applied sciences

MDPI

Article

Practical, Low-Cost Fault Injection Attacks on Personal Smart Devices

Shaked Delarea [1] and Yossi Oren [1,*]

[1] Department of Software and Information Systems Engineering, Faculty of Engineering Sciences, Ben Gurion University of the Negev, Beer-Sheva 8410501, Israel

* Correspondence: yos@bgu.ac.il

Abstract: Fault attacks are traditionally considered under a threat model that assumes the device under test is in the possession of the attacker. We propose a variation on this model. In our model, the attacker integrates a fault injection circuit into a malicious field-replaceable unit, or FRU, which is later placed by the victim in close proximity to their own device. Examples of devices which incorporate FRUs include interface cards in routers, touch screens and sensor assemblies in mobile phones, ink cartridges in printers, batteries in health sensors, and so on. FRUs are often installed by after-market repair technicians without properly verifying their authenticity, and previous works have shown they can be used as vectors for various attacks on the privacy and integrity of smart devices. We design and implement a low-cost fault injection circuit suitable for placement inside a malicious FRU, and show how it can be used to practically extract secrets from a privileged system process through a combined hardware-software approach, even if the attacker software application only has user-level permissions. Our prototype produces highly effective and repeatable attacks, despite its cost being several orders of magnitude less than that of commonly used fault injection analysis lab setups. This threat model allows fault attacks to be carried out remotely, even if the device under test is in the hands of the victim. Considered together with recent advances in software-only fault attacks, we argue that resistance to fault attacks should be built into additional classes of devices.

Keywords: fault injection; fault injection attacks; hardware attacks; cryptography

Citation: Delerea, S.; Oren, Y. Practical, Low-Cost Fault Injection Attacks on Personal Smart Devices. *Appl. Sci.* **2022**, *1*, 0. https://doi.org/

Academic Editor: Guy Gogniat; Vianney Lapotre; Maria Mushtaq

Received: 7 December 2021
Accepted: 30 December 2021
Published:

1. Introduction

Fault injection attacks (FIA) are physical interventions that exploit the circuit's direct implementation [1]. In contrast to exploitation of software vulnerabilities, FIA attacks usually require the attacker to have physical access to the victim's device. Because of that, countermeasures are often considered for application in specific fields that are required to operate in a challenging environment such as space and automotive, and for specific low-power high-performance applications [2,3].

While fault attacks are known to be much more effective than software-only attacks in their ability to corrupt a device's ordinary execution flow [4], setting up an environment for injecting transient faults is a complex process, which includes the usage of expensive equipment such as oscilloscopes, XYZ stages, high-end pulse generators and amplifiers [5–7]. Defense from these attacks was therefore considered out of scope for many devices that are not expected to be subjected to such intensive physical intervention.

In this work we argue that fault attack countermeasures need to be deployed in more settings. We demonstrate a novel method for performing fault attacks outside the lab. Our key motivating factor is the existence of a large and unregulated market for hardware field-replaceable units (FRUs). Examples of devices that incorporate FRUs include interface cards in routers, touch screens and sensor assemblies in mobile phones, ink cartridges in printers, batteries in health sensors, and so on. As noted by Shwartz et al. [8], third-party FRU installations often use cheap components of unknown pedigree, and thus may introduce,

knowingly or unknowingly, counterfeit or malicious components into otherwise secure devices. Shwartz et al. divided attacks based on malicious FRUs into two different classes: **first-order attacks**, in which the malicious FRU falsifies interactions with the device in the ways a standard user would, but without the user's consent, and **second-order attacks**, which go beyond exchanging properly formed data, and attempt to cause a malfunction in the device driver and compromise the operating system kernel. Our work considers a new form of **third-order attacks**, which do not rely on data exchange between the FRU and the device at all, but instead exploit its physical proximity to launch fault injection attacks.

Specifically, in this paper we make the following contributions:

1. We propose a new attack model for fault attacks, in which the adversary triggers a fault on the device under test (DUT) using a malicious FRU.
2. We design and build an EM fault injection device based on inexpensive components, and show how this injection device can disrupt the normal course of execution of a processor commonly used in smart devices.
3. We present a proof-of-concept attack, in which a fault generated by our injection device leaks confidential information from a high-privileged service in the device under test. Specifically, we show how to extract a private key used by a privileged signing service running in Linux, using the default PyCrypto library, by combining our fault injection method with attack code running with regular user permissions.
4. We discuss various countermeasures which can be used against our attack, and present a software-only approach which is capable of reliably stopping our proof-of-concept attack. This software-based countermeasure was submitted to the authors of the PyCrypto library.

Our chosen attack model offers a unique combination of challenges and advantages. First, we note that our attacker device must be small and inexpensive enough to be mass-produced, and power-efficient enough to run on the limited power provided to the FRU by the host device. Second, we cannot apply any sample preparation steps to the device under test (DUT) such as thinning, flipping or decapsulation. In terms of advantages, we note that the FRU is placed in very close proximity to the device under test, often inside the case and its associated EM shielding. This allows the attacker precise control over the location of the attack, and can also be used to provide trigger information for precisely timing the attack, as we further discuss in Section 4.4.

The main novelty of our work is demonstrating further evidence that shows that active fault injection attacks must be considered for additional classes of devices, both inside and outside of the laboratory environment. This is also the primary research gap between this work and other existing works focusing on fault injection attacks, which are taking the environment in which fault injections are conducted to outside of the lab.

2. Methods

2.1. Fault Generating Device

The purpose of the EM fault injector is to generate a momentary high magnetic field, which affects the registers of the CPU and causes some of the bits to flip, resulting in a fault [1]. To create such a magnetic field, we pass a high current pulse through a copper coil. We do so by connecting the coil to a spark gap and generating a high DC voltage that allows an arc to form and current to flow through the gap. We could also add a ferrite core to the coil in order to increase the intensity of the magnetic field it creates, but we found that using a simple coiled wire was sufficient in producing the desired result. The mathematical model behind electromagnetic fault injections is thoroughly covered in [9].

In order to create a spark through air, we need to reach a very high voltage for a short duration. We achieve this by using an adaptation of a simple commercial mosquito killer, as described in [10]. The circuit consists of a DC to AC oscillator, commonly referred to as a joule thief, and a voltage multiplier. We connected the multiplier's output to a coil, followed by a gap of 3 mm between two electrodes. The electrodes were made of exposed electric wire.

Since the circuit involves very high DC voltage at its output, we took a precautionary measure and controlled its operation using an optical isolator (OI). That way, the spark gap circuit works only when a control signal is given, which allows the current flow at the output of the isolator. The entire circuit is shown in Figure 1. The left block is the OI unit, which, when a control signal is applied, allows current to flow to the EM fault injector and start the charge–discharge cycle, and turns on the LED to signal the circuit is activated. The oscillator block, which is a joule thief circuit, then generates an AC square wave, which is the input to the voltage multiplier block that charges the capacitor at the output until a high enough voltage causes the gap to be bridged and a peak current to flow through the coil. The entire circuit, including the spark gap generator, fits into a small plastic case, as shown in Figure 2.

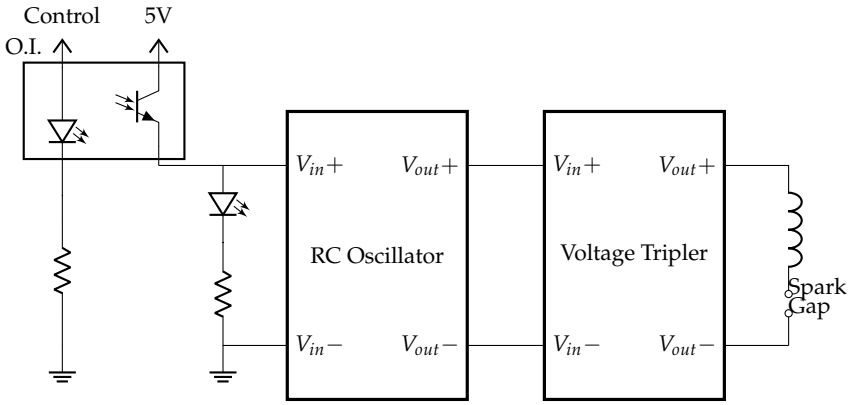

Figure 1. The EM fault injector. When a control signal is applied, the optic isolator (O.I) allows current flow to the oscillator, which feeds a voltage tripler. When a voltage sufficient to bridge the gap is reached, a high current and a magnetic field are generated in the coil.

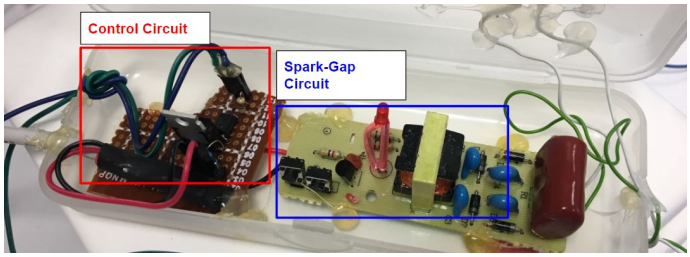

Figure 2. The EM fault injector. Both circuits are placed inside of the plastic case and the spark gap can be seen glued to the top of the case. The wires to the left and the right are for the control and the spark gap circuit, respectively.

2.2. Attack Model

We consider an attacker who has physical access to the DUT and user-land code execution permissions. The attacker is using a limited API from a privileged service, which is performing a crucial cryptographic operation. For example, the attacker asks for a privileged service to sign a message using RSA-CRT, using a private key which is not known to the attacker in user-land. Another similar scenario is an attacker gaining physical access over a device and wishing to extract secret keys from the secure enclave of the device, such as the TrustZone in ARM. We assume that the attacker uses a limited API only, and that the software application behind that API is completely bug-free and has no security vulnerabilities that the attacker may exploit. As we note further in Section 4.4, the attacker does not necessarily have to be in physical possession of the device to launch this attack, if the fault generating device is embedded inside a malicious FRU.

Our working principle assumes that by subjecting the device to a transient fault, we expect it to malfunction for a short period, during which it will perform a faulty computation, but that after the fault it will continue execution and not crash or reboot.

The attacker asks for the privileged service to perform the cryptographic operation in a loop. While the service runs, the attacker tries to inject transient faults to the device and waits for a faulty computation to take place. Once such an event occurs, the attacker collects the leaked information from the device and exploits it. The exploitation may be an unauthorized memory access, or a faulty RSA-CRT signature which can allow an attacker to reveal the private key, using the method described by Joye et al. in [11].

2.3. Comparison to Other Attack Models

The traditional attack models to attack software assume a logical flaw in the target's code. That flaw is exploited by an attacker to gain control over the device. In contrast, the proposed attack model is hardware-based. It assumes the target is running code that is completely bug-free and working exactly as intended. The hardware attack model aims to exploit weaknesses in the hardware implementation of the target to achieve the same goal as a software attack. Most common hardware attack models that use electromagnetic fault-injection attacks such as in [5] describe the attack setting to be inside of a laboratory equipped with high-end equipment. The discussed attack model proposes a new approach, in which fault attacks are applied outside of the lab setting, allowing the attack to be applied in additional scenarios and increasing its impact.

3. Results

In order to evaluate the EM fault injector, we study its effect on processors which are common among smart devices. The processor chosen to be examined is the ARMv7-based BCM2837 system-on-chip (SoC). This processor architecture and the BCM2837 technical specifications (4× ARM Cortex-A53 cores operating at 1.2 GHz) are similar to the SoC found on a common smart phone. As our attack model describes, the attacker has limited execution privileges within a Linux environment. The system hosts a privileged cryptographic service running as root, which is signing messages with a private key using RSA-CRT. The service is receiving requests from a standard inter-process-communication (IPC) interface such as sockets. To test the EM fault injector, the private key is extracted from the privileged service in the system using fault attacks.

3.1. Experiment Setup

The DUT is placed on an XYZ stage and the coiled wire of the EM fault injector, which is where the electromagnetic interference is most concentrated, is placed directly on top of the processor. This allows the coil position to be changed in a precise and controlled manner. Allowing the tester to learn the most effective locations in a specific DUT that are most vulnerable, and return to that point to gain repeatable results. The device is connected via serial connection to a computer that operates the DUT, controls the XYZ Stage and restarts the device if necessary after a crash.

3.2. ATMega328p

We conducted the same experiment described in [12], where a for-loop counter was disrupted. The program we used was virtually identical to the one in [12]. The program checks at the end of each loop if the counter is equal to some fixed value (2^{14} in our case), and if it is not, the program stops. Thus, under regular conditions, the program should never stop. We tested the EM fault injector on the ATMega328p in ten separate experiments. Each time, we expected one of the following to occur: the device may crash and restart; the device may continue its execution; finally, the device may output a different, faulty computation. In 8 out of the 10 cases, the device printed a faulty computation and stopped the execution of the program. In one particular case out of the 10, despite the fact the

program was supposed to stop after identifying a bad counter value, the program did detect the faulty value but continued to the next loop.

3.3. BCM2837

The coiled wire was placed 1.1 cm above the chip. The transient fault produced caused the device to reach one of three states: the device does not respond and requires manual reboot; the device may compute a correct signature, and it appears that nothing else is affected; finally, the device may return a faulty signature. In our experiments, the device indeed printed a faulty signature, as shown in Figure 3.

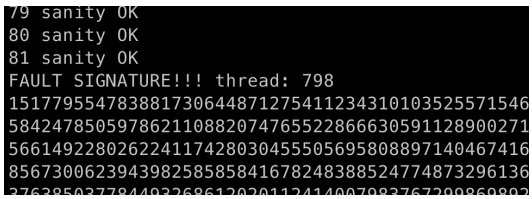

Figure 3. Output of the Raspberry Pi while running a repetitive RSA-CRT-Sign operation from the PyCrypto library. For testing purposes, the program prints every second a *"sanity"* message indicating the device is still alive and has not crashed.

The privileged signing service is using the default PyCrypto library to perform signing of a message with a private key that is hard-coded in the program. It is important to note that we are using a popular library implementation of the cryptographic primitives and did not implement our own variant. Because the attacker has full user-land control, it is assumed that they are able to utilize all cores of the system and monopolize execution time. This eases the precise timing requirement, as when the spark gap bridges, the cores will be very likely to operate on the user's code.

In order to learn which areas of the chip are most susceptible to EMFIs, we created the following experiment setup: a central PC was connected to the DUT and was capable of starting and stopping the RSA-CRT program; we recall that the PC acts as a regular user-land attacker but is still able to make the privileged program perform an RSA-CRT operation by using its exposed API. The PC was connected to the EM fault injector as well, to activate its control circuit, that is, to activate it, causing it to start producing EM pulses. Lastly, the PC was connected to an XYZ stage, which controlled the location of the Raspberry Pi and was able to move it in precise increments. This setup is described in Figure 4.

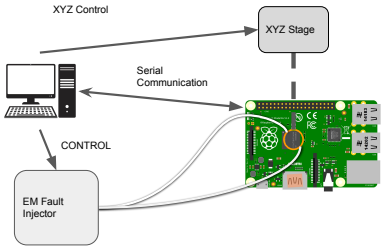

Figure 4. Cartography setup. A PC is connected to the XYZ stage's arm to be able to move the magnetic coil across the surface of the chip, connected via serial interface to the DUT and connected to the EM fault injector to activate it.

We started the cartography process by dividing the surface of the chip into a 5×5 grid. For each point, we activated the EM fault injector, which generated a pulse. As soon as it did, we checked if the device was still operating by checking its sanity outputs to the console, and if not, we checked if the DUT had printed a faulty signature, or had crashed.

After aggregating the results of 19 experiment runs as described above, we created a heat map of the vulnerable points, based on the amount of successful faults we were able to

receive for each point in the 5×5 grid. . It is important to note that the point at $(2,3)$ has a success rate of 19 out of 19 attempts, as described in Figure 5, which means that when the coiled wire was placed around that area, a faulty signature was returned to the attacker and the device did not crash, with a success rate of 100%. We assume this is because that is the location of the CPU cores within the SoC which are under heavy usage during the operation. The heat map, overlaid with a photograph of the chip, is shown in Figure 6.

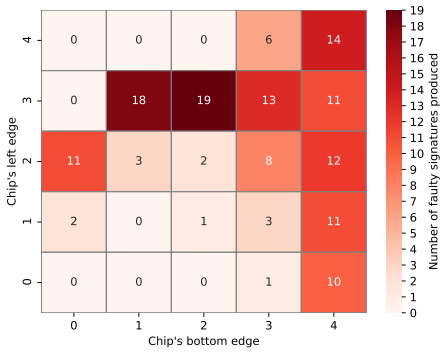

Figure 5. Heat map of vulnerable points after running the experiment 19 times, using a 5×5 grid. Each cell value represents the number of times a successful faulty signature was returned from the program while the DUT itself continued execution and did not crash.

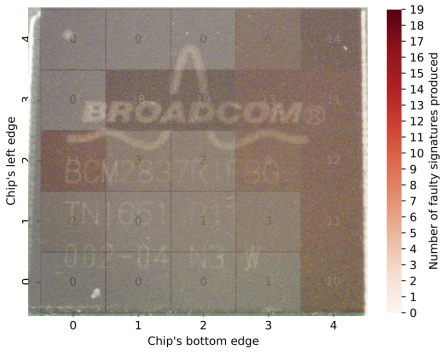

Figure 6. Cartography results overlaid with a photograph of the chip.

After determining the most susceptible area that will increase the likelihood of a successful glitch, we ran the experiment an additional 400 times to test that specific point, out of which 258 caused the device to crash, 7 were unaffected and the faulty signature was leaked 135 times.

4. Discussion

4.1. Comparative Analysis

Our technique can be compared to two approaches. The first approach is software-only attacks such as buffer overflows. The second approach is full-featured fault attacks using specialized lab equipment.

Our approach has similarities to the software-based approach, in that it can be applied in the field to devices already in the hands of the victim. In particular, it can be carried out after the device has been initialized with user-supplied secrets, after it has been unlocked with a fingerprint or face ID, or during a very specific privacy-related computation such as a cryptocurrency transfer. On the other hand, it has a disadvantage over the software-only approach, in that it requires additional hardware and cannot be immediately applied to all devices. Nevertheless, due to the widespread use of counterfeit and untrusted replacement components for smartphones and similar devices, it is within the attack model to assume a device has some malicious component, such as a battery or a screen [13].

Our approach also has similarities to the lab-based fault approach, in that it can break implementations that are completely secure and have no bugs or vulnerabilities. The disadvantage of our approach compared to the lab approach are the limitations on the capabilities of the attack hardware, due to the limited form factor, limited cost and limited power budget. In addition, there are additional preparation steps that are feasible for analysts to carry out in the lab but are not applicable in our attack model. These include chip depackaging, delayering or exposure to adverse environmental conditions such as temperature, vibration or radiation [1].

4.2. Related Work

The use of fault attacks against cryptographic operations first appeared in [14,15], which suggested how an attacker injecting faults to a device running a cryptographic operation may be able to extract the symmetric secret key or the asymmetric private key. Schmidt and Hutter [16] showed an extremely low-cost fault injection lab setup, in which EMFI was performed by a spark gap generator constructed from a simple gas lighter. That spark gap generator was only able to be operated mechanically, and the target discussed in that work was a simple 8-bit microcontroller. In the current work, we show how a low-cost fault injection device is able to disrupt the execution of a powerful processor running Linux.

Karaklajić et al. describe in [1] several common methods that have been studied for fault injection, and could be considered as alternatives for designing our fault injection device: clock glitches, optical and thermal attacks, voltage glitching and electromagnetic fault injections (EMFI).

In this work, we were interested in selecting an attack method that was both simple and inexpensive enough to be incorporated into a malicious FRU, while remaining accurate and effective enough to launch practical attacks. Therefore, we chose to use EMFI attacks, since they combine high locality and accuracy and do not require the DUT to be prepared using lab methods. Since FRUs are placed very close to the DUT, often inside the device's external EM shielding, the attack is feasible even with a limited power budget.

Exposure of the DUT's IC to an external electromagnetic field can cause the internal registers, as well as the memory, to change their state even for an encapsulated chip. An example was shown in [16], where a lab setup incorporating a low-cost spark gap generator was used to inject EM fault attacks to a microcontroller. This method is completely non-invasive, allowing the injection of faults using only physical proximity to the targeted device, and has the advantage of a relatively high spatial and temporal locality.

Riviere et al. [5] thoroughly analyzed EM fault injection on an ARMv7 microcontroller, analyzing how an instruction skip may occur on the chip when an EM fault attack takes place, by studying the changes in the data and instruction caches of the microprocessor. That work demonstrated how a high-end lab setup can be used to induce highly reliable EM faults, but was out of the reach of attackers without access to a high-end equipped laboratory. In our work, we showed how EMFI can be conducted outside of laboratory settings, and with low-cost equipment available to virtually anybody.

In 2019, O'Flynn [17] demonstrated an EMFI attack using the ChipSHOUTER[18] device to attack a critical part of the code in the USB stack of a cryptocurrency wallet. The attach let the attacker extract more bytes from the memory than legally allowed, thereby breaking the security of various cryptocurrency wallets. Since the attack was done using EMFI, the attack left no physical evidence on the device for the victim to be able to tell they had been attacked. At the time of writing this paper, the price of the ChipSHOUTER device used for O'Flynn's attack was USD 3200. In this work, we presented a device which can be easily assembled with low-cost components costing no more than USD 10.

In the last few years, fault injection attacks have also been demonstrated using software-based techniques. These works, starting with [19], exploit the energy management mechanisms found in smart phones, PCs and other smart devices, allowing them to be deployed by a remote attacker. By controlling the voltage and the operating frequency of the CPU cores of the device, an attacker can cause faults in the main processing unit.

Recent works [20–22] have extended this attack to ARM processors, demonstrating the extraction of secret keys from the ARM TrustZone enclave [23], as well as Intel's Software Guard Extensions (SGX). While these works show a new and powerful software-based attack vector, they all require the attacker to have **root privileges** on the victim's machine.

4.3. Countermeasures

Hardware failure detection is a common consideration for chip designers [24], however, they often do not consider an intended fault injected to the circuit. Many types of defenses have been proposed and tested against fault attacks. In this subsection we briefly discuss countermeasures that can handle potential attacks from malicious FRUs.

While hardware-oriented countermeasures are considered to be the most effective, applying such countermeasures may often increase the complexity and cost of such systems. A simpler approach to defend against fault attacks is through software modifications. A common software countermeasure used in cryptographic libraries to protect against fault in the RSA-CRT signing process is to verify the signature before returning from the function. This countermeasure is present in several embedded-oriented cryptographic libraries, but is not present in the standard PyCrypto library. As a defense from our attack, we implemented such a fix on the PyCrypto library, and verified it using the same attack parameters discovered previously. In out testing, out of 30 successful glitches, the faulty signature was never leaked. We provided the patch to the PyCrypto maintainers in May 2020.

4.4. Toward Malicious FRUs

Shwartz et al. [13] showed how malicious field replaceable units (FRUs) on a smart phone can attack software running on the phone's main CPU. We believe that our work opens a path to a new class of attacks, in which fault injection devices are placed inside replaceable units of smart devices. In such a setting, they gain the capacity to inject faults to the main processor, compromising the security of the system while the victim is unaware of the attack. In our work, we show that such a low-cost fault injection device can be assembled and is able to disrupt the regular execution of a powerful processor. The gap between our EM fault injector and a malicious FRU can be described in two parts.

The first part is reducing the size of the device to allow placing it inside of a small device, such as a smart phone. The circuit described in this work can be simple and can be printed to a very small printed circuit board (PCB). Small hardware has been demonstrated to be capable of creating high voltage spikes in devices such as the USB Killer, which is a device designed to collect power from the USB port and send high voltage pulses on the data lines, damaging the host PC [25]. In regard to the placement of the PCB inside an existing FRU, a malicious aftermarket device may contain a lot of free real estate inside for the attacker to use. For example, an attacker compromising a charging case can take advantage of the large surface to place larger capacitors. Another example of a malicious external device is a charging surface, which already emits EM energy and with various changes to its circuitry may act as our EM fault injector. Batteries are also large and can be made with malicious components inside. Establishing a covert channel between the malicious FRU and the user-space program allows the attacker to carry out attacks remotely on demand, resulting in a stealth attack.

The second part is using precise injection of faults to attack a specific target, both temporally and spatially. To address the temporal challenge, as shown in [20], a malicious actor who is free to execute code inside its unprivileged environment can simply ask the privileged service to perform an operation over and over again. In [20], such a method was used to perform DFA on the AES encryption algorithm and extract the encryption key. The spatial challenge may not be difficult to address in the FRU setting, since a component such as a replacement phone screen is always fitted precisely in the same place related to the phone's main CPU. The gap between our work and final malicious FRU product amounts ultimately to technical engineering work.

While performing a physical attack such as FI remotely is powerful, the attacker has to be able to overcome the engineering challenges but also to adjust the malicious FRU to fit specific smart devices; as the malicious FRU is powerful, its primary limitation is being tailored to a specific target device.

4.5. Conclusions

In this paper, we showed a set of techniques that allowed simple and effective fault injection attacks on electronic devices. The construction of the device described in the previous section was demonstrated using off-the-shelf components that cost less than USD 10, making this kind of attack feasible for mass-produced malicious FRUs. We also showed how, through the process of fault cartography, the specific locations of the target that are most susceptible to the attack can be identified. This enables more effective and targeted attack protocols to be developed for specific targets. Our evaluation used common targets and a common software library to show the generality of the techniques introduced in this paper. This suggests a new vector for fault attacks against consumer devices, strengthening the claim that resistance to fault attacks should be built into additional classes of devices.

Author Contributions: Conceptualization, S.D and Y.O.; methodology, S.D and Y.O.; resources, Y.O.; data curation, S.D.; writing—original draft preparation, S.D.; writing—review and editing, Y.O.; supervision, Y.O.; project administration, Y.O.; funding acquisition, Y.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the BGU Cyber Security Research Center.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Karaklajic, D.; Schmidt, J.; Verbauwhede, I. Hardware Designer's Guide to Fault Attacks. *IEEE Trans. Very Large Scale Integr. Syst.* **2013**, *21*, 2295–2306.
2. Guo, J.; Xiao, L.; Mao, Z. Novel Low-Power and Highly Reliable Radiation Hardened Memory Cell for 65 nm CMOS Technology. *IEEE Trans. Circuits Syst. Regul. Pap.* **2014**, *61*, 1994–2001. https://doi.org/10.1109/TCSI.2014.2304658.
3. Guo, J.; Liu, S.; Zhu, L.; Lombardi, F. Design and Evaluation of Low-Complexity Radiation Hardened CMOS Latch for Double-Node Upset Tolerance. *IEEE Trans. Circuits Syst. Regul. Pap.* **2020**, *67*, 1925–1935. https://doi.org/10.1109/TCSI.2020.2973676.
4. Cui, A.; Housley, R. *BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection*; WOOT; USENIX Association: Berkeley, CA, USA, 2017.
5. Rivière, L.; Najm, Z.; Rauzy, P.; Danger, J.; Bringer, J.; Sauvage, L. *High Precision Fault Injections on the Instruction Cache of ARMv7-M Architectures*; HOST; IEEE Computer Society: Washington, DC, USA, 2015; pp. 62–67.
6. Bukasa, S.K.; Lashermes, R.; Lanet, J.; Legay, A. *Let us Shock Our IoT's Heart: ARMv7-M under (Fault) Attacks*; ARES; ACM: New York, NY, USA, 2018; pp. 33:1–33:6.
7. Beckers, A.; Kinugawa, M.; Hayashi, Y.; Fujimoto, D.; Balasch, J.; Gierlichs, B.; Verbauwhede, I. *Design Considerations for EM Pulse Fault Injection*; CARDIS; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11833, pp. 176–192.
8. Shwartz, O.; Shitrit, G.; Shabtai, A.; Oren, Y. *From Smashed Screens to Smashed Stacks: Attacking Mobile Phones Using Malicious Aftermarket Parts*; EuroS&P Workshops; IEEE: New York, NY, USA, 2017; pp. 94–98.
9. Omarouayache, R.; Raoult, J.; Jarrix, S.; Chusseau, L.; Maurine, P. *Magnetic Microprobe Design for EM Fault Attack*; EMC EUROPE: Eruges, Belgium, 2013; pp. 949–954.
10. Makama, J.; Alpha, M.; Kure, N.; A., B.; Daniel, T.; .S, I.; Adoyi, E. Design and Construction of a Tripler Circuit for a Mosquitor Zapper. *Am. J. Eng. Res.* **2016**, *5*, 256–260.
11. Joye, M.; Lenstra, A.K.; Quisquater, J. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *J. Cryptol.* **1999**, *12*, 241–245.
12. O'Flynn, C. Fault Injection using Crowbars on Embedded Systems. *IACR Cryptol. ePrint Arch.* **2016**, *2016*, 810.
13. Shwartz, O.; Cohen, A.; Shabtai, A.; Oren, Y. *Shattered Trust: When Replacement Smartphone Components Attack*; WOOT; USENIX Association: Berkeley, CA, USA, 2017.
14. Boneh, D.; DeMillo, R.A.; Lipton, R.J. *On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)*; EUROCRYPT; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1233, pp. 37–51.
15. Biham, E.; Shamir, A. *Differential Fault Analysis of Secret Key Cryptosystems*; CRYPTO; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1294, pp. 513–525.

16. Schmidt, J.M.; Hutter, M. Optical and EM Fault-Attacks on CRT-based RSA: Concrete Results. In Proceedings of the Austrochip 2007, 15th Austrian Workhop on Microelectronics, Graz, Austria, 11 October 2007; Verlag der Technischen Universität Graz: Graz, Osterreich, 2007; pp. 61–67.
17. O'Flynn, C. *MIN()imum Failure: EMFI Attacks against USB Stacks*; WOOT @ USENIX Security Symposium; USENIX Association: Berkeley, CA, USA, 2019.
18. ChipShouter. Available online: http://store.newae.com/chipshouter-kit/ (accessed on 26 December 2021).
19. Tang, A.; Sethumadhavan, S.; Stolfo, S.J. *CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management*; USENIX Security Symposium; USENIX Association: Berkeley, CA, USA, 2017; pp. 1057–1074.
20. Qiu, P.; Wang, D.; Lyu, Y.; Qu, G. *VoltJockey: Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies*; ACM CCS; ACM: New York, NY, USA, 2019; pp. 195–209.
21. Murdock, K.; Oswald, D.; Garcia, F.D.; Van Bulck, J.; Gruss, D.; Piessens, F. *Plundervolt: Software-Based Fault Injection Attacks against Intel SGX*; IEEE: New York, NY, USA, 2020.
22. Kenjar, Z.; Frassetto, T.; Gens, D.; Franz, M.; Sadeghi, A. V0LTpwn: Attacking x86 Processor Integrity from Software. *arXiv* **2019**, arXiv:1912.04870.
23. Pinto, S.; Santos, N. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* **2019**, *51*, 130:1–130:36.
24. Khalil, K.; Eldash, O.; Kumar, A.; Bayoumi, M. Machine Learning-Based Approach for Hardware Faults Prediction. *IEEE Trans. Circuits Syst. Regul. Pap.* **2020**, *67*, 3880–3892. https://doi.org/10.1109/TCSI.2020.3010743.
25. USB Killer. Available online: https://kukuruku.co/post/usb-killer/ (accessed on 26 December 2021).