Secure Offloading of Intrusion Detection Systems from VMs with Intel SGX

Tomoharu Nakano
Kyushu Institute of Technology
naka_tomo@ksl.ci.kyutech.ac.jp

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

***Abstract*—Virtual machines (VMs) inside clouds need to be monitored using intrusion detection systems (IDS). Since host-based IDS can be easily disabled by intruders, IDS offloading with VM introspection (VMI) is used to securely run IDS outside a target VM. However, offloaded IDS can be still attacked because it runs on top of a vulnerable operating system (OS). Various systems have been proposed to protect offloaded IDS, but no systems provide an appropriate execution environment to IDS. This paper proposes *SGmonitor* for enabling the secure execution of IDS offloaded from VMs inside clouds using Intel SGX. SGmonitor executes IDS in SGX *enclaves* and preserves confidentiality and integrity. It provides secure VMI for memory and storage by using encryption and integrity checking. To make the development of offloaded IDS easier, it provides the in-kernel API to in-enclave IDS and enables transparent access to OS data in VMs. We have implemented SGmonitor in Xen with SGX support and showed that the overhead of in-enclave IDS was 31% in compensation for much stronger security.**

1. Introduction

Recently, clouds are becoming targets of attackers because they consolidate too many virtual machines (VMs) into a small number of network locations. Unfortunately, some of the VMs have vulnerabilities and can be penetrated by attacks. In preparation for attacks, intrusion detection systems (IDS) are of more significance to prevent sensitive information inside VMs from being stolen. Host-based IDS runs inside target VMs, but it can be easily disabled by intruders. To securely execute IDS outside target VMs, IDS offloading with VM introspection (VMI) [1] has been proposed. Using this technique, intruders in target VMs cannot disable offloaded IDS.

However, even if IDS offloading is performed, offloaded IDS still suffers from attacks. IDS is usually offloaded to privileged VMs and runs on top of the full-fledged operating system (OS), which often has many vulnerabilities. If it is attacked by exploiting such vulnerabilities, it can be disabled as before IDS offloading. In addition, sensitive information that IDS obtains from VMs can be eavesdropped on. So far, various systems have been proposed to protect offloaded IDS [2]–[5], but no systems provide an appropriate execution environment to IDS.

In this paper, we propose *SGmonitor* for securely executing IDS offloaded from VMs inside clouds using Intel SGX. SGX is a processor feature for securely executing programs in *enclaves*. The execution of in-enclave IDS can prevent tampering of IDS and information leakage from IDS even inside clouds. Since only small enclaves are newly added to the trusted computing base (TCB) for IDS, SGmonitor can keep the size of the TCB small. Enclaves run as part of an SGX application and therefore the impact on the security and performance of the entire virtualized system is limited.

For a rich but minimum execution environment, SGmonitor provides the in-kernel API to in-enclave IDS. It transforms IDS code and transparently obtains OS data from the memory of target VMs. It protects obtained memory data using encryption and integrity checking. Also, SGmonitor securely obtains file data from encrypted virtual disks via the in-enclave filesystem. We have implemented SGmonitor using Xen supporting SGX [6]. Our experiments showed that the overhead of our in-enclave IDS was 31%.

The organization of this paper is as follows. Section 2 describes IDS offloading in clouds and the issues. Section 3 proposes SGmonitor for secure IDS offloading with Intel SGX. Section 4 explains the implementation of SGmonitor and Section 5 shows our experimental results. Section 6 describes related work and Section 7 concludes this paper.

2. IDS Offloading in Clouds

IDS offloading with VMI [1] securely runs IDS outside a target VM and monitors the system running inside the VM. Offloaded IDS analyzes OS data in the memory of the VM and obtains the system state using *memory introspection*. Also, it analyzes the filesystems in the virtual disks of the VM and examines files and directories using *storage introspection*. Thus, it can securely monitor the target system and detect attacks as if it ran inside a VM. For example, it can find malware by obtaining the list of running processes and searching for specific data in files.

However, even if IDS is offloaded outside target VMs, it can be still attacked. Offloaded IDS usually runs in privileged VMs, e.g., the management VM called Dom0 in Xen and the root partition in Hyper-V. Otherwise, it runs as host processes without VMs as in KVM. In any case, it runs on top of the full-fledged OS. Since such an OS has many vulnerabilities, it is difficult to protect offloaded IDS from

attackers. If attackers can access the memory of offloaded IDS, they can easily steal sensitive information that IDS obtains from the target VMs. Once they disable offloaded IDS, they can intrude into the target VMs without detection.

To tackle this issue, many systems have been proposed for the secure execution of offloaded IDS [2]–[5]. For example, IDS can be offloaded to more secure VMs [2]. Even system administrators cannot access such secure VMs. However, such VMs can be still vulnerable because they provide a rich execution environment including the full-fledged OS. This largely increases the size of the TCB for IDS. Without using secure VMs, IDS can be embedded into the hypervisor [3], which has a much smaller attack surface. One downside is to increase the size of the hypervisor and make the TCB for the entire virtualized system larger. This increases the risk at which the hypervisor is compromised and the entire system is affected. In addition, it is difficult to execute sophisticated IDS because the hypervisor provides only a poor execution environment to IDS.

IDS can be offloaded to the outside of the entire virtualized system including the hypervisor using nested virtualization [4]. Even if the entire virtualized system is attacked, IDS can continue to monitor target VMs. However, nested virtualization degrades the performance of the virtualized system largely. IDS can be run in remote hosts outside clouds [5]. This approach can protect IDS from intruders in clouds more easily. Since it cannot execute IDS inside clouds, system administrators have to prepare hosts for running offloaded IDS outside clouds. Also, it is costly to transfer monitored data to remote IDS.

As such, any previous systems do not provide an appropriate execution environment for offloaded IDS. The secure execution of offloaded IDS needs a system that satisfies the following four requirements. (1) Offloaded IDS should run inside the same cloud as target VMs to reduce the management cost and communication overhead. (2) The increase in the size of the TCB for offloaded IDS should be minimized to keep the attack surface as small as possible. (3) IDS offloading should not have a large impact on the security or performance of the entire virtualized system. (4) A rich but minimum execution environment is necessary to run sophisticated IDS, while the increase in the size of the TCB should be suppressed.

3. SGmonitor

This paper proposes *SGmonitor* for securely monitoring VMs by protecting offloaded IDS with Intel SGX inside clouds.

3.1. Threat Model

We assume that processors are equipped with flawless SGX. SGX is often used under the assumption that any software is untrusted, but we relax this too strong assumption to achieve a new application of SGX. In this paper, we assume that only the hypervisor inside clouds is trusted. The hypervisor provides the basis for the memory management of VMs, on which offloaded IDS relies to monitor VMs. Such a relaxed assumption has been also used in previous work using SGX [7]. The trustworthiness of the hypervisor can be confirmed by various techniques, e.g., remote attestation with TPM and tamper detection with hardware features [8]–[10].

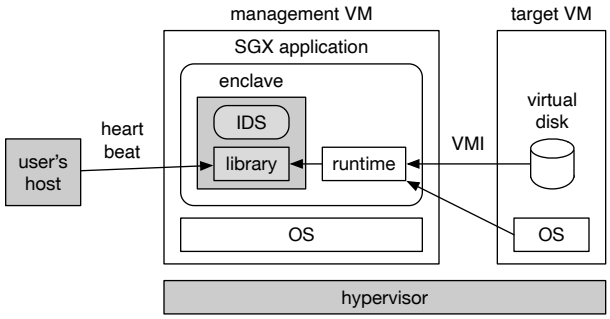


Figure 1: The system architecture of SGmonitor. Gray components are trusted.

In contrast, we do not trust the other software stack. Adversaries can compromise the entire management VM including its full-fledged OS. We assume that adversaries attempt to tamper with IDS offloaded to the management VM and obtain sensitive information of target VMs via offloaded IDS.

3.2. Secure IDS Offloading with SGX

SGmonitor securely offloads IDS using SGX from target VMs in clouds. SGX is a processor feature that enables the secure execution of programs in an untrusted environment using protection domains called *enclaves*. Fig. 1 illustrates the system architecture of SGmonitor. Offloaded IDS is created as SGX applications and runs in the management VM on top of the hypervisor. An SGX application consists of trusted enclaves and an untrusted SGmonitor runtime. Each enclave runs IDS and the SGmonitor library. Only this small library is added to the TCB for offloaded IDS. Since an SGX application is just one process, the execution of IDS does not affect the security and performance of the entire virtualized system.

SGmonitor can securely execute offloaded IDS in enclaves even inside clouds. First, attackers cannot eavesdrop on sensitive information obtained from target VMs by IDS thanks to the memory encryption of enclaves. Only IDS in enclaves can access the memory and disk data of target VMs. Second, attackers cannot tamper with in-enclave IDS because SGX always checks the integrity of enclave memory. They cannot disable monitoring functions of IDS by modifying IDS at runtime. Third, attackers cannot launch malicious IDS in enclaves to illegally obtain sensitive information from target VMs. At the launch time of an enclave, SGX checks the digital signature of the IDS. In addition, SGmonitor checks that the launched IDS is legitimate using remote attestation.

Unfortunately, SGX cannot prevent attackers from stopping in-enclave IDS with SGX applications. If attackers can obtain administrative privileges or compromise the OS, they can easily stop SGX applications. To detect this type of attack, SGmonitor confirms the correct execution of IDS by securely sending heartbeats from the outside of clouds to the SGmonitor library. If the library does not respond to a heartbeat, the remote user can notice that offloaded IDS is stopped by an attacker. Since the heartbeats are generated using a secret key, attackers cannot respond to the heartbeats correctly. For secure heartbeats, SGmonitor requires a user's host outside a cloud, but its role is minimum unlike the previous work [5].

3.3. Secure VMI with SGX

SGmonitor enables in-enclave IDS to securely monitor target VMs using VMI. This is challenging because the functionality of enclaves is strictly restricted. For memory introspection, IDS first invokes the SGmonitor library in an enclave and communicates with the trusted hypervisor via the untrusted SGmonitor runtime. Using the runtime is necessary because in-enclave IDS cannot directly invoke the hypervisor. Then, IDS securely obtains OS data in the memory of a target VM and monitors the system state in the VM. SGmonitor encrypts all the data passed between an enclave and the hypervisor and checks the integrity to prevent the untrusted runtime from eavesdropping on or tampering with the obtained OS data.

Since it is troublesome to analyze OS data using low-level memory introspection, SGmonitor provides the in-kernel API to make the development of sophisticated IDS easier. The in-kernel API enables IDS to be developed like OS kernel modules loaded into target VMs. IDS developers can reuse OS code as much as possible. To bridge the gap between the in-kernel API and low-level memory introspection, SGmonitor transforms IDS code at compile time so that memory introspection is transparently performed in enclaves to target VMs. As a result, developers can develop in-enclave IDS without considering that IDS is offloaded and is executed in enclaves.

For storage introspection, IDS uses the in-enclave filesystem provided by the SGmonitor library to securely monitor files in the virtual disks of a target VM. It is not secure that the library invokes the runtime at a higher-level interface and uses the filesystem of an untrusted OS. Since the in-enclave filesystem cannot directly access virtual disks, it invokes the SGmonitor runtime at the block-level interface. Then, the runtime reads the specified blocks from virtual disks. To prevent the untrusted runtime from eavesdropping on disk data, SGmonitor uses encrypted virtual disks to run target VMs. The library obtains encrypted disk data via the runtime and securely decrypts it inside an enclave. Using encrypted disks is mandatory to protect VMs inside clouds even without IDS offloading.

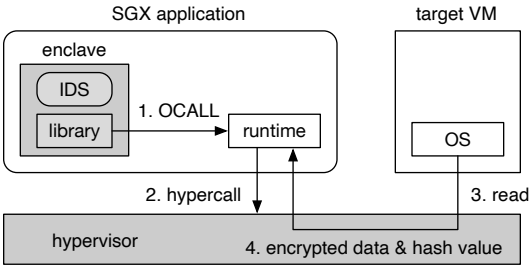


Figure 2: Memory introspection in SGmonitor.

4. Implementation

We have implemented SGmonitor using Xen-SGX 4.7 [6]. SGX requires a secure memory area for enclaves, called the enclave page cache (EPC). EPC is reserved by BIOS and the size is limited. Xen-SGX adds the management function of EPC to the hypervisor and enables VMs to use virtual SGX. SGmonitor creates a privileged VM called an *IDS VM* for running offloaded IDS with enclaves and allocates EPC to it. Unlike traditional IDS offloading, IDS cannot be offloaded to Dom0 in Xen because Xen-SGX does not support SGX virtualization for para-virtualized VMs. The IDS VM becomes one of the management VMs. Therefore, we assume that the entire software stack of the IDS VM is untrusted including its OS.

We used the Intel SGX SDK 1.9 to develop SGX applications containing in-enclave IDS. An SGX application executes in-enclave IDS by using the enclave call (ECALL) mechanism in SGX. ECALL enables untrusted code outside an enclave to securely invoke trusted code in the enclave.

4.1. Secure Memory Introspection

To obtain OS data in the memory of a target VM, IDS first invokes the SGmonitor library in an enclave and the library then invokes the SGmonitor runtime outside the enclave, as illustrated in Fig. 2. This invocation is done using the outside call (OCALL) mechanism in SGX. OCALL enables trusted code inside an enclave to invoke untrusted code outside the enclave. In this OCALL, the library passes the virtual address of OS data to the runtime. Then, the runtime invokes the hypervisor using a newly added hypercall for obtaining memory data of a VM.

To prevent the requested virtual address from being eavesdropped on or tampered with by the untrusted SGmonitor runtime, SGmonitor uses encryption and integrity checking. Upon invoking the runtime, the library generates a sequence number and calculates the hash value of the requested virtual address and the sequence number. Then, it encrypts the virtual address and the sequence number. The hypervisor decrypts the received request, calculates the hash value from the virtual address and the sequence number, and compares the calculated value with the received hash value. If the two values do not match, the hypervisor returns an error.

If the integrity of the request is kept, the hypervisor walks the page tables of the guest OS in the VM. Using the page tables, it translates the passed guest virtual address into a guest physical address. Then, it translates the obtained address into a host physical address using the extended page tables (EPT) for the VM. Finally, it obtains the memory data of the page including the requested OS data.

Then, the hypervisor calculates the hash value of the obtained data and the received sequence number. It encrypts the data and returns that with the hash value to the SGmonitor library via the runtime. The library decrypts the received response, calculates the hash value from the memory data and the sequence number, and compares the value with the received hash value. The sequence number is required to prevent the replay attack, in which the runtime illegally returns legitimate OS data captured before.

We have ported AES functions with AES-NI of wolfSSL to enclaves and the hypervisor. The SGmonitor library avoids executing the CPUID instruction for checking the availability of the AES-NI feature in CPUs. The execution of that instruction is not allowed inside enclaves. To use AES-NI in the hypervisor, SGmonitor saves and restores the XMM registers before and after invoking AES functions, respectively. This is because it was not assumed that the hypervisor uses those registers. Also, we have ported the SHA-256 function of OpenSSL to the hypervisor. The SGmonitor library uses the SHA-256 function provided by the SGX SDK in enclaves.

The SGmonitor library caches the obtained memory data in enclave memory. If IDS requires the same OS data or the other data in the cached pages, the library does not need to invoke the hypervisor. However, it is well known that the performance of enclaves largely degrades due to the overhead of memory encryption and decryption when used enclave memory exceeds the size of EPC [11]. Therefore, the library periodically removes older data in the cache and keeps the cache size appropriate. For example, it can flush the cache when IDS completes checking a set of malware.

4.2. Secure Storage Introspection

To enable in-enclave IDS to access files in target virtual disks, we have ported the virtual filesystem (VFS) and the ext4 filesystem in Xvisor 0.2.10 [12] as an in-enclave filesystem. Xvisor is a lightweight hypervisor for embedded systems. The ported filesystem provides minimum functions for monitoring virtual disks. Since Xvisor does not support several important ext4 features, e.g., extents usually used in Linux, we have added the necessary support. To delegate block-level accesses to the SGmonitor runtime, the SGmonitor library also provides a thin block layer. That layer issues OCALL with a disk offset and a data size.

The SGmonitor runtime performs actual access to encrypted virtual disks, as illustrated in Fig. 3. In advance, SGmonitor shares virtual disks located in Dom0 or network storage with the IDS VM using NFS. Then, it mounts partitions in the virtual disks and creates device maps. The runtime reads the specified blocks from the device maps and returns the data to the library in an enclave. Since all the partitions in virtual disks are encrypted, the runtime cannot eavesdrop on sensitive information. The SGmonitor library decrypts the received data and passes it to the filesystem.

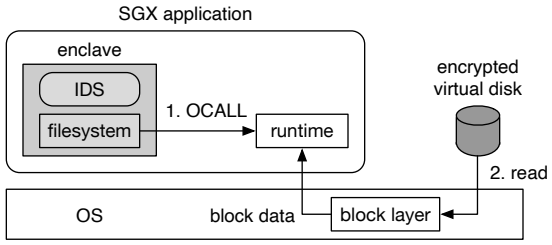


Figure 3: Storage introspection in SGmonitor.

```
#include <linux/sched.h>

void detect_process(void)
{
  struct task_struct *p;

  for_each_process(p) {
    if (strcmp(p->comm, "kworkerds") == 0)
      printk("Mining tools installed\n");
  }
}
```

Figure 4: A simple code example for detecting malicious processes by name.

4.3. Execution Environment for IDS

To enable in-enclave IDS to be developed like Linux kernel modules, SGmonitor provides an execution environment for using the in-kernel API. Fig. 4 shows a simple code example for traversing the process list in a target VM and detecting malicious processes by name. Developers can use the header files of the Linux kernel such as sched.h to write IDS code. They can use the kernel data structure such as `task_struct`, which contains the process name in the `comm` array. They can also use kernel macros such as `for_each_process` for examining all the processes and global kernel variables such as `init_task`, which is used in the `for_each_process` macro. In addition, they can use helper functions such as `strcmp` and `printk`, which are provided in the Linux kernel.

Using the LLView framework [13], SGmonitor compiles IDS code written with the in-kernel API and transforms the emitted LLVM intermediate representation to transparently obtain OS data from a target VM. Specifically, it inserts the invocation of the function for accessing the memory of a target VM with OCALL just before each load instruction. The inserted function invokes the SGmonitor runtime outside the enclave to issue the hypercall for obtaining the specified memory data, as described in Section 4.1. It does nothing if the target of the load instruction is a local variable. Then, SGmonitor modifies that load instruction to read the obtained data. In addition, it replaces kernel variables with

the corresponding virtual addresses used in the guest OS. The mapping from kernel symbols to their virtual addresses is obtained from the System.map file.

To monitor files and directories in virtual disks, SGmonitor provides the user-space API, which is a subset of the one provided by the C standard library. For example, developers can use functions such as read, stat, and readdir. Note that SGmonitor does not provide functions for modifying files and directories because VMI is used only for obtaining information.

4.4. Secure Heartbeats

To send heartbeats securely, a user's host generates a random number as a challenge and sends it to the SGmonitor runtime. Then, the runtime invokes the SGmonitor library in an enclave using ECALL. The library calculates the hash value of the received challenge and the secret key shared with the user's host. As a response, it returns the hash value to the user's host via the runtime. The user's host also calculates the hash value of the generated challenge and the shared secret key. If the hash value matches the received response, it is guaranteed that IDS is running normally. Since the hash value includes a secret key, only correct IDS can return correct responses.

4.5. Secure Key Sharing

SGmonitor securely shares a symmetric key for encrypting memory data between the hypervisor and an enclave. First, the SGmonitor library in an enclave generates an encryption key and encrypts it using the public key of the hypervisor. The encrypted key is securely passed to the hypervisor using OCALL and a hypercall. Then, it is decrypted using the corresponding private key in the hypervisor. Since only the hypervisor can access its private key, the SGmonitor runtime cannot decrypt the shared key.

To prevent an illegitimate enclave from registering an encryption key to the hypervisor, SGmonitor relies on a user's host outside a cloud. When an enclave is launched, it is remotely attested to by the trusted attestation server at a user's host. The server can accept only legitimate enclaves running pre-registered IDS and establish a secure communication channel. The SGmonitor library securely passes an encrypted key to the server and obtains its digital signature using the server's private key. It passes the encrypted key with the signature to the hypervisor. Then, the hypervisor verifies the signature using the server's public key.

SGmonitor uses two more keys for disk encryption and secure heartbeats. For disk encryption, a user's host securely sends a key for encrypting a virtual disk using the secure communication channel established with the SGmonitor library by remote attestation. Then, the library registers the key to the hypervisor. For secure heartbeats, a user's host and the library securely share a secret key similarly.

TABLE 1: The detection of malware using memory introspection.

malware	process name	module name	TCP port	UDP port	mmap file	kernel symbol	packet socket
kworkerds	✓					✓	
adore		✓				✓	
sebek		✓					
slapper			✓				
bindshell			✓				
tcpdump			✓				
scalper				✓			
suckit					✓		
sniffer							✓

5. Experiments

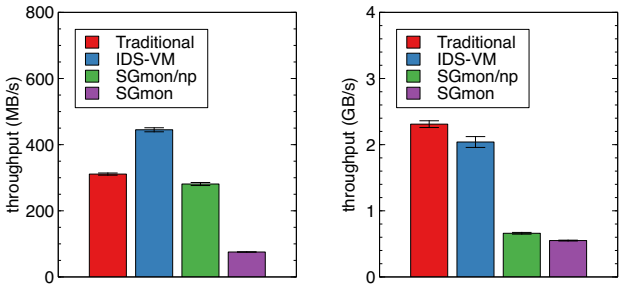
We conducted several experiments to show the effectiveness of IDS offloaded with SGmonitor. For comparison, we used three execution environments in addition to SGmonitor (SGmon). To identify the encryption overhead, we ran IDS using an unencrypted virtual disk without encrypting memory data or checking the integrity (SGmon/np). To examine the overhead of SGX, we also ran IDS without an enclave (IDS-VM). In this environment, IDS directly issued the hypercall for obtaining memory data and accessed a virtual disk without OCALL. To examine the impact by using an IDS VM, we ran IDS in Dom0 and used a local virtual disk without NFS (Traditional).

We used a PC with an Intel Xeon E3-1225 v5 processor, 8 GB of memory, and an HDD of 1 TB. We ran Xen-SGX 4.7 and created an IDS VM and a target VM. For each VM, we assigned two virtual CPUs, 2 GB of memory, and a virtual disk of 50 GB and ran Linux 4.4. For the IDS VM, we allocated 93 MB of EPC, which was the maximum size. We measured the execution time of IDS for each execution environment 10 times. To flush the cache at the OS level, we rebooted the PC every time we ran IDS.

5.1. Developed In-enclave IDS

We have developed in-enclave IDS that has similar detection capabilities to chkrootkit [14]. This IDS uses the in-kernel API although the original chkrootkit uses the user-space API through a shell script and external commands. As shown in Table 1, our IDS examines 11 features using memory introspection to detect 9 types of malware. Specifically, it checks process names, module names, TCP and UDP port numbers, the names of memory-mapped files, kernel symbols, and processes using packet sockets. Using storage introspection, the IDS examines the existence of 240 malicious files and searches for specific strings included in 9 files to detect 57 types of malware. In addition, it examines specific strings included in 55 system commands. For this string search, we have implemented a function equivalent to strings command and ported the regular expression functions from the musl library [15].

The code size of this IDS is only 396 KB. In the SGmonitor library (247 KB), the size of the regular expression functions is the largest (106 KB). The size of the filesystem



(a) Memory (b) Storage

Figure 5: The performance of VMI.

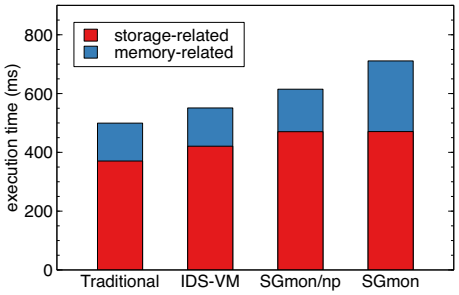


Figure 6: The execution time of the IDS.

is relatively small (53 KB) and only 11% of that of the ext4 kernel module in Linux. Note that our filesystem includes the VFS layer as well. This code size is enough small even for small EPC.

5.2. Effectiveness of In-enclave IDS

To confirm that our in-enclave IDS could detect malware, we first emulated various types of malware in the target VM. For example, we established a network connection to port 2001 used by the Scalper worm. We created a file named `/etc/xig` created by Rocke Monero Miner in the virtual disk of the VM. As a result, the IDS could detect all types of malware.

Next, we used a virtual disk called Metasploitable 2 [16] as a target of our IDS. This virtual disk includes various actual vulnerabilities and backdoors. Our IDS could detect a backdoor by checking the configuration file of inetd, for example. Since SGmonitor does not currently support the old 32-bit Linux kernel running in Metasploitable 2, our IDS could not analyze its OS data. It is not so difficult to support it because SGmonitor enables developers to reuse the source code of Linux as much as possible.

5.3. Performance of VMI

To examine the performance of memory introspection, we measured the throughput of obtaining OS data from the memory of the target VM. We created a benchmark program that read 50 MB of memory data in total. In this experiment, the SGmonitor library did not cache obtained memory data to always read the VM's memory. As shown in Fig. 5(a), the monitoring performance in the IDS VM was 43% higher than that in Dom0. The difference between Dom0 and the IDS VM is that Dom0 is para-virtualized but the IDS VM is fully virtualized. This difference affected the execution performance of the hypercalls for obtaining memory data. Using an enclave, the performance degraded by 37% due to the overhead of OCALL. The overhead of encryption and integrity checking was larger and reached 73%. In summary, SGmonitor degraded the performance of memory introspection by 76%, compared with the traditional method.

To examine the performance of storage introspection, we measured the throughput of reading a file in an encrypted virtual disk. We created a benchmark program that read a text file of 50 MB. As shown in Fig. 5(b), the monitoring performance in the IDS VM was 12% lower than in Dom0. This is because the benchmark in the IDS VM accessed the virtual disk located in Dom0 using NFS. When the virtual disk is located in network storage, the performance would be almost the same. Using an enclave, the performance degraded by 68% due to frequent OCALL at the block level. The overhead of decryption itself was small and only 16%. In summary, SGmonitor degraded the performance of storage introspection by 76%, compared with the traditional method.

5.4. IDS Performance

We measured the execution time of our IDS. In this experiment, the SGmonitor library cached all the obtained memory data. For the other methods including the traditional IDS offloading, the obtained memory data was also cached. Fig. 6 shows the average execution time and the breakdown for memory- and storage-related detection. Compared with the traditional IDS offloading, the overhead of our in-enclave IDS was 31% using SGmonitor. This overhead is much smaller than those of simple memory and storage introspection. This is because the IDS not only obtains data from the target VM but also inspects the obtained data.

For malware detection with memory introspection, the SGmonitor library obtained 6.1 MB of memory data in total from the target VM. This data was cached in EPC, but the impact on EPC was small. Compared with the traditional IDS offloading, this detection took 74% longer time in SGmonitor. In contrast, the execution time was only 5% longer in SGmonitor without data protection. This means that the overhead of SGX and our introspection library was small, but that of the encryption and integrity checking of memory data was large. For malware detection with storage introspection, the SGmonitor library obtained 15.6 MB of file data in total from the target VM. Compared with the traditional IDS offloading, the detection time was 17% longer in SGmonitor. Surprisingly, the overhead of decrypting disk data was negligible because it took a much longer time to perform pattern matching of the strings included in files.

6. Related Work

As described in Section 2, there are various systems for the secure execution of offloaded IDS [2]–[5]. In this section, we describe the other work on secure monitoring.

Several systems for securely running IDS with SGX have been proposed although they cannot run offloaded host-based IDS unlike SGmonitor. S-NFV [17] runs part of IDS in enclaves. It moves the states of virtual network functions in network function virtualization (NFV) and the state processing code to an enclave. As an example, it secures the per-flow state of the Snort IDS, which is network-based IDS. It minimizes the amount of code running in an enclave, but it is difficult to securely split an NFV application into one running in a trusted enclave and the other running in an untrusted outside world.

SEC-IDS [18] runs the entire Snort IDS in an enclave with almost no modification. To enable running legacy IDS in an enclave, it uses the Graphene-SGX library OS [19]. In addition, it uses DPDK to efficiently acquire network packets in an enclave. As long as the state of Snort fits into the size of EPC, SEC-IDS achieves near-native performance. However, the size of the TCB is largely bloated by Snort, necessary libraries, and Graphene-SGX. The lines of code are more than one million only for Graphene-SGX.

Using the other hardware features, several systems have been proposed to securely monitor the hypervisor or the OS. They can be applied to the monitoring of VMs. Copilot [20] and HyperCheck [9] send memory data to a remote host using a dedicated PCI add-in card and the system management mode (SMM) of Intel processor, respectively. They need to run IDS in remote hosts outside clouds. HyperGuard [8] monitors the hypervisor using SMM inside a target host. HyperSentry [10] enables an agent to securely run inside the hypervisor using SMM. Flicker [21] runs IDS using AMD SVM and Intel TXT securely. However, the entire system has to be stopped during the execution of IDS for security.

7. Conclusion

This paper proposed SGmonitor for the secure execution of offloaded IDS with Intel SGX. We have implemented SGmonitor in Xen-SGX and showed that the performance overhead of in-enclave IDS was 31%. For IDS executed asynchronously and infrequently, this overhead could be acceptable in exchange for much stronger security. One of our future work is to provide an extended execution environment such as user-space API to host-based IDS running in enclaves. To avoid the increase of the TCB and support existing userland IDS, we are planning to use SCONE [11] as a library OS unlike SEC-IDS [18]. Also, SGmonitor needs to support the integrity checking of disk data. Another direction is to use SMM for securely obtaining the memory data of VMs, instead of using the trusted hypervisor.

Acknowledgment

The research results have been achieved by the "Resilient Edge Cloud Designed Network (19304)," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

References

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

[2] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service Cloud Computing," in *Proc. ACM Conf. Computer and Communications Security*, 2012, pp. 253–264.

[3] Y. Oyama, T. Giang, Y. Chubachi, T. Shinagawa, and K. Kato, "Detecting Malware Signatures in a Thin Hypervisor," in *Proc. Annual ACM Symp. Applied Computing*, 2012, pp. 1807–1814.

[4] S. Miyama and K. Kourai, "Secure IDS Offloading with Nested Virtualization and Deep VM Introspection," in *Proc. European Symp. Research in Computer Security, Part II*, 2017, pp. 305–323.

[5] K. Kourai and K. Juda, "Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds," in *Proc. IEEE Int. Conf. Cloud Computing*, 2016, pp. 43–50.

[6] K. Huang, "Introduction to Intel SGX and SGX Virtualization," Xen Project Developer and Design Summit, 2017.

[7] H. Shuang, W. Huang, P. Bettadpur, L. Zhao, I. Pustogarov, and D. Lie, "Using Inputs and Context to Verify User Intentions in Internet Services," in *Proc. ACM SIGOPS Asia-Pacific Workshop on Systems*, 2019, pp. 76–83.

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

[9] 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.

[10] 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*, 2010, pp. 38–49.

[11] 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*, 2016, pp. 689–703.

[12] A. Patel, M. Daftedar, M. Shalan, and M. El-Kharashi, "Embedded Hypervisor Xvisor: A Comparative Analysis," in *Proc. Euromicro Int. Conf. Parallel, Distributed, and Network-Based Processing*, 2015, pp. 682–691.

[13] 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*, 2019, pp. 47–53.

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

[15] R. Felker et al., "musl libc," https://musl.libc.org/.

[16] Rapid7, "Metasploitable 2," https://metasploit.help.rapid7.com/docs/metasploitable-2.

[17] 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*, 2016, pp. 45–48.

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

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

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

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