Electromagnetic Fault Injection as a New Forensic Approach for SoCs

Clément Gaine, Driss Aboulkassimi, Simon Pontie, Jean-Pierre Nikolovski, Jean-Max Dutertre

Electromagnetic Fault Injection as a New Forensic Approach for SoCs

Clément Gaine*, Driss Aboulkassimi*, Simon Pontié*, Jean-Pierre Nikolovski*, Jean-Max Dutertre†
*CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne, F-13541 Gardanne FRANCE
*Univ. Grenoble Alpes, CEA, LETI, MINATEC Campus, F-38054 Grenoble, FRANCE
†Mines Saint-Etienne, CEA-Tech, Centre CMP, Departement SAS, F - 13541 Gardanne France
Email: *firstname.lastname@cea.fr, dutertre@emse.fr

***Abstract*—Smartphones have a complex hardware and software architecture. Having access to their full memory space can help solve judicial investigations. We propose a new privilege escalation technique in order to access hidden contents and execute sensitive operations. While classical forensic tools mostly exploit software vulnerabilities, it is based on a hardware security evaluation technique. Electromagnetic fault injection is such a technique usually used for microcontrollers or FPGA security characterization. A security function running at 1.2 GHz on a 64-bit SoC with a Linux-based OS was successfully attacked. The Linux authentication module uses this function to verify the password correctness by comparing two hash values. Hence, this work constitutes a step towards smartphones privilege escalation through electromagnetic fault injection. This approach is interesting for addressing forensic issues on smartphones.**

***Index Terms*—Electromagnetic Fault Injection; SoC; Forensic; Smartphones;**

I. Introduction

The security of sensitive data stored in integrated circuits (IC) can be altered by passive hardware attacks such as side-channel (e.g. the electromagnetic (EM) side-channel [1]) or active attacks (e.g. physical tampering or fault injection). Various techniques are used to stress the attacked devices, such as laser [2], voltage glitch [3], electromagnetic perturbations [4]–[11] or even the software-induced CLKSCREW attack [12]. Faults Injections intentionally disrupt the operations of a system and obtain behaviors not anticipated by designers. This can be data flow or control flow modifications. The exploitation is obtained by inverting the result of a conditional test, by skipping a branch instruction or exiting a loop prematurely (as discussed in Section V).

In this paper, we focus on Electromagnetic Fault Injection (EMFI) by applying a transient EM pulse on the device under test. A voltage pulse is transmitted to a fault injection probe, an electromagnetic coil, to generate this field. It targets the physical implementation of a microprocessor to corrupt its control flow. This may allow a privilege escalation or an information leakage. In the case of smartphones, their exploitation can reveal a large amount of personal data.

EMFI is routinely used for security characterization of microcontrollers: circuits that have a simple architecture and run at low frequency. Smartphones use complex hardware and software architectures. They also run at higher frequency (1 GHz or above). This explains why there was to date no report of successful data extraction from a smartphone through an EMFI technique. EMFI may also be used to complement conventional forensic tools (e.g. using fault injection to recover boot-ROM dumps in order to identify weaknesses that can be exploited in a standard methods).

In our research, we studied a smartphone-grade 64-bit System on a Chip (SoC) running a Linux OS. Our intent was to ascertain the ability to use privilege escalation through EMFI for forensic purposes. Interesting effects created on the target by the EMFI process were identified during preliminary tests (i.e. the fault model).

This article is organized as follows. Section II reviews the state-of-the-art of EMFI and introduces the aim of our work. Our experimental setup and settings are described in section III. Section IV reports on our first experiments on our SoC target. Then, an attack path is identified and validated in section V. In section VI we discuss the use of the technique in a forensic context. Section VII concludes the paper.

II. Related and previous work

A. State of the art of EMFI attacks

To the best of our knowledge the first time that physical attacks have been introduced as a potential forensic application was by Aboulkassimi et al. [1] in which the authors showed how to use side channel analysis for recovering data from a mobile device. We extend this approach to active hardware security techniques. EMFI was mainly used against microcontroller or FPGA targets running at low frequencies, as in [4]. This has been the subject of numerous publications, and studies about fault models (i.e. the characteristics of the induced faults). Zussa et al. [5] proposed that faults are produced by timing violations. This assumption was reinforced with Schaumont et al. [6] and their study at different levels: wire-level, chip-network level and gate-level. Most of injections target the CPU, through some (as Menu et al. [13]) rather faulted data transfers from the Flash memory to the data buffer of a cortex-M microcontroller.

So far little work has been done about SoC, due to their complexity: advanced hardware technology, numerous software layers with an OS, high operating frequencies. However,

WIFS'2020, December, 6-11, 2020, New York, USA.
978-1-7281-9930-6/20/$31.00

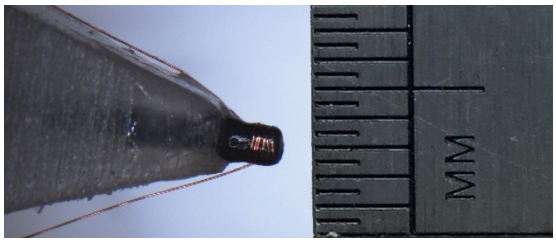

Fig. 1. Probe designed for EM Injection on a SoC target

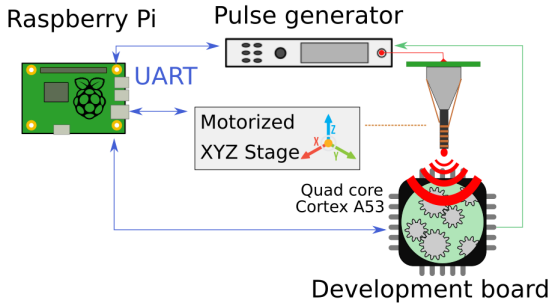

Fig. 2. EM pulse injection setup

in the past few months, new results on SoCs have begun to emerge in the field of hardware security. Yet, EMFI has not been proven for forensic purposes in this context.

Regarding EMFI on SoCs, Majeric [7] [8] successfully conducted a Differential Fault Analysis (DFA) to extract the secret key of a software Advanced Encryption Standard (AES) running on a A9-Cortex (32-bit ARM core). On the same device, Proy et al. [9] caused software loop exits.

Majeric et al. [7] also studied a A7-Cortex (32-bit ARM core). They were able to bypass a secure boot with the repeatability of 0.78%. They also tried to realize EMFI attacks on a software running in a Trusted Execution Environment (TEE) on a A53-Cortex (64-bit ARM core).

Recently, Trouchkine et al. [10] worked on a A54-Cortex (a 64-bit ARM core). Studying loops, they showed that the EMFI effects were different in bare metal mode w.r.t. using an OS. The difference is related to how the cache memories are managed. In the case of a bare metal target, they were able to attribute the faults to the cache memories (L1 and L2) and to the Memory Management Unit. Ait el Mehdi [11] tried to bypass the lock screen of a 64-bit ARM target running Android 8 with EMFI. He proposed a theoretical attack scheme which was demonstrated in simulation, but its practical implementation failed.

Other fault injection methods have also been used with SoCs. A laser attack on Android smartphone targeting the secure boot sequence was made by Vasselle et al. [2]. Timmers et al. [3] showed that Voltage Fault Injection allows a full control of a Linux OS without any logical vulnerability.

A new class of fault attack that can be exploited on SoCs is the CLKSCREW attack [12]. Controlling the Dynamic and Voltage Frequency Scaling system (DVFS) allows to modify the power voltage and clock frequency by software means for the purpose of conducting an attack.
Instruction skips are predominantly the effects of fault injections [3], [4], [9], however Proy extended the classification with 3 other observable effects at Instruction Set Architecture (ISA) level: register most-significant half-word reset, register corruption or source operand substitution.
Among of all the publications presented, there are two objectives: (1) extracting secrets [1], [8], [12] , or (2) bypassing security features [2], [3], [11], [12]. However, of all these publications, only one was in a forensic context [1]. This latter work is based on passive method, we chose here to conduct our research using an active method: EMFI.

B. Contributions, EMFI in the forensic field

In this paper, we propose a new technique using physical attacks as a new approach for EMFI based forensic application. We focus on an A53-Cortex core. It is a 64-bit multi-core SoC used in many smartphones, mainly at entry and mid-range price levels. We propose a fault model at the ISA level. We also propose an attack scenario on the same target running an OS. [3] has shown that fault injection can exploit various attack paths. We limited ourselves to one path, but with another method: EMFI. This technique could be used on a critical command, allowing Super User privilege escalation on Linux (`su`). This is the first study on a 64-bit chip with a complete software architecture, usable in a smartphone. This can be used for forensic purposes to access protected or hidden data or to execute code.

III. Experimental setup

A. Targeted SoC

The targeted SoC, based on 4 ARM Cortex cores A53 with 32-bit and 64-bit support, is part of a development board for mobile platforms. Its operating frequency can be set between 800 MHz and 1.2 GHz. An external 1 GB RAM is soldered on the board. So, the chip is directly accessible via EM injection. Removing a part of the plastic package of the chip or modifying the board was not necessary to perform EMFI

The target was flashed with a Linux-based system: the Sumo release of the Yocto Project with a Linux kernel 4.14. Yocto allows having a custom Linux distribution, however we haven't made any major system change.

B. EMFI platform and injector

The source of the EM perturbation is a voltage pulse delivered to an injection probe. This probe is a coil for creating a magnetic field, which is guided by a ferrite rod. For this experiment, the pulse generator can provide a voltage pulse amplitude of up to 400 V with a rise time of 2 ns in nominal operation. The EM injection probe is shown in Figure 1. A thin ferrite rod of 750 µm in diameter was chosen together with a thin enameled copper wire, featuring 5 turns. Its characteristics improve the locality of the induced perturbations. This is necessary for EM injection on small advanced technological nodes used in SoC. Due to the locality of the EM perturbation radiated, the probe must be placed close to the chip. The probe position is finely controlled by a XYZ motorized stage.

A Unit Control (Raspberry Pi Board) completes the electromagnetic fault injection platform, presented in Figure 2. It communicates with the board using 2 UARTs: one for the control console and the other for the data. A signal trigger is produced through one GPIO of the SoC to trigger the voltage pulse generator. A controlled delay may be added between the reception of the trigger signal and the actual pulse generation.

IV. Identification of a physical vulnerability

Inducing faults into a SoC is more difficult than into a microcontroller due to many factors. This is due first, to a small technological node. Thus we have to make a very precise location scan using an EM probe with a small diameter. Moreover, the target size (55 mm^2 for the targeted SoC) is larger than that of general-purpose microcontrollers (about ten mm^2). This enlarges the area to be explored. Due to the higher operating speed, the cycle time is also shorter, which implies increasing the time resolution and accuracy of the scans. Besides, there are many uncontrollable desynchronization sources. Finally, this requires a higher EM field and therefore more powerful equipments. On another aspect, the multiple reboots and the loading time of the OS on a SoC is long, which lengthens the time needed to perform the data acquisition campaigns.

The first part of our methodology is to learn how to inject faults (ie identify a physical vulnerability). A simple test code is used to find the time and space injection parameters and deduce the fault model.

A. Identifying when to fire

The aimed target is complex, for the sake of simplicity we chose to force the CPU frequency at 1.2 GHz. Then we always used the same configuration, in order to improve repeatability.

We use the Code Under Test (CUT) given in Listing 1, to obtain faults by EMFI. To alleviate the synchronization constraints, this test code was chosen long enough (300 ns) to be easily faultable. It consists in a series of subtractions by one from an initial value passed successively to ten registers and iterated 32 times. We use no loop in the test code in order to highlight the fault effect on the registers rather than alterations of the loop control flow, which are harder to interpret.

```
//Initialisation 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
```

Listing 1. Code working with registers

The registers we are manipulating, x_{19} to x_{28}, are 64-bit registers that are dumped in memory at the end and readback. When no fault is injected, the expected result corresponds to a decreasing series of hexadecimal numbers (from 39 to 30), as shown in the reference A line of Table I.

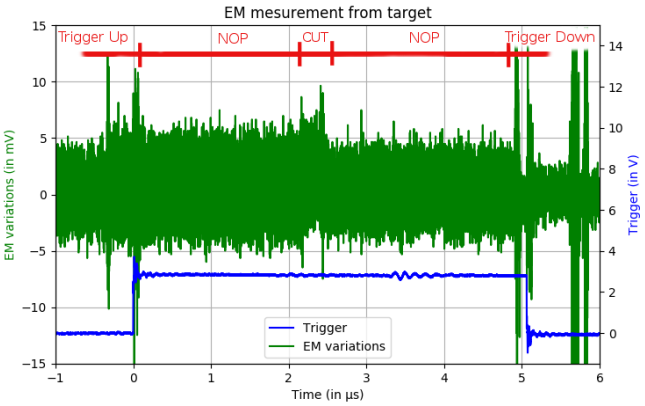

Fig. 3. Delay identification using SEMA, without EM fire

A Simple ElectroMagnetic Analysis (SEMA) was used to identify the relevant timing of the EMFI. It consists in recording an image of the computational activity of the target through its EM emanations, which are sampled using a passive EM probe connected to an oscilloscope through a low noise amplifier. While listening, it is not possible to perform EM shots to avoid damage to the acquisition system. In figure 3, we can see, in green color, the EM trace captured via a Langer probe and its amplifier positioned near the processor. The trigger signal is colored in blue. It is raised, then no operation instructions (NOP) are executed, followed by the test code and other NOPs and finally the trigger is lowered. NOPs allow us to isolate the CUT, which also makes it easier to view in SEMA. We can observe the EM emissions produced by the CUT 2.1 µs after the trig up with a 300 ns duration.

Delay and jitter are introduced by the code and characteristics of the equipment. The minimum delay between the GPIO raising instruction and the shot is about 420 ns and the jitter is 50 ns. The use of a trigger signal and the SEMA analysis of the target activity allowed us to synchronize the EM perturbation with the CUT. As a result, any timing uncertainty is suppressed. The next step was to find a spatial location of the injection probe that makes it possible to inject faults into the target.

B. Identifying where to fire

It turns out that only one Central Processing Units (CPU) out of the four embedded in the targeted SoC was sensitive to EMFI. The other three CPUs couldn't be faulted. In order to keep the parameters identical throughout the campaign, it was chosen to run the test programs only on this CPU. Injecting a fault without controlling the CPUs is nevertheless possible, but it lengthens the duration of the campaigns. Similar results were obtained with different chips of the same reference.

The first tests were also done with a large diameter probe, 1.5 mm of ferrite's diameter, to save time by identifying the sensitivity of a larger area. To test the sensitivity of a larger area, tests should be done with the maximum available voltage to highlight sensitive areas. The area is considered sensitive when freezing or rebooting the SoC. Once a sensitive area is found, we reduce the probe diameter and the voltage to locate the most sensitive point, and look for some exploitable faults.

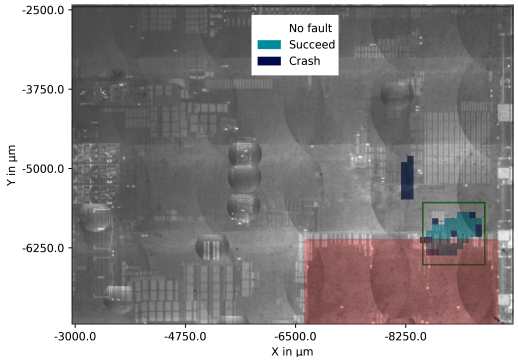

Fig. 4. Localization of EMFI-sensitive areas on an IR image of the target

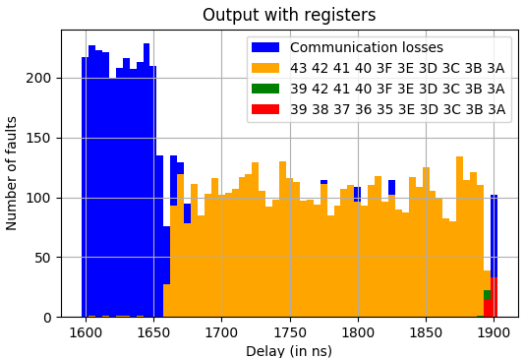

Fig. 5. Number of faults by function type versus pulse delay

C. Results

By iterating, the best configuration (position, timing, stress intensity) could be found. Figure 4 displays the effects obtained due to EMFI overlaid to a picture of the target silicon die. These effects are classified as: *no fault* when the target behavior is undisturbed, *succeed* when the registers return is corrupted, and *crash* when the target initiates an autoreboot or freezes (i.e. communication is lost). One result of this campaign is presented in Figure 4. The categorization of data has been simplified and is as follows: undisturbed results (no fault), modifications of registers (succeed) and crashes including auto-reboots and freeze. Auto-reboots correspond mainly to a report of an illegal modification of the Program Counter, which is corrected by reboot. We consider a freeze when the card does not respond to any request on UARTs.

The green box that contains the region with the vulnerability measures $1\,mm$ by $1\,mm$. The interesting area, with fault on when manipulating the registers, is lower than $0.4\,mm^2$. It's located in an area that appears to be close to the CPU represented in red. The values returned from the registers are given in Table I.

Ref.	Occurrences	Result (x19, ..., x28)	Proportion
A	27287	39,38,37,36,35,34,33,32,31,30	71.0%
B	5314	Communication lost	13.8%
C	4899	43,42,41,40,3F,3E,3D,3C,3B,3A	12.7%
D	48	39,38,37,36,35,3E,3D,3C,3B,3A	0.1%
E	28	39,42,41,40,3F,3E,3D,3C,3B,3A	0.1%
	...	...	...

TABLE I
RESULT OBTAINED WITH EMFI ON THE CODE IN LISTING 1

During a campaign, pulse delays and positions are varied, thus maximizing the types of faults. Approximately 30% of faulty results are obtained. The majority are communication losses (14%, reference B), due to a restart of the board or a closing of the application. This mostly happens in the first 50 ns of code execution. This can be seen in figure 5 which depicts the obtained faults as a function of the delay between the trigger signal and the actual EMFI.

Reference C corresponds to EMFI induced between the 1^{st} and 310^{th} `sub` instructions (13% of the results). We analyze it as an instruction skip (i.e. the fact that one instruction is not executed). It breaks a chain of 10 successive subtractions made on the ten registers. As a result, the returned values are higher than expected (an increase by `0x0A`). Note that, we cannot conclude on the exact number of skipped instructions because any value between 1 and 9 would produce the same faulted result. This analysis is further strengthened by reading the time repartition of these faults in Fig. 5 (i.e. from 50 ns to 300 ns of the faults window).

Reference D and E correspond to fault induced during the 10 lasts `sub`. Their frequency rate are around 0.1%. The first part of the results correspond to the expected values, but the following ones are incremented by 10 (highlighted in red). This means that the execution of one instruction have been skipped, and we observe the values of the previous round. By considering the case of the fault injection mechanism reported in reference E, skipping the `sub x20,x19,#1` instruction left x20 content unchanged at 42. This faulted value breaks the chain of subtractions: the subsequent `sub` instructions also produce faulted values (with an increment of 10 w.r.t the correct values). Note that skipping several successive instructions produces the same result. These results therefore occur when the shooting takes place during the last instructions. The timing analysis in Figure 5 confirms it. These results show that we can modify the behavior of the SoC. The effect obtained is that of an instruction skip. The code used does not make it possible to determine the number of instruction skips made (between 1 and 10). However, further testing showed that only one instruction is skipped. The rest of the methodology is devoted to change the behavior of a security function.

V. PRIVILEGE ESCALATION EXPLOITING THE PHYSICAL VULNERABILITY

A. Analyze the code to identify an attack path

The results obtained with a test code are now applied to a more complex code used by a security application. EM-induced instruction skip can be exploited. By choosing the instruction to skip, as an instruction branch, we should successfully modify the control flow. Indeed, in the case of a loop, if we skip an instruction branch, we should exit it prematurely. So we looked for these instructions in a software code.

It was chosen to try to achieve an escalation of privilege in order to have a full access to the system. The command chosen was the `su` command of Linux. This Substitute User command is used to acquire privileges of another user account. `su` uses the `libpam` library (Pluggable Authentication Modules LIBrary) to authenticate a user before providing new privileges.

libpam asks the user to provide an administrator password, once verified and validated, it opens an administrator console with full permission. Obtaining administrator rights with a wrong password is considered as a successful attack.

A first attack path is to modify the control flow of the authentication function. A jump of the branch instruction that check the password allows to continue the elevating procedure even if the password is wrong. However a protection introduces a jitter of 1 second if the password is wrong. We don't know the password, so we'll have a random delay of about a second introduced before the targeted branch. The synchronization must be accurate to the nanosecond, so this makes it impossible in practice to attack. We therefore looked for another instruction before this random delay.

The validity of the entered password is not assessed by comparing it directly to the stored password. At first it is hashed (using a hash function) and then compared to the hash of the stored password. The strcmp function from the LIBC library is used to compare the two hash strings. This chain has a length of 106 bytes and consists of the hash id, the seed and the hash separated by $. Two examples are shown below.

- 6wWxFc—tJdeOI05—KNO$lAAh—w8Th...
- 6wWxFc—tJdeOI05—KNO$Uung—4U7s...

Comparison of the 2 hashes (ASCII representation)

The strcmp function compares 8-char blocks from left to right. If it encounters two different blocks, the check is stopped and the two strings are reported as different. Regardless of the password entered, the first two 64-bit words are identical because they correspond to the prefix of the hash function. A premature loop exit during the 1st or 2nd word comparison would cause all passwords to be interpreted as valid passwords. Nevertheless, an injection at the 3rd comparison will also cause a loop exit, but the password will be considered as wrong and won't allow the privilege escalation.

In order to validate this analysis, the behavior in the event of a loop exit was tested. An extract of the strcmp function is shown, in listing 2. This code loads two words, performs the comparison, checks if there is a difference (or for the end of the chain), then continues the comparison or stops. The Compare and Branch on Zero instruction (cbz), highlighted in red, is in charge of branching to the beginning of the function to perform the next block comparison, or to end the comparison.

A first validation was conducted on simulation basis, we used the gdb debugger tool to run the strcmp function on our target and emulate the effect of skipping the cbz instruction. An instruction skip on the 1st or 2nd cbz instruction exit the loop without detecting any password difference. It validates the proof of concept of this vulnerability. The control flow could theoretically be modified, the next step was to assess the practical feasibility of this vulnerability.

B. Preparation to facilitate the characterization of strcmp

With the previous simulation results, the exploitation of this vulnerability seems attractive. The rest of the methodology is to prove the vulnerability of strcmp in an experimental way.

```
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)
```

Listing 2. Code of strcmp

We decided to start by proving the concept only on a strcmp comparison code identical to that used by su.

For the best understanding of fault injection mechanism to obtain privilege escalation, the adopted approach consist in starting by strcmp which is used in su, before attacking su itself. The idea was to ascertain experimentally the existence of a vulnerability on a simpler case. A code calls it and passes two different character strings in argument. These strings are entered directly into the code, which, unlike the su code, doesn't make calls to the flash memory. This allows a less important jitter than in the real case. We could directly monitor the output of strcmp, which is not possible with the su code, in order to check if there has been a modification of the control flow. The trigger is set just after the reception of the password.

We managed to get a success rate of 2%, which corresponds to a success every 2 minutes, for a code running always on the same CPU with a fixed probe position at a fixed frequency.

C. Results and Exploitation

The success obtained in the previous part encouraged us to demonstrating the proof of concept of a more complex attack. As we already observed that only one core of the target is vulnerable to EMFI, we use a program to launch the su process on this very specific CPU. It has been chosen to always send the same incorrect password. A trigger was added in a modified libpam. This trigger is managed by the module libpam_misc and the file misc_conv.c. This trigger is set up after the password reception and set down after the call of the strcmp function. The area of interest, bordered by the trigger, is 200 ns.

The test was successful when the operating frequency was set at 1.2 GHz, but it was also successful with the Dynamic Voltage and Frequency Scaling activated. Once the settings have been made, we obtained 21 success for 6,000 tests. This is equivalent to obtaining a success every 300 shots, which corresponds to a successful authentication every 15 minutes.

We can authenticate quite regularly, but the board restarts as soon as the authentication is successful. In order to circumvent this limitation and in the scenario of an attack, one can add to the call of su the parameters "chmod + s /bin/bash" allowing to make persistent the root access to the terminal.

The command had time to be executed before the restart. Thus, we can set the SETUID to allow a permanent escalation of privilege. This is observed by the flag "s" instead of "x" for all users in the Unix permissions. These results show the effectiveness of EM injection attacks on a SoC for the purpose of inducing privilege escalation.

VI. Towards a forensic use

A. Discussion of results

The first compromise we made on a real scenario was to move most processus to different CPU to keep only one CPU for the code under test. Regarding this point, tests have shown that this only influences the success time rate and, by multiplying the campaign time by the number of CPUs we can find equivalent results. This has been implemented, and, in order to have a uniform distribution of the code under test on all the CPUs, a program soliciting the processor is used. This allows a frequent change of the CPU where the code to be faulted runs, and therefore in our case, decreases only by a factor of 4 the success rate. Forcing the code to run on a CPU therefore has the effect of speeding up the manipulation.

The study was done on a SoC, which can also be used with Android OS, but the attack should be adapted.

Using a trigger signal in order to have an ideal synchronization was particularly interesting to understand the fault model. For a forensic use, this technique cannot be used because it assumes root access to the SoC in order to authorize the use of GPIOs. In a more realistic case, we would look for another way to synchronize as explained hereafter.

B. Synchronize EMFI without a GPIO

In the case of forensic use on a real mobile device, we cannot have all the accesses we have on a development board, so it is wise to pose the problem of EMFI synchronization without using a trigger from a GPIO.
The first idea when we want to perform a simple synchronization would be to use a fake usb keyboard to enter the password and get a good synchronization. Before carrying out such an analysis, the jitter due to the code was estimated. A first analysis showed a standard deviation of approximately 5 ms between the reception of the password by the `su` and the call of the `strcmp` function. By keeping the optimal parameters of the injection bench and according to our estimates, several weeks experimentation should be conducted to generate one successful fault injection. This could be achieved, but the lapse time of attack strongly encourages the development of tools to improve synchronization. A solution could be to exploit the EM fields emitted by the processor in order to generate a trigger signal.

C. Use Cases for Forensic

One of the problematic points in the forensic is to bypass secure boot or root access. Commercial forensic solutions work using specific software flaws. EMFI is an alternative because it targets the hardware. Our results show that EMFI, similarly to software attacks, can be used to gain access to the data of a smartphone, execute or plan the execution of a code.

EMFI-based hardware forensic may also be used as a first step in order to increase the potentiality of the usual forensic software tools. Indeed, EMFI could provide new contents to be analyzed. We can imagine using EMFI to unlock security in order to dump a boot-ROM. Then, the extracted code can be analyzed, in order to identify a software flaw and to be exploited by using conventional tools. Thus, the joint use of these two techniques can push forward the limits of forensic tools.

VII. Conclusion and future works

This paper shows that it is possible to apply successfully EM fault injection on a smartphone-grade SoC despite the complexity of the hardware and software architectures of the target. The reported exploitation case consists in bypassing the root privilege protection of the target. The proof of concept has been demonstrated, even if the real scenario would require a synchronization tool. As the objective of this article is to present a new approach for forensic, we also suggested some use cases. Future works will focus on bypassing a secure boot on a Smartphone, using the EMFI approach.

References

[1] D. Aboulkassimi, M. Agoyan, L. Freund, J. Fournier, B. Robisson, and A. Tria, "ElectroMagnetic analysis (EMA) of software AES on Java mobile phones," in 2011 IEEE International Workshop on Information Forensics and Security, 2011.

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

[3] N. Timmers and C. Mune, "Escalating Privileges in Linux Using Voltage Fault Injection," Fault Diagnosis and Tolerance in Cryptography, FDTC, 2017.

[4] A. Dehbaoui, J.-M. Dutertre, B. Robisson, and A. Tria, "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES," 2012.

[5] L. Zussa, J.-M. Dutertre, J. Clédière, B. Robisson, and A. Tria, "Investigation of timing constraints violation as a fault injection means," 27th Conference on Design of Circuits and Integrated Systems (DCIS), 2012.

[6] P. Schaumont, M. Ghodrati, B. Yuce, S. Gujar, C. Deshpande, and L. Nazhandali, "Inducing local timing fault through EM injection," DAC, 2018.

[7] F. Majéric, "Etude d'attaques matérielles et combinées sur les 'System-on-Chip'," Ph.D. dissertation, 2018.

[8] F. Majéric, E. Bourbao, and L. Bossuet, "Electromagnetic security tests for SoC," IEEE International Conference on Electronics, Circuits and Systems, ICECS 2016, 2016.

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

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

[11] N. Ait el Mehdi, "Analyzing the Resilience of Modern Smartphones Against Fault Injection Attacks," Ph.D. dissertation, DEFL, 2019.

[12] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," USENIX, 2017.

[13] A. Menu, S. Bhasin, J. M. Dutertre, J. B. Rigaud, and J. L. Danger, "Precise spatio-temporal electromagnetic fault injections on data transfers," Fault Diagnosis and Tolerance in Cryptography, FDTC, 2019.