Sharing Experience and Knowledge: How to verify Secure Boot against Modify Attack

Yangjae Lee
SysCore Lab.
Sejong University
Seoul, South Korea
leelambjae@gmail.com

Keon-Ho Park
SysCore Lab.
Sejong University
Seoul, South Korea
imguno0629@naver.com

Daeseon Choi
Dept. Department of Medical Information
Kongju National University
Kongju, South Korea
sunchoi@kongju.ac.kr

Ki-Woong Park*
Dept. Information Security
Sejong University
Seoul, South Korea
woongbak@sejong.ac.kr

***Abstract*— To enhance the mobility of embedded devices, the board has become smaller. However, this characteristic makes the board to be stolen easier. In this case, when the board which contains critical information is stolen, the attacker can steal the data easily because they can conduct every attack from physical attack to cyber-attack. Therefore, implementing the secure boot is an indispensable thing for embedded system security. However, does secure boot can really recognize the modify attack? In this paper, we detail our experiences which building and executing verification model to test the secure boot against modify attack. We expect that these experiences serve two benefits for readers. First, readers can get the knowledge of T2080 RDB which we used for exemplary, Second, it can raise questions about what verification model is the best for verifying the secure boot against modify attack.**

Keywords— verification method; embedded system; secure booting; security methodology

I. Introduction

To protect boot procedure, many kinds of secure boot have released [5-7]. However, we can't really be sure that the secure boot can protect the modify attack. Therefore, we make a verification model against modify attack to verify the secure boot. In our experiences which building and executing verification model, we expect that readers to obtain the two benefits. The first benefit is knowledge of T2080RDB which we used for exemplary. Because our verification model requires prior knowledge of the service, readers can obtain the information about T2080RDB which we have investigated. The second benefit is that readers can raise questions which verification model is the best for verifying the secure boot against modify attack. Because we investigated some verification model for other service and analyzed each verification model, readers can raise the question which verification model is the best.

II. What Verification Model Most Suitable for Secure Boot Against Modify Attack?

In order to make most suitable verification model for the secure boot against modify attacks, we analyzed other verification models for other services. The detailed procedure of each verification model is listed in Table Ⅰ.

SiChoon Noh et al. proposed verification method of ensuring web application security [1]. To protect the threats, they present the verification model for web application security, the verification model procedure is consisting of four steps. Myong-Yeal Lee et al. proposed a safe smart car application plan [2]. They look at types of smart car and deriving the security threats based on various scenarios. Jeom Goo Kim. proposed an automatic method which verifies the network security system and verification method based on scenarios [3]. To verify the network security system, they found the problems of current networks system based on scenarios. Balzarotti, Davide, et al. describe the testing methodology that used in testing the security of real-world electronic voting system [4].

By analyzing the other verification model for other services, we found that there are some common features. First, most of the verification model gathered information about the services to protect. After that, they analyzed the information. For example, in the verification model of web application [1], they set the scope to protect and analyzed the target application. Second, most of the verification model find threats to the service. In the verification model for smart car [2], the anticipated threats and risks are analyzed based on existing attack cases and information about services. Third, they make the attack scenarios by linking the threats. In the verification model for real-world electronic voting systems [4], they made realistic attack scenarios according to the procedure of the voting system. Finally, they verify the security by executing the attack scenarios or attacks. In the evaluation technique of network security system, they experimented and analyzed in a small school network according to scenario-based verification method. Some verification models additionally proposed countermeasures in verification model. Analyzing the several verification models and investigating the secure boot, we draw one verification model for secure boot against modify attack. In

*: Corresponding author

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NRF-2017R1C1B2003957, NRF-2016R1A4A1011761).

TABLE Ⅰ. The Procedure of Each Verification Model

Target Application	*Step 1*	*Step 2*	*Step 3*	*Step 4*
[1]	Set the scope of security verification	Analyze the target application	Select the security checklists	Verify the proof of security
[2]	Look at types and policies of smart car	Make scenarios which can occur	Analyze the smart car security threats based on scenarios	Propose the countermeasures
[3]	Build the database for security checklists	Define and grade the risks of security	Establish a verification environment based on the scenario	Test the automatic verification tool and proof security
[4]	Information gathering such as copy machine, source code, and documentation.	Analysis the system. Such as information flow and vulnerability	Draw the attack scenarios according to the procedure	Execute attack and report the results

our verification model, to find the attack surface we analyzed the secure boot first. After finding the attack surface against modify attack, we found risks that can occur if attack surfaces are modified. Combining these risks, we made scenarios. Executing these scenarios, we try to verify the security of secure boot.

III. Design of Verification Model

To test the secure boot, we use T2080 RDB, which supports secure boot and be used for military application [8-10]. To make the verification model for T2080RDB secure boot, we form the verification model in four steps. First, to define which components to protect, the boot process should be analyzed. After analyzing the boot process, components to protect should be defined. Second, found the risk that can occur if components are modified. Third, make the attack scenarios that can exploit the system by combining the risk. When we make scenarios based on risks and components, numerous scenarios were derived. But verifying all the scenarios manually consumes a lot of resources. In this paper, we make two attack scenarios for the case study, which steal the data and get the total access based on the attacker's primary goal. At the last, scenarios are actually executed on the T2080 secure boot to determine whether the attack succeed or fail. If it fails, by analyzing which function of T2080 secure boot is able to defend, assure the safety of T2080 secure boot.

A. Defining and analyzing the components to protect

In this paper, we divided secure boot process of T2080 into three steps based on the secure boot phase. The first step is the Pre-Boot Phase. When the power is applied to the system, Pre-Boot Phase checks the security status of the system. Also, Pre-Boot Phase uses Security Fuse Processor (SFP) and Pre-Boot Loader (PBL). SFP physically burn fuses during device provisioning and send One Time Programmable Master Key (OTPMK) and Super Root Key Hash (SRKH) to other hardware securely. PBL reads a command file from a location determined by the Reset Configuration Word (RCW) and performs a store of a value to the ESBC pointer register within the SoC. In the case study, because SFP is programmed in hardware, we assume that to modify the SFP is impossible [11].

The second step is the Internal Secure Boot Code (ISBC) Phase. In the ISBC Phase, check the integrity of U-boot, Barker Code of CSF Header, and public key.

The third step is the External Secure Boot Code (ESBC) Phase. The ESBC Phase checks the integrity of the Boot Script, the Root File System (Rootfs), the Device Tree Blob (DTB), and the Linux kernel.

According to the T2080 secure boot process, we can define 8 components to protect. Table Ⅱ shows a description of components to protect.

The components of Table Ⅱ are used to validate the boot images in T2080RDB secure boot. By defining the which components to protect, attack points can be specified, and attacks can be blocked effectively.

TABLE Ⅱ. Description of Components to Protect

Components	*A Description of Components*	*Elements related to Security*
U-boot	U-boot is bootloader, and there are secure U-boot and normal u-boot.	Secure U-boot checks the integrity of uImage, Rootfs, and DTB.
Rootfs	The filesystem that contains a program that supports the system operating.	Rootfs can be superseded to other Rootfs which have viruses.
DTB (Device Tree Blob)	The data structure which describes the hardware components of the system.	DTB can be superseded to other DTB which recognize illegal hardware.
uImage	uImage is a compressed kernel image.	To load another kernel, uImage can be superseded to other uImage.
Bootscript	Contain commands which u-boot supports	Define the address of uImage, rootfs, DTB, and CSF Header
RSA Key Pair	Using the OpenSSL RSA function, make the public key and private key.	RSA public key is used to sign the signature of boot images. RSA private key is used decrypt the signature to verify the integrity.
CSF Header	Contain Barker Code, public key, private key, the signature of boot image.	To verify the integrity of the boot image, compare booted image and decrypted signature of CSF Header.
Memory Mapping	Address where the boot images are saved when porting the boot images to T2080RDB.	User can store boot images at any address.

B. Risks to Components to Protect

The previously defined components to protect may be modified by the physical attack such as JTAG and insertion of the external media device and cyber-attack such as viruses, worms, and trojan horses. Table Ⅲ shows the risk that could arise if the components to protect were modified.

TABLE Ⅲ. Risk of Attack of Components

Components	***Risk of modifying***
U-boot	If U-boot is modified to normal U-boot, it cannot check the integrity even DTB, uImage, Rootfs is modified.
Rootfs	The modified Rootfs which contain virus can be mounted.
DTB	The modified DTB would recognize the unexpected device.
uImage	The modified uImage would disable every security solutions running on the kernel level. Allow attackers to access the stored data without any authentication.
Bootscript	The modified Bootscript can turn off the secure boot. Allow attackers to boot arbitrary boot images by modifying the address of images.
RSA key Pair	Allow attackers to generate the CSF Header using the modified private key.
CSF Header	Generate the CSF Header of the modified image for verification.
Memory Mapping	Attacker can store the modified image to any address.

C. Drawing Attack Scenario

When we generate attack scenarios based on components, there were several cases such as scenarios which only one component is modified or a scenario which various components were simultaneously modified. Therefore, there are many attack scenarios which can occur. If all the images simply substituted to another image, 2^8 scenarios can be derived as shown in Fig. 1.

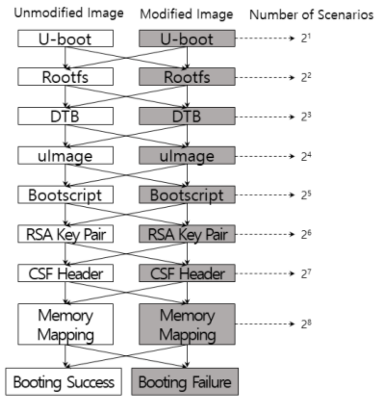


Fig. 1. Attack Scenario That Can Be Derived

If configurations of the image are changed or substituted with other images, there will be more scenarios. In this section, according to the attacker's primary goal, we make two attack scenarios for the case study.

One of the attacker's goal is to steal the data in T2080RDB. To steal the data, the attacker needs to turn off the secure boot of T2080RDB by substituting the secure U-boot to normal U-boot and delete the validate command in Bootscript. It can be expected that secure boot is turned off. Because secure boot is turned off, the attacker can modify the kernel and steal data as shown in Fig. 2.

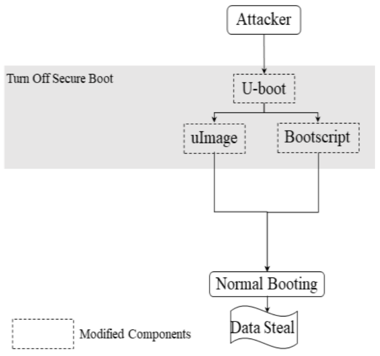


Fig. 2. Data Steal Scenario

Another attacker's goal is to get access. To get the access, the attacker needs to create new RSA key pair and modified CSF Headers on every image such as uImage, DTB, Rootfs, and Bootscript. Because every image is substituted to the attacker's image, it can be expected that the attacker can get the total access as shown in Fig. 3.

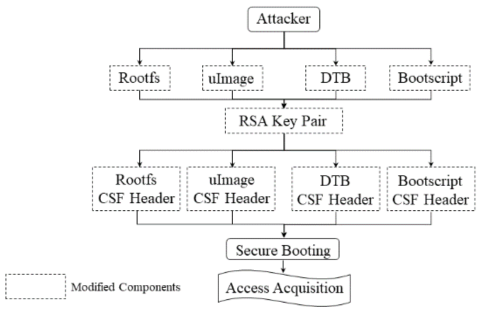
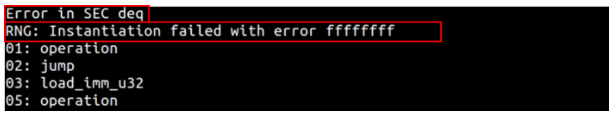


Fig. 3. Access Acquisition Scenario

Verification of security based on Scenarios

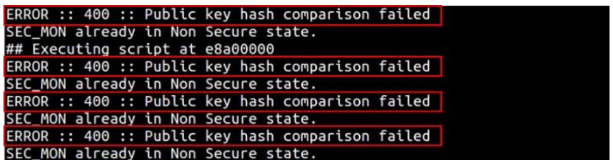
We got the two attack scenarios in the previous section. When we test the data steal attack, the attack was prevented in the RCW. Because once the RCW is programmed into ITS, it is impossible to reprogram it. Therefore, even if secure U-boot is substituted to normal U-boot and Bootscript is modified, secure boot does not turn off. As shown in Fig. 4 when we execute the attack scenario, the error message shows us that there is an error in SEC dequeue.



```
Error in SEC deq
RNG: Instantiation failed with error ffffffff
01: operation
02: jump
03: load_imm_u32
05: operation
```

Fig. 4. Error Message of Data Steal Scenario

When we test the access acquisition attack, the attack was prevented in SRKH. This is because once the SRKH is programmed into the security fuse processor, it is impossible to reprogram it. Therefore, even if a new CSF Header is created for the modified image, it cannot be decrypted because RSA public key which used to create the modified image has not been programmed into the SFP which has SRKH. As shown in Fig. 5 when T2080 secure boot verifies the images, error message was printed because the public key hash and SRKH are different.



```
ERROR :: 400 :: Public key hash comparison failed
SEC_MON already in Non Secure state.
## Executing script at e8a00000
ERROR :: 400 :: Public key hash comparison failed
SEC_MON already in Non Secure state.
ERROR :: 400 :: Public key hash comparison failed
SEC_MON already in Non Secure state.
ERROR :: 400 :: Public key hash comparison failed
SEC_MON already in Non Secure state.
```

Fig. 5. Error Message of Access Control Acquisition Scenario

IV. The Future of Verification Model

In this paper, we shared our experiences which building and executing verification model to test the secure boot against modify attack. We constructed the verification model in four steps of data collection, risk identification, scenario generation, and scenario verification. In our verification model, we could find numberless scenarios and verify the scenarios. However, this verification model is restricted to modify attack. Therefore, the verification model should be constructed according to services. Because the verification model has many forms and every service has its own verification model. If services find its own suitable verification model, it can enhance the security level of the service.

References

[1] SiChoon Noh, IkSoo Jun, Kuinam J. Kim, Verification Method of Insuring Web Application Security. Journal of Information and Security, 3(2), pp. 11-20. 2003.

[2] Myong-Yeal Lee and Jae-Pyo Park, An analysis on invasion threat and a study on countermeasures for Smart Car. Journal of the Korea Academia-Industrial Cooperation Society, Vol. 18, No. 3, pp. 374~380, 2017.

[3] Jeom Goo Kim, A Study on Evaluation Technique of Network Security System. Journal of Information and Security, 9(2), pp. 33-39. 2009.

[4] Balzarotti, D., Banks, G., Cova, M., Felmetsger, V., Kemmerer, R., Robertson, W., ... & Vigna, G. “Are your votes really counted?: testing the security of real-world electronic voting systems. “In Proceedings of the 2008 international symposium on Software testing and analysis July 2008, pp. 237-248, ACM.

[5] NXP Semiconductors, QorIQ T2080 Reference Manual, 2016.

[6] XILINX, Zynq-7000 All Programmable SoC Technical Reference Manual, 2018.

[7] Qualcomm Technologies, Secure Boot and Image Authentication Technical Overview, 2017.

[8] Curtis-Wright, VPX3-133-3U-VPX-NXP-T2080-Single-Board-
Computer-product-sheet, 2013.

[9] Curtis-Wright, VPX3-152-3U-VPX-NXP-T2080-Single-Board-
Computer-product-sheet, 2013.

[10] Curtis-Wright, VPX6-195-6U-VPX-Freescale-T2080-SBC-product-
sheet, 2013.

[11] NXP Semiconductors, QorlQ SDK v2.0-1703 Documentation, 2017.