# Keyspector: Secure Monitoring of IoT Devices Using RISC-V Keystone

Takahito Iwano
*Kyushu Institute of Technology*
*iwano@ksl.ci.kyutech.ac.jp*

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

***Abstract*—The Internet of Things (IoT) devices have become increasingly widespread in recent years. Since IoT devices tend to suffer from attacks from the Internet, they need monitoring using intrusion detection systems (IDS). However, IDS running inside a target system can be easily disabled by intruders. To address this issue, several approaches have been proposed to securely execute IDS using trusted execution environments (TEEs) such as Intel SGX and Arm TrustZone. Nevertheless, existing approaches have several drawbacks, e.g., substantial overhead for accessing the memory of the target system and too high privileges to execute IDS. This paper proposes *Keyspector* for enabling the secure execution of IDS using Keystone, which is a TEE for RISC-V processors. Keyspector executes IDS inside a secure execution environment called an enclave, which has relatively low privileges. For efficient monitoring, it enables only an enclave running IDS to share the memory of the target system using the security monitor running below the system. Using the shared memory, IDS can directly monitor the data of the target system. We have implemented Keyspector in the security monitor and the Eyrie runtime and confirmed that the overhead of the IDS running in an enclave was 10%, compared with the traditional IDS.**

***Index Terms*—IoT, host-based IDS, TEE, RISC-V, Keystone**

## 1. Introduction

The Internet of Things (IoT) has rapidly proliferated in recent years. The number of IoT devices is expected to increase to 40.6 billion by 2034 [1]. Since IoT devices, such as home routers, printers, TVs, and cars, are connected to servers or other devices via the Internet, they are exposed to a high risk of attacks. In fact, between 2023 and 2024, the number of detected malware targeting IoT devices increased by 45% year over year. In addition, traffic attempting to deliver malware payloads to IoT devices rose by 12% [2]. This indicates more active attacks to infect these devices. Examples of such attacks include distributed denial-of-service (DDoS) attacks by exploiting compromised IoT devices [3], [4], [5]. Incidents involving unauthorized modification of programs running inside IoT devices also occur [3], [5], [6], [7].

Since IoT devices tend to suffer from attacks from the Internet, they need monitoring using intrusion detection systems (IDS). However, host-based IDS running inside a target system can be easily disabled by intruders. To address this issue, several approaches have been proposed to securely execute IDS using trusted execution environments (TEEs) provided by recent processors. For example, IDS can be securely executed inside a protection domain called an enclave in Intel SGX [8]. It monitors system information by accessing the memory of the target system. Since enclaves cannot directly access system memory, substantial overhead is imposed to securely obtain memory data. Similarly, IDS can be executed in the secure world of Arm TrustZone [9]. It can directly access the memory of the target system running in the normal world, but the secure world has higher privileges than necessary for IDS. Executing IDS in this environment introduces potential risks.

To address these issues, this paper proposes *Keyspector* for enabling the secure execution of IDS using Keystone [10], which is a TEE for RISC-V processors. RISC-V is an open-source instruction set architecture that has recently gained attention in IoT devices. Like SGX, Keyspector executes IDS inside an enclave, which has relatively low privileges. For efficient monitoring, it enables IDS to directly access the memory of the target system by sharing the system memory with the enclave using a mechanism called Physical Memory Protection (PMP). To prevent intruders from eavesdropping on the system memory by creating malicious enclaves, the security monitor underlying enclaves allows an enclave to share the system memory only when the identity of IDS running in the enclave is confirmed by attestation. Then, a lightweight operating system (OS) running inside the enclave maps the shared memory into the address space of the IDS in a read-only manner. Using the shared memory, the IDS can monitor data in the OS of the target system.

We have implemented Keyspector in the security monitor and the lightweight OS called the Eyrie runtime in an enclave. Then, we have developed an IDS that collects system information provided by the proc filesystem of the target system. This IDS translates the virtual addresses of the monitored OS data of the target system into physical ones using the page tables located in the system memory. It obtains the information on the page tables from the security monitor. In addition, using LLView [11] ported to RISC-V, the IDS can transparently access the OS data of the target system using the source code of the OS. We conducted

several experiments using the developed IDS and confirmed that it could collect the same system information as that obtained inside the target system. We also measured the collection time of system information and confirmed that the overhead of the IDS running in an enclave was 10%, compared with the traditional IDS running in the target system.

The organization of this paper is as follows. Section 2 describes the monitoring of IoT devices using TEEs. Section 3 proposes Keyspector for enabling the secure execution of IDS using Keystone, and Section 4 explains its implementation. Section 5 presents the security analysis, and Section 6 shows the experimental results. Section 7 describes related work, and Section 8 concludes this paper.

## 2. Monitoring of IoT Devices Using TEEs

Since IoT devices are connected to the Internet, they are highly exposed to external attacks. In high-functionality IoT devices equipped with the OS, such as home routers, printers, TVs, and cars, it is unfortunately difficult to eliminate all the vulnerabilities that may become entry points for intruders. Therefore, IDS is required to detect attacks against IoT devices. IDS monitors abnormal behaviors of the system inside an IoT device and notifies administrators of possible intrusions. However, it is not easy to securely execute host-based IDS inside a target system. Once the system is compromised, IDS itself may be tampered with by intruders. Although IDS can be protected by running it inside the OS, even such IDS can be compromised if the OS contains vulnerabilities.

To securely execute IDS, using trusted execution environments (TEEs) provided by recent processors has been proposed. One of the TEEs is Intel SGX [12], which enables an application to create a protection domain called an enclave. At the time of launching, SGX verifies the digital signature of the application code, so that programs tampered with by attackers cannot be executed. In addition, SGX ensures the integrity of the enclave memory and thereby prevents attackers from modifying the running code. Since the enclave memory is encrypted, data inside the enclave cannot be eavesdropped on by attackers. In this way, programs running inside an enclave can be securely executed under the protection of the CPU without trusting even the OS.

For example, SSdetector [8] prevents IDS from being eavesdropped on or tampered with by executing IDS inside an SGX enclave, as shown in Fig. 1(a). IDS monitors the target system by accessing the OS data in the system memory. However, it cannot directly access the system memory inside an enclave because an SGX enclave has the same privileges as an application. Therefore, it invokes BIOS running below the OS and securely obtains the memory data of the OS in System Management Mode (SMM). It encrypts and integrity-checks the memory data to prevent eavesdropping and tampering by the OS or other software. However, the invocation of BIOS and the protection of data incur significant overhead in obtaining memory data. In addition,

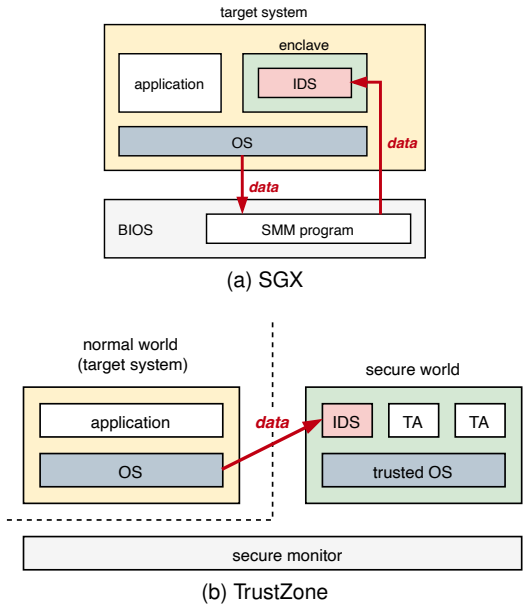



Figure 1. IDS using two types of TEEs.

the entire target system is halted while the SMM program is executed in BIOS. This performance impact on the target system is also considerable. Furthermore, SGX is currently supported only on server-grade Intel processors [13]. Thus, it cannot be applied to IoT devices.

On the other hand, Arm processors are commonly used in IoT devices and provide a TEE called TrustZone [14]. TrustZone provides two execution environments known as worlds. The normal world runs a general-purpose OS such as Android or Linux, while the secure world executes trusted applications (TAs) that need to be isolated from the normal world. A dedicated, trusted OS also runs in the secure world, independently of the OS in the normal world. The transition between the two worlds is managed by the secure monitor. Since system resources are strictly partitioned by the secure monitor, resources allocated to the secure world cannot be accessed from the normal world.

As shown in Fig. 1(b), the ITZ library [9] enables TAs in the secure world to access and analyze the system memory in the normal world. Using this library, various IDS can be developed. For example, IDS can perform integrity checks by calculating the hash value of the kernel memory and comparing it with that in a clean state, or by retrieving the values of the kernel variables. However, the secure world has higher privileges than necessary for IDS, such as access to devices. If IDS is compromised, these elevated privileges may be taken over by attackers. In fact, various vulnerabilities in TAs and the trusted OS have been reported [15]. Therefore, developing IDS in the secure world requires great caution. Many devices do not allow users to execute custom TAs.

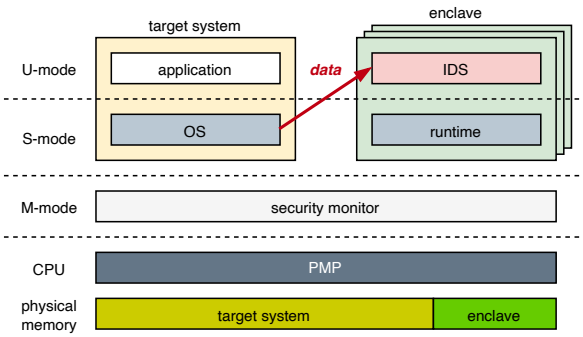



Figure 2. The system architecture of Keyspector.

## 3. Keyspector

This paper proposes *Keyspector* for enabling the secure execution of IDS using Keystone, which is a TEE for RISC-V processors. RISC-V is an open-source instruction set architecture that has recently gained attention and is expected to be widely adopted in IoT devices. Fig. 2 illustrates the system architecture of Keyspector. Keyspector executes IDS inside a Keystone enclave and prevents it from being eavesdropped on or tampered with. Both the enclave and the target system run on top of the security monitor and are strictly isolated from each other. The security monitor is software running in the highest privilege called M-mode.

The threat model of Keyspector is as follows. We assume that RISC-V processors and the security monitor are trusted and free of vulnerabilities. Also, we assume that attackers cannot intrude into enclaves running IDS by exploiting the vulnerabilities of software inside the enclaves. As for attacks on the target system, we assume attackers who attempt to compromise the system remotely over the network. Attackers inside the target system can take root privileges and control the OS and enclaves completely. They can create new enclaves and terminate existing enclaves, including ones running IDS. In addition, they can change the scheduling of enclaves.

Keyspector enables IDS inside an enclave to directly access the memory of the target system. In Keystone, the security monitor uses a mechanism called Physical Memory Protection (PMP) to logically isolate the memory of each enclave from that of the target system. As a result, enclaves are generally not allowed to access the target system memory except for the specific regions used for exchanging information. Keyspector modifies the accessible memory range and permissions in PMP for an enclave so that the enclave can share the entire target system memory. Then, a lightweight OS called a runtime inside the enclave maps the system memory into the memory address space of the IDS.

IDS monitors OS data in the target system by accessing the mapped system memory. Since it refers to OS data by virtual addresses, it needs to translate the virtual addresses of OS data into physical ones. The mapped system memory can be accessed by its base address and physical addresses. This address translation is performed using the page tables contained in the system memory. The address of the page tables is stored in the CPU register in RISC-V and saved by the security monitor at the time of switching from the target system to the enclave. Therefore, IDS invokes the security monitor via the runtime to obtain the register value. To allow users to develop IDS without being aware of such address translation, Keyspector embeds the translation logic into IDS at compile time so that address translation is performed transparently. Using this feature, IDS can be developed like a kernel module by leveraging the source code of the OS.

The security monitor in Keyspector allows access to the system memory only for the enclave in which IDS is running. If all the enclaves were permitted to share the system memory, the intruders who compromise the target system could execute malicious enclaves to eavesdrop on OS data in the system memory. In Keystone, the security monitor calculates the hash value of the software running inside the enclave. This hash value is used for remote attestation to confirm that the software is not tampered with at boot time. In Keyspector, the security monitor additionally checks that the hash value matches that of a pre-registered IDS and grants access to the system memory only for the enclave running the pre-registered IDS. As an alternative, the verifier at a remote host can check whether the hash value matches that of the IDS during remote attestation. Then, it notifies the security monitor of the result.

To detect attacks preventing the execution of IDS by intruders inside the target system, the security monitor inspects the execution state of the enclave running IDS. In Keyspector, the target system creates, runs, and terminates enclaves. Therefore, if intruders gain administrative privileges in the target system, they could forcibly terminate the enclave and disable the IDS. Since it is difficult to prevent this type of attack, the security monitor periodically checks for the presence of the enclave running IDS and detects the unauthorized termination of IDS. Even if intruders do not terminate the enclave, they could prevent IDS from being scheduled. To confirm that IDS is actively running, the security monitor checks the execution time of the enclave based on context switches between the target system and the enclave.

Unlike SGX, Keyspector allows its enclave to directly access the entire system memory. This eliminates the overhead of encryption and integrity checks to protect the retrieved memory data. In contrast, an SGX enclave can access only the memory of the application in which it is running. To access the system memory, it must rely on external trusted components such as BIOS and perform indirect memory access. Therefore, data protection mechanisms are required between the enclave and the external component. Additionally, Keyspector enables the security monitor to track the execution status of the enclave, whereas SGX lacks programmable and trusted privileged software to perform such monitoring. Compared with the secure world in TrustZone, the enclave in Keyspector has much lower privileges and can execute IDS more securely. We assume that the enclave has no vulnerabilities, but even if the enclave is compromised, attackers can only eavesdrop

on the system memory. In exchange for this advantage, the enclave in Keyspector can be more easily disabled. However, such attacks can be detected by the security monitor.

## 4. Implementation

We have implemented Keyspector in the security monitor and the lightweight OS called the Eyrie runtime in RISC-V Keystone.

### 4.1. Memory Sharing with Enclaves

PMP is a memory protection mechanism that restricts software access to physical memory regions. The security monitor can configure PMP using two types of registers: pmpaddr and pmpcfg. The pmpaddr register is used to define the base address and size of a memory region, which is aligned to four-byte boundaries. The pmpcfg register is used to specify access permissions such as read, write, and execute for each memory region. There are 16 sets of these two registers, depending on hardware implementation, and the index number of each register determines its priority. The security monitor assigns the highest priority for its own memory region, while it assigns the lowest priority for the memory region used for the untrusted host system. For an enclave, it assigns the in-between priority for the memory region. PMP checks are applied to all memory accesses performed in S-mode for running the OS and U-mode for running processes.

The security monitor re-configures PMP according to the execution context by changing the above-mentioned two registers. In Keystone, when switching the context from the host system to an enclave, the security monitor disables access to the system memory and enables access only to the physical memory assigned to the enclave. This provides an isolated execution environment to the enclave. Strictly speaking, the security monitor also enables the enclave to access only a part of the system memory so that the enclave can receive parameters from the host system and return results to it. This context switch occurs when a process in the host system launches an enclave or when it is rescheduled after being scheduled out. Conversely, when switching the context from the enclave back to the host system, the security monitor enables access to the system memory and disables access to the memory assigned to the enclave. This prevents the untrusted host system from accessing the memory region of the enclave. This context switch occurs when the enclave terminates or when the process that invokes the enclave is scheduled out.

In Keyspector, when switching the context from the host system to the enclave running IDS, the security monitor enables read access to the system memory, instead of completely disabling access, as shown in Fig. 3. This allows the enclave to directly monitor the system memory. Since the security monitor does not allow the enclave to modify the system memory, the enclave does not affect the behavior of the host system erroneously. When switching the context from the enclave to the host system, the security monitor

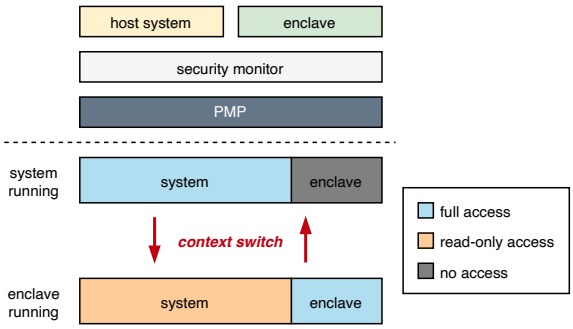



Figure 3. Switching the access permissions of physical memory.

enables all access to the system memory so that the host system can access its memory without any restrictions.

To allow IDS in an enclave to access the system memory, the Eyrie runtime maps the system memory onto the address space of the IDS. When an enclave is launched, the loader is first executed and then loads the runtime in the enclave. At this stage, the loader maps the physical memory assigned to the enclave onto the kernel address space of the runtime. In Keyspector, the loader additionally maps the system memory onto the process address space. Since only a single application runs in each enclave, the process address space is also only one. For this mapping, the loader sets up the page tables of the process so that only read access in U-mode is permitted. After that, the runtime loads IDS into the process address space. Keyspector guarantees that the mapped system memory does not overlap with the virtual address ranges used by the IDS for the stack and heap.

### 4.2. Controlled Memory Sharing

To limit enclaves that can share the system memory, Keyspector uses the hash value of software running inside an enclave. When the hash value matches that of the pre-registered IDS, the security monitor allows the enclave to share the system memory. When an enclave is launched, the security monitor calculates the hash values of the loader, the runtime, and the application that are executed inside the enclave, as shown in Fig. 4(a). These hash values are usually used for remote attestation, which can detect tampering with software running in the enclave. In Keyspector, the security monitor additionally checks whether the calculated hash values are the same as those for the pre-registered IDS. If the legitimate IDS is executed in the enclave, the security monitor sets a flag for permitting memory sharing to that enclave. Upon a context switch from the host system to an enclave, the security monitor enables the enclave to share the system memory if the flag is set. Otherwise, the system memory is not shared.

As an alternative approach, the attestation server used in remote attestation can determine whether the hash value matches that of a legitimate IDS. As shown in Fig. 4(b), the security monitor first calculates the hash value of software running inside an enclave and generates a signed report

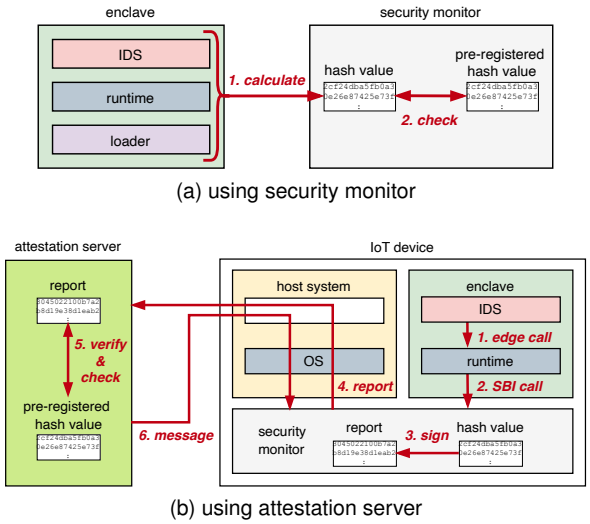



Figure 4. The controlled sharing of the system memory with enclaves.

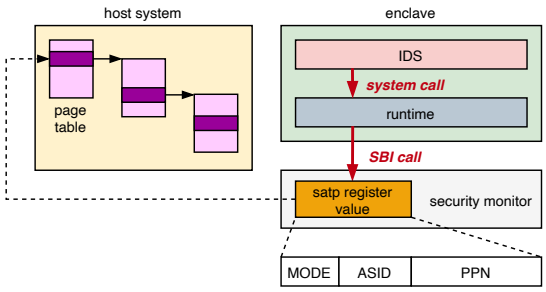



Figure 5. Obtaining the value of the satp register used by the target system.

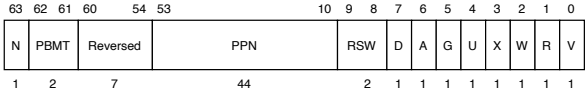



Figure 6. The internal structure of a PTE.

including the hash value. Then, it invokes the process that launched the enclave in the host system using an edge call mechanism and sends the signed report to the attestation server via that process. The attestation server checks whether the received hash value matches that for the pre-registered IDS. Then, it generates a signed message including the matching result and sends it back to the security monitor via the host system. The security monitor verifies the received message and sets the flag if the legitimate IDS is executed in the enclave. One advantage of this method is that a new hash value can be registered without modifying the security monitor when the hash value changes, e.g., by the updates of the IDS.

### 4.3. Accessing OS Data

IDS in an enclave obtains the kernel data of the host OS and monitors the untrusted host system. To access OS data, it translates the virtual address of the OS data into a physical address. Then, it adds the obtained physical address to the virtual address of the mapped system memory, so that it can identify the virtual address of the OS data in the address space of the IDS. This address translation can be done using the page tables of the host system, which are stored in the system memory. The physical address of the page tables is held in the satp register in RISC-V, but this register contains the address of the page tables used by the enclave during the execution of the enclave. The value of the satp register used by the host system is saved in the security monitor when the context is switched from the host system to an enclave. Therefore, IDS first invokes the runtime using a new system call, as shown in Fig. 5. Then, the runtime invokes the security monitor using a supervisor binary interface (SBI) call. Finally, the security monitor returns the saved value of the satp register to the IDS through the runtime.

Using the obtained physical address of the page tables, IDS walks through the page tables to find the page table entry (PTE) corresponding to a given virtual address. Since RISC-V supports multiple address translation modes, IDS obtains the used mode from the MODE field of the satp register. The current implementation supports Sv39 and Sv57. Sv39 uses 39-bit virtual addresses and three-level page tables, and Sv57 uses 57-bit virtual addresses and five-level page tables. It is not difficult to support the other modes such as Sv48 and Sv64.

Fig. 6 shows the internal structure of a PTE in RISC-V. The R and X fields indicate that the page is readable and executable, respectively. If IDS finds a non-leaf PTE where both fields are not set, it examines the next-level PTE indicated by the PPN field. If it reaches a leaf PTE where either the R or X field is set, it calculates a physical address using the PPN field and the lower 12 bits of the virtual address as an offset. If IDS encounters a leaf PTE before reaching the third level, it treats the page as a superpage of 1 GB or 2 MB, depending on the level. The result of the address translation is cached so that the page tables are not traversed again for virtual addresses that belong to the same page.

### 4.4. Developed IDS

Keyspector uses the LLView framework [11] to enable the transparent address translation of OS data. LLView is a framework that allows developers to write programs for analyzing OS data using the source code of the Linux kernel. Specifically, developers can use kernel data structures, global variables, inline functions, and macros defined in the kernel header files. Thus, they can reuse kernel code and write analysis programs like kernel modules. LLView compiles the written program and inserts code for address translation before each `load` instruction in the generated LLVM intermediate representation. To support LLView in cross-compilation for RISC-V, Keyspector specifies the 64-bit RISC-V architecture in both `clang` and `llc` and sets the ABI to `lp64d` in `llc`.

TABLE 1. THE PSEUDO FILES GENERATED BY THE DEVELOPED IDS.

| file path | system information |
|---|---|
| /proc/meminfo | statistics about system memory usage |
| /proc/net/tcp | information about TCP connections |
| /proc/stat | kernel and system statistics |
| /proc/sys/kernel/osrelease | kernel version |
| /proc/sys/kernel/pid_max | maximum process ID |
| /proc/tty/drivers | tty drivers loaded in the kernel |
| /proc/uptime | uptime of the system and idle time |
| /proc/<PID>/auxv | ELF interpreter information |
| /proc/<PID>/stat | status information about the process |
| /proc/<PID>/status | memory usage and status information |

```
for_each_online_cpu(i) {
    struct kernel_cpustat kcpustat;

    kcpustat_cpu_fetch(&kcpustat, i);
    idle = get_idle_time(&kcpustat, i);
    :
}
```

Figure 7. The program for obtaining CPU times.

Using LLView, we have developed IDS that analyzes OS data in the system memory to collect system information provided by the proc filesystem in Linux. The proc filesystem generates pseudo files that reflect the state of the Linux kernel and provides information about processes, network sockets, hardware, and system resources. Table 1 shows the pseudo files generated by the developed IDS. Fig. 7 shows a part of the program that collects per-CPU statistics. In this program, the `for_each_online_cpu` macro is used to traverse all the CPUs. The information is retrieved from the `kernel_cpustat` structure.

## 5. Security Analysis

Using the memory isolation provided by PMP, Keyspector can protect running IDS from attackers in the target system. It prevents attackers from tampering with and eavesdropping on IDS running in enclaves and OS data obtained from the target system. The configuration of PMP can be changed only by the security monitor running below the target system. Before the execution of IDS, attackers could inspect the binary files of IDS, but the files do not contain any secrets such as private keys. Similarly, Keyspector cannot prevent attackers from tampering with the files and launching altered IDS using them. However, the security monitor does not permit sharing the memory of the target system with such illegitimate IDS. The hash value of IDS does not match the pre-registered one stored in the security monitor or the attestation server. Therefore, attackers cannot eavesdrop on the memory of the target system via malicious IDS.

Keyspector cannot prevent IDS in enclaves from being disabled by attackers, but it can detect such attacks using the security monitor. Attackers can terminate enclaves with the processes that launched them using the functionality of the host OS. For this attack, the security monitor periodically checks the existence of the enclaves running IDS. If the enclaves do not exist, Keyspector can detect the attack. Also, attackers can prevent IDS from being scheduled because the host OS manages process scheduling. For this attack, the security monitor checks the interval between context switches from the target system to the enclave. If the enclave is not executed for a long time, Keyspector can detect the attack. These checks cannot be avoided because the security monitor is invoked at each timer interrupt.

Since IDS in enclaves monitors OS data in the target system to detect attacks, attackers could hide attack traces by altering OS data. This type of attack is effective for some of the OS data, e.g., removing malicious processes from the process list. However, the target system does not work correctly for most of the OS data because the attack affects the entire target system. Instead, attackers could replace the page tables with the ones altered to point to fake OS data just before a context switch to the enclave. As a result, IDS fails to monitor actual OS data. Since the altered page tables are referred to only by IDS in the enclave, the target system still works correctly. This type of attack can be detected by checking the used page tables in the security monitor at timer interrupts. It is difficult for attackers to replace the page tables just before the interrupt.

TABLE 2. HARDWARE SPECIFICATIONS AND RUNNING SOFTWARE.

| | emulator | real device |
|---|---|---|
| CPU | 4 cores | Freedom U740 SoC (4 cores) |
| memory | 2 GB | 16 GB |
| firmware | OpenSBI 1.1 | OpenSBI 1.1 |
| OS | Linux 6.1.32 | Linux 6.1.43 |

## 6. Experiments

To examine the effectiveness of Keyspector, we conducted several experiments using the developed IDS. This IDS collects system information necessary to generate ten types of pseudo files in Table 1, which are provided by the proc filesystem in Linux. For comparison, we used the traditional method that reads the proc filesystem inside the target system. In this experiment, we used both the RISC-V emulator and the HiFive Unmatched Rev B board manufactured by SiFive [16]. For the emulator, we ran QEMU 6.2.0 on a PC with an Intel Core i7-14700 processor and 64 GB of memory. Table 2 shows the hardware specifications and running software. The emulator and the real device supported different address translation modes, specifically, Sv57 and Sv39, respectively.

### 6.1. Capability Tests of IDS

We executed our IDS inside an enclave using Keyspector and displayed the contents of the generated pseudo files for confirmation. Fig. 8 shows a part of the displayed system information. To confirm the correctness of the obtained information, we compared this output with the information obtained by reading the proc filesystem inside the target

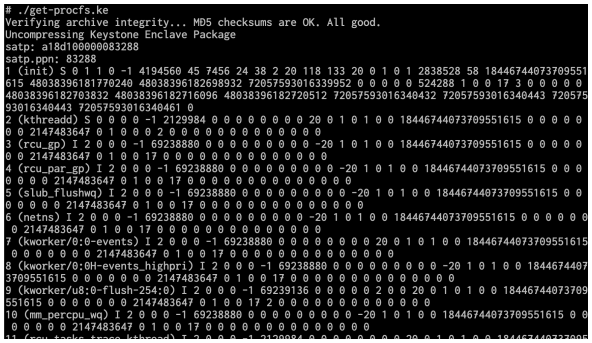
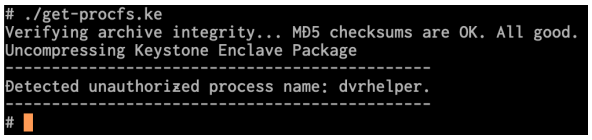



Figure 8. The execution results of our IDS.


(a) unauthorized communication




(b) unauthorized process

Figure 9. The detection of possible infection with IoT malware.

system. This was performed immediately after the execution of our IDS so that the change in system states was minimized. As a result, the output of our IDS matched the information obtained in the target system. From this result, it was confirmed that our IDS running in an enclave could correctly obtain the system information.

Next, we examined whether our IDS in an enclave could detect IoT malware. First, we ran a program that performed communication to TCP port number 2323, which was used by Mirai [3], in the target system. Then, our IDS obtained the list of TCP connections and checked the destination port numbers. As shown in Fig. 9(a), we confirmed that our IDS could detect unauthorized communication and report possible infection with the Mirai malware. Second, we ran a program whose process name was dvrhelper, which was used by Mukashi [17], a variant of Mirai, in the target system. Then, our IDS obtained the status information of all processes and checked the process names. Consequently, it was confirmed that our IDS could detect the unauthorized process executed by the Mukashi malware, as shown in Fig. 9(b).

Finally, we slightly modified the IDS program so that its hash value would differ from the one registered in the security monitor and executed it inside the enclave. Fig. 10 shows the runtime output when this IDS accessed the target system memory. From this result, it was confirmed that even if an intruder executes an unauthorized program inside the enclave, the information in the target system memory cannot be eavesdropped.

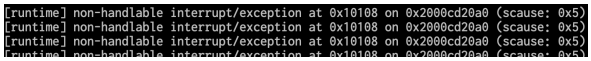



Figure 10. The execution results of an illegitimate IDS.

TABLE 3. THE NUMBER OF MONITORED OBJECTS.

| | emulator | real device |
|---|---|---|
| processes | 62 | 110 |
| TCP connections | 1 | 4 |
| tty drivers | 11 | 13 |

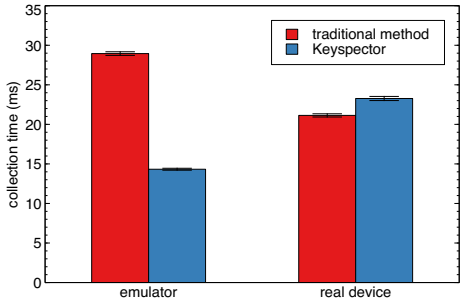



Figure 11. The collection time of system information.

## 6.2. Collection Time of System Information

We measured the total time required for the IDS to collect system information provided by the proc file system using Keyspector and the traditional method. Table 3 shows the number of monitored objects in the target system. The number of processes includes kernel threads and the process necessary for the IDS. In Keyspector, there also exists a process for running the IDS in an enclave. We executed the IDS more than ten times and calculated the average and standard deviation of the collection time. Fig. 11 shows the total collection time measured on the emulator and the real device. Compared with the traditional method, Keyspector reduced the collection time by a factor of 2 on the emulator. In contrast, the collection time was 10% longer than the traditional method on the real device.

To investigate the large difference between the emulator and the real device, we measured the time required to collect the system information necessary for generating each pseudo file. Fig. 12 shows the breakdown of the collection time. For process-related information, the collection time is the sum of the time taken to collect information for all processes. On the real device, Keyspector was faster or slower for each pseudo file. For example, it was 31x faster for pid-auxv, while it was 2.3x slower for tcp. In contrast, Keyspector was faster for almost all pseudo files on the emulator. In particular, it was much faster for process-related information, e.g., 31x for pid-auxv and 3.5x for pid-stat. This is the reason why Keyspector was faster than the traditional method on the emulator in total.

Next, we examined why the collection time of process-related information was much longer in the traditional method only on the emulator. To estimate the overhead of accessing the filesystem, we copied the files and directories

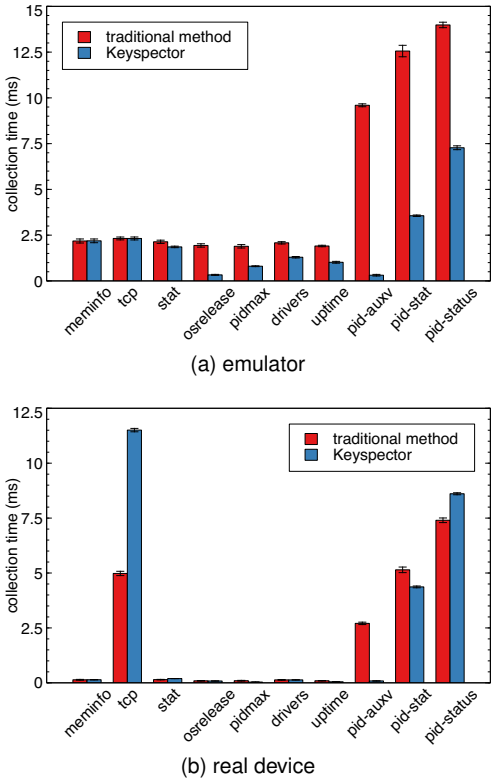



Figure 12. The breakdown of the collection time.

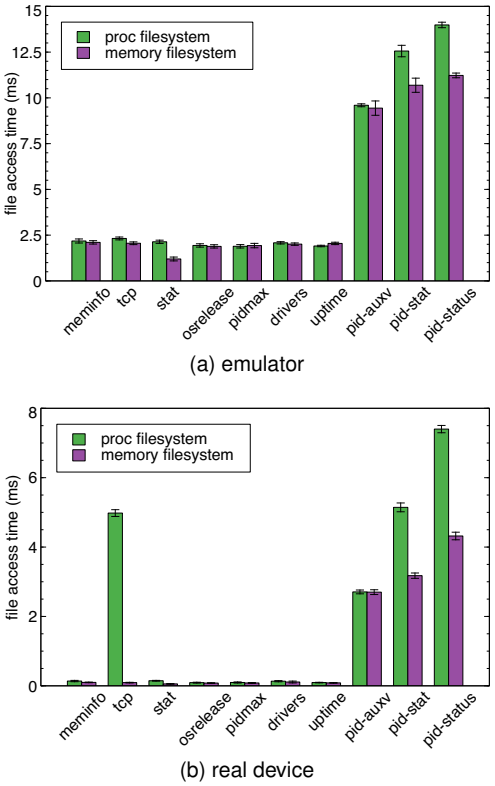



Figure 13. The file access time for each file in the traditional method.

in the proc filesystem to the memory filesystem. Then, we read the files from the memory filesystem, instead of the proc filesystem. Fig. 13 shows the file access time for each file in both filesystems. The difference is the time for generating a pseudo file in the kernel. On the emulator, the file access time occupied a large portion of the collection time. In contrast, the portion was relatively small on the real device, particularly in pid-stat, pid-status, and tcp. This means that file access is slower on the emulator or that the generation of pseudo files is slower on the real device.

Finally, we examined the reason why the collection time of tcp was much longer in Keyspector only on the real device. From Fig. 13, it was revealed that the generation time of tcp was much longer. Therefore, we measured the number of address translations performed by the IDS in an enclave. Fig. 14 shows the results on the emulator and the real devices. Most of the address translations hit in the software cache and did not walk the page tables of the target system. The number of address translations was large in three pseudo files. For pid-stat and pid-status, the collection time was proportional to the number of address translations on both execution environments.

For tcp, the collection time was not so long on the emulator, although the number of address translations was also large. On the real device, in contrast, the collection time was proportional to the number of address translations. The number of TCP connections was 4x larger on the real device, but its impact on the number of address translations and the collection time was small. As a result of our deep

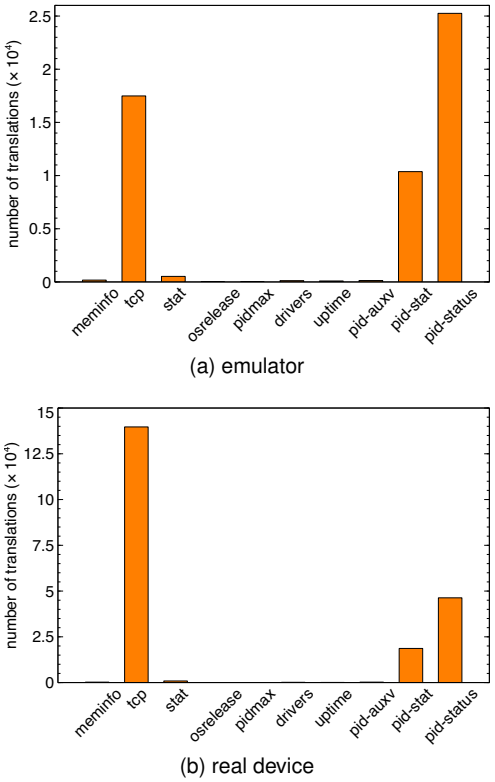



Figure 14. The number of address translations in Keyspector.

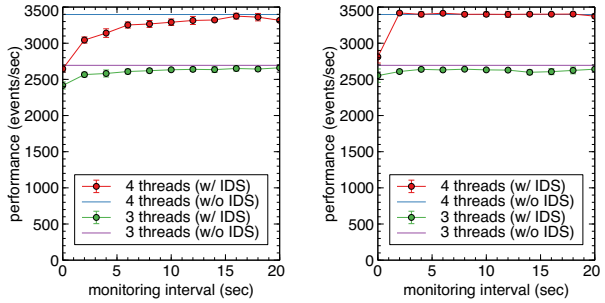



Figure 15. The impact on CPU performance (emulator).

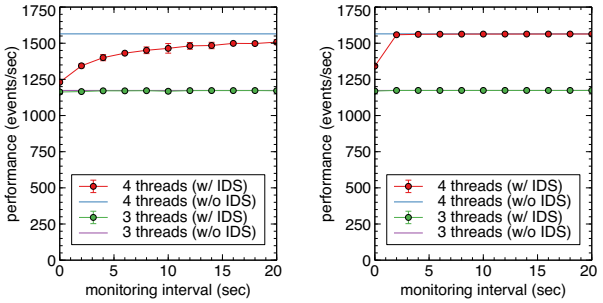



Figure 16. The impact on CPU performance (real device).

investigation, we found that the total size of the hash tables used to manage TCP connections in the kernel was 8x larger on the real device. Since our IDS traversed the hash tables to obtain the list of TCP connections, the number of address translations increased by a factor of 8. From this result, it was shown that the overhead of Keyspector came from address translation performed in software on the real device.

### 6.3. Impact on System Performance

To examine the impact of the execution of IDS on system performance, we executed the sysbench [18] benchmark in the target system. Then, we measured the CPU performance and the memory read performance while we periodically ran the IDS using Keyspector and the traditional method. In this experiment, we configured the IDS to collect system information at intervals ranging from 0 to 20 seconds in 2-second steps. For comparison, we measured the performance without running the IDS. Since the emulator and the real device had four CPU cores, we executed sysbench using three and four threads. The IDS could use one CPU core when sysbench used three CPU cores, while the IDS competed with sysbench for one CPU core when sysbench used four CPU cores.

Figs. 15 and 16 show the benchmark results of the CPU performance on the emulator and the real device, respectively. On the emulator, even when sysbench used only three CPU cores, the performance was degraded by 1.2-10% by executing the IDS. In contrast, the performance was not degraded on the real device. This means that the emulator has some performance impact on the system. When sysbench used four CPU cores, performance degradation was 0.7-22% and 3.6-21% in Keyspector on the emulator and the real device, respectively. This overhead was reduced by lowering the frequency of the execution of the IDS. In contrast, the performance was degraded by 17% in the traditional method only when we executed the IDS continuously. This means that the traditional method is more lightweight.

Figs. 17 and 18 show the benchmark results of the memory read performance. The performance was almost the same as the CPU performance, although the performance impact was slightly different.

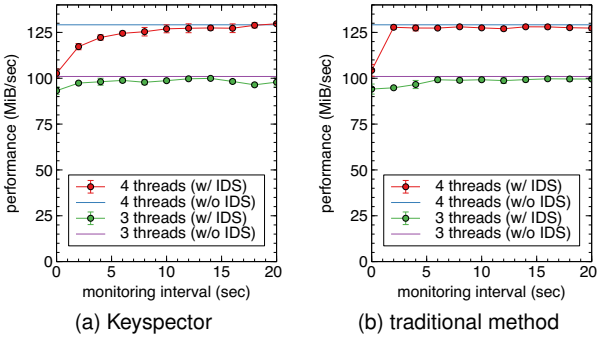



Figure 17. The impact on memory read performance (emulator).

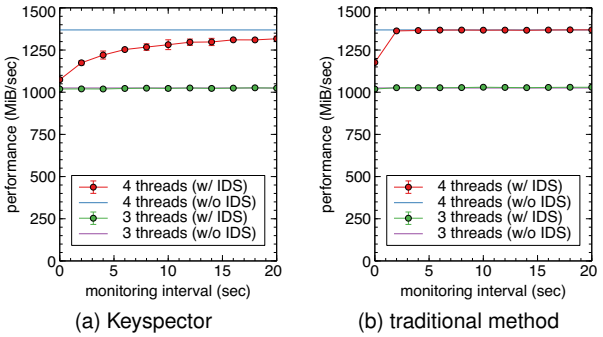



Figure 18. The impact on memory read performance (real device).

## 7. Related Work

HyperGuard [19], SPECTRE [20], and HyperCheck [21] achieve the secure execution of host-based IDS by using System Management Mode (SMM), which is one of the operation modes of Intel and AMD processors. These systems trigger a System Management Interrupt (SMI) to invoke the SMI handler in BIOS, which is a program executed in SMM. The SMI handler executes the entire HIDS in HyperGuard and SPECTRE. Therefore, BIOS must be updated whenever HIDS is updated. In addition, the entire system is paused while the SMI handler is executed. HyperCheck runs only a network driver in SMM and transfers memory data to HIDS running in a remote host. It can update HIDS easily

and minimize performance degradation caused by SMM, but the security of HIDS itself is not guaranteed. In contrast, Keyspector can easily update HIDS securely running in an enclave and suppress performance degradation.

SSdetector [8] enables the secure execution of host-based IDS by combining Intel SGX and SMM. It prevents IDS from being eavesdropped on or tampered with by executing IDS inside an SGX enclave. Since an SGX enclave cannot directly access the system memory, IDS obtains memory data by invoking the SMI handler via the untrusted code outside the enclave. To prevent eavesdropping and tampering by the untrusted application code or an untrusted OS, the enclave and the SMI handler need to encrypt and integrity-check exchanged data. If the process running the enclave is terminated, IDS in the enclave also stops. To detect such attacks, a remote host periodically sends heartbeats to IDS to confirm the correct execution. In Keyspector, the enclave can securely and efficiently access the system memory without data protection. The security monitor can check the correct execution of IDS without remote hosts.

S-NFV [22] and SEC-IDS [23] enable network-based IDS to securely execute using SGX. S-NFV protects the internal states of Snort [24] by storing them in an enclave. These internal states are accessed only by the code running in the same enclave. The code is invoked through a secure API outside the enclave. This prevents attackers from eavesdropping on information such as per-flow network states. SEC-IDS runs the entire Snort inside an enclave with almost no modification. It uses the Graphene-SGX library OS [25] to execute the existing IDS in an enclave. Although Snort can be securely executed inside an enclave, it may fail to detect attacks correctly if packets are tampered with before being passed to Snort. Keyspector could also support existing IDS by extending the lightweight OS running in an enclave. Even network-based IDS could be executed in an enclave by obtaining packets stored in the system memory.

The ITZ library [9] enables virtual machine introspection (VMI) using Arm TrustZone. It treats the normal world running the target system as a kind of virtual machine. From the secure world, it monitors the memory of the normal world using an API similar to LibVMI [26]. In the secure world, a microkernel and a virtual machine monitor are running. VMI tools using the ITZ library run on top of the microkernel. Since the normal world cannot access the secure world, intruders in the target system cannot disable the VMI tools. However, the secure world has too high privileges. Therefore, VMI tools can do more than just monitor the system. In Keyspector, an enclave running IDS has much lower privileges than the secure world, although it is extended to be able to read the system memory.

Trusted Monitor [27] also executes IDS securely in the secure world using TrustZone. The IDS uses CPU performance counters to perform anomaly detection. It collects the values of performance counters and uses a machine learning model for anomaly detection. The model is trained with the values obtained when the system is operating normally. Trusted Monitor runs an agent in the normal world to periodically invoke IDS in the secure world. If the agent does not invoke IDS for a certain period, a watchdog timer resets the entire system. To achieve IDS using performance counters, the system is configured so that the normal world cannot access the performance counters. This limits the capability of the target system running in the normal world. In Keyspector, IDS in an enclave could also use performance counters to detect anomalies.

Elasticlave [28] is a memory model that enables flexible memory sharing between enclaves and between an enclave and the host system in RISC-V. Since Keystone does not support direct memory sharing between enclaves, it is necessary to exchange data through the host system using the shared memory established between each enclave and the host system. Elasticlave enables enclaves to directly share memory with each other. It can assign different access permissions for each enclave to access shared memory and allows each enclave to change the assigned permissions if necessary. In addition, it supports exclusive access to shared memory. This memory model achieves a good balance between security and flexibility. Elasticlave assumes that enclaves and the host system share a part of their memory, while Keyspector shares the entire system memory with an enclave.

## 8. Conclusion

This paper proposes Keyspector for enabling the secure execution of IDS using Keystone, which is a TEE for RISC-V processors. Keyspector runs IDS in an enclave and allows it to directly access the memory of the target system. This is achieved by re-configuring access permissions in PMP using the security monitor when context switches occur between the target system and the enclave. To prevent information leakage via malicious enclaves, Keyspector limits enclaves that can share the system memory by extending the attestation mechanism. We have implemented Keyspector in the security monitor and the Eyrie runtime and developed the IDS that collects the OS data of the target system. Experimental results show that Keyspector can efficiently obtain system information provided by the proc filesystem in Linux.

One of our future work is to implement a mechanism for combining controlled memory sharing with remote attestation. We need to implement remote attestation in RISC-V and extend it securely. Also, we would like to implement the monitoring of the execution states of IDS in the security monitor. Another direction is to collect other types of system information provided by the proc filesystem in Linux and detect various types of attacks. We are planning to implement the proc filesystem in the lightweight OS running in an enclave and provide the collected system information to IDS.

## Acknowledgments

This work was supported by JST K Program Grant Number JPMJKP24U4, Japan.

## References


[1] Statista. Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2034. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2] Zscaler Inc. Zscaler ThreatLabz 2024 Mobile, IoT, and OT Threat Report. [Online]. Available: https://www.zscaler.com/campaign/threatlabz-mobile-iot-ot-report

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proc. 26th USENIX Security Symp.*, 2017, pp. 1093–1110.

[4] H. Griffioen and C. Doerr, "Examining Mirai's Battle over the Internet of Things," in *Proc. ACM SIGSAC Conf. on Computer and Communications Security*, 2020, pp. 743–756.

[5] O. Alrawi, C. Lever, K. Valakuzhy, R. Court, K. Snow, F. Monrose, and M. Antonakakis, "The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle," in *Proc. 30th USENIX Security Symp.*, 2021, pp. 3505–3522.

[6] C. Miller and C. Valasek, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, 2015.

[7] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet," in *Proc. 26th Annual Network and Distributed System Security Symp.*, 2019.

[8] Y. Koga and K. Kourai, "SSdetector: Secure and Manageable Host-based IDS with SGX and SMM," in *Proc. 22nd IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications*, 2023, pp. 539–548.

[9] M. Guerra, B. Taubmann, H. P. Reiser, S. Yalew, and M. Correia, "Introspection for ARM TrustZone with the ITZ Library," in *Proc. 18th IEEE Int. Conf. on Software Security and Reliability*, 2018, pp. 123–134.

[10] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: An Open Framework for Architecting Trusted Execution Environments," in *Proc. 15th European Conf. Computer Systems*, 2020.

[11] Y. Ozaki, S. Kanamoto, H. Yamamoto, and K. Kourai, "Reliable and Accurate Fault Detection with GPGPUs and LLVM," in *Proc. 16th IEEE Int. Conf. on Cloud Computing*, 2023, pp. 540–546.

[12] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," in *Proc. 2nd Int. Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[13] Intel Corp. Intel Processors Supporting Intel SGX. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions-processors.html

[14] ARM Ltd., "ARM Security Technology – Building a Secure System Using TrustZone Technology," White Paper, 2009.

[15] D. Cerdeira, J. Martins, N. Santos, and S. Pinto, "ReZone: Disarming TrustZone with TEE Privilege Reduction," in *Proc. 31st USENIX Security Symp.*, 2022, pp. 2261–2279.

[16] SiFive Inc. HiFive Unmatched Rev B. [Online]. Available: https://www.sifive.com/boards/hifive-unmatched-revb

[17] Palo Alto Networks. New Mirai Variant Targets Zyxel Network-Attached Storage Devices. [Online]. Available: https://unit42.paloaltonetworks.com/new-mirai-variant-mukashi/

[18] A. Kopytov. sysbench. [Online]. Available: https://github.com/akopytov/sysbench

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

[20] 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. on Dependable Systems and Networks*, 2013, pp. 1–12.

[21] F. Zhang, J. Wang, K. Sun, and A. Stavrou, "HyperCheck: A Hardware-AssistedIntegrity Monitor," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 4, pp. 332–344, 2014.

[22] 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.

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

[24] M. Roesch, "Snort – Lightweight Intrusion Detection for Networks," in *Proc. 13th USENIX Conf. on System Administration*, 1999, pp. 229–238.

[25] C. Tsai, D. E. 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.

[26] B. D. Payne, "Simplifying Virtual Machine Introspection Using LibVMI," Sandia National Laboratories, Tech. Rep., 2012.

[27] C. Eichler, J. Röckl, B. Jung, R. Schlenk, T. Müller, and T. Hönig, "Profiling with Trust: System Monitoring from Trusted Execution Environments," *Design Automation for Embedded Systems*, vol. 28, no. 1, pp. 23–44, 2024.

[28] J. Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, "Elasticlave: An Efficient Memory Model for Enclaves," in *Proc. 31st USENIX Security Symp.*, 2022, pp. 4111–4128.