SeCore: Continuous Extrospection with High Visibility on Multi-core ARM Platforms

Penghui Zhang, Bernard Ngabonziza, Haehyun Cho, Ziming Zhao, Adam Doupé, Gail-Joon Ahn
Arizona State University
{pzhang57,bngabonz,hcho67,zmzhao,doupe,gahn}@asu.edu

ABSTRACT

We present SeCore, which is a novel continuous extrospection system on multi-core ARM platform. SeCore leverages ARM TrustZone technology to keep one core in the secure world and assure the integrity of the static kernel data and code in the normal world. By breaking the original time-sharing paradigm of such systems, SeCore enables continuous coprocessor-like monitoring with high visibility into the rich execution environment on mobile and IoT platforms. By ensuring that secure tools execute on certain physical CPU cores, the system's attack surface is also significantly reduced.

ACM Reference Format:
Penghui Zhang, Bernard Ngabonziza, Haehyun Cho, Ziming Zhao, Adam Doupé, Gail-Joon Ahn. 2018. SeCore: Continuous Extrospection with High Visibility on Multi-core ARM Platforms. In *Proceedings of Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, March 19–21, 2018 (CODASPY '18),* 3 pages.
https://doi.org/10.1145/3176258.3176948

1 INTRODUCTION

Existing mobile and IoT systems and applications are fraught with vulnerabilities and prone to many attacks. In particular, attacks that can compromise OS kernels are a growing threat. When attackers take the control of an OS kernel, they can hide their traces, steal sensitive information, and install backdoors. Therefore, monitoring and protecting OS kernel has received significant attention. Existing techniques of monitoring and protecting the security of OS can be categorized into two classes:

1) Host based intrusion detection, where the intrusion detection system (IDS) runs as a monitor on its host from a higher privileged mode, such as a hypervisor and collects information used to identify possible intrusions on that host [2, 5]. The approach, which is named virtual machine introspection (VMI), significantly increases the size of the trusted computing base (TCB).

Hardware-isolated execution environment (HIEE), such as ARM TrustZone, enables another way of host based kernel protection on mobile and IoT platforms by running kernel protection tools in an isolated execution environment called the trusted execution environment (TEE), whose memory and peripherals can be physically isolated from a rich execution environment (REE) [1, 4, 7].

However, existing HIEE based solutions suffer from many if not all following limitations: i) a compromised REE kernel can perform many types of denial-of-service attacks (DoS) on the TEE by refusing to relinquish the control of CPU [6]; ii) it is required that the security world returns as soon as possible. Otherwise, running inside the TEE for a long time could jeopardize the stability and usability of the normal world commodity OS; iii) the context switches from the TEE to the REE, and vice versa are very expensive, which significantly increase the performance overhead and power consumption.

2) Secure coprocessor based intrusion detection, where the state and data of the host is collected and processed by monitoring software running on an external hardware coprocessor [3]. Secure coprocessors have many advantages including isolated and continuous monitoring and low interaction with the target OS. However, compared with VMI they retain relatively low visibility into the host, since they only play the role of an outside peer.

In this paper, we present a hardware-based security framework, namely SeCore, in which one or more general purpose computing cores will stay in a hardware-isolated or safe execution environment to monitor the system running in the rich execution environment. Different from the existing paradigm of hardware-isolated environments, where CPU cores switch between a privileged mode (secure world in TrustZone) and a normal mode (normal world in TrustZone), the proposed idea breaks the time-sharing paradigm of cores, which enables both coprocessor-like continuous monitoring and host-based-solution-like high visibility into the rich execution environment within a CPU. Different from virtual machine introspection where the guest OSes run inside the hypervisor, in SeCore the monitored OS or hypervisor do not run directly inside the monitoring system and tools. Indeed, the inspection tools run outside the rich execution environment but retain high visibility of the rich execution environment. We call this kind of inspection *extrospection*.

2 DESIGN AND IMPLEMENTATION OF SECORE

SeCore has three major steps in achieving extrospection: In Step 1, SeCore changes the boot process of the dedicated cores and make sure world switching instructions will never be executed on this cores. In Step 2: SeCore accesses the monitored system from the monitoring Cores. The challenges in enabling monitoring cores to access the monitored system include configuring the undocumented on-chip registers, since most vendors do not implement the standard memory space controller. We reverse engineer available firmware to understand how different SoCs support the memory configurations. In Step 3: SeCore monitors the normal world continuously.

SeCore has three features. First, tamper-resistance. Since SeCore and the nomral world are isolated via the ARM TrustZone security extension, the normal world does not have the authority

CODASPY '18, March 19–21, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5632-9/18/03.
https://doi.org/10.1145/3176258.3176948

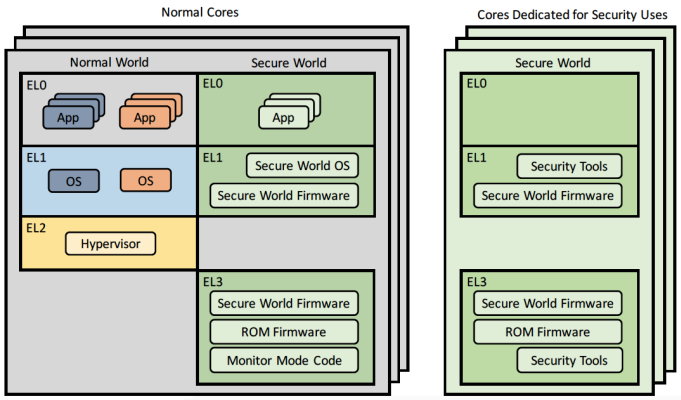

Figure 1: SeCore Architecture.

to access any resouces in SeCore, which makes SeCore tamper-resistant. Second, Continuous. We do not need to stop the normal world's execution for SeCore to perform extrospection. Both SeCore and the normal world operating system run concurrently. Third, stealty. It is difficult for the normal world to detect SeCore's extrospection is executing, which makes SeCore stealthy.

We implement SeCore on a Hikey board which has 8 ARM Cortex-A53 1.2GHz cores and 2GB of memory. Regarding the software stack, the secure world side runs OP-TEE, combined with ARM Trusted Firmware (ATF), while the normal world runs Linux.

2.1 SeCore Boot Process

In the normal boot process of a TrustZone-enabled platform, all cores boot in the secure world monitor mode <mon|s>. The cores are divided into one primary core and secondary cores. The system initializes the primary core first and let the primary core to wake the secondary cores up to get initialized. A bootloader that runs in <mon|s> sets up the environment, such as the stack for each mode, before it transfers control to the secure world operating system. The secure world operating system finishes its own initialization in <svc|s> and gives control back to a monitor by executing the SMC instruction. The code that runs in <mon|s> then switches the core's state from secure to non-secure. Then, the operating system running in the normal world, such as Linux and Windows, takes over and starts executing in <svc|ns> mode. When the normal world operating system needs services from the secure world, it executes SMC that forces the core to enter <mon|s> mode. Then, the monitor code will dispatch the request accordingly.

SeCore's initialization is different from the normal cores since the secure world operating system does not gives control to the normal world after finishing the initialization of SeCore.

Listing 1 shows how we implement SeCore during the booting procedure inside function `vector_cpu_on_entry`. Register `x0` stores the number of CPU after the execution of function `get_core_pos`. Function `vector_cpu_on_entry` allows us to select one or more secondary cores as SeCore according to our needs. In this example, we just select one core as SeCore and SeCore is implemented to initialize the 7th core which should stay in the secure world from Line 9. After selecting 7th core as SeCore, it jumps to `secore_func` which goes to execute SeCore functions.

2.2 Accessing Normal World Memory

SeCore first needs to allow the secure world to access all the memory space, which includes RAM, peripherals, etc., at boot time. Note that SeCore will also make sure some memory address space, such as the secure world's physical memory, cannot be accessed from the normal world at this step. This can be done by configuring TrustZone Address Space Controller (TSC). However, on some platforms the TSC is replaced by a vendor-specific address space controller, which normally does not come with documentations.

We identified such as a memory controller on HiKey board at memory address 0xF7121000 that is very similar with TSC. Even though it is undocumented, the PIs were able to reverse engineer the associated code and perform several experiments to understand how it works. It turns out that, when `secur_boot_lock` signal is high, its register `SEC_LOCKDOWN_SELECT` will be set as read-only. Also, other registers specified in `SEC_LOCKDOWN_SELECT` will be set as read-only. And, they can only be unset by a poweron reset. So, SeCore software module can first configures the controller at address 0xF7121000 including `SEC_RGN_MAP`, `SEC_RGN_ATTRIB`, and others to specify secure world can see the whole physical memory space and normal world can only see some of it. Then, SeCore lock the registers, which cannot be reverted until next power-on reset.

2.3 Continuous Extrospection

SeCore monitors the normal world kernel static memory by staying outside the environment. SeCore functions execute in the secure world exception level 3 (EL3), which has the highest privilege to monitor all the memory in the normal world. Therefore, SeCore has the ability to extrospect the static memory regions of the normal world kernel, which stays in the normal world exception level (EL1).

Even though TrustZone architecture allows the secure world to access all the memory of the normal world, the memory area still needs to be mapped before the secure world can actually access it. After the initialization of cores, SeCore needs to access the static code and data in the normal world kernel memory. However, the addresses SeCore has are the virtual addresses. To solve the problem,

```
LOCAL_FUNC vector_cpu_on_entry , :
adr   x16, thread_cpu_on_handler_ptr
ldr   x16, [x16]
blr   x16
mov   x19, x0
bl get_core_pos

/* select SeCore*/
cmp   x0, #7
beq   secore_func

mov   x1, x19
ldr   x0, =TEESMC_OPTEED_RETURN_ON_DONE
smc   #0
b  .  /* SMC should not return */
END_FUNC vector_cpu_on_entry

...

LOCAL_FUNC secore_func , :
/* execute functionalities of SeCore */
b  secore_func_in_c
b  .
END_FUNC secore_func
```

Listing 1: Code of Core Initialization in SeCore

SeCore needs to translate these virtual addresses to the physical addresses first, and then it can access the static memory region of the normal world kernel using the physical addresses. We implemented a function called `va2pa_in_sec()` to translate the starting and ending virtual addresses of the static kernel memory regions in the normal world to physical addresses so that SeCore can read the static code and data in the normal world kernel memory.

After SeCore gets the virtual address corresponding to the static memory region of the normal world kernel, it has the ability to monitor the normal world kernel and to check its integrity. In SeCore, integrity checking and monitoring take place continuously. As the static data in the normal world kernel memory is consecutively mapped, binaries in the static memory regions of the normal world kernel are read by accessing through the starting to the ending physical addresses which are converted from the virtual addresses by SeCore at first. To measure the integrity of static data and code, we compute the original hash values of the binaries using a cryptographic hash function before all processes begin to execute.

We store these hash values in order to check the integrity of static code and data later. After this, processes start being executed, and we implement a function `integrity_check()` running in SeCore to check the integrity continuously, by hashing the memory regions of the normal world kernel again, and comparing the hash values with the original one. If the hash values match with each other, the integrity of static code and data is guaranteed. If the hash values do not match with each other, it means that the static kernel memory region in the normal world has been tampered.

3 FUTURE WORK

The current version of SeCore checks the static code and data in the normal world kernel memory, which is linearly mapped in the memory. This feature offers us a convenient way to read all the data in binary as long as we know the starting and ending addresses of one specific memory region. However, the attackers would attempt to tamper the dynamic code to make the attacks successful. Unlike the static code and data, it is much more difficult to check the integrity of dynamic data memory region, as the data is changing all the time when the system is running. It is a challenge for an integrity check to distinguish between a normal operation and a potential tampering behaviour. In the future, we plan to enhance SeCore to monitor dynamic areas as well.

Given the number of vulnerabilities discovered in hypervisors, it is imperative to design a framework that is not only able to perform virtual machine introspection but also hypervisor inspection. However, no existing solution that can provide an unified framework to monitor and protect kernel and hypervisor simultaneously. We plan to extend SeCore for hypervisor inspection in the future.

A major limitation of secure coprocessors based intrusion detection systems is their visibility into the host is limited due to the fact that they play the role of outside peers. This characteristic makes it impossible to perform event-triggered monitoring without modifying the host system. We plan to extend SeCore with the functionality of event-triggered extrospection in the future.

4 CONCLUSION

We presented SeCore, an innovative continuous high visibility extrospection technique on multi-core ARM platform in this paper. SeCore exploits ARM TrustZone technology to keep one core in the secure world forever, assuring the computing integrity of data. By breaking the original time-sharing paradigm of such systems, SeCore enables continuous coprocessor-like monitoring with high visibility into the rich execution environment on such mobile and IoT platforms. And by ensuring that secure tools execute on certain physical CPU cores, the system's attack surface is significantly reduced. Also, with the increasing number of mobile CPU cores and based on the results of evaluation, SeCore only introduces a negligible overhead.

REFERENCES

[1] Ahmed M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. 2014. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 90–102.

[2] Tal Garfinkel, Mendel Rosenblum, et al. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection.. In *Ndss*, Vol. 3. 191–206.

[3] Hojoon Lee, Hyungon Moon, Ingoo Heo, Daehee Jang, Jinsoo Jang, Kihwan Kim, Yunheung Paek, and Brent Kang. 2017. KI-Mon ARM: A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object. *IEEE Transactions on Dependable and Secure Computing* (2017).

[4] Hyungon Moon, Hojoon Lee, Jihoon Lee, Kihwan Kim, Yunheung Paek, and Brent Byunghoon Kang. 2012. Vigilare: toward snoop-based kernel integrity monitor. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 28–37.

[5] Nick L Petroni Jr and Michael Hicks. 2007. Automated detection of persistent kernel control-flow attacks. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 103–115.

[6] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, et al. 2016. fTPM: A Software-Only Implementation of a TPM Chip. In *USENIX Security Symposium*. 841–856.

[7] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Sushil Jajodia. 2014. Trustdump: Reliable memory acquisition on smartphones. In *European Symposium on Research in Computer Security*. Springer, 202–218.