# Electromagnetic Fault Injection on SoCs

**Clément Gaine**[1], Driss Aboulkassimi[1], Simon Pontié[1], Jean-Pierre Nikolovski[1], Jean-Max Dutertre[2]

[1]*Univ. Grenoble Alpes, CEA, LETI, MINATEC Campus, F-38054 Grenoble, France*
[2]*Mines Saint-Etienne, CEA-Tech, Centre CMP, Departement SAS, F-13541 Gardanne France*

JAIF'2021, September 23, 2021, Paris

# State of the art

## Complex SoC-type target vulnerability to physical attacks

-Mobile phones contain a large amount of personal data

-Observation attacks - Side-channel [Aboulkassimi et al., 2011, Leignac et al., ]

-Perturbation attacks - Fault injection:

- Laser [Vasselle et al., 2017]
- Voltage [Timmers and Mune, 2017]
- Clock-based [Tang et al., 2017]
- EM [Majéric et al., 2016, Proy et al., 2019, Trouchkine et al., 2019]

# EMFI on complex target

## Complex SoC-type versus microcontroller

- Complex hardware architecture (cache memory, CPUs, ...)
- Complex software layer (OS, ...)
- High operating frequencies (>1GHz)
- Large silicon area with a small technology node
- More security features (TrustZone, TEE, ...)

New topic, mostly on academic targets.

# EMFI on complex target

## Complex SoC-type versus microcontroller

- Complex hardware architecture (cache memory, CPUs, ...)
- Complex software layer (OS, ...)
- High operating frequencies (>1GHz)
- Large silicon area with a small technology node
- More security features (TrustZone, TEE, ...)

New topic, mostly on academic targets.

## Use-case

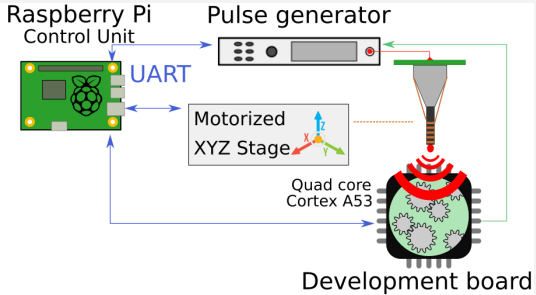
EMFI on SoC for forensic[Gaine et al., 2020] - ExFiles Project[a]

[a] https://exfiles.eu/

# Overview

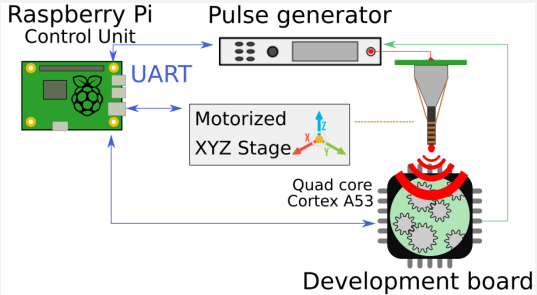
# EMFI platform and targeted SoC

## EMFI platform



# EMFI platform and targeted SoC

## EMFI platform




## Targeted Soc

64-bit 4-core SoC
Operating frequency up to 1.2GHz
Linux OS

# Physical vulnerability analysis of SoC under test

## How to inject faults?

-Characterization step: running a chosen test
-When to inject?
-Where to inject?

# When to fire?

## Challenge 1: time synchronization

-High operating speed requires a higher resolution time and accuracy
-Hardware and software complexity
-Many uncontrollable desynchronization sources (50ns jitter)

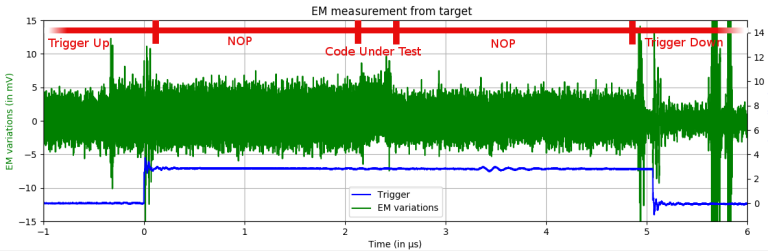
# When to fire?

## Challenge 1: time synchronization

-High operating speed requires a higher resolution time and accuracy
-Hardware and software complexity
-Many uncontrollable desynchronization sources (50ns jitter)

## Searching efficient delay for fault injection : Based on Side-Channel Analysis

Simple ElectroMagnetic Analysis used to identify the timing



# When to fire?

## Code Under Test

A 320 instructions code de relax synchronization constraints

```
//Initialization x28 = 368 = 0x170
mov x28, #0170
//Following sequences repeated 32 times
sub x19, x28, #0x1
sub x20, x19, #0x1
sub x21, x20, #0x1
...
sub x28, x27, #0x1
```

These are 320 instructions

-> i.e. 270 ns

# When to fire?

## Code Under Test

A 320 instructions code de relax synchronization constraints

```
//Initialization x28 = 368 = 0x170
mov x28, #0170
//Following sequences repeated 32 times
sub x19, x28, #0x1
sub x20, x19, #0x1
sub x21, x20, #0x1
...
sub x28, x27, #0x1
```

The EM perturbation effect

is seen at readback

# Where to fire?

**Challenge 2: Spatial resolution**

-Large area to explore
-Small technological node ($28nm$)
-Active CPU executing the code is unknow

# Where to fire?

## Challenge 2: Spatial resolution

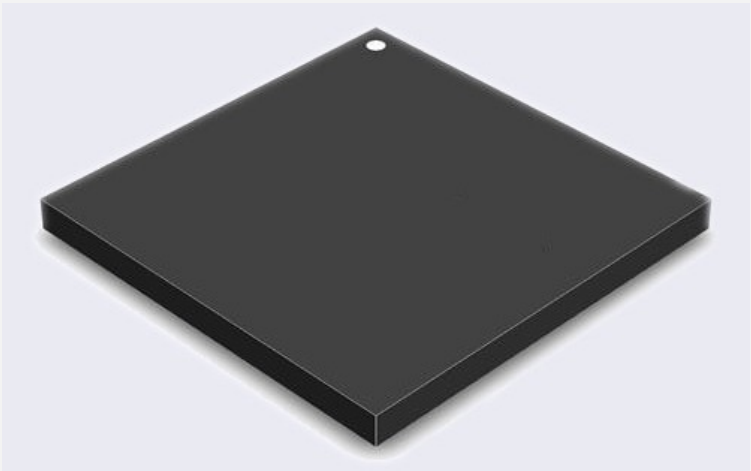
-Large area to explore
-Small technological node ($28nm$)
-Active CPU executing the code is unknow

## Searching efficient probe location (X,Y) for fault injection

Force the program to run on one CPU only
Scan with a $750\mu m$ probe diameter
Pulse voltage at maximum, then reduce the voltage when a sensitive area is identified

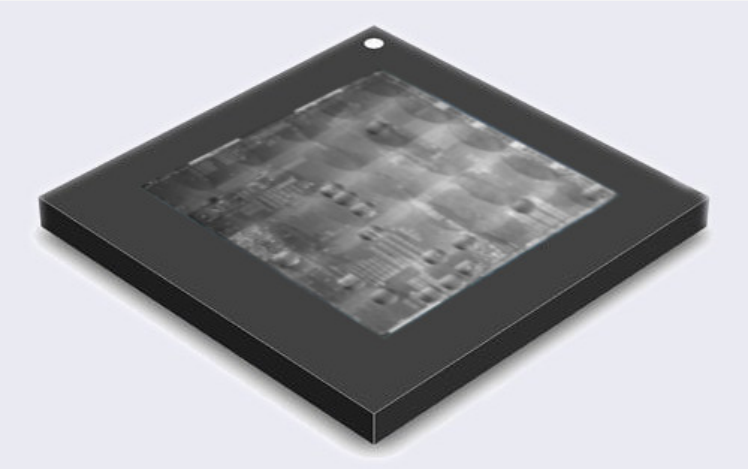
# Where to fire ?

## Localization of EMFI-sensitive areas



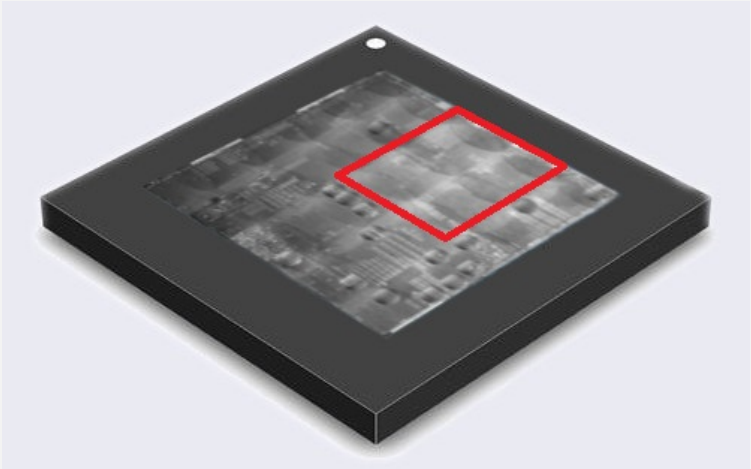
# Where to fire ?

## Localization of EMFI-sensitive areas



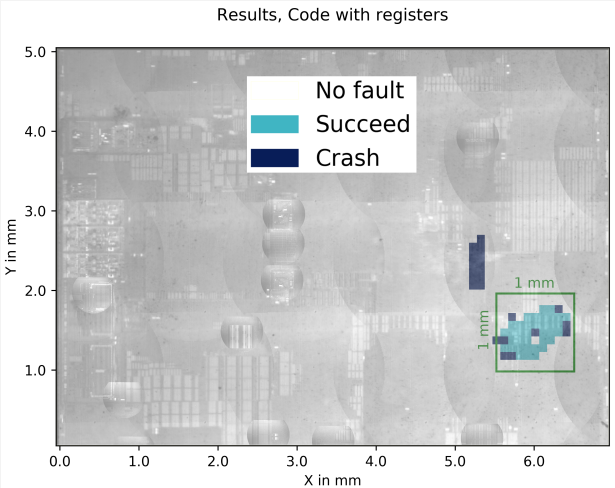
# Where to fire ?

## Localization of EMFI-sensitive areas



# Where to fire ?

## Localization of EMFI-sensitive areas



# Results

## Result Analysis

| Occurrences | Result (x19, ..., x28) | Occ. rate | Timing (in ns) |
|---|---|---|---|
| 27287 | 39,38,37,36,35,34,33,32,31,30 | 71.0% | 1600 - 1900 |
| 5314 | Communication lost | 13.8% | 1600 - 1900 |
| 4899 | 43,42,41,40,3F,3E,3D,3C,3B,3A | 12.7% | 1650 - 1890 |
| 48 | 39,38,37,36,35,3E,3D,3C,3B,3A | 0.1% | 1900 |
| 28 | 39,42,41,40,3F,3E,3D,3C,3B,3A | 0.1% | 1900 |
| ... | ... | ... | ... |

## Fault model identification

Instruction skip

# Vulnerability exploitation: privilege escalation

## Starting point

We know how to inject fault
We identified a fault model

# Vulnerability exploitation: privilege escalation

## Starting point

We know how to inject fault
We identified a fault model

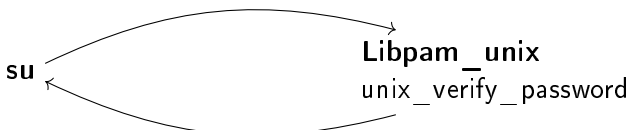
## How to elevate privileges?

Hypothesis: User access without root password
su command of Linux -> From unprivileged user to root

# Analyze the su code to identify an attack path

**su**

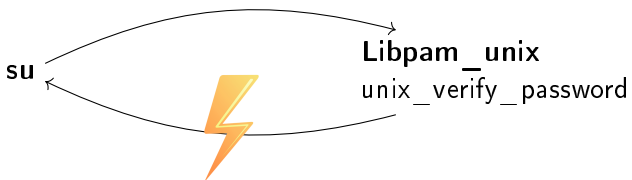
Flag setuid => starts with admnistrator rights.
- Authentication succeed -> root console. Otherwise -> user console.



# Analyze the su code to identify an attack path

## su

Flag setuid => starts with admnistrator rights.
- Authentication succeed -> root console. Otherwise -> user console.



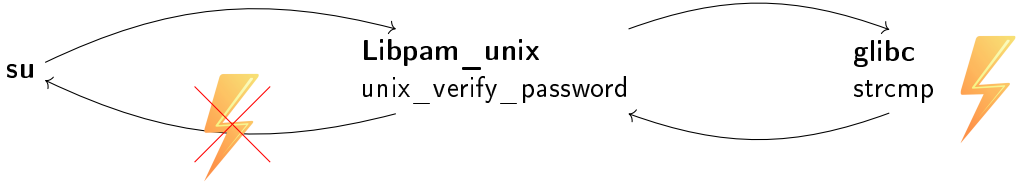

## Attack path 1 - Change libpam return to su

libpam is protected against Brute Force and Side-Channel Analysis (random time)
We aim for a nanosecond in a 1.5s interval

# Analyze the su code to identify an attack path

## su

Flag setuid => starts with admnistrator rights.
- Authentication succeed -> root console. Otherwise -> user console.




## Attack path 1 - Change libpam return to su

libpam is protected against Brute Force and Side-Channel Analysis (random time)
We aim for a nanosecond in a 1.5s interval

## Attack path 2 - strcmp control flow

Checks the validity of the password provided by _unix_verify_password

# Analyze the strcmp code to identify an attack path

## strcmp function

Compare the hashes of the entered and stored password bytes by bytes

## Hashes compared by strcmp

- $6$wWxFc|tJdeOl05|KNO$IAAh|w8Th... -> Hash of "root" = root password
- $6$wWxFc|tJdeOl05|KNO$Uung|4U7s... -> Hash of "fail" = test password

word 1 | word 2 | word 3 | word ...

# Results and exploitation

## Comparison of two hashes by strcmp

```
L(loop_misaligned):
  ...
  ldr data1, [src1], #8
  ldr data2, [src2], #8
  sub tmp1, data1, zeroones
  orr tmp2, data1, #REP8_7f
  eor diff, data1, data2 /*Non-zero if differences found.*/
  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
  orr syndrome, diff, has_nul
  cbz syndrome, L(loop_misaligned)
  b L(end)
```

# Results and exploitation

## Comparison of two hashes by strcmp

```
L(loop_misaligned):
 ↑  ...                                                    1st round
 |  ldr data1, [src1], #8 //$6$wWxFc
 |  ldr data2, [src2], #8 //$6$wWxFc
 |  sub tmp1, data1, zeroones
 |  orr tmp2, data1, #REP8_7f
 |  eor diff, data1, data2 /*Non-zero if differences found.*/
 |  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
 |  orr syndrome, diff, has_nul
 |  cbz syndrome, L(loop_misaligned) //continue the comparison
    b L(end)
```

# Results and exploitation

## Comparison of two hashes by strcmp

```
L(loop_misaligned):
 ↑  ...                                                    2nd round
 |  ldr data1, [src1], #8 //tJdeOIO5
 |  ldr data2, [src2], #8 //tJdeOIO5
 |  sub tmp1, data1, zeroones
 |  orr tmp2, data1, #REP8_7f
 |  eor diff, data1, data2 /*Non-zero if differences found.*/
 |  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
 |  orr syndrome, diff, has_nul
 |  cbz syndrome, L(loop_misaligned) //continue the comparison
    b L(end)
```

# Results and exploitation

## Comparison of two hashes by strcmp

```
L(loop_misaligned):
 ↑  ...                                                    3rd round
 |  ldr data1, [src1], #8 //KNO$IAAh
 |  ldr data2, [src2], #8 //KNO$Uung
 |  sub tmp1, data1, zeroones
 |  orr tmp2, data1, #REP8_7f
 |  eor diff, data1, data2 /*Non-zero if differences found.*/
 |  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
 |  orr syndrome, diff, has_nul
 |  cbz syndrome, L(loop_misaligned) //stop the comparison
    b L(end)
```

# Results and exploitation

## Comparison of two hashes by strcmp

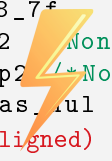
```
L(loop_misaligned):
  ...
  ldr data1, [src1], #8
  ldr data2, [src2], #8
  sub tmp1, data1, zeroones
  orr tmp2, data1, #REP8_7f
  eor diff, data1, data2 /*Non-zero if differences found.*/
  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
  orr syndrome, diff, has_nul
  cbz syndrome, L(loop_misaligned)
  b L(end)
```

EMFI during 1st or 2nd cbz instruction

# Results and exploitation

## Comparison of two hashes by strcmp

```
L(loop_misaligned):
  ...
  ldr data1, [src1], #8
  ldr data2, [src2], #8
  sub tmp1, data1, zeroones
  orr tmp2, data1, #REP8_7f
  eor diff, data1, data2 /*Non-zero if differences found.*/
  bic has_nul, tmp1, tmp2 /*Non-zero if NUL terminator.*/
  orr syndrome, diff, has_nul
  cbz syndrome, L(loop_misaligned)
  b L(end)
```



EMFI during 1st or 2nd cbz instruction

## Results

21 success for 6,000 tests -> 1 success every 15 minutes

# Towards a blackbox

## Blackbox exploitation issues

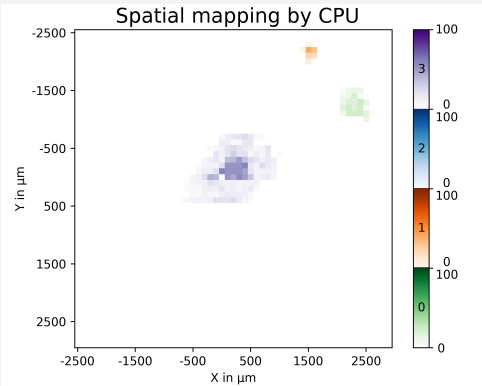
- Choice of a CPU
- Choice of a frequency
- Trigger for synchronization

## A new code under test

Alloing to maximize the faults observable number
15% of fault to 60%

# Choice of CPU

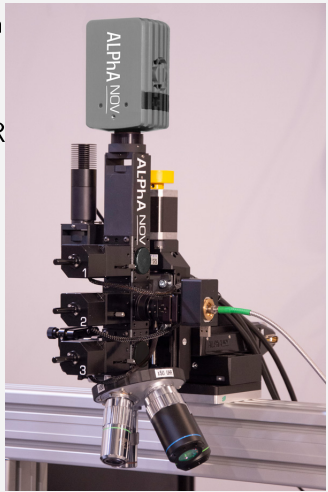
## Different location for different CPU



# Choice of CPU

## Photo-emission

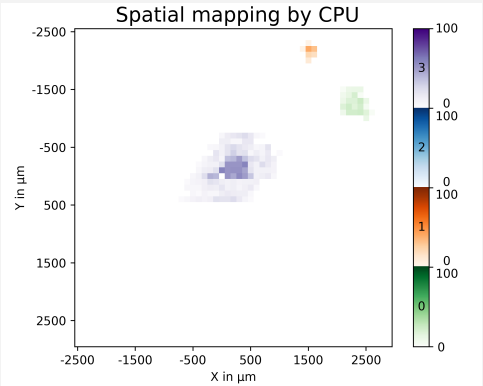
- Operation of an IC generates infrared photons via the rear side
- Loop code on one CPU
- Capture and analysis of these emissions via an IR camera

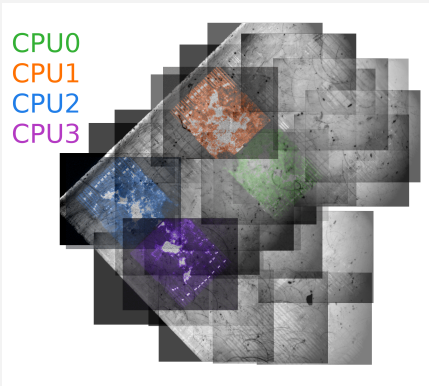



Photoemission optical bench from Alphanov

# Choice of CPU

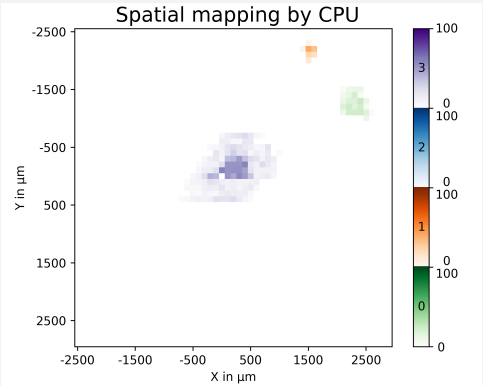
## Photo-emission

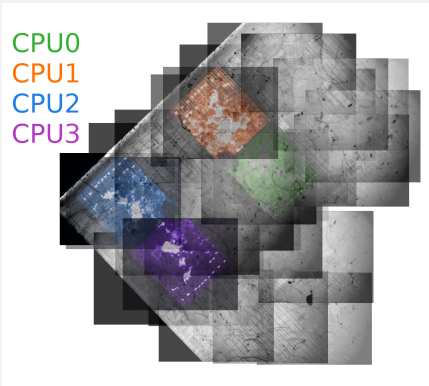


# Choice of CPU

## Photo-emission

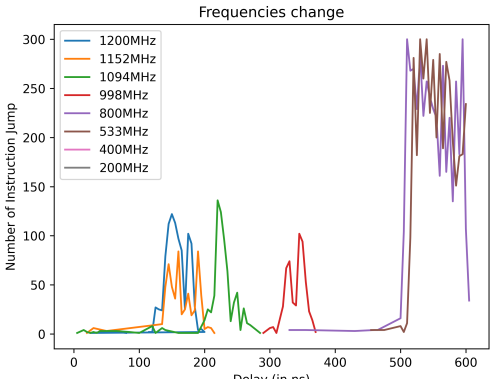



3/4 CPU are faultable, with different success rates

# Choice of frequency

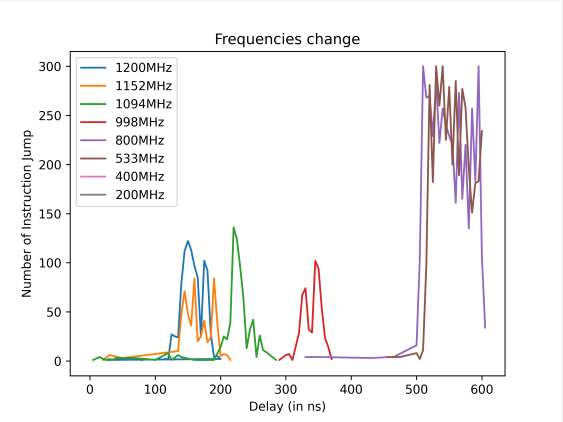
## Different frequencies




- 200 and 400 MHz = No fault
- >533MHz = Fault

# Choice of frequency

## Different frequencies




- 200 and 400 MHz = No fault
- >533MHz = Fault

Possible to fault at different frequencies, by adapting the EM pulse delay.

# Synchronization

## Currently

Development board with GPIO trigger

## Perspectives

Use of a fake usb keyboard to enter a password
-> Jitter >5 ms = several weeks of campaign

Tool to improve the synchronization
-> EM fields emitted by the processor ?

# Conclusion

-SoCs are sensitive to EMFI

# Conclusion

-SoCs are sensitive to EMFI
-Method for successful EMFI on SoC

# Conclusion

-SoCs are sensitive to EMFI
-Method for successful EMFI on SoC
-Exploitation case in bypassing the root privilege protection

# Questions ?

# References I


Aboulkassimi, D., Agoyan, M., Freund, L., Fournier, J., Robisson, B., and Tria, A. (2011).
ElectroMagnetic analysis (EMA) of software AES on Java mobile phones.
In WIFS 2011.

Gaine, C., Aboulkassimi, D., Pontié, S., Nikolovski, J.-p., and Dutertre, J.-m. (2020).
Electromagnetic Fault Injection as a New Forensic Approach for SoCs.

Leignac, P., Potin, O., Rigaux, J.-B., Dutertre, J.-M., and Pontie, S.
Comparison of side-channel leakage on rich and trusted execution environments.

Majéric, F., Bourbao, E., and Bossuet, L. (2016).
Electromagnetic security tests for SoC.
ICECS 2016.

Proy, J., Heydemann, K., Berzati, A., Majéric, F., and Cohen, A. (2019).
Studying EM Pulse Effects on Superscalar Microarchitectures at ISA Level.
ACM International Conference Proceeding Series.

Tang, A., Sethumadhavan, S., and Stolfo, S. (2017).
CLKSCREW: Exposing the perils of security-oblivious energy management.
USENIX.

Timmers, N. and Mune, C. (2017).
Escalating Privileges in Linux Using Voltage Fault Injection.
Fault Diagnosis and Tolerance in Cryptography, FDTC.

# References II

Trouchkine, T., Bukasa, S. K., Escouteloup, M., Lashermes, R., and Bouffard, G. (2019).
Electromagnetic fault injection against a System-on-Chip, toward new micro-architectural fault models.

Vasselle, A., Thiebeauld, H., Maouhoub, Q., Morisset, A., and Ermeneux, S. (2017).
Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot.
Fault Diagnosis and Tolerance in Cryptography, FDTC.