
# Qualcomm BootROM

## A journey through Sahara

# AGENDA

I. Introduction

II. Sahara

III. From AP to CP

IV. CP and more

V. Conclusions


# I. Introduction


# Alexander Kozlov

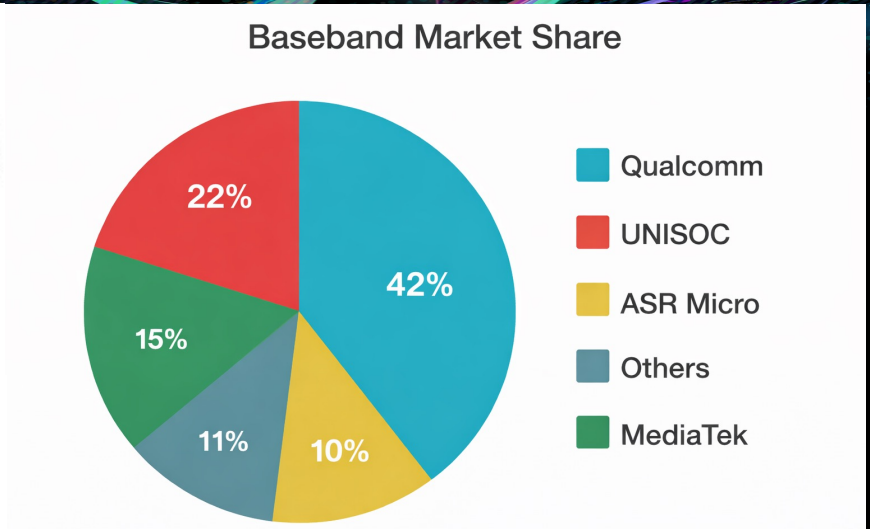
Principal Security Researcher, Kaspersky

# Sergey Anufrienko

Group Manager, Kaspersky

# MDM9207: Why and where?

- Smart Utilities
- Retail & Security
- Logistics
- Networking




# MDM9207: Why and where?

# MDM9207: Security assumptions

- ARM Boot ROM only executes verified code*
- TrustZone restricts access to CP (MSS) memory
- CP Boot ROM cannot be read
- CP Boot ROM only executes verified code*

MSS - Modem Sub-System
CP - Communication Processor

* on fused/production devices

# Previous work

CENSUS
Cybersecurity Engineering

Attacking Hexagon: Security Analysis of Qualcomm's aDSP

Dimitrios Tatsis (@dtouch3d)

tatsisd@census-labs.com

Recon Montreal 2019

Exploiting Qualcomm WLAN and Modem Over the Air

Xiling Gong of Tencent Blade Team

In-Depth Analyzing and Fuzzing for Qualcomm Hexagon Processor

BLADE Tencent Blade

Xiling Gong of
Bo Zhang of Tencent

- This presentation belongs t
- Xiling Gong is on be

THE ROAD TO QUALCOMM TRUSTZONE APPS FUZZING

November 14, 2019

Research By: Slava Makkaveev

Unveiling the Mysteries of Hexagon QDSP6 JTAG

A Journey into Advanced Theoretical Reverse Engineering

Alisa Esage
Zero Day Engineering Research & Training
Black Hat Asia 2025, Singapore

Analysis of Qualcomm Secure Boot Chains

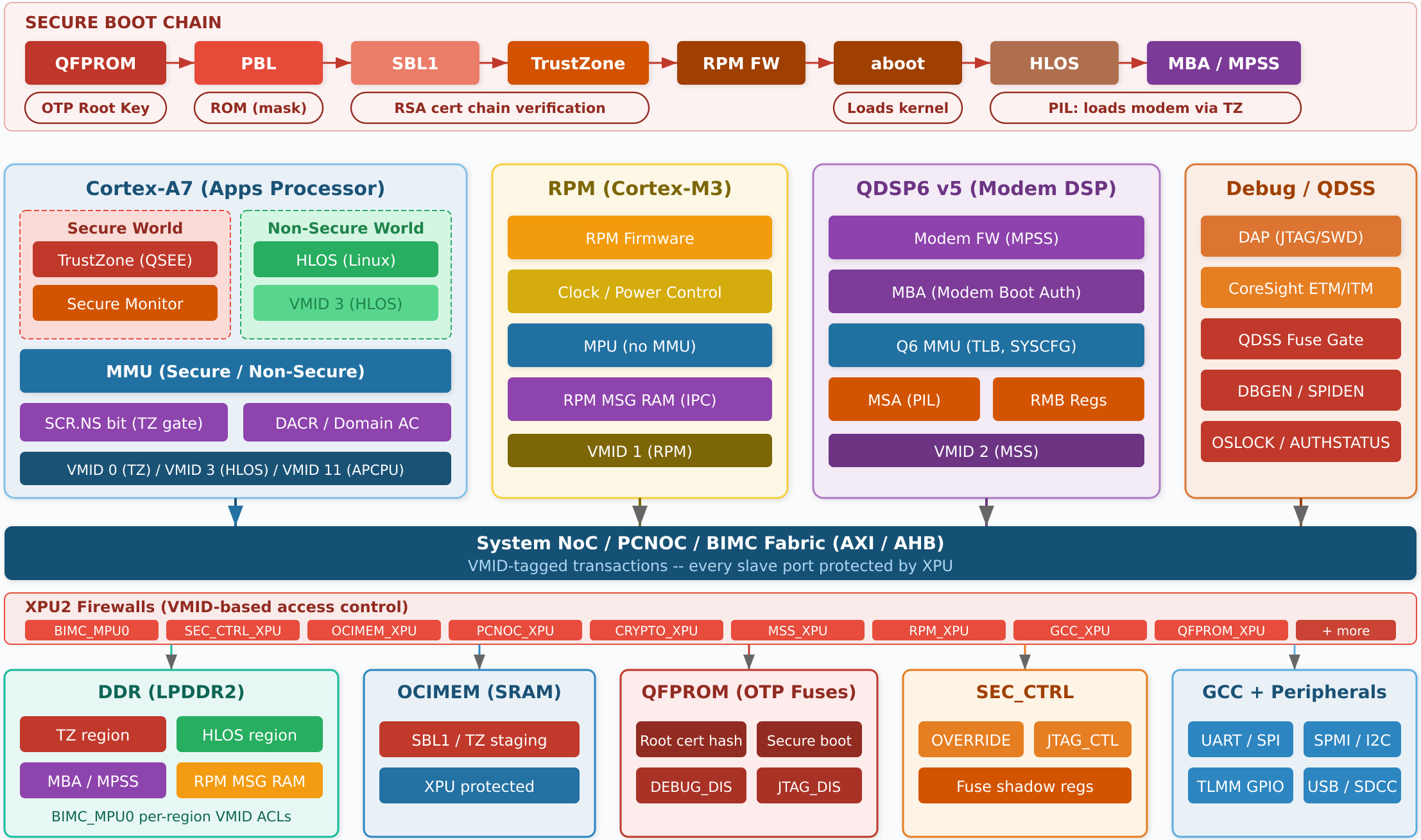
Posted Thu 24 October 2019
Author Elouan Appere
Category Reverse-Engineering
Tags secure boot, reverse-en

Exploiting Qualcomm EDL Programmers (1): Gaining Access & PBL Internals

By Roee Hay (@roeehay) & Noam Hadad
*January 22, 2018 **

QPSIIR-909 ALEPH-2017029 CVE-2017-13174 CVE-2017-5947

# MDM9607 / MSM SoC Security Architecture




# II. Sahara

# ARM PBL

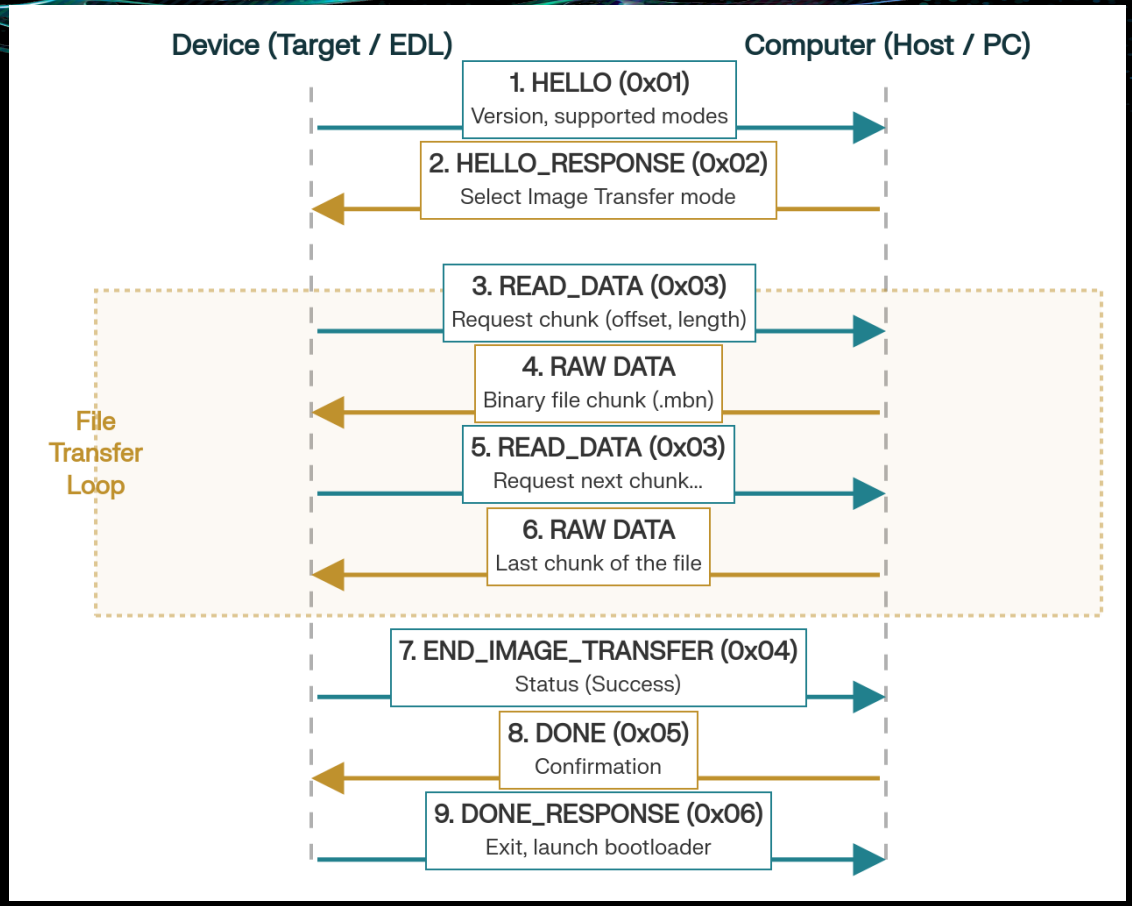
- Use OEM-signed programmer
- Just one well-known overflow vuln
- Dump PBL
- Start address 0x10000

```
00107EC0  05 00 A0 E1 70 80 BD E8  2E 2F 61 70 70 73 2F 70  ..........apps/p
00107ED0  62 6C 5F 73 61 68 61 72  61 2E 63 00 10 40 2D E9  bl_sahara.c..@-.
00107EE0  48 40 9F E5 00 00 A0 E3  00 10 94 E5 00 00 51 E3  H@............Q.
00107EF0  0D 00 00 1A AB 0B 00 EB  12 F7 FF EB 00 00 50 E3  ..............P.
00107F00  04 00 00 1A 28 20 9F E5  28 00 8F E2 00 30 A0 E3  ....(·.....(....
00107F10  D2 10 A0 E3 DE F8 FF EB  00 00 A0 E3 E3 0B 00 EB  ................
00107F20  00 00 50 E3 01 10 A0 03  00 10 84 05 10 80 BD E8  ..P.............
00107F30  1C 35 00 08 00 81 07 00  2E 2F 61 70 70 73 2F 70  .5......./apps/p
00107F40  62 6C 5F 73 61 68 61 72  61 2E 63 00 00 02 00 E3  bl_sahara.c.....
00107F50  1E FF 2F E1 05 0C 00 EA  01 00 A0 E3 10 40 2D E9  ../............-.
00107F60  55 0C 00 EB 08 10 9F E5  00 00 A0 E3 00 00 81 E5  U...............
00107F70  10 80 BD E8 1C 35 00 08  10 40 2D E9 00 40 A0 E3  .....5.....@-.....
00107F80  8B F7 FF EB 00 00 50 E3  01 40 A0 03 04 00 A0 E1  ......P.....@.....
00107F90  10 80 BD E8 00 00 50 E3  10 40 2D E9 04 00 00 1A  ......P...@-.....
00107FA0  32 29 A0 E3 00 30 A0 E3  38 00 8F E2 35 13 00 E3  2)...0..8...5...
00107FB0  B7 F8 FF EB 40 00 9F E5  A1 E6 FF EB 00 00 50 E3  ....@.........P.
00107FC0  04 00 00 1A 34 20 9F E5  00 30 A0 E1 14 00 8F E2  ....4·.....0....
00107FD0  3B 13 00 E3 AE F8 FF EB  00 00 A0 E3 36 0C 00 EB  ;...........6...
00107FE0  00 00 A0 E3 10 80 BD E8  2E 2F 61 70 70 73 2F 70  ..........apps/p
00107FF0  62 6C 5F 73 61 68 61 72  61 2E 63 00 5C 30 00 08  bl_sahara.c.\0..
00108000  00 81 0C 00 70 40 2D E9  00 00 51 E3 00 30 A0 E1  ....p@-...Q....0..
00108010  02 50 A0 E1 00 00 A0 E3  00 00 55 13 00 40 A0 E1  .P........U..@..
00108020  0C 00 00 0A 08 00 53 E3  00 C0 A0 E3 04 20 A0 E3  ......S.........
00108030  03 F1 9F 37 31 00 00 EA  68 80 10 00 70 80 10 00  ...71...h...p...
00108040  90 80 10 00 B4 80 10 00  00 81 10 00 00 81 10 00  ................
```

PBL – Primary Boot Loader

# Sahara: what do we know

- Emergency Download Mode (EDL)
- Implemented in ARM PBL (Boot ROM)
- Device identifies itself through USB



# Sahara: what do we learn

- Not || but &&
- So what if size > 0xEFFE?
- Bypass p_addr checks
- Got arbitrary write primitive

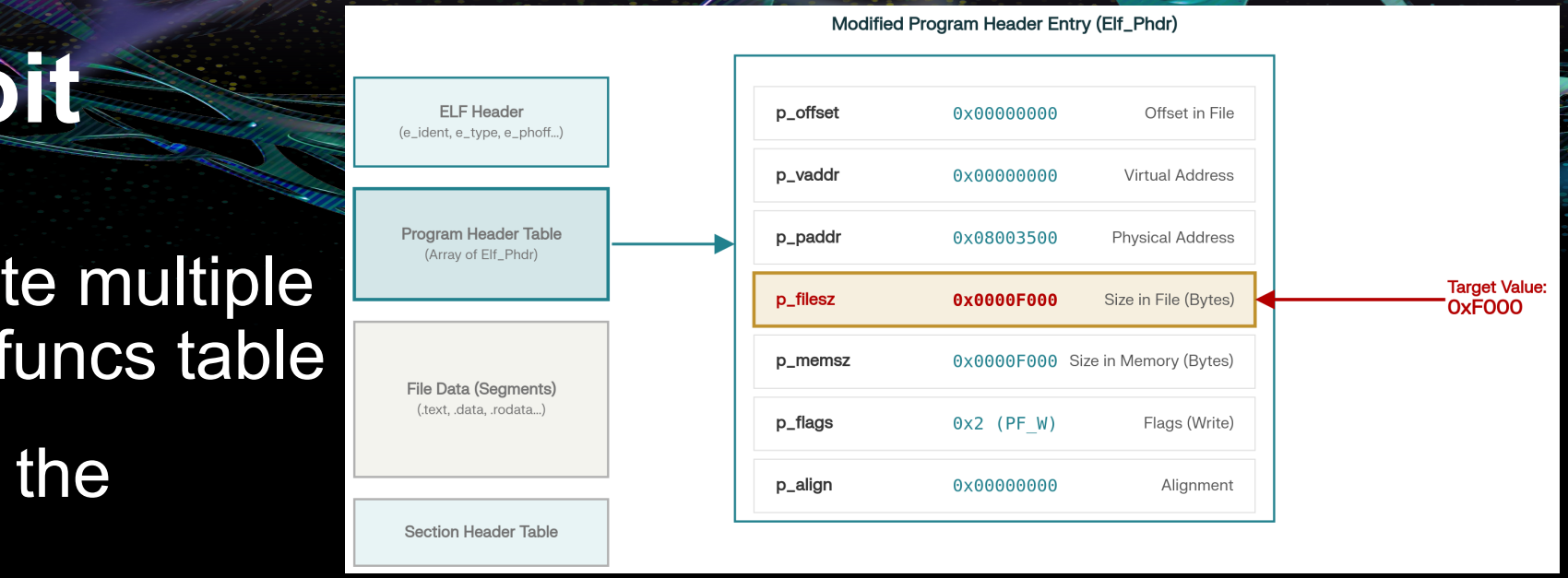
```
int __fastcall sub_104ECC(_DWORD *p_addr, unsigned int size)
{
  int v2; // r2
  int v3; // r1

  v2 = 0;
  if ( size < 0xFFDF69C3 )
  {
    v3 = size + 0x20963C;
    if ( p_addr )
    {
      if ( (unsigned int)(v3 - 0x20963D) <= 0xEFFE )
      {
        v2 = 1;
        *p_addr = 0x20963C;
      }
    }
  }
  return v2;
}
```

```
if ( !(*(int (__fastcall **)(int *, unsigned int))(dword_80034EC + 0x34))(off_80034C8 + 0xF, v9)// sub_1082BC // sub_104ECC
  && !off_80034C8[0xF] )
{
  return 38;
}
if ( !p_filesz || (v5 = SAHARA_read_data_from_USB_BULK(p_offset, p_filesz, v9, off_80034C8[0xF])) == 0 )
{
  v10 = (*(int (__fastcall **)(int *))(dword_8003518 + 0x10))(off_80034C8);// sub_107E80
  if ( v10 )
    p_filesz = 0;
  else
    v5 = 0x21;
  if ( !v10 )
    return v5;
```

# Sahara: exploit

1. Gain ability to write multiple times via rewrite funcs table
2. Write out code to the 0x215000
3. Rewrite 0x1940000 with OUR addr
4. Write ZERO to reboot
5. Profit



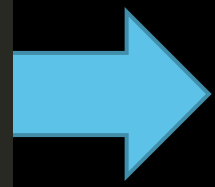

```
  mcr(15, 0, a1, 13, 0, 0);
__mcr(15, 0, (unsigned int)&unk_200000, 2, 0, 0);
v1 = sub_1003C8();
__mcr(15, 0, sub_1003BC(v1), 8, 7, 0);
__mcr(15, 0, 0, 2, 0, 2);
__mcr(15, 0, (unsigned int)&unk_200000, 2, 0, 0);
v2 = __mrc(15, 0, 1, 0, 0);
```

```
00200058                DCD 0x1600C16
0020005C                DCD 0x1700C16
00200060                DCD 0x1800C16
00200064                DCD 0x1900C16
00200068                DCD 0x1A00C16
```

| | | |
|---|---|---|
| **Base Address** | `0x190xxxx` | Covers **0x1900000** to **0x19FFFFF** (1 MB range). |
| **Target Reach** | `0x1940000` | **Accessible** (It is 256 KB offset from the base). |
| **Permissions** | `AP[2:0] = 011` | **Full Access** (Read/Write for both Kernel and User). |
| **Execution** | `XN = 1` | **Blocked** (Execute-Never; code cannot run here). |

# Sahara: Code Execution

```
if *(DWORD)(0x1940000) & 1 == 1
{
    v2 = (int) (*(DWORD)(0x1940000) & 0xFFFFFFFD);
    return v2();
}
```



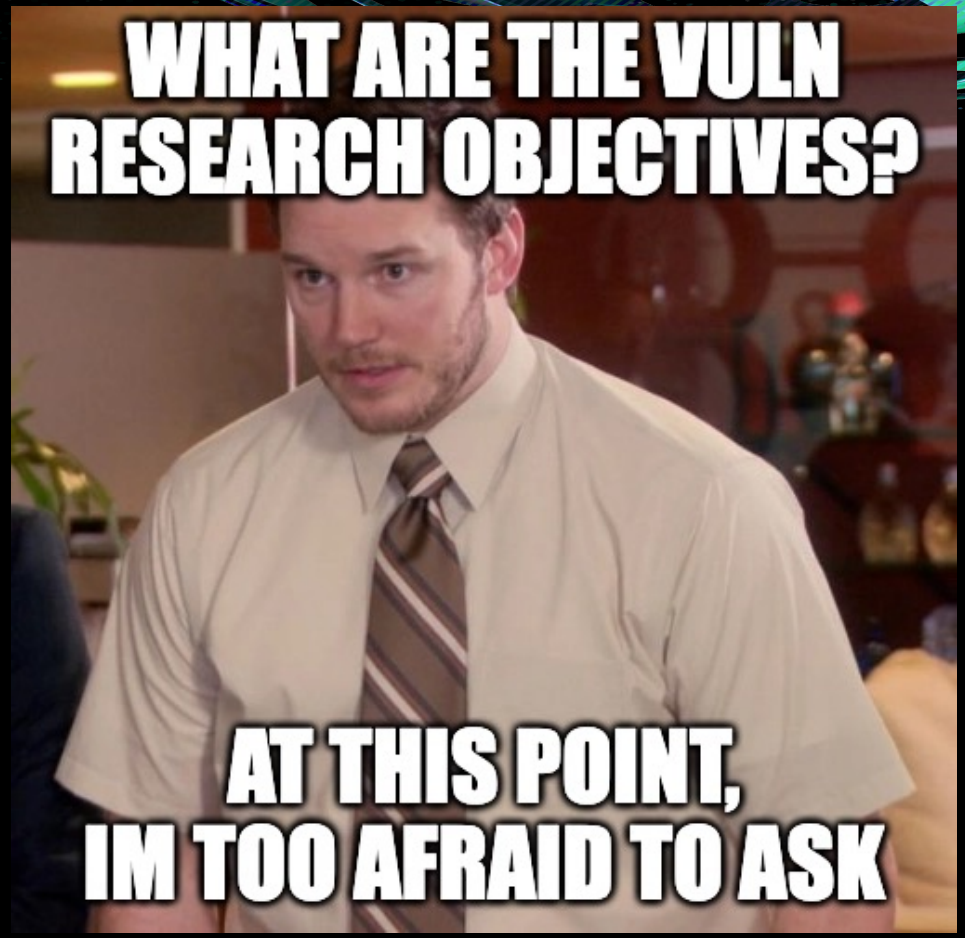
BootROM Code Execution

# Security assumptions (revisited)

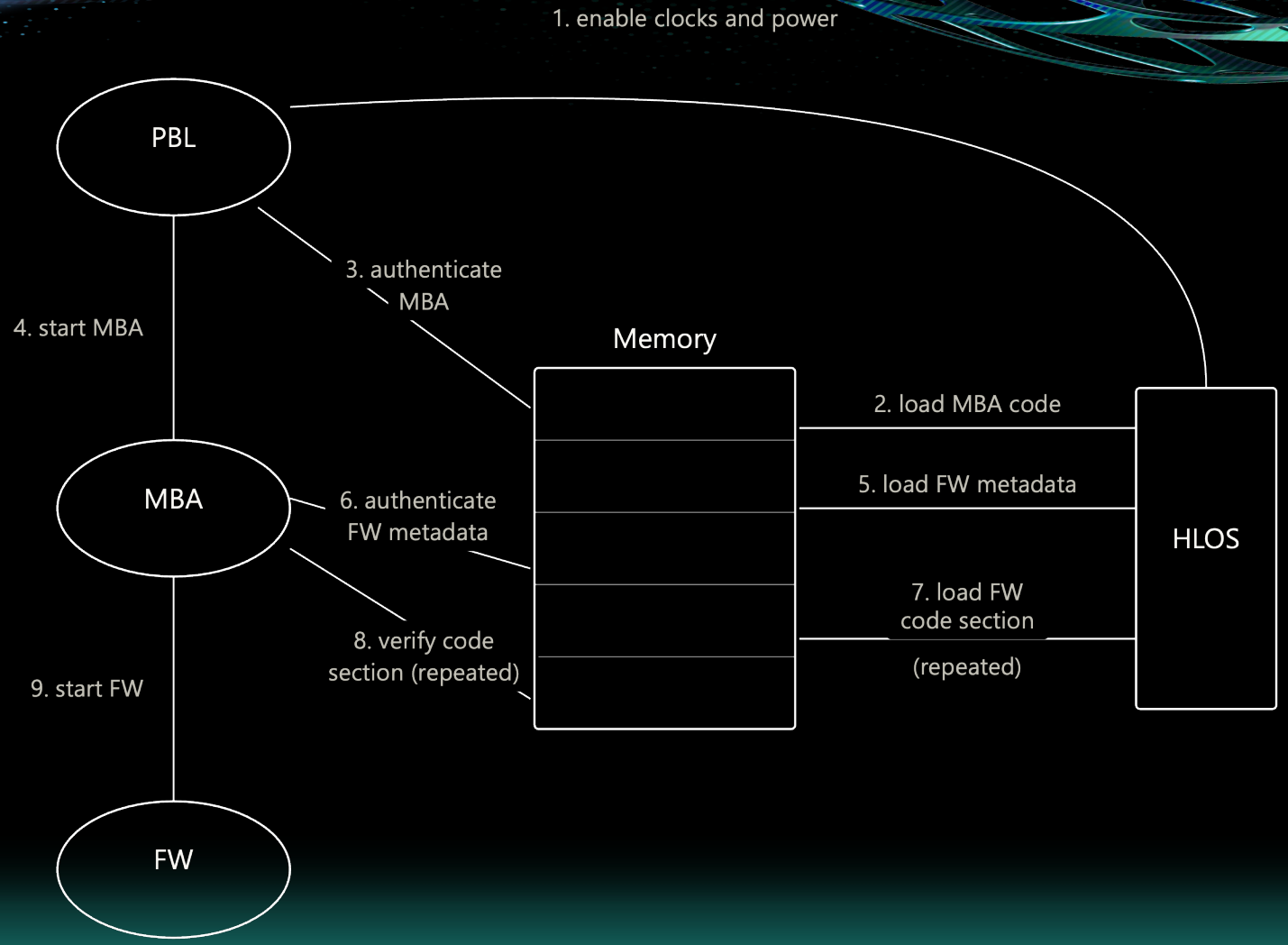
- ARM ROM only executes verified code
- CP ROM cannot be read
- CP ROM only executes verified code


# III. From AP to CP

# What’s next?


WHAT ARE THE VULN RESEARCH OBJECTIVES?
AT THIS POINT, IM TOO AFRAID TO ASK

# PIL (Peripheral Image Loading)



# TrustZone: what and why

- Runs on ARM in monitor mode
- Sets up XPUs to protect subsystem memory
- Locks down critical memory areas
- Enforces debug policy
- Manages QFPROM fuses
- Provides other services via SMC calls

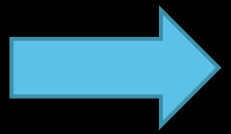
XPU – eXternal Protection Unit
SMC – Secure Monitor Call

# TrustZone: XPU disable

```
int xpu2_configure(int param_1,int param_2)

{
  int iVar1;
  int iVar2;

  iVar2 = 1;
  iVar1 = xpu2_validate_id();
  if (iVar1 != 0) {
    iVar1 = xpu2_is_configured(param_1);
    if (iVar1 == 0) {
      iVar2 = 2;
    }
    else if (*(int *)(*(int *)(&xpu_tbl1 + param_1 * 8) + 100) == 0) {
      iVar2 = 3;
    }
    else {
      dsb();
      iVar2 = (**(code **)(*(int *)(&xpu_tbl1 + param_1 * 8) + 100))
                        (*(undefined4 *)(&xpu_tbl2 + param_1 * 8),param_2);
      dsb();
    }
  }
  return iVar2;
}
```



```
undefined4 xpu2_configure(void)

{
  return 0;
}
```

# TrustZone: XPU disable

```
/ # dd if=/dev/modem_mem bs=1 skip=$((0x0)) count=$((0x200)) | hexdump -C
00000000  00 c0 00 78 06 40 00 67  00 c5 00 78 12 c0 00 67  |...x.@.g...x...g|
00000010  00 c0 00 a2 00 d0 c0 56  02 c0 c0 57 02 c2 00 58  |.......V...W...X|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 1a 00 00 00  |................|
00000040  00 00 00 00 43 4f 4e 46  49 47 5f 41 44 56 41 4e  |....CONFIG_ADVAN|
00000050  43 45 44 5f 4d 45 4d 4f  52 59 0a 0a 51 55 52 54  |CED_MEMORY..QURT|
00000060  4b 5f 70 68 79 73 5f 69  73 6c 61 6e 64 0a 51 55  |K_phys_island.QU|
00000070  52 54 4b 5f 71 75 69 63  6b 5f 6c 6f 61 64 5f 4d  |RTK_quick_load_M|
00000080  4d 55 0a 51 55 52 54 4b  5f 74 6c 62 5f 64 75 6d  |MU.QURTK_tlb_dum|
00000090  70 0a 51 55 52 54 4b 5f  74 6c 62 5f 64 75 6d 70  |p.QURTK_tlb_dump|
000000a0  5f 72 65 6c 6f 63 73 0a  51 55 52 54 4b 5f 74 6c  |_relocs.QURTK_tl|
000000b0  62 6c 6f 63 6b 5f 65 6e  74 72 69 65 73 0a 54 4c  |block_entries.TL|
000000c0  42 5f 4c 41 53 54 5f 52  45 50 4c 41 43 45 41 42  |B_LAST_REPLACEAB|
000000d0  4c 45 5f 45 4e 54 52 59  0a 5b 77 5d 70 6f 6f 6c  |LE_ENTRY.[w]pool|
000000e0  5f 63 6f 6e 66 69 67 73  0a 5b 77 5d 71 75 72 74  |_configs.[w]qurt|
000000f0  6f 73 5f 74 6c 62 5f 72  65 63 6c 61 69 6d 0a 71  |os_tlb_reclaim.q|
00000100  75 72 74 6f 73 5f 62 6f  6f 74 5f 6d 61 70 70 69  |urtos_boot_mappi|
00000110  6e 67 73 0a 71 75 72 74  6f 73 5f 6d 6d 61 70 5f  |ngs.qurtos_mmap_|
00000120  74 61 62 6c 65 0a 71 75  72 74 6f 73 5f 78 6d 6c  |table.qurtos_xml|
00000130  5f 70 61 73 73 74 68 72  6f 75 67 68 00 00 00 00  |_passthrough....|
00000140  b4 42 01 84 a4 06 70 82  a0 0e 70 82 0c 04 70 82  |.B....p...p...p.|
00000150  d0 e4 3b 84 7f 00 00 00  70 e0 3b 84 50 f1 4f 84  |..;.....p.;.P.O.|
00000160  88 42 01 84 c0 f5 3d 84  18 f7 3d 84 00 00 00 00  |.B....=...=.....|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000200
512+0 records in
512+0 records out
512 bytes (512B) copied, 0.006368 seconds, 78.5KB/s
```

First page of modem FW region read from Linux

# SBL:

- Initialize DDR memory

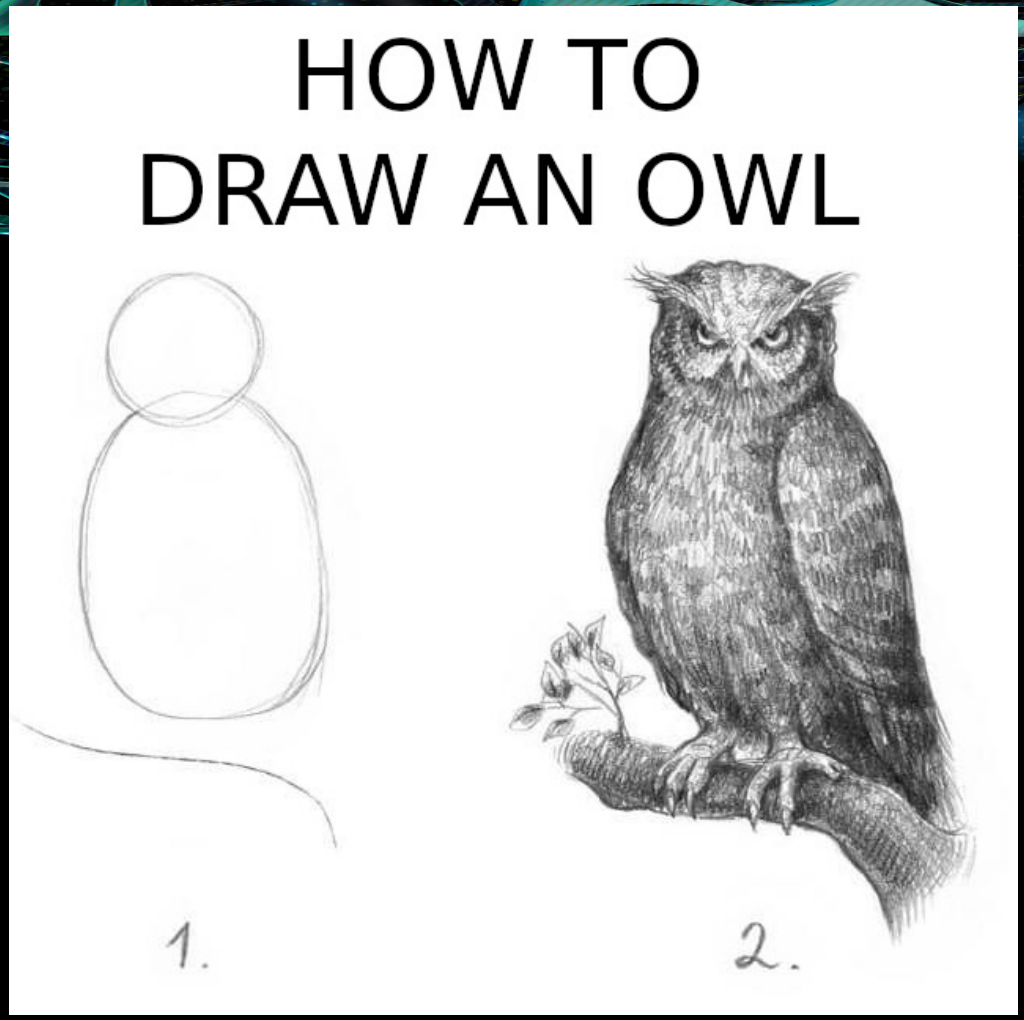
Don’t need these

- Boot QSEE (TrustZone)
- Boot RPM (Resource Power Managment) core
- Verify and run aboot (Android Bootloader)

SBL – Secondary Boot Loader

# SBL: write our own ~~OS~~

- Runs right after ARM PBL with highest privileges
- Nothing else is running (TZ/RPM)
- Initializes DDR, clocks, regulators
- Can start/stop Hexagon core with custom FW image
- CLI shell via USB




8.5k SLoC bare-metal Rust

# MDMShell

- R&W ARM/Hexagon memory
- Individual control of clock/reset signals
- PMIC control via SPMI (System Power Management Interface)
- Activate & authorize JTAG

```
 __  __  ____  __  __  ____  _          _ _
|  \/  ||  _ \|  \/  | / ___|| |__   ___| | |
| |\/| || | | | |\/| | \___ \| '_ \ / _ \ | |
| |  | || |_| | |  | |  ___) | | | |  __/ | |
|_|  |_||____/|_|  |_| |____/|_| |_|\___|_|_|
           [ modem debug shell ]

[   1.089] MSS clocks enabled, RMB base: 0x4020000
[   1.228] JTAG enabled (QDSD + ARM debug)
Welcome to nut-shell! Type '?' for help.
@/> ls
  reboot  - Reboot (edl/fastboot)
  memtest  - Test DDR memory
  md  - Memory hexdump: md <hex_addr> [length]
  mw  - Memory write: mw <hex_addr> <hex_data|0xWORD>
  rr  - Read registers: rr <hex_addr> [count]
  rmb  - Dump RMB (Remote Message Buffer) registers
  q6  - Dump QDSP6 subsystem register state
  load  - Load binary to memory: load <hex_addr>
  go  - Jump to address: go <hex_addr>
  fuses  - Dump QFPROM security fuse state
  clocks  - Dump GCC clock branch status
  q6start  - Start Q6 core: q6start <hex_addr> [--auth|--direct] [--noreset]
  q6stop  - Stop Q6 core and re-enable MSS clocks
  q6reset  - Q6 reset control: q6reset [assert|release]
  uptime  - Show time since boot
  dump  - Dump memory as hex: dump <hex_addr> <size>
  spmi  - SPMI: spmi r <sid> <addr> | w <sid> <addr> <val> | ch | vdd
  mbaread  - Read Q6 memory via MBA: mbaread <hex_addr> <size>
  mbarr  - Read Q6 registers via MBA: mbarr <hex_addr> [count]
  mbamw  - Write Q6 register via MBA: mbamw <hex_addr> <hex_value>
  qdsd  - Debug over SD: qdsd [status|swduart|jtag|off]
  dbgauth  - ARM debug auth
  led  - LED control via PM8019 GPIO
  clk  - clk <on|off> <NAME> [bit] | clk list
```

# Downloaded CP PBL

```
0012238c  51 B0 4D 20 51 55 4...        ds          "Q",B0h,"M QUALCOMM COPYRIGHT"
001223a3  4D 4F 44 45 4D 20 ...         ds          "MODEM BOOT ROM VERSION: 1.0"
 001223bf 4D 4F 44 45 4D 20 ...         ds          "MODEM PBL VERSION"
```

# Security assumptions (revisited)

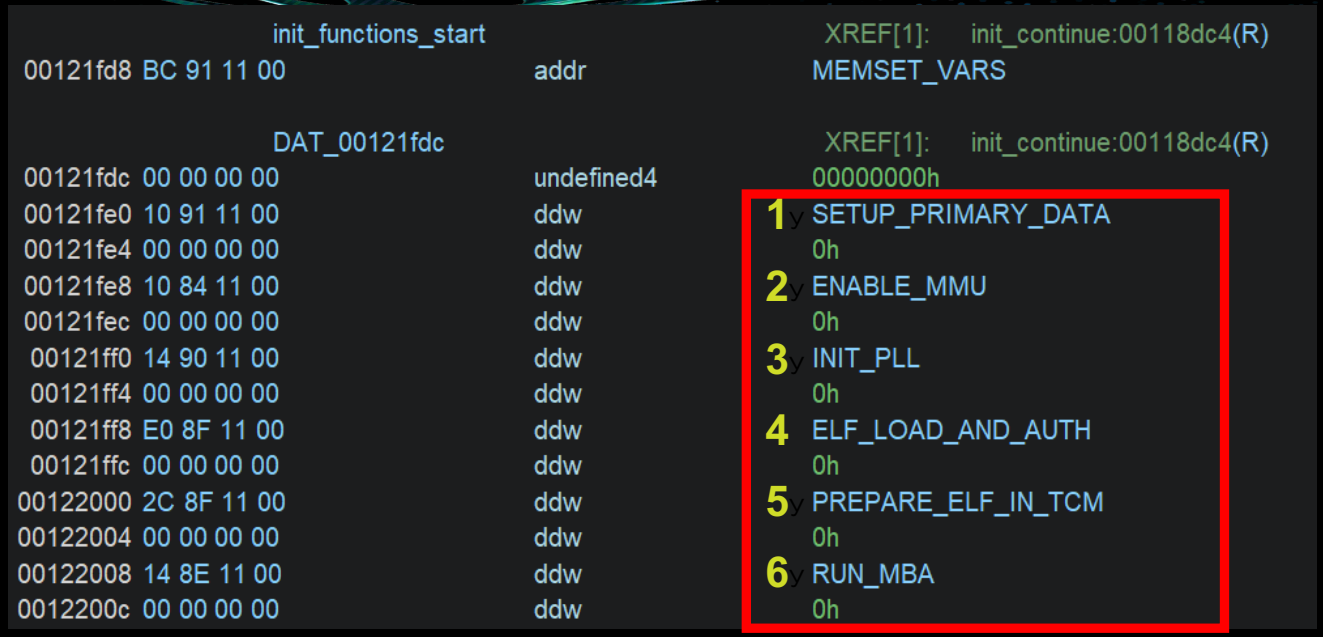
- ARM ROM only executes verified code
- CP ROM cannot be read
- CP ROM only executes verified code


# IV. CP and more

# CP

# CP: PBL

- PBL has 6 stage loading process
- It uses MMU to prevent some attack vectors
- Almost the same as in ARM PBL



# CP: How to misuse warm boot feature

- PBL checks RSA certs, but…
- There is a FUSE to enable bypass (obvious)
- There is an internal CP (MSS) register accessible from ARM (not obvious)

FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

MSS REG

```
if (*(DWORD)0x04180058 & 0x02 != 0)
{
    v2 = (int) (*(DWORD)(0x04180400));
    return v2(&EVB);
}
```

# CP: How to misuse warm boot feature

- PBL ch
  but…
- There i
  bypass
- There i
  (MSS)
  from AP

FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

MSS REG

```
if (*(DWORD)0x04180058 & 0x02 != 0)
{
    v2 = (int) (*(DWORD)(0x04180400));
    return v2(&EVB);
}
```

# CP: How to misuse warm boot feature

- Initialize CP (clocks, power)
- Initialize DDR
- Put Warm Boot code to DDR
- Put addr of Warm Boot code to reg 0x4180400
- Set reg 0x4180058 to 0x3
- Start CP

FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

MSS REG

```
if (*(DWORD)0x04180058 & 0x02 != 0)
{
    v2 = (int) (*(DWORD)(0x04180400));
    return v2(&EVB);
}
```

# CP: How to misuse warm boot feature

- Initialize CP (clocks, power)
- Initialize DDR
- Put ~~Warm Boot~~ YOUR code to DDR
- Put addr of ~~Warm Boot~~ YOUR code to reg 0x4180400
- Set reg 0x4180058 to 0x3
- Start CP
- Profit

```
[0x04020000] MBA_IMAGE:          0x8fa00000
[0x04020004] PBL_STATUS:         0x00000000
[0x04020008] MBA_COMMAND:        0x00000000
[0x0402000c] MBA_STATUS:         0x00000000
[0x04020010] PMI_META_DATA:      0x00000000
[0x04020014] PMI_CODE_START:     0x00000000
[0x04020018] PMI_CODE_LENGTH:    0x00000000
[0x0402001c] PROTOCOL_VERSION:   0x00000000
[0x04020020] MBA_DEBUG_INFO:     0xaabbccdd
```

# Security assumptions (revisited)

- ARM ROM only executes verified code
- CP ROM cannot be read
- CP ROM only executes verified code

# JTAG: Where the right pins are?

## Debug(SWD/UART/Trace) over SDC2 enabling #309

Open

fxsheep opened on Oct 4, 2023 · edited by fxsheep

Edits

According to the TRM( `lm80-p0436-100_d_snapdragon_410e_apq8016e_tech_reference_manual_revd.pdf` , page 3357), msm8916 supports NIDnT(Narrow Interface for Debug and Test) feature, implemented in a QDSD(Qualcomm Debug over SD?) block inside TLMM.

# JTAG: Where the right pins are?

Debug(SWD/UART/ ing #309

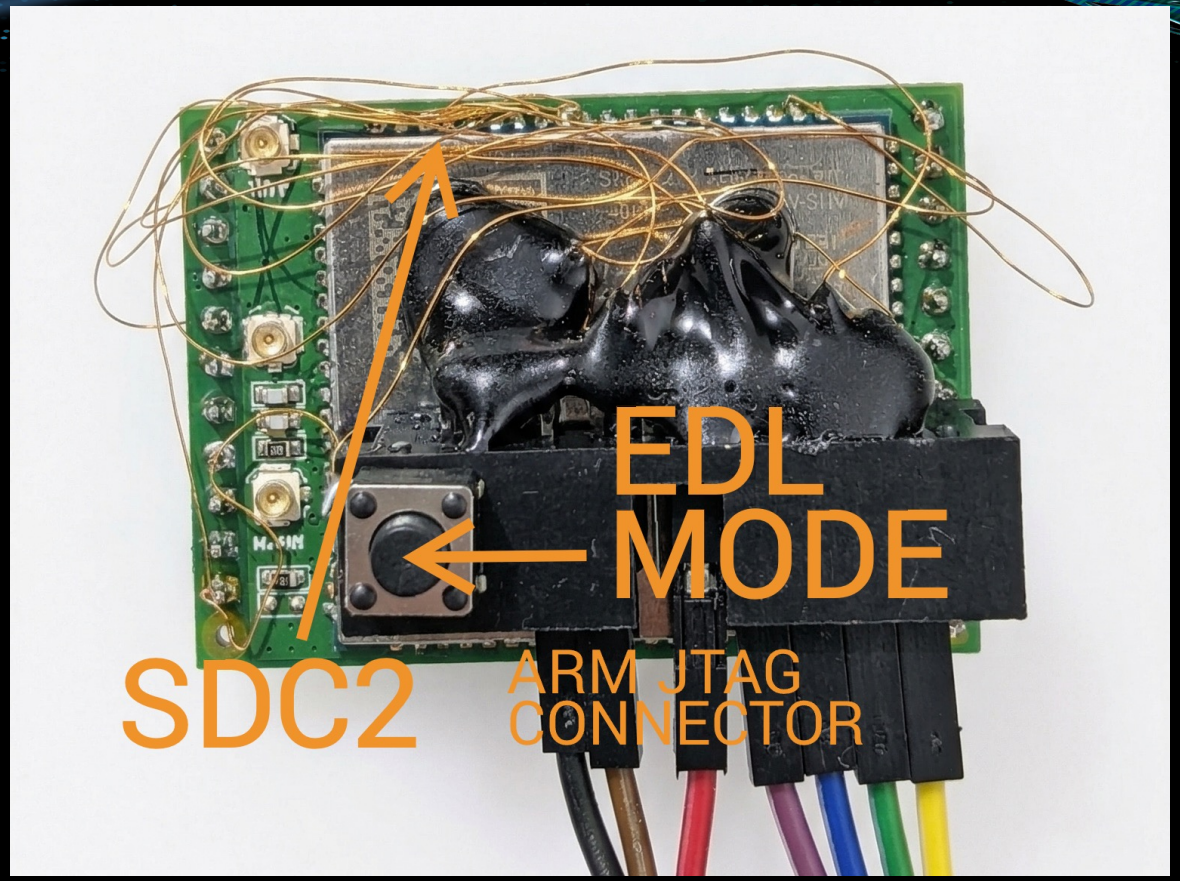
Open

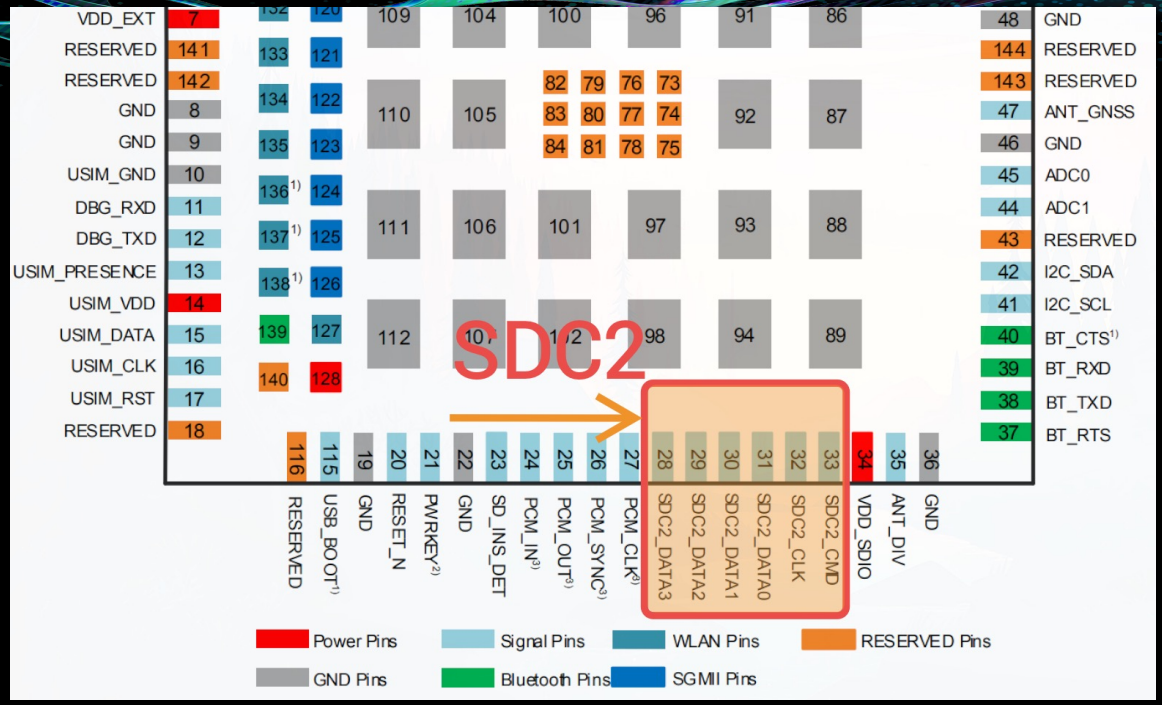
fxsheep opened on Oct 4, 2023 · e

Edits

According to the TRM( lm80-p043 anual_revd.pdf , page 3357), msm8916 supports NIDnT(Narrow Interface f ualcomm Debug over SD?) block inside TLMM.

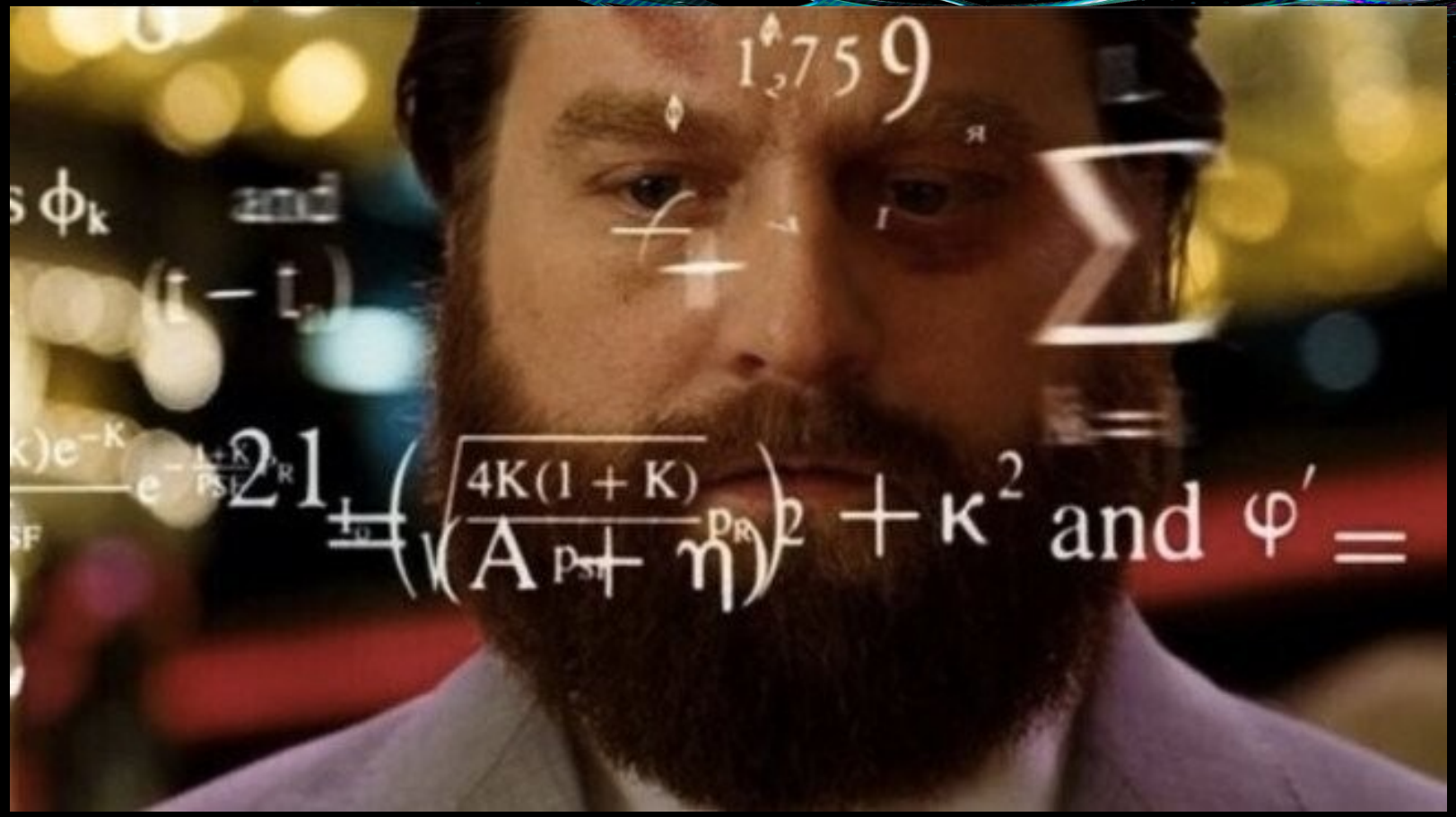
# JTAG: Where the right pins are?



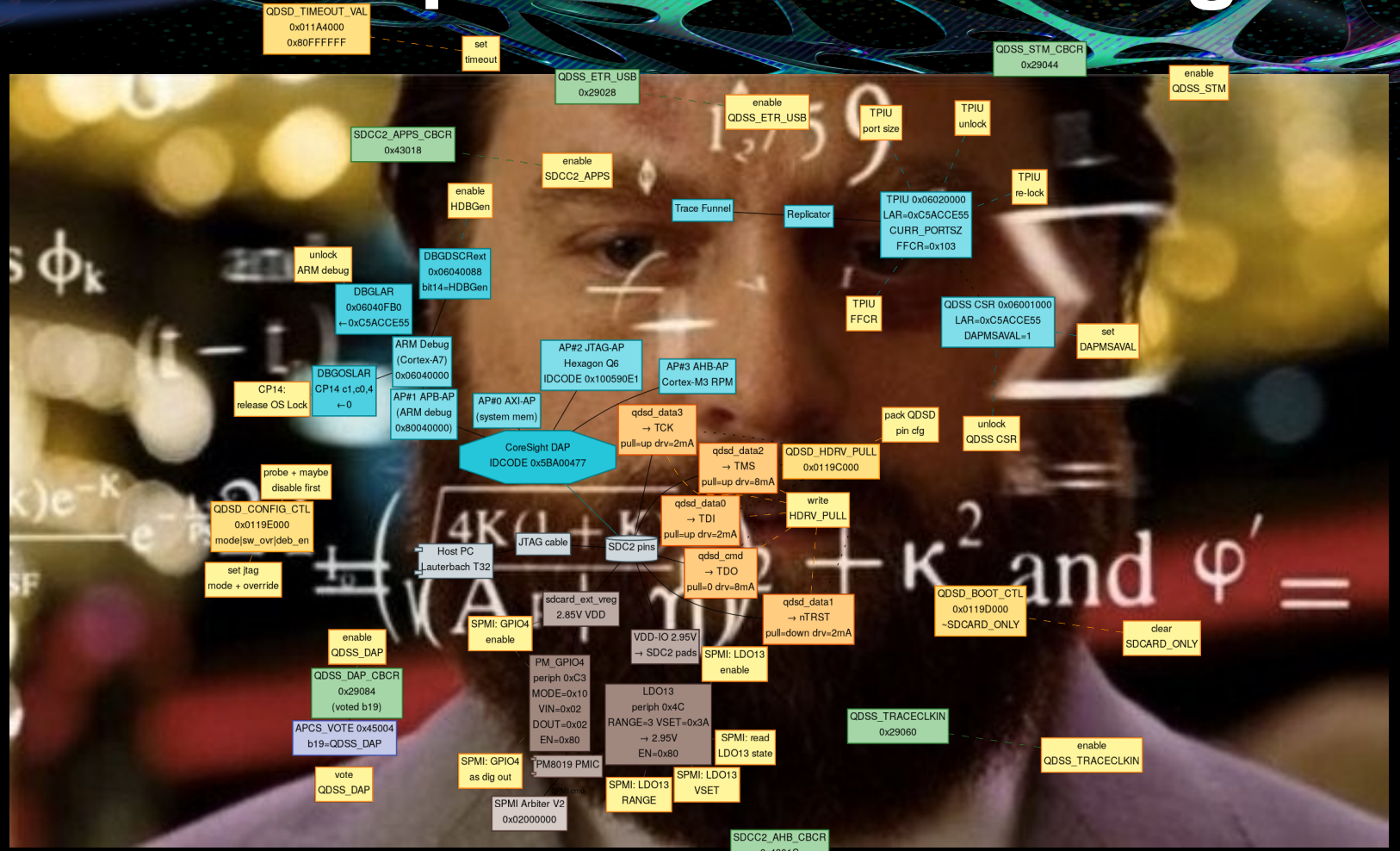




Quectel EC25-E Modem Module Pinout
(based on the **MDM9207** chipset)

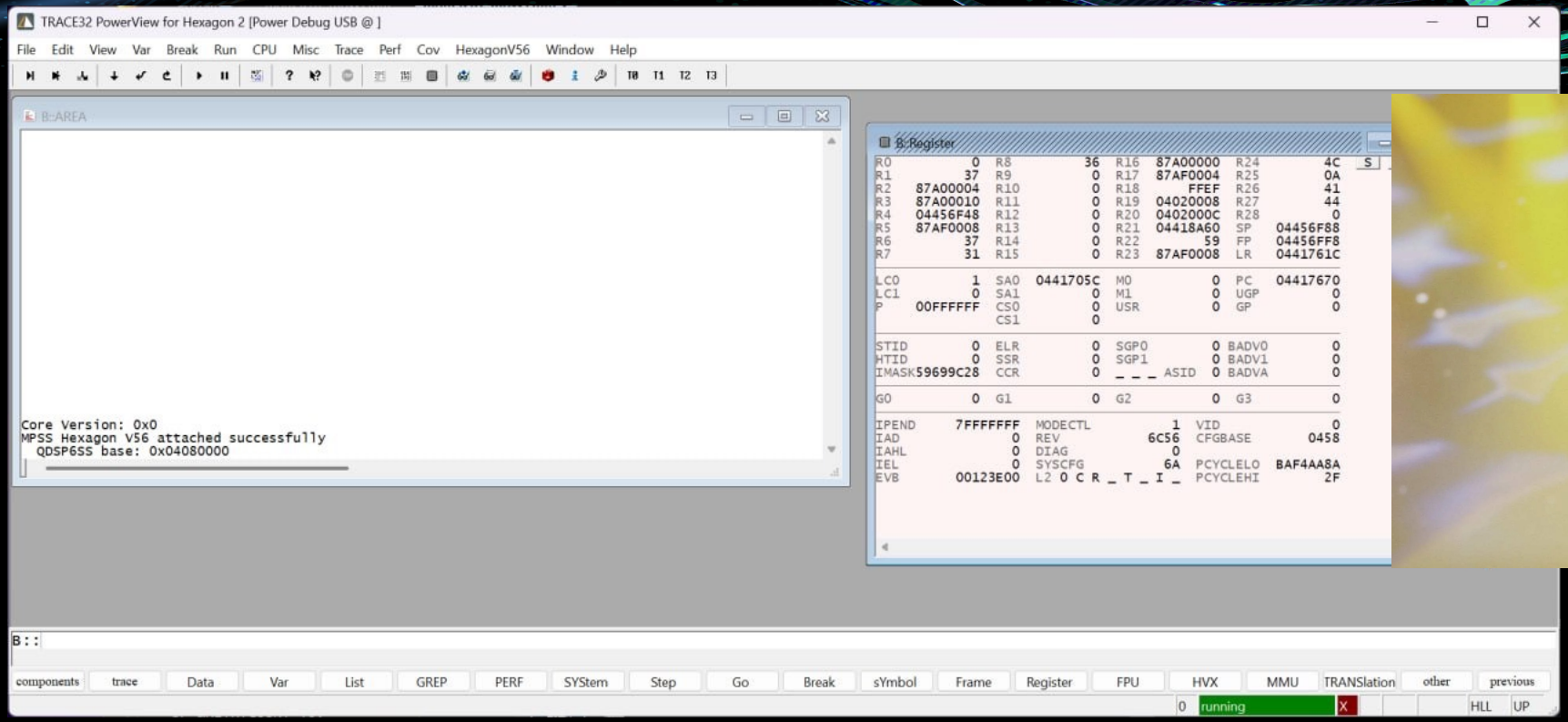
# JTAG: When the pins are not enough



# JTAG: When the pins are not enough



# JTAG: Enabled




# Fuses? What Fuses?

HEXAGON FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

ARM FUSE

```
if (*(DWORD)0x000a602C & 0x20 == 0)
{
    v2 = (int) (*(DWORD)(0x00210000));
    return v2(&EVB_ARM);
}
```

```
000a6000 40050000
000a6004 00000003
000a6008 00000000
000a600c 00000005
000a6010 0242f800
000a6014 d41c008e
000a6018 00003c50
000a601c 00000000
000a6020 00000000
000a6024 00000000
000a6028 00000000
000a602c 000002e1
```

# Fuses? What Fuses?

HEXAGON FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

ARM FUSE

```
if (*(DWORD)0x000a602C & 0x20 == 0)
{
    v2 = (int) (*(DWORD)(0x00210000));
    return v2(&EVB_ARM);
}
```

```
000a6000 00000100 00
000a6004 00000100 03
000a6008 00000100 00
000a600c 00000100 05
000a6010 00000100 00
000a6014 00000100 8e
000a6018 00000100 50
000a601c 00000100 00
000a6020 00000100 00
000a6024 00000100 00
000a6028 00000100 00
000a602c 00000100 00
000a6030 00000100 e1
```

# Fuses? What Fuses?

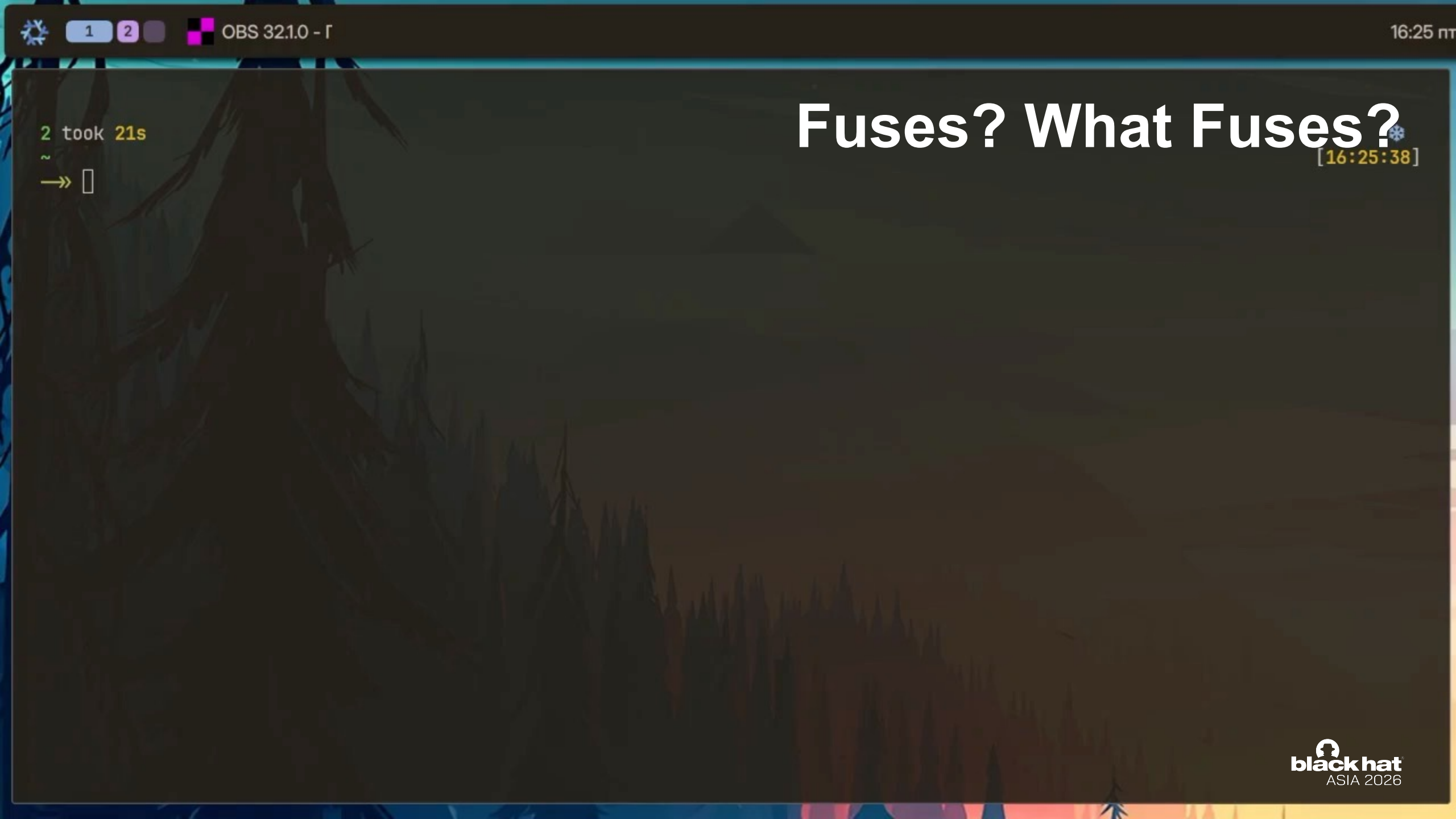
HEXAGON FUSE

```
if (*(DWORD)0x000a6018 & 0x40 == 0)
{
    v2 = (int) (*(DWORD)(0x04080010)) << 4;
    return v2(&EVB);
}
```

ARM FUSE

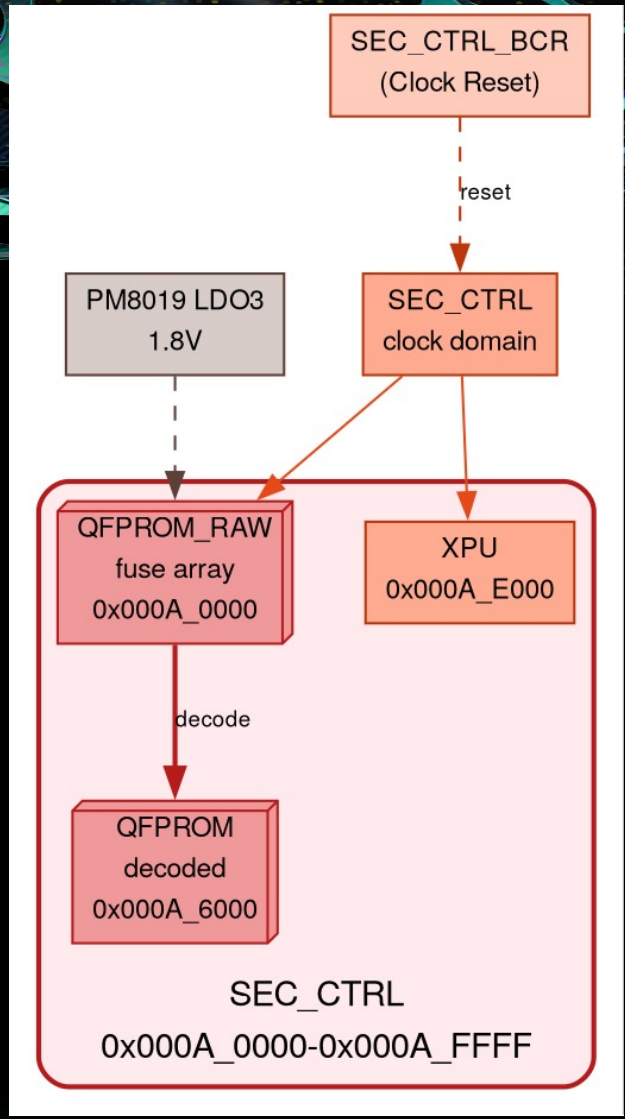
```
if (*(DWORD)0x000a602C & 0x20 == 0)
{
    v2 = (int) (*(DWORD)(0x00210000));
    return v2(&EVB_ARM);
}
```

```
000a6000 cafebabe 0 00
000a6004 cafebabe 0 03
000a6008 cafebabe 0 00
000a600c cafebabe 0 05
000a6010 cafebabe 0 00
000a6014 cafebabe 0 8e
000a6018 cafebabe 0 50
000a601c cafebabe 0 00
000a6020 cafebabe 0 00
000a6024 cafebabe 0 00
000a6028 cafebabe 0 00
000a602c cafebabe 0 e1
```


1 2
OBS 32.1.0 - Г
16:25 пт
2 took 21s
~
Fuses? What Fuses?
[16:25:38]
black hat
ASIA 2026

# Fuses? What Fuses?

- SEC_CTRL holds Fuses
- If you RESET SEC_CTRL clock, the counter just stops…
- And the last read Fuse value is mirrored to ALL other Fuses
- Profit




# V. Conclusions

# Timeline

## CVE-2026-25262

- MDM9x07
- MDM9x45
- MDM9x65
- MSM8909
- MSM8916
- MSM8952
- SDX50.

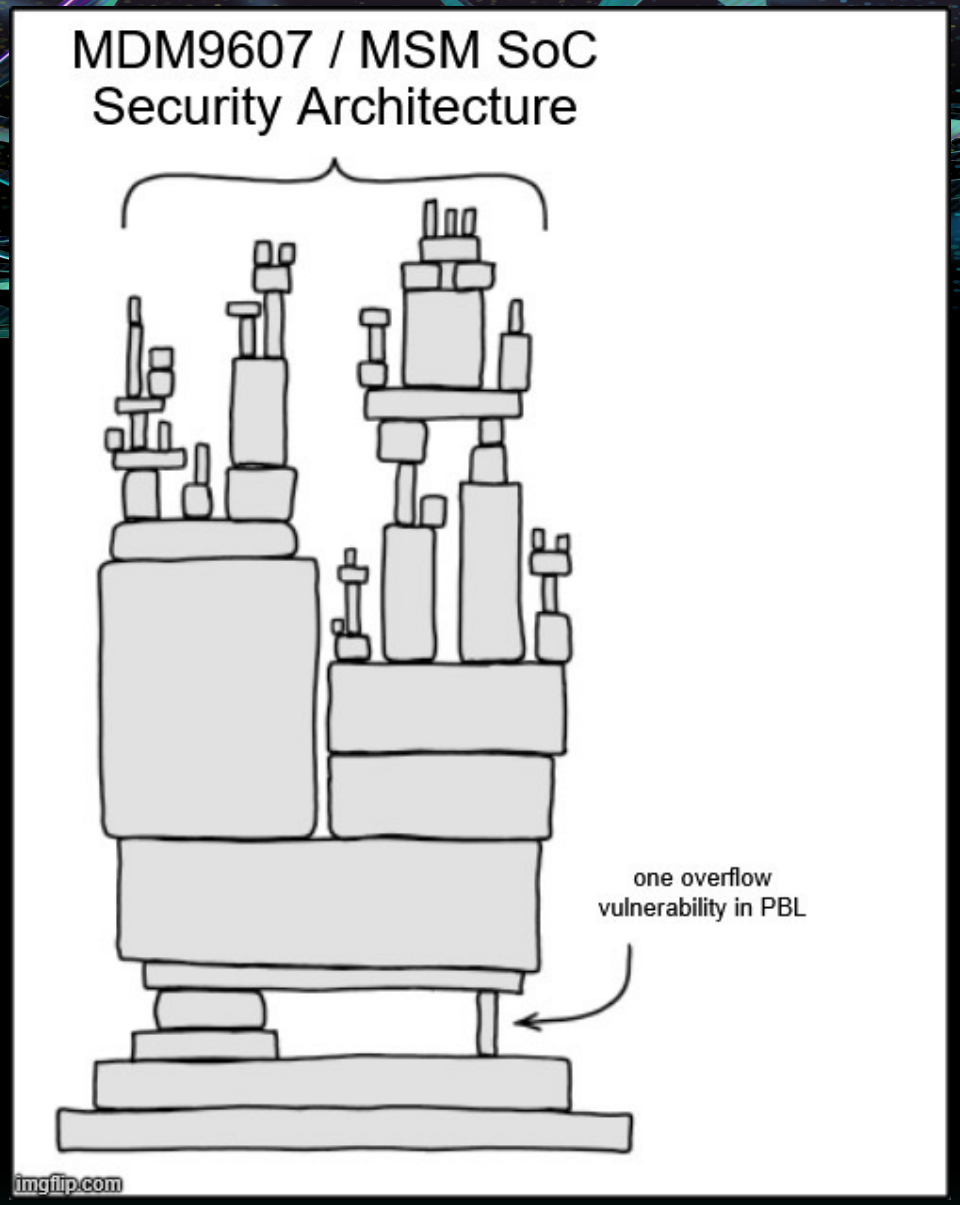
March 2025 April 2025 May 2026

# Mitigation guidelines

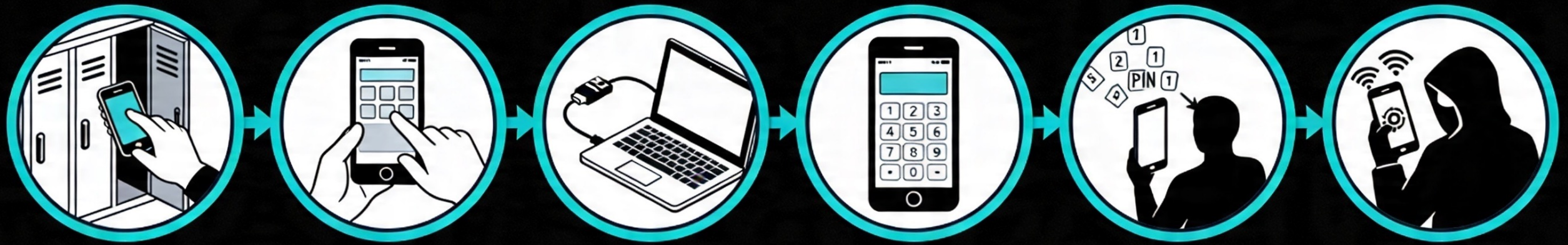
- Just
- Update
- Your
- SoCs
- Physically

# Security assumptions

- ARM ROM only executes verified code
- CP ROM cannot be read
- CP ROM only executes verified code
- BONUS: Enabled JTAG (both ARM and Hexagon)



# Kill Chain: example



# Future plans

1. Fuzzing Hexagon Instructions (in Progress)
2. Research devices on AXI/APB/AHB buses
3. ?????


# THANK YOU!

@n0um3n0n
Alexander.a.Kozlov@kaspersky.com
@madprogrammer
Sergey.Anufrienko@kaspersky.com