iCORE: Continuous and Proactive Extrospection on Multi-core IoT Devices

Penghui Zhang
Arizona State University
Penghui.Zhang@asu.edu

Haehyun Cho
Arizona State University
haehyun@asu.edu

Ziming Zhao
Rochester Institute of Technology
zhao@mail.rit.edu

Adam Doupé
Arizona State University
doupe@asu.edu

Gail-Joon Ahn
Arizona State University
Samsung Research
gahn@asu.edu
gailjoon.ahn@samsung.com

ABSTRACT

In this paper, we present iCORE, a novel continuous and proactive extrospection system with high visibility on IoT devices deploying multi-core ARM platforms. Dedicated cores named *Isolated Cores* are configured to stay in the TrustZone secure world upon system boot to perform monitoring functionalities to extrospect static normal world kernel memory area proactively, continuously, and stealthily. Different from the existing TrustZone paradigm, in which secure world serves as the slave of the normal world, iCORE makes the secure world play a master role. Therefore, iCORE remains stealthy and proactive to perform monitoring functionalities. The evaluation results show that iCORE is effective and imposes negligible performance degradation using the `SPEC CPU2017` benchmark.

CCS CONCEPTS

• **Security and privacy** → *Embedded systems security*; *Trusted computing*;

KEYWORDS

ARM TrustZone, Extrospection, CPU isolation

ACM Reference Format:
Penghui Zhang, Haehyun Cho, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. 2019. iCORE: Continuous and Proactive Extrospection on Multi-core IoT Devices. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3297280.3297364

1 INTRODUCTION

To protect OS kernels of IoT devices from potential attacks, two classes of monitoring mechanisms were proposed [17]. First, "in-the-box" approaches, refer to the security tools that reside in the OS kernel. Second, "out-of-the-box" approaches, are the tools who stay outside of the kernel. However, both of the mechanisms have limitations. For example, "in-the-box" security tools can be easily compromised if the OS kernel is under attacker's control. Also, "out-of-the-box" tools rely on vulnerable hypervisors and external hardware components. The shortcomings make it difficult to keep the security tools safe or to reduce the cost of virtualization and additional hardware deployment.

Recently, hardware isolated execution environments, such as ARM TrustZone [30], AMD SVM [1], and Intel TXT [15], were proposed to provide a trusted environment for secure execution outside of the normal execution environment. Among the isolated architectures, ARM TrustZone is exploited most on IoT devices [42]. In the existing ARM TrustZone paradigm, the secure world is designed as a slave of the normal world, because it only executes the operations that the normal world requests. This architecture facilities the researchers to develop security tools that reside outside of the monitored system, such as in TZ-RKP [6] and SPROBES [13]. However, to implement such security tools, the developers have to make invasive changes in the normal world OS kernel to interrupt certain functionalities. Besides, these methodologies burden the normal world OS kernel by frequently performing a world switch. Additionally, due to the traditional relationship between two worlds, the functionalities of the security tools can be never invoked if the attacker who compromises the normal world chooses not to. Therefore, IoT devices still encounter with potential attacks even though such security tools reside.

In this paper, we present iCORE, a novel continuous and proactive extrospection system with high visibility on IoT devices deploying multi-core ARM platforms exploiting ARM TrustZone extensions, to overcome the aforementioned limitations of current security tools. Dedicated cores named *Isolated Cores* are assigned to the secure world in the full power cycle starting from the system booting period to proactively, continuously, and stealthily extrospect the normal world. Secure boot procedure is deployed to help ensure the initialization of iCORE, which, in turn, provides a small trust computing base (TCB). The implementation of iCORE does not require any changes on the normal world operating system. Moreover, iCORE ameliorates the traditional master-normal-world and slave-secure-world concept by making the secure world play a role as a master of the system. Therefore, iCORE can execute its functionalities independently even if the normal world is compromised by an attacker.

iCORE is designed as an integrity monitor with the attributes of continuousness, stealthiness, proaction, and high visibility. First,

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297364

continuousness allows iCORE to detect any malicious modifications in time when monitoring the normal world operating system. Event-driven monitoring methods, such as [6, 13, 20, 29], are easy to implement, but malicious operations cannot be detected or responded if certain events do not occur. Second, iCORE is staying and operating the monitoring functionalities in the secure world. Therefore, attackers in the normal world cannot detect its existence. Third, iCORE is out of the control of the normal world, hence it can provide proactive and independent monitoring functionalities over the normal world. Last, the high visibility is brought by the design of ARM TrustZone architecture that the secure world where iCORE resides can access all the resources of the normal world with the highest privilege.

There are several challenges in designing iCORE. First, all cores are initialized with secure boot procedure to have the integrity of memory protected. During the booting procedure, the integrity of system images is protected stage by stage from malicious modification. Therefore, to deploy iCORE in the secure world, we need to prevent dedicated cores from entering the normal world. Second, in the traditional TrustZone paradigm, the secure world performs as a slave of the normal world to execute certain functionalities as the normal world requires. Instead, iCORE, which stays in the secure world, should seek the self-decision-making power to execute the operations by itself, not from the normal world. Third, the secure world has access to the normal world memory using physical address after memory registration, but the processes in the normal world exploit virtual addresses to acquire memory data [3]; since the normal world cannot be trusted from iCORE's point of view while page tables from the normal world are used to convert virtual addresses to physical ones, iCORE has to develop its one methodology to translate the addresses.

The contributions of this paper are as follows:

- We analyze and discuss the limitations of current monitoring and protection mechanisms on IoT devices. The limitations include the security of proposed tools, high cost to deploy, the requirement of manual effort in implementation, and low performance of the system.
- We introduce iCORE, a novel proactive and continuous extrospection with high visibility for IoT devices based on multicore ARM platforms exploiting ARM TrustZone extensions. iCORE overturns the traditional master-normal-world and slave-secure-world paradigm. iCORE does not require any changes on neither the normal world or the secure world OS.
- We implement and evaluate iCORE. The evaluation results show that a system with iCORE can execute operations with negligible overhead.

2 BACKGROUND

In this section, we present different monitoring mechanisms and the ARM TrustZone architecture.

2.1 Monitoring Mechanisms

2.1.1 In-the-box vs Out-of-the-box. To prevent and detect potential attacks on OS kernels, researchers developed two categories of approaches, namely "in-the-box" approach and "out-of-the-box" approach [17]. "In-the-box" approach utilizes system software where the protection and monitoring tools reside in the monitored system. One instance is kGuard [18], which is a compiler plug-in to reinforce the kernel to detect return-to-user attacks. Such approaches have been proved less practical because the security tools can be disabled or even eliminated if the OS kernel is compromised [20]. Based on this observation, the research community has realized that the monitoring and protection tools should be correctly isolated from the monitored system so that attacks on the monitored system will not affect the security tools [6]. As a result, "out-of-the-box" approach was proposed. "Out-of-the-box" approaches, such as [6, 13, 20, 29], purport to deploy security tools outside of the monitored system. The "out-of-the-box" approaches can be further categorized into hypervisor-assisted and hardware-assisted approaches based on the components that the security tools reside and take advantage of [20].

2.1.2 Hypervisor-assisted vs Hardware-assisted. The essence of hypervisor-assisted approaches is to utilize virtualization to provide security tools with a higher-privileged and isolated execution environment. Residing in a hypervisor or a virtual machine monitor (VMM), which runs on host's hardware to control the hardware and guest operating systems, the security tools such as [16, 22, 33, 36, 39] can inspect the monitored operating system along with its interaction with hardware resources, and thus can detect potential attacks. However, the hypervisor itself is fragile because it also contains large code base bringing countless vulnerabilities, such as [25, 26, 28]. Although sacrificing the whole hypervisor as a security tool can avoid the hypervisor being compromised, it is an unrealistic way because this idea will block the virtualization functionalities [13]. Furthermore, the high-cost of virtualization on IoT devices makes it difficult to deploy hypervisor-assisted protection approaches [6].

The hardware-assisted approach is also prevalent, where researchers develop their security tools with the help of isolated hardware. Some protection methodologies are designed to monitor kernel objects using external hardware [20, 29], which increase the cost of security tool deployment. Since 2004, new isolated hardware architectures, such as ARM TrustZone [30], AMD SVM [1], and Intel TXT [15], were proposed to provide a trusted area for secure executions. The essence of the isolated architectures is to physically segment the system resources into two worlds, named as the normal world (also as rich execution environment, REE) for conventional processing, and the secure world (also as trusted execution environment, TEE) for secure processing, respectively. This design guarantees that the security-sensitive data can be properly protected in the secure world [14]. With the assist of isolated hardware architectures, researchers have been transferring their focus to developing security tools based on the trusted execution environment. Among the security tools based on TEE, TZ-RKP [6] and SPROBES [13] are recent examples, which exploit the secure world to deploy the security tools. Both of the security tools protect the Linux kernel code area by trapping the normal world page table update operations and judging in the secure world if the operation is legitimate.

However, the execution time of workload increases due to performing world switch each time when a page table update occurs. In addition, modifications on both the normal and secure world OS before implementing the security tools are needed to interrupt the

page table updates. Also, the attackers who have successfully compromised the normal world can opt out the functionalities of the secure world.

2.1.3 Event-driven vs Continuous. Lunt et al. [23] proposed a prototype real-time intrusion-detection expert system (IDES). In IDES, discrete and continuous measures were discussed. A discrete measure is used on a finite and unordered set of range of values, while a continuous measure is deployed for an infinite and continuous set of range of values.

Some of the modern monitoring and protection mechanisms implement a discrete variant, event-driven, aiming to detect after certain events happen. Vigilare [29], Ki-Mon [20], TZ-RKP [6], and SPROBES [6] use different techniques to implement their security tools, but they all leverage the concept of event-driven.

Other techniques, such as HookSafe [44] and SecVisor [37], aim to protect the kernel code continuously, which means the protection methods keep the integrity of the kernel not affected by events taking place.

2.2 ARM TrustZone Architecture

2.2.1 Overview of TrustZone. ARM TrustZone [30] is designed as a hardware-assisted security extension to ARM architecture, such as ARM Cortex-A and Cortex-M. ARM TrustZone physically partitions the system into two worlds, namely the normal world and the secure world. Each world has its own banked registers and memory which are running on the world-specific operating systems and applications. Trusted applications (TA) can execute secure processes in the secure world. Client applications (CA) are running in the normal world to operate conventional processes. Shared memory and general registers are used to communicate between the secure world and the normal world. Processes that are running in the normal world must call the `smc` [4] instruction to trigger one of the services in the secure world in order to request the security-sensitive data, and the secure world will send the data back to the normal world if the request is accepted. The services that execute in the secure world, however, can access the resources in the normal world without permission from the normal world.

The secure world is proposed to help the normal world operate security-sensitive executions with high privileges to protect the secure data from being attacked and leaked. So it only executes the functionalities that the normal world requests. Hence, the secure world by the default design plays a role as a slave of the normal world even with the highest privileges among the whole system. Take comparing Apple Touch ID [43] as an example. The normal world calls `smc` instruction along with the fingerprint collected from the sensor to request a Touch ID comparison in the secure world. The secure world then executes certain functionalities and returns the result back to the normal world. However, it cannot invoke such operation without the request of the normal world. From this perspective, the functionalities of the secure world will not be executed if the normal world never calls them.

2.2.2 Core Initialization with Secure Boot. ARM Trusted Firmware (ARM-TF) is one of the major booters that is designed to initialize the ARM cores with TrustZone extension. Both cold boot, where the system is switched on physically, and warm boot, where cores have already been initialized, should go to ARM-TF reset entry point. Afterwards, different booting procedures step into their own initialization sequence.

For the cold boot, the initialization of hardware, including core, platform, and architecture, is performed first. The primary core, which is selected when it is released from reset and executes mainly the cold boot path, starts with the C runtime initialization. And the other cores, called secondary cores, are placed in a platform-specific state and wait to be woken up after the primary core finishes initializing enough functionalities [5].

As for the warm boot procedure, the system goes to the warm boot entry point to continue the configuring PSCI, platform, architectural, and generic setup, along with PSCI state maintenance [5].

During the initialization, secure boot procedure is exploited to protect the integrity of all the secure world software images from being unauthorized or illegally modified, applying cryptographic checks to every stage of the secure world booting procedure [2]. For example, a trusted vendor would sign the image that she plans to execute on the device with her private key and then send the image along with the signature to this device. The corresponding public key is stored and protected from being substituted to verify whether the image has been tampered with and whether it is from the trusted vendor. The secure boot procedure also exploits the concept of chain of trust, meaning that starting from the root of trust that located in on-SoC ROM, every other software component can be verified by its higher level component before being executed.

2.2.3 Normal World Memory Access from Secure World. User and kernel processes running in the normal world have their own private virtual address memory space, which is the contribution of MMU. When a process in the normal world wants to access the memory through the virtual address, MMU will convert the virtual address through translation tables to the corresponding physical address to access the memory.

Although the secure world can access all the resources in the normal world, physical addresses are required by the secure world to access the specific normal world memory. Since there is no function designed by default to convert the virtual address to the physical address, we design one which will be discussed in detail in Section 4. After acquiring the physical address, the secure world can access the static kernel memory in the normal world directly. In addition, the static kernel data is linearly mapped in the normal world memory. Therefore, with the starting and ending physical addresses, the secure world can load the whole data stored in the static memory area [8].

3 ASSUMPTIONS AND THREAT MODEL

3.1 Assumptions

We assume that iCORE is implemented on IoT devices deploying multi-core ARM platform with ARM TrustZone extension. We consider the secure world as trusted and the normal world as our monitoring target. Also, we assume that the system will not be compromised or attacked during the cold boot procedure. To prevent the system from being tampered during the booting time, ARM secure boot [2] procedure can be leveraged to initialize the processors in order to guarantee the integrity of the whole memory area.

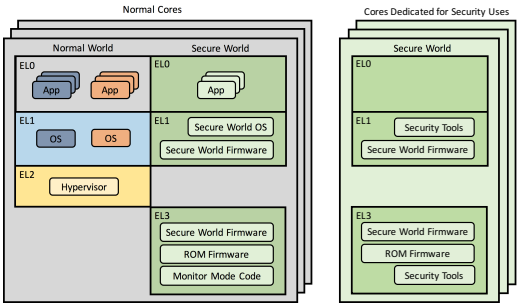

Figure 1: iCORE architecture.

3.2 Threat Model

The normal world OS kernel mainly encounters the threats from kernel level rootkits. As aforementioned, the normal world OS is vulnerable and can be compromised any time after the system has booted [11]. Therefore, kernel-level rootkits can be installed to take control of and attack the normal world OS. To hide the evidence of their intrusion, rootkits will tamper with some parts of the OS kernel such as setting a hook in certain system calls, which reside in the static kernel memory area.

Furthermore, attackers may notice that some security tools reside in the normal world OS. To avoid the detection from the security tools, the rootkits can deploy *transient attacks* [29], which do not tamper with the system permanently, in the kernel. By exploiting transient attacks, rootkits mitigate the permanent modifications on the kernel memory, which sets a camouflage to cheat on the security tools. However, iCORE checks the integrity of the static kernel memory area of the normal world continuously. Any modification on the monitored area will be detected. Consequentially, temporary modifications through transient attacks can still be detected by iCORE.

4 SYSTEM DESIGN

In this section, we present iCORE, a novel continuous and proactive extrospection with high visibility on IoT devices to monitor and detect the integrity of the static kernel memory in the normal world.

Figure 1 illustrates the architecture of iCORE. Dedicated cores assigned as iCORE for security uses are deployed and running only in the secure world, meanwhile, the other cores are processing the conventional workload with the access of both normal and secure worlds. Security tools of iCORE are residing in exception level EL1 in the secure world OS. With the privilege of EL1, iCORE has the right to access the normal world kernel memory that has the same exception level as iCORE. According to our design, iCORE can access and monitor the static kernel memory of the normal world continuously and proactively without notifying or getting permission from the normal world.

In the following subsections, we demonstrate how dedicated cores are chosen as iCORE and initialized during a modified booting procedure deployed in the secure world. Meanwhile, cryptographic checks are applied to each stage of the secure world boot procedure, pointing to protect the integrity of the secure world image from unauthorized and malicious modification. Then, we discuss a mechanism that iCORE acquires data stored in the static kernel memory region of the normal world and checks the integrity of the specific memory area with the acquired data proactively and continuously.

4.1 Initialization of iCORE

Figure 2 shows the flow chart of iCORE's initialization. The sequence of core initialization is from the primary core to the secondary cores and finally to iCORE. When the device is cold booted, the primary core is first invoked by a trusted booting firmware, for example, ARM-TF, and initialized in the secure world with exception level EL3. The primary core executes the boot path in the trusted booting firmware following such steps: Application Processor Trusted ROM (Boot Loader Stage 1, BL1), Trusted Boot Firmware (BL2), and EL3 Runtime Software (BL3). After the initialization of primary core is finished by the trusted firmware with EL3, the firmware then switches the control to the secure world OS with EL1 as shown in Step ① to continue the initialization of the primary core. In Step ②, the secure world OS should switch back to the trusted booting firmware after it finishes primary core initialization to perform the world switch operation with `smc` instruction through the trusted firmware. After the normal world OS gains the control from the secure world in Step ③, the primary core will have its initialization finalized in the normal world. The process of initializing secondary cores is similar to that of the primary core. After the primary core finishes its initialization in the normal world OS, a switch to the secure world that the trusted booting firmware generates to wake the secondary cores up, as shown in Step ④. Secondary cores are also initialized through the firmware to the secure world OS (Step ⑤), switching the world by the trusted booting firmware (Step ⑥), and eventually to the normal world OS (Step ⑦).

The regular cold boot procedure of an ARM multi-core platform indicates that the initialization of each core goes through the secure world and the normal world. As discussed in Section 3, the normal world can be compromised anytime so that the normal world is our monitoring target and cannot be trusted. To prevent iCORE from being infected, iCORE should never go to the normal world from the beginning of the initialization. To achieve this goal, we modify the boot procedures of iCORE by redirecting the core sequence to execute the monitoring functions. After the secondary cores finish the initialization in the normal world, a world switch to the secure world (Step ⑧) occurs to start the initialization of iCORE. iCORE is first initialized in the trusted booting firmware and then goes to the secure world OS to continue its booting (Step ⑨), as what other cores do. However, iCORE never switches to the normal world to finalize the initialization. Instead, it dedicates itself to stay in the secure world forever by executing security functionalities (Step ⑩) to monitor the normal world kernel memory and to detect potential malicious modification.

The normal world OS kernel waits for the cores to come online or time out. For example, when the normal cores switch to the normal world eventually, the normal world OS detects the normal cores online within certain time limit and notifies the successful detection to the system. When the normal world OS tries to detect iCORE, however, iCORE never returns to the normal world. After the time exceeds the threshold, the normal world OS stops waiting the dedicated cores to execute the following operations but informs the

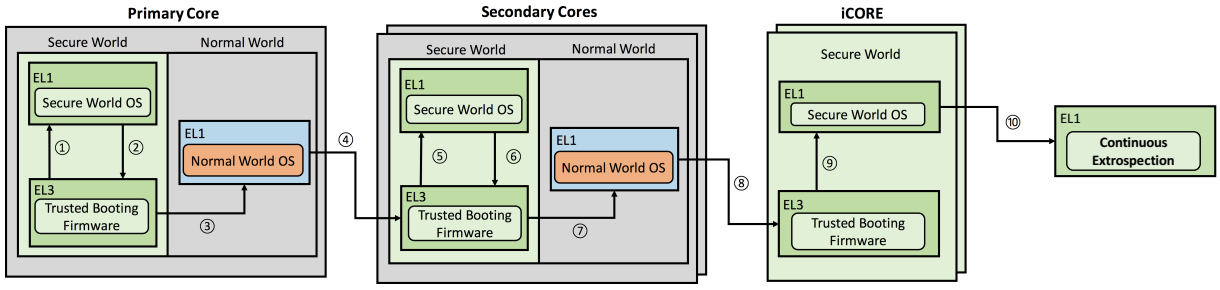

Figure 2: Core initialization with iCORE.

system during the booting period. Therefore, it does not influence the normal initialization and execution of the system that iCORE never returns to the normal world. By now, iCORE officially gets out of control of the normal world to be able to execute the monitoring functionalities independently and proactively.

In summary, by initializing and staying only in the secure world, iCORE can avoid the detection of potential attacks coming from the normal world. The normal world can still continue its conventional executions without iCORE coming online.

4.2 Continuous and Proactive Extrospection

To provide continuous and proactive extrospection with high visibility on the static kernel memory blocks of the normal world, iCORE needs to access the designated memory area, load the data, and check the integrity. Figure 3 shows the process of the extrospection of iCORE.

Though as originally designed in ARM TrustZone the secure world has the privilege to access all the resources in the normal world, which facilities iCORE to protect the normal world with high visibility, two features form an obstacle for iCORE. The first feature is that the normal world memory is not yet registered in the secure world, but the secure world can only access registered memory region. And the other one is that the addresses of kernel memory area are stored as virtual ones, the secure world, however, exploits the physical addresses to access the normal world memory. If iCORE plans to monitor one specific kernel memory region of the normal world, the corresponding memory region should be properly registered in the secure world beforehand and the starting and ending physical addresses must be obtained by converting the virtual addresses that the normal world use. However, ARM architecture does not offer any facilities for the secure world to register the normal world memory or to convert the virtual addresses. Hence, we design a function for iCORE by traversing a given virtual address on the normal world translation tables to retrieve the corresponding physical address. Additionally, we modify the configuration of memory layout to register the normal world memory in the secure world before the system boots. The detailed implementation of registering memory and converting virtual addresses will be discussed later in Section 5. Afterwards, iCORE can load the data used for integrity checking from the specific static kernel memory region of the normal world.

iCORE checks the integrity by the following steps. First, when the normal world OS is waiting for iCORE online during its initialization, iCORE loads the data from the monitored memory region and

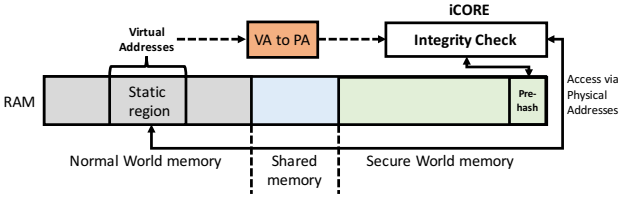

Figure 3: Extrospection of iCORE.

calculates a hash value of the data, named as `pre-hash`. Based on the assumption in Section 3, the attackers cannot compromise the normal world OS until the system is completely booted. Therefore, we can trust `pre-hash` as the root of trust. Also, `pre-hash` is stored in the secure world memory that cannot be accessed by the normal world and any trusted applications because trusted applications are in EL0, lower privilege than that of iCORE, EL1. Next, when the normal world OS is ready and the user processes start to execute, iCORE again accesses the same memory region and retrieve the hash value, named `current-hash`. Then iCORE compares `current-hash` with `pre-hash`. If these two values are equal, it indicates that the integrity of the monitored memory region is guaranteed. Otherwise, iCORE will report it as tampered memory area with memory dumped to be further analyzed.

Such extrospection provided by iCORE is continuous. iCORE is repeatedly computing `current-hash` of the monitored area and comparing those two values to guarantee the integrity. Moreover, the monitoring functionalities of iCORE are proactive. iCORE is completely running independently without permissions or requests from the normal world since it has got rid of the control of the normal world during the initialization.

5 IMPLEMENTATION

We implemented the prototype of iCORE and tested it on a Hikey LeMaker board, which has 8 ARM Cortex-A53 1.2GHz cores and 2GB of memory. Regarding the software stack of our testing environment, the secure world side runs OP-TEE version 2.0.0 [31], combined with ARM-TF [5], while the normal world runs Linux distribution with kernel version 4.15. we open source the prototype with the expectation that it will be exploited and extended further by security researchers [1].

[1] https://github.com/FormerBuckeye/iCore.git

```
1  LOCAL_FUNC vector_cpu_on_entry , :
2      ...
3      bl  get_core_pos
4      /* begin to select iCORE*/
5      cmp x0, #7
6      beq iCORE_func
7      /* end*/
8      ...
9      smc #0
10     b   .   /* SMC should not return */
11     END_FUNC vector_cpu_on_entry
12     ...
13     LOCAL_FUNC iCORE_func , :
14     /* execute functionalities of iCORE */
15     b   do_extrosepction
16 END_FUNC iCORE_func
```

Listing 1: Code of core initialization in iCORE.

5.1 Core Initialization

As mentioned earlier in Section 4, all the cores are firstly initialized in the secure world by the ARM-TF and then the secure world OS, in our case, OP-TEE. Eventually, all the cores are returned to the normal world to finalize the initialization by the normal world OS. We analyze the source code of OP-TEE and modify it to select dedicated cores as iCORE. In OP-TEE source code, Function `vector_cpu_on_entry()` in File `thread_a64.S` is responsible for switching from the secure world to the normal world using `smc #0` instruction after the core initialization is finished in the secure world. The detailed iCORE selection procedure we implement has been shown in Listing 1.

Function `get_core_pos()` in Line 3 gets the number of the current core and stores the result to Register `x0`. Line 5-6 is the modified code to assign dedicated cores. In our implementation, we select one secondary core as iCORE from the perspective of performance efficiency. There are 8 cores in the Hikey LeMaker board, one primary core numbered as 0 and seven secondary cores numbered through 1 to 7. We choose the core No.7 as iCORE because as discussed in Section 4, iCORE will be booted after all normal cores finish initialization. When core No.7 is initializing, Line 6 forces it to execute Function `iCORE_func()` to invoke iCORE functionalities, instead of executing `smc #0` to switch back to the normal world. In the normal world side, Function `_cpu_up()` in File `cpu.c` of the normal world OS is waiting for core No.7 to switch to the normal world. After time exceeds the limit, `_cpu_up()` will only notify the system that core No.7 fails to come online. Sequentially, the normal world OS ignores the offline core and continues to execute the following operations. By now, iCORE loses the control of the normal world and can perform the monitoring functionalities proactively and independently. The whole initialization does not require any changes on the normal world OS, which lighten the workload for developers to deploy iCORE architecture.

5.2 Memory Acquisition

After iCORE gets initialized and started to execute the functionalities, the static kernel memory region of the normal should be properly acquired by iCORE to load data. To make the secure world access the static kernel memory region of the normal world, as aforementioned

```
1  static struct map_area bootcfg_memory_map[] =
2  {
3      {
4          .type = MEM_AREA_NSEC_SHM,
5          .pa = DRAM0_BASE, .size = DRAM0_SIZE,
6          .cached = true, .secure = false,
7          .rw = true, .exec = false,
8      },
9      ...
10 }
```

Listing 2: Memory registration.

in Section 4, two tasks should be resolved. First, the normal world memory should be correctly registered in the secure world. Second, the virtual addresses used by the normal world should be converted to the corresponding physical addresses for the secure world to use.

We conquer the first task by modifying the boot configuration memory map in the secure world OS. Specifically, the secure world exploits `bootcfg_memory_map` to manage the memory layout provided to the TEE core. `bootcfg_memory_map` is a structure type of `map_area` to record the memory area registered in the secure world. In this structure, every memory area that the secure world needs to access is listed along with its type (all types of memory area have been enumerated in `mmu.h`), starting physical address, size, and attribute. Hence, we register the normal world memory to the secure world by adding the corresponding memory layout in `bootcfg_memory_map`. Listing 2 demonstrates the memory layout that we insert. The type of the memory layout is `MEM_AREA_NSEC_SHM`, which represents that the normal world memory now is registered as non-secure shared RAM between the normal world and the secure world; the starting physical address is `DRAM0_BASE`, which is the base address of the DRAM of the normal world; the size is `DRAM0_SIZE`, which is the size of DRAM of the normal world; and it can be cached, can be read and written, cannot be executed, and is non-secure.

Afterwards, we should solve the second task. There is no related instruction or function for the secure world to convert the virtual address to the physical address by default. In the normal world, one process accesses the memory by providing the virtual address in its own private space to MMU, and MMU then looks up the page tables to calculate the physical address. The secure world has to traverse the same page tables to convert the virtual address. Hence, we implement the converting function called `va2pa_in_sec()` in iCORE before accessing the memory of the normal world. Specifically, for a given virtual address in AArch64, the most significant bits determine the base address of the page table, and then bits [41:29] and [28:16] are the index of Level 2 and Level 3 page tables to find the corresponding page table entry level by level. Finally, we combine the bits [47:29] of the page table entry and the least significant bits of the virtual address to get the proper physical address.

5.3 Continuous and Proactive Extrospection

Finishing two tasks mentioned above, iCORE has the ability to monitor the normal world kernel and to check its integrity. As the static data in the normal world kernel memory is consecutively stored, iCORE can load data in the static memory regions of the normal world kernel by accessing from the starting to the ending physical

addresses. However, iCORE can only obtain the virtual addresses of the corresponding memory area by analyzing File `system.map`, which is generated during the kernel compiling and records the virtual addresses of symbols used in the normal world kernel. Function `va2pa_in_sec()` is then exploited to convert the virtual addresses to physical addresses to help iCORE access the data.

To measure the integrity of static data and code, first iCORE deploys SHA-1 cryptographic hash algorithm to compute the root of trust, `pre-hash`, of the loaded data before the normal world OS finishes the initialization. The `pre-hash` is then stored in the secure world memory. Second, when all the processes starts to execute, iCORE loads the data from the same normal world kernel memory region and computes the `current-hash`. Next, iCORE compares `current-hash` with `pre-hash`. The result of two hash-value comparison will determine whether the static kernel memory of the normal world has been tampered with.

To provide continuous monitoring functionalities, except the first step (because `pre-hash` is the root of trust), others should be processed repeatedly. iCORE leverages an infinite loop to achieve the continuous extrospection. Thus, in each loop iCORE can check the integrity by calculating and comparing the hash values. All the monitoring procedures are also executed by iCORE independently, without requesting or getting permission from the normal world, because the normal world cannot detect the existence of iCORE. Consequently, iCORE shows continuousness and proactiveness to detect the malicious modification in the monitored memory areas, and will not be affected or disabled by the normal world.

6 EVALUATION

After implementing iCORE, we answer the following questions that may be asked: How effective is iCORE? Can iCORE be detected or even terminated from the normal world? Losing one core brings a negative impact on the performance of IoT and mobile devices, can such a device be qualified to meet the daily requirements?

6.1 Effectiveness of iCORE

To evaluate the effectiveness of iCORE to protect the static kernel memory area of the normal world against the attacks and vulnerabilities that are probably exploited, we design experiments trying to (1) tamper the monitored memory area, and (2) terminate iCORE from the normal world. To experiment the first task, we deploy Loadable Kernel Module in the normal world attempting to read and write the monitored static kernel memory area with root privilege. Unsurprisingly, iCORE detects all the modifications on the memory region. It indicates that iCORE can detect any malicious modification on the static kernel memory area of the normal world from potential attacks such as malicious code injection [21] and vulnerabilities that are exploited to tamper with the static kernel memory such as allowing privileged users to modify a limited range of kernel memory in syscall interface of bridging [27].

Secondly, to verify that iCORE cannot be detected or disabled from the normal world, we check the `cpuinfo` of the normal world OS that records the CPU information. The content in `cpuinfo` only shows the information of CPU 0 to 6. It indicates that the normal world cannot detect the existence of iCORE and thus it does not provide any methods for attackers in the normal world to detect or even disable iCORE.

Table 1: Performance overhead when checking different size of static memory area.

Benchmark application	Monitoring whole static memory	Monitoring specific static memory
perlbench	2.28%	1.25%
mcf	3.61%	2.42%
omnetpp	3.14%	2.02%
xalancbmk	12.50%	2.79%
x264	2.38%	0.77%
deepsjeng	1.17%	1.01%
leela	1.92%	0.73%
xz	3.95%	2.64%

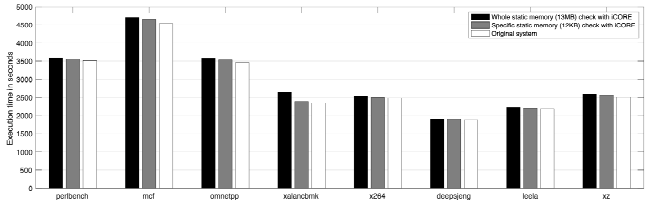

Figure 4: Evaluation result in SPEC CPU2017.

6.2 Performance with iCORE

iCORE may impact the performance of the device by frequently checking the whole static kernel memory of the normal world. Besides the impact of losing one core, frequent memory access may influence the speed of workload execution as well. Reading normal world memory may fill up L2 cache shared among cores and the data bus between CPUs and memory. Therefore, the data required by processes in the normal world would be delivered slower than the original system. We evaluate the performance of the system with iCORE by checking the whole static kernel memory of the normal world and by checking a portion of kernel text (code) area to determine how the impact of iCORE scales when it monitors different sizes of static kernel memory. According to our analysis, the size of whole static kernel memory is around 13MB while that of selected text (code) area is about 12KB.

We evaluate the performance of the normal world OS using the `SPEC CPU2017` to measure the impact of permanently losing one core to the secure world. Figure 4 represents the evaluation result under 3 situations, a system with 7 cores and iCORE checking the whole static memory of the normal world, one with 7 cores and iCORE checking a small portion of the static memory, and the original system with 8 cores, in `SPEC CPU2017`'s 8 benchmarks of `intrate` suit. Table 1 shows the performance overhead of the system when checking a different size of the static kernel memory area of the normal world. From the evaluation result, iCORE unsurprisingly brings the overhead on CPU performance of the system because of losing one core and frequently accessing the memory. When checking the whole static kernel memory of the normal world, iCORE generates

1.17% (*deepsjeng*) to 12.50% (*xalancbmk*) performance overhead compared with the original system.

According to the analysis of `SPEC CPU2017` in [32], benchmark *xalancbmk* has the highest percentage of branch instructions, over 30%, while the benchmarks *omnetpp*, *leela*, and *deepsjeng* only have 15% of branch instructions. Besides, *xalancbmk* consumes a considerable percentage of their execution time on cache and memory related operations. This explains why iCORE has a more overhead fraction on operating *xalancbmk*: executing this benchmark application requires much more memory/cache accesses than operating the other ones; however, the system with iCORE does not provide as much computation power as the original system does because it only has 7 cores to process the workload in the benchmark, and iCORE uses L2 cache lines and data bus between CPU and memory for the monitoring purpose.

We also evaluate that monitoring smaller portion of static kernel memory of the normal world, which brings the system less performance degradation as shown in Figure 4. The performance overhead of checking 12KB static memory on *xalancbmk* is 2.79% compared with that on the original system. The CPU performance increases, but not significantly, even though iCORE only checks 0.1% of the whole static kernel memory of the normal world. One possible reason is that the Linux scheduler helps the system work efficiently by checking the load balancing of each CPU dynamically and distributing the routine workload to the low CPU usage core so that the tasks can be executed as fast as possible.

As a consequence, the system with iCORE, checking even the whole static kernel memory, can still finish the same workload within the negligible performance overhead.

7 DISCUSSION AND FUTURE WORK

7.1 Response After Detection

After detecting the malicious modification in the monitored memory region of the normal world, iCORE should have corresponding measures to report and analyze the memory area. However, there exists a semantic gap that iCORE only detects the modification but has no ability to extract the semantic information such as the current CPU info, the active processes, or potential hooks on system calls. iCORE has to get a hand from memory forensic tools [38] developed to analyze and extract digital artifacts.

Memory forensic tools generally require the memory sample file, which iCORE can provide. If there are malicious modifications occurred under the monitoring of iCORE, the normal world memory can be dumped since iCORE has the ability to access the whole normal world resources. Afterwards, the dumped memory file will be sent to memory forensic tools outside the device. We can leverage one of the memory forensic tools such as Volatility [12]. The memory forensic tools, then, can analyze the memory sample and extract semantic-rich information, such as the active process list, potential hooks on system calls, and a list of opened files by the normal world OS kernel. Therefore, by means of the memory forensic tools, the semantic gap between iCORE and the normal world can be narrowed down. Also, the further analysis after detecting malicious modifications can be possible.

7.2 Potential Threats Against iCORE

iCORE is designed to detect malicious modifications on the static kernel memory area of the normal world. However several types of threats can be exploited to potentially hazard iCORE.

First, during the booting procedure of a system with iCORE, a threat that the attacker is attempting to modify the kernel image file may occur. To protect the kernel binary from maliciously tampering, we implement secure boot procedure [2] on ARM platform that is mentioned in Section 4. Hence, secure boot procedure guarantees that iCORE can be booted safely. Once iCORE is booted, the attacker cannot detect or turn off the functionalities of iCORE, because it is completely isolated from the monitored system.

Secondly, as we mention in Section 4, to load memory data, iCORE exploits `system.map` to get the starting and ending address of the static kernel memory area of the normal world. Some may concern that the addresses where the static kernel data is stored can be modified by the attackers so that iCORE cannot work properly with incorrect addresses. This is avoidable because File `system.map` is generated before the system booting. Static kernel data of the normal world will be loaded according to the file during the initialization of the system. Before the attacker takes control of the normal world, iCORE has already gained the physical addresses of the monitored memory area. Hence, the changes of the addresses do not influence the correctness of iCORE's functionalities.

Thirdly, when iCORE detects the malicious modification on the monitored memory area, the normal world memory can be dumped for further analysis. However, the normal world RAM could be tampered again by the attacker to erase the evidence of crimes. To avoid the situation mentioned above, TrustDump [41] provides an approach to allow the device user to switch into the secure world safely when the normal world OS crashes or is compromised in order to cease the operations in the normal world. Therefore, the normal world memory cannot be further tampered with and the memory dumped by iCORE can be trusted.

7.3 Extension of iCORE

Our current design of iCORE checks the static code and data in the normal world kernel memory. However, the attackers would attempt to tamper with the dynamic code to make the attacks successful. Unlike the static code and data, it is much more difficult to check the integrity of the 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 modification and a potential tampering behavior. Petroni et. al [34] propose an architecture to provide the integrity to dynamic kernel data using a specification language-based approach. Also, the code-reuse attacks are popular now which execute the existing code in the memory instead of code injection to by-pass the integrity check of security tools. In the future, we will develop iCORE to monitor the dynamic memory area and protect the normal world from the code-reuse attacks.

In addition to the integrity check improvement, iCORE will be able to overcome the semantic gap. As aforementioned, iCORE can exploit the data read from the static memory region in the normal world to check the integrity. However, the real meaning of the data iCORE reads remains unknown, so we can leverage the existing

Table 2: Comparison of the related works with iCORE.

Criteria	kGuard [18]	SIM [39]	NOVA [40]	Vigilare [29]	MIPE [7]	TZ-RKP [6]	SPROBES [13]	SecVisor [37]	iCORE
Resides out of the monitored system	○	●	●	●	●	●	●	●	●
Provides proactive monitoring	●	●	●	○	○	○	○	●	●
No need for virtualization supports	●	○	○	●	●	●	●	○	●
No need additional hardware	●	●	●	○	●	●	●	●	●
No need to modify monitored system	●	●	●	●	○	○	○	●	●
In-time tamper detection	○	○	○	○	○	○	○	○	●

●: satisfying the criteria, ○: not satisfying the criteria.

memory forensic tools to extract the semantic-rich info from the memory dump file. Nevertheless, analyzing the dump file in device is necessary because transferring the dumped memory file out of the device can be disabled by the attacker if she has the physical control of the device. The following work of iCORE will be attempting to narrow the semantic gap itself to enforce the security since the bridge between binaries and the kernel structures will help iCORE to monitor the real critical information.

8 RELATED WORK

Hardware/Hypervisor-assisted Monitoring Methods. As we mentioned in Section 2, modern monitoring methods are classified into hardware-assisted and hypervisor-assisted methods, the shortcomings and advantages of both of which have also been discussed. Other efforts have been made to expand the monitoring fields and to mitigate the negative impact brought by these two categories.

Articles [19, 24, 40] proposed another method called tiny-hypervisor. It is a thin software TCB (trusted computing base) that has a small amount of code to reduce the attack interface to monitor and protect the system, which performs well on monitoring virtual machines. However, they still require virtualization support on the target system, which is costly deployed on all mobile devices. Since most mobile and IoT devices are deploying ARM TrustZone extension, iCORE will have a better performance than the tiny-hypervisors do.

As for hardware-assisted methods that we mentioned in Section 2, Vigilare [29] and Ki-Mon [20] provided additional external equipment to protect the static and dynamic kernel objects from being tamped, respectively. Davi et. al [10] designed a hardware-based control flow integrity defense for embedded systems against return-oriented programming (ROP) attacks. Based on this work, HAFIX [9] is proposed as a defensive extension against ROP attacks using backward edges. However, it is costly and inconvenient to take another device dedicated to real-time protection and monitoring purposes along with the mobile devices. To avoid using additional equipment, TZ-RKP [6] and SPROBES [13] take advantage of ARM TrustZone extension to trap the page table updates, switch to the secure world through `smc` instruction, and verify the write signal on the monitored memory area. MIPE [7] does the similar work to protect the memory of the normal world. It checks in the secure world if the previous status of a physical address that the normal world maps when a page fault occurs has already been mapped to a virtual address. These methods still require operations in the normal world to switch to the secure world, which slows the execution down and can be disabled by the compromised normal world. iCORE provides a possibility to the research on developing security tools based on ARM TrustZone that the monitoring tool in the secure world can struggle out of control of the normal world with self-decision-making power so that the attackers in the normal world cannot disable its functionalities.

Continuous Monitoring Mechanisms. Current continuous monitoring mechanisms exploit trampolines, which are used to store addresses pointing to interrupt service routines, or hooks pre-installed in the monitored system to change the control flow of the existing execution path to the monitoring functionalities. This requires that the hooks and trampolines should be properly protected from tampered so that the monitoring mechanism can keep their integrity and availability. Articles [33, 39] implement trampolines that are guaranteed by a hypervisor level memory protection approach to monitor the VM.

From another perspective, ARM TrustZone is also exploited to deploy the continuous monitoring. Mechanisms such as TIMA (TrustZone Integrity Measurement Architecture) [35, 45] implement the monitoring code in TrustZone and implant the trampoline in the secure world kernel to enhance the integrity of the static memory block in the normal world. The purpose of iCORE is also to provide a secure environment based on ARM TrustZone for normal world OS. However, iCORE is more effective because it does not require additional modifications on the normal world OS source code and can get access to all the normal world memory with high visibility due to its highest privilege state. Thus, iCORE is an improvement of the traditional continuous monitoring mechanism.

9 CONCLUSION

In conclusion, we present iCORE, an innovative continuous and proactive extrospection system with high visibility technique on IoT devices deploying multi-core ARM platform in this paper. iCORE exploits ARM TrustZone technology to dedicate one core in the secure world forever, assuring the computing integrity of static kernel memory region of the normal world. By breaking the original time-sharing paradigm of such systems, iCORE enables continuous co-processor-like monitoring with high visibility into the rich execution environment on such mobile and IoT platforms based on the design of ARM TrustZone architecture that the secure world can access all the resources in the normal world. iCORE plays a role as master and monitors the pre-selected memory area proactively so that the normal world cannot detect or disable the functionalities of it because iCORE gets out of control of the normal world since the system booting procedure. And by ensuring that security 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 the evaluation, iCORE only introduces a negligible overhead.

ACKNOWLEDGEMENT

This material is based upon work supported in part by Samsung Research, Samsung Electronics, the Center for Cybersecurity and Digital Forensics at Arizona State University, the Global Research Laboratory Program through the National Research Foundation of Korea funded by the Ministry of Science, ICT under Grant NRF-2014K1A1A2043029, and the National Science Foundation under Grant No. 1642031.

REFERENCES

[1] AMD. 2005. Secure Virtual Machine Architecture Reference Manual. https://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf.

[2] ARM. 2009. ARM Security Technology Building a Secure System using TrustZone Technology. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html.

[3] ARM. 2015. ARM Cortex-A Series Programmer's Guide for ARMv8-A. http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/index.html.

[4] ARM. 2016. SMC CALLING CONVENTION System Software on ARM Platforms. http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM$_{D}$EN0028B$_{S}$MC$_{C}$alling$_{C}$onvention.pdf.

[5] ARM. 2017. ARM Trusted Firmware. https://github.com/ARM-software/arm-trusted-firmware.

[6] 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 21st ACM Conference on Computer and Communications Security (CCS)*. Scottsdale, AZ, 90–102.

[7] Rui Chang, Liehui Jiang, Wenzhi Chen, Yang Xiang, Yuxia Cheng, and Abdulhameed Alelaiwi. 2017. MIPE: a practical memory integrity protection method in a trusted execution environment. *Cluster Computing* 20, 2 (2017), 1075–1087.

[8] P Daniel, Cesati Marco, et al. 2007. Understanding the Linux kernel.

[9] Lucas Davi, Matthias Hanreich, Debayan Paul, Ahmad-Reza Sadeghi, Patrick Koeberl, Dean Sullivan, Orlando Arias, and Yier Jin. 2015. HAFIX: Hardware-assisted flow integrity extension. In *Proceedings of the 52nd Annual Design Automation Conference*. San Francisco, CA.

[10] Lucas Davi, Patrick Koeberl, and Ahmad-Reza Sadeghi. 2014. Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation. In *Proceedings of the 51st Annual Design Automation Conference*. San Francisco, CA.

[11] World Economic Forum. 2018. The Global Risks Report 2018, 13th Edition. http://www3.weforum.org/docs/WEF$_{G}$RR18$_{R}$eport.pdf.

[12] VOLATILITY FOUNDATION. 2017. Volatility Framework - Volatile memory extraction utility framework. https://github.com/volatilityfoundation/volatility.

[13] Xinyang Ge, Hayawardh Vijayakumar, and Trent Jaeger. 2014. SPROBES: Enforcing kernel code integrity on the trustzone architecture. In *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*. San Jose, CA.

[14] GlobalPlatform. 2016. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide. http://www.globalplatform.org/mediaguidetee.asp.

[15] Intel. 2014. Intel Trusted Execution Technology (Intel TXT). https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html.

[16] Xuxian Jiang and Xinyuan Wang. 2007. Out-of-the-box Monitoring of VM-based High-Interaction Honeypots. In *Proceedings of the 10th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Queensland, Australia, 198–218.

[17] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. 2007. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*. 128–138.

[18] Vasileios P Kemerlis, Georgios Portokalidis, and Angelos D Keromytis. [n. d.]. kGuard: Lightweight Kernel Protection against Return-to-User Attacks.. In *Proceedings of the 21st USENIX Security Symposium (Security)*.

[19] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. 2011. L4Android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*. Chicago, IL, 39–50.

[20] 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).

[21] Anthony Lineberry. 2009. Malicious Code Injection via/dev/mem. *Black Hat Europe* (2009), 11.

[22] Lionel Litty, H Andrés Lagar-Cavilla, and David Lie. 2008. Hypervisor Support for Identifying Covertly Executing Binaries.. In *Proceedings of the 17th USENIX Security Symposium (Security)*. Boston, MA, 243–258.

[23] Teresa F Lunt and R Jagannathan. 1988. A prototype real-time intrusion-detection expert system. In *Proceedings of the 9th IEEE Symposium on Security and Privacy (Oakland)*. Oakland, CA, 59–66.

[24] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*. Glasgow, Scotland UK, 315–328.

[25] MITRE. 2017. CVE-2017-15589 Detail. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15589.

[26] MITRE. 2017. CVE-2017-7228 Detail. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7228.

[27] MITRE. 2018. CVE-2018-1068 Detail. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1068.

[28] MITRE. 2018. CVE-2018-7542 Detail. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7542.

[29] 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 19th ACM Conference on Computer and Communications Security (CCS)*. Raleigh, NC, 28–37.

[30] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. 2016. Trustzone explained: Architectural features and use cases. In *Proceedings of the IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*. Pittsburgh, PA, 445–451.

[31] OP-TEE. 2018. OP-TEE Trusted OS Documentation. https://www.op-tee.org/.

[32] Reena Panda, Shuang Song, Joseph Dean, and Lizy K John. 2018. Wait of a Decade: Did SPEC CPU 2017 Broaden the Performance Horizon?. In *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. Vienna, Austria, 271–282.

[33] Bryan D Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. 2008. Lares: An architecture for secure active monitoring using virtualization. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (Oakland)*. Oakland, CA, 233–247.

[34] Nick L Petroni Jr, Timothy Fraser, AAron Walters, and William A Arbaugh. 2006. An Architecture for Specification-Based Detection of Semantic Integrity Violations in Kernel Dynamic Data.. In *Proceedings of the 15th USENIX Security Symposium (Security)*. Vancouver, Canada, 289–304.

[35] Daniel Plastina, Jonathan Cain, and Michael Novak. 2005. Methods, systems, and computer-readable media for generating an ordered list of one or more media items. US Patent App. 11/089,696.

[36] Mendel Rosenblum and Tal Garfinkel. 2005. Virtual machine monitors: Current technology and future trends. *Computer* 38, 5 (2005), 39–47.

[37] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. 2007. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*. Stevenson, WA, 335–350.

[38] Pavitra Shankdhar. 2018. 22 Popular Computer Forensics Tools [Updated for 2018]. https://resources.infosecinstitute.com/computer-forensics-tools/#gref.

[39] Monirul I Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. 2009. Secure in-vm monitoring using hardware virtualization. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*. Chicago, IL, 477–487.

[40] Udo Steinberg and Bernhard Kauer. 2010. NOVA: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*. ACM, 209–222.

[41] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Sushil Jajodia. 2014. Trustdump: Reliable memory acquisition on smartphones. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*. Wroclaw, Poland, 202–218.

[42] Arijit Ukil, Jaydip Sen, and Sripad Koilakonda. 2011. Embedded security for Internet of Things. In *Proceedings of the 2nd National Conference on Emerging Trends and Applications in Computer Science*. Shillong, India.

[43] USMAN. 2013. Apple's Secure Enclave for Touch ID And Its Importance Detailed. (2013). http://www.iphoneincanada.ca/iphone-5s/apples-new-secure-enclave-details/.

[44] Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. 2009. Countering kernel rootkits with lightweight hook protection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*. Chicago, IL, 545–554.

[45] White Paper: An Overview of the Samsung Knox Platform. 2016. Samsung Knox. https://kp-cdn.samsungknox.com/df4184593021d7b8fabfdfeff5c318ba.pdf.