20 ways past secure boot

Job de Haas
Riscure Security Lab
Who am I …

Job de Haas

• Principal Security Analyst at Riscure
• Testing security on: Set-top-boxes, mobile phones, smart cards, payment terminals, ADSL routers, VoIP modems, smart meters, airbag controllers, USB tokens, …
• Before: Pentesting network security (since 1991)

Riscure

• Services: Security Test Lab
• Product: Side Channel Tools
• Full range testing: detailed hardware to white-box crypto and obfuscation
Overview

• Introduction on secure boot

• Hardware related threats

• Demo

• Logical threats
Secure boot?

• Not talking about UEFI
• Not talking about Microsoft lockdown
• Does not mean it does not apply

Lockdown

The coming war on general-purpose computing

By Cory Doctorow - Share this article


HITB KUL2013
Secure boot everywhere
Targets have in common

- Want to protect against persistent attacks
- Often nearby
- More often than not: the user

Double edged sword:
- Protects users against evil agencies and common thieves
- Protects corporations against their users
- Can deny users control of their hardware
Secure boot theory

- Root key internal
- Chain of trust
Secure boot example

- Secure boot failure:
  - Arbitrary code execution
  - Possible persistent attacks
  - Stepping stone for further attacks
20 ways to ...

- Did not try to classify and cross-classify all weaknesses
- Many different ways to count them
- Tried to find sufficiently different ones…
Hardware related threats
20. debug access to boot stage (JTAG)

- JTAG can allow full low level control of execution
- Can be very difficult to do without in production
- Physical complexity of connecting is overestimated

Mitigation:
- Secure designs can disable or lock JTAG
- Solution is chip dependent
19. Debug/service functionality

- UART is almost as persistent as JTAG
- Many devices leave some form of access for debug/service purposes
- What is the point of using u-boot to check the signature of the kernel, while commands are present like:

  ⇒ help mw
  mw - memory write (fill)
  Usage: mw [.b, .w, .l] address value [count]

- Example: Nook boot lock exploit (2012)
Nook boot UART exploit

• Post by hkvc:

UBOOTPROMPT> md.l 80e84808
----- This should show 1a00000a
...
UBOOTPROMPT> mw.l 80e84808 e1a00000
------------- This modify with NOP

UBOOTPROMPT> md.l 80e84808
----- should show e1a00000
...
UBOOTPROMPT> mmcinit 0; fatload mmc 0:1 0x81000000 flashing_boot.img; booti 0x81000000

Now it should boot with out giving a signature error.

Mitigation:
• Every chain in the boot process matters
• At least use some device unique authentication
18. Overriding boot source medium

- Boot source is selectable. Can a user override it (straps)?
- Does a system have different rules based on source?
- Also very effective as stepping stone (no brick)

- Automotive ECU’s have a ‘boot pin’
- JIG’s sold to reflash/remap firmware

Mitigation:
- Disable undesired functionality
- There should not be any unauthenticated exception for booting

Source: http://www.briskoda.net
17. TOCTOU race conditions

- Integrity check is performed on content in external storage
- Then the code is read or directly executed from the external storage
- Typical case: boot from external NOR flash
- Attack: After the integrity check alter stored code
- Nokia BB5 unlock by Dejan Kaljevic (2007):

**Mitigation:**
- Protect the memory interface for code execution
- Load code in (D)RAM
16. Timing attacks

- May allow guessing much faster than brute-force
- Typical on compare (HMAC)
- Hash calculated with symmetric key is stored with firmware. Boot calculates same and compares (20 bytes)
- memcmp has different timing if byte is correct or wrong
- Example: Xbox 360

Mitigation:
- Side channel leakage review
Timing attack with Infectus board

source: http://beta.ivancover.com
XBOX 360 timing attack procedure

- Store rogue bootloader
- Init hash in memory
- Init hash byte counter
- Reset XBOX
- Observe failure
- Register time
- Increase hash byte
- Reset XBOX
- Observe failure
- Later?
- Yes
- Final?
- Yes
- Success!
- No
- Increase byte counter

Brute forcing $16 \times 128 = 2048$ values takes about 2 hrs.
15. Glitch sensitivity

- Glitching is an effective way to subvert execution flow
- Examples of glitch sensitive coding:
  - using infinite loops
  - single comparisons (signature verification)
  - binary layout (return skipping)
  - using external memories
- Seldom a persistent attack; effective as stepping stone
- PS3: http://rdist.root.org/2010/01/27/how-the-ps3-hypervisor-was-hacked/
- XBOX 360: reset glitch attack: http://www.free60.org/Reset_Glitch_Hack

Mitigation:
- Fault injection review:
Examples of glitch sensitive code

HITB KUL2013
Glitch demo

• XMEGA target
• Fake secure boot implementation:
  ▪ Mimics signature verification
  ▪ Prints message

• Goal:
  Manipulate the target to failing the signature check and execute main code

• Method:
  ▪ Electromagnetic Fault Injection
Is it a real attack?

- Slot machine EMP jammer
Slot machine EMP jamming

http://www.youtube.com/watch?v=dew0KD_-ypw
secure_boot = fake_signaturecheck();

if (secure_boot) {
    sprintf(counter_msg, "Secure booting!\n");
    for (i=0;counter_msg[i] != 0; i++) {
        serial_send(counter_msg[i]);
    }
} else {
    sprintf(counter_msg, "Insecure booting!\n");
    for (i=0;counter_msg[i] != 0; i++) {
        serial_send(counter_msg[i]);
    }
    while(1);
}

... 

sprintf(counter_msg, "Lets go!\n");
Typical FI set up

Configure

Data

Reset

Inject Fault
EM-FI Transient Probe
Research probes

The EM-Probes from left to right: Probe 1, 2.3, 2.4, 2.5, 3, and 4

<table>
<thead>
<tr>
<th>Probe Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probe 1</td>
<td>Horizontal coil, 4mm diameter, ferrite core</td>
</tr>
<tr>
<td>Probe 2.3</td>
<td>Vertical coil, 3mm diameter, no core</td>
</tr>
<tr>
<td>Probe 2.4</td>
<td>Vertical coil, 4mm diameter, no core</td>
</tr>
<tr>
<td>Probe 2.5</td>
<td>Vertical coil, 5mm diameter, no core</td>
</tr>
<tr>
<td>Probe 3</td>
<td>Horizontal coil, 4mm diameter, EP5 ferrite core</td>
</tr>
<tr>
<td>Probe 4</td>
<td>Vertical coil, 4mm diameter, ferrite core</td>
</tr>
</tbody>
</table>
DEMO
Logical threats
13. Design mistakes

• Making wrong assumptions or adding risky features

Examples:
• Empty signature is accepted as good
• One flag means: no signing necessary
• Early removal of signature: iPhone 2G,3G (2008)
• [http://theiphonewiki.com/wiki/Pwnage](http://theiphonewiki.com/wiki/Pwnage)

Mitigation:
• Design review / Implementation review
12. Accessibility of boot ROM after boot

- Having access to the binary code of a boot rom allows detailed analysis
- Useful for:
  - Logical attacks
  - Locating glitch points
- Value is difficult to quantify:
  - Also in closed ROMs bugs are found
  - Breakthrough in some cases was clearly delayed
- Examples: original Xbox, iPhone, etc.

Mitigation:
- Disable ROM access (when leaving ROM execution)
- Execute-only ROM (less secure, hard to use)
11. Crypto sanitization

- After the boot code uses cryptographic engines they may become available for generic code
- State can be reused, registers may be read
- Attack: create more signatures, decrypt/encrypt more code

**Mitigation:**

- Clear key and data registers of crypto engines and any other memory used for storing sensitive data
- Better too much than too little
10. Firmware Upgrade / Recovery flaws

- Important feature to mitigate flaws in the field
- Don’t worry about the firmware update, but worry about the mechanism itself
- Updated firmware should follow same rules as installed fw

Examples:
- Many phone and game lock-down mechanisms subverted

Mitigation:
- Limit the functionality!
- Prevent rollback: can negate fixes
- Better to have ‘debug upgrade’ than debug built-in
9. Relying on unverified code

• Typical example: verified (ROM) code copied to RAM and used later
• Runtime flaws can lead to code modification before use

Examples:
• iPhone: http://rdist.root.org/2008/03/17/apple-iphone-bootloader-attack/
• SamyGo.tv: RSA disabler application (2010)

Mitigation:
• Using a single instance of critical code is good; do not copy but execute in place (ROM)
8. Service backdoor / password

- Everyone understands this can be bad
- More often: “It is bad, but not for my application”
- And then later the application requirements change
- Strong solutions require significant infrastructure

**Examples:**
- Many car tuning ECU cables/software
- ‘Magic’ authentication allows firmware mods, changing car keys, mileage

**Mitigation:**
- Depends on use case
- Make use of connected world to improve possibilities
Typical bootloader screens

![Typical bootloader screens](image-url)
7. State errors

- Where is state stored?
- How can a state sequence be influenced?
- Suspend/resume example: State is stored insecurely, which allows a local exploit to subvert the boot process on resume → maximum privilege escalation

Mitigation:
- Analyze all state variables in the boot sequence (exception handling, suspend/resume, storage, integrity)
- Consider both logical and fault injection threats
Custom boot loader menu

Wallaby Patch Tool 1.3/5.14

- Show PW stats
- Deactivate PW
- Activate PW
- Wipe PW
- Return Main

Searching ...
Found Heap at AC0C7000
Password not set and deactivated
Press ACTION
6. Driver weaknesses

- Boot code has several functions:
  - Boot from different media including file system (USB, SD, MMC, UART, NOR, NAND, SPI)
  - Ensure fall back and restore mechanisms
  - Perform parsing of firmware image formats
- Input parsing problems can lead to overflows, integer sign problems, etc. etc.
- Example: iPhone exploits
  - [http://theiphonewiki.com/wiki/Usb_control_msg(0xA1, _1)_Exploit](http://theiphonewiki.com/wiki/Usb_control_msg(0xA1, _1)_Exploit)
  - [http://theiphonewiki.com/wiki/Limera1n_Exploit](http://theiphonewiki.com/wiki/Limera1n_Exploit)

**Mitigation:**
- Code review, fuzzing,
- Limiting functionality to bare minimum, code reuse
5. ROM patching functionality

- Desired for maximum in-field updataibility
- Hook based techniques
- Can act as a boomerang
- Used in smart cards both for security fixes as exploitation

Mitigation:
- Don’t use it?
- If you need it, think again how to limit attacker possibilities

Source: http://www.innoozest.com
4. Decryption ≠ Authentication

- Some schemes add encryption of boot code
- Some misinterpret this for authentication / integrity
- ECB, CBC mode all allow small changes

Example:
- Nokia DCT4 2\textsuperscript{nd} stage loader u\_2nd.fia could be patched to load unencrypted 3\textsuperscript{rd} stage
- \url{http://www.dejankaljevic.org/download/dct4_rd.zip} 2002/2005

Mitigation:
- Always verify authenticity
- First verify, then decrypt
3. Inappropriate signing area

• If anything is left unsigned, what can it be used for?

Examples:
• iPhone 3GS, Samsung Galaxy S4
• http://theiphonewiki.com/wiki/0x24000_Segment_Overflow
• http://blog.azimuthsecurity.com/2013/05/exploiting-samsung-galaxy-s4-secure-boot.html

Mitigation:
• Do not use headers, pointers, addresses without/before checking authenticity
2. Key management

- Disclosing signing keys
- Also:
  - Signing development boot loaders with production keys
- Last year: we identified issue with a device in the field
- Vendor currently working on mitigation

**Mitigation:**

- Starting from the first key you create, implement proper key management: storage, access, lifetime, revocation
- Provide for test keys and test devices to limit exposure
1. Weak signing keys/methods

PS3 Epic Fail

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
```

Source: http://events.ccc.de/congress/2010
Console Hacking 2010
1. Weak signing keys/methods

- Know and understand the weaknesses of the algorithms and protocols used

Examples:
- RSA small exponent signature verification
- PS3 ECDSA signatures with same ‘random’
- [http://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html](http://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html)

Mitigation:
- Cryptographic review
Parting thoughts

• The purpose and function determine what is a sufficiently strong implementation

• High security applications need to consider many aspects including side channel and fault injection attacks

• But: proper design principles go a long way

• Learn your lessons from the past

• And pay attention to detail…

Source: http://www.jeffalytics.com
Riscure North America
71 Stevenson Street, Suite 400
San Francisco, CA 94105
USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
www.riscure.com

Contact: Job de Haas
dehaas@riscure.com
Principal Security Analyst
Riscure Security Lab

Riscure North America
71 Stevenson Street, Suite 400
San Francisco, CA 94105
USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
www.riscure.com