

Safety \neq Security

A security assessment of state of the art ASIL-D certified microcontrollers

(extended abstract)

Abstract—insert abstract here

Keywords—insert keywords here

TODO

use of first person consistency

present / past tense consistency in sections discussing results

fi parameter plots

fi number of attempts tables

Microcontroller units (MCU) form the foundation of the modern car. They are used to implement functionalities that can be as simple as opening the window when a button is pressed, or as complicated as Vehicle to Vehicle communications and autonomous driving systems. They can perform non-critical tasks such as controlling the windscreen wipers or extremely critical ones like making sure airbags are deployed in case of emergency.

Over the last two decades the number of electronic controllers in vehicles has increased, making evident the need of ensuring the reliability and safety of these MCUs. The ISO 26262 standard [6] – in particular part 10, Annex. A and part 5, Annex. D – recognizes the pivotal part that MCUs play in the ecosystem of a vehicle and establishes a framework for the development of safe automotive electronic systems. Part 5 of the standard gives a number of examples of safety mechanisms that can be implemented to mitigate different electric faults that could affect the operation of an electronic system and threaten the safety of the vehicle and passengers. Although these examples and recommendations intend to protect against faults occurring during the normal use of the vehicle, some of these safety measures are also often recommended in the security literature as countermeasures against fault injection (FI) attacks (eg. Bar-El et al. [1]). It is therefore easily taken for granted that MCUs implementing ISO 26262 recommendations (i.e. ASIL chips) are also protected against FI attacks.

Our presentation summarizes our investigations in assessing the effectiveness of the ISO 26262's recommendations to resist against FI. We first compare the resistance to FI of two ASIL-D certified MCUs with a standard (non-ASIL) automotive MCU in a controlled test environment. Next, we explore the risks of FI attacks on ASIL-D chips by disabling the debug interface protection mechanism in our targets. Finally, we discuss the conse-

quences and threats these attacks pose in the automotive context and provide recommendations for mitigating the aforementioned threats.

I. AUTOMOTIVE CONTEXT, ISO 26262 AND FAULT INJECTION

TABLE I
ASIL CATEGORIES

Category	QM	ASIL-A	ASIL-B	ASIL-C	ASIL-D
Assurance level	Low				High

Automotive standards and literature, especially with regards to hardware, primarily deal with safety. In this context, faults are considered as an unintended, abnormal and often random condition caused by the hostile environment in which the hardware resides (e.g. high temperatures, noisy power supply, vibrations, electromagnetic (EM) pulses produced by coils, etc). ISO 26262 provides recommendations to mitigate and minimize the impact of these unintentional faults and defines the Automotive Safety Integrity Level (ASIL) risk classification scheme from level A through D. ASIL-D complying MCUs are required to adhere to the most stringent requirements on safety and fault-tolerance in order to achieve absence of unreasonable risk. A fifth level, QM, is defined for parts that only require basic quality management. Fault injection is mentioned in the ISO 26262 only as a method of testing the compliance of the ASIL requirements.

When shifting to the domain of security, faults are no longer the effect of random and unintended events. They are the result of *glitches* in the environment that are deliberately placed, precisely timed and carefully tuned by an attacker with the purpose causing unintended behavior – the *fault* – which is leveraged to compromise the chip security. FI attacks are in general used to alter the program flow or the processed data. If this is possible, an attacker could, for example, bypass the Secure Boot process by skipping the signature verification and executing unauthenticated code [11], or recover cryptographic keys using a Differential Fault Analysis (DFA) attack [2, 3, 5].

For this work, two techniques of injecting faults have been considered: voltage glitching – a fast variation of the power supply voltage – and EM glitching – a strong and localized EM pulse. A voltage glitching device is cheap (tools can be bought for even less than 200€, do-it-yourself solutions are even cheaper than that) but the attack is not localized, as it affects the entire chip, and requires modifying the target board by removing the capacitors. EM glitching on the other hand requires no modifications on the board and the glitches are local (i.e. they affect a small area of the chip) but the equipment required is far more expensive.

II. TARGETS

To investigate the effectiveness of safety mechanisms as countermeasures against fault injection attacks, two ASIL-D certified MCUs that implement these mechanisms have been selected, from two different manufacturers. Although we are convinced that we would have obtained similar results with any other ASIL-D MCU in the market, we omit the chip models because we are still in the process of disclosing the findings to the manufacturers in a coordinated manner. The first target, codename ASILD1, implements an ARM Cortex-R4 core and the second target, codename ASILD2, a PowerPC e200. These two targets were selected because they represent two of the most prevalent architectures in the safety electronic industry. For reference, a non-ASIL target has been chosen, code-named QM1. This target implements an ARM Cortex-M0 and can be used for applications which require only the basic QM rating.

From all the safety mechanisms implemented in ASIL-D MCUs, we are only interested in investigating the ones that have an effect on transient faults as they could also mitigate the glitches used by an FI attacker. In both selected targets these mechanisms include a dual core CPU in lockstep configuration (or ‘Simple Time Redundancy with Comparison’) and memories with error correction codes (ECC) parity bits, as recommended by ISO 26262 part 5.

III. SETUP

A. FI tools

Commercially available hardware [9] and software [10] from is used for injecting glitches. The central piece to the hardware setup is the VCGLitcher, an FPGA powered device capable of generating voltage glitches with 2ns of precision, or generating input to an EM pulse emitter device. During the characterization experiments where the target is fully under control, a GPIO pin of the target is used to generate a signal that triggers the VCGLitcher. In the experiments where the target is not controlled, the icWaves tool is used instead. This device analyzes the power consumption of the target and generates a trigger signal when a programmable reference pattern is detected. The setup is controlled on a Windows host machine running the Inspector FI software.

B. Target preparations

Before running the experiments, the hardware and software of the targets needs to be prepared.

Standard low-cost development boards are used as target for the three MCUs under test. For voltage glitching, the power traces of the PCevB are modified if needed. The purpose of this modification is to isolate the MCU power supply from the rest of the circuit, so a voltage glitch affects only to the MCU and not to other parts. Additionally, the decoupling capacitors of the MCU are removed as they affect the shape of the voltage glitch. For EM glitching, no modifications to the boards are needed.

In order to determine the effect of the glitch, in particular whether or not it caused a useful fault, MCUs are programmed with software that outputs the contents of several internal registers over a UART communication line. This is especially useful in determining whether or not a particular safety mechanism has been triggered.

Besides the more theoretical characterization experiments a more realistic investigation of the targets discussed in this work as well, by investigating the debugging interface present in these targets is attacked. In order to do this investigation, these targets first have to be locked. This is done by programming one or more values to an OTP section in its memory, depending on how the locking mechanism works in a specific target.

C. Glitch parameters

The effectiveness of a glitch depends on the moment in time when the glitch is injected and the shape of the glitch. These can be modeled through parameters like the glitch length, glitch intensity (voltage or EM field) and time offset from the trigger signal. In the case of EM glitching, the point in the chip where the EM field is applied is also important.

The methodology used for finding the optimal parameters - that is, those with the highest success rate - involves running multiple FI campaigns. In each campaign, thousands of glitches are attempted varying the parameters within certain range. The initial FI campaign runs on very broad parameters. The parameters of the successful glitches found are used to restrict the search range in the next campaign and after few iterations, the optimal parameters are found.

A glitch can have various effects on a target, depending on the aforementioned parameters. A glitch is considered successful when it causes a fault in the target that results in the behavior that the attacker is looking for. A glitch can also results in crashes or unexpected behaviour. In this work, special attention is paid to glitches that are detected or not by the safety mechanisms under investigation. When a glitch is mentioned as detected,

it means that it was detected by one or more of these mechanisms. Classification is based on internal error registers and externally observable error output pins.

IV. CHARACTERIZATION

The general effectiveness of the safety mechanisms in preventing FI is investigated in a characterization test which involves two experiments in a controlled environment. In both experiments a customized trigger signal is generated by the firmware on the target to indicate to the setup when to inject a glitch.

A. Scenario 1 – unroll

In the first experiment, *unroll*, a sequence of increment instructions is performed on the same CPU register. This should produce a predictable final value in the counter register. The final value of the register is sent over a serial connection at the end of the sequence together with the internal state of the MCU and its fault detectors. A pseudo code version of this setup is given in Listing 1.

Listing 1. *unroll* scenario pseudo code

```
trigger_up()
asm(add r1 #1)
asm(add r1 #1)
...
asm(add r1 #1)
asm(add r1 #1)
trigger_down()
send_serial(r1)
```

A successful glitch affects the value of *r1* by altering the content of the register itself or by changing the code flow and skipping some increment instructions, without triggering the fault detectors.

B. Scenario 2 – auth

unroll is one of the most basic experiments that can be performed and a good starting point for characterizing MCUs. The goal of the test is to confirm that the target is sensitive to FI. If successful glitches are found, additional experimentation should be done involving more realistic scenarios. The additional experiment chosen in this work is a conditional branch setup – *auth*. In this experiment an *if*-statement determines the program flow, simulating a situation where some authentication check is performed, as shown in Listing 2.

Listing 2. *auth* scenario pseudo code

```
flag = 1
...
trigger_up()
if (flag == 0):
    send_serial_authenticated()
else
    send_serial_denied()
trigger_down()
```

A successful glitch affects the *if*-statement in such a way that the unintended branch – *send_serial_authenticated()* – is executed.

C. Results

TABLE II
AN OVERVIEW OF THE SUCCESS RATES OF CHARACTERIZATION EXPERIMENTS

	unroll		auth	
	Power	EM	Power	EM
ASILD1	87%	0.2%	60%	0.2%
ASILD2	0%	18%	N/A	57%

The previous two experiments were run in different FI campaigns during three weeks. A total of ~720,000 and ~550,000 glitches were injected in ASILD1 and ASILD2 respectively.

The final results of the characterization part of this work are given in Table II. These percentages represent the rates of successful and undetected glitches obtained for the last FI campaign of each characterization experiment, after finding and using the most effective glitch parameters. Around 10,000 glitches were injected in each of these last FI campaigns. Other glitches - not reflected in this table - were able to affect successfully the program execution but were detected by the MCU safety mechanisms.

For ASILD1, EM glitching was attempted for the only purpose of verifying its feasibility. The EM FI campaigns did not attempt to find the optimal parameters. For example, the entire chip surface is scanned while obtaining these results, rather than a specific sensitive point. Therefore only a 0.2% of success rate was achieved. If the optimal parameters are found and used, it is reasonable to expect a success rate similar to voltage glitching.

On ASILD2, voltage glitching proved to be ineffective in the *unroll* experiment. We find a possible explanation in the internal power distribution and the on-chip voltage regulator that could be filtering the glitches. We did not try voltage glitching in the following ASILD2 experiments, limiting them to EM glitching. Furthermore, after careful configuration of the target none of our glitches were reported as detected. A plausible explanation for this is that detected faults immediately result in a target reset, not leaving enough time to investigate these channels to determine what exactly happened. Note that the results reported in Table II were achieved with parameters that did not cause any resets, ensuring that if the detected set of the glitches is contained in the reset category, none of the glitches were detected.

Analyzing the information collected during the ASILD1 campaigns, we find that the most effective detection mechanism is the lockstep output comparison, which was triggered in 84% to 98% of the detected glitches, varying from experiment to experiment. Flash and RAM parity errors were triggered in 18% to 25%


of the detected glitches. Other safety mechanism were triggered in less than 3.5% of the detected glitches. Note that a glitch can trigger many detection mechanisms at the same time. ASILD2 did not

The primary conclusion to be drawn from these characterization experiments is that it is possible to find glitch parameters, for both ASILD1 and ASILD2, which result in a relatively high rate of successful and undetected glitches. Contrary to expectations, all the previous experiments used a single glitch to inject the same fault in both lockstep CPUs. Moreover, when using two glitches separated by the same number of clock cycles as the separation of the lockstep execution pipelines, the success rate decreases drastically.

V. BREAKING JTAG PROTECTION


TABLE III
AN OVERVIEW OF THE SUCCESS RATES OF THE DIFFERENT JTAG EXPERIMENTS

password QM1	password ASILD1	records ASILD2	life cycle ASILD2
80%	1.3%	0.5%	XX%

Previously discussed characterization experiments show that it is possible to use FI on ASIL-D MCUs to affect the code execution flow and bypass instructions, even if lockstep processing units and ECC protected memories are used. These experiments are run in a white-box manner in a controlled environment with full control over the software, including a trigger signal. Real attacks are normally executed in a black-box manner where the attacker has no control over the CPU and no access to the code. In order to have a better understanding of the risks that FI attacks pose on a real application, a different type of FI experiment  prepared that better [represents/simulates/emulates] a non-controlled environment. As debug interface access is one of the first assets that an attacker tries to obtain, the resistance of the debug interface protection mechanisms present in our targets against FI attacks is assessed.

Most modern MCUs have a permanent protection for closing the debug interfaces. This protection is usually enabled at the end of the manufacturing process of the embedded system in order to protect the firmware contained in the MCU. It is a common practice for many semiconductor manufacturers to implement this debug interface protection in the ROM code executed at boot time, before the application firmware is executed. This ROM usually reads the protection configuration from an one-time programmable (OTP) memory section or a Flash word and configures the registers of the debug interface module accordingly. An attacker using FI to bypass the configuration read or the debug module configuration stage effectively unlocks the debug interface on a protected device, which is regarded here as successful.

Because the ROM is immutable, the boot process firmware cannot be modified to generate a trigger for the glitching device. This makes a successful fault injection attack significantly harder, as an important parameter that has to be tuned. The attacker has to rely purely on measuring the time after releasing the reset to inject the glitch. Some MCU datasheets give some general information about the boot process, but the precise timing information required to trigger the glitch is typically missing. Additionally, time between release of the reset and the area in which a glitch is needed is typically not time-constant. If ROM code is obtainable, the attacker can reverse it to determine exactly when the debug protection mechanism is configured and when to inject the fault, but normally the ROM is proprietary and not accessible. Injecting glitches randomly at boot time is possible when the boot-up time is short – less than ~10,000 clock cycles from the reset to the user application execution – but is not feasible when more time is required, as is the case with the tested targets. As an alternative, side channel analysis can be used to determine when the debug protection mechanism is configured.

We observe that by comparing the power consumption traces of the boot process in a protected and an unprotected chip, an attacker can find the moment in time where the debug interface protection is enabled and get the timing information needed for injecting a glitch. In this paper this identification technique and its results  are presented for ASILD1 and ASILD2 [and QM1].

A full attack – power analysis to find the timing and injecting voltage glitches to unprotect the JTAG debug interface – is run on targets ASILD1 and QM1. A slightly incomplete attack that does include power analysis to find vulnerable points and some preliminary successful fault injection results are presented in this section for the ASILD2 target. However, this target requires additional investigation to comfortably claim that the JTAG protection can be broken.

Note that of all the debug interfaces supported by our targets – JTAG, SWD, UART, CAN – we choose attacking JTAG because it is the most common one. All descriptions of how specific targets implement their protection mechanisms below are based only on information found in publicly available documentation and resources.

A. ASILD1

The debugging interface in ASILD1 can be locked by programming a 128-bit value to a certain OTP area. This value goes through some internal logic both implemented in hardware and software upon which the decision is made to put the interface in a locked state or to keep it in an unlocked state. By profiling power consumption during boot time in both configurations and comparing the two, a small section of this time frame shows a distinct difference in power consumed, as shown in Figure 1.

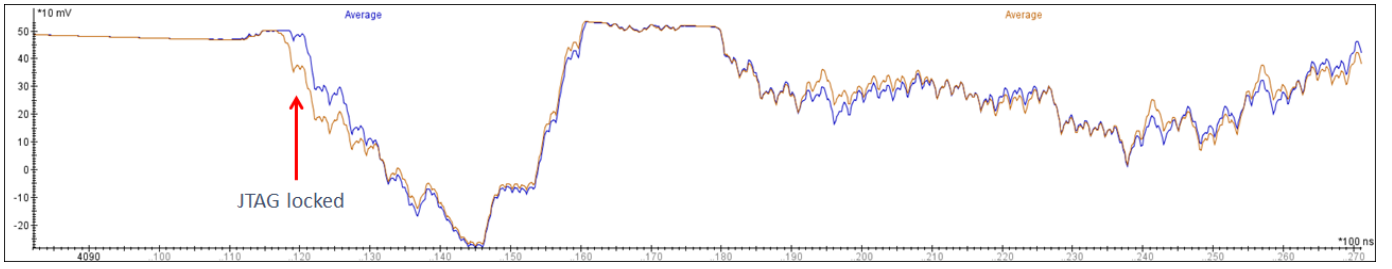


Fig. 1. Difference between locked chip (brown) and unlocked chip (blue) for ASILD1

~259,000 glitches were injected in total for a broad range of different glitch parameters that include this 200ns window, resulting in three successful faults. After tuning the glitch, 35,200 glitches with the optimal parameters were injected. 1.3% (486) of the attempts succeeded to keep the JTAG unprotected without triggering any lockstep or memory faults. An additional 1.2% of the glitches succeeded to keep the JTAG unprotected but were detected by the safety mechanisms. Finally, 3.2% of the attempts had no apparent effects other than triggering the fault detector mechanisms, bringing the total amount of detected faults in this setting to 4.4%. This means that the ASILD1 target can be unprotected in less than one minute and half approximately, after performing the preparatory profiling and tuning steps.

B. ASILD2

ASILD2 has several configuration parameters that determine whether its debugging interface is locked or not:

- 1) Life cycle state – This state indicates whether the target is still in production, or in the field. Early states bypass all other protection mechanisms listed below.
- 2) Censorship state – This is a binary state, where censored blocks access to the debugging interface and uncensored does the opposite.
- 3) Four debug lock bits – These bits control access to the different JTAG clients.
- 4) JTAG password – The reference password against which a given password is compared to unlocked a locked target.

Only when all four of these configuration parameters are set to their locked state is the interface properly protected. 2 and 3 are configured indirectly through values (records) programmed sequentially to an OTP section of memory, which can be overwritten by programming additional records to this section. These records are loaded during the boot sequence and later verified as well. By default values 2 and 3 are set to “censored” and “locked”, so in order for them to be successfully attacked through these records, records have to be set that first set them to open and then later records that set them to closed again. This requirement makes this attack fairly configuration specific.

By investigating the power consumption under a configuration with two, 102 and 120 of such records, two

points show a very clear dependency on the amount of values programmed, likely corresponding to the load and verify operations. The first point is shown in Figure 2, the second in Figure 3. We also observe that timing between these two points changes depending on the amount of records programmed. This is not visible from the figures shown.

As characterization attempts with power glitching were unsuccessful, only EM glitching is used at this stage. After scanning the entire chip surface by injecting XXX,000 glitches, YY% was successful in corrupting the configuration process during which the records are loaded from OTP and sent to their destination modules. In our experiments we were only able to ZZZ. A very small amount of the injected faults, AABBBBB, was reported as causing a configuration error.

Item 1, the life cycle state, is configured directly by specific OTP section. By programming certain values to certain addresses, the state can be advanced from unrestrictive production states to more restrictive in-field states. Being able to successfully influence the state that is configured during boot offers a much more powerful attack, as it is not hindered by the very circumstantial configuration requirements needed for attacking values 2 and 3.

Preliminary investigation into this attack method reveal that changing this life cycle state clearly affects power consumption during the boot sequence. However, we were not able to convert this information into a successful attack, so further investigation into this is required.

C. QM1

[SETUP TBA]

For comparison purposes, we repeated the experiment on the target QM1 with ~89,000 glitching attempts in total. Using the optimal glitch parameters, 80% of the ~10,800 attempts succeeded in unprotected the JTAG interface, which means that an attack take less than 5 seconds to succeed.

D. Stray conclusion

It is not possible to directly compare results on ASILD1 and QM1 because both chips are implemented with

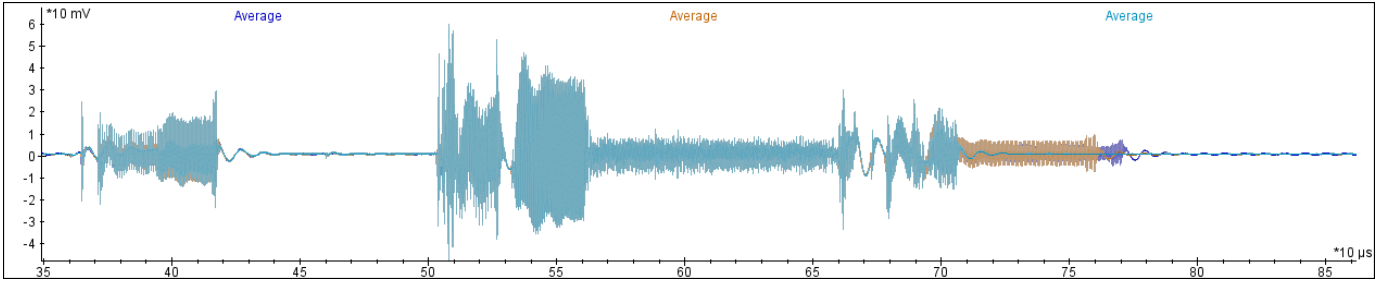


Fig. 2. Difference between two records (cyan), 102 records (brown) and 120 records (blue) programmed for ASILD2, point one: load values

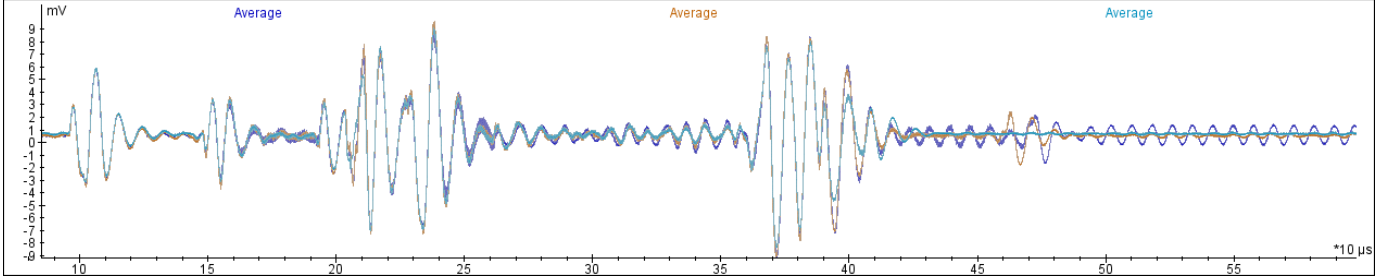


Fig. 3. Difference between two records (cyan), 102 records (brown) and 120 records (blue) programmed for ASILD2, point two: verify values

different technologies which could have different sensitivity to FI attacks. However, the big gap between success rates indicates that some of the ASIL-D safety mechanisms could slightly mitigate the FI attacks, but cannot fully prevent them.

VI. SECURITY CONSEQUENCES

The experiments described in section IV and section V prove that FI attacks can bypass the detection mechanisms outlined above in state of the art ASIL-D certified MCUs. Although the safety mechanisms detect some glitches, they are not good enough to prevent a FI attack.

The fact that it is possible to bypass instructions or execute an unintended branch means that the integrity of the code execution cannot be assured. Any security feature enabled and any check executed in the software can be manipulated by the attacker as previous step toward gaining access to assets. Various assets can be targeted, depending on the attacker profile and the attack scenario. The most common assets include firmware extraction, firmware manipulation, runtime control and access to debug interfaces.

The firmware contained in MCUs is usually the most valuable secret of Tier1 manufacturers and OEMs. The firmware implements those functionalities that add value to the product and differentiate the brand from the competitors. Taking for example a brake controller assistant or an autonomous driving system, there are thousands of hours invested in research and development of the perfect algorithms for stopping the car safely or driving it autonomously through the city. A competitor extracting the software of an ECU would have access to those proprietary technologies and could copy them, or use them to improve their own technology

and gain an unfair competitive edge. Extracting the software can also be an intermediate step for achieving a bigger objective from a security point of view. An attacker could reverse engineer the extracted software to understand its operation and find a vulnerability. This has been done several times in the past when researchers [4, 7, 12] reversed the firmware of the remote key entry systems and immobilizers. Vulnerabilities in the cryptographic algorithms were found, which allowed cloning the keys to open and start the car. Similarly, researchers [8] reversed the extracted software to find remotely exploitable vulnerabilities which gave control of the car without even needing physical access to it.

Modifying the firmware is another common objective of attackers. Typically this attacker profile includes car tuners who seek to modify the performance of their vehicles by altering the software. These persons normally are not aware of the risk that those unlawful modifications could pose to their and others lives. A less common profile could be subjects that are purposely looking to inflict damage in the vehicle or to its occupants by the means of sabotage. In both cases, the possibility of altering the MCU firmware without any tamper evidence is appealing and it should be prevented by the manufacturers.

Runtime control and access to debug interfaces are usually intermediate steps that are necessary for gaining any of the assets mentioned. For example, the attack executed in section V for gaining JTAG access could be used to extract and modify the firmware in the MCU. An additional benefit of gaining runtime control or debug access is the possibility of experimenting and debugging a live target, which could help to reverse and understand the entire system. Runtime control can be gained using

FI through the means of accessing the debug interface, bypassing the Secure Boot or using more advanced techniques like manipulating the CPU's program counter.

Finally, the goal of an FI attack could be simply to disable a security feature. For example, this could be the case for the immobilizer that verifies the response of the key in order to allow starting the engine. By using FI, the attacker could bypass the verification check and start the car without the legitimate key.

VII. RECOMMENDATIONS

As described before, safety mechanisms implemented in ASIL-D chips can reduce the success rate of FI attacks but not fully prevent them. In the absence of other hardware countermeasures against fault injection, software mitigations like execution flow control, redundancy or random delays should be implemented. The reader can refer to the extensive documentation detailing these countermeasures.

It is not only responsibility of the embedded system developers (e.g. Tier1 manufacturers) to implement these countermeasures in their software. As explained in section V, ROM code can also be the target of FI attacks and therefore, System on Chip (SoC) manufacturers should also implement countermeasures when developing ROM code.

When using a MCU with a vulnerable ROM, embedded system developers can mitigate the risk by using the ROM patching mechanism, if such a mechanism is present. For example, ARM Cortex MCUs usually implement the Flash Patch and Breakpoint Unit (FPB) which can be used to patch ROM code. Depending on the MCU manufacturer, the FPB can be enabled at reset time and patch the boot routine, or can only be enabled after booting and patch only the ROM drivers used in the application. In the experiment described in section V, 1.2% of the glitching attempts succeeded to unprotect the debug interface protection mechanism, but lockstep or memory errors were detected. The application developers should check for these errors or any other unexpected state (e.g. unexpected values in a JTAG configuration register) immediately after booting and reset the chip if an inconsistency is found.



that, compared to standard non-ASIL MCUs, the safety mechanisms implemented in ASIL-D chips can help to mitigate FI attacks, but not fully prevent them, nor can they reduce the risk FI attacks pose to an acceptable level. We advise MCU manufacturers and embedded system developers to implement both hardware and software countermeasures against fault injection.

VIII. CONCLUSION

We assessed the resistance of two ASIL-D MCUs against FI attacks. We succeeded in identifying and attacking the JTAG protection by using power analysis and voltage glitching in one ASIL-D target and extracted secret information from the firmware. We succeeded in identifying the same on another ASIL-D target and obtained promising results in attacking this point with EM glitching, but have not yet been able to successfully attack this target in the same manner. We observed

REFERENCES

- [1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer’s Apprentice Guide to Fault Attacks. *Proceedings of the IEEE*, 94(2):370–382, February 2006. ISSN 0018-9219. doi: 10.1109/JPROC.2005.862424.
- [2] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Annual International Cryptology Conference*, pages 131–146. Springer, 2000.
- [3] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of cryptology*, 14(2):101–119, 2001.
- [4] Flavio D. Garcia, David Oswald, Timo Kasper, and Pierre Pavlidès. Lock It and Still Lose It—On the (In) Security of Automotive Remote Keyless Entry Systems. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.
- [5] Christophe Giraud. Dfa on aes. In *International Conference on Advanced Encryption Standard*, pages 27–41. Springer, 2004.
- [6] ISO26262. ISO 26262, Road vehicles — Functional safety. <https://www.iso.org/obp/ui/#iso:std:iso:26262:-1:ed-1:v1:en:term:1.134>.
- [7] Markus Kasper, Timo Kasper, Amir Moradi, and Christof Paar. Breaking KeeLoq in a flash: On extracting keys at lightning speed. In *International Conference on Cryptology in Africa*, pages 403–420. Springer, 2009.
- [8] Charlie Miller and Chris Valasek. Remote exploitation of an unaltered passenger vehicle. *Black Hat USA*, 2015, 2015.
- [9] Riscure B.V. Hardware. <https://www.riscure.com/security-tools/hardware/>, .
- [10] Riscure B.V. Inspector FI. <https://www.riscure.com/security-tools/inspector-fi/>, .
- [11] Niek Timmers, Albert Spruyt, and Marc Witteman. Controlling PC on ARM Using Fault Injection. In *Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop On*, pages 25–35. IEEE, 2016.
- [12] Roel Verdult, Flavio D. Garcia, and Baris Ege. Dismantling megamos crypto: Wirelessly lockpicking a vehicle Immobilizer. In *Supplement to the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 703–718, 2015.