

MASTER IN

**CYBER—
SECURITY**

**UNLOCK
YOUR POTENTIAL**

Lab 2: Fault Injection simulation and countermeasures



UNIVERSIDAD
DE MURCIA



Facultad de
Informática



Introduction

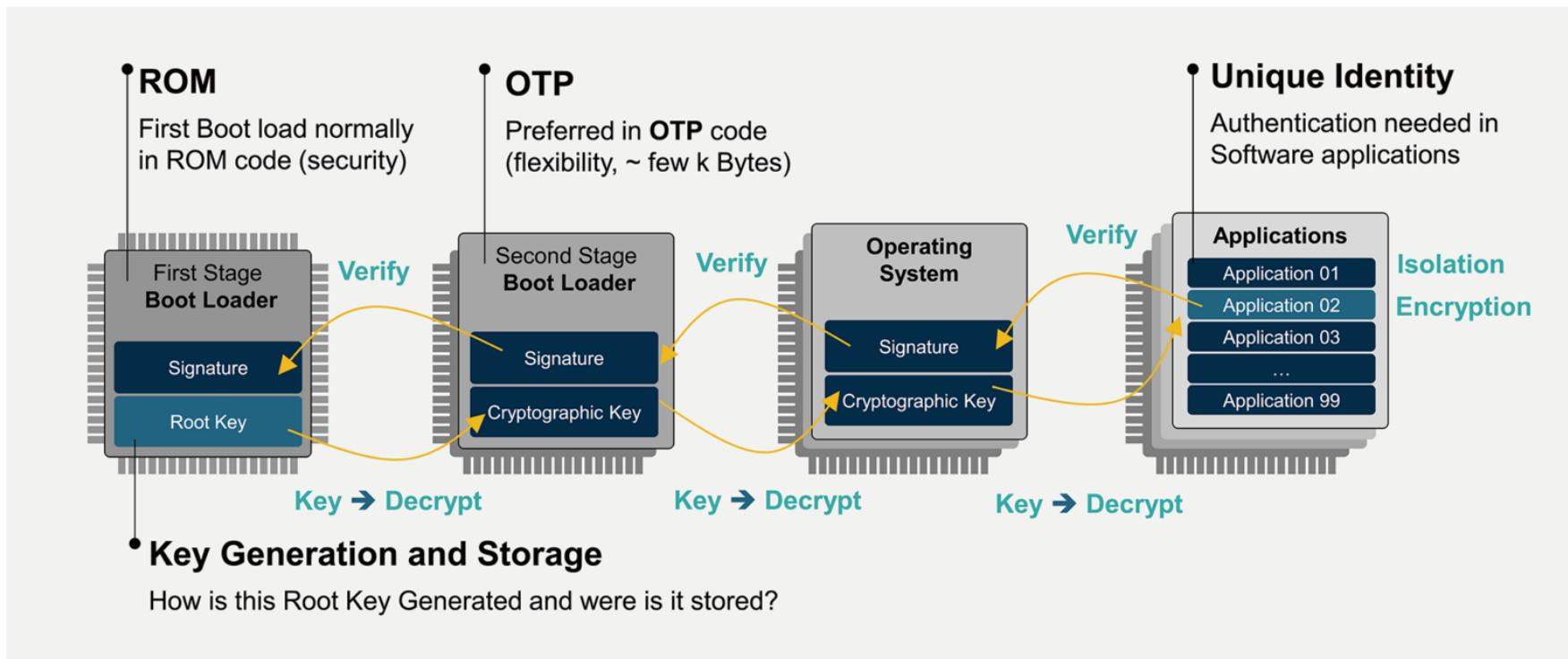


- The aim of this lab is to introduce fault injection and its countermeasures.
- Concretely we will use a simulator for fault injection attacks used with bootloader code
- You will modify the code to introduce countermeasures and analyse the results.

Secure Boot

- The target code will be a simplified bootloader operating during a secure boot process

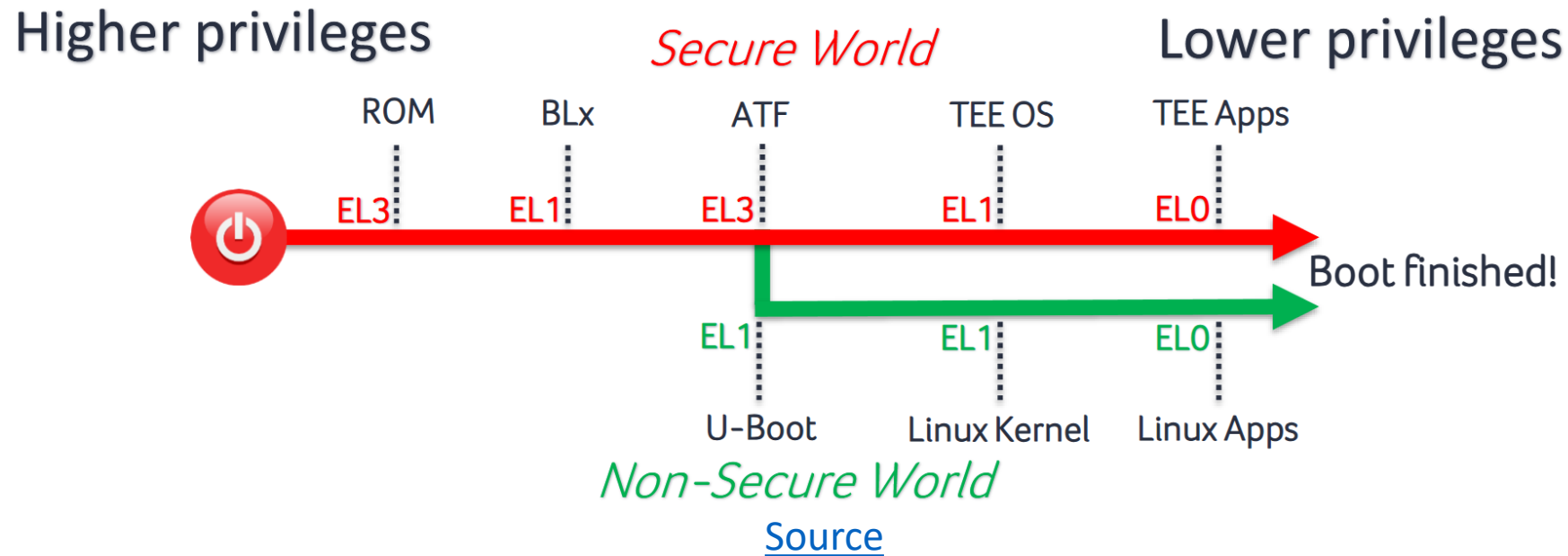
Secure Boot Flow Four Sages



Source <https://www.pufsecurity.com/technology/secure-boot/>

Secure Boot

- The target code will be a simplified bootloader operating during a secure boot process



Fault injection

- Intentionally cause errors in a system in order to compromise the security of the system.
 - Modifications on the contents of memory, registers or the executed instructions.
- The attack may result in glitches following different models, we will consider two:
 - A fault/glitch causes the micro-controller to skip an instruction.
 - E.g., `beq 1008f8` may be completely skipped (substituted for `nop`).
 - Flipping a bit in the encoding of a specific instruction.
 - E.g., `cmp r3, #0` may be changed into `cmp r3, \#1`.
 - Note this also may affect the instruction itself, not only constant/register values

Fault injection software countermeasures

- **Redundant operations:** The main purpose of performing redundant operations (e.g., check) is to force an attacker to successfully perform multiple successive glitches.
- **Defensive coding:**
 - **Non-trivial constants:** Using “magic numbers” for constants, enums... (good practices are to include a good mix of 1s and 0s, high hamming distances...) and “closed-on-fail” principles.
 - **Initialize to error values:** The variables before a check is initially set to an error value, avoiding some simple attacks by e.g. jumping instructions.
 - **Error by default:** If an unexpected event happens, then a fault can be assumed and an error notified (default switch branch, “unreachable” branch...)
- **Timing countermeasures:** Not a direct countermeasure to the fault injection effect (will not be seen in the simulator). Fault attacks are timing critical, so introducing randomized access, random delays... makes it more difficult to target the glitch.

Fault injection software countermeasures

- **Fault detection test:** It attempts to detect fault injections by verifying that some arithmetic invariants are respected, and branch to return an error instead of the numerical result of the algorithm in case of invariant violation.
- **Infective computing:** It uses arithmetic invariants to compute a neutral element (that is not valid if fault happens) that is used to infect the result of the algorithm before returning it to make it unusable by the attacker.
- **Control flow integrity:** Advanced yet with limited applicability technique based on introducing checks that the correct execution flow of the program has occurred, e.g. through CRCs.

Tools

- The FiSim simulator

Riscure FiSim (demo edition)

Clean Build Verify Simulate

bin
include
src
Makefile.AArch32

```
main.c
19: bool is_sec_boot_en;
20:
21: void* img_base = IMG_LOAD_ADDR;
22: size_t img_size;
23:
24: otp_init();
25:
26: otp_is_sec_boot_enabled(&is_sec_boot_en);
27:
28: if (is_sec_boot_en) {
29:     goto do_boot;
30: }
31:
32: serial_puts("Start Secure Boot...\n");
33:
34: otp_get_img_hash(otp_img_hash);
35:
36: flash_load_img(img_base, &img_size);
37:
38: MD5_Init(&ctx);
39: MD5_Update(&ctx, img_base, img_size);
40: MD5_Final(img_hash, &ctx);
41:
42: if (is_sec_boot_en && memcmp(img_hash, otp_img_hash, sizeof(otp_img_hash))) {
43:     serial_puts("Auth failed!\n");
44:     _SET_SIM_FAILED();
45: }
46:
47: do_boot:
48:     serial_puts("Boot next stage\n");
49:
50:     boot_next_stage();
51: }
```

C:\Users\Jesus\Downloads\FiSim.2020-11.win\Content\Example\bin\aaarch32\bl1.elf

Verify functional behavior...

Start Secure Boot...

Boot next stage

Start Secure Boot...

Auth failed!

Verification finished. Bootloader is verified to work as intended.

Tools

- The FiSim simulator

Riscure FiSim (demo edition)

Clean Build Verify Simulate

bin
include
src
Makefile.AArch32

```

main.c
19: bool is_sec_boot_en;
20:
21: void* img_base = IMG_LOAD_ADDR;
22: size_t img_size;
23:
24: otp_init();
25:
26: otp_is_sec_boot_enabled(&is_sec_boot_en);
27:
28: if (is_sec_boot_en) {
29:     goto do_boot;
30: }
31:
32: serial_puts("Start Secure Boot...\n");
33:
34: otp_get_img_hash(otp_img_hash);
35:
36: flash_load_img(img_base, &img_size);
37:
38: MD5_Init(&ctx);
39: MD5_Update(&ctx, img_base, img_size);
40: MD5_Final(img_hash, &ctx);
41:
42: if (is_sec_boot_en && memcmp(img_hash, otp_img_hash, sizeof(otp_img_hash))) {
43:     serial_puts("Auth failed!\n");
44:     __SET_SIM_FAILED();
45: }
46:
47: do_boot:
48:     serial_puts("Boot next stage\n");
49:
50:     boot_next_stage();
51: }

```

TransientNopInstructionModel

80001278: BL #0x80000410 -> NOP
8000042C: STR r2, [r3, #4] -> NOP
80001280: BL #0x800003b0 -> NOP
800003B8: MOV r4, r0 -> NOP
800003C8: BL #0x8000032c -> NOP
800003C4: MOV r0, #0x10 -> NOP
80000348(1/5): TST r3, #2 -> NOP
80000358(1/5): STR r0, [r3, #8] -> NOP
80000360(1/5): STR r2, [r3, #4] -> NOP
80000344(1/5): LDR r3, [r3] -> NOP
80000354(1/5): MOV r2, #2 -> NOP
80000390(1/5): STR r3, [r1] -> NOP
800003CC: CMP r0, #0 -> NOP
8000038C(1/5): LDR r3, [r3, #0xc] -> NOP
800003D4: UBFXNE r3, r3, #0x11, #1 -> NOP
800003D8: STRBNE r3, [r4] -> NOP
80001264: POP {fp} -> NOP
80000498: POP {fp} -> NOP
800012E0: CMP r3, #0 -> NOP
800012E4: BNE #0x800012f8 -> NOP
800002FC: BNE #0x800002e8 -> NOP

TransientSingleBitFlipInstructionModel

80000000: LDR r4, [pc, #0x24] -> LDR r4, [pc, #0x24
80000004: LDR r5, [pc, #0x24] -> LDR r5, [pc, #0x24
8000000C: BEQ #0x80000024 -> BEQ #0x8000002c
80000024: BL #0x8000126c -> BL #0x8000128c (2x)
80001278: BL #0x80000410 -> BLGT #0x80000410 (2
80000418: MOV r3, #0x12000000 -> MOV r3, #0x320
8000042C: STR r2, [r3, #4] -> STR r2, [r3] (8x)
80000428: MOV r2, #1 -> ADC r2, r0, #1 (2x)
8000127C: SUB r0, fp, #0xc1 -> SUB r0, pc, #0xc1 (8

C:\Users\Jesus\Downloads\FiSim.2020-11.win\Content\Example\bin\aaarch32\bl1.elf

Starting simulation... This will take several minutes.

Start correct sign trace

Start wrong sign trace

Finished correct sign trace

Finished wrong sign trace

1760/1099 1651/1097 2/54483

Exercise and report

- Modify the bootloader implementation to introduce countermeasures to the discovered fault attacks
 - At least 5
 - The countermeasures must be varied in their approach redundancy, constants and initialization, respond to faults detected/errors by default...
 - Tackling different parts of the code will also be valued.
- For each countermeasure implemented, the report must analyse:
 - The vulnerabilities tackled
 - The results of the application of the countermeasure
 - The trade-offs introduced
- Analyse the results obtained when all countermeasures are applied, including remaining vulnerable points detected by the simulator and some other potential fault scenarios and models that could bypass the mitigations.

Exercise and report

- Conducting the lab: Perform the specified tasks and document it as explained in the lab guide.
 - Go step-by-step carefully reading the guide.
- Reporting: Follow the style guide that can be found in the [virtual classroom](#) (some small things can be changed, e.g. using Latex fonts/formatting)
- Deadline: 18th May 2025



UNIVERSIDAD
DE MURCIA



Facultad de
Informática

