Characterization of a Cortex-M4 based microcontroller using optical fault injection

Research Project 1 by students of the Security and Network Engineering Master
University of Amsterdam

J.Hupkens: jhupkens@os3.nl
D.Rusek: drusek@os3.nl

February 9th, 2019

Abstract—Fault injection techniques introduce faults into a target in order to alter its intended behavior by controlled environmental changes. In this research we use backside laser fault injection to characterize a Cortex-M4 microcontroller. Through different experiments, we can prove that the intended behavior of the microcontroller can be changed by modifying instructions. This attack is further extrapolated into a real-world attack, where an authentication mechanism is circumvented.

Keywords - ARM; backside optical fault injection; Cortex-M4; instruction modification; STM32F417IG.

1 Introduction

Fault injection techniques introduce faults into a target by controlled environmental changes, in order to alter its intended behavior [1]. One of the first examples of fault injection originates from the observation that chips were affected by the presence of radioactive particles in the packaging material when shipped. Due to the interaction of these particles with the chip, bits would flip and the behaviour of the chip would change [2].

There are several methods of intentional fault injection such as varying the supply of voltage, introducing variations into an external clock, changing the temperature or introducing faults via optical means [2]. These injections could result in glitches impacting the functionality of a microcontroller so that it is unable to function in the intended way. Security models such as password verification, secure boot or separation of OS privileges, often rely on the correct execution of software by hardware. Fault injection can be a powerful tool in circumventing these security models [3].

The goal of this project is to characterize the effects of laser pulses injected into the backside (silicon substrate side) of an ARM Cortex-M4 32-bit microcontroller (MCU). The Cortex-M4 is used in embedded systems and Internet of Things (IoT) devices [4]. In order to evaluate the target's behavior, we will use a software-based test suite and develop experiments trying to affect different parts of the inner workings of the MCU. The main focus is on modifying instructions and values stored in a register. The laser glitches will be introduced using Riscure's Diode Laser Station (DLS).

This paper is structured as follows: Section 2 outlines the research question, section 3 presents various fault injection techniques and background information about the ARM architecture. Section 4 explains the experiment setup, whereas Section 5 focuses on describing the research methodology. Next, Sections 6 and 7 present the results and discuss their practical application, followed by the conclusion in Section 8. Finally, Section 9 presents future work.

2 Research question

The main research question for this project is defined as follows:

> What is the security impact of injecting laser glitches into a Cortex-M4 based microcontroller?

To support the main research question the following sub-questions have been defined:

- How may laser glitches be injected into the MCU so that it results in a fault?
- What are the optimal variables for the laser to introduce faults in the Cortex-M4 MCU?
- What behavioral changes occur in the MCU when injecting laser faults?

3 Related work and background

3.1 Fault injection techniques

Semiconductors are sensitive to light and when exposed, they might switch transistors from one state to another [5]. As a result, a value in a register or an instruction could be modified [6]. When executed by the microcontroller, it produces a different outcome than originally intended.

There are several techniques of introducing such faults, which have been researched in the past. Spruyt defined a fault and attack model for voltage glitching of XMEGA microcontroller [7], while Gratchoff [8] focused on introducing faults into the CPU's program counter so that it points to an arbitrary address. If successful, it enables the attacker to run arbitrary code on a secure device. Moro et al. [9] researched the effects of electromagnetic fault injection on a microcontroller. Optical fault injection, which is generated with a strong source of light was first presented in the research paper by Skorobogatov et al. [10], where SRAM memory of a microcontroller was targeted with a budget photo flash light. As a result, the researchers were successful in changing any individual bit of an SRAM array. Later, laser fault injection targeting smart cards was performed[5].

This research contributes to the area of optical fault injection, where the characterization of the Cortex-M4 microcontroller is performed. Riscure has developed a new version of the laser [11], used for security testing of embedded hardware, which is less expensive than other models. Our goal is to conduct a research to verify whether the new laser is capable of introducing glitches into the Cortex-M4 microcontroller. We will focus on two main attack vectors: instruction modification and register value modification.

3.2 ARM architecture

The focus of this research is on a 32-bit Cortex-M4 microcontroller using the ARMv7-M architecture with a Thumb instruction set. The microcontroller has 13 general purpose registers (from r0 to r12) and a number of special purpose registers (from r13 to r15) [12].

Register r13 is used as the stack pointer. The Cortex-M4 uses a full descending stack, which means that the stack pointer holds the address of the last stacked item in memory. The link register is register r14, storing the return address when a subroutine call is made. Register r15 is the program counter, which is incremented by the size of the instruction executed.

The ARMv7 architecture is bi-endian for data access, while instructions are always fetched in little endian format [12], which is important when analyzing obtained results.

4 TEST ENVIRONMENT

The following hardware components are part of the test environment:

- An ARM Cortex-M4 STM32F417IG microcontroller, mounted on a Printed Circuit Board (PCB), hereafter referred to as Cortex-M4;
- A Riscure Diode Laser (DLS) [11], used for performing fault injection, mounted on the EM probe station [13];
- A Riscure Spider [14], responsible for finding the right moment in time to glitch;
- Riscure Glitch Amplifier [15], used to supply power to the microcontroller.

The test environment is shown in Figure 1. The test laptop is running a Python framework, which is used to perform experiments and store the results in a database. The laptop is connected to three devices: the microcontroller, the Spider and the EM probe station. The Spider is connected to the laser and to the microcontroller. It receives the glitch trigger from the microcontroller and through the Glitch Amplifier, the Spider supplies the power to the MCU. The Spider itself is capable of delivering enough voltage to the board, however, the Glitch Amplifier is needed to also deliver sufficient current. By supplying the power via the Spider we are capable of performing hardware based resets by taking the power away from the MCU. The laser and the target are placed in a metal safety box, which prevents anybody being exposed to the laser beams fired at the MCU. The laser we use for this research is a category 4 laser [16] and should be handled with care.

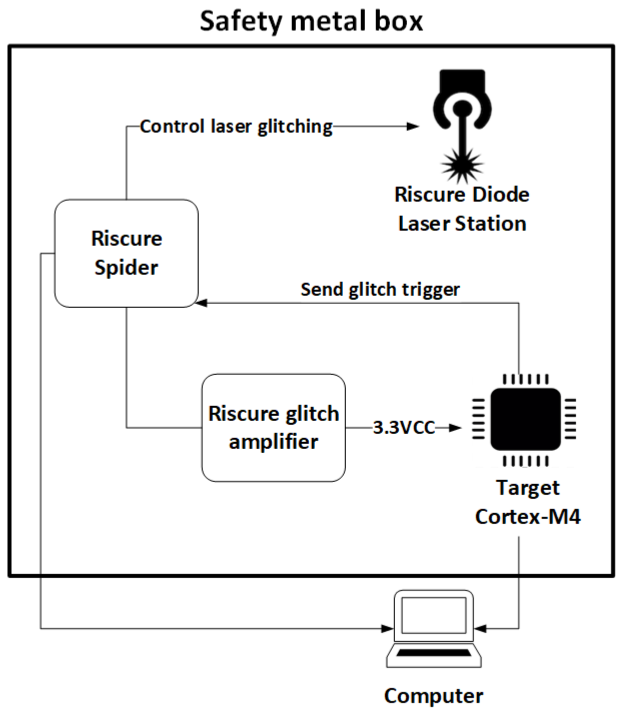


Fig. 1: The test environment for backside laser fault injection.

4.1 Device Under Test (DUT)

The target of this research is a Cortex-M4 microcontroller, which is widely used in embedded systems and Internet of Things (IoT) devices [4]. It has a True Random Number Generator (TRNG) and a hardware cryptographic processor build in, supporting the DES, 3DES and AES algorithms [17].

Laser fault injections can be performed by shooting laser beams either to the frontside (metal layer side) or the backside (silicon substrate side) of the chip [3]. The frontside of the chip provides good visibility in the layout. However, it does not allow targeting desired locations due to a metal layer, which reflects the laser beam [3] [18]. Backside injection allows for targeting the microcontroller more precisely with regards to geometric location [18], but it requires more advanced preparation of the chip.

In this research, we will focus on backside injection. The Cortex-M4 microcontroller is mounted on a PCB, which allows us to interface with it easily. Riscure prepared the PCB with a hole exposing the bottom of the MCU. Using acid, they decapsulated the bottom of the chip, further exposing the substrate. Figure 2 shows the backside of the target with the exposed silicon substrate and Figure 3 shows the picture of the die, taken with an infrared camera.

4.2 Diode Laser Station (DLS) specifications

The laser used in this research, Riscure's Diode Laser Station, uses Near Infrared (NIR) light with a wavelength of 1064 nm. NIR light with this wavelength is necessary to perform backside fault injections, because the laser beam needs to cross the entire wafer of the chip. The wafer is made of silicon, which is translucent for light of this wavelength [19], therefore, allowing the laser to reach the light-sensitive features on the die [18].

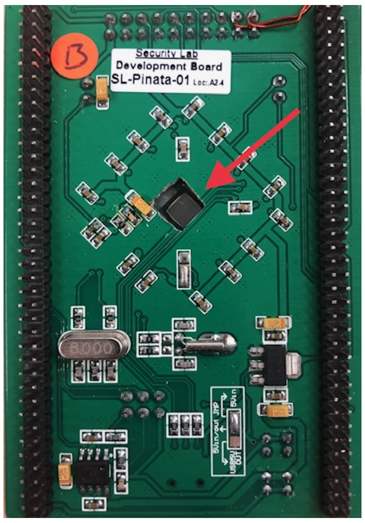


Fig. 2: Picture of the Cortex-M4 with exposed substrate, indicated with the red arrow.

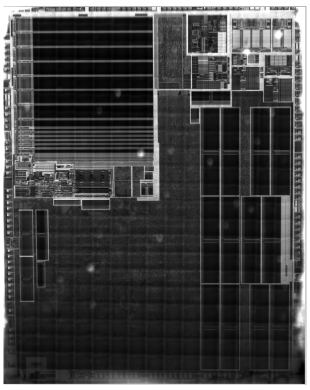


Fig. 3: Picture of the Cortex-M4 microcontroller die made with an infrared camera.

In order to cause faults, a laser beam can be injected into a target either as a continuous light source (constant output power) or as a laser pulse (laser beam in the form of pulses) [3]. It has been proven that in case of fault injection, a continuous wave does not allow for precise location of the faults, because it can affect other regions of the chip [20]. Laser pulses are more effective, as they can be switched on for short periods of time. The laser has a peak output power of 20W and the pulse duration can be configured between 20ns - 100μs [11]. The laser has a Mitutoyo NIR 5x magnifying objective installed. The lens influences the spot size, which is the area on the chip affected by the laser. To be able to aim precisely, it needs to be as narrow as possible.

When working with the laser fault injection, there are several parameters that need to be taken into account. The glitch power is the wattage of the laser pulse, given in percentages of the maximum value. The glitch delay is the time after the trigger is received by the Spider and the laser pulse is deployed. The step size is the distance between two points, defining how much the laser moves to before reaching the next point on the target. The area specifies the location on the chip, which is scanned.

5 Research Methodology

Our research consists of a series of experiments to determine the impact of laser glitches injected into the target. Our focus is on detecting behavioral changes that can be extrapolated into real-world attacks. Using Riscure's software-based test suite, we aim to detect faults resulting from the modification of instructions and the modification of register values. The test framework is able to execute multiple tests at the same time. Experiments are written in C or ARM Assembly.

5.1 Experiments

5.1.1 *Counter increment*

This test contains different types of instructions. Its goal is to prove that useful faults can occur and to verify the robustness and reliability of the test setup. The script executes 32,768 loop iterations while a counter is incremented by 1 and another counter being decreased by 1. During this process, the laser pulse is injected. Upon completion, the board returns a value. The knowledge of the starting value of the counter and the number of increment instructions allows for precalculating the result and therefore makes verification possible whether all instructions were executed correctly. A counter value different than the expected one indicates that the laser changed the intended behavior of the MCU. The expected return value for this test is 0x00008000, where 0x0000 is the decreasing counter, counting back from 32,768 to zero and 0x8000 is the incremental counter. Appendix A, Listing 9 shows part of the code used in the experiment.

5.1.2 *ADD loop*

The next step is to investigate whether we can determine which instruction is affected. For this purpose, we created an Assembly based test using multiple ADD instructions. The ADD instruction is repeated 10,000 times, during which a laser pulse is injected. When the board returns value 0x2710, it means that the execution of the program was not affected. The code is shown in Appendix A, Listing 10.

5.1.3 *Bitwise increment*

Another experiment, allowing to indicate which instruction was affected during the execution of the program is the bitwise increment. Register r1 is initialized with the value 1 after which the next consecutive power of two is repeatedly added to that value, setting every bit separately until a byte is filled. In the end, this results in the value 255. If the outcome is different, we should be able to see which instruction was modified, because that bit will still have zero as a value. The expected return value for this test is 0xff. Appendix A, Listing 11 shows the code of the test.

5.1.4 *Register value modification*

The goal of this test is to investigate whether a laser glitch can change a value residing in a register. Four registers are initialized with known values, as shown in Appendix A, Listing 7. A no-operation (NOP) instruction in form of **mov r1,r1** instruction is executed 10,000 times, during which a laser pulse is injected. After the execution of the program, the values are read from the registers. A successful glitch is identified when a register contains a different value than the one which was loaded.

5.1.5 *Authentication bypass*

This experiment is an example of a practical attack, which can be accomplished with laser fault injection. It implements an 'if' statement, which compares the password stored on the board with the password send as a payload. If the password matches, the MCU will reply with value 0x9000, otherwise value 0x6986 is returned. The laser pulse is introduced during the execution of the password authentication with the intention to bypass it by modifying an instruction. This could result in obtaining unauthorized access. Appendix A, Listing 13 shows the code in C used for this experiment.

5.2 Glitch repeatability

An important aspect of successful fault injection attack, or any experiment, is the repeatability of the obtained results. In order to achieve this, the experiments will be conducted multiple times. During the first attempt we target the entire board with the laser beam and the goal is to identify areas where successful glitches occur, without setting specific laser parameters. Detailed experiments follow, where variables such as glitch power, glitch duration, glitch delay and glitch location are set to fixed values in order to approximate the repeatability of the successful glitches from the global scan. By taking this approach, we are able to define, per experiment, a set of variables, which result in the highest ratio of successful glitches and we make sure that these faults can be repeated.

6 Results

This section presents the outcome of the experiments, followed by the analysis of the obtained data, which proves that modifying instructions with laser fault injection is possible. The code used in the experiments can be found in GitLab repository [21]. The test suite is able to differentiate between several types of results, which allow us to identify the behavioral changes caused by the laser glitches injected into the MCU. 'Expected' indicates that obtained value matches the expected value and the laser had no impact on the operation of the target. When the obtained value differs from the expected value, the glitch has affected the operation of the target and is labeled a 'successful glitch'. 'Reset' is shown, when the injected glitch alters the code, but the execution cannot continue and the target resets or 'mutes'. This could happen e.g. in case the board attempts to execute an illegal instruction or due to excessive glitch power. 'Timeout' indicates a Spider timeout, which takes place when the Spider doesn't receive a trigger signal from the target. This can happen in case the target did not yet recover from a previous, successful glitch.

By combining results from the experiments, we identified four areas on the target, where successful glitches occur. These areas are shown in Figure 4 and are referred to in this section. Moreover, we discovered that setting the glitch power between 20 - 25% of the maximum 20W was the most effective in introducing successful glitches. Setting the power to a lower value significantly decreased the amount successful glitches, whereas setting it to a higher value resulted in muting the response of the target. Other values like glitch length and glitch delay varied per experiment and thus it was difficult to define the most effective values for detailed scan.

6.1 Results of the experiments

This section outlines the results of the experiments. Table 1 shows the initial parameters used in every experiment. Glitch delay is excluded from the table including initial values, because it was experiment specific.

6.1.1 Counter increment

As shown in Table 1, we ran the experiment with random parameters and a glitch delay of 450ns, in order to identify the values resulting in the highest amount of successful glitches. Out of total 498,434 attempts over a

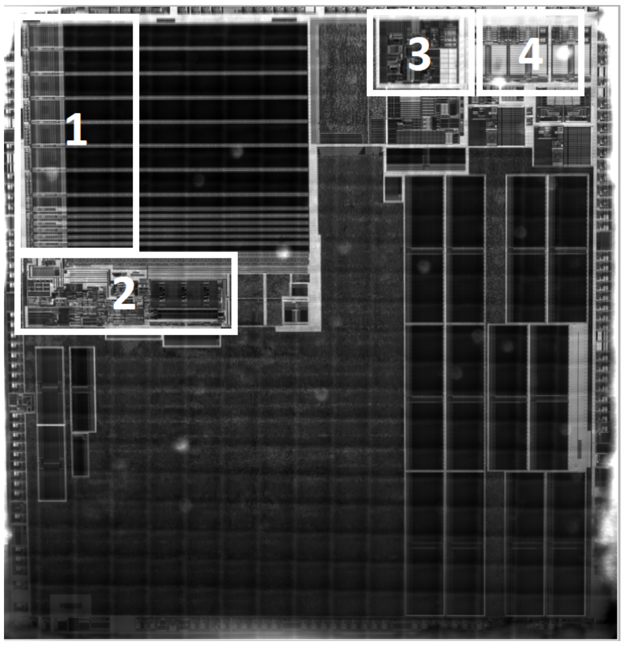


Fig. 4: Four areas on the die with the highest rate of successful glitches.

Parameter	Initial value
Glitch power	Random between 10% and 30%
Glitch length	Random between 10ns and 100ns
Step size	120μm
Location	Global scan

Table 1: Initial parameters for every experiment.

period of 7 hours with random parameters, we obtained 26 successful glitches (0.005%), 6656 mutes/resets (1.34%), 2 timeouts (<0.01%) and 491,751 times the laser pulse did not influence the chip's operation (98.66%). The experiment yields successful glitches in area 4 as shown on Figure 4. When re-running the experiment for approximately half an hour with parameters outlined in Table 2, we managed to increase the amount of successful faults to 0.04%.

Even though successful, the test is designed in such a way, that there are many different memory and register operations, e.g. MOV, ADD, SUB, LDR, STR and CMP. Hence, it is very difficult to determine which instruction was affected by the laser pulse. In order to define which instruction is affected by the laser pulse, additional experiments were performed.

Parameter	Final value
Glitch power	Random between 20% and 25%
Glitch length	Random between 80ns and 100ns
Glitch delay	450ns
Step size	100μm
Location	Area 3,4

Table 2: Final parameters for the counter increment experiment.

6.1.2 ADD loop

The experiment was successful in introducing glitches and changing the intended behavior of the system. The initial parameters set for the experiment are the same as shown in Table 1, with a random glitch delay between 8ns and 50,000ns. Out of total 564,877 attempts over a period of approximately 11 hours, we obtained 8,055 successful glitches (1.43%), 23,588 mutes/resets (4.18%) and 533,234 times the laser pulse had no influence on the MCU's behavior

(94.40%). Successful glitches appeared in areas 1, 2 and 4 in Figure 4. Next, we modified the parameters, as shown in Table 3, in order to approximate the repeatability of a successful glitch. When re-running the experiment for approximately half an hour, we managed to increase the percentage of successful glitches to 50.77%.

Parameter	Final value
Glitch power	25%
Glitch length	Random between 80ns and 100ns
Glitch delay	48000ns
Step size	120μm
Location	Area 1,2

Table 3: Final parameters for the ADD loop experiment.

6.1.3 Bitwise increment

The initial parameters set for the experiment are the same as shown in Table 1. In the first experiment, we obtained 3,829 successful glitches (0.81%) out of 472,615 attempts ran over the period of approximately 9 hours. There are 13,923 (2.95%) mutes/resets, 2 timeouts (<0.01%) and 454,861 times the laser pulse had no influence on the MCU's behavior (96.24%). Successful glitches were seen in areas 1, 2 and 4 as shown in Figure 4. After narrowing down the target area and modifying parameters which are shown in Table 4, the success rate increased to 36.14% with the experiment duration of approximately half an hour.

Parameter	Final value
Glitch power	25%
Glitch length	Random between 80ns and 100ns
Glitch delay	Random between 50ns and 200ns
Step size	120μm
Location	Area 1

Table 4: Final parameters for the bitwise increment experiment.

6.1.4 Register value modification

The goal of this experiment was to verify whether it is possible to modify a register with a laser pulse. Out of 492,532 experiments, we identified 5,091 successful glitches (1.03%), 20,088 mutes and resets (4.08%) and 467,353 times the laser pulse had no influence on the MCU (94.89%).

Parameter	Final value
Glitch power	25%
Glitch length	Random between 80ns and 100ns
Glitch delay	500ns
Step size	120μm
Location	Area 1

Table 5: Final parameters for register value modification experiment.

After focusing the laser on area 1 which had the highest density of the successful glitches and re-running the experiments with the parameters shown in Table 5, the percentage of success increased to 48.09%.

6.1.5 Authentication bypass

Out of 565,906 attempts over 11 hours, we identified 275 (0.05%) successful glitches, 27,361 mutes/resets (4.83%), 322 timeouts (0.06%) and 537,948 (95%) times the laser pulse had no influence on the MCU's behavior. Successful glitches were visible in area 1 and 2 as shown in Figure 4.

The second experiment focused specifically on those two areas and was run with the decreased step size of 100 μs as shown in Table 6. The results demonstrated that out of 142,515 experiments, which were done in approximately 7 hours, we located 318 (0.22%) successful glitches.

Parameter	Final value
Glitch power	Random between 10% and 30%
Glitch length	Random between 8ns and 100ns
Glitch delay	Random between 8ns and 800ns
Step size	120μm
Location	Area 1,2

Table 6: Final parameters for authentication bypass experiment.

6.2 Analysis of the results

6.2.1 Instruction modification with ADD loop

In the ADD loop experiment, unused general purpose registers were initialized with well recognizable values. Register r0 had the value of **0xdeadbeef** and register r1 was used for the incremented value. Listing 1 shows the expected output versus values of the most frequently occurring successful glitches. A more complete list can be found in Appendix B, Table 7.

By analyzing the results of the experiment we can prove that modifying instructions with a laser pulse is possible. As shown in Listing 1, there are three results appearing frequently. A value starting with **0xdead** is amongst these frequent results. To obtain this value, either the laser pulse changed multiple bits in register r1 several times or the value of register r0 was loaded into r1.

```
Expected output:
0x00002710

Value of register r0:
0xdeadbeef

Successful glitch:
0xdeadd77f
0xeadc0789
0x00001890
```

Listing 1: The values of most frequently occurring successful glitches of the ADD loop experiment.

The ADD instruction used in ADD loop experiment is **add.w r1, r1 #1**, which means increase the value of the source register (second r1) by 1 and write it to the destination register (first r1). Figure 5 provides an overview of the ADD instruction encoding in ARM [12].

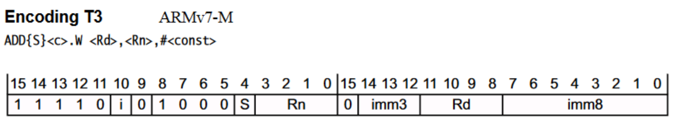


Fig. 5: Overview of the ADD instruction encoding.

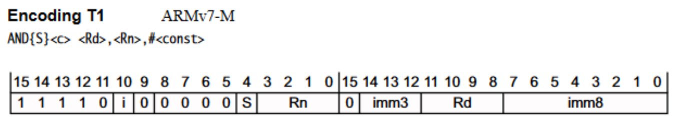
Listing 2 shows the ADD instruction before and after a successful glitch where r0 got loaded into r1. Through the binary representation, it is visible that 1 bit in the source register Rn was modified in order to create the new instruction.

```
ADD instruction in binary:
11110 0 0 1000 0 0001 0 000 0001 00000001

ADD instruction in binary after glitch:
11110 0 0 1000 0 0000 0 000 0001 00000001
```

Listing 2: The ADD instruction presented in binary representation.

Another frequently occurring result was **0x00001890**, which is a counter value lower than the expected **0x00002710**. Since it is not likely all the bits were set to zero by the laser several times, we looked whether this was caused by an instruction. The AND instruction is an instruction which differs only a single bit with the original ADD instruction and it is shown in Figure 6.



Encoding T1 ARMv7-M

AND{S}<c> <Rd>,<Rn>,#<const>

15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0	15	14 13 12	11 10 9 8	7 6 5 4 3 2 1 0
1	1	1	1	0	i	0	0	0	0	0	S	Rn	0	imm3	Rd	imm8

Fig. 6: Overview of the AND instruction encoding.

The AND instruction ”performs a bitwise AND of a register value and an immediate value, and writes the result to the destination register” [12]. This immediate value is comprised of the following fields in the AND instruction: i, imm3 and imm8, as shown in Figure 6. Using the values from the original ADD instruction the immediate value would be **000000000001**. Only the last bit of the value in r1 at that point in time is taken into account for the bitwise AND operation, which means that the value of the counter would be reset to either 0 or 1. Listing 3 shows the AND instruction after the successful glitch.

```
ADD instruction in binary:
11110 0 0 1000 0 0001 0 000 0001 00000001

AND instruction in binary after glitch:
11110 0 0 0000 0 0001 0 000 0001 00000001
```

Listing 3: The comparison of ADD and AND instruction after glitch, presented in binary representation.

Due to a fixed glitch delay, we always injected the laser at the same point in time. Therefore, the counter consistently reached the value of **0x00001890** in every experiment. Because when we subtract **0x00001890** from the most frequently occurring value **0xea dc0789**, we obtain **0xea dbee f9**. This shows that the instruction was modified and the value of register r0 was loaded into register r1, which also happened in the same point of time as the AND instruction change. Moreover, when modifying the value of the glitch delay to a lower one, we discovered that the values of the counter changed consistently to a higher one. This is because glitching earlier meant the counter had a longer time to increase after the AND instruction was executed. The same was true for changing the glitch delay to a higher value, which resulted in a lower value of the counter. This shows we can control the outcome of the experiment.

6.2.2 *Instruction modification with bitwise increment*

Another example of instruction modification comes from the bitwise increment experiment explained in Section 5.1.3. Listing 4 shows the expected outcome and the glitched result, which was returned most frequently. It is visible that the values differ with 1 bit. This means that the injected laser pulse affected the intended behavior of the microcontroller. Since the affected bit is in the 3rd position, the glitched instruction was the ADD operation with value 4 in Listing 5. It is possible that the value in the instruction was modified to another one with value zero, which would leave the bit unchanged.

```
Expected output:
0xff
Expected output in binary:
1111 1111
Successful glitch:
0xfb
Successful glitch in binary:
1111 1011
```

Listing 4: The value of the most frequently occurring successful glitch of the bitwise experiment.

```
...
add.w   r1, r1, #2
add.w   r1, r1, #4
add.w   r1, r1, #8
add.w   r1, r1, #16
...
```

Listing 5: ADD instruction affected by the laser pulse marked in bold, the result is value 0xfb.

The repeatability of this glitch is proven, by changing the order of the executed instructions as shown in Listing 6. In this case, the most frequently occurring glitch was 0xf7, which meant that the ADD instruction with value 8 was modified. This proves that we can modify a specific instruction.

```
...
add.w   r1, r1, #2
add.w   r1, r1, #8
add.w   r1, r1, #4
add.w   r1, r1, #16
...
```

Listing 6: Changed order of the ADD instruction execution, the result is value 0xf7.

6.2.3 *Register value modification*

The integrity of values stored in registers is important for the correct functioning of the microcontroller. In our experiment, we focused on general purpose registers, which were initialized with known values as shown in Listing 7. The expected output of the experiment is **0xfacade00deadbeefcafebabefacefeed**. Although we ran multiple tests for several days, we were not able to modify register values. Instead, we again noticed instruction modification.

```
r0:  fa ca de 00
r4:  ca fe ba be
r5:  fa ce fe ed
r6:  de ad be ef
```

Listing 7: Correct output of the register value modification experiment.

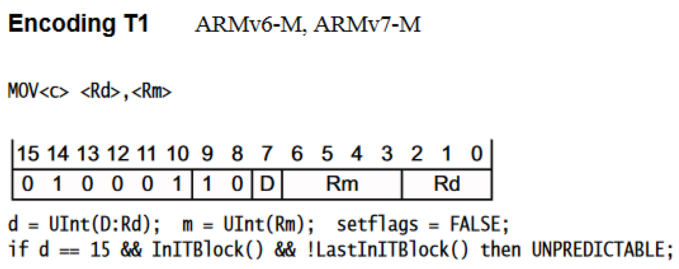
```
0xcade0000deadbeefcafebabefacefeed
0xde000000deadbeefcafebabefacefeed
0x00000000deadbeefcafebabefacefeed
```

Listing 8: Outcome of the register value modification experiment.

Listing 8 clearly shows that the value of **facade00** is shifted to the left. This can be explained by the NOP instructions, which we implemented so that the laser has enough time to introduce a glitch. The operation of the program was too fast and there was no time to glitch. Thus, instruction **mov r1, r1** was used, which is essentially an instruction that does nothing but provides the necessary time.

When analyzing the results, we noticed that this MOV instruction implemented as a NOP was modified into the Linear Shift Left (LSL) instruction. LSL is one of the MOV instructions listed in the ARM manual [12]. The encoding for this instruction is shown in Figure 8. When comparing MOV and LSL instruction, we see that they differ with a single bit. If this bit is changed, the MOV instruction becomes the LSL instruction.

Even though we were able to flip a single bit, it doesn't explain the results we obtained. With one bit flip, we would still have register r1 for source and destination register, instead of register r0. This means that is highly likely we managed to flip 3 bits. The difference between instructions is shown in Figures 7 and 8.



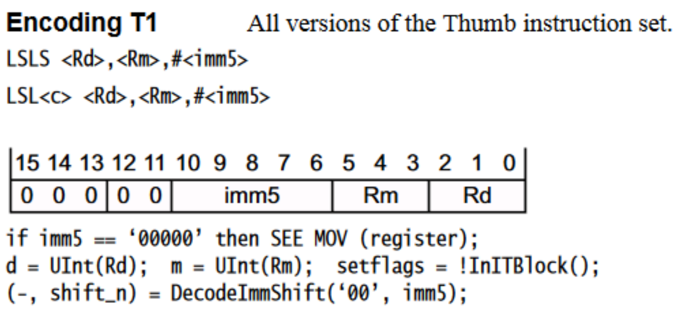
Encoding T1 ARMv6-M, ARMv7-M

MOV<c> <Rd>,<Rm>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	D	Rm				Rd		

```
d = UInt(D:Rd);  m = UInt(Rm);  setflags = FALSE;
if d == 15 && InITBlock() && !LastInITBlock() then UNPREDICTABLE;
```

Fig. 7: Overview of the MOV instruction encoding[12].



Encoding T1 All versions of the Thumb instruction set.

LSLS <Rd>,<Rm>,#<imm5>

LSL<c> <Rd>,<Rm>,#<imm5>

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Rm			Rd		

```
if imm5 == '00000' then SEE MOV (register);
d = UInt(Rd);  m = UInt(Rm);  setflags = !InITBlock();
(-, shift_n) = DecodeImmShift('00', imm5);
```

Fig. 8: Overview of the LSL instruction encoding [12].

This shows that the value for the LSL, with 3 bits flipped, is **11000**. Listing 8 shows that this shift occurs only in one of the results. However, for the other results to occur, up to three additional bits have to be flipped. Further research is needed to better understand these results.

7 Discussion

The results of the experiments show that we are able to alter the behavior of the target by modifying instructions or their arguments with laser pulses. Other researchers will be able to reproduce our results under similar circumstances and with similar equipment.

Obtained results can also be applied to e.g. bypass future authentication mechanisms. If a password check can be compromised by flipping a single bit, other researches can now draw conclusions that the attack is feasible without the need to practically prove it.

However, we were not able to modify register values. Due to the limited time, we did not perform a global scan with a significantly smaller step and spot size, which could have resulted in finding the precise location of the register and modifying it.

The experiments have shown that there are 4 interesting areas on the board as shown in Figure 4. This is where the majority of the successful glitches occured. We suspect that area 1 is where a type of memory is located and areas 3 and 4 is where the CPU resides. However, we have no means to prove this theory and further research is needed. We do not know which component is located in area 2.

In order to mitigate the threats posed by laser fault injection, chip manufacturers can deploy hardware or software countermeasures. Creating a physical barrier such as a metal shield, covering parts of the chip is a way to prevent laser beams from penetrating it. Another countermeasure is the implementation of the photodetectors or light sensors, which can detect scanning laser beams and terminate operation [5] [3]. Furthermore, when designing a PCB it should be taken into account that the system ceases operation when an attacker attempts to drill a hole to access the Cortex-M4. When it comes to the software countermeasures, implementing random delays in checks will increase the difficulty of a successful glitch [5].

8 Conclusion

In order to answer our main research question, we first need to elaborate on the sub-questions. The first one, focused on the ways laser faults can be injected. In our research, we decided to perform the backside laser fault injection over frontside, because it allows for precise targeting of features on the die, once it is exposed. Furthermore, based on the recommendations found in the literature, we used laser pulses in order to target certain areas of the die, instead of continuous wave, which is not precise enough.

The second sub-question was about finding the optimal variables for the laser to introduce faults in to the target. During our research we discovered that many variables for laser fault injection exist and we noticed that each experiment had their own set of optimal variable values, except for the glitch power. When setting the value of this variable between 20 - 25% of the maximum 20W it resulted in the highest amount of successful glitches throughout all the experiments.

The last sub-question was about the type of behavioral changes that can occur in the MCU when injecting laser faults. We set out to test for two: modifying instruction and modifying register values. We noticed that we can reliably modify an instruction and thus change the intended execution of the program with several experiments. In the proven cases this was achieved by changing the value of a bit from 1 to 0. In some cases this effectively changed the entire instruction. The password authentication mechanism implemented on the target was successfully bypassed, by exploiting the possibility to modify instructions.

To answer our main question "What is the security impact of injecting laser glitches into a Cortex-M4 based microcontroller?", we can conclude that the results of the experiments have shown that the target is vulnerable to backside laser fault injection. While instruction modification was proven successful with high success ratio during several experiments, modifying register values was not accomplished in this research. Nevertheless, the consequences of instruction modification showed to be severe enough to state that the security impact of backside laser fault injection on the MCU is high.

9 Future work

As part of the future work, further research into modifying registers is necessary. The experiments could be performed with a different objective magnification e.g. 20x or 50x. Changing the objective narrows down the spot size and allows for more precision when firing the laser.

In this research we have proven to change instructions or their parameters by changing the value of a bit from 1 to 0. It would be interesting to verify whether it is possible to change a 0 to a 1, which could trigger different instructions.

Also, further research could be done into specific functionalities of the Cortex-M4, such as the Read-Data Protection (RDP) or the True Random Number Generator (TRNG). The RDP has three levels of read/write protection and with optical glitching it could be attempted to downgrade the protection level, allowing an attacker to e.g. flash its own code on the target while this was not permitted. The operation of the TRNG could potentially be disrupted with the optical glitching, which could result in producing less random numbers used in cryptographic operations.

Finally, other microcontrollers from the ARM Cortex family could be investigated, which implement more advanced security functionalities such as a Memory Protection Unit (MPU) or a TrustZone and attempt to bypass those with laser fault injection.

10 Acknowledgements

The authors would like to thank Niek Timmers and Nils Wiersma from Riscure for their help and support.

References

[1] Niek Timmers and Cristofaro Mune. *KERNELFAULT: R00ting the Unexploitable using Hardware Fault Injection*. URL: https://www.slideshare.net/MSbluehat/kernelfault-r00ting-the-unexploitable-using-hardware-fault-injection (visited on 07/01/2018).

[2] Hagai Bar-El et al. “The sorcerer's apprentice guide to fault attacks”. In: *Proceedings of the IEEE* 94.2 (2006), pp. 370–382.

[3] Nikolaos Athanasios Anagnostopoulos. “Optical fault injection attacks in smart card chips and an evaluation of countermeasures against them”. MA thesis. University of Twente, 2014.

[4] ARM. *White Paper: Cortex-M for Beginners - An overview of the Arm Cortex-M processor family and comparison*. URL: https://community.arm.com/processors/b/blog/posts/white-paper-cortex-m-for-beginners-an-overview-of-the-arm-cortex-m-processor-family-and-comparison (visited on 31/01/2018).

[5] Jasper GJ Van Woudenberg, Marc F Witteman and Federico Menarini. “Practical optical fault injection on secure microcontrollers”. In: *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2011, pp. 91–99.

[6] Niek Timmers and Cristofaro Mune. “Escalating privileges in Linux using voltage fault injection”. In: *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2017 Workshop on*. IEEE. 2017, pp. 1–8.

[7] Albert Spruyt. “Building fault models for microcontrollers”. In: *University of Amsterdam, Amsterdam, Tech. Rep* (2012), pp. 2011–2012.

[8] James Gratchoff. “Proving the wild jungle jump”. In: (2015).

[9] Nicolas Moro et al. “Electromagnetic fault injection: towards a fault model on a 32-bit microcontroller”. In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2013, pp. 77–88.

[10] Sergei P Skorobogatov and Ross J Anderson. “Optical fault induction attacks”. In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[11] Riscure. *Laser Station 2*. URL: https://www.riscure.com/uploads/2017/07/ls2_datasheet.pdf (visited on 07/01/2018).

[12] ARM. *ARM®v7-M Architecture Reference Manuals*. URL: https://web.eecs.umich.edu/~prabal/teaching/resources/eecs373/ARMv7-M_ARM.pdf (visited on 28/01/2019).

[13] Riscure. *EM Probe Station*. URL: https://www.riscure.com/uploads/2017/07/datasheet_emprobestation.pdf (visited on 05/02/2018).

[14] Riscure. *Spider: A Security Test Tool for Side Channel Analysis and Fault Injection Testing*. URL: https://www.riscure.com/product/spider/ (visited on 07/01/2018).

[15] Riscure. *Glitch Amplifier II: a Security Test Tool for Power Glitches on Embedded Devices*. URL: https://www.riscure.com/product/glitch-amplifier/ (visited on 31/01/2018).

[16] Laser Safety. *Class 4 (IV) laser safety information*. URL: http://www.lasersafetyfacts.com/4/ (visited on 08/01/2018).

[17] ST. *STM32F417IG*. URL: https://www.st.com/en/microcontrollers/stm32f417ig.html (visited on 07/01/2018).

[18] Stephan De Castro et al. “Frontside versus backside laser injection: a comparative study”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.1 (2016), p. 7.

[19] ACEPT W3 Group. *Silicon - a Material Transparent to Infrared*. URL: https://www.asu.edu/courses/phs208/patternsbb/PiN/rdg/silicon/index.htm (visited on 28/01/2019).

[20] Tuba Kiyan, Christof Brillert and Christian Boit. “Timing analysis of scan design integrated circuits using stimulation by an infrared diode laser in externally triggered pulsing condition”. In: *Microelectronics Reliability* 48.8-9 (2008), pp. 1327–1332.

[21] Jasper Hupkens Dominika Rusek. *Laser fault injection test scripts*. URL: https://gitlab.os3.nl/drusek/laser-fault-injection/tree/master (visited on 15/01/2019).

APPENDIX A
CODE IN C AND ARM ASSEMBLY USED IN THE EXPERIMENTS

Full code base can be found in Gitlab [21].

Counter increment

```
GPIOC->BSRRL = GPIO_Pin_2;
while (payload_len) {
    payload_len--;
        upCounter++;
}
GPIOC->BSRRH = GPIO_Pin_2;
```

Listing 9: Counter increment code in C used in the experiment.

Add loop

```
...
add.w    r1, r1, #1
...
```

Listing 10: ADD instruction in ARM assembly used in the experiment.

Bitwise increment

```
...
add.w    r1, r1, #2
add.w    r1, r1, #4
add.w    r1, r1, #8
add.w    r1, r1, #16
add.w    r1, r1, #32
add.w    r1, r1, #64
add.w    r1, r1, #128
...
```

Listing 11: Bitwise increment instructions in ARM assembly used in the experiment.

Modify register values

```
...
mov.w    sl, #0
add.w    sl, sl, #56832   ; 0xde00
add.w    sl, sl, #13238272         ; 0xca0000
add.w    sl, sl, #4194304000       ; 0xfa000000
mov.w    r6, #239          ; 0xef
add.w    r6, r6, #48640   ; 0xbe00
add.w    r6, r6, #11337728         ; 0xad0000
add.w    r6, r6, #3724541952       ; 0xde000000
mov.w    r4, #190          ; 0xbe
add.w    r4, r4, #47616   ; 0xba00
add.w    r4, r4, #16646144         ; 0xfe0000
add.w    r4, r4, #3388997632       ; 0xca000000
mov.w    r5, #237          ; 0xed
add.w    r5, r5, #65024   ; 0xfe00
...
```

Listing 12: ARM assembly code used in the modify register value experiment.

Authentication bypass

```
...
for (i = 0;i < 4; i++) {
if (rxBuffer[i] == password[i]) {
charsOK = charsOK + 1;
}
...
```

Listing 13: C code used in the authentication bypass experiment.

APPENDIX B
THE VALUES OF THE MOST FREQUENT SUCCESSFUL GLITCHES

Count	Value
82	01 01f1 01
72	ea dc07 94
70	00 0019 22
57	00 0019 21
46	00 0026 83
32	de add8 1a
31	00 0019 20
30	08 0047 95
28	de add8 1d
27	00 0018 a5
23	ea dc07 91
19	00 0018 a2
19	00 0018 94
17	ea dc07 96
16	00 0026 80
15	ea dc07 92
15	00 0027 00
14	00 0018 a6
13	00 0019 25
11	00 0018 a1
10	00 0026 84
10	00 0018 a4
8	f1 0101 01
8	de add8 1c
8	00 0026 ff
8	00 0019 1f
8	00 0018 9d

Table 7: ADD loop experiment. The values of the most frequent successful glitches.

Count	Value
939	00 0000 fb
158	00 0000 fc
138	00 0000 f8
22	00 0001 1f
11	00 0001 a9
10	79 0000 fc
8	00 0001 9f
5	ae 00ae fb
5	79 0000 f8
5	00 0001 a1

Table 8: Bitwise increment experiment. The values of the most frequent successful glitches.

Count	Value
10537	00 0000 00de adbe efca feba befa cefe ed
360	f5 95bc 00de adbe efca feba befa cefe ed
22	da 0000 00de adbe efca feba befa cefe ed
21	fa cefe edde adbe efca feba befa cefe ed
13	ed 0000 00de adbe efca feba befa cefe ed
13	95 bc00 00de adbe efca feba befa cefe ed

Table 9: Modify register values experiment. The values of the most frequent successful glitches.