Exploration of system-on-chip secure-boot vulnerability to fault-injection by side-channel analysis

Clement Fanjas, Simon Pontie, Driss Aboulkassimi, Jessy Clediere

Exploration of System-on-Chip Secure-Boot Vulnerability to Fault-Injection by Side-Channel Analysis

Clément Fanjas
CEA-Tech, Centre CMP,
Équipe Commune CEA Tech - Mines Saint-Étienne,
F-13541 Gardanne, France
Université Grenoble Alpes, CEA, Leti,
F-38000 Grenoble, France
clement.fanjas@cea.fr

Driss Aboulkassimi
CEA-Tech, Centre CMP,
Équipe Commune CEA Tech - Mines Saint-Étienne,
F-13541 Gardanne, France
Université Grenoble Alpes, CEA, Leti,
F-38000 Grenoble, France
driss.aboulkassimi@cea.fr

Simon Pontié
CEA-Tech, Centre CMP,
Équipe Commune CEA Tech - Mines Saint-Étienne,
F-13541 Gardanne, France
Université Grenoble Alpes, CEA, Leti,
F-38000 Grenoble, France
simon.pontie@cea.fr

Jessy Clédière
CEA-Leti, 17 av. Des Martyrs, 38 054 Grenoble, France
jessy.clediere@cea.fr

***Abstract*—Fault-Injection might be a useful tool to bypass security features that may obstruct the work of forensic experts. For instance, injecting a fault could modify the target control-flow and compromise its security. When the attacker knowledge about the target software implementation and hardware architecture is limited, discovering a Fault-Injection vulnerability becomes a serious challenge. Another issue is identifying when the targeted vulnerability is executed. To the best of our knowledge, this paper proposes a new methodology to solve these problems for the first time on System-on-Chip (SoC). The first step is to improve the knowledge of security feature implementations. Then deviations in the control-flow induced by forged inputs can be combined with Side-Channel observations to identify vulnerabilities. The next step is to define a trigger as close as possible in time and prior to these vulnerabilities. At this stage, Electromagnetic Fault-Injection (EMFI) can be put in practice to bypass the targeted security feature. As a proof of concept, we bypassed the Secure-Boot of a smartphone grade SoC. Three theoretical vulnerabilities in the Secure-Boot architecture of our target are identified using this new methodology and successfully exploited by EMFI.**

***Index Terms*—Side-Channel Analysis, Electromagnetic Fault-Injection, System-On-Chip, Secure-Boot, Hardware Attacks**

I. Introduction

Most modern smartphones come with security features such as Secure-Boot. This mechanism ensures the authenticity and integrity of the different programs processed in order to complete the boot and to start the Operating System (OS). This security feature consists in a systematic authentication of the boot stage *n+1* prior to its execution by the boot stage *n* currently executed. It is an interesting feature to study, in particular in a forensic context where it can obstruct the work of forensic experts. Fault-Injection may be a solution to bypass this security feature. Injecting a fault with the right timing may alter the program control-flow and guide it in an advantageous state for the attacker. During the boot phase, programs that are executed earlier are granted higher privileges. Therefore an attacker would want to bypass the first authentication to get the highest privileges.

Having the knowledge of the executed code would be a great asset. However the first program executed is generally unreachable for an attacker, which raises problematics such as:

P1. Finding a vulnerability to Fault-Injection without having access to the code.
P2. Finding the timing when the vulnerability happens.
P3. Identifying the best signal to trigger the Fault-Injection.
P4. Identifying the best Fault-Injection parameters.

This paper presents a methodology to answer P1, P2 and P3. The problematic P4 is not addressed in this paper. The Electromagnetic Fault-Injection (EMFI) parameters are set up prior to the final experiment in a fully controlled environment. Our methodology works as follows:

1) First the target Secure-Boot is analyzed in order to find potential vulnerabilities.
2) Then we forge images that create deviations in the target control-flow according to the vulnerabilities. We assume it is possible to load a customized image on the target.
3) These deviations leak in Side-Channel, creating notice-

able divergences between Electromagnetic (EM) traces from authentic and corrupted images.
4) A trigger signal as close as possible prior to the divergence is identified in order to get a jitter as low as possible.
5) The delay between the trigger signal and the divergence, and its jitter are characterized.
6) Finally, the trigger signal is used with the delay measured to inject a fault with the right timing.

Thanks to this methodology, the first authentication of the Secure-Boot of a smartphone grade System-on-Chip (SoC) has been successfully bypassed for three vulnerabilities.
We will present these results according to the following plan: Section II presents a state of the art on Secure-Boot attacks and on related works. Section III presents the target and its Secure-Boot. Section IV presents the methodology and how we applied it. The results of the experiments are presented in Section V. Finally, Section VI concludes this paper.

II. State of the Art

A. Secure-Boot attacks

Hardware attacks, especially Fault-Injection, have already been used to bypass Secure-Boot [1, 2, 3, 4, 5, 6]. Such attacks have various synchronization requirements. Some attacks do not require timing precision. For instance, [1] presents a scenario in which an attacker can inject a fault during the copy phase of a malicious image composed of a shellcode and pointers. A fault any time during the pointers copy could set the Program Counter value to the pointer value. Another attack from [2] shows that it is possible to bypass the Secure-Boot of a smartphone SoC by using static faults. In those two examples, the need of synchronization is relaxed. The fault can be injected any time in a certain temporal window. In contrast, attacks such as [5, 4, 3, 6] target a vulnerability that happens at a specific timing. A trigger close to this instant will increase the attack success rate by reducing the jitter. It can be a communication signal [5, 3, 6, 7, 8], a reset signal [3] or a Side-Channel based trigger [4]. Also knowing the delay between the trigger and the vulnerability reduces the time-domaine exploration, thus increasing the attack success rate. When this delay is not precisely known, the attacker has to explore a temporal window until a successful fault [3, 6, 7]. An attacker could also look for Side-Channel information on the target behavior to find the vulnerability delay [5, 8, 9].

B. Side-Channel Analysis

Side-Channel Analysis is based on the fact that data manipulated by a target can leak in power consumption or EM emanations. However information about the target behavior may also leak alongside this data, such analysis is called a Simple Power Analysis (SPA) [10]. Other works exploit this fact, for instance finding some information about the target code via a Side-Channel Analysis has already been explored in [11, 12].
Also, some similar works use Side-Channel to monitor the target control-flow in order to detect malicious program [13, 14]. In [9] the authors propose to identify the execution timing of a specific targeted instruction by matching a template waveform with a target waveform using a Sum of Absolute Difference (SAD). This methodology allows to identify all potential timing of the targeted instruction in the targeted trace. Our work lays on the same idea that information about the target control-flow leaks in Side-Channel. [5] proposes a similar approach to roughly measure the vulnerability timing on a microcontroller. However for a much more complex target that runs at high frequency such as a SoC, a more resilient methodology might be useful. In order to find this divergence, we use a methodology close to trace alignment techniques that have already been explored in the literature [15, 16, 17].

III. Target and Secure-Boot

Our target is the same target presented in [4]. It is a smartphone grade SoC on a development board. The SoC is based on 4 cores Cortex A53 up to 1.2GHz. During the earlier phases of the boot, only one core is active, and it is running at 800MHz. The Secure-Boot is natively disabled on our target, but it can be enabled.
A first program named the First Stage Boot Loader (FSBL) is executed from a Read Only Memory (ROM). Then, the Secondary Stage Boot Loader (SSBL) is loaded in SRAM and authenticated by the FSBL before execution. The SSBL again loads and authenticates the next boot stage, etc. This boot flow is represented in Fig. 1. The FSBL is the only program that is not authenticated since the ROM is enclosed in the SoC, it is considered secure by default. The SSBL is the first program loaded that can be modified. A malicious SSBL could be used to get privileges at Exception Level 3 (EL3), giving access to the processor in secure state. This would give the attacker privileges over critical resources such as the TEE. However, the experiment has to be performed in a black-box context since the attacker has no access to the FSBL binary or code. In this paper we propose a methodology to find the right timing to inject a successful fault without having access to the targeted FSBL code.

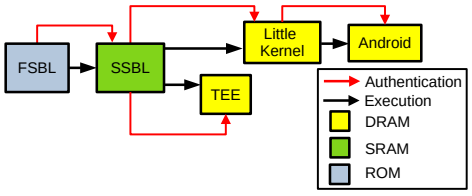

Fig. 1. Secure boot of our target

In order to be authenticated, the SSBL image has to be signed beforehand. The signature step works as follows:
1) A RSA private key K_n is used to sign the image hash.
2) The corresponding public key k_n is saved in the certificate *n*.
3) The certificate *n* is signed with a private key K_{n-1}.
4) The corresponding public key k_{n-1} is saved in the certificate *n-1*.

5) Step 3 and 4 are repeated with the certificate *n-1* and a new key pair until the root certificate.

This process is illustrated in Fig. 2A. The root certificate is a particular case. Its hash should have already been burned into fuses during the procedure to enable the Secure-Boot. In Fig. 1 each authentication involves authenticating both the image and every certificate. An authentication of an image or a certificate is performed according to the following steps:

1) The certificate/image hash is computed.
2) The certificate/image signed hash is verified using the public key saved in the previous certificate.
3) The computed hash is compared with the verified hash. The comparison result is used to continue or stop the boot.
4) Step 1,2 and 3 are repeated until the root certificate.
5) The root certificate computed hash is compared to the value stored in fuses.

Fig. 2B represents the full authentication of an image and its certificate chain. Each certificate or image authentication

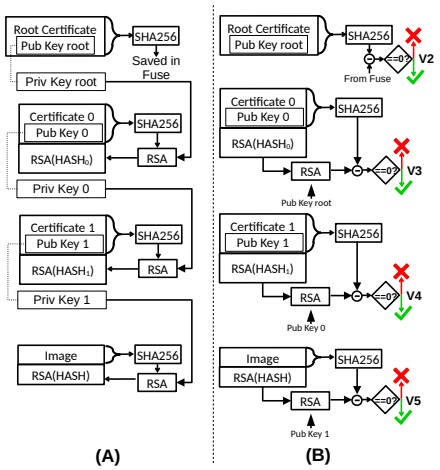

Fig. 2. (A) Image signature performed by the authority | (B) Image authentication performed by the target

contains one final comparison that represents a potential vulnerability to Fault-Injection. An attacker could try to fault a comparison to force its own certificate or image to be accepted. Furthermore, there is an other vulnerability when the FSBL check if the Secure-Boot is activated or not. The Secure-Boot status is written in fuse. The FSBL reads the fuse and perform, or not, the Secure-Boot depending from the fuse content. The SSBL certificate chain is composed of three certificates. Therefore there is at least 5 potential Fault-Injection vulnerabilities in the SSBL authentication:

V1. The check of the Secure-Boot status
V2. The root certificate authentication
V3. The certificate 0 authentication
V4. The certificate 1 authentication
V5. The image authentication

In this paper we explore V1, V2 and V3. We did not explore V4 and V5, we suppose that these vulnerabilities are already close to V3.

IV. Methodology to find Vulnerabilities timing

In order to bypass the authentication, we need to find the right Fault-Injection timing. Without any knowledge of the FSBL code, our methodology relies on a Side-Channel Analysis to find this timing. A Langer probe (RF3-mini) and a preamplifier (PA303/PA306) are used to capture the target EM emanations. The EM emanations can be captured from above the chip or above the decoupling capacitors.

A. SCA of the certificate 0 authentication: V3

We can use the signals between the SoC and the EMMC memory as a trigger signal to get as close as possible to the vulnerability in a similar way as [3]. Indeed, the image has to be loaded before being authenticated. Therefore, the authentication is likely to be performed shortly after the last activity on the EMMC communication bus. The first vulnerability studied is the vulnerability V3 in the authentication of the certificate 0. To find the best timing, we need to induce a modification of the target behavior that would leak in Side-Channel. By using forged inputs such as forged SSBL images, it is possible to change the target control flow. This change will leak in Side-Channel, resulting in the divergence between traces. Therefore, we compared the EM traces when the SSBL is signed correctly and when it is not. In Fig. 3 two classes of traces are represented:

- **Valid trace** in blue: Trace when a fully valid image is used.
- **Invalid trace** in green: Trace when a malicious image is used. A valid root certificate is used but the others certificates are malicious.

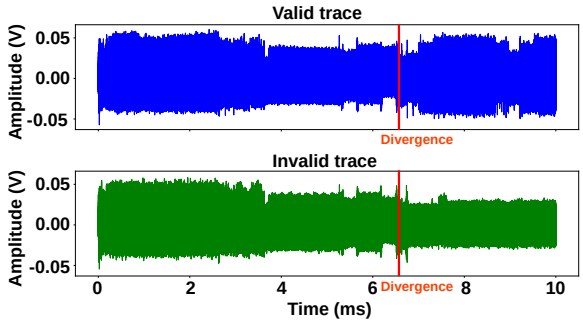

Fig. 3. Valid trace vs Invalid trace

The malicious image is first correctly signed with a certificate chain that begins with our own root certificate which hash does not match the value saved in fuses. We replaced this root certificate by the right one used in a valid image. So the root certificate hash matches the value saved in fuse, but the certificate 0 signature can't be verified by this root certificate. Our goal is to trigger the injection at the right timing to force the SoC to accept our malicious certificate 0.

On Fig. 3 the divergence between the two classes seems to appear 6ms after the trigger. A more in-depth analysis is needed to find the perfect timing. Multiple traces are needed to measure the mean timing. However due to the jitter,

the divergence is desynchronized between multiple traces. Therefore we propose a methodology to reliably find the divergence on each trace. A trace is composed of patterns that carry information about the processor activity. Patterns are persistent between traces captured in the same conditions. When a pattern is identified in a trace, it can be identified in other traces by using a comparison metric such as the Pearson correlation coefficient. Our methodology to find the divergence between two classes of traces is:

1) We use large traces like Fig. 3 to roughly localize the divergence timing.
2) We select a reference trace in one of the two classes around the divergence timing.
3) We manually select the patterns in our reference trace.
4) Using a sliding window correlation, we search each pattern in the other traces from both classes. The divergence is identified when a pattern is only present in one of the two classes. It should be the same class as the reference trace.

Fig. 4 shows an example of the sliding correlation technique. A reference pattern (in red) is retrieved in the test trace (in green) at the maximum of the sliding window correlation function (t_{max}). This sliding correlation can be replaced by other techniques such as Sum of Absolute Difference or Phase-Only Correlation (POC) [16].

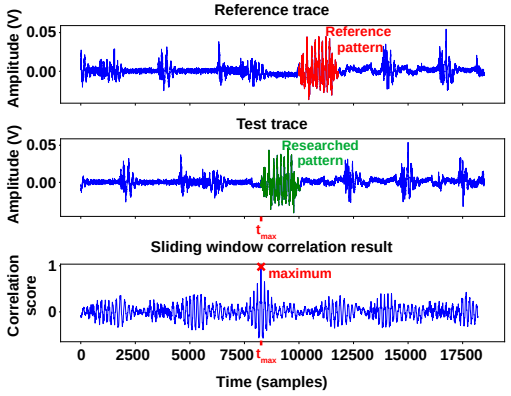

Fig. 4. Example of sliding window correlation to find a pattern

Timing identification of the diverging pattern is a necessary condition for a successful Fault-Injection. We applied the methodology presented on traces captured arround the V3 divergence. The reference trace is arbitrarily selected in the valid traces. Eventually, a pattern that is only present in the valid traces is found. On Fig. 5 several traces from both classes are realigned and superimposed to clearly illustrate the divergence between classes. When the diverging pattern has been identified, we can measure the delay before it appears in each trace. For 76 invalid traces, we measured a mean delay of 6.733ms after the trigger signal, the standard deviation of this delay is 1.87µs.

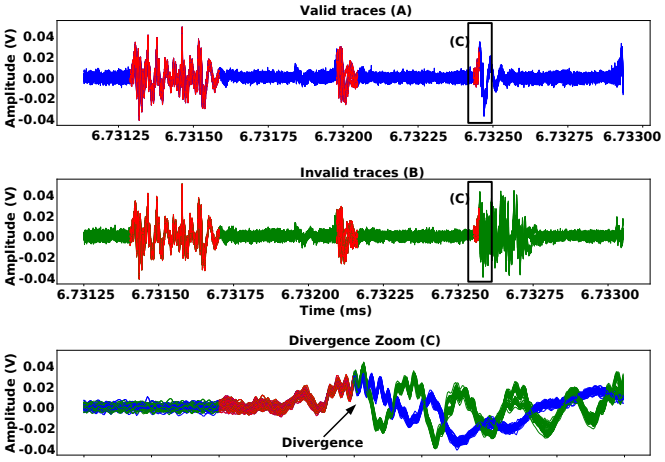

Fig. 5. Traces of both classes superimposed and aligned on the last common pattern, with a zoom on the divergence.

B. SCA of the Secure-Boot status check: V1

The same methodology is applied to find the timing for the check of the Secure-Boot status. Traces of a target with enabled Secure-Boot and traces of a target with disabled Secure-Boot are compared. The trigger signal used is the same as the previous section. The divergence happens closer to the trigger signal. We use a reference trace that belongs to the Secure-Boot enabled class to find the patterns around the divergence timing. We search these patterns in both classes until we find a pattern that appears only in the Secure-Boot enabled class. Fig. 6 shows the analysis result.

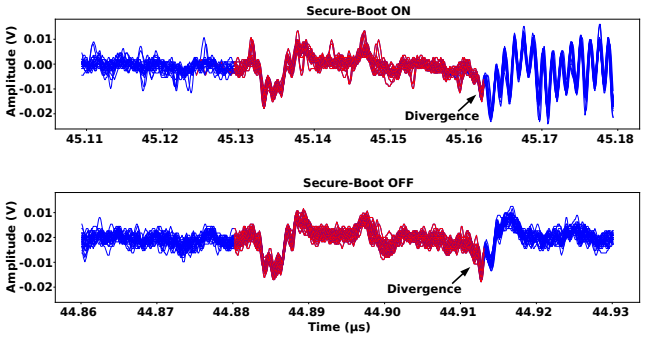

Fig. 6. Traces when Secure-Boot enabled vs traces when Secure-Boot disabled around the divergence timing

The delay between the trigger signal and this pattern has a mean value of 45.12µs and a standard deviation of 49ns measured from 67 traces with Secure-Boot enabled.

C. SCA of the root certificate authentication: V2

The previous subsections show how to find the divergence with no knowledge of the code executed. However, an authentication always ends with a comparison of two hashes. It is generally performed using a "memcmp" function (memory compare) which compares two inputs word per word. We made the asumption that the comparison was performed in a similar way as illustrated in Fig. 7 with 8-bits words.

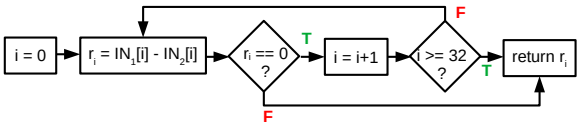

Fig. 7. Hypothesis on the memcmp architecture. IN_1 and IN_2 are the inputs.

We may identify this "memcmp" in Side-Channel by finding a pattern that repeats itself depending from the number of valid bytes in common between the two hashes that are compared. We bruteforced different root certificates with hashes that partially match the hash stored in fuse.

- 0 valid byte hash: **b043c4dd67.....26bbd6a24c**
- 1 valid byte hash: **e1eb1c648d.....0ed518fbb7**
- 2 valid bytes hash: **e11cebc2da.....ec0aa76ba2**
- hash stored in fuse: **e11c9b94af.....57256a5a08**

Then we tried to find the divergence between the traces when the SSBL is signed with these root certificates. Eventually, a pattern is found that seems to diverge according to the number of valid bytes. Multiple segments are noticed when analysing traces of Fig. 8. Blue and black segments are common between all traces. Red segments are only in traces with at least one valid byte. Green segment is only in the 2 bytes valid trace.

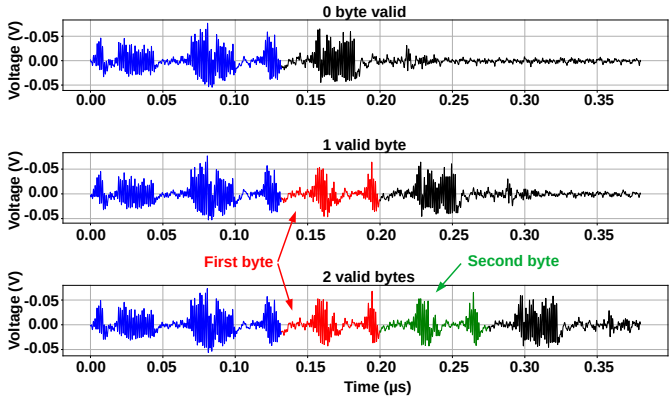

Fig. 8. Memcmp pattern for hash with 0, 1 and 2 valid bytes.

With this approach we are able to identify the "memcmp" function with higher confidence than the previous vulnerabilities. For this experiment we used the Frequency Detector described in [4] as trigger. On 65 traces we measured a mean delay of 4.95µs between the trigger signal and the pattern. The standard deviation of this delay is 492ns.

V. Proof of Concept: Secure-Boot Bypass

The goal of the previous section is to identify interesting timing for a Fault-Injection that could bypass the Secure-Boot. We measured the delay between divergences, corresponding to V1, V2 and V3, and their respective triggers. In this section, we present the setup and the results of the Fault-Injection using the different identified timings.

A. Setup

The setup used for the final experiment is described in Fig. 9. We use a voltage pulser that injects a 400V pulse in an active probe positioned above the target. The pulse width is 6ns and its rise time is 2.5ns. The methodology used to find the probe position and the pulse parameters is performed in a fully controlled environement and is very similar to the methodology presented in [18, 4]. In the final experiment, the trigger signal used is the communication signals between the SoC and the EMMC for V1 and V3. For V2 we use the Frequency Detector described in [4] sets to trigger on the activation of the 20.5MHz frequency.

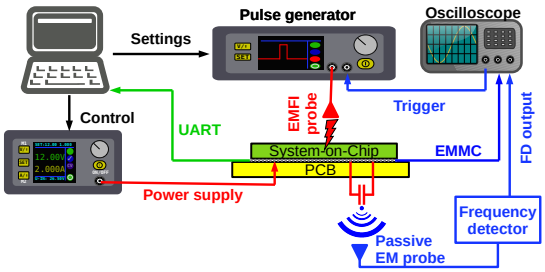

Fig. 9. Attack setup.

B. Bypass of the certificate 0 authentication: V3

The goal of this experiment is to execute a malicious SSBL image. The certificate chain of this image contains a malicious certificate 0 signed with our own private key. However the root certificate is the right one. So the target should authenticate the root certificate, then it should try and fail to authenticate the certificate 0. A fault injected at the right time might force the target to continue the Secure-Boot despite the malicious certificate 0. We used the delay of 6.733ms previously identified in section IV-A as delay between the trigger and the injection. On 25000 experiments (17 hours), we were able to boot our malicious SSBL 2 times.

C. Bypass of the Secure-Boot status check: V1

The target reads fuses to determine if the Secure-Boot is enabled or not. A fault during the comparison of the result could disabled the Secure-Boot for this boot, allowing us to load and execute our own malicious SSBL without any authentication. The injection is triggered using the communication signals between the EMMC and the SoC. We used the mean delay of 45.13µs previously identified. However after 50000 attempts (78 hours) without results, we gave up on bypassing the Secure-Boot. A possible explanation might be the use of function pointer in the code. In this case, finding the divergence is not enough to achieve a successful bypass. Therefore we used another approach, we assumed that there might be vulnerabilities between the trigger signal and the divergence. So we explored each nanosecond between the trigger and the timing identified. The whole campaign duration was 80 hours. Eventually we get 3 success:

- Success 1: 25.88µs.
- Success 2: 29.52µs.
- Success 3: 29.64µs.

Giving the low jitter, Success 2 and 3 are probably caused by the same vulnerability. However, Success 1 seems to be caused by an other vulnerability. Also it shows that the divergence may not be the right timing for the vulnerability, but it gives an upper bound to the time-domaine exploration.

D. Bypass of the root certificate authentication: V2

To attack the root certificate authentication, a modified image with a malicious invalid root certificate is used. The two first bytes of this malicious certificate hash match the two first bytes of the expected hash. We assume that injecting a fault at the right timing may cause a premature loop exit such as presented in [18] with a return value of 0. The activation of a specific frequency in the EM emanations is used as trigger. The Fault-Injection is triggered with the delay previously identified. With this setup, we performed 55000 experiments (66 hours) which resulted in 10 successes. The better jitter measured for V2 might explain this higher success rate.

VI. Conclusion

This paper proposes a methodology to answer problematics P1 and P2 in the context of a complex target such as a SoC and when the attacker has no access to the code. This methodology exploits the fact that information about the processor control-flow leaks in Side-Channel. The research of divergent patterns in EM traces allows to identify relevant timing for Fault-Injection. These timing are also used to find the best trigger, which partially answer P3. It does not ensure a success, but it gives boundaries to the timing window in which the vulnerability is. By forging specific images, we were able to create different behaviors in Side-Channel. This allows us to reverse the architecture of specific function such as the memcmp function. We performed the proof of concept by bypassing the Secure-Boot of our target for three vulnerabilities.

This methodology is expected to be used when the attacker does not have the target code. It allows to find some vulnerabilities, but not necessarily all the vulnerabilities. A full scan of the delay between the EMMC trigger and the end of the last authentication would be useful to compare results. However the full authentication chain takes several milliseconds, which would take months to fully explore. An alternative would be to use a classic methodology using a known code and GPIO to find the vulnerabilities, and then compare the results with our methodology results. P4 is not explored in this work. A controlled software is used to find the best Fault-Injection parameters. Also, only one SoC was studied, and it is an old model that is not used anymore in new smartphones.

Finally, we believe that the methodology presented in this paper could be used on others target and other scenarios. In particular, bypassing the Secure-Boot at the FSBL level could be used in forensic investigation to execute code controlled by the forensic experts with TEE privileges, which would lead the way to other attacks.

VII. Acknowledgment

The experiments were done on the Micro-PackS™ platform. This work has benefited from a government grant managed by the National Research Agency under France 2030 with reference ANR-22-PECY-0009

References

[1] N. Timmers, A. Spruyt, and M. Witteman, "Controlling pc on arm using fault injection," in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 25–35, IEEE, 2016.

[2] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, and S. Ermeneux, "Laser-induced fault injection on smartphone bypassing the secure boot-extended version," *IEEE Transactions on Computers*, vol. 69, no. 10, pp. 1449–1459, 2018.

[3] O. Bittner, T. Krachenfels, A. Galauner, and J.-P. Seifert, "The forgotten threat of voltage glitching: a case study on nvidia tegra x2 socs," in *2021 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, pp. 86–97, IEEE, 2021.

[4] C. Fanjas, C. Gaine, D. Aboulkassimi, S. Pontié, and O. Potin, "Combined fault injection and real-time side-channel analysis for android secure-boot bypassing," in *Smart Card Research and Advanced Applications: 21st International Conference, CARDIS 2022, Birmingham, UK, November 7–9, 2022, Revised Selected Papers*, pp. 25–44, Springer, 2023.

[5] C. O'Flynn, "Bam bam!! on reliability of emfi for in-situ automotive ecu attacks," *Cryptology ePrint Archive*, 2020.

[6] N. Kühnapfel, R. Buhren, H. N. Jacob, T. Krachenfels, C. Werling, and J.-P. Seifert, "Em-fault it yourself: Building a replicable emfi setup for desktop and server hardware," in *2022 IEEE Physical Assurance and Inspection of Electronics (PAINE)*, pp. 1–7, IEEE, 2022.

[7] *Espressif ESP32: Bypassing Secure Boot using EMFI*. Available at https://raelize.com/blog/espressif-systems-esp32-bypassing-sb-using-emfi/.

[8] H. Marco and A. Vicent, *Vlind Glitch: A Blind VCC Glitching technique to bypass the secure boot of the Qualcomm MSM8916 mobile SoC*. Available at https://www.cyberintel.es/publications/2022-12-07_BlackHat_Europe_pub/.

[9] S. Nashimoto, D. Suzuki, R. Ueno, and N. Homma, "Bypassing isolated execution on risc-v using side-channel-assisted fault-injection and its countermeasure," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 28–68, 2022.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pp. 388–397, Springer, 1999.

[11] T. Eisenbarth, C. Paar, and B. Weghenkel, "Building a side channel based disassembler.," *Trans. Comput. Sci.*, vol. 10, pp. 78–99, 2010.

[12] J. Park, X. Xu, Y. Jin, D. Forte, and M. Tehranipoor, "Power-based side-channel instruction-level disassembler," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.

[13] Y. Liu, L. Wei, Z. Zhou, K. Zhang, W. Xu, and Q. Xu, "On code execution tracking via power side-channel," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1019–1031, 2016.

[14] Y. Han, S. Etigowni, H. Liu, S. Zonouz, and A. Petropulu, "Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1095–1108, 2017.

[15] N. Debande, Y. Souissi, M. Nassar, S. Guilley, T.-H. Le, and J.-L. Danger, ""re-synchronization by moments": An efficient solution to align side-channel traces," in *2011 IEEE International Workshop on Information Forensics and Security*, pp. 1–6, IEEE, 2011.

[16] N. Homma, S. Nagashima, Y. Imai, T. Aoki, and A. Satoh, "High-resolution side-channel attack using phase-based waveform matching," in *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*, pp. 187–200, Springer, 2006.

[17] Q. Tian, A. Shoufan, M. Stoettinger, and S. A. Huss, "Power trace alignment for cryptosystems featuring random frequency countermeasures," in *2012 Second International Conference on Digital Information Processing and Communications (ICDIPC)*, pp. 51–55, IEEE, 2012.

[18] C. Gaine, D. Aboulkassimi, S. Pontié, J.-P. Nikolovski, and J.-M. Dutertre, "Electromagnetic fault injection as a new forensic approach for socs," in *2020 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2020.