Practical steps to evaluate and protect Secure Boot implementations on embedded devices

Riscure | The Netherlands

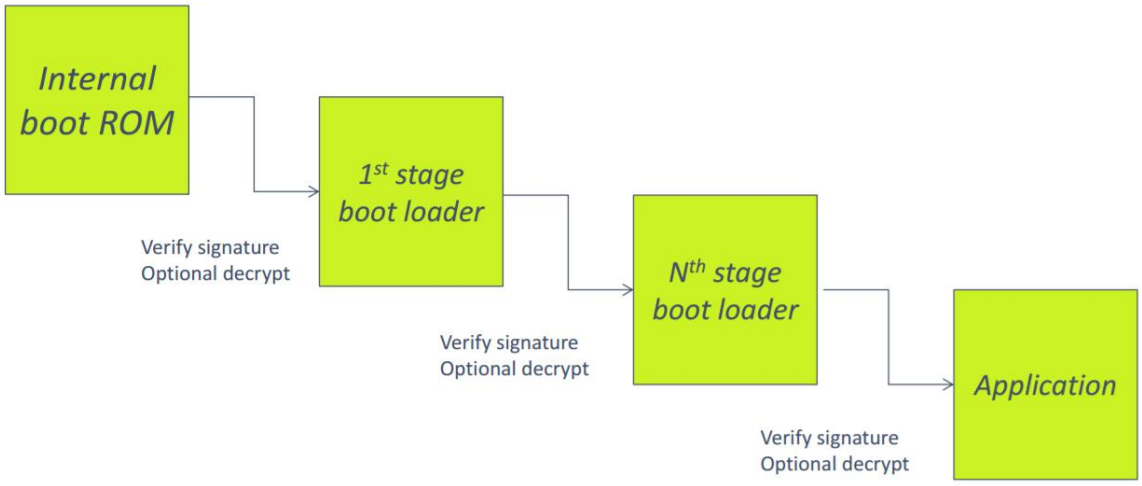
Executive summary:

The majority of mobile, embedded and IoT systems implement some form of Secure Boot to preserve the integrity, protect senstive content and operations on a device, and reduce chances of a severe compromise of a system. Smartphones and tablets, IoT devices, industrial automation systems, game consoles and other solutions rely on Secure Boot: a critically important method of setting the chain of trust. Although Secure Boot alone does not guarantee reliable protection from all types of attacks on devices, its compromise proves to be the most damaging, leading to risks of financial, reputational and legal damage for a vendor. Existing requirements for immunity of Secure Boot are high, yet Riscure's security experts typically find serious vulnerabilities in the majority of implementations during the first stage of security evaluation. This paper reveals common weaknesses in Secure Boot implementations and proposes practical steps to enhance security of this critical element of an embedded system.

1 The importance of protecting Secure Boot

Secure Boot solves a relatelively simple task of initializing a computer system: i.e. running a small piece of code which then gives way to a more complex software – either firmware of an embedded device or a feature-rich operating system of a smartphone or a computer. A while ago this operation was scarcely protected, but because of ever increasing malicious attack attempts, at some point it became critical to verify and control the integrity of a code being run on a system. Secure boot is implemented to protect the intellectual property contained in a firmware, check the integrity of software code to make sure it was not modified for a malicious purpose, maintain security of protected content such as a console game, video/audio stream, mobile payment authentication or sensitive user data. Thus, reliability of the Secure Boot mechanism benefits both manufacturers and consumers.

In theory, the Secure Boot mechanism involves a series of stages, in each stage a new piece of code is executed, after necessary checks for integrity and optional decryption of protected content. Due to a wide variety of embedded platforms, in practice every system typically has its own unique specifics.



The diversity inevitably leads to an increased attack surface and potential compromise of Secure Boot can lead to significant damage to an OEM, customer and/or end user. Examples of successful attacks are frequently found in the news: circumvention of copyright protection in gaming consoles and set-top boxes, jailbreaking of smartphones, attacks on engine control units in automotive and so on.

Riscure has 16 years of experience analyzing security of embedded systems and has been evaluating Secure Boot implementations for more than 10 years. On average, our experts typically break security of 90% of implementations during the first stage of evaluation. Despite the relative simplicity of Secure Boot compared to the rich environment of Android or Windows operating systems, which allows better control of protection features, there is a clear need to improve protection of this critical element of almost every device.

2 The security insight: common weaknesses

Every Secure Boot implementation is somewhat unique, which is why it is virtually impossible to foresee all potential problems. Evaluating security of a system from the attack side, an approach that is being successfully embraced by Riscure, pursues a different goal:

- We put systems under test using both the latest and well-known attack methods;
- We accumulate security expertise on a particular subset of systems (embedded devices)
- We share this intelligence with hardware manufacturers and software developers, so that they can use it to improve their products.

Riscure has a very strict policy on disclosure of new attack methods and vulnerabilities found in our customers' systems. The scope of security problems related to Secure Boot presented below is therefore incomplete: only well-known methods, already used in-the-wild and publicly disclosed are included, however they provide a cornerstone for the majority of secure boot problems. In this document we have divided security challenges in two groups: logical and physical. Logical vulnerabilities are more common and more frequently exploited, but are often easy to fix via the means of a software update. Physical vulnerabilities are less frequently exploited, some of them require a major effort from an attacker, but when a compromise occurs, it is often impossible to mitigate a problem on devices already in circulation. Applying a fix to a design that has already entered production stage also comes at a cost.

A view on system security from an attackers perspective does not isolate logical and physical vulnerabilities. In many cases a newly identified attack involves a combination of the two, and a major compromise of a system may be triggered by a series of seemingly benign software and hardware issues. Also, attackers may well use a hardware attack to identify software vulnerabilities which then in turn can be used to attempt a highly scaleable logical attack. A proper security evaluation therefore offers an independent view of an entire application, delivers a proper security insight and leads to cost savings, if involved early in the design phase.

Logical threats

Challenge	Example	Mitigation
Design Error	A serious programming error that significantly harms security of a system: acceptance of an empty signature as a valid one, circumvention of an integrity check by a certain flag in the code.	Human errors are possible in the development process. The only approach to reduce the impact of such errors is to educate developers on the secure programming techniques and conduct a design review from a security perspective.
Boot ROM accessibility during normal device operation	Partial or full availability of binary code of a Secure Boot implementation can simplify the process of an attack: the code can be reviewed and	Access to unencrypted ROM code should be disabled after the boot sequence finishes.

	reverse engineered to look for bugs, identify potential logical attacks and other weaknesses.	
Cryptographic implementation errors	A diverse subset of errors, includes reuse of cryptographic engine in full, or in part, after the boot sequence. This presents a possibility to deconstruct the encryption/decryption process and intrude during the boot process.	Both keys and data registers of cryptographic engines have to be erased from the memory after the boot sequence finishes.
Firmware Upgrade issues	The ability to circumvent device protection typically comes not from the new firmware itself, but from an upgrade mechanism. Additional functionality may be exploited, if not secured properly.	A general mitigation practice is to reduce functionality of an upgrade mechanism to a bare minimum and prevent rollback to a previous firmware version.
Service backdoor	Hardcoded functionality circumventing protection for engineering/debug or customer support purposes is a frequent cause of a security breach.	Mitigation depends on use case, at best it is recommended to avoid any 'backdoor' functionality in production. Can be identified via Design Review assessment and such evaluation methods as Model Based Testing and Fuzzing.
Insecurity of State variables	Insecure handling of a system state variable may lead to maximum privelege escalation. For example, exploitation of suspend/resume state may be used to replace original firmware with a modified one.	Design review of all state variables in the boot sequence, such as suspend/resume, exception handling, storage, integrity from the security perspective. State variables have to be resistant to both software-based and physical attacks (see more below on Fault

		Injection).
Driver weaknesses	Weak security of drivers (handling interfaces like USB, MMC, UART, etc) may lead to exploitation.	Code review and penetration testing to identify weaknesses in software. Potential risks can be addressed by limiting functionality of a driver to a minimum.
Relying on decryption for authentication	Misinterpreting successful decryption of boot code as verification. Reversing the encryption process by an attacker leads to a compromise.	It is recommended to always implement code encryption together with signature verification..
Code signing errors	A scenario where a smaller part of the code remains unsigned may still lead to a security breach.	All parts of the boot image have to be verified for authenticity, including headers, pointers and addresses.
Weak key management	A typical problem is when an adversary obtains signing keys. Signing development boot loaders with production keys may contribute to the problem.	Proper key management with detailed policy for storage, access, lifetime and revocation. Separate keys for development purposes to limit exposure.
Weak signing methods	General weakness of a code signing method that leads to a compromise.	Cryptographic review to ensure proper implementation of cryptographic algorithm in use.

Hardware threats

Problem	Example	Mitigation
JTAG/UART and other means of debug access to boot stage	Availability of a debug access to boot stage in a production device with high privileges often allows to bypass	Disabling or locking debug interface, if possible. Camouflaging debug ports is achievable on a vendor side,

	security mechanisms entirely. Frequently introduced by a chip manufacturer and cannot be fully controlled by an OEM.	although it provides only minimal protection.
Selectable boot source	Ability to select a boot source leads to a security breach.	Disabling undesired functionality. Elimintating the option for unauthenticated booting.
Race Condition	Typical case is when integrity checking and execution is conducted on a code stored externally. Stored code is altered after integrity check, and before execution.	Protection of external storage medium to eliminate the possibility of code modification.
Timing Attacks/Side Channel Analysis	Utilizing side channels (such as electromagnetic emissions of a chip) to extract a secret or, in general, identify boot stages with a potential for another type of attack.	Evaluating algorithms that 'leak' sensitive data via side channel and modifying them.
Fault Injection	Deliberate introduction of 'faults' such as non-standard voltage or electromagnetic pulse to disrupt normal device operation.	Detailed in Section 3 of this document.

3 Fault Injection attacks against Secure Boot

Compared to logical and simple physical (see JTAG example ebove) attacks, Fault Injection can be considered an 'elite' attack method. It is often overlooked by developers in light of more abundant logical problems, but we believe it deserves additional attention. Unlike Side Channel Analysis that only monitors the state of hardware, Fault Injection is often an invasive method. Physical intrusion in operation of a system requires an immense number of trials to identify a successful attack scenario, with expensive hardware and significant expertise. As a result, there are just a few *publicly disclosed* examples of successful Fault Injection attacks, such as the "reset glitch" circumvention of Secure Boot routine of Xbox 360 gaming console. However, like any other complex attack method, the cost of Fault Injection tends to decrease over time.

We believe that embedded systems developers need to improve their Fault Injection competencies, even though it is still easier to attack their products via logical errors:

- Vulnerabilities exploited via Fault Injection do not always require presence of a logical vulnerability. Therefore, even a product with 'flawless' software can be successfully attacked.

- Fault Injection vulnerabilities are hardware-based, and it is not always possible to mitigate them via a software update.
- Fault Injection hardware and methods are becoming ever more accessible. A simple, although very limited setup can already be assembled for less than 250 USD.
- Rapid adoption of smart and IoT devices in all aspects of our lives will attract interest of research community (which will benefit security) and cybercrime (when they see a potential for return-on-investment).
- The majority of embedded systems can now be attacked via logical vulnerabilities. Yet, as the industry matures, security of future products will be improved. At some point Fault Injection attacks will become mainstream, since they offer an opportunity to go as far as full firmware extraction and/or identify easily scaleable logical attacks on devices with a large numeric or global presence. Given that certain embedded systems (for example, smart meters and industrial automation) have a life span 10+ years, it is better to protect them now, in order to avoid costly recalls and reputation damage in the future.

Typical Fault Injection routines introduce 'glitches' by altering the clock and/or the operating voltage. Normal operation can also be disrupted via the means of an alectro-magnetic or laser impulse. As a result, a device usually halts its normal operation or is physically damaged, but under certain circumstances (identified by the process of trial and error) the 'glitch' may allow to circumvent Secure Boot protection methods. The goals pursued by Fault Injection include creating the condition to execute arbitraty code, access unencrypted Boot code for further assessment, identify the cryptographic algorithms in use or to force the device to 'leak' cryptographic keys. Fault Injection may also lead to inception of one of the the logical vulnerabilities listed above such as modifying the system state for further exploitation. The success of a Fault Injection attack relies on physical properties of hardware as well as certain software programming approaches that are prone to 'glitching'.

Therefore, the mitigation of threats related to Fault Injection relies on three methods:

- Test your product for Fault Injection vulnerabilities in-house or with a security lab.
- Embrace secure programming methods that reduce the risk of a successful Fault Injection attack (although it cannot be eliminated completely, the success threshold can be raised substantially).
- Introduce countermeasures to detect Fault Injection attacks.

4 The solution: Hard work and No silver bullet

In this paper we have covered a variety of vulnerabilities that can be exploited to circumvent the protection of Secure Boot implementations, from simple errors such as a software backdoor or JTAG override, to the most sophisticated flaws that can be discovered via Side Channel Analysis or triggered via Fault Injection. Unfortunately, they cannot be addressed by a single security method. A typical approach to security involves addressing vulnerabilities one by one, as they are discovered internally or by a third party, at best ensuring a balance between investment in security intelligence and losses inflicted by successful attacks. We think that a better approach is to speed up this process dramatically by following the three directions of the security-by-design strategy:

1) Train your software developers to deliver secure code. Help them view their code from an attacker's perspective to spot vulnerabilities and thus avoid typical attack scenarios.

2) Conduct a comprehensive security evaluation of your product, internally or externally. Doing this earlier in the development process simplifies subsequent certification and reduces a risk of a costly redesign as a product enters production stage.
3) Build your own security intelligence. Invest in 'future mainstream' Fault Injection and Side Channel Analysis evaluation methods so that your products stay secure during their entire lifecycle.

5 What Riscure offers: Tools, Services and Training

Riscure specializes in embedded systems security, with 16 years experience in security evaluations, strong R&D for security tools and 90+ highly skilled experts. We offer the comprehensive portfolio of Tools, Services and Training that help our customers deliver protected software and hardware solutions. The goal is to share our knowledge with our clients, and we believe this is the better way of securing the embedded world.

Tools

Riscure is the leading developer of specialized Side Channel Analysis, Fault Injection and White Box Crypto evaluation tools. The state-of-the-art Inspector FI solution includes more than 10 hardware modules to cover all aspects of Fault Injection evaluations powered by custom-built software for automated assessment. Combining this with support and training programs, we provide substantial cost savings on research and development for academic institutions, hardware manufacturers and government agencies around the world.

Read more about Riscure Tools. Contact Tools sales team.

Services

Riscure provides security evaluation and certification services for embedded systems and applications developers. We offer tailor-made evaluations to enhance security, save time and budget by avoiding costly design changes, speed up the certification process for customers in Automotive, Electronic/Mobile payment and PayTV industries, as well as IoT vendors. We are an officially accredited lab for such certification programs as EMVco, Common Criteria and GlobalPlatform.

Read more about Riscure Tools. Contact Services sales team.

Riscure Training Academy

Riscure Training Academy was born to serve the needs of our Tools customers, who requested to be trained on specifics of Side Channel Analysis and Fault Injection Methods. Now we offer a comprehensive secure development training program with the goal to share knowledge on Reverse Engineering and Software Exploitation techniques, and help our customers build robust embedded systems applications. Specialized training courses are also available, sharing practical intelligence on security of White Box Cryptography systems and SmartCards. Combined together, Tools, Services and Training help our customers integrate the best security practices in all stages of product development, from design to certification. Customized education programs are also available, including on-site training and Security Coaching to help businesses improve their security intelligence according to their unique specifics.

Read more about Training Academy. Contact Training sales team.

riscure

www.riscure.com

Riscure BV
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands

Phone: + 31 (0) 15 251 4090
inforequest@riscure.com

Riscure North America
550 Kearny St. Suite 330
San Francisco, CA 94108
USA

Phone: +1 (650) 646 9979
inforequest@na.riscure.com

www.riscure.com

Riscure BV
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands

Riscure North America
550 Kearny St. Suite 330
San Francisco, CA 94108