Fault Injection Characterization on modern CPUs

From the ISA to the Micro-Architecture

Thomas TROUCHKINE[1], Guillaume BOUFFARD[1,2], Jessy CLÉDIÈRE[3]

[1] National Cybersecurity Agency of France (ANSSI)

[2] Information Security Group, École Normale Supérieure

[3] CEA, LETI, MINATEC

December 12, 2019

Digital systems usage

- Secure elements

✓ Certified

- Mobile devices

✗ Not fully certified

Both are powered by System On Chips (SoCs)

System On Chip differences

Secure element (Evaluated)

simple CPU

Mobile device (Non-evaluated)

complex CPU

System On Chip differences

Secure element (Evaluated)

- simple CPU
- few modules

Mobile device (Non-evaluated)

- complex CPU
- multiple modules

System On Chip differences

Secure element (Evaluated)

- simple CPU
- few modules
- internal memory only

Mobile device (Non-evaluated)

- complex CPU
- multiple modules
- internal and external memory

System On Chip differences

Secure element (Evaluated)

- simple CPU
- few modules
- internal memory only
- few communication interfaces

⟹ "Small" attack surface

Mobile device (Non-evaluated)

- complex CPU
- multiple modules
- internal and external memory
- multiple interfaces

⟹ "Large" attack surface

System On Chip differences

Secure element (Evaluated)

- simple CPU
- few modules
- internal memory only
- few communication interfaces

⟹ "Small" attack surface

Mobile device (Non-evaluated)

- complex CPU
- multiple modules
- internal and external memory
- multiple interfaces

⟹ "Large" attack surface

Focus on the CPU behavior against Fault Injections

Fault characterization overview

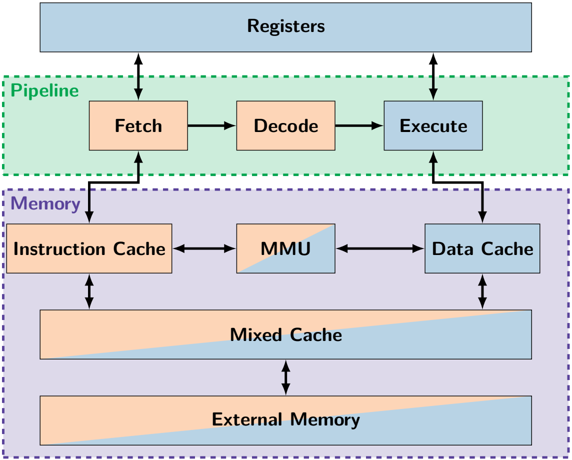
On simple CPUs

- **Pipeline** (Balasch, Gierlichs, and Verbauwhede 2011)
- **Buses** (Moro et al. 2013)
- **Cache** (Rivière et al. 2015)
- **Memory** (Kumar et al. 2018)

On complex CPUs

- **Registers/Instructions** (Proy et al. 2019)

Complex CPU model (functional view)



Micro-architectural blocks manipulating: Data Instructions

Buses: ⟷

Analysis framework

Adversary model

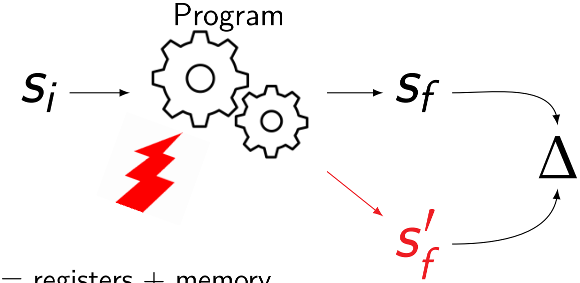
- Developer access
- No hardware debug tools (JTAG,*etc*)

Analysis framework

Adversary model

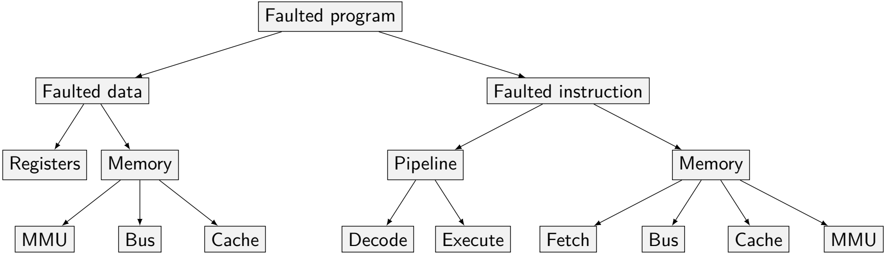
- Developer access
- No hardware debug tools (JTAG,*etc*)

How to characterize the fault model ?



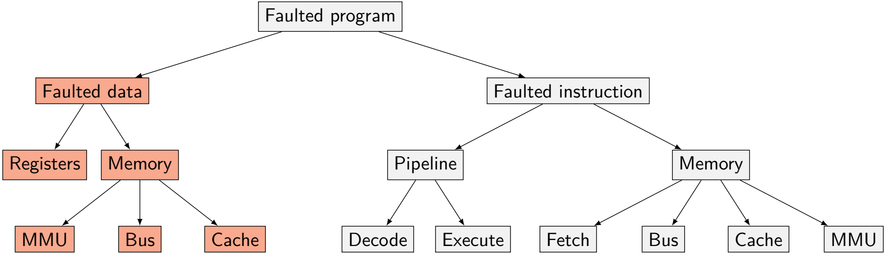
S (state) = registers + memory

Analysis paths



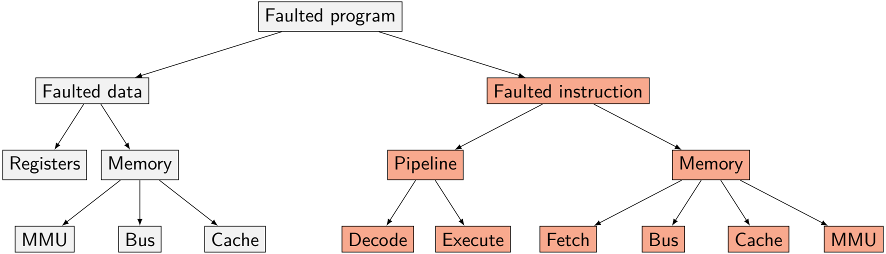
Applied on BCM2837 and Intel Core I3
against electromagnetic perturbations.

Analysis paths



Applied on BCM2837 and Intel Core I3 against electromagnetic perturbations.

Analysis paths

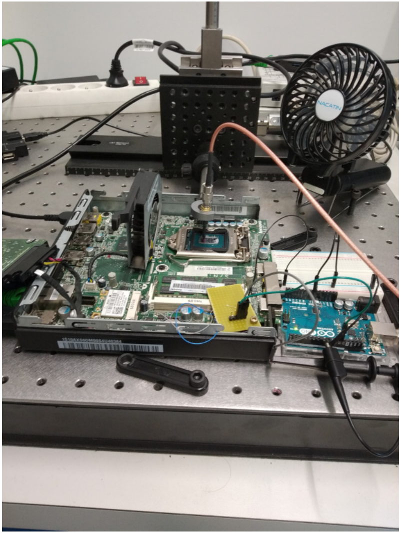


Applied on BCM2837 and Intel Core I3 against electromagnetic perturbations.

Step by step analysis on Intel Core i3

The bench

- High voltage (800V/16A) pulse generator
- Home-made electromagnetic probe (copper wire and ferrite)
- Arduino-based defibrillator
- GPIO based external trigger
- Fan based cooling system
- Test programs run over Debian 9



Step by step analysis on Intel Core i3

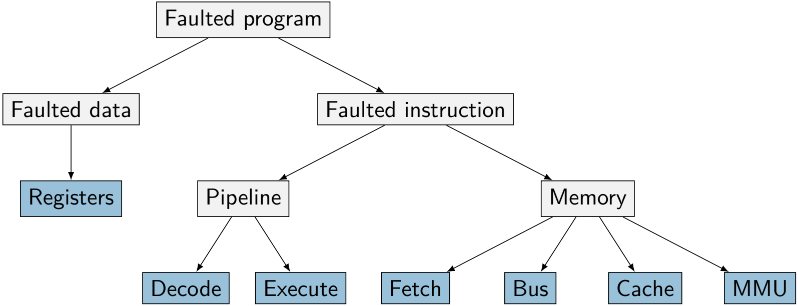
Tested program

```
trigger_up();
mov rbx, rbx;
... # several times
mov rbx, rbx;
trigger_down();
```

Criteria

- No state modification
- No data memory access
- Wide time injection window

Analysis paths



Step by step analysis on Intel Core i3

Initial values

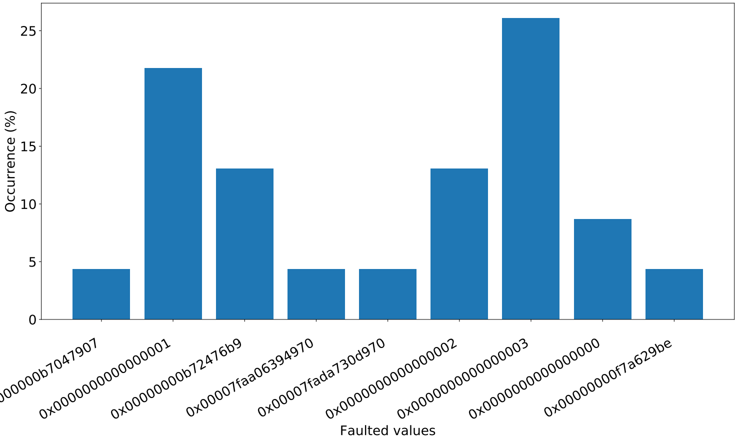
- No simple arithmetic relations between them
- Set with all bit flipped also used

Register	Initial value	Register	Initial value
rax	0x8000000000000001	r9	0x0100000000000080
rbx	0x4000000000000002	r10	0x0080000000000100
rcx	0x2000000000000004	r11	0x0040000000000200
rdx	0x1000000000000008	r12	0x0020000000000400
rsi	0x0800000000000010	r13	0x0010000000000800
rdi	0x0400000000000020	r14	0x0008000000001000
r8	0x0200000000000040	r15	0x0004000000002000

Step by step analysis on Intel Core i3

Distribution of the faulted values (`mov rbx, rbx`)

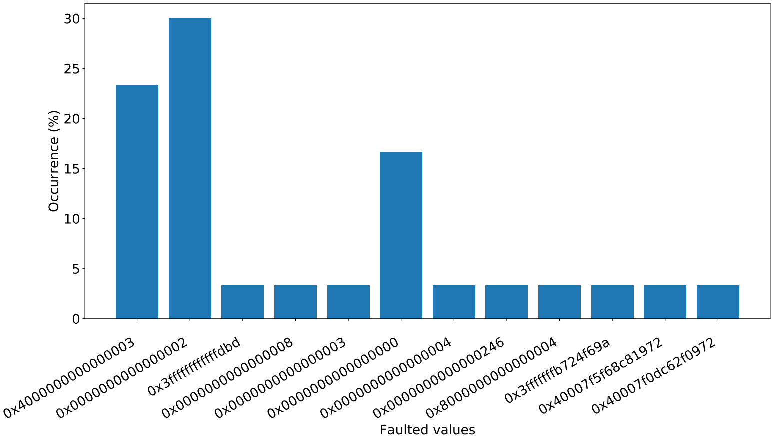
The only faulted register is always `rbx`.



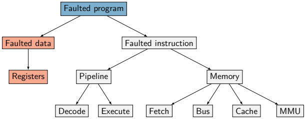
Step by step analysis on Intel Core i3

Distribution of the faulted values (`orr rbx, rbx`)

The only faulted register is always `rbx`.



Step by step analysis on Intel Core i3



Fault on registers ?

- Hypothesis with a first experiment → bits reset
- Confirmation with a second experiment (new initial values)

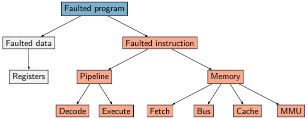
13 to 30% of the cases:

$$\texttt{0x2} = \texttt{0x4000000000000002 and 0x2}$$

8.7 to 16.7% of the cases:

$$\texttt{0x0} = \texttt{0x4000000000000002 and 0x0}$$

Step by step analysis on Intel Core i3


Faulted program

Faulted data

Faulted instruction

Registers

Pipeline

Memory

Decode

Execute

Fetch

Bus

Cache

MMU

Fault on instructions ?

$$instruction = opcode + destination + operands$$

In 36.6 to 56.5% of the cases: second operand faulted

`mov rbx, rbx`

→ `mov rbx, rax` (38.46%)

→ `mov rbx, rcx` (15.38%)

→ `mov rbx, rdi` (46.15%)

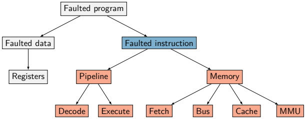
`orr rbx, rbx`

→ `orr rbx, rax` (77.78%)

→ `orr rbx, rcx` (22.22%)

Instruction corruption fault model

Step by step analysis on Intel Core i3


Faulted program
Faulted data
Faulted instruction
Registers
Pipeline
Memory
Decode
Execute
Fetch
Bus
Cache
MMU

Fault in the pipeline or in the memory ?

Usual memory fault models:

- X Instruction skip
- X Instruction replacement
- X Instruction repetition
- ✓ Instruction corruption

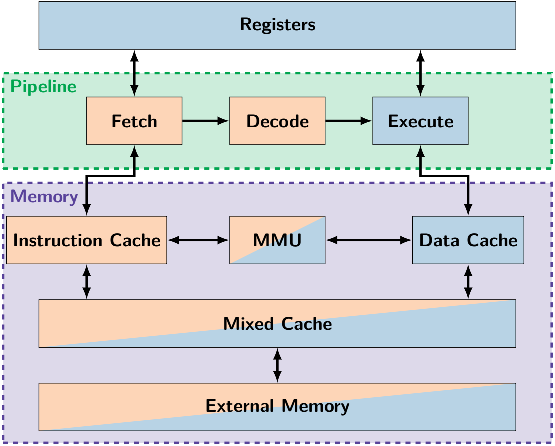
Usual pipeline fault models:

- ✓ Instruction corruption

Where does the fault appear ?

(fetch, decode, execute, cache, bus)

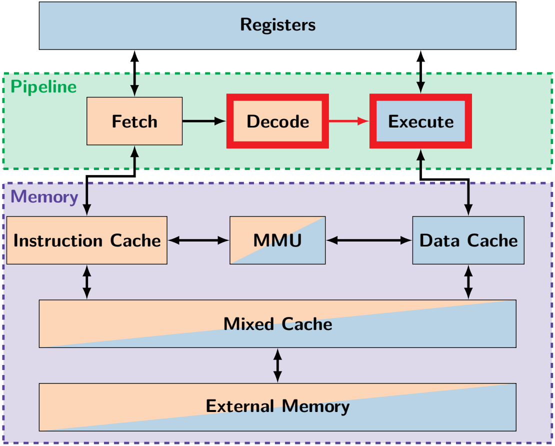
Step by step analysis on Intel Core i3



Micro-architectural blocks manipulating: Data Instructions

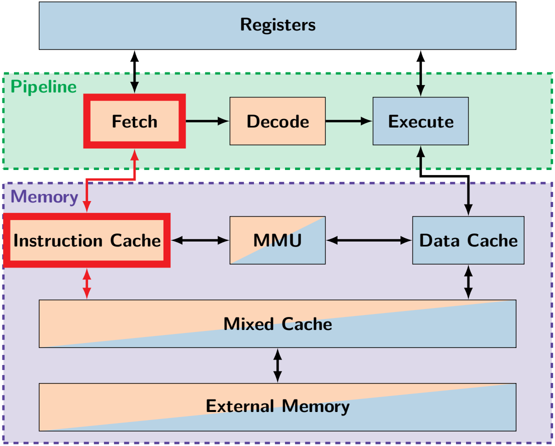
Buses: ⟷

Step by step analysis on Intel Core i3



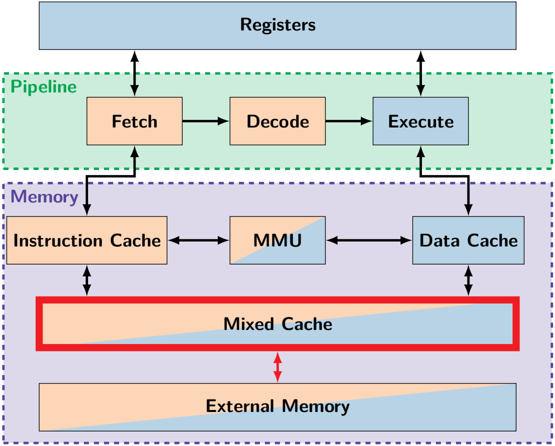
Is fault after decoding ? → fault instructions with different encoding

Step by step analysis on Intel Core i3



Is the fault in the dedicated part of the memory ? → fault data memory access

Step by step analysis on Intel Core i3



Is the fault in the mixed part of the memory ? → fault data memory access

Step by step analysis on Intel Core i3

Results of the characterization

- Fault identified in the registers and the dedicated to instructions memory subsystem (fetch or instruction cache)
- Instruction fault targets the second operand
- Data fault is a bit reset

Conclusion

On the method

- ✓ Able to identify faulted micro-architectural blocks using ISA
- ✓ No hardware access or tools needed
- X Non exhaustive
  - Some faulted values are not identified
  - Some optimizations (shadow registers, *etc*) are not considered

Next steps

About the characterization method

- Improve the model by adding micro-architectural blocks (pre-fetch, line buffers, *etc*)
- Confirm the relevance of these blocks in the fault characterization
- Enlarge the scope of the used values for the fault characterization

About the fault model usage

- Integrate the fault model characterization into an attack evaluation process
- Focus on the perturbation of other modules than the CPU

Questions?

References

[BGV11] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. "An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs". In: *Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*. Ed. by Luca Breveglieri et al. IEEE Computer Society, 2011, pp. 105–114. ISBN: 978-1-4577-1463-4.

[Kum+18] Dilip S. V. Kumar et al. “An In-Depth and Black-Box Characterization of the Effects of Laser Pulses on ATmega328P”. In: *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*. Ed. by Begül Bilgin and Jean-Bernard Fischer. Vol. 11389. Lecture Notes in Computer Science. Springer, 2018, pp. 156–170. ISBN: 978-3-030-15461-5.

[Mor+13] Nicolas Moro et al. “Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller”. In: *Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, August 20, 2013*. Ed. by Wieland Fischer and Jörn-Marc Schmidt. IEEE Computer Society, 2013, pp. 77–88. ISBN: 978-0-7695-5059-6.

[Pro+19] Julien Proy et al. “A First ISA-Level Characterization of EM Pulse Effects on Superscalar Microarchitectures: A Secure Software Perspective”. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES 2019, Canterbury, UK, August 26-29, 2019.* ACM, 2019, 7:1–7:10. ISBN: 978-1-4503-7164-3.

[Riv+15] Lionel Rivière et al. “High precision fault injections on the instruction cache of ARMv7-M architectures”. In: *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*. IEEE Computer Society, 2015, pp. 62–67. ISBN: 978-1-4673-7420-0.