TRAITOR :
a multi clock-glitch attack platform reproducing EMI effects at low-cost.

par Ludovic Claudepierre

© Inria / Photo C. Morel
Plan

1. Electromagnetic Injection (EMI)

2. Clock behaviour in presence of EMI

3. TRAITOR
Hacking IoT with Fault attack

- Fault attack: runtime modification of the firmware
- Applications: retrieve a crypto-key, bypass any security mechanism
- Main difficulty: microcontroller is a black-box
Plan

1. Electromagnetic Injection (EMI)

2. Clock behaviour in presence of EMI

3. TRAITOR
Faustine

Inside the Faraday cage: magnetic probe in the close vicinity of the targeted chip

Waveform generation:
- Delay generator
- Signal generator
- Amplifier
### Capability of EMI

#### Virtual NOP by modifying OPCODE [Moro et al. 2013]
- Random change of the OPCODE
- No side effects
- Behaviour: as if the targeted instruction was a NOP

#### Corrupt data [Moro et al. 2013]
- On LDR instruction
- Random change of the loaded data

#### Skip the fetch of instructions [Rivière et al. 2015]
- Skip the fetch of new instructions
- Re-execute the previously fetched instructions
Fault attacks by Electromagnetic Injection

Pros:
- Non-invasive ✓
- Reproducible ✓

Cons:
- Many parameters to tune ×
- Low success rate (30%) ×
- Expensive hardware apparatus ×
- Limited number of fault ×
What is the cause of that unusual behaviour?

What if we take control of the clock signal and recreate this glitch whenever we want?
Plan

1. Electromagnetic Injection (EMI)

2. Clock behaviour in presence of EMI

3. TRAITOR
EMI efficiency vs probe location

- Strong influence of EMI on clock signal
- Sensitive location = analog feeding pins (including PLL)
- Crystal clock only → fault rate close to 0%

**Figure 1** – Comparison of fault injection mapping with STM32F100RB-LQFP64 PIN map.

**Injection parameter**
- 4 sinus periods
- Frequency : 275 MHz
- Power : 175 W
- Delay : 188.5 ns
**Clock generation by Phase-Locked Loop (PLL)**

- **Input reference** = Crystal (8 MHz on for STM32F100RB)
- **VCO output** wired to clock tree
- **Phase-frequency detector** → phase comparison VCO vs Crystal
- **Phase difference** → voltage correction on VCO
- **Advantage** =
  - Frequency higher than with crystal only
  - Frequency chosen by user
Hypothesis on mechanism

- Global injection inefficient
- Shape of the glitch $\simeq$ shape of VCO output when phase jump
- Hypothesis:
  - disruption on one of the comparator input
  - detection of phase-jump
  - voltage correction on VCO
  - glitch on VCO output

Theoretical VCO signal due to phase jump.

Future works

- Confirm the hypothesis by simulation
- Determine the relation between glitch amplitude and phase-difference
- Deduce the shape of the radiated wave for a more efficient EMI
<table>
<thead>
<tr>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Electromagnetic Injection (EMI)</td>
</tr>
<tr>
<td>2. Clock behaviour in presence of EMI</td>
</tr>
<tr>
<td>3. TRAITOR</td>
</tr>
</tbody>
</table>
Reproducing EMI effects in a cheaper way

TRAITOR = FPGA Artix-7
Target = STM32F100RB

Pros:
- Cheap (∼ 100€) ✓
- A lot of glitches in a single execution ✓
- High success rate (∼ 99%) ✓
- Easily transportable ✓

Cons:
- Access to the crystal required ✗

Can completely edit the targeted program during its execution
TRAITOR signal generation

Methods of generation

- Generation of 2 signals slightly unphased
- Glitch: $Clk_{out} = (Clk_1 \oplus Clk_2) \cdot Clk_1$
- Parameterization of delays by user
- Switch output to the glitch according to these delays
DEM0 : Hacking an almost secure PIN implementation

```c
if (check_result(result)){
    // State 0
    Green_light_on( ) ;
    Blue_light_off( ) ;
    if (check_result(result)){
        // State 1
        Green_light_on( ) ;}
    else{
        // State 2
        Blue_light_on( ) ;}
}
else{
    // State 3
    Blue_light_on( ) ;
    Green_light_off( ) ;
    if (check_result(result)){
        // State 4
        Green_light_on( ) ;}
    else{
        // State 5
        Blue_light_on( ) ;}
}
```

Target = **STM32F100RB**

**Fault on Double PIN verification**

By default : wrong code PIN is sent to the device ⇒ Blue

- **STATE 1** : Green ⇒ Right PIN or Intrusion undetected
- **STATE 2** : Blue + Green ⇒ Intrusion warning
- **STATE 4** : Blue + Green ⇒ Intrusion warning
- **STATE 5** : Blue ⇒ Wrong PIN
DEM0 : Hacking an almost secure PIN implementation

```c
if (check_result(result)){
    // State 0
    Green_light_on( ) ;
    Blue_light_off( ) ;
    if (check_result(result)){
        // State 1
        Green_light_on( ) ;
    }
    else{
        // State 2
        Blue_light_on( ) ;
    }
}
```

```
800057c: f000 f91e   bl   80007bc <check_result>
8000580: 4603   mov   r3, r0
8000582: 2b00
cmp   r3, #0
8000584: d027   movs   r2, #1
8000586: 2201   ldr   r0, [pc, #484] ; (8000774 <main+0x4f0>)
8000588: f44f 7100  mov.w  r1, #512 ; 0x200
800058a: 4879   ldr   r3, [pc, #472] ; (8000778 <main+0x4f4>)
800058c: f402 fb54  bl 8002c3a <HAL_GPIO_WritePin>
8000590: f44f 7180  mov.w  r1, #256 ; 0x100
8000592: 4876   ldr   r0, [pc, #440] ; (8000774 <main+0x4f0>)
8000594: 681b   bl 80007bc <check_result>
8000596: 4603   mov   r3, r0
8000598: 2b00
cmp   r3, #0
800059a: d009   mov   r0, r3
800059c: 4b73   ldr   r3, [pc, #460] ; (800077c <main+0x4f8>)
800059e: 2201   movs  r2, #1
80005a0: 601a   str   r2, [r3, #0]
80005a2: 2201   ldr   r3, [r3, #0]
80005a4: f44f 7100  mov.w  r1, #512 ; 0x200
80005a6: 486e   ldr   r0, [pc, #440] ; (8000774 <main+0x4f0>)
80005a8: f002 fb3d  bl 8000b2c3a <HAL_GPIO_WritePin>
80005aa: e033   b.n 800062a <main+0x3a6>
80005ac: 4b6e   ldr   r3, [pc, #440] ; (800077c <main+0x4f8>)
80005ae: 2202   movs  r2, #2
80005b0: 601a   str   r2, [r3, #0]
80005b2: 2201   ldr   r3, [r3, #0]
80005b4: f44f 7100  mov.w  r1, #512 ; 0x200
80005b6: 486e   ldr   r0, [pc, #440] ; (8000774 <main+0x4f0>)
80005b8: f002 fb3d  bl 800062a <HAL_GPIO_WritePin>
80005ac: 2b00   ldr   r3, [pc, #460] ; (800077c <main+0x4f8>)
80005ae: 2201   movs  r2, #1
80005b0: 601a   str   r2, [r3, #0]
80005b2: 2201   ldr   r3, [r3, #0]
80005b4: f44f 7180  mov.w  r1, #256 ; 0x100
80005b6: 4869   ldr   r0, [pc, #420] ; (8000774 <main+0x4f0>)
80005b8: f002 fb3d  bl 8002c3a <HAL_GPIO_WritePin>
80005ba: e029   b.n 800062a <main+0x3a6>
```
DEMO: Hacking an almost secure PIN implementation

800057c:  f000 f91e   bl   80007bc <check_result>
8000580:  4603   mov   r3, r0
8000582:  2b00   cmp   r3, #0
8000584:  d027   beq.n  80005d6 <main+0x352>
8000586:  2201   movs   r2, #1

2 possibilities to bypass the tests

- CMP not executed (in the hypothesis, the ASPR register is by default in the right state)
- Beq not executed → branch “PIN ok”

Fault model

- Skip instruction fetch and re-execute the instruction(s) previously fetched
- Cortex-M3 = instruction fetched 2 by 2
- !!! Depending in instructions around, fault is not that easy !!!
TRAITOR capabilities

Instruction fault

- Execute twice: mov, ldr, add, push, pop
- Skip fetch of str, mov, ldr, add, push, pop, bl, cmp, bx
- No fetch of some instructions induces most of the time (except str) to re-execute the already fetched instructions
- If wide instruction (32 bits), 1 instruction “nop”.

Application

- Bypass counters by incrementing artificially
- Bypass function (particularly security functions) to avoid countermeasures
- Activation of dead code
- Activation of back-doors
- Rewriting completely the code at run-time combining the previous items
Fun Facts

Glitch voltage influence

- Fault on MOV, LDR, ADD, STR, BL $\sim [630 \text{ mV} ; 950 \text{ mV}]$
- Depending on the code, for a same clock edge, different voltage induce different effects

Exotic behaviour 01

- Fault just after fetch BL $[630 \text{ mV} ; 1,3 \text{ V}]$
- LR data copied in the destination register of the fourth instruction before branch
- When replacing LDR by MOV Rd, Rm, LR copied in Rm

Exotic behaviour 02

- NOP of $LDR \ R0$, and $LDR \ R1$, glitch $= [550 \text{ mV} ; 670 \text{ mV}]$ and $[770 \text{ mV} ; 870 \text{ mV}]$
- **Get out of the function** after the $MOV \ R3, \ #0$, glitch $= [670 \text{ mV} ; 770 \text{ mV}]$
- Strange behaviour independent of the instructions after the branch
Conclusions - Perspectives

Conclusions on TRAITOR

- Light and transportable platform, easy to use
- Take control of clock signal and inject fault
- Multi-fault → can edit a program at run-time and deeply change its goal

Perspectives

- Continue to experiment faults on instruction set
- Applied TRAITOR to other target (TI chip for example)
- Applied multi-fault on real program → application case
Thank you!

Board of an everyday object with STM32F2 and its Crystal

Questions?