SSdetector: Secure and Manageable Host-based IDS with SGX and SMM

Yoshimichi Koga
Kyushu Institute of Technology
koga@ksl.ci.kyutech.ac.jp

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

***Abstract*—Host-based intrusion detection systems (HIDS) are used to monitor the internals of target systems. It is essentially difficult to execute HIDS securely inside target systems. For example, it is not guaranteed that HIDS can obtain correct information from compromised systems. If HIDS is tampered with by intruders, it would be easily disabled. So far, various techniques have been proposed to securely execute HIDS using the security features of processors, e.g., System Management Mode (SMM) and SGX in Intel processors. However, strongly secure HIDS sacrifices its manageability, whereas manageable HIDS is less secure. In practice, it is important to achieve not only the security but also the manageability of HIDS. This paper proposes *SSdetector* for achieving both security and manageability by combining SGX and SMM. SSdetector securely runs HIDS inside an SGX enclave, which is a protected region inside an SGX application. Since HIDS is developed as an SGX application, the management of HIDS is easier. To securely obtain system information in memory, in-enclave HIDS invokes the *SMM monitor* running in an isolated execution environment created by BIOS. SSdetector protects information passed between in-enclave HIDS and the SMM monitor by encryption and integrity checking. We have implemented SSdetector in UEFI BIOS and examined the performance of HIDS collecting system information necessary for the proc filesystem.**

***Index Terms*—Intel SGX, system management mode, host-based IDS, BIOS**

1. Introduction

As systems become huge and complex, attacks against systems are increasing. Since it is difficult to remove all the vulnerabilities from systems, intrusion detection systems (IDS) are required as countermeasures against attacks. In particular, host-based IDS (HIDS) monitors the internal states of a target system and notifies the symptoms of attacks to administrators. However, it is not easy to securely run HIDS because HIDS has to essentially run *inside* a target system. For example, it is not guaranteed that HIDS can obtain correct information from the target system after the system is compromised. If HIDS is tampered with by intruders inside the system, it cannot detect attacks anymore.

Various techniques have been proposed to securely run HIDS using the security features of processors. However, all of them have issues in either security or manageability. For example, HIDS using System Management Mode (SMM) in Intel processors has been proposed [1]–[4]. SMM-based HIDS can securely monitor system information in memory using an isolated execution environment created by BIOS. However, it is difficult or insecure to update HIDS and deploy new HIDS. Recently, HIDS using Intel SGX has been proposed [5], [6]. SGX-based HIDS securely runs inside a trusted execution environment (TEE) called an *enclave*. It is more manageable than the SMM-based one, but it needs to rely on the hypervisor to obtain memory data. This is not secure enough because many vulnerabilities in hypervisors have been reported [7].

To address these issues, this paper proposes *SSdetector* for achieving both the security and manageability of HIDS by combining SGX and SMM. SSdetector runs HIDS inside an SGX enclave to prevent HIDS from being eavesdropped on and tampered with. Since HIDS can be developed as an SGX application running on the operating system (OS) of the target system, the management of HIDS becomes easier. Unfortunately, in-enclave HIDS cannot directly obtain the memory data of the target system. In SSdetector, in-enclave HIDS invokes the *SMM monitor* running in BIOS and securely obtains memory data. Since the SMM monitor provides only common and minimum functions, it does not need to be updated. SSdetector prevents eavesdropping and tampering of information passed between in-enclave HIDS and the SMM monitor by encryption and integrity checking.

We have implemented SSdetector using Intel SGX SDK [8] and TianoCore [9], which is an open-source implementation of UEFI BIOS. In SSdetector, the SMM monitor is invoked by triggering a system management interrupt (SMI) and obtains requested memory data by translating a virtual address of OS data into a physical one. We have developed in-enclave HIDS that collects system information from the OS memory and generates the contents of pseudo files provided by the proc filesystem in Linux, using LLView [10]. We conducted several experiments using SSdetector and confirmed that our HIDS could collect system information correctly. In addition, we measured the time needed to collect system information and examined the performance overhead caused by SGX, SMM, encryption, and integrity checking.

The organization of this paper is as follows. Section 2 describes the secure execution of HIDS using general-

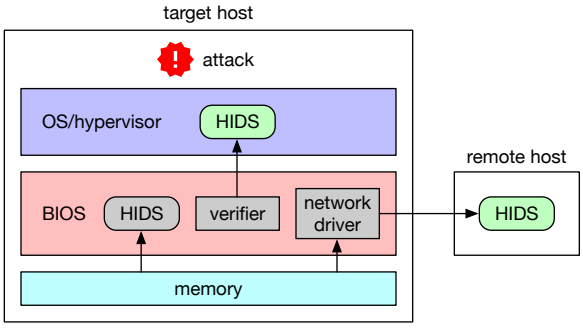


Figure 1: Three examples of SMM-based HIDS.

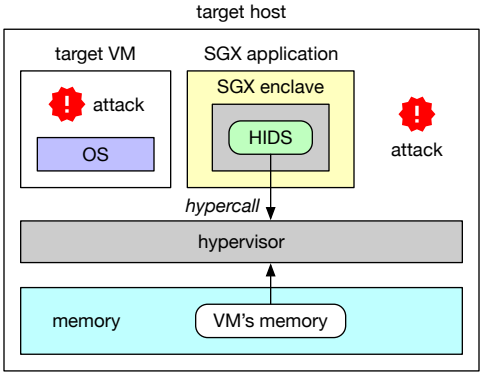


Figure 2: An example of SGX-based HIDS.

purpose processors. Section 3 proposes SSdetector for achieving both the security and manageability of HIDS by combining SGX and SMM. Section 4 explains the implementation of SSdetector and Section 5 shows our experimental results. Section 6 describes related work and Section 7 concludes this paper.

2. Secure Execution of HIDS

The requirements for securely running HIDS are (1) HIDS can detect attacks without relying on the functions of the target system, and (2) HIDS is not eavesdropped on or tampered with even if attackers intrude into the target system. When HIDS monitors the target system using the functions of the target system, it is not guaranteed that HIDS can obtain correct information from that system after the system is compromised. If HIDS is tampered with by intruders, it cannot detect attacks after that. If attackers can eavesdrop on system information that HIDS obtains, they could steal part of the sensitive information included in that via HIDS.

So far, various techniques have been proposed to satisfy these requirements using the security features of general-purpose processors. However, they have several issues in terms of not only the security but also the manageability of HIDS. For example, HIDS that directly monitors the memory of the target system using SMM has been proposed [1]–[4]. SMM is one of the processor modes provided by Intel processors. It is available only in BIOS and provides an isolated execution environment, which cannot be accessed by any systems software such as the OS. The code running in SMM is located in the dedicated memory region called system management RAM (SMRAM) and can be accessed only in SMM. The SMM code is invoked by triggering a software interrupt called an SMI. For example, an SMI can be triggered by writing data to specific I/O ports.

HyperGuard [1] and SPECTRE [4] monitor the integrity of the hypervisor code and the internal states of the operating system in SMM, respectively, as depicted in Fig. 1. HIDS cannot be attacked by intruders inside the target system. However, it is not easy to update HIDS for supporting new versions of OSes and deploy new HIDS because the entire HIDS is embedded into BIOS. HyperSentry [3] securely runs HIDS inside the hypervisor after the handler in SMM verifies it and disables interrupts. It is not necessary to modify BIOS for updating HIDS, but the hypervisor needs to be updated and rebooted. HyperCheck [2] runs only a network driver in SMM and sends the memory data of the target system to HIDS running at a remote host. This method makes the management of HIDS easier, but HIDS is less secure because it is not protected at a remote host.

Recently, HIDS that directly monitors the memory of virtual machines (VMs) using SGX has been proposed [5], [6]. SGX is a security feature of Intel processors and enables an SGX application to create a trusted execution environment called an enclave inside it. The code in an enclave can be executed securely due to the protection of a processor. Since the digital signature of the code is verified by SGX at the launch time, any code modified by attackers cannot be executed in enclaves. In addition, SGX guarantees the integrity of the enclave memory, so that attackers cannot tamper with the code running in the enclave. Attackers cannot also eavesdrop on data inside the enclave because the enclave memory is encrypted.

SGmonitor [5] runs HIDS in an SGX enclave and monitors the internal states of a target VM by obtaining memory data from the VM, as illustrated in Fig. 2. In-enclave HIDS invokes the trusted hypervisor running under the VM using a hypercall and securely obtains the memory data of the VM. This is because enclaves cannot directly access the memory of VMs. Intruders inside and outside the VM cannot eavesdrop on or tamper with in-enclave HIDS. However, if attackers can compromise the hypervisor, they can interfere with the behavior of in-enclave HIDS. Since the hypervisor is complex software, it has many vulnerabilities [7]. In addition, SGmonitor can monitor only virtualized systems because it needs the hypervisor to securely obtain memory data.

Table 1 compares security and manageability among previous SGX- and SMM-based HIDS. To summarize, strongly secure HIDS such as HyperGuard, SPECTRE, and HyperSentry sacrifices its manageability. In contrast, manageable HIDS such as HyperCheck and SGmonitor tends to be less secure. As such, all of them do not satisfy both security and manageability. The security of HIDS is mandatory, whereas

TABLE 1: Comparison among SGX- and SMM-based HIDS.

	security	manageability
SPECTRE [4]	✓	
HyperSentry [3]	✓	
HyperCheck [2]		✓
SGmonitor [5]		✓

its manageability is also important in practice.

3. SSdetector

This paper proposes SSdetector for achieving both the security and manageability of HIDS by combining SGX and SMM. We trust processors and assume that SGX and SMM provided by processors do not have any hardware vulnerabilities. In contrast, we do not trust any software except for code running in SGX enclaves and SMM. We configure BIOS so that it can be updated only when administrators can access a physical machine. Then, we focus on remote attacks against a target system including the OS. Also, we assume that the remote attestation server for SGX is trusted and well-protected.

3.1. How to Combine SGX and SMM

There are two approaches to combining SGX and SMM for secure and manageable HIDS. One approach is to apply SGX to SMM-based HIDS. It is impossible to apply SGX to HyperGuard, SPECTRE, and HyperSentry because an SGX enclave cannot be created inside BIOS, the hypervisor, and the OS. The only possible solution is to run HIDS in an SGX enclave at a remote host in HyperCheck. This makes HIDS more secure and keeps its manageability. However, various network drivers still need to be implemented in SMM. This means that the manageability of the SMM part of HIDS is low. In addition, network communication imposes a large overhead when HIDS obtains memory data on demand like SPECTRE. Bulk transfer of memory data can reduce this overhead, but the enclave page cache (EPC) used for the SGX enclaves is a limited resource. If the EPC is used up by bulk transfer, the performance of the SGX enclaves degrades largely.

Therefore, SSdetector adopts the other approach, which applies SMM to SGX-based HIDS. SSdetector runs an SGX application for HIDS inside the target system, as illustrated in Fig. 3. It securely executes HIDS in an SGX enclave created in the application. Like previous SGX-based HIDS, in-enclave HIDS cannot directly access the memory data of the target system because memory is managed by the underlying OS. Therefore, SSdetector invokes the SMM monitor in BIOS using SMIs. The SMM monitor securely runs in SMM and just obtains requested data in memory. Since SSdetector does not rely on the hypervisor, it can support non-virtualized systems. Compared with the hypervisor, the SMM monitor is much smaller and provides a narrower interface, so that it is more difficult to attack.

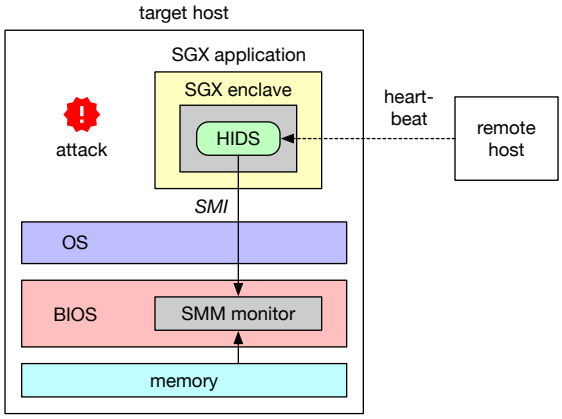


Figure 3: The system architecture of SSdetector.

3.2. Secure and Manageable HIDS

An SGX enclave prevents HIDS from being eavesdropped on and tampered with by using processors' capabilities. SGX encrypts enclave memory and checks its integrity. Similarly, SMM prevents the SMM monitor from being attacked by running it in an isolated execution environment. SMM prevents attackers from accessing SMRAM used by the SMM code. In addition, SSdetector protects information passed between in-enclave HIDS and the SMM monitor. The SMM monitor encrypts obtained memory data to prevent attackers in the middle from eavesdropping on the data. In-enclave HIDS detects that attackers in the middle tamper with obtained memory data or return crafted memory data by checking the hash value of the data and a secret key. The key used for the encryption and integrity checking is generated by in-enclave HIDS and is securely shared with the SMM monitor using public key encryption.

Using remote attestation, SSdetector prevents malicious applications from obtaining memory data by invoking the SMM monitor using SMIs. When an SGX enclave is launched, it is remotely attested to by the trusted attestation server. If in-enclave HIDS is legitimate, it sends the generated encryption key to the server via the secure communication channel established by remote attestation. The key is digitally signed using the private key of the server. When in-enclave HIDS passes the encryption key and its signature to the SMM monitor, the SMM monitor verifies the signature using the public key of the server. If the verification fails, the SMM monitor does not accept the encryption key or return memory data.

SGX cannot prevent attackers from stopping SGX applications because they are normal processes running on top of the OS. If attackers can obtain administrative privileges or compromise the OS, they can easily stop HIDS with SGX applications. To detect this type of attack, SSdetector confirms the correct execution of HIDS by sending heartbeats from a remote host. If in-enclave HIDS does not respond to a heartbeat, the remote host can notice that HIDS is stopped by attackers. Since the heartbeats are encrypted by

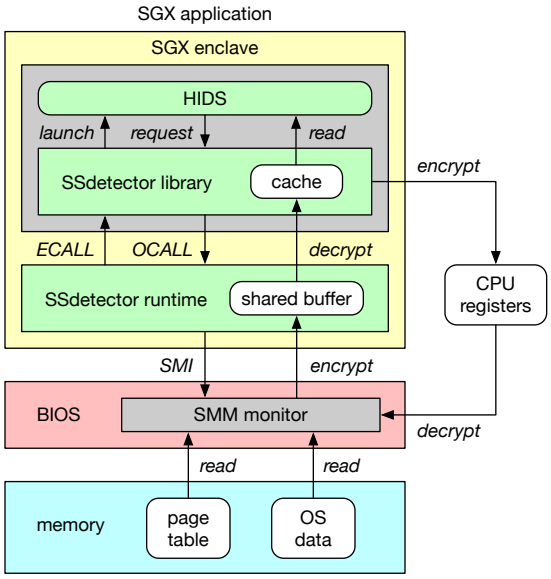


Figure 4: The detailed architecture of SSdetector.

the remote host, attackers cannot respond to the heartbeats correctly. In addition, attackers can mount denial of service (DoS) attacks by discarding requests from HIDS or responses from the SMM monitor. This type of attack is detectable by a request timeout.

In SSdetector, the management of HIDS is much easier than that of previous SMM-based HIDS. To update HIDS, e.g., according to an OS upgrade, SSdetector just restarts an SGX application for the HIDS. When deploying new HIDS, SSdetector starts a new SGX application for it. Since an SGX application runs on top of the OS as a normal application, restarting it can easily be done without rebooting the target host. The SMM monitor does not need to be updated because it is common for all the HIDS. It just obtains requested memory data and returns it to in-enclave HIDS. Even if HIDS runs in other hosts with different hardware, the SMM monitor does not need to be modified because it does not depend on specific hardware such as NICs.

4. Implementation

An SGX application for HIDS consists of an enclave and the SSdetector runtime, as illustrated in Fig. 4. In the enclave, HIDS and the SSdetector library run. The library provides the interface to the runtime. The runtime mediates between in-enclave HIDS and the SMM monitor. Since it is not protected by SGX, we assume that it can be attacked by intruders inside the target system. We used Intel SGX SDK 2.13.3 [8] to implement an SGX application for HIDS. We have implemented the SMM monitor in TianoCore, which is open-source UEFI BIOS, using EDK II [9].

4.1. SSdetector Library and Runtime

When SSdetector starts an SGX application for HIDS, the application runs the SSdetector runtime in it. Then, the runtime creates an enclave and loads an enclave binary for HIDS. The binary contains HIDS and the SSdetector library. To launch in-enclave HIDS, the runtime securely invokes the pre-registered function of the SSdetector library using an SGX mechanism called an enclave call (ECALL). At this time, the runtime allocates a buffer for sharing information and passes its address to the library in the enclave. The enclave can access the memory outside it, although the runtime cannot access enclave memory for security. When the ECALL function is invoked in the library, the library starts to run the function of HIDS.

In-enclave HIDS analyzes OS data in memory to monitor the target system. When it requires memory data for OS data, it invokes the SSdetector library. Then, the library securely invokes the SSdetector runtime running outside the enclave using an SGX mechanism called an outside call (OCALL). At this time, the library passes the virtual addresses of the requested OS data and the shared buffer to the runtime. To protect the address information from attackers, it encrypts these virtual addresses. It applies AES encryption to a 16-byte block consisting of two 64-bit addresses. Then, it splits the encrypted data into two 8-byte data and stores them in two 64-bit CPU registers.

The OCALL function in the SSdetector runtime triggers an SMI to invoke the SMM monitor in BIOS. It triggers an SMI by writing data to the I/O port of 0xb2. To permit this write, the SGX application runs with administrative privileges and configures the permission using the ioperm system call in Linux. The reason why in-enclave HIDS invokes the SMM monitor via the runtime is that the code in an enclave cannot trigger SMIs. Since the runtime does not change the values of the two CPU registers where encrypted virtual addresses are stored, the values stored by the library are passed to the SMM monitor as they are.

When the SMM monitor is completed, encrypted memory data and the hash value of the unencrypted memory data and the encryption key are stored in the passed shared buffer. When the OCALL function in the SSdetector runtime finishes, the SSdetector library also receives the data via this buffer. Then, it decrypts the memory data and recalculates its hash value. If the recalculated hash value is different from the received one, it can detect tampering with either the encrypted memory data or the received hash value. Otherwise, in-enclave HIDS uses the received memory data to access OS data in the target system.

The library caches the decrypted memory data inside the enclave to reduce the number of invocations of the SMM monitor. If memory data is kept in the cache, the SSdetector library can return them immediately without invoking an OCALL to the SSdetector runtime. SSdetector limits the data size preserved in the cache. One reason is that SGX can use the EPC of only a limited size. When the EPC is used up, the memory access performance of the code in an enclave largely degrades due to paging. The other reason is that HIDS cannot timely detect attacks from stale data. Therefore, the library evicts data in a least recently used (LRU) fashion and removes expired data periodically.

4.2. SMM Monitor

To enable the SMM monitor to access the entire memory of the target system, SSdetector uses UEFI BIOS instead of traditional BIOS. UEFI BIOS runs in 64-bit mode and can access memory that exceeds 4 GB. However, TianoCore limits the memory region that can be accessed in SMM. SSdetector removes this limitation so that the SMM monitor can obtain the memory data of the target system. Specifically, TianoCore limits memory access by creating the page tables for SMM. SSdetector extends the page tables so that the SMM monitor accesses the entire memory. It sets up the page tables so that read-only access is permitted to the extended memory regions. Therefore, attackers cannot compromise the target system even if the SMM code is compromised.

When the SSdetector runtime triggers an SMI, the SMM monitor in UEFI BIOS concatenates two 8-byte data stored in the two 64-bit CPU registers. Then, it decrypts the combined 16-byte block using AES. From the decrypted block, it extracts two 64-bit virtual addresses of OS data and the shared buffer. Next, the SMM monitor translates the decrypted virtual address of OS data into a physical one and obtains the corresponding memory data. This is because the SMM monitor needs to access the OS memory using the physical addresses of OS data. In contrast, HIDS analyzes the memory of the target system using the virtual addresses of OS data. If HIDS itself has to translate a virtual address into a physical one, it would depend on the function of the underlying OS, which can be compromised by intruders.

Therefore, the SMM monitor securely performs address translation. It obtains the value of the CR3 register from the currently used processor and specifies the page tables in the OS memory. Using the page tables, it translates the virtual address of OS data into a physical one. Similarly, it translates the virtual address of the shared buffer into a physical one. Then, it reads the corresponding memory data using the physical address of the OS data. In addition, it calculates the hash value of the memory data and the encryption key and stores it in the shared buffer. Finally, it encrypts the memory data and stores it in the shared buffer.

4.3. Secure Key Sharing

The SSdetector library in the enclave generates an AES key used for encrypting memory data. It encrypts the generated key using the RSA public key for the SMM monitor and passes it to the SMM monitor by triggering an SMI. To distinguish between registering the AES key and obtaining memory data, the library writes a different value to the same I/O port. The SMM monitor decrypts the received AES key using its RSA private key. Since only the SMM monitor can access this private key, attackers in the target system cannot steal it. Attackers can access the unencrypted binary file of an SGX application for HIDS, but only the RSA public key is stored in that file.

TABLE 2: The nine pseudo files of the proc filesystem generated by our HIDS.

pseudo file	contents
stat	kernel and system statistics
meminfo	statistics about memory usage
uptime	uptime of the system
tty/drivers	list of tty drivers
sys/kernel/osrelease	Linux kernel version
sys/kernel/pid_max	maximum process ID
[pid]/stat	status information on the process
[pid]/status	human-readable [pid]/stat
[pid]/auxv	information passed to the process

4.4. HIDS Developed with LLView

Using the LLView framework [10], we have developed in-enclave HIDS that analyzes OS data in memory. LLView enables developers to write HIDS code that runs outside the target OS, using the source code of the Linux kernel. Developers can use data structures, global variables, inline functions, and macros included in the Linux kernel headers. LLView compiles HIDS code and generates LLVM intermediate representation called bitcode. Then, it transforms the bitcode and inserts a call to the function for obtaining necessary memory data before all the load instructions. In SSdetector, the function invokes an OCALL and obtains memory data from the SMM monitor by triggering an SMI.

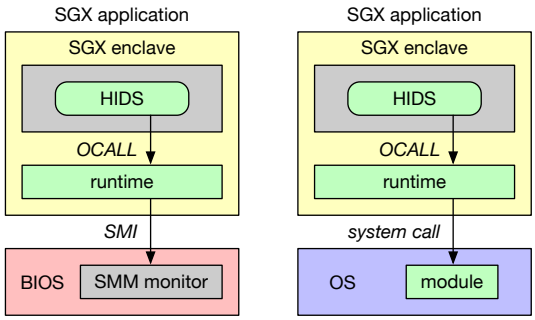
The developed HIDS collects system information necessary for the proc filesystem in Linux. The proc filesystem provides pseudo files and generates their contents dynamically. For example, it provides information about the system, processes, networks, and devices. This filesystem is often used to monitor the target system in application-level HIDS running on top of the OS. In the current implementation, our HIDS generates the contents of nine pseudo files shown in Table 2. These files are actually used by chkrootkit [11] to detect various attacks.

5. Experiments

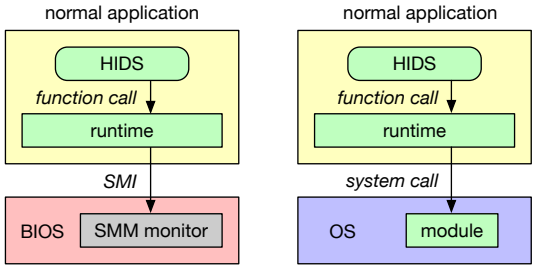
To show the effectiveness of SSdetector, we conducted several experiments using the HIDS we have developed, which is described in Section 4.4. In this experiment, we ran SSdetector in a VM because it is difficult to change the BIOS of a physical machine (PM). To create a VM supporting SGX, we used KVM SGX v5.6.0-rc5-r2 and QEMU SGX v4.0.0-r1, which supported SGX virtualization. We assigned one virtual CPU and 6 GB of memory to this VM and ran Linux 5.8.18 in the VM. As UEFI BIOS, this VM used OVMF, which is a port of TianoCore to QEMU. To run this VM, we used a PM with Intel Core i7-9700 and 16 GB of memory. This PM used AMI UEFI BIOS F.14. To develop HIDS using LLView, we used LLVM 12.0.0.

5.1. Behavior of HIDS in SSdetector

We ran our HIDS in an SGX enclave using SSdetector. To examine the correctness of collected system information,



(a) SSdetector (b) SGX only



(c) SMM only (d) no SGX/SMM

Figure 5: The four combinations of SGX and SMM.

the HIDS showed the generated contents of the pseudo files only in this experiment. For comparison, we obtained the contents of the corresponding pseudo files directly from the proc filesystem. As a result, the contests were the same between our HIDS and the original proc filesystem except for slight differences. We confirmed that our HIDS could collect correct system information from the memory of the target system via the SMM monitor.

5.2. Overhead of Using SGX and SMM

We measured the time needed to collect all the necessary system information in our HIDS. Compared with SSdetector using both SGX and SMM (Fig. 5(a)), we conducted this experiment for (1) the system using SGX only (Fig. 5(b)), (2) the one using SMM only (Fig. 5(c)), and (3) the one without SGX or SMM (Fig. 5(d)). For the systems without using SGX, the HIDS directly invoked the SSdetector runtime, instead of invoking OCALLs. For the systems without using SMM, the HIDS obtained memory data by accessing the device driver installed into the OS, instead of triggering SMIs. Note that these systems without using SGX and/or SMM are less secure than SSdetector.

Fig. 6 shows the time for collecting system information. To focus on the overhead of using SGX and SMM, we disabled encryption or integrity checking for data protection in this experiment. Compared with no SGX/SMM, SSdetector took 5.1x longer to collect system information. The variance in SSdetector was quite large due to using SGX. The SGX performance was unstable in a VM, although it was stable in a PM. This is probably because of the implementation of SGX virtualization.

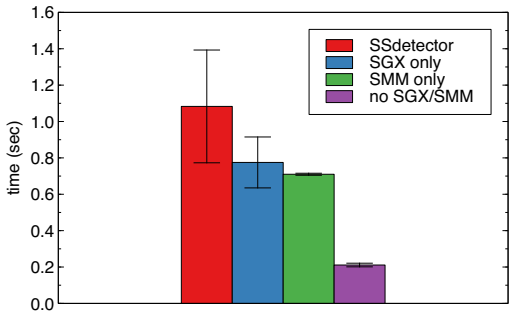
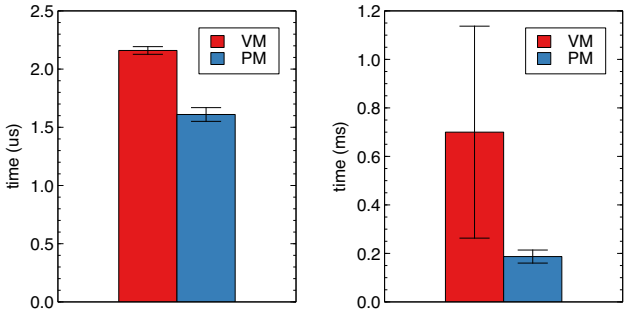


Figure 6: The collection time of system information.



(a) OCALL (b) SMI

Figure 7: The differences in the execution time between a VM and a PM.

Comparing SGX only and no SGX/SMM, it is estimated that the overhead due to using SGX is 564 ms. This occupies 52% of the time taken in SSdetector. The breakdown of this overhead is 1148 invocations of OCALLs and the execution of the HIDS in the enclave. In contrast, it is estimated that the overhead due to using SMM is approximately 499 ms, comparing SMM only and no SGX/SMM. This occupies 46% of the time taken in SSdetector. The breakdown of this overhead is 1148 invocations of SMIs and the execution of the SMM monitor in BIOS. From these results, both SGX and SMM degrade the performance of HIDS almost equally.

5.3. Overhead Estimation in a PM

The above results can be different in a PM because SGX and SMM are virtualized in a VM. For SGX, OCALLs can be affected by SGX virtualization. To examine the differences between a VM and a PM, we measured the execution time of an OCALL in both environments. In this experiment, we did nothing in the invoked OCALL function. As shown in Fig. 7(a), the execution time of an OCALL was only 0.54 μs faster in a PM. Since our HIDS needs 1148 OCALL invocations to collect necessary system information, it is estimated that the overhead could be reduced by 0.62 ms. Compared with the execution time of the HIDS, this reduction is negligible.

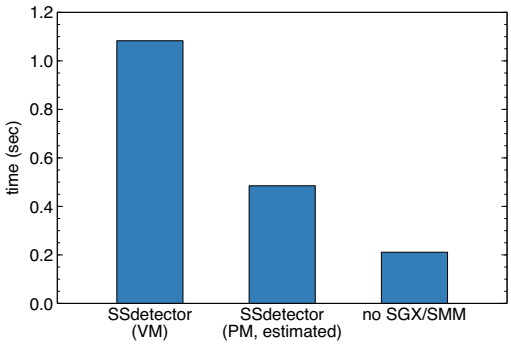


Figure 8: The estimated collection time of system information in a PM.

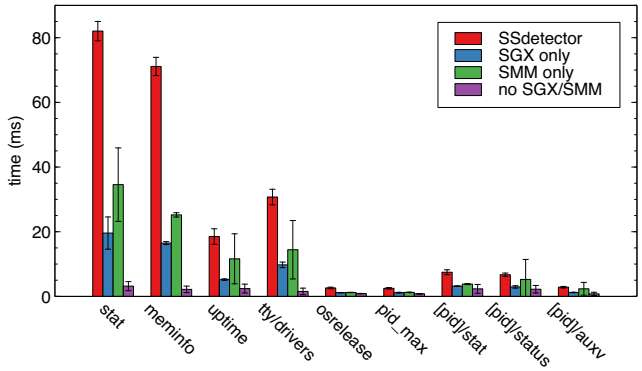


Figure 9: The collection time of system information for each pseudo file.

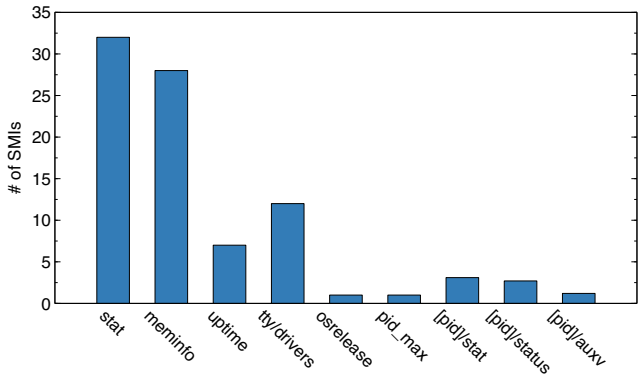


Figure 10: The number of triggered SMIs.

For SMM, one of the largest factors that cause performance differences is SMIs. We measured the execution time of an SMI in a VM and a PM. In this experiment, we did nothing in the invoked SMM code. As shown in Fig. 7(b), the execution time of an SMI in a PM was 0.52 ms faster than in a VM. For a total of 1148 SMIs, it is estimated that the overhead could be reduced by 597 ms. This reduction is 84% of the time for SMM only and occupies 55% of the time taken in SSdetector. As a result, SSdetector would take only 2.3x longer than the system using no SGX/SMM to collect system information, as shown in Fig. 8. Another large factor is the execution performance in SMM. Since the SMM execution is slower than the normal one [3] but does not suffer from virtualization overhead in a PM, the execution time of the SMM monitor would change in a PM. This investigation is our future work because we could not execute the SMM monitor in a PM.

5.4. Detailed Overhead of SGX and SMM

To examine the overhead of using SGX and SMM in further detail, we measured the time needed to individually generate the contents of each pseudo file. Fig. 9 shows the collection time of system information for each pseudo file. The pseudo files starting from [pid] mean the system information corresponding to the process whose ID is [pid]. The shown collection time is the average of 286 processes running in the target system. Note that the sum of all the collection times is larger than the total collection time shown in Section 5.2. This is because obtained memory data was cached inside the enclave when the HIDS generated all the pseudo files consecutively.

Fig. 10 shows the number of SMIs triggered while our HIDS generated the contents of each pseudo file. For the pseudo files starting from [pid], the average time is shown for each process. The sums of the numbers of triggered SMIs for [pid]/stat, [pid]/status, and [pid]/auxv are 888, 771, and 330, respectively. Compared with the results of Fig. 9, it is shown that the collection time for each pseudo file is proportional to the number of triggered SMIs.

As opposed to the results in Fig. 6, the collection time for each pseudo file in SMM only was 5-123% longer than in SGX only. When the HIDS generated all the pseudo files, the time in SMM only was rather 9% shorter. This might be related to the instability of SGX virtualization. In addition, the overhead of SSdetector was up to 33x larger than the system using no SGX/SMM, depending on pseudo files. To generate all the necessary pseudo files, [pid]/stat and [pid]/status are dominant because there are many processes. For these pseudo files, SSdetector was only 3x slower.

5.5. Overhead of Data Protection

To examine the overhead of encryption and integrity checking for data protection, we compared the time for collecting system information for all the pseudo files with or without data protection. Compared with SSdetector with both encryption and integrity checking, we measured the time in (1) the system with encryption only, (2) the one with integrity checking only, and (3) the one with no protection. SSdetector used AES-CBC with a 128-bit key for encryption and SHA-256 for integrity checking. The reason why we used AES-CBC is that the Crypto package in EDK II supported only that as AES modes of operation.

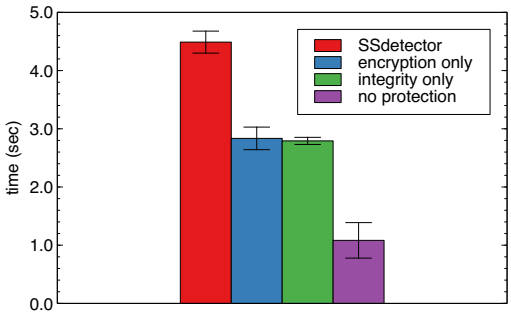
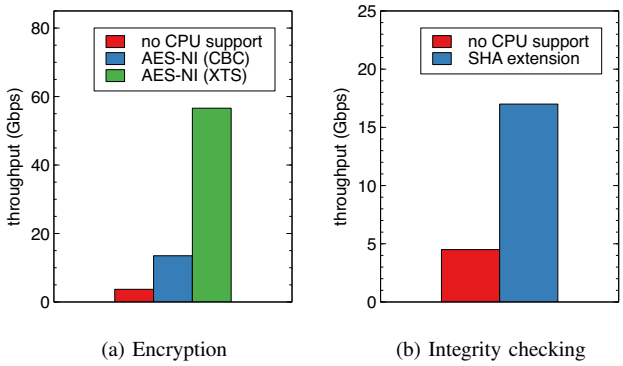


Figure 11: The overhead of data protection.



(a) Encryption (b) Integrity checking

Figure 12: The performance enhancement of data protection by CPU support.

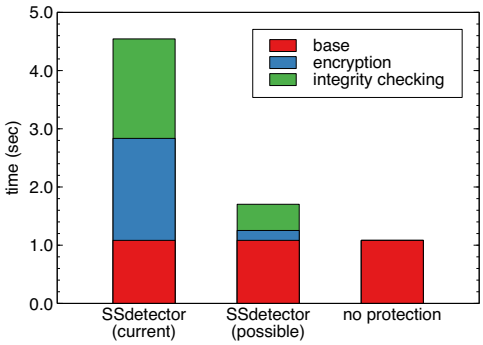


Figure 13: The possible reduction of the overhead of data protection by full CPU support.

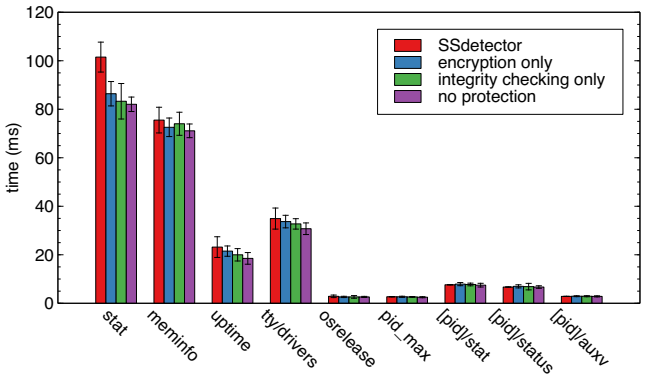


Figure 14: The overhead of data protection in collecting system information for each pseudo file.

Fig. 11 shows the collection time in these four systems. The overhead due to encryption was 1.75 seconds, which occupies 39% of the time taken in SSdetector. Similarly, the overhead due to integrity checking was 1.71 seconds, which occupies 38% of the time taken in SSdetector. From these results, it is shown that the time for data protection occupies 77% of the collection time in SSdetector. This is 4.2x longer than the collection time for no data protection.

This overhead of data protection could be reduced by using existing CPU support. Currently, SSdetector uses only the AES-NI instruction set in an SGX enclave. The SMM monitor does not use hardware encryption support. This is because the implementation of the Crypto package in EDK II did not use AES-NI. According to our experiment, AES-NI improved the performance of AES-CBC with a 128-bit key by 3.6x, as shown in Fig. 12(a). Furthermore, if the SMM monitor uses AES-XTS with a 128-bit key, the encryption performance could be further improved by 4.2x. Therefore, the encryption overhead could be reduced to 0.17 seconds.

Also, we measured the performance of SHA-256 in another PC with Intel Core i7-12700, which supports the SHA Extensions [12]. Since the SHA Extensions improved the performance by 3.8x, as shown in Fig. 12(b), the overhead of integrity checking could be reduced to 0.45 seconds. As a result, the collection time in SSdetector could be reduced to 2.29 seconds by fully utilizing CPU support, as shown in Fig. 13. This is only 1.6x longer than the system with no data protection.

To examine the overhead due to data protection in detail, we measured the generation time of the contents of each pseudo file with or without data protection. Fig. 14 shows the collection time for each pseudo file. The overhead due to data protection was 23% at maximum, compared with the system with no protection. This overhead was 77% and much larger when SSdetector collected all the system information, as shown in Fig. 11. One of the possible reasons is that it takes longer to collect a small amount of system information even without data protection. If this is the case, the overhead due to data protection is relatively small.

6. Related Work

SMM-based HIDS. HyperGuard [1] checks the integrity of the hypervisor by obtaining its code from memory. It runs the entire HIDS in BIOS, so that BIOS has to be updated whenever HIDS is updated. Updating BIOS requires the reboot of the target system. SPECTRE [4] triggers SMIs using a hardware timer and analyzes the memory

of the hypervisor, the OS, and applications using semantic information. Like these systems, running the entire HIDS in SMM largely affects the performance of both HIDS and the target system because SMM is slow and stops the entire target system.

HyperSentry [3] verifies a management agent in SMM and runs it inside the hypervisor. Therefore, the hypervisor needs to be rebooted whenever a management agent is updated. To securely run the agent, it disables interrupts and stops all but one CPU core. Therefore, the target system completely stops for a longer time as the time taken in HIDS increases. In SSdetector, the target system stops only while the SMM monitor obtains memory data.

HyperCheck [2] just transfers memory data to a remote host in SMM, whereas HIDS at the remote host monitors received memory data. This can suppress performance degradation due to SMM, but the security of the HIDS is not guaranteed. In addition, it is necessary to develop a network driver running in SMM per NIC. SSdetector needs to develop only SGX applications for HIDS. The SMM monitor does not need to be updated because it is common for HIDS.

SGX-based IDS. SGmonitor [5] runs HIDS in an SGX enclave and obtains the memory data of VMs via the hypervisor. It can also monitor the virtual disks of VMs by running filesystems in an enclave. SCwatcher [6] enables existing HIDS to run in an enclave using SCONE [13] or Occlum [14], which is an execution environment for SGX. It provides the proc filesystem in an enclave so that HIDS can obtain the system information in a VM using the existing interface. These systems need to trust a relatively large hypervisor and limit the target to virtualized systems. In contrast, SSdetector trusts only small BIOS and can monitor non-virtualized systems.

Not only HIDS but also network-based IDS (NIDS) using SGX has been proposed. S-NFV [15] protects Snort by storing only part of its internal states in an enclave. The states can be accessed only by the code running in the same enclave, which is invoked by secure API from the outside of the enclave. This can protect information on network flows. SEC-IDS [16] can run Snort with almost no modification in an enclave. To run existing Snort in an enclave, it uses the Graphene-SGX library OS [17]. It achieves almost the same performance as Snort without SGX by efficiently obtaining network packets using DPDK. However, in-enclave Snort could not detect attacks correctly if packets are tampered with while the Snort obtains them. In SSdetector, attackers cannot tamper with memory data while HIDS obtains it, thanks to SMM and data protection.

Systems Using Both SGX and SMM. SSdetector is the first system that combines SGX and SMM for HIDS, but several systems using SGX and SMM have been proposed for other purposes. Scotch [18] provides accurate and transparent resource accounting for clouds. It tracks the resources consumed by each VM in SMM whenever a context switch and an I/O interrupt occurs. Then, it relays the accounting information to the SGX enclave to securely analyze it later.

Aurora [19] enables the code in an SGX enclave to securely use devices such as clocks and NICs. When an enclave accesses such devices, it invokes the SMM code by triggering an SMI. The SMM code runs a device driver to access the device. The enclave and the SMM code exchange encrypted data via shared memory. Thus, a secure path is established between them. In SSdetector, the SMM code accesses system memory instead of devices.

KShot [20] enables kernel patches to be lively applied to the OS kernel without trusting the OS or the patch system. It securely downloads kernel patches in an enclave, preprocesses them, and stores it in the reserved memory. Then, it invokes the SMM code, which applies the patches to the kernel by dynamically rewriting the kernel memory.

7. Conclusion

This paper proposes SSdetector for achieving both the security and manageability of HIDS by combining SGX and SMM. SSdetector executes HIDS in an SGX enclave securely and makes the management of HIDS easier by running as an SGX application. It invokes the SMM monitor in BIOS to securely obtain the memory data of the target system using encryption and integrity checking. The information passed between in-enclave HIDS and the SMM monitor is protected by encryption and integrity checking. We confirmed that our HIDS could collect system information necessary for the proc filesystem. In addition, we showed that the overhead due to using SGX, SMM, encryption, and integrity checking was not small but could be reduced. by using full CPU support in PMs.

One of our future work is to run various HIDS using SSdetector. It is necessary to run existing HIDS in an SGX enclave using the library OS, as proposed in SCwatcher [6]. If HIDS performs time-consuming attack detection in addition to the collection of system information, the overhead of SSdetector could be reduced. In addition, we need to reduce the overhead of encryption and integrity checking using AES-NI in the SMM monitor and the SHA Extensions. Also, we would like to apply SSdetector to PMs and show the actual overhead.

Acknowledgment

This work was partially supported by JST, CREST Grant Number JPMJCR21M4, Japan. These research results were partially obtained from the commissioned research (No.05501) by National Institute of Information and Communications Technology (NICT), Japan.

References

[1] J. Rutkowska and R. Wojtczuk. Preventing and Detecting Xen Hypervisor Subversions. Black Hat USA, 2008.

[2] J. Wang, A. Stavrou, and A. Ghosh. HyperCheck: A Hardware-assisted Integrity Monitor. In *Proc. Int. Symp. Recent Advances in Intrusion Detection*, pages 158–177, 2010.

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

[4] F. Zhang, K. Leach, K. Sun, and A. Stavrou. SPECTRE: A Dependable Introspection Framework via System Management Mode. In *Proc. 43rd Annual IEEE/IFIP Int. Conf. Dependable Systems and Networks*, pages 1–12, 2013.

[5] T. Nakano and K. Kourai. Secure Offloading of Intrusion Detection Systems from VMs with Intel SGX. In *Proc. IEEE Int. Conf. Cloud Computing*, pages 297–303, 2021.

[6] T. Kawamura and K. Kourai. Secure Offloading of User-level IDS with VM-compatible OS Emulation Layers for Intel SGX. In *Proc. IEEE Int. Conf. Cloud Computing*, pages 157–166, 2022.

[7] T. Ngoc, B. Teabe, A. Tchana, G. Muller, and D. Hagimont. Mitigating Vulnerability Windows with Hypervisor Transplant. In *Proc. 16th European Conf. Computer Systems*, pages 162–177, 2021.

[8] Intel Corp. Intel Software Guard Extensions SDK for Linux. https://01.org/intel-softwareguard-extensions.

[9] EDK II Project. EDK II. https://github.com/tianocore/edk2.

[10] Y. Ozaki, S. Kanamoto, H. Yamamoto, and K. Kourai. Detecting System Failures with GPUs and LLVM. In *Proc. ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 47–53, 2019.

[11] N. Murilo and K. Steding-Jessen. chkrootkit – Locally Checks for Signs of a Rootkit. http://chkrootkit.org/.

[12] S. Gulley, V. Gopal, K. Yap, W. Feghali, J. Guilford, and G. Wolrich. Intel SHA Extensions. White Paper, 2013.

[13] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *Proc. USENIX Symp. Operating Systems Design and Implementation*, pages 689–703, 2016.

[14] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan. Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX. In *Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pages 955–970, 2020.

[15] M. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-NFV: Securing NFV States by Using SGX. In *Proc. ACM Int. Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48, 2016.

[16] D. Kuvaiskii, S. Chakrabarti, and M. Vij. Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX). In *arXiv:1802.00508*, 2018.

[17] C. Tsai, D. Porter, and M. Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *Proc. USENIX Annual Technical Conf.*, pages 645–658, 2017.

[18] K. Leach, F. Zhang, and W. Weimer. Scotch: Combining Software Guard Extensions and System Management Mode to Monitor Cloud Resource Usage. In *Proc. Int. Symp. Research in Attacks, Intrusions, and Defenses*, pages 403–424, 2017.

[19] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie, and T. Yang. Establishing Trusted I/O Paths for SGX Client Systems with Aurora. *IEEE Trans. Information Forensics and Security*, 15:1589–1600, 2019.

[20] L. Zhou, F. Zhang, J. Liao, Z. Ning, J. Xiao, K. Leach, W. Weimer, and G. Wang. KShot: Live Kernel Patching with SMM and SGX. In *Proc. 50th Annual IEEE/IFIP Int. Conf. Dependable Systems and Networks*, pages 1–13, 2020.