Secure boot under attack: Simulation to enhance fault injection & defenses

Martijn Bogaard
Senior Security Analyst
martijn@riscure.com / @jmartijnb

Niek Timmers
Principal Security Analyst
niek@riscure.com / @tieknimmers

Today’s agenda

Today’s agenda

- Crash course secure boot on embedded devices

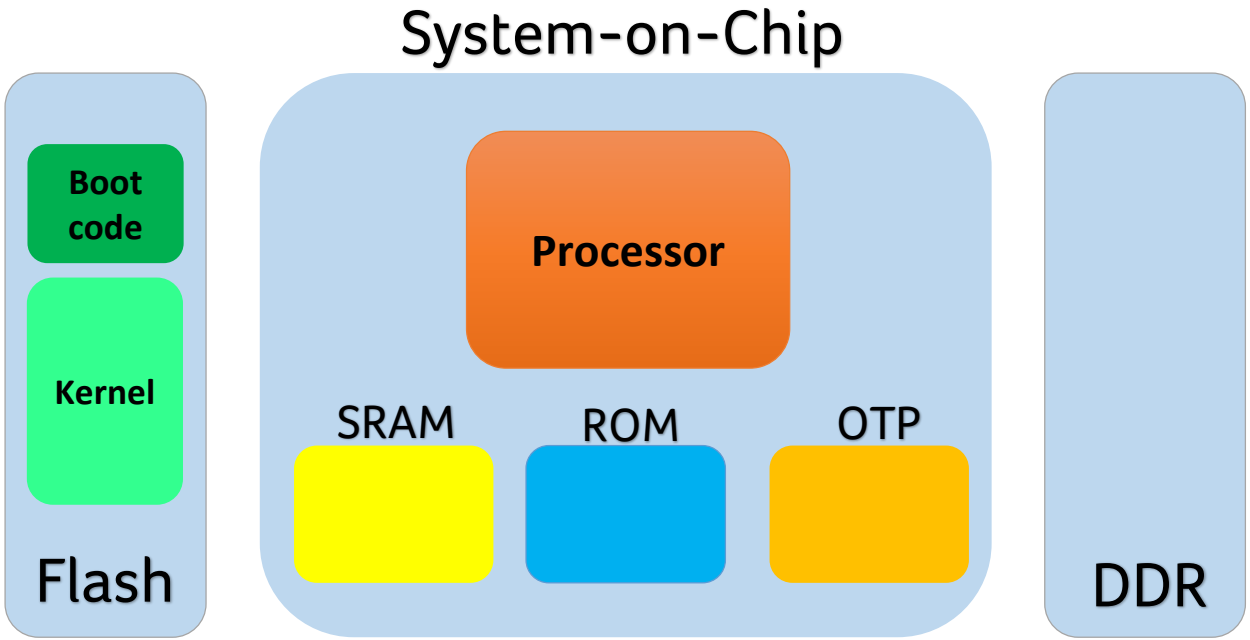
Today’s agenda

- Crash course secure boot on embedded devices
- Crash course fault injection (FI) attacks

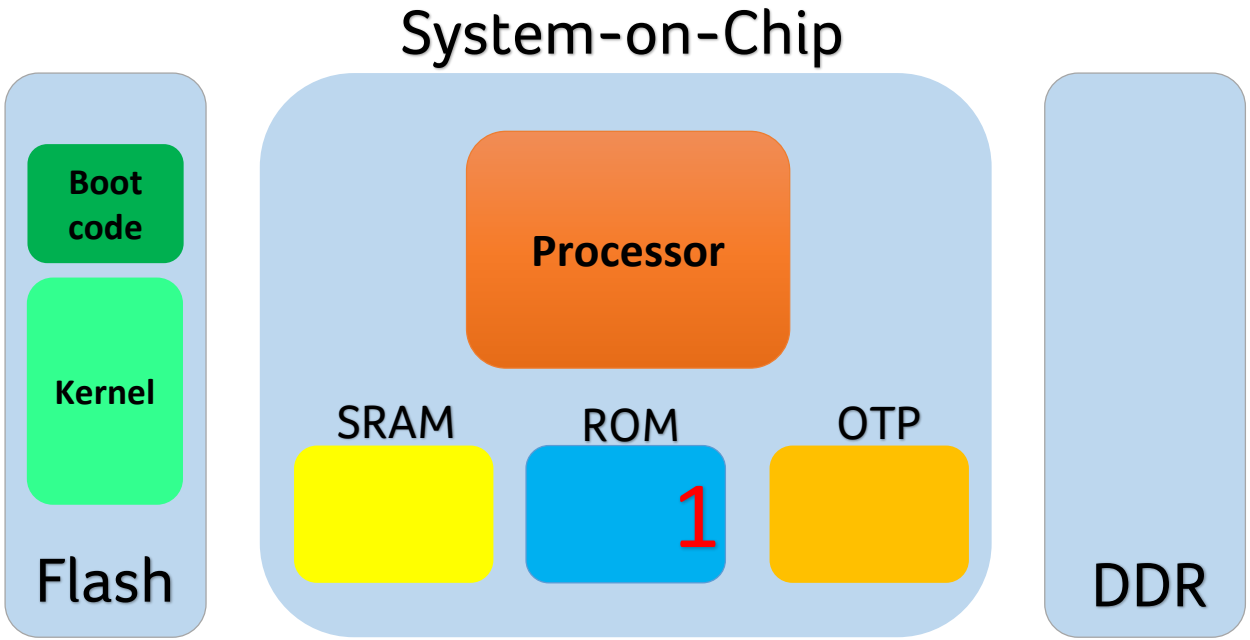
Today’s agenda

- Crash course secure boot on embedded devices
- Crash course fault injection (FI) attacks
- **Using simulation to identify FI vulnerabilities**

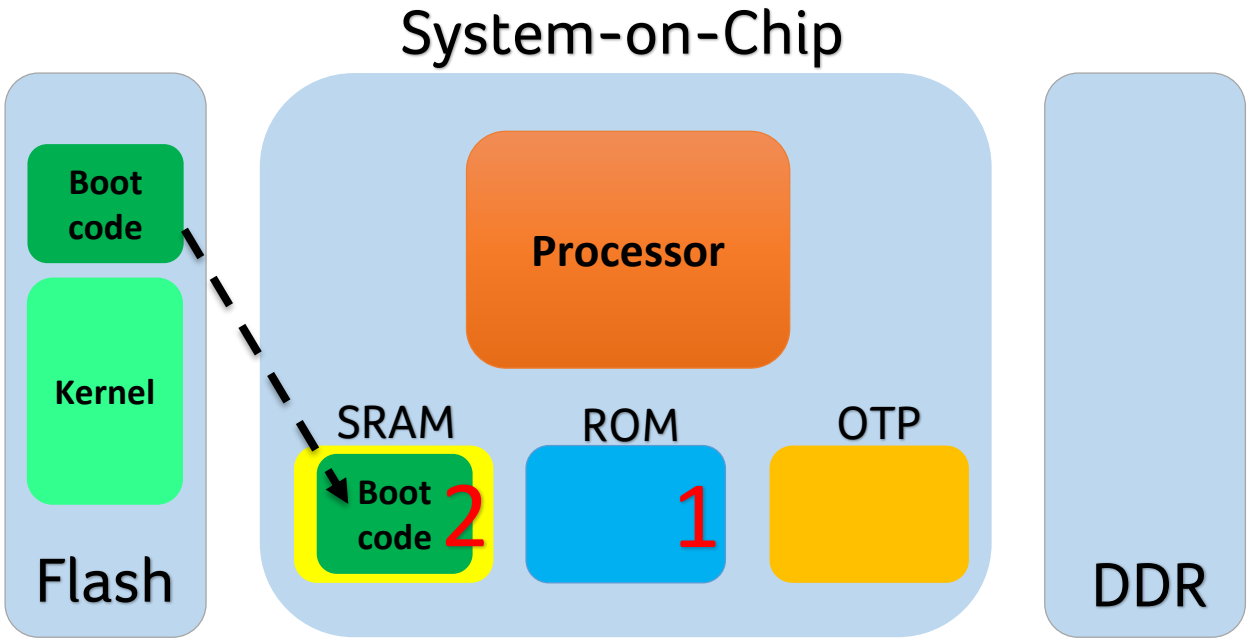
Why do we need secure boot?



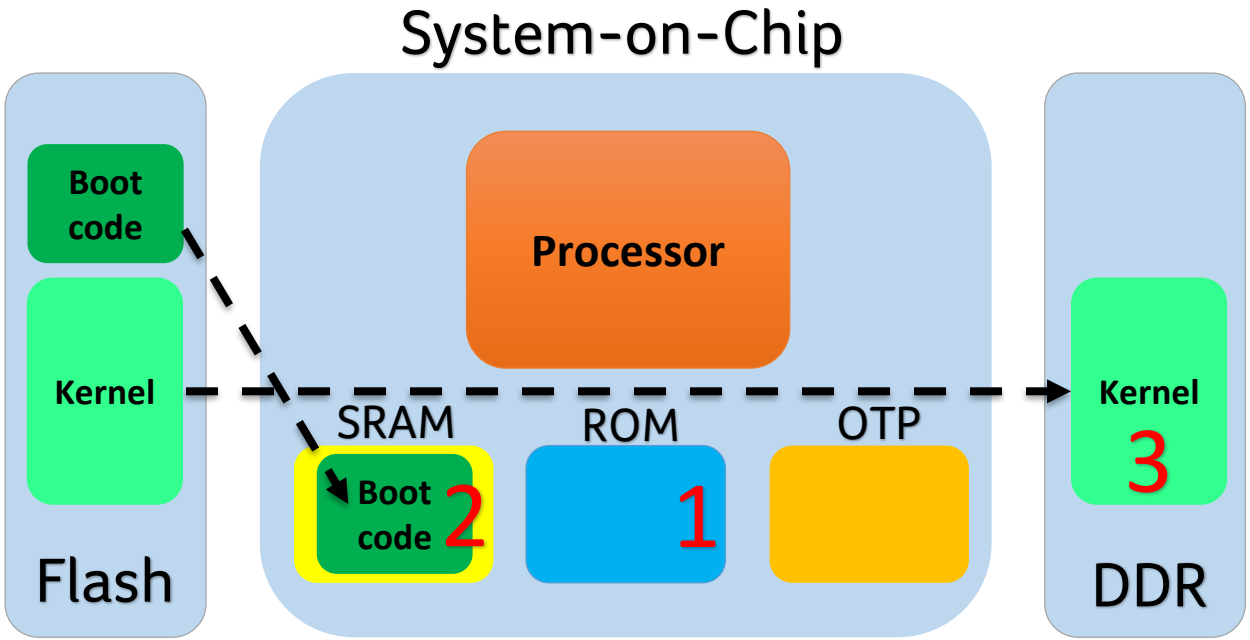
Why do we need secure boot?



Why do we need secure boot?

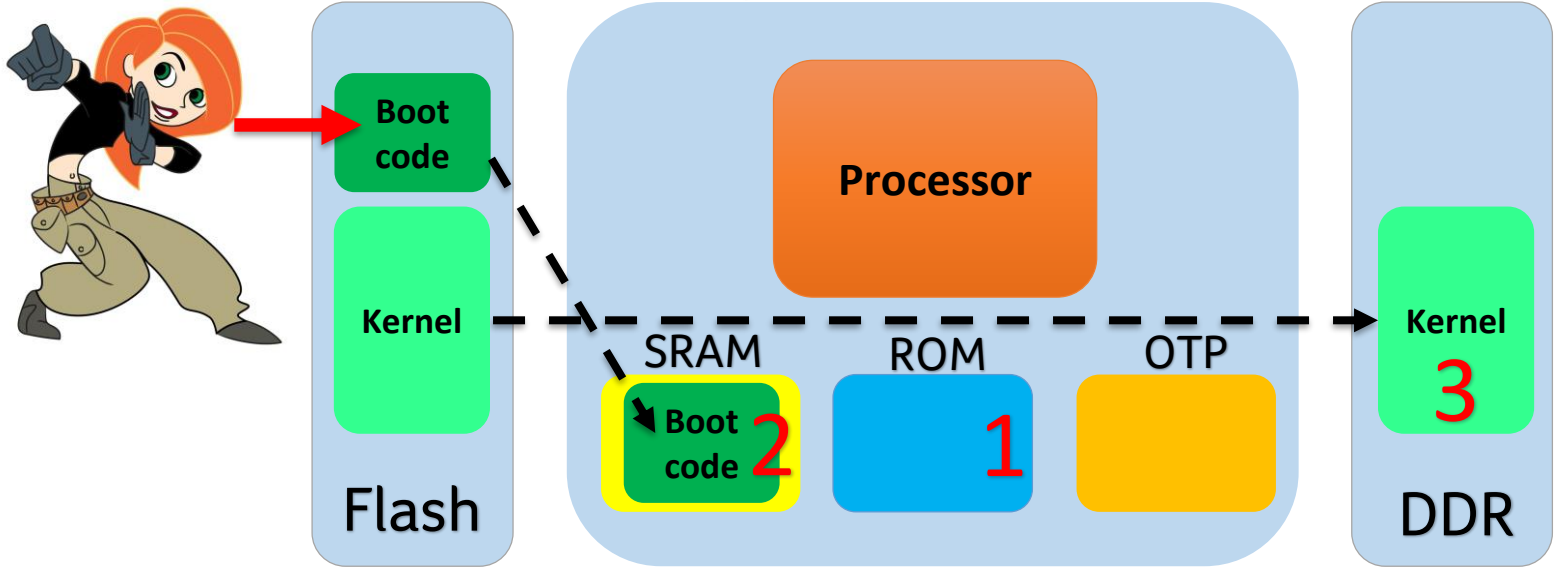


Why do we need secure boot?

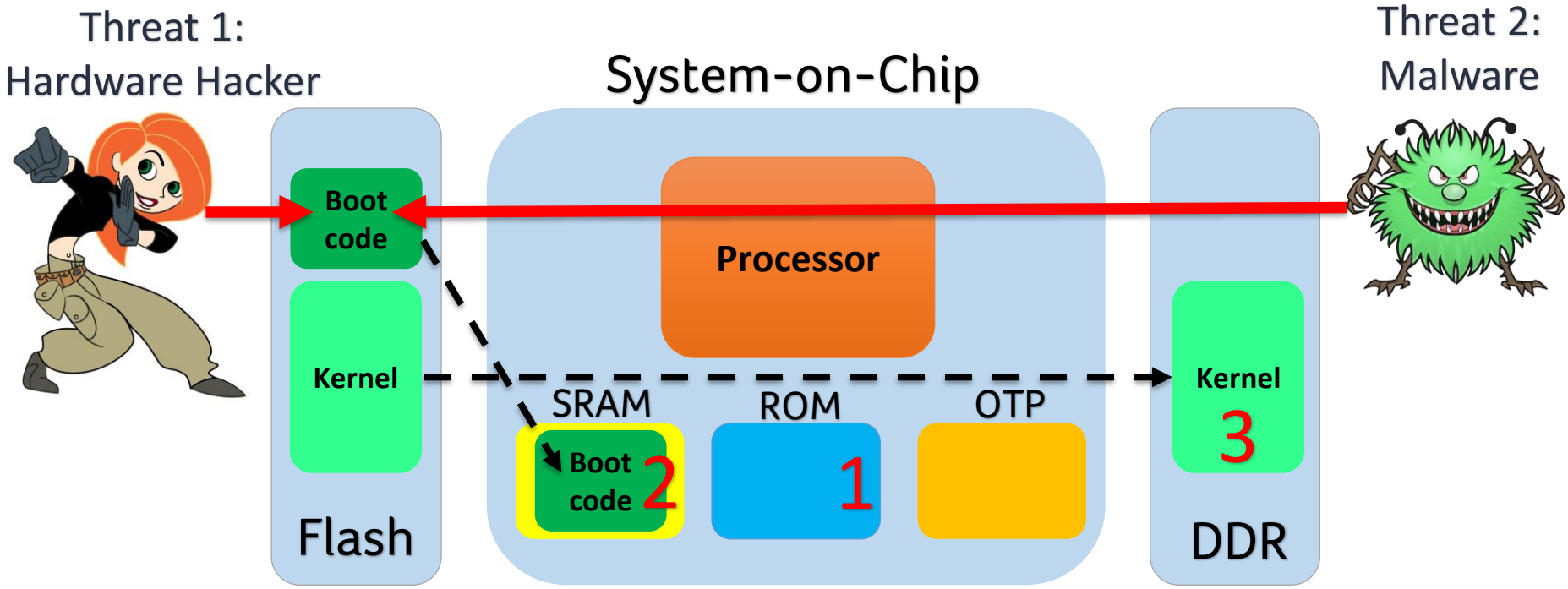


Why do we need secure boot?

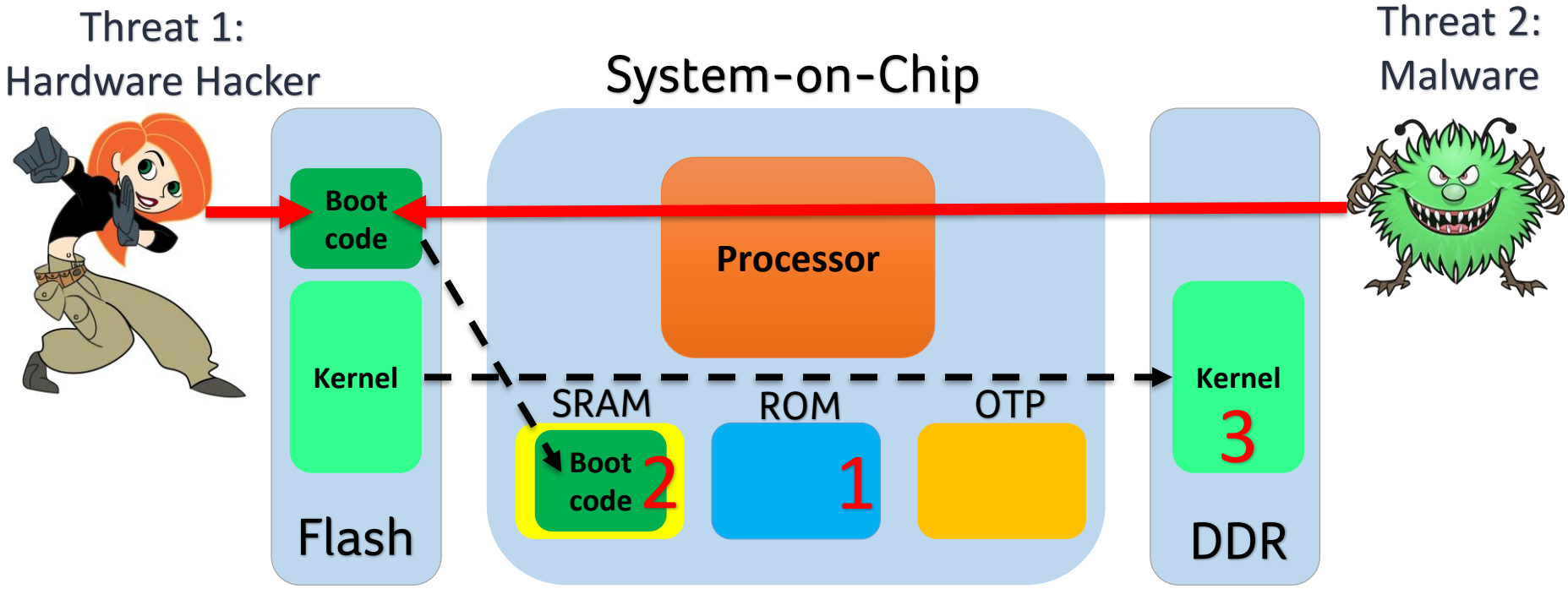
Threat 1:
Hardware Hacker



Why do we need secure boot?



Why do we need secure boot?



Secure boot assures integrity of code/data in cold storage!

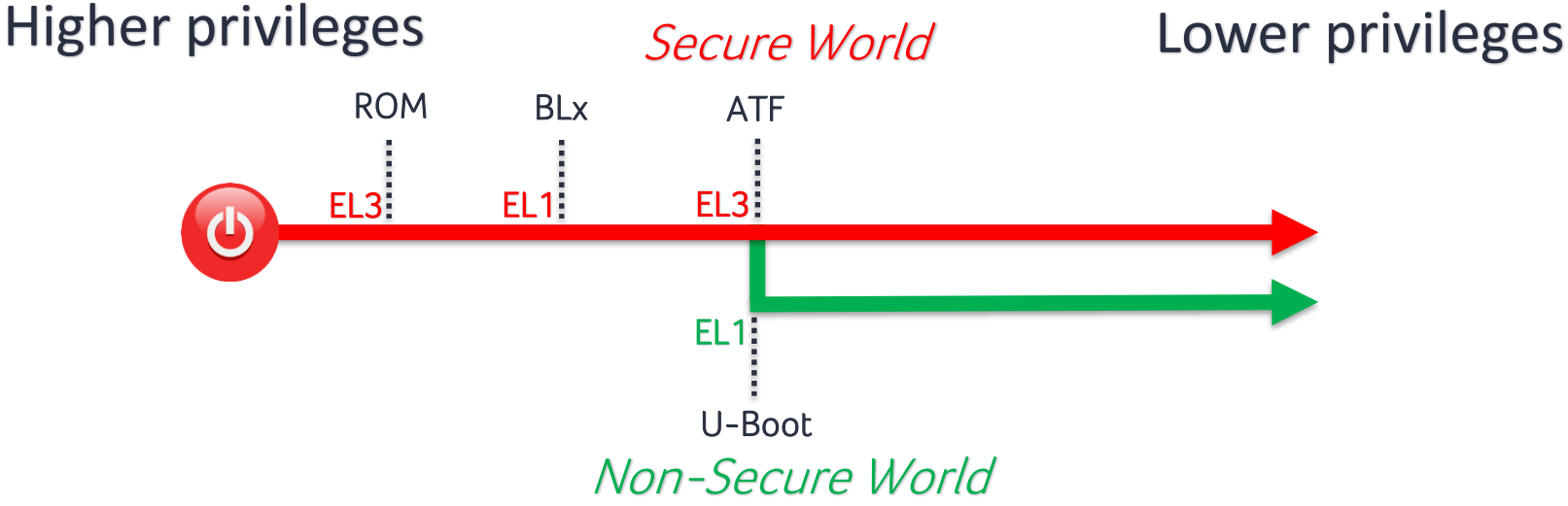
The real world is more complex!

The real world is more complex!

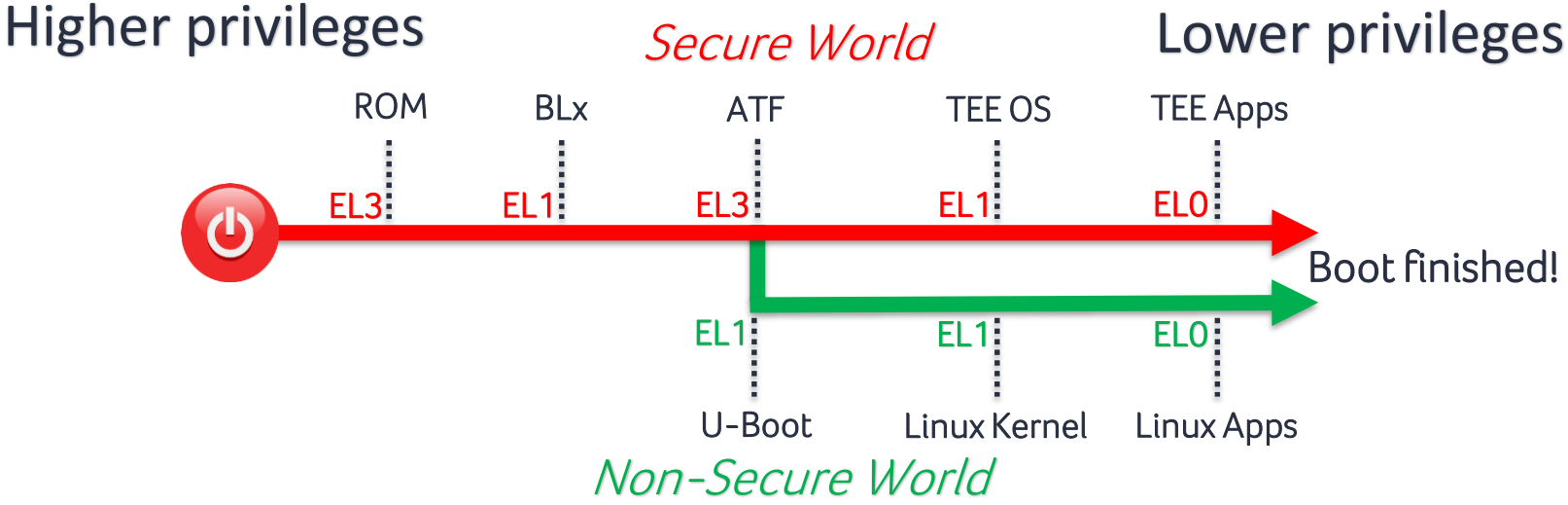
The real world is more complex!

The real world is more complex!

The real world is more complex!



The real world is more complex!



The chain can break at any stage. Early is better!

Breaking Secure Boot early

Breaking Secure Boot early

- Early boot stage run at the highest privilege
  - E.g. unrestricted access

Breaking Secure Boot early

- Early boot stage run at the highest privilege
  - E.g. unrestricted access
- Security features often not initialized yet
  - E.g. access control

Breaking Secure Boot early

- Early boot stage run at the highest privilege
  - E.g. unrestricted access
- Security features often not initialized yet
  - E.g. access control
- Access assets that are not accessible after boot
  - E.g. ROM code and keys

What makes Secure Boot secure?

What makes Secure Boot secure?

Unbreakable cryptography… Right?

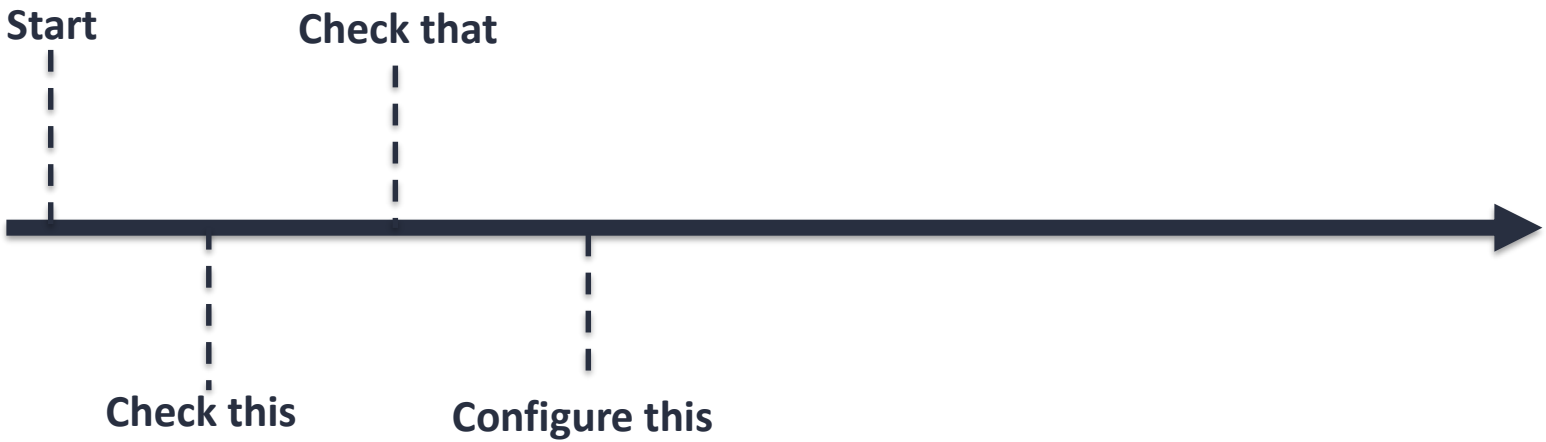
Flow of a typical boot stage

Flow of a typical boot stage

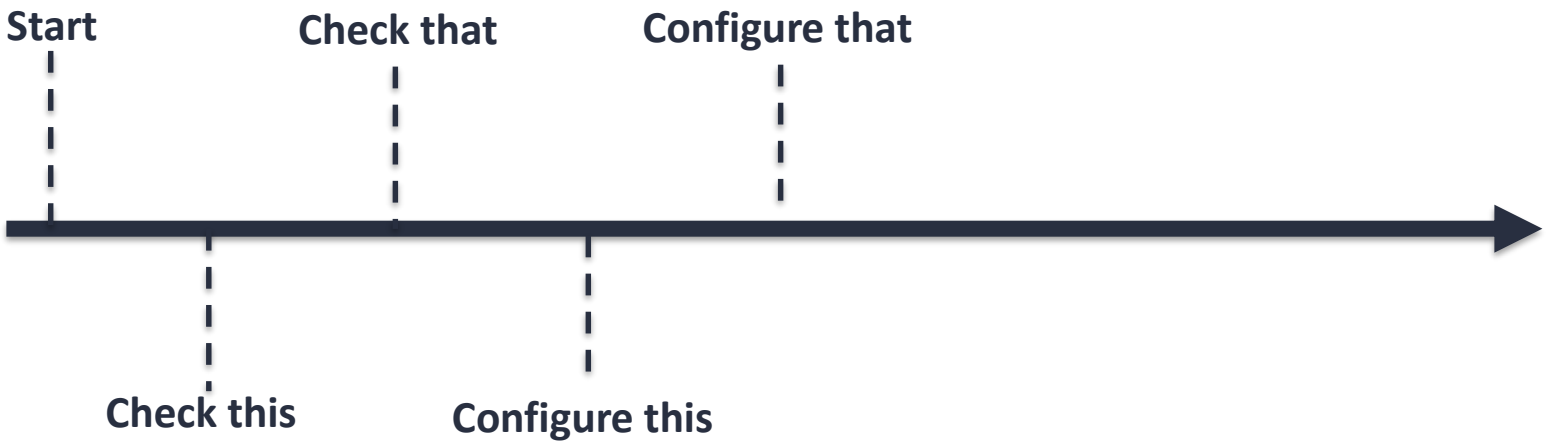
Flow of a typical boot stage

Flow of a typical boot stage

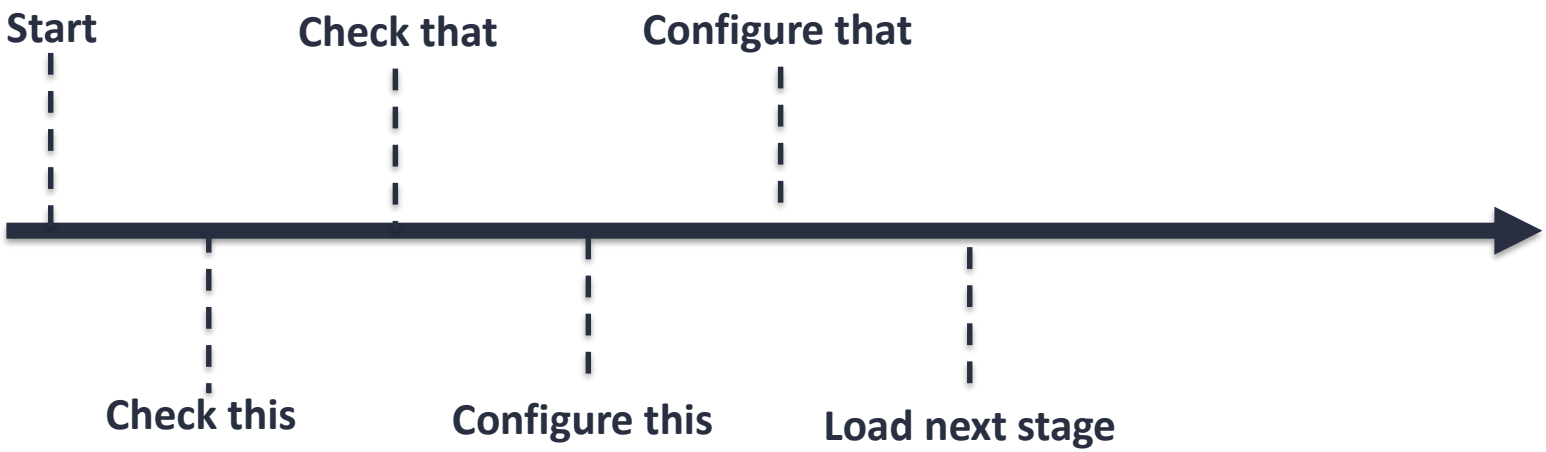
Flow of a typical boot stage



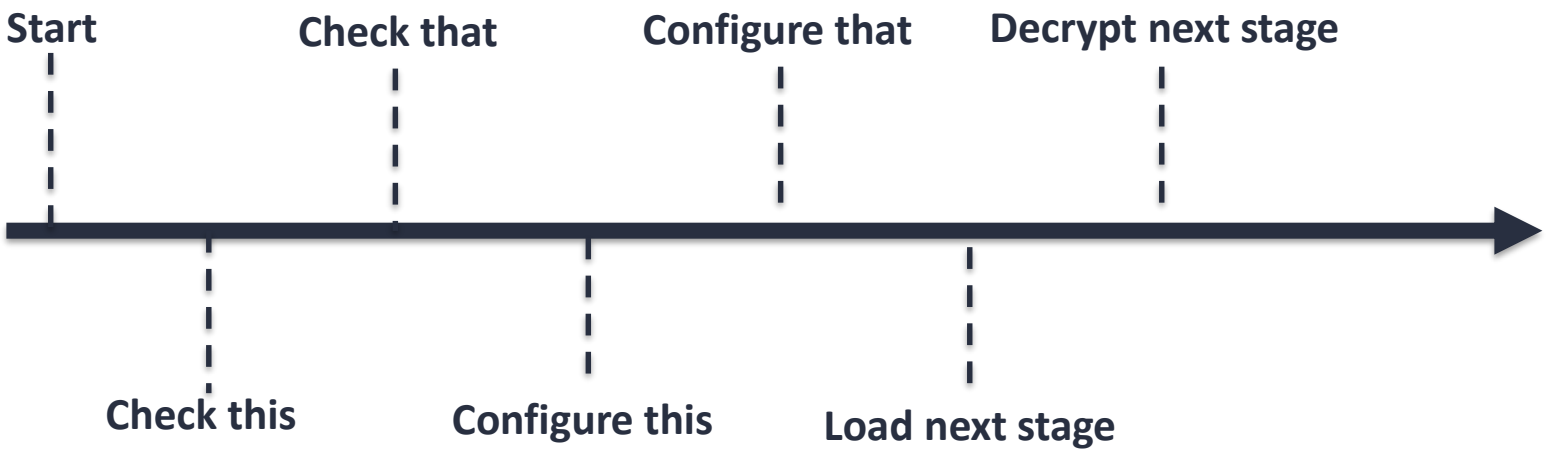
Flow of a typical boot stage



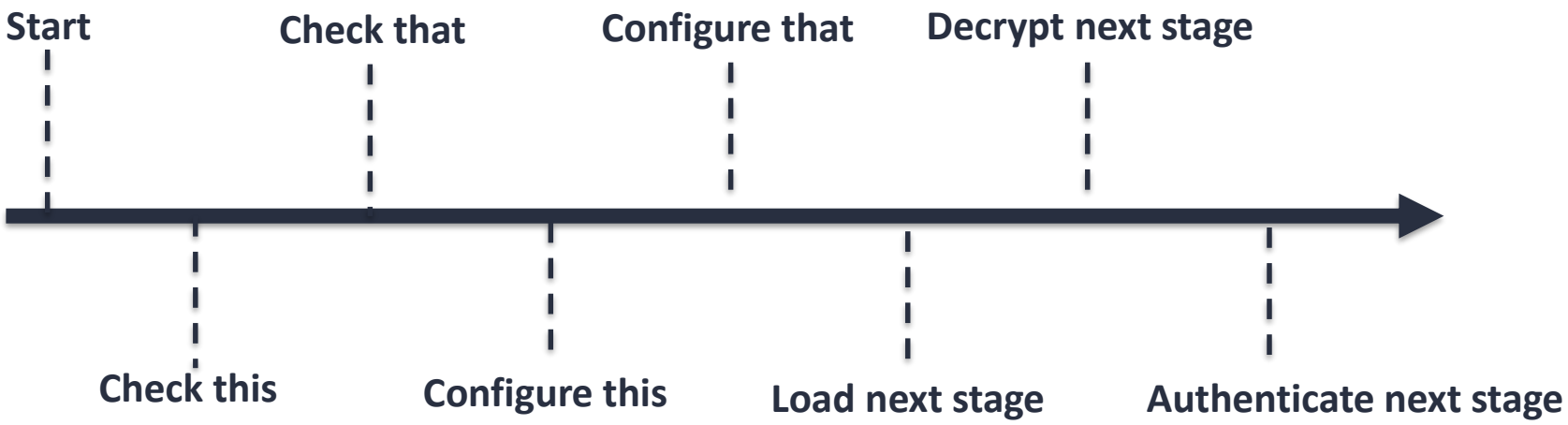
Flow of a typical boot stage



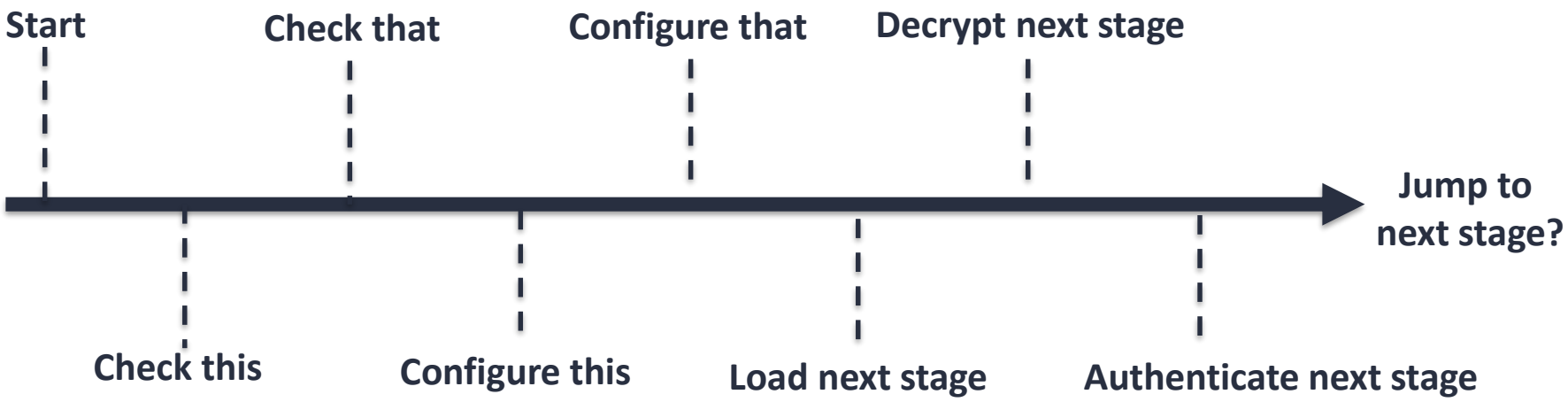
Flow of a typical boot stage



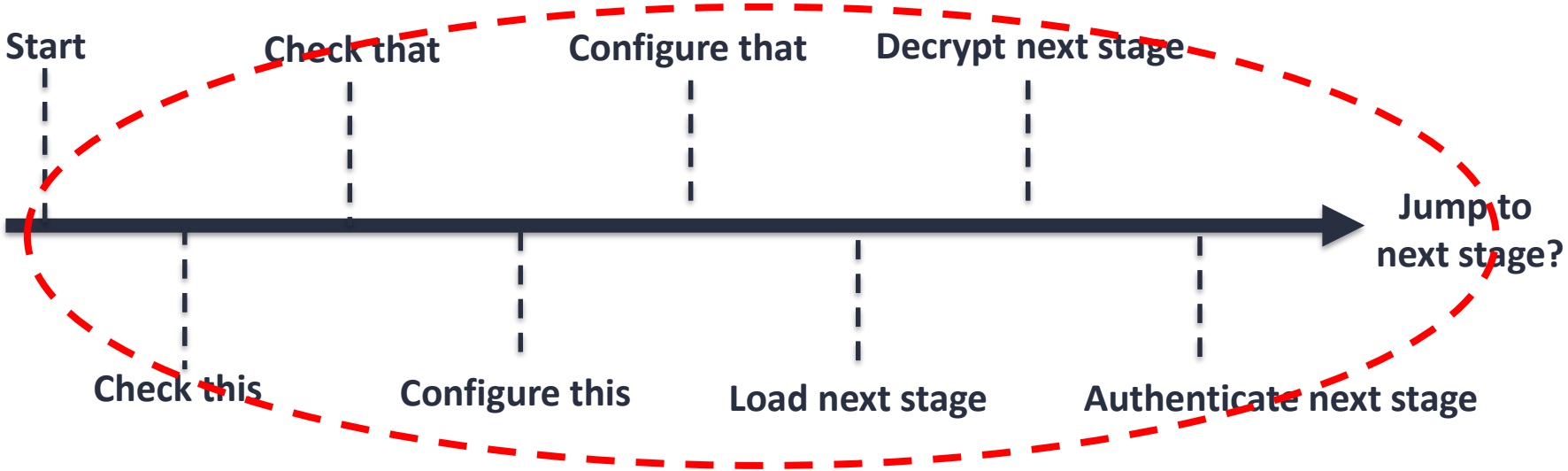
Flow of a typical boot stage



Flow of a typical boot stage

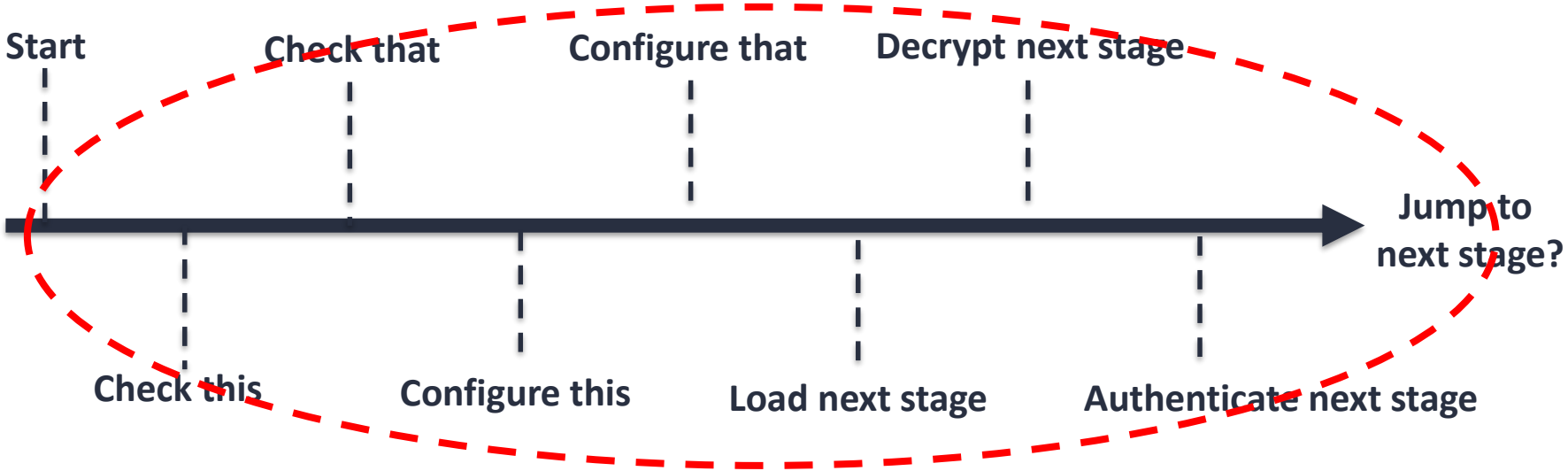


Flow of a typical boot stage



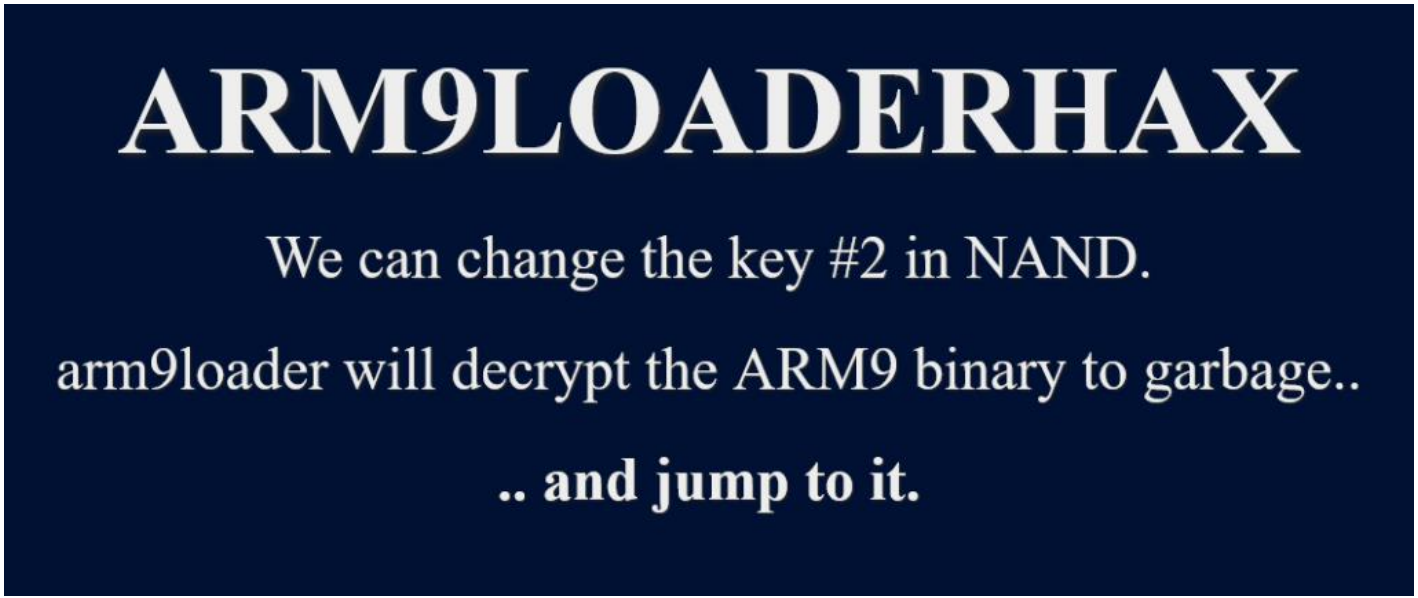
Lots of functionality! What can go wrong?

Flow of a typical boot stage



Lots of functionality! What goes wrong!?

No authentication!



https://smealum.github.io/3ds/32c3/#/95

Software vulnerabilities!

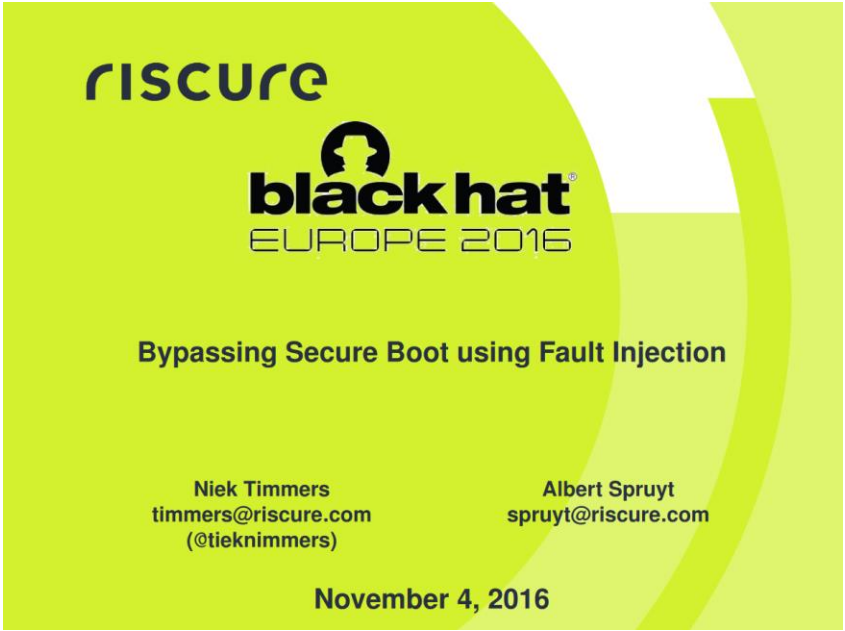
CVE-2018-18439, CVE-2018-18440 - U-Boot verified boot bypass vulnerabilities

From: Andrea Barisani <andrea.barisani () f-secure com>
Date: Fri, 2 Nov 2018 05:30:02 +0100

```
Security advisory: U-Boot verified boot bypass
==============================================

The Universal Boot Loader - U-Boot [1] verified boot feature allows
cryptographic authentication of signed kernel images, before their execution.

This feature is essential in maintaining a full chain of trust on systems which
are secure booted by means of an hardware anchor.

Multiple techniques have been identified that allow to execute arbitrary code,
within a running U-Boot instance, by means of externally provided
unauthenticated data.
```

https://seclists.org/oss-sec/2018/q4/125

Hardware vulnerabilities!



https://www.blackhat.com/docs/eu-16/materials/eu-16-Timmers-Bypassing-Secure-Boot-Using-Fault-Injection.pdf

Why hardware attacks on secure boot?

Why hardware attacks on secure boot?

- Usually a small code base

Why hardware attacks on secure boot?

- Usually a small code base
- Limited attack surface

Why hardware attacks on secure boot?

- Usually a small code base
- Limited attack surface
- Should be extensively reviewed

Why hardware attacks on secure boot?

- Usually a small code base
- Limited attack surface
- Should be extensively reviewed
- Difficult / impossible to fix after deployment

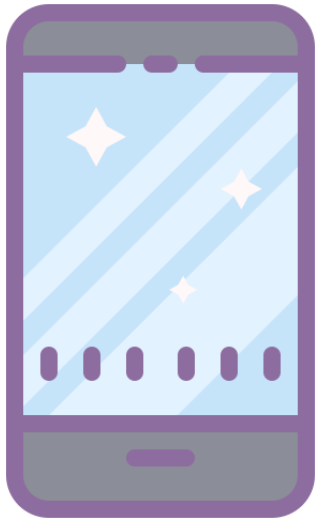
Why hardware attacks on secure boot?

- Usually a small code base
- Limited attack surface
- Should be extensively reviewed
- Difficult / impossible to fix after deployment

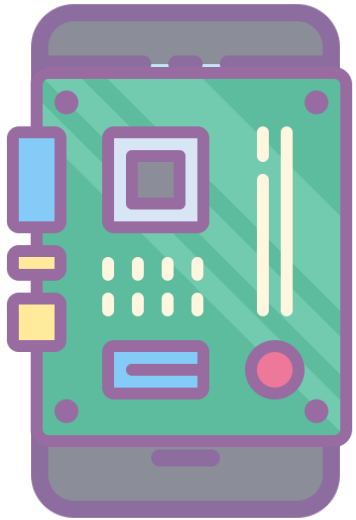
Software vulnerabilities not guaranteed to be present!

Voltage Fault Injection in practice

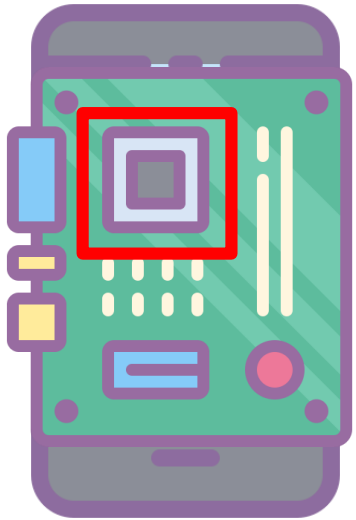
Voltage Fault Injection in practice



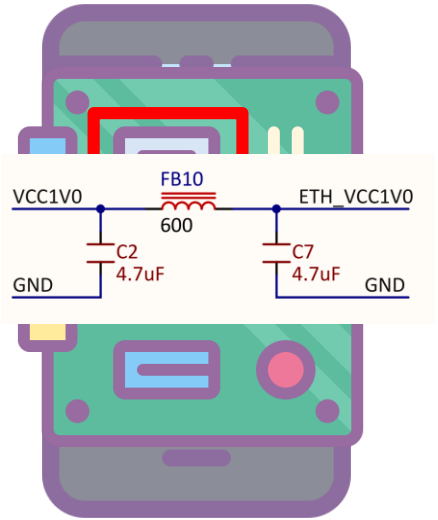
Voltage Fault Injection in practice



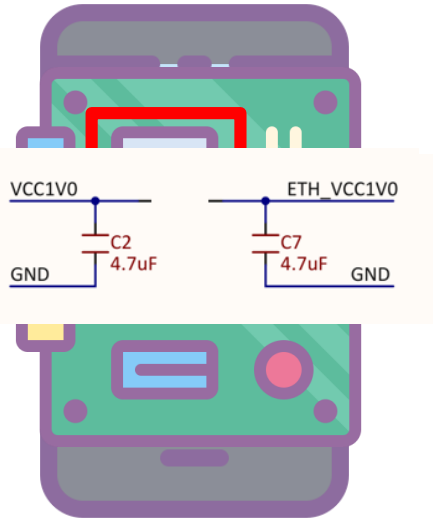
Voltage Fault Injection in practice



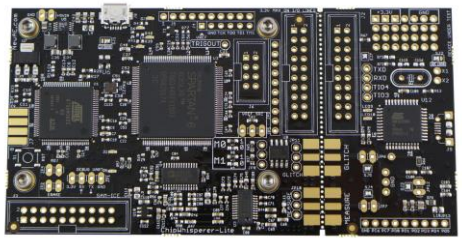
Voltage Fault Injection in practice

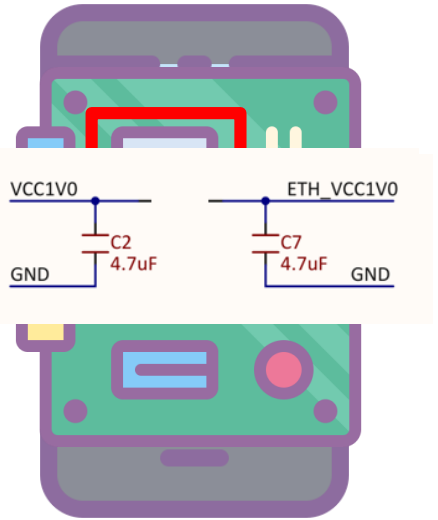


Voltage Fault Injection in practice

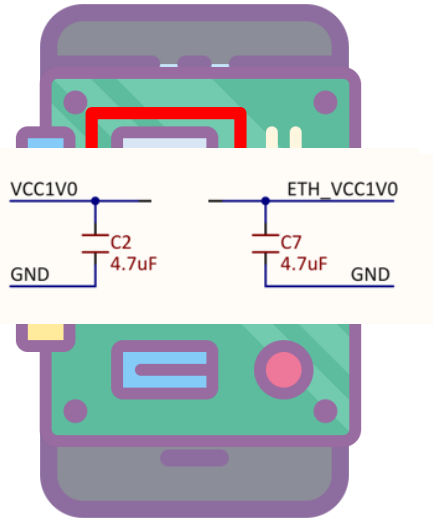


Voltage Fault Injection in practice

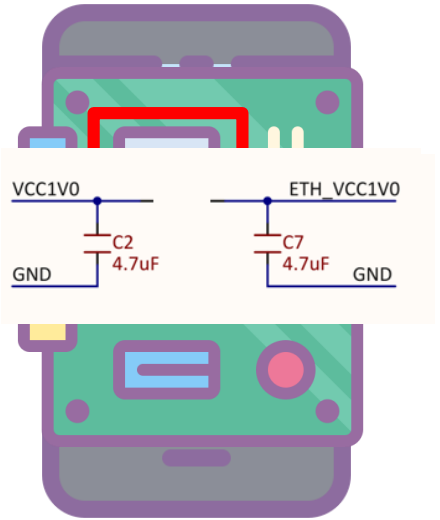




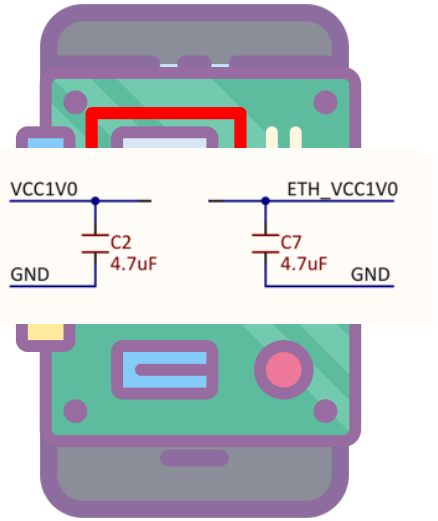
Voltage Fault Injection in practice



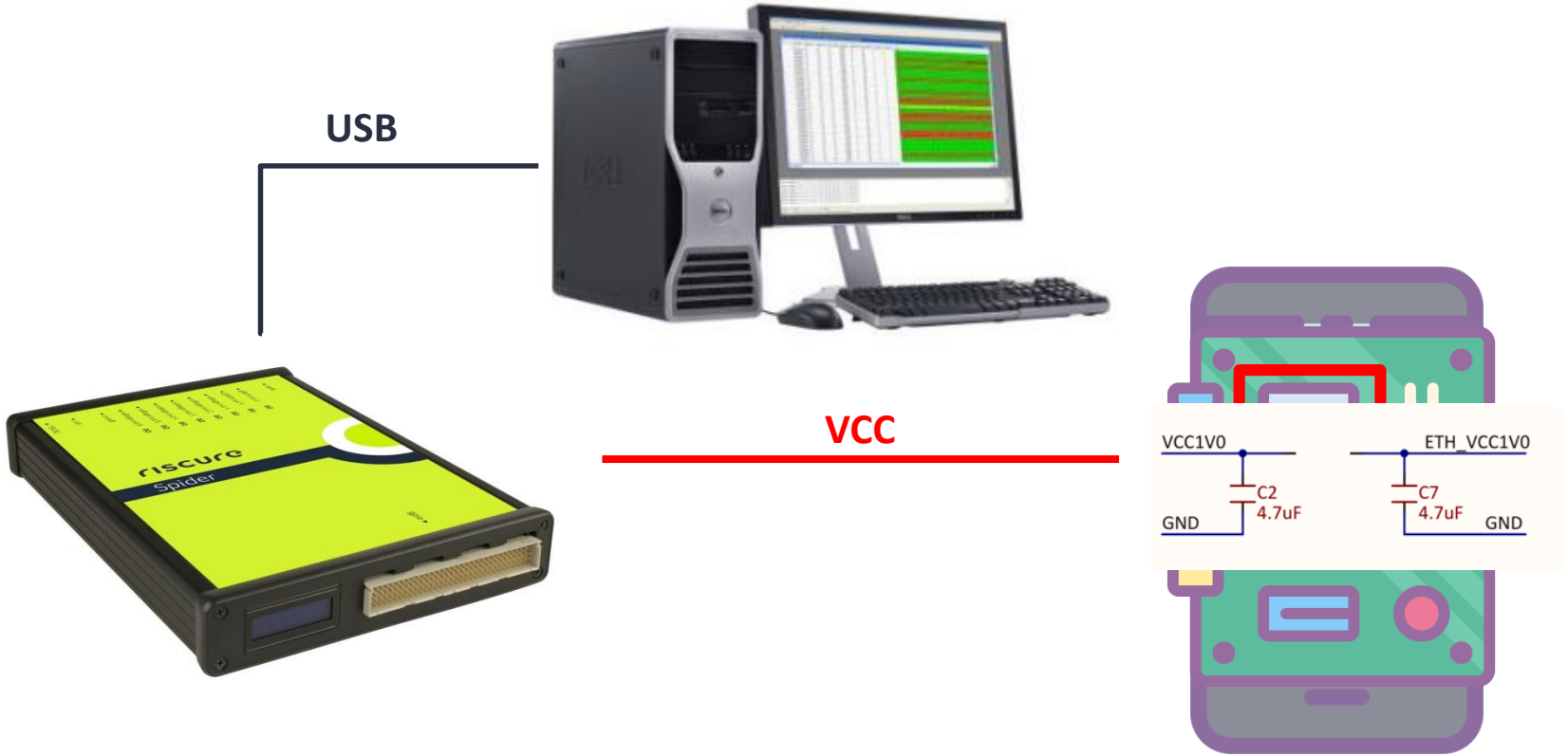
Voltage Fault Injection in practice



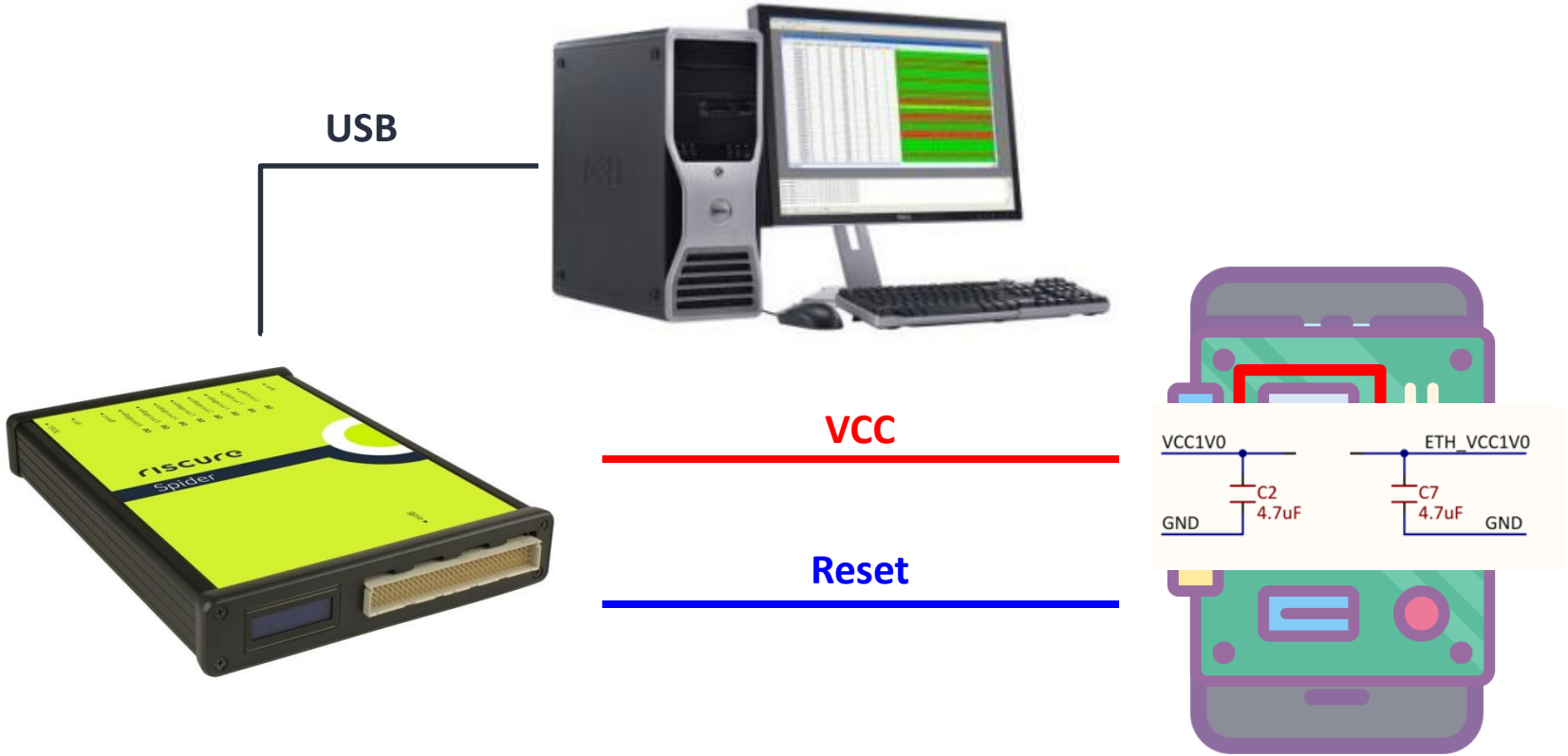
Voltage Fault Injection in practice

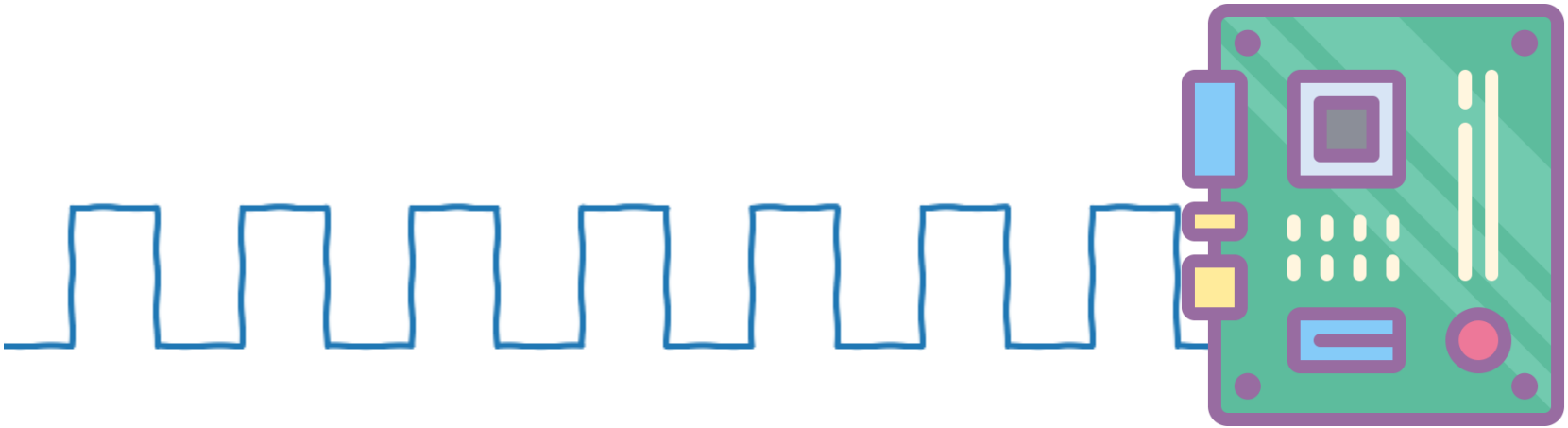


Voltage Fault Injection in practice

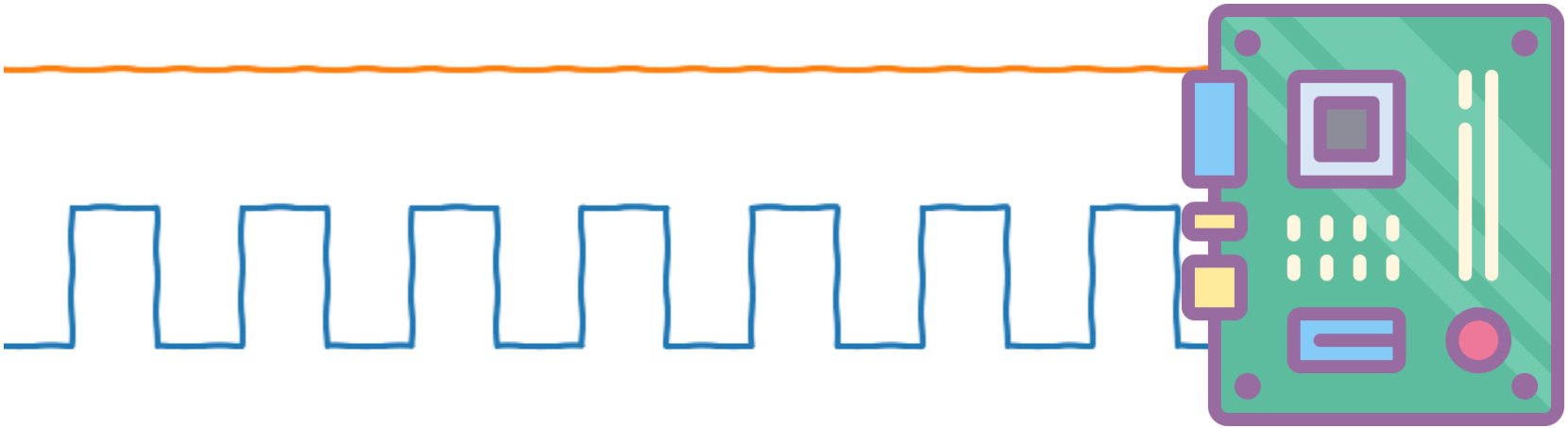


Voltage Fault Injection in practice

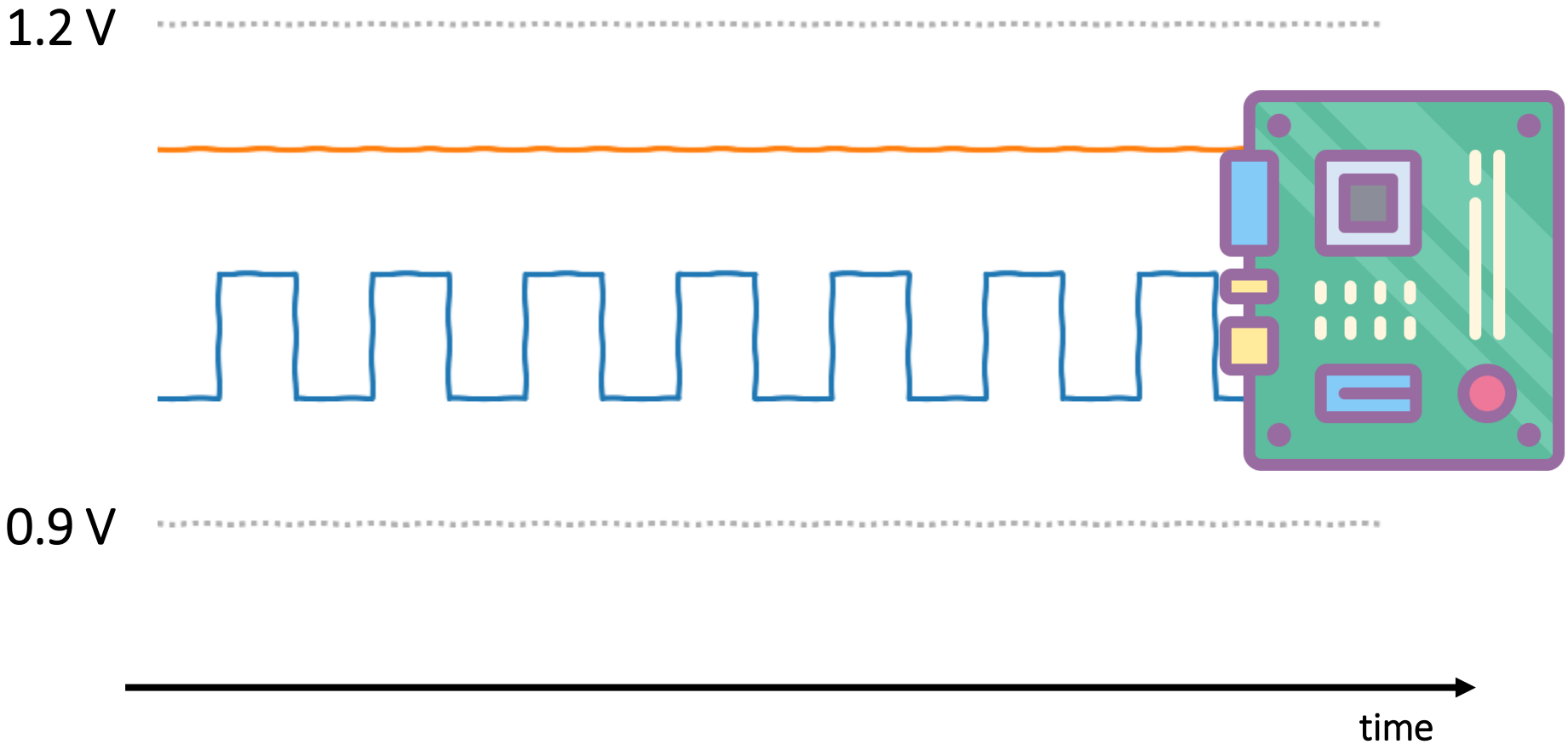


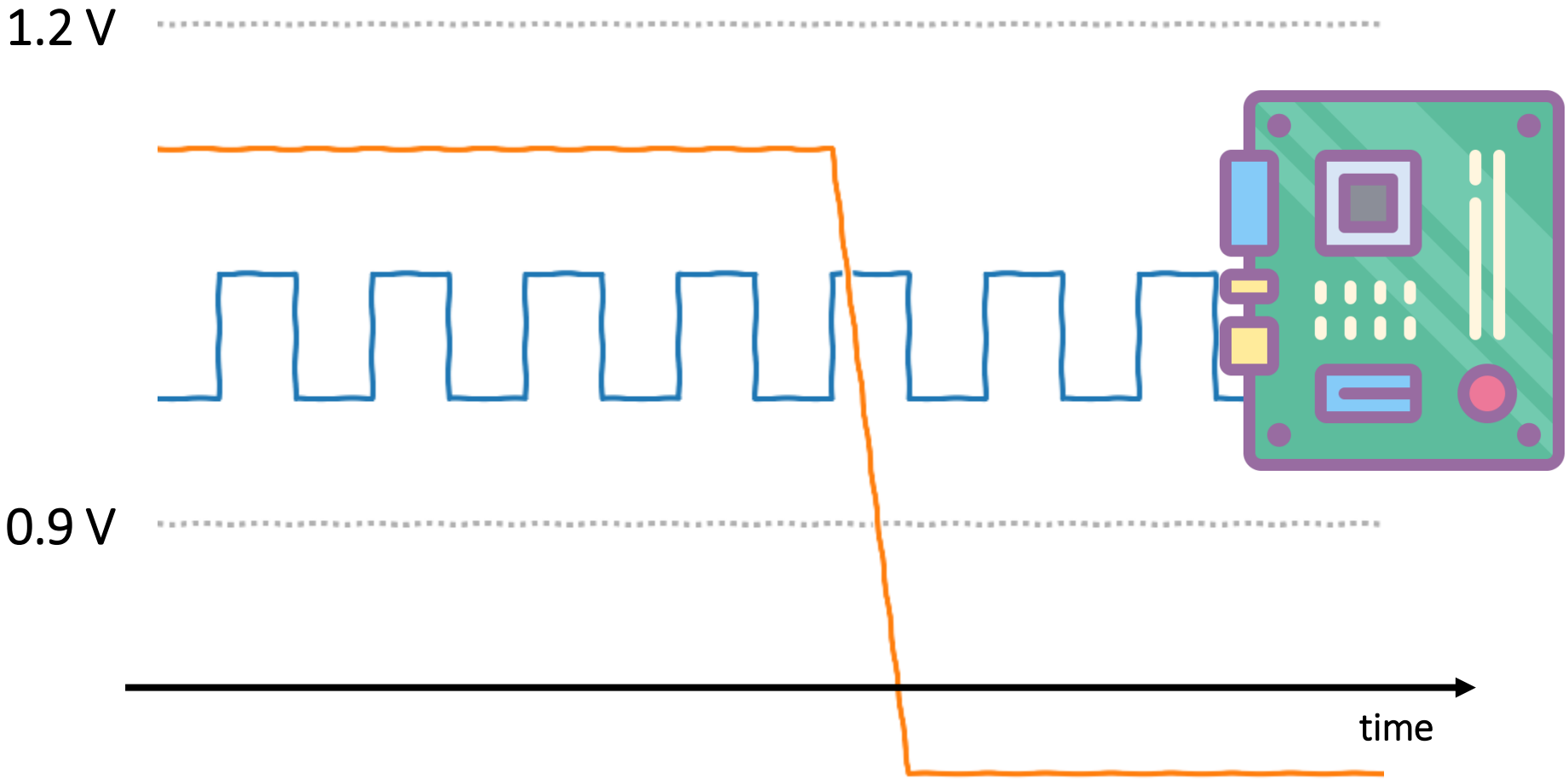


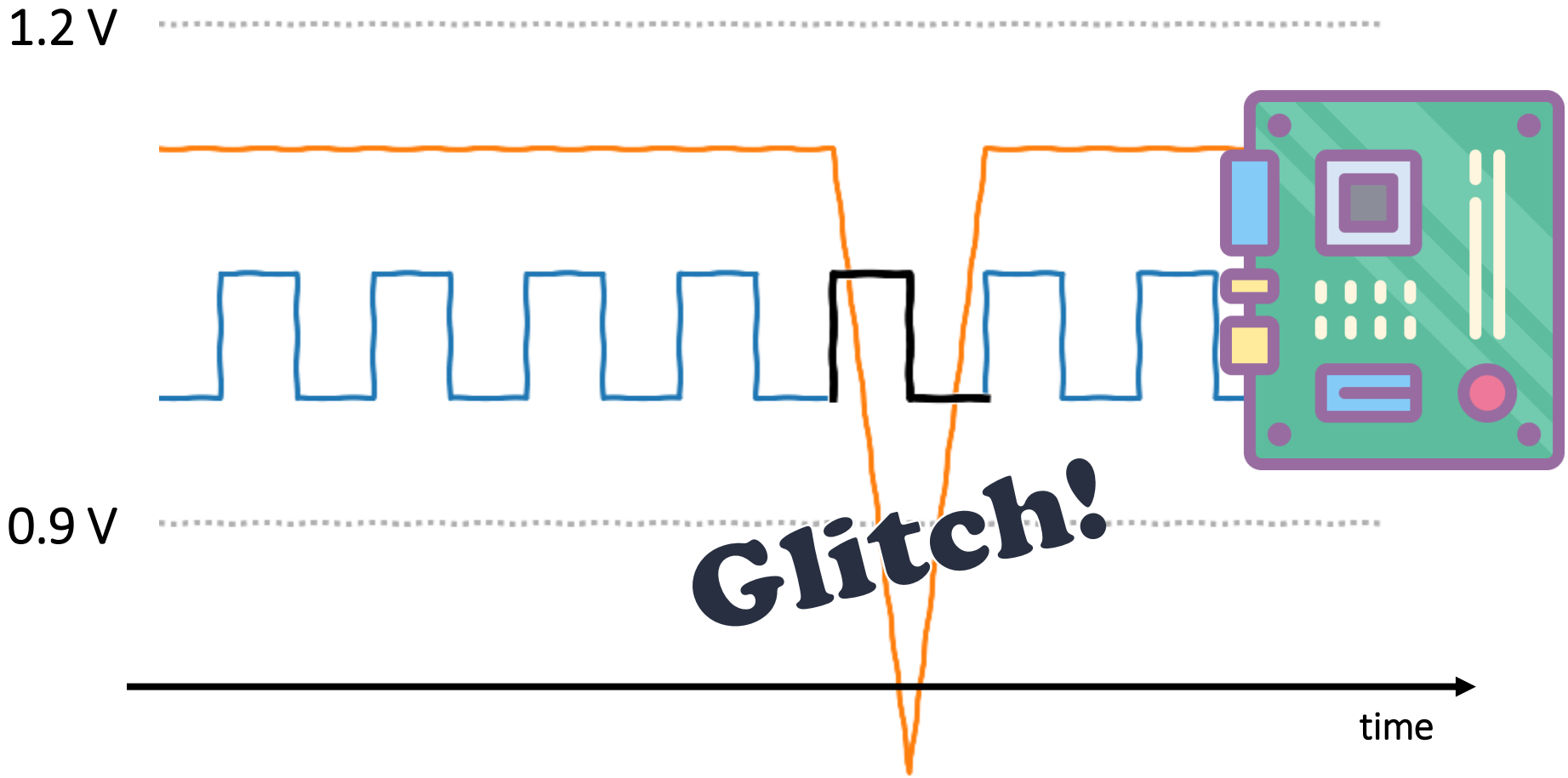
time



time


1.2 V
0.9 V
time


1.2 V
0.9 V
time


1.2 V
0.9 V
Glitch!
time

Let’s do this live on stage!

What could possibly go wrong....

Fault Injection Demo

Fault Injection Demo

BL1 ---------- U-Boot ----------▶

We do not modify U-Boot in flash.

Fault Injection Demo

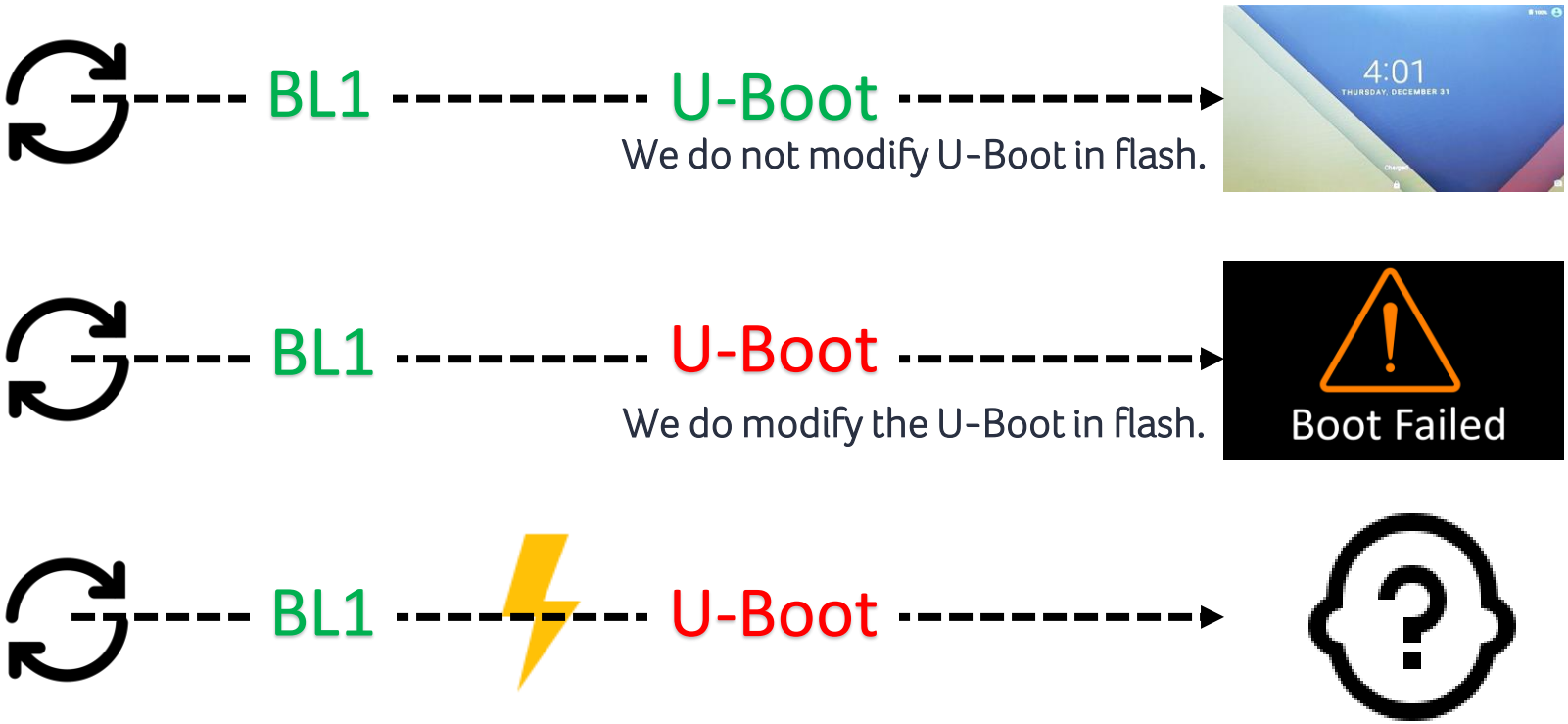
BL1 ---------- U-Boot ----------▶

We do not modify U-Boot in flash.

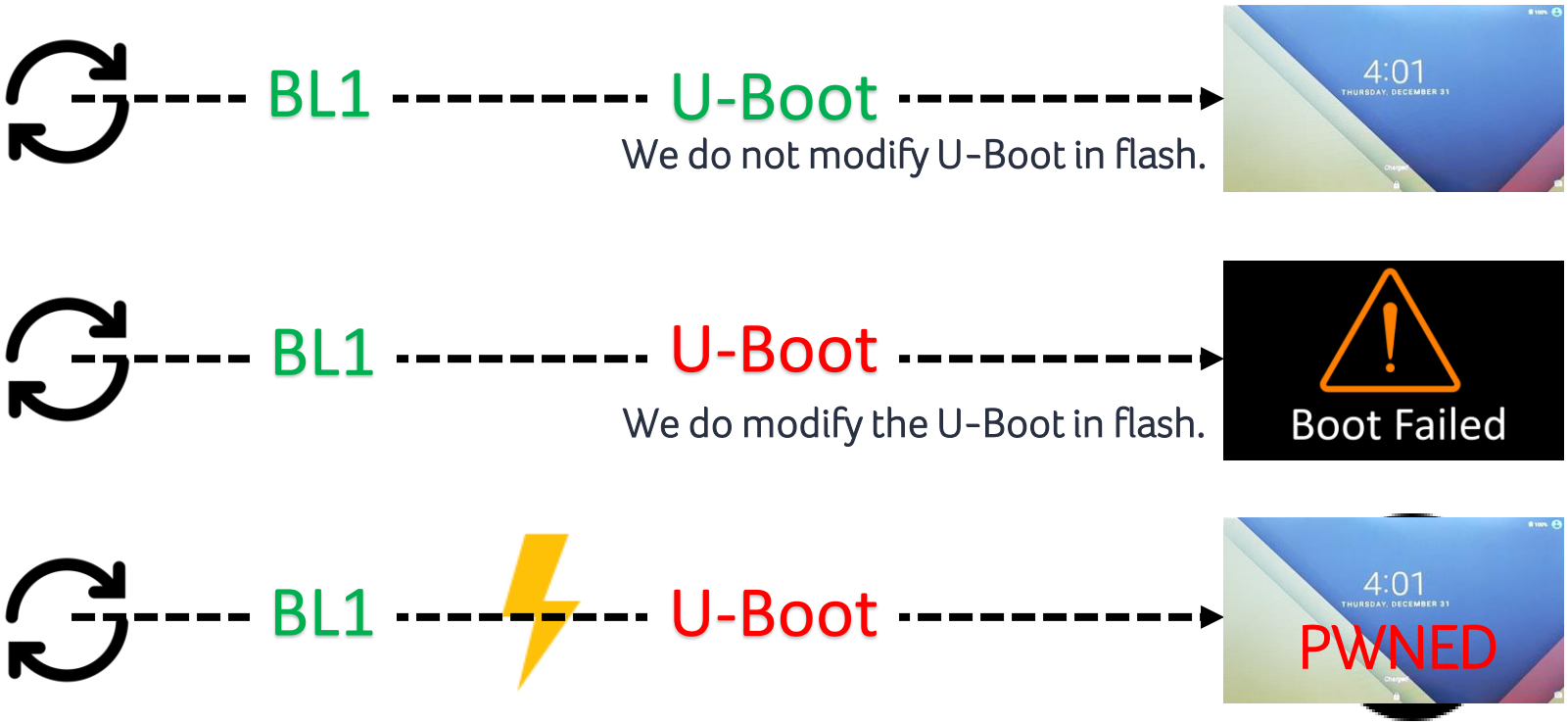
BL1 ---------- U-Boot ----------▶

We do modify the U-Boot in flash.

Fault Injection Demo



Fault Injection Demo



Successful Glitch!

id	vcc	glitch_vcc	glitch_delay	glitch_length	Data	Color
10	0.95	0.667265712035	66052	2416	S SB: 01/00 BL1 FIP Auth Boot BL3-1 ◆R◆◆O: Configuring TZ Controller NOTICE: [redacted] NOTICE: INFO: BL3-1: [redacted] [3.584]inter uboot shell uboot#	3
9	0.95	0.679053968415	66364	2401	S SB: 01/00 BL1 FIP Auth Auth FIP failed	4
8	0.95	0.67376919023	66761	2413	S SB: 01/00 BL1 FIP Auth Auth FIP failed	4
7	0.95	0.663793835686	66989	2400	S SB: 01/00 BL1 FIP Auth	5
6	0.95	0.679428856288	66946	2417	S SB: 01/00 BL1 FIP Auth Auth FIP failed	4
5	0.95	0.66061010504	66565	2405	S SB: 01/00 BL1 FIP Auth	5
4	0.95	0.674809077665	66223	2406	S SB: 01/00 BL1 FIP Auth	5
3	0.95	0.665179681471	66712	2413	S SB: 01/00 BL1 FIP Auth	5
2	0.95	0.667440366992	66417	2402	S SB: 01/00 BL1 FIP Auth	5
1	0.95	0.672698361993	66211	2415	S SB: 01/00 BL1 FIP Auth	5
0	0.95	0.661686836904	66532	2415	S SB: 01/00 BL1 FIP Auth Auth FIP failed	9

Want to know more? Please meet us after the talk!

Why does this work? What goes wrong?

Difficult to answer. But, behaviorally we can say a lot!

What can we do with our glitches?

What can we do with our glitches?

- Modify memory contents

What can we do with our glitches?

- Modify memory contents
- Modify register contents

What can we do with our glitches?

- Modify memory contents
- Modify register contents
- Modify the executed instructions !!!

What can we do with our glitches?

- Modify memory contents
- Modify register contents
- Modify the executed instructions !!!

We can change the intended behavior of software!

What about *unglitchable* hardware?

What about *unglitchable* hardware?

Yes. But... difficult & expensive.

What about using only software?

What about using only software?

Sure.

Typical Software FI Countermeasures*

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Typical Software FI Countermeasures*

- Redundant checks

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Typical Software FI Countermeasures*

- Redundant checks
- Defensive coding
  - e.g. initialize return values as ‘error’

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Typical Software FI Countermeasures*

- Redundant checks
- Defensive coding
  - e.g. initialize return values as ‘error’
- Code flow integrity
  - i.e. assure the code follows the intended path

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Typical Software FI Countermeasures*

- Redundant checks
- Defensive coding
  - e.g. initialize return values as ‘error’
- Code flow integrity
  - i.e. assure the code follows the intended path
- Random delays

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

Typical Software FI Countermeasures*

- Redundant checks
- Defensive coding
  - e.g. initialize return values as ‘error’
- Code flow integrity
  - i.e. assure the code follows the intended path
- Random delays

This sounds easy…

* https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf

It is not.

It is not.

```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

boot();
```

It is not.

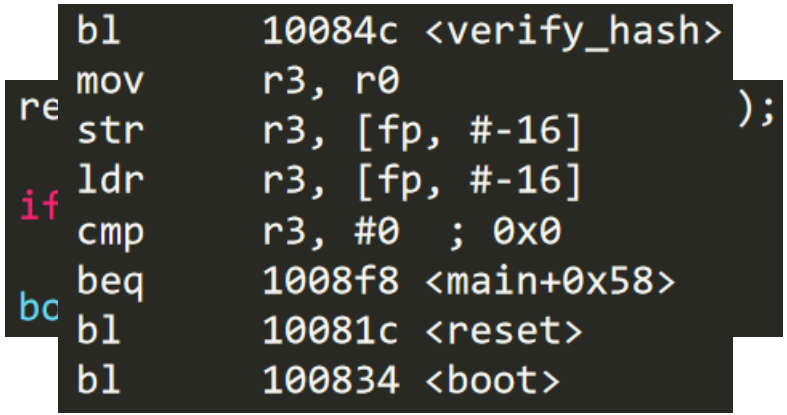
```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

boot();
```

```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

if(result) { reset(); }

boot();
```

Redundant checks needs multiple glitches?

Remember, we can modify instructions using glitches!

It is not.

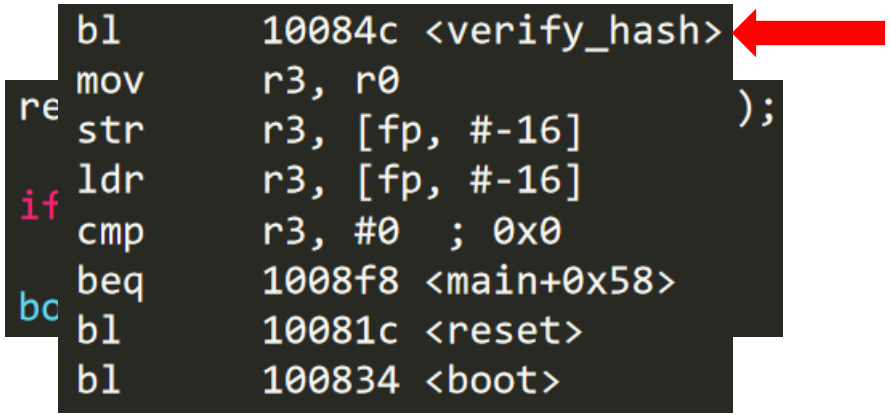



```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

if(result) { reset(); }

boot();
```

Redundant checks needs multiple glitches?

Remember, we can modify instructions using glitches!

It is not.

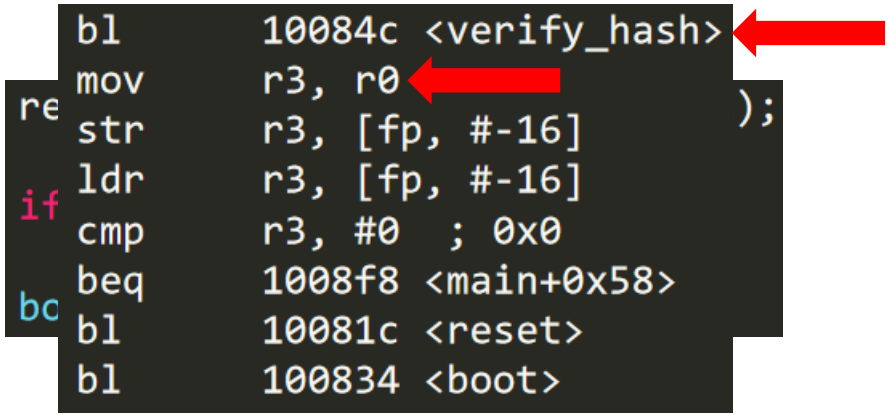



```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

if(result) { reset(); }

boot();
```

Redundant checks needs multiple glitches?

Remember, we can modify instructions using glitches!

It is not.

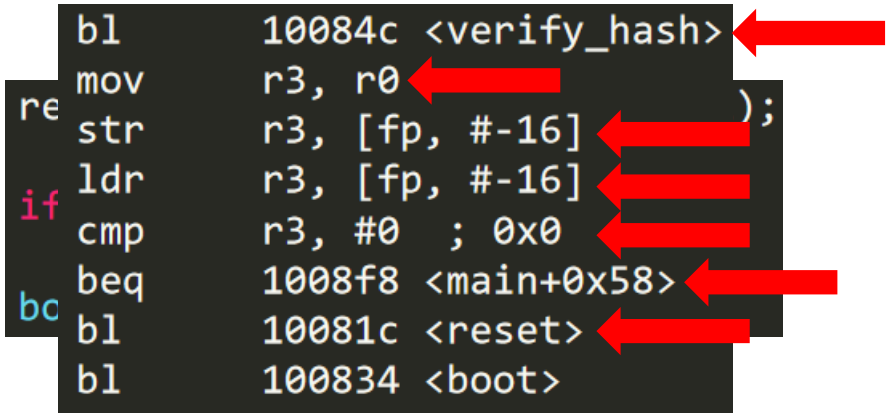



```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

if(result) { reset(); }

boot();
```

Redundant checks needs multiple glitches?

Remember, we can modify instructions using glitches!

It is not.




```
result = verify_hash(H1, H2, 32);

if(result) { reset(); }

if(result) { reset(); }

boot();
```

Redundant checks needs multiple glitches?

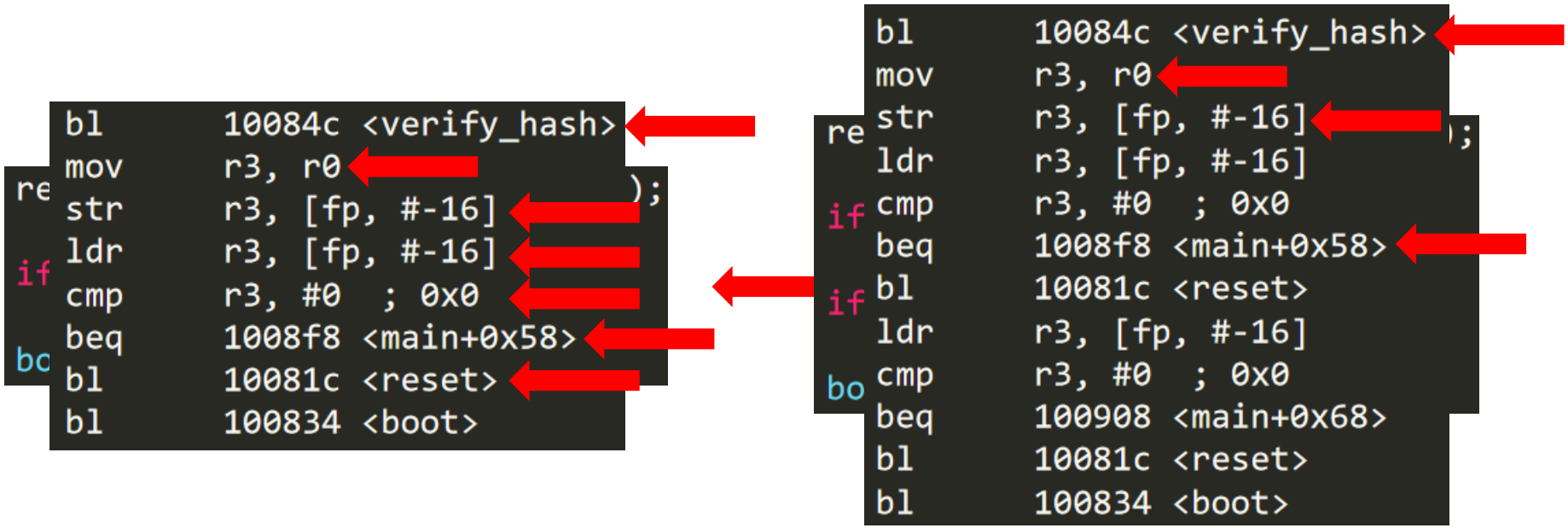
Remember, we can modify instructions using glitches!

It is not.

Redundant checks needs multiple glitches?

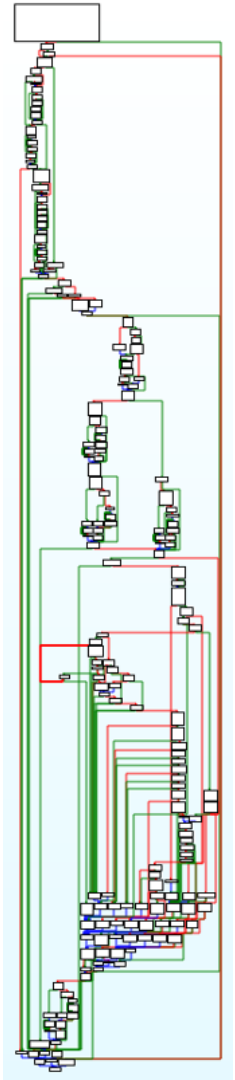
Remember, we can modify instructions using glitches!

It is not.



Redundant checks needs multiple glitches?

Remember, we can modify instructions using glitches!



Where can we bypass secure boot using a glitch?

We need automation to do this efficiently.

We?!?

The challenges of attackers & defenders are actually very similar!

Attackers vs Defenders

How can I glitch this device?

How do I know my glitch was succesfull?

How do I know where to glitch?

Which attack method is better for this target?

How can my code be attacked?

How can I make my code more robust?

How can I give an attacker as little information as possible?

What is the effect of these changes on the glitchability?

What is the effect of this type of glitches on my target?

Attackers vs Defenders

- No symbols, only the binary
- Limited knowledge / documentation of hardware

- Source code and a binary with symbols
- Documentation available

Attackers vs Defenders

- No symbols, only the binary
- Limited knowledge / documentation of hardware

- Source code and a binary with symbols
- Documentation available

Biggest difference:
Attackers need to reverse engineer the binary!

Our solution?

Our solution?

Simulation!

Simulation

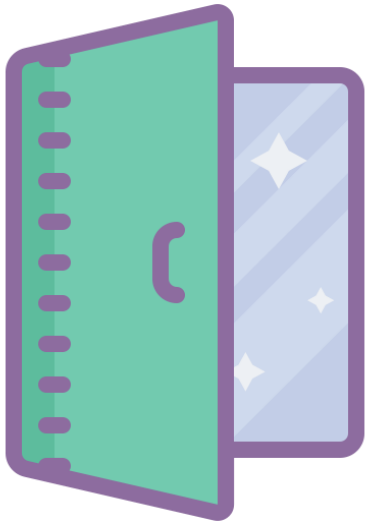
- Not a new idea!
- Several existing simulators already available.
- Nonetheless challenging to give useful results...

Simulation

- Not a new idea!
- Several existing simulators already available.
- Nonetheless challenging to give useful results...

Why? Bunch of challenges...

Challenge #1

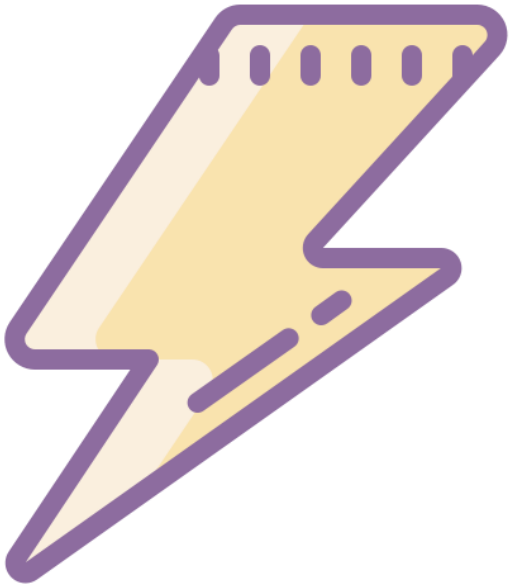


No hardware simulator = No fault simulator

Challenge #2

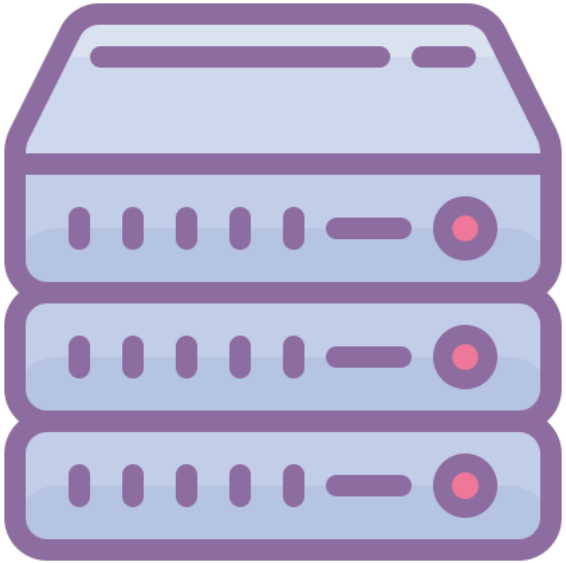
Changing the binary is no option.

Challenge #3



Detecting successful glitches.

Challenge #4



Using reasonable computational power.

Challenge #5

Realistic simulation.

What type of simulator do we use?

What type of simulator do we use?

- HDL simulator?

What type of simulator do we use?

- HDL simulator?
- Full system emulators? (Gem5, QEMU, ...)

What type of simulator do we use?

- HDL simulator?
- Full system emulators? (Gem5, QEMU, ...)
- Smartcard simulators ?!?...

What type of simulator do we use?

- HDL simulator?
- Full system emulators? (Gem5, QEMU, ...)
- Smartcard simulators ?!?...
- ???

What type of simulator do we use?

- HDL simulator?
- Full system emulators? (Gem5, QEMU, ...)
- Smartcard simulators ?!?...
- ???
- Our own?!?

Introduction to FiSim

- Main ideas
  - Shortest path to reasonable results
  - Speed over accuracy
  - Reusing existing components
  - Binary-based; can be used by attackers *and* defenders
- Glitches can be modelled by their observable effects in SW
  - Effects described through fault models

FiSim Features

- Unicorn & Capstone based
- Implements 2 realistic* fault models
  - Skipping individual instructions
  - Flipping a bit in the instruction encoding
  - Many more possible, easy to add

* https://www.riscure.com/uploads/2017/09/Controlling-PC-on-ARM-using-Fault-Injection.pdf

FiSim Features

- Unicorn & Capstone based
- Implements 2 realistic* fault models
  - Skipping individual instructions
  - Flipping a bit in the instruction encoding

} corruption

  - Many more possible, easy to add

* https://www.riscure.com/uploads/2017/09/Controlling-PC-on-ARM-using-Fault-Injection.pdf

FiSim Features

- Unicorn & Capstone based
- Implements 2 realistic* fault models
  - Skipping individual instructions
  - Flipping a bit in the instruction encoding
  - Many more possible, easy to add

} corruption

Instruction corruption

```
MOV R0, R1        11100001101000000000000000000000
MOV R0, R2        11100001101000000000000000000010
```

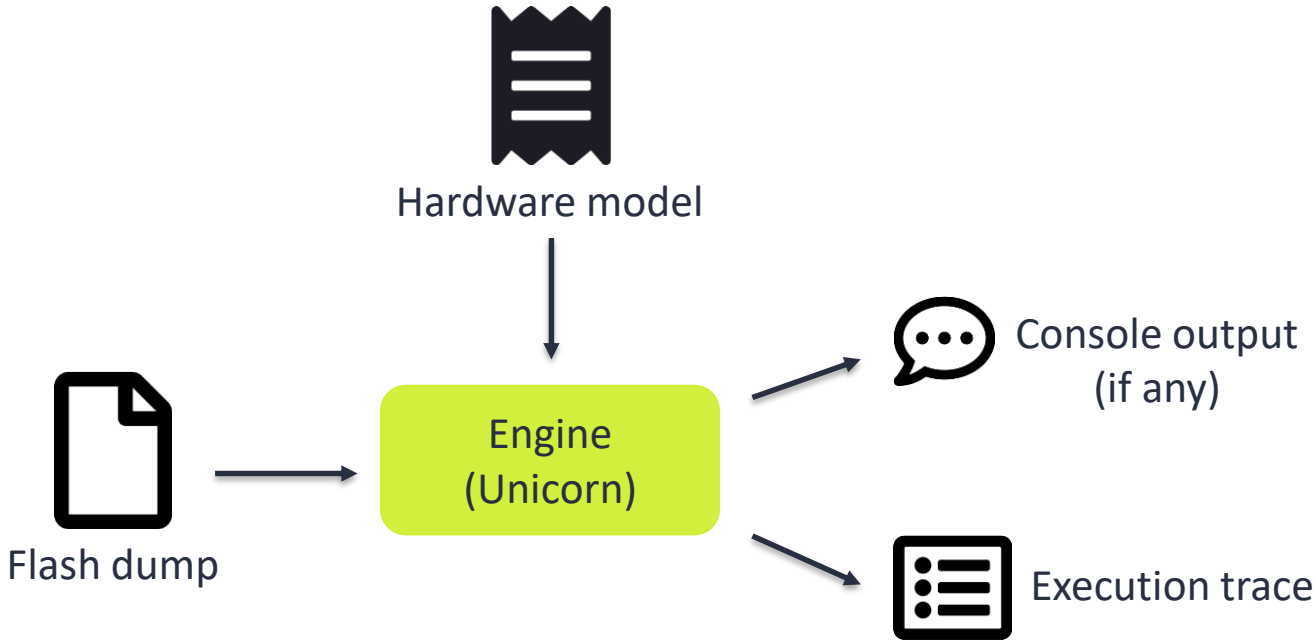
* https://www.riscure.com/uploads/2017/09/Controlling-PC-on-ARM-using-Fault-Injection.pdf

We tested several real bootloaders successfully!

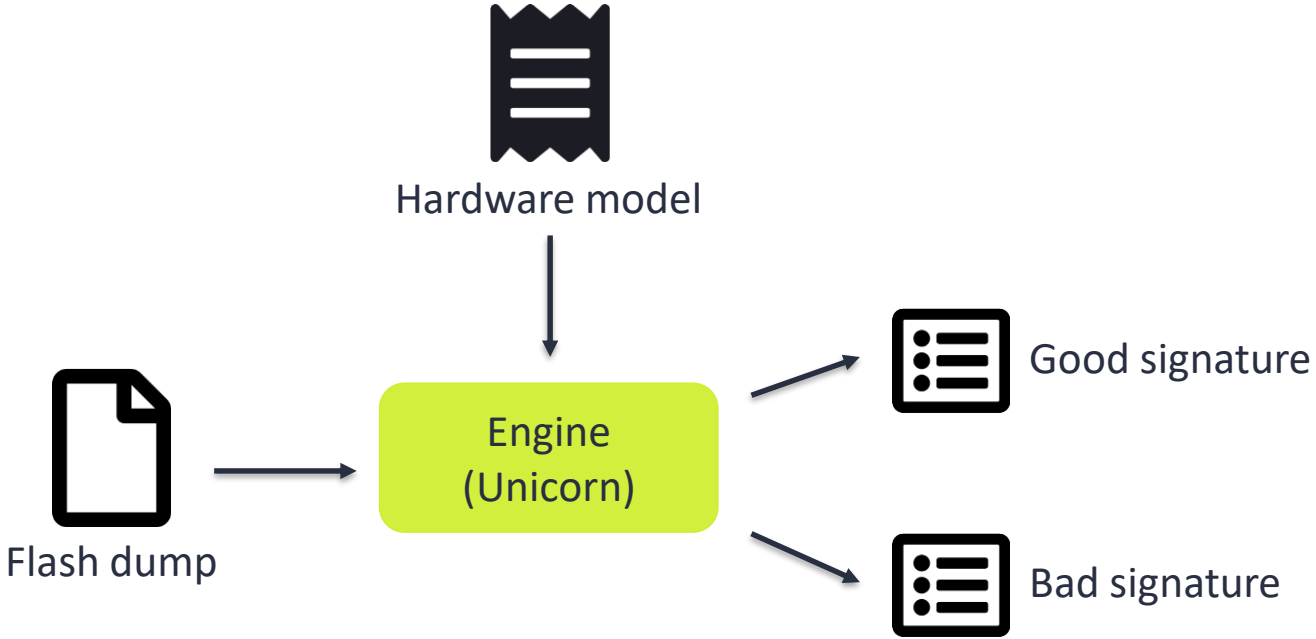
We tested several real bootloaders successfully!

Let’s dive into the architectural details...

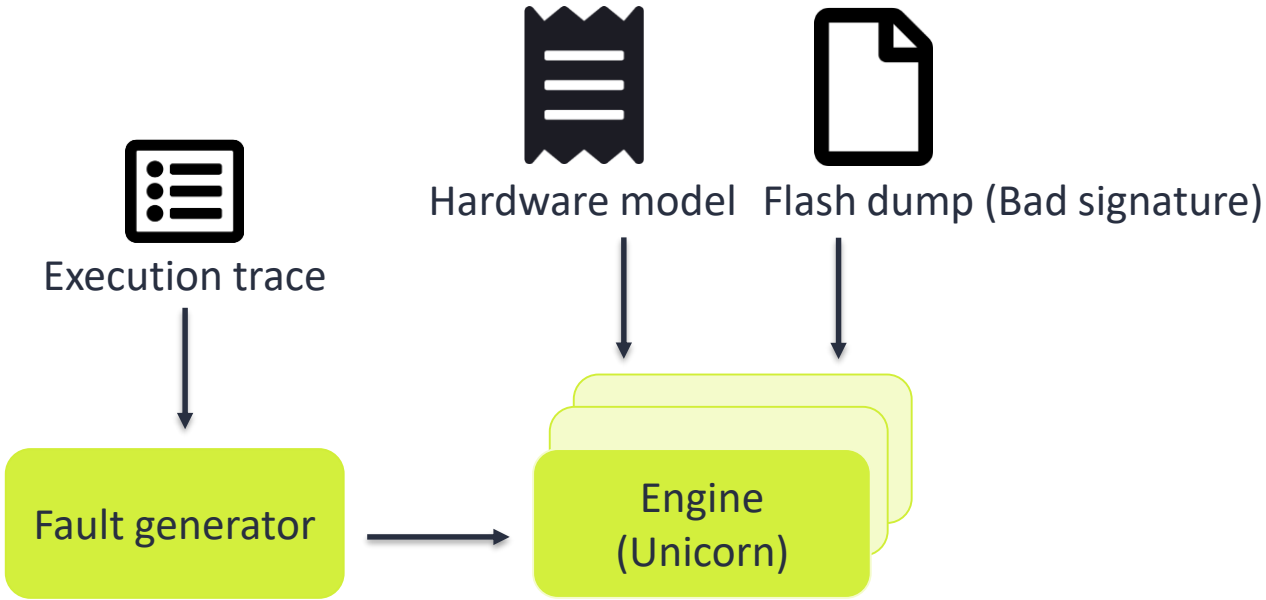
FiSim Architecture



FiSim Architecture



FiSim Architecture



Hardware Model

```
var simConfig = new Config {
    EntryPoint = binInfo.Symbols["_start"].Address,
    StackBase = 0x80100000,
    MaxInstructions = 2600000,
    AddressSpace = new AddressSpace {
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // mmc
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // gpio
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // uart
        {          , new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RWX } }, // BL31
        {          , new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RWX } }, // ???
        {          , new MemoryRegion { Size = 0x100000, Permission = MemoryPermission.RW } }, // fip / uboot
        {          , new MemoryRegion { Size = 0x100000, Permission = MemoryPermission.RW } }, // "HEAP"
        { 0x80000000, new MemoryRegion { Data = flashBin, Size = 0x10000, Permission = MemoryPermission.RWX } }, // Code
        { 0x80100000, new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RW } }, // Stack
        { 0x1234000, new HwPeripheral( onWrite: (eng, address, size, value) => { eng.RequestStop(Result.Failed); }) }, // Auth failed trigger
    },
    BreakPoints = {...},
    Patches = {...}
};
```

Hardware Model

```
var simConfig = new Config {
    EntryPoint = binInfo.Symbols["_start"].Address,
    StackBase = 0x80100000,
    MaxInstructions = 2600000,
    AddressSpace = new AddressSpace {
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // mmc
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // gpio
        {          , new MemoryRegion { Size = 0x1000, Permission = MemoryPermission.RW } }, // uart
        {          , new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RWX } }, // BL31
        {          , new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RWX } }, // ???
        {          , new MemoryRegion { Size = 0x100000, Permission = MemoryPermission.RW } }, // fip / uboot
        {          , new MemoryRegion { Size = 0x100000, Permission = MemoryPermission.RW } }, // "HEAP"
        { 0x80000000, new MemoryRegion { Data = flashBin, Size = 0x10000, Permission = MemoryPermission.RWX } }, // Code
        { 0x80100000, new MemoryRegion { Size = 0x10000, Permission = MemoryPermission.RW } }, // Stack
        { 0x1234000, new HwPeripheral( onWrite: (eng, address, size, value) => { eng.RequestStop(Result.Failed); }) }, // Auth failed trigger
    },
    BreakPoints = {...},
    Patches = {...}
};
```

```
public override void OnWrite(SimulatorEngine engine, ulong address, uint size, ulong value) {
    var stateData = engine.GetDeviceState<StateData>(this);

    if ((address & 0xFFF) == 0) { // src
        stateData.SrcAddr = value;

        stateData.State |= State.SrcSet;
    }
    else if ((address & 0xFFF) == 4) { // dst
        stateData.DstAddr = value;

        stateData.State |= State.DstSet;
    }
    else if ((address & 0xFFF) == 8) { // key
        stateData.KeyAddr = value;

        stateData.State |= State.KeySet;
    }
    else if ((address & 0xFFF) == 12) { // algo
        stateData.Algorithm = (CryptoAlgorithm)value;
    }
    else if ((address & 0xFFF) == 13) { // mode
    }
    else if ((address & 0xFFF) == 14) { // start
        if (stateData.State == State.Ready) {
            byte[] cipherText;
            byte[] key;

            if (stateData.Algorithm == CryptoAlgorithm.AES) {
                cipherText = new byte[16];
                key = new byte[16];
            }
            else if (stateData.Algorithm == CryptoAlgorithm.RSA) {
                cipherText = new byte[128];
                key = new byte[162];
            }
            else {
                throw new NotSupportedDeviceOperationException(engine, "Unknown crypto algorithm");
            }

            engine.MemRead(stateData.SrcAddr, cipherText);
            engine.MemRead(stateData.KeyAddr, key);

            if (stateData.Algorithm == CryptoAlgorithm.AES) {
                var plaintext = _decryptAes(cipherText, key);

                engine.MemWrite(stateData.DstAddr, plaintext);
            }
            else if (stateData.Algorithm == CryptoAlgorithm.RSA) {
                try {
                    var plaintext = _decryptRsa(cipherText, key);

                    engine.MemWrite(stateData.DstAddr, plaintext);
                }
                catch (Exception e) {
                    throw new InvalidDeviceOperationException(engine, "RSA decryption failed", e);
                }
            }
            else {
                throw new NotSupportedDeviceOperationException(engine, "Unknown crypto algorithm");
            }
        }
        else {
            throw new InvalidDeviceOperationException(engine, "Engine not ready");
        }
    }
```

Hardware Model

```
var simConfig = new Config {
    EntryPoint = binInfo.Symbols["_start"].Address,
    StackBase = 0x80100000,
    MaxInstructions = 2600000,
    AddressSpace = new AddressSpace {...},
    BreakPoints = {...},
    Patches = {
        { binInfo.Symbols["serial_init"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["msdelay"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["mmc_clk_init"].Address, AArch32Info.A32_MOV_R0_0_RET },
        { binInfo.Symbols["mmc_read"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["pmc_release"].Address, AArch32Info.A32_RET },
        /* ... */
    }
};
```

Hardware Model

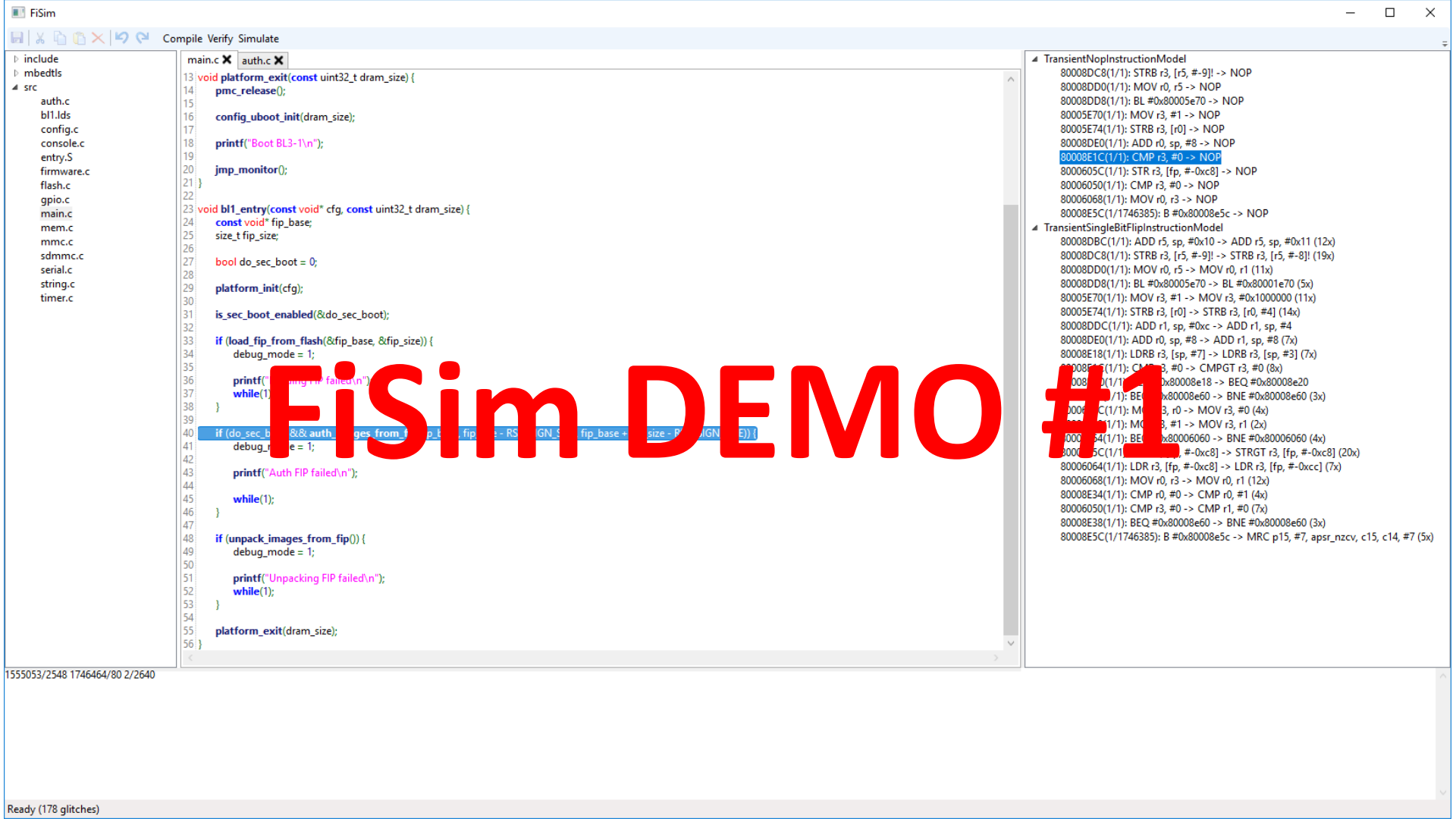
```
var simConfig = new Config {
    EntryPoint = binInfo.Symbols["_start"].Address,
    StackBase = 0x80100000,
    MaxInstructions = 2600000,
    AddressSpace = new AddressSpace {...},
    BreakPoints = {...},
    Patches = {
        { binInfo.Symbols["serial_init"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["msdelay"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["mmc_clk_init"].Address, AArch32Info.A32_MOV_R0_0_RET },
        { binInfo.Symbols["mmc_read"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["pmc_release"].Address, AArch32Info.A32_RET },
        /* ... */
    }
};
```

Hardware Model

```
var simConfig = new Config {
    EntryPoint = binInfo.Symbols["_start"].Address,
    StackBase = 0x80100000,
    MaxInstructions = 2600000,
    AddressSpace = new AddressSpace {...},
    BreakPoints = {...},
    Patches = {
        { binInfo.Symbols["serial_init"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["msdelay"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["mmc_clk_init"].Address, AArch32Info.A32_MOV_R0_0_RET },
        { binInfo.Symbols["mmc_read"].Address, AArch32Info.A32_RET },
        { binInfo.Symbols["pmc_release"].Address, AArch32Info.A32_RET },
        /* ... */
    }
};
```

Note: attacker needs to hardcode addresses!

```
BreakPoints = {
    { binInfo.Symbols["config_init"].Address, eng => { eng.RegWrite(Arm.UC_ARM_REG_R0, value: 0x80100000); } }, // DRAM cfg
    { binInfo.Symbols["mmc_read"].Address, eng => {
        // mmc_bread(..., unsigned long start, unsigned blkcnt, void *dst)
        var start = eng.RegRead(Arm.UC_ARM_REG_R1);
        var blkcnt = eng.RegRead(Arm.UC_ARM_REG_R2);
        var dst = eng.RegRead(Arm.UC_ARM_REG_R3);

        var offset = (start * 512);
        var len = blkcnt * 512;

        if (offset + len > (ulong) fipBin.Length) {
            throw new Exception( message: "Try to read more from MMC than is provided in FIP");
        }

        var data = new byte[len];
        Array.Copy( sourceArray: fipBin, sourceIndex: (long)offset, destinationArray: data, destinationIndex: 0, length: (long)len );

        eng.Write(dst, data);

        eng.RegWrite(Arm.UC_ARM_REG_R0, blkcnt);
    } },

    { binInfo.Symbols["jmp_monitor"].Address, eng => {
        var result = eng.Compare( address: 0x40000000,
            expectedData: new byte[] { 0x0C, 0x00, 0x00, 0x14, 0x6D, 0x6F, 0x6E, 0x69, 0x74, 0x6F, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00 })
            ? Result.Completed
            : Result.Failed;

        eng.RequestStop(result);
    } },

    /* ... */
```


FiSim
Compile Verify Simulate
include
mbedtls
src
auth.c
bl1.lds
config.c
console.c
entry.S
firmware.c
flash.c
gpio.c
main.c
mem.c
mmc.c
sdmmc.c
serial.c
string.c
timer.c
main.c
auth.c
void platform_exit(const uint32_t dram_size) {
pmc_release();
config_uboot_init(dram_size);
printf("Boot BL3-1\n");
jmp_monitor();
}
void bl1_entry(const void* cfg, const uint32_t dram_size) {
const void* fip_base;
size_t fip_size;
bool do_sec_boot = 0;
platform_init(cfg);
is_sec_boot_enabled(&do_sec_boot);
if (load_fip_from_flash(&fip_base, &fip_size)) {
debug_mode = 1;
printf("
while(1)
}
if (do_sec_b
debug_
printf("Auth FIP failed\n");
while(1);
}
if (unpack_images_from_fip()) {
debug_mode = 1;
printf("Unpacking FIP failed\n");
while(1);
}
platform_exit(dram_size);
}
FiSim DEMO #1
TransientNopInstructionModel
80008DC8(1/1): STRB r3, [r5, #-9]! -> NOP
80008DD0(1/1): MOV r0, r5 -> NOP
80008DD8(1/1): BL #0x80005e70 -> NOP
80005E70(1/1): MOV r3, #1 -> NOP
80005E74(1/1): STRB r3, [r0] -> NOP
80008DE0(1/1): ADD r0, sp, #8 -> NOP
80008E1C(1/1): CMP r3, #0 -> NOP
8000605C(1/1): STR r3, [fp, #-0xc8] -> NOP
80006050(1/1): CMP r3, #0 -> NOP
80006068(1/1): MOV r0, r3 -> NOP
80008E5C(1/1746385): B #0x80008e5c -> NOP
TransientSingleBitFlipInstructionModel
80008DBC(1/1): ADD r5, sp, #0x10 -> ADD r5, sp, #0x11 (12x)
80008DC8(1/1): STRB r3, [r5, #-9]! -> STRB r3, [r5, #-8]! (19x)
80008DD0(1/1): MOV r0, r5 -> MOV r0, r1 (11x)
80008DD8(1/1): BL #0x80005e70 -> BL #0x80001e70 (5x)
80005E70(1/1): MOV r3, #1 -> MOV r3, #0x1000000 (11x)
80005E74(1/1): STRB r3, [r0] -> STRB r3, [r0, #4] (14x)
80008DDC(1/1): ADD r1, sp, #0xc -> ADD r1, sp, #4
80008DE0(1/1): ADD r0, sp, #8 -> ADD r1, sp, #8 (7x)
80008E18(1/1): LDRB r3, [sp, #7] -> LDRB r3, [sp, #3] (7x)
80008E1C(1/1): CMP r3, #0 -> CMPGT r3, #0 (8x)
80008E20(1/1): BEQ #0x80008e18 -> BEQ #0x80008e20
80008E60(1/1): BEQ #0x80008e60 -> BNE #0x80008e60 (3x)
80006050(1/1): MOV r3, r0 -> MOV r3, #0 (4x)
80006060(1/1): MOV r3, #1 -> MOV r3, r1 (2x)
80006064(1/1): BEQ #0x80006060 -> BNE #0x80006060 (4x)
8000605C(1/1): STR r3, [fp, #-0xc8] -> STRGT r3, [fp, #-0xc8] (20x)
80006064(1/1): LDR r3, [fp, #-0xc8] -> LDR r3, [fp, #-0xcc] (7x)
80006068(1/1): MOV r0, r3 -> MOV r0, r1 (12x)
80008E34(1/1): CMP r0, #0 -> CMP r0, #1 (4x)
80006050(1/1): CMP r3, #0 -> CMP r1, #0 (7x)
80008E38(1/1): BEQ #0x80008e60 -> BNE #0x80008e60 (3x)
80008E5C(1/1746385): B #0x80008e5c -> MRC p15, #7, apsr_nzcv, c15, c14, #7 (5x)
1555053/2548 1746464/80 2/2640
Ready (178 glitches)

What did we glitch in the first demo?

What did we glitch in the first demo?

Who knows??!

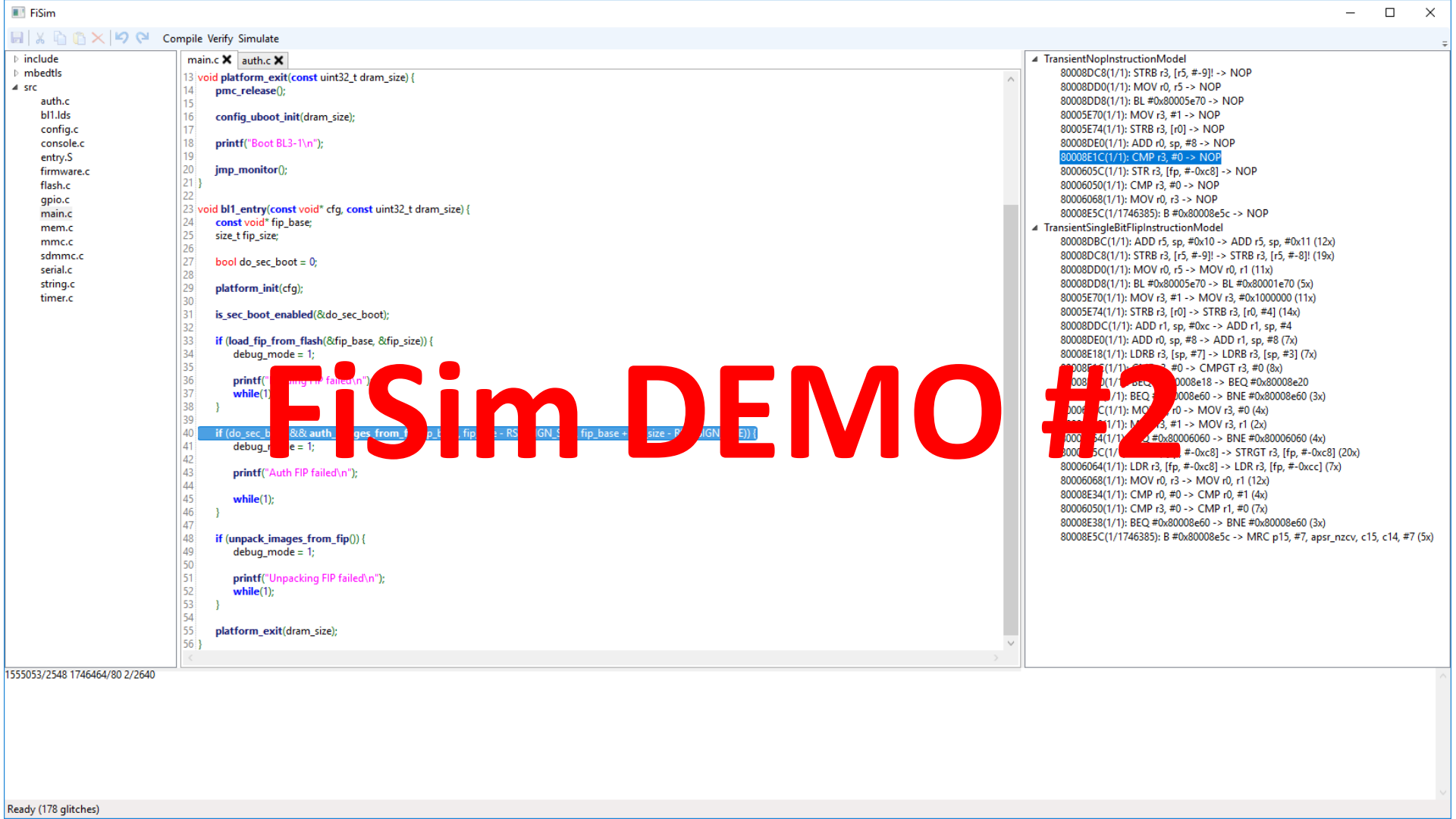
What did we glitch in the first demo?

Many possibilities....

Let’s harden our bootloader...

Let’s harden our bootloader...

What if we authenticate twice?


FiSim
Compile Verify Simulate
include
mbedtls
src
auth.c
bl1.lds
config.c
console.c
entry.S
firmware.c
flash.c
gpio.c
main.c
mem.c
mmc.c
sdmmc.c
serial.c
string.c
timer.c
main.c
auth.c
void platform_exit(const uint32_t dram_size) {
pmc_release();
config_uboot_init(dram_size);
printf("Boot BL3-1\n");
jmp_monitor();
}
void bl1_entry(const void* cfg, const uint32_t dram_size) {
const void* fip_base;
size_t fip_size;
bool do_sec_boot = 0;
platform_init(cfg);
is_sec_boot_enabled(&do_sec_boot);
if (load_fip_from_flash(&fip_base, &fip_size)) {
debug_mode = 1;
printf("
while(1)
}
if (do_sec_b
debug_
printf("Auth FIP failed\n");
while(1);
}
if (unpack_images_from_fip()) {
debug_mode = 1;
printf("Unpacking FIP failed\n");
while(1);
}
platform_exit(dram_size);
}
FiSim DEMO #2
TransientNopInstructionModel
80008DC8(1/1): STRB r3, [r5, #-9]! -> NOP
80008DD0(1/1): MOV r0, r5 -> NOP
80008DD8(1/1): BL #0x80005e70 -> NOP
80005E70(1/1): MOV r3, #1 -> NOP
80005E74(1/1): STRB r3, [r0] -> NOP
80008DE0(1/1): ADD r0, sp, #8 -> NOP
80008E1C(1/1): CMP r3, #0 -> NOP
8000605C(1/1): STR r3, [fp, #-0xc8] -> NOP
80006050(1/1): CMP r3, #0 -> NOP
80006068(1/1): MOV r0, r3 -> NOP
80008E5C(1/1746385): B #0x80008e5c -> NOP
TransientSingleBitFlipInstructionModel
80008DBC(1/1): ADD r5, sp, #0x10 -> ADD r5, sp, #0x11 (12x)
80008DC8(1/1): STRB r3, [r5, #-9]! -> STRB r3, [r5, #-8]! (19x)
80008DD0(1/1): MOV r0, r5 -> MOV r0, r1 (11x)
80008DD8(1/1): BL #0x80005e70 -> BL #0x80001e70 (5x)
80005E70(1/1): MOV r3, #1 -> MOV r3, #0x1000000 (11x)
80005E74(1/1): STRB r3, [r0] -> STRB r3, [r0, #4] (14x)
80008DDC(1/1): ADD r1, sp, #0xc -> ADD r1, sp, #4
80008DE0(1/1): ADD r0, sp, #8 -> ADD r1, sp, #8 (7x)
80008E18(1/1): LDRB r3, [sp, #7] -> LDRB r3, [sp, #3] (7x)
80008E1C(1/1): CMP r3, #0 -> CMPGT r3, #0 (8x)
80008E20(1/1): BEQ #0x80008e18 -> BEQ #0x80008e20
80008E60(1/1): BEQ #0x80008e60 -> BNE #0x80008e60 (3x)
80006050(1/1): MOV r0, r0 -> MOV r3, #0 (4x)
80006060(1/1): MOV r3, #1 -> MOV r3, r1 (2x)
80006064(1/1): B #0x80006060 -> BNE #0x80006060 (4x)
8000605C(1/1): -0xc8] -> STRGT r3, [fp, #-0xc8] (20x)
80006064(1/1): LDR r3, [fp, #-0xc8] -> LDR r3, [fp, #-0xcc] (7x)
80006068(1/1): MOV r0, r3 -> MOV r0, r1 (12x)
80008E34(1/1): CMP r0, #0 -> CMP r0, #1 (4x)
80006050(1/1): CMP r3, #0 -> CMP r1, #0 (7x)
80008E38(1/1): BEQ #0x80008e60 -> BNE #0x80008e60 (3x)
80008E5C(1/1746385): B #0x80008e5c -> MRC p15, #7, apsr_nzcv, c15, c14, #7 (5x)
1555053/2548 1746464/80 2/2640
Ready (178 glitches)

Limitations / Future work

- Is instruction corruption the only fault model?
  - We do not know…
  - Other fault models likely applicable too!
- What is the impact of instruction / data caches?

Limitations / Future work

- Is instruction corruption the only fault model?
  - We do not know…
  - Other fault models likely applicable too!
- What is the impact of instruction / data caches?

Testing remains critical!

Takeaways

Takeaways

- Fault attacks are effective to bypass secure boot

Takeaways

- Fault attacks are effective to bypass secure boot
- Simulating is effective for attackers and defenders

Takeaways

- Fault attacks are effective to bypass secure boot
- Simulating is effective for attackers and defenders
- Actual testing still required for assurance

Thank you! Any questions?

Or come to us...

Martijn Bogaard
Senior Security Analyst
martijn@riscure.com / @jmartijnb

Niek Timmers
Principal Security Analyst
niek@riscure.com / @tieknimmers