
Boot security in the MCU
elttam

$ whoami

Zoltan

Daniel

elttam

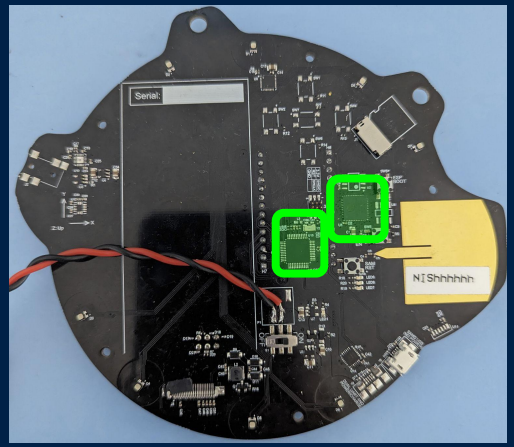
Agenda

1. Introduction
2. ESP32
3. STM32
4. Outroduction

Microcontrollers (MCU)

- Essentially a small computer:
  - Processor
  - Memory
  - Peripherals
  - I/O

- Found in a multitude of devices:
  - Crypto Wallets (e.g. Trezor, Ledger)
  - Wearables (e.g. Apple Airpods)
  - Smart Home (e.g. Google Nest)



elttam

Secure Boot

… in a nutshell, protects the MCU against running unverified code.

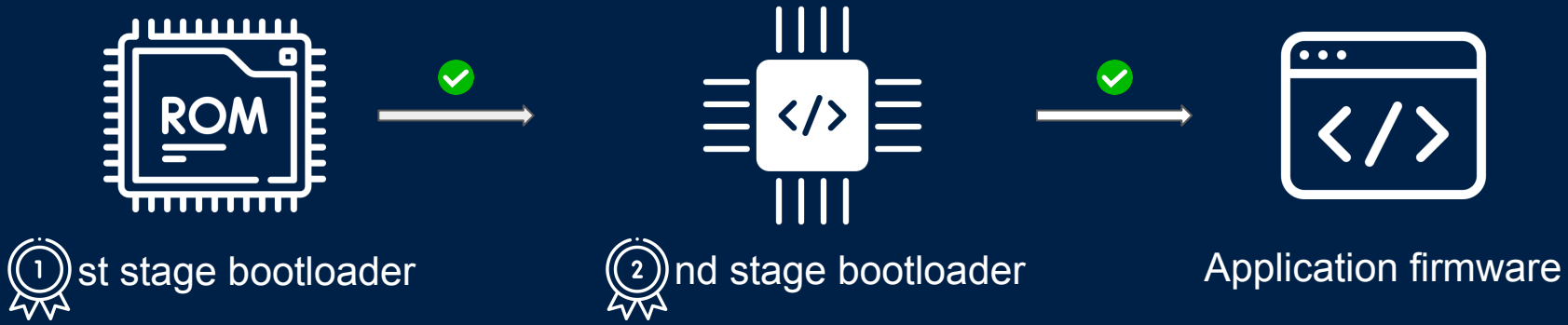
Protection

Detection

Recovery

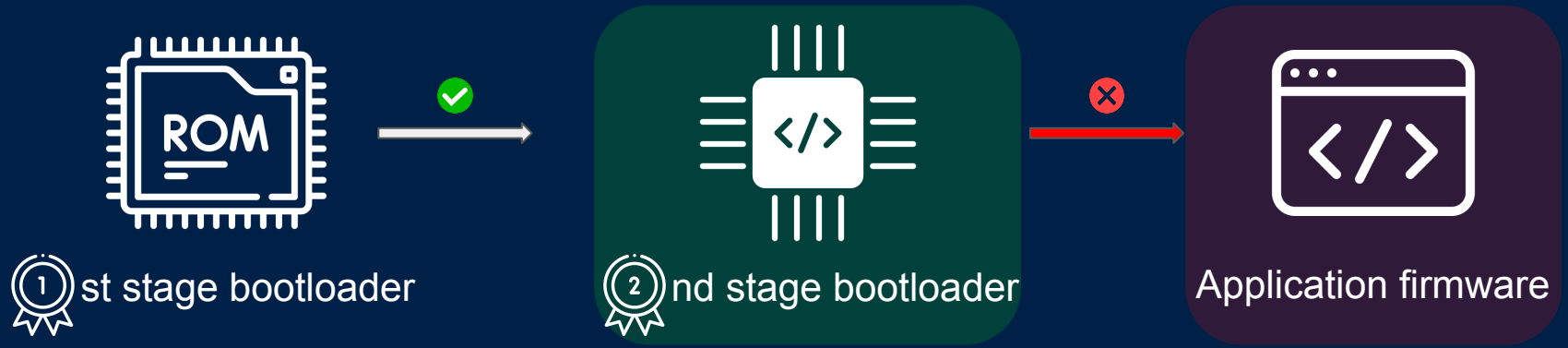
elttam

Secure Boot



elttam

Our focus



“We are not doing this because it’s easy, we are doing it because we thought it would be easy.”

elttam

Espressif Systems ESP32

elttam

Target Selection

esp-idf releases:
v5.3 v5.2 v5.1 v5.0
v4.4 v4.3 v4.2 v4.1 v4.0
v3.3 v3.2 v3.1 v3.0
v2.1 v2.0

ESP32 Family

ESP32-series:
ESP32-D0WD-V3
ESP32-D0WDR2-V3
ESP32-U4WDH
ESP32-D0WD
ESP32-D0WDQ6
ESP32-D0WDQ6-V3
ESP32-S0WD
ESP32-PICO-V3
ESP32-PICO-V3-02
ESP32-PICO-D4

C2-series:
ESP32-C2
ESP8684H2
ESP8684H4

C6-series:
ESP32-C6
ESP32-C6FH4
ESP32-C6FH8

C3-series:
ESP32-C3
ESP32-C3FN4
ESP32-C3FH4
ESP32-C3FH4AZ
ESP32-C3FH4X
ESP8685H4

S2-series:
ESP32-S2
ESP32-S2FH2
ESP32-S2FH4
ESP32-S2FN4R2
ESP32-S2R2

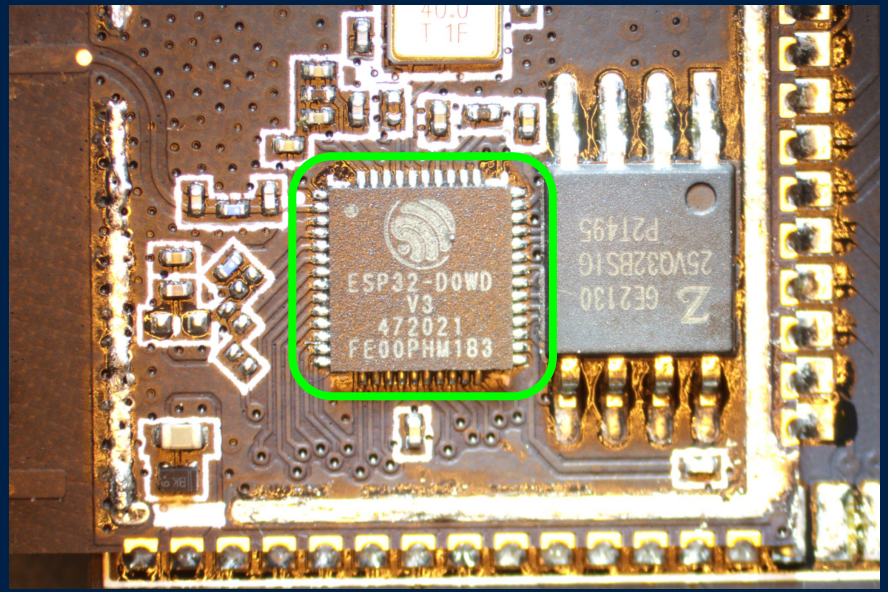
S3-series:
ESP32-S3
ESP32-S3R2
ESP32-S3R8
ESP32-S3-PICO-1-N8R2
ESP32-S3R8V
ESP32-S3FN8
ESP32-S3FH4R2
ESP32-S3-PICO-1-N8R8

P-series:
ESP32-P4NRW16
ESP32-P4NRW32

H-series:
ESP32-H2FH2
ESP32-H2FH4

Target Selection

- ESP32-D0WD-V3
  - Xtensa® dual-core 32-bit **LX6** CPU
  - Security features:
    - 1024 bit OTP (eFuse)
    - Crypto Hardware Acceleration
    - Secure Boot v2
    - Flash Encryption
- esp-idf framework
  - releases 5.1, 4.1, 3.1, 2.1



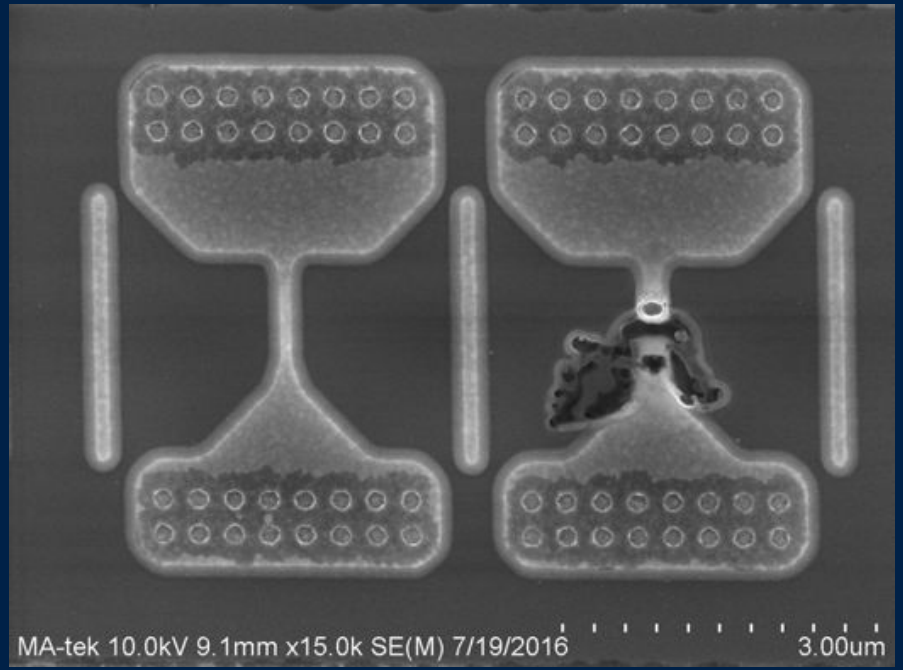
elttam

Doc Review

“It is highly recommended that Secure Boot be enabled on all production devices.”

elttam

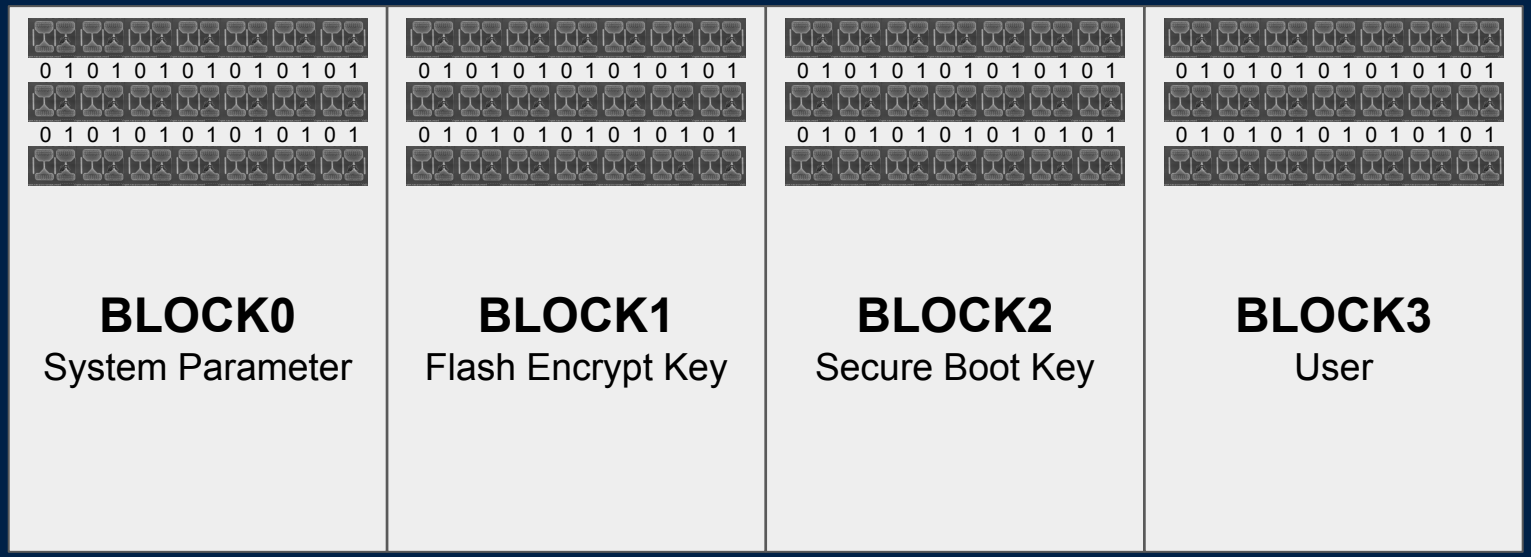
One Time Programmable (OTP) memory / eFuse



source

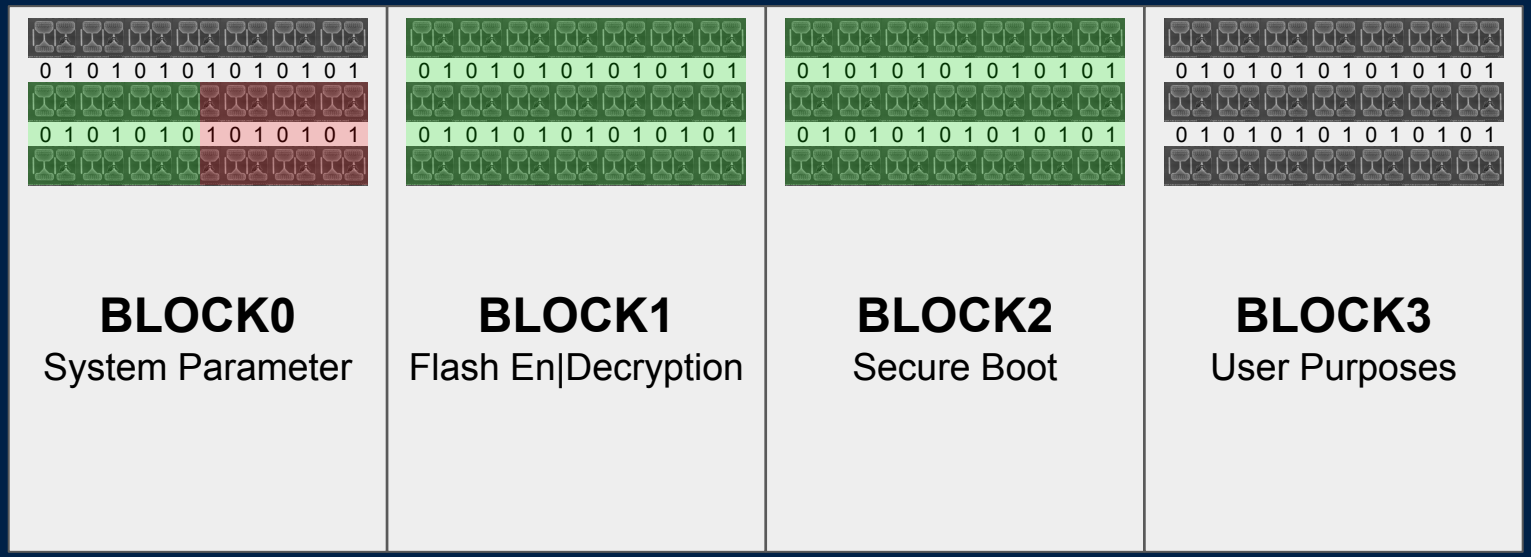
elttam

eFuse blocks: Allow for permanent configuration



elttam

eFuse blocks: Also supports read/write protection



elttam

e.g. No write protect on MAC_FACTORY (BLOCK0)

```
vscode@4a7f9f125b0a:/opt/hello_world$ espefuse.py burn_bit BLOCK0 33 34 36 39 41 44 45 46 47 48 49 50 51 52 53 54 56 57 58 60 62 63 65 67 68 70 71 72
espefuse.py v4.7.0
Connecting....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting....
Detecting chip type... ESP32

=== Run "burn_bit" command ===
bit_number:   [223].....................................................[0]
BLOCK0    : 0x00000000000000000000000000000000000001dad77ff29600000000
BLOCK0          (                ) [0 ] regs_to_write: 00000000 d77ff296 000001da 00000000 00000000 00000000 00000000

Check all blocks for burn...
idx, BLOCK_NAME,          Conclusion
[00] BLOCK0               is not empty
        (written ): 0x00000004001000000000007380000a20000e3d29ffdd6b6ff00000000
        (to write): 0x00000000000000000000000000000000000001dad77ff29600000000
        (coding scheme = NONE)
.
This is an irreversible operation!
Type 'BURN' (all capitals) to continue.
BURN
BURN BLOCK0  - OK (all write block bits are set)
Reading updated efuses...
Successful
vscode@4a7f9f125b0a:/opt/hello_world$
```

d2:9f:fd:d6:b6:ff

↓

d3:df:ff:ff:f6:ff

elttam

e.g. JTAG_DISABLE (BLOCK0)

```
vscode@4a7f9f125b0a:/opt/hello_world$ espefuse.py summary | grep JTAG_DISABLE
  JTAG_DISABLE (BLOCK0)                          Disable JTAG                                = False R/W (0b0)
vscode@4a7f9f125b0a:/opt/hello_world$
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 3 ESP-IDF

```
[esp32.cpu0] Target halted, PC=0x400D5524, debug_reason=00000001
Set GDB target to 'esp32.cpu0'
[esp32.cpu1] Target halted, PC=0x400846DA, debug_reason=00000000
[New Thread 1073413024]
[New Thread 1073413712]
[New Thread 1073413368]
[New Thread 1073412272]
[New Thread 1073412616]
[New Thread 1073410900]
[Switching to Thread 1073413024]

Thread 2 "main" hit Temporary breakpoint 1, app_main () at /opt/hello_world/main/hello_world_main.c:16
16      {
(failed reverse-i-search)`dump': Quit
(failed reverse-i-search)`dumpdump binary memory rodata.bin': Quit
(gdb) dump binary memory rom.bin 0x40000000 0x40070000
(gdb) dump binary memory rodata.bin 0x3f400000 0x3fbfffff
(gdb)
```

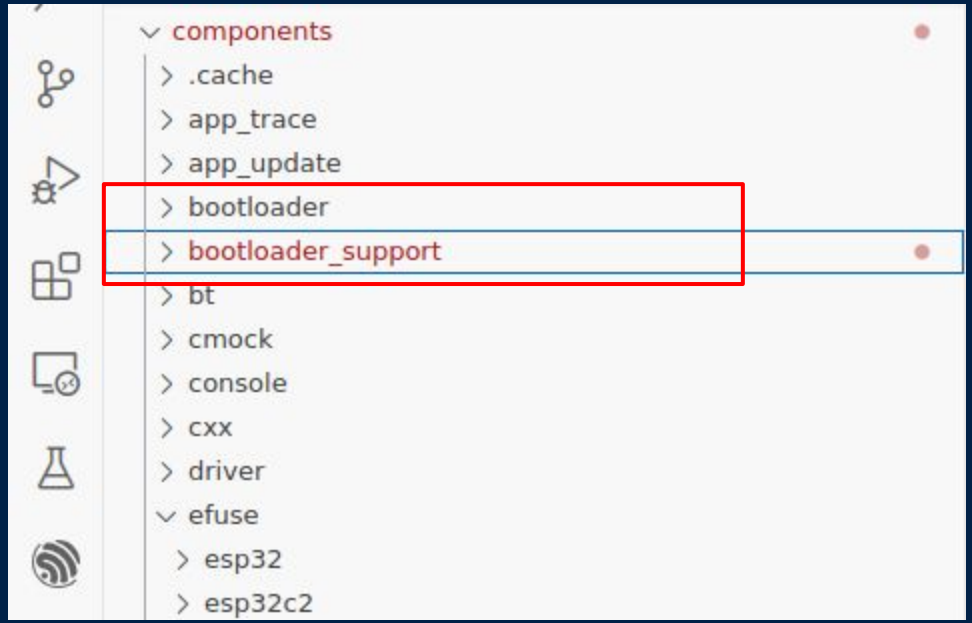
```
vscode@4a7f9f125b0a:/opt/hello_world$ strings ./rodata.bin | head
hello_world
23:51:11
Sep 17 2024
v5.1.4-802-g8af42a08cf
cpu_start
[0;32mI (%lu) %s: App cpu up.
[0;32mI (%lu) %s: Multicore app
[0;32mI (%lu) %s: Pro cpu up.
[0;31mE (%lu) %s: Running on single core variant of a chip, but app is built with multi-core support.
[0;31mE (%lu) %s: Check that CONFIG_FREERTOS_UNICORE is enabled in menuconfig
vscode@4a7f9f125b0a:/opt/hello_world$
```

elttam

Code Audit Approach

- MCU is configured according to best practices
- Audit all the things!

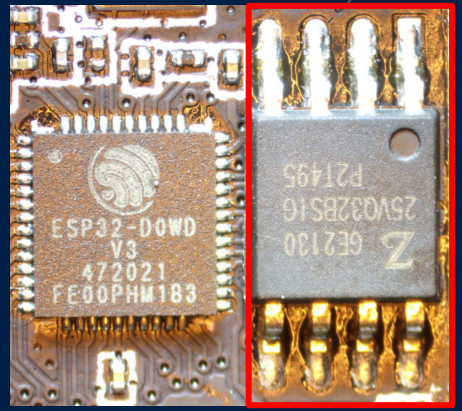
- … aka read all code from `call_start_cpu0()`

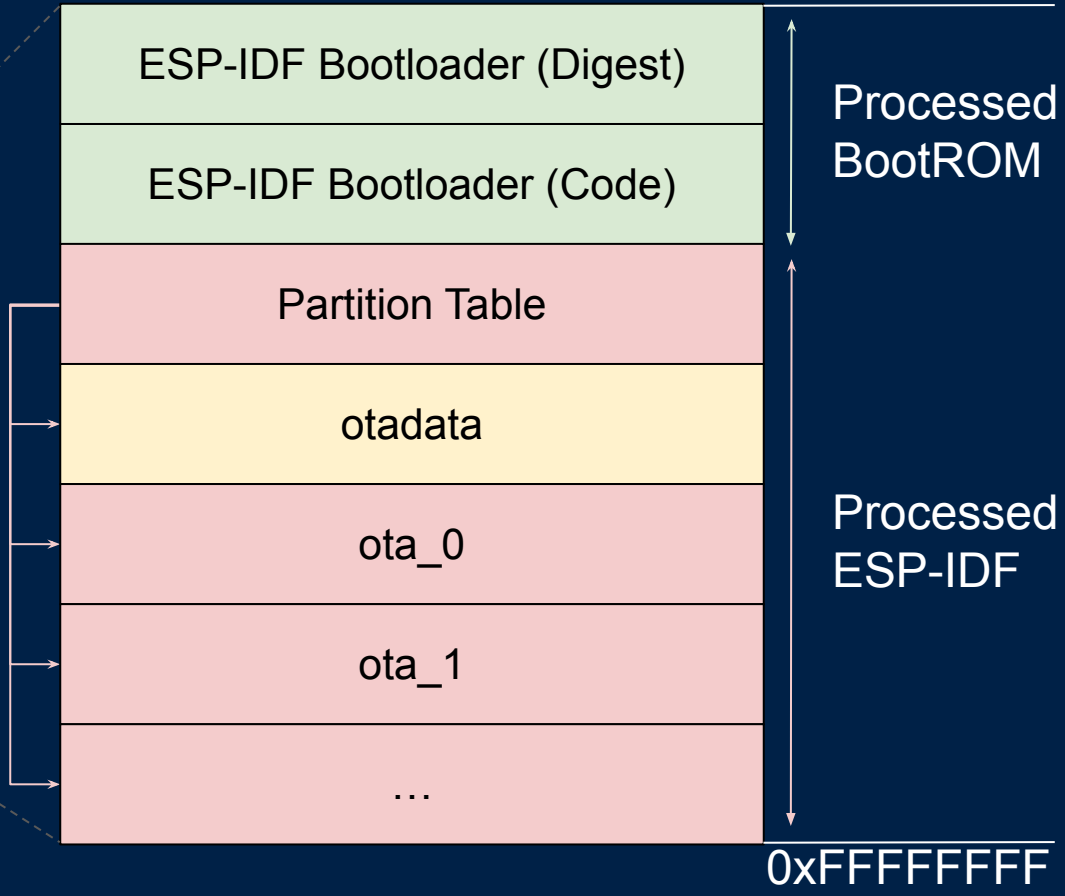


elttam

```
void __attribute__((noreturn)) call_start_cpu0(void)
{
    // ...
    // 1. Hardware initialization
    if (bootloader_init() != ESP_OK) {
        bootloader_reset();
    }
    // 2. Select the number of boot partition
    bootloader_state_t bs = {0};
    int boot_index = select_partition_number(&bs);
    if (boot_index == INVALID_INDEX) {
        bootloader_reset();
    }
    // 3. Load the app image for booting
    bootloader_utility_load_boot_image(&bs, boot_index);
}
```

SPI Flash Layout





elttam

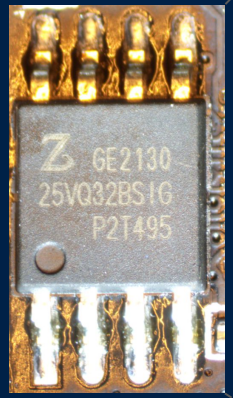
```
void __attribute__((noreturn)) call_start_cpu0(void)
{
    // ...
    // 1. Hardware initialization
    if (bootloader_init() != ESP_OK) {
        bootloader_reset();
    }
    // 2. Select the number of boot partition
    bootloader_state_t bs = {0};
    int boot_index = select_partition_number(&bs);
    if (boot_index == INVALID_INDEX) {
        bootloader_reset();
    }
    // 3. Load the app image for booting
    bootloader_utility_load_boot_image(&bs, boot_index);
}
```

```
void bootloader_utility_load_boot_image(const bootloader_state_t *bs, int
start_index)
{
    int index = start_index;
    esp_partition_pos_t part;
    esp_image_metadata_t image_data = {0};
    // ...
    for (index = start_index; index >= FACTORY_INDEX; index--) {
        part = index_to_partition(bs, index);
        if (part.size == 0) {
            continue;
        }
        if (check_anti_rollback(&part) && try_load_partition(&part, &image_data))
        {
            set_actual_ota_seq(bs, index);
            load_image(&image_data);
        }
```

Anti Rollback

- Updates received over-the-air are written to an unused partition.
- The update must have a version number greater than or equal to the currently running firmware.
- Prevents installing **signed** firmware containing a known vulnerability which can be exploited.

elttam



ESP-IDF Bootloader (Digest)

ESP-IDF Bootloader (Code)

Partition Table

otadata

ota_0

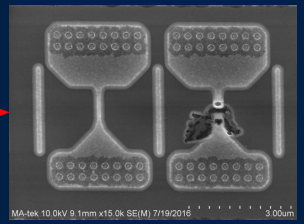
ota_1

2. change active partition

3. next boot

1. ota firmware update

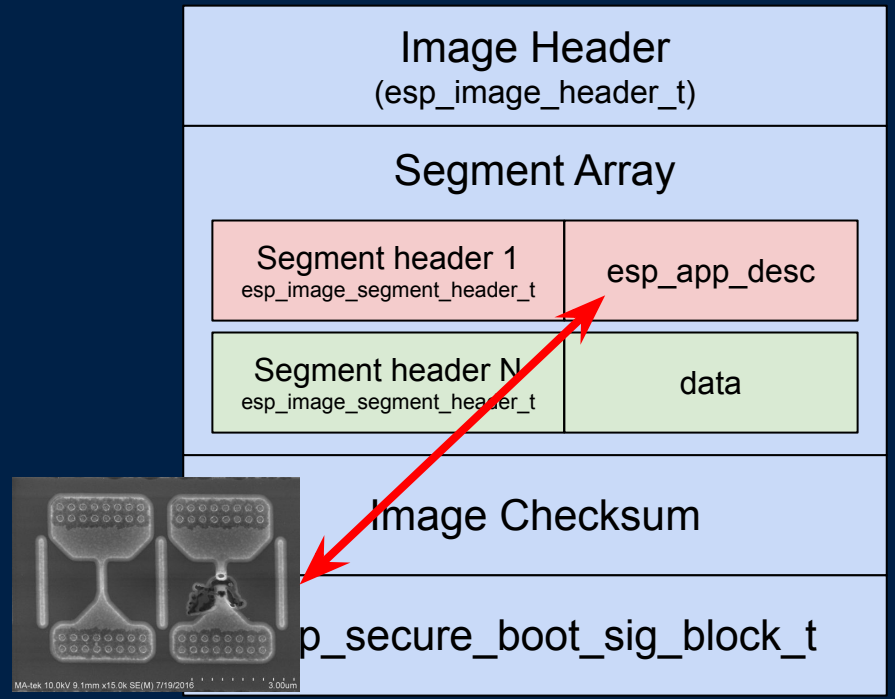
4. blow fuse



elttam

Anti Rollback

- After the first segment header in an application image is an esp_app_desc structure.
- The structure contains a `secure_version` field.
- This must be ≥ `ESP_EFUSE_SECURE_VERSION` stored in eFuse `BLOCK3`.



elttam

```
void bootloader_utility_load_boot_image(const bootloader_state_t *bs, int
start_index)
{
    int index = start_index;
    esp_partition_pos_t part;
    esp_image_metadata_t image_data = {0};

    for (index = start_index; index >= FACTORY_INDEX; index--) {
        part = index_to_partition(bs, index);
        if (part.size == 0) {
            continue;
        }
        if (check_anti_rollback(&part) && try_load_partition(&part, &image_data))
        {
            set_actual_ota_seq(bs, index);
            load_image(&image_data);
        }
```

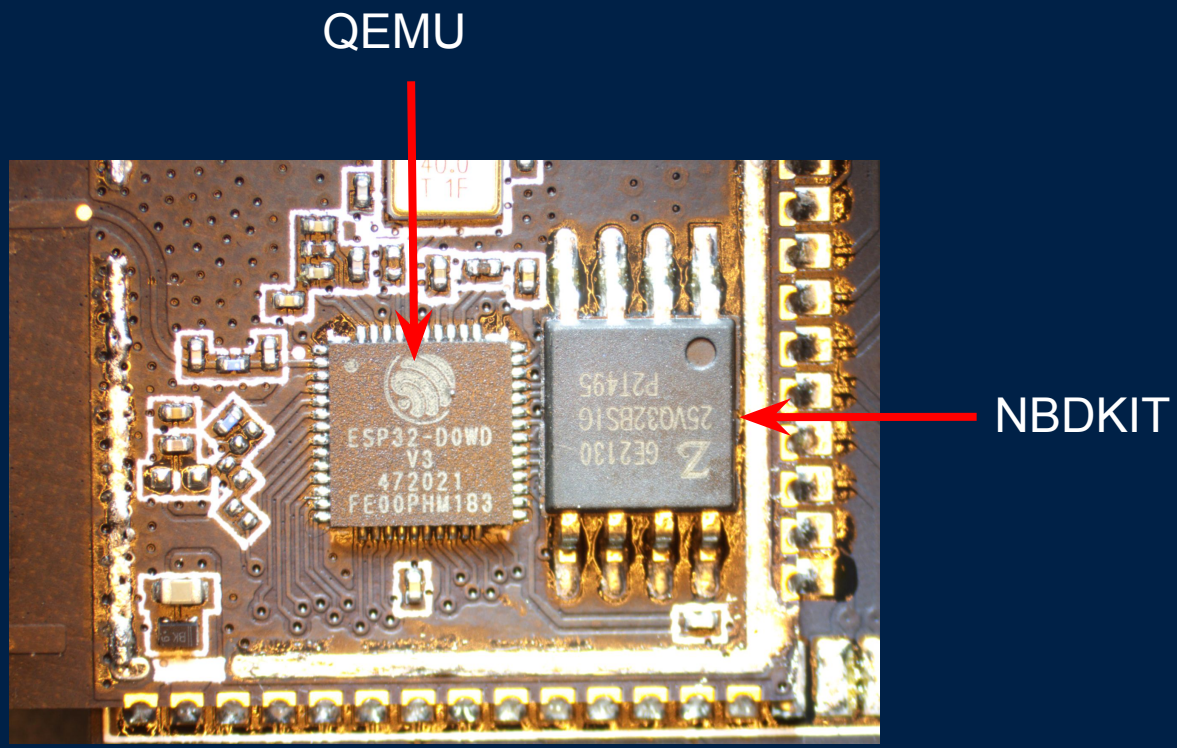
```
void bootloader_utility_load_boot_image(const bootloader_state_t *bs, int
start_index)
{
    int index = start_index;
    esp_partition_pos_t part;
    esp_image_metadata_t image_data = {0};

    for (index = start_index; index >= FACTORY_INDEX; index--) {
        part = index_to_partition(bs, index);
        if (part.size == 0) {
            continue;
        }
        if (check_anti_rollback(&part) && try_load_partition(&part, &image_data))
        {
            set_actual_ota_seq(bs, index);
            load_image(&image_data);
        }
```

Time of check

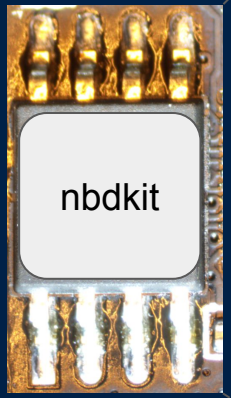
Time of use

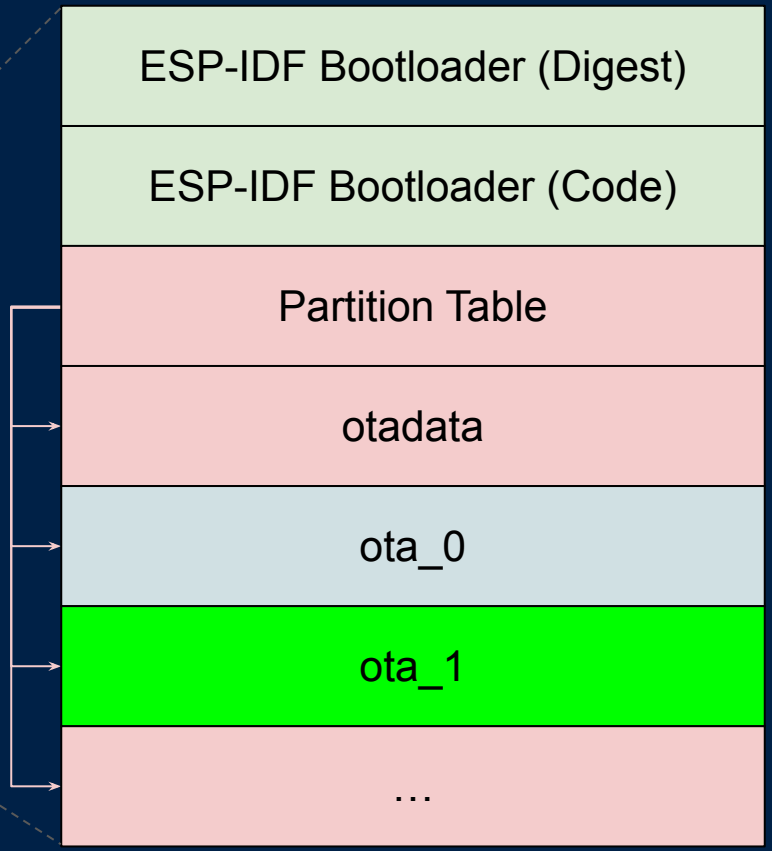
PoC


QEMU
NBDKIT
ESP32-D0WD
V3
472021
FE00PHM183
25VQ32BSIG
P21495

elttam

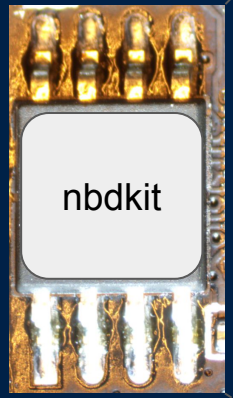
Time of check

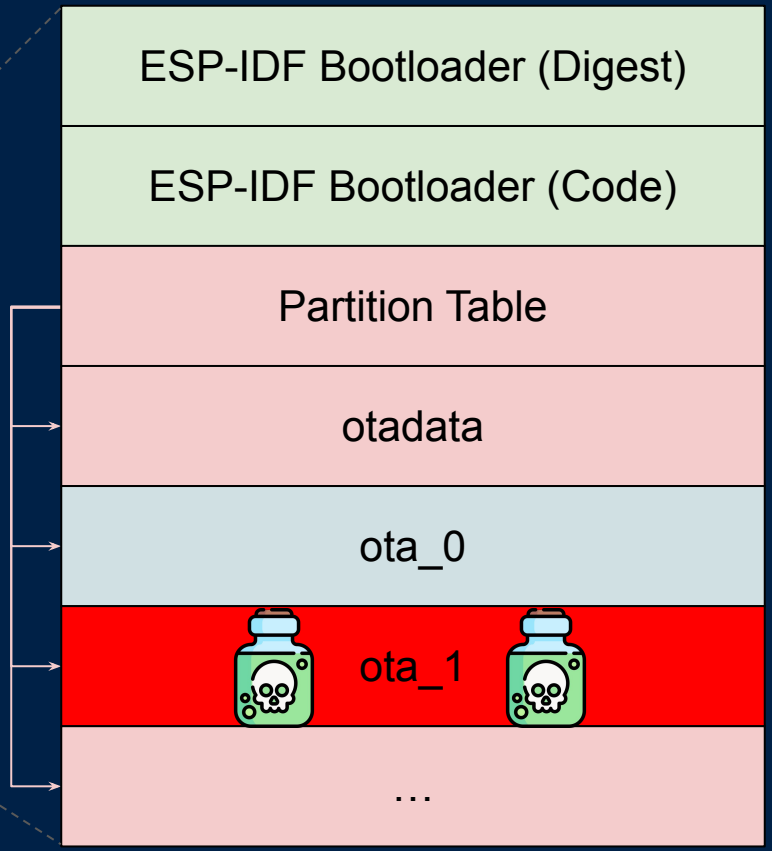




elttam

Time of use





elttam

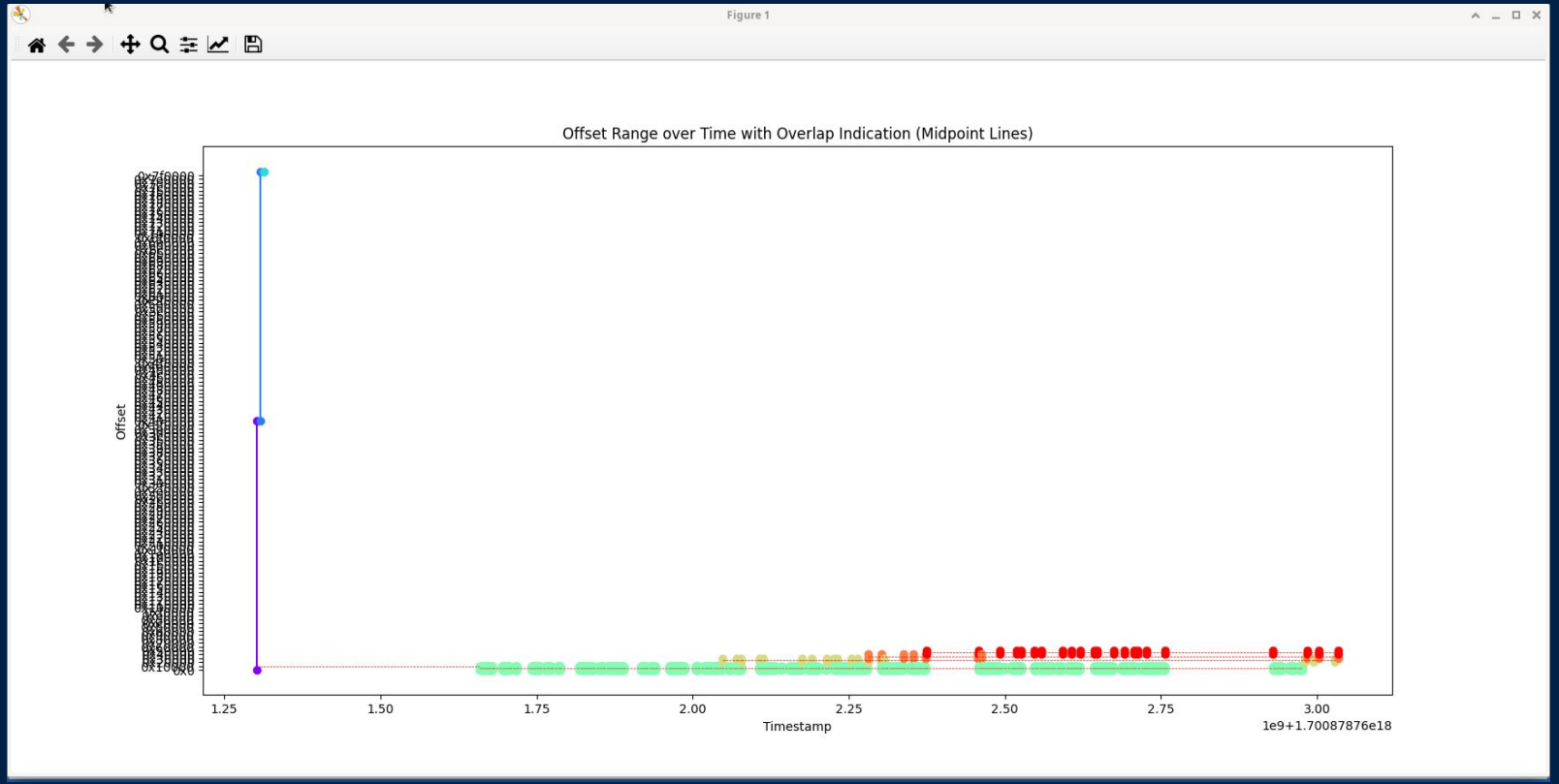
CVE-2024-28183

Shout outs josep68_!

```
I (22111) boot: Enabled a check secure version of app for anti rollback
I (22113) boot: Secure version (from eFuse) = 10
I (30827) esp_image: segment 0: paddr=00020020 vaddr=3f400020 size=0a390h ( 41872) map
I (34619) esp_image: segment 1: paddr=0002a3b8 vaddr=3ffb0000 size=02240h (  8768) load
I (38353) esp_image: segment 2: paddr=0002c600 vaddr=40080000 size=03a18h ( 14872) load
I (42058) esp_image: segment 3: paddr=00030020 vaddr=400d0020 size=149b0h ( 84400) map
I (45762) esp_image: segment 4: paddr=000449d8 vaddr=40083a18 size=08d8ch ( 36236) load
I (49457) esp_image: segment 5: paddr=0004d76c vaddr=00000000 size=02814h ( 10260)
I (53186) esp_image: Verifying image signature...
I (4657) boot: Loaded app from partition at offset 0x20000
I (4658) secure_boot_v1: bootloader secure boot is already enabled. No need to generate digest.
I (4661) boot: Checking secure boot...
I (4663) secure_boot_v1: bootloader secure boot is already enabled, continuing..
I (4665) boot: Disabling RNG early entropy source...
I (5968) cpu_start: Multicore app
I (6346) cpu_start: Pro cpu start user code
I (6357) cpu_start: cpu freq: 160000000 Hz
I (6359) cpu_start: Application information:
I (6360) cpu_start: Project name:     hello_world
I (6368) cpu_start: App version:      a6b2033-dirty
I (6372) cpu_start: Secure version:   0
I (6372) cpu_start: Compile time:     Dec 15 2023 15:04:09
I (6375) cpu_start: ELF file SHA256:  3eff680d7...
I (6377) cpu_start: ESP-IDF:          v5.3-dev-892-g692c1fcc52
[0] 0:pbdkit- 1:[tmux]*                                    "28302ff51740"
```

elttam

toctou tooling



elttam

```
static esp_err_t process_segment_data(...)
{
    const uint32_t *data = (const uint32_t *)bootloader_mmap(data_addr, data_len);
    uint32_t *dest = (uint32_t *)load_addr;
    const uint32_t *src = data;
    for (size_t i = 0; i < data_len; i += 4) {
        int w_i = i / 4; // Word index
        uint32_t w = src[w_i];
        if (checksum != NULL) {
            *checksum ^= w;
        }
        if (do_load) {
            dest[w_i] = w ^ ((w_i & 1) ? ram_obfs_value[0] : ram_obfs_value[1]);
        }
        const size_t SHA_CHUNK = 1024;
        if (sha_handle != NULL && i % SHA_CHUNK == 0) {
            bootloader_sha256_data(sha_handle, &src[w_i],
                                   MIN(SHA_CHUNK, data_len - i));
```

Cache says no, maybe other architectures?

“SPI0 is entirely dedicated to the flash cache the ESP32 uses to map the SPI flash device it is connected to into memory.”

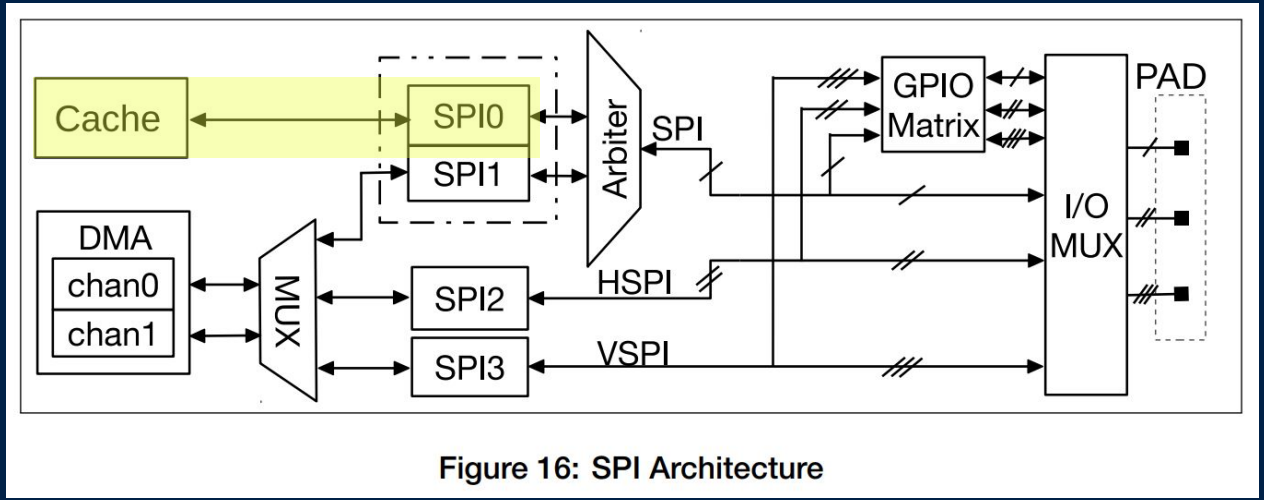


Figure 16: SPI Architecture

elttam

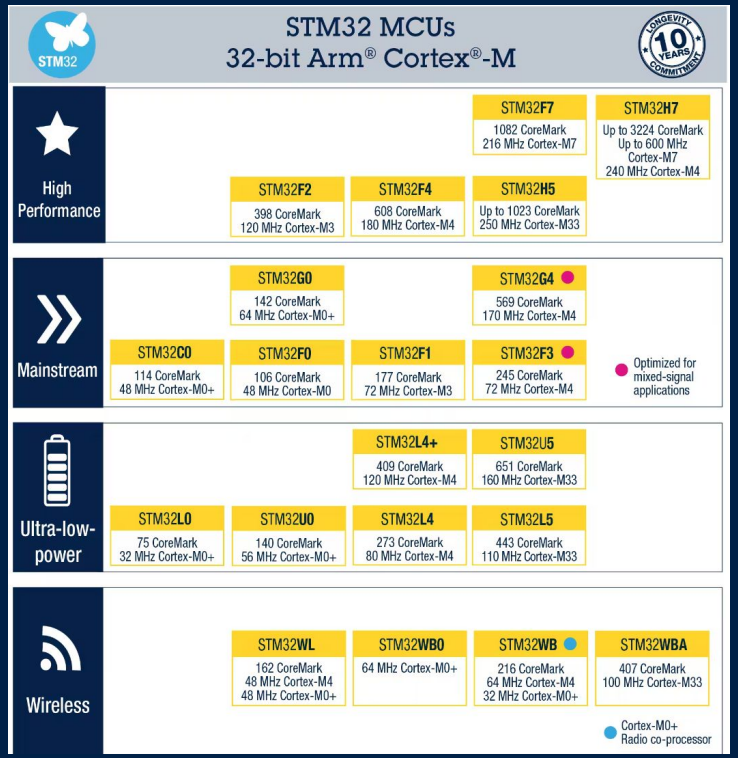
tldr;

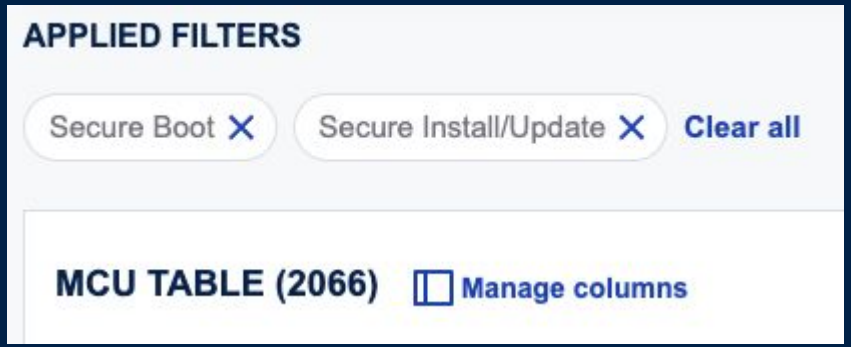
- TOCTOU seems pretty juicy attack surface for ESP32.
- First pass of the rest of the boot flow seems good.
- Still see potential for esoteric issues relating to multiple architecture support (LX6, LX7, RISC-V, etc.).
- Great choice of target to begin learning about MCU secure boot!

STMicroelectronics STM32

elttam

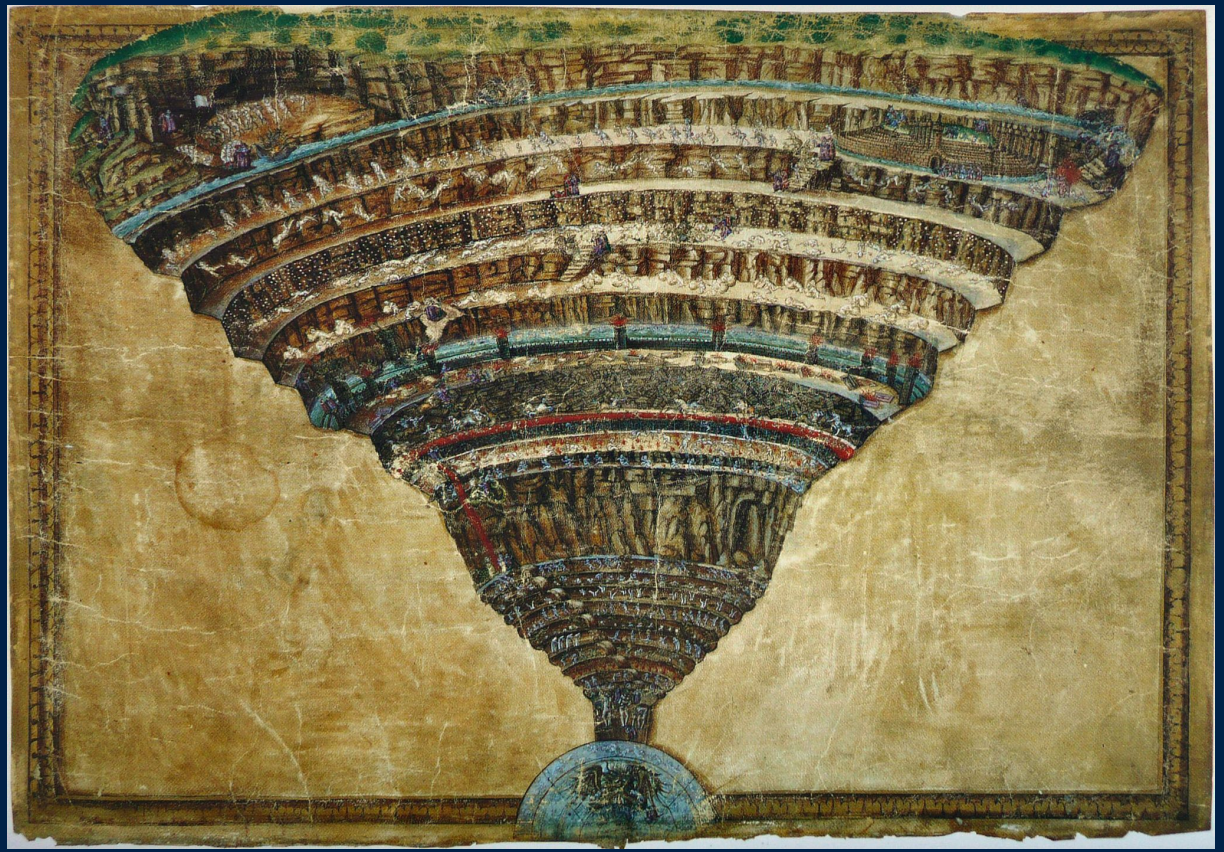
STM32





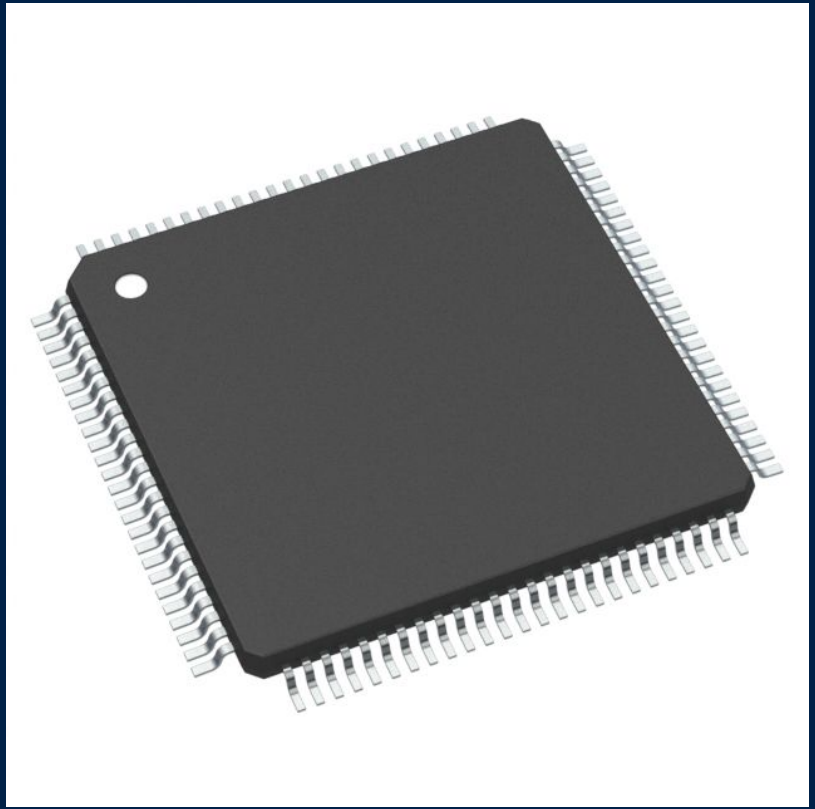
elttam

Documentation review



STM32L4S5

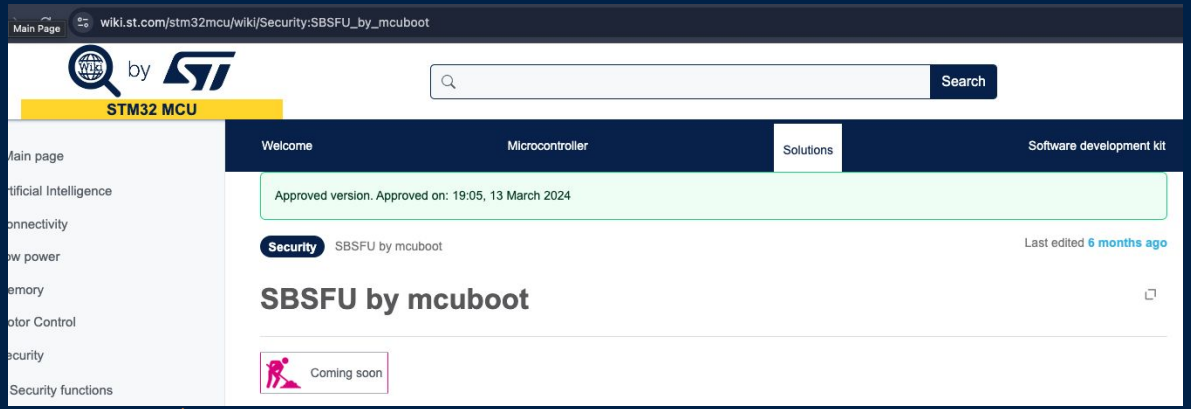
- The MCU we used during our investigations
  - Specifically the B-**L4S5I**-IOT01A discovery kit
- Arm® Cortex®-M4
- 2-Mbyte Flash
- 640 Kbytes of SRAM
- All the communications interfaces
  - USB OTG, IC2, USART, SPI, CAN, Bluetooth v4.1, NFC, Wifi, etc.



elttam

STM32 Secure Boot Vendor Recommendations

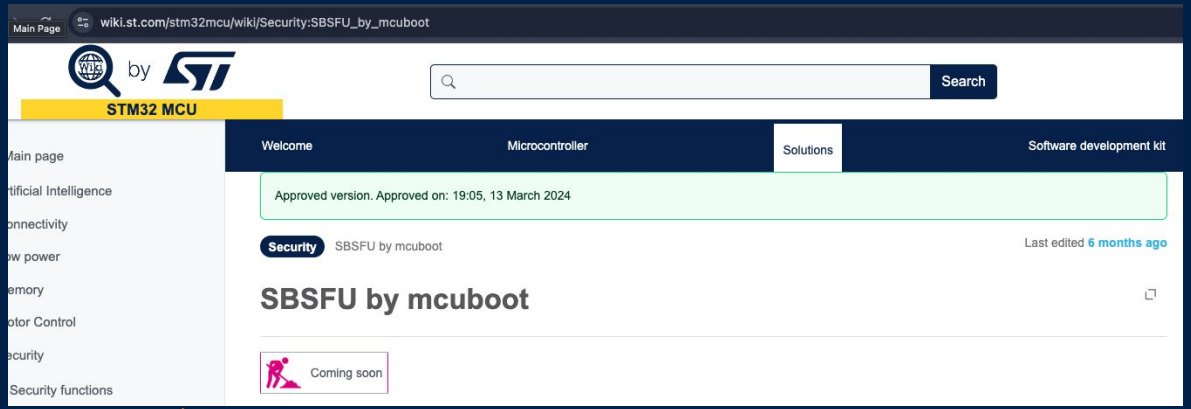
- X-CUBE-SBSFU
  - “Legacy”
- SBSFU by mcuboot
  - Seems to be the new standard, not available yet though
- STiRoT



elttam

STM32 Secure Boot Vendor Recommendations

- X-CUBE-SBSFU
  - Primary first-party secure boot implementation at the time
- SBSFU by mcuboot
  - Wasn’t available during our research
- STiRoT



elttam

X-CUBE-SBSFU

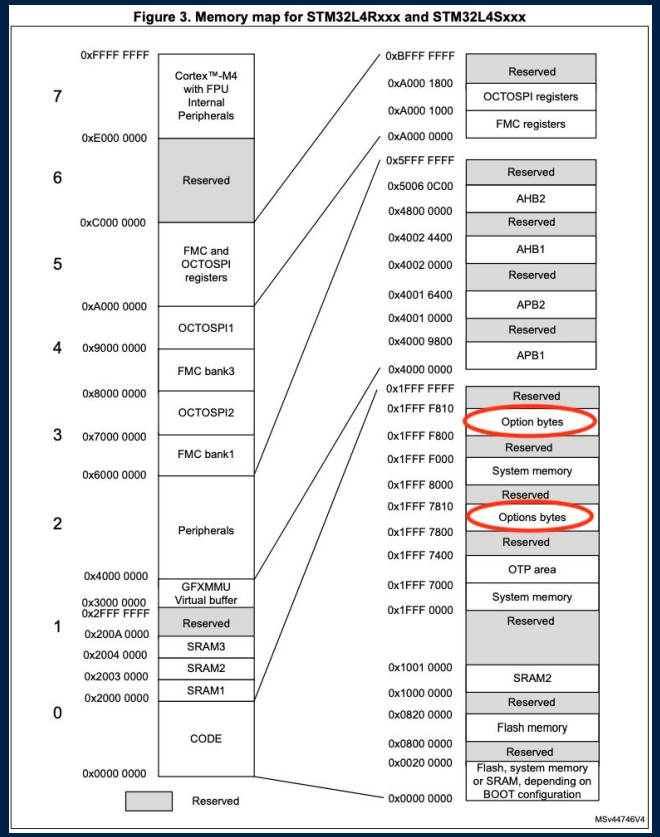
Strategies

- Single code entry: SBSFU
- Immutable SBSFU code and secrets (RDP)
- Protected and isolated enclave for secrets (KMS or STSAFE)
- Limited SBSFU attack surface
- Disable JTAG (RDP)
- System monitoring

Common element?

STM32 Security Features

- Relies on ARM
- Readout Protection (RDP)
  - Level 0-2
- Memory Protection Unit (MPU)
- Write Protection (WRP)
- Proprietary code readout protection (PCROP)
- Firewall



elttam

RDP

RDP level 0

RDP level 1

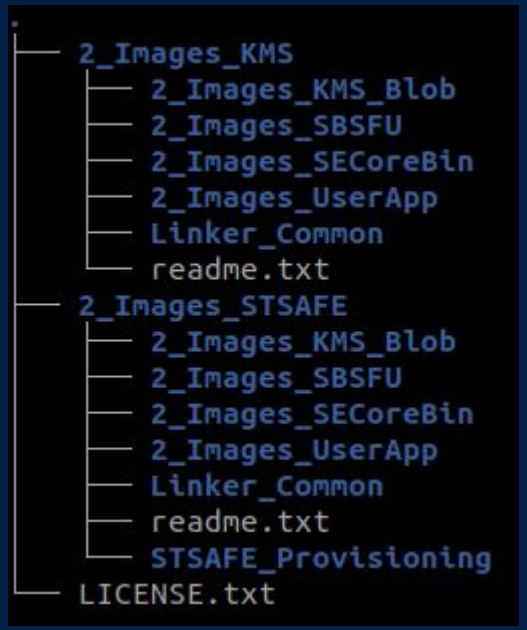
RDP level 2

nRDP	RDP	Protection
0x55	0xAA	RDP Level 0
Any other combination		RDP Level 1
0x33	0xCC	RDP Level 2

Readout Protection Level Configuration

elttam

SBSFU - STSAFE



Secure boot high level steps

1. Board init
   a. HAL, peripherals, etc.
2. STSAFE init
   a. STSAFE HAL driver init
3. Secure Engine Startup
4. Check/Apply Security Protections
   a. Static - RDP, WRP, PCRoP
   b. Runtime - FW, DMA, IWDG, DAP, Anti-tamper, Clock monitor, Temperature monitor
5. SBSFU state machine init and start

elttam

STSAFE Init

```c
int32_t StSafeA_HostKeys_Init()
{
#if (USE_PRE_LOADED_HOST_KEYS)
  /* This is just a very easy example to retrieve keys pre-loaded at the end of the MCU Flash
     and load them into the SRAM. Host MAC and Chiper Keys are previously pre-stored at the end
     of the MCU flash (e.g. by the SDK Pairing Application example) .
     It's up to the user to protect the MAC and Chiper keys and to find the proper
     and most secure way to retrieve them when needed or to securely keep them into
     a protected volatime memory during the application life */

  /* Host MAC Key */
  uint32_t host_mac_key_addr = FLASH_BASE + FLASH_SIZE - 2U * (STSAFEA_HOST_KEY_LENGTH);

  /* Host Cipher Key */
  uint32_t host_cipher_key_addr = FLASH_BASE + FLASH_SIZE - (STSAFEA_HOST_KEY_LENGTH);

  /* Set and keep the keys that will be used during the Crypto / MAC operations */
  (void)memcpy(aHostMacKey, (uint8_t *)host_mac_key_addr,    STSAFEA_HOST_KEY_LENGTH);
  (void)memcpy(aHostCipherKey, (uint8_t *)host_cipher_key_addr, STSAFEA_HOST_KEY_LENGTH);
#endif /* USE_PRE_LOADED_HOST_KEYS */

  return 0;
}
```

elttam

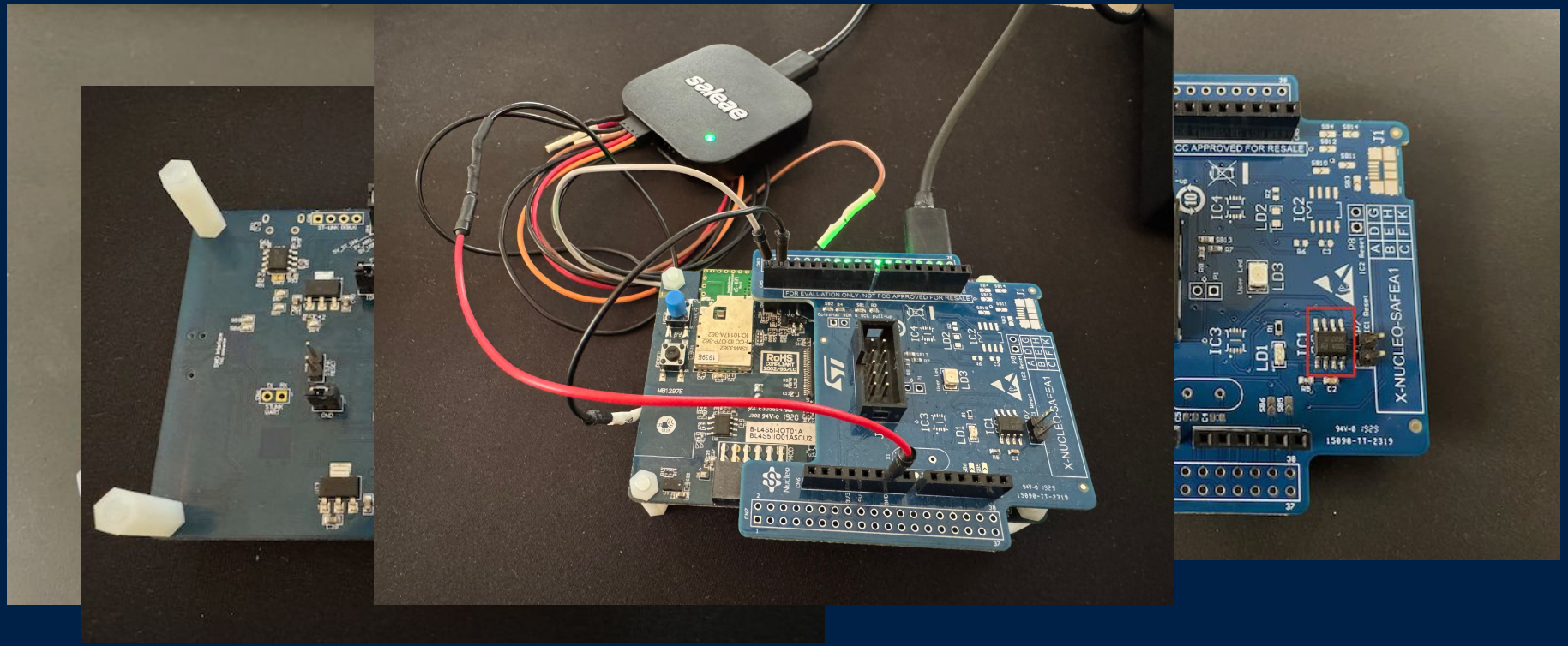
SBSFU State Machine

```
  static void (* fnStateMachineTable[])(void) = {SFU_BOOT_SM_CheckStatusOnReset,
#if (SECBOOT_LOADER == SECBOOT_USE_LOCAL_LOADER) || (SECBOOT_LOADER == SECBOOT_USE_STANDALONE_LOADER)
                                                 SFU_BOOT_SM_CheckNewFwToDownload,
                                                 SFU_BOOT_SM_DownloadNewUserFw,
#endif /* (SECBOOT_LOADER == SECBOOT_USE_LOCAL_LOADER) || (SECBOOT_LOADER == SECBOOT_USE_STANDALONE_LOADER) */
#ifdef KMS_ENABLED
                                                 SFU_BOOT_SM_CheckKMSBlobToInstall,
                                                 SFU_BOOT_SM_InstallKMSBlob,
#endif /* KMS_ENABLED */
                                                 SFU_BOOT_SM_CheckUserFwStatus,
                                                 SFU_BOOT_SM_InstallNewUserFw,
                                                 SFU_BOOT_SM_VerifyUserFwSignature,
```

```
#define SFU_SET_SM_IF_CURR_STATE(Status, SM_STATE_OK, SM_STATE_FAILURE) \
  do{                                                                   \
    m_StateMachineContext.PrevState = m_StateMachineContext.CurrState;  \
    if (Status == SFU_SUCCESS){                                         \
      m_StateMachineContext.CurrState = SM_STATE_OK;                    \
    }                                                                   \
    else {                                                              \
      m_StateMachineContext.CurrState = SM_STATE_FAILURE;               \
    }                                                                   \
  }while(0) /*!< Set a State Machine state according to the 'Status' value*/
```

elttam

B-L4S5I-IOT01A and X-NUCLEO-SAFEA1



elttam

STSAFE-A110

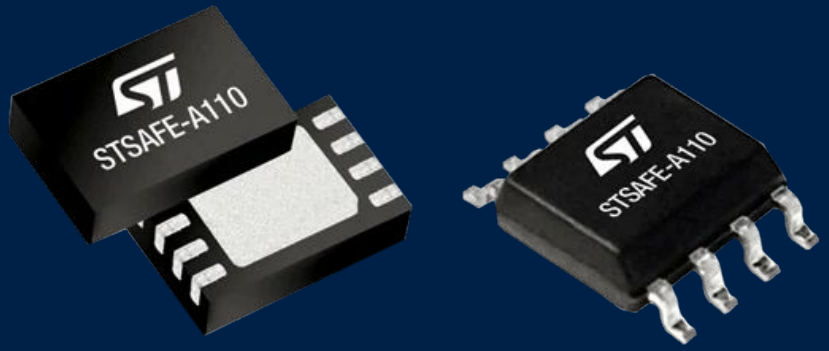
Pretty much black-box hardware specs

- 6 Kbytes of configurable non-volatile memory

External security enclave

I2C communication

- Attack surface!



elttam

STSAFE Commands

Init/Config

StSafeA_Init()

General Purpose

StSafeA_Echo()
StSafeA_Reset()
StSafeA_GenerateRandom()
StSafeA_Hibernate()

Data Partition

StSafeA_DataPartitionQuery()
StSafeA_Decrement()
StSafeA_Read()
StSafeA_Update()

PKI

StSafeA_GenerateKeyPair()
StSafeA_GenerateSignature()
StSafeA_VerifyMessageSignature()
StSafeA_EstablishKey()

Admin

StSafeA_ProductDataQuery()
StSafeA_I2cParameterQuery()
StSafeA_HostKeySlotQuery()
StSafeA_PutAttribute()
StSafeA_DeletePassword()
StSafeA_VerifyPassword()
StSafeA_RawCommand()

Local Envelope

StSafeA_LocalEnvelopeKeySlotQuery()
StSafeA_GenerateLocalEnvelopeKey()
StSafeA_WrapLocalEnvelope()
StSafeA_UnwrapLocalEnvelope()

elttam

CVE-2023-50096: STSAFE-AXX Buffer Overflow

- Reported by Zoltan Madarassy, elttam in 2023
- Allows code execution on the main MCU via the STSAFE-AXX I2C bus

elttam

```
int8_t StSafeA_ReceiveBytes(StSafeA_TLVBuffer_t *pOutBuffer)
{
resp_length = (pOutBuffer->LV).Length;
[...]
  if (resp_length + 1 < STSAFEA_HEADER_LENGTH) {
    if ((pOutBuffer->LV).Data != (uint8_t *)0x0) {
      for (; (status_code != STSAFEA_BUS_OK && (loop < (STSAFEA_I2C_POLLING_MAX /
STSAFEA_I2C_POLLING_STEP))); loop = loop + STSAFEA_I2C_POLLING_STEP) {
        statuscode = (*HwCtx.BusRecv)((uint16_t)((HwCtx.DevAddr & 0x7fff) << 1),
                                      (pOutBuffer->LV).Data,resp_length + 3);
        status_code = (int8_t)statuscode;
        if (status_code == STSAFEA_BUS_NACK) {
          (*HwCtx.TimeDelay)(STSAFEA_I2C_POLLING_STEP);
        }
      }
```

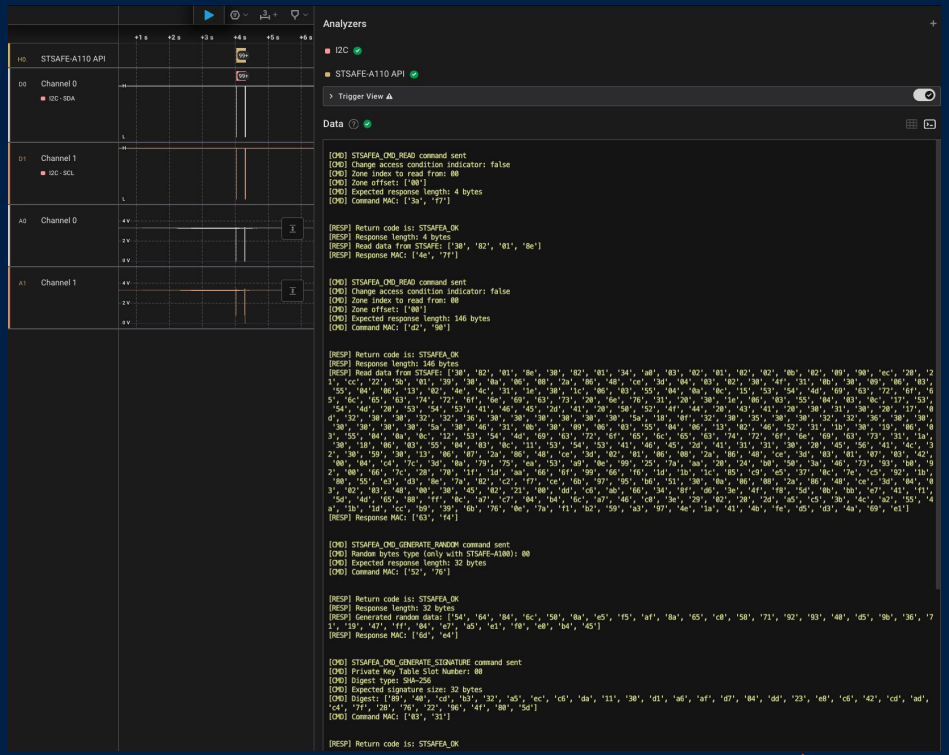
```
pOutBuffer->Header = *(pOutBuffer->LV).Data;
(pOutBuffer->LV).Length =
      (ushort)(pOutBuffer->LV).Data[2] + (ushort)(pOutBuffer->LV).Data[1] * 0x100;
memcpy((pOutBuffer->LV).Data,(pOutBuffer->LV).Data + 3,(uint)resp_length);
```

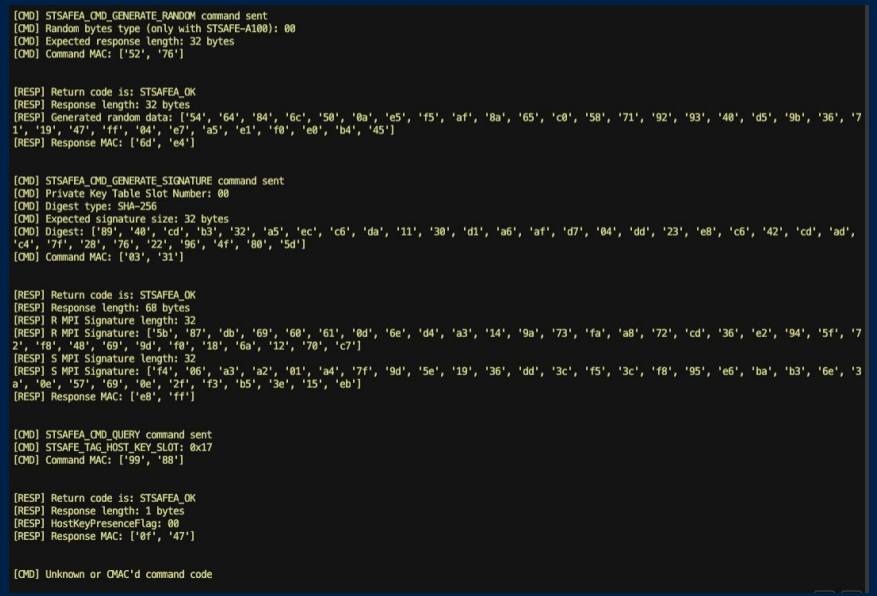
elttam

```
      if ((resp_length < (pOutBuffer->LV).Length) && (status_code == '\0')) {
        status_code = -1;
        for (loop = 1; (status_code != STSAFEA_BUS_OK && (loop < (STSAFEA_I2C_POLLING_MAX /
STSAFEA_I2C_POLLING_STEP))); loop = loop + STSAFEA_I2C_POLLING_STEP) {
          statuscode = (*HwCtx.BusRecv)((uint16_t)((HwCtx.DevAddr & 0x7fff) << 1),
                                        (pOutBuffer->LV).Data,(pOutBuffer->LV).Length + 3);
          status_code = (int8_t)statuscode;
          if (status_code == STSAFEA_BUS_NACK) {
            (*HwCtx.TimeDelay)(STSAFEA_I2C_POLLING_STEP);
          }
        }
```

elttam

Demo: STSAFE-A1XX Protocol Analyser Logic Plugin





https://github.com/elttam/stsafe-a110-i2c-decoder

elttam

Outroduction

Where else be dragons?

- Execute in place (XiP)
- Ultra Low Power (ULP) Coprocessors and alternate boot flows
- DMA
- Padding
- Block alignment
- UART
- WiFi and BLE stack
- BootROM implementation
- Fault injection

Future work

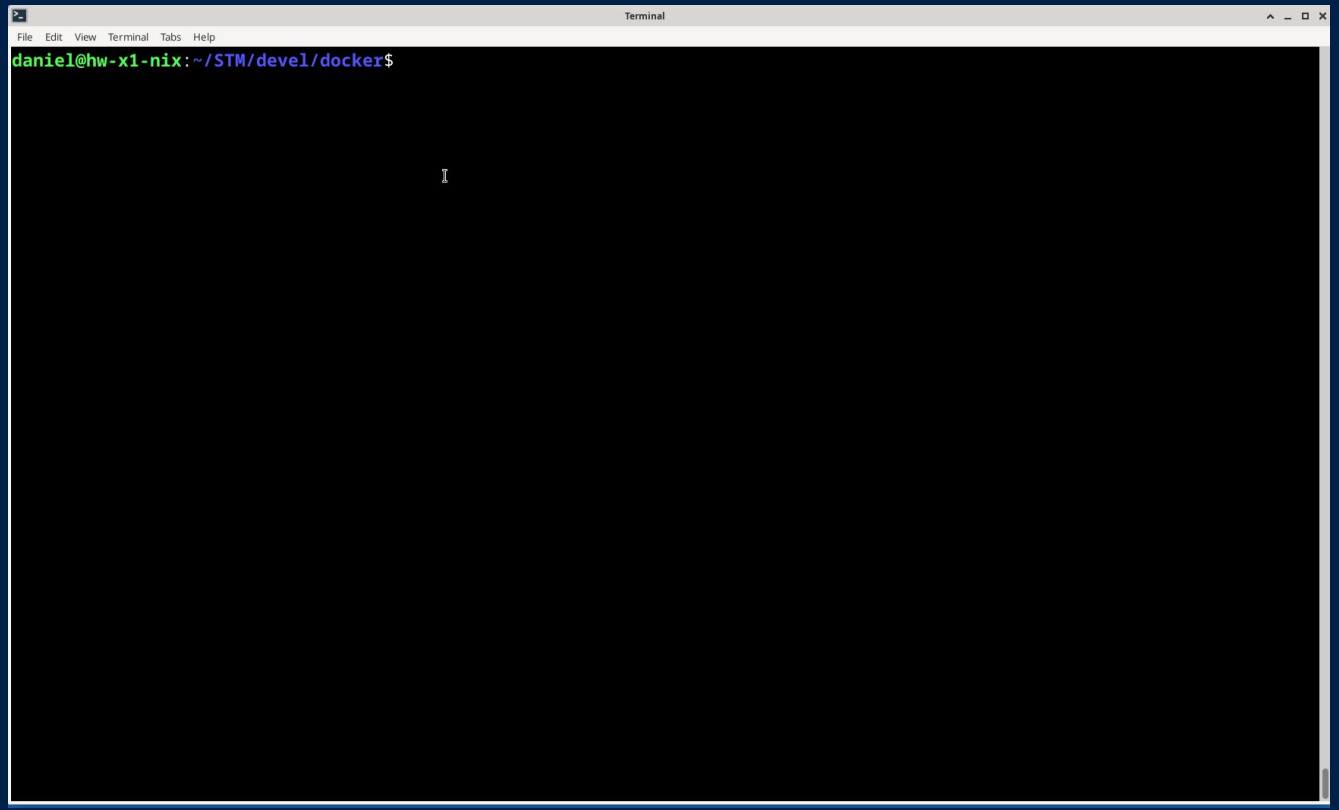
More STM32 auditing

Look at more vendors

Iterative tool improvement

Collab

https://github.com/elttam/boot-security-in-the-mcu

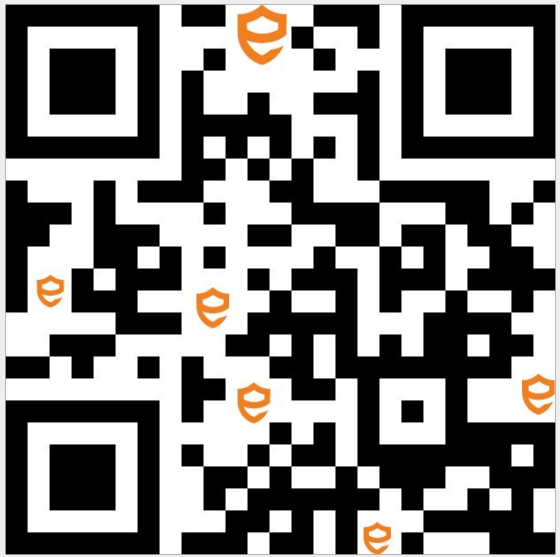


Prior Work & Thanks

- LimitedResults
- Yashin Mehaboobe
- Santiago Cordoba Pellicer
- Travis Goodspeed, author of Microcontroller Exploits published by NoStarchPress
- Karim M. Abdellatif, Olivier Hériveaux, Adrian Thillard, Ledger
- Espressif (fast and detailed responses!)
- STMicroelectronics (responses!)

Thank you

Any questions?



elttam