Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot

Aurélien Vasselle, Hugues Thiebeauld
Eshard
Bordeaux, France
aurelien.vasselle@eshard.com

Quentin Maouhoub, Adèle Morisset, Sébastien Ermeneux
ALPhANOV
Bordeaux, France
adele.morisset@alphanov.com

***Abstract*—This paper describes the outcome of a laser attack study on an Android smartphone targeting specifically the secure boot sequence. Laser fault injection has become a classical attack path in the secure chip industry to investigate potential security mitigation. The implementation of such attacks on a recent mobile phone remains relatively unexplored and represents different challenges, both at hardware and software levels. In this paper, we show how the device is crafted to get a direct access to the silicon and explain the corresponding experimental setup. By inserting our own software into the boot sequence, it was possible to achieve a fine characterization of the die sensitivity to light. With the knowledge of potential perturbations, it was possible to build an attack scenario allowing to malevolently get the highest level of privilege within the mobile phone.**

***Index Terms*—Laser Fault Injection; Smartphone; SoC; Secure Boot; Hardware security; Reverse engineering; TrustZone; TEE**

I. Introduction

A. Laser Fault Injection

Hardware fault injection has become a common practice to stress the security of embedded devices like secure controllers, RAM, processors, etc. These techniques can be obtained with unexpected variation of physical properties, such as clock or voltage glitches, substrate biasing [1] or even electromagnetic [2] and light beam interaction [3].

Optical fault injection is a technique allowing to induce a physical disruption inside a chip aiming to create a faulty behavior during code execution. Practical results were first published by Skorobogatov and Anderson [4], using a focused camera flash to obtain flipped bits by locally interfering with an integrated circuit. Relatively shortly after this publication, laser systems turned out to be a very powerful mean to inject faults on semiconductor devices. Indeed, laser systems combine a high power with fine control over the surface and time to achieve finer disruptions. The laser beam generates electron-hole pairs when passing through the silicon with an appropriate photon energy of at least 1.12eV (silicon band-gap), corresponding to a wavelength of at most 1100nm. The electrical field generated by the transistors can make these charges drift and create a transient photocurrent. This spatially and temporally localized photocurrent may have an incidence on the circuit logic. It can cause multiple bit errors as well as controlled and reproducible single bit-flip by choosing the appropriate laser parameters [5].

Cryptographic attacks assisted by fault injection into secure devices have proven the ability to obtain bits of a secret key since the Differential Fault Analysis (DFA) [6] [11]. All algorithms are theoretically vulnerable to laser fault attacks: lots of studies were performed on Data Encryption Standard (DES) [7], Advanced Encryption Standard (AES) [8][9][10] and RSA [11]. As a result of the cat and mouse game, dedicated countermeasures have been designed to tackle the threats behind faults. A first principle is to detect the physical disruption using sensors. In particular light injection can be prevented with photosensitive semiconductor structures implemented on the sensitive areas of the die [12]. A more generic approach makes use of algorithmic protections at hardware or software level to insure redundancy and/or error correction of the program and data flow [13]. State of the art laser attacks now require to get very fine beams to find a way between sensors and other hardware protections. Moreover, applying different laser pulses on the silicon in time and space became a way to circumvent logical protections in secure software. Exploring laser attack techniques to validate the security of a product is important, particularly when the silicon can be directly accessed without big efforts [14].

A lot of parameters must be considered when implementing a laser fault injection attack: wavelength, spot size, location on the chip (X, Y), focus (Z), timing, pulse width and intensity. Only a proper combination of these physical parameters can lead to valuable faults and be exploited through a practical attack scenario.

B. Secure Boot

Secure Boot refer to a sequence of cryptographic verifications insuring the authenticity and conformity of bootloader software in order to sustain a Chain of Trust. By doing so, the device can securely initialize a Trusted Execution Environment (TEE), starting from the SoC bootrom code, inherently secure. It allows software, peripheral and I/O protection, code isolation and security management. ARM leadership on the smartphone market has placed TrustZone, their hardware-based security solution, inside almost all the devices offering secure boot.

To ensure the origin of any software and make sure it can be considered secure, the device relies on software verifications involving asymmetric cryptography, in which trusted authorities can sign a code with a secret private key. Doing so,

978-1-5386-2948-2 /17 $31.00 © 2017 IEEE
DOI 10.1109/FDTC.2017.18

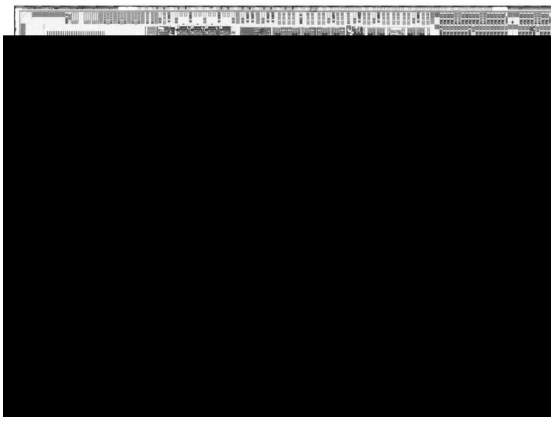

Fig. 1: Secure boot program flow

everyone has the opportunity to authenticate the author by using the corresponding public key. This is particularly critical when dealing with several entities and companies.

In most cases, secure boot is an elusive attack target since a standard user has no way to either interfere with the executed software or even understand it, provided that it is properly ciphered and/or obfuscated. Even if software attacks have exploited design flaws in the TEE to obtain secure code execution [16][17], only few attacks targeted the secure boot. Thanks to its limited attack surface, this code is by nature more likely to be flawless. This explains why it is particularly valuable to consider physical attack scenarios during the early stages of boot. Previous work has demonstrated a successful voltage glitch attack [18] exploiting ARMv7 program counter specificities.

C. Motivations

As seen in the smartcard industry over the past 15 years, hardware attacks can take security considerations one step further by stressing software and cryptographic implementations considered secure. Yet, their applicability to mobile phones is not obvious. Among these examples the laser, if well mastered, constitutes the tool with the greatest potential to perform fault injection on a complex SoC, thanks to its high precision and efficiency.

However, the meandering route towards a working proof of concept requires to overcome several technical challenges. Indeed, exposing the SoC die within a mobile phone represents the first obstacle since the Package-on-Package technology became usual in mobile phones or smart devices industry: the SoC is confined between the PCB and stacked DRAM dies, and therefore it cannot be easily exposed to laser beams. When mechanical issues are overcome, the physical dimensions involved in the technology still represents a challenge by the huge parametric exploration space, which cannot be entirely exhausted. Furthermore, mobile phones' software architecture is complex and built upon several levels of privileges. The whole platform security heavily relies on the secure boot sequence, which makes a manifest target for fault injection as explored in [18]. As far as our knowledge, no published work showing a laser fault injection targeting an off-the-shelf SoC has been released.

Exploring this critical boot sequence allowed us to get a better understanding of the security protocols and how much they could be stressed. Although the hardware and software manipulations can be tricky, the outcome of breaking the secure boot sequence is worth the effort when considering the stakes of a successful attack: secure privilege escalation at bootloader level or signature forgery. It may turn out to be worthwhile for critical assets.

On one hand, the main purpose of this study is to forge a better opinion about the risk that laser attacks may present and the corresponding impacts for embedded devices security. On the other hand, we wanted to assess how practical and difficult it would be to set up such an attack.

II. Gaining Access and Knowledge

The purpose of this section is to explain how the device under test (DUT) was altered for our experimental setup in order to be prepared for optical fault injection. After finding entry points, the reverse engineering phase led to valuable understanding of the architecture and a smart physical access method.

A. Low level Software and Hardware Access

The first step towards our goal is to take control of the device at an early stage of the boot sequence. For this we had to exploit a software weakness which allowed us to flash part of the firmware without being detected by the TEE security. This gives access to debug functionalities, allowed for reverse engineering and later for useful modifications of the boot sequence.

B. Architecture Overview - Reverse Engineering

One indisputable factor of success during the preparation of the device was the software reverse engineering. We mostly used Hex-Ray IDA pro as static analysis tool for firmware disassembly and decompilation, in cohesion with Zynamics' Bindiff, to easily compare functions reversed from other firmwares or similar devices. The study proceeded in background by accumulation of small mechanisms understanding, lecture of the available documentation and substantive thinking.

We were able to dump every piece of code composing the boot sequence at the runtime, bypassing any encryption and memory protection. The firmware is composed of few hundred kilobytes of compact code and the overall program flow is simple - apart from the security perspective. This confirmed the initial statement: achieving software attacks on the boot sequence is very uncertain, considering the very low number of entry-points and the architectural strength.

After rearranging the puzzle pieces, the objective was to understand the big picture (see Figure 1)

On reset (or power-on), the device sets up a basic environment to allow a safe execution of its boot sequence. For

example, it sets up the different clock ratios, and the main clock at $1.4\,\mathrm{GHz}$.

Once the environment is stable, the SoC chooses its boot media and retrieves the firmware (FW) that is then loaded in its internal SRAM memory for verification.

The secure memory in which the FW is copied cannot be modified by the user at runtime, thus, the CPU can safely validate the firmware integrity via a checksum and then process to the cryptographic verification. First the public key is authenticated with an Hash-based Message Authentication Code (HMAC) to make sure that it has not been modified. Then it can check the FW signature with this public key to confirm the code origin and author. Lastly, the now-trusted software can be AES deciphered with a hardware-backed master key in order to be, ultimately, executed. The overall design of the cryptographic system is robust and mastered.

These steps should be executed between each program block handover in order to build a chain of trust. Exploring different devices, we could see that the first stage is very well protected, but factually the further you progress in the chain of trust, the more these protections decline.

Noticeably, we never encountered any countermeasure against physical attacks, and worse several weaknesses were found. From a functional point of view, the secure boot security can be bypassed by conditional "ifs" already hardcoded in the Read-Only Memory (ROM). These bypasses cannot be reached by software as hardware fuses permanently disabled the program path. But without resilience, every physically-induced misbehavior during this boot sequence will result in totally different program flows. Among them several bypasses can happen. This is in direct contrast with the security architecture complexity, but makes us enthusiastic about the purpose of our study.

C. Physical Access

In the smartcard security industry, gaining a non-destructive physical access to the chip imply opening packages with mechanical or chemical preparation to remove the different layers of plastic, metal, or epoxy. Once uncovered, the chip may require additional preparation in case of sophisticated attacks. For instance bulk thinning with dedicated machinery enable to reduce laser dispersion when attacking the backside.

For a mobile phone, the accessibility is different. When targeting the SoC, a simple hot air gun allows clearing direct access to the backside surface, taking apart the 3D package stack. But the main issue remains: The chip is located underneath its DRAM in a Package-on-Package (PoP) [19] assembly. In other words we need to somehow keep the device working while obtaining optical access for a laser pulse illumination.

The most generic solution is to relocate the DRAM with a dedicated PCB aside from the main die but it requires highly accurate routing and State of the Art industrial technologies. The development of such solutions was considered and may be mandatory on other devices, but for our target device, we took a simpler exploration path that kept the attack costs as low as possible.

At this stage we had reversed enough information concerning the target smartphone boot sequence to know that it was possible to modify the hardware configuration at the bootloader level. As a result, we can take control of the executed code before most of the device initialization, in order to bypass the DRAM setup. Our code is thus located inside the SoC internal SRAM memory.

For our analysis, the device does not have to boot all the way to the Linux operating system or Android. The fault injection setup was focused on the secure boot sequence, and these modifications allow developing a complete software environment to test the security of the lowest cryptographic layers.

III. Fault Injection Setup

A. The Target

The most relevant specifications of our targeted system on chip architecture are:

- Quad-core Cortex A9 @ 1.4 GHz - 0.71ns clock period
- 8-Stage pipeline with variable length, NEON extension
- 32 KB Instruction and Data L1 cache with 1 MB shared L2 cache
- Out-of-order execution and Dynamic branch prediction
- 64 KB ROM, internal 256 KB RAM, and eFuse key storage
- Hardware Crypto-engine, TrustZone Hardware
- Multiple clock and power domains over the SoC

Speculative execution, related to caches and dynamic code optimization make software fluctuating at runtime, and are very complex to analyze without precise knowledge of the architecture. But these observations do not apply during the boot, which is inherently predictable. However, the challenge lies in uncharted obstacles remaining: the clock speed, the pipeline architecture and the transistor density. Our concerns are more focused on the semiconductor processes at the CMOS 32nm High-K Metal Gate (HKMG) technological node.

Using a microscope station as described below, we measured a silicon thickness of $125\,\mu\mathrm{m}$, which is thin enough to allow infrared imaging and provide a suitable laser penetration throughout the substrate. This avoids thinning a large die over $1\,\mathrm{cm}^2$ of surface. Backside IR imaging provided a good way to observe the semiconductor properties and the overall system architecture as depicted in Figure 2. Moreover, due to the high number of metal layers, at least 9, the backside approach turned out to be a mandatory choice for the infrared laser.

The manufacturer gives a gate pitch of $110\,\mathrm{nm}$, with an average density of $3\,000\,000\,\mathrm{gates/mm}^2$. A rough approximation of the number of logic gates in the ARM cores leads to 27 million, over a surface of $9\,\mathrm{mm}^2$.

B. Bench Hardware

We used a commercially available fault injection bench [15]. A single-mode laser emitting around $1\,\mu\mathrm{m}$ with Pulse-On-Demand capabilities at the nanosecond is used within a dedi-

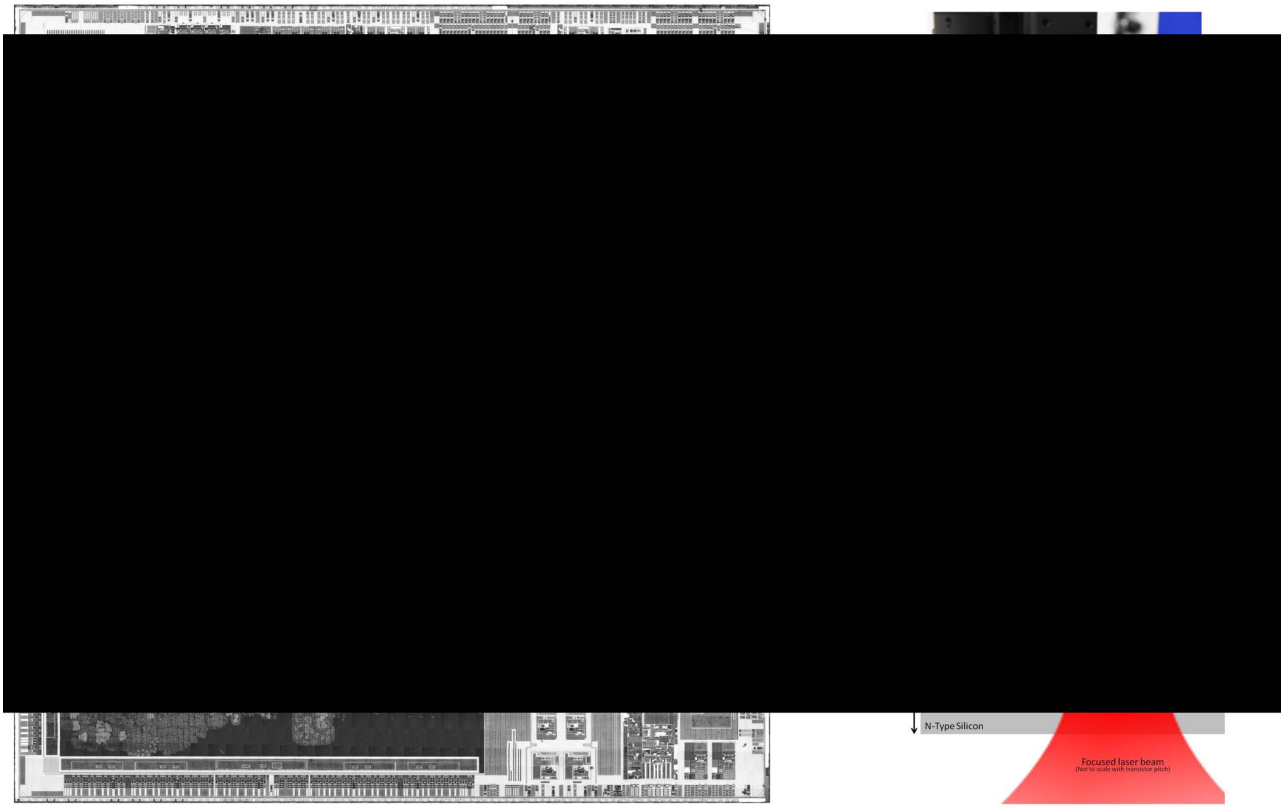

Fig. 2: (a) Backside infrared imagery of the SoC (150 MPixels) - (b) Fault injection setup - (c) Cross-section view of the DUT

cated microscope station. The optical system setup consisted of a laser microscope mounted on X, Y, Z translation stages fixed on a marble arch for stability. We were thus able to scan the SoC entirely with the laser spot, keeping the sample stationary and did not meet any ergonomic or positioning issues.

The setup is able to reach sub-micrometer laser spot size on the surface of the sample with an adequate objective lens. We used a 20X special objective which allows us to measure a $3.4\,\mu\mathrm{m}$ spot size at Full Width Half Maxmimum (FWHM) using the knife-edge technique. The laser spot size was probably slightly larger within the silicon. This is huge compared to the transistor size ($32\,\mathrm{nm}$ gate and $110\,\mathrm{nm}$ contact pitch at metal 1 level). A large spot size is not a problem when empirically scanning the device for fault effects. However, a smaller one helps to differentiate faults effects and localize more precisely areas of interest. Indeed, even when the spot size does not allow to target a single logic cell, it remains possible to inject single-bit faults in the chip as demonstrated by Agoyan [5].

As the substrate is thin and the technology is low power, it is natural to think that a physical disruption in the core logic is easily achievable. However, a high laser power precision is beneficial for two related reasons. First to generate more exploitable fault effects, as it is necessary to target the thin power band between an energy level without injected fault and a threshold where the device mostly crashes. And most importantly, to effectively increase the spot locality: factually, at the given gate density, a $10\,\mu\mathrm{m}$ spot covers 942 gates. However, when considering the Gaussian intensity distribution of the single-mode laser, 109 gates receive at least half the maximum intensity inside the $3.4\,\mu\mathrm{m}$ spot at FWHM; and half the total energy is located in a $1.95\,\mu\mathrm{m}$ spot, illuminating only 35 gates. Using a single-mode laser, finding the critical energy required to disrupt the logic allows to refine the faulted area to much smaller dimensions, in the order of magnitude of the $32\,\mathrm{nm}$ technological node.

The device is clocked at $1.4\,\mathrm{GHz}$ during most of the boot sequence. This corresponds to a $710\,\mathrm{ps}$ period. The laser jitter is very low ($< 8\,\mathrm{ps}$) which is important for the repeatability of the tests. Most of our tests were performed using a specially made trigger setup which generated a significantly higher jitter of $3\,\mathrm{ns}$, corresponding to 4 operations. With this setup, timed attacks have to be repeated several times to confirm a fault effect.

C. Software Setup

A dedicated software controls all the instruments used in the bench to automate the tests as much as possible.

We developed a bare-metal software environment at firmware level on the phone in order to configure the smartphone SoC and board, communicate with the main computer

and execute our function under tests (FUT). The device is set up to actively wait for a command, acknowledge its reception and immediately jump into the test loop. The laser pulse is then triggered on emission of a character by the phone to eliminate as much jitter as possible for the laser pulse delay.

Hardware modifications have been made to power-up the smartphone automatically via the programmable power supply, without human intervention, so that the software could decide to reboot the device when necessary.

An area to scan is selected with help of the infrared camera imaging. The position parameters on the X-Y-Z axis, the pulse width, the diode current and the pulse delay are all set respectively to a minimum value, a maximum value and a step size in the software. Then the program loops through each combination of these parameters.

For each sequence of tests, the initial answer of the DUT is stored in the PC as a standard answer, it depends on the internal SRAM reset values, and was reset after each reboot of the device. There are 3 cases for the response of a laser pulse:

- The device delivers the expected result, no fault is detected
- The device delivers no answer, which usually means it has crashed.
- The device delivers an answer different to the standard one, meaning that a fault was successfully injected in the chip.

In cases 2 and 3, the answer (or lack thereof) is logged in the file with the current parameters, and the device is rebooted. Then, the next combination of parameters is tested.

It is important to highlight that ARM exception provides fault resilient architecture to a certain degree: we implemented exception handlers from scratch and configured the device in order to log special events (data abort, prefetch abort, undefined instruction, etc). Nevertheless laser fault injection is able to put the device in an unreachable state where it generally crashes without logs.

Most parameters, except for the positions in X and Y, are either fixed or with a small number of iterations for a given test. After scanning roughly the selected zone, more tests can be done on areas where faults were successfully injected in the device, but with a smaller step size to better localize the interesting points.

D. Estimed durations

To start exploring the huge dimensional space of the experimental parameters we decided to have a simple coarse-fine approach, in which we proceeded to an overall scan in order to have an idea of the adequate energy levels and sensitive locations of the chip, as described in the next section.

The scanning process is very time-consuming, so the limits and step sizes must be considered carefully when initiating a sequence of tests. The delay between pulses is actually limited by the communication speed, which allows only 11520 characters per second. Indeed the more data you decide to log, the longer the tests will last. But the device also needs to be reset when it crashes, which takes approximatively two seconds.

To give the reader an order of magnitude, scanning over the $9\,\mathrm{mm}^2$ CPU block with 10 values of pulse width and energy per location, will require 2.25 million pulses at $20\,\mu\mathrm{m}$ steps, which is about 4 times the spot size; 20 million at spot size and 100 million at half the spot size. Even with an average of 5 laser pulses per second, it represents full weeks of experimental testing.

Still, the temporal exploration is not yet initiated. As the device is quite fast, even focusing 1 ms of the 300ms-long secure boot, corresponds to the processing of more than a million instructions.

E. Fault Parameters Sensitivity

In this experiment we try to set aside the time dimension and observe the fault sensitivity, however keep in mind that the time delay of the laser pulse into the boot sequence is by far the largest dimension.

The misbehaviors appeared quickly and allowed to identify core 0, handling the boot sequence and all our functions under test.

Depending on the areas, coarse scans resulted in between 5 to 40% effective fault injections. About a half of them can be considered as crashes. Indeed the most common case is that the phone is left unresponsive during the logs. We could identify 26% of the crashes as caught exceptions, probably caused by a bad pipeline feeding. In this case the device can recover from its faulty behavior and switch in exception mode to log valuable information about its internal state. The location of this crash effects highlights the areas of prefetching, instruction decode and data manipulation.

Over some regions the chip behavior can be so unpredictable that some faults cannot be categorized. For example 4% of them contained glitches in the log sequence and required human analysis. Due to the high amount of data generated, and with an attacker state of mind, we decided to focus on simpler faults and discard complex/multiple fault effects.

The Figure 3 shows the result of fault sensitivity characterization of a simple FUT: a loop counting until a given limit. Blue areas led to successful fault injection and in the orange area, the output can be considered as crash. This scan was performed with a hundred unique combinations of pulse width and diode current and in the end, each effect is stacked in order to conglomerate glow intensity. The resulting color corresponds to the likelihood of the effect over the range of assessed energies.

It delimits the area of the main integer pipeline and a restricted set of instruction execution areas. Changing the FUT helps finding area of each instructions processing. For example, this can be done for floating point instructions to highlight the NEON pipeline.

The optimal parameters could be guessed from these experimental scans, to perform finer scans around area of interest afterward. The Figures 4 and 5, which corresponds to Figure 3

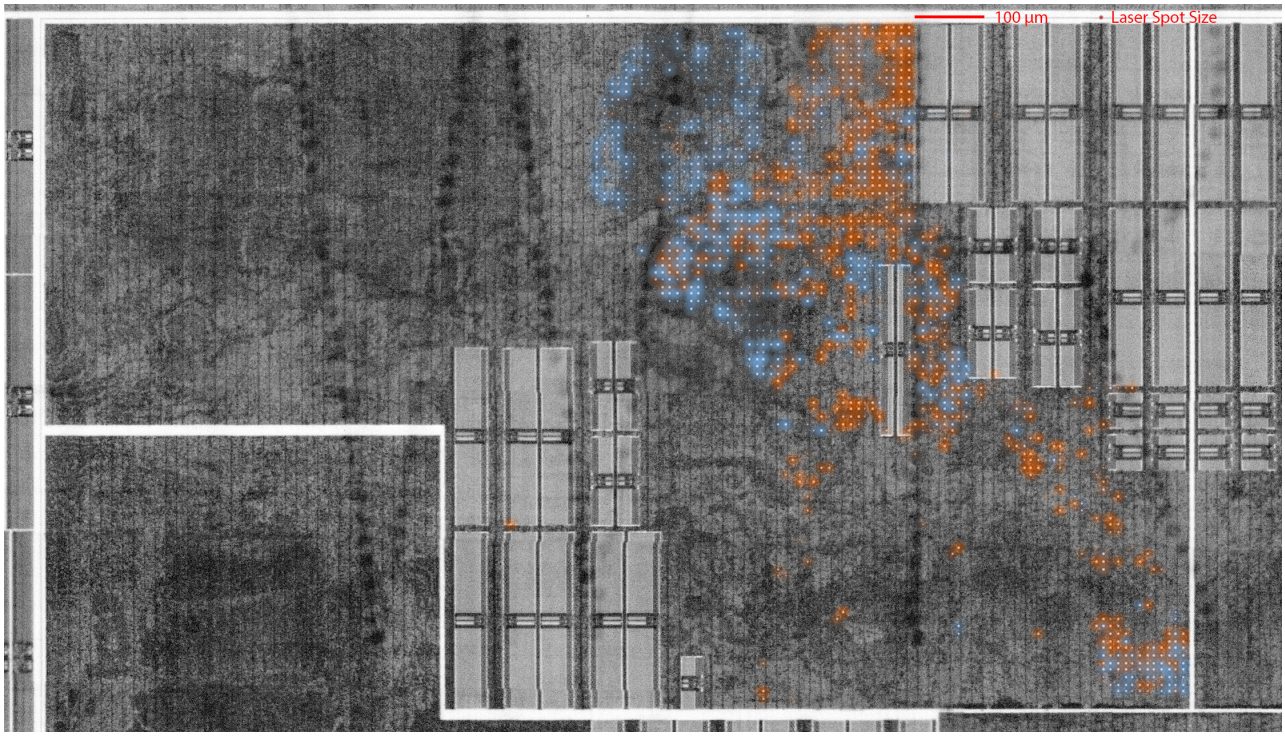

Fig. 3: Core0 sensitive areas

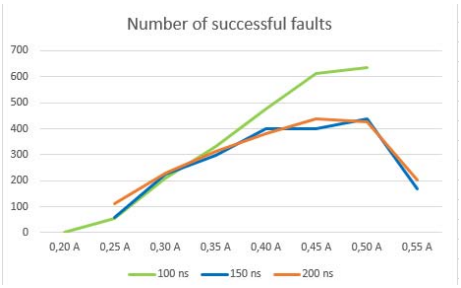

Fig. 4: Number of successful faults depending on the diode current and pulse width

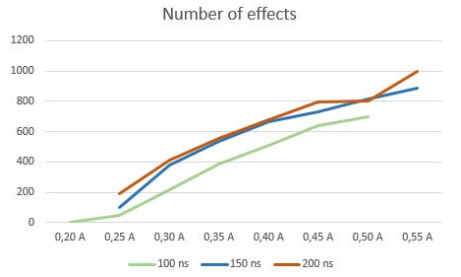

Fig. 5: Number of effects depending on the diode current and pulse width

scan, illustrates that the critical pulse width limit, at which it no longer affect the physical response of the chip, might be higher than expected. There is a significant difference between 100 ns and 150 ns pulses at most power levels.

We later used this result in order to cause static fault effects with long pulse width: At constant pulse energy, a longer pulse is less intense and spacially refined.

IV. Characterization Results

The remaining half of faulty behaviors, where the device did not crash and the log could be statistically examined, showed a considerable number of effects on the main register values. An exhaustive classification would allow to understand what is happening at architectural level through the 8-stages pipeline. However, this is hardly conceivable without dedicated tools such as an ARM in-circuit debugger and tracer. Unfortunately this requires additional privileges on the hardware, unavailable with our blackbox approach.

A. Dynamic Faults

In this work, dynamic faults refers to alterations occurring during the processing of a specific operation. Basically the fault effect can affect a given state (Instruction Queue, Decode, Renaming, Execute, Write Back) of the pipeline; causing the CPU to retrieve false operands, execute unexpected instructions or write erroneous results in a possibly different output register.

We witnessed a large number of effects over the parametric space available. It is possible to characterize a short sequence of instructions, searching for desired effect in the program

00000000	FFFFFFFF
00002000	FFFFEFFF
00008000	FDFFDFFF
00400000	FFFFFFFD
00008000	FFFFF9FE
00408000	00000000

Fig. 6: Example of static fault injection over unused registers

output, expecting to obtain identical behaviors during the actual boot sequence.

Dynamic faults results led to numerous attack paths. However, as the FW is compact, most of them required an instruction level fault, without impact on the surrounding instructions. Indeed the so called timed attacks are quite hard to implement and simpler attack scenarios, removing the time dimension were later intended.

B. Static Faults

On the contrary static faults differentiate themselves by their simplicity: they allow to flip a bit of currently unused registers at any given time of the processing. If the register is actively processed, there is a possibility of overwriting the fault effect.

The examples given in Figure 6 shows that most of the imaginable scenarios can happen. The precision of laser fault injection helps to separate bit-flip behaviors locally. Strangely they often affect R6 and R7 registers, more than others.

We obtained a high repeatability of these fault effects. Mostly because the pulse width was way greater than the clock cycle and we were not bothered by timing considerations at all.

As static fault effects do not require timing precision, they represent a powerful instrument for exploitation. We decided to build our attack scenarios with static faults, making use of the ARM architecture.

First we noticed on Figure 7 the possibility to fault the Current Program Status Register (CPSR) which holds the comparison flags as well as the core configuration. Thus it was possible at any given time to:

- Green dots: enable or disable interrupts (IRQ, FIQ and Imprecise Aborts)
- Blue dots: make the processor switch to another mode
- Purple dots: change the value of comparisons flags (Negative, Zero, Carry or Overflow) as well as Greater than or Equal (GE) flags of SIMD instructions.

On the Figure 7 orange glow represent the probability of crashes. We can see that CPSR fault are located in reliable areas and can be reproduced with accuracy. As individual bits of CPSR can be targeted, this can be exploited in several scenarios. However, the interaction of our faulty behavior with Out-of-Order execution remains unknown.

We believe that this kind of static fault injection can be done with every configuration register of the ARM core. They are

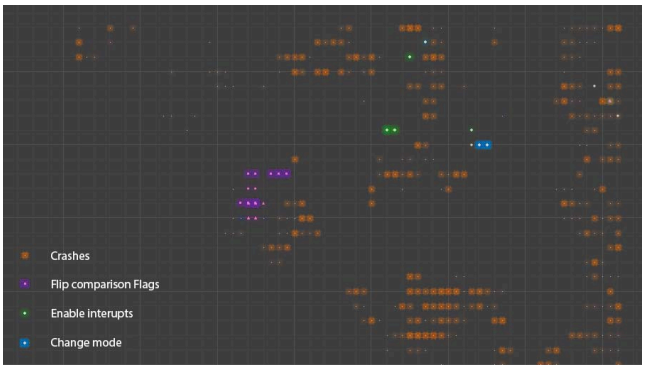

Fig. 7: Static faults on CPSR register

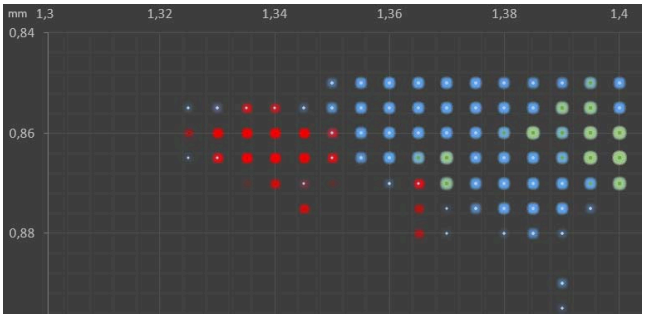

Fig. 8: Static faults on SCR register

plenty of possibilities, which extends the experimental process with additional (and slow) logs. But the most critical register in the secure boot architecture is the Secure Configuration Register (SCR), which holds the privileges granted to that core. In particular, the Non Secure bit (NS), at location 0, is set to 1 to indicate that the core is in non secure mode.

On Figure 8 green dots represent the probability to reset the bit 0 of SCR, blue dots represent bit flips on the other bits of SCR and red dots crashes. We managed to obtain flipped bits on the majority of positions in the register, even if the ARM architecture effectively uses only the first 9 bits, the other being reserved for future use.

Once the correct parameters were found, the bit reset fault can be done without crash, with a success rate of more than 95%. This fine scan allows to inject single bit flips in the SCR register with a large spot. The overall granularity of this scan is $5\,\mu\mathrm{m}$, about the spot size. Single and reproducible bit-flips are in the end achievable with a relatively large spot and very long pulses. This latest result is particularly sensitive as faulting the mobile phone in the green area grants the highest privilege levels on the device, bypassing the secure boot architecture without having to understand the cryptographic implementation or achieve a finely timed attack.

V. Mitigations

We performed this attack on a widespread but specific device. Nonetheless, the ARM architecture dominates the mobile market and the targeted hardware differs little from latest

technologies. We believe that same implementation issues and physical challenges will apply on future technological nodes such as 10nm, even if the transistor dimensions meets the atomic size. Especially, it is critical to insure mastery of the laser beam shape and its energy distribution. Right now we are getting to a point where this kind of attack scenario became practicable and present a growing risk.

However mitigations techniques exist and have shown their efficiency in other industries, such as the smartcard or payTV. At the physical level first, it is possible to prevent or detect a fault injection by applying dedicated filters or sensors respectively. The drawback remains certainly the silicon surface to protect and therefore the cost it would incur. But simpler and affordable protections such as hardware redundancy or error correcting codes could easily be implemented for the core configuration registers, without exponential costs; and the ROM software in charge of the secure boot could be harden by adding checks of the program flow and fault resilience.

With regards to the importance of the boot process, our opinion is to take a better care of this sequence regardless the complexity of physical attacks. Even if these protections are not at the state of the art, it possibly might be enough to prevent exploitable scenarios. The difficulty lies on the ability to define where the protections should be implemented. The right balance between cost of the protections and security level can only be achieved by a good knowledge of the threats and possibilities offered by fault injection attacks, and their related countermeasures.

VI. Conclusion

In this paper, we showed that despite its apparent complexity, a laser fault injection attack on a mobile phone embedding recent technologies, such as the Package-on-Package, is achievable. Targeting the secure boot sequence, we managed to compromise security architecture with a laser pulse and consequently get the highest privileges on the phone model. Getting this level of privilege grants rights over a number of secure resources in the phone, such as the TEE.

This study showed that implementing such an attack requires to overcome a number of technical issues. First, an in-depth exploration and reverse engineering of the software layers and the related security mechanisms. But also physical access to the main processor, which is the most challenging issue, particularly because recent devices are highly integrated. However, all hardware modifications we applied could be undone.

The experimental setup allowed us to characterize the laser fault effects empirically. It will require further work to obtain precise information and build a fault model. Nevertheless we showed that the laser fault injection is capable of modifying the behavior of a SoC, and more particularly a high-speed Cortex A9, with various and numerous fault effects. Among them static fault effects detach themselves by their simplicity and efficiency within a real-world attack scenario.

Such a technique is an effective tool to explore in depth the core device security with a high success rate and the ability of a deep characterization. It does not appear relevant for large audience attack, but remains powerful for sophisticated attacks when targeting critical assets, performing forensic analyses, or simply evaluating the risk behind a security function.

References

[1] Philippe Maurine, Karim Tobich, Thomas Ordas, Pierre Yvan Liardet. *"Yet Another Fault Injection Technique : by Forward Body Biasing Injection",* YACC2012: Yet Another Conference on Cryptography, Sep 2012, Porquerolles Island, France.

[2] N. Moro, A. Dehbaoui, K. Heydemann, B. Robisson, E. Encrenaz *"Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller",* Proceedings of the IEEE, vol. 100, n111, pp. 3056-3076, Nov 2012.

[3] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, *"Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures",* 10th workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2013, Santa-Barbara : United States

[4] S. P. Skorobogatov and R. J. Anderson *"Optical Fault Induction Attacks",* Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers, B. S. Kaliski, K. Ko and C. Paar, ds., Berlin, Heidelberg, Springer Berlin Heidelberg, 2003, pp. 2-12.

[5] M. Agoyan, J. M. Dutertre, A. P. Mirbaha, D. Naccache, A. L. Ribotta and A. Tria, *"How to flip a bit?",* IEEE 16th International On-Line Testing Symposium, 2010.

[6] E. Biham and A. Shamir, B, *"Differential fault analysis of secret key cryptosystems",* Proc. CRYPTO, 1997, pp. 513525.

[7] P. Loubet-Moundi, F. Olivier, D. Vigilant, *"Static Fault Attack on Hardware DES Registers",* IACR Cryptology ePrint Archive (2011)

[8] C. Giraud *"DFA on AES",* Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers

[9] G. Canivet, P. Maistri, R. Leveugle, J. Cldire, F. Valette and M. Renaudin, *"Glitch and Laser Fault Attacks onto a Secure AES Implementation on a SRAM-Based FPGA",* Journal of Cryptology, vol. 24, n12, pp. 247-268, 2011.

[10] C. Roscian, J. M. Dutertre and A. Tria, *"Frontside laser fault injection on cryptosystems - Application to the AES' last round -",* IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2013.

[11] D. Boneh, R. A. DeMillo, R. J. Lipton, *"On the Importance of Eliminating Errors in Cryptographic Computations",* Journal of Cryptology 14(2), pages 101119, 2001.

[12] R. Leveugle, P. Maistri, P. Vanhauwaert, F. Lu, G. D. Natale, M. L. Flottes, B. Rouzeyre, A. Papadimitriou, D. Hly, V. Beroulle, G. Hubert, S. D. Castro, J. M. Dutertre, A. Sarafianos, N. Boher, M. Lisart, J. Damiens, P. Candelier and C. Tavernier, *"Laser-induced fault effects in security-dedicated circuits",* 22nd International Conference on Very Large Scale Integration (VLSI-SoC), 2014.

[13] M. Karpovsky, K. J. Kulikowski, A. Taubin *"Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard",* International Conference on Dependable Systems and Networks, 2004

[14] R. Torrance, D. James *"The State-of-the-Art in IC Reverse Engineering",* CHES '09 Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems

[15] Alphanov, *PDM+ (Pulse-on-Demand Modules) and S-LMS (Single Laser Microscope Station),* http://www.alphanov.com/40-optoelectronics-systems-and-microscopy-single-spot-laser-station.html, [Online; accessed 2017].

[16] D. Shen, *"Exploiting Trustzone on Android",* In Black Hat US, 2015

[17] Laginimaineb, *"Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption",* http://bits-please.blogspot.fr/2016/06/extracting-qualcomms-keymaster-keys.html, [Online; accessed 2017]

[18] N. Timmers, A. Spruyt and M. Witteman, *"Controlling PC on ARM Using Fault Injection",* Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016.

[19] K. Gutierrez, G. Coley, *"PCB Design Guidelines for 0.4mm Package-On-Package",* Application Report SPRAAV1B May 2009, and SPRAAV2A November 2013