FATAL FURY ON ESP32: TIME TO RELEASE HW EXPLOITS

LimitedResults
ZeroNights 2019

12-13 November, St. Petersburg

$ whoami

- Limited
    - By Time, $$$, Skills too…
- Results
    - www.LimitedResults.com
- Offensive Side
    - Focus on HW, Low-Level Vulns…
- No Affiliation
- Time to play!

POWER ON

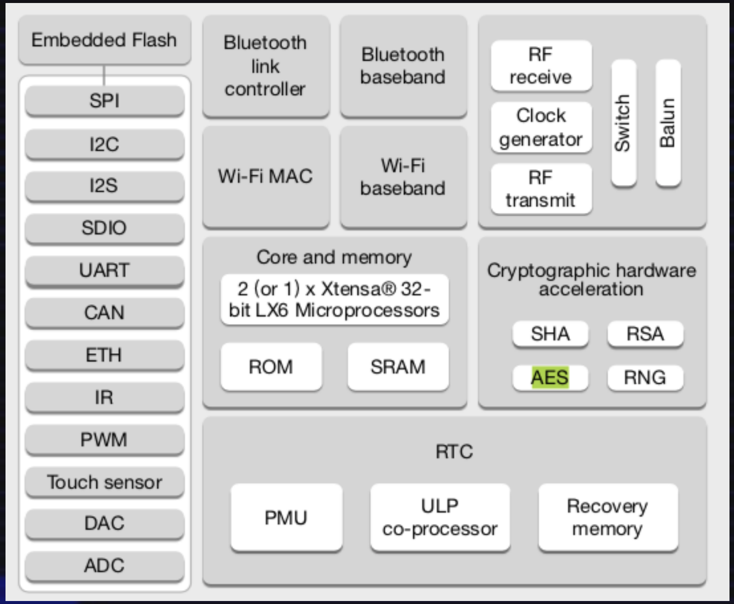
INTRODUCTION

The Entry Point

- Last April, I decide to ~~break~~ investigate into the ESP32
  - System-on-Chip (SoC) released in 2016 by Espressif
  - Widely-deployed (> 100M of devices) [1]
  - Wireless MCU/SoC Market leader
  - Claim to have ‘State-of-the-Art’ Security
  - 12 years-longevity commitment
- General Use
  - IoT
  - Wireless peripheral

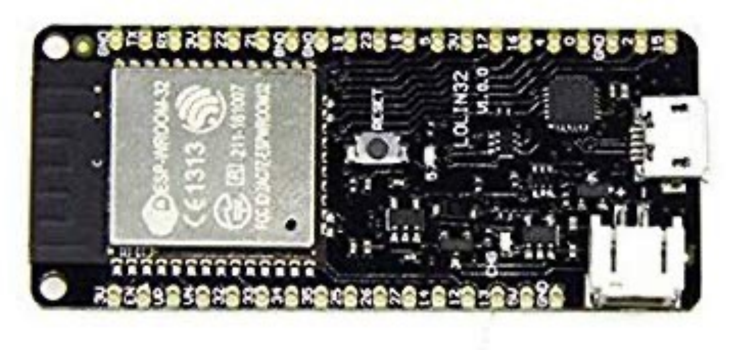
The target

- ESP32
  - Techno 40nm node
  - QFN 6*6, 48 pins
- Overview
  - Wi-FI (2.4GHz) & BT v4.2
  - Ultra Low-Power
  - Xtensa Dual-Core LX6
    - up to 240MHz
    - ROM, SRAM, no CPU caches
  - GPIOs, Touch sensor, ADC…
  - 4 SPI, 3 UART, Ethernet…
  - No USB



ESP32 Form Factor

- ESP32 SiP module (ESP32-WROOM-32)
  - Easy to integrate in any design
  - Flash storage 4MB
  - FCC certified
- ESP32 Dev-Kit (Lolin ESP32)
  - Micro-USB
    - Power
    - ttyUSB0 port
    - Pin headers
- Limited Cost
  - 15$



ESP32 Software

- Esp-idf Dev. Framework on Github
  - xtensa-esp32-elf toolchain
  - Set of Python Tools (esptool)
- Good Quality of Documentation
  - Datasheet and TRM available [2]
- Arduino core supported
  - I don’t like pre-compiled libraries, I don’t use it
- Official Amazon AWS IoT Platform
  - FreeRTOS, Mongoose OS…

Agenda Today

- Focus on Built-in Security
  - Just Grep the Datasheet
- Four Points
  - Crypto HW accelerator
  - Secure Boot
  - Flash Encryption
  - OTP
- Let’s start!

1.4.4 Security

- Secure boot
- Flash encryption
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration:

OPTIONS MENU

SETTINGS

The Limited Plan

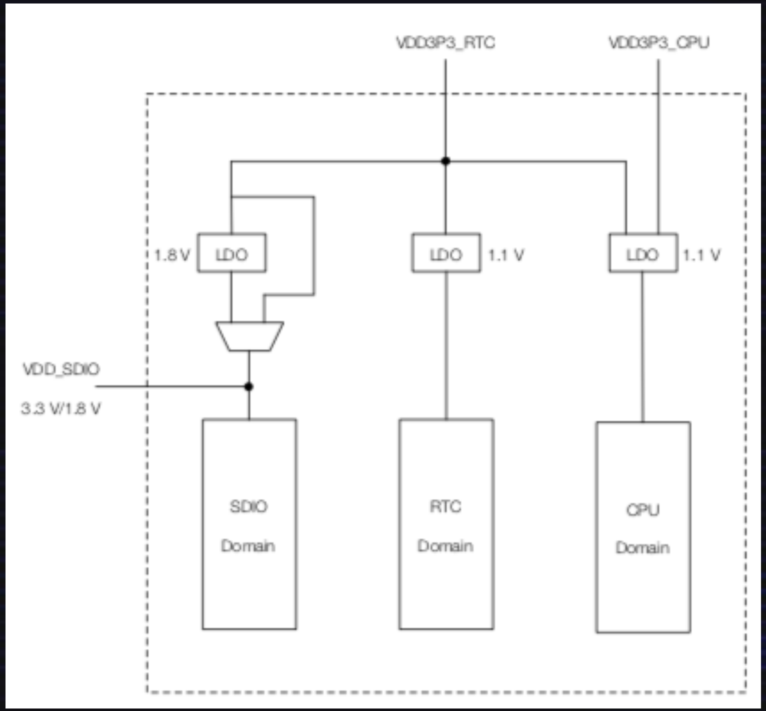
- The Context
  - 3 months to investigate (spare time)
- My Objective
  - Break one by one the Security Features
    - Physical Access Required (plausible attack scenario nowadays)
- So, I will probably use HW Techniques
  - Fault Injection, Side Channel maybe?
  - Micro-soldering, PCB modification
  - Reverse
  - And Code Review ☺

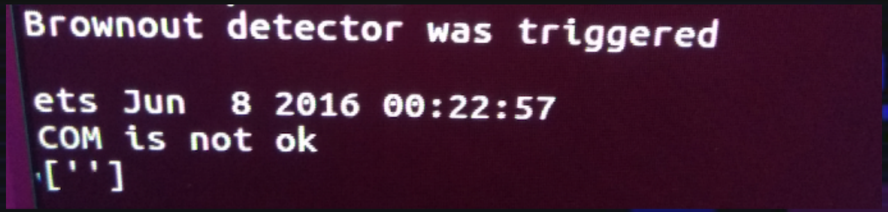
Fault Injection

- Voltage glitching
    - Well-known, still efficient and Low-cost FI technique nowadays
        - Public ressources about voltage glitching [3][4][…]
- Goal
    - Perturb the Power of the chip to induce a fault during critical SW/HW operations
- Expected effects
    - Skip instruction
        - Checks…
    - Data/Code modification
        - Branch conditions…
    - Sometimes difficult to predict/understand
        - especially with complex CPU architecture (cache effects?, pipeline?…)

Power domains inside ESP32

- 3 separate Power domains
- CPU domain shares two Power Signals
  - VDD3P3_CPU && VDD3P3_RTC (not common)
- Low Drop-out regulators (LDO)
  - Stabilize internal voltages
    - Filter effect against glitches?
- Brownout Detector (BOD)
  - « If the BOD detects a voltage drop, it will trigger a signal shutdown and even send a message on UART »
    - Able to detect glitches?
- BoD only effective on VDD_RTC
- So, I will Glitch on VDD3P3_CPU

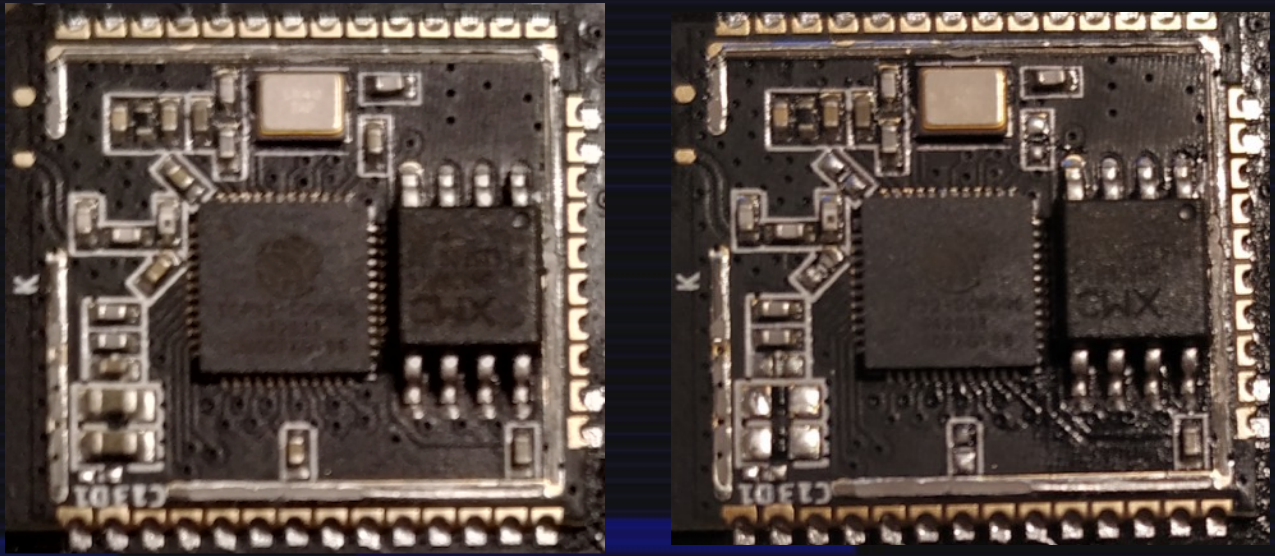




```
Brownout detector was triggered

ets Jun  8 2016 00:22:57
COM is not ok
['']
```

Target Preparation

- ESP-WROOM-32 Module
  - Shield is removed
- No silkscreen but Schematic available
- I remove Capacitors connected to VDD_CPU and VDD_RTC

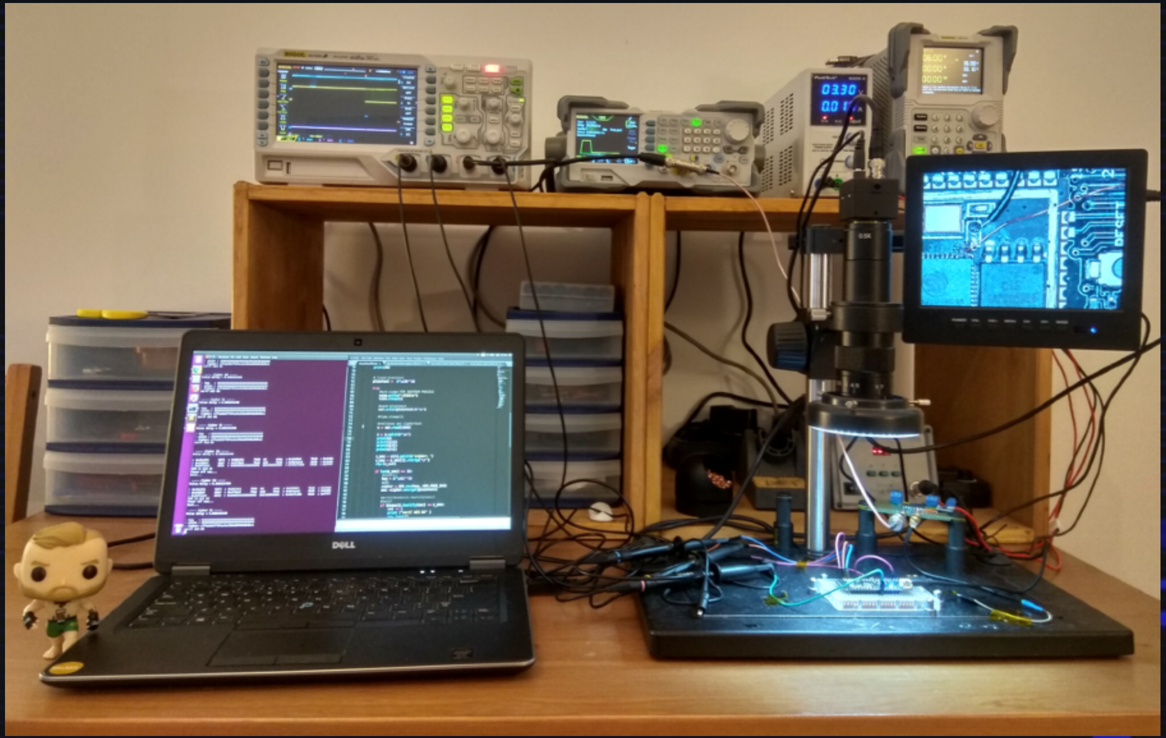


PCB Modification

- Three steps
  - Expose the VDD_CPU trace (Pin 37)
  - Cut the trace
  - Solder the glitch output to VDD_CPU pin and GND

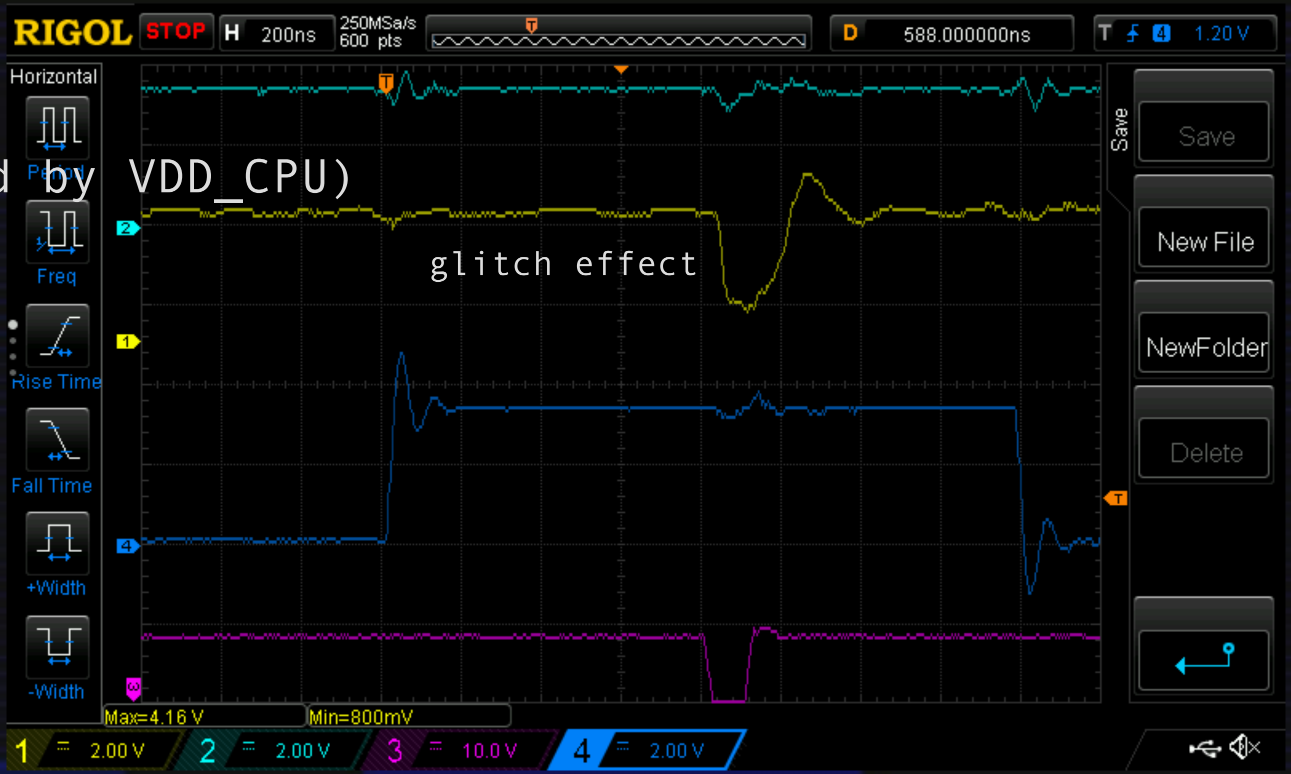
HW Setup

- Home-made Glitcher (10$)
  - Based on MAX4619
  - Add passive components
  - SMA connectors
- Synchronised by a Scope
- Triggered by Signal Generator
  - USB commands to set parameters
    - Delay
    - Width
    - Voltage
- Python scripts for full-control
  - Can run during days…



Voltage Glitching effect

- Effect Looks good



UART
(powered by VDD_CPU)

glitch effect

GPIO

cmd

LEVEL 1

THE CRYPTO-CORE

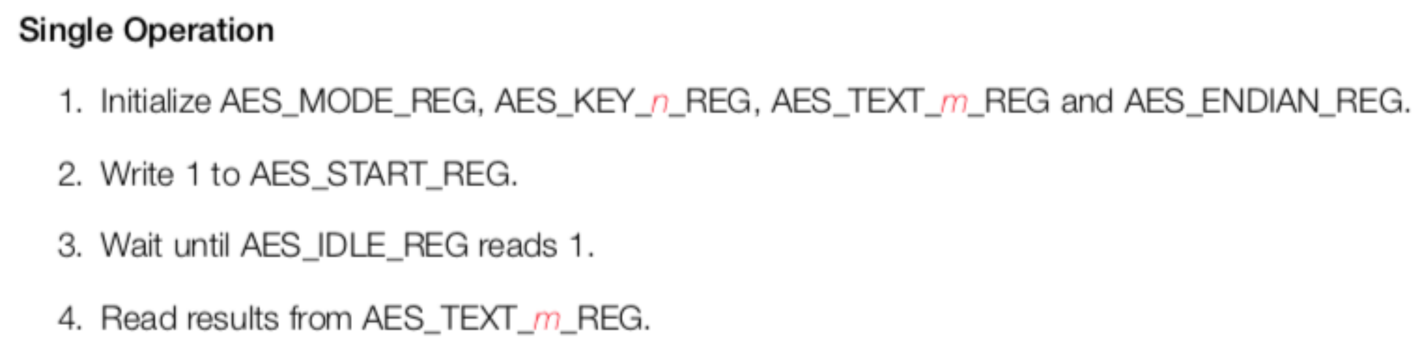
Crypto-Core/ Crypto-Accelerator

- Just a peripheral to speed-up the computation
  - AES, SHA, RSA…
- Why is it interesting to pwn?
  - Espressif Crypto-Lib
  - HW accel. used by default in MBedTLS
    - MBedTLS is the ARM crypto-library (all IoT are using it)
- My Goal
  - Focus on the CPU/Crypto interface (crypto-driver)
    - Do not expect to find ‘pure’ Software Vulns
  - Looking for vulns triggered by Fault Injection
- It is Time for Code Review

arm MBED

Design Weakness

- AES operation
  - Datasheet



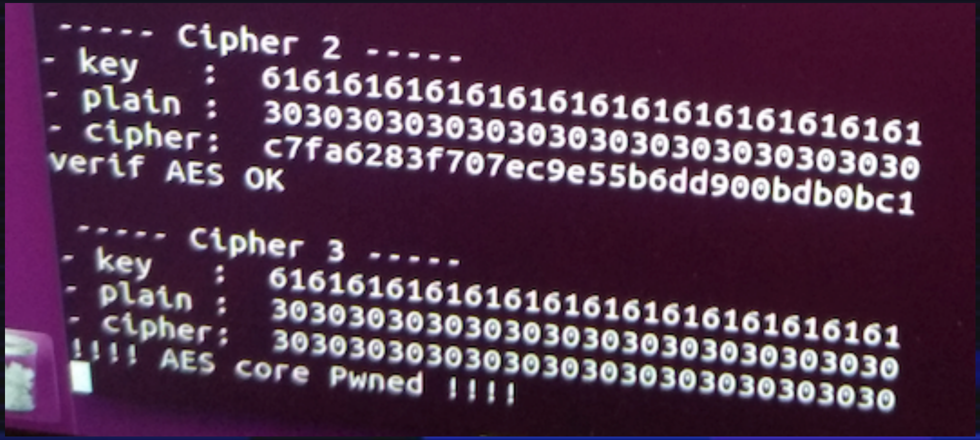
Single Operation

1. Initialize AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_*m*_REG and AES_ENDIAN_REG.
2. Write 1 to AES_START_REG.
3. Wait until AES_IDLE_REG reads 1.
4. Read results from AES_TEXT_*m*_REG.

- Design Weakness
  - AES_TEXT_m_REG registers used to store plaintext and also ciphertext
- Encrypt-In-Place can be risky
  - If something goes wrong during AES call, pretty sure I can retrieve the plaintext
    - Pretty cool & simple to exploit as first PoC

Vuln n*1 = AES Bypass

- Previous Weakness is confirmed
- Multiple spots to trigger
  - AES call
  - The while condition
  - The For loop
- PoC
  - Output = Input

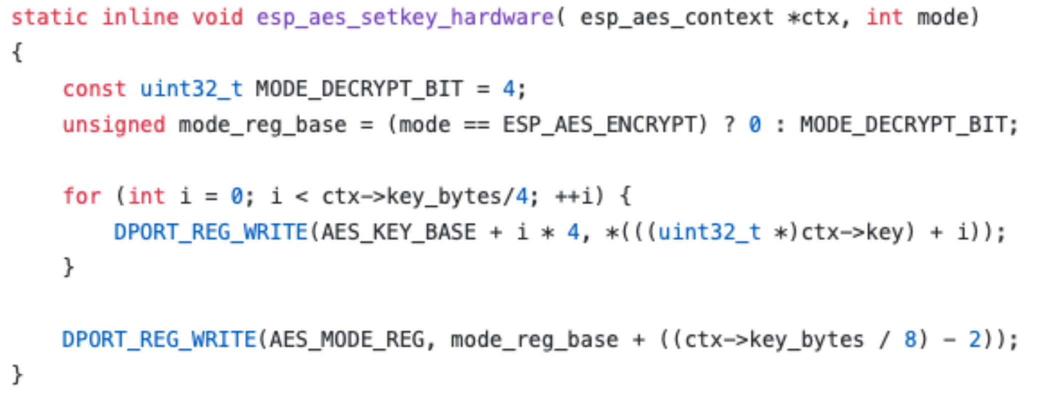


```
----- Cipher 2 -----
- key   : 61616161616161616161616161616161
- plain : 30303030303030303030303030303030
- cipher: c7fa6283f707ec9e55b6dd900bdb0bc1
verif AES OK

----- Cipher 3 -----
- key   : 61616161616161616161616161616161
- plain : 30303030303030303030303030303030
- cipher: 30303030303030303030303030303030
!!!! AES core Pwned !!!!
```

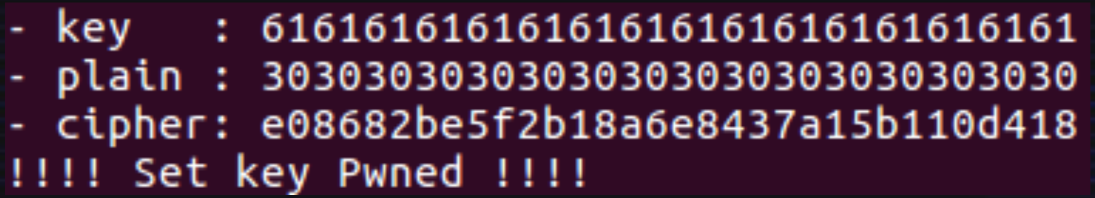
```
 * Call only while holding esp_aes_acquire_hardware().
 */
v4.0-dev-141-g106dc0590-dirty
static inline void esp_aes_block(const void *input, void *output)
{
    const uint32_t *input_words = (const uint32_t *)input;
    uint32_t *output_words = (uint32_t *)output;
    uint32_t *mem_block = (uint32_t *)AES_TEXT_BASE;

    for(int i = 0; i < 4; i++) {
        mem_block[i] = input_words[i];
    }

    DPORT_REG_WRITE(AES_START_REG, 1);

    DPORT_STALL_OTHER_CPU_START();
    {
        while (_DPORT_REG_READ(AES_IDLE_REG) != 1) { }
        for (int i = 0; i < 4; i++) {
            output_words[i] = mem_block[i];
        }
    }
    DPORT_STALL_OTHER_CPU_END();
}
```

Vuln n*2 = AES SetKey

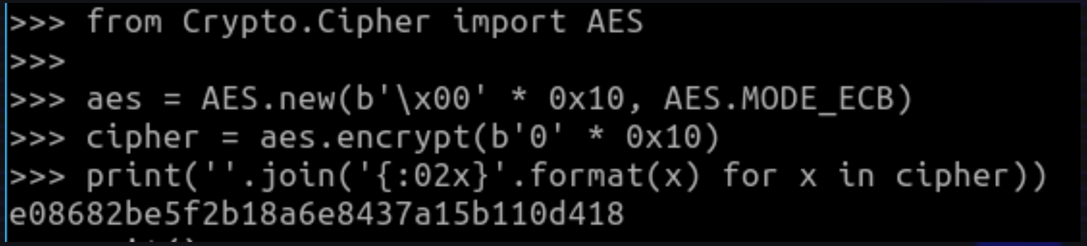
- Vuln to trigger
  - Unprotected loop for to load the key into the crypto-core
- PoC
  - Key ZEROized
  - Persistent key value until the next setkey()
  - Nice for attacking AES Cipher Block Chaining Mode



```c
static inline void esp_aes_setkey_hardware( esp_aes_context *ctx, int mode)
{
    const uint32_t MODE_DECRYPT_BIT = 4;
    unsigned mode_reg_base = (mode == ESP_AES_ENCRYPT) ? 0 : MODE_DECRYPT_BIT;

    for (int i = 0; i < ctx->key_bytes/4; ++i) {
        DPORT_REG_WRITE(AES_KEY_BASE + i * 4, *(((uint32_t *)ctx->key) + i));
    }

    DPORT_REG_WRITE(AES_MODE_REG, mode_reg_base + ((ctx->key_bytes / 8) - 2));
}
```



```
- key    : 61616161616161616161616161616161
- plain  : 30303030303030303030303030303030
- cipher: e08682be5f2b18a6e8437a15b110d418
!!!! Set key Pwned !!!!
```



```
>>> from Crypto.Cipher import AES
>>>
>>> aes = AES.new(b'\x00' * 0x10, AES.MODE_ECB)
>>> cipher = aes.encrypt(b'0' * 0x10)
>>> print(''.join('{:02x}'.format(x) for x in cipher))
e08682be5f2b18a6e8437a15b110d418
```

Crypto-Core Conclusion

- Crypto-core does not improve security
- Six Vulns with PoCs in AES and SHA
  - Espressif HwCrypto in esp-idf 4.0
  - ARM MbedTLS v2.13.1
- Resp. disclosure
  - No answer from Espressif & ARM during 1 month ☹
  - BugBounty Program from ARM MBedTLS is Fake ☹
  - Silent Patch attempt ☹
- I am (a little bit) in a FURY now…
- …and I am going to pwn HARDer

LEVEL 2

SECURE BOOT

Role of Secure Boot

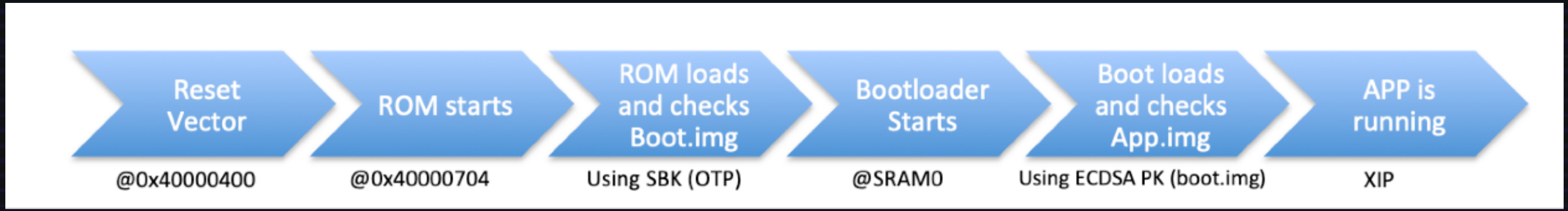
- Protector of FW Authenticity
- Avoid FW modification
  - Easy to flash new Firmware in SPI Flash
  - CRC? Not sufficient sorry…
- It will Create a Chain of Trust
  - BootROM to Bootloader until the App
- It Guarantees the code running on the device is Genuine
  - Will not boot if images are not properly signed

ESP32 SecBoot during Production

- Secure Boot Key (SBK)
  - SBK burned into E-Fuses BLK2
  - This SBK cannot be readout or modified (R/W protected)
  - Used by bootROM to perform AES-256 ECB
- ECDSA key pair
  - Created by the App developer
  - Priv. Key used to sign the App
  - Public Key integrated to bootloader.img
- The Bootloader Signature
  - 192 bytes header = 128 bytes of random + 64 bytes digest
    - Digest = SHA-512(AES-256((bootloader.bin + ECDSA PK), SBK))
  - Random at 0x0 in Flash Memory layout, digest at 0x80

Sec. Boot on the Field

- Boot process



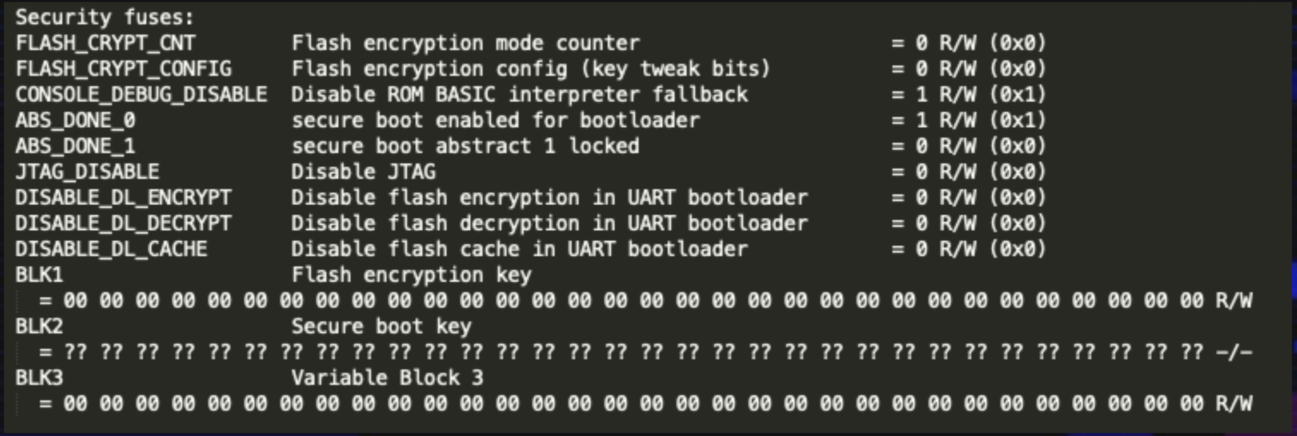
Reset Vector → ROM starts → ROM loads and checks Boot.img → Bootloader Starts → Boot loads and checks App.img → APP is running

@0x40000400 | @0x40000704 | Using SBK (OTP) | @SRAM0 | Using ECDSA PK (boot.img) | XIP

- Verification Mechanisms
  - BootROM (Stage 0)
    - Compute Digest with SBK and compare with 64-bytes Digest at 0x80
  - ECDSA verification by the Bootloader (Stage 1)
    - Micro-ECC is used
- I will Focus on Stage 0
  - Signature based on Symmetric Crypto
  - SBK = AES-Key used to sign the bootloader (CRITICAL ASSET)
    - Stored in E-Fuses, R/W protected

Set the Secure Boot

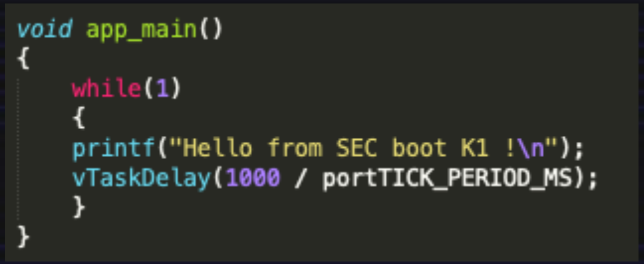
- Can be done automatically by ESP-IDF Framework…
- But I prefer to do it manually
  - Burn the Secure Boot Key into BLK2
    - $ espefuse.py burn_key secure_boot ./secure-bootloader-key-256.bin
  - Burn the ABS_DONE fuse to activate the sec boot
    - $ espefuse.py burn_efuse ABS_DONE_0
- E-Fuses Map
  - espefuse.py summary
- Look JTAG fuse ☺



```
Security fuses:
FLASH_CRYPT_CNT        Flash encryption mode counter                      = 0 R/W (0x0)
FLASH_CRYPT_CONFIG     Flash encryption config (key tweak bits)           = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE  Disable ROM BASIC interpreter fallback             = 1 R/W (0x1)
ABS_DONE_0             secure boot enabled for bootloader                 = 1 R/W (0x1)
ABS_DONE_1             secure boot abstract 1 locked                      = 0 R/W (0x0)
JTAG_DISABLE           Disable JTAG                                       = 0 R/W (0x0)
DISABLE_DL_ENCRYPT     Disable flash encryption in UART bootloader        = 0 R/W (0x0)
DISABLE_DL_DECRYPT     Disable flash decryption in UART bootloader        = 0 R/W (0x0)
DISABLE_DL_CACHE       Disable flash cache in UART bootloader             = 0 R/W (0x0)
BLK1                   Flash encryption key
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
BLK2                   Secure boot key
  = ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? -/-
BLK3                   Variable Block 3
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
```

Secure boot in Action

- Signed App (using SBK)

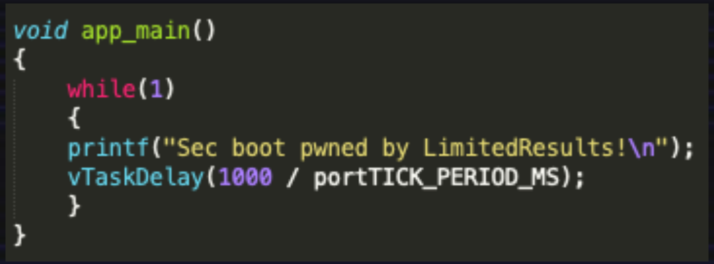


```
void app_main()
{
    while(1)
    {
    printf("Hello from SEC boot K1 !\n");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}
```

- make flash, then it runs

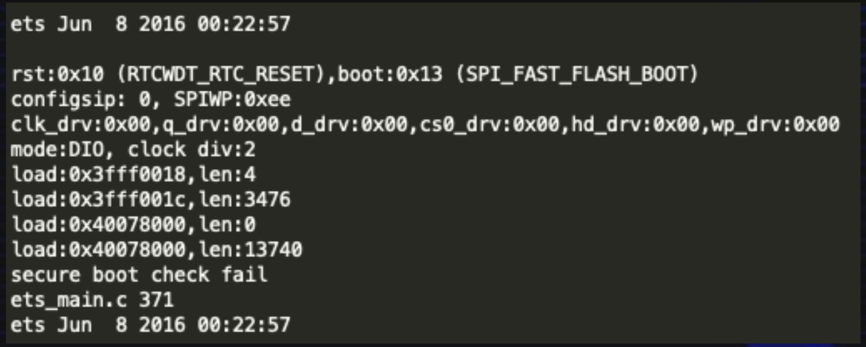
```
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8556
load:0x40078000,len:12064
load:0x40080400,len:7088
entry 0x400807a0
D (88) bootloader_flash: mmu set block paddr=0x00000000 (was 0xffffffff)
I (38) boot: ESP-IDF v4.0-dev-667-gda13efc-dirty 2nd stage bootloader
...
I (487) cpu_start: Pro cpu start user code
I (169) cpu_start: Starting scheduler on PRO CPU.
Hello from Sec boot K1 !
Hello from Sec boot K1 !
```

- Unsigned App (no Key)



```
void app_main()
{
    while(1)
    {
    printf("Sec boot pwned by LimitedResults!\n");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}
```

- Flash it then Fail



```
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:3476
load:0x40078000,len:0
load:0x40078000,len:13740
secure boot check fail
ets_main.c 371
ets Jun  8 2016 00:22:57
```

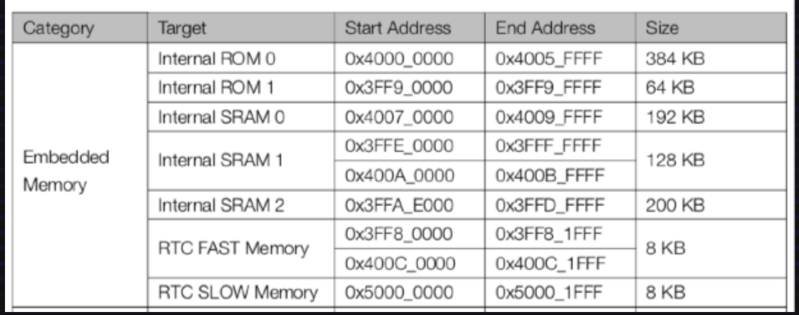
- Stuck in stage0 (perfect)

Bypass the Secure Boot

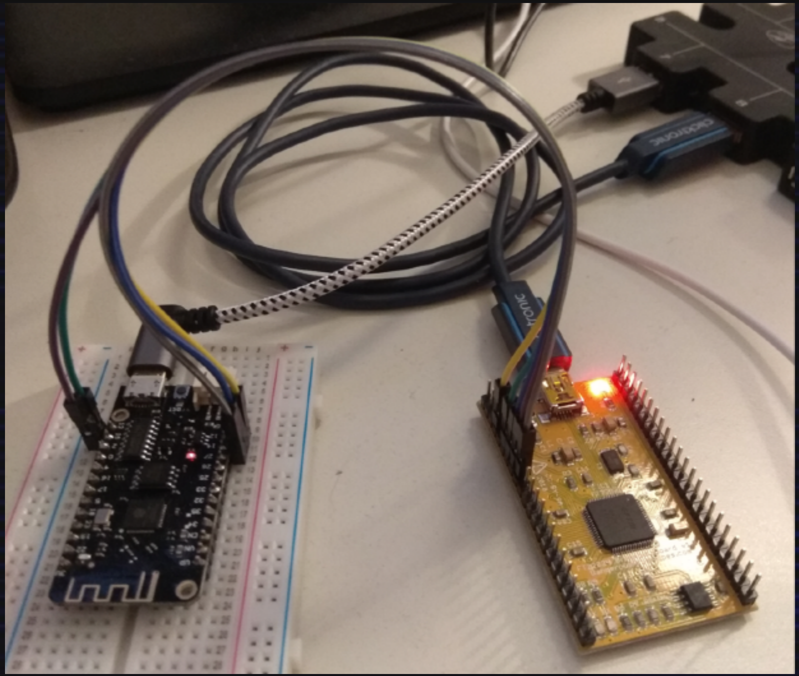
- Why?
  - To have code exec
- How?
  - Force ESP32 to execute my unsigned bootloader to load my unsigned app
- Focus on BootROM
  - Always Nice to exploit BootROM vulns
  - Always Difficult to Fix BootROM vulns
- So, I need to reverse the BootROM image
- But first, I need to dump it…

Dump the BootROM

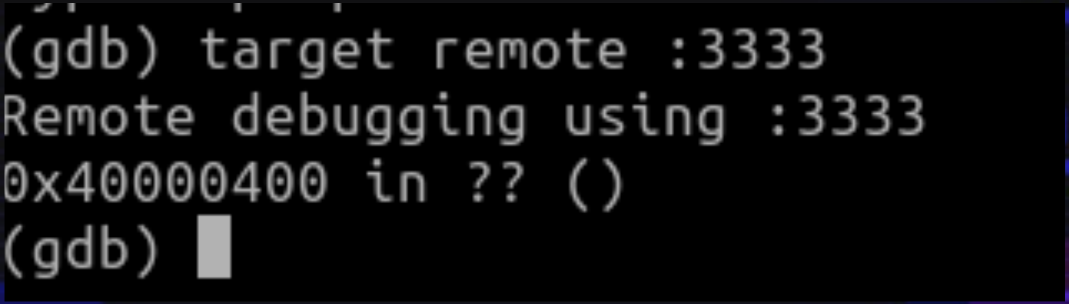
- Memory map



Category	Target	Start Address	End Address	Size
Embedded Memory	Internal ROM 0	0x4000_0000	0x4005_FFFF	384 KB
	Internal ROM 1	0x3FF9_0000	0x3FF9_FFFF	64 KB
	Internal SRAM 0	0x4007_0000	0x4009_FFFF	192 KB
	Internal SRAM 1	0x3FFE_0000	0x3FFF_FFFF	128 KB
		0x400A_0000	0x400B_FFFF	
	Internal SRAM 2	0x3FFA_E000	0x3FFD_FFFF	200 KB
	RTC FAST Memory	0x3FF8_0000	0x3FF8_1FFF	8 KB
		0x400C_0000	0x400C_1FFF	
	RTC SLOW Memory	0x5000_0000	0x5000_1FFF	8 KB



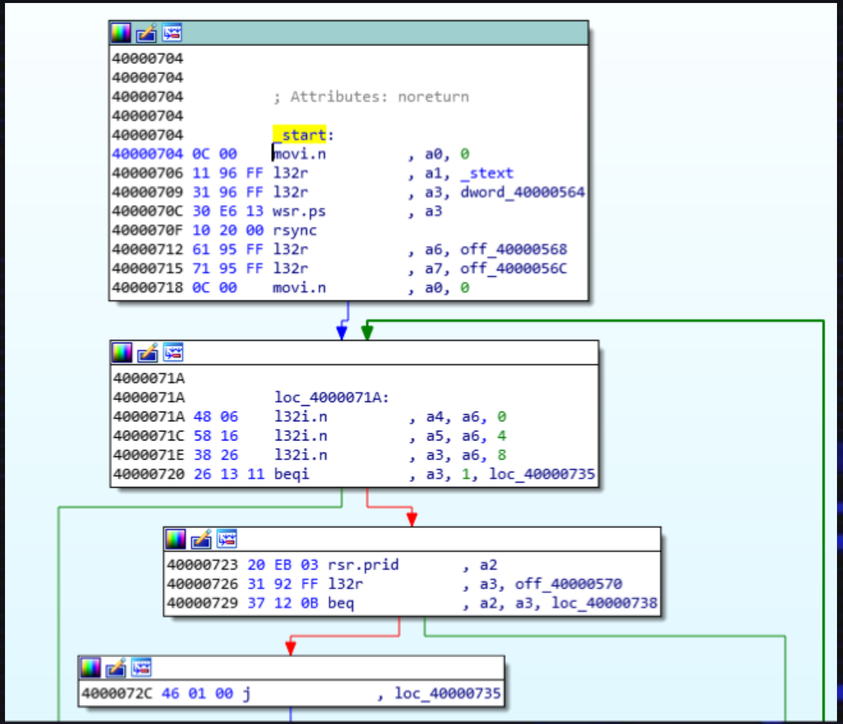
- Remember I didn't burn JTAG DISABLE E-Fuse?
  - FT2232H board (20$)
  - OpenOCD + xtensa-esp32-gdb
- Full Debug Access
  - Reset Vector 0x40000400
- BootROM dumped



```
(gdb) target remote :3333
Remote debugging using :3333
0x40000400 in ?? ()
(gdb) █
```

BootROM Reverse

- Xtensa is ‘exotic’ arch
  - registers windowing, lengths of instr…
  - ISA [5]
- IDA
  - ida-xtensa plugin from @themadinventor
- Secure_boot.h
  - List all the ROM functions
    - They deprecated since…
- Call my ‘little bro’ to check my mess
  - @wiskitki
- At the end, not perfect but doable
  - _start at 0x40000704 (as expected)



```
40000704
40000704
40000704          ; Attributes: noreturn
40000704
40000704          _start:
40000704 0C 00    movi.n          , a0, 0
40000706 11 96 FF l32r            , a1, _stext
40000709 31 96 FF l32r            , a3, dword_40000564
4000070C 30 E6 13 wsr.ps          , a3
4000070F 10 20 00 rsync
40000712 61 95 FF l32r            , a6, off_40000568
40000715 71 95 FF l32r            , a7, off_4000056C
40000718 0C 00    movi.n          , a0, 0
```

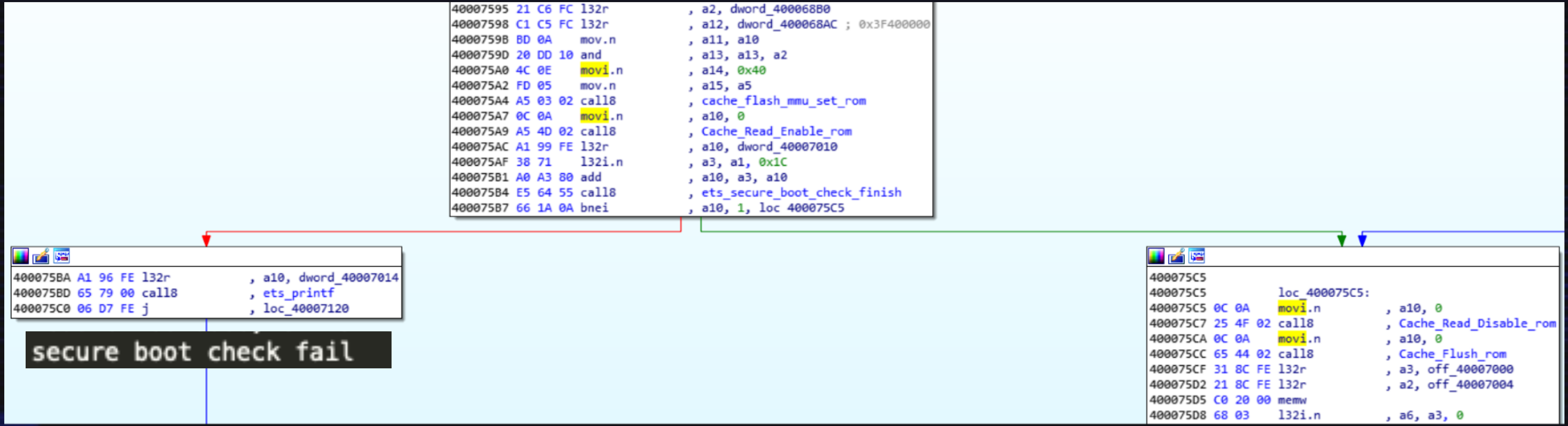
```
4000071A
4000071A          loc_4000071A:
4000071A 48 06    l32i.n          , a4, a6, 0
4000071C 58 16    l32i.n          , a5, a6, 4
4000071E 38 26    l32i.n          , a3, a6, 8
40000720 26 13 11 beqi            , a3, 1, loc_40000735
```

```
40000723 20 EB 03 rsr.prid        , a2
40000726 31 92 FF l32r            , a3, off_40000570
40000729 37 12 0B beq             , a2, a3, loc_40000738
```

```
4000072C 46 01 00 j               , loc_40000735
```

The BootROM Vuln

- After ets_secure_boot_check_finish()



```
40007595 21 C6 FC l32r           , a2, dword_400068B0
40007598 C1 C5 FC l32r           , a12, dword_400068AC ; 0x3F400000
4000759B BD 0A    mov.n          , a11, a10
4000759D 20 DD 10 and            , a13, a13, a2
400075A0 4C 0E    movi.n         , a14, 0x40
400075A2 FD 05    mov.n          , a15, a5
400075A4 A5 03 02 call8          , cache_flash_mmu_set_rom
400075A7 0C 0A    movi.n         , a10, 0
400075A9 A5 4D 02 call8          , Cache_Read_Enable_rom
400075AC A1 99 FE l32r           , a10, dword_40007010
400075AF 38 71    l32i.n         , a3, a1, 0x1C
400075B1 A0 A3 80 add            , a10, a3, a10
400075B4 E5 64 55 call8          , ets_secure_boot_check_finish
400075B7 66 1A 0A bnei           , a10, 1, loc 400075C5
```

```
400075BA A1 96 FE l32r           , a10, dword_40007014
400075BD 65 79 00 call8          , ets_printf
400075C0 06 D7 FE j              , loc_40007120
```

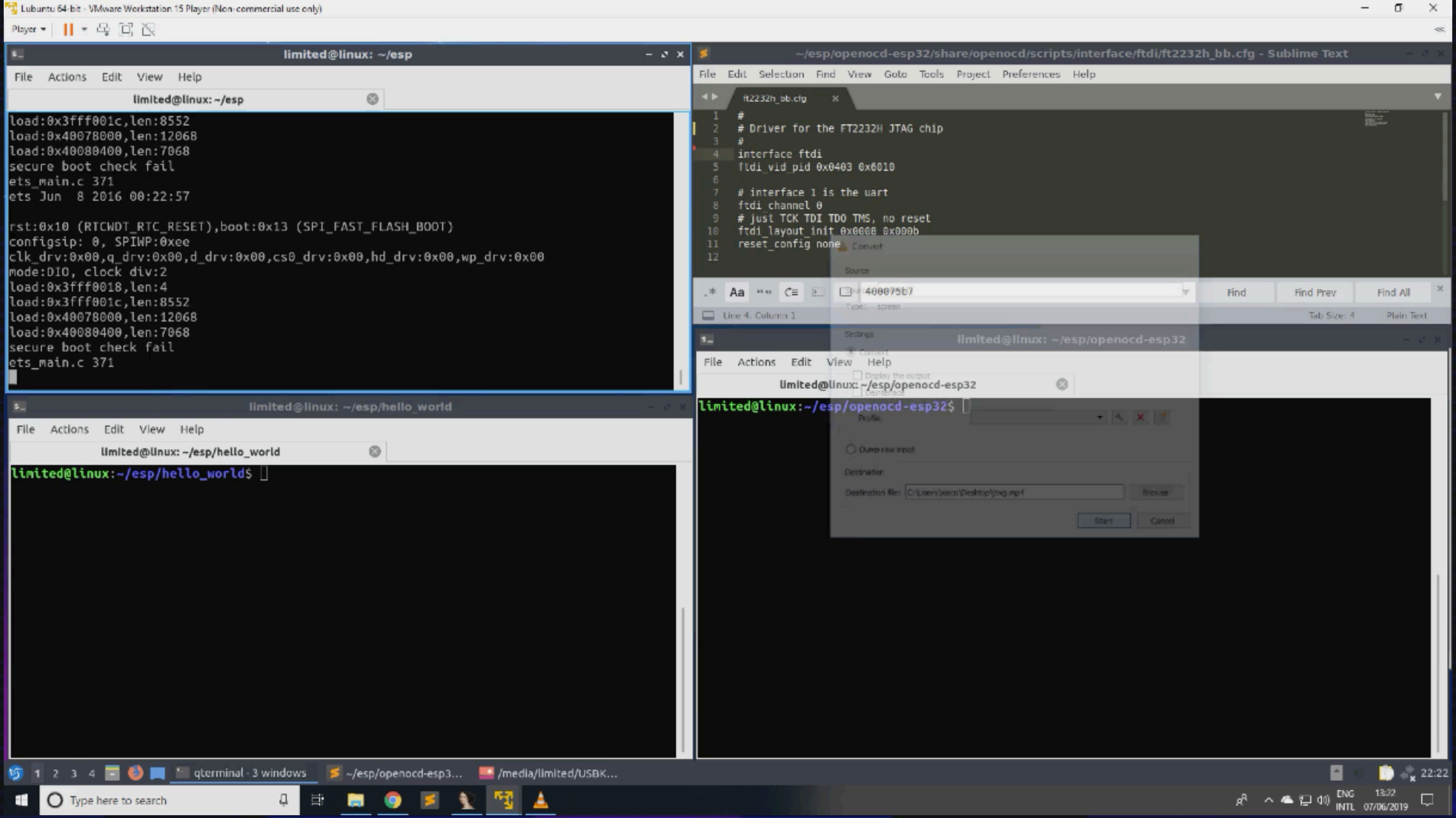
secure boot check fail

```
400075C5
400075C5          loc_400075C5:
400075C5 0C 0A    movi.n         , a10, 0
400075C7 25 4F 02 call8          , Cache_Read_Disable_rom
400075CA 0C 0A    movi.n         , a10, 0
400075CC 65 44 02 call8          , Cache_Flush_rom
400075CF 31 8C FE l32r           , a3, off_40007000
400075D2 21 8C FE l32r           , a2, off_40007004
400075D5 C0 20 00 memw
400075D8 68 03    l32i.n         , a6, a3, 0
```

- Bnei (Branch if not equal immediate)
  - Depends on a10 Register value (storing sec_boot_check() retvalue)
- I want PC jump to 0x400075C5 to execute the bootloader

Jtag Exploit Validation

- Set a10 register = 0 via JTAG to bypass secboot



```
load:0x3fff001c,len:8552
load:0x40078000,len:12068
load:0x40080400,len:7068
secure boot check fail
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8552
load:0x40078000,len:12068
load:0x40080400,len:7068
secure boot check fail
ets_main.c 371
```

```
#
# Driver for the FT2232H JTAG chip
#
interface ftdi
ftdi_vid_pid 0x0403 0x6010

# interface 1 is the uart
ftdi_channel 0
# just TCK TDI TDO TMS, no reset
ftdi_layout_init 0x0008 0x000b
reset_config none
```

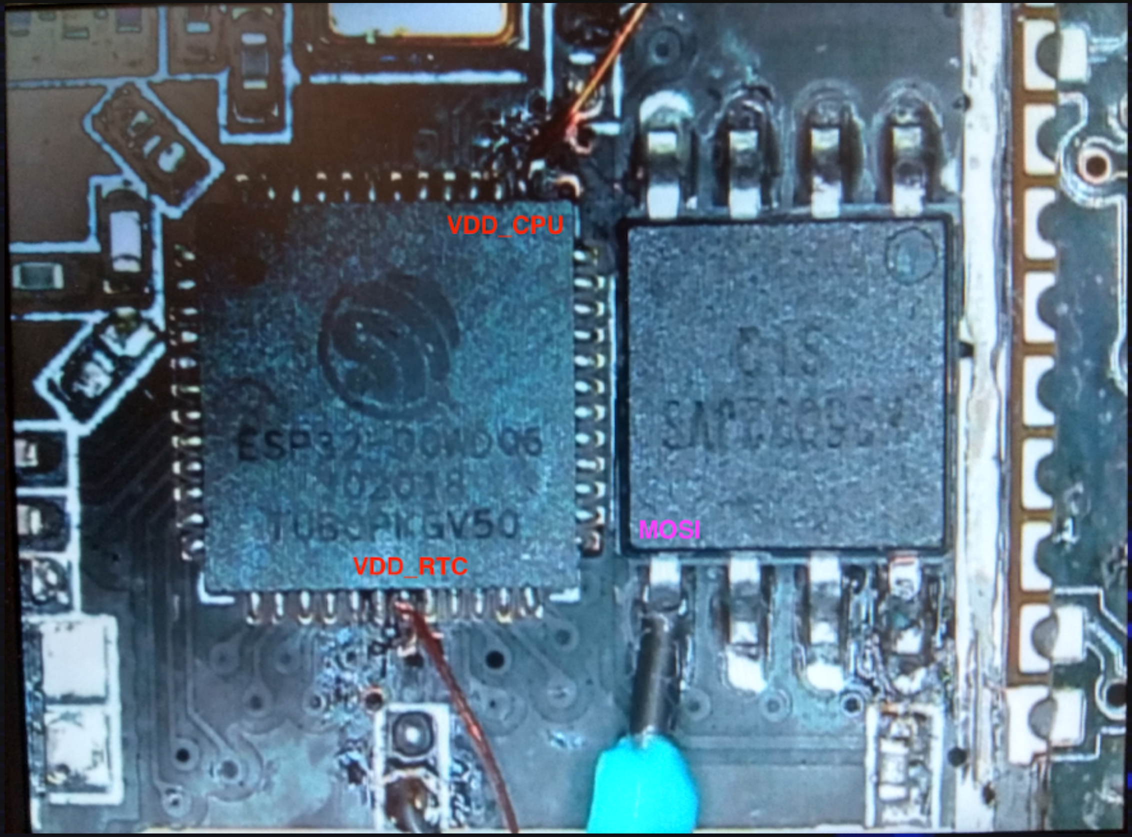
Find: 400075b7

```
limited@linux:~/esp/hello_world$
```

```
limited@linux:~/esp/openocd-esp32$
```

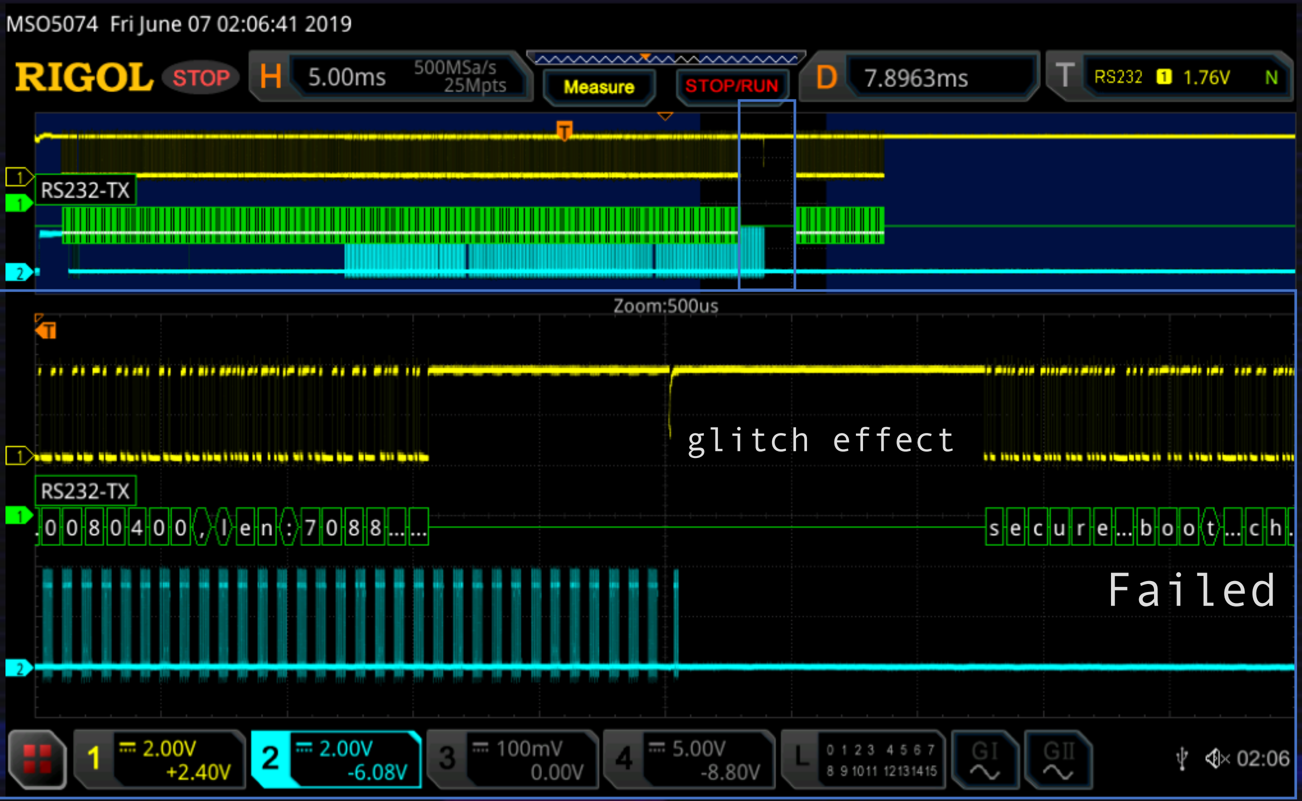
Time to Pwn (for Real)

- Real Life
  - JTAG is disabled
  - I could not find a way to exploit this Vuln by SW
- So, Fault Injection is my only way here
  - Simultaneous glitch on VDD_CPU && VDD_RTC
  - SPI MOSI is probed to have a timing information



First attempts during BootROM

- Previous BootROM Reverse is helpful
  - to determine Fault injection Timing



ZOOM

Serial

glitch effect

SPI

Failed

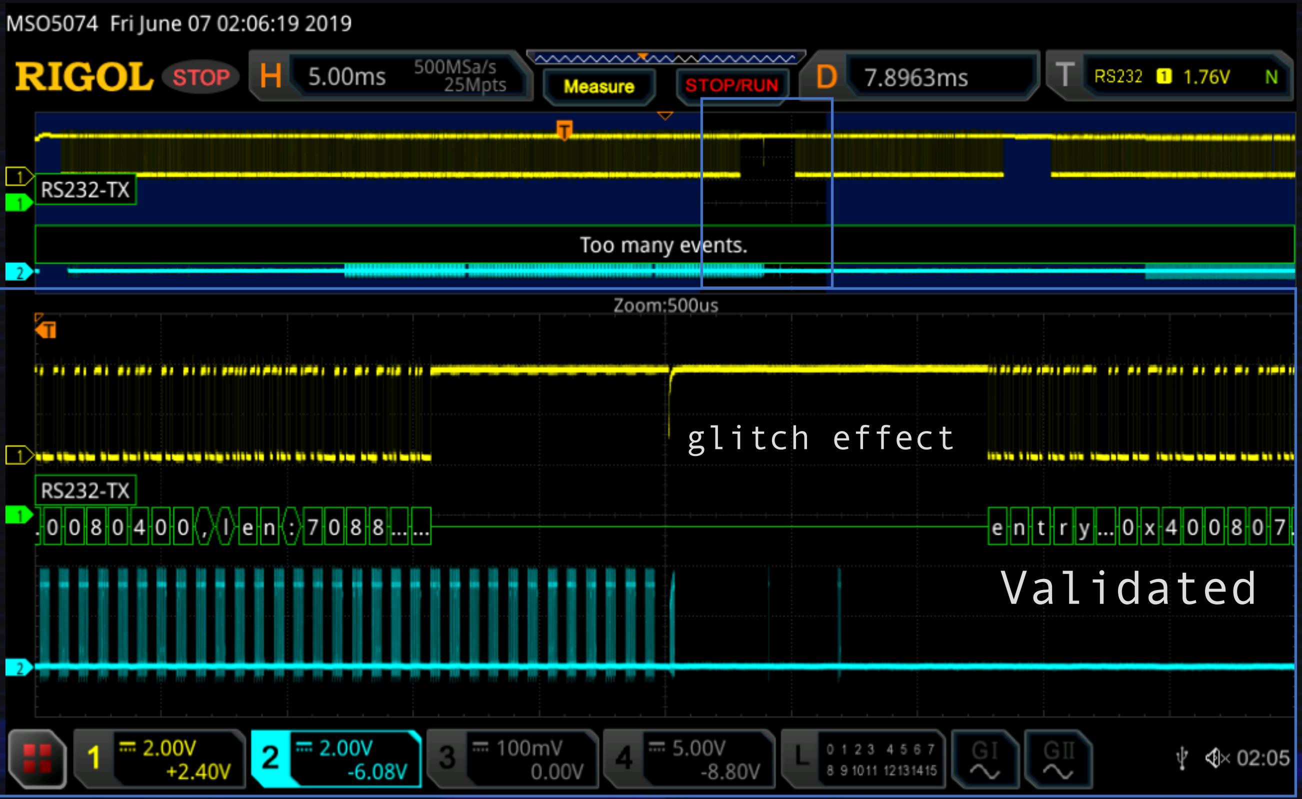
Successful Sec.Boot Bypass

- CPU is jumping to the entry point, Bootloader is executed. Done

ZOOM

Serial

SPI



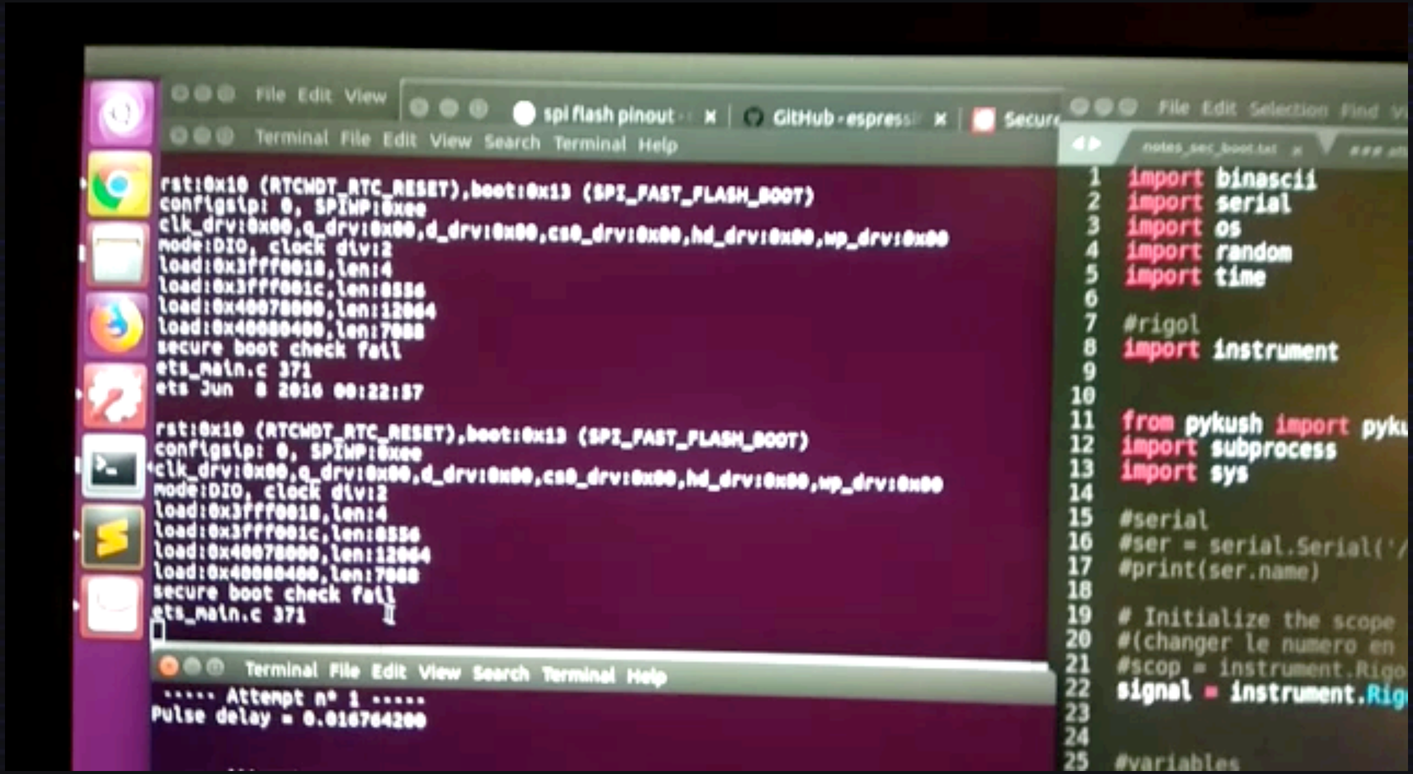
```
.0080400,\len:7088......
entry...0x400807
```

glitch effect

Validated

PoC Secure Boot

- Sorry for the tilt



```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8556
load:0x40078000,len:12064
load:0x40080400,len:7088
secure boot check fail
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:8556
load:0x40078000,len:12064
load:0x40080400,len:7088
secure boot check fail
ets_main.c 371
```

```
----- Attempt n° 1 -----
Pulse delay = 0.016764200
```

```python
import binascii
import serial
import os
import random
import time

#rigol
import instrument


from pykush import pyku
import subprocess
import sys

#serial
#ser = serial.Serial('/
#print(ser.name)

# Initialize the scope
#(changer le numero en
#scop = instrument.Rigo
signal = instrument.Rig


#variables
```

Secure Boot Conclusion

- Secure Boot Bypass exploit
  - Stage 0 (bootROM Vuln)
  - Triggered by Fault Injection
  - Not persistent if Reset occurs
  - No way to Fix this without ROM revision
- Resp. disclosure
  - PoC sent on June 4
  - Security Advisory on Sept. 2
    - CVE-2019-15894 (requested by Vendor)
    - Patched by Flash Encryption always enabled
    - A security lab, called Riscure, found the same vuln
- No silent patch attempt this time…

LEVEL 3

FLASH ENCRYPTION

Role of Flash Encryption

- Protector of FW Confidentiality
  - Protect against Binary extraction and Reverse
- Without FE, it is easy to extract sensitive data
  - Ex: LIFX Wi-Fi lightbulbs [6]
- Firmware Encryption more and more present Today
  - Security by obscurity…
- Espressif recommends Secure Boot + Flash Encryption for maximum Security

DON'T TOSS THAT BULB, IT KNOWS YOUR PASSWORD

by: Tom Nardi

68 Comments

January 29, 2019

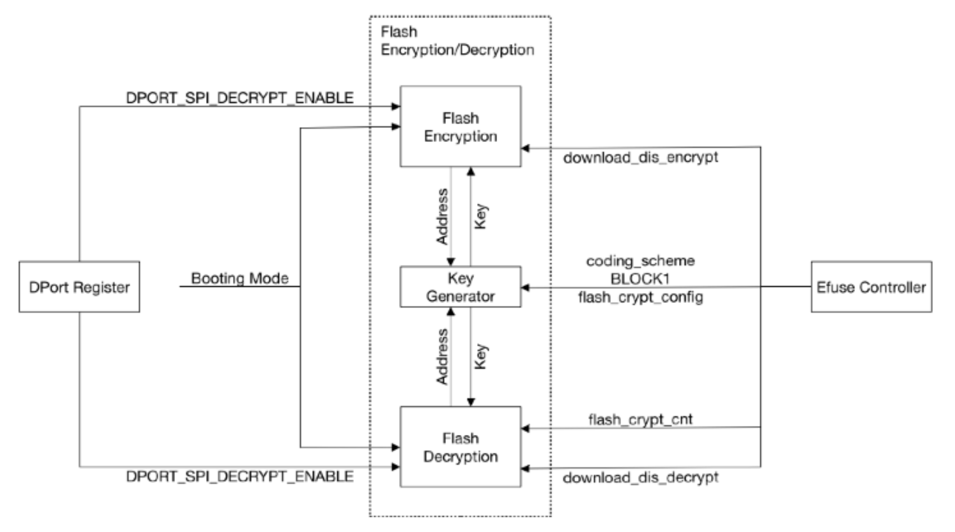
By Daniel Oberhaus | Jan 31 2019, 6:27pm

This Hacker Showed How a Smart Lightbulb Could Leak Your Wi-Fi Password

The "moderate to severe" vulnerabilities discovered by the hacker LimitedResults have since been fixed, according to the smart bulb company LIFX.

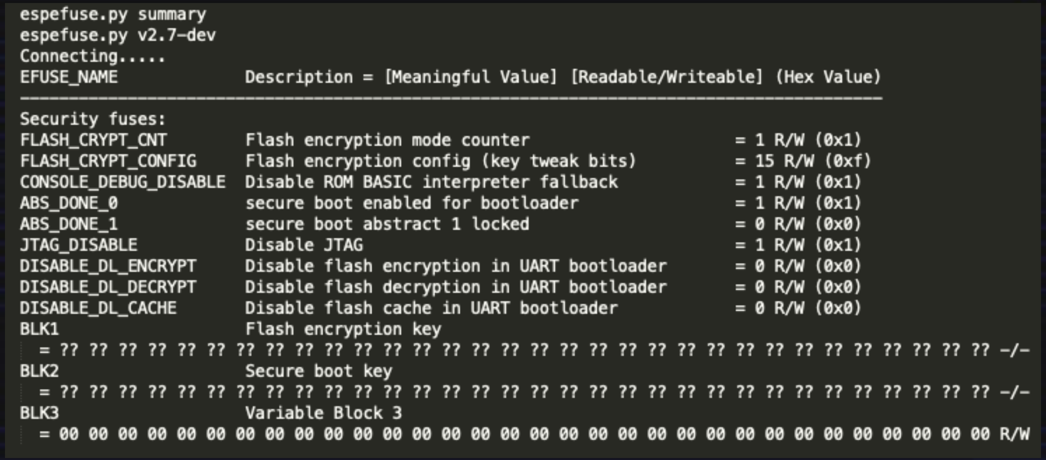
Flash Encryption Review

- HW AES Enc./Dec. Block in Flash Memory Controller
  - Fetch Key from E-Fuses and other parameters
  - Decrypt/Encrypt I/D into a Cache
  - SW cannot access
- Flash Encryption Key (FEK)
  - AES-Key used to decrypt the FW
    - Stored in E-Fuses BLK1 (R/W protected)
  - CRITICAL ASSET (of course)

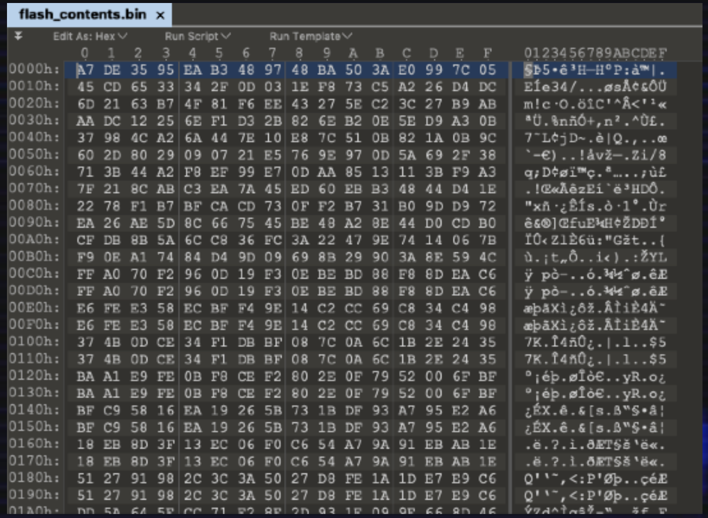


Set the Flash Encryption

- Burn the FEK into BLK1
  - $ espefuse.py --port /dev/ttyUSB0 burn_key flash_encryption my_flash_encryption_key.bin
- Activate the Flash Encryption
  - $ $ espefuse.py burn_efuse FLASH_CRYPT_CONFIG 0xf
  - $ espefuse.py burn_efuse FLASH_CRYPT_CNT
- Flash encrypted FW into ESP32
- Verify E-Fuses Map
- Verify encrypted FW



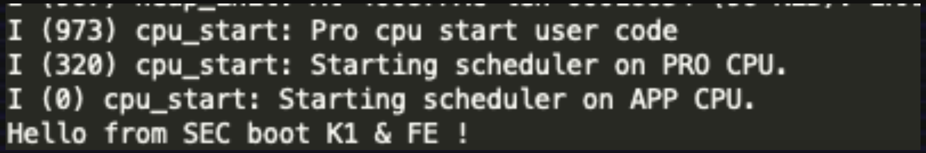
```
espefuse.py summary
espefuse.py v2.7-dev
Connecting.....
EFUSE_NAME           Description = [Meaningful Value] [Readable/Writeable] (Hex Value)
------------------------------------------------------------------------------------------
Security fuses:
FLASH_CRYPT_CNT      Flash encryption mode counter                    = 1 R/W (0x1)
FLASH_CRYPT_CONFIG   Flash encryption config (key tweak bits)         = 15 R/W (0xf)
CONSOLE_DEBUG_DISABLE Disable ROM BASIC interpreter fallback           = 1 R/W (0x1)
ABS_DONE_0           secure boot enabled for bootloader               = 1 R/W (0x1)
ABS_DONE_1           secure boot abstract 1 locked                    = 0 R/W (0x0)
JTAG_DISABLE         Disable JTAG                                     = 1 R/W (0x1)
DISABLE_DL_ENCRYPT   Disable flash encryption in UART bootloader      = 0 R/W (0x0)
DISABLE_DL_DECRYPT   Disable flash decryption in UART bootloader      = 0 R/W (0x0)
DISABLE_DL_CACHE     Disable flash cache in UART bootloader           = 0 R/W (0x0)
BLK1                 Flash encryption key
  = ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? -/-
BLK2                 Secure boot key
  = ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? -/-
BLK3                 Variable Block 3
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 R/W
```



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: A7 DE 35 95 EA B3 48 97 48 BA 50 3A E0 99 7C 05
0010h: 45 CD 65 33 34 2F 0D 03 1E F8 73 C5 A2 26 D4 DC
0020h: 6D 21 63 B7 4F 81 F6 EE 43 27 5E C2 3C 27 B9 AB
0030h: AA DC 12 25 6E F1 D3 2B 82 6E B2 0E 5E D9 A3 0B
0040h: 37 98 4C A2 6A 44 7E 10 E8 7C 51 0B 82 1A 0B 9C
0050h: 60 2D 80 29 09 07 21 E5 76 9E 97 0D 5A 69 2F 38
0060h: 71 3B 44 A2 F8 EF 99 E7 0D AA 85 13 11 3B F9 A3
0070h: 7F 21 8C AB C3 EA 7A 45 ED 60 EB B3 48 44 D4 1E
0080h: 22 78 F1 B7 BF CA CD 73 0F F2 B7 31 B0 9D D9 72
0090h: EA 26 AE 5D 8C 66 75 45 BE 48 A2 8E 44 D0 CD B0
00A0h: CF DB 8B 5A 6C C8 36 FC 3A 22 47 9E 74 14 06 7B
00B0h: F9 0E A1 74 84 D4 9D 09 69 8B 29 90 3A 8E 59 4C
00C0h: FF A0 70 F2 96 0D 19 F3 0E BE BD 88 F8 8D EA C6
00D0h: FF A0 70 F2 96 0D 19 F3 0E BE BD 88 F8 8D EA C6
00E0h: E6 FE E3 58 EC BF F4 9E 14 C2 CC 69 C8 34 C4 98
00F0h: E6 FE E3 58 EC BF F4 9E 14 C2 CC 69 C8 34 C4 98
0100h: 37 4B 0D CE 34 F1 DB BF 08 7C 0A 6C 1B 2E 24 35
0110h: 37 4B 0D CE 34 F1 DB BF 08 7C 0A 6C 1B 2E 24 35
0120h: BA A1 E9 FE 0B F8 CE F2 80 2E 0F 79 52 00 6F BF
0130h: BA A1 E9 FE 0B F8 CE F2 80 2E 0F 79 52 00 6F BF
0140h: BF C9 58 16 EA 19 26 5B 73 1B DF 93 A7 95 E2 A6
0150h: BF C9 58 16 EA 19 26 5B 73 1B DF 93 A7 95 E2 A6
0160h: 18 EB 8D 3F 13 EC 06 F0 C6 54 A7 9A 91 EB AB 1E
0170h: 18 EB 8D 3F 13 EC 06 F0 C6 54 A7 9A 91 EB AB 1E
0180h: 51 27 91 98 2C 3C 3A 50 27 D8 FE 1A 1D E7 E9 C6
0190h: 51 27 91 98 2C 3C 3A 50 27 D8 FE 1A 1D E7 E9 C6
```

How to break Flash Encryption?

- I did some tests (believe me…)
  - Did not find particular Weakness to access the Key by SW or to Attack by DFA
- My Last Hope is Side Channel Analysis
  - to target the Bootloader decryption
- But my setup is too ‘limited’
  - SPI bus producing a lot of Noise
  - Cannot control the SPI frames
    - Use a kind of SPI emulator but BIG FAIL
  - I tried DPA, CPA…
    - Low SNR, No good Leakage…
- 8-9 NIGHTS, ZERO result…
- K.O



```
I (973) cpu_start: Pro cpu start user code
I (320) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello from SEC boot K1 & FE !
```

Flash Encryption Conclusion

- I lost…

EXTRA-COIN

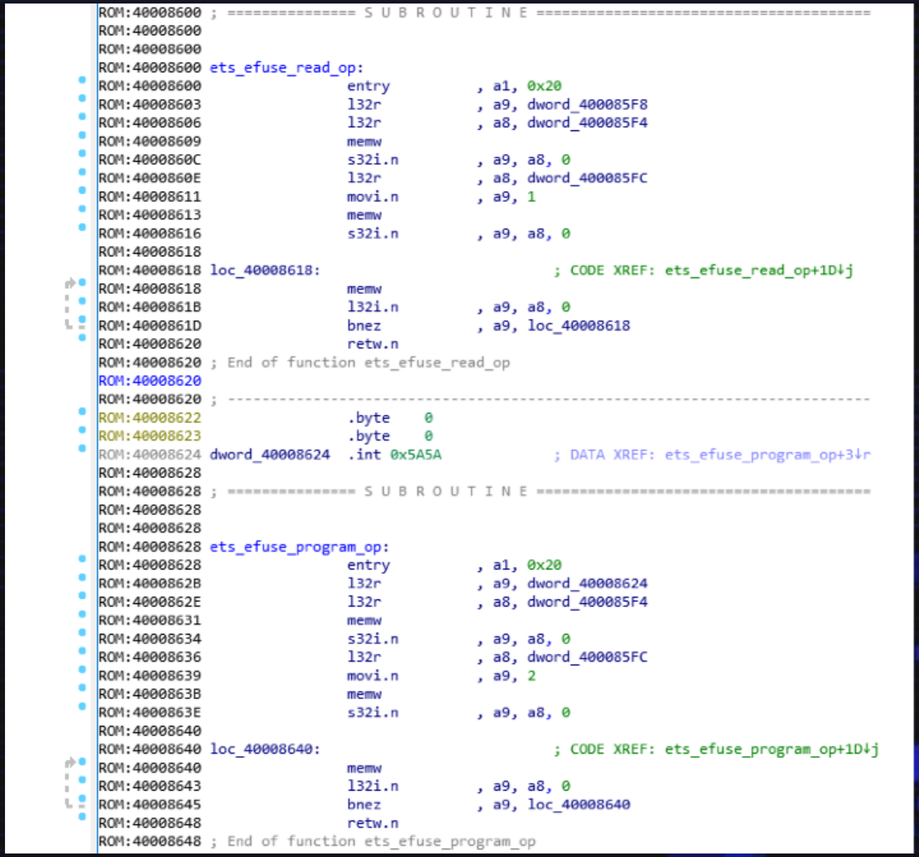
OTP/E-FUSES: THE MOTHER OF VULNS

Role of OTP/E-Fuses

- One-Time-Programmable (OTP) Memory based on E-Fuses
  - Non-Volatile-Memory inside the ESP32
  - An e-Fuse can be ‘programmed’ just ‘One-Time’ from 0 to 1
  - Once burned, no possibility to rewrite it or to wipe it
- Organisation
  - EFUSE_BLK0 = ESP32 configuration
  - EFUSE_BLK1 = Flash Encryption Key (FEK)
  - EFUSE_BLK2 = Secure Boot Key (SBK)
  - EFUSE_BLK3 = reserved for User Application
- According to Espressif, these E-Fuses are R/W protected and cannot be readout/modified once protection bits set
- E-Fuses are managed by the E-Fuses Controller, a dedicated piece of HW inside the ESP32

ESP32 E-Fuses Reverse

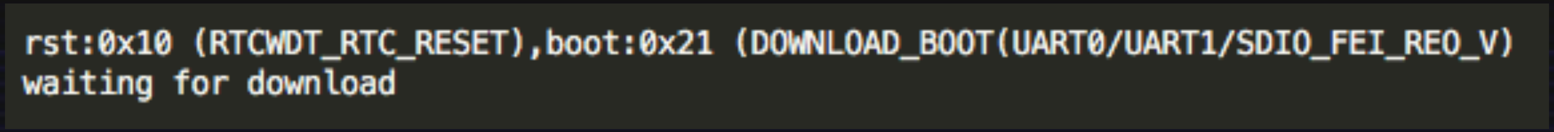
- Only two identified functions
- Used during a ‘Special Boot mode’
  - interesting…
- BootROM never touch OTP values
- It means only the E-Fuses Controller has access to OTP
  - Pure HW Process
  - Has to be set before BootROM



```
ROM:40008600 ; =============== S U B R O U T I N E =======================================
ROM:40008600
ROM:40008600
ROM:40008600 ets_efuse_read_op:
ROM:40008600                 entry           , a1, 0x20
ROM:40008603                 l32r            , a9, dword_400085F8
ROM:40008606                 l32r            , a8, dword_400085F4
ROM:40008609                 memw
ROM:4000860C                 s32i.n          , a9, a8, 0
ROM:4000860E                 l32r            , a8, dword_400085FC
ROM:40008611                 movi.n          , a9, 1
ROM:40008613                 memw
ROM:40008616                 s32i.n          , a9, a8, 0
ROM:40008618
ROM:40008618 loc_40008618:                           ; CODE XREF: ets_efuse_read_op+1D↓j
ROM:40008618                 memw
ROM:4000861B                 l32i.n          , a9, a8, 0
ROM:4000861D                 bnez            , a9, loc_40008618
ROM:40008620                 retw.n
ROM:40008620 ; End of function ets_efuse_read_op
ROM:40008620
ROM:40008620 ; ---------------------------------------------------------------------------
ROM:40008622                 .byte    0
ROM:40008623                 .byte    0
ROM:40008624 dword_40008624  .int 0x5A5A             ; DATA XREF: ets_efuse_program_op+3↓r
ROM:40008628
ROM:40008628 ; =============== S U B R O U T I N E =======================================
ROM:40008628
ROM:40008628
ROM:40008628 ets_efuse_program_op:
ROM:40008628                 entry           , a1, 0x20
ROM:4000862B                 l32r            , a9, dword_40008624
ROM:4000862E                 l32r            , a8, dword_400085F4
ROM:40008631                 memw
ROM:40008634                 s32i.n          , a9, a8, 0
ROM:40008636                 l32r            , a8, dword_400085FC
ROM:40008639                 movi.n          , a9, 2
ROM:4000863B                 memw
ROM:4000863E                 s32i.n          , a9, a8, 0
ROM:40008640
ROM:40008640 loc_40008640:                           ; CODE XREF: ets_efuse_program_op+1D↓j
ROM:40008640                 memw
ROM:40008643                 l32i.n          , a9, a8, 0
ROM:40008645                 bnez            , a9, loc_40008640
ROM:40008648                 retw.n
ROM:40008648 ; End of function ets_efuse_program_op
```

Special Boot Mode

- Special Boot Mode (Download_Boot)
  - Management mode to Flash FW, and Set E-Fuses
  - IO0 connected to GND then Power-up

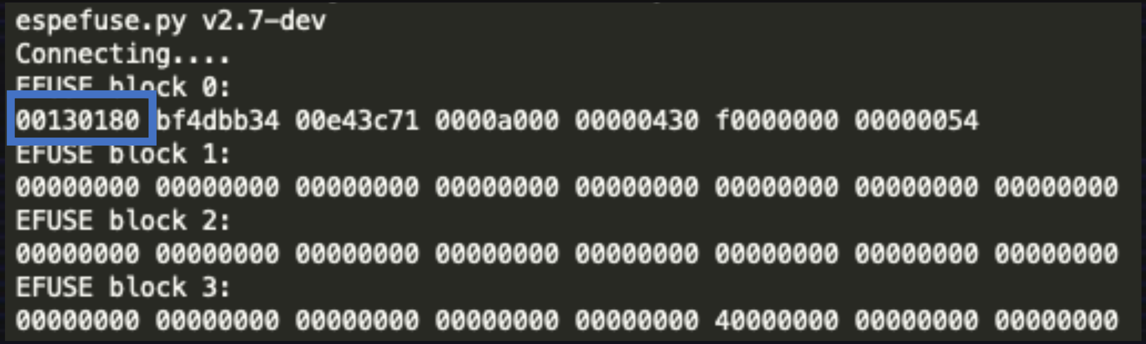


```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x21 (DOWNLOAD_BOOT(UART0/UART1/SDIO_FEI_REO_V)
waiting for download
```

- Esptool is python utility to communicate with the ROM functions
  - Dedicated commands available from UART0 to deal with E-Fuses
    - dump, program,…

E-Fuses Protection

- Any attempt to read BLK1 or BLK2 returns 0x00
  - $ espefuse.py --port /dev/ttyUSB0 dump



```
espefuse.py v2.7-dev
Connecting....
EFUSE block 0:
00130180 bf4dbb34 00e43c71 0000a000 00000430 f0000000 00000054
EFUSE block 1:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 2:
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
EFUSE block 3:
00000000 00000000 00000000 00000000 00000000 40000000 00000000 00000000
```

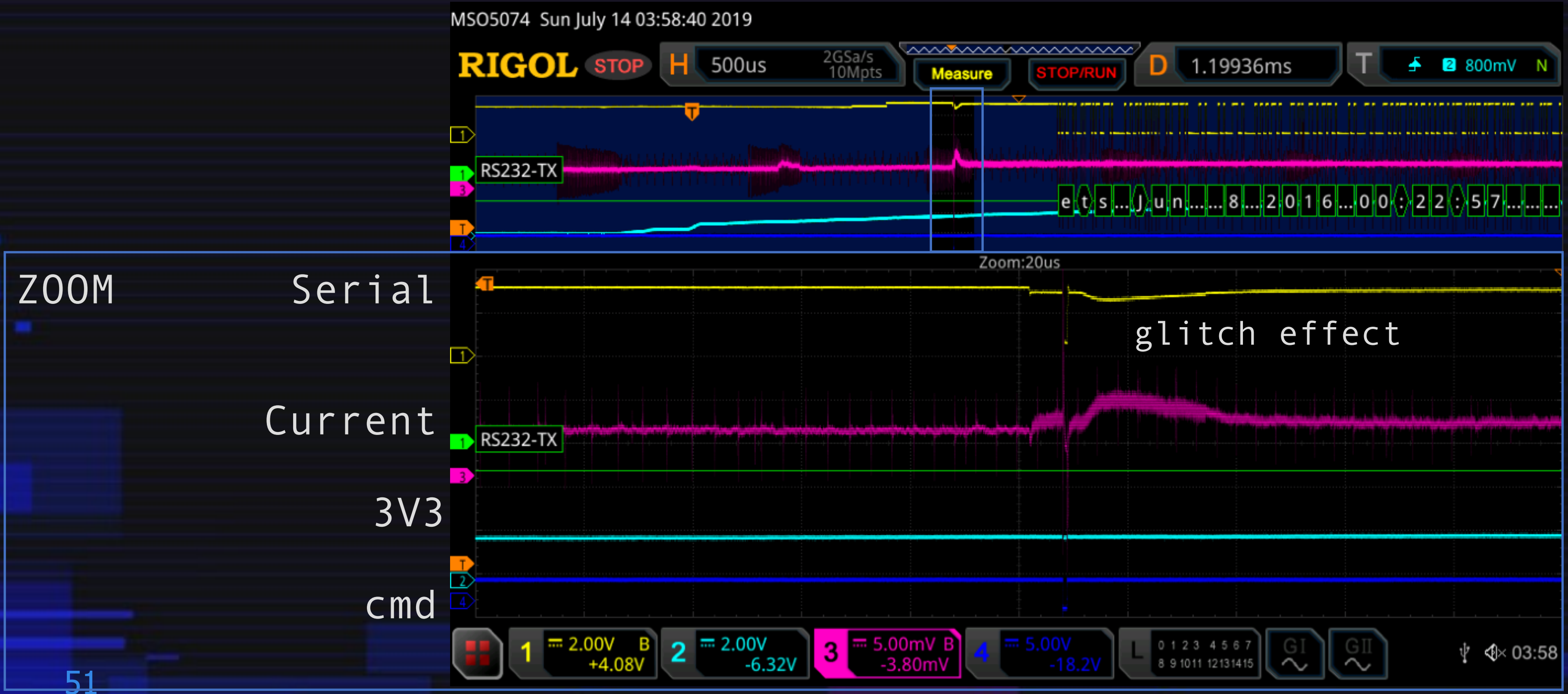
- Identification of R/W Protection bits in BLK0
- 00130180 = 00000000 00010011 00000001 1000 0000
  - These two bits are the Read protection bits

Wait LR, where is the Vuln?

- I Have no vuln here…
- But I know
  - BootROM does not manage the E-Fuses
  - Obviously, E-Fuses Controller does the job before
  - Special boot mode called ‘Download_Boot’
  - Read protection bits have been identified
- The idea
  - Glitch the E-Fuses Controller initialization to modify the R/W protections
  - Then send Dump command in Special Mode
  - And get back BLK1 (FEK) and BLK2 (SBK)

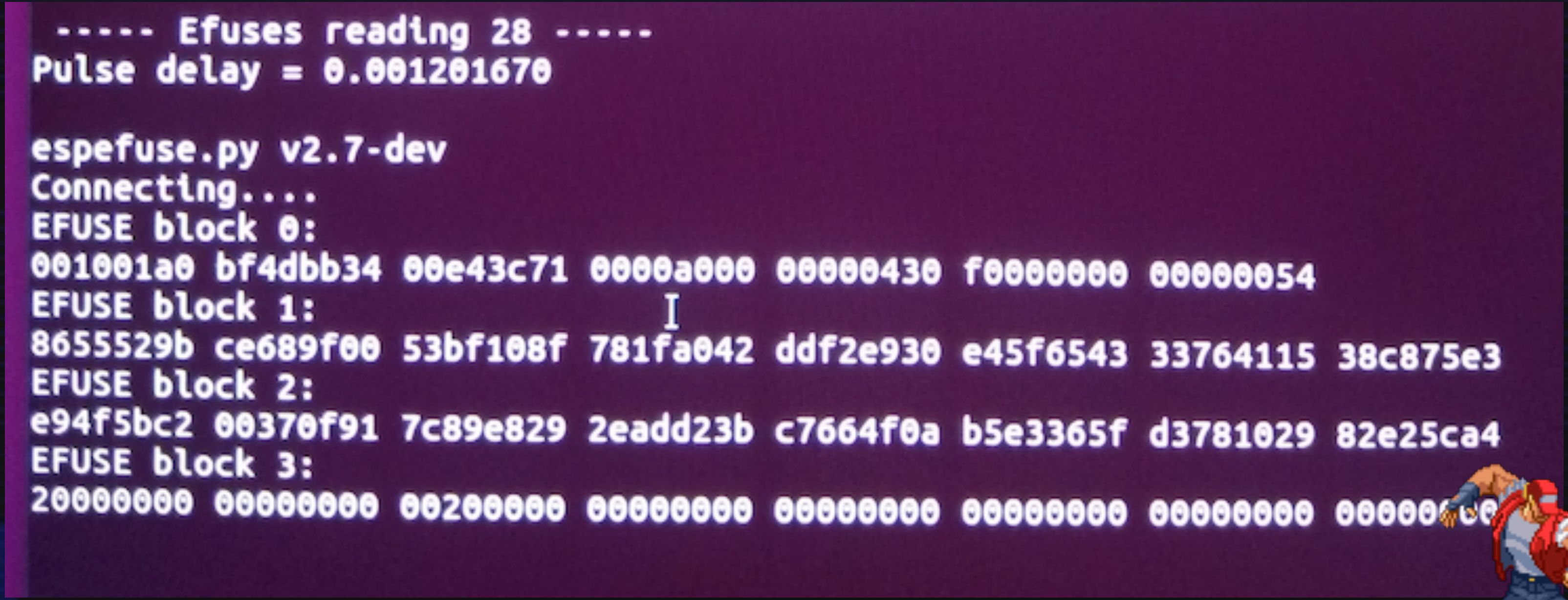
FATAL Attack

- Simple Power Analysis on VDD_CPU to identify
- Glitch during this identified HW process



FATAL Results

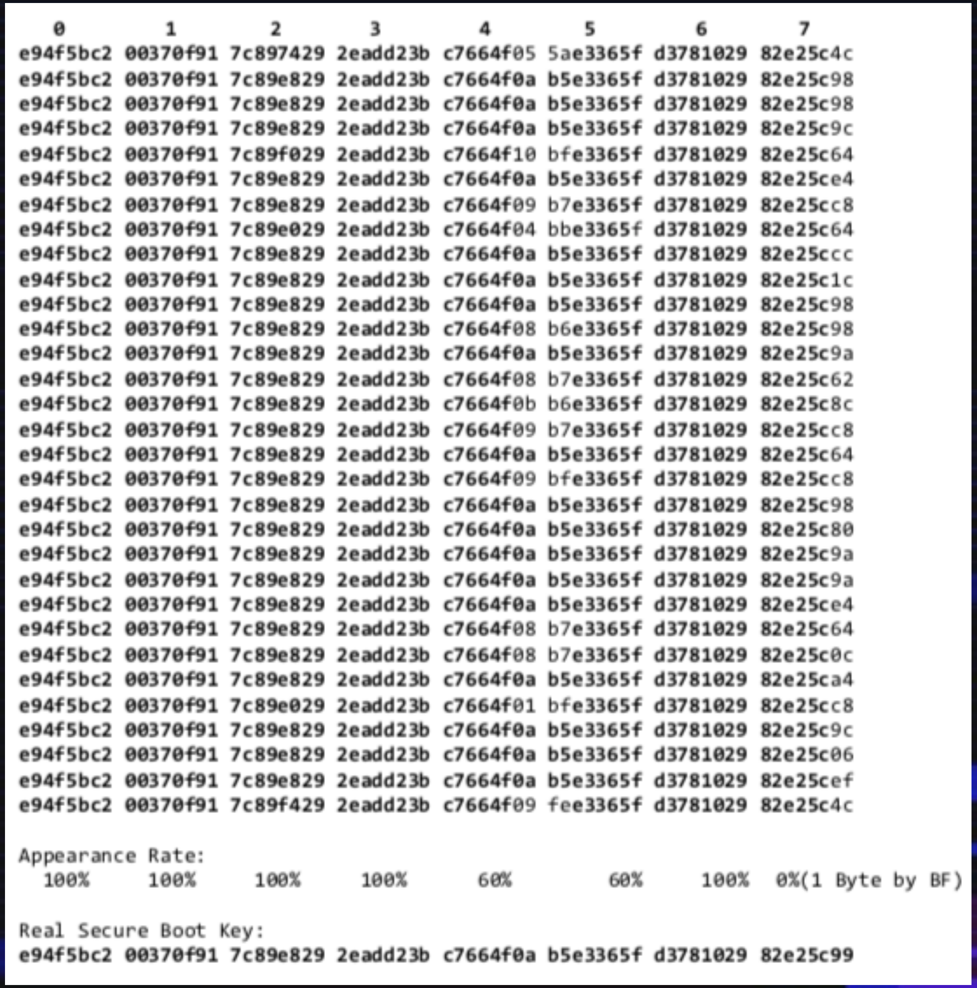
- PoC sent to vendor (on July 24)



```
 ----- Efuses reading 28 -----
Pulse delay = 0.001201670

espefuse.py v2.7-dev
Connecting....
EFUSE block 0:
001001a0 bf4dbb34 00e43c71 0000a000 00000430 f0000000 00000054
EFUSE block 1:
8655529b ce689f00 53bf108f 781fa042 ddf2e930 e45f6543 33764115 38c875e3
EFUSE block 2:
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25ca4
EFUSE block 3:
20000000 00000000 00200000 00000000 00000000 00000000 00000000 00000
```

One more step

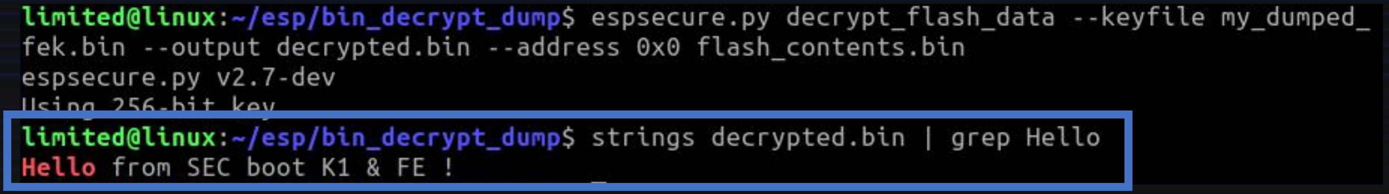
- Sadly, the dumped Keys are not exactly True values
  - Remember I burned the keys ☺
- Offline Statistical Analysis on 30-50 dumped key values
  - just Keep the most recurrent Bytes (here SBK analysis)
- 1 Byte still unknown and has to be Brute Forced (worst case)
  - Same for FEK



```
   0        1        2        3        4        5        6        7
e94f5bc2 00370f91 7c897429 2eadd23b c7664f05 5ae3365f d3781029 82e25c4c
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c98
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c98
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c9c
e94f5bc2 00370f91 7c89f029 2eadd23b c7664f10 bfe3365f d3781029 82e25c64
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25ce4
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f09 b7e3365f d3781029 82e25cc8
e94f5bc2 00370f91 7c89e029 2eadd23b c7664f04 bbe3365f d3781029 82e25c64
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25ccc
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c1c
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c98
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f08 b6e3365f d3781029 82e25c98
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c9a
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f08 b7e3365f d3781029 82e25c62
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0b b6e3365f d3781029 82e25c8c
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f09 b7e3365f d3781029 82e25cc8
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c64
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f09 bfe3365f d3781029 82e25cc8
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c98
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c80
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c9a
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c9a
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25ce4
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f08 b7e3365f d3781029 82e25c64
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f08 b7e3365f d3781029 82e25c0c
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25ca4
e94f5bc2 00370f91 7c89e029 2eadd23b c7664f01 bfe3365f d3781029 82e25cc8
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c9c
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c06
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25cef
e94f5bc2 00370f91 7c89f429 2eadd23b c7664f09 fee3365f d3781029 82e25c4c

Appearance Rate:
  100%     100%     100%     100%      60%         60%     100%  0%(1 Byte by BF)

Real Secure Boot Key:
e94f5bc2 00370f91 7c89e829 2eadd23b c7664f0a b5e3365f d3781029 82e25c99
```

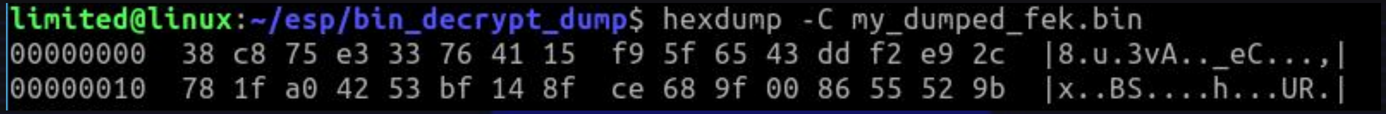
FATAL Exploit step 1: Decrypt FW

- Dump the encrypted FW
  - By Download Mode or by dumping the Flash Content
- Perform FATAL Glitch to extract FEK/SBK values
  - Run Statistical analysis
- Confirm the True FEK (by decrypting FW)



```
limited@linux:~/esp/bin_decrypt_dump$ espsecure.py decrypt_flash_data --keyfile my_dumped_
fek.bin --output decrypted.bin --address 0x0 flash_contents.bin
espsecure.py v2.7-dev
Using 256-bit key
limited@linux:~/esp/bin_decrypt_dump$ strings decrypted.bin | grep Hello
Hello from SEC boot K1 & FE !
```

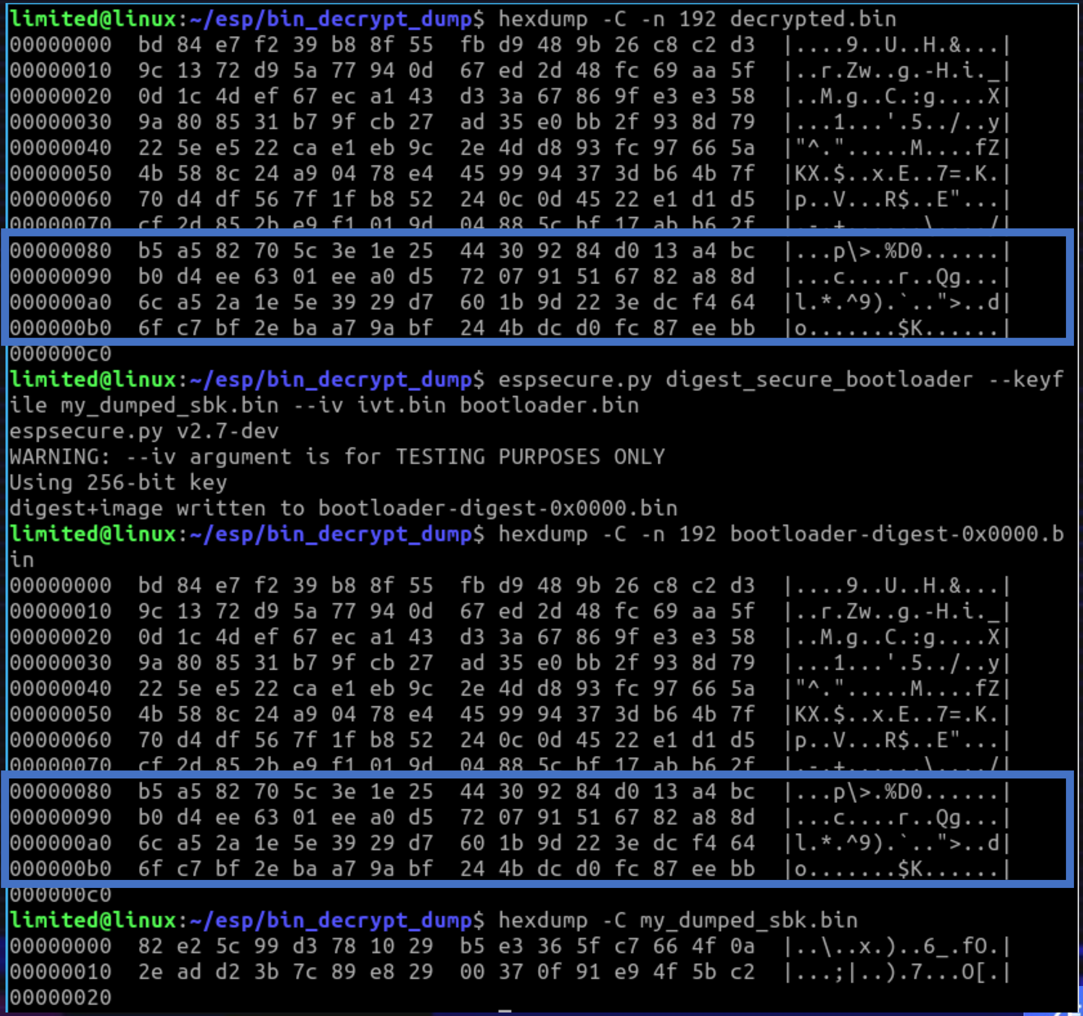
  - IMPORTANT to respect this byte order in key.bin



```
limited@linux:~/esp/bin_decrypt_dump$ hexdump -C my_dumped_fek.bin
00000000  38 c8 75 e3 33 76 41 15  f9 5f 65 43 dd f2 e9 2c  |8.u.3vA.._eC...,|
00000010  78 1f a0 42 53 bf 14 8f  ce 68 9f 00 86 55 52 9b  |x..BS....h...UR.|
```

FATAL Exploit step 2: Sign Your Code

- Firmware is now decrypted
- dd ivt.bin (the first 128 random bytes at 0x00 in decrypted.bin)
- dd Bootloader.bin at 0x1000
- Confirm the true SBK
  - digest computation command
- Write your Code
  - a little FW backdoor maybe? ☺
- Compile images
  - using FEK and SBK
- Flash new FW



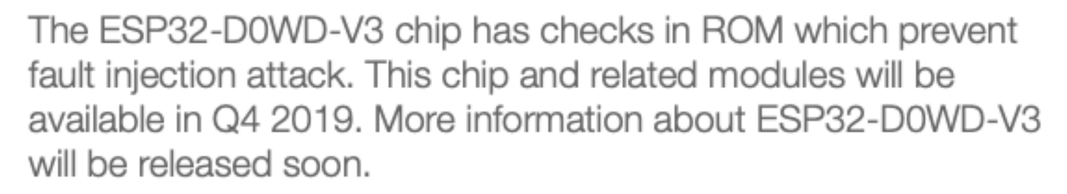
```
limited@linux:~/esp/bin_decrypt_dump$ hexdump -C -n 192 decrypted.bin
00000000  bd 84 e7 f2 39 b8 8f 55  fb d9 48 9b 26 c8 c2 d3  |....9..U..H.&...|
00000010  9c 13 72 d9 5a 77 94 0d  67 ed 2d 48 fc 69 aa 5f  |..r.Zw..g.-H.i._|
00000020  0d 1c 4d ef 67 ec a1 43  d3 3a 67 86 9f e3 e3 58  |..M.g..C.:g....X|
00000030  9a 80 85 31 b7 9f cb 27  ad 35 e0 bb 2f 93 8d 79  |...1...'.5../..y|
00000040  22 5e e5 22 ca e1 eb 9c  2e 4d d8 93 fc 97 66 5a  |"^.".....M....fZ|
00000050  4b 58 8c 24 a9 04 78 e4  45 99 94 37 3d b6 4b 7f  |KX.$..x.E..7=.K.|
00000060  70 d4 df 56 7f 1f b8 52  24 0c 0d 45 22 e1 d1 d5  |p..V...R$..E"...|
00000070  cf 2d 85 2b e9 f1 01 9d  04 88 5c bf 17 ab b6 2f  |.-.+......\..../|
00000080  b5 a5 82 70 5c 3e 1e 25  44 30 92 84 d0 13 a4 bc  |...p\>.%D0......|
00000090  b0 d4 ee 63 01 ee a0 d5  72 07 91 51 67 82 a8 8d  |...c....r..Qg...|
000000a0  6c a5 2a 1e 5e 39 29 d7  60 1b 9d 22 3e dc f4 64  |l.*.^9).`..">..d|
000000b0  6f c7 bf 2e ba a7 9a bf  24 4b dc d0 fc 87 ee bb  |o.......$K......|
000000c0
limited@linux:~/esp/bin_decrypt_dump$ espsecure.py digest_secure_bootloader --keyf
ile my_dumped_sbk.bin --iv ivt.bin bootloader.bin
espsecure.py v2.7-dev
WARNING: --iv argument is for TESTING PURPOSES ONLY
Using 256-bit key
digest+image written to bootloader-digest-0x0000.bin
limited@linux:~/esp/bin_decrypt_dump$ hexdump -C -n 192 bootloader-digest-0x0000.b
in
00000000  bd 84 e7 f2 39 b8 8f 55  fb d9 48 9b 26 c8 c2 d3  |....9..U..H.&...|
00000010  9c 13 72 d9 5a 77 94 0d  67 ed 2d 48 fc 69 aa 5f  |..r.Zw..g.-H.i._|
00000020  0d 1c 4d ef 67 ec a1 43  d3 3a 67 86 9f e3 e3 58  |..M.g..C.:g....X|
00000030  9a 80 85 31 b7 9f cb 27  ad 35 e0 bb 2f 93 8d 79  |...1...'.5../..y|
00000040  22 5e e5 22 ca e1 eb 9c  2e 4d d8 93 fc 97 66 5a  |"^.".....M....fZ|
00000050  4b 58 8c 24 a9 04 78 e4  45 99 94 37 3d b6 4b 7f  |KX.$..x.E..7=.K.|
00000060  70 d4 df 56 7f 1f b8 52  24 0c 0d 45 22 e1 d1 d5  |p..V...R$..E"...|
00000070  cf 2d 85 2b e9 f1 01 9d  04 88 5c bf 17 ab b6 2f  |.-.+......\..../|
00000080  b5 a5 82 70 5c 3e 1e 25  44 30 92 84 d0 13 a4 bc  |...p\>.%D0......|
00000090  b0 d4 ee 63 01 ee a0 d5  72 07 91 51 67 82 a8 8d  |...c....r..Qg...|
000000a0  6c a5 2a 1e 5e 39 29 d7  60 1b 9d 22 3e dc f4 64  |l.*.^9).`..">..d|
000000b0  6f c7 bf 2e ba a7 9a bf  24 4b dc d0 fc 87 ee bb  |o.......$K......|
000000c0
limited@linux:~/esp/bin_decrypt_dump$ hexdump -C my_dumped_sbk.bin
00000000  82 e2 5c 99 d3 78 10 29  b5 e3 36 5f c7 66 4f 0a  |..\..x.)..6_.fO.|
00000010  2e ad d2 3b 7c 89 e8 29  00 37 0f 91 e9 4f 5b c2  |...;|..).7...O[.|
00000020
```

OTP/EFuses FATAL Conclusion

- FATAL exploit leading to SBK & FEK extraction
  - Breaking Secure Boot and Flash Encryption
- An attacker can decrypt the Firmware (access IP and sensitive data)
- An attacker can sign & run his own (encrypted) code PERSISTENTLY
- Low Cost, Low Complexity
- Easy to reproduce
- No Way to fix
- All ESP32 versions vulnerable

Vendor Reaction

- Resp. disclosure
  - PoC sent on July 24
  - CVE-2019-17391 (req. by Vendor)
  - Disclosure Today



The ESP32-D0WD-V3 chip has checks in ROM which prevent fault injection attack. This chip and related modules will be available in Q4 2019. More information about ESP32-D0WD-V3 will be released soon.

- Security Advisory on November 1 [7]
- No way to Fix but…
  - They propose to buy their new chip version ☺ ☺ ☺
- Millions of vulnerable Devices on the field for the coming years
- What about devices offered for sales? Who want broken platforms?

Final Conclusion

- Developers are Now aware
  - Attacker with physical access can compromise ESP32 security badly
- Fix?
  - No fix on current ESP32 version
  - Chip is broken FOREVER
- I identified several companies using Esp32 security features in their products…
- General Message for Vendors
  - Don't patch silently, Reward instead
- New Results coming soon
  - Stay tuned ;)

ZERO NIGHTS 2019 EDITION

References & Credits

- Fatal Fury Animations
  - www.fightersgeneration.com
- Espressif
  - [1] Espressif 100-Millions chip shipments
- ESP32
  - [2] Datasheet, TRM
- Fault injection references
  - [3] Chris Gerlinsky (@akacastor)
  - [4] Colin O'Flynn (@colinoflynn)
- Xtensa
  - [5] ISA Manual
- LIFX Pwn
  - [6] LIFX Pwn
- Security Advisory
  - [7] CVE-2019-17391

Thank you!

@LimitedResults