Poster: PA-Boot: A formally Verified Processor Authentication Protocol for SMP Secure Boot

Zhuoruo Zhang, Rui Chang, Qinming Dai, Kui Ren
Zhejiang University
{zhangzhuoruo, crix1021, qinm_dai, kuiren}@zju.edu.cn

Abstract—**Multi-processor SoC systems are widely used nowadays. Symmetric multiprocessing (SMP) is a popular architecture for multiprocessor systems, which classifies the processors into a bootstrap processor (BSP) and multiple application processors (APs). The authenticity of APs is under-examined since the APs are trusted by default. However, it turns out to be vulnerable when introducing the supply chain attack aiming to replace or counterfeit the APs.**

In this paper, we systematically investigate critical stages of SMP system boot process, and discover a fundamental and non-trivial attack due to the lack of AP authentication. To mitigate the new attack, we propose PA-Boot, the first formally verified processor authentication protocol for secure boot of SMP systems. PA-Boot is proved to be functionally correct, and is guaranteed to detect different adversarial behaviors, e.g., replacement of AP, tampering with certificates, and man-in-the-middle attacks. The fine-grained formal specification and its fully mechanized security proofs are carried out in the theorem prover Isabelle/HOL with 305 lemmas/theorems and $\sim$7,000 LOC. We also implement PA-Boot in C as a prototype, and the performance and security evaluations show that it can identify boot process attacks efficiently and improve the security of SMP systems.

I. Introduction

Attacks during boot process are arguably the most difficult to defend against because at this stage in a device's lifecycle, traditional defences such as firewalls and anti-viruses are not in place, and attacks are difficult to detect [1]. A common defence against boot attacks is authenticated or secure boot [2]. Recent works have shown a growing interest in secure boot verification both in academia and industry. However, despite extensive research focusing on boot process for uniprocessors, little attention has been paid to whether and how these verification techniques can be applied to symmetric multiprocessing (SMP) systems. SMP refers to the computer architecture where multiple identical processors are interconnected to a single shared main memory, with full accessibility to all the I/O devices. Nowadays, given the increasing popularity of SMP systems, it is imperative to guarantee the security of their boot process.

In this paper, we focus on a fundamental but neglected problem for SMP secure boot, which is a specific type of attack stemming from the authentication of the application processors (AP). Specifically, while all the processors in a compliant system are functionally identical, SMP specification classifies them into two types: the bootstrap processor (BSP) and the application processors (AP). In the boot process, each processor first executes boot code in its own bootROM. Then, the BSP is responsible for initializing the system and booting the operating system; APs are activated only after the operating system is up and running with the same privilege as the BSP. General permission includes access to peripherals, memory or other hardware devices. Nevertheless, this mechanism renders a severe vulnerability. Specifically, an attacker can replace the BSP with a bootkit implanted one and tamper with the bootROM and mutable code. Therefore, this malicious AP can communicate with the BSP with the intent of getting access to secret information or authorized services, even tampering with the system runtime data.

The boot process has been developed to modern SMP systems for years, and not enough attention is paid to the security of the boot process since the APs are considered as trusted by default. However, it turns out to be vulnerable in our analysis when introducing the prominent supply chain attack. More specifically, the customer's device can be intercepted and implanted with additional chip or a modified version of the chip, either at the manufacturing site or on its way from the manufacturer to the customer. Known attacks using modified, replaced, or counterfeit chips raise the question of whether such a shift of trust is justified [3].

To meet the requirement for stronger security in such a scenario, we dig into the SMP system boot process and develop a processor authentication protocol, called PA-Boot, to validate the authenticity of APs. We also specify and verify PA-Boot based on formal techniques, providing a strong assurance of its correctness and security. In addition, guided by principles of practicality and performance, we implement PA-Boot as a prototype. It offers the user a trusted and efficient way to ensure whether the processor is genuine or has been manipulated. To the best of our knowledge, this is the first study that systematically verifies the authenticity of APs in SMP boot process based on formal methods.

In detail, the technical contributions of our work are as follows:

- **Systematical investigation:** We systematically investigate critical stages of SMP system boot process and discover a novel attack in AP authentication.
- **Formally verified PA-Boot:** To address the above problem, we propose a processor authentication protocol, named PA-Boot, that can satisfy security goals for secure boot in the presence of an active attacker executing simultaneously with the protocol. We provide a mechanically checked formal specification for PA-Boot

TABLE I: Security statistics for three types of attacks.

Error code: ❶=version error; ❷=datatype error; ❸=reserved space is not empty; ❹=length field error; ❺=verifying $Cert_{root}$ error; ❻=verifying $Cert_{BSP}$ error; ❼=total length error; ❽=RSA decryption error; ❾=DH public key error; ❿=verifying $Cert_{AP}$ error

Category	Tamper with Packets in Shared Memory				Tamper with Certificates in Flash			Replace the AP
Attack	P_{Cert}	P_{Chall}	P_{RC}	P_{Resp}	$Cert_{root}$	$Cert_{BSP}$	$Cert_{AP}$	AP'
Total Time	2,502	2,453	2,418	2,627	10,000	10,000	10,000	10,000
Time	1 1 1 5 1,272 1,222	6 6 12 14 4 2,411	1 3 8 12 6 783 1,605	8 4 8 12 6 870 1,219	10,000	8 9,991	8 9,991	10,000
Error Code	❶ ❷ ❸ ❹ ❺ ❻	❶ ❷ ❸ ❹ ❼ ❽	❶ ❷ ❸ ❹ ❼ ❾ ❽	❶ ❷ ❸ ❹ ❼ ❾ ❽	❺	success ❻	success ❿	❽

in Isabelle/HOL, a state-of-the-art interactive theorem prover. More precisely, we use 91 functions/definitions to specify the protocol model covering system behaviors, adversary capabilities and security properties. In order to verify integrity, authenticity, and confidentiality for SMP secure boot, 305 lemmas/theorems are proved to ensure that the model satisfies the security properties. About 7000 lines of Isabelle code are implemented in total.

- **Implementation and evaluation:** We implement PA-Boot in C based on a code-to-spec review and validate its performance and security by extensive evaluations. The results show that it offers adequate performance in practice and can improve the security of SMP systems.

II. Workflow

Figure 1 depicts PA-Boot's workflow. First, we find vulnerabilities in SMP processor authentication process, and design PA-Boot as a mitigation approach. Based on it, a threat model is explicitly described with respect to the attacker's fundamental target, security assets and three potential attacks. Then, security goals are defined coarsely in terms of integrity, authenticity and confidentiality. Afterward, following the informal threat model and security goals, we formally derive M_A in Isabelle/HOL [4], a high-level functional specification of PA-Boot. In addition, the security goals are formally specified into high-level security properties and are verified to be satisfied by M_A. Next, to describe a more detailed design of PA-Boot, we specify M_C, which is an instance of M_A. It succinctly captures the key attributes and behaviors of various components in the threat model. Based on M_C, we extend each of the high-level security properties into a low-level one that is stronger and more precise. M_C is proved to hold all of them based on refinement relation. Afterward, based on human review and logical equivalence transformation from M_C, a practical processor authentication protocol is implemented in C. As it follows the formally verified protocol design, it is guaranteed to meet our security goals. Finally, we evaluate the protocol in terms of performance and security, and the results indicate its high efficiency and security.

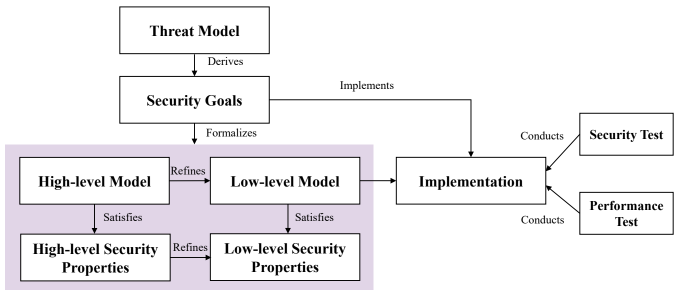


Fig. 1: Workflow

III. Implementation and Evaluation

We implemented our PA-Boot with about 3000 lines of C code as a prototype, along with roughly 1000 lines for performance and security tests. Specifically, we measure the running time of the certificate generation stage and the processor authentication stage 10,000 times each. The average running time is around 28ms for certificate generation, and around 110ms for processor authentication. For security evaluation, we simulate three types of attacks, i.e., man-in-the-middle attacks, flash compromise, and AP replacement, 10,000 times each. We then record the return value to analyze if our system can discover the attacks as expected (see Table I). In summary, the results show that PA-Boot offers adequate performance and completely complies with our security goals.

IV. Conclusion

The security of boot process depends on the weakest link in the authentication chain. This paper address this point by proposing a formally verified PA-Boot to mitigate the new boot attack resulting from AP replacement in SMP system. This paper intends to rethink the security design of SMP boot process and motivate the researchers/developers to draw more attention to the "known-safe" or "assumed-safe" components in the existing systems. Therefore, this work is of great value for identifying boot process attacks and improving the boot ecosystem's overall security.

References

[1] Zhe Tao et al., "DICE*: A Formally Verified Implementation of DICE Measured Boot", in 30th USENIX Security Symposium, 2021.

[2] J. Douglas et al., "Dyad: A System for Using Physically Secure Coprocessors", in Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment, 1991.

[3] Felix Bohling et al., "Subverting Linux' integrity measurement architecture", in Proceedings of the 15th International Conference on Availability, Reliability and Security, 2020.

[4] Nipkow et al., "Isabelle/HOL: a proof assistant for higher-order logic", in Springer Science & Business Media, 2002.

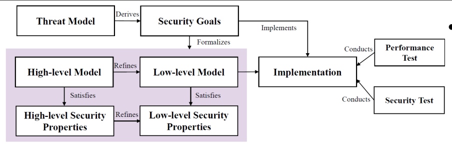
Poster: PA-Boot: A formally Verified Processor Authentication Protocol for SMP Secure Boot

Zhuoruo Zhang, Rui Chang, Qinming Dai, Kui Ren
ICSR, Zhejiang University

1. Contribution and Workflow

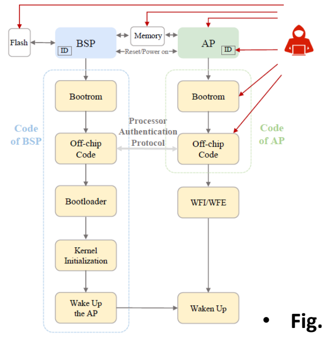
Discovery, Design, Verification, Implementation, Evaluation

- **Discovery of a new attack.** After a systematic investigation in SMP system secure boot, find a fundamental but neglected vulnerability due to the lack of AP.
- **Proposing a mitigation method**. Define a threat model describing the attacker's fundamental target, security assets, and three types of attacks. Based on this, define security goals in terms of integrity, authenticity and confidentiality. Design a processor authentication protocol, called PA-Boot, that can satisfy the security goals in the presence of an active attacker executing concurrently with the protocol.
- **Formal Specification and Verification**. Formalize the protocol and verify it against the security properties. The mechanical verification is performed in Isabelle/HOL, a state-of-the-art interactive theorem prover.
- **Implementation and evaluation**. Implement the protocol in C and validate its effectiveness, scalability and security by extensive evaluations.

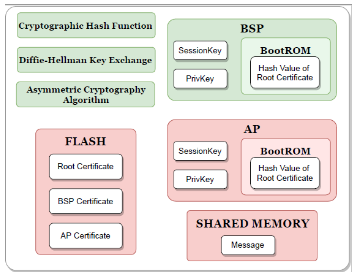


- **Fig. 1:** *Workflow.*

2. Motivation and Threat Model

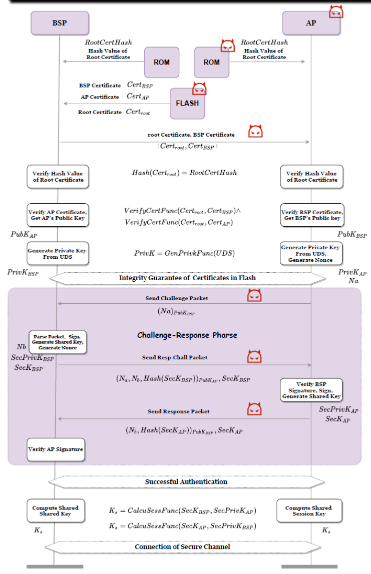


- **Fig. 3:** *Security assets in threat model.*


Cryptographic Hash Function
Diffie-Hellman Key Exchange
Asymmetric Cryptography Algorithm
BSP
SessionKey
PrivKey
BootROM
Hash Value of Root Certificate
FLASH
Root Certificate
BSP Certificate
AP Certificate
AP
SessionKey
PrivKey
BootROM
Hash Value of Root Certificate
SHARED MEMORY
Message

- **Fig. 2:** *Adversarial behaviors.*

3. Protocol Design


BSP
AP
RootCertHash
Hash Value of Root Certificate
ROM
ROM
RootCertHash
Hash Value of Root Certificate
BSP Certificate $Cert_{BSP}$
AP Certificate $Cert_{AP}$
Root Certificate $Cert_{root}$
FLASH
root Certificate, BSP Certificate
Verify Hash Value of Root Certificate
$Hash(Cert_{root}) = RootCertHash$
Verify Hash Value of Root Certificate
Verify AP Certificate, Get AP's Public Key
Verify BSP Certificate, Get BSP's Public Key
$PubK_{AP}$
$PubK_{BSP}$
Generate Private Key From UDS
$PrivK = GenPrivkFunc(UDS)$
Generate Private Key From UDS, Generate Nonce
$PrivK_{BSP}$
$PrivK_{AP}$
Integrity Guarantee of Certificates in Flash
Send Challenge Packet
Challenge-Response Phase
Parse Packet, Sign, Generate Shared Key, Generate Nonce
Send Resp-Chall Packet
Verify BSP Signature, Sign, Generate Shared Key
Send Response Packet
Verify AP Signature
Successful Authentication
Compute Shared Session Key
Compute Shared Session Key
Connection of Secure Channel

Certificates Validation. The immutable hash value of root certificate is stored in bootROM, serving as a chain of trust. Use this to help validate the legality of certificates.

Challenge-Response. Obtains the other processor's public key from the validated certificate for encrypted communication. After authentication, use DH algorithm to generate shared session key for subsequent secret communications between the two processors.

- **Fig. 4:** *The normal flow of PA-BOOT, vulnerabilities marked with red devils.*

4. Formal Specification and Verification

- Formalize PA-Boot as a state machine, consisting of states and events. Formalize security goals as three security properties.
- Prove if the model satisfies the security properties.

Security Properties Covering all Execution Traces

- **Security Property under Normal Execution:** Starting from initial state where the initial configuration has not been tampered, and there is no adversary behaviors during execution, then the final state must be an ideal state
  $$If\ \Gamma \vdash \langle ops, s\rangle \Rightarrow t, \mathcal{P}(ops)\ and\ \mathcal{R}(t)\ \ then\ \mathcal{V}(t)$$
- **Security Property under Tampered Configuration:** Starting with compromised initial configuration, the system should terminate in a bad state
  $$If\ \Gamma' \vdash \langle ops, s\rangle \Rightarrow t\ and\ \mathcal{R}(t)\ \ then\ \neg\mathcal{V}(t)$$
- **Security Property under Adversarial behaviors:** Any adversary behaviors during system execution should be detected, resulting in a bad final state
  $$If\ \Gamma \vdash \langle ops, s\rangle \Rightarrow t, \neg\mathcal{P}(ops)\ and\ \mathcal{R}(t)\ \ then\ \neg\mathcal{V}(t)$$

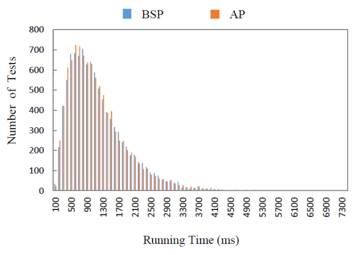
5. Implementation and Evaluations

Table 1: *Statistics for specification, proofs and implementation.*

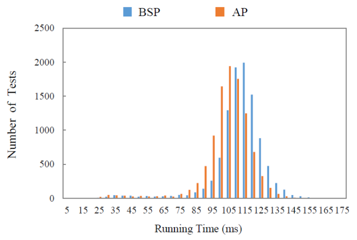
Isabelle Code (~7100LoC)					C Code
Item	Specification		Proofs		
	Locale/Definition	LoC	Lemma/Theorem	LoC	
High-level Specification M_A	1	~50	1	~50	3,680 LoC
Low-level Specification M_C	90	~850	304	~6,150	
Total	91	~900	305	~6,200	

Implement PA-Boot in C, and Evaluate It in Two-fold

- **Performance:** Measure the running time of the certificate generation stage and the processor authentication stage, respectively.
- **Security**: Simulate three types of attacks, i.e., man-in-the-middle attacks, flash compromise, AP replacement, 10,000 times each. Record the return value to analyze if our system can discover the attacks as expected.



Runtime of certificate generation stage



Runtime of processor authentication stage

Table 2: *Security statistics for three types of attacks.*

Error code: ❶=version error; ❷=datatype error; ❸=reserved space is not empty; ❹=length field error; ❺=verifying $Cert_{root}$ error; ❻=verifying $Cert_{BSP}$ error; ❼=total length error; ❽=RSA decryption error; ❾=DH public key error; ❿=verifying $Cert_{AP}$ error

Category	Tamper with Packets in Shared Memory				Tamper with Certificates in Flash			Replace the AP
Attack	P_{Cert}	P_{Chall}	P_{RC}	P_{Resp}	$Cert_{root}$	$Cert_{BSP}$	$Cert_{AP}$	AP'
Total Time	2,502	2,453	2,418	2,627	10,000	10,000	10,000	10,000
Time	1 1 1 5 1272 1222	6 6 12 14 4 2411	1 3 8 12 6 783 1605	8 4 8 12 6 870 1219	10,000	8 9,991	8 9,991	10,000
Error Code	❶ ❷ ❸ ❹ ❺ ❻	❶ ❷ ❸ ❹ ❼ ❽	❶ ❷ ❸ ❹ ❼ ❾ ❽	❶ ❷ ❸ ❹ ❼ ❾ ❽	❺	success ❻	success ❿	❽

Conclusion: PA-Boot offers adequate performance and completely complies with our security goals.

Contact: Zhuoruo Zhang (zhangzhuoruo@zju.edu.cn), Rui Chang (crix1021@zju.edu.cn), Qinming Dai (qinm_dai@zju.edu.cn) and Kui Ren (kuiren@zju.edu.cn).