. Introduction

In recent years, computer systems suffer from various attacks from the Internet. The users usually protect their systems with security software such as antivirus. Such security software monitors the system with the functionalities of the operating system. However, the operating system is not an exception of the attacks. Many kernel rootkits such as Adore-ng [1], Knark [2], and SucKIT [3] have been found. If the operating system is compromised, the monitoring results of security software become unreliable.

To increase the reliability of security software, the integrity of the operating system should be guaranteed. However, if monitoring systems run on top of or inside the operating system, it is difficult to examine that the operating system is not compromised. Therefore, several methods for running monitoring systems underneath the operating system have been proposed, but they are not sufficient. For example, remote attestation using the TPM [4] guarantees the integrity of the operating system only at boot time. Hardware supports enable monitoring systems to be executed securely [5], [6], [7], [8], but the system performance degrades because a monitoring system cannot be run with the other tasks on the other CPU cores. Virtualization technology is promising to run monitoring systems underneath the operating system [9], [10], [11], but the hypervisors implemented by software may have vulnerabilities [12], [13].

In this paper, we propose a framework for securely monitoring operating systems using SPEs in Cell/B.E., which is named *SPE Observer*. Cell/B.E. has two types of cores: a PPE for the operating system and SPEs for parallel applications. SPE Observer executes a monitoring system on an SPE to check the integrity of the operating system on the PPE. To protect the integrity and confidentiality of the monitoring system from the attacks, SPE Observer runs the SPE in the *isolation mode*[1]. Since the isolation mode still allows the PPE to stop SPEs for controlling the whole system, SPE Observer monitors the running status of the monitoring system by sending heartbeats from an external *security proxy*. In addition, to mitigate the performance degradation of applications due to using SPEs for the monitoring purpose, SPE Observer schedules monitoring systems so that SPEs are not always occupied.

We have implemented SPE Observer in PlayStation 3 and developed a monitoring system that checked the integrity of the kernel memory as an example. We modified the Linux kernel to allow monitoring systems to access the kernel memory and to schedule SPEs appropriately. Using SPE Observer, we conducted several experiments to confirm that the integrity monitor could detect compromised operating systems and how the application performance was affected. It was shown that the integrity monitor on an SPE could distinguish between legitimate and compromised kernels. Scheduling of monitoring systems could mitigate the performance degradation of parallel applications. Particularly, the performance of an application whose threads were synchronized became 4.6 times higher at the scheduling interval of 200 ms.

The rest of this paper is organized as follows. Section II describes issues in the existing frameworks for monitoring operating systems. Section III proposes SPE Observer and Section IV explains its implementation. Section V shows the results of experiments using SPE Observer. Section VI describes related work in detail and Section VII concludes this paper.

II. Monitoring Operating Systems

To increase the reliability of the system, it is necessary to check the integrity of the operating system. However,

[1]Strictly speaking, this is a mode for an SPU inside an SPE. In this paper, we identify an SPU with an SPE for simplicity.

it is not easy to securely monitor the operating system. Monitoring systems for the operating system often run on top of the operating system using system calls [14], [15]. If the operating system is compromised by the attackers, e.g., using kernel rootkits, the results of the system calls cannot be trusted. For example, a compromised operating system may prevent the notification of security breaches. It can even stop monitoring systems although the risk of being detected increases. Monitoring systems can be embedded into the operating system kernel [16], [17]. They can directly monitor the operating system through the kernel memory without depending on the functions of the operating system, but it is not guaranteed that monitoring systems work correctly. Monitoring systems running inside a compromised operating system can be also compromised easily.

To solve this problem, many researchers have proposed methods for monitoring the operating system with the virtual machine technology [9], [10], [11]. The hypervisor virtualizes the hardware to create multiple virtual machines, in each of which the operating system runs. Since the hypervisor runs underneath the operating system, it can securely monitor the operating system in a virtual machine. In several architectures such as Xen, Hyper-V, and vSphere Hypervisor, privileged virtual machines can also monitor the operating system in normal virtual machines. In either case, the monitoring system can run securely without interference with a compromised operating system. However, the hypervisor and the privileged virtual machines can be also compromised because they are constructed by software. For example, it has been reported that a normal virtual machine could attack the hypervisor [12], [13].

With the help of hardware such as the TPM [4], the remote attestation of the operating system can be performed. It guarantees the integrity of the operating system if a chain of trust is established from the TPM. For example, ROM code measures the BIOS, the BIOS measures the boot loader, and then the boot loader measures the operating system, in turn. All the calculated hash values of these components are stored in the TPM securely and verified by a trusted third party. It is difficult for the attackers to compromise the TPM, which is a root of trust. However, this static attestation is performed only once at boot time. If the operating system is compromised at runtime, that cannot be detected. Dynamic attestation performed at runtime has been proposed for applications [18], but it cannot be applied to the operating system if the operating system is the lowest software layer.

Using several capabilities of processors, monitoring systems can be securely executed at runtime. System Management Mode (SMM) provided in the x86 processor families is used for monitoring the integrity of the operating system [5], [6], [7]. When a CPU enters SMM via interrupts, it can securely execute monitoring systems without interferences by the attackers. Also, Intel TXT and AMD SVM enable monitoring systems to be executed in an execution environment that has not been compromised [8]. However, while a monitoring system is running in such an execution environment, all the other tasks including the operating system are suspended to avoid attacks from the other CPU cores. This is critical for coming many-core processors because only one core can be used during monitoring.

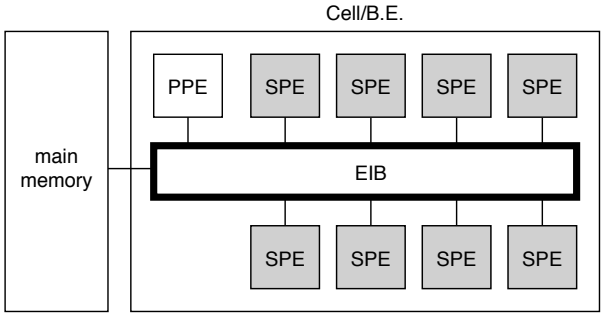

Figure 1. The architecture of Cell/B.E.

Therefore, we use Cell Broadband Engine (Cell/B.E.) providing a unique security feature called the *isolation mode* [19]. Cell/B.E. is a heterogeneous multicore processor that consists of a PowerPC Processor Element (PPE) and Synergistic Processor Elements (SPEs), as depicted in Figure 1. A PPE is a control processing core and executes an operating system and regular processes. SPEs are arithmetic processing cores and execute parallel applications. Each SPE consists of a Synergistic Processor Unit (SPU) and the memory called a local store, which is isolated from the main memory physically. It loads a program into its local store and moves data between the main memory and the local store using DMA. The isolation mode of an SPE can protect the local store from the other cores and all the devices. Nevertheless, an isolated SPE can securely run with the other SPEs and the PPE in parallel.

III. SPE Observer

This paper proposes *SPE Observer*, which is a framework for monitoring operating systems using isolated SPEs in Cell/B.E. Figure 2 illustrates the system architecture of SPE Observer. In SPE Observer, a monitoring system runs on an SPE in the isolation mode. An external security proxy bridges internal and external networks and monitors the running status of the monitoring system.

A. Threat Model

We assume that the attackers can compromise the operating system on the PPE and applications running on SPEs in the normal mode. However, we do not assume that there are remotely-exploitable hardware vulnerabilities in Cell/B.E. That is, the local stores of SPEs are correctly protected in the isolation mode and the secret key in the hardware cannot be extracted. Note that PlayStation 3 with Cell/B.E. has been cracked and its secret key has been exposed. We believe

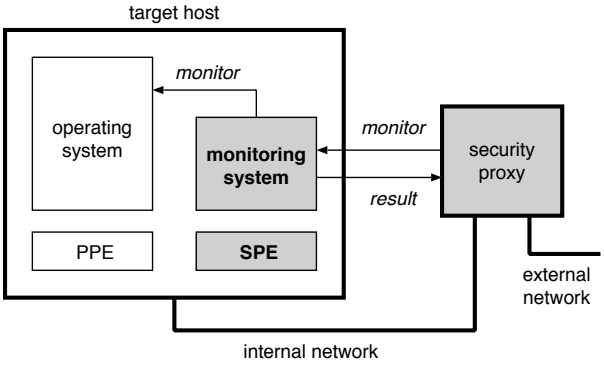

Figure 2. The system architecture of SPE Observer.

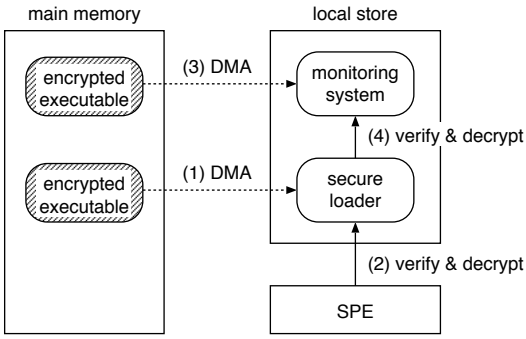

Figure 3. Loading a monitoring system using the secure loader.

that the vulnerability is due to an implementation issue in PlayStation 3 but is not inherent in Cell/B.E. In addition, we do not consider any attacks against the security proxy. Since the security proxy provides only the services of bridging networks and monitoring the running status of monitoring systems, protecting the security proxy is relatively easy.

B. Secure Execution in the Isolation Mode

SPE Observer securely executes monitoring systems on SPEs with the isolation mode enabled. The isolation mode protects the local store in an SPE and prohibits the PPE and the other SPEs from accessing the local store. Therefore, the integrity of monitoring systems running on isolated SPEs is guaranteed because their code and data are located in the protected local stores. The attackers cannot modify the running program or processing data on isolated SPEs by DMA from the outside. Also, the confidentiality of monitoring systems running on SPEs is guaranteed. The attackers cannot analyze the programs of monitoring systems or steal sensitive information such as secret keys in the local stores.

SPE Observer securely loads a monitoring system into the local store in an SPE using a secure loader (or isolation loader) [20]. Figure 3 shows the procedure. First, it loads an encrypted executable of the secure loader into the local store in an SPE. The SPE verifies the digital signature of the secure loader. If the secure loader is not modified, the SPE decrypts it with the secret key embedded into the hardware. Next, the secure loader loads an encrypted executable of a monitoring system into the local store. It verifies the digital signature of the monitoring system and decrypts it with its secret key.

The secure loader guarantees the integrity and confidentiality of the executables of monitoring systems. For the integrity, the attackers cannot make SPE Observer to load compromised monitoring systems into the local stores of isolated SPEs. The secure loader verified by the hardware checks their integrity. Since the attackers do not know the private key for signatures, they cannot sign their own monitoring systems correctly. For the confidentiality, the attackers cannot decrypt the executables of monitoring systems because they do not know the secret key for decryption. The key is embedded into the secure loader and the secret key for decrypting the secure loader is in the hardware.

As such, the isolation mode and the secure loader maintain the integrity and confidentiality of monitoring systems executed on SPEs. However, the PPE can still stop monitoring systems running on SPEs even if SPEs run in the isolation mode. This function is necessary because the PPE controls the whole system including SPEs in usual computation. For example, the PPE has to terminate an infinitely-looping program in an SPE. To detect the PPE maliciously stopping monitoring systems, we use a security proxy, which is described in the next section. Note that the confidentiality of monitoring systems is still guaranteed even if the attackers can stop SPEs. When the isolation mode in an SPE is disabled, the contents of the local store and the registers are erased by the hardware. The PPE cannot steal data of the terminated monitoring system by DMA or loading another program into the SPE.

C. Security Proxy

To monitor the running status of monitoring systems on SPEs, an external security proxy sends heartbeats to monitoring systems periodically. The security proxy judges that a monitoring system is stopped if it loses the responses to its heartbeats. The timeout period of heartbeats has to be configured carefully to prevent false positives. If the attackers on the PPE stop monitoring systems, they can avoid detecting a compromised operating system. This means that security software running on the operating system also becomes untrustworthy. However, the external monitoring by the security proxy could deter the attackers from stopping monitoring systems on SPEs.

The security proxy sends encrypted messages for heartbeats so that only a legitimate monitoring system can respond correctly. It is difficult for the outside attackers to decrypt the messages. Since the secret key for decryption is stored in the local store of an SPE, the attackers cannot steal

it thanks to the memory protection in the isolation mode. After a monitoring system decrypts the received message, it generates a new message depending on the decrypted message, encrypts it, and sends it to the security proxy. The outside attackers cannot also decrypt the reply message.

If the security proxy receives an incorrect response or no response in a certain period, it cuts the network between the monitoring host and the outside. In this case, it considers that the host has been compromised. Since the security proxy forwards packets from the monitoring host to the outside and vice versa, it can drop all the packets between them. This prevents the attackers from mounting outgoing attacks from the compromised host or from leaking sensitive information to the outside. Also, the outside attackers cannot access the compromised host using backdoors. Although attack programs may continue to run in the compromised host, the damages are localized on the host. The attackers cannot stop monitoring systems if they want to use the network.

Also, the security proxy receives monitoring results from monitoring systems on SPEs. Like heartbeats, the results are encrypted by the monitoring systems and the outside attackers cannot modify them without being detected by the security proxy. If the result indicates that the integrity of the operating system is not preserved, the security proxy ceases network bridging to separate the monitoring host.

D. Scheduled Monitoring

SPE Observer needs at least one SPE for executing a monitoring system, whereas Cell/B.E. is a processor that gains performance by parallel processing with many SPEs. If monitoring systems always occupy one or more SPEs, the performance of the other applications degrades in proportion to the number of occupied SPEs. However, an operating system may not always need to be monitored as frequently as possible. For example, it would be sufficient to check the integrity of the operating system kernel every second.

To mitigate the performance degradation of applications, SPE Observer enables monitoring systems to be scheduled. Whenever it executes a monitoring system, it performs a context switch in a victim SPE if necessary and securely loads the executable of the monitoring system into the local store using the secure loader. Although such scheduling suffers from the extra overhead of starting monitoring systems every time, the other applications can use all the SPEs when monitoring systems do not run. The SPE scheduling depends on the operating system that may be compromised, but the security proxy is responsible for monitoring the periodic execution of monitoring systems.

IV. Implementation

We have implemented SPE Observer using the Cell/B.E. SDK 3.1 and the Security SDK, which were provided by IBM. The target operating system running on the PPE is Linux 2.6.27.

A. Secure Execution on SPEs

SPE Observer emulates the isolation mode using the Security SDK because we could not obtain the secure loader that supported the isolation mode of SPEs at the hardware level. Unlike the hardware-level isolation mode, the secure loader that is not encrypted is loaded into the local store of an SPE. To make a monitoring system occupy one SPE while it is running, SPE Observer executes it with the SPE_NOSCHED flag. This flag prevents the SPE scheduler from switching the context of the specified thread.

B. Monitoring the Kernel Memory

To allow an SPE used for a monitoring system to access the kernel memory, SPE Observer first clears the Problem-State bit in the status register of the memory flow controller (MFC). The MFC is included in each SPE and performs DMA transfers between the local store and the main memory. The Problem-State bit is usually set so that the MFC can access only the process memory.

Second, SPE Observer registers an address mapping for the kernel memory to a segment lookaside buffer (SLB) of the SPE. In Cell/B.E., the main memory including the kernel memory is accessed by using effective addresses. An effective address is mapped onto a virtual address, which is mapped onto a physical address. The SLB is a translation table from effective addresses to virtual addresses and is included in each SPE. An SPE can access the main memory if the corresponding address mapping exists in the SLB.

As an example, we have developed a monitoring system that checks the integrity of the kernel memory. This integrity monitor obtains the text and read-only data segments of the operating system kernel using DMA. Note that the read-only data segment includes the system call table. Then it calculates the SHA-1 hash value of these memory regions using libspucrypt and compares the value with the pre-calculated one. To overlap DMA transfers from the kernel memory with the calculation of the hash value, the integrity monitor performs double buffering. It prepares two buffers, one for calculation and the other for a DMA transfer. While it calculates the hash value of a memory block in one buffer, it transfers the next memory block to the other buffer using DMA.

It is possible to implement other types of monitoring systems for SPE Observer. For example, a monitoring system can monitor dynamic data in the kernel, such as the process list and the module list, by using a technique similar to virtual machine introspection [9]. This technique enables monitoring systems to access the kernel data structures through debug information of the kernel.

C. Heartbeats

To implement heartbeats from the security proxy, SPE Observer runs a *relay process* on the PPE, as illustrated in Figure 4. The relay process forwards encrypted messages

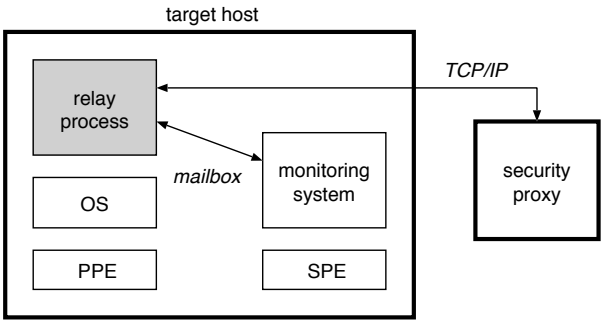

Figure 4. Heartbeats forwarded by the relay process.

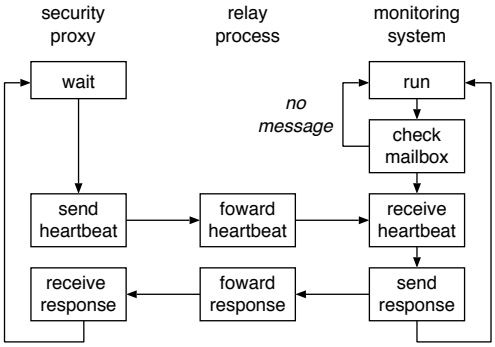

Figure 5. The flow chart of handling heartbeats.

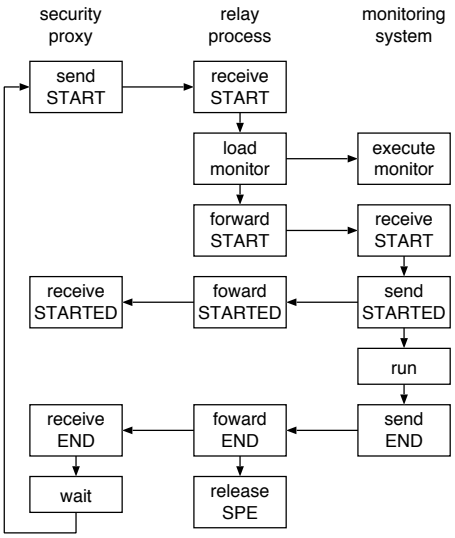

Figure 6. The flow chart of scheduling monitoring systems.

between the security proxy and a monitoring system on an SPE. It is possible to directly communicate between the security proxy and an SPE by running protocol stacks in the SPE [21]. However, this approach makes it difficult to implement a monitoring system in the same SPE because implementing protocol stacks consume a large amount of local store.

The security proxy and the relay process on the PPE communicate using TCP/IP, while the relay process and a monitoring system communicate using a mailbox of an SPE. A mailbox is prepared for passing data between the PPE and an SPE via mailbox channels in the MFC. Using a mailbox, the PPE can communicate with SPEs running even in the isolation mode.

Figure 5 is the flow chart of handling heartbeats between these three components. When the security proxy sends a heartbeat, it generates a random message, encrypts it with a pre-shared key, and sends it to a target host. The pre-shared key is embedded into a monitoring system and is shared with the security proxy. The relay process receives the encrypted message and passes it to the specified monitoring system on an SPE using its mailbox. The monitoring system reads the message in the mailbox, decrypts it, and generates a reply message depending on the random message in the heartbeat. Then it encrypts the reply message and writes it to its mailbox. After the relay process reads the reply message in the mailbox and sends it to the security proxy, the security proxy decrypts the message and checks the correctness of the reply.

Although the relay process on the PPE can be compromised, this heartbeat mechanism still works well. If a compromised relay process modifies encrypted messages between the security proxy and a monitoring system, either the monitoring system or the security proxy can detect that by checking decrypted messages. If the relay process drops messages, the security proxy can detect that by the timeout. As a result, the security proxy can cease bridging the networks. Since the security proxy sends encrypted random messages that change every time, it is difficult for the relay process to mount replay attacks.

D. Scheduling of Monitoring Systems

Monitoring systems are scheduled by the security proxy and the SPE scheduler in a target host. The security proxy determines when to execute monitoring systems. The flow chart of this scheduling is illustrated in Figure 6. When it starts a monitoring system, it sends the START message to the relay process on the PPE. Then the relay process loads the specified monitoring system into an SPE and sends the START message to the monitoring system via the mailbox of the SPE. When the monitoring system reads the START message, it returns the STARTED message to the security proxy via the relay process. When the monitoring system completes its task, it sends the END message with the monitoring result to the security proxy, terminates itself, and releases the SPE. After the security proxy receives the END message, it waits for the next time for starting the monitoring system. Note that these messages are encrypted like heartbeats. If the attackers modify or drop the messages, the security proxy can detect that.

When a monitoring system is loaded into an SPE, the

SPE scheduler in the Linux kernel selects one of the SPEs. It first looks for idle SPEs and allocates one of them if such SPEs exist. If there is no idle SPE, the SPE scheduler selects one of the SPEs used by applications as a victim and saves the contents of the local store and the state of the SPE in the main memory for a context switch. Then the monitoring system is executed with the SPE_NOSCHED flag.

We modified the SPE scheduler so that the most appropriate SPE is selected as a victim. The existing SPE scheduler had a bug in that a program executed with the SPE_NOSCHED flag is not scheduled forever as long as there is no idle SPE. To work around this bug, our SPE scheduler maximizes the priority of a program executed with that flag, so that the program is scheduled immediately. In addition, the existing SPE scheduler switched the context of only a particular SPE. For fairness, our SPE scheduler selects one of the SPEs in a round-robin fashion.

V. Experiments

For a machine with Cell/B.E., we used Sony Computer Entertainment PlayStation 3, which had six SPEs available for the users. Each SPE had 256 KB of a local store. The machine had 256 MB of XDR memory and 80 GB of a disk. We installed Fedora 9 to this machine. For a security proxy, we used a PC with one Intel Xeon processor E5630, 4 GB of memory, and two Gigabit Ethernet NICs. These two machines were connected with a Gigabit Ethernet switch.

A. Integrity Checking of the Kernel

We examined that the integrity monitor on an SPE could detect the kernel modification in SPE Observer although the detection algorithm itself is outside the scope of this paper. As described in Section IV-B, the integrity monitor compares the hash value of the text and read-only data segments in the kernel with the pre-calculated one. For comparison, we ran on the PPE the original Linux kernel, the kernel whose system call table was modified, and the kernel whose function for a system call was modified. Consequently, the integrity monitor could correctly judge that only the original kernel was not compromised. This means that the integrity monitor can detect kernel rootkits that modify the system call table, such as Adore-ng and Knark. It can detect SucKIT, which modifies the pointer itself to the table.

Next, we measured the time needed for checking the integrity of the kernel. The integrity monitor on an SPE obtained 12 MB of the kernel memory by DMA transfers and calculated the SHA-1 hash value. It periodically checked heartbeats from the security proxy and returned the responses whenever it received heartbeats. The security proxy sent a heartbeat every second. The execution time of this integrity monitor was 24 ms, which was from when the PPE started to load the integrity monitor until it released the used SPE. For calculating the hash value, it took 17 ms, which was 70 % of the total execution time. The remaining time was consumed mainly for responding heartbeats. The overhead of DMA transfers was hidden by the hash calculation.

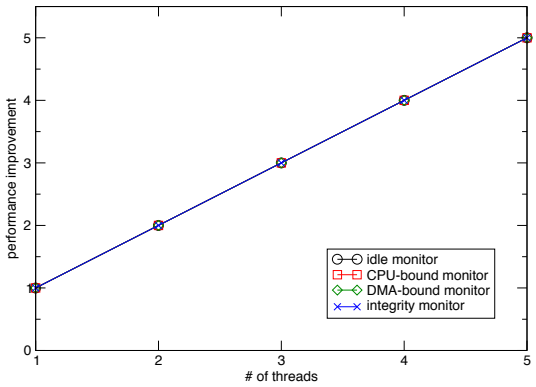

Figure 7. The performance of a CPU-bound application for various monitoring systems.

B. Impacts on Application Performance

We executed various types of monitoring systems on one SPE and examined how each monitoring system affected the performance of parallel applications. For monitoring systems, we used the integrity monitor of the kernel, a CPU-bound monitor, a DMA-bound monitor, and an idle monitor. The integrity monitor repeated DMA transfers and hash calculation in this experiment. The CPU- and DMA-bound monitors repeated only calculation and DMA transfers, respectively, and the idle monitor did nothing. The security proxy did not send heartbeats to these monitoring systems to prevent external disturbance. For each monitoring system, we executed three types of applications as we changed the number of application threads between one and five. Since an application and a monitoring system used six SPEs at most, context switches did not occur for SPEs.

First, we executed a CPU-bound application and measured the execution time. This application performed a certain amount of calculation with the specified number of threads. Figure 7 shows the performance of this application when we executed one of various monitoring systems together. The performance is normalized by that in sequential execution. The results show that any monitoring systems did not affect the performance of the CPU-bound application at all because both computation and DMA transfers did not compete with each other. The performance increased in proportion to the number of application threads. When the application used five threads, the performance became 5 times.

Second, we executed a DMA-bound application, which obtained a certain amount of memory from the main memory by DMA transfers. We measured the time needed for completing the DMA transfers. Figure 8 shows the normalized performance of this application with one of various monitoring systems. Even when we executed the idle monitor, the

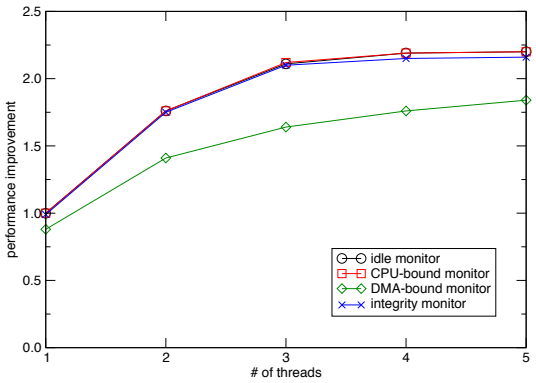

Figure 8. The performance of a DMA-bound application for various monitoring systems.

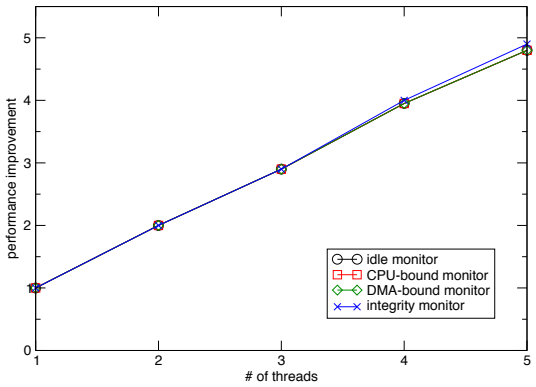

Figure 9. The performance of matrix multiplication for various monitoring systems.

DMA-bound application did not scale. Compared with the sequential execution, the application performance improved up to only 2.2 times even if the application used many threads. This is because the application consumed most of the available DMA bandwidth. Therefore, when we executed the DMA-bound monitor, the performance of the DMA-bound application degraded by 12 % to 22 %.

On the other hand, when the CPU-bound monitor was executed together, the application performance was almost the same as when the idle monitor was run. These monitoring systems did not compete with the DMA transfers performed by the application. Similarly, even if the application was executed with the integrity monitor, the performance was not nearly affected. Since the integrity monitor performed DMA transfers to obtain the kernel memory, it was possible to compete with the DMA transfers by the application. However, calculating the hash value was dominant in the integrity monitor, as analyzed in Section V-A. The integrity monitor consumed a small portion of the DMA bandwidth.

Third, we executed an application for matrix multiplication, which was provided by IBM. This application calculated the multiplication of two matrices using the specified number of threads, while the threads communicated with each other for synchronization. Figure 9 shows the application performance and the results were very similar to those in the CPU-bound application.

C. Impacts on Applications with Six Threads

We examined how the integrity monitor affected applications running with six threads, which are the same number of SPEs available to the users in PlayStation 3. Applications for Cell/B.E. are often customized for the fixed number of threads. The number is usually the same as that of available SPEs in the target machine. This means that the performance of applications using six threads is important while a monitoring system is running. Since the integrity monitor occupied one SPE in the isolation mode, applications could use only five SPEs for six threads. In this experiment, SPE Observer did not schedule the integrity monitor.

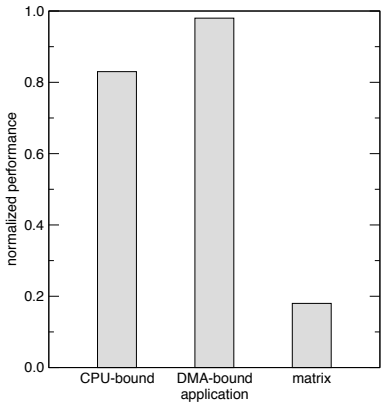

Figure 10. The performance degradation of applications with six threads.

We used three applications in the previous section. Figure 10 shows the performance when we executed each application using six threads with the integrity monitor. The shown performance is normalized by that when we executed each application without running the integrity monitor. The performance of the CPU-bound application degraded by 17 %. Since the application with six threads had to be run on only five SPEs, the SPE scheduler switched the thread contexts in a round-robin fashion. Therefore, the application performance became $\frac{5}{6}$ of that when the application could use six SPEs.

For the DMA-bound application, the application performance degraded only slightly. In other words, the execution time did not nearly change even when six threads were run on five SPEs. Since the integrity monitor on one SPE did not perform DMA transfers very much, five SPEs could consume most of the DMA bandwidth. This means that each SPE for the application could consume more DMA bandwidth than when the application used six SPEs. As a result, the performance of DMA transfers by each SPE

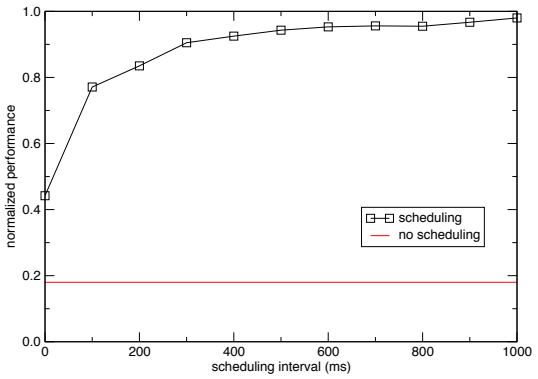

Figure 11. The performance improvement of matrix multiplication by scheduling the integrity monitor.

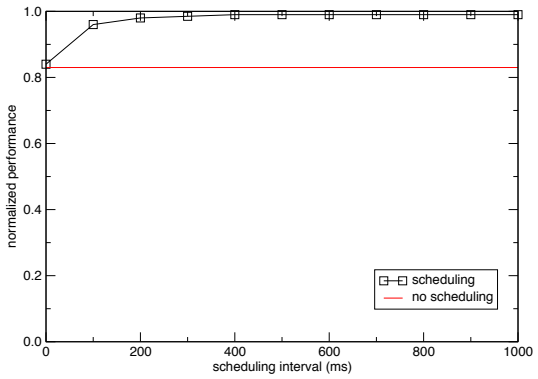

Figure 12. The performance improvement of a CPU-bound application by scheduling the integrity monitor.

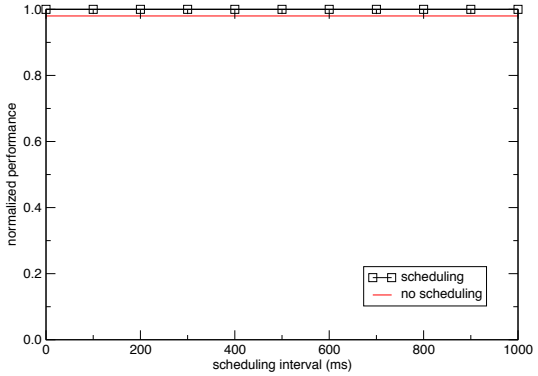

Figure 13. The performance improvement of a DMA-bound application by scheduling the integrity monitor.

improved to nearly $\frac{6}{5}$. This compensated the degradation by decreasing the number of available SPEs to $\frac{5}{6}$.

On the other hand, the performance of matrix multiplication degraded extraordinarily. This application synchronized between all the threads with mailboxes of SPEs whenever each thread performed a small amount of computation. If even one thread is delayed, the others have to wait for that. When six threads are executed on only five SPEs, five threads immediately complete their computation of a certain turn, but one thread is delayed. Since the SPE scheduler is not aware of the idleness of threads on SPEs, it simply performs context switches every 100 ms, which is the default timeslice. In other words, the delayed thread is not scheduled although there are idle SPEs. This is the reason of the performance degradation in this application.

D. Performance Improvement by Scheduling

To examine whether the performance of matrix multiplication can be improved by scheduling the integrity monitor on an SPE, we scheduled the integrity monitor at various intervals. Here, a scheduling interval is the time from when the security proxy receives the END message from the integrity monitor until it sends the START message. Figure 11 shows the changes of the normalized performance when we changed the scheduling interval. Even if the interval was short, the application performance improved dramatically. This reason is that the delayed thread was allocated to the SPE that had been used by the integrity monitor. The integrity monitor was re-scheduled immediately, but it was allocated to another SPE, which was probably idle, because our SPE scheduler selects a victim SPE in a round-robin fashion. When the interval is more than 200 ms, the performance degradation was less than 17 %, which is the performance gained by one SPE.

Next, we examined how scheduling the integrity monitor affected the performance of the CPU- and DMA-bound applications. While the performance of these applications did not degrade very much even without scheduling, scheduling incurs performance overheads due to context switches. Figure 12 and Figure 13 show the changes of the performance of the CPU- and DMA-bound applications, respectively, when we changed the scheduling interval. For the CPU-bound application, the performance gradually improved from 0.83 as the interval became longer. When the interval was 100 ms, the performance degradation was only 4 %. For the DMA-bound application, the performance degraded by 2 % at maximum when the interval was short, but no performance degradation occurred if the interval was longer than 100 ms.

E. Impacts on Monitoring Performance

We examined how much the integrity monitor on an SPE was affected by running applications when SPE Observer scheduled the integrity monitor. We executed the integrity monitor together with one of three applications with six threads. For comparison, we also executed the integrity monitor alone. In this experiment, SPE Observer scheduled the integrity monitor at the interval of 200 ms. We measured the time from when the security proxy started sending the START message until it received the END message.

Figure 14 shows the execution time of the integrity monitor for various applications. Not only CPU-bound but

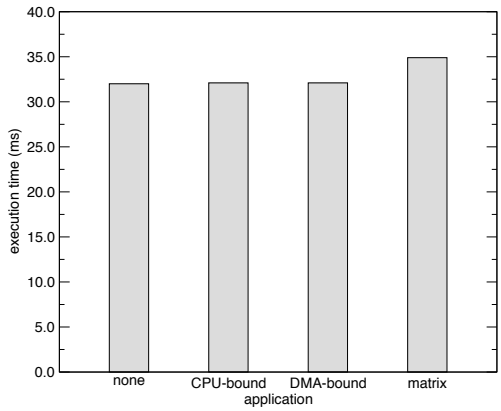

Figure 14. The execution time of the integrity monitor for various applications with scheduling.

also DMA-bound applications did not affect the performance of the integrity monitor. However, matrix multiplication degraded the performance by 9 %. This is probably because the SPE scheduler could not preempt an SPE for the integrity monitor when the relay process received the START message.

VI. Related Work

A. Hardware-assisted Secure Monitoring

Copilot [22] monitors the kernel memory by using a special PCI card inserted in a target host. The Copilot monitor on the PCI card obtains the contents of the physical memory by DMA and sends them to a remote host via the dedicated network. Then the remote host checks the integrity of the kernel text and jump tables. The attackers in the target host cannot compromise the Copilot monitor on the PCI card or the remote host. While Copilot needs special hardware, SPE Observer uses only general-purpose hardware.

HyperGuard [5] monitors the integrity of the hypervisor by using System Management Mode (SMM) provided in the commodity x86 processor families. A CPU enters SMM via System Management Interrupt (SMI) and executes a SMI handler. In SMM, the CPU can securely execute code in System Management RAM (SMRAM), which cannot be accessed in the normal mode called the protected mode. HyperGuard triggers SMI by timer interrupts and the SMI handler on SMRAM checks the memory of the hypervisor. HyperGuard can be applied to monitor the integrity of the operating system in a non-virtualized environment.

However, there are several drawbacks of using SMM. First, while a CPU runs in SMM, all the regular tasks including the operating system are suspended to maintain the integrity. In other words, all cores other than the one executing a SMI handler have to be frozen. SPE Observer inherently supports multicore and a monitoring system on an SPE in the isolation mode can run with the other tasks in parallel. Second, SMM is much slower than the protected mode. The isolation mode of SPEs does not suffer from such a performance penalty. Third, it is not easy to execute various monitoring systems in SMM because the SMI handler is a part of BIOS. SPE Observer allows executing arbitrary programs with correct signatures for monitoring systems.

HyperCheck [6] also uses SMM to monitor the physical memory, but it runs only a network device driver as the SMI handler. The driver makes the NIC to read the physical memory using DMA and send packets whose payloads are the memory contents. The remote host receives the packets and checks the integrity of the hypervisor. Since SMM is only used for setting up the NIC and the integrity checking is outsourced, the performance degradation due to SMM is small. In addition, HyperCheck can read and verify CPU registers. However, it is necessary to implement various device drivers running in SMM. In SPE Observer, monitoring systems do not depend on devices.

HyperSentry [7] allows a measurement agent inside the hypervisor to be securely executed using SMM although the hypervisor may have been compromised. The SMI handler is invoked via Intelligent Platform Management Interface, which is an out-of-band communication channel with a remote host. Then, the handler verifies the agent inside the hypervisor, disables interrupts, and runs the agent for collecting the detailed information on the hypervisor. Finally, the measurement output is attested by the remote host. In HyperSentry, not only the SMI handler but also the agent cannot run with the other tasks.

Flicker [8] is an infrastructure for executing security-sensitive code such as rootkit detectors, using the hardware support such as Intel TXT and AMD SVM. When such code needs to be executed, Flicker suspends the current execution environment including the operating system, securely executes the code using late launch, and resumes the previous execution environment. Late launch enables code execution without interferences by the attackers. However, it also stops all CPU cores other than the one used by the executed code. While the security-sensitive code is running, the other applications cannot be running. Moreover, researchers point out that it is possible to attack Intel TXT in SMM [23].

B. Applications of the Isolation Mode of SPEs

As a usage example of the isolation mode, the code verification service has been proposed [20]. This service enables an SPE to check the integrity of applications run on the PPE. When the operating system loads an application on the PPE, it requests an SPE for the verification. The service on the SPE obtains the memory used for the application by DMA transfers and verifies the signature of the application using the public key inside the SPE. Only if the verification succeeds, the operating system runs the application. The isolation mode maintains the integrity of the public key and protects the service itself from the outside attackers. This is similar to our SPE Observer, but the goal of SPE

Observer is detecting the modification to the long-running operating system periodically, not only at boot time. SPE Observer needs SPE scheduling to achieve this efficiently and a security proxy for reliable detection.

The isolation mode is also used for privacy-preserved data mining [24]. In volunteer computing like SETI@Home, the privacy of processing data should be protected from volunteers. The proposed system sends encrypted data to a volunteer's computer, decrypts them in isolated SPEs, and performs data mining. Since the secret key is shared only between the data owner and the application running on isolated SPEs, any volunteers cannot decrypt the received data. Furthermore, the decrypted data on isolated SPEs cannot be stolen from the PPE or the other SPEs.

VII. Conclusion

In this paper, we proposed SPE Observer, a framework for securely monitoring operating systems using SPEs in Cell/B.E. SPE Observer uses the isolation mode of SPEs to guarantee the integrity and confidentiality of monitoring systems. It also monitors the running status of monitoring systems on SPEs from the external security proxy. To mitigate the performance degradation due to occupying SPEs for monitoring, SPE Observer schedules monitoring systems. We have implemented SPE Observer in PlayStation 3 and conducted several experiments. The results show that monitoring systems on SPEs can check the integrity of the operating system and that the application performance can be improved by appropriate scheduling of monitoring systems.

One of our future work is to develop various monitoring systems for SPE Observer and examine its effectiveness. For example, it is necessary to check the integrity of kernel modules and find hidden processes in the kernel. Another is to develop middleware that runs in SPEs and enables the operating system in the PPE to detect idle SPEs.

Acknowledgment

This research was supported in part by JST, CREST.

References

[1] Stealth, "Adore-ng Rootkit," http://stealth.openwall.net/rootkits/.

[2] T. Miller, "Analysis of the KNARK Rootkit," SecurityFocus, 2001.

[3] sd and devik, "Linux On-the-fly Kernel Patching without LKM," Phrack issue 58, article 0x07, 2001.

[4] Trusted Computing Group, "TPM Main Specification," http://www.trustedcomputinggroup.org/, 2011.

[5] J. Rutkowska, R. Wojtczuk, and A. Tereshkin, "Xen 0wning Trilogy," Black Hat USA, 2008.

[6] J. Wang, A. Stavrou, and A. Ghosh, "HyperCheck: A Hardware-Assisted Integrity Monitor," in *Proc. Int. Symp. Recent Advances in Intrusion Detection*, 2010, pp. 158–177.

[7] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in *Proc. Conf. Computer and Communications Security*, 2010, pp. 38–49.

[8] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in *Proc. European Conf. Computer Systems*, 2008, pp. 315–328.

[9] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Proc. Symp. Network and Distributed System Security*, 2003, pp. 191–206.

[10] X. Jiang, X. Wang, and D. Xu, "Stealthy Malware Detection through VMM-based "Out-of-the-box" Semantic View Reconstruction," in *Proc. Conf. Computer and Communications Security*, 2007, pp. 128–138.

[11] O. Hofmann, A. Dunn, S. Kim, I. Roy, and E. Witchel, "Ensuring Operating System Kernel Integrity with OSck," in *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2011, pp. 279–290.

[12] CVE, "CVE-2008-2004," 2008.

[13] ——, "CVE-2008-4405," 2008.

[14] N. Murilo and K. Steding-Jessen, "chkrootkit," http://www.chkrootkit.org/.

[15] Trend Micro, Inc., "OSSEC," http://www.ossec.net/.

[16] McAfee, Inc., "Carbonite."

[17] T. Lawless, "StMichael," http://sourceforge.net/projects/stjude/.

[18] C. Kil, E. Sezer, A. Azab, P. Ning, and X. Zhang, "Remote Attestation to Dynamic System Properties: Towards Providing Complete System Integrity Evidence," in *Proc. Int. Conf. Dependable Systems and Networks*, 2009, pp. 115–124.

[19] SCEI, "Cell Broadband Engine Architecture Version 1.02," 2007.

[20] M. Murase, K. Shimizu, W. Plouffe, and M. Sakamoto, "Effective Implementation of the Cell Broadband Engine Isolation Loader," in *Proc. Computer and Communications Security Conf.*, 2009, pp. 303–313.

[21] Y. Kawamura, T. Yamazaki, T. Ishiwata, K. Horie, and H. Kyusojin, "Network Processing on an SPE Core in Cell Broadband Engine," in *Proc. Symp. High Performance Interconnects*, 2008, pp. 119–128.

[22] N. Petroni, Jr., T. Fraser, J. Molina, and W. Arbaugh, "Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor," in *Proc. Conf. USENIX Security Symp.*, 2004.

[23] R. Wojtczuk and J. Rutkowska, "Attacking Intel Trusted Execution Technology," Black Hat DC, 2009.

[24] H. Wang, H. Takizawa, and H. Kobayashi, "A Performance Study of Secure Data Mining on the Cell Processor," in *Proc. Int. Symp. Cluster Computing and the Grid*, 2008, pp. 633–638.