RÉPUBLIQUE FRANÇAISE

Liberté
Égalité
Fraternité

Security of Complex Software

How to Move from Characterising the Fault Effects to Exploitation?

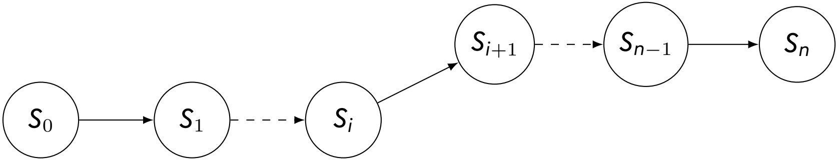
Guillaume Bouffard

Co-joint work with: Alexandre Iooss and Thomas Trouchkine

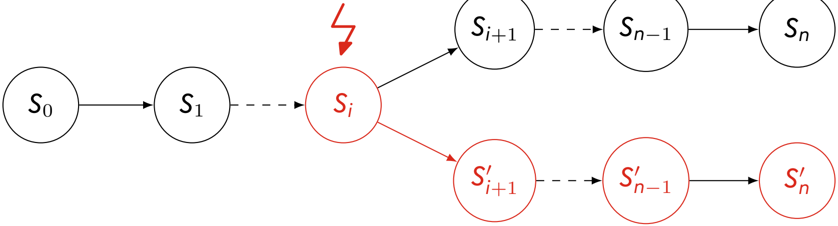
Agence nationale de la sécurité des systèmes d'information

European Cyber Week 2023 — BITFLIP 2023 — 22 november 2023

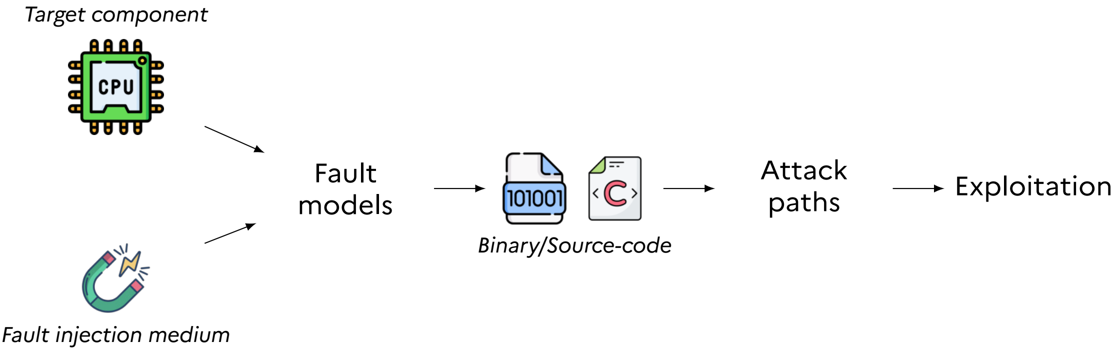
Fault Injection Attacks: Effects on Software



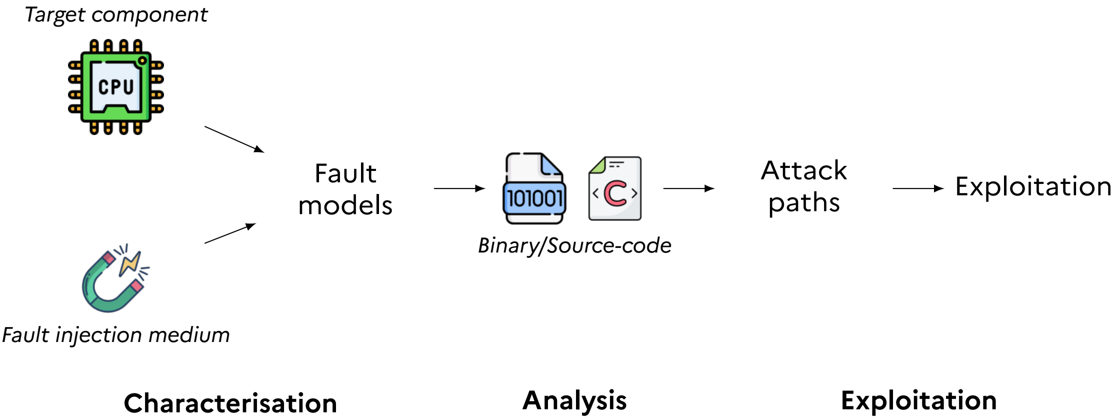
Fault Injection Attacks: Effects on Software



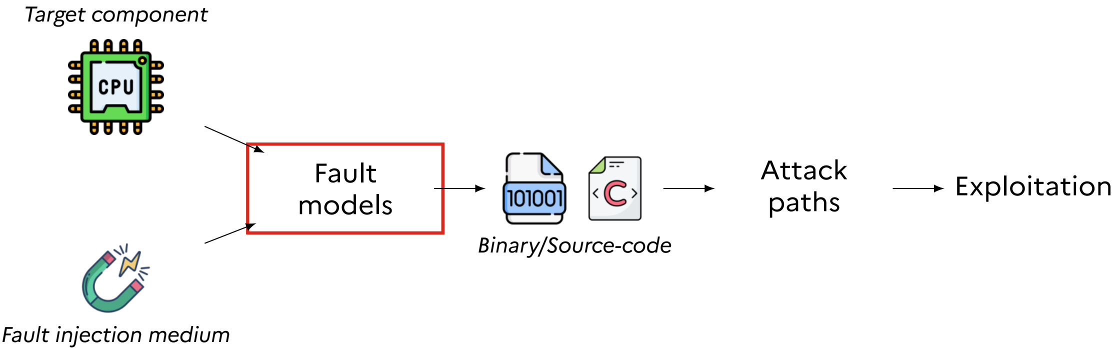
Scenario of a Fault Injection Attack



Scenario of a Fault Injection Attack



Scenario of a Fault Injection Attack



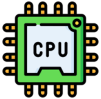
Characterisation **Analysis** **Exploitation**

Scenario of a Fault Injection Attack

Characterisation approaches:

- From software POV: T. Trouchkine's PhD Thesis [Tro21].
- From hardware logic: A. Marotta's PhD Thesis [Mar].

Target component



Fault injection medium

Fault models →

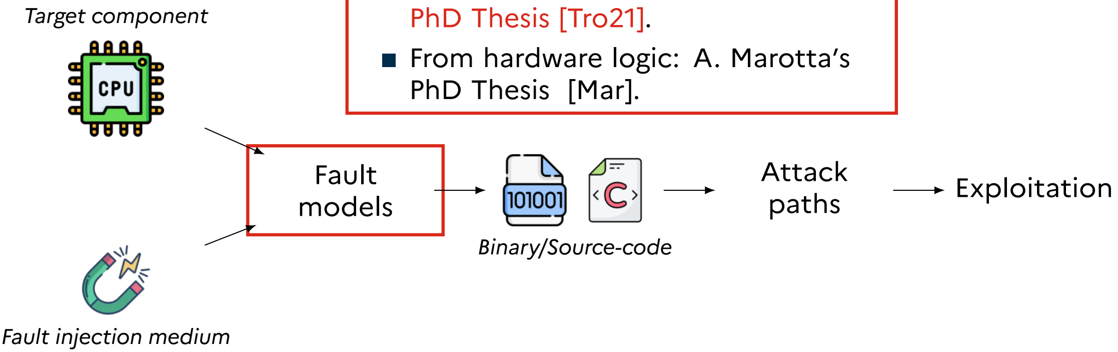
Binary/Source-code → Attack paths → Exploitation

Characterisation **Analysis** **Exploitation**

Scenario of a Fault Injection Attack

Characterisation approaches:

- From software POV: T. Trouchkine's PhD Thesis [Tro21].
- From hardware logic: A. Marotta's PhD Thesis [Mar].



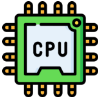
Characterisation **Analysis** **Exploitation**

Scenario of a Fault Injection Attack

Characterisation approaches:

- From software POV: T. Trouchkine's PhD Thesis [Tro21].
- From hardware logic: A. Marotta's PhD Thesis [Mar].

Target component



Fault injection medium

Fault models

Binary/Source-code

Attack paths

Exploitation

Characterisation

Analysis

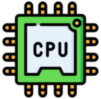
Exploitation

Scenario of a Fault Injection Attack

Characterisation approaches:

- From software POV: T. Trouchkine's PhD Thesis [Tro21].
- From hardware logic: A. Marotta's PhD Thesis [Mar].

Target component



Fault injection medium

Fault models

Binary/Source-code

Attack paths → Exploitation

Characterisation

Analysis

Exploitation

Security analysis of sudo [Tro21, ITB23].

Fault Injection Attacks

To succed a Fault Injection (FI) attack, one need to:

- Know where attacks? $\Rightarrow$ FI medium (x, y, z, θ) position on the target component;
- Know how attacks? $\Rightarrow$ FI medium parameters;
- Know when attacks? $\Rightarrow$ Time t to confuse target program.

Fault Injection Attacks (cont.)

1. Where?/How?: Characterize the sensitivity of the target component to a fault medium [TBC21, PHB+19]
2. When?: Transfer this sensitivity to a target application [GHHR23, Dur16]
3. Conducting an attack through exploitation [Wer22, TBE+21]

Fault Injection Attacks (cont.)

1. Where?/How?: Characterize the sensitivity of the target component to a fault medium [TBC21, PHB+19]
2. When?: Transfer this sensitivity to a target application [GHHR23, Dur16]
3. Conducting an attack through exploitation [Wer22, TBE+21]

Can one determine when to induce a fault in a complex software by characterizing (where/how) the effect of a fault?

Fault Injection Attacks (cont.)

1. Where?/How?: Characterize the sensitivity of the target component to a fault medium [TBC21, PHB+19]
2. When?: Transfer this sensitivity to a target application [GHHR23, Dur16]
3. Conducting an attack through exploitation [Wer22, TBE+21]

Can one determine when to induce a fault in a complex software by characterizing (where/how) the effect of a fault?

Target: The sudo application on a Raspberry Pi 4 running the Raspberry Pi OS[1] (derived from Debian).

[1] **See:** `https://www.raspberrypi.com/software/`

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

1. Characterisation

Analysis of fault effects on a complex CPU [Tro21]

During a fault, at least one microarchitecture block is disturbed.

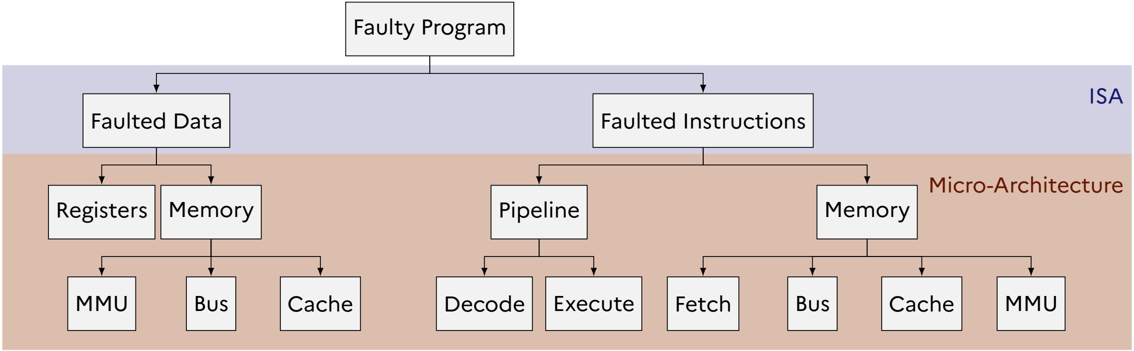
Analysis of fault effects on a complex CPU [Tro21]

During a fault, at least one microarchitecture block is disturbed.

Modus Operandi

- A test program is faulted during its execution.
- Fault different test programs to gather information on the behavior of microarchitectural blocks.

Analysis of fault effects on a complex CPU [Tro21]



Characterisation approach

Test program 1

```
orr r4, r4;
/*
 * Arbitrary number
 * of repetitions
 */
orr r4, r4;
```

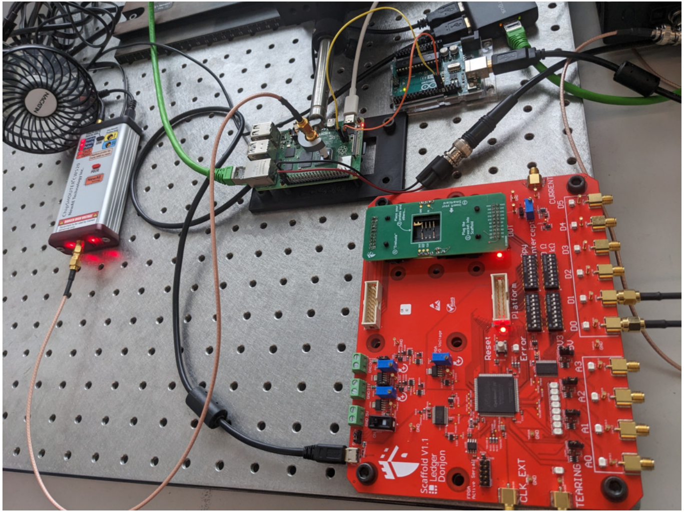
Test program 2

```
and r4, r4, #255;
/*
 * Arbitrary number
 * of repetitions
 */
and r4, r4, #255;
```

Table: Initial values of the registers.

Register	**Initial value for** `ORR R4, R4` **repetition**	**Initial value for** `AND R4, #255` **repetition**
r0	0xFFFE0001	0xFFFE0001
r1	0xFFFD0002	0xFFFD0002
r2	0xFFFB0004	0xFFFB0004
r3	0xFFF70008	0x000000FF
r4	0xFFEF0010	0xFFEF0010
r5	0xFFDF0020	0xFFDF0020
r6	0xFFBF0040	0xFFBF0040
r7	0xFF7F0080	0xFF7F0080
r8	0xFEFF0100	0xFEFF0100
r9	0xFDFF0200	0xFDFF0200

Test Bench


Scaffold V1.1
Ledger
Donjon
CLK_EXT
TEARING
Reset
Error
Platform
CURRENT
D5
D4
D3
D2
D1
D0
A3
A2
A1
A0

Sensitivity Map and Results

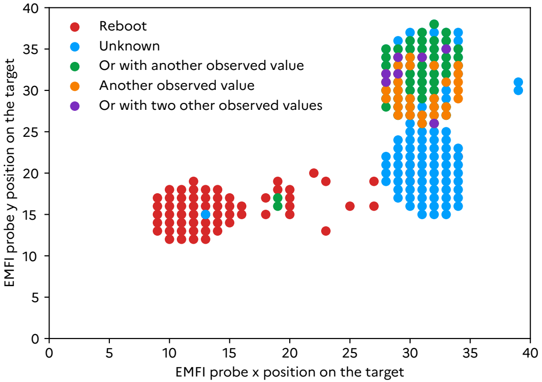


Figure: `ORR R4, R4`

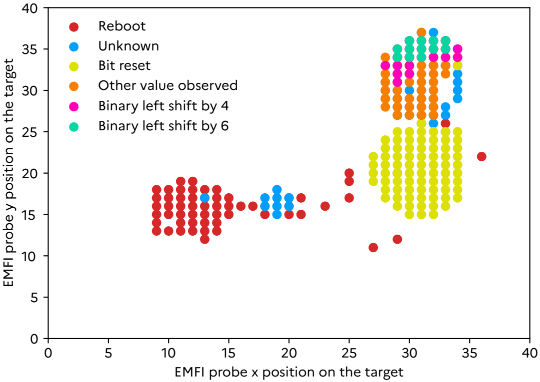


Figure: `AND R4, R4, #255`

Sensitivity Map and Results

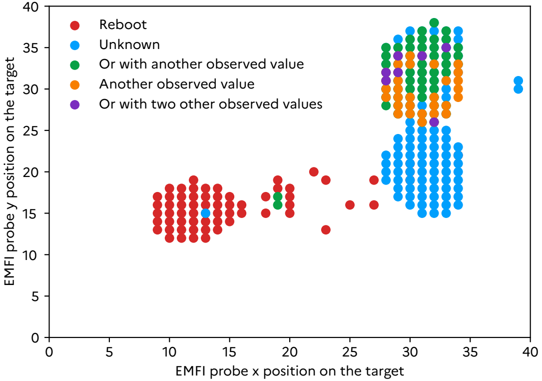


Figure: `ORR R4, R4`

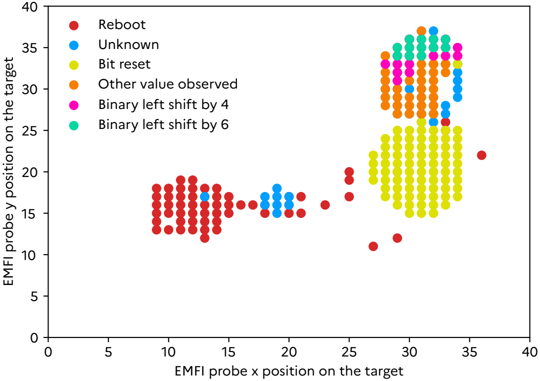


Figure: `AND R4, R4, #255`

☞ Setting the 8 least significant bits to 0 in the executed instructions.

2. Fault Model Transposition

Fault Model Transposition

Simulation Goal: Determine when to inject the fault into the execution flow to achieve a desired effect. We focus on simple fault.
Target: Binary sudo from Raspberry Pi OS[2] (based on Debian)

[2] `https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2022-09-26/`

Fault Model Transposition

Simulation Goal: Determine when to inject the fault into the execution flow to achieve a desired effect. We focus on simple fault.
Target: Binary `sudo` from Raspberry Pi OS[2] (based on Debian)

Authentication rules in `/etc/pam.d/common-auth`

```
# here are the per-package modules (the "Primary" block)
auth        [success=1 default=ignore]        pam_unix.so nullok
# here's the fallback if no module succeeds
auth        requisite                         pam_deny.so
```

`sudo` calls PAM, which uses the `pam_unix.so` module.

[2] https://downloads.raspberrypi.org/raspios_lite_armhf/images/raspios_lite_armhf-2022-09-26/

Dynamic Instrumentation of `pam_unix.so` with Rainbow

Rainbow is a fault injection simulator developed by the Ledger Donjon, based on Unicorn-Engine (QEMU).

⚠ `sudo` is **dynamically** linked with glibc and PAM.

Ledger-Donjon/rainbow

Dynamic Instrumentation of `pam_unix.so` with Rainbow

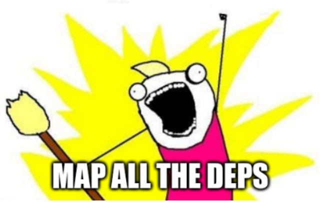
Rainbow is a fault injection simulator developed by the Ledger Donjon, based on Unicorn-Engine (QEMU).

⚠ sudo is **dynamically** linked with glibc and PAM.

We implemented a loader based on CLE in Rainbow.

Dynamic dependencies of `pam_unix.so`

`libpam.so.0, libcrypt.so.1, libselinux.so.1, libnsl.so.2, libtirpc.so.3, libc.so.6, ld-linux-armhf.so.3, libaudit.so.1, libdl.so.2, libpcre2-8.so.0, libgssapi_krb5.so.2, libpthread.so.0, libcap-ng.so.0, libkrb5.so.3, libk5crypto.so.3, libcom_err.so.2, libkrb5support.so.0, libkeyutils.so.1, libresolv.so.2`



Ledger-Donjon/rainbow

Fault Model Simulation in Rainbow

```python
def fault_model(emu):
    # Get PC value
    pc = emu["pc"]
    # Get next instruction
    instr = emu[pc]

    # Patch and run modified instruction
    i = int.from_bytes(instr, "little") & 0xFFFF_FF00
    instr_patched = i.to_bytes(4, "little")
    emu[pc] = instr_patched
    emu.start(pc, 0, count=1)

    # Restore correct instruction
    emu[pc] = bytes(instr)
```

Exhaustive Testing in Simple Fault

```python
emu = rainbow_arm()
emu.load("arm-libs/pam_unix.so")
emu[0xE0000000] = f"toto\x00".encode()
emu[0xF0000000] = f"$6$hH.15uU5laaxuXHY$anemvMyc.gFyc[...]nSGEO.\x00".encode()
emu["r0"] = 0  # pamh (used for pam_syslog calls)
emu["r1"] = 0xE0000000  # const char *p
emu["r2"] = 0xF0000000  # char *hash
emu["r3"] = 1  # unsigned int nullok
pc_stopped = emu.start_and_fault(fault_model, i, 0x00405b40, 0, count=1000)
print(emu["r0"])  # if 0, then auth is successful
```

Variant of the Ledger blog post [IS22]. We hook `malloc`, `calloc`, and `free`.
22 minutes or 3.2 seconds with SHA2-256 hooked (`crypt_r`).

Exhaustive Testing in Simple Fault: Results

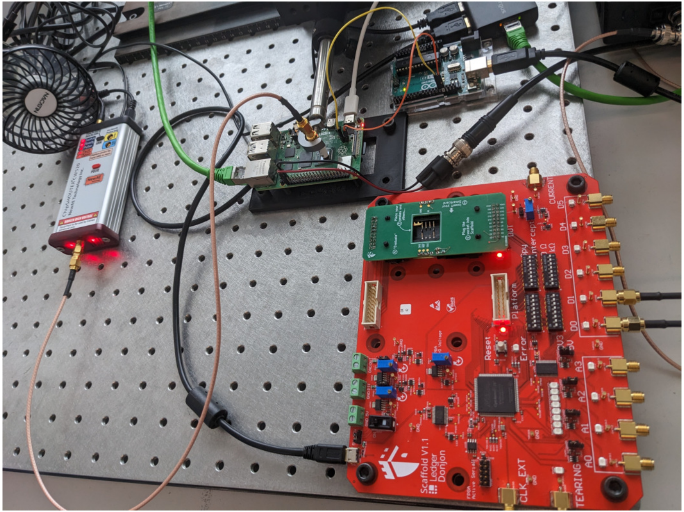
Fault Found

```
; i=1201 (in pam_unix.so)
MOVNE   R4, #7
; become
MOVNE   R4, #0
```

```
          ; [...]
@121:  movne      r4,#0x7     ; r4 <- 0 with fault
@122:  cmp        r3,#0x0
@123:  cpyne      r3,r7
@124:  movne      r1,#0x0
@125:  beq        LAB_00405cb4
    LAB_00405ca4:
@126:  strb       r1,[r3],#0x1
@127:  ldrb       r2,[r3,#0x0]
@128:  cmp        r2,#0x0
@129:  bne        LAB_00405ca4
    LAB_00405cb4:
@130:  cpy        r0,r7
@131:  bl         free
@132:  cpy        r0,r4       ; returns the value of r4
@133:  ldmia      sp!,{r4,r5,r6,r7,r8,r9,r10,pc}
```

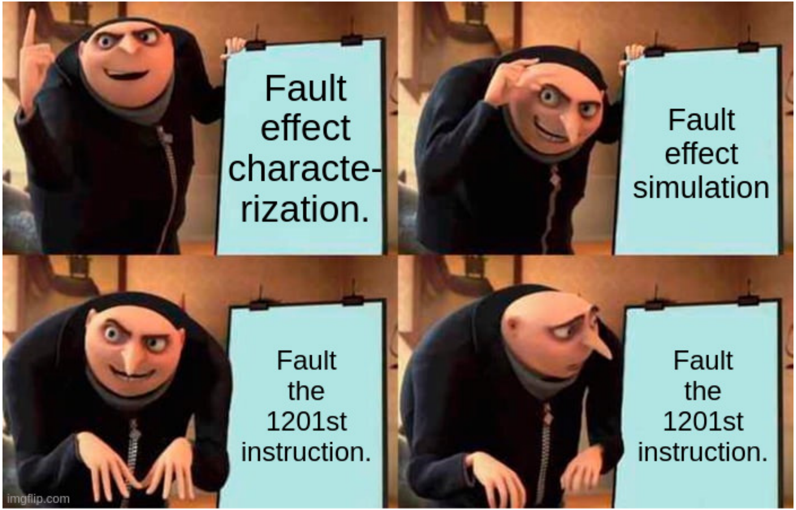
3. Exploitation

Test Bench



Scaffold V1.1
Ledger
Donjon
Reset
Error
Platform
CLK_EXT
TEARING
CURRENT
D5
D4
D3
D2
D1
D0
A3
A2
A1
A0

Transition to Reality



Transition to Reality: Jitter Measurement

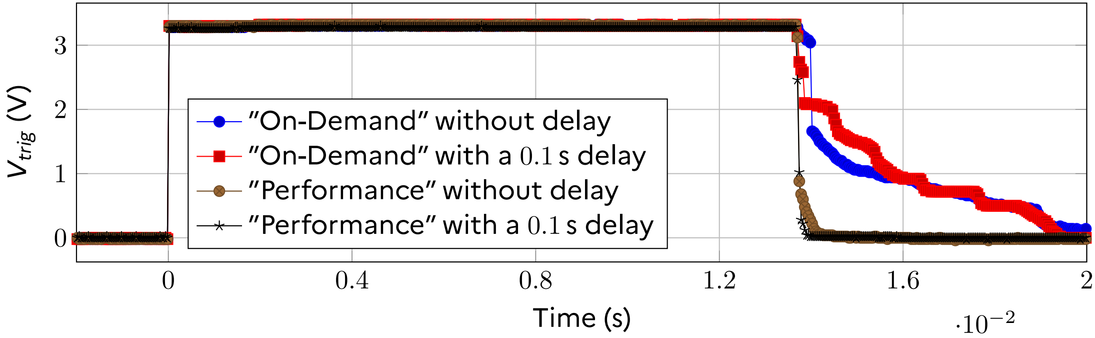
Modus Operandi

- Creation of a wrapper to launch sudo via Netcat and trigger a GPIO.
- sudo forced on CPU core 3.

Transition to Reality: Jitter Measurement

Modus Operandi

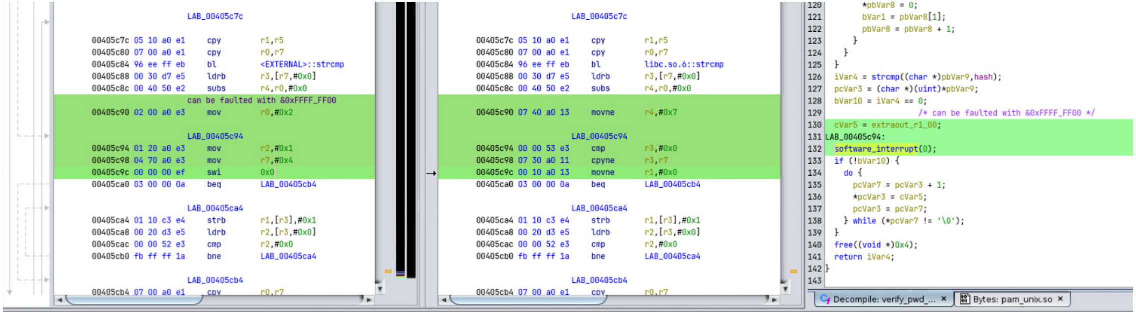
- Creation of a wrapper to launch sudo via Netcat and trigger a GPIO.
- sudo forced on CPU core 3.



RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Trick to Find the Fault Instant

Use of an *open-sample*



Trick to Find the Fault Instant

Problem: The delay introduced by the `syscall` and parent process.

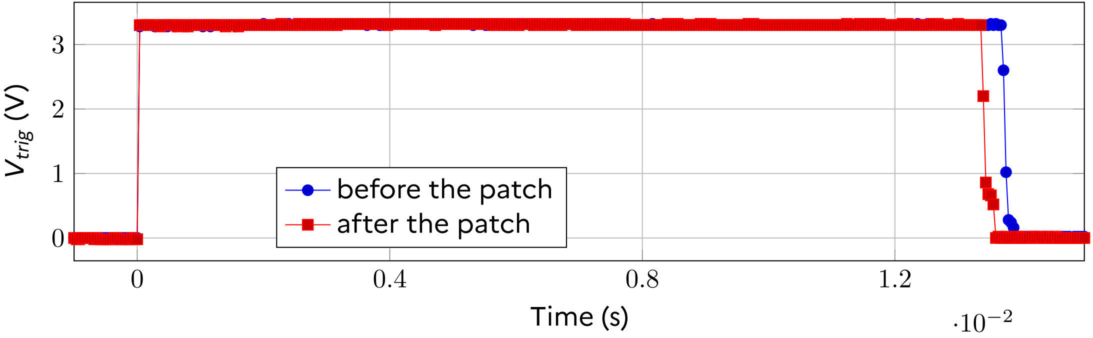


Figure: Average of 1024 calls to sudo with the incorrect password, CPU performance mode, delay of 0.1 s.

Limitations

- The expected fault model was not consistently found at the same FI medium position.

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial `sudo` bypass:
  - `sudo --validate`

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial `sudo` bypass:
  - `sudo --validate` $\Rightarrow$ success

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial sudo bypass:
  - `sudo --validate` ⇒ success
  - `sudo whoami`

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial `sudo` bypass:
  - `sudo --validate` ⇒ success
  - `sudo whoami` ⇒ not success

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial sudo bypass:
  - `sudo --validate` ⇒ success
  - `sudo whoami` ⇒ not success
  - The reasons for this difference between simulation and reality are currently unknown.

Limitations

- The expected fault model was not consistently found at the same FI medium position.
- Partial sudo bypass:
  - `sudo --validate` ⇒ success
  - `sudo whoami` ⇒ not success
  - The reasons for this difference between simulation and reality are currently unknown.
- Require an open-sample to determine real-world injection timing:
  - overcoming with a cycle-accurate emulation ⇒ it's rarely available for complex systems.
  - Using side-channel simulators to identify the injection point on a simulated trace, but precise modeling of the target is currently challenging.

Conclusion

In this work:

- Transposition of the fault model to a complex program (sudo)

⚠ Transition from analyzing a binary to a bench attack

- Use of open-source tools:
  - Ledger-Donjon/rainbow (proposed patches submitted).
  - Ledger-Donjon/scaffold
  - newaetech/ChipSHOUTER

Conclusion

In this work:

- Transposition of the fault model to a complex program (sudo)

⚠ Transition from analyzing a binary to a bench attack

- Use of open-source tools:
  - Ledger-Donjon/rainbow (proposed patches submitted).
  - Ledger-Donjon/scaffold
  - newaetech/ChipSHOUTER

What's next?

- Can we do without an *open-sample*?
- How to protect against it?
- Transition to RISC-V.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Questions?

Security of Complex Software

How to Move from Characterising the Fault Effects to Exploitation?

Guillaume Bouffard
Laboratory of Hardware and Software Architectures
ANSSI
`guillaume.bouffard@ssi.gouv.fr`

Credit: icons downloaded from Flaticon

Bibliographie

[Dur16] Louis Dureuil, *Analyse de code et processus d'évaluation des composants sécurisés contre l'injection de faute.*, Ph.D. thesis, Grenoble Alpes University, France, 2016.

[GHHR23] Antoine Gicquel, Damien Hardy, Karine Heydemann, and Erven Rohou, *SAMVA: static analysis for multi-fault attack paths determination*, Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, April 3-4, 2023, Proceedings (Elif Bilge Kavun and Michael Pehl, eds.), Lecture Notes in Computer Science, vol. 13979, Springer, 2023, pp. 3–22.

[IS22] Alexandre Iooss and Victor Servant, *Integrating fault injection in development workflows*, Aug 2022.

Bibliographie (cont.)

[ITB23] Alexandre Iooss, Thomas Trouchkine, and Guillaume Bouffard, *Pew Pew, I'm root! - De la caractérisation à l'exploitation: un voyage plein d'embûches*, September 2023.

[Mar] Amélie Marotta, *Electromagnetic injection fault models and countermeasures for RISC-V sw FPGA processors*, Ph.D. thesis, Université Rennes.

[PHB+19] Julien Proy, Karine Heydemann, Alexandre Berzati, Fabien Majéric, and Albert Cohen, *A first isa-level characterization of EM pulse effects on superscalar microarchitectures: A secure software perspective*, Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019, ACM, 2019, pp. 7:1–7:10.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Bibliographie (cont.)

[TBC21] Thomas Trouchkine, Guillaume Bouffard, and Jessy Clédière, *EM fault model characterization on socs: From different architectures to the same fault model*, 18th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2021, Milan, Italy, September 17, 2021, IEEE, 2021, pp. 31–38.

[TBE+21] Thomas Trouchkine, Sébanjila Kevin Bukasa, Mathieu Escouteloup, Ronan Lashermes, and Guillaume Bouffard, *Electromagnetic fault injection against a complex cpu, toward new micro-architectural fault models*, J. Cryptogr. Eng. **11** (2021), no. 4, 353–367.

[Tro21] Thomas Trouchkine, *SoC physical security evaluation*, Ph.D. thesis, Université Grenoble Alpes, March 2021.

[Wer22] Vincent Werner, *Optimizing identification and exploitation of fault injection vulnerabilities on microcontrollers.*, Ph.D. thesis, Grenoble Alpes University, France, 2022.