논문 2015-10-38

SeBo: Secure Boot System for Preventing Compromised Android Linux

Tong Min Kim, Se Won Kim, Chuck Yoo*

Abstract : As the usage of mobile devices becomes diverse, a number of attacks on Android also have increased. Among the attacks, Android can be compromised by flashing a new image of compromised Android Linux. In order to solve this problem, we propose SeBo (Secure Boot System) which prevents compromised Android Linux by guaranteeing secure boot environment for mobile devices based on ARM TrustZone architecture. SeBo checks the hash value of the Android Linux image before the Android Linux executes. SeBo detects all the attacks within 5 seconds. Moreover, since SeBo only trusts the Secure Bootloader from Secure World, SeBo can reduce the additional overhead of checking the Normal Bootloader from Normal World.

Keywords : Secure boot, Embedded system, Embedded security, Android, Linux, ARM TrustZone

Ⅰ. Introduction

The growing market of mobile devices has led to an increment of various purposes of mobile applications. Among them, some applications can be trusted if certified.

However, even though the application is trusted, the potential of data leakage still exists when the mobile OSes are compromised. The mobile OSes such as Android and iOS can be compromised by rooting or jailbreaking attacks. These attacks allow malicious applications to obtain and to manage the root privilege, which can be a great threat for sensitive data.

There are two kinds of rooting attacks in Android [1]. One of them is exploiting system vulnerabilities, but recently, the number of the attacks has decreased as it has become harder to find vulnerabilities in Android any more. This is because Android uses two types of access control [2, 3], DAC (Discretionary Access Control) and MAC (Mandato ry Access Control).

The other method is flashing compromised image. This method cannot be protected by the dual access control mentioned above as attackers can newly flash a whole compromised Android Linux image (ALimage).

In order to prevent the compromised ALimage from being flashed, this paper proposes SeBo (Secure Boot) checker and SeBo monitor. SeBo checker and SeBo monitor verify the current ALimage whether it is trusted or not before the current Android starts up.

SeBo is implemented based on ARM TrustZone architecture. In the architecture, ARM TrustZone provides two types of the worlds; Secure world and Normal world. In this paper, SeBo runs in Secure world and Android Linux in Normal world, respectively. By dividing SeBo and Android Linux into two worlds, these systems can be isolated from each other [4]. Therefore, the isolated SeBo in Secure world can check the Android Linux without any threats from Normal world. In this regard, SeBo guarantees secure environment

*Corresponding Author(hxy@os.korea.ac.kr)
Received: 31 Jan. 2015, Revised: 19 May 2015, Accepted: 27 May 2015.
T.M. Kim, S.W. Kim, C. Yoo: Korea University
※ 이 논문은 국가과학기술연구망(KREONET)을 이용하여 한국연구재단의 지원을 받아 수행된 연구임(No.2010-0029180)

for mobile devices.

This paper is organized as follows. Section 2 describes background concepts of ARM TrustZone. Section 3 briefly analyzes attack examples. Section 4 proposes the design and implementation of SeBo. Section 5 compares the evaluation results between SeBo and native. Related works on Secure Boot are presented in section 6. Finally, we conclude in section 7.

Ⅱ. ARM TurstZone

ARM TrustZone architecture guarantees a system to be secure, if security intellectual features are suitably configured. This paper use a number of the security IPs in order to ensure secure boot sequence for preventing compromised Android Linux. SeBo uses following features:

1. Operation Mode

ARM TrustZone architecture provides two executing domains called Secure world and Normal world. Within a world, OS can directly execute privileged instructions without any modification because both worlds have all the seven modes USR, FIQ, IRQ, UND, SVC, SYS, and ABORT that are defined by ARM architecture[5]. In addition, because the CPU mode and world domains are orthogonal, each world can execute its own OS and applications.

Secure world has a special mode called monitor mode to manage both worlds. The monitor mode takes charge of switching the worlds and sending the device interrupts to the corresponding worlds [6]. For the purpose, a new instruction called SMC (Secure Monitor Call) is introduced. When a guest OS issues an SMC instruction, the SMC exception is generated, and it is handled by the SMC handler in monitor mode of Secure world. The SMC handler saves the CPU context of the current world and restores that of opposite world. Then, the world is switched.

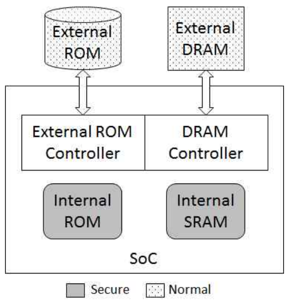

Fig. 1 Location of Internal and external memories

2. Internal ROM and Internal SDRAM

ARM TrustZone architecture supports additional ROM and RAM called Internal ROM (iROM) and Internal SRAM (iSRAM). They can not be modified or replaced by simple reprogramming attacks [6]. Fig. 1 shows that the iROM and iSRAM are inside of the SoC and the external ROM and external DRAM are outside of the SoC.

We store and execute both SeBo checker and SeBo monitor in the iROM and the iSRAM because iROM and iSRAM ensure security. On the other hand, as external ROM and DRAM can be compromised, we store and execute Android Linux in external ROM and DRAM. Detailed description will be discussed in the *Design and Implementation* section.

3. Boot Sequence

ARM TrustZone architecture recommends two kinds of bootloaders [6]. With proper configurations of IPs, the first bootloader, Secure bootloader, can be implemented based on internal iROM and iRAM, and the other, Normal bootloader, can be implemented based on external ROM and DRAM, respectively. Fig. 2 describes the recommended sequence of the two kinds of bootloaders.

In order to separate and protect SeBo checker and SeBo monitor from the Android Linux in the Normal world, we locate two kinds of bootloaders as recommended. At first, Secure bootloader starts right after power on the mobile device. Then,

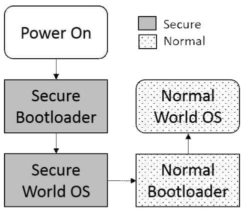

Fig. 2 ARM recommended sequence of Secure and Normal Bootloader

Secure world OS starts. The Secure world OS controls the Normal world components such as Normal bootloader and Normal world OS. Since the Secure world OS can be replaced with simpler system such as monitor systems, we use SeBo monitor instead of the Secure world OS. Finally, SeBo monitor controls Normal world components. Among Normal world components, we use Android Linux instead of the Normal world OS. The boot sequence in this paper will also be described in detail in the Section V.

Ⅲ. Attack Examples

Among the two kinds of rooting method mentioned above, we mainly focus on the attack examples of the latter method; flashing compromised image. There are two possible attack examples when flashing compromised image. Fig. 3 shows these two attack examples compare to the ideal operation; compromised ALimage with or without compromised Normal bootloader. The ideal operation neither have compromised ALimage nor compromised Normal bootloader.

In the ideal operation, the manufacturer's Normal bootloader loads the manufacturer's ALimage on the kernel load address and starts decompressing the image. However, since these manufacturer's ALimage and bootloader are in the Normal world, attackers can replace them with the compromised ones.

If an attacker replaces the manufacturer's ALimage with the compromise one, the manufacturer's Normal bootloader loads the compromised ALimage. This is because a

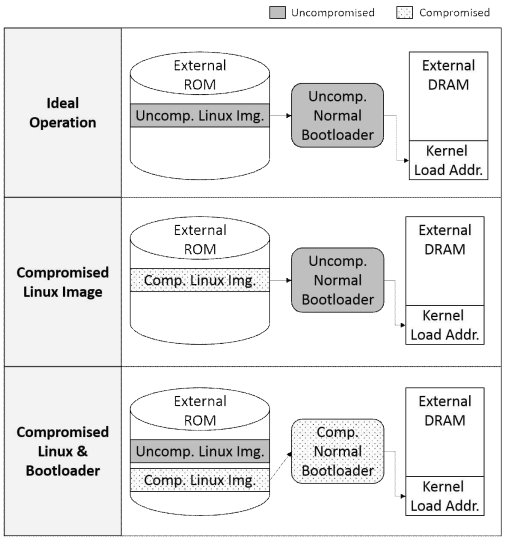

Fig. 3 Attack examples of flashing compromised ALimage

manufacturer's Normal bootloader usually loads an ALimage according to the fixed address without any checking. In other words, the manufacturer's Normal bootloader only loads the ALimage from the fixed address whether the ALimage is compromised or not. Therefore, if the attacker locates the compromised ALimage on the fixed address, the manufacturer's Normal bootloader loads the compromised ALimage.

In the case of an attacker replaces both manufacturer's ALimage and bootloader with the compromised ones, the attacker can load an ALimage from any address via the compromised Normal bootloader. Namely, the attacker can load the compromised ALimage instead of the manufacturer's avoiding the fixed address.

Ⅳ. Design and Implementation of SeBo Checker and SeBo Monitor

With SeBo checker and SeBo Monitor, the above attack examples can be defended since the SeBo checker verifies the Android Linux in the external DRAM. In order to prevent the

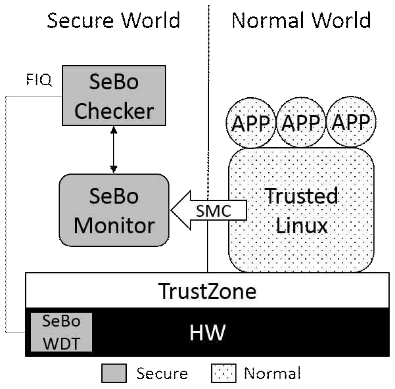

Fig. 4 The architecture of SeBo components and trusted Android Linux

compromised Android Linux in the Normal world, both SeBo checker and SeBo Monitor should be separated from the Normal world. Fig. 4 shows the architecture of SeBo checker and SeBo monitor separated from the Normal world.

From now on, we assume that all of the trusted Android Linux calls SeBo monitor via SMC instruction before the Android Linux executes. In the case of compromised Android Linux, we divide it into two types: the one that does not call the SeBo monitor and the other that can call SeBo monitor forcefully.

The trusted Android Linux passes all the checks of SeBo components whereas both case of the compromised Android Linux cannot pass the checks.

1. SeBo Components

SeBo components fall into two main components: SeBo monitor and SeBo checker. Each role is as follows.

1.1 SeBo Monitor

On the recommendation of ARM TrustZone architecture, a monitor should be a security critical gatekeeper that provides the interface between Normal world and Secure world via the following exceptions: an interrupt, and external abort, or an explicit call via an SMC instruction [5].

SeBo monitor is also implemented as recommended. Additionally, SeBo monitor receives SMC from the Android Linux in the Normal world. Then, SeBo monitor executes SeBo checker. After that, SeBo monitor sends the start and the end addresses of current ALimage to the SeBo checker. If the Android Linux does not call SMC and bypasses the SeBo checker, the SeBo checker erases data and turns off the power.

1.2 SeBo Checker

SeBo checker calculates the hash value of the addresses received from the SeBo monitor. After calculating the current hash value, SeBo checker compares the current hash value with the original hash value. The original hash value can be stored in a secure storage such as on-SoC ROM by the manufacturer in the first place. When the manufacturer wants to update the kernel, the manufacturer can send a new kernel ALimage with a new original hash value by Over The Air (OTA) management [7].

If the original hash value does not match with the current hash value the SeBo checker erases data and turns off the power. On the contrary, if the original hash value matches with the current hash value, SeBo checker disables SeBo Watch Dog Timer.

1.3 SeBo Watch Dog Timer

The SeBo Watch Dog Timer (WDT) detects the compromised Android Linux that bypassed the SeBo checker. SeBo WDT uses Global timer in ARM SoC [8]. SeBo WDT is implemented in the monitor code of Secure World by the reference to [9].

If the compromised Android Linux bypasses the SeBo checker, the SeBo WDT expires because the SeBo checker does not disable the SeBo WDT. After the SeBo WDT expired, the SeBo WDT sends FIQ to the SeBo checker and the SeBo checker erases data and turns off the power.

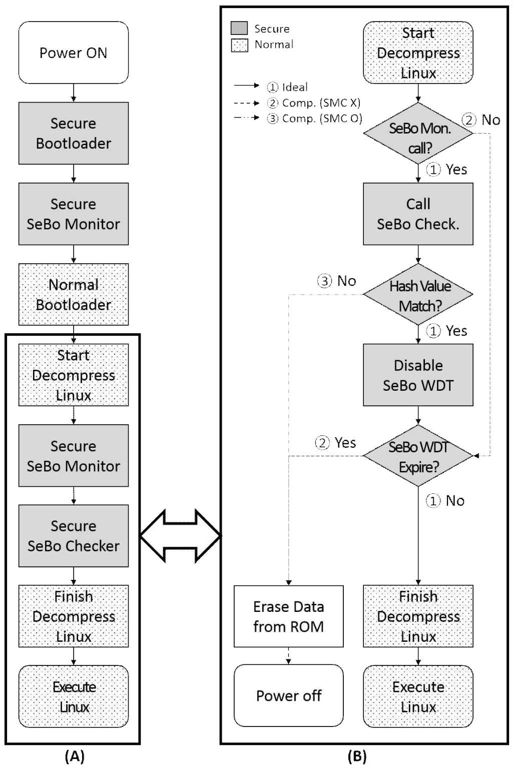

Fig. 5 The flow charts of the SeBo boot sequence

2. SeBo Boot Sequence

In this section, we describe the boot sequence of the above SeBo components called SeBo boot sequence. Fig. 5 shows the flow charts of the overall (Fig. 5(A)) and the detailed (Fig. 5(B)) SeBo Boot sequence.

As shown in Fig. 5(A), SeBo monitor and SeBo checker check the Android Linux after the Normal bootloader starts to decompressing the ALimage. Specifically, the SeBo monitor receives SMC, during the decompressing time. After that, the SeBo monitor temporary stops the decompressing and sends the start and end addresses to the SeBo checker. Then, the SeBo checker starts checking the Android Linux. Finally, the SeBo checker finishes the checks and completes the decompressing before executing the Android Linux.

Fig. 5(B) shows three possible cases of scenarios that can either pass or fail the checks of the SeBo checker.

The first scenario, in Fig. 5(B)-①, describes the ideal operation of trusted Android Linux. The trusted Android Linux can pass all checks and be executed.

The second scenario, shown in Fig. 5(B)-②, describes the operation of compromised Android Linux that does not contain SMC instruction. Without SMC instruction, The compromised Android Linux cannot call SeBo monitor and bypasses the SeBo checker. Bypassing the SeBo checker expires the SeBo WDT. Then, the SeBo checker erases data from ROM and turns off the power.

The third scenario, shown in Fig. 5(B)-③, describes the operation of compromised Android Linux that contains SMC instruction. If the attacker forcibly inserts the SMC instruction in the compromised Android Linux to avoid SeBo WDT expiration, the SeBo checker detects it by comparing its hash value with the original hash value. Mismatching hash value leads the SeBo checker to erase Data from ROM and to turn off the power.

V. Evaluation

We implemented SeBo in DTK4412 board for evaluation. DTK4412 board is an Android Linux platform based on Samsung Exynos 4412 quad core processor. DTK4412 includes u-boot 2010.12, Android 4.0.4 (Ice Cream Sandwich) and Linux 3.0.15.

SeBo monitor has 2054 lines of assembly code and SeBo checker has 1165 lines of C code. For trusted ALimage, we added 6 lines of assembly code to call SMC instruction. We first analyze the performance result of SeBo. Then we describe the detection results of SeBo according to each attack example.

1. Performance Evaluation

Since SeBo operates in the boot sequence, SeBo cause no effect on the runtime performance.

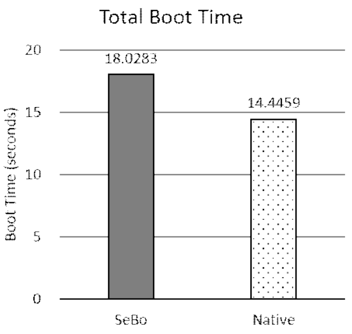

Fig. 6 Total boot time of SeBo and Native

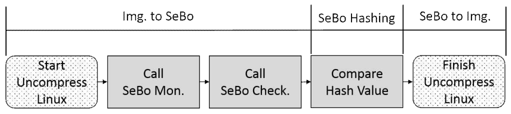

Fig. 7 SeBo operation divided by three phases

Therefore, we focus on the performance of boot time and analyze it accordingly. Fig. 6 describes the total boot time of SeBo and native. The native boot time means general boot time without SeBo.

The Y-axis of Fig. 6 describes total boot time which is measured in seconds. Comparing the two, SeBo boot time delays only about 3 seconds more than that of the native. In other words, SeBo finishes all its operation within 4 seconds. SeBo operation can be divided by three phases as shown in Fig. 7.

In Fig. 8, the X-axis describes the execution time and Y-axis indicates SeBo execution divided by three phases. As shown in Fig.8, comparing hash values between the original and the current occupies most of the operation time. In this paper, we use MD5 hash function since the MD5 calculates the hashed value faster than that of SHA-1 [10]. Though MD5 is faster than the other hash functions, it takes much time to calculate the whole ALimage value because of its various operation process [11]. Therefore, due to the characteristics of the MD5 hash function, the amount of time increases rapidly depending on the size of ALimage. For example, the size of

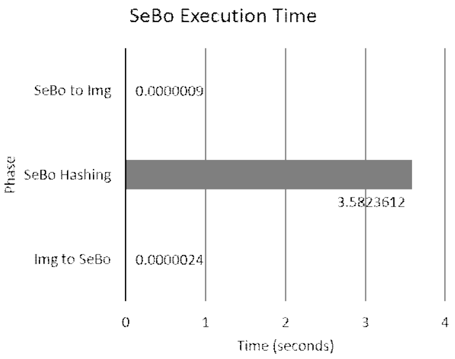

Fig. 8 The execution time of SeBo

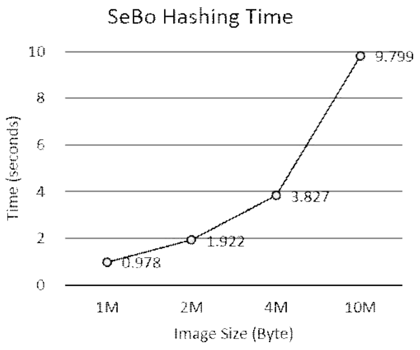

Fig. 9 SeBo hashing times according to ALimage sizes

our experiment ALimage was 3.8Mbyte and it took 3.58 seconds for SeBo to hash the values. The other hashing times according to their ALimage sizes are shown in Fig. 9.

We set SeBo WDT to be expired within 5 seconds considering the fact that most of ALimage rarely over 5Mbyte.

2. Detection Results

In order to evaluate SeBo, we made two types of compromised ALimages and a compromised bootloader. The first type of ALimage has SMC instruction and the other does not have SMC instruction. Since the size of a general ALimage is within 4Mbyte, we limit the size of ALimages by 4Mbyte. The compromised bootloader loads a ALimage from

Table 1. The detect results and detected time

	Without SMC	With SMC
Trusted Img. & Trusted Bootldr.	-	Undetected (-)
Comp. Img.	Detected (5s)	Detected (3.8s)
Comp. Img. & Comp. Bootldr.	Detected (5s)	Detected (4.0s)

uncommon address. Table 1 describes the detect results and the detected time of four attack examples. The detected time is measured after a bootloader loads an ALimage on a external DRAM. We also include the trusted case for the comparison.

As shown in Table 1, the ideal case with trusted ALimage and trusted bootloader was undetected by SeBo. However, SeBo detected all the four cases of compromised ALimages.

It takes the exact 5 seconds for SeBo to detect any compromised ALimage that does not have SMC instruction. This is because SeBo WDT expires within 5 seconds if the compromised ALimage bypasses the SeBo checker.

If a compromised ALimage has SMC instruction, SeBo took about 4 seconds to detect the ALimage. Especially, the SeBo checker takes most of the time due to generating hash value of the current ALimage and comparing it with the original hash value.

Ⅵ. Related Work

For comparing the related work, we introduce Samsung KNOX Workspace. Samsung KNOX guarantees platform security with three strategies: Customizable Secure Boot, ARM TrustZone-based Integrity Measurement Architecture (TIMA), and a kernel with built-in Security Enhanced Android access Control [12]. Among them, we focused on the Customizable Secure Boot since it has the same goal with different approach. Table 2 summarizes similarities and differences between SeBo and KNOX.

Table 2. Similarities and differences between SeBo and KNOX

	SeBo	KNOX
Goal	Preventing compromised Normal OS & Normal bootloader	
Based on	ARM TrustZone	
Checks	- ALImage	- Normal Bootloader - ALImage
Operates in	- Boot sequence	- Boot sequence - Runtime

Both SeBo and KNOX prevents compromised Normal OS and bootloader and guarantees secure boot. They are also based on the ARM TrustZone architecture.

Even though the goal is the same, their approach is different. KNOX not only checks ALimage but also Normal bootloader, which means, KNOX trusts Normal bootloader and allows Normal bootloader to check the integrity of the ALimage. SeBo, on the other hand, only checks the ALimage and does not check Normal bootloader. Instead of trusting the Normal bootloader in the Normal World, SeBo trusts SeBo checker in the Secure World and let SeBo checker to check the integrity of the ALimage. Therefore, SeBo can check the ALimage safely without trusting the components in the Normal World. Such a method also can reduce the overhead of checking the Normal Bootloader.

KNOX provides two kinds of boot mechanism; Secure Boot and Trusted Boot. Similar to SeBo, Secure Boot operates only in the boot sequence and not in the system runtime. However, Trusted Boot consistently checks the integrity of the system in the system runtime. Since the scope of SeBo is only limited to the integrity of the boot sequence, checking the integrity of the system runtime is additionally required for the future work.

Ⅶ. Conclusion and Future Work

The results of this study suggests that SeBo can provide secure boot environment by preventing not only compromised OS but also compromised Normal bootloader. From the result, SeBo detects all the attacks within 5 seconds only with 3225 lines of additional code. Moreover, SeBo is lightweight because SeBo focuses only on the boot component from Secure World, free of superfluousness.

Since SeBo hashes the ALimage value only with the MD5, performance comparison between MD5 and the other hash functions is needed for the future work. Furthermore, checking mechanism in the system runtime such as data access control [13, 14], is required.

References

[1] S. Yuru, X. Luo, C. Qian. “Rootguard: Protecting rooted android phones," IEEE Computer Vol. 47, No. 6, pp. 32-40, 2014.

[2] S. Smalley, “The case for SE Android,” In Linux Security Summit 2011. http://selinuxproject.org/~jmorris/lss2011_slides/caseforseandroid.pdf.

[3] S. Smalley, R. Craig. “Security Enhanced (SE) Android: Bringing Flexible MAC to Android," NDSS (Vol. 310, pp. 20-38), 2013.

[4] T.M. Kim, S.W. Kim, C. Yoo, “Tiny Monitoring Platform for Protecting Data against Compromised Mobile Operating Systems," Proceeding of Autumn Conference on IEMEK (in Korean).

[5] Technologies, A.R.M “ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition, " URL: ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition

[6] Technologies, A.R.M “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

[7] Aghera, P., Bok, A., Chintada, S., Rao, S., & Rinaldi, A. (2003). U.S. Patent Application 10/652,352.

[8] Technologies, A.R.M “Chapter 4. Global timer, private timers, and watchdog registers,” http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0407i/BEJHAGEE.html

[9] LWN.net, “ARM:global_timer: Add ARM global timer support,” https://lwn.net/Articles/549648/

[10] D. Menascé, “Security performance." IEEE Internet Computing, Vol. 7, No. 3, pp. 84-87, 2003.

[11] R. Rivest, “The MD5 message-digest algorithm,” http://tools.ietf.org/html/rfc1321?ref=driverlayer.com, 1992.

[12] Technologies, Samsung “Samsung KNOX Workspace" https://www.samsungknox.com/en/products/knox-workspace/technical

[13] J. Shin, Y. Kim, W. Park, C. Park, “A Secure Data Management Framwork based on ARM TrustZone for Cloud Storage Services," Proceeding of Autumn Conference on IEMEK (in Korean).

[14] J. Shin, Y. Kim, W. Park, C. Park, "A Method for Data Access Control and Key Management in Mobile Cloud Storage Services," J. IEMEK Embed. Syst. Appl., Vol. 8, No. 6, pp. 303-309, 2013 (in Korean).

Tong Min Kim (김 동 민)

Received the B.S. degrees in computer science from Dongduk Women's University. She is currently pursuing her M.S. degree in College of Informatics, Korea University, Seoul, Korea. Her research interests include operating system, embedded system and computer security.

Email: tmkim@os.korea.ac.kr

Se Won Kim (김 세 원)

Received the B.S. and M.S. degrees in computer science from Korea University. He is currently pursuing his Ph.D. degree in College of Informatics, Korea University, Seoul, Korea. His research interests include realtime system, embedded system and power management.

Email: swkim@os.korea.ac.kr

Chuck Yoo (유 혁)

Received the B.S. and M.S. degrees in electronic engineering from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees in computer science from University of Michigan, Ann Arbor, USA. He worked as a researcher in Sun Microsystems Laboratory from 1990 to 1995. He is a professor in the College of Information and Communications, Korea University, Seoul, Korea since 1995. His research interests include operating systems, embedded systems, virtualization, and multimedia streaming.

Email: hxy@os.korea.ac.kr