Fuzzing for Power Grids: A Comparative Study of Existing Frameworks and a New Method for Detecting Silent Crashes in Control Devices

Marie Louise Uwibambe, Yanjun Pan, Qinghua Li
Department of Electrical Engineering and Computer Science
University of Arkansas
{uwibambe, yanjunp, qinghual}@uark.edu

***Abstract*—Power grids serve as the backbone of modern society, supplying electricity for a wide range of critical uses, including the residential, commercial, and industrial sectors. As the power grid infrastructure becomes more reliant on networked digital technologies, it also becomes more vulnerable to cyberattacks. To ensure the resilience and security of the power grid, it is necessary to identify security vulnerabilities in power electronic devices. Fuzzing is an effective technique for detecting security flaws in software/firmware by subjecting the system under test to a series of unexpected inputs, which elicits unexpected behavior such as system crashes. While some work has been done on fuzzing power grid systems, there still lacks a comprehensive, comparative study of existing solutions. Furthermore, new methods are still needed to detect unexpected behaviors of power electronic devices during fuzzing. In this paper, we first conduct a survey and a comparative study of existing fuzzing frameworks for power grid systems. We then hypothesize and verify the use of electromagnetic waves as a method to detect silent system crashes in power electronic devices, aiming to advance fuzzing in this field.**

***Index Terms*—Fuzzing, Power grid, Electromagnetic waves**

I. Introduction

With the increasing reliance on digital technologies, power grid systems are becoming more vulnerable to cyberattacks [1]. For instance, in 2015, Ukraine's power system suffered a major hack, leading to widespread blackouts [2]. Many cyberattacks exploit vulnerabilities in software, including device firmware, to compromise targeted systems [3], [4]. Much work has been done to detect and mitigate attacks in power grids [5], [6]. However, it is crucial to discover vulnerabilities before attackers exploit them.

Fuzzing has received considerable attention in recent years as an effective method for discovering software vulnerabilities by injecting invalid, malformed, or unexpected inputs into a software system and observing its ensuing behavior [7]. Advances in machine learning have substantially enhanced its effectiveness by performing optimizations, generating high-quality test inputs, and conducting crash analysis [8]–[10]. Symbolic and concolic executions have also been used to improve code coverage and explore deeper program paths [11], [12]. Moreover, the evolution of specialized fuzzing techniques has given rise to domain-specific fuzzing frameworks targeting diverse areas, including kernels [13], database systems [14], and embedded systems [15].

Fuzzing has also been applied to power grids in some recent work such as [16]–[19]. However, significant limitations persist in adapting fuzzing techniques to the realm of power grids. The distinctive attributes of power grids, such as utilization of proprietary software and adherence to safety considerations, pose significant challenges necessitating tailored fuzzing techniques. Moreover, while a system crash during fuzzing typically serves as an indicative marker of software flaws, power grid devices frequently experience silent crashes, and as a result, defects and vulnerabilities remain unseen due to the absence of necessary interfaces.

In this paper, we first conduct a survey and a comparative study of existing fuzzing frameworks for power grid systems, and point out their limitations. To the best of our knowledge, this is the first comparative study of fuzzing techniques for power grid systems. Then, in order to enhance fuzzing in power electronic devices, we propose the utilization of electromagnetic (EM) waves emitted by the device hosting the program-under-test to detect silent crashes.

The rest of the paper is organized as follows. Section II discusses the unique requirements of fuzzing for power grids. Section III provides a comparison of fuzzing frameworks. Section IV discusses how we leverage EM waves to detect silent crashes, followed by our conclusion.

II. Fuzzing for the Power Grid

While power grid fuzzing follows a similar structure to that of software applications, it possesses unique characteristics that demand special consideration, particularly in the context of target identification, input delivery, and system monitoring. We will explore these aspects in the following.

A. Target Identification

Unlike traditional software applications, power grids consist of a complex network of interconnected components, including control systems, sensors, and physical devices that communicate via various protocols. The complexity of these systems makes it challenging to fuzz all the components together and demands for specialized divide-and-conquer approaches to make the problem tractable. When fuzzing power grids, usually one component is targeted, and only the code within its reach is fuzzed. The most common power grid fuzzing targets are human-machine interfaces (HMIs) [20], smart meters [21], [22], and programmable logic controllers (PLCs) [23], [24].

B. Input Delivery

Due to the prevalence of proprietary software, sending inputs (test cases) to the system-under-test is challenging for power grids. Consequently, fuzzing power grid systems requires domain-specific knowledge, including understanding the protocols and data formats utilized in the energy industry, which can differ significantly from those employed in software applications. Reverse engineering techniques have been utilized to comprehend the structure, message formats, and communication patterns of proprietary protocols. This helps design test cases that can be accepted by the system-under-test. Various machine learning techniques have been employed to analyze the protocol message structure and create fuzzing packets [25]–[27].

C. System Behavior Monitoring

Monitoring the behavior of power electronic devices and controllers is different from monitoring regular software behaviors. To enable seamless monitoring of the tested system, several efforts have been made to handle the complex nature of power electronics. [28] uses an address sanitizer to monitor memory errors, while fieldfuzz [29] uses pinging to check the liveliness of the system. However, as mentioned earlier, tracing crashes in power grid devices is difficult due to silent crashes, system complexity, and limited access.

III. Survey and Comparative Study of Existing Fuzzing Frameworks

A. Existing Frameworks

Several studies have proposed tools uniquely designed for fuzzing power grid systems. IECFuzzer [30], for instance, is a mutation-based framework that focuses on IEC 61850 protocol implementations. This framework employs two types of mutations. The first is based on random generation, which produces binary data of random or indefinite lengths. This approach aids in broadening the coverage of variability. The second type is based on sample data variation. It involves extracting the value of the field intended for mutation from the data packet in the sample space and then applying operations like expansion, reduction, transformation, or replacement.

The work in [24] achieves parallel fuzzing of IEC 60870-5-104 protocol-based systems by assuming that given multiple identical instances of a device under test, their behaviors should be similar when fed identical test data and have the same access rights. Individual instances of the tested service run in Docker containers to offer an automated creation of the test environment. A framework [28] was designed to detect internal software flaws. It uses an address sanitizer (Asan) that uncovers memory-specific bugs. Asan forces the system under test to crash if a memory problem is triggered. However, this resulted in a performance overhead.

Dantas et al. [22] developed eFuzz, a fuzzing framework that detects flaws in smart metering devices based on the DLMS/COSEM communication protocol. eFuzz was tailored to interact with the meter under test through the Application Protocol Data Unit (APDU). Test cases are generated by altering APDU fields while also ensuring that they are accepted by the target. eFuzz can detect power grid protocol implementation errors by fuzzing the smart meter. However, it is still limited to only a portion of the DLMS/COSEM implementation. Other researchers employed a mutation-based technique to fuzz a physical DLMS/COSEM smart meter by submitting queries to the smart meter server [21].

PCFuzzer [18] is a fuzzing framework that detects vulnerabilities in programmable controllers by combining state guidance and network/physical condition monitoring. The state guidance is based on feedback obtained from the properties of bytes in protocol traffic.

The work in [20] connects an error to the responsible input using automated video analysis that examines the HMI's real-time video stream and maps it to the causal input recorded in the log history.

Several other fuzzing frameworks have been developed to find vulnerabilities in various Industrial Control Systems (ICS) applications, such as water treatment and waste treatment. Although these tools were not specifically designed with power grids in mind, they remain useful for power grids since their target protocols are employed in power grids. Most of these tools learn message semantics using recurrent neural network models, such as seq2seq, SeqGan, and GAN [18], [19], [25]–[27].

B. Analytical Comparison

Table I presents an analytical comparison of power grid fuzzing frameworks. They can be categorized based on various criteria, and we discuss some of them below.

- *Protocol.* The Protocol or communication standard used is critical in establishing the strategy for fuzzing. Since power grid systems use a variety of communication protocols (e.g., IEC 61850, DNP3, and Modbus), fuzzing frameworks must be customized to the complexities of each protocol to provide effective vulnerability discovery and exploitability evaluation. We show the communication protocols that are compatible with the framework. Some frameworks provide support for multiple communication protocols; those are labeled as "cross-protocol".
- *Target.* The fuzzing approach is significantly influenced by the target component under testing. The fuzzing technique might vary in terms of test input generation and protocol interactions, depending on whether the target component is a control system, smart meter, or other power electronic equipment.

TABLE I
FUZZING FRAMEWORKS FOR POWER GRIDS

	Framework	Protocol/Standard	Target	Environment	Testing Objective	Automation	Scope	Is source code needed?
1	Efuzz [22]	DLMS/COSEM	Smart meter	Device	Bugs	Fully automated	PG	No
2	(Wang, Chien-Lung,al., 2022) [21]	DLMS/COSEM	Smart meter	Device	Bugs	Fully automated	PG	No
3	IECFuzzer [30]	IEC61850	PLC	Device	Bugs	Semi-automated	ICS	No
4	(Ilgner;et al) [24]	IEC 60870-5-104	Protocol implementation	Simulated	Bugs	Fully automated	PG	No
5	LZFuzz [29]	Cross-protocol	SCADA equipment	Device	System Vulnerability	Fully automated	PG	Yes
6	ICSFuzz [23]	IEC 61131-3	PLC	Device	Bugs	Semi-automated	ICS	No
7	(Fen, et al) [21]	Cross-protocol	Protocol implementation	Simulated	System Vulnerability/ Bugs	Fully automated	ICS	No
8	MaskFuzzer [15]	Cross-protocol	Protocol implementation	Device	Bugs	Fully automated	ICS	No
9	PCFuzzer [18]	Cross-protocol	Programmable Controllers	Device	Bugs	Semi-automated	ICS	No
10	(Visky, Gábor, Lavrenovs, Maennel, 2021) [20]	IEC 61850	Protocol implementation	Device	System Vulnerability	Fully automated	PG	No
11	(Winter, et al) [28]	IEC 61850	Protocol implementation	Simulated	Bugs	Semi-automated	PG	No

- *Environment.* Due to the expenses associated with real physical testbeds, simulated environments are often utilized in power grid security analysis. We specify whether the framework was designed and tested on a real (physical device) or a simulated environment.
- *Testing objective.* Certain frameworks focus on maintaining overall system resilience by ensuring the system is not vulnerable, while others concentrate on identifying specific bugs. The first scenario is denoted as "System Vulnerability," and the second one is indicated as "bugs."
- *Scope.* Frameworks can be classified based on whether they are intended solely for Power Grid systems (PG) or are suitable for various Industrial Control Systems (ICS).

C. Experimental Comparison

The absence of publicly available benchmark datasets that cover a wide range of scenarios and vulnerabilities makes the experimental comparison for power grid fuzzing frameworks challenging. Furthermore, existing frameworks demonstrate various objectives, with rare instances where frameworks align in terms of their focus on certain environments, components, protocols, and other related attributes. As a result of their specialized features, making meaningful comparisons between different frameworks becomes complicated. However, testing commodity software found in power electric equipment is more doable at this time.

Power grids use modern PLCs that run commodity operating systems (e.g., Linux). These operating systems introduce additional vulnerabilities. Techniques for fuzzing Linux binaries in PLCs have been proposed [23], [31]. AFL variants [32] are the most prevalent frameworks used for commodity software fuzzing. Using the technique suggested in [31], we investigated their performance on Linux binaries found in PLCs. The system under test is an emulated instance of a Wago PFC-100 PLC (ARM), hosted on an 8GB memory Linux VM.

As shown in Table II, AFL++ outperforms AFL by 18 times in terms of path depth. Deeper paths indicate the exploration of complex and less frequently traversed code paths. Furthermore, AFL++ achieves a larger branch coverage and can easily discover new paths during the fuzzing session. Given these findings, AFL++ emerges as a preferable option for conducting efficient commodity software testing on PLCs.

TABLE II
AFL VARIANTS PERFORMANCE ON LINUX-BASED PLCS

	Branch coverage (%)	Path depth	New paths
AFL	2	1	0
AFL++	71.3	18	33

D. Limitations of Existing Fuzzing Frameworks

Poor versatility. Existing fuzzing frameworks for power grid systems suffer from scope restrictions. These frameworks frequently focus on certain communication protocols, fuzzing environments, or device types and models. An ideal power grid fuzzing solution should allow users to define various protocols and accommodate the heterogeneous nature of power grid components. Thus, it is important to explore ways to establish seamless compatibility with the rapidly evolving power grid systems, such as IEC61131-based systems, regardless of their unique characteristics.

Limited feedback. Generally, existing frameworks lack important feedback mechanisms such as the impact of test cases and coverage measurements. Effective feedback is crucial for further optimization and prioritizing remediation efforts.

Lack of test-beds. Current research efforts use minimal test-beds, often employing just one or no physical power grid device and relying on emulation for the rest. This approach achieves cost efficiency and mitigates potential damage associated with live power grid testing. However, the extensive use of simulation raises concerns regarding the effectiveness of such simulations in replicating the dynamic complexities of a live power grid. Thus, research efforts should look into the fidelity of simulations and the development of dedicated test-beds for fuzzing.

IV. Silent Crash Identification via Electromagnetic Waves

As previously mentioned, crash detection is an important method for monitoring the tested system during fuzzing. On the other hand, ICSs like power electronic devices have silent crashes that are difficult to detect. To assist with crash identification, we propose leveraging side-channel information from electromagnetic (EM) waves. EM wave analysis has been used as a non-intrusive way to monitor firmware changes in embedded systems [33] and detect control command attacks [34]. However, it has not been used to identify silent crashes in the context of fuzzing before.

The current flowing through the printed circuit board (PCB) of the power grid component generates a magnetic field, which further interacts with an electric field, resulting in the formation of an EM wave. This EM wave is emitted by the wire on the circuit, functioning as an antenna. External EM sensors positioned close to the PCB can capture these signals. Specifically, we utilize these near-field EM waves as a side channel to detect silent crashes on a device. If a silent crash occurs during fuzzing, it can be identified from normal execution by examining the changes in EM signals emitted by the device's chips.

A. Overview

We designed a statistical model to characterize both normal operations and crashes. This model relies on the mean value, a single number that serves as a representative of a list of numbers. By calculating the mean value for the measured EM waves, we obtain a representation of the signal's typical behavior, providing a reliable baseline for comparison.

This method accomplishes two goals: simplifying the representation of measurements, which is especially valuable when dealing with large and complex data, and facilitating the detection of deviations from the established baseline. These deviations may indicate abnormal situations, such as crashes.

B. Signal Analysis: Modeling and Prediction

Phase 1: Operation modeling. In this phase, we create a model representing the normal operation using the signals collected during the program's normal execution.

We utilize both negative and positive data points within the signal. A positive data point corresponds to a voltage measurement (that is generated from EM waves) when the signal's amplitude exceeds the predetermined trigger point. These positive values indicate the degree to which the voltage exceeds the trigger point. Similarly, negative data points are represented as negative numbers, indicating the extent to which the voltage falls below the trigger point.

We use the symbols $\mathbf{V}_n^+$ and $\mathbf{V}_n^-$ to denote the vectors holding positive and negative data points in normal conditions. Similarly, in the crash scenario, we use the symbols $\mathbf{V}_c^+$ and $\mathbf{V}_c^-$. We achieve the modeling in this stage by averaging the parameters discussed above, as shown in Eq.1.

$$\overline{V}_n^+ = mean(\mathbf{V}_n^+) \quad \text{and,} \quad \overline{V}_n^- = mean(\mathbf{V}_n^-). \tag{1}$$

Similarly, by using the signal collected in the presence of a crash, we model crashes with

$$\overline{V}_c^+ = mean(\mathbf{V}_c^+) \quad \text{and,} \quad \overline{V}_c^- = mean(\mathbf{V}_c^-). \tag{2}$$

Phase 2: Operation Identification. The parameters computed from the modeling are employed for signal analysis, facilitating the classification of signals into two distinct categories: normal execution and instances of crashes. This analysis is implemented through the following steps.

1) We first divide the signal into smaller chunks. A chunk, denoted as $\mathbf{C}$, consists of a block of consecutive data points, with each chunk containing exactly n data points (in our case, n is set to 10). We further segment each chunk into positive and negative data points by comparing them to a predetermined trigger point, resulting in two corresponding vectors: $\mathbf{C}^+$ and $\mathbf{C}^-$.
   The rationale behind this segmentation lies in the co-existence of both crash-related and normal execution traces within a single signal. Hence, we classify each individual chunk independently before combining the results to obtain the final classification. It is worth noting that while using a smaller value for n may seem more precise, it can lead to a higher rate of false negatives. Furthermore, this method reduces the computational workload required for prediction.
2) Next, we proceed with Phase 1, where we calculate the averages for both the negative and positive data points within each chunk.
$$\overline{C}^+ = mean(\mathbf{C}^+) \quad \text{and,} \quad \overline{C}^- = mean(\mathbf{C}^-) \tag{3}$$
3) We measure the distance, Δ, which quantifies the proximity of the chunk to either a crash-related or a normal operational trace. In both of these cases, the negative and positive data points are analyzed separately.
   The signal's proximity to normal operation is computed as shown below.
$$\Delta_n^+ = |\overline{C}^+ - \overline{V}_n^+| \quad \text{and,} \quad \Delta_n^- = |\overline{C}^- - \overline{V}_n^-| \tag{4}$$
   Similarly, for crash closeness, we compute the following.
$$\Delta_c^+ = |\overline{C}^+ - \overline{V}_c^+| \quad \text{and,} \quad \Delta_c^- = |\overline{C}^- - \overline{V}_c^-| \tag{5}$$
4) Next, we classify the chunks into two classes by applying the following conditions.
   Normal operation. $\Delta_n^+ < \Delta_c^+$ and $\Delta_n^- < \Delta_c^-$
   Crash. $\Delta_n^+ > \Delta_c^+$ and $\Delta_n^- > \Delta_c^-$
   If a chunk does not fall into either of the two categories, the chunk is divided into two distinct parts. The first part inherits the classification of the left adjacent chunk, while the second part inherits the classification of the right adjacent chunk.
5) Finally, we aggregate the prediction for each chunk to summarise the transition of states (normal to crash operations and vice-versa). We use the symbol Y to indicate an array of combined predictions and y_i to symbolize the prediction of the i^{th} chunk.
$$Y = \{y_1, y_2, y_3..., y_n\} \tag{6}$$

C. Experiment Setup

System-under-test. The experiment was conducted in a simulated environment hosted on a Linux-based system. The system under test comprises a server simulated using the PyModbusTCP library [35], along with two services (Service 1 and Service 2) functioning as clients. The two services communicate through the server as follows: Service 1 updates the server registers with the latest data, while Service 2 retrieves these updates from the server for its computations. Service 1 was exposed to various input values, including out-of-bounds values, to monitor the application's behavior in the event of a Service 1 crash.

To enhance the integrity of the experiment, other services were disabled, leaving only the core components necessary for the experiment. This configuration ensured the fidelity of the signals, protecting them from potential interference by background services.

Devices' configuration. We recorded EM signals using a Rigol Near Field Probe (NFP-3-P1 model) positioned 2 cm from the host machine's CPU. This probe has a testing range of 10 cm, ensuring that the EM signals in this range were properly collected.

The probe was connected to a Rigol DS1104Z oscilloscope with a 2 MHz sampling rate. For each experimental iteration, the samples were converted into a spectral representation of 1200 signal strength traces measured in voltage. We also experimented with a higher sample rate of 10 MHz; however, this change did not yield noticeable improvements. Consequently, we retained the original 2 MHz sampling rate, which effectively preserved essential signal information while maintaining manageable computational complexity.

The signal collection is illustrated in Fig. 1. Figure 2 depicts the complete experiment setup, encompassing both the EM wave collection process and subsequent analysis.

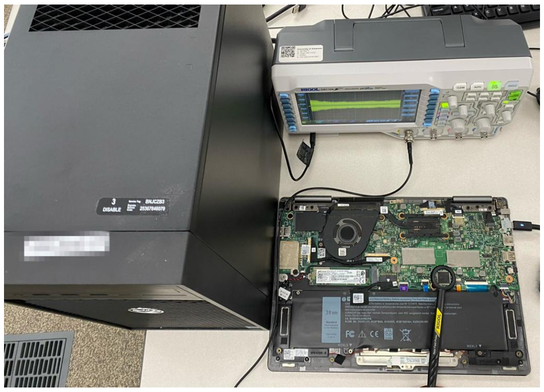

Fig. 1. The signal collection setup showing the probe placed over the PCB board and its Connection to an oscilloscope.

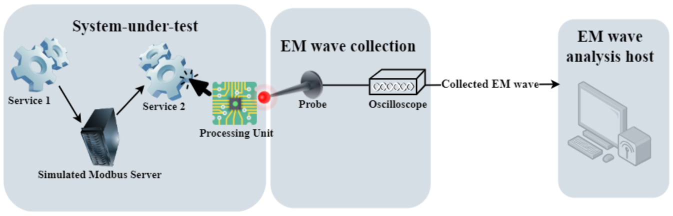

Fig. 2. Experiment setup.

TABLE III
COMMAND-SPECIFIC MODEL PERFORMANCE

Command type	**Accuracy(%)**
Register writing	97.5
Register Reading	95.283
External resource reading	98.0
Mathematical computation	95.28
Average performance	96.51

D. Performance Analysis

In our experimental trials, we introduced a variety of commands into the program-under-test, achieving an average accuracy of over 96%, as detailed in Table III. These commands include *register write*, *register reads*, *external resource access*, and various *mathematical computations*. The diversity in commands ensured that the model would remain effective even in the event of server-side randomness. Figure 3 further provides an example of crash detection using our model.

We observe that crashes caused by diverse triggering events have unique waveforms. This disparity in waveforms has the potential to cause confusion during classification operations, especially when the system encounters events that are not covered by the model. A naive approach to address this would be to create an all-encompassing model that accounts for a wide range of crash situations. However, when it comes to silent crashes, the basic issue is that not all causative events are known, thus, making full coverage an impossible goal. An alternative approach is to model the waveform patterns of normal system operations while treating any other waveform as potentially suspicious.

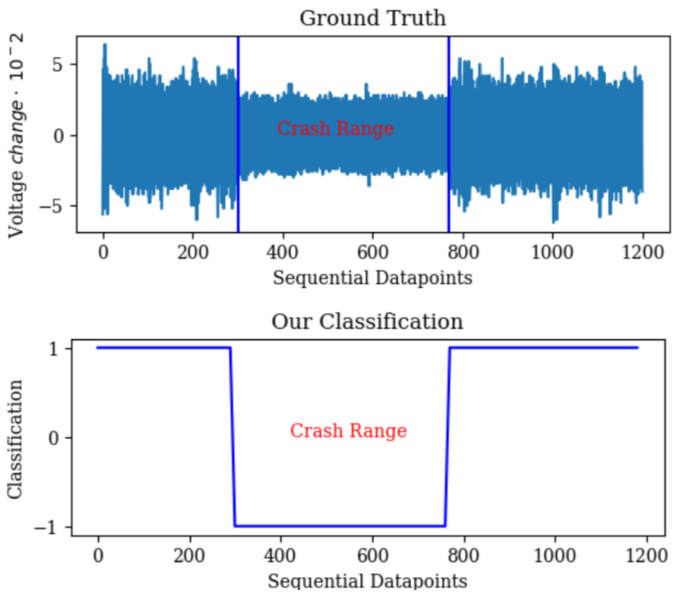

Fig. 3. An example of crash detection using the proposed method.

Conclusion

In this paper, we surveyed and compared fuzzing frameworks for power grid systems. We highlighted the unique requirements in fuzzing power grid systems and proposed leveraging electromagnetic waves to monitor silent crashes in power electronic devices. Future researches can concentrate on developing common datasets, benchmarks, test-beds, and best practices for power grid fuzzing. Another direction is to research machine learning-based innovative techniques of drawing feedback from power grid side channels such as power use and EM emissions to monitor target behaviors in fuzzing.

Acknowledgment

This work is supported in part by the NSF under award number 1751255. We also thank the National Center for Reliable Electric Power Transmission (NCREPT) for granting us access to their testing resources.

References

[1] Chih-Che Sun, Adam Hahn, and Chen-Ching Liu. Cyber security of a power grid: State-of-the-art. *International Journal of Electrical Power & Energy Systems*, 99:45–56, 2018.

[2] Julia E Sullivan and Dmitriy Kamensky. How cyber-attacks in ukraine show the vulnerability of the us power grid. *The Electricity Journal*, 30(3):30–35, 2017.

[3] Philip Huff and Qinghua Li. Towards automated assessment of vulnerability exposures in security operations. In *EAI International Conference on Security and Privacy in Communication Networks (SecureComm)*, pages 62–81, 2021.

[4] Fengli Zhang, Philip Huff, Kylie McClanahan, and Qinghua Li. A machine learning-based approach for automated vulnerability remediation analysis. In *IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2020.

[5] Fengli Zhang, Yatish Dubasi, Wei Bao, and Qinghua Li. Detection and localization of data forgery attacks in automatic generation control. *IEEE Access*, pages 1–1, 2023.

[6] Yatish Dubasi, Ammar Khan, Qinghua Li, and Alan Mantooth. Security vulnerability and mitigation in photovoltaic systems. In *IEEE International Symposium on Power Electronics for Distributed Generation Systems (PEDG)*, pages 1–7, 2021.

[7] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang. Fuzzing: a survey for roadmap. *ACM Computing Surveys*, 54(11s):1–36, 2022.

[8] Jianzhong Su, Hong-Ning Dai, Lingjun Zhao, Zibin Zheng, and Xiapu Luo. Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, 2022.

[9] Pei-Yi Lin, Chia-Wei Tien, Ting-Chun Huang, and Chin-Wei Tien. Icpfuzzer: proprietary communication protocol fuzzing by using machine learning and feedback strategies. *Cybersecurity*, 4(1):1–15, 2021.

[10] Yan Wang, Peng Jia, Luping Liu, Cheng Huang, and Zhonglin Liu. A systematic review of fuzzing based on machine learning techniques. *PloS one*, 15(8):e0237749, 2020.

[11] Lannan Luo, Qiang Zeng, Bokai Yang, Fei Zuo, and Junzhe Wang. Westworld: Fuzzing-assisted remote dynamic symbolic execution of smart apps on iot cloud platforms. In *Annual Computer Security Applications Conference*, pages 982–995, 2021.

[12] Sebastian Poeplau and Aurélien Francillon. Symqemu: Compilation-based symbolic execution for binaries. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

[13] Kyungtae Kim, Dae R Jeong, Chung Hwan Kim, Yeongjin Jang, Insik Shin, and Byoungyoung Lee. Hfl: Hybrid fuzzing on the linux kernel. In *NDSS*, 2020.

[14] Zu-Ming Jiang, Jia-Ju Bai, and Zhendong Su. Dynsql: Stateful fuzzing for database management systems with complex and valid sql query generation. In *USENIX Security Symposium*, 2023.

[15] Joobeom Yun, Fayozbek Rustamov, Juhwan Kim, and Youngjoo Shin. Fuzzing of embedded systems: A survey. *ACM Computing Surveys*, 55(7):1–33, 2022.

[16] Honggang Wu, Li Gong, Ao Liu, Yi Zhang, and Jianwei Yang. Enipfuzz: A seqgan-based ethernet/ip protocol fuzzing test framework. In *IEEE International Conference on Electronics Technology (ICET)*, pages 1278–1282, 2022.

[17] Dongliang Fang, Zhanwei Song, Le Guan, Puzhuo Liu, Anni Peng, Kai Cheng, Yaowen Zheng, Peng Liu, Hongsong Zhu, and Limin Sun. Ics3fuzzer: A framework for discovering protocol implementation bugs in ics supervisory software by fuzzing. In *Annual Computer Security Applications Conference*, pages 849–860, 2021.

[18] Puzhuo Liu, Yaowen Zheng, Zhanwei Song, Dongliang Fang, Shichao Lv, and Limin Sun. Fuzzing proprietary protocols of programmable controllers to find vulnerabilities that affect physical control. *Journal of Systems Architecture*, 127:102483, 2022.

[19] Weifeng Sun, Bowei Zhang, Jianqiao Ding, and Min Tang. Maskfuzzer: A maskgan-based industrial control protocol fuzz testing framework. In *IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 51–57, 2022.

[20] Gábor Visky, Arturs Lavrenovs, and Olaf Maennel. Status detector for fuzzing-based vulnerability mining of iec 61850 protocol. In *European Conference on Cyber Warfare and Security*, page 454, 2021.

[21] Chien-Lung Wang, Jou-An Shih, I-En Liao, and Chun-Tsai Chien. An evaluation of cybersecurity risks of dlms/cosem smart meter using fuzzing testing. In *IET International Conference on Engineering Technologies and Applications (IET-ICETA)*, pages 1–2. IEEE, 2022.

[22] Henrique Dantas, Zekeriya Erkin, Christian Doerr, Raymond Hallie, and Gerrit van der Bij. efuzz: A fuzzer for dlms/cosem electricity meters. In *Workshop on Smart Energy Grid Security*, pages 31–38, 2014.

[23] Dimitrios Tychalas, Hadjer Benkraouda, and Michail Maniatakos. {ICSFuzz}: Manipulating {I/Os} and repurposing binary code to enable instrumented fuzzing in {ICS} control applications. In *USENIX Security Symposium*, pages 2847–2862, 2021.

[24] Petr Ilgner and Radek Fujdiak. Fuzzing framework for iec 60870-5-104 protocol. In *International Conference on Computer Science and Software Engineering*, pages 190–194, 2022.

[25] Wenpeng Wang, Zhixiang Chen, Ziyang Zheng, and Hui Wang. An adaptive fuzzing method based on transformer and protocol similarity mutation. *Computers & Security*, 129:103197, 2023.

[26] Tao Fen, Deming Li, and Zhanting Yuan. An industrial network protocol fuzzing framework based on deep adversarial networks. In *International Conference on Computer Engineering and Application (ICCEA)*, pages 590–596. IEEE, 2023.

[27] Zhenhua Yu, Haolu Wang, Dan Wang, Zhiwu Li, and Houbing Song. Cgfuzzer: A fuzzing approach based on coverage-guided generative adversarial networks for industrial iot protocols. *IEEE Internet of Things Journal*, 9(21):21607–21619, 2022.

[28] Eugen Winter and Michael Rademacher. Fuzzing of scada protocols used in smart grids. *Energy Inform*, 3:1–3, 2020.

[29] Andrei Bytes, Prashant Hari Narayan Rajput, Constantine Doumanidis, Nils Ole Tippenhauer, Michail Maniatakos, and Jianying Zhou. Fieldfuzz: In situ blackbox fuzzing of proprietary industrial automation runtimes via the network. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2023.

[30] Tengfei Tu, Hua Zhang, Boqin Qin, and Zhuo Chen. A vulnerability mining system based on fuzzing for iec 61850 protocol. In *International Conference on Frontiers of Manufacturing Science and Measuring Technology (FMSMT)*, pages 589–597, 2017.

[31] Dimitrios Tychalas and Michail Maniatakos. Iffset: In-field fuzzing of industrial control systems using system emulation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 662–665. IEEE, 2020.

[32] M. Zalewski. American fuzzy lop, 2017. http://lcamtuf.coredump.cx/afl/.

[33] Yi Han, Sriharsha Etigowni, Hua Liu, Saman Zonouz, and Athina Petropulu. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In *ACM conference on computer and communications security (CCS)*, pages 1095–1108, 2017.

[34] Kylie McClanahan, Jingyao Fan, Qinghua Li, and Guohong Cao. Protecting control commands using low-cost em sensors in the smart grid. In *IEEE Conference on Communications and Network Security (CNS)*, 2023.

[35] PyModbusTCP. Version 0.2.0. https://github.com/sourceperl/pyModbusTCP.