Efficient attack-surface exploration for electromagnetic fault injection

Daniele Antonio Emanuele Carta[3][0000−0001−7053−3695],
Vittorio Zaccaria[2][0000−0001−5685−9795],
Gabriele Quagliarella[1], and
Maria Chiara Molteni[1][0000−0003−2901−2972]

[1] Security Pattern, Italy
[2] Politecnico di Milano, Italy
[3] STMicroelectronics, Italy

Abstract. Electromagnetic Fault Injection is a physical attack that aims to disrupt the operation of hardware circuits to bypass existing confidentiality and integrity protections. The success probability of the attack depends, among other things, on many different variables such as the probe used to inject the pulse, its position, the pulse intensity, and duration. The number of such parameter combinations and the stochastic nature of the induced faults make a comprehensive search of the parameter space impractical. However, it is of utmost importance for hardware circuit manufacturers to identify these vulnerability points efficiently and introduce countermeasures to mitigate them.
This work presents a methodology to efficiently identify the subregion of the attack parameter space that maximizes the occurrence of a *informative* fault. The idea of this work consists in applying a multidimensional bisection method and exploiting the equilibrium between a pulse that is too strong and one that is too weak to produce a disruption on the circuit's operation. We show that such a methodology can outperform existing methods on a concrete, state-of-the-art embedded multicore platform.

Keywords: Electromagnetic Fault Injection, Parameters Search, Optimization, Methodology, Fault Model, System On Chip

1 Introduction

Today, System-on-Chips (SoCs) are increasingly used for sensitive tasks such as secure payments, critical infrastructure management, and other mission critical applications characterized by confidentiality and integrity constraints. However, SoCs are complex architectures that might present a vast attack surface, which is difficult to protect. For this reason, they are increasingly equipped with Trusted Execution Environments (TEEs), which are small, isolated processing environments whose attack surface is more easily under control.

Ensuring a completely safe TEE is not a simple task, as the *boomerang* attack has shown [12]; However, even if the attack surface could be reduced to zero, fault injection (FI) could still be used to force the system to work outside its nominal conditions and expose otherwise absent vulnerabilities, perhaps justifying such an increase in research efforts.

D. A. E. Carta completed this work while at Security Pattern.

Fault injection is all about disrupting the nominal operation of a circuit by invalidating design-time assumptions around the environment. A successful injection could be used to trigger instruction execution skips or corruption in working data with obvious consequences[1]; In fact, one of the most defining aspects of FI is the need to have physical access to the target[2]. Injection can be performed in several ways that vary in terms of equipment cost, invasivity, and robustness, e.g. altering the working temperature of the system, the clock signal, the power supply, and/or the system internal signals. The latter effect could be induced through either microprobing, a coherent light source (if the circuit has been decapped), or by injecting electromagnetic pulses (EMFI).

EMFI is particularly interesting because it represents a potentially cheaper (see the ChipShouter project [17]) than other methods but with more degrees of freedom. However, its cost-performance trade-off is characterized by less precise control over the fault injection position (with respect to optical or microprobing attacks) and the significant range of equipment configurations that can be used to perform it, such as the electromagnetic probe position, the voltage, the intensity, etc., which we call the *attack surface*.

This work stems from our efforts to overcome the two limits of conventional approaches, i.e.,

- using trial and error tests with the risk of leaving out interesting exploitable points [20], and
- targeting the identification of a *single* (X, Y) faulty point by adopting some sort of occurrence ratio with the side effect of reducing fault differentiation [8].

This work proposes a target-agnostic methodology to efficiently search the EMFI attack surface for potentially exploitable configurations. We overcome the inherent limits of an exhaustive search (which is unfeasible) and a random search (which is suboptimal) by addressing, through a multidimensional bisection method, the probe position problem and the pulse configuration problem.

This paper is structured as follows. Section 2 introduces the state-of-the-art EMFI attacks to facilitate understanding of the motivation and problem statement of this work. Section 3 introduces the actual methodology, which is then validated by appropriate experimentation in Section 4. Section 5 concludes the paper with an outline of possible future work.

2 State of the art

Research efforts on EMFI have focused on understanding its effects (inference of the fault model), improving fault success rates, and building/validating attacks. The last two challenges are based on the tooling to perform the EMFI and the

[1] Being able to skip a branch instruction could, for example, bypass security checks.

[2] This is not a requirement as some fault-injection attacks might work even remotely (e.g., *clkscrew* [22] and *rowhammer*).

development of methodologies that integrate it with the fault model, a particular methodological issue being the exploration of the attack surface. When dealing with EMFI against programmable microcontrollers (MCU), we can identify a broad division between practical approaches, targeting FPGA or ASIC SoCs, and methodological contributions, summarized in Table 1.

2.1 EMFI on FPGA

FPGA technology, due to its lower clock frequency and hardware complexity, was a great starting point for white-box analysis of EMFI effects. For example, Moro et al. [16] have built an RTL model that predicted timing constraint violations on flash memory bus transfers. Their experiments (on a 56MHz FPGA target) confirmed that an attacker could corrupt instructions fetched from memory. Similarly, Ordas et al. [19] have introduced a more refined model, which takes into account the corruption of internal registers' (flip-flop) data, essentially making it independent of the clock frequency. With a similar white-box approach, Menu et al. [15] derive a model that explains the corruption of data fetches from flash memory. Other researchers [4] have provided evidence and theoretical justification for a successful EMFI with pulses that are shorter than the target clock cycle. For example, Dutertre et al. [6] have introduced an instruction skip model that shows 100% repeatability on a single precise instruction that could be extended to deal with more than one instruction in different moments.

2.2 EMFI on ASIC SoCs

Commercial ASIC-based SoCs (generally based on application-class MPUs) introduce a whole new level of complexity in fault modeling. Researchers cannot apply white-box approaches anymore, as they do not control the underlying technology, and have to work with clock frequencies higher than their FPGA counterparts which makes synchronization difficult. Hummel et al. [10] is one of the first approaches in this field to successfully deal with a precise synchronization between raised exceptions and pulse timing. Ang et al. [3] try to overcome the synchronization problem by employing a second-order EMFI attack, which consists of attacking a secondary component to affect the primary target (by targeting an external DRAM running at a 40MHz clock to disrupt the execution of the faster processor). This and other approaches, such as Kuhnapfel et al. [11], are characterized by relatively low-cost equipment ranging from $350 to $7000. Other works resorted to trial-and-error approaches to explain faults at higher levels. Proy et al. [20] (inspired by Dereuil et al. [5]) are among the first to define a CPU fault model based on the Instruction Set Architecture, while Trouchkine et al. [23] try to explain faults using architectural features such as register, pipeline, MMU, and caches. Finally, Gaine et al. [8] present an interesting hybrid approach consisting of privilege escalation in a Linux environment; here the target is a 1.2 GHz mobile SoC for which they have a white-box view. They are the first to introduce the concept of crash susceptibility, which we will exploit in the remainder of this work. However, they were unable to carry out

the planned attack in a real-world scenario due to serious timing-synchronization issues with the fast target.

Type	Year	Work	Target
MCU	2013	Moro et al. [16]	ARM Cortex-M3
	2017	Ordas et al. [19]	Xilinx Spartan 3-1000 ARM Cortex-M4
	2019	Menu et al. [15]	Atmel SAM3X8E ARM Cortex-M3
		Dumont et al. [4]	Custom designed
	2021	Dutertre et al. [6]	ATmega328P
SoC	2014	Hummel [10]	ARM Cortex-A8
	2017	Ang et al. [3]	Cisco 8861 IP Phone Broadcom BCM11123 SoC
	2019	Proy et al. [20]	ARM Cortex-A9
		Trouchkine et al. [23]	ARM BCM2837 x86 Intel Core i3-6100T
	2020	Gaine et al. [8]	ARM Cortex-A53
	2022	Kuhnapfel et al. [11]	x86 AMD Ryzen 5 2600
Methods	2013	Omarouayache et al. [18]	Probes
		Carpi et al. [2]	Smartcards
	2019	Madau et al. [13]	ARM Cortex-M3 ARM Cortex-M4
		Maldini et al. [14]	ARM Cortex-M4
	2022	Gaine et al. [9]	Probes

Table 1: Summary of the state-of-the-art for EMFI.

2.3 Existing methodologies

Methodological approaches are more interested in maximizing the amount of information that can be obtained from an experimentation campaign than in a successful exploit. In fact, there is an overall underrated aspect of fault injection, that is, how and where to reliably reproduce a fault in the first place. The probe reliability and selection problem, originally addressed by Omarouayache et al. [18] is less difficult today than it was 10 years ago. Toolkit producers such as NewAE, eShard, Riscure and other vendors commercialize state-of-the-art probes with their offerings (whose accuracy obviously depends on the cost). However, identification and exploration of the configuration setup for fault injection is still in its infancy, although initial steps have been proposed in 2013 by Carpi et al. [2] in the field of Voltage Fault Injection. They address the problem of identifying the subspace of the duration and intensity values of pulses that could produce an actual fault with a two-step process, that is, trying to optimize the parameters separately. Maldini et al. [14] bring this work to EMFI through an evolutionary algorithm that tries to find the optimal geometric and

pulse intensity values that maximize *fault occurrence ratio* while keeping some of the configuration fixed (pulse duration). Madau et al. [13] offer an alternative methodology to locate the best areas to obtain unexpected behaviors on the surface of the chip; Each surface point, starting from a predefined grid, is rated using a *susceptibility criterion* that requires measuring electromagnetic emissions. In its testing environment, the criterion has efficiently led to the identification of 50% of the surface that produces a covering of 80% of the faulty surface. However, the susceptibility criterion requires expensive equipment to measure electromagnetic emissions. Furthermore, the criterion test is performed with a fixed pulse intensity and duration, while different durations and intensities could provide different results.

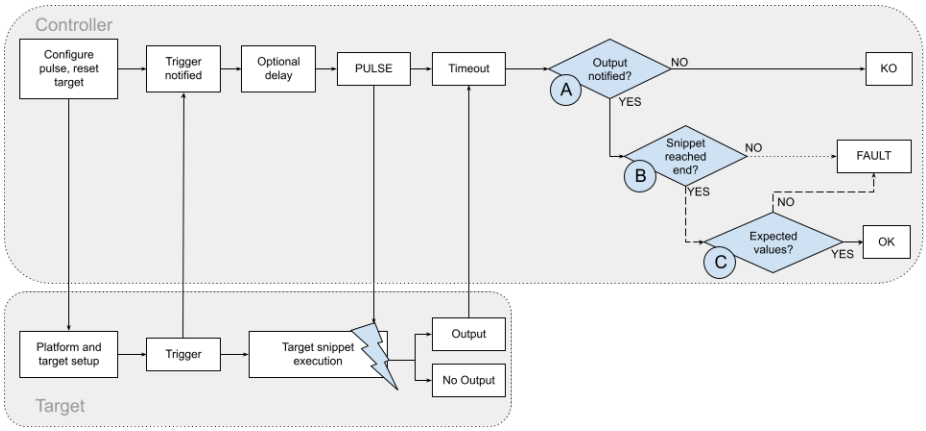


Fig. 1: Flowchart of evaluating the outcome of a single EMFI. A `FAULT` may be exploitable or not depending on the path that lead to it. Exploitable `FAULT`s follow the dotted arrow path, and Non-exploitable `FAULT`s follow the dashed arrow path.

The current state of affairs is not satisfactory for several reasons. First of all, each of the above approaches has a set of setup variables which are fixed to some value perhaps identified through trial and error. This is done, of course, to limit the complexity of the analysis of the attack surface, but could leave some interesting exploitable points out of scope. Our work aims to provide a methodology that starts with instruments and setup capabilities and leaves nothing behind without an explanation.

Second, most of the existing approaches adopt some sort of *occurrence ratio* as a maximization objective to find a single (X, Y) chip surface coordinate. Instead, we aim to derive multiple points of the attack surface to enhance fault differentiation in the hope that nonfrequent faults are more informative.

Third, as suggested by other authors [2], there are better strategies than random search [21] to improve both efficiency and efficacy. In fact, probe movement

associated with random search introduces too much error and should be reduced as much as possible. However, so far, there has been no clear indication on how probe coordinates should be explored.

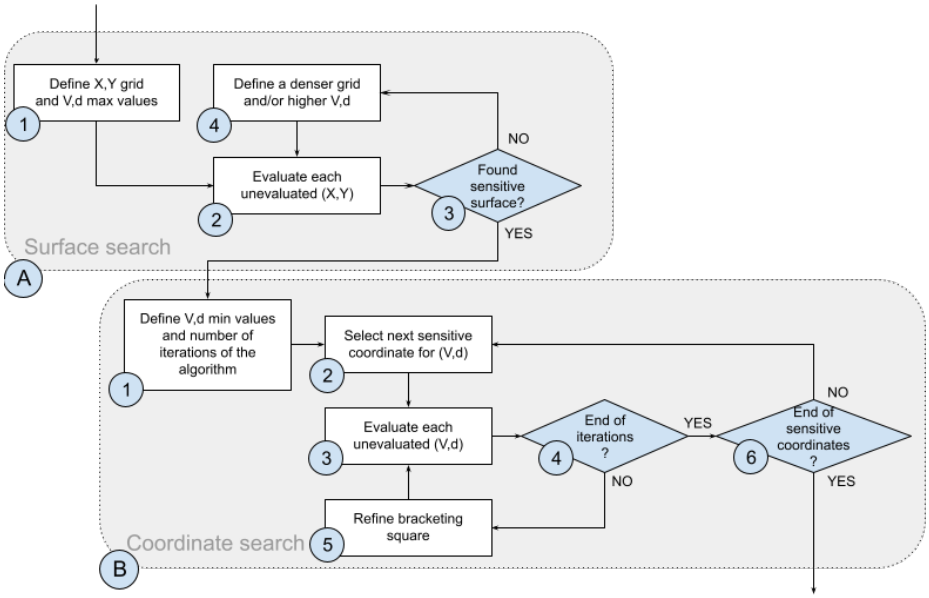


Fig. 2: Flowchart of the proposed search methodology.

3 Methodology

Following [14], we assume a *controller/target* evaluation setup such as the one represented in Figure 1. The controller is responsible for guiding the injection probe on the target by modifying the coordinates (X, Y) of the probe, the intensity V and the duration d of the square pulse. Each injection of faults is modeled as a function $\text{EMFI}(X, Y, V, d)$ with three possible outcomes.

- `OK`: the target output is as expected.
- `KO`: the target locks up, freezes, resets, or does not produce a result.
- `FAULT`: anything else; this is the most rare behavior and is divided further into:
  - Informative: the `FAULT` that does not prevent the code under test from reaching its end, but does not show expected values.
  - Noninformative: any other `FAULT` such as processor *exceptions* of any kind.

Since EMFI is a probabilistic attack, we will need to work with statistics associated with n fault injections, which will provide, for each outcome $o \in \{\texttt{OK}, \texttt{KO}, \texttt{FAULT}\}$, its probability $P_o(X, Y, V, d, n)$. The problem is to efficiently identify the subregion of the attack space $X \times Y \times V \times d$ that maximizes probability $P_{\texttt{FAULT}}$ for each coordinate that meets a susceptibility criterion, without resorting to a random search where n is usually set depending on time constraints.

The proposed methodology is based on the idea that $P_{\texttt{FAULT}}$ is non-negligible where $P_{\texttt{OK}}$ and $P_{\texttt{KO}}$ balance out. In fact, we rely on the idea that a pulse "too weak" ($P_{\texttt{OK}} \gg P_{\texttt{KO}}$) is not sufficient to cause the target fault. At the same time, a pulse that is "too strong" ($P_{\texttt{KO}} \gg P_{\texttt{OK}}$) may disturb execution too much. Our strategy is carried out in two steps: 1) reducing the physical surface of the target (X, Y) to only points susceptible to faults (*susceptible surface search*), and 2) identifying the intensity V and duration d of the pulse through a multidimensional bisection algorithm [1] (*coordinate search*). The methodology, shown in Figure 2, is agnostic to the target architecture and relies only on the observability of an outcome, which could be a led lighting up or a log message from a debug console. The two methodological steps are outlined in the following two subsections.

3.1 Surface search

The search for the susceptible area (Figure 2, A) consists of first defining a grid G of coordinates and the maximum intensity and duration of the pulse. This is done by measuring the spatial dimensions of the target $X_{min}, Y_{min}, X_{max}, Y_{max}$, choosing a grid *step* according to the precision of the probe positioning mechanism and defining the maximum value of intensity V_{max} and duration d_{max} exactly below the values that risk damaging the target.

Then, finally, evaluate

$$\text{EMFI}(X_i, Y_j, V_{max}, d_{max}, \bar{n})$$

on the grid G and a number of experiments $\bar{n}$ and derive the subset of "susceptible" surface points (S), i.e., those points that show at least some `KO` or `FAULT` result:

$$\begin{aligned} S(G) = \{(\bar{X}, \bar{Y}) | (\bar{X}, \bar{Y}) \in G \wedge \\ P_{\texttt{KO}}(\bar{X}, \bar{Y}, V_{max}, d_{max}, \bar{n}) + P_{\texttt{FAULT}}(\bar{X}, \bar{Y}, V_{max}, d_{max}, \bar{n}) > 0\} \end{aligned}$$

If no susceptible points are found, run the procedure again on a grid with a smaller step, higher intensity, and/or duration of the pulse.

3.2 Coordinate search

This phase of the methodology (Figure 2, B) is based on the idea that $P_{\texttt{FAULT}}$ is non-negligible where $P_{\texttt{OK}}$ and $P_{\texttt{KO}}$ balance out. In practice, we formalize the

coordinate search problem by finding the root of the following equation for susceptible points $S(G)$ and a fixed number of experiments $\bar{n}$:

$$E(V,d) = \{P_{\texttt{KO}}(\bar{X},\bar{Y},V,d,\bar{n}) - P_{\texttt{OK}}(\bar{X},\bar{Y},V,d,\bar{n}) \simeq 0, (\bar{X},\bar{Y}) \in S(G)\} \quad (1)$$

Note that the above equation potentially defines a line in the (V,d) space which is the solution of interest. To optimally search for this line, we assume that the function E is smooth, that is, small changes in (V,d) bring small changes to E and that it increases monotonically with V and d. These conditions allow for the use of a proper adaptation of the multidimensional bisection algorithm [1], which will allow the identification of rectangular regions that contain the target line (V,d). Each such rectangular region is called "bracketing rectangle" and is such that at least two of its vertices i,j trigger a sign difference for E of at least magnitude $2\epsilon, \epsilon \geq 0$, where ϵ is a control parameter of the bisection method[3]:

$$E(V_i,d_i) < -\epsilon \wedge E(V_j,d_j) > \epsilon \quad (2)$$

The algorithm starts by considering the coordinates of a rectangular region of the space (see Figure 3)

$$R = \{(V_{min},d_{min}),(V_{max},d_{min}),(V_{max},d_{max}),(V_{min},d_{max})\}$$

If the rectangle is bracketing, then it is divided into 4 equal subrectangles; the search is then repeated for those subrectangles that are bracketing until a maximum number of iterations is reached or there are no more bracketing rectangles.

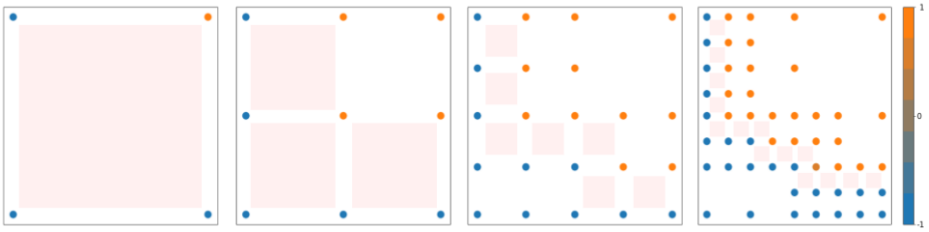


Fig. 3: Example iterations (0 to 3) of the Bi-dimensional bisection with $\epsilon = 0$. Vertices with $E > 0$ are highlighted in orange, those with $E < 0$ are highlighted in blue, while bracketing rectangles are highlighted in red. In iteration 1 the top right rectangle is not bracketing since it has no vertex V with $E(V,d) < 0$. Note that measurement units for X and Y axis are different (Volts vs milliseconds)

The maximum number of iterations I_{max} is given by the discrete nature of the parameters V and d:

[3] It is an indirect stop criterion for the bisection method. The higher ϵ, the lower the bar will be set to recognize the rectangles as bracketing rectangles, and thus continue the search.

$$I_{max} = min\left(\left\lfloor \log_2\left(\frac{V_{max} - V_{min}}{V_{step}}\right)\right\rfloor, \left\lfloor \log_2\left(\frac{d_{max} - d_{min}}{d_{step}}\right)\right\rfloor\right) \tag{3}$$

where V_{step} and d_{step} are determined by the precision of the equipment / setup.

In the worst-case scenario (that is, a function with roots right above the bottom left perimeter of the initial bracketing rectangle) and at iteration I, the bisection algorithm must evaluate $2^{I+2} - 3$ vertices $\bar{n}$ times; thus, we obtain the following bound for the number of experiments N:

$$N \leq 4\bar{n} + \left(\sum_{I=1}^{I_{max}} 2^{I+2} - 3\right)\bar{n} \tag{4}$$

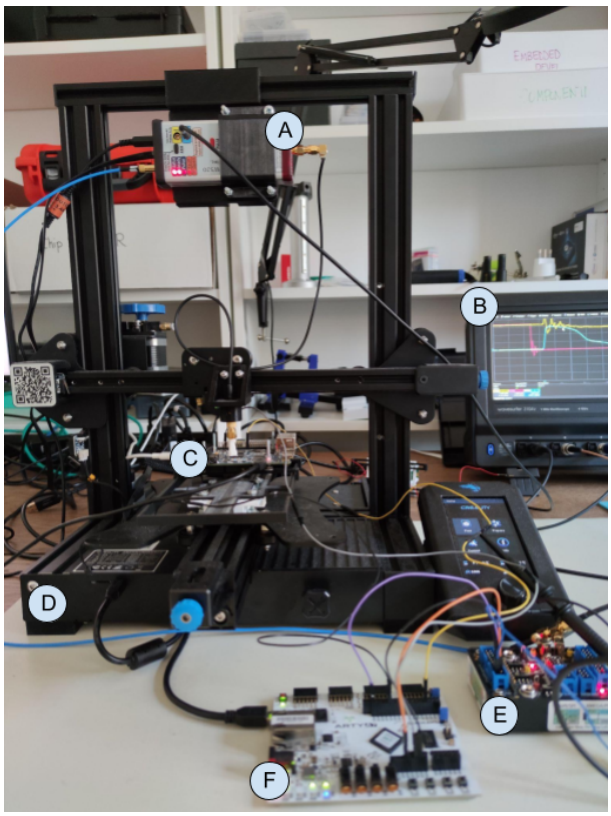


Fig. 4: Chipshouter (A), Oscilloscope (B), Target (C), 3D printer (D), Voltage translator (E) and FPGA (F).

4 Experimental validation

This section presents an experimental validation of the methodology presented in the previous section. We will introduce the setup of the injection platform and the target, as well as a qualitative and quantitative evaluation of the efficacy in identifying informative faults.

The hardware and software components of our setup are built around the ChipShouter platform for a comprehensive budget of less than 5K€ (excluding oscilloscope) and a standard laptop used to control the following parameters (see Figure 4):

- duration of d pulse injection to as low as 10ns, through an Artix-7 35T Arty at 100MHz
- (X, Y) position of the fault injection probe, through a 3d printer with a 0.1mm resolution
- intensity V of the pulse (directly on the ChipShouter).

The setting allows us to produce pulses with d ranging from 10 to 600 ns with a 10ns resolution and V ranging from 150 to 500 V, with a 1V resolution. To control the platform, we used the following software tools:

- Raiden[7], an open source FPGA project to handle the delay between the target and the pulse triggers. It also controls the duration of the trigger, allowing the pulse to last a fixed number of clock cycles. Finally, it resets the target to perform new experiments.
- OctoPrint, a 3D-printer control application.
- A Python app that orchestrates Raiden, Octoprint, and the ChipShouter APIs to configure and collect the target output through a serial interface.

The target is an ARMv7 dual core, dual issue SoC that mounts a Cortex A7 with eight pipeline stages with data and instruction caches disabled. It runs at 600MHz and does not perform any speculative execution. The chip has not been decapped, and we do not have information on the internal layout. The target has a serial port that is used by the central workstation to read the output of the experiments performed on it. The chosen target offers a standard procedure for building and deploying everything necessary for a robust and secure boot chain. We position our victim code in the First Stage Boot Loader (FSBL) of Trusted Firmware-A. Putting the victim code at this point in the boot-chain simplifies the collection and interpretation of the results; in particular, the code runs on a single core and allows us to minimize the time window for testing.

The victim code has a standard template; the initial part of the template triggers the pulse through a GPIO pin:

```
;Pulse trigger
            bl    set_gpio
            mov   r0, #89            ; 0x59
            bl    clk_enable
```

```
ldr   r3, [pc, #124]      ; address for gpio high
movs  r2, #128            ; 0x80
str   r2, [r3, #0]        ; set gpio high
```

It then sets each register from r0 to r12 to a unique value (`r0=0x41414141`, `r1=0x42424242` ... `r12=0x53535353`) to recognize any unexpected/random change in its content:

```
;REGISTERS SETTING
              mov.w  r0, #1094795585   ; 0x41414141
              mov.w  r1, #1111638594   ; 0x42424242

              mov.w  ip, #1397969747   ; 0x53535353
```

The actual victim code (which belongs to a class of codes introduced below) is then executed, followed by a print on the serial port of the architectural state.

The code has a size limitation because it has to fit into the internal SRAM, according to the FSBL platform guidelines. The size of the FSBL image allows enough instructions to hit after accounting for the actual delay between the GPIO high instruction and the actual arrival of the electromagnetic pulse.

We designed the victim code snippets to stress the three main microarchitectural blocks of the processor: the arithmetic units, the memory subsystem (load/store unit), and the branching unit.

- A sequence of `NOP`s.
- A sequence of `ADD`s which increment register r0 by one.
- A sequence of `LDR`s instructions.
- A single `bne` instruction that jumps to itself. This snippet does not produce any output, and its `OK` and `KO` behaviors are indistinguishable (called *Loop* in the following).

4.1 Trigger and timing synchronization.

As highlighted above, we achieve synchronization between the pulse and the victim code through a GPIO pin that is controlled directly by the victim. Figure 5 shows the view, captured via oscilloscope, of the timing of the signals involved.

First, the victim sets the GPIO to high (Figure 5,A); in turn, this triggers the ChipShouter (Figure 5,B), and finally the actual electromagnetic pulse is produced (Figure 5,C). (Figure 5,D) is the actual amount of time that occurs between the victim's trigger instruction and the actual impact on the execution of the instructions (Figure 5, F). The ChipShouter delay is less than 100 ns (Figure 5, E).

4.2 Surface search

The surface mapping is performed using a 1mm grid step. Given that the edge of the square chip surface is 13mm, the resulting grid G corresponds to 169 points.

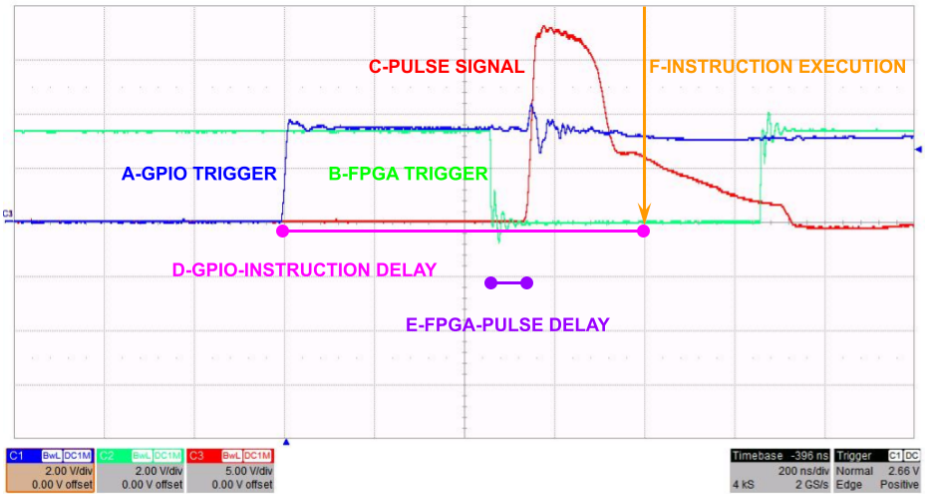


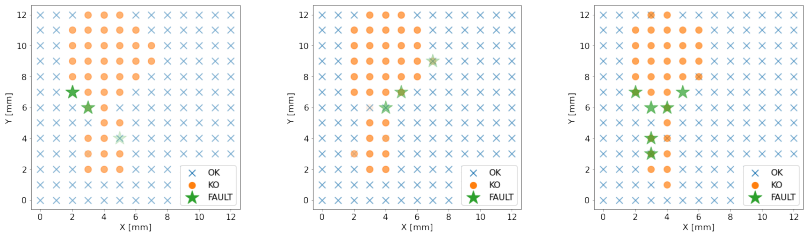
Fig. 5: Timing synchronization scheme from the oscilloscope perspective. The horizontal steps of the grid represent a period of 200 ns.

According to the previous description, we used a V_{max} at 500V, a duration of $d_{max} = 600$ ns (which is the maximum available on the ChipShouter as of 2022) and a number of experiments per point $\bar{n} = 8$.

First, we performed a surface search using the `NOP` victim code. The overall resulting dimension of the susceptible sub-grid $S(G)$ is 42, that is, 25.8% of the entire grid (see Figure 6a). In this phase, we observed very few `FAULT`s, and the majority of experiments were `KO`. The victim code `ADD` behaves similarly (see Figure 6b). Interestingly, the `FAULT`s are located on the perimeter of the `KO` sub-grid and are essentially exceptions. We also found a case where a faulty behavior did not trigger an exception, i.e., the value of a register in the computation was modified. The victim code `LDR` behaves similarly (see Figure 6c) to the other two with exceptions classified as link register abort (`LRABT`), and Data Abort. The Loop victim code is more difficult to characterize, as one can only observe either exceptions or sudden control changes that force the CPU out of the loop. Even in this case, we observed the `FAULT`s on the perimeter of the previous susceptible surface.

4.3 Coordinate search

We sampled a few points within the subgrid $S(G)$ by using $\epsilon = 0$, thus forcing the maximum iterations of the bisection method to $I = 5$. The algorithm converged towards $P_{\texttt{FAULT}}$ ranging from 30% to 80%. Once the bisection converged, some coordinates of $S(G)$ showed very different (V, d) profiles, which appeared even before reaching the maximum I, as Figure 7 shows. In particular, some points produced a high probability of fault in the upper right quadrant (Figure 7a),



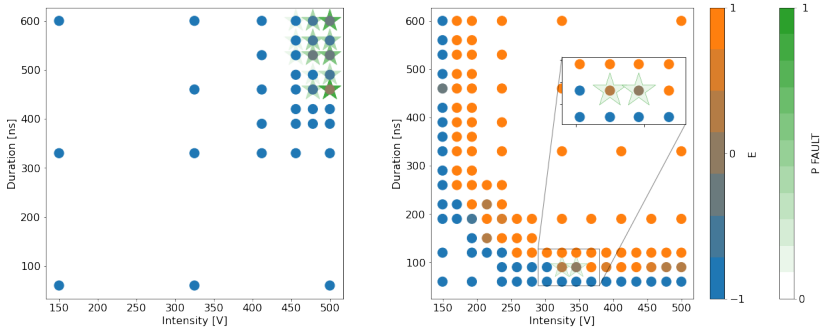
(a) `NOP`s susceptible surface.

(b) `ADD`s susceptible surface.

(c) `LDR`s susceptible surface.

Fig. 6: Surface search for different code snippets executions. Each point coordinate is evaluated 8 times at max intensity and duration of the pulse.

while some others were characterized by a very low maximum probability in the lower left quadrant (Figure 7b), which incidentally goes against some results reported earlier [8]. We do not have conclusive explanations for this conflicting behavior, which, we think, could be better explained with a decapped chip.



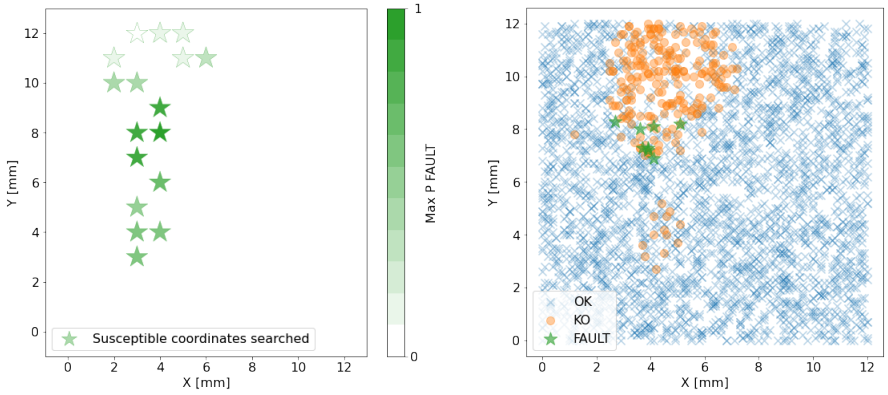
(a) (3,7) coordinate search, 4 iterations.
Min $|E| = 0$, Max $P_{\texttt{FAULT}} = 100\%$.

(b) (4,12) coordinate search, 4 iterations.
Min $|E| = 0.1$, Max $P_{\texttt{FAULT}} = 10\%$.

Fig. 7: Coordinate search. Each point evaluation corresponds to 10 experiments. The color of the round points represents the E value (range $[-1, +1]$) for the configuration. The color of the stars represents max $P_{\texttt{FAULT}}$ achieved for the configuration.

Figure 8a shows all coordinates tested with the maximum $P_{\texttt{FAULT}}$ obtained; By comparison, Figure 8b shows the results obtained when both $S(G)$ and bisection are replaced by random sampling. Given the striking difference in precision, we further investigated the efficacy of the random search at some coordinates in $S(G)$ comparing it with the bisection method (Figures 9 and 10), using as many random experiments as the amount needed for the four bisection iterations. The

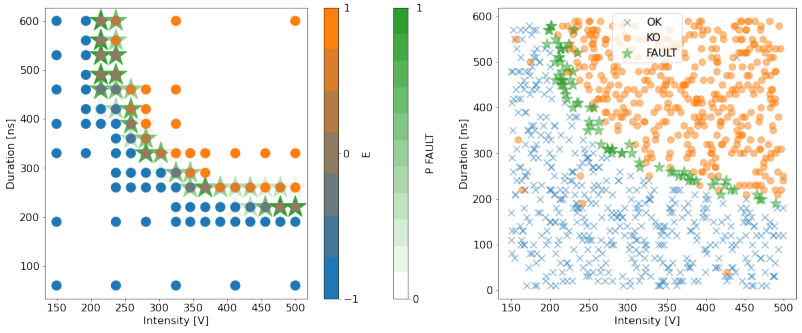
proposed bisection method obtained a number of faults that is almost triple the random one.



(a) Each coordinate color reports its max $P_{\texttt{FAULT}}$ achieved using the methodology ($I = 4, n = 10$ corresponding to 17790 experiments performed).

(b) 3200 random experiments.

Fig. 8: Validation tests on susceptibility criterion.



(a) Bisection search.

(b) Random search.

Fig. 9: Validation tests on coordinate (4,8). 960 experiments per Random and Bisection search.

On a selected subset of coordinates, we evaluated the importance of the threshold ϵ in equation 2. We expected that the lower the threshold, the closer we get to the roots of the E function in equation 1, and potentially the higher

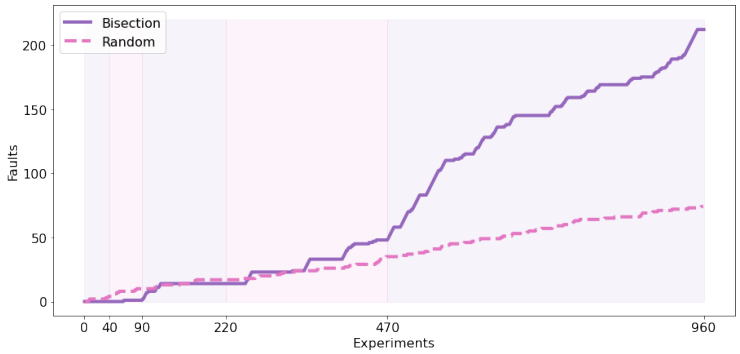


Fig. 10: Comparison between bisection and random search upon a fixed coordinate. Number of total Faults per experiments performed. Upon experiments intervals [0,40],[40,90],[90,220],[220,470] and [470,960] execute iterations from 0 to 4.

the probability of a fault. Figure 11 shows a perceived almost linear relationship between the two.

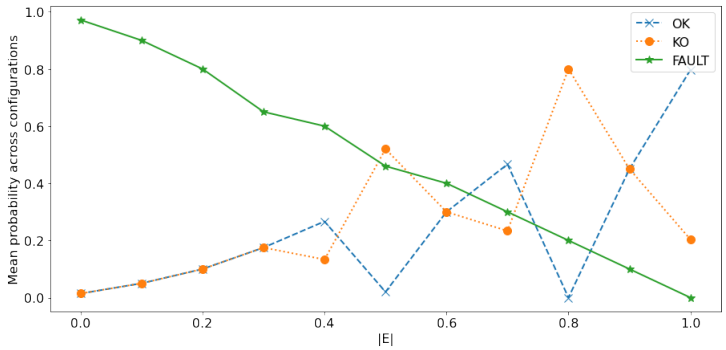


Fig. 11: Fault probability relationship with achieved $|E|$

4.4 Testing a fault model

The previous methodology allowed us to identify some potential coordinates of interest to be further investigated. Although the following is outside the scope of the methodology, we report some results of this additional investigation. In particular, we focus on the `ADD` victim code. Recall that the `ADD` victim code is composed of an unrolled loop of `ADD` instructions that increment the R0 register by a deterministic amount. Inspecting some of the sensible coordinates, we found

that the final value of the R0 register was off by a small margin relative to the expected value, indicating a potential *instruction skip*. These coordinates are characterized by a low $P_{\texttt{FAULT}}$ (thus potentially discarded by other approaches); one of them in particular shows 166 total `FAULT`s, of which 154 are noninformative, 5 reflect the skip of two instructions, and 7 the skip of a single one. We tried, in the same coordinates, a different snippet (`SUB`) and we obtained a similar behavior.

It is well known that instruction skips, when applied to branch instructions, might be the most dangerous exploitable effect. After all, in principle, you could skip complete security checks by skipping a branch. We thus tried a snippet consisting of a branch jumping on itself; after 59 experiments using random values over the (V, d) domain, we have obtained the result that the loop was effectively broken (for 455 V and 200ns of duration). We were able to reproduce this fault with a probability of 2.2%. Even if these results might seem promising, we must underline that it is extremely difficult to target a single branch instruction in a realistic setting (i.e. one that does not jump to itself all the time).

What is thus the effectiveness of the equipment in targeting a single instruction on a 600 MHZ processor? To answer this question, we relied on a particular victim snippet of a single `ADD` and several `NOP`s surrounding it. We then observed the address reported by the `LRABT` exceptions that we have induced by varying the timing offset of the pulse (see Figure 12). Some experiments allowed us to determine (by linear regression on the reported addresses) what was the most likely offset to skip the victim `ADD`. However, even concentrating on that offset, we have found that on average we were producing exceptions both before and after `ADD` and never `ADD` itself. Our conclusion is that the current equipment does not provide adequate accuracy when targeting a single instruction.

5 Conclusion and future work

In this work, we presented a general methodology to identify possible EMFI attack coordinates in a large parameter space. The methodology does not discard any point that could produce a fault (i.e., it has high coverage) and has been proven to reduce the search space in a specific use case by five times. In particular, we were able to produce faults with an average probability of 26.7% in all susceptible coordinates, some coordinates reaching 97.6%. The proposed bisection method has found a number of faults that is 3 times higher than a random search on selected coordinates and corroborates our idea that fault points lie at the equilibrium between `OK` and `KO` points. As future steps, the proposed methodology needs to be further investigated to characterize the range of its applicability, especially on simpler and slower targets and on systems equipped with an EMFI mitigation, and to account for probe height variability.

References

1. Bachrathy, D., Stépán, G.: Bisection method in higher dimensions and the efficiency number. Periodica polytechnica. Mechanical engineering **56**, 81–86 (01

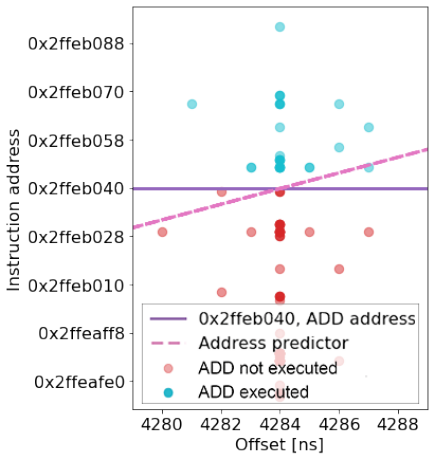


Fig. 12: Varying the timing offset allows to target a range of instruction addresses. The predicted address for the pulse offset 4284ns was the victim `ADD` but we were not able to make it skip, almost all experiments impacting either before or after it.

2012). https://doi.org/10.3311/pp.me.2012-2.01
2. Carpi, R.B., Picek, S., Batina, L., Menarini, F., Jakobovic, D., Golub, M.: Glitch it if you can: parameter search strategies for successful fault injection. In: International Conference on Smart Card Research and Advanced Applications. pp. 236–252. Springer (2013)
3. Cui, A., Housley, R.: BADFET: Defeating modern secure boot using Second-Order pulsed electromagnetic fault injection. In: 11th USENIX Workshop on Offensive Technologies (WOOT 17). USENIX Association, Vancouver, BC (Aug 2017), https://www.usenix.org/conference/woot17/workshop-program/presentation/cui
4. Dumont, M., Lisart, M., Maurine, P.: Modeling and simulating electromagnetic fault injection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **40**(4), 680–693 (2021). https://doi.org/10.1109/TCAD.2020.3003287
5. Dureuil, L., Potet, M.L., Choudens, P.d., Dumas, C., Clédière, J.: From code review to fault injection attacks: Filling the gap using fault model inference. In: International conference on smart card research and advanced applications. pp. 107–124. Springer (2015)
6. Dutertre, J.M., Menu, A., Potin, O., Rigaud, J.B., Danger, J.L.: Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures. Microelectronics Reliability **121**, 114133 (2021). https://doi.org/https://doi.org/10.1016/j.microrel.2021.114133, https://www.sciencedirect.com/science/article/pii/S0026271421000998
7. G., W., A., L.: Raiden github repository. https://github.com/IBM/raiden (2020)
8. Gaine, C., Aboulkassimi, D., Pontié, S., Nikolovski, J.P., Dutertre, J.M.: Electromagnetic fault injection as a new forensic approach for socs. In: 2020 IEEE International Workshop on Information Forensics and Security (WIFS). pp. 1–6 (2020). https://doi.org/10.1109/WIFS49906.2020.9360902

9. Gaine, C., Nikolovski, J.P., Aboulkassimi, D., Dutertre, J.M.: New probe design for hardware characterization by electromagnetic fault injection. In: 2022 International Symposium on Electromagnetic Compatibility – EMC Europe. pp. 299–304 (2022). https://doi.org/10.1109/EMCEurope51680.2022.9901104
10. Hummel, T.: Exploring effects of electromagnetic fault injection on a 32-bit high speed embedded device microprocessor. Master's thesis, University of Twente (2014)
11. Kühnapfel, N., Buhren, R., Jacob, H.N., Krachenfels, T., Werling, C., Seifert, J.P.: Em-fault it yourself: Building a replicable emfi setup for desktop and server hardware. arXiv preprint arXiv:2209.09835 (2022)
12. Machiry, A., Gustafson, E., Spensky, C., Salls, C., Stephens, N., Wang, R., Bianchi, A., Choe, Y.R., Kruegel, C., Vigna, G.: Boomerang: Exploiting the semantic gap in trusted execution environments. In: NDSS (2017)
13. Madau, M.: A methodology to localise EMFI areas on Microcontrollers. Theses, Université Montpellier (Nov 2019), https://tel.archives-ouvertes.fr/tel-02478873
14. Maldini, A., Samwel, N., Picek, S., Batina, L.: Optimizing electromagnetic fault injection with genetic algorithms. In: Automated Methods in Cryptographic Fault Analysis, pp. 281–300. Springer (2019)
15. Menu, A., Bhasin, S., Dutertre, J.M., Rigaud, J.B., Danger, J.L.: Precise spatio-temporal electromagnetic fault injections on data transfers. In: 2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). pp. 1–8. IEEE (2019)
16. Moro, N., Dehbaoui, A., Heydemann, K., Robisson, B., Encrenaz, E.: Electromagnetic fault injection: Towards a fault model on a 32-bit microcontroller. In: 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 77–88 (2013). https://doi.org/10.1109/FDTC.2013.9
17. NewAE: Chipshouter github repository. https://github.com/newaetech/ChipSHOUTER (2019)
18. Omarouayache, R., Raoult, J., Jarrix, S., Chusseau, L., Maurine, P.: Magnetic Microprobe design for EM fault attack. In: EMC EUROPE: Electromagnetic Compatibility. EMC EUROPE, Bruges, Belgium (Sep 2013), https://hal.archives-ouvertes.fr/hal-01893856
19. Ordas, S., Guillaume-Sage, L., Maurine, P.: Electromagnetic fault injection : the curse of flip-flops. Journal of Cryptographic Engineering **7** (09 2017). https://doi.org/10.1007/s13389-016-0128-3
20. Proy, J., Heydemann, K., Berzati, A., Majéric, F., Cohen, A.: 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. pp. 1–10 (2019)
21. Raelize: Qualcomm ipq40xx: Breaking into qsee using fault injection. https://raelize.com/blog/qualcomm-ipq40xx-breaking-into-qsee-using-fault-injection (2021)
22. Tang, A., Sethumadhavan, S., Stolfo, S.: {CLKSCREW}: Exposing the perils of {Security-Oblivious} energy management. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1057–1074 (2017)
23. Trouchkine, T., Bouffard, G., Clédière, J.: Fault injection characterization on modern cpus. In: IFIP International Conference on Information Security Theory and Practice. pp. 123–138. Springer (2019)

Funded by the European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Funded by
the European Union