electronics

MDPI

Systematic Review

A Systematic Review of Fault Injection Attacks on IoT Systems

Aakash Gangolli *, Qusay H. Mahmoud and Akramul Azim

Department of Electrical, Computer and Software Engineering, Ontario Tech University, Oshawa, ON L1G 0C5, Canada; qusay.mahmoud@ontariotechu.ca (Q.H.M.); akramul.azim@ontariotechu.ca (A.A.)
* Correspondence: aakash.gangolli@ontariotechu.net

Abstract: The field of the Internet of Things (IoT) is growing at a breakneck pace and its applications are becoming increasingly sophisticated with time. Fault injection attacks on IoT systems are aimed at altering software behavior by introducing faults into the hardware devices of the system. Attackers introduce glitches into hardware components, such as the clock generator, microcontroller, and voltage source, which can affect software functioning, causing it to misbehave. The methods proposed in the literature to handle fault injection attacks on IoT systems vary from hardware-based attack detection using system-level properties to analyzing the IoT software for vulnerabilities against fault injection attacks. This paper provides a systematic review of the various techniques proposed in the literature to counter fault injection attacks at both the system level and the software level to identify their limitations and propose solutions to address them. Hybrid attack detection methods at the software level are proposed to enhance the security of IoT systems against fault injection attacks. Solutions to the identified limitations are suggested using machine learning, dynamic code instrumentation tools, hardware emulation platforms, and concepts from the software testing domain. Future research possibilities, such as the use of software fault injection tools and supervised machine learning for attack detection at the software level, are investigated.

Keywords: fault injection attack; attack detection; software vulnerability analysis; machine learning; software fault injection; software testing

Citation: Gangolli, A.; Mahmoud, Q.H.; Azim, A. A Systematic Review of Fault Injection Attacks on IoT Systems. *Electronics* **2022**, *11*, 2023. https://doi.org/10.3390/electronics11132023

Academic Editor: Ahmed Abu-Siada

Received: 7 June 2022
Accepted: 24 June 2022
Published: 28 June 2022

1. Introduction

As the application of IoT systems spreads across several domains, such as healthcare, smart homes, and autonomous vehicles, the security of these systems becomes increasingly important. While the attack surface for such systems is huge, significant work has been undertaken to categorize, analyze, and counter them. Fault injection attacks (FIA) inject faults into the IoT system's hardware devices that result in abnormal behavior of the software. The attacker exploits this aberrant behavior for a variety of purposes, including obtaining personal information, disrupting program flow in order to bypass critical security protections, and illegal system access and control. The faults introduced into the hardware components can be transient faults or persistent faults that remain in the system and exploit it repeatedly. Clock glitch, voltage glitch, and electromagnetic fault injection (EMFI) are examples of transient faults. These are active attacks that occur while the system is running. Fault injection attacks are different from other attacks on IoT systems as they span multiple layers of the system. The attack is carried out at the physical layer on the IoT system's hardware devices, which affects the functioning of software components and programs on other layers of the system. The software types that can be affected include device drivers, the operating system, and application software [1]. For instance, the authors of [2] analyzed how fault injection attacks are carried out on cryptographic devices to trick the encryption algorithm into using a zero-encryption key. The attacker can then use a zero key to decrypt and steal sensitive data. These types of attacks pose a significant threat to safety-critical IoT devices. Since these attacks are initiated at the physical layer and affect the software at

various layers of the IoT architecture, the methods proposed in the literature to counter fault injection attacks range from attack detection using system-level physical and network properties to software vulnerability analysis against the effects of such attacks.

Frameworks proposed in [3–5] enable the detection of fault injection attacks using physical properties of the system, such as the voltage level, temperature level, and clock frequency, or by monitoring the electromagnetic field around the IoT system [6]. These frameworks use different methods, such as formal analysis, machine learning, and deep learning, to analyze the data and subsequently detect or predict the attacks. The authors of [4] proposed a framework in which a separate sensor board consisting of various digital sensors was used to continuously monitor the IoT system properties and used an AI core to predict any abnormal events. On the other hand, studies such as [7–9] analyzed the IoT software for vulnerabilities against the effects of fault injection attacks. Software vulnerability analysis techniques aim to test the IoT software and identify exploitable vulnerabilities either by replicating actual fault injection attacks on the IoT system or by simulating the attacks directly on the IoT software. The authors of [8] proposed combining simulation-based software vulnerability detection with hardware-level verification of the detected vulnerabilities. While attack detection methods have drawbacks, such as the requirement of a separate physical hardware setup for monitoring the system-level properties, simulation-based software vulnerability analysis techniques are not always able to completely safeguard against all system-level threats due to fault injection attacks.

In this paper, a systematic literature review (SLR) of various frameworks proposed to counter fault injection attacks on IoT systems is presented. The main contributions of the paper are as follows:

- Analysis of the primary studies that propose frameworks to counter fault injection attacks on IoT systems using attack detection and software vulnerability analysis identifies the limitations and research gaps for each category.
- Proposal of hybrid attack detection methods at the software level that combine concepts from both categories, such as the use of software fault injection, machine learning, and code instrumentation tools, to address the limitations and improve the security of IoT systems against fault injection attacks.

The remainder of this paper is organized as follows: Section 2 provides a summary of the relevant background, terminologies, and definitions necessary to understand the paper. Section 3 compares this literature review with other related surveys and establishes the need for this survey. Section 4 details the method used to conduct the systematic literature review, including the research questions addressed, the search process used to identify the primary studies, the inclusion criteria, and the data extraction and analysis from the primary studies undertaken. Section 5 presents the results, which consist of the list of primary studies found and the answers to the research questions. This section presents a detailed discussion of the limitations of existing methods and possible research directions to address them. Finally, Section 6 concludes the paper and offers ideas for future work.

2. Background

This section provides background information for understanding fault injection attacks and their effects on IoT systems.

2.1. Fault Injection Attacks on IoT Systems

Fault injection attacks inject faults into the hardware and devices of an IoT system with the intention of modifying the software behavior. These attacks on IoT software invalidate the common perception that hardware faults have no consequences on IoT software functioning. These attacks can be carried out by introducing faults into various hardware components, such as the external clock generator, voltage source, and input/output (I/O) devices connected to the IoT system. The techniques used to carry out such attacks are called fault injection techniques and include clock glitch, voltage glitch, electromagnetic field injection, and optical injection, amongst many others. Figure 1 shows an example of a

clock glitch fault injection attack (CGFI) that injects glitches into the system clock. Since integrated circuits latch data and control signals at the rising or falling edge of the clock, the clock glitches lead to propagation delays in the logic blocks, causing the IoT system software to function abnormally for a brief period and to produce an unexpected output. EMFI and optical attacks use electromagnetic field and light as the means of injecting faults into the system, respectively. The faults injected by such methods may be transient glitches or long-lived faults, such as changes to the memory region. EMFI-injected faults may affect the microcontroller operation or interfere with the system voltage, causing the executing software to behave abnormally. Faults introduced into I/O devices alter the input to the IoT software, causing it to behave abnormally. The altered software behavior is subsequently used by the adversary for malicious purposes, such as bypassing authentication mechanisms due to control flow alterations [10], unsecured booting of the system firmware, and data theft using side-channel analysis [11], amongst many others.

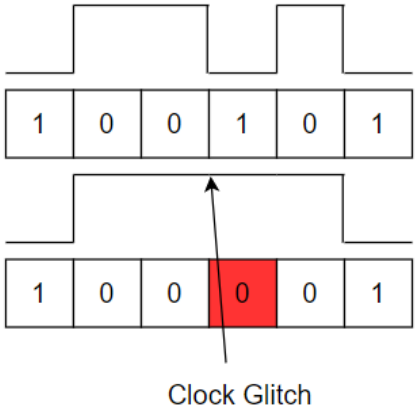

Figure 1. Clock Glitch Fault Injection Attack.

2.2. Software Effects of Fault Injection Attacks in IoT

The faults injected by the adversary into the hardware components of an IoT system subsequently propagate to and affect the software at different layers of the IoT architecture. This stage is commonly referred to as the fault propagation stage. The software effects of fault injection attacks are most commonly analyzed at the instruction level or the microarchitectural level because these levels have the closest interaction with the IoT system hardware components, such as the microprocessor, clock generator, and I/O devices. The instruction level comprises the conversion of high-level software programs into assembly program instructions, whereas the microarchitectural level is concerned with the hardware components of a microprocessor involved in the correct execution of the assembly instructions, such as the registers, execution unit, and memory. The types of software usually targeted by attackers include cryptography software, access control software, device firmware, and application software. The low-level effects of fault injection attacks are combined and utilized in a targeted manner by the attackers, which results in software effects such as instruction skip, and data corruption and manipulation. After the fault has successfully penetrated into the software and taken effect, the attacker subsequently exploits these faults for malicious purposes, such as obtaining unauthorized system access or corrupting and accessing the software output to perform side-channel analysis attacks. This is the final stage of the attack, called the fault exploitation stage. The following explains, and presents examples of, the two important software effects of fault injection attacks.

2.2.1. Control Flow Modifications

Fault injection attacks can cause one or more assembly instructions of the software to skip, resulting in control flow modifications. As a result of control flow modification or instruction skip, the attacker may be successful in bypassing the authentication check in a PIN verification software process used in an IoT device to obtain unauthorized device

access. Figure 2 shows a control flow modification resulting in the authentication of an invalid user PIN.

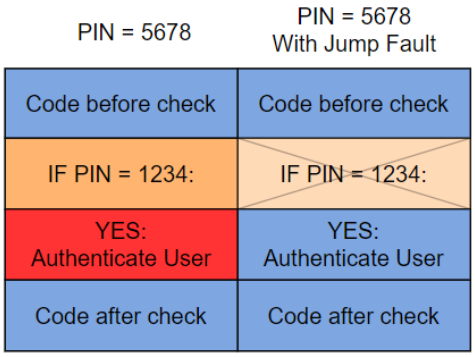

Figure 2. Instruction Skip in Pin Verification Software.

There have been successful control flow modifications performed on cryptography devices reported in the literature which have forced the device software to use a zero-encryption key [12]. If the attacker gets access to the encrypted data, they can easily decrypt it using a zero key and get access to confidential data that was supposed to be protected. Various types of instruction-level changes create control flow modifications. Single-instruction skips lead to the skipping of a single assembly instruction in the software at a time. Barbu et al. [13] demonstrated the use of single instruction skip attacks to bypass authentication mechanisms on a java card 3.0. A single instruction skip has limited value because it only affects one assembly instruction, which is not sufficient to replicate the system-level effects of fault injection attacks. Multiple instruction skips lead to the simultaneous skipping of two or more instructions. Dehbaoui et al. [14] demonstrated the use of EMFI attack on AES implementation to achieve multiple instruction skips and retrieve the encryption key using differential analysis of the ciphertexts. Methods proposed in [14–17] demonstrated multiple instruction skipping. Menu et al. [16] demonstrated multiple instruction skips on an ATmega328P microcontroller using electromagnetic fault injection as the method of physical attack. Breier et al. [17] utilized a laser injection attack to implement an instruction skip on the AES encryption algorithm running on an ATmega328P microcontroller.

2.2.2. Data Corruption and Manipulation

In this category, fault injection attacks affect the data being used in the targeted software. This data can range from user-controlled data, hard-coded data, and configuration properties, amongst many others. The intention of the attacker is to modify the data flow of the program as intended by corrupting a single bit, a single byte, or multiple bytes of data. Such attacks can also benefit from software bugs that may lead to unexpected behavior, such as buffer overflow, memory changes, and memory corruption. There are various ways in which the data can be corrupted, such as flip, set, reset, and random changes at different levels, including bit, byte, or word levels. For instance, Barenghi et al. [2] demonstrated how different fault injection techniques, such as clock glitch, laser injection, and EMFI can be used to corrupt the cryptographic keys used in different encryption programs. On the other hand, Joye et al. [18] proposed several countermeasures, such as parity-based error detecting codes (EDCs), computation randomization, and signature detection against data corruption due to instruction skipping.

2.3. Fault Models

Fault models are used to model the effects of fault injection attacks on the IoT system. Hardware fault models are created by analyzing the different ways in which fault injection attacks can induce faults in the hardware components of an IoT system in order to affect their operation. These include clock-glitch-induced faults, temperature-induced faults, and

voltage-induced faults, amongst many others. Hardware fault models are utilized in attack detection methods that use system-level properties to monitor the IoT system. Software fault models are created by considering the different effects that the injected faults into the hardware can have on the system software. For example, instruction-level fault models are created by replicating the effects of fault injection attacks at the software program instruction-level. A single-bit fault model is used to replicate attacks that affect a single bit of the software at a time. For example, a bit flip fault model modifies a single bit from 1 to 0 or 0 to 1. Similarly, there exist byte-level and word-level fault models. The jump fault model is an example of a byte-level fault model which modifies a single byte in the jump instructions extracted from the software. The jump fault model can be used to simulate the instruction-skipping effect of fault injection attacks on IoT software. Software fault models are utilized in simulation-based software vulnerability detection methods against fault injection attacks.

3. Related Work

This section analyzes other surveys in the literature that address fault injection attacks on IoT systems and devices to determine their shortcomings and justify the need for this systematic literature review. Kazemi et al. [19] reviewed methods proposed in the literature that utilize clock and voltage fault generators to physically replicate fault injection attacks on the hardware devices of microcontroller based IoT systems. Their review article only considered hardware design validation methods against clock and voltage fault injection attacks, and focused on the methods for hardware security domain with primary studies solely from the perspective of microcontroller-based IoT applications. They also proposed a standard hardware testing platform for IoT systems and devices to be used during the hardware design and system software development. Polychronou et al. [20] reviewed the various fault injection attack vectors that do not require physical contact with the hardware components of an IoT device. The review presented novel attack methods based on the hardware vulnerabilities at the microarchitectural level in IoT devices. The objective of the survey was to inform hardware designers of IoT devices about potential hardware vulnerabilities against fault injection attacks. The survey also analyzed hardware-based attack detection methods for attack vectors that targeted micro-architectural hardware vulnerabilities in IoT devices and proposed modifications for precise detection. The modifications proposed were based on the observed microarchitectural side-effects of the various attack vectors. For instance, cache monitoring was proposed for faults injected into the IoT device memory. The survey was primarily concerned with analyzing the hardware side-effects of attack vectors, such as cache miss, cache hit, and changes in on-chip thermal monitors. Bilgiday et al. [1] presented a survey on fault attack threats against embedded software that manage the security of embedded devices and systems, such as cryptography and access control software. The survey categorized the hardware fault attacks on embedded software according to the level of attack, including application level, instruction level, microarchitecture level, circuit level, and environment level. The survey focused on hardware attacks at the microarchitecture level and below, which are architecture-dependent and inapplicable to a wide variety of embedded devices. The primary studies included in the survey covered fault exploitation techniques and vulnerability analysis of embedded security devices that used hardware-based testing.

The above-mentioned surveys analyzed the various fault injection attack techniques and detection methods at the hardware level. There exist other surveys in the literature that have analyzed software vulnerability analysis techniques. Software vulnerability analysis techniques are used to determine the fault tolerance of the IoT software against the effects of fault injection attacks. Dureuil et al. [21] evaluated cryptography software implementations on smartcard-embedded devices for robustness against fault injection attacks. They proposed methodologies to detect software vulnerabilities in the cryptography software that resulted in the smartcard providing unauthorized access. The survey examined various fault models that have been used to simulate fault injection attacks on smart card devices at

the architecture level and proposed predictive vulnerability rate as an evaluation metric for software vulnerability analysis. However, the proposed metric is specific to cryptography software implementations used in smartcard devices. Eslami et al. [22] presented a survey that brought together and compared hardware-based and software-based approaches to simulate fault injection attacks on digital devices. The survey covered hardware-based fault injection and emulation-based fault injection using circuit instrumentation techniques and hardware reconfiguration. Within the domain of software-based fault injection, the survey analyzed static and dynamic methods for performing software fault injection. Software fault injection is a simulation of the effects of fault injection attacks directly on the software. The primary focus of the survey was on how fault injection attacks can be performed using hardware emulation and software fault injection techniques rather than on attack detection and software vulnerability analysis. Qasem et al. [23] presented a survey on automatic software vulnerability detection in embedded devices and firmware. The primary focus of their survey was on static code analysis, dynamic code analysis, and symbolic execution techniques to analyze software vulnerabilities in embedded and IoT device firmware. The methods analyzed by the survey were not specific to hardware-based fault injection attacks, which require techniques to analyze the software at least at the instruction level or below. Lou et al. [24] reviewed methods for microarchitectural side-channel attacks. Since fault injection attacks are used as a precursor for side-channel analysis, many of the methods discussed in the survey are also applicable to fault injection attacks and give an insight into novel countermeasures against fault injection attacks. The survey analyzed various software countermeasures by categorizing them based on the IoT architecture levels at which they are implemented. The countermeasures included in the survey are analyzed at three levels: application, system, and architecture. Countermeasures, such as monitoring run-time behavior and static software vulnerability detection, are proposed to secure the IoT system at the application level, whereas architecture-level strategies include hardware-resource partitioning for different running processes and hardware vulnerability identification. The survey covered hardware designs and software implementations of various cryptography devices that are vulnerable and analyzed the existing countermeasures. The analysis led to the suggestion of new attack vectors, rendering existing countermeasures ineffective. The primary focus of the paper was to discover new attack vectors and suggest software countermeasures for them.

The literature surveys presented in [1,19,20,25] analyzed the various hardware attack vectors to identify existing hardware vulnerabilities in components such as the microcontroller, clock generator, and voltage supply in the IoT system or device. The purpose of these surveys was to inform IoT hardware designers of the threats posed by fault injection attacks. The surveys presented in [1,2,24] were specifically aimed at cryptography devices and software, making the survey results specific and not generalizable for all IoT systems. The existing surveys do not focus on fault injection attack detection methods. There is a need to analyze attack detection methods based on the type of IoT system features used to detect the attacks, such as application, circuit, or physical level features. This will help to discover their limitations and potential solutions. Simulation-based software vulnerability analysis techniques are an effective method to safeguard against fault injection attacks but have limitations, such as lack of usability in real-time and adaptability to new attack vectors. This systematic literature review is intended to identify the limitations and research gaps for each category of solutions proposed to counter fault injection attacks, with a major focus on simulation-based software vulnerability analysis techniques. There is a lack of surveys that compile the primary research studies on attack detection and software vulnerability analysis methods. A combined analysis of existing methods in both categories is used to propose hybrid software-level attack detection methods that combine concepts from both categories, such as the use of software fault injection, machine learning, and code instrumentation tools to improve the security of IoT systems against fault injection attacks. The survey also investigates the possibility of utilizing concepts from the software test-

ing domain, such as [26,27], for IoT software vulnerability analysis against fault injection attacks.

4. Method

This systematic review follows the guidelines proposed by Kitchenham et al. [28]. This section discusses the methodology used to conduct the literature review and the many processes involved. It details how each stage was completed.

4.1. Research Questions

This study intends to produce responses to the following research questions (RQs):

RQ1: What solutions have been proposed for dealing with fault injection attacks?

Answering this question will shed light on the different types of solutions that have been proposed to deal with fault injection attacks on IoT systems. For instance, some studies have analyzed the system's physical and network features to detect fault injection attacks, while others have conducted software vulnerability analysis to ensure that the appropriate software countermeasures are in place to withstand the impact, even if the attack is successful.

RQ2: What are the limitations of the current solutions proposed and how can they be addressed?

By addressing these concerns, a better understanding of the field's current deficiencies is possible, which helps us to explore potential solutions for addressing the limitations of existing attack detection and software vulnerability methods.

RQ3: Can the limitations of existing attack detection and software vulnerability methods be addressed by using hybrid solutions from both categories and techniques from the software testing domain?

Answering this question will shed light on how hybrid solutions, machine learning, software testing concepts, and code instrumentation tools can be utilized to make IoT systems more resilient to fault injection attacks.

4.2. Inclusion Criteria

After the research questions were constructed, an inclusion criterion was determined that was required to be satisfied by all the primary studies included in this systematic literature review. This helped to establish a ground rule and to avoid any bias towards any particular study. The primary studies were required to satisfy at least one of the following inclusion criteria (IC):

- IC1: *Addresses fault injection attacks on IoT systems*
- IC2: *Addresses detection and prevention of fault injection attacks in an IoT system or device*
- IC3: *Addresses software vulnerability analysis related to an IoT system or device*

4.3. Search Process to Find Primary Studies

The search process utilized to find the primary studies included searching and thoroughly scanning multiple sources. The objective of this search was to locate the maximum number of papers that were published related to fault injection attacks and solutions proposed to deal with these attacks. The initial search for the primary studies was performed on the Google Scholar search engine, which enables the search of indexed scholarly published material. Google Scholar searches through the databases of most of the popular publications. Apart from this, a manual search was also performed on the popular publication websites, such as IEEE Xplore, ScienceDirect, ACM Digital Library, MDPI, and Springer. From the search results, the primary studies were selected by reviewing the title and abstract and skimming through the content of the papers. Subsequently, all the results from different search engines were merged, and any duplicate primary studies were

eliminated from the list. The following list of keywords were used in the search engines to find the primary studies:

- *"IoT embedded security physical fault injection attacks"*
- *"IoT embedded fault injection attack detection"*
- *"IoT embedded fault injection attack software vulnerability analysis"*

4.4. Data Extraction and Analysis from Primary Studies

The information below was extracted from the primary studies in order to answer the research questions presented above:

- The source of publication
- The authors and their institution
- The year of publication
- The task performed by the security framework
- The methodology and techniques used in the proposed solution

5. Results

The results presented in this section provide answers to the proposed research questions. Research gaps in attack detection methodologies and software vulnerability detection against fault injection attacks are identified and the use of emerging technologies, such as code instrumentation tools, machine learning, and software testing concepts to address them is proposed. By addressing these gaps, IoT systems can be made more secure and resistant to fault injection attacks and their associated consequences.

5.1. Search Results

The search process resulted in the identification of 28 primary studies. Table 1 provides an overview of the important primary studies. These primary studies are then used for data extraction and analysis, to assist in answering the proposed research questions in this systematic literature review. Table 1 also shows the statistics for the primary studies, such as the year of publication, the source, and whether the primary study was published in a journal, conference, or a book chapter. Hardware frameworks to counter fault injection attacks [29] have been available since 2013, whereas software vulnerability analysis and countermeasures [30,31] gained traction from 2014. Recent studies in both categories utilized machine-learning techniques to analyze monitored data for attack detection [3,32] and to identify vulnerability patterns for software vulnerability detection [7,33]. IEEE Xplore has published the majority of the studies in this field.

Table 1. Statistics of primary studies.

Primary Study	Year	Source	Type
Shrivastwa et al. [32]	2021	Springer	B
Richter-Brockmann et al. [34]	2021	IACR	J
Lacombe et al. [35]	2021	HAL	C
Dutertre et al. [15]	2021	ScienceDirect	J
Wei et al. [3]	2020	ScienceDirect	J
Given-Wilson et al. [8]	2020	Springer	J
Koylu et al. [33]	2020	IEEE Xplore	C
Brejon et al. [36]	2019	ACM	C
Mahmoud et al. [37]	2019	ACM	C
Khosrowjerdi et al. [7]	2018	IEEE Xplore	C

Table 1. *Cont.*

Primary Study	Year	Source	Type
Benevenuti et al. [2]	2017	IEEE Xplore	J
Deshpande et al. [38]	2016	IEEE Xplore	C
Kaliorakis et al. [39]	2015	IEEE Xplore	C
Rivière et al. [9]	2015	Springer	C
Holler et al. [40]	2015	IEEE Xplore	C
Riviere et al. [41]	2015	IEEE Xplore	C
Potet et al. [31]	2014	IEEE Xplore	C
Moro et al. [30]	2014	Springer	J
Hiroaki et al. [29]	2013	IEEE Xplore	C

C = Conference, J = Journal, B = Book Chapter.

5.2. Addressing RQ1: Categorizing the Solutionss

Table 2 provides an overview of the categorization of primary studies used for this systematic literature review. The proposed solutions can be broadly classified into two categories. This section provides an overview of each category and discusses in detail some methods from both categories that represent the state of the art and are both necessary and sufficient for determining the strengths and limitations of existing methods in each category.

Table 2. Overview of primary studies.

Primary Study	Type	Details
Shrivastwa et al. [32]	AD	EMFI and CGFI detection with FPGA-based sensor fusion monitoring
Facon et al. [42]	AD	EMFI detection with FPGA-based digital smart monitor
Deshpande et al. [38]	AD	FPGA-based monitor for clock glitch attack
Hiroaki et al. [29]	AD	Faulty clock monitor for crypto device
Benevenuti et al. [2]	AD	For SRAM FPGAs using functional redundancy
Wei et al. [3]	AD	Semi-supervised detection of voltage glitch attacks using neural network
Richter-Brockmann et al. [34]	SVA	Using physical fault injection framework
Given-Wilson et al. [8]	SVA	Formal methods and model checking
Brejon et al. [36]	SVA	Model checking based vulnerability analysis
Kaliorakis et al. [39]	SVA	Using micro-architectural fault injection simulation
Potet et al. [31]	SVA	Using model checking and dynamic symbolic execution against laser attacks
Rivière et al. [9]	SVA	Combined source code and assembly code analysis
Lacombe et al. [35]	SVA	Combined toolchain for static and symbolic program analysis
Holler et al. [40]	SVA	Using QEMU-based microarchitectural simulator
Riviere et al. [41]	SVA	EFS + Lazart for multi-level simulation
Khosrowjerdi et al. [7]	SVA	Learning-based fault injection test case prioritization
Dutertre et al. [15]	SVA	For multiple instruction skips on AES using EMFI simulation
Koylu et al. [33]	SVA	RNN-based control flow modification detection in RSA
Moro et al. [30]	SVA	Using instruction skip attacks on cryptography software
Mahmoud et al. [37]	SVA	Silent data corruption detection using software testing techniques

AD = Attack Detection, SVA = Software Vulnerability Analysis.

5.2.1. Attack Detection

The faults injected into the hardware components of an IoT system affect various properties at different levels of the system, such as the environment and hardware properties. Attack detection techniques proposed in the literature monitor these system properties to detect any abnormal variations and predict the attack. This section analyzes the existing attack detection methods in detail to discover their vulnerabilities and limitations.

Physical level attack detection methods monitor the physical properties of the systems that are expected to be disturbed by the fault injection attack. Two types of physical properties are monitored to detect the attacks: operational properties and hardware-level properties. Operational properties include temperature, electromagnetic field, and optical intensity, amongst many others, that indicate the operating conditions of the IoT system. Hardware properties include system voltage, clock frequency, and I/O device state, amongst many others. Monitoring these operational and hardware properties enables the detection of any sudden and unexpected variations in the physical conditions of the IoT system due to fault injection attacks. Physical level attack detection methods usually utilize a separate hardware setup alongside the IoT system to monitor the physical properties. Machine learning techniques are used to analyze the collected physical system data and identify any abnormal patterns representing attack conditions. Facon et al. [42] presented a machine-learning-based detection approach to electromagnetic fault injection (EMFI) attacks. A group of digital sensors on a field-programmable gate array (FPGA) board monitored the operating conditions of a crypto-accelerator, such as the temperature, clock frequency, voltage, and reset line stability of the FPGA. The proposed framework also monitored other hardware properties, such as the I/O ports of the system and the processor load. The monitored physical features were utilized to predict attack and nominal scenarios using different supervised machine-learning algorithms, such as support vector machines (SVM), logistic regression classifier (LRC), naive bayes classifier (NBC), and multi-layered perceptron (MLP). The results of the machine-learning-based detection were compared to the baseline approach using sensor value thresholding. The results demonstrated that machine-learning algorithms are highly capable of detecting intricate deviations in the physical operating conditions resulting from fault injection attacks and detected the attacks with an average accuracy improvement of 25–30%.

Jiang Wei [3] utilized a semi-supervised machine-learning technique to detect voltage glitch fault injection attacks on digital IoT systems. The technique utilized a feedforward neural network with a single hidden layer to find hidden variables, improving the detection performance, with the major motive being reducing the learning time complexity of the detection framework. An extreme machine-learning (ELM) classifier was used to perform the clustering on a voltage glitch experiment dataset at the gate-level. The proposed approach achieved an average accuracy and f1-score of 87%. Methods such as [5,29,38] utilized only hardware-level properties to detect fault injection attacks. Deshpande et al. [38] introduced a configurable ring oscillator-based timing monitor to detect fault injection attacks which cause timing glitches in the hardware components of the IoT system, such as changes in clock duration or frequency. The timing monitor was also able to detect voltage manipulations. The proposed monitoring device was implemented on an FPGA, which made it reconfigurable and applicable to a wide range of IoT systems and devices. Benevenuti et al. [5] presented a fault injection attack detection framework for SRAM-based FPGAs by duplicating certain functional modules of cryptography devices, such as the substitution box (S-box), which is a part of the advanced encryption standard (AES). The framework used functional redundancy as a means of comparing the output of duplicated modules in the device and detecting fault injection attacks. The proposed approach was evaluated using a bit flip software fault model to replicate the effect of a laser-based optical fault injection attack on a Xilinx 7 Series SRAM-based FPGA. Unlike the above-mentioned methods, the proposed approach could detect system-level threats, such as silent data corruption (SDC) and functional interruption (SEFI), for FPGA-based

applications. Other hardware-based attack detection methods proposed in the literature include [2,6,13,14,17,43].

The most recent methods in the literature have combined data from multiple sources to detect fault injection attacks [4,32]. Shrivastwa et al. [32] proposed a hardware-based monitoring framework whereby a smart monitor was used to continuously monitor the system and predict EMFI and CGFI attacks on a target FPGA board. The proposed smart monitor was implemented on an FPGA board and consisted of several digital sensors, such as temperature, voltage, and other precision sensors, placed at different locations on the FPGA board. The data collected by all these sensors was aggregated using data fusion techniques. Sensor data fusion was utilized to avoid incorrect readings or local anomalies from saturated digital precision sensors and improved the reliability of the smart monitor. The aggregated data was processed in two stages to detect and classify the fault injection attack. The attack detection stage was evaluated using several supervised machine-learning algorithms, such as the Gaussian naïve Bayes classifier (GNBC), logistic regression classifier (LRC), support vector machines (SVM), and multi-layer perceptron (MLP). The detection stage was trained using supervised machine-learning algorithms as they are simple and low-latency algorithms compared to artificial intelligence (AI) techniques, such as neural networks (NN). This ensured minimal prediction latency, which helped in activating the countermeasures as soon as possible. An AI core was further used to classify the type of detected perturbation as nominal, SMFI, or CGFI. The AI core used to perform the classification is a complex stage and has a higher latency than the detection stage. The framework was evaluated using high level synthesis (HLS) and an FPGA board and demonstrated that the proposed framework performed marginally better than threshold-based attack detection. The framework distinguished between EMFI and CGFI with a 77.25% accuracy while maintaining a false negative rate of 0. Though approaches using sensor fusion and multi-source data fusion to detect fault injection attacks are currently limited, these solutions are becoming more popular due to their end-to-end system security. Sahu et al. [4] proposed methods utilizing sensor fusion and the combination of various data sources, such as physical and network properties, to detect a wide variety of attacks at different levels of an IoT system, including fault injection attacks. Frameworks such as [4,44,45] can also help in discovering features at other IoT system levels that are indicative of fault injection attacks. For instance, the effects of fault injection attacks on the network and software layer of an IoT system can be studied in detail using such frameworks.

5.2.2. Software Vulnerability Analysis

The faults injected into the hardware components of an IoT system ultimately affect the software functioning. The second approach to counter fault injection attacks is by implementing the appropriate software countermeasures that make the software resistant to the effects of fault injection attacks. Fault injection tolerant software necessitates analyzing the software to identify the potential exploitable vulnerabilities. The methods proposed in the literature to detect software vulnerabilities to fault injection attacks can be categorized as physical testing and simulation-based analysis techniques.

Physical testing techniques utilize physical testing equipment to perform fault injection attacks on various hardware components of the IoT system and subsequently monitor the software for any abnormal behavior. The methods in [46–49] have injected faults into various hardware components of an IoT system, including the microprocessor, clock generator, and voltage supply, to discover new software vulnerabilities. Kazemi et al. [46] performed clock glitch attacks on a medical IoT device to identify control flow vulnerabilities in the software executing on a microcontroller. The proposed approach was evaluated on a password-checking procedure running on an ARM-based microcontroller with multiple conditional statements, and it was concluded that the number of authentication steps has a linear relationship with the software fault tolerance. The authors of [47,48] described frameworks to perform voltage glitch attacks on IoT devices using Intel software guard extensions (Intel SGX). Intel SGX is used to isolate the memory region for multiple pro-

cesses executing on a microprocessor. These studies demonstrate the catastrophic effects of voltage glitch attacks causing faulty operation of Intel SGX resulting in overlapping memory regions for low-privilege and high-privilege processes. Bossuet et al. [49] described a physical testing setup to perform multi-spot laser fault injection attacks on cryptography devices in order to induce instruction-level faults in the software. Experiments performed on two characterization codes demonstrated the ability of the testing setup to induce bit level faults at the program instruction-level of the executing software.

Simulation-based software vulnerability analysis is the second category of solutions used to test the software fault tolerance against fault injection attacks. Higher-level software fault models can simulate limited attack effects because they do not correspond to software modifications caused by hardware-injected faults. For example, software fault models at the source code level can only modify the variables and programming statements that do not correspond to the effects of fault injection attacks, such as single-instruction-skipping and changes in conditional branches. Simulation-based vulnerability analysis techniques simulate the fault injection attack effects directly on the software at the assembly instruction or microarchitectural level. **Instruction-level software fault injection tools**, such as Chaos Duck [50] and Simplifi [51], simulate the effects of fault injection attacks by modifying the assembly instructions of the software using several fault models described in Section 2.3. For instance, the Chaos Duck tool extracts the assembly instructions from the executable software and injects faults using non-operation (NOP), jump (JMP), flip (FLP), and zero-byte (ZERO) fault models. The tool runs the faulty executables and records the execution output, which can be analyzed to predict if the injection created a vulnerability. Potet et al. [31] described the Lazart tool to simulate fault injection attacks on IoT software. Lazart simulates fault injection attacks by operating on the low-level virtual machine intermediate representation (LLVM-IR) [52] of the software, instead of modifying the software directly at the assembly instruction level. This allows the Lazart tool to combine various low-level software fault models and simulate the attacks at an intermediate level for a higher similarity to the effects of actual hardware fault injection attacks. The mutant executables generated by modifying the LLVM-IR program are executed using symbolic execution to generate test cases. The results of these symbolic executions can determine attack paths or sometimes be inconclusive. Riviere et al. [9] improved the Lazart tool by combining it with an embedded fault simulator (EFS) [41] to simulate multiple fault injection attacks. Lazart operates statically at compile time on the intermediate instruction representations, whereas EFS operates dynamically at run time on the assembly level instructions. Combining the two tools enabled the simulation of a wide range of attacks at multiple levels of the IoT system. Experiments performed on the PIN verification software demonstrated its ability to detect a higher number of vulnerabilities compared to Lazart. For instance, Lazart exhibited a detection rate of 16.6%, while the combined approach improved the detection rate to 21.4%.

Microarchitectural-level software fault injection tools simulate the effects of fault injection attacks by modifying the way in which assembly instructions are executed by the microprocessor. These tools simulate the attacks by injecting faults into components of the microprocessor, including registers, instruction cache, instruction execution unit and buffers, amongst many others. Höller et al. [40] described a quick emulator (QEMU)-based software fault injection tool supporting the majority of the processor architectures used in IoT systems. This QEMU-based simulator was used to inject faults into the memory, registers, and execution unit of the processor. Utilizing QEMU makes the simulator applicable to a wide range of processor architectures. The simulated attacks lead to incorrect execution of assembly instructions by the processor, which can be used to implement control flow errors, memory faults, and data corruption. For instance, modifying register bits leads to incorrect loading of data stored in variables, producing an unexpected output. The QEMU-based fault injection tool was evaluated on a password-checking procedure against control flow errors, memory access errors, and data corruption. Kaliorakis et al. [39] presented a fault injection simulation approach for the x86-64 processor architecture using

the MARSSx86 simulator [53]. The proposed approach achieved a speedup of 2.92× compared to the baseline approach which utilized full execution on the processor instead of the simulator. However, the proposed simulator is only applicable to the x86-64 architecture, which limits its application in IoT systems that primarily use ARM-based processors. Microarchitectural-level software fault injection tools can also be used to simulate attack effects at the instruction level, though they are more difficult to implement.

The methods proposed in the literature utilize various analysis techniques on the software fault injection output to detect instances that create vulnerabilities. The majority of the software vulnerability analysis methods utilize model checking or machine learning to analyze the software fault injection output and automatically detect software vulnerabilities to fault injection attacks. Model-checking techniques use system-modeling techniques to build an expected model of the IoT software along with expected features and properties. Formal analysis methods are used on the models created from the software fault injection output and their properties are verified against the expected properties. A difference in the model properties obtained from software fault injection indicates that the injected fault created a software vulnerability. Formal analysis methods are mathematical procedures used in software verification. The authors of [34,36,54] utilized model-checking methods to analyze cryptography software for vulnerabilities. Given-Wilson et al. [8] described a framework for run-time verification of fault-injected software binaries using model checking to validate the expected software properties defined as variables and constants at certain positions in the software. The fault-injected software binaries are created using instruction-level fault models, such as NOP and FLP. Model checking, applied on the fault injected binary execution, classifies it as vulnerable, correct, incorrect, or crashed. Potet et al. [31] used model checking on dynamic symbolic execution results to detect control flow modifications caused by simulated laser fault injection attacks.

Machine-learning methods utilize pattern recognition to identify vulnerability patterns using static and dynamic software properties. The methods described in [7,33,55] have used machine-learning techniques to identify software vulnerabilities. Khosrowjerdi et al. [7] described a method combining model checking and machine learning to detect vulnerabilities in safety-critical software including cryptography and access control software. A finite state modeling (FSM) technique is used to build the system models. The software fault injection process is carried out using the quick emulator (QEMU) and GNU debugger (GDB) and system models are generated for each injected fault. The proposed method utilizes machine learning on reverse-engineered system models to automate test case generation. This prevents executing each fault-injected binary. The generated test cases identify software fault injection points that are more likely to result in exploitable vulnerabilities. Model checking is used on the execution output of the generated test cases to identify if an injected fault creates a vulnerability. Koylu et al. [33] proposed the use of a recurrent neural network (RNN) to detect program flow modifications in the RSA cryptography software due to instruction-level simulation of fault injection attacks. An RNN was trained on the control flow graphs generated by symbolic execution of fault injected executables to identify whether or not the injected fault created a vulnerability. Lacombe et al. [35] utilized static source code analysis to detect fault injection points in the software and subsequently used dynamic symbolic execution to validate the attack paths. Prioritizing fault injection points using static analysis saves a significant amount of the time required for the time-consuming dynamic symbolic execution, which may also be non-terminating. These non-terminating conditions are identified using the static analysis step and are avoided.

5.3. Addressing RQ2: Analysis and Limitations

This section analyzes the significant primary studies reported in the literature that are sufficient to determine the advantages and limitations of existing solutions in the two categories of attack detection and software vulnerability analysis methods.

5.3.1. Attack Detection

The physical level attack detection methods described in Section 5.2.1 [3,5,19,29,32,38,42] all require a separate physical hardware monitoring setup alongside the target IoT system or device to detect fault injection attacks. This makes such attack detection methods expensive, and the hardware monitoring setup needs to be adjusted for every IoT application. For instance, Shrivastwa et al. [32] utilized an FPGA-based digital smart monitor, whereas Wei Jiang [3] utilized hardware equipment to monitor gate-level properties of integrated circuits (ICs), such as voltage levels at different points, in order to analyze circuit-level properties and detect fault injection attacks. Shrivastwa et al. [32] strategically placed the digital precision sensors of the smart monitor at different locations on the FPGA board on which the IoT system is implemented. These sensor locations need to be manually modified for each IoT application. The use of physical properties, such as temperature, optical intensity, and voltage, by attack detection frameworks also makes them vulnerable to false data injection attacks and adversarial machine learning [56]. The physical data used by the detection frameworks as an input can be manipulated by adversarial attacks, leading to a high number of false negatives in the attack predictions. False triggering of system-level countermeasures on detection of an attack can make it possible for adversarial attackers to perform side-channel analysis by observing the system behavior. This creates a new source of side-channel attacks that can be used to steal confidential data from cryptographic devices used in IoT systems or alter the functionality of safety-critical and access control IoT software [57,58]. Another important concern with attack detection methods is the machine learning or AI algorithm to be used. Shrivastwa et al. [32] achieved high prediction and classification accuracy due to the use of supervised machine-learning algorithms. However, this poses a significant challenge of creating the necessary training data for the machine-learning algorithm. The IoT system or device needs to be subjected to fault injection attacks in different scenarios in order to collect the necessary labeled physical data representing nominal and attack conditions. Subjecting every IoT system to controlled fault injection attacks can be a difficult task, and the physical equipment required to perform such experiments may need modification according to the IoT system. This makes supervised machine-learning techniques difficult to utilize for attack detection due to the lack of a labeled dataset. It is not possible to generate such a dataset containing physical properties, such as temperature and voltage, using attack simulation techniques. A possible solution to this is to utilize a hardware description language (HDL), such as Verilog and VHDL, to simulate the IoT system hardware. This still requires simulating fault injection attack conditions in HDL, which remains an open challenge. The difficulty in creating a labeled dataset has led to other methods to utilize semi-supervised or unsupervised machine-learning techniques [3]. However, the use of AI-based frameworks creates problems such as performance and timing overhead for the detection module, especially in resource-constrained IoT systems. The above-mentioned limitations of attack detection methods are evident by the lack of frameworks proposed in the recent literature.

Responses to the major limitations of attack detection methods that need to be pursued in future research are as follows:

- Standard instruments to replicate fault injection attacks on commonly used devices in IoT systems, such as ARM-based microprocessors, I/O devices, clock generators, etc.
- Generic attack detection frameworks using supervised machine-learning algorithms that provide better results but present the challenge of creating a labeled dataset. Such frameworks must be adaptable to multiple IoT systems with minor modifications.
- A generic labeled dataset generation workflow for attack detection applicable to a wide range of IoT systems and devices, preferably using emulation-based methods to avoid replicating fault injection attacks on the actual IoT system or device.
- Countering adversarial machine learning and false data injection attacks on the fault injection attack detection mechanism. Such attacks are commonly carried out on the physical properties of IoT systems, such as temperature, voltage level, electromagnetic field, and optical intensity, amongst many others.

5.3.2. Software Vulnerability Analysis

The limitations of physical testing techniques used for IoT software vulnerability analysis are similar to those of attack detection methods. Physical testing techniques require replicating actual fault injection attacks on IoT systems, similar to attack detection methods. However, the effects of the attack on the IoT software are observed and inspected to discover exploitable software vulnerabilities to fault injection attacks and determine appropriate software countermeasures. Physical testing techniques require a hardware setup to replicate the attacks, which can be expensive and needs to be modified for every IoT application. For instance, the clock glitch generator device presented by Kazemi et al. [46] injects glitches into the microcontroller clock cycles specifically for a medical IoT device and discovers control flow software vulnerabilities. Similarly, voltage glitch attack generators proposed in [47,48] are specific to IoT devices using Intel microprocessors enabled with Intel SGX, whereas the multi-spot laser attack setup proposed in [49] is specific to cryptography devices. The hardware testing setup currently proposed in the literature is not applicable to a wide range of IoT systems and devices and is instead application-specific. This necessitates major modifications in the hardware setup for testing every IoT system for software vulnerabilities to fault injection attacks. Moreover, it is difficult to carry out targeted fault injection attacks that result in exploiting the software vulnerabilities. Only those attacks which cause the software to behave unexpectedly are useful. The methods proposed in [19,48,49] demonstrate that thousands of fault injections need to be performed in experiments to detect a small number of software vulnerabilities. This conclusively demonstrates that physical testing techniques cannot be generic and can only be used to detect software vulnerabilities in application-specific IoT systems and devices that can afford the time and cost investment of developing appropriate fault injection equipment.

Simulation-based software vulnerability analysis techniques analyze the IoT software by simulating the effects of fault injection attacks directly on the software using several fault models. The software fault models utilized in the domain of hardware-based fault injection attacks are created by analyzing the software effects at two levels of the IoT software: the instruction level and the microarchitectural level. Instruction-level simulation tools modify the assembly program instructions of the IoT software to replicate the effects of fault injection attacks using fault models such as NOP, FLP, and JMP. For instance, the Chaos Duck tool uses the NOP fault model to replace an assembly instruction of the software with the non-operation instruction for x86 and ARM-based instruction sets. Instruction-level simulation tools described in the literature, such as [41,50,51], are specific to certain instruction set architecture families, such as x86 and ARM. While these architectures cover the majority of the IoT systems and devices, other instruction set architectures and extensions, such as ZipCPU and RISC-V, commonly used in FPGA boards [59–61] to implement several IoT applications are not supported by these tools. This is a major limitation of the existing instruction-level fault injection tools. The existing instruction-level tools modify each assembly instruction at the bit, byte, or word level. This allows the tools to only simulate fault injection attacks that affect a single assembly instruction of the IoT software. The effectiveness of instruction-level tools is currently limited due to the inability to simulate high-level effects of fault injection attacks, which simultaneously affect multiple assembly program instructions. For instance, control flow alterations due to skipping of multiple simultaneous instructions cannot be simulated using the existing instruction-level tools. The combined Lazart and EFS tool [41] tackled this problem by combining multiple low-level software fault models to precisely simulate high-level hardware attacks. However, the use of LLVM-IR limits its application to only Linux-based systems, which is a major limitation. Moreover, Lazart has been developed specifically for laser fault injection attacks on cryptography devices. Microarchitectural-level simulation tools simulate the attacks by injecting faults into microprocessor components responsible for executing the assembly instructions. For instance, the QEMU-based tool proposed in [40] injects faults into the memory, registers, and execution unit of the processor to modify the manner in which the assembly instructions are executed. Microarchitectural-

level fault injection simulation tools [39,40] utilize microprocessor architecture emulators, such as QEMU and MARSSx86 [53]. The attack effects are simulated by modifying different sections of the simulator tools responsible for executing the assembly instructions. The microprocessor architecture simulators allow the fault injection tools to be compatible with a wide range of processor architectures. For instance, QEMU supports ARM, x86, MIPS64, PowerPC, and MicroBlaze processor emulations, which are widely used in IoT systems and devices. Microarchitectural-level fault injection simulators are difficult to implement as they require complex modifications to the processor modules in order to affect the instruction execution. Although simulated processor-level modifications allow microarchitectural-level tools to simulate effects that closely replicate actual hardware effects of fault injection attacks, instruction-level tools are easier to implement as they directly modify the assembly instructions and do not require knowledge of the processor architecture and working. Much of the literature has utilized simulation tools for the vulnerability analysis of safety-critical IoT software, such as cryptography software, password-checking procedures, and device firmware [23,30,34,36,62]. However, fault injection attacks can also affect the application software of an IoT system. There is a lack of studies that utilize software vulnerability analysis to identify threats to application-level IoT software, such as the control daemon of home automation software. Detecting vulnerabilities in application software will also help detect system-level threats due to fault injection attacks as application software is in the topmost layer of the IoT architecture and is usually directly connected to the physical environment of the system.

The fault injection attack simulation output is analyzed to predict if the injected fault creates a software vulnerability to a fault injection attack. The software vulnerability analysis methods proposed in the literature utilize various types of output from the software fault injection process for the analysis. For instance, the Chaos Duck instruction-level software fault injection tool [50] runs the fault-injected executables and records the execution output. This execution output can contain run-time software properties such as the output logs, error code, and time out. Potet et al. [31] analyzed the control flow graphs generated dynamically from the fault-injected executables. The techniques used in the literature to analyze the software fault injection output include model checking [34,36,54] and machine learning [7,33,55]. Model-checking methods utilize certain pre-defined model properties of the IoT software, such as security variables. This static nature of model-checking methods cannot detect unseen software vulnerabilities without manually updating the expected model properties. Machine-learning techniques learn patterns from the existing software vulnerabilities and are capable of detecting unseen vulnerabilities. Machine-learning techniques have also been used in the literature to prioritize fault injection points, resulting in a higher probability of a software vulnerability [7]. Most existing methods utilize unsupervised, semi-supervised, and AI techniques due to the lack of a labeled dataset. Unsupervised machine-learning methods do not perform as well as supervised machine-learning techniques, especially for unseen data instances, as such methods usually rely on clustering-based techniques to differentiate between vulnerable and non-vulnerable injection instances. On the other hand, AI-based techniques, such as [7,33], have a significant overhead in terms of timing and computing resources and are not suitable for the majority of resource-constrained IoT systems. An alternative is to utilize supervised machine-learning techniques that perform well, in addition to having less timing and computing overhead. However, this involves the cumbersome process of creating a labeled dataset for every IoT application and presents the challenge of creating a standard labeled dataset generation workflow. There is also a lack of studies that utilize machine learning to prioritize fault injection points to detect the maximum possible vulnerabilities in the IoT software.

Software vulnerability analysis techniques are utilized to detect vulnerabilities in the IoT software to fault injection attacks that modify the system behavior by causing the software to behave abnormally. The detected vulnerabilities are subsequently filled using the appropriate software countermeasures, such as software module duplication, use of

user-defined variables and data types, and multiple redundant checks for conditional statements. However, these implemented countermeasures can themselves be affected by fault injection attacks, making them ineffective. Papadimitriou et al. [11] targeted the hardened versions of the AES cryptography software and demonstrated that it remains vulnerable to fault injection attacks. We conduct a fault injection experiment on the hardened VerifyPIN software included in the FISSC dataset [63] using the Chaos Duck software fault injection tool. Figure 3 shows a FLP fault injected at an instruction with address 0xc5f authenticating an invalid user PIN.

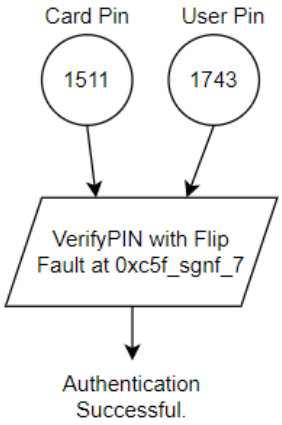

Figure 3. FLP Fault in VerifyPIN Granting Unauthorized Access.

Moreover, the existing software vulnerability analysis techniques in the literature cannot be used in real-time to detect any dynamically created vulnerabilities in the live environment of the IoT system. This creates an opportunity to borrow the concepts from existing simulation-based software vulnerability analysis techniques and to utilize them to detect fault injection attacks in real-time at the software level.

The major limitations for software vulnerability analysis methods that need to be addressed in future research are as follows:

- Physical testing techniques to discover software vulnerabilities are limited by the lack of a generic hardware setup for different IoT systems and targeted fault injection attacks that trigger the software vulnerabilities.
- Simulation-based software vulnerability analysis techniques are limited by the microprocessor and instruction set architecture family supported by the software fault injection method. They are also limited by the difficulty in replicating high-level software effects of fault injection attacks by combining low-level fault models.
- There exist limited primary studies in the literature analyzing application software for vulnerabilities to fault injection attacks.
- The existing methods do not use supervised machine-learning techniques for detecting software vulnerabilities due to the lack of a generic labeled dataset generation workflow.
- The existing software vulnerability analysis techniques cannot be used in real-time to detect dynamically created software vulnerabilities.

5.4. Addressing RQ3: Future Research Directions and Solutions

This section proposes possible solutions to the limitations described in the previous section.

5.4.1. Attack Detection

The major limitation of attack detection methods is the replication of fault injection attacks on the actual IoT system or device. This can be addressed by utilizing software-based emulation tools such as QEMU and MARSSx86. Software-based emulation tools create software components that imitate the hardware behavior using the host computer running

the tool. For instance, QEMU emulates the ARM processor behavior by utilizing software components that replicate the behavior of hardware components of the ARM processor, such as the arithmetic logic unit (ALU), registers, instruction decoder, and execution unit. QEMU supports emulation for a wide range of processors, including x86, ARM, SPARC, RISC-V, MicroBlaze, and other commonly used processor architectures in IoT systems and devices. Studies such as [39,40] utilized software-based emulation tools to simulate the effects of fault injection attacks and used them for software vulnerability analysis. The methodologies used in these studies can be used to replicate fault injection attacks on emulated processors for attack detection. The simulation techniques used in these studies modify and inject faults into the emulated microprocessor components, which is very similar to fault injection attacks that inject glitches into the hardware components of an IoT system. This makes it possible to use the described methods to perform fault injection attacks on emulated hardware, which can be used to create a labeled dataset for attack detection using supervised machine learning. This would avoid the need for a separate hardware setup to replicate fault injection attacks on the actual microprocessor used in the IoT system, reducing the cost and effort required to conduct the fault injection experiments. The use of a software-based processor emulation tool can address the limitations of replicating fault injection attacks and standard labeled dataset generation workflow. Another major limitation of attack detection methods is the threat of adversarial machine learning on the physical system properties, such as temperature, voltage, light intensity, and electromagnetic field, used to monitor the IoT system operating conditions and detect fault injection attacks. There is a lack of such studies in the literature that consider the threat of adversarial machine learning on fault injection attack detection frameworks for IoT systems and devices. Utilizing supervised machine learning as described above in conjunction with techniques such as adversarial training with perturbation or noise [64], gradient masking [65], and input regularization [66] can be useful to counter adversarial machine learning and false data injection attacks on the detection mechanisms.

5.4.2. Software Vulnerability Analysis

Physical testing techniques to detect software vulnerabilities are not a popular option as they require replicating fault injection attacks on the actual IoT system. This limitation has been addressed by studies such as [39,40] which utilized software-based emulation platforms for microprocessors. However, these methods operate at the microarchitectural level, which makes it difficult to implement fault injection experiments by modifying the emulated microprocessor components. The processor architecture knowledge should be known to implement such fault injections. Instruction-level fault injection simulations performed in [41,50,51] only require knowledge of the instruction set architecture family, rather than the functioning of microprocessor components. The fault models currently used by such tools to simulate the effects of fault injection attacks are limited. Chaos Duck [50] is a generic software fault injection tool which can be used to test any software against instruction-level faults. The FLP fault model utilized in Chaos Duck does not flip multiple bits simultaneously, which limits its application to IoT software. The tool is also unable to simulate high-level effects of attacks, such as control flow alterations due to skipping of multiple simultaneous instructions. The application of multiple simultaneous faults using combinations of existing fault models can be conducted experimentally to simulate the high-level effects of fault injection attacks on the IoT software. These modifications to the generic instruction-level software fault injection tools can be useful to detect software vulnerabilities to fault injection attacks for a wide range of IoT software, unlike the existing application-specific methods.

The fault injection simulation output is analyzed using machine learning to detect injection instances that create software vulnerabilities [7,33]. The accuracy obtained in [33] varied from 35–100% for different fault models. These methods utilize instruction-level and micro-architectural level features, such as the instruction flow and register contents, to train AI algorithms to detect abnormal patterns and conditions. The existing methods

give acceptable performance but use features that are not always easy to monitor. The use of control flow graphs and code property graphs to detect vulnerabilities in IoT software is limited in the literature [7,35]. Concepts from general software vulnerability analysis can be used to detect vulnerabilities in IoT software to fault injection attacks. For instance, Yamaguchi et al. [67] presented a machine-learning-based framework trained to identify search patterns for taint style vulnerabilities in the software. The framework identifies patterns for taint-style vulnerabilities affecting the data using code property graphs to monitor the data flow during symbolic execution of the software. This graph-based approach is capable of identifying corruption of data inputs to functions resulting in unexpected function output, which can be useful in detecting data corruption and modification in IoT software due to fault injection attacks. On the other hand, Kim et al. [68] proposed utilizing static analysis as a first stage to detect vulnerable sections of the software and then using dynamic analysis as a second stage to reduce the number of false positives obtained from the first stage. The static analysis is performed on the abstract syntax trees (AST) of the source code, containing the syntactical information conveyed by the source code. Fuzz testing is used for dynamic execution. This approach minimizes the use of dynamic execution by restricting its application to vulnerabilities identified by static analysis in order to inject a fault and confirm the vulnerability. A limitation of such methods [67] is that they are specific to a particular combination of data source and sink, requiring the machine-learning model to be trained for each unique combination. The methods also require proper data sanitization rules to be established beforehand for each source–sink combination. These concerns need to be addressed before such techniques can be used on IoT software. Currently, there are limited studies that utilize machine learning and AI techniques to prioritize software fault injection points that result in software vulnerabilities, particularly in the field of fault injection attacks on IoT systems and devices [7,37]. Research into the adaption of generic software testing and analysis concepts at the executable level, such as test case prioritization and vulnerability impact analysis, should be performed to improve the detection performance and timing aspect of existing software vulnerability analysis methods. Though vulnerability impact analysis may add significant timing overhead to the testing process, it helps to determine which software vulnerabilities create larger impacts at the system-level amongst the thousands of vulnerabilities found in large-scale IoT software. Finally, localization of the detected vulnerabilities remains an outstanding issue. The vulnerabilities detected by using instruction-level simulation tools indicate the assembly instruction which is vulnerable to fault injection attacks. However, to determine the appropriate source code countermeasure, the vulnerability needs to be traced back, which is currently lacking. The use of dynamic binary instrumentation tools such as PIN [69] and DynamoRio [70] should be investigated in this regard. PIN is a dynamic binary instrumentation tool which supports Intel IA-32, x86-64 and MIC instruction-set architectures, whereas Dynamo Rio is another tool that allows the manipulation and monitoring of software during run-time and supports IA-32, AMD64, ARM, and AArch64 processor architectures. Dynamic binary instrumentation tools can be utilized to generate debugger symbols, which can help relate the assembly instructions to the approximate source code location. This presents future research possibilities of integrating dynamic code instrumentation tools with fault injection simulation tools to create an integrated environment for IoT software vulnerability analysis against fault injection attacks.

5.4.3. Software-Level Attack Detection

As mentioned in Section 5.3.2, software vulnerability analysis techniques lack the ability to be used in real-time to detect dynamically created software vulnerabilities. These dynamically created vulnerabilities can exist because of unexpected input to the application software due to fault injection attacks or previously unseen vulnerabilities. For instance, the fault injection attack may manipulate the input data to the IoT software, which can result in a buffer overflow despite the implemented software countermeasures [62,71]. This necessitates detecting such attack effects at the software-level in real-time. Software

fault injection tools which have been utilized in simulation-based software vulnerability analysis can be used for software-level attack detection. Instruction-level software fault injection tools, such as Chaos Duck [50] and Simplifi [51], can be used to train supervised machine-learning models for real-time software-level attack detection. For instance, the Chaos Duck tool injects instruction-level faults into the extracted assembly instructions from a software executable and creates faulty executables. The recorded execution output of these faulty executables contains the output logs, error logs, exit code, and time out. This execution output can be transformed into a labeled dataset which can be used to train supervised machine-learning models. These models can be used in a live IoT environment to predict fault injection attacks at the software level. However, additional software run-time properties need to be recorded in the execution output in order to detect only those software anomalies resulting from fault injection attacks. The feasibility of software-level attack detection using a simulation tool should be analyzed in future research. Application-level input data used to execute the software can be a useful indicator of such attacks because the software behavior under simulated faults is dependent on the input to the software. The combination of application-level input data to the software and run-time software features to detect fault injection attacks at the software level should be investigated.

6. Conclusions

This systematic literature review introduces fault injection attacks on IoT systems with an emphasis on the various high-level effects of fault injection attacks on IoT software and fault models used to analyze the attack effects on IoT software. A thorough examination of the various methods proposed in the literature for dealing with such attacks is conducted by categorizing them into system-level attack detection and software vulnerability analysis methods. Unlike other related surveys, the combined analysis of attack detection and software vulnerability analysis methods provides a complete understanding of different methods proposed in the literature to counter fault injection attacks on IoT systems at the system level and the software level. This allows the creation of hybrid solutions that utilize concepts from both categories, such as the use of hardware emulation platforms and software fault injection tools to detect attacks at the software level.

Attack detection methods can be used in real-time but are limited due to the replication of fault injection attacks on the actual IoT system and the use of a separate physical hardware monitoring setup. Simulation-based software vulnerability analysis methods do not require a hardware setup but lack real-time usability in a live IoT environment. The limitations of both categories are addressed in this systematic literature review using techniques such as machine learning, dynamic code instrumentation tools, and hardware emulation platforms. The feasibility of utilizing software testing concepts to address the limitations of software vulnerability analysis methods against fault injection attacks is also discussed. The use of software testing concepts addresses existing challenges in the literature, such as identifying data corruption and modification in the software due to fault injection attacks on IoT systems. A hybrid software-level attack detection method is proposed using an instruction-level software fault injection tool and supervised machine learning.

Fault injection attacks are increasingly used to target a wide range of IoT systems and devices and are also used as a precursor to side-channel analysis attacks in order to disrupt the system functioning or steal confidential data. Various types of software executing in IoT systems and devices are affected by such attacks, including device drivers, access control software, and application software. This systematic literature review provides a comprehensive overview of various methods used to counter fault injection attacks at the system level and the software level, along with consideration of their limitations and potential solutions for future research.

Author Contributions: Writing—original draft preparation: A.G.; supervision and writing—review and editing: Q.H.M.; supervision and writing—review and editing: A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Yuce, B.; Schaumont, P.; Witteman, M.F. Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. *J. Hardw. Syst. Secur.* **2018**, *2*, 111–130. [CrossRef]
2. Barenghi, A.; Breveglieri, L.; Koren, I.; Naccache, D. Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures. *Proc. IEEE* **2012**, *100*, 3056–3076. [CrossRef]
3. Jiang, W. Machine Learning Methods to Detect Voltage Glitch Attacks on IoT/IIoT Infrastructures. *Comput. Intell. Neurosci.* **2022**, *2022*, 6044071. [CrossRef] [PubMed]
4. Sahu, A.; Mao, Z.; Wlazlo, P.; Huang, H.; Davis, K.; Goulart, A.; Zonouz, S. Multi-Source Multi-Domain Data Fusion for Cyberattack Detection in Power Systems. *IEEE Access* **2021**, *9*, 119118–119138. [CrossRef]
5. Benevenuti, F.; Kastensmidt, F.L. Evaluation of fault attack detection on SRAM-based FPGAs. In Proceedings of the 2017 18th IEEE Latin American Test Symposium (LATS), Punta del Este, Uruguay, 27–29 October 2017; pp. 1–6. [CrossRef]
6. Breier, J.; Bhasin, S.; He, W. An electromagnetic fault injection sensor using Hogge phase-detector. In Proceedings of the 2017 18th International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 14–15 March 2017; pp. 307–312. [CrossRef]
7. Khosrowjerdi, H.; Meinke, K.; Rasmusson, A. Virtualized-Fault Injection Testing: A Machine Learning Approach. In Proceedings of the 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST), Vasteras, Sweden, 9–13 April 2018; pp. 297–308. [CrossRef]
8. TGiven-Wilson; Jafri, N.; Legay, A. Combined software and hardware fault injection vulnerability detection. *Innov. Syst. Softw. Eng.* **2020**, *16*, 101–120. [CrossRef]
9. Rivière, L.; Potet, M.-L.; Le, T.-H.; Bringer, J.; Chabanne, H.; Puys, M. Combining High-Level and Low-Level Approaches to Evaluate Software Implementations Robustness Against Multiple Fault Injection Attacks. *Found. Pract. Secur.* **2015**, *8930*, 92–111.
10. Delarea, S.; Oren, Y. Practical, Low-Cost Fault Injection Attacks on Personal Smart Devices. *Appl. Sci.* **2022**, *12*, 417. [CrossRef]
11. Papadimitriou, A.; Nomikos, K.; Psarakis, M.; Aerabi, E.; Hely, D. You can detect but you cannot hide: Fault Assisted Side Channel Analysis on Protected Software-based Block Ciphers. In Proceedings of the 2020 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Frascati, Italy, 19–21 October 2020; pp. 1–6. [CrossRef]
12. Valencia, F.; Oder, T.; Güneysu, T.; Regazzoni, F. Exploring the Vulnerability of R-LWE Encryption to Fault Attacks. In Proceedings of the Fifth Workshop on Cryptography and Security in Computing Systems, Online, 24 January 2018; pp. 7–12. [CrossRef]
13. Barbu, G.; Thiebeauld, H.; Guerin, V. Attacks on Java Card 3.0 Combining Fault and Logical Attacks. In *International Conference on Smart Card Research and Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 148–163.
14. Dehbaoui, A.; Mirbaha, A.-P.; Moro, N.; Dutertre, J.-M.; Tria, A. Electromagnetic Glitch on the AES Round Counter. In *Constructive Side-Channel Analysis and Secure Design*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 17–31.
15. Dutertre, J.-M.; Menu, A.; Potin, O.; Rigaud, J.-B.; Danger, J.-L. Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures. *Microelectron. Reliab.* **2021**, *121*, 114133. [CrossRef]
16. Menu, A.; Dutertre, J.-M.; Potin, O.; Rigaud, J.-B.; Danger, J.-L. Experimental Analysis of the Electromagnetic Instruction Skip Fault Model. In Proceedings of the 2020 15th Design Technology of Integrated Systems in Nanoscale Era (DTIS), Marrakesh, Morocco, 1–3 April 2020; pp. 1–7. [CrossRef]
17. Breier, J.; Jap, D.; Chen, C.-N. Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES. In Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, Denver, CO, USA, 16 October 2015; pp. 99–103. [CrossRef]
18. Joye, M.; Tunstall, M. *Fault Analysis in Cryptography*; Springer: Berlin/Heidelberg, Germany, 2012; p. 147.
19. Kazemi, Z.; Hely, D.; Fazeli, M.; Beroulle, V. A Review on Evaluation and Configuration of Fault Injection Attack Instruments to Design Attack Resistant MCU-Based IoT Applications. *Electronics* **2020**, *9*, 1153. [CrossRef]
20. Polychronou, N.-F.; Thevenon, P.-H.; Puys, M.; Beroulle, V. A Comprehensive Survey of Attacks without Physical Access Targeting Hardware Vulnerabilities in IoT/IIoT Devices, and Their Detection Mechanisms. *ACM Trans. Des. Autom. Electron. Syst.* **2021**, *27*, 1–35. [CrossRef]
21. Dureuil, L.; Potet, M.-L.; de Choudens, P.; Dumas, C.; Clédière, J. From Code Review to Fault Injection Attacks: Filling the Gap Using Fault Model Inference. In *Smart Card Research and Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 107–124.
22. Eslami, M.; Ghavami, B.; Raji, M.; Mahani, A. A survey on fault injection methods of digital integrated circuits. *Integration* **2020**, *71*, 154–163. [CrossRef]

23. Qasem, A.; Shirani, P.; Debbabi, M.; Wang, L.; Lebel, B.; Agba, B.L. Automatic Vulnerability Detection in Embedded Devices and Firmware: Survey and Layered Taxonomies. *ACM Comput. Surv.* **2021**, *54*, 1–42. [CrossRef]
24. Lou, X.; Zhang, T.; Jiang, J.; Zhang, Y. A Survey of Microarchitectural Side-Channel Vulnerabilities, Attacks, and Defenses in Cryptography. *ACM Comput. Surv.* **2021**, *54*, 1–37. [CrossRef]
25. Potestad-Ordóñez, F.E.; Tena-Sánchez, E.; Acosta-Jiménez, A.J.; Jiménez-Fernández, C.J.; Chaves, R. Hardware Countermeasures Benchmarking against Fault Attacks. *Appl. Sci.* **2022**, *12*, 2443. [CrossRef]
26. Shah, I.A.; Rajper, S.; ZamanJhanjhi, N. Using ML and Data-Mining Techniques in Automatic Vulnerability Software Discovery. *Int. J. Adv. Trends Comput. Sci. Eng.* **2021**, *10*, 2109–2126. [CrossRef]
27. Eceiza, M.; Flores, J.L.; Iturbe, M. Fuzzing the Internet of Things: A Review on the Techniques and Challenges for Efficient Vulnerability Discovery in Embedded Systems. *IEEE Internet Things J.* **2021**, *8*, 10390–10411. [CrossRef]
28. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering—A systematic literature review. *Inf. Softw. Technol.* **2009**, *51*, 7–15. [CrossRef]
29. Igarashi, H.; Shi, Y.; Yanagisawa, M.; Togawa, N. Concurrent faulty clock detection for crypto circuits against clock glitch based DFA. In Proceedings of the 2013 IEEE International Symposium on Circuits and Systems (ISCAS), Beijing, China, 19–23 May 2013; pp. 1432–1435. [CrossRef]
30. Moro, N.; Heydemann, K.; Encrenaz, E.; Robisson, B. Formal verification of a software countermeasure against instruction skip attacks. *J. Cryptogr. Eng.* **2014**, *4*, 145–156. [CrossRef]
31. Potet, M.-L.; Mounier, L.; Puys, M.; Dureuil, L. Lazart: A Symbolic Approach for Evaluation the Robustness of Secured Codes against Control Flow Injections. In Proceedings of the 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation, Cleveland, OH, USA, 31 March–4 April 2014; pp. 213–222. [CrossRef]
32. Shrivastwa, R.-R.; Guilley, S.; Danger, J.-L. Multi-source Fault Injection Detection Using Machine Learning and Sensor Fusion. In *Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 93–107.
33. Köylü, T.Ç.; Reinbrecht, C.R.W.; Hamdioui, S.; Taouil, M. RNN-Based detection of fault attacks on RSA. In Proceedings of the IEEE International Symposium on Circuits and Systems, Daegu, Korea, 22–28 May 2020; pp. 1–5. [CrossRef]
34. Richter-Brockmann, J.; Shahmirzadi, A.R.; Sasdrich, P.; Moradi, A.; Güneysu, T. Fiver–robust verification of countermeasures against fault injections. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 447–473. [CrossRef]
35. Lacombe, G.; Feliot, D.; Boespflug, E.; Potet, M.-L. Combining Static Analysis and Dynamic Symbolic Execution in a Toolchain to detect Fault Injection Vulnerabilities. September 2021. Available online: https://www.proofs-workshop.org/2021/papers/paper2.pdf (accessed on 20 April 2022).
36. Bréjon, J.-B.; Heydemann, K.; Encrenaz, E.; Meunier, Q.; Vu, S.-T. Fault Attack Vulnerability Assessment of Binary Code. In Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems, Valencia, Spain, 21 January 2019; pp. 13–18. [CrossRef]
37. Mahmoud, A.; Venkatagiri, R.; Ahmed, K.; Misailovic, S.; Marinov, D.; Fletcher, C.W.; Adve, S.V. Minotaur: Adapting Software Testing Techniques for Hardware Errors. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Providence, RI, USA, 13–17 April 2019; pp. 1087–1103. [CrossRef]
38. Deshpande, C.; Yuce, B.; Ghalaty, N.F.; Ganta, D.; Schaumont, P.; Nazhandali, L. A Configurable and Lightweight Timing Monitor for Fault Attack Detection. In Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, USA, 11–13 July 2016; pp. 461–466. [CrossRef]
39. Kaliorakis, M.; Tselonis, S.; Chatzidimitriou, A.; Gizopoulos, D. Accelerated microarchitectural Fault Injection-based reliability assessment. In Proceedings of the 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), Amherst, MA, USA, 12–14 October 2015; pp. 47–52. [CrossRef]
40. Höller, A.; Krieg, A.; Rauter, T.; Iber, J.; Kreiner, C. QEMU-Based Fault Injection for a System-Level Analysis of Software Countermeasures Against Fault Attacks. In Proceedings of the 2015 Euromicro Conference on Digital System Design, Madeira, Portugal, 26–28 August 2015; pp. 530–533. [CrossRef]
41. Rivière, L.; Bringer, J.; Le, T.-H.; Chabanne, H. A novel simulation approach for fault injection resistance evaluation on smart cards. In Proceedings of the 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Graz, Austria, 13–17 April 2015; pp. 1–8. [CrossRef]
42. Facon, A.; Guilley, S.; Ngo, X.; Nguyen, R.; Perianin, T.; Shrivastwa, R. High Precision EMFI Detector using Machine Learning and Sensor Fusion. Available online: https://www.cesar-conference.org/wp-content/uploads/2019/10/resume_IAD_soum15-2.pdf (accessed on 12 April 2022).
43. Dutertre, J.-M.; Mirbaha, A.-P.; Naccache, D.; Ribotta, A.-L.; Tria, A.; Vaschalde, T. Fault Round Modification Analysis of the advanced encryption standard. In Proceedings of the 2012 IEEE International Symposium on Hardware-Oriented Security and Trust, San Francisco, CA, USA, 3–4 June 2012; pp. 140–145. [CrossRef]
44. Lee, H. Framework and development of fault detection classification using IoT device and cloud environment. *J. Manuf. Syst.* **2017**, *43*, 257–270. [CrossRef]
45. Jiang, W.; Wen, L.; Zhan, J.; Jiang, K. Design optimization of confidentiality-critical cyber physical systems with fault detection. *J. Syst. Archit.* **2020**, *107*, 101739. [CrossRef]

46. Kazemi, Z.; Fazeli, M.; Hely, D.; Beroulle, V. Hardware Security Vulnerability Assessment to Identify the Potential Risks in A Critical Embedded Application. In Proceedings of the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), Napoli, Italy, 13–15 July 2020; pp. 1–6. [CrossRef]
47. Qiu, P.; Wang, D.; Lyu, Y.; Tian, R.; Wang, C.; Qu, G. VoltJockey: A New Dynamic Voltage Scaling-Based Fault Injection Attack on Intel SGX. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2021**, *40*, 1130–1143. [CrossRef]
48. Chen, Z.; Vasilakis, G.; Murdock, K.; Dean, E.; Oswald, D.; Garcia, F.D. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual, CA, USA, 11–13 August 2021; pp. 699–716.
49. Bossuet, L.; de Laulanié, L.; Chassagne, B. Multi-Spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks. In Proceedings of the Smart Card Research and Advanced Applications: 20th International Conference, CARDIS 2021, Lübeck, Germany, 11–12 November 2021.
50. Zavalyshyn, I.; Given–Wilson, T.; Legay, A.; Sadre, R.; Rivière, E. Chaos Duck: A Tool for Automatic IoT Software Fault-Tolerance Analysis. In Proceedings of the 2021 40th International Symposium on Reliable Distributed Systems (SRDS), Chicago, IL, USA, 20–23 September 2021; pp. 46–55. [CrossRef]
51. Grycel, J.; Schaumont, P. SimpliFI: Hardware Simulation of Embedded Software Fault Attacks. *Cryptography* **2021**, *5*, 15. [CrossRef]
52. Lattner, C.; Adve, V. LLVM: A compilation framework for lifelong program analysis & transformation. In Proceedings of the International Symposium on Code Generation and Optimization, CGO, San Jose, CA, USA, 20–24 March 2004; pp. 75–86. [CrossRef]
53. Patel, A.; Afram, F.; Ghose, K. Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors. In Proceedings of the 1st International Qemu Users' Forum, Grenoble, France, 18 March 2011; pp. 29–30.
54. Given-Wilson, T.; Jafri, N.; Lanet, J.-L.; Legay, A. An Automated Formal Process for Detecting Fault Injection Vulnerabilities in Binaries and Case Study on PRESENT. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, Australia, 1–4 August 2017; pp. 293–300. [CrossRef]
55. Padmanabhuni, B.M.; Tan, H.B.K. Buffer Overflow Vulnerability Prediction from x86 Executables Using Static Analysis and Machine Learning. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Washington, DC, USA, 1–5 July 2015; Volume 2, pp. 450–459. [CrossRef]
56. Li, J.; Yang, Y.; Sun, J.S.; Tomsovic, K.; Qi, H. ConAML: Constrained Adversarial Machine Learning for Cyber-Physical Systems. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, Hong Kong, China, 7–11 June 2021; pp. 52–66. [CrossRef]
57. Gravellier, J.; Dutertre, J.-M.; Teglia, Y.; Moundi, P.L.; Olivier, F. Remote Side-Channel Attacks on Heterogeneous SoC. In *Smart Card Research and Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 109–125.
58. Shepherd, C.; Markantonakis, K.; van Heijningen, N.; Aboulkassimi, D.; Gaine, C.; Heckmann, T.; Naccache, D. Physical fault injection and side-channel attacks on mobile devices: A comprehensive analysis. *Comput. Secur.* **2021**, *111*, 102471. [CrossRef]
59. Höller, R.; Haselberger, D.; Ballek, D.; Rössler, P.; Krapfenbauer, M.; Linauer, M. Open-Source RISC-V Processor IP Cores for FPGAs—Overview and Evaluation. In Proceedings of the 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, 10–14 June 2019; pp. 1–6. [CrossRef]
60. Gray, J. GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In Proceedings of the 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), Washington, DC, USA, 1–3 May 2016; pp. 17–20. [CrossRef]
61. Lee, J.; Chen, H.; Young, J.; Kim, H. RISC-V FPGA Platform Toward ROS-Based Robotics Application. In Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 31 August–4 September 2020; p. 370. [CrossRef]
62. Nashimoto, S.; Homma, N.; Hayashi, Y.; Takahashi, J.; Fuji, H.; Aoki, T. Buffer overflow attack with multiple fault injection and a proven countermeasure. *J. Cryptogr. Eng.* **2017**, *7*, 35–46. [CrossRef]
63. Dureuil, L.; Petiot, G.; Potet, M.-L.; Le, T.-H.; Crohen, A.; de Choudens, P. FISSC: A Fault Injection and Simulation Secure Collection. In *Computer Safety, Reliability, and Security*; Springer: Cham, Switzerland, 2016; pp. 3–11.
64. Tramer, F.; Boneh, D. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*; Curran Associates Inc.: Vancouver, BC, Canada, 2019; p. 32. [CrossRef]
65. Zhang, H.; Wang, J. Defense against adversarial attacks using feature scattering-based adversarial training. In *Advances in Neural Information Processing Systems*; Curran Associates Inc.: Vancouver, BC, Canada, 2019; p. 32. [CrossRef]
66. Miyato, T.; Maeda, S.; Koyama, M.; Ishii, S. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 1979–1993. [CrossRef]
67. Yamaguchi, F.; Maier, A.; Gascon, H.; Rieck, K. Automatic Inference of Search Patterns for Taint-Style Vulnerabilities. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 797–812. [CrossRef]
68. Kim, S.; Kim, R.Y.C.; Park, Y.B. Software vulnerability detection methodology combined with static and dynamic analysis. *Wirel. Pers. Commun.* **2016**, *89*, 777–793. [CrossRef]
69. Luk, C.-K.; Cohn, R.; Muth, R.; Patil, H.; Klauser, A.; Lowney, G.; Wallace, S.; Reddi, V.J.; Hazelwood, K. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *SIGPLAN Not.* **2005**, *40*, 190–200. [CrossRef]

70. Bruening, D.; Zhao, Q.; Kleckner, R. DynamoRIO: Dynamic Instrumentation Tool Platform. February 2009. Available online: https://www.dynamorio.org (accessed on 25 April 2022).
71. Bukasa, S.K.; Lashermes, R.; Lanet, J.-L.; Leqay, A. Let's Shock Our IoT's Heart: ARMv7-M under (Fault) Attacks. In Proceedings of the ARES 2018: Proceedings of the 13th International Conference on Availability, Reliability and Security, Online, 27 August 2018; p. 33. [CrossRef]