Journal of Systems Architecture 119 (2021) 102240

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes

Zhen Ling [a,*], Huaiyu Yan [a], Xinhui Shao [a], Junzhou Luo [a], Yiling Xu [b], Bryan Pearson [c], Xinwen Fu [d]

[a] Southeast University, Nanjing, China
[b] Alibaba Group, Hangzhou, China
[c] University of Central Florida, Orlando, FL, USA
[d] University of Massachusetts Lowell, Lowell, MA, USA

ARTICLE INFO

Keywords:
Internet of Things
Integrity
TrustZone

ABSTRACT

With the extensive application of IoT techniques, IoT devices have become ubiquitous in daily lives. Meanwhile, attacks against IoT devices have emerged to compromise IoT devices by tampering with system pre-installed programs or injecting new malware. To mitigate these attacks, integrity enforcement of IoT systems has been proposed. The integrity of an IoT device system includes load-time integrity and runtime integrity. In this paper, we design an IoT system based on ARM TrustZone to enforce the system integrity. First, we establish the root of trust and propose a hybrid booting approach consisting of both secure boot and trusted boot to enforce the system load-time integrity. Second, we investigate a paging-based process integrity measurement method to measure the NW processes and conduct remote attestation based on the measurement results ensuring the NW runtime process integrity. We implement an IoT prototype system on a NXP i.MX6Q SABRE SD development board to assess its feasibility. Real-world experiment results demonstrate that our prototype introduces negligible performance overhead to the original system.

1. Introduction

The widespread usage of smart devices in various industries and fields brings a new era of Internet of Things (IoT). It is estimated that a total number of 11.7 billion IoT devices are actively connected to the Internet at the end of 2020, occupying 54% of overall online devices, and 30 billion IoT connections are expected by 2025 [1]. The global IoT market size has reached $250 billion in 2019 and is predicted to reach $1463 billion by 2027 [2]. However, despite of the rapid growth of IoT device number and market size, security has been overlooked due to the lagging IoT security standards, inadequate investment in security development as well as the lack of security awareness.

In recent years, extensive research efforts have been conducted to attacks against IoT devices, including hardware attacks, operating system (OS)/firmware attacks, and software attacks. (1) Hardware attack: For IoT devices deployed in public places, such as surveillance cameras, attackers can have physical access to them and leverage hardware interfaces like universal asynchronous receiver/transmitter (UART) and joint test action group (JTAG) interfaces to illegally tamper with the internal IoT system [3–5]. (2) OS/Firmware attack: The operating system image or firmware are usually stored in flash memory for IoT devices and can be updated through network. The contents of a flash memory can be maliciously modified through hardware attacks and a malicious firmware image can be used for updating [6]. (3) Software attack: Software vulnerabilities of IoT devices, like stack overflow, command injection, etc. [7], can be leveraged to inject malware or maliciously modify existing programs. All of these attacks involve tampering with IoT system software, thus damaging the integrity of the original IoT system.

To mitigate such attacks and enforce system integrity, some research works have leveraged virtualization techniques to conduct system runtime execution monitoring (REM) [8,9]. These solutions rely on virtual machine monitors (VMM), namely hypervisors, that may contain vulnerabilities due to their large codebase, and can be maliciously modified before system boot. Meanwhile, the performance overhead introduced by the virtualization-based REM is intolerable for low-cost IoT devices.

Compared to x86 instruction set architecture, ARM, with virtues of energy efficiency, is more suitable for low-cost IoT devices and has

* Corresponding author.
E-mail addresses: zhenling@seu.edu.cn (Z. Ling), huaiyu_yan@seu.edu.cn (H. Yan), xinhuishao@seu.edu.cn (X. Shao), jluo@seu.edu.cn (J. Luo), yiling.xyl@alibaba-inc.com (Y. Xu), bpearson@knights.ucf.edu (B. Pearson), xinwen_fu@uml.edu (X. Fu).

https://doi.org/10.1016/j.sysarc.2021.102240
Received 30 March 2021; Received in revised form 7 July 2021; Accepted 9 July 2021
Available online 15 July 2021

dominated the embedded system market, especially the mobile market [10]. Additionally, recent ARM processors provide a system-level security solution called TrustZone [11], which provides system-level isolation by dividing both system hardware and software resources into two domains, namely the Secure World (SW) and the Normal World (NW). The SW is more privileged and can be leveraged to conduct REM on the NW.

In order to enforce a strong system integrity policy, we leverage ARM TrustZone technology to ensure both the load-time integrity and the runtime integrity of the IoT system.

To enforce load-time integrity, we first establish the root of trust (RoT) based on the OCROM and eFuse. Then, we propose a hybrid booting approach consisting of the secure boot of the SW and the trusted boot of the NW. The secure boot involves establishing a chain of trust (CoT) initiated from the RoT for the SW boot images to ensure the SW load-time integrity, while the trusted boot involves measurements of the NW boot images and a remote attestation is conducted to verify the NW load-time integrity.

On such basis, we investigate a paging-based process integrity measurement and attestation method to monitor the NW status from SW. A periodical measurement is conducted inside the SW on the code segments of each NW process and the measurement results are sent to a remote attestation server. The NW runtime process integrity is verified if the received measurement results match with some pre-calculated reference values.

We implement a prototype system on a Freescale i.MX6Q SABRE SD development board [12] and evaluate its effectiveness against all these attacks. According to experimental results, our system introduces negligible performance overhead to the original IoT system.

In summary, our contributions in this paper are listed as follows:

- We propose a hybrid booting approach based on ARM TrustZone technology to enforce system load-time integrity.
- We investigate a paging-based process integrity measurement and attestation method to enforce runtime process integrity.
- We implement a prototype system on a Freescale i.MX6Q SABRE SD development board. Extensive empirical experiment results demonstrate that our system can effectively defend and detect different IoT attacks with little performance overhead.

The rest of this paper is organized as follows. Section 2 provides the necessary background information on TrustZone, secure boot and trusted boot. The system overview is presented in Section 3. The details of the hybrid booting approach and the paging-based process integrity measurement and attestation method are introduced in Sections 4 and 5, respectively. Section 6 evaluates the system effectiveness and performance overhead. Related work is reviewed in Section 8. Finally, we conclude this paper in Section 9.

2. Background

2.1. TrustZone overview

The ARM TrustZone technology [13] is a system-level security extension to the ARM architecture since ARMv6. TrustZone divides the system into two domains, Secure World (SW) and the Normal World (NW), and enforces strong isolation between these worlds in terms of both hardware and software resources.

TrustZone leverages dedicated hardware components to enforce hardware resource isolation. An additional processor bit, Non-Secure (NS) bit, indicates the current CPU state and is propagated through the Advanced eXtensible Interface (AXI) system bus to the peripherals and the memory. A peripheral can be configured as secure or non-secure using TrustZone Protection Controller (TZPC), and a secure peripheral can only be accessed by the SW when NS bit is cleared. Additionally, the physical memory is separated into two isolated parts, i.e., the secure memory and the non-secure memory, via TrustZone Address Space Controller (TZASC). The secure memory can only be accessed by SW and any attempted access from NW is blocked, while the non-secure memory can be accessed from both worlds. The switch between these two worlds is accomplished via a Secure Monitor Call (SMC) instruction.

Based on the hardware isolation mechanism provided by TrustZone, both SW and NW run separated software suites, including different operating systems and user-level applications. Generally, a rich OS and client applications (CA) run in the NW while a secure OS and trusted applications (TA) run in the SW. Programs in the SW have full access to all system resources while programs in the NW can only access NW resources but not those belonging to the SW. Therefore, security critical tasks are often deployed inside the SW to be protected from an insecure NW.

2.2. Secure boot and trusted boot

Secure boot is a mechanism that establishes a Chain of Trust (CoT) on all system boot images. Secure boot relies on the public key cryptography to verify image signatures before their execution [14]. A pair of public and private key is generated for image signing and verification. The private key is used to sign an image offline while the public key is used to verify the image signature before one image is executed. The whole secure boot process usually involves several images. The image of the former boot stage verifies the image of the next boot stage, which in turn forms a verification chain, known as the CoT. During the secure boot, a single signature verification failure can terminate the whole system booting process.

As for trusted boot, all system boot images are measured in each boot stage [15]. The measurement results are accumulated to generate a measurement list which uniquely identifies the particular firmware images executed so far. The measurement list can be used for attestation. During trusted boot, an attestation failure will not terminate the system, but the user may be alerted via a smart app.

Both secure boot and trusted boot anchor their trust on a root of trust (RoT), which is inherently trusted. Therefore, the RoT is usually established based on some invariable storage media whose content cannot be modified once programmed.

3. System overview

In this section, we present the threat model and the basic idea of the system design.

3.1. Threat model

We assume that attackers have physical access to IoT devices. They can launch hardware attacks [3–5], OS/firmware attacks [6] and software attacks [7] against IoT devices. Before the IoT devices are powered up, the attackers can tamper with the firmware images of both the SW and the NW stored in the flash memory. During system runtime, the attackers can inject malware in the NW and tamper with NW built-in programs arbitrarily. Sophisticated hardware attacks like bus snooping attacks [16], cold boot attacks [17] and cache side channel attacks [18] are out of the scope of this paper. We only consider the security of the code section of a program, i.e., *.text*.

We assume that the program in ROM is secure since the On-Chip ROM (OCROM) is read-only and difficult to tamper with. We also assume that the attackers cannot compromise the run-time SW; therefore, SW code is secure from software attacks. Finally, we assume that the remote attestation server is secure and trustworthy.

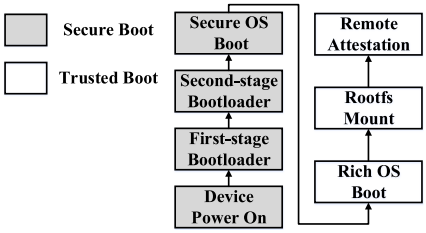


Fig. 1. The hybrid booting sequence.

3.2. System design

We propose a hybrid secure and trust booting method and a process integrity measurement and attestation method to ensure the system load-time integrity and run-time process integrity, respectively.

The hybrid booting procedure is comprised of the secure boot of the SW and the trusted boot of the NW. Fig. 1 illustrates the hybrid booting sequence. On powering up, the first-stage bootloader starts to run first. It loads the second-stage bootloader into memory, verifies its integrity and transfers control to it after a successful signature verification. Then the second-stage bootloader loads the rest firmware images, namely the secure OS kernel image, the rich OS kernel image and the filesystem image, into the memory and verifies the secure OS kernel image so as to enforce the load-time integrity of the SW. Due to secure boot, the integrity of the secure OS kernel is verified. Therefore, the secure OS kernel can be treated as the trusted base for the trusted boot of the NW. During trusted boot, the secure OS kernel measures both the rich OS kernel image and filesystem image and then transfers control to the rich OS. After the rich OS starts, the measurement results are sent to the remote attestation server to verify NW load-time integrity.

After the hybrid boot, programs running in the NW provides IoT device's functionalities. We implement a monitoring module in the secure OS to measure the memory pages of the code segments of processes in the rich OS periodically. After being encrypted with a remote attestation key, the measurement results are sent to the remote attestation server to verify the integrity of run-time NW processes. In this paper, this design is targeted to Linux-based IoT systems based on TrustZone. However, the proposed technique can be revised to apply to all kinds of IoT systems.

4. Hybrid booting approach

In this section, we first present the design of RoT and then elaborate on the secure boot for the SW and the trusted boot for the NW.

4.1. Root of trust

As the trusted base for the hybrid boot, the RoT is first established based on the OCROM and eFuse. The OCROM is a read-only memory with write-protection, thus it is difficult to tamper with the codes stored inside. In addition, the on-chip eFuse is a one-time-programmable (OTP) electronic element, whose contents cannot be modified once programmed. Therefore, the OCROM and the eFuse are leveraged as the system RoT since both are immune to being tampered with. The first-stage bootloader is stored in the OCROM, responsible for verifying the integrity of the second-stage bootloader using a public key. The hash of the public key is stored in the eFuse and used to verify the public key's integrity.

4.2. Secure boot

Secure boot is used to boot up the SW to ensure the integrity of the SW. The secure boot for the SW involves two phases, the offline image signing phase and the online secure boot phase.

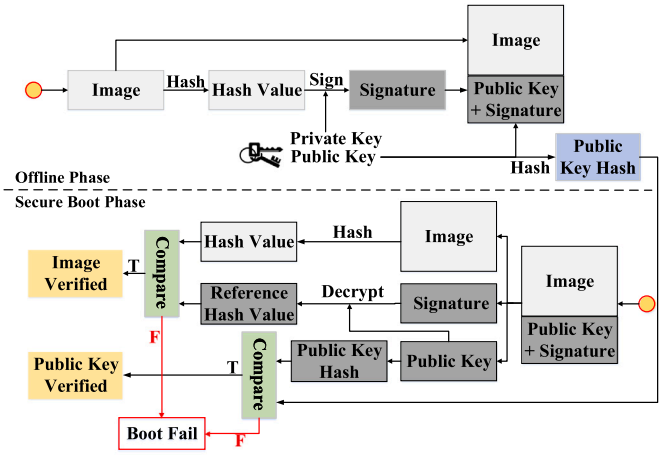


Fig. 2. Secure boot processes.

4.2.1. Offline phase

The second-stage bootloader image and the secure OS kernel image are measured and signed offline, as shown in the upper half of Fig. 2. A hash of the second-stage bootloader image is calculated and used as its measurement result. The private key of the second-stage bootloader PR_1 is used to sign the measurement result and the hash of the corresponding public key PU_1 is stored in the eFuse. The second-stage bootloader image, as well as PU_1 and the signature, is stored in the flash memory. Additionally, the secure OS kernel image is measured and signed in a similar way using another private key PR_2. The secure OS kernel image, as well as its public key PU_2 and signature, is stored in the flash memory while the hash of PU_2 is stored in the second-stage bootloader.

4.2.2. Secure boot phase

A CoT can be established based on the first-stage bootloader. On powering up, the first-stage bootloader acts as the trusted base of the secure boot. It passes the control to the second-stage bootloader after successfully loading and verifying the integrity of the second-stage bootloader, as shown in the lower half of Fig. 2. Then, the second-stage bootloader verifies the integrity of the secure OS kernel and attestation CAs control to the secure OS kernel if the verification succeeds. A single verification failure can terminate the secure boot process and in turn the system aborts.

Let P_0 be the first-stage bootloader, P_1 be the second-stage bootloader and P_2 be the secure OS kernel. The steps that P_{i-1} verifies P_i are shown as follows.

(1) P_{i-1} locates P_i, as well as the attached public key and signature of P_i in the memory using the parameters passed to it.
(2) P_{i-1} calculates the hash of the public key, and compares the resulting hash with the one it possesses. In particular, the first-stage bootloader uses the public key hash stored in the eFuse to verify the public key of the second-stage bootloader. The booting process terminates if there is a mismatch.
(3) P_{i-1} restores the measurement result m from the signature using the public key.
(4) P_{i-1} makes a fresh hash calculation on P_i to obtain a new measurement m' and compares it with m. P_i starts its execution if m' matches m, otherwise, the system terminates.

Once the chain of trust is established, the programs in the SW can be trusted after a successful secure boot. Additionally, the isolation mechanism provided by TrustZone ensures that programs in the NW have no access to SW resources. Thus, the SW can be treated as the trusted base for the trusted boot and provide necessary secure storage used by the trusted boot.

Fig. 3. CAAM blob structure.

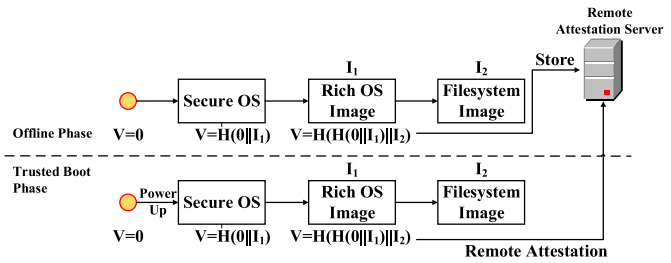


Fig. 4. Trusted boot processes.

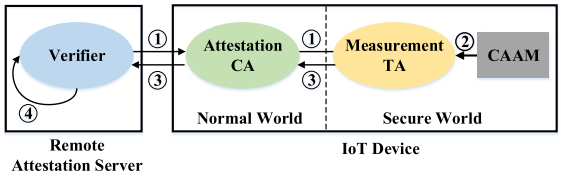


Fig. 5. Remote attestation process.

4.3. Trusted boot

After the secure OS kernel gets started, trusted boot is used to boot up the NW to ensure its integrity. The trusted boot for the NW involves two phases: the offline hash chain calculation phase, and the online trusted boot phase. Furthermore, the remote attestation key needs to be securely stored in the flash memory.

4.3.1. Offline phase

We design a hash chain to measure the NW images, as shown in the upper half of Fig. 4. The initial hash value is set to 0 ($V = 0$). The hash value is updated by concatenating the current value V and next image I on the chain, $V = Hash(V \| I)$. The NW consists of two images: the rich OS kernel image and the file system image. Therefore, the final value of the hash chain is calculated as $V = Hash(Hash(0 \| I_1) \| I_2)$ and it is stored in the remote attestation server as the reference value for NW integrity verification.

The final hash value V is encrypted with the remote attestation key before being sent to the remote attestation server for NW integrity verification. The remote attestation key is a symmetric encryption key and is generated offline. Both the remote attestation server and the IoT device have a copy of this key. Since the remote server is trusted, we only consider the secure storage of the attestation key in the local IoT device.

We design the secure key storage based on the Cryptographic Acceleration and Assurance Module (CAAM) module of the i.MX6Q development board used in our design. The CAAM provides a Blob mechanism to protect secret data across system power cycles. To this end, an on-chip Secure Non-Volatile Storage (SNVS) can be used to provide a 256 bit Master Key (MK) for CAAM, and a Random Number Generator (RNG) inside CAAM is used as a 256-bit blob key.

A common CAAM Blob structure consists of an encrypted Blob key, the encrypted remote attestation key and a Message Authentication Code (MAC) tag, as shown in Fig. 3. Since the remote attestation key is encrypted, the Blob can be securely stored in the flash memory. The steps to produce a Blob are explained as follows:

(1) A 256-bit random Blob Key is generated using the RNG.
(2) The remote attestation key is encrypted using AES-CCM with the Blob Key. A MAC of the remote attestation key is also calculated and appended to the encrypted attestation key to ensure its integrity.
(3) CAAM derives a 256-bit Blob Key Encryption Key (BKEK) using the MK and employs it to encrypt the Blob Key, generating the encrypted Blob Key.
(4) The Blob is generated by concatenating these three components and finally stored in the flash memory.

In order to prevent the NW from obtaining the remote attestation key, the Central Security Unit (CSU) is leveraged to configure CAAM as a secure peripheral. Accordingly, the CAAM is accessible only to the SW and any access attempt from the NW will be blocked. In addition, the SNVS is a secure peripheral by default. Since the remote attestation key can only be restored and accessed in the SW, its integrity is enforced.

4.3.2. Trusted boot phase

During the trusted boot phase, the secure OS kernel measures the NW images to obtain the final hash value and starts the NW. The secure OS kernel measures the rich OS image and the file system image using the same offline method, as shown in the lower half of Fig. 4. After the NW is fully booted, the final hash value is encrypted using the remote attestation key and sent to the remote attestation server for NW load-time integrity verification. The remote attestation key is restored from the CAAM blob and used to encrypt the final hash value in the SW. The encrypted hash value is sent to the remote attestation server through the NW. There are three components involved in the remote attestation process, including a measurement TA, an attestation CA and a remote verifier. The remote attestation process shown in Fig. 5 is illustrated in details as follows:

(1) The NW attestation CA establishes a TLS connection with the remote verifier and requests a $Nonce$ which is randomly generated to resist replay attacks. The attestation CA passes the received $Nonce$ to the measurement TA through shared memory.
(2) The measurement TA leverages CAAM to restore the remote attestation key K from the Blob. First, the CAAM generates the BKEK using the MK. Then, the Blob Key is recovered by decrypting the encrypted Blob Key via the BKEK. Finally, the CAAM decrypts the encrypted remote attestation key with the Blob Key to recover the remote attestation key K, and uses the MAC to verify its integrity. The measurement TA saves K in the SW and the isolation mechanism provided by TrustZone ensures that the NW has no access to it. The final hash value V and the $Nonce$ are encrypted with K using AES-128-CBC, i.e., $E = AES-128-CBC(Nonce \| V, K)$.
(3) The measurement TA sends E to the attestation CA through shared memory and the attestation CA sends E to the remote verifier through the TLS connection.
(4) The verifier uses the local stored hash value V', $Nonce'$, and remote attestation key K' to verify the integrity of the NW. After decrypting E with K' to obtain the final hash V sent from the IoT device, the Verifier compares V to V' and $Nonce$ to $Nonce'$. If both match, the NW load-time integrity is verified. If the verification of the final hash value V fails, the NW integrity is damaged. If the verification of $Nonce$ fails, the IoT device is under replay attacks.

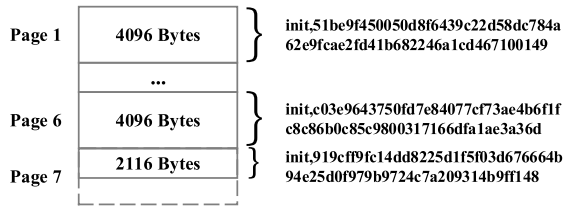


Fig. 6. Measurement results of the *init* code segment.

5. Paging-based process integrity measurement and attestation method

In this section, we present the basic idea of process integrity measurement. Then we elaborate on the process integrity measurement method and the process integrity remote attestation method.

5.1. Basic idea

We propose a paging-based process integrity measurement and attestation method to ensure the integrity of NW processes at the runtime. Recall that the SW is trusted at the load-time and runtime as the secure boot of the SW and TrustZone hardware isolation techniques are applied. However, NW programs are still untrusted at the runtime despite of the NW trusted boot, since the NW may be invaded by attackers who can inject malware or tamper with built-in programs in the NW. Note that, different from PC environment, after an IoT device is deployed, the system always executes the same set of pre-installed programs instead of installing new programs on user demand. As a result, we can perform offline measurement on the code sections of the pre-installed programs at the page granularity level in the NW and store measurement results as reference values on the remote attestation server. Then we measure the code segments of the runtime processes residing on the memory page using a measurement TA in the SW, and finally send the results to the remote attestation server to verify the integrity of processes.

5.2. Offline program measurement

All of the program code are stored in the *.text* section of the corresponding ELF files of the programs. However, the code is loaded and run in the memory in terms of the paging mechanism. Therefore, the *.text* sections of all NW programs can be divided into several segments in terms of a page size (i.e., 4 KB) offline. The hash values of each segment are calculated and saved on the attestation server as the reference values to verify the integrity of NW processes.

We take the first user-level process (i.e., *init*) as an example. The size of its code segment is of 0x6844 bytes, occupying 7 pages in terms of a 4 KB page size. The last part that cannot occupy one full page is handled in accordance with its actual size. A SHA256 hash of each page is calculated, generating 7 structures of $\{processname, pagehash\}$, as shown in Fig. 6. The $\{processname, pagehash\}$ structures of all NW ELF files are calculated and saved in a hash table on the attestation server as the reference for process integrity verification.

5.3. Runtime process integrity measurement

The SW measurement TA measures the code segment of each process periodically in the memory. After encrypting the measurement results with the remote attestation key, the measurement TA sends them to the attestation server that verifies the runtime process integrity.

In Linux, processes are managed using the process descriptor *task_struct* shown in Fig. 7. Each *task_struct* manages one process and contains all information of that process, including process ID,

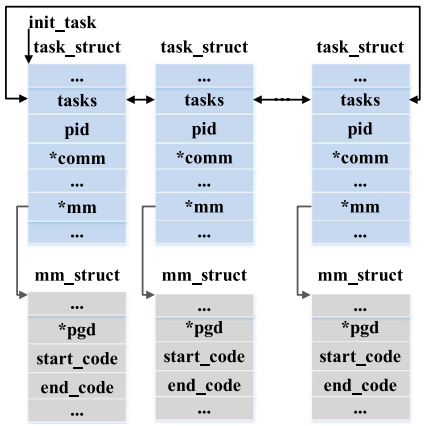


Fig. 7. Linux *task_struct* and *mm_struct*.

process name, address space, etc. All *task_structs* are organized as a doubly-linked list by the field *tasks*. The virtual address of process 0's *task_struct*, named *init_task*, is stored in the kernel symbol table file, i.e., *System.map*. Starting from *init_task*, all *task_structs* can be traversed and the information of all processes can be collected. The field *mm* of *task_struct* points to a memory descriptor *mm_struct* which is used to manage the virtual address space of a process. The fields *start_code* and *end_code* describe the starting and ending address of the process code segment respectively and can be used to locate the code segment of a process in the memory.

Since the SW and NW have different virtual memory address spaces, the NW virtual addresses should be translated into physical addresses. Then these physical addresses are mapped to SW virtual addresses. Linux divides a process' virtual address space into two parts, i.e, the kernel space and the user space. The kernel space uses the linear address translation method. There is a fixed interval $va2pa\_offset$ between a kernel space virtual address *va_kernel* and its corresponding physical address *pa_kernel*, as shown in Eq. (1).

$$pa\_kernel = va\_kernel - va2pa\_offset \tag{1}$$

The user space conducts address translation using paging. The field *pgd* of *mm_struct* points to the base address of the page table. A user space virtual address *va_user* can be translated to its corresponding physical address *pa_user* through page table walk *page_table_walk*, as shown in Eq. (2).

$$pa\_user = page\_table\_walk(pgd, va\_user) \tag{2}$$

The secure OS uses one-level paging structure to manage the SW memory space. A physical address *pa* is mapped into the SW virtual address space using Eq. (3).

$$va = page\_table(pa) \tag{3}$$

The SW measurement TA traverses the code segments of all NW processes starting from *init_task*. Since both *task_struct* and *mm_struct* are in the NW kernel space, the measurement TA obtains the physical address of *init_task* according to Eq. (1) and maps the physical address into the SW virtual address space using Eq. (3) to parse the structure of process 0's *task_struct*. The *tasks* field of process 0's *task_struct* points to that of process 1's, i.e., the *init* process. Afterwards, the physical address corresponding to the *tasks* field of *init*'s *task_struct* can be obtained according to Eq. (1). Then, the physical address of *init*'s *task_struct* is calculated based on the *tasks*'s offset inside *task_struct*.

After obtaining the physical address of *init*'s *task_struct*, the measurement TA reads its code segment as shown in Fig. 8, and the steps are as follows:

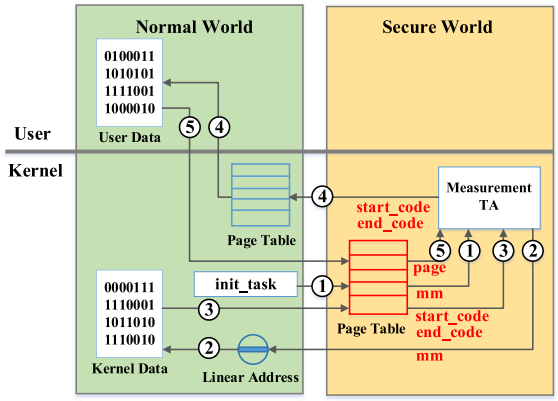


Fig. 8. Measuring pages of NW process code segments.

(1) The measurement TA maps the physical address of *init*'s *task_struct* into the SW virtual address space using Eq. (3) and obtains the *mm* field in accordance with the structure of *task_struct*. Note that the obtained *mm* contains the NW virtual address of *init*'s *mm_struct*.
(2) The measurement TA obtains the physical address of *init*'s *mm_struct* according to Eq. (1).
(3) The physical address of *init*'s *mm_struct* is transformed to the corresponding SW virtual address according to Eq. (3). The measurement TA obtains *start_code*, *end_code*, and *pgd* from *mm_struct*. Note that the obtained pointers also contain NW virtual addresses.
(4) According to *start_code* and *end_code* as well as the page size, the measurement TA calculates the number of pages the *init* process's code segment occupies and the starting NW virtual address of each page. The physical address of *init*'s page table can be located using *pgd*. Since the code segment of *init* is in the NW user space, the virtual address of each page is transformed to its corresponding physical address according to Eq. (2). Besides, the measurement TA determines whether a page currently resides in the physical memory according to the *Present* bit of its corresponding page table entry.
(5) The physical address of each page is transformed to the corresponding SW virtual address according to Eq. (3) and the measurement TA reads and measures the content of each page.

The SHA256 hash values of pages in the memory are calculated and concatenated to form a measurement result M of one process. The format of the result M is "##process name##number of page hashes##page hash 1, page hash 2, page hash 3, ..., page hash n".

5.4. Process integrity attestation

The NW process measurement results generated by the SW measurement TA are used as the attestation information and forwarded to the remote verifier by the NW attestation CA for NW process integrity remote attestation, as shown in Fig. 9. We design a protocol for the remote attestation. The detailed workflow is illustrated as follows.

(1) The IoT device requests a $Nonce$ from the remote verifier. After establishing a TLS connection to the remote verifier, the NW attestation CA requests a $Nonce$ from the verifier and passes it to the SW measurement TA. The measurement TA makes a secure copy of the $Nonce$ in the SW memory.
(2) The measurement TA reads the memory pages of the ith NW process' code segment and calculates its measurement M_i.

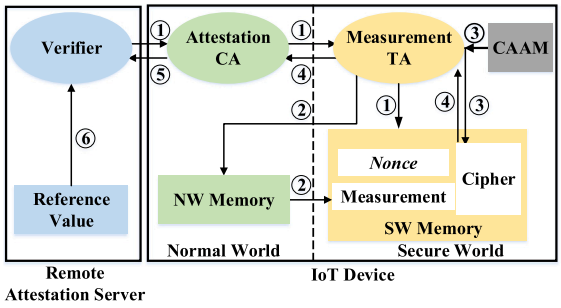


Fig. 9. NW process integrity remote attestation.

(3) The measurement TA encrypts the attestation information. M_i and $Nonce$ are encrypted with the remote attestation key K used in the trusted boot phase. Then we can obtain the ciphertext $E_i = AES-128-CBC(Nonce \| M_i, K)$.
(4) The measurement TA passes the ciphertext E_i to the attestation CA. The measurement TA obtains the next *task_struct* and if it corresponds to process 0, the measurement TA will execute step 5, otherwise jump back to step 2.
(5) The attestation CA sends the ciphertext set $E = \{E_1, E_2, \ldots, E_n\}$ to the verifier through the TLS connection.
(6) The verifier uses the remote attestation key to decrypt the ciphertext set $E = \{E_1, E_2, \ldots, E_n\}$, and obtains $Nonce$ and the measurement result M_i of each process. After the $Nonce$ is verified successfully, the verifier restructures M_i to a series of $\{processname, pagehash\}$ pairs. Each $\{processname, pagehash\}$ pair is searched in the hash table generated offline. If one pair can be found, the integrity of the corresponding page is verified. Otherwise, the verifier tries to search for the process name in the hash table. If the process name is in the hash table, it indicates that a pre-installed program has been tampered with, and if there is no match for the process name, it indicates that a new malicious process exists. Only if all pages are verified successfully, the runtime NW process integrity verification passes.

6. Evaluation

In this section, we first present the experimental setup, and then evaluate the effectiveness and performance of our system.

6.1. Experimental setup

We implement a trusted air quality monitoring prototype on a Freescale i.MX6Q SABRE SD development board, as shown in Fig. 10. The prototype senses the surrounding air quality status, i.e., particulate matter (PM2.5) levels, and acts as an MQTT client to publish the resulting statistics to an MQTT broker running on a remote server, as shown in Fig. 11. We implement and deploy the remote server in a PC running Ubuntu 14.04 LTS and develop an Android app to subscribe the air quality statistics from the server.

Besides, a secure boot module and a trusted boot module are implemented in the second-stage bootloader and the secure OS, respectively. A runtime process integrity measurement TA and attestation CA is developed and deployed in the SW and NW, respectively. Without loss of generality, the remote server is also used for remote attestation and the load-time and runtime integrity information is sent to the Android app so as to inform the user of the system status of the IoT system.

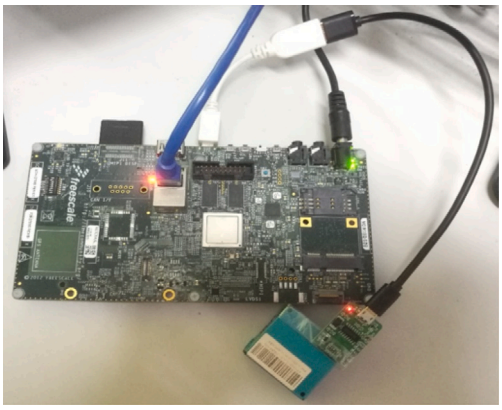


Fig. 10. Trusted air quality monitoring prototype.

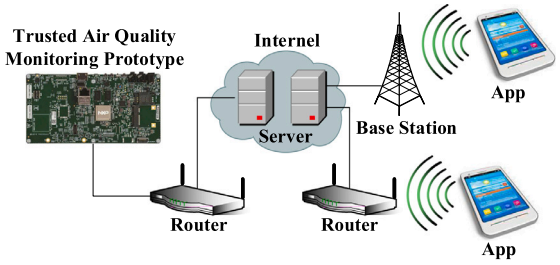


Fig. 11. Air quality monitoring prototype architecture.

6.2. Effectiveness

The evaluation of the secure boot process in the SW is aimed to verify whether the second-stage bootloader can detect any violation if the image of the secure OS, the public key, or the signature is tampered with. In the experiments, four different secure OS images are evaluated: an intact one, one with a tampered secure OS, one with a tampered public key, and one with a tampered signature. Only the intact image can boot successfully, while the others fail to boot up due to verification errors. The results show that the secure boot can enforce the load-time integrity of the SW.

The evaluation of the trusted boot process in the NW is aimed to verify whether our prototype can report the abnormal system status to the attestation server if the rich OS or the filesystem image is tampered with. After we tamper with the rich OS or the filesystem image, our prototype can boot up and function normally without being shut down by force. However, the remote attestation server has already detected the abnormal system status which can be sent to the user informing that the device is not trusted any more. The results show that by use of the trusted boot in the NW, even if NW images are tampered with, the NW programs can be executed, while the abnormal system status can be verified by the remote attestation server.

We assess the runtime process integrity measurement and attestation so as to verify whether our prototype can detect a newly-inserted malware or a tampered pre-installed program in the NW. We insert a malware into the NW. After the malware starts, the attestation server detects the malware and shows its name. Furthermore, we tamper with the code segment of a pre-installed program *serial_arm* in the NW. After restarting *serial_arm*, the attestation server indicates that a pre-installed program is tampered with. The results show that the paging-based process integrity measurement and attestation method can enforce runtime process integrity of the NW.

6.3. Performance

The performance evaluation of the hybrid boot is designed to measure the consumed time during the secure boot and trusted boot. We record the time consumed by both the secure boot module and the trusted boot module as well as the total booting time of second-stage bootloader and the secure OS, respectively. We conduct the timing measurement of the hybrid booting process for 30 times and take the average of the time. As the results shown in Table 1, the secure boot module introduces little overhead in the second-stage bootloader. Also, the trusted boot module slows down the secure OS booting process dramatically. The reason is that the filesystem image has a total size of 107 MB, and it takes a lot of time measuring it. Under real circumstances, the filesystem image can be compressed down to less than 1 MB, which can erase such performance bottleneck. In addition, a total booting time of approximate 9.2 s is tolerable in terms of user experience.

We evaluate the performance overhead introduced by the measurement TA and attestation CA in the paging-based process integrity measurement and attestation method. We use LMBench [19] to evaluate the system performance. In the experiments, we measure and compare the execution time of various Linux system services with the measurement TA and attestation CA enabled and disabled. We continuously call each system service for 1000 times, the call interval of each system service is 250 ms, and the whole performance evaluation lasts about 30 min. As shown in Table 2, when the measurement TA and attestation CA are enabled, the delay introduced to the evaluated services fluctuates between −0.55% and +0.67%. The results show that our the measurement TA and attestation CA introduce negligible performance overhead to the original system, and it is feasible to actually deploy our prototype system. Note that the time interval of NW process integrity measurement event is determined based on the trade-off between performance and security. Due to the limited computing resources of IoT devices, frequent measurement events will jeopardize the whole system performance. Additionally, a period adaptation way can be taken to actively adjust the measurement time interval at the runtime [20].

7. Security analysis and limitations

This section conducts security analysis on both the hybrid booting approach and the paging-based process integrity measurement and attestation method and discusses their limitations.

7.1. The hybrid booting approach

The hybrid booting approach ensures that the system starts from a legal state. The root of trust in our hybrid booting approach is established based on the eFuse and OCROM which are tamper-proof. Starting from the RoT, a chain of trust is established through the secure boot phase and a single image verification failure will terminate the whole booting process. After a successful secure boot, the SW measures the NW images and the measurement results are used for remote attestation. If the NW images are maliciously modified by an attacker, the remote attestation will fail and the user is alerted. Therefore, any offline modification to both the SW and the NW images will be detected and the system can only be in normal operation after a successful hybrid boot.

7.2. The paging-based process integrity measurement and attestation method

Both the SW measurement TA and the measurement results are secure from the NW. The secure boot ensures that only pre-installed SW programs will run inside the SW. Base on the hardware isolation mechanism provided by TrustZone, the SW measurement TA cannot be compromised by the NW. Additionally, the measurement results are

Table 1
Results of hybrid boot performance evaluation.

	Secure/trusted boot module booting time (ms)	Total booting time (ms)	Ratio (%)
Second-stage bootloader	23.7	6430.0	0.37
Secure OS	1276.0	2863.0	44.57

Table 2
Results of LMBench performance evaluation.

System service	Program integrity measurement and data transfer OFF (μs)	Program Integrity Measurement and Data Transfer ON (μs)	Difference (%)
null syscall	0.4230	0.4253	+0.54
open/close	10.1292	10.1865	+0.57
pagefault	1.2594	1.2678	+0.67
signal handler install	1.1063	1.1082	+0.17
fork+exit	1159.5902	1153.2701	−0.55
fork+exec	3410.6390	3405.9838	−0.14
select(250fd)	16.4555	16.4623	+0.04

encrypted inside the SW and then forwarded to the NW for network transmission. Only the SW and the remote attestation server have access to the decryption key and the NW can never get the plaintext.

Our measurement method now relies on the integrity of NW Linux paging structure and process management kernel objects, i.e., *task_structs*. Therefore, our method is vulnerable to malware capable of self-hiding, for example, transient rootkits [21]. Meanwhile, the semantic gap issue involved in all REM projects is still an open research topic [22] and existing TrustZone-based approaches can provide security protection for such kernel objects [16,23]. We plan to dedicate these semantic invariant protection topics to our future work.

8. Related work

8.1. Research and application of TrustZone

TrustZone is researched and widely deployed on different computing devices, including mobile devices and IoT devices. (1) Mobile Devices. Most ARM-based mobile devices are protected by TrustZone-based TEE, such as smart phones produced by Apple, Samsung [24], Huawei [25], Xiaomi, etc. (2) IoT Devices. TrustZone is used to protect IoT smart devices. For example, Ukil et al. [26] proposed to provide data security for IoT devices based on the TrustZone isolation mechanism. TrustShadow [27] leverages TrustZone to protect programs from untrusted Rich OSes. The program is placed in the SW to be isolated from the Rich OS. Its requests for OS services are forwarded to the Rich OS and the returning results are verified by TrustShadow. Recently, TrustZone is leveraged to realize real-time communication for hybrid dual-OS systems [28].

8.2. System load-time integrity verification

System load-time integrity verification techniques, e.g., secure boot and trusted boot, are employed to defend offline firmware tampering attacks. Both secure boot and trusted boot require offline system integrity measurement before system usage [29] and verify each component step by step from the root of trust forming a chain of trust.

The hardware-based RoT has the virtues of stability, reliability and small attack surfaces and therefore is preferred over the software-based ones [15]. For example, NXP's i.MX 6 series applications processors implement High Assurance Boot (HAB) with boot ROM and eFuse as the ROT [30]. Trusted Platform Module (TPM) [31], Mobile Trusted Module (MTM) [32], Battery Backup Random Access Memory (BBRAM) [33] can be leveraged to implement hardware-based RoTs. The fingerprint of on-chip Static Random Access Memory (SRAM) can be used to restore the seed for device key generation and thus provide RoT for TrustZone SW [34].

8.3. System runtime integrity verification

System runtime integrity verification is widely deployed to detect malicious or abnormal behaviors in computer systems, such as malware injection and modification of pre-installed programs. For instance, DRIVE [35] verifies the integrity of processes by comparing the memory image of the process with the corresponding executable binary image. Chang et al. [9] propose a page-based process integrity verification method by measuring the pages of one executable program's code segment in a virtual machine. Upon each page fault triggered by demand paging, the missing page is measured and its integrity is verified before it is loaded into memory. Wang et al. [36] propose a data integrity detection method based on edge computing [37] where self-balancing binary search trees are leveraged to accelerate the data auditing process in the cloud. Recently, machine learning technologies have been leveraged to detect malware [38] and software vulnerabilities [39].

Hardware-based process integrity measurement and attestation have been widely researched. For example, Hristozov et al. [40] propose a Device Identity Composition Engine (DICE)-based system runtime integrity verification method for lightweight MCU-powered IoT devices. Wang et al. [41] propose a hardware-based Instruction Stream Integrity Checker(ISIC) to measure the integrity of instruction blocks during program execution. Wehbe et al. [42] propose to connect a target embedded device to an external hardware monitor. The hardware monitor is responsible for measuring the pages of the target system's processes and comparing the measurement results with the pre-calculated ones stored in its secure storage.

9. Conclusion

This paper designs a hybrid booting approach consisting of both secure boot and trusted boot to enforce the IoT system load-time integrity. On this basis, the paging-based runtime process integrity measurement and attestation method is designed and implemented. The trusted SW measures and verifies process integrity of the NW to enforce the runtime process integrity of the system. An IoT prototype system is implemented on an IMX6Q SABRE SD development board. Extensive evaluations are performed to demonstrate the effectiveness of the system.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported in part by National Key R&D Program of China 2018YFB0803400, 2018YFB2100300 and 2017YFB1003000, US National Science Foundation (NSF) Awards 1643835, 1931871, and 1915780, US Department of Energy (DOE) Award DE-EE0009152, National Natural Science Foundation of China Grant Nos. 62022024, 61972088, 61632008, 62072103, 62072102, 62072098, 61972083, and 62061146001, Jiangsu Provincial Natural Science Foundation for Excellent Young Scholars Grant Nos. BK20190060, Jiangsu Provincial Key Laboratory of Network and Information Security Grant Nos. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China Grant Nos. 93K-9, Collaborative Innovation Center of Novel Software Technology and Industrialization. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time, 2020, https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/.

[2] Internet of Things (IoT) market size, share & covid-19 impact analysis, 2020, https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307. (Accessed July 2020).

[3] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, Y. Jin, Security analysis on consumer and industrial IoT devices, in: Proceedings of 21st Asia and South Pacific Design Automation Conference, ASP-DAC, Macao, Macao, pp. 519–524. [Online]. Available: https://doi.org/10.1109/ASPDAC.2016.7428064.

[4] O. Arias, J. Wurm, K. Hoang, Y. Jin, Privacy and security in Internet of Things and wearable devices, IEEE Trans. Multi Scale Comput. Syst. 1 (2) (2015) 99–109, http://dx.doi.org/10.1109/TMSCS.2015.2498605.

[5] G. Hernandez, O. Arias, D. Buentello, Y. Jin, Smart nest thermostat: A smart spy in your home, in: Proceedings of the 17th Black Hat USA, Las Vegas, USA, 2014.

[6] K. Liu, M. Yang, Z. Ling, H. Yan, Y. Zhang, X. Fu, W. Zhao, On manually reverse engineering communication protocols of linux based IoT systems, IEEE Internet Things J. (2020).

[7] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, X. Fu, Security vulnerabilities of Internet of Things: A case study of the smart plug system, IEEE Internet of Things J. 4 (6) (2017) 1899–1909, http://dx.doi.org/10.1109/JIOT.2017.2707465.

[8] X. Jiang, X. Wang, D. Xu, Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction, in: P. Ning and S.D.C. di Vimercati and P.F. Syverson (Ed.), Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS, Alexandria, Virginia, USA, 2007, pp. 128–138. [Online]. Available: https://doi.org/10.1145/1315245.1315262.

[9] C. Chang, X. Chen, S. Wang, Q. Xiao, Research on dynamic integrity measurement model based on memory paging mechanism, Discrete Dyn. Nat. Soc. 2014 (2014).

[10] How arm came to dominate the mobile market, 2021, https://www.techspot.com/article/1989-arm-inside/. (Accessed March 2021).

[11] Arm ltd. arm trustzone technology, 2020, https://developer.arm.com/ip-products/security-ip/trustzone. (Accessed November 2020).

[12] RD-IMX6Q-SABRE: Sabre board for smart devices based on the i.MX 6quad applications processors, 2020, https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/sabre-board-for-smart-devices-based-on-the-i-mx-6quad-applications-processors:RD-IMX6Q-SABRE. (Accessed November 2020).

[13] Arm ltd. ARM security technology. Building a secure system using trustzone® technology, 2020, https://documentation-service.arm.com/static/5f1ffa25bb903e39c84d7e98?token=. (Accessed November 2020).

[14] i.MX secure boot on HABv4 supported devices, 2020, https://www.nxp.com/docs/en/application-note/AN4581.pdf. (Accessed November 2020).

[15] B. Parno, J.M. McCune, A. Perrig, Bootstrapping trust in commodity computers, in: Proceedings of the 31st IEEE Symposium on Security and Privacy, S&P, Berleley/Oakland, California, USA, 2010, pp. 414–429. [Online]. Available: https://doi.org/10.1109/SP.2010.32.

[16] H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek, B.B. Kang, Vigilare: Toward snoop-based kernel integrity monitor, in: T. Yu and G. Danezis and V. D. Gligor (Ed.), Proceedings of the 19th ACM Conference on Computer and Communications Security, CCS, Raleigh, NC, USA, 2012, pp. 28–37. [Online]. Available: https://doi.org/10.1145/2382196.2382202.

[17] J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, E.W. Felten, Lest we remember: Cold-boot attacks on encryption keys, Commun. ACM 52 (5) (2009) 91–98, http://dx.doi.org/10.1145/1506409.1506429.

[18] N. Zhang, K. Sun, D. Shands, W. Lou, Y.T. Hou, TruSense: Information leakage from TrustZone, in: Proceedings of the 37th IEEE Conference on Computer Communications, INFOCOM, Honolulu, HI, USA, 2018, pp. 1097–1105. [Online]. Available: https://doi.org/10.1109/INFOCOM.2018.8486293.

[19] L.W. McVoy, C. Staelin, lmbench: Portable tools for performance analysis, in: Proceedings of the USENIX Annual Technical Conference, San Diego, California, USA, 1996, pp. 279–294.

[20] X. Dai, A. Burns, Period adaptation of real-time control tasks with fixed-priority scheduling in cyber-physical systems, J. Syst. Archit. 103 (2020) 101691, http://dx.doi.org/10.1016/j.sysarc.2019.101691.

[21] S. Wan, J. Sun, K. Sun, N. Zhang, Q. Li, SATIN: A secure and trustworthy asynchronous introspection on multi-core ARM processors, in: Proceedings of 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, Portland, OR, USA, 2019, pp. 289–301. [Online]. Available: https://doi.org/10.1109/DSN.2019.00040.

[22] B. Jain, M.B. Baig, D. Zhang, D.E. Porter, R. Sion, SoK: Introspections on trust and the semantic gap, in: Proceedings of the 35th IEEE Symposium on Security and Privacy, S&P, Berkeley, CA, USA, 2014, pp. 605–620. [Online]. Available: https://doi.org/10.1109/SP.2014.45.

[23] H. Lee, H. Moon, I. Heo, D. Jang, J. Jang, K. Kim, Y. Paek, B.B. Kang, KI-Mon ARM: A hardware-assisted event-triggered monitoring platform for mutable kernel object, IEEE Trans. Dependable Secur. Comput. 16 (2) (2019) 287–300, http://dx.doi.org/10.1109/TDSC.2017.2679710.

[24] Secure, deploy and manage with knox suite, 2020, https://www.samsungknox.com/en. (Accessed November 2020).

[25] Huawei, privacy protection, 2020, https://www.huawei.com/en/sustainability/stable-secure-network/privacy-protection. (Accessed November 2020).

[26] A. Ukil, J. Sen, S. Koilakonda, Embedded security for Internet of Things, in: Proceedings of the 2nd National Conference on Emerging Trends and Applications in Computer Science, St. Anthony's College, Shillong, Meghalaya, 2011, pp. 1–6.

[27] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, T. Jaeger, TrustShadow: Secure execution of unmodified applications with ARM TrustZone, in: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys, Niagara Falls, NY, USA, 2017, pp. 488–501. [Online]. Available: https://doi.org/10.1145/3081333.3081349.

[28] P. Dong, Z. Jiang, A. Burns, Y. Ding, J. Ma, Build real-time communication for hybrid dual-os system, J. Syst. Archit. 107 (2020) 101774, http://dx.doi.org/10.1016/j.sysarc.2020.101774.

[29] M. Gasser, A. Goldstein, C. Kaufman, B. Lampson, The digital distributed system security architecture, in: Proceedings of the 12th National Computer Security Conference, 1989, pp. 305–319.

[30] AN4581 i.MX secure boot on HABv4 supported devices, 2020, https://www.nxp.com/docs/en/application-note/AN4581.pdf. (Accessed November 2020).

[31] A. Tomlinson, Introduction to the TPM, in: Smart Cards, Tokens, Security and Applications, second ed., 2017, pp. 173–191.

[32] J.-E. Ekberg, Mobile trusted module (MTM)–An introduction, 2007.

[33] J.G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S.W. Smith, S.H. Weingart, Building the IBM 4758 secure coprocessor, Computer 34 (10) (2001) 57–66, http://dx.doi.org/10.1109/2.955100.

[34] S. Zhao, Q. Zhang, G. Hu, Y. Qin, D. Feng, Providing root of trust for ARM TrustZone using on-chip SRAM, in: Proceedings of the 4th International Workshop on Trustworthy Embedded Devices, TrustED, Scottsdale, Arizona, USA, 2014, pp. 25–36. [Online]. Available: https://doi.org/10.1145/2666141.2666145.

[35] A. Rein, DRIVE: Dynamic runtime integrity verification and evaluation, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS, Abu Dhabi, United Arab Emirates, 2017, pp. 728–742. [Online]. Available: https://doi.org/10.1145/3052973.3052975.

[36] T. Wang, Y. Mei, X. Liu, J. Wang, H.-N. Dai, Z. Wang, Edge-based auditing method for data security in resource-constrained Internet of Things, J. Syst. Archit. 114 (2021) 101971, http://dx.doi.org/10.1016/j.sysarc.2020.101971.

[37] T. Wang, Y. Lu, J. Wang, H.-N. Dai, X. Zheng, W. Jia, EIHDP: Edge-intelligent hierarchical dynamic pricing based on cloud-edge-client collaboration for IoT systems, IEEE Trans. Comput. (2021) 1, http://dx.doi.org/10.1109/TC.2021.3060484.

[38] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, K. Ren, Android HIV: A study of repackaging malware for evading machine-learning detection, IEEE Trans. Inf. Forensics Secur. 15 (2020) 987–1001, http://dx.doi.org/10.1109/TIFS.2019.2932228.

[39] G. Lin, S. Wen, Q.-L. Han, J. Zhang, Y. Xiang, Software vulnerability detection using deep neural networks: A survey, Proc. IEEE 108 (10) (2020) 1825–1848, http://dx.doi.org/10.1109/JPROC.2020.2993293.

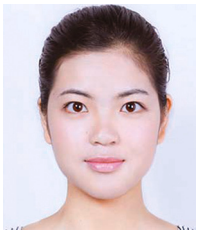
[40] S. Hristozov, J. Heyszl, S. Wagner, G. Sigl, Practical runtime attestation for tiny iot devices, in: Proceedings of the 2018 Workshop on Decentralized IoT Security and Standards, San Diego, CA, USA, vol. 18, 2018.
[41] X. Wang, W. Wang, B. Xu, P. Du, L. Li, M. Liu, A fine-grained hardware security approach for runtime code integrity in embedded systems, J.UCS 24 (4) (2018) 515–536.
[42] T. Wehbe, V.J.M. III, D.C. Keezer, Hardware-based run-time code integrity in embedded devices, Cryptography 2 (3) (2018) 20, http://dx.doi.org/10.3390/cryptography2030020.

Zhen Ling received the B.S. degree (2005) and Ph.D. degree (2014) in Computer Science from Nanjing Institute of Technology, China and Southeast University, China, respectively. He is an associate professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He won ACM China Doctoral Dissertation Award and China Computer Federation (CCF) Doctoral Dissertation Award, in 2014 and 2015, respectively. His research interests include network security, privacy, and Internet of Things.

Huaiyu Yan received the B.S. degree in software engineering from Southeast University, Nanjing, China, in 2019. Currently, he is working toward the Ph.D. in computer science and engineering at Southeast University, Nanjing, China. His current research interests include Internet of Things, privacy and security.

Xinhui Shao received the B.S degree in communication engineering from Shanghai University, Shanghai, China, in 2019. Currently, he is working toward the master degree in cyber science and engineering at Southeast University, Nanjing, China. His current research interests include Internet of Things, privacy and security.

Junzhou Luo received the B.S. degree in applied mathematics and the M.S. and Ph.D. degrees in computer network, all from Southeast University, China, in 1982, 1992, and 2000, respectively. He is a full professor in the School of Computer Science and Engineering, Southeast University, Nanjing, China. He is a member of the IEEE Computer Society and co-chair of IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design, and he is a member of the ACM and chair of ACM SIGCOMM China. His research interests are next generation network architecture, network security, cloud computing, and wireless LAN.



Yiling Xu received the B.S. degree (2016) in digital media technology and the M.S degree (2019) in Computer Science from Jiangnan University, China and Southeast University, China, respectively. She is a Software Engineer in Test in Alibaba Group, Hangzhou, China. Her research interests include Internet of Things, privacy and security.

Bryan Pearson received his B.S. degree in Computer Science from Stetson University in 2018. He is currently working towards the Ph.D degree in Computer Science at the University of Central Florida. His research interests include Internet of Things security and privacy, fuzz testing, and binary analysis.

Dr. Xinwen Fu is a Professor with the Department of Computer Science, University of Massachusetts Lowell, Lowell, MA, USA. He received the B.S. degree in 1995 from Xi'an Jiaotong University, Xi'an, China, the M.S. degree in electrical engineering in 1998 from the University of Science and Technology of China, Hefei, China, and the Ph.D. degree in computer engineering in 2005 from Texas A&M University, College Station, TX, USA. His current research interests include computer security and privacy, and digital forensics. His research was reported by various media such as Wired and aired on CNN and CCTV 10. He is a senior member of IEEE.