CYBER INTELLIGENCE

Vlind Glitch:

A Blind VCC Glitching technique to bypass the secure boot of the Qualcomm MSM8916 mobile SoC

Hector Marco & Vicent Arnau

EX FILES

The EXFILES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 883156

Agenda

- About us
- Motivation & Goals
- Attack Scope
- Vlind Glitch Attack
- Demo
- Conclusions

About us

Dr. Hector Marco

- Founder of Cyber Intelligence S.L.
- Working in Cybersecurity > 15 years

Mr. Vicent Arnau

- Hardware Security Researcher
- Specialized in Smartphone Security

Cyber Intelligence S.L.

- Company based in Spain.
- Specialized in software and hardware security.
- https://cyberintel.es
- security@cyberintel.es

Motivation & Goals

EXFILES Project

- Motivation
  - Encrypted phones are often a key factor in criminal cases.
  - Data stored may contain critical evidences.
  - Current devices are strongly encrypted.

- Main Goals
  - To find ways to access protected evidences.
  - By using software and hardware methods for data extraction.
  - Create and develop tools to improve reverse engineering.

Motivation & Goals

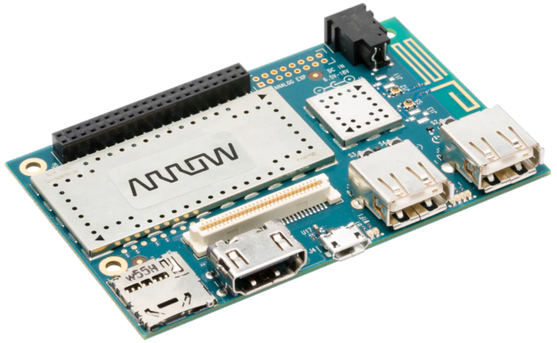
About this Talk

- Main goals
  - Bypass the Secure Boot of a real hardware (BFU) using VCC Glitching.
  - Run Arbitrary code with maximum privileges (EL3).
  - Provide a generic method that does not require reversing engineering.

- Device Under Test:
  - We started with a device we can enable secure boot (have the keys!)
  - **Board:** `DragonBoard 410c`
  - **SoC:** `MSM8916/APQ8016`

Attack Scope

Target: DragonBoard 410c



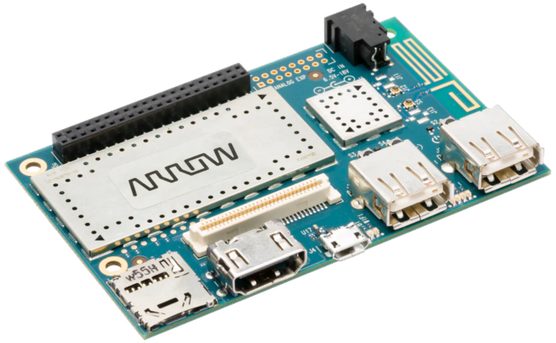
- The board follows Qualcomm's Secure Boot.

- The Board comes with Secure Boot disabled
  - Do not confuse this with HASH verification!
  - Modifying the bootloader code will result in a fail because a HASH mismatch but not because of the secure boot.

- Secure boot must be enabled
  - We have the keys → easier setup, debug, etc.
  - The approach can be used on devices we do not have the keys.

Attack Scope

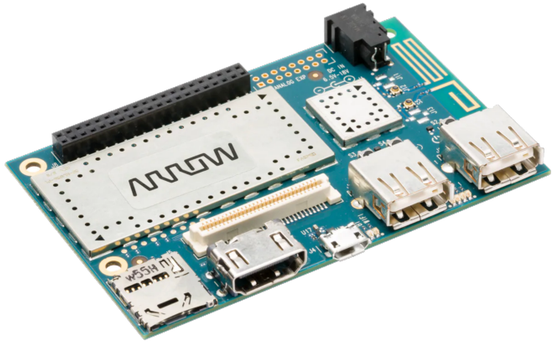
Secure Boot



- To bypass the secure boot we need to decide in which point.
- We aim to bypass it during the boot process
  - Multiple verifications → Multiple opportunities
- PBL: Primary bootloader in ROM (also called BootROM)
  - The first code executed by the CPU
- SBL: is the secondary boot loader
  - Loads next stages (normal and secure world).
- EDL is the Emergency Downloader Mode
  - After a system reboot specifying that the PBL need to boot in EDL mode.
  - When SBL is corrupted, eMMC is not working, etc.

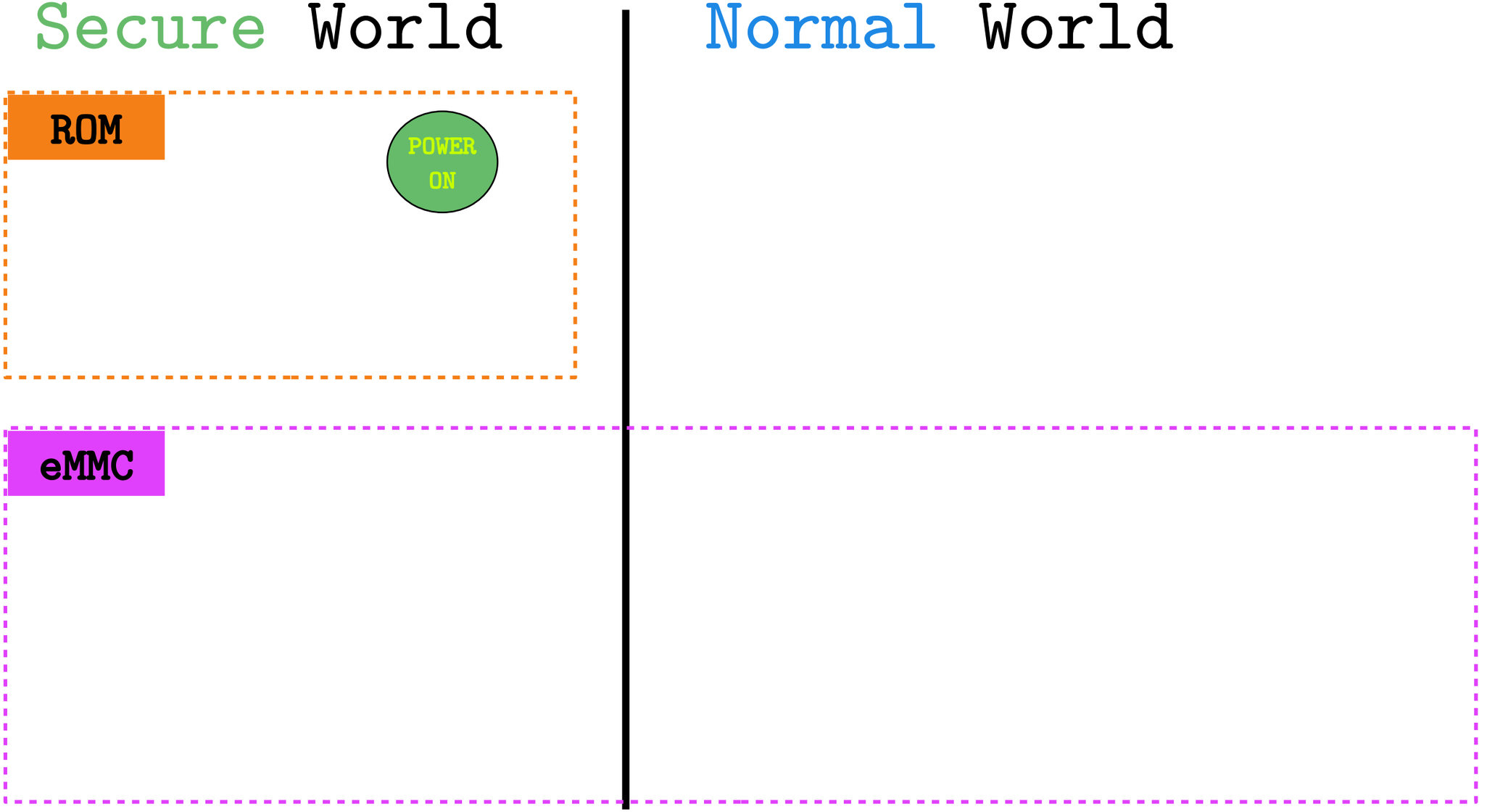
Attack Scope

Secure Boot

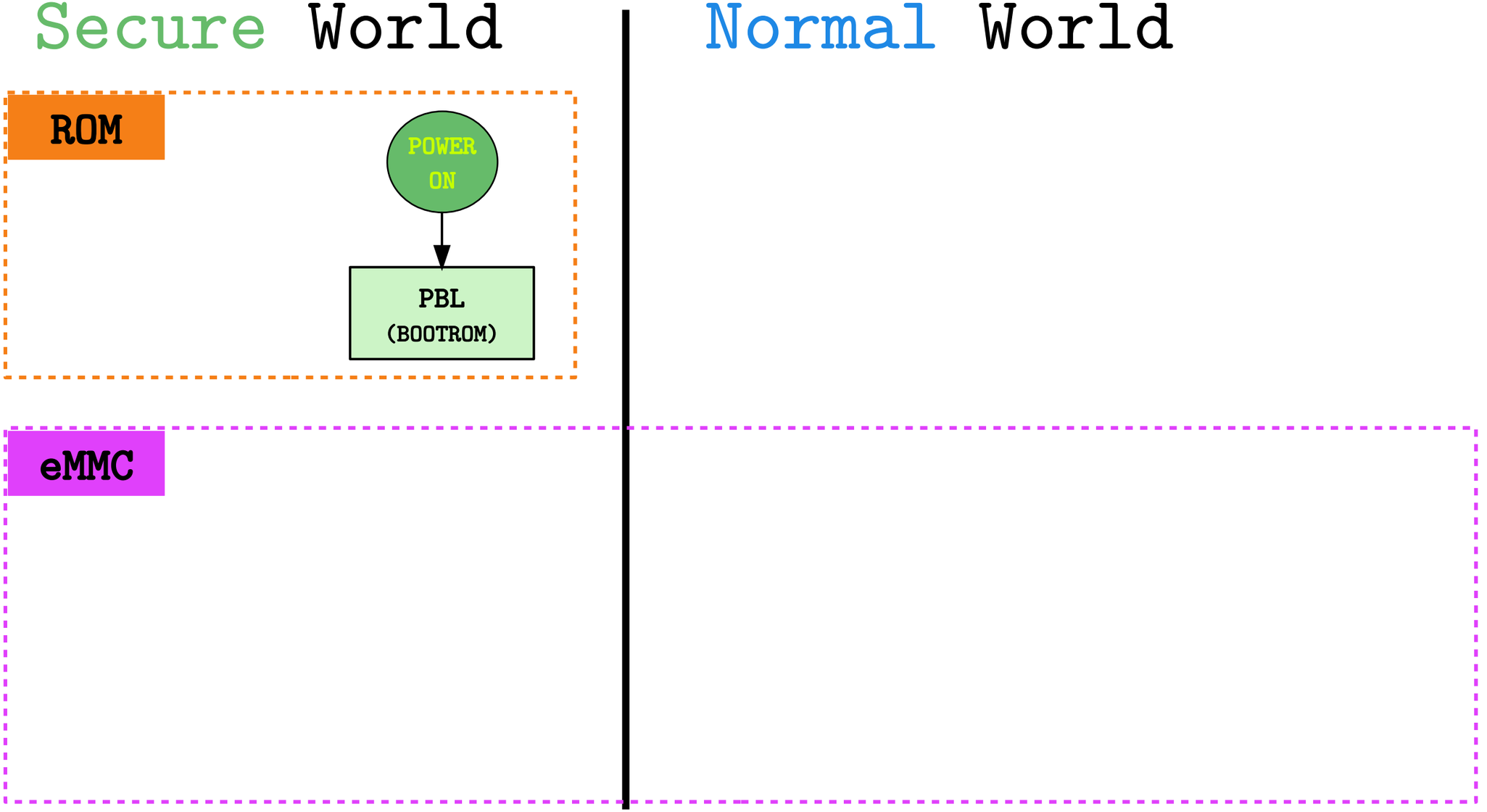


- EDL can load programs via USB to add extra capabilities
  - Those programs are named “programmers”
  - Can be used to re-flash partitions of the eMMC
  - Useful when partitions are corrupted and phone does not boot
- In the DragonBoard 410c the EDL:
  - Runs with maximum privileges (EL3).
  - Accepts only programmers with valid signatures/keys.
- Qualcomm’s EDL Image verification
  - Our attack bypasses the programmer image verification.
  - Same can be applied to the SBL.
  - The attack does not depend on flash partitions.

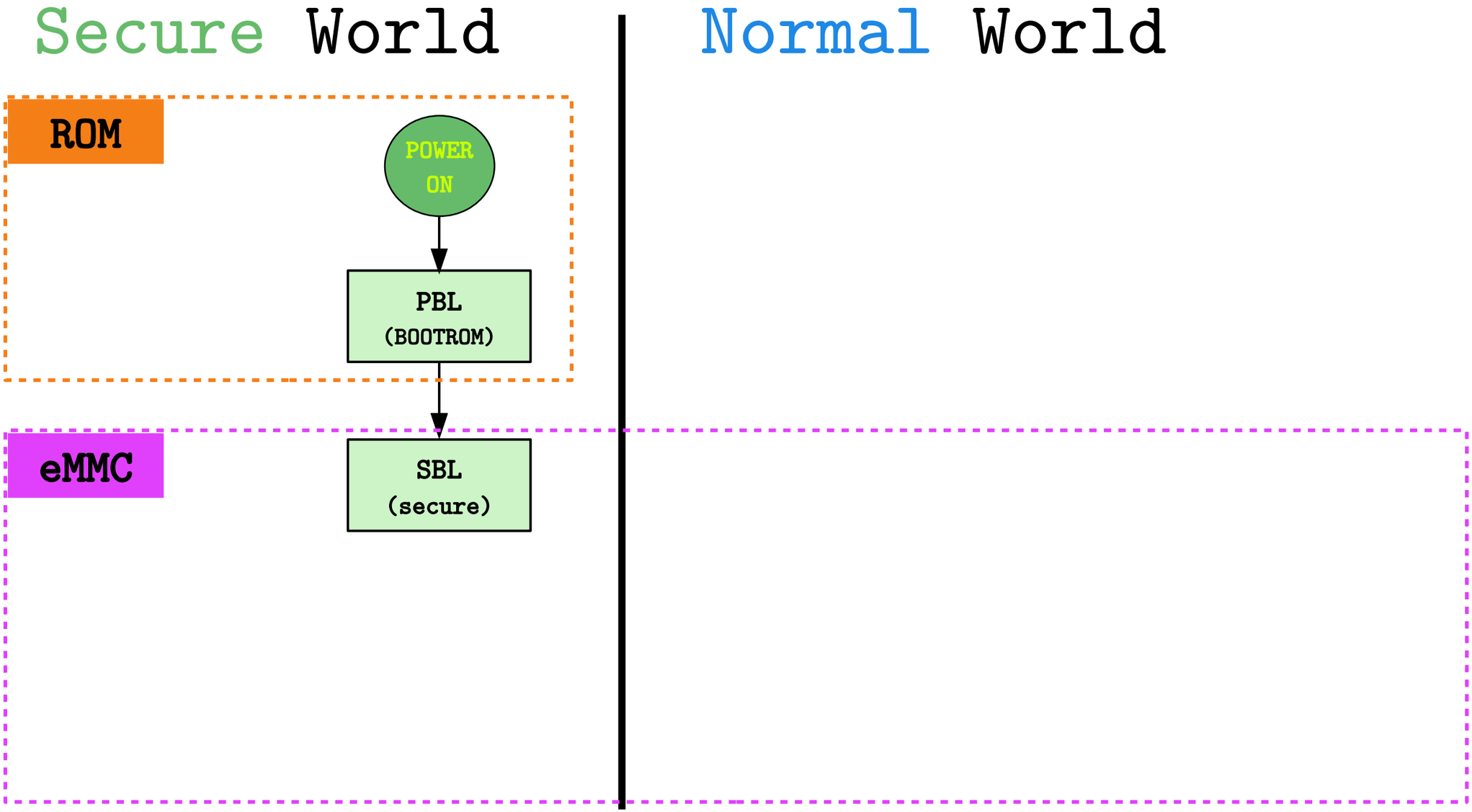
Attack Scope



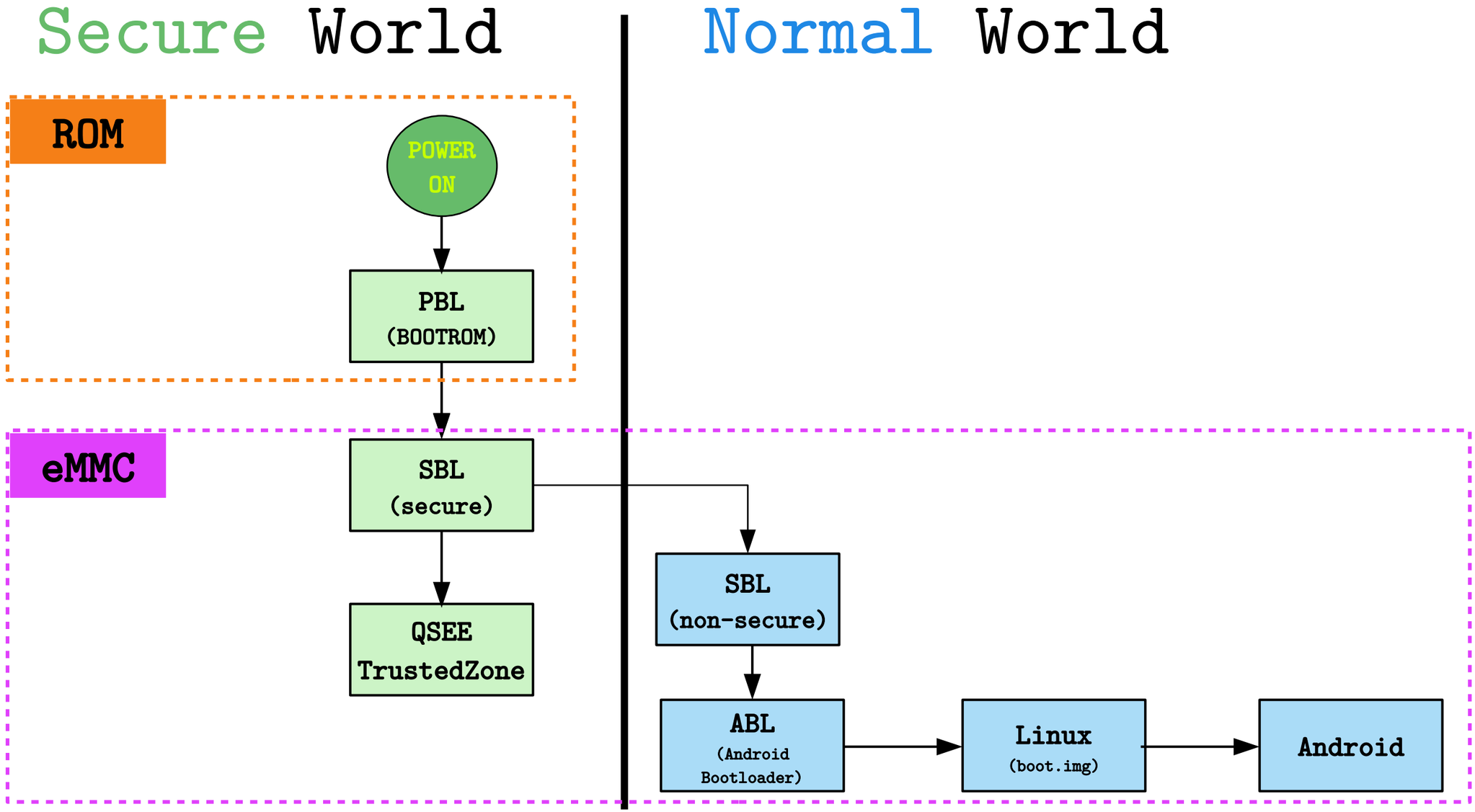
Attack Scope



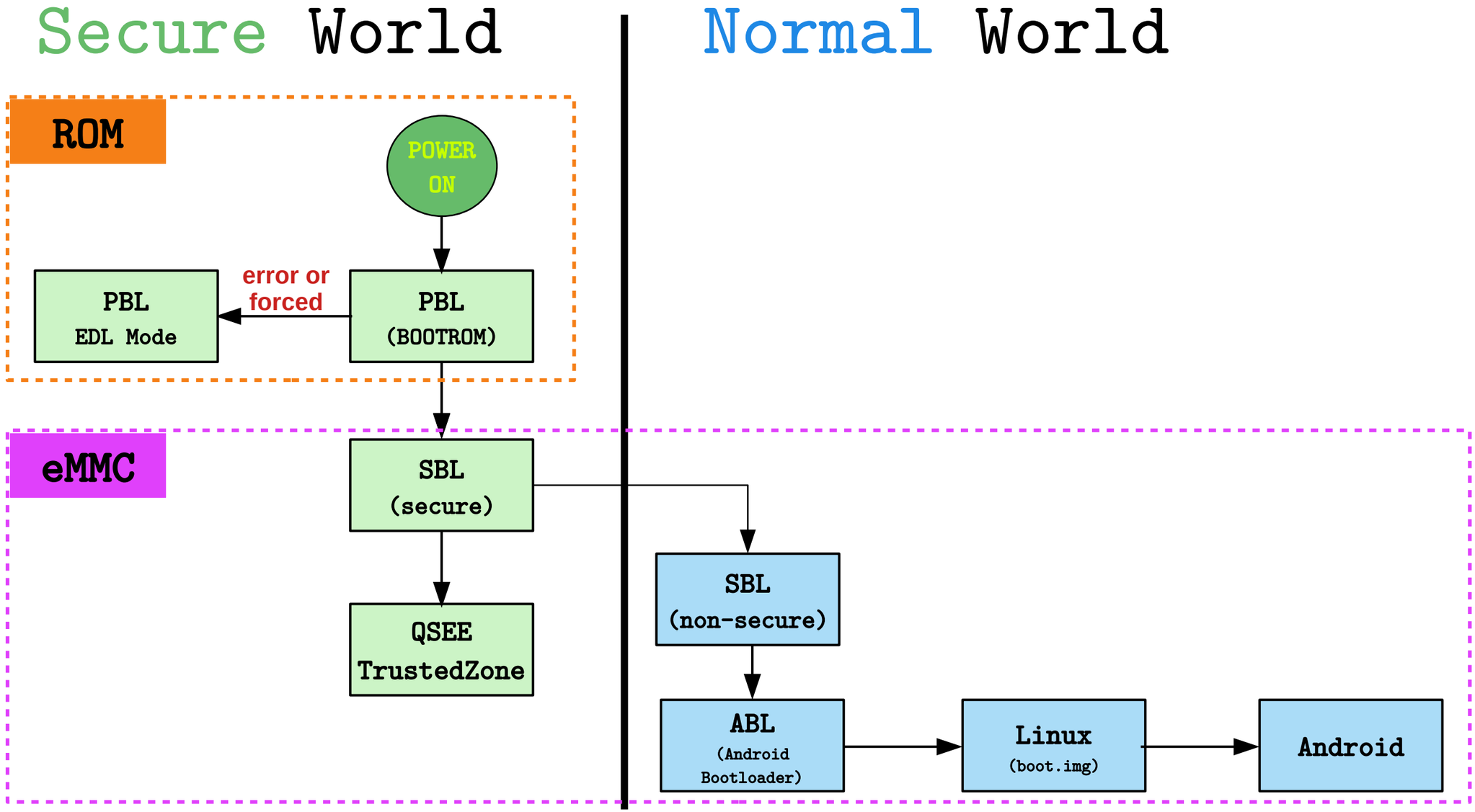
Attack Scope



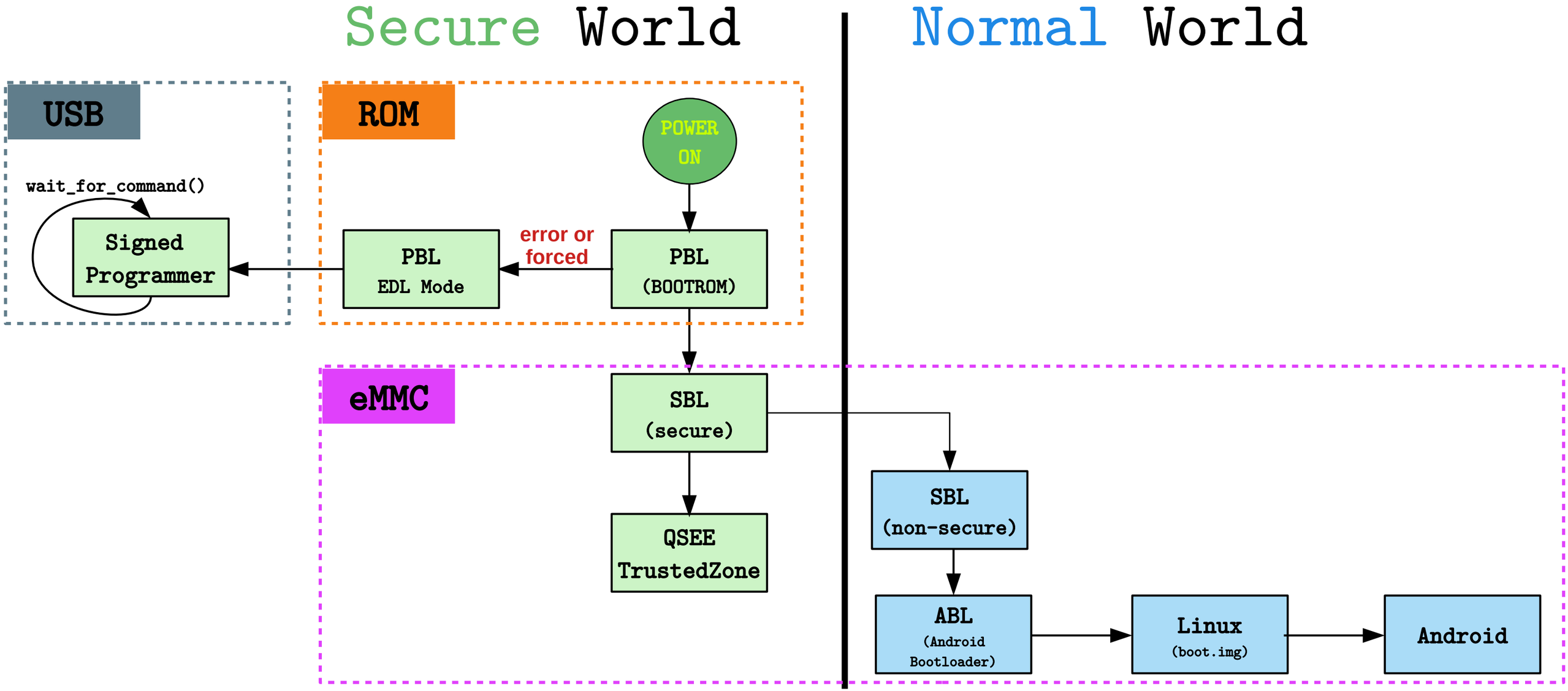
Attack Scope



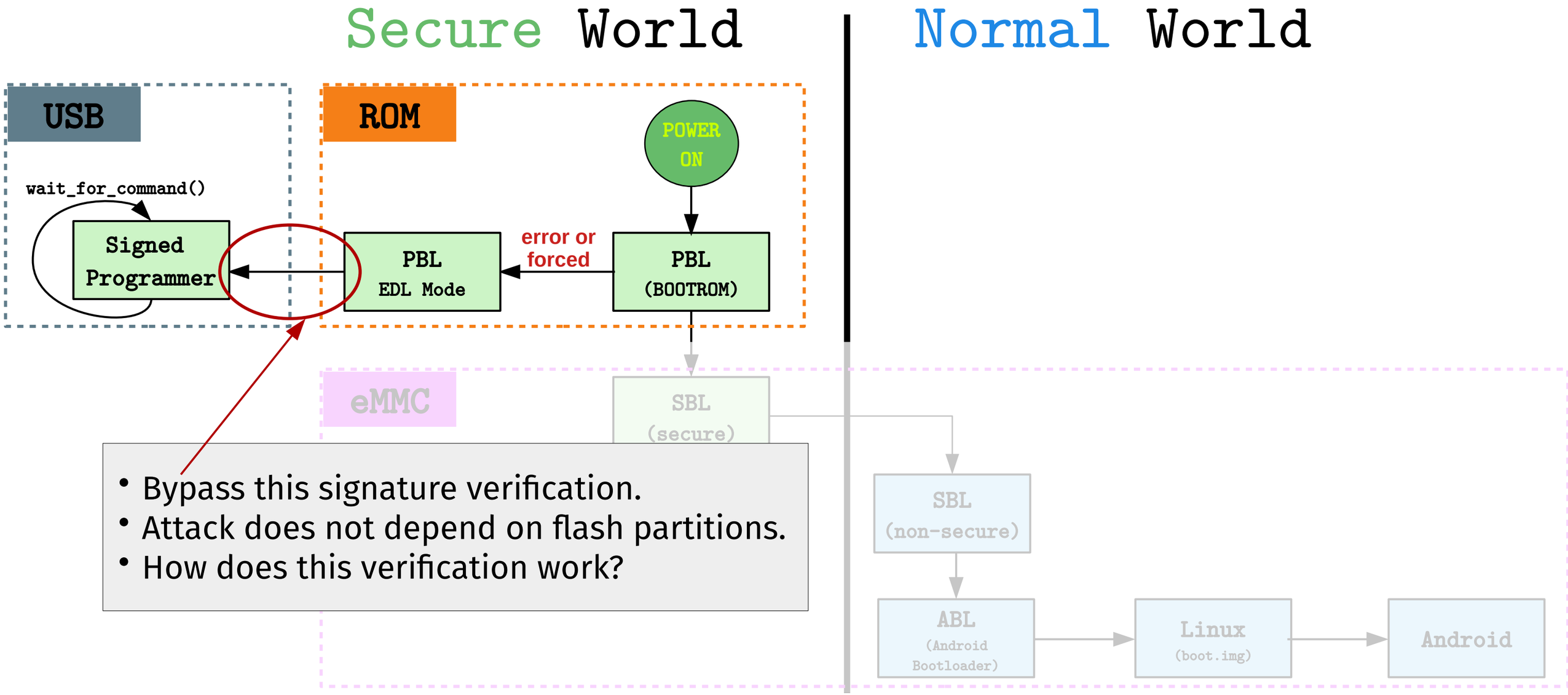
Attack Scope



Attack Scope



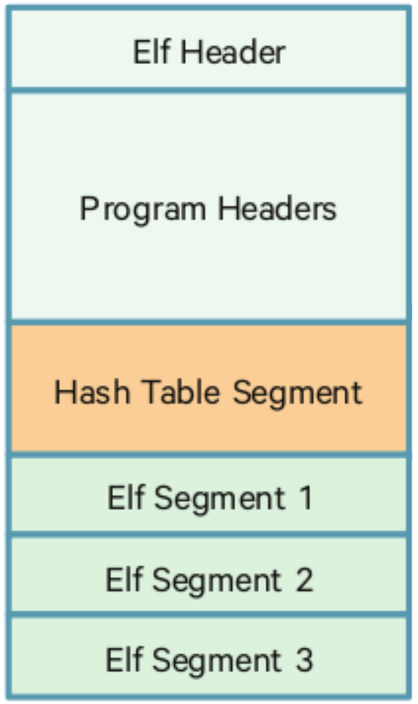
Attack Scope

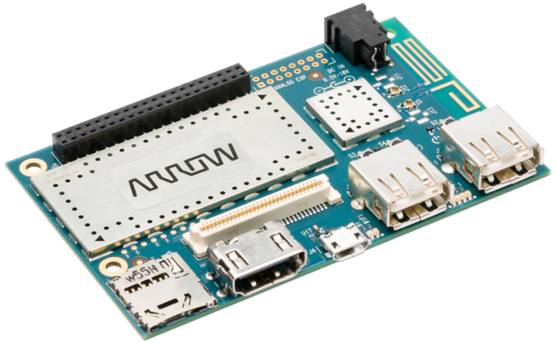


- Bypass this signature verification.
- Attack does not depend on flash partitions.
- How does this verification work?

Attack Scope

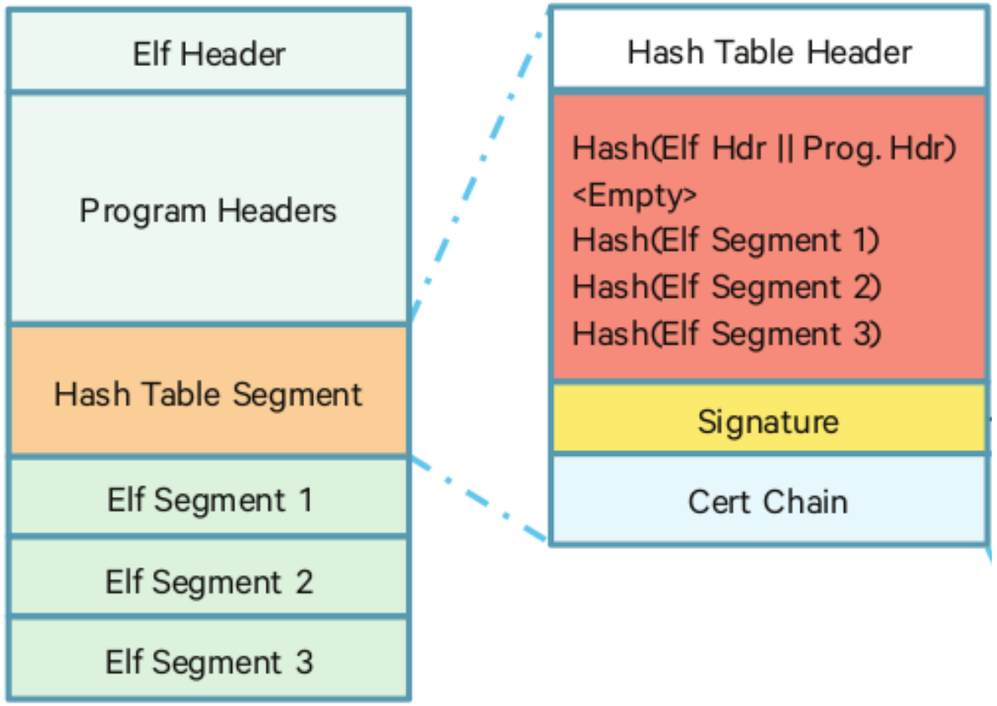
EDL Image verification

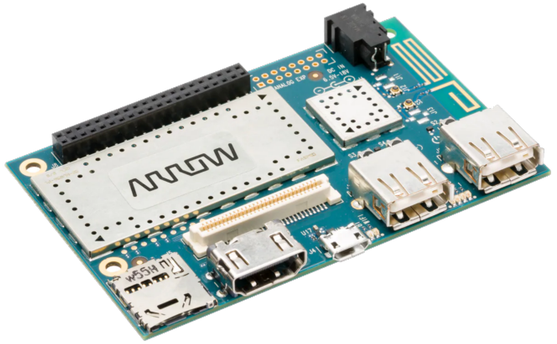




Attack Scope

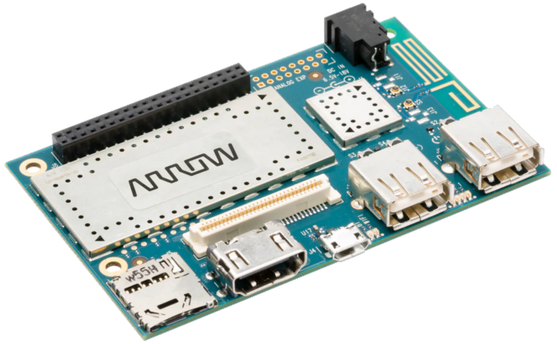
EDL Image verification

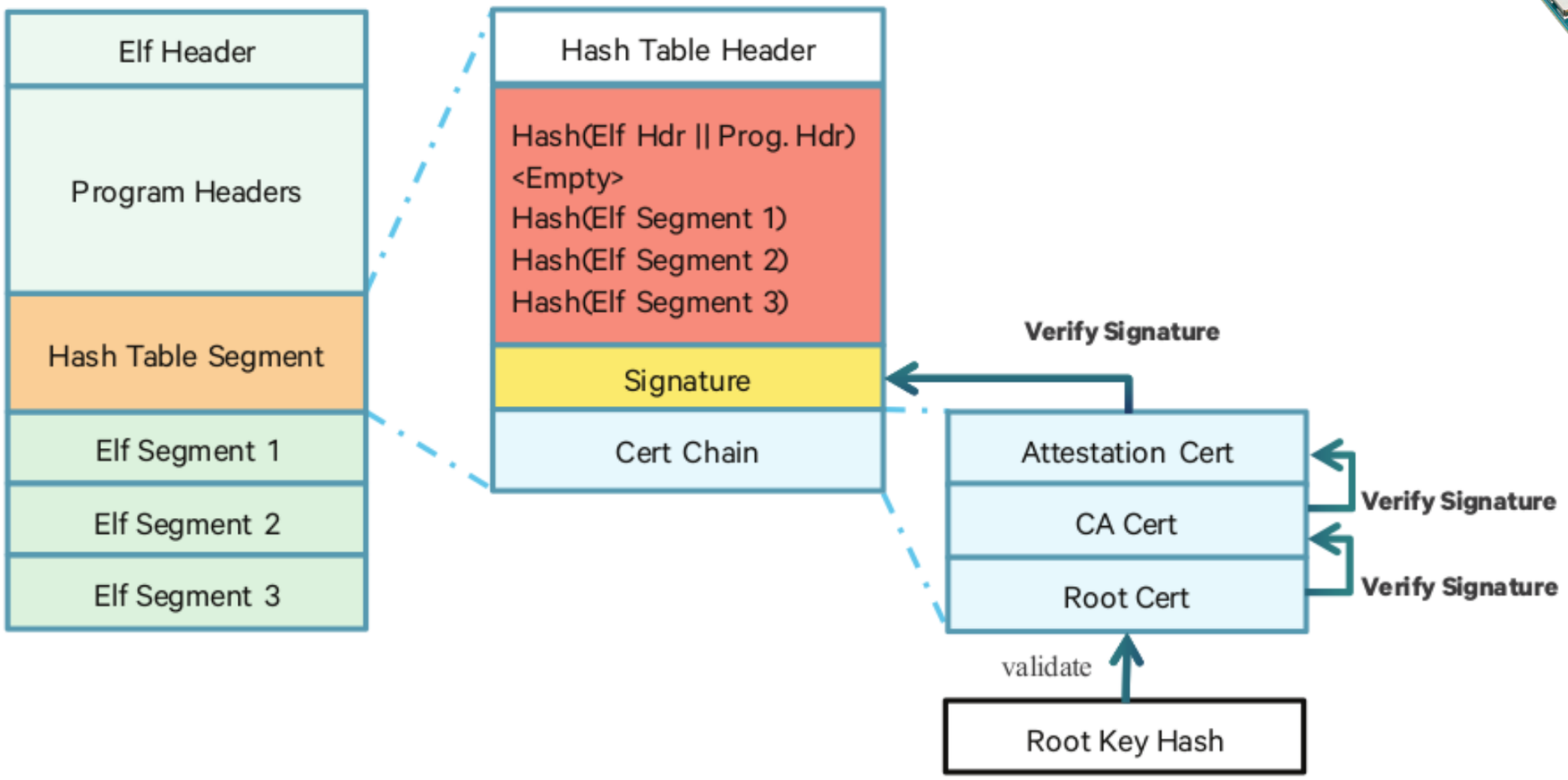




Attack Scope

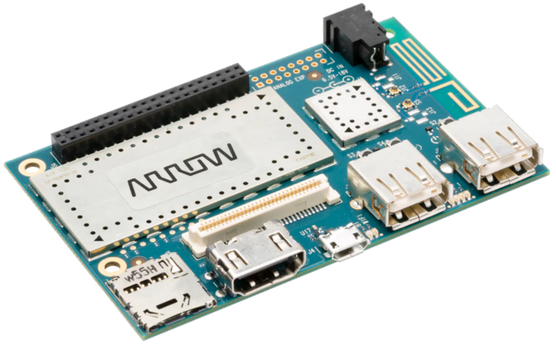
EDL Image verification

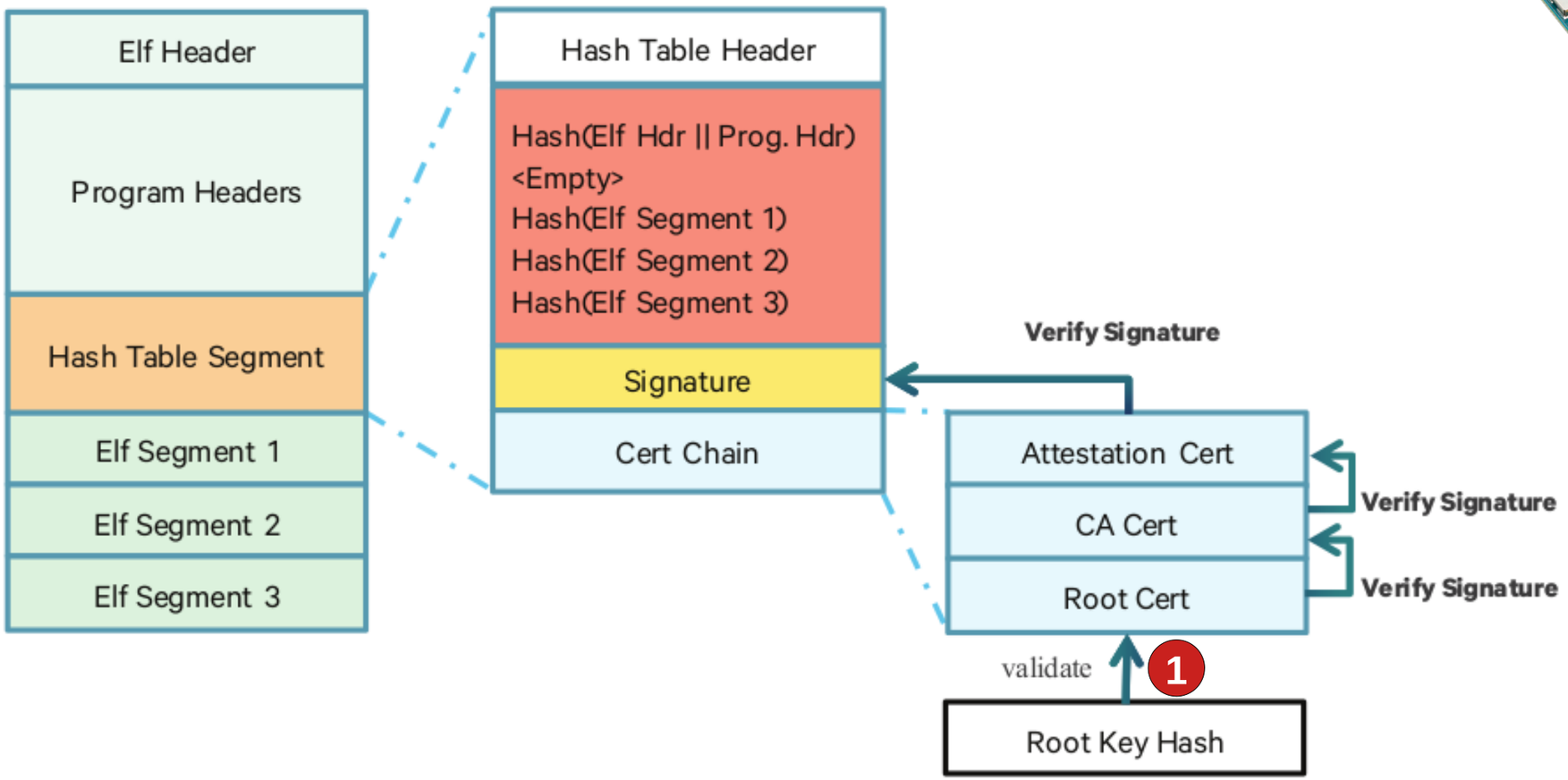




Attack Scope

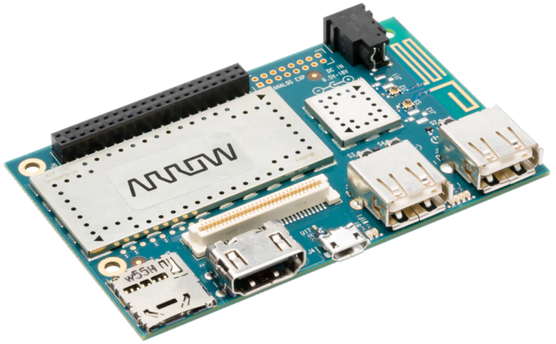
EDL Image verification

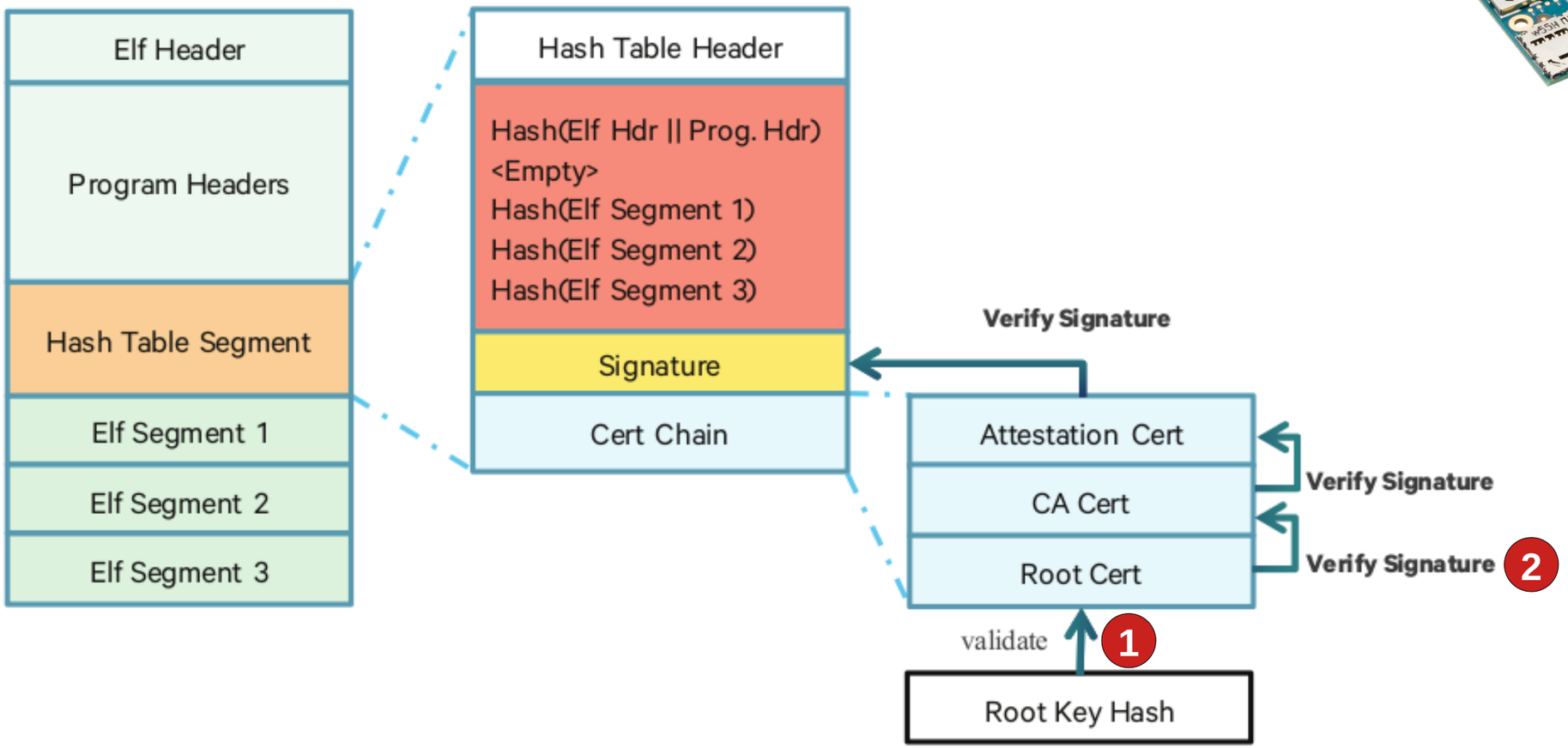




Attack Scope

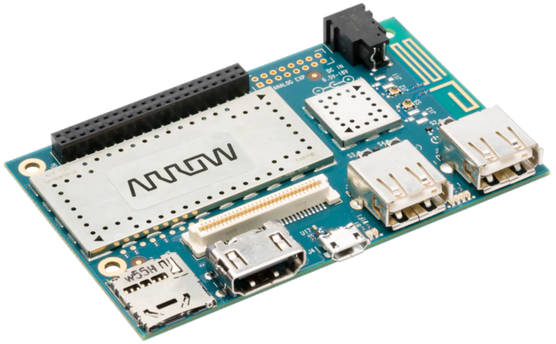
EDL Image verification

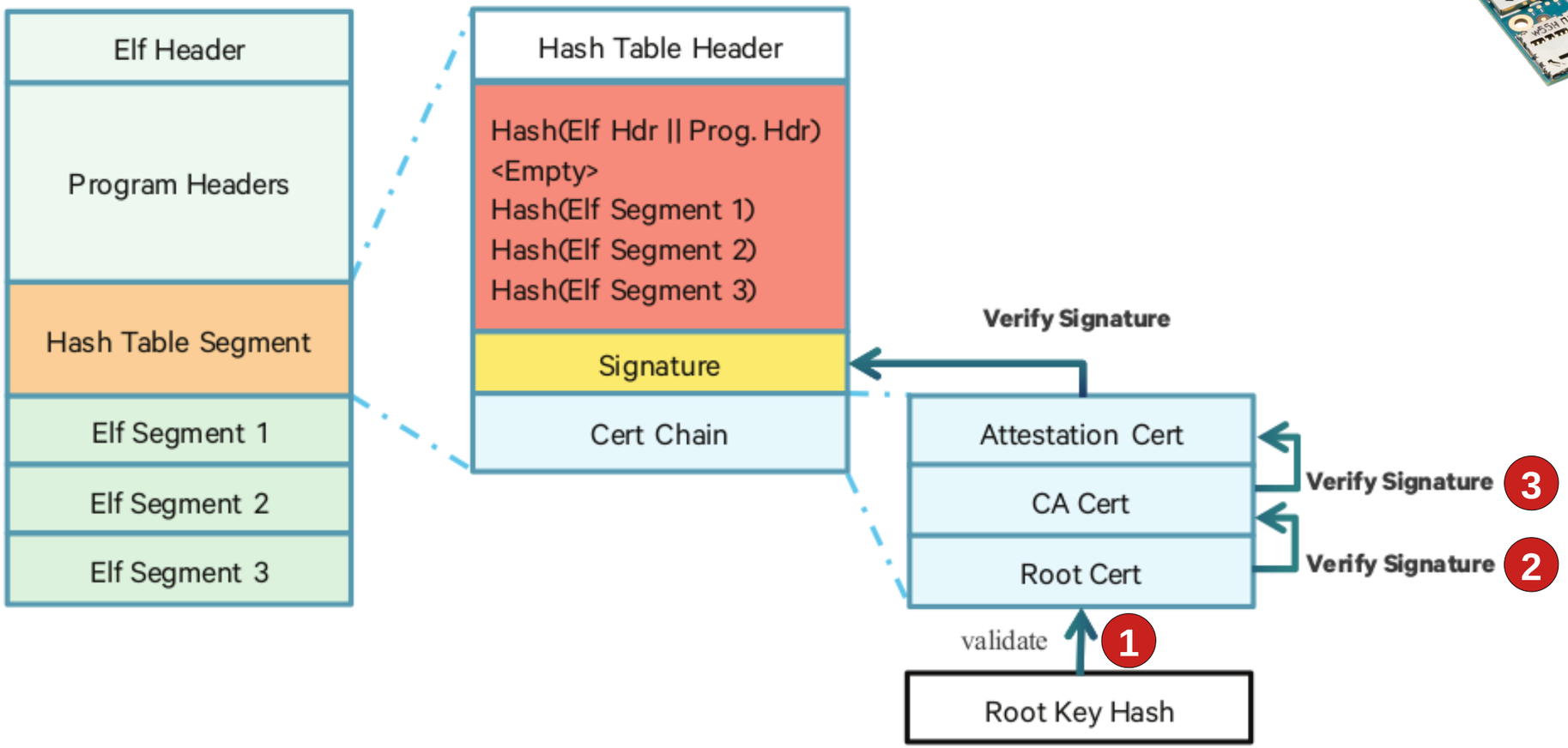




Attack Scope

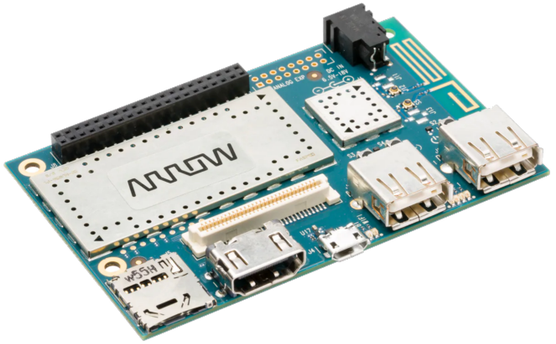
EDL Image verification

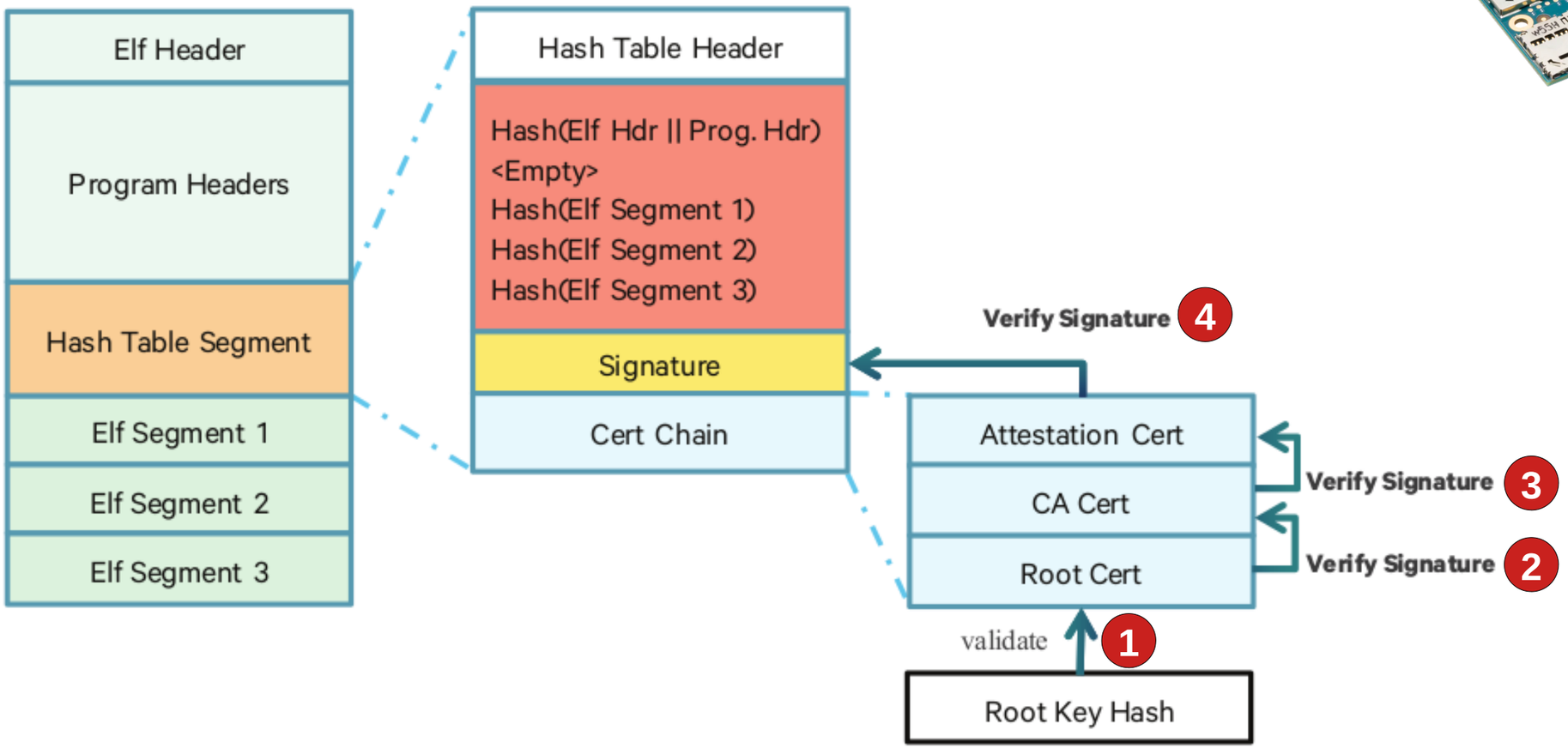




Attack Scope

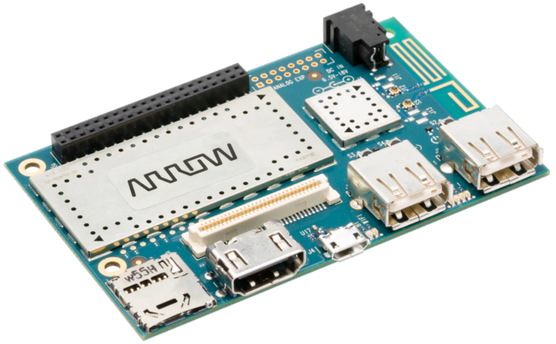
EDL Image verification

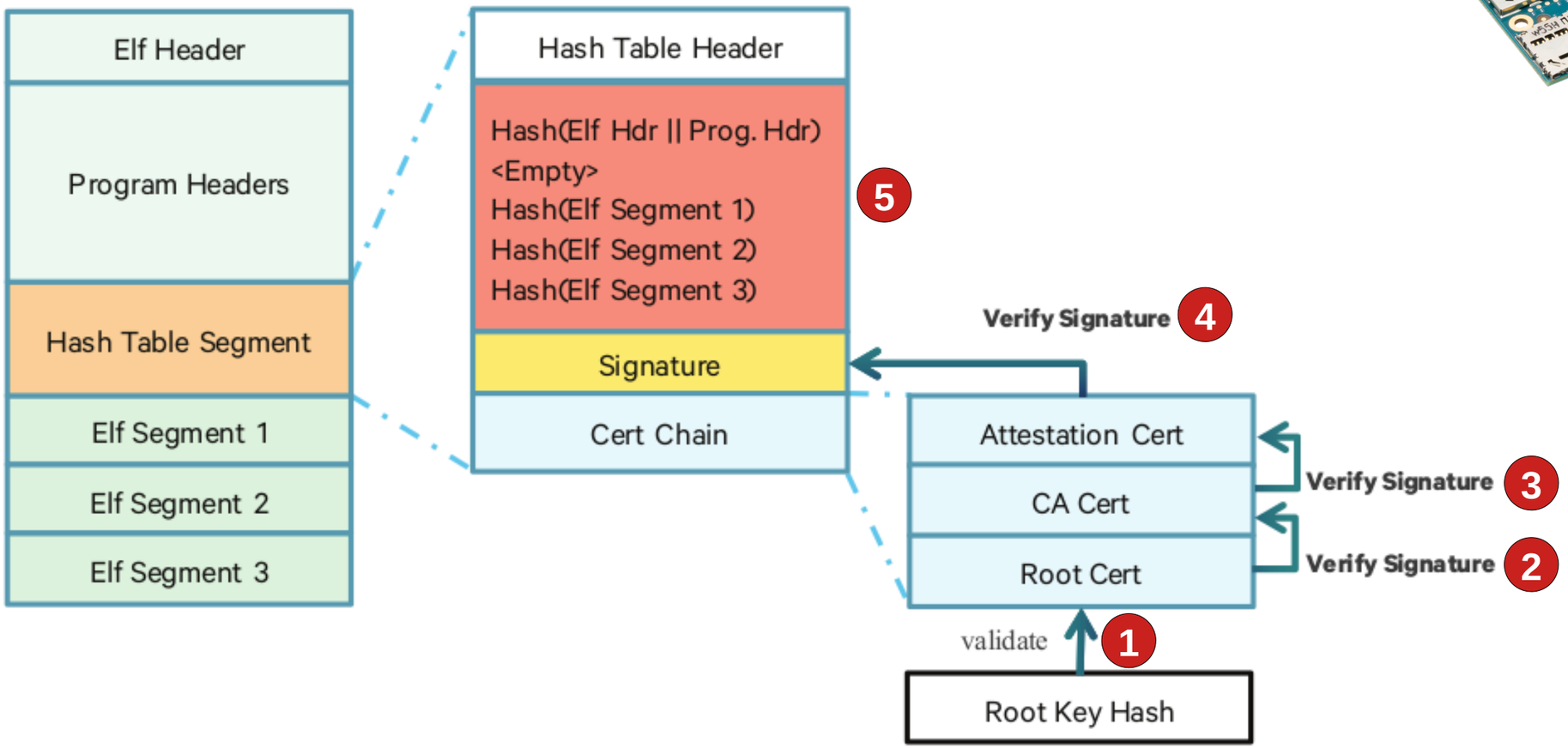




Attack Scope

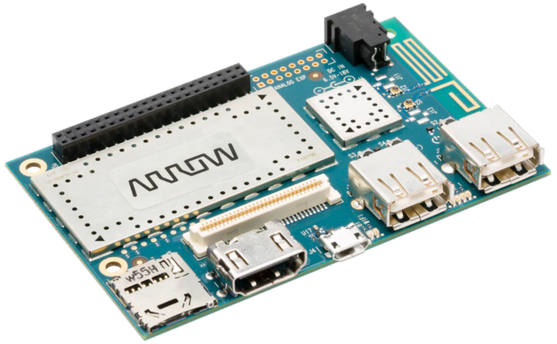
EDL Image verification

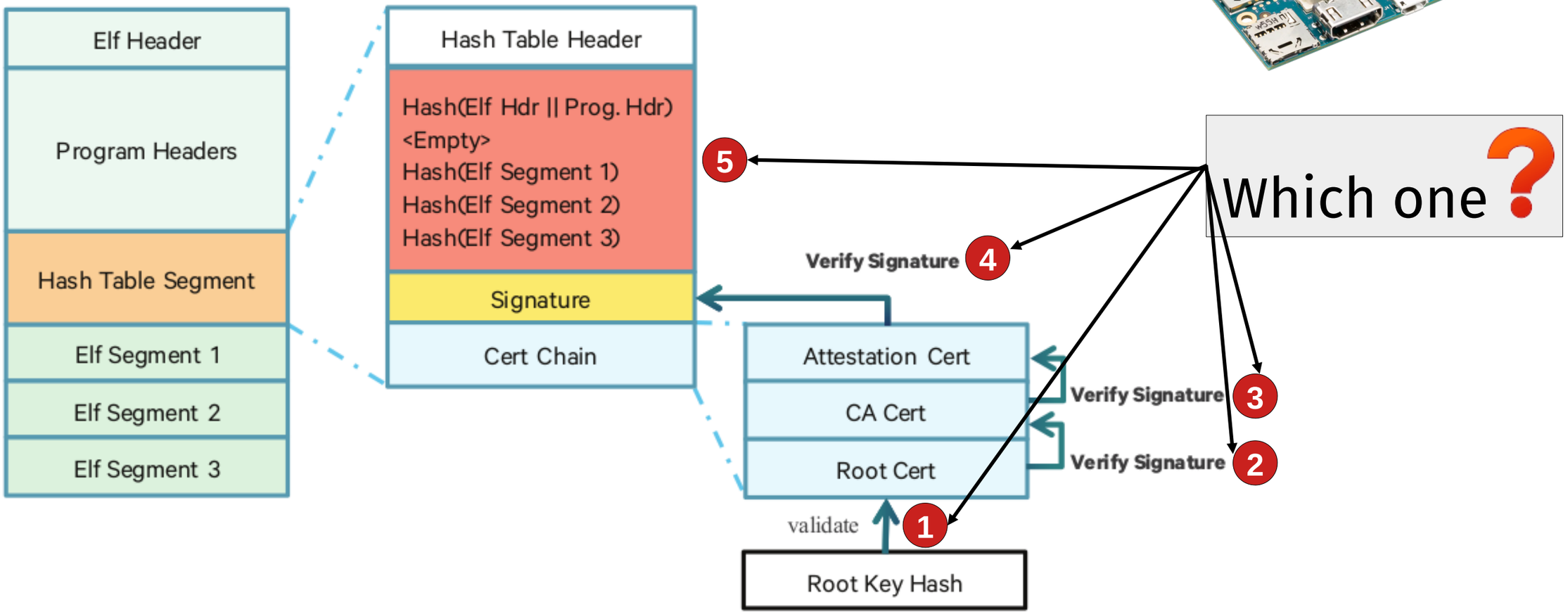




Attack Scope

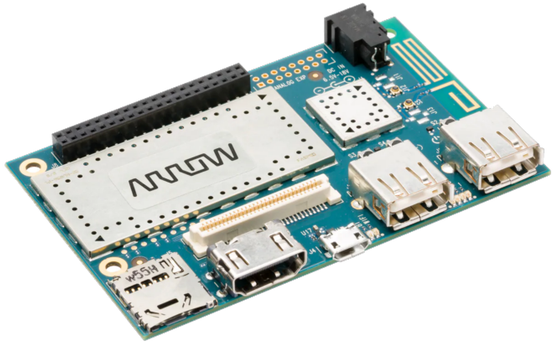
Choosing a target

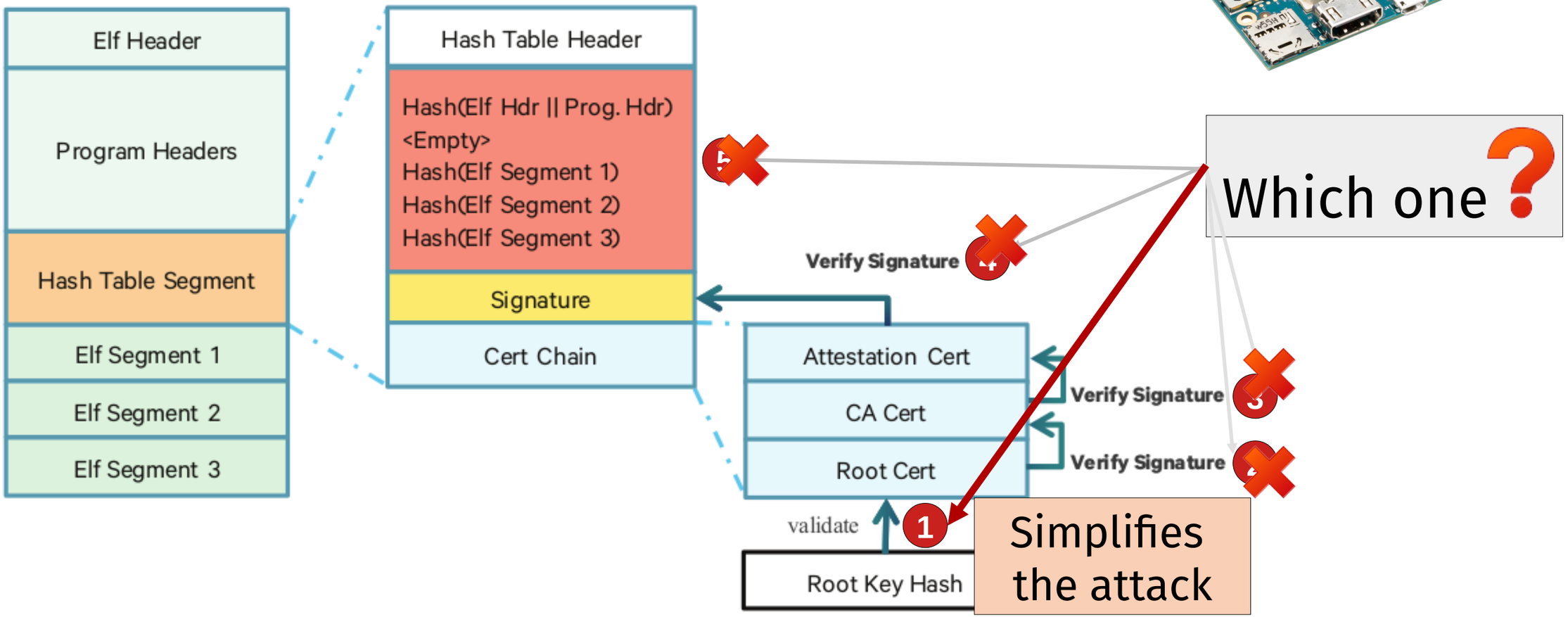




Attack Scope

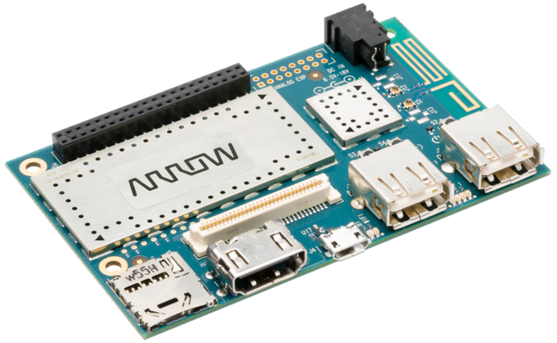
Choosing a target





Attack Scope

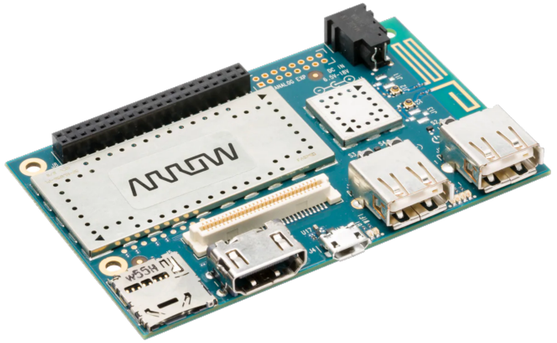
Attack Overview



- The goal is to bypass the first verification
- How we obtained the PBL in the first place to reverse it?
  - We didn’t. This is one of the goals!
  - We are blind and don’t know what we are glitching.
- How do we exactly determine the first verification then?
  - We don’t. We just guess a probably timing area.
- How we determine that area?
  - We send to the board a programmer with a valid root key and another with a wrong key.

Attack Scope

Attack Overview

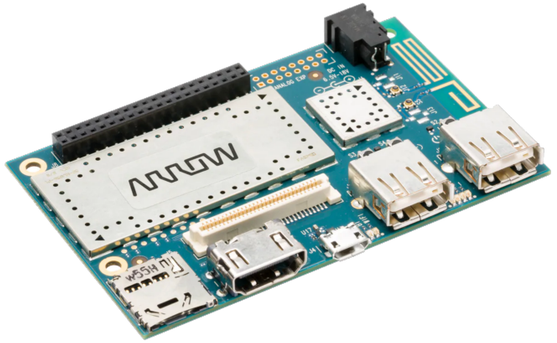


- The goal is to bypass the first verification
- How we obtained the PBL in the first place to reverse it?
  - We didn't. This is one of the goals!
  - We are blind and don't know what we are glitching.
- How do we exactly determine the first verification then?
  - We don't. We just guess a probably timing area.
- How we determine that area?
  - We send to the board a programmer with a valid root key and another with a wrong key.

Accepted

Attack Scope

Attack Overview



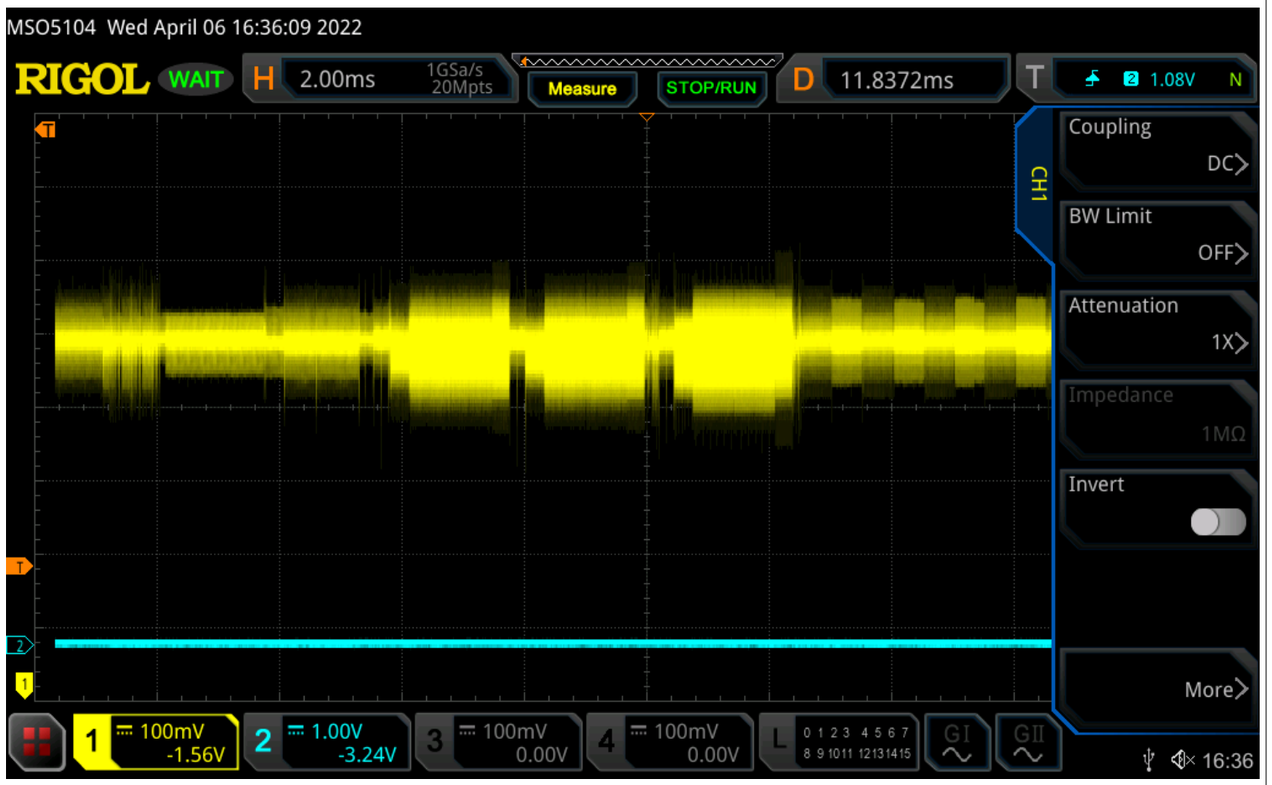
- The goal is to bypass the first verification
- How we obtained the PBL in the first place to reverse it?
  - We didn't. This is one of the goals!
  - We are blind and don't know what we are glitching.
- How do we exactly determine the first verification then?
  - We don't. We just guess a probably timing area.
- How we determine that area?
  - We send to the board a programmer with a valid root key and another with a wrong key.

Accepted

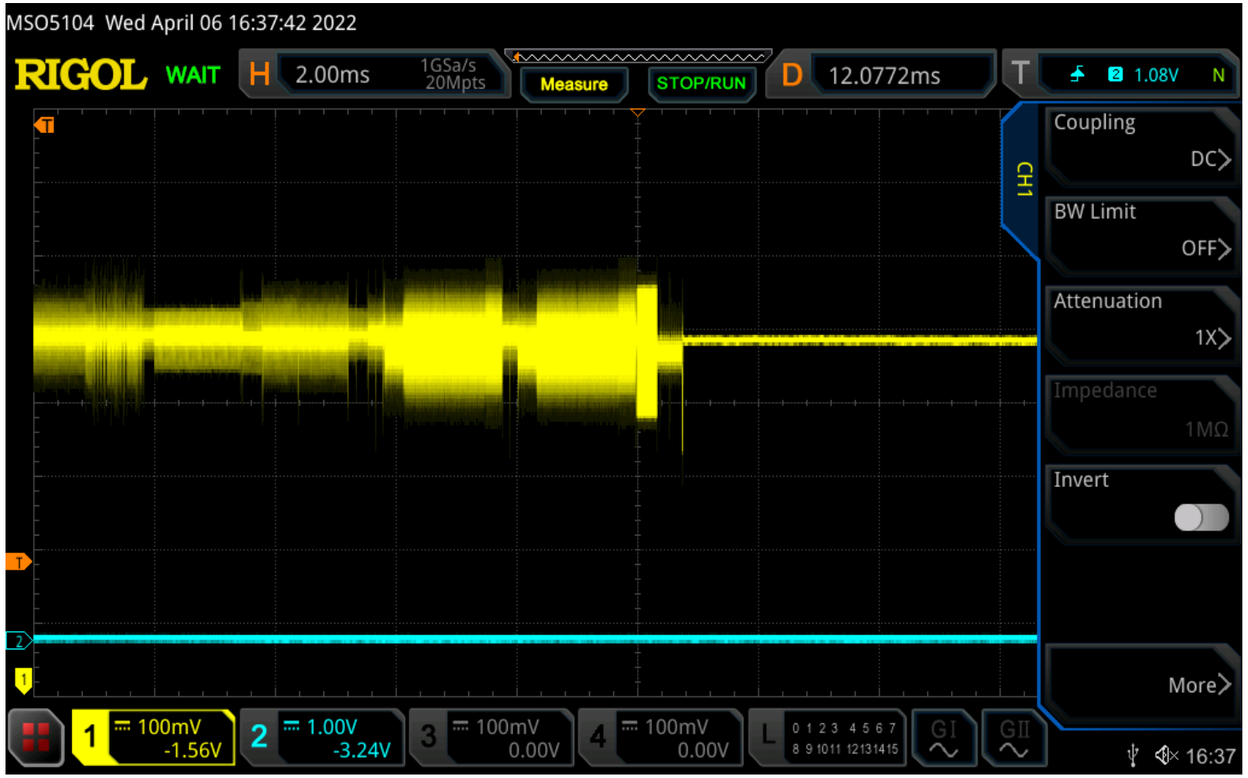
Rejected

Attack Scope

Attack Overview



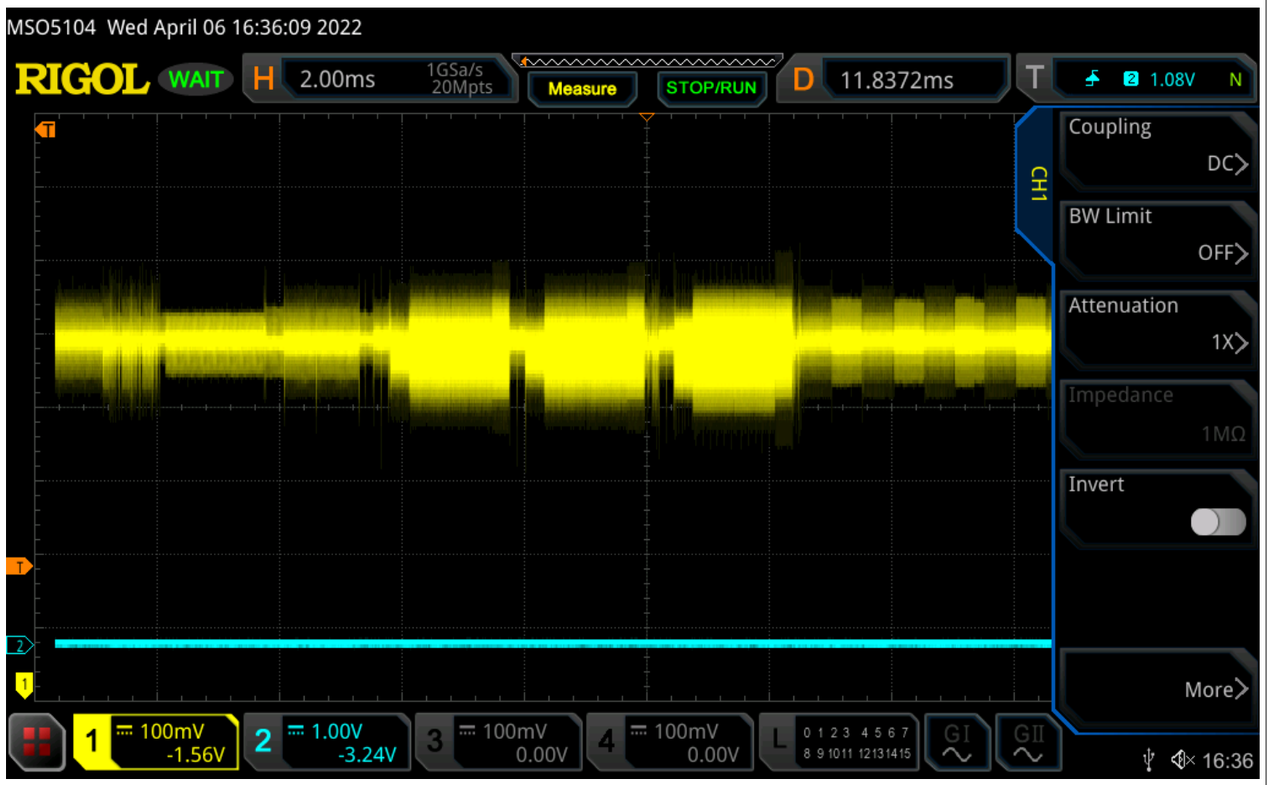
Power trace when loading a programmer with a **valid** Root Key



Power trace when loading a programmer with an **invalid** Root Key

Attack Scope

Attack Overview



Power trace when loading a programmer with a **valid** Root Key

Power trace when loading a programmer with an **invalid** Root Key

Full Attack Overview

USB Sniffer/ Trigger

Glitch generator

Power Supply

7v

GPIO

Full Attack Overview

1 Send a crafted `programmer.elf` using **`./edl`**

USB Sniffer/ Trigger

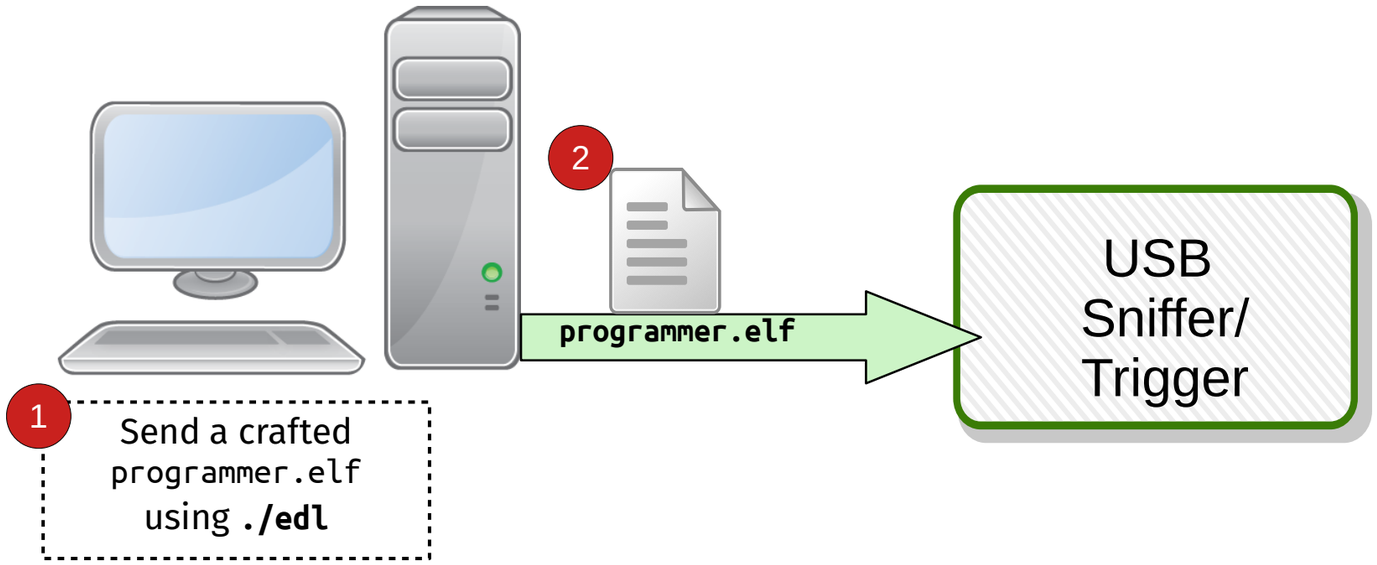
Glitch generator

Power Supply

7v

GPIO

Full Attack Overview



USB Sniffer/ Trigger

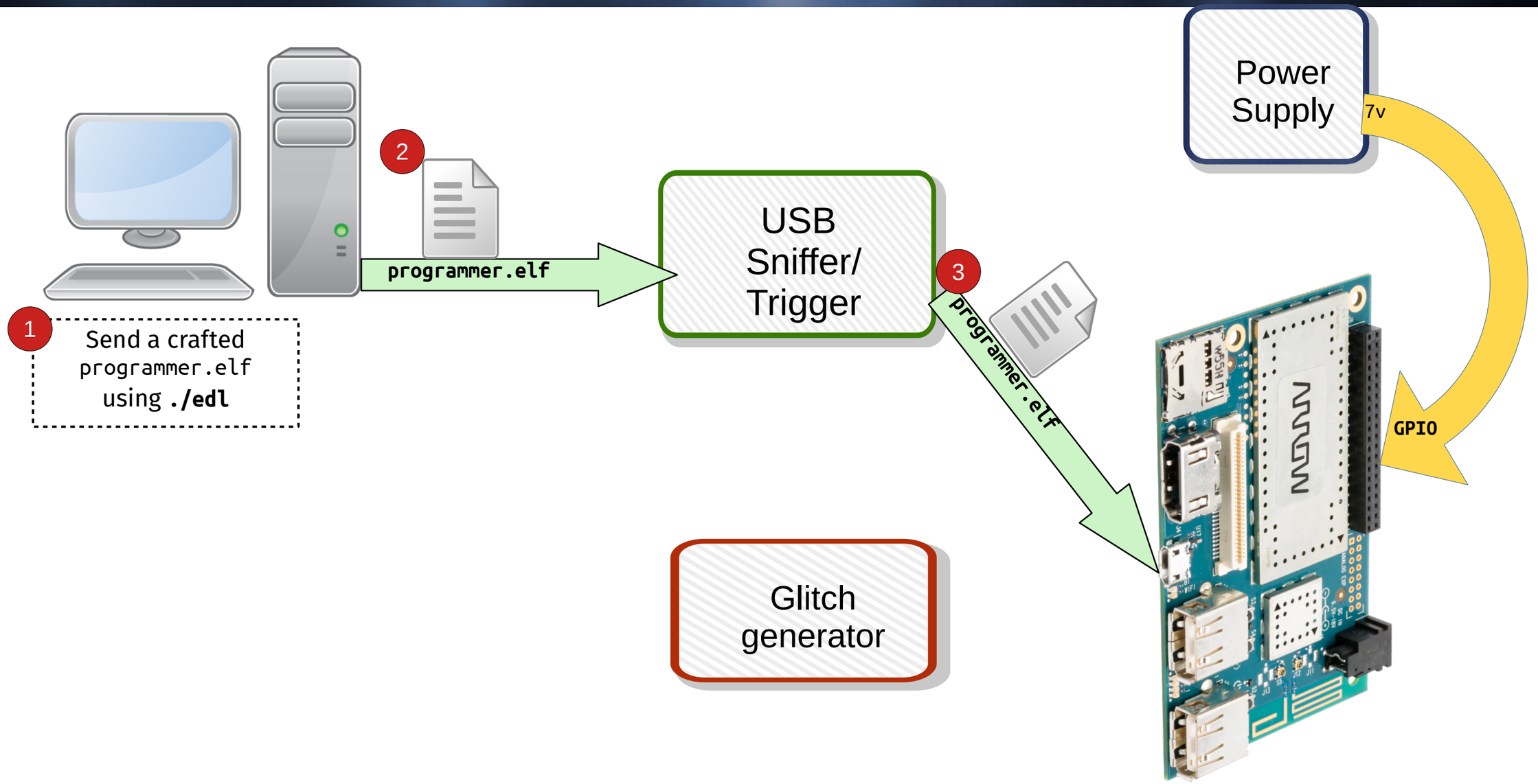
Glitch generator

Power Supply

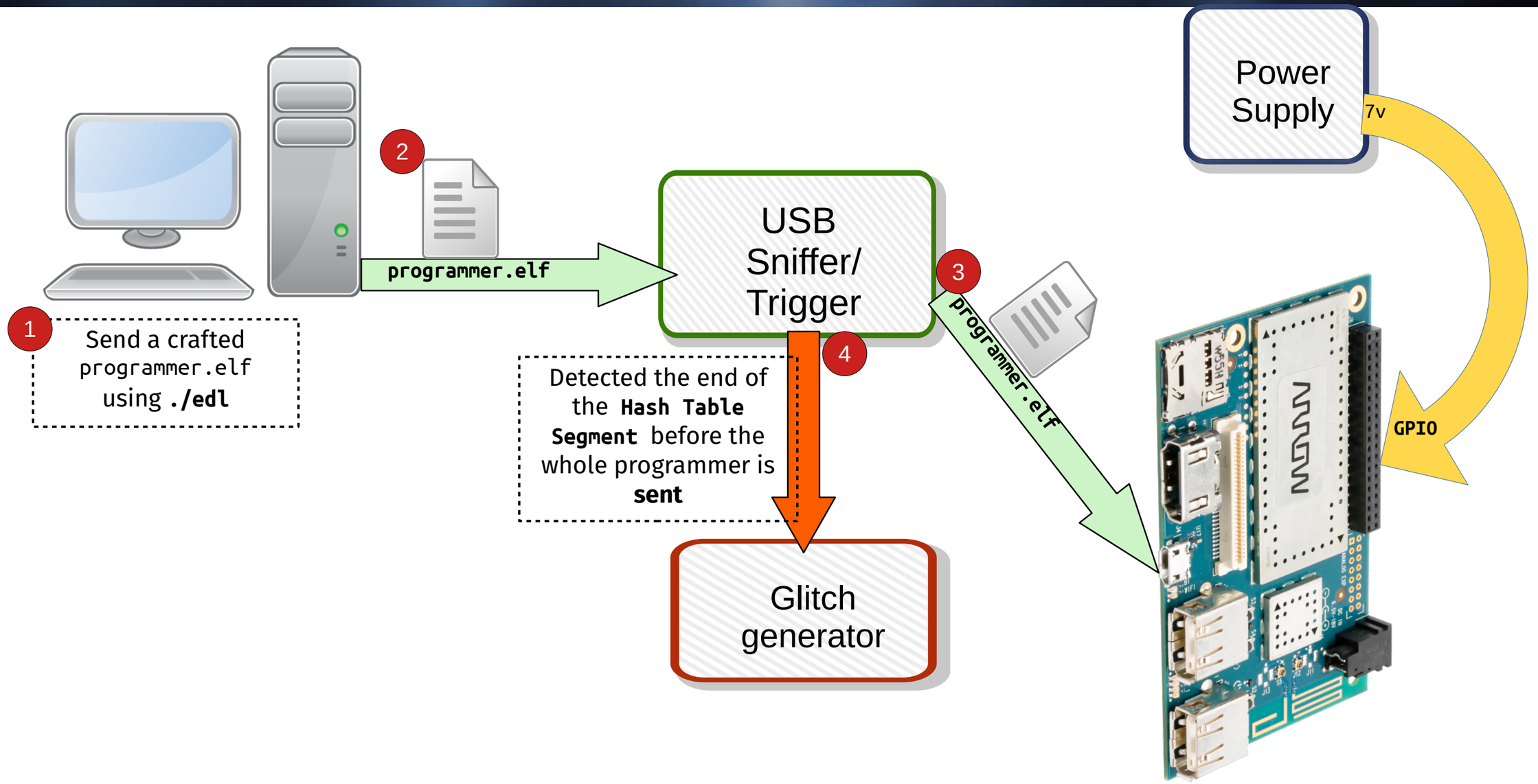
7v

GPIO

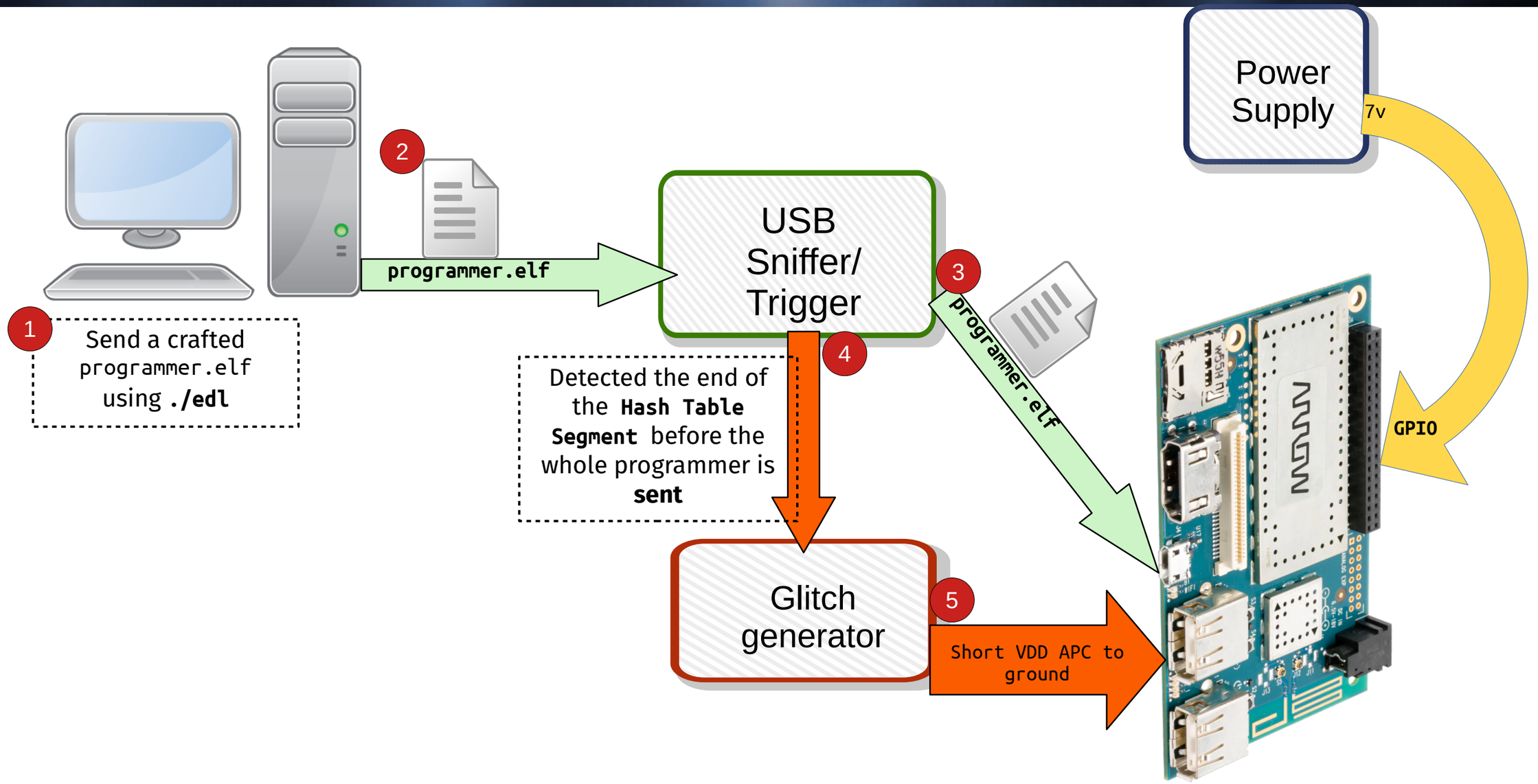
Full Attack Overview



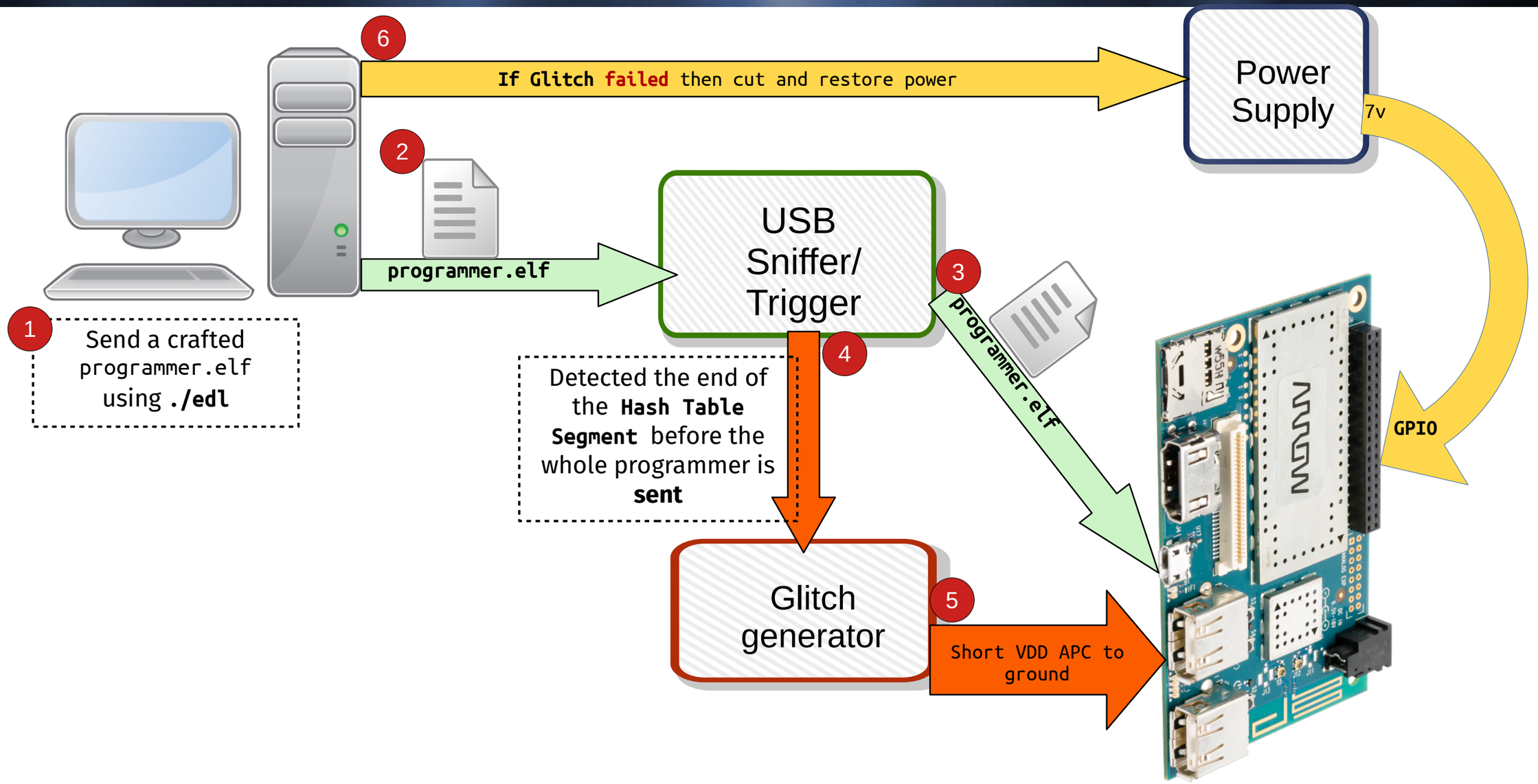
Full Attack Overview



Full Attack Overview

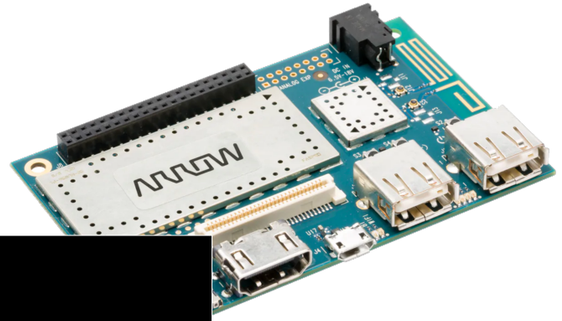


Full Attack Overview



Vlind Glitch

Equipment



Equipment	Functionality
PC	Main control.
NewAE Chip Whisperer Nano	FPGA board. Controls glitching MOSFET.
TotalPhase Beagle USB 5000	USB sniffing. Triggers Chip Whisperer.
Relay	Resets device under test.
DragonBoard 410c	Device under test.

Vlind Glitch

Enabling Secure Boot

- With Qualcomm-provided tools.

https://developer.qualcomm.com/hardware/dragonboard-410c/softw 120%

Qualcomm developer network

Solutions Software **Hardware** Downloads Forums Community About Us

Hardware › App Processors & Platforms › DragonBoard 410c Development Board › DragonBoard 410c Software

DragonBoard 410c Development Board

- Tutorial Videos
- Kits & Accessories
- **Software**
- Forum
- Projects

Software

Use the software, documents and video tutorial resources below to jump start your development with Snapdragon® 410E for embedded computing and the DragonBoard™ 410c by Arrow Electronics.

Forum support for the DragonBoard 410c is available at 96boards.org.

Windows 10 IoT Core

Windows 10 IoT Core Board Support Package

View License Agreement

Windows 10 IoT BSP for DragonBoard 410c Customization Guide

Vlind Glitch

Enabling Secure Boot

- With Qualcomm-provided tools.

https://developer.qualcomm.com/hardware/dragonboard-410c/softw 120%

Qualcomm developer network

Solutions Software **Hardware** Downloads Forums Community About Us

Hardware › App Processors & Platforms › DragonBoard 410c Development Board › DragonBoard 410c Software

DragonBoard 410c Development Board

- Tutorial Videos
- Kits & Accessories
- **Software**
- Forum
- Projects

Software

Use the software, documents and video tutorial resources below to jump start your development with Snapdragon® 410E for embedded computing and the DragonBoard™ 410c by Arrow Electronics.

Forum support for the DragonBoard 410c is available at 96boards.org.

Windows 10 IoT Core

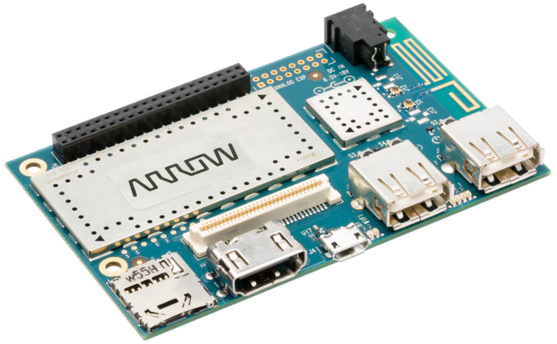
Windows 10 IoT Core Board Support Package → `sectools`

View License Agreement

Windows 10 IoT BSP for DragonBoard 410c Customization Guide → **Chapter 11** `Secure Boot enablement`

Vlind Glitch

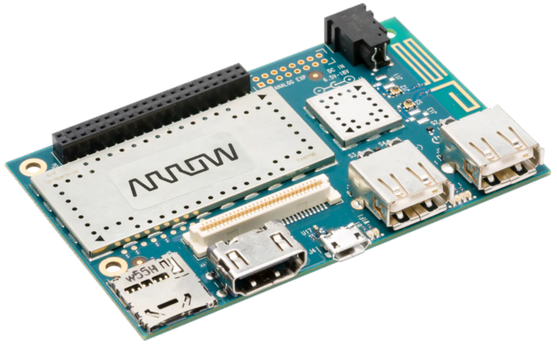
Enabling Secure Boot



- We generated our own keys.
- In some models modification of configuration files is required.

Vlind Glitch

Enabling Secure Boot

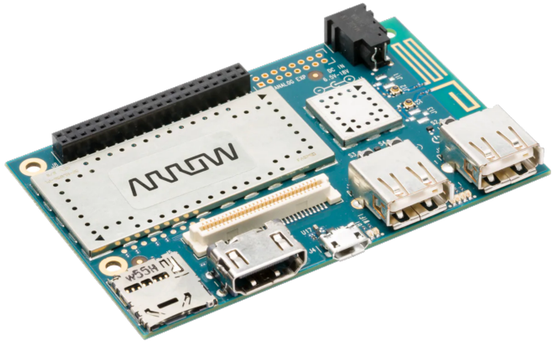


- We generated our own keys.
- In some models modification of configuration files is required.

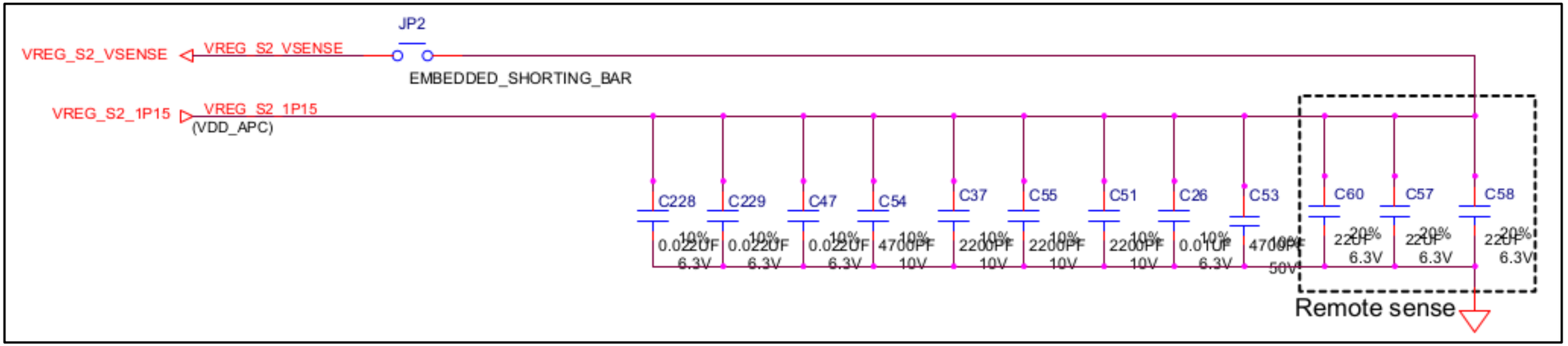
```
v@v-HP-Desktop-M01-F1xxx:~/edl$ ./edl info
Qualcomm Sahara / Firehose Client V3.53 (c) B.Kerler 2018-2021.
main - Using loader /home/v/ws/ci-repos/fault-injection/dragonboard-410c/programmers/
main - Waiting for the device
main - Device detected :)
main - Mode detected: sahara
Device is in EDL mode .. continuing.
sahara -
-----------------------
HWID:                0x007060e100000005 (MSM_ID:0x007060e1,OEM_ID:0x0000,MODEL_ID:0x000
CPU detected:        "APQ8016"
PK_HASH:             0x58677bf07e4b274b17d4e75cc44dbebbc9406c5c4cc9568a9ce1ab28d25be498
Serial:              0x1c5846be
```

Vlind Glitch

Soldering to the Chip Whisperer

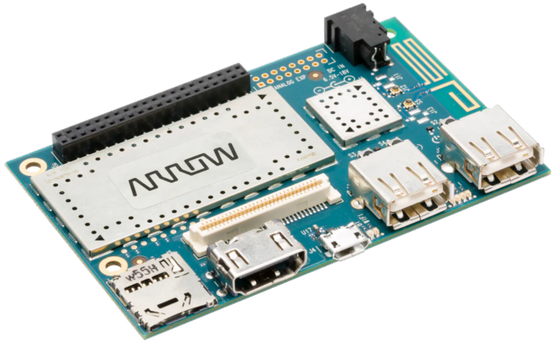


- We removed a capacitor from the Application Processor Core (APC) rail.
- Direct access to the power rail.

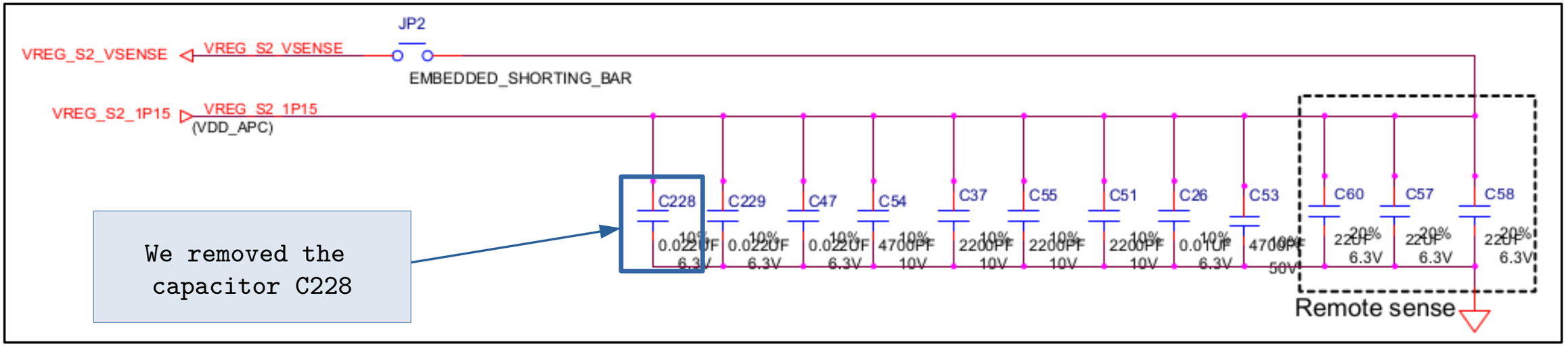


Vlind Glitch

Soldering to the Chip Whisperer

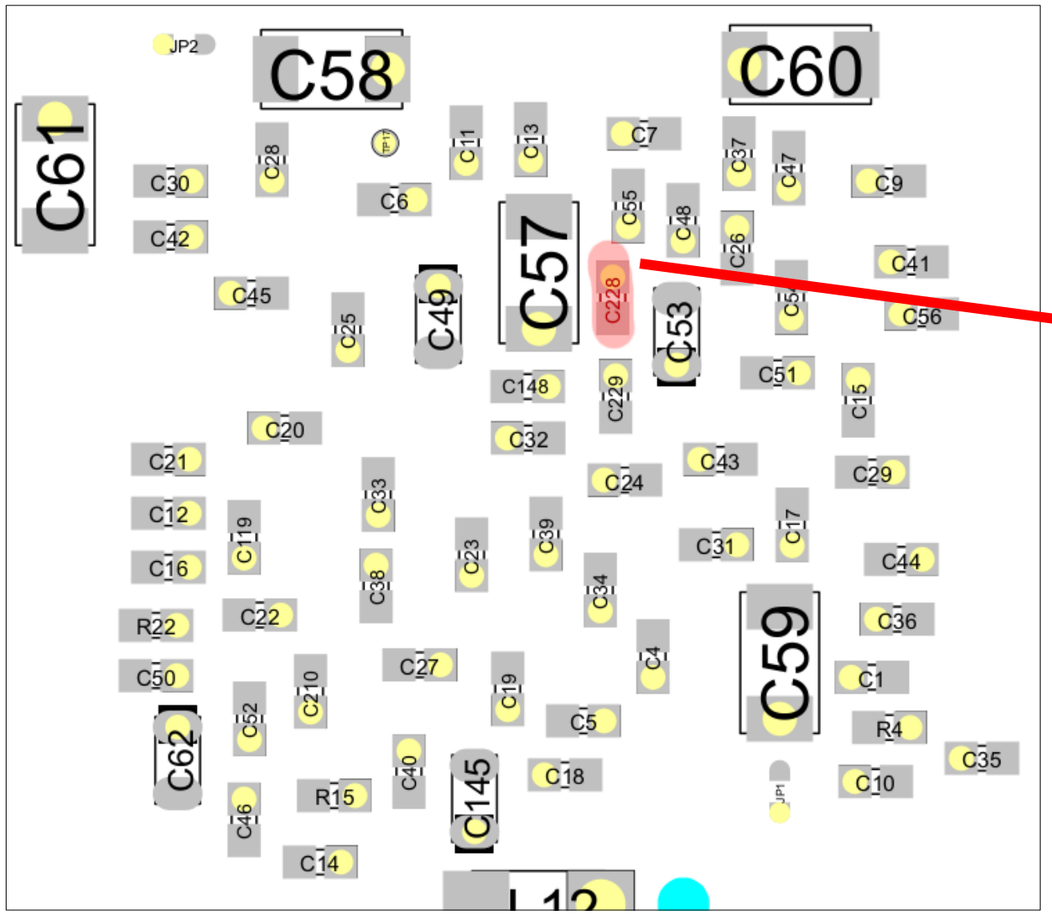


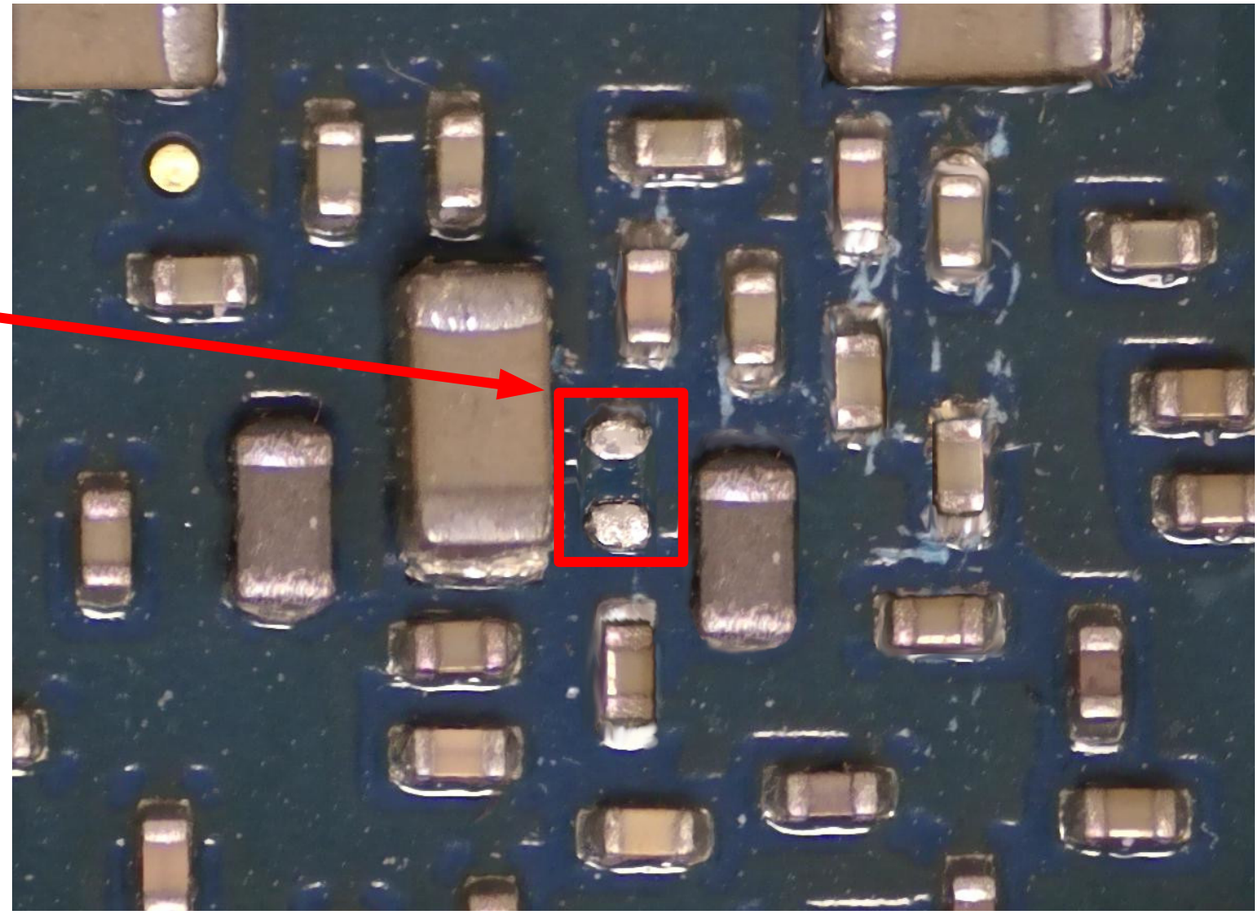
- We removed a capacitor from the Application Processor Core (APC) rail.
- Direct access to the power rail.



Vlind Glitch

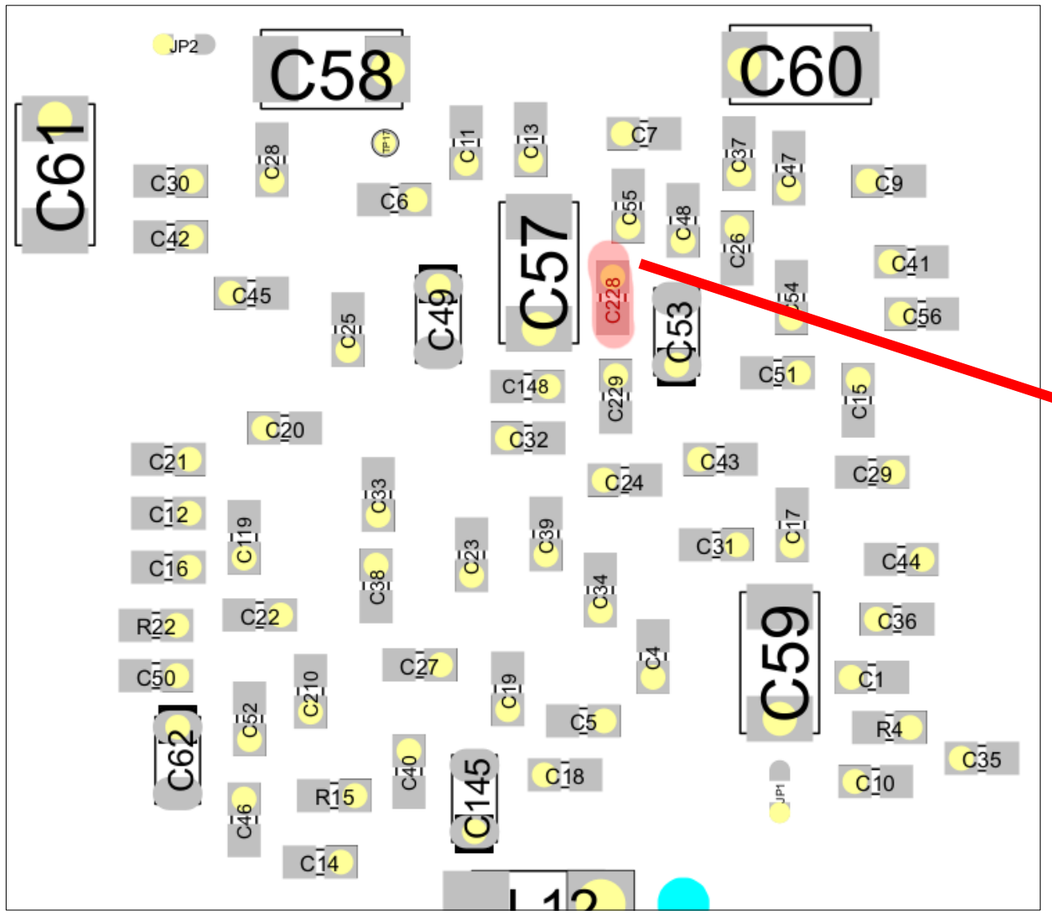
Soldering to the Chip Whisperer

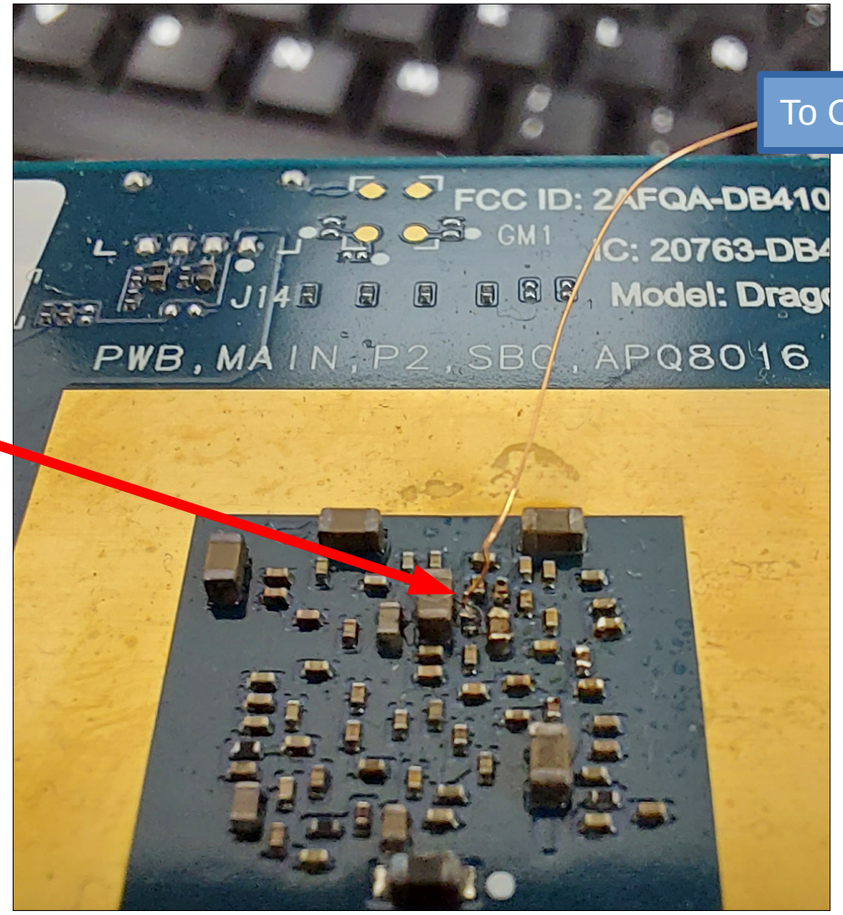




Vlind Glitch

Soldering to the Chip Whisperer

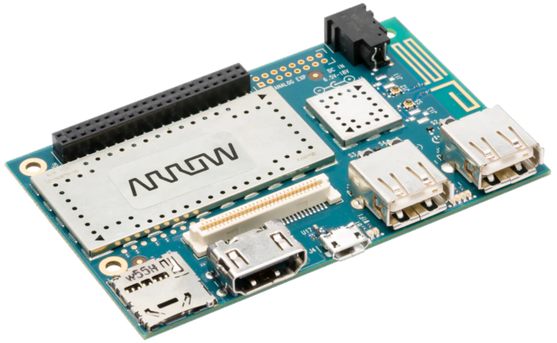




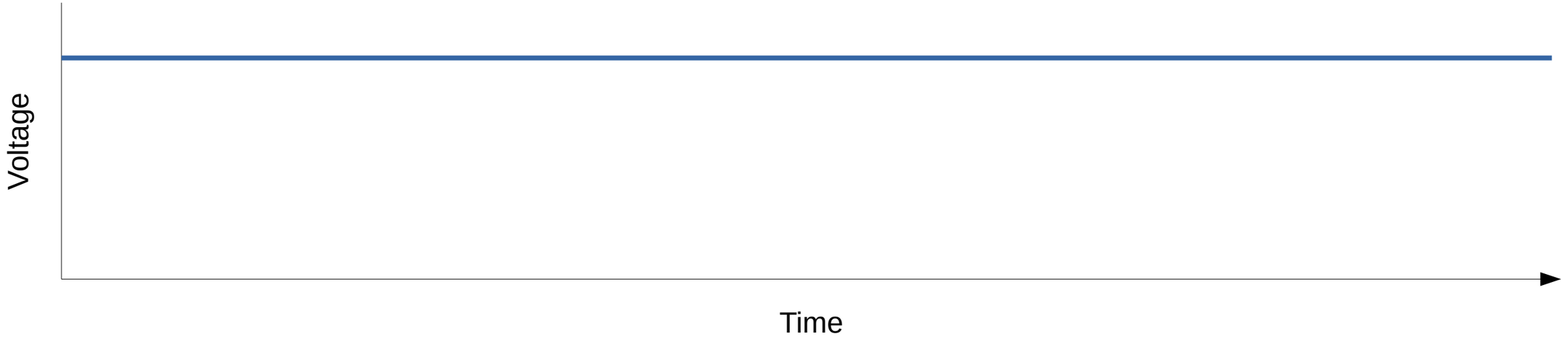
To Chip Whisperer

Vlind Glitch

Characterising the DragonBoard

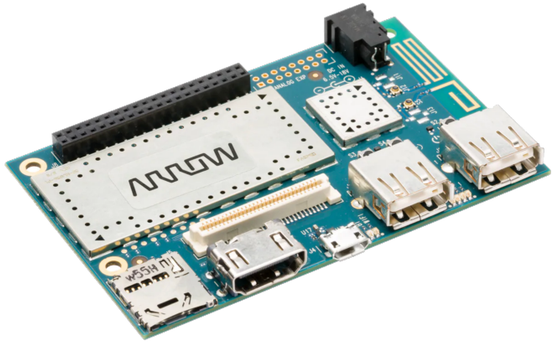


- Learn how different glitch parameters affect the device:
  - The minimum glitch pulse duration to influence the DragonBoard.
  - The maximum glitch pulse duration that does not restart the DragonBoard.

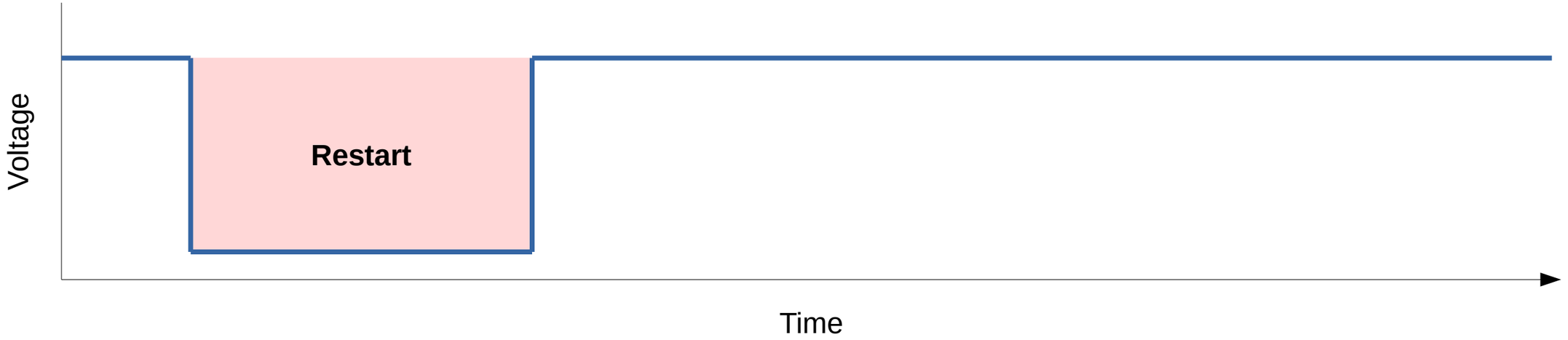


Vlind Glitch

Characterising the DragonBoard

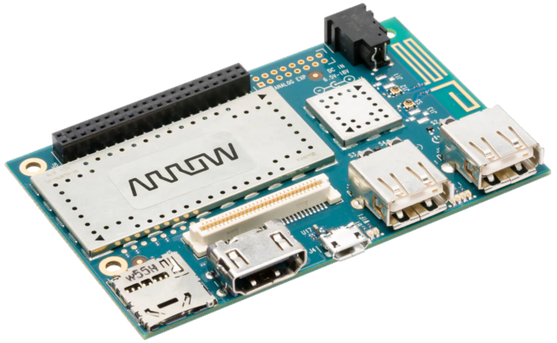


- Learn how different glitch parameters affect the device:
  - The minimum glitch pulse duration to influence the DragonBoard.
  - The maximum glitch pulse duration that does not restart the DragonBoard.

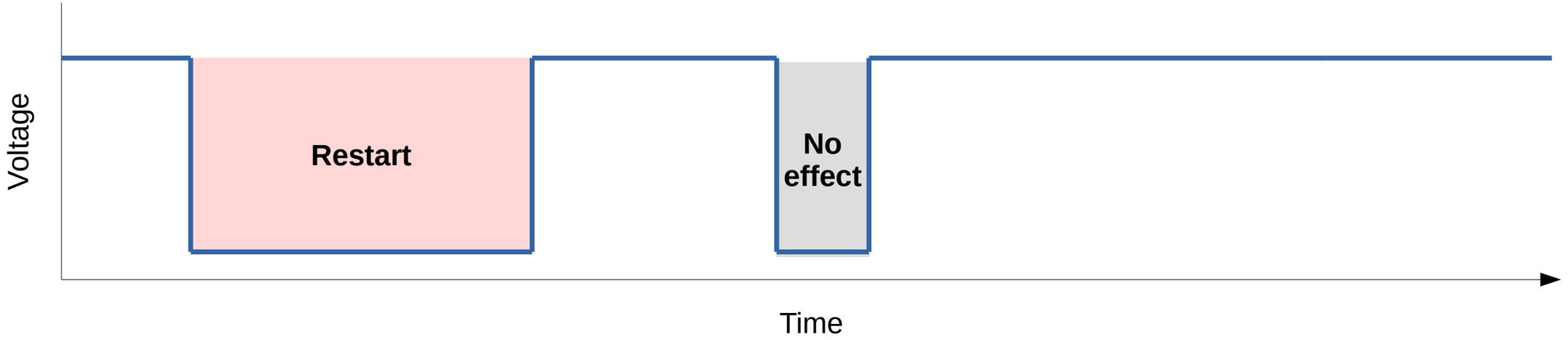


Vlind Glitch

Characterising the DragonBoard

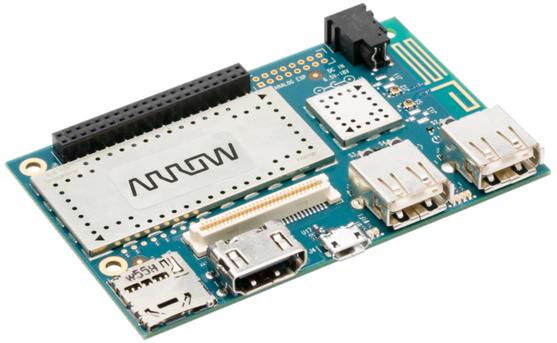


- Learn how different glitch parameters affect the device:
  - The minimum glitch pulse duration to influence the DragonBoard.
  - The maximum glitch pulse duration that does not restart the DragonBoard.

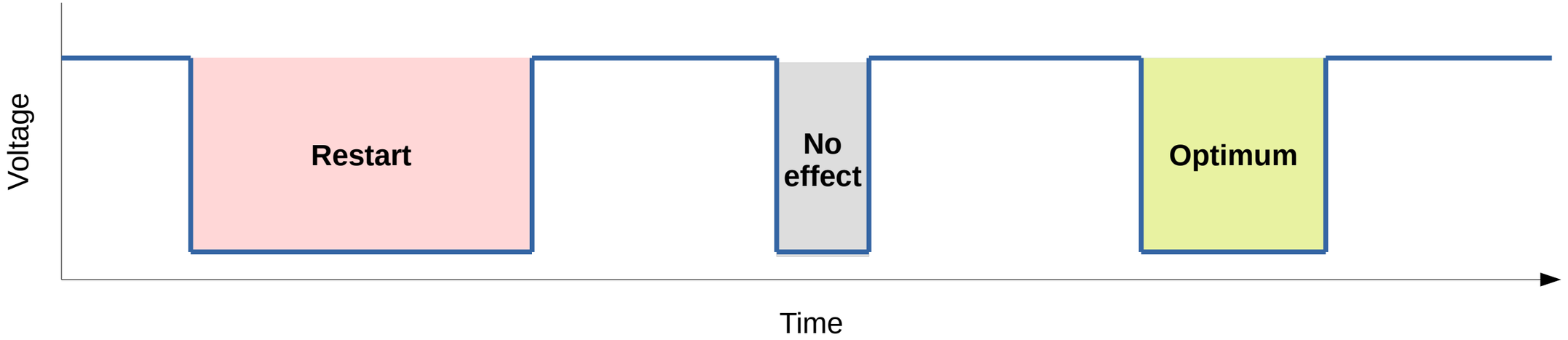


Vlind Glitch

Characterising the DragonBoard

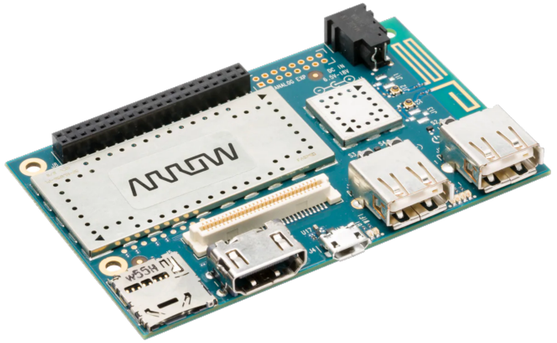


- Learn how different glitch parameters affect the device:
  - The minimum glitch pulse duration to influence the DragonBoard.
  - The maximum glitch pulse duration that does not restart the DragonBoard.



Vlind Glitch

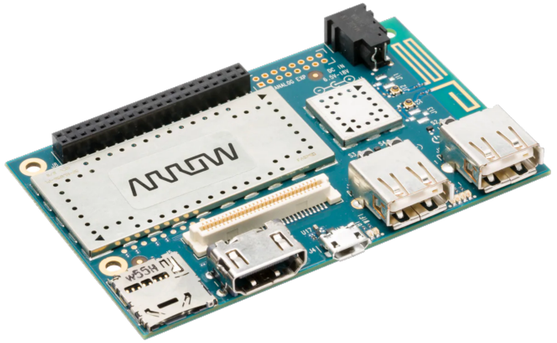
Characterising the DragonBoard



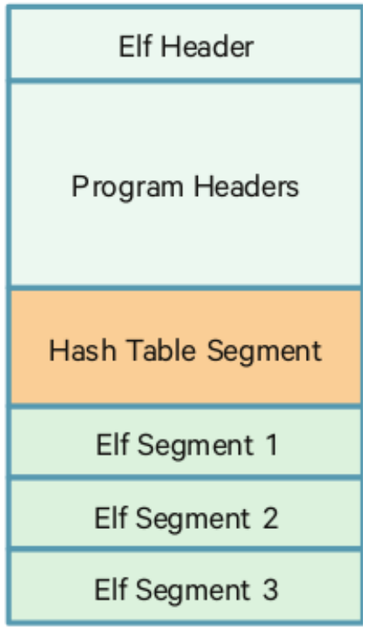
- We have the keys → We can sign modified programmers.
  - We replaced a programmer function with our characterisation code.
  - We sent the programmer to the board via EDL.
  - This code is executed when a specific programmer command is received by the board.
    - In our case **`firmwarewrite()`**

Vlind Glitch

Characterising the DragonBoard

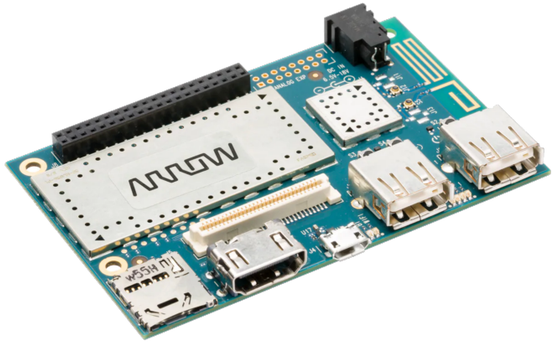


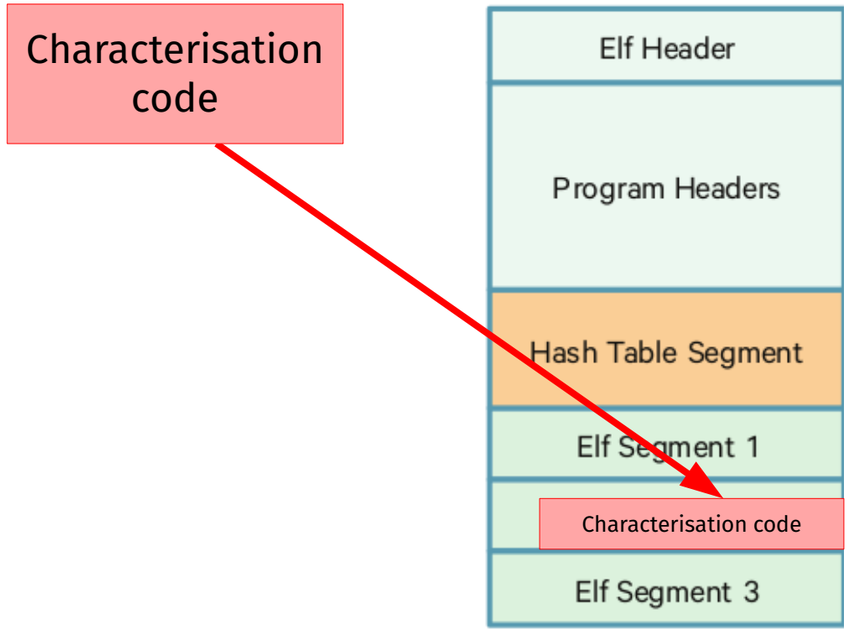
Characterisation code



Vlind Glitch

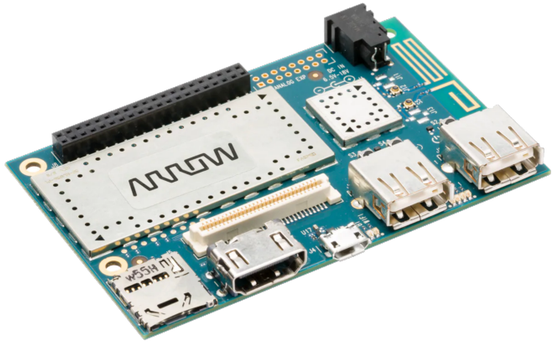
Characterising the DragonBoard

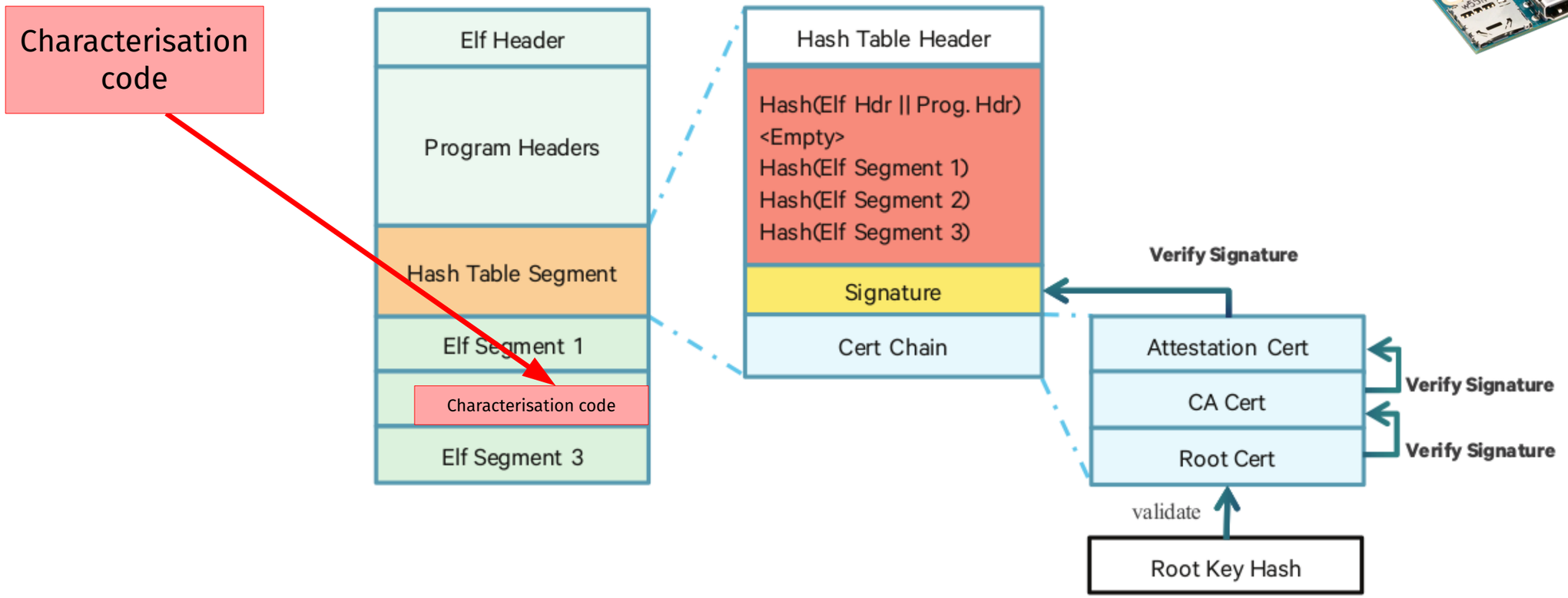




Vlind Glitch

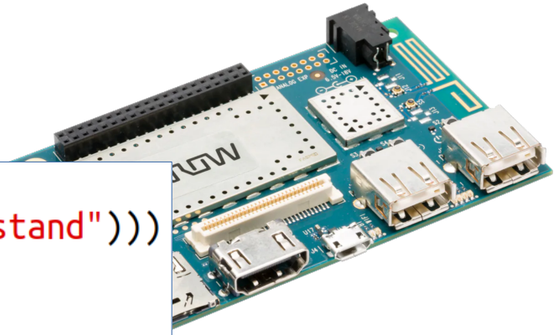
Characterising the DragonBoard

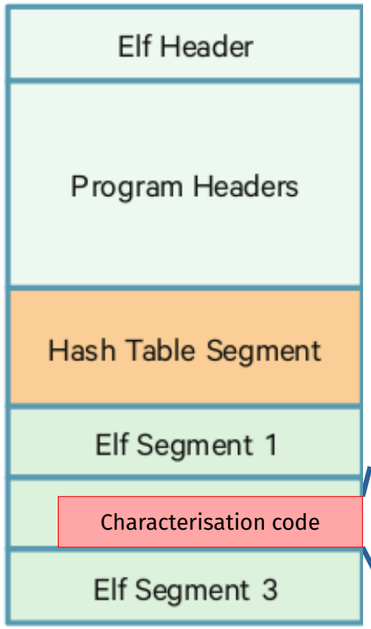




Vlind Glitch

Characterising the DragonBoard

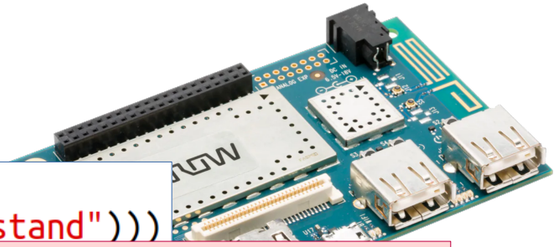




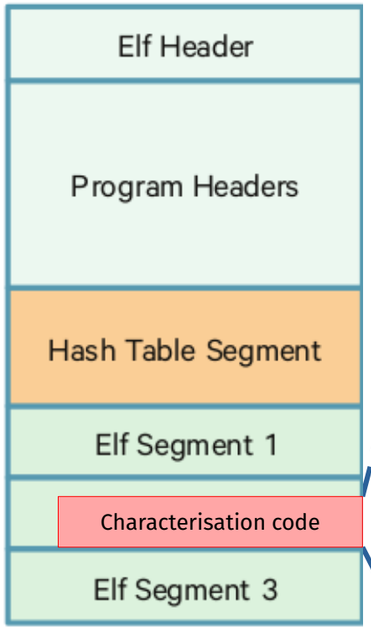

```c
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

Vlind Glitch

Characterising the DragonBoard



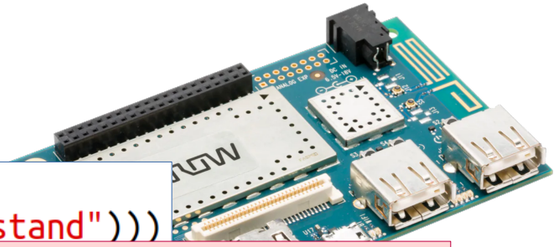
Programmer modified function.

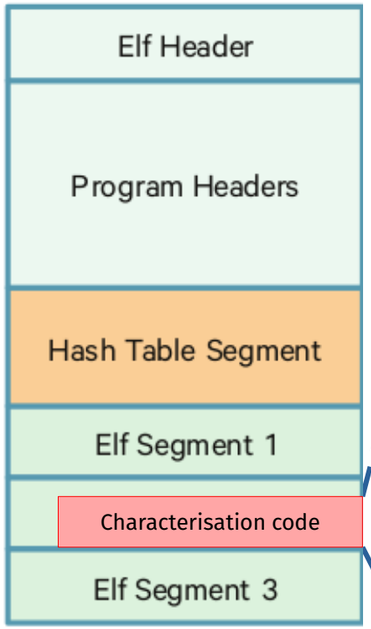



```
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

Vlind Glitch

Characterising the DragonBoard





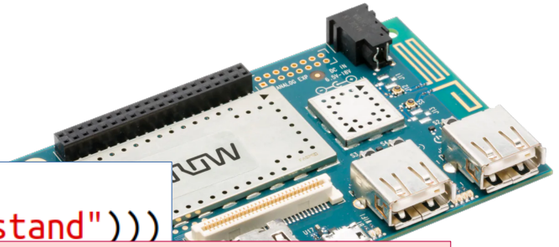

```
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

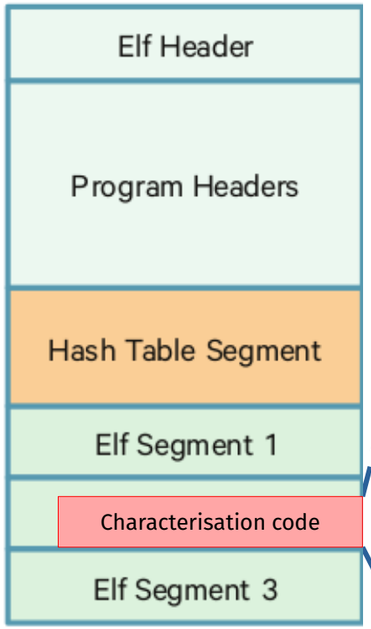
Programmer modified function.

Characterisation trigger high.

Vlind Glitch

Characterising the DragonBoard






```
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

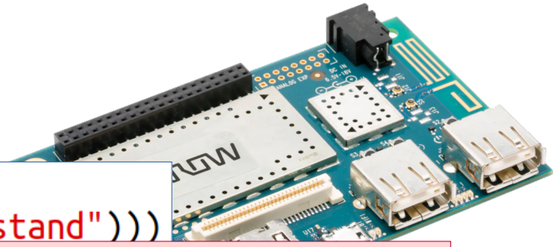
Programmer modified function.

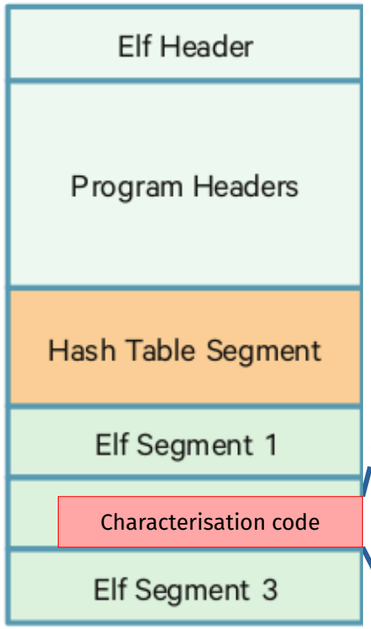
Characterisation trigger high.

Long operation.

Vlind Glitch

Characterising the DragonBoard






```
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

Programmer modified function.

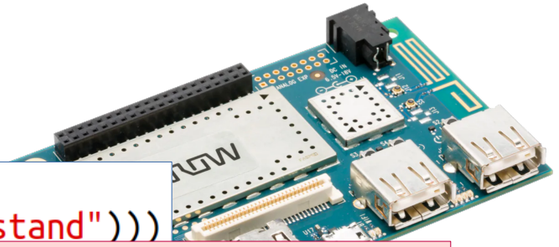
Characterisation trigger high.

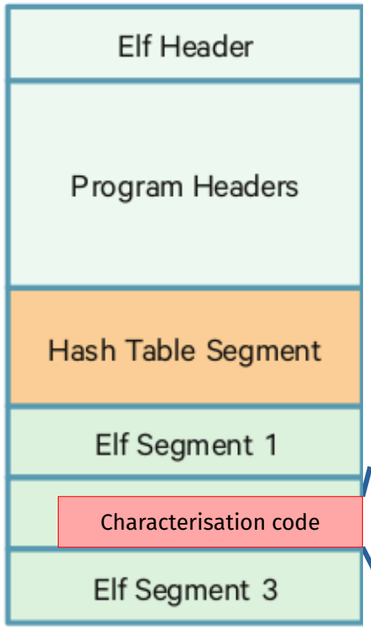
Long operation.

Characterisation trigger low.

Vlind Glitch

Characterising the DragonBoard






```
static int firmwarewrite(void)
        __attribute__((noinline)) __attribute__((section(".text.stand")))
{
        int i, j;
        // '+1' since we are in Thumb mode.
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);

        // Set GPIO_12 high, this will be our trigger
        trigger_high();
        for(i = 0; i<50000; i++){
                for(j = 0; j<50000; j++){
                        i += j;
                }
        }
        // Deassert GPIO_12
        trigger_low();

        if(i == 1249975001)
                usb_log("Expected.");
        else
                usb_log("Success!");

        return 0;
}
```

Programmer modified function.

Characterisation trigger high.

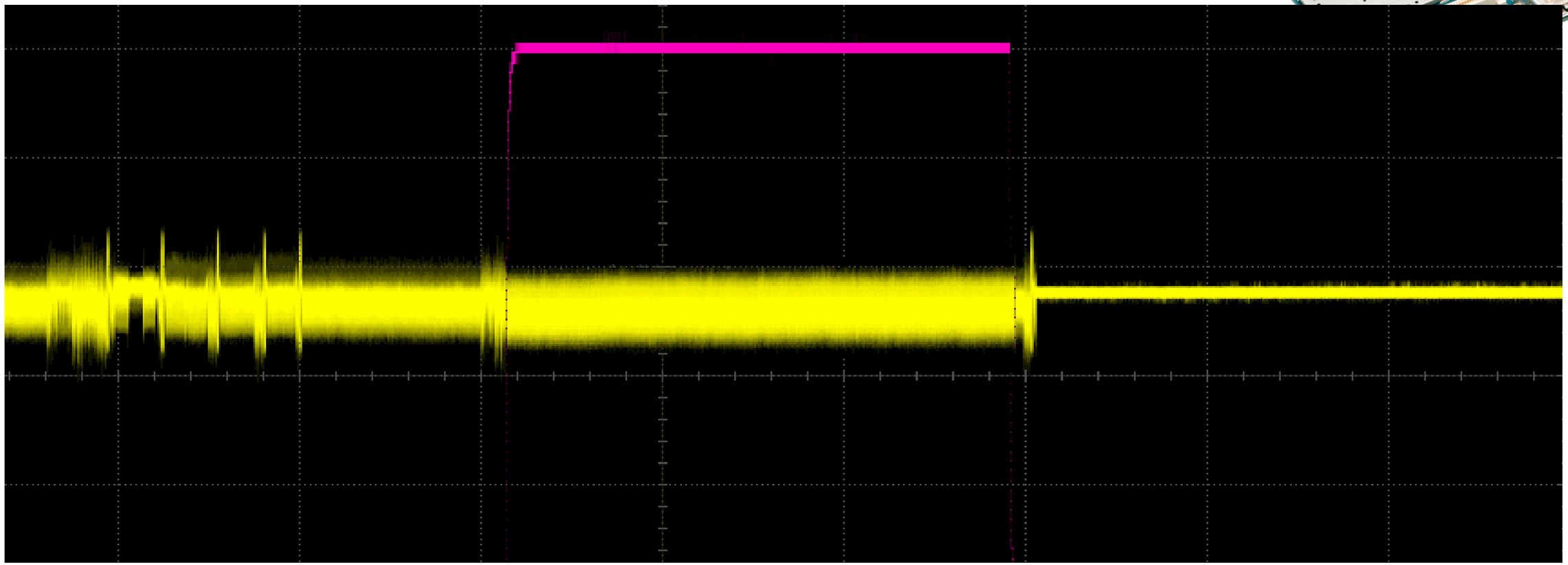
Long operation.

Characterisation trigger low.

Always "Expected" unless affected.

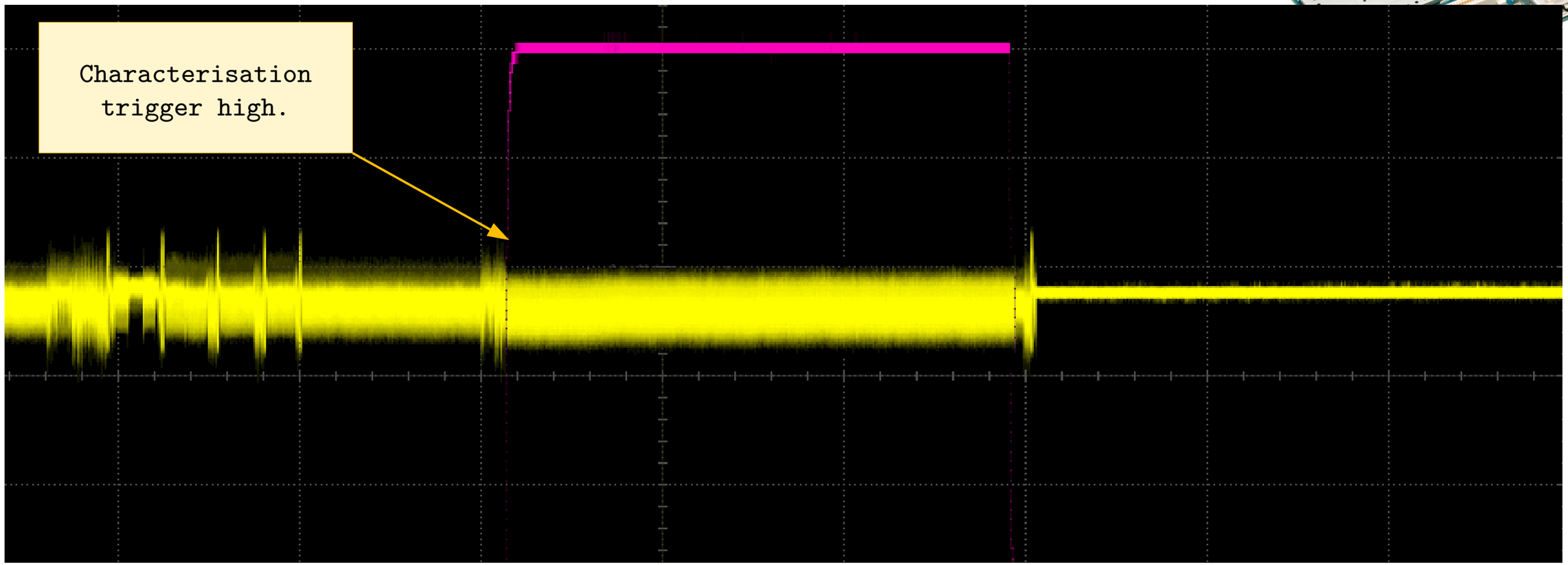
Vlind Glitch

Characterising the DragonBoard



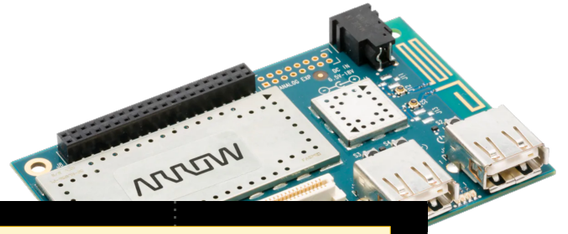
Vlind Glitch

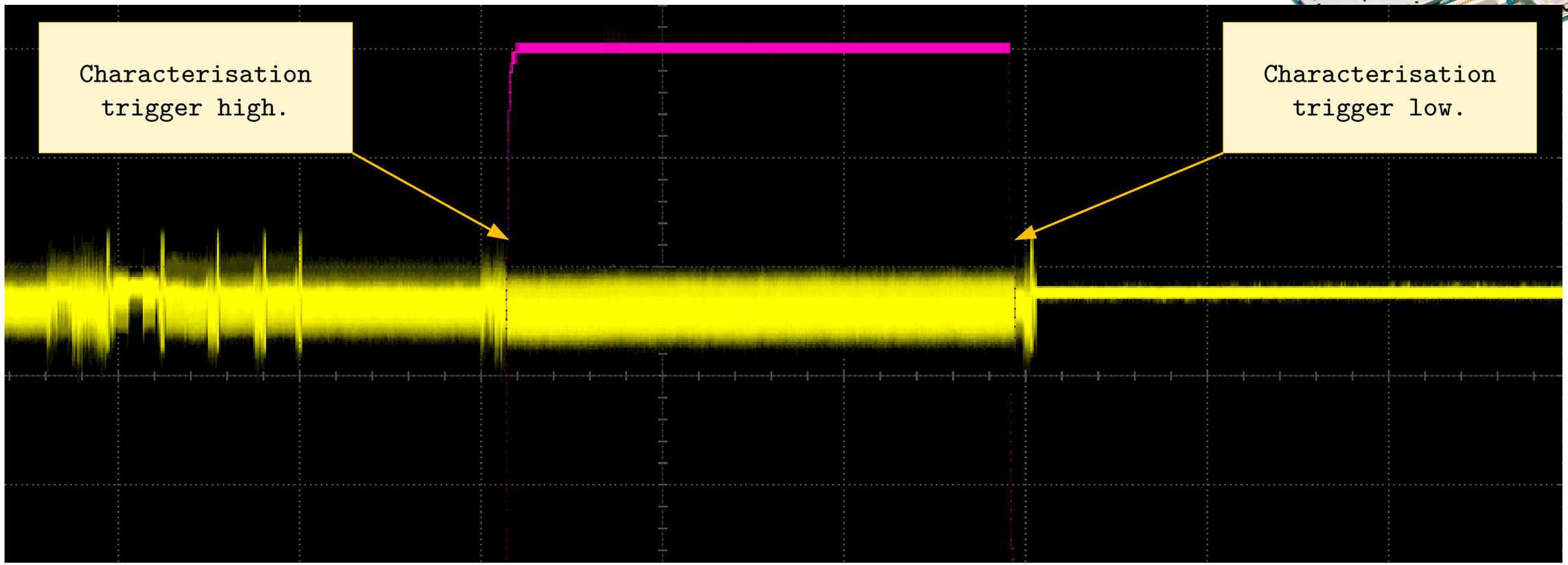
Characterising the DragonBoard



Vlind Glitch

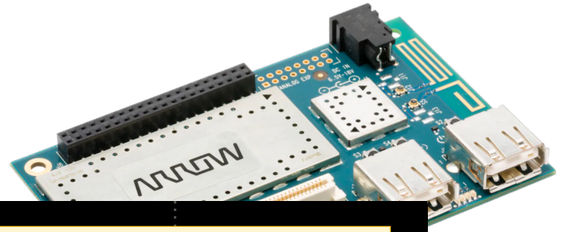
Characterising the DragonBoard

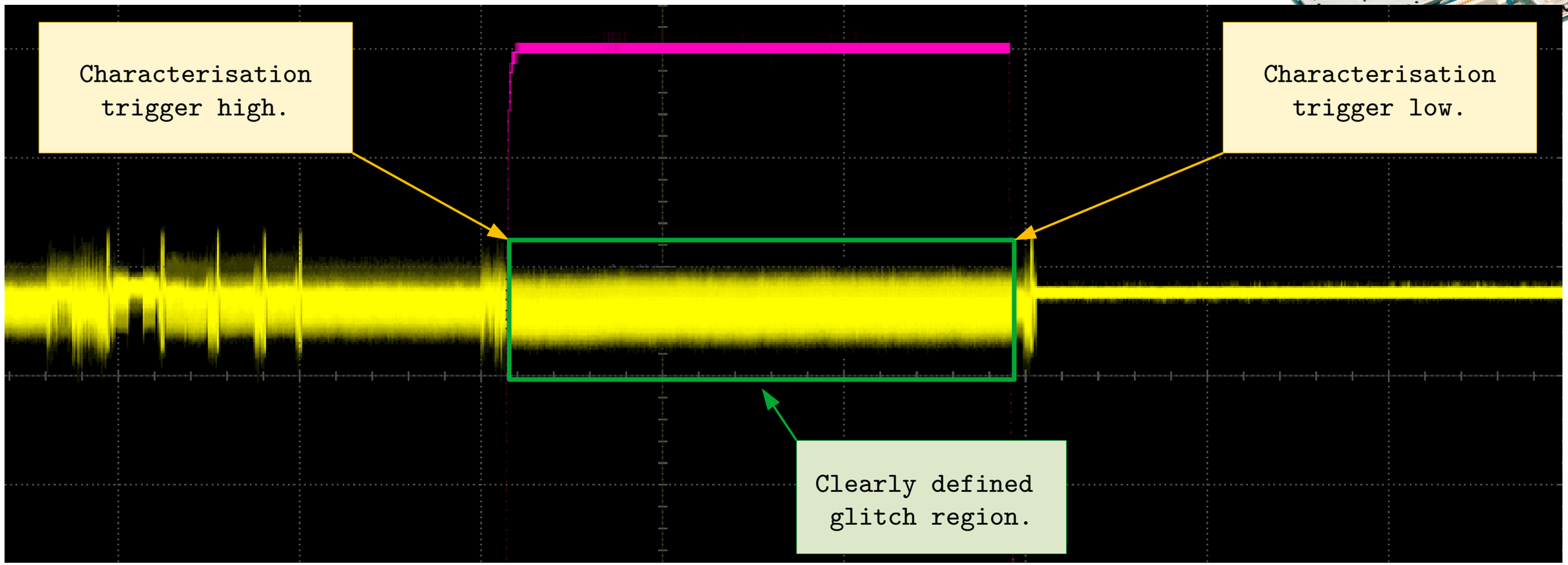




Vlind Glitch

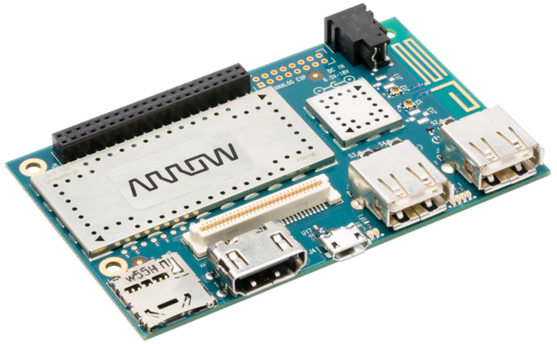
Characterising the DragonBoard





Vlind Glitch

Characterising the DragonBoard



- Characterisation made our lives easier:
  - Bypass still doable without characterisation; less efficient use of work.
  - We now know the best glitching parameters.
  - Not every glitch will be a success, but an enough number of them will eventually lead to one.

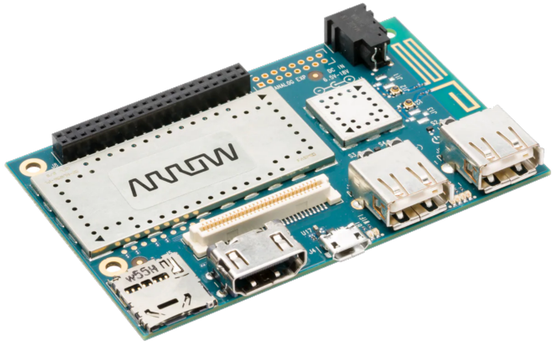
Vlind Glitch

Bypassing Secure Boot

- Attacking a real device:
  - We do not have the keys!
  - We can not run the characterisation step BUT
  - We have learnt the characterisation parameters range.
  - If we modify a programmer with custom code:
    - EDL will reject it (remember we do not have the keys).
- Goal: glitch EDL when verifying the modified programmer.
  - Influence the device when verifying the Root Key Hash.
  - With the proper parameters to bypass the verification.

Vlind Glitch

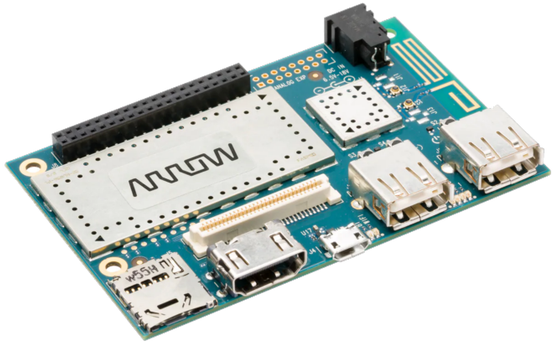
Bypassing Secure Boot



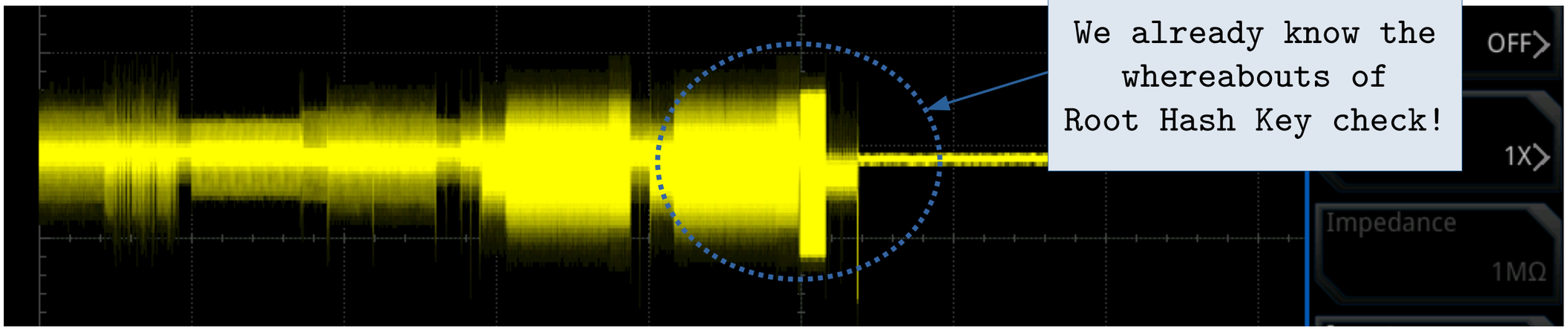
- Getting timing right:
  - Beagle allows us to trigger on USB data → Serves as time anchor.
  - Root Hash Key check has to happen after the programmer is received.
  - We use that information to set up a timing window.

Vlind Glitch

Bypassing Secure Boot

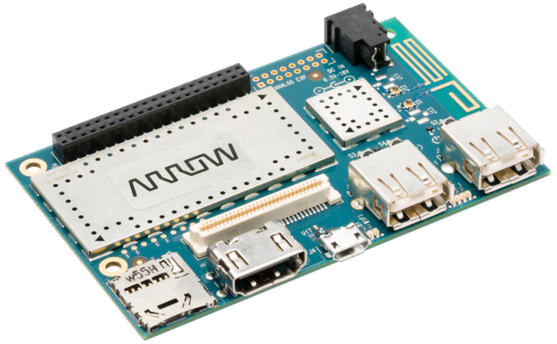


- Getting timing right:
  - Beagle allows us to trigger on USB data → Serves as time anchor.
  - Root Hash Key check has to happen after the programmer is received.
  - We use that information to set up a timing window.



Vlind Glitch

Bypassing Secure Boot

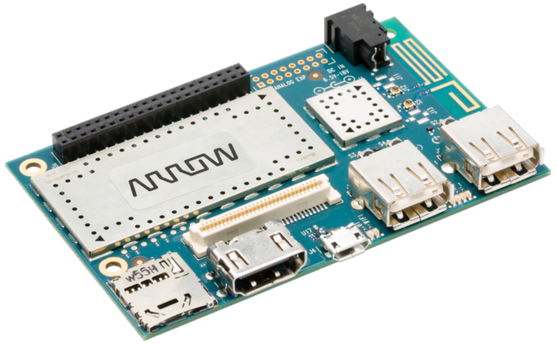


- Modifying programmer – PoC:
  - We modify a valid programmer to return us a string.
  - We generated new keys, different to the ones used to enable Secure Boot.
  - We signed the modified programmer with the new keys
    → Forces Root Key Hash check failure, programmer will be rejected.

```
int firmwarewrite(void) {
        void (*usb_log)(char *s, ...) = (0x080054DC + 1);
        usb_log("Cyber Intelligence 2022: \
                We are executing code!");
        return 0;
}
```

Vlind Glitch

Bypassing Secure Boot

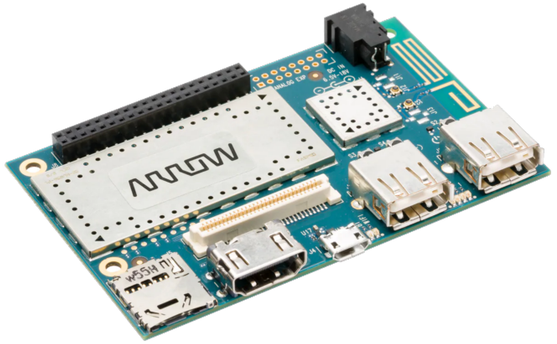


- Putting it all together:
  - Glitch parameters from the characterisation step.
  - Timing windows from power analysis and Beagle.
  - Loading a non-verified programmer.

- Time to start glitching!

Vlind Glitch

Bypassing Secure Boot

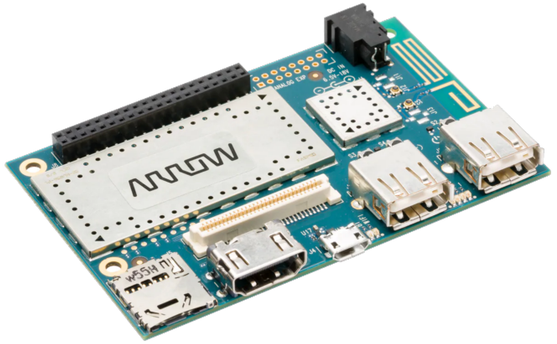


- Challenges:
  - USB stack is inherently jittery:
    - Non-deterministic delays → Trigger signal may be imprecise.
  - As in the characterisation step, not every glitch attempt is a success.

- Benefits:
  - Generic methodology → Applicable to numerous devices.
  - No need for reverse engineering, code or implementation knowledge.

Vlind Glitch

Bypassing Secure Boot



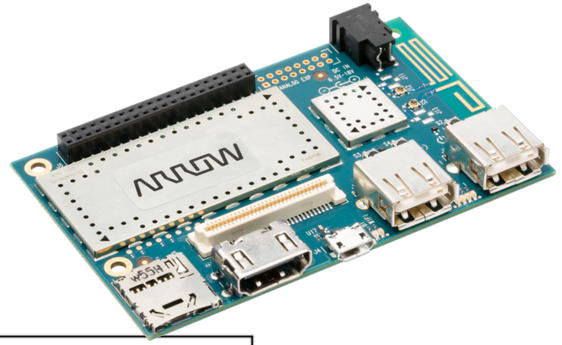
- Challenges:
  - USB stack is inherently jittery:
    - Non-deterministic delays → Trigger signal may be imprecise.
  - As in the characterisation step, not every glitch attempt is a success.

- Benefits:
  - Generic methodology → Applicable to numerous devices.
  - No need for reverse engineering, code or implementation knowledge.

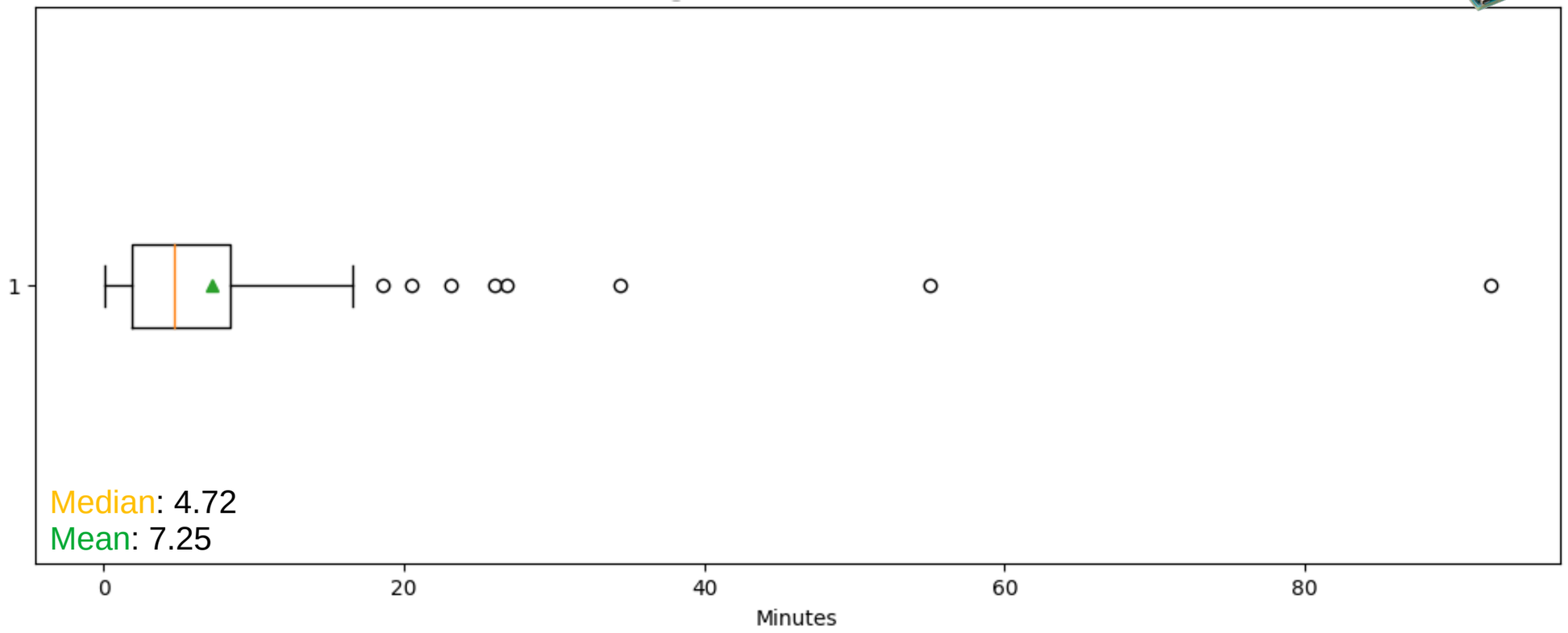
Two of our main objectives!

Vlind Glitch

Bypassing Secure Boot

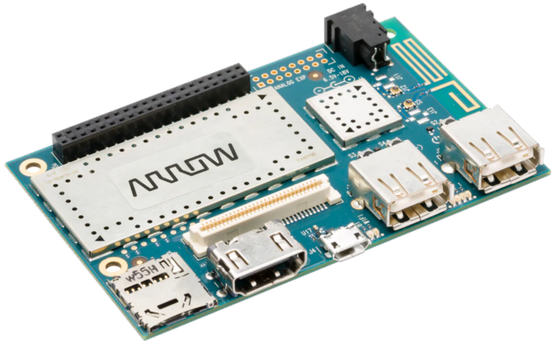


- Bypasses: 125 in 15 hours.

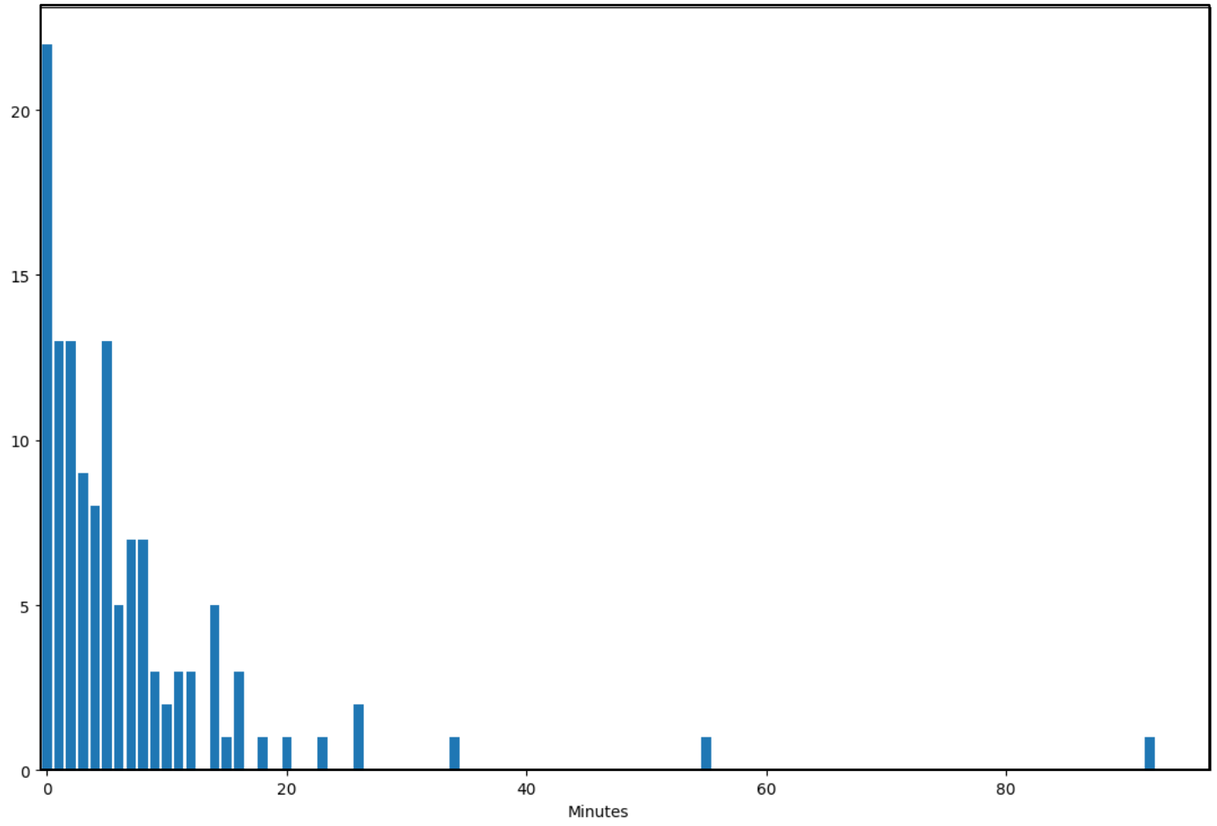


Vlind Glitch

Bypassing Secure Boot

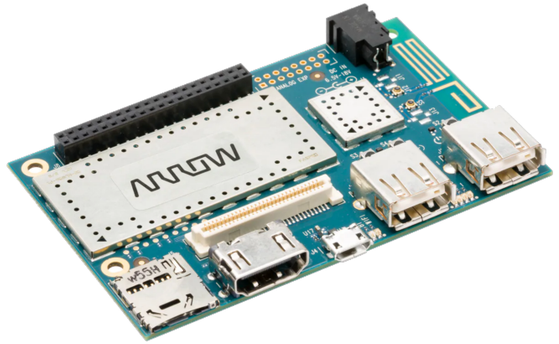


- Bypasses: 125 in 15 hours.



DEMO

Bypassing Secure Boot with Vlind Glitch



Conclusions

- Vlind Glitch is a powerful technique to bypass secure boot.
- The technique do not require the `PBL`, how the secure boot is implemented or the line of code to be glitched.
- We successfully bypassed the secure boot of the `DragonBoard 410c`.
  - On average we get a bypass each 7.25 minutes.
- After the attack the stability of the board was not affected.
  - We successfully dumped the `bootrom/PBL`.
  - We compared this `PBL` with the original and they were the same.
- With the BootROM we are not blind anymore → Software vulnerabilities.

EXFILES Grant Agreement No. 883156

“The **EXFILES** project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement
No **883156**.”

If you need further information, please contact the coordinator:

Technikon Forschungs- und Planungsgesellschaft mbH
Burgplatz 3a, 9500 Villach, AUSTRIA
Phone: +43 4242 233 55 Fax: +43 4242 233 55 77
Mail: **coordination@exfiles.eu**