GAFLERNA Ahoy! Integrating EM Side-Channel Analysis into Traditional Fuzzing Workflows

Jorge Barredo
IKERLAN, Mondragon Unibertsitatea
Arrasate/Mondragón, Spain
jbarredo@ikerlan.es

Justyna Petke, David Clark
University College London
London, United Kingdom
j.petke|david.clark@ucl.ac.uk

Daniel Blackwell
University College London
London, United Kingdom
daniel.blackwell.14@ucl.ac.uk

Maialen Eceiza
IKERLAN
Arrasate/Mondragón, Spain
meceiza@ikerlan.es

Jose Luis Flores
University of the Basque Country
Donostia/San Sebastián, Spain
joseluis.flores@ehu.eus

Mikel Iturbe
Mondragon Unibertsitatea
Arrasate/Mondragón, Spain
miturbe@mondragon.edu

Abstract

Fuzzing, a powerful tool for vulnerability discovery, is limited by the coarse-grained, binary nature of its crash detection oracle. The use of sanitizers strengthens this oracle but requires recompilation or binary rewriting, and is limited to known patterns of vulnerabilities. We investigate an alternative way to strengthen the implicit oracle that is suitable for small (IoT-sized) devices: electromagnetic (EM) side-channel analysis. By integrating this into a fuzzing campaign we are able to detect anomalous program states through physical execution patterns. GAFLERNA, our EM-enhanced AFL++ framework, achieves a 87% correlation with sanitizer findings in the best case, without modifying the executable, while discovering 104 *new paths* to known crashes across four real-world programs. This reveals the potential for hardware-level feedback to extend fuzzing and analyse IoT programs where only the binary code is available.

CCS Concepts

• **Security and privacy → Software security engineering**.

Keywords

Fuzzing, Side-channel Analysis

ACM Reference Format:
Jorge Barredo, Justyna Petke, David Clark, Daniel Blackwell, Maialen Eceiza, Jose Luis Flores, and Mikel Iturbe. 2025. GAFLERNA Ahoy! Integrating EM Side-Channel Analysis into Traditional Fuzzing Workflows. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3696630.3728497

1 INTRODUCTION

Program analysis has traditionally relied predominantly on software-level observations. Whilst grey-box fuzzing has proven effective for security testing [8, 10, 25], and sanitizers make it possible for these fuzzers to detect a greater range of error types, both approaches remain constrained by their software-level nature, unable to identify anomalous execution patterns visible only through hardware behaviour. Furthermore, coverage feedback and sanitizers cannot be used when fuzzing IoT devices where the firmware cannot be obtained. We propose feeding electromagnetic (EM) Side-Channel Analysis (SCA) [21, 31] into fuzzing workflows to guide the fuzzer using program behaviours observed through physical execution; this could reveal vulnerabilities invisible to traditional means.

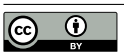
The evolution of vulnerability detection reflects our understanding of how bugs manifest in modern systems. Traditional fuzzing relies predominantly on crashes caused by interactions between the program and the OS, e.g. segmentation faults, out-of-memory errors, but sanitizers have expanded detection capabilities, e.g., AddressSanitizer (ASan) [35] catches use-after-free and out-of-bounds bugs, MemorySanitizer (MSan) [38] detects uninitialized memory usage. However, these approaches remain limited to software-level.

An EM SCA approach is relevant for Internet of Things (IoT) devices, whose firmware is challenging to emulate and analyse – if it can be obtained at all. Fortunately, they typically operate in low-noise environments that enable precise EM measurements [34]. These signals can provide insights into program behaviour, e.g. via unexpected power consumption [15] or EM emissions [17, 24, 27].

The research community has traditionally focused on software-level feedback mechanisms, e.g., `RIoTFuzzer` [23], while recent work exploring hardware-level fuzzing has limited scope. Multiple tools leverage SCA for differential analysis [29], including `DifFuzz` [28], `QFuzz` [30], `ct-fuzz` [16] and Brennan *et al.* [6]; however they all use the SCA to detect information leaks, and the SCA provides no feedback to help guide the fuzzer. Similarly, Basu *et al.* [4] and `SIGFuzz` [32] demonstrate promising hardware analysis capabilities but do not make this feedback available to the fuzzer. Sperl *et al.* [37] *do* use physical features as coverage proxies to guide the fuzzer, but their method applies to a limited set of devices.

We introduce `GAFLERNA`, a novel framework that enhances vulnerability detection by combining `AFL++` with feedback from EM-based SCA. Our approach analyses the EM responses generated by the execution of the fuzzing corpus inputs from a target program to guide the fuzzer toward potentially vulnerable program states. We seek to answer the following questions:

RQ1: Can EM-based SCA provide detection capabilities comparable to sanitizer instrumentation? Given that many programs cannot be augmented with sanitizers (e.g., black-box fuzzing where the binary is not available to modify), we investigate whether SCA can offer similar detection capabilities through hardware-level observations.



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3728497

RQ2: How effectively can EM anomalies guide fuzzing toward discovering additional vulnerabilities? We examine whether anomalous EM signals can direct the fuzzer toward vulnerable execution paths.

Our initial evaluation across 4 real-world programs shows promising results. While standard AFL++ discovered 23 paths to crashes, GAFLERNA revealed 1,714 anomalous states leading to 116 crash paths. Notably, 87% of these findings correlated with sanitizer detections in the best case, improving root cause analysis and yielding more comprehensive regression test sets for patch verification. This demonstrates the potential of EM-based SCA feedback in fuzzing workflows, particularly where sanitizer augmentation is unfeasible.

Contributions. Here, we propose a methodology that provides EM SCA feedback to grey-box fuzzing (Section 2); we present GAFLERNA, that implements the methodology (Section 3); and we demonstrate its effectiveness across a range of real-world programs (Section 4).

2 METHODOLOGY

This section outlines the methodology to provide EM-SCA feedback to traditional grey-box fuzzing workflows. An exploration fuzzing campaign generates a corpus of inputs (Section 2.1), while a calibration phase (Section 2.2) collects EM responses from the Device Under Test (DUT) to train anomaly detection algorithms (Figure 1). Each corpus input is then executed in the DUT, with its EM response analysed for anomalies (Section 2.3). We then filter out non-anomalous inputs and use the remaining ones as seeds for a subsequent fuzzing campaign of equal duration (Section 2.3.3).

2.1 Exploration Fuzzing phase

An instrumented program undergoes conventional fuzzing with initial seeds for a fixed duration. We prioritise developer-provided seeds, which maximise the variety of functionality in the baseline.

2.2 Side-Channel Calibration phase

Calibration is necessary to establish a baseline of non-anomalous EM responses from the exploration fuzzing phase (Figure 1). We assume initial seeds do not trigger vulnerabilities. We collect EM responses caused by executing the seeds – these are used to train the anomaly detection model. This calibration phase establishes a baseline for normal hardware behaviour patterns, recognizing that many valid execution paths not covered by seeds will produce similar patterns. Only significant deviations from these patterns are flagged as anomalous, rather than all previously unseen paths.

These anomalies may indicate underlying hardware behavioural irregularities, such as cache trashing, or infinite loops.

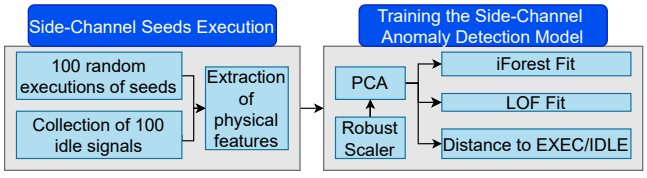


Figure 1: Methodology for the side-channel calibration phase.

2.2.1 Side-Channel Seeds Execution. We randomly execute seeds in the DUT until we collect EM responses, labeled as *EXEC* signals (100 in our case). With our seed corpora containing fewer than 100 seeds, some seeds are executed multiple times. Furthermore, the same number of idle signals (*IDLE*) are collected from the DUT when no program is running, establishing a baseline that helps us distinguish between actual program execution patterns and background EM emissions.

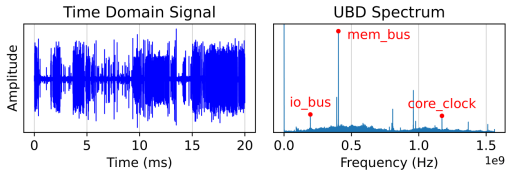


Figure 2: Sample EM signal in time domain (left) and frequency domain (right). Frequency bands correspond to specific hardware components (core clock, memory, I/O bus). Note that even at idle, these signals show non-flat activity.

Hardware components are identifiable in the frequency domain as they operate at specific frequencies (core clock, memory bus, I/O bus as shown in Figure 2). Each EM signal is captured with an EM probe and analysed to extract physical features (54 in our case). This feature extraction strategy reduces storage requirements while preserving signal characteristics. Quantitatively, a raw 10 ms signal comprising one million samples occupies 7.63 MB of memory, whereas its corresponding set of extracted features requires only 1.4 KB, enabling efficient storage for extended fuzzing campaigns.

2.2.2 Training the Side-Channel Anomaly Detection Model. *EXEC* and *IDLE* traces undergo normalization using Robust Scaler [2] and reduction through Principal Component Analysis (PCA) [1].

The preprocessed traces serve as training data for two well-established outlier detection procedures: Isolation Forest (iForest) [22] and Local Outlier Factor (LOF) [7], chosen for their complementary strengths in anomaly detection [9]. iForest detects global outliers effectively, LOF local ones. Other algorithms, identifying pattern deviations, would likely produce similar outcomes.

2.3 Side-Channel Operational phase

Once calibration is done, each corpus input from the initial exploratory fuzzing is executed in the DUT, and its EM response is analysed to detect whether it is anomalous, revealing program states that may indicate vulnerabilities without triggering crashes detectable by binary oracles. Any non-anomalous inputs are removed from the corpus — only the anomalous ones are retained (Figure 3).

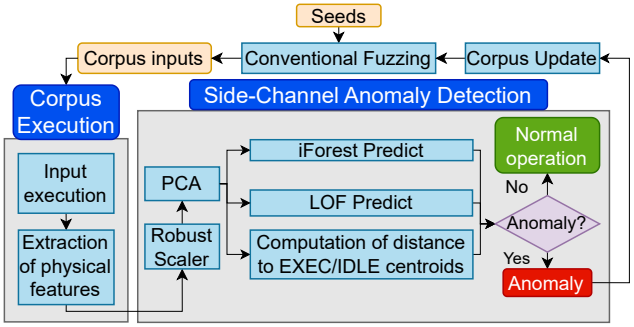


Figure 3: Methodology for the operational phase.

2.3.1 Side-Channel Corpus Execution. To prevent false positives, each corpus input is executed in the DUT multiple times (10 in our experiments). The EM response closest to the median of these executions is selected, and undergoes Robust Scaler and PCA.

2.3.2 Side-Channel Anomaly Detection and Corpus Update. Corpus signals are evaluated using the anomaly detection model, with inputs marked anomalous if the three thresholds are met. The corpus is filtered to retain anomalous inputs, guiding the fuzzer toward vulnerable paths. Detection confidence is quantified as follows:

$$c_i = 0.3 * \frac{IFthold - IF_i}{IFthold} + 0.3 * \frac{LOF_i}{LOFthold} + 0.2 * \frac{min(dEXEC_i, dIDLE_i)}{dThold} + 0.2 * \frac{min(dEXEC_i, dIDLE_i)}{max(dEXEC_i, dIDLE_i)} \quad (1)$$

where *IF* denotes *Isolation Forest*, *LOF* denotes *Local Outlier Factor*, and *dEXEC*/*dIDLE* denotes the normalized distance of a trace to the centroids of the *EXEC*/*IDLE* clusters.

2.3.3 Fuzzing Campaign Resumption. We employ the anomalous corpus inputs from the exploratory campaign as seed inputs for a follow-up fuzzing campaign of equal duration. While our methodology focuses on this single iteration, the process could theoretically be repeated, with each campaign's anomalous inputs serving as the seeds for the subsequent iteration.

3 STUDY DESIGN

We developed GAFLERNA, a framework that augments AFL++ [11], one of the most widely-used fuzzers. While AFL++ and other grey-box fuzzers predominantly rely on code coverage for guidance, GAFLERNA enables black-box detection of anomalies that manifest in hardware behaviour but do not result in crashes.

Core Architecture. GAFLERNA extends AFL++ through an EM analysis that extracts execution patterns from hardware signals, and a fuzzing control layer that integrates these insights into AFL++.

Fuzzing Setup. We leverage AFL++ for fuzzing and Claude.ai (https://claude.ai/) for seed generation, following recent research showing AI-generated seeds improve bug detection [5, 26, 33]. For each program we generated 4-16 initial seeds (Table 2) with program-specific valid inputs, preferring AI-generated seeds over developer seeds for consistency across targets.

Side-Channel Setup. We employ as DUT a Raspberry Pi 3B (RPi 3B) with a 1.2 GHz quad-core ARM Cortex-A53 processor. To ensure clean signal capture, we run on a single core at maximum frequency. A **Langer RF-R 0.3-3 H-Field Probe** captures the EM signals, positioned using heat-map guidance [39]. The setup includes a Tektronik MSO56 oscilloscope triggered by a RPi's GPIO for precise correlation between program states and hardware behaviour.

3.1 Evaluation Procedure

To address RQ1, we analysed detection results, visualising fuzzing corpus distributions as spirals – where the centre represents initial seeds and expansion reflects new inputs – and compared GAFLERNA's performance against binaries for the two programs recompiled with AddressSanitizer enabled. For RQ2, we evaluated three one-hour configurations: traditional fuzzing, traditional fuzzing seeded with crash sources from exploratory fuzzing (to assess whether conventional fuzzing can find alternative paths to known vulnerabilities when guided directly to vulnerable program regions), and our approach, each with 30-minute exploration fuzzing followed by 30-minute fuzzing resumption for each configuration.

Target Selection. We selected four real-world programs with known vulnerabilities (Table 1). We compiled two targets with AddressSanitizer and two without onto RPi 3B, evaluating GAFLERNA in contexts where ASan augmentation is unavailable. This setup shows how EM signals can reveal anomalies missed by conventional fuzzing and sanitizers, especially memory-related issues.

Table 1: Evaluated programs with known vulnerabilities, marked with CVE numbers. Sanitizer name provided if used.

Program Name	Sanitizer	CVE Number	Source
`OpenSSL-1.0.1f`	ASan	CVE-2014-0160	[13]
`md2roff 1.7`	-	CVE-2022-34913	[14]
`xpdf 3.02`	-	CVE-2019-13288	[19]
`libxml2-v2.9.2`	ASan	CVE-2015-8317	[13]

4 STUDY RESULTS

4.1 RQ1: Detection capability of side-channel anomalies in conventional fuzzing

Table 2 shows the proportion of corpus inputs flagged as anomalous by the SCA at the end of the exploratory fuzzing campaign, ranging from 76.00% in OpenSSL to 0.76% in libxml2. High detection confidence scores (Equation 1) support the reliability of these anomalies.

Table 2: Detected anomalies in the fuzzing corpus

Program Name	#seeds	Corpus Entries	Anomalies (corpus %)	Detection confidence (%)
`OpenSSL`	16	400	304 (76.00%)	72.00
`md2roff`	4	1116	690 (61.83%)	92.00
`xpdf`	5	1403	691 (49.25%)	92.00
`libxml2`	16	3812	29 (0.76%)	79.00

Distribution of anomalies. Distribution of anomalies is visualised in Figure 4 using a novel spiral representation. Anomalies emerge as the fuzzer explores new execution paths, while the central region (initial phase) remains largely anomaly-free, validating calibration for establishing a normal behaviour baseline.

> **Finding 1:** Calibration ensures that initial mutations are normal, with side-channel anomalies emerging when the fuzzer discovers significantly deviating execution patterns.

Insights for non-sanitized programs. A discovery emerged from our analysis: in `xpdf 3.02`, 87.69% of anomalies detected through EM signals corresponded to crashes when subsequently compiled with AddressSanitizer. This suggests that SCA could enable vulnerability detection even when sanitizer augmentation is infeasible, such as in black-box testing contexts.

> **Finding 2:** SCA can identify potential vulnerabilities in non-sanitized programs, providing AddressSanitizer-like detection capabilities through hardware-level observation.

4.2 RQ2: Crash detection capability through side-channel anomalies

Table 3 presents crash results of our test programs across standard AFL++ and GAFLERNA with a filtered corpus to isolate EM improvements. Each 30-minute phase was repeated 20 times.

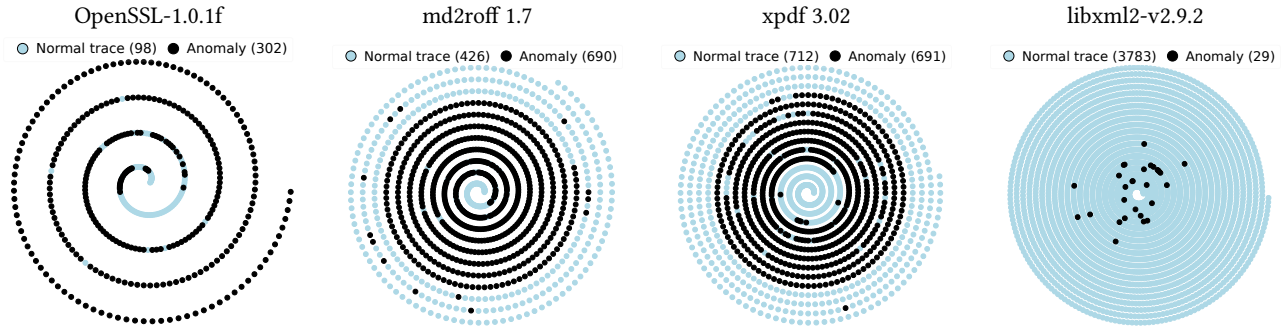


Figure 4: Spiral figures from the evaluated fuzzing corpuses. Each spiral visualises the complete corpus of inputs from the exploratory fuzzing campaign, starting from initial seeds (centre) and spiralling outwards. Initial mutations appear as normal traces in the central region, as their execution patterns match the calibration baseline. As the spiral opens, new execution patterns are detected, and they might be considered anomalous if they deviate significantly from this baseline.

Table 3: Crash comparison across standard AFL++ and GAFLERNA (total/unique crashes). Unique crashes in bold.

Program Name	Expl. Phase	Operational Phase		Gain†
		AFL++	GAFLERNA	
OpenSSL	**5**	0/**0**	968/**5**	+5
md2roff	**117**	8051/**9**	4065/**96**	+87
xpdf	**7**	17/**14**	3/**3**	-11
libxml2	**24**	0/**0**	37/**12**	+12

†Gain: Additional unique crashes discovered by GAFLERNA vs. standard AFL++

Comparison with conventional fuzzing. Hardware-level feedback provides insights into program behaviour missed by traditional fuzzing. In md2roff, GAFLERNA uncovered 96 unique crashes (compared to 9 by AFL++), while for OpenSSL, after finding the Heartbleed vulnerability during exploration (5 crashes), GAFLERNA's operational phase discovered 5 distinct paths to this vulnerability (968 crashes) while AFL++'s operational phase found none.

Comparison with crash-seeded fuzzing. Evaluating against AFL++ seeded with exploration phase crash sources gives an unrealistic advantage, as it leverages future knowledge of vulnerabilities to guide the fuzzing process - unavailable in real-world vulnerability discovery. While this crash-seeded variant found more total crashes by following known vulnerable paths (796 for AFL++ vs 968 for GAFLERNA in OpenSSL, 9511 vs 4065 in md2roff, 82 vs 37 in libxml2), GAFLERNA discovered novel crash patterns using only side-channel guidance. These discoveries are evidenced by comparable crash detection rates (5 for AFL++ vs 8 for GAFLERNA in OpenSSL, 96 vs 87 in md2roff, 3 vs 12 in xpdf, 12 vs 19 in libxml2). Thus, GAFLERNA achieved these results while exploring significantly less code (exploring 16.5% less in OpenSSL), suggesting that side-channel feedback effectively guides the fuzzer toward vulnerable states even with more targeted exploration.

Finding 3: Hardware-level feedback guides fuzzing to discover novel paths to known vulnerabilities that traditional approaches miss, aiding root cause analysis.

4.3 Limitations and Future Work

Our approach encounters several technical challenges. Collecting EM traces introduces performance overhead, as each execution requires multiple measurements to obtain reliable signals. However, this cost is offset by GAFLERNA's ability to provide hardware-level feedback, unavailable to sanitizers, and to detect anomalies in black-box context where binary recompilation is unfeasible. Additionally, the hardware setup demands precise probe positioning to capture consistent signals from specific CPU regions, while isolating target signals from ambient noise. Furthermore, the approach is also sensitive to concurrent processor activities, which can affect EM emissions. Consequently, future work will focus on optimizing data collection, mitigating noise effects, and extending the approach to black-box fuzzing of IoT devices through EM analysis.

5 RELATED WORK

Research mainly integrates fuzzing principles into SCA rather than the reverse, often focusing on specific areas like Java programs [6], hardware drivers [36], and mobility systems [12]. Frameworks leveraging resource [40] and power consumption metrics [3, 18, 20] have emerged, with work on SCA evasion [15]. EM emissions enable novel attacks [17, 24, 27] and fuzzing applications for specific contexts [4, 32, 37]. GAFLERNA provides SCA feedback to fuzzing, potentially revealing vulnerabilities existing methods might miss.

6 CONCLUSIONS

This paper introduces a novel approach to enhance fuzzing by integrating side-channel analysis, leveraging EM emissions for hardware-level feedback. Initial evaluation of GAFLERNA shows how hardware signals reveal program behaviours invisible to traditional instrumentation. This work bridges hardware analysis and fuzzing, opening opportunities in signal interpretation and efficient data collection. Early results with real-world vulnerabilities suggest this cross-layer approach could change how we detect subtle bugs.

ACKNOWLEDGEMENTS

CRITIC Project Grant PLEC2024-011222 funded by AEI/10.13039/501100011033, FEDER and UE. This work has received funding from UK EPSRC #EP/S022503/1. Mikel Iturbe is partially supported by the Basque Government (grant number IT1676-22).

REFERENCES

[1] Hervé Abdi and Lynne J. Williams. 2010. Principal component analysis. *WIREs Computational Statistics* 2, 4 (2010), 433–459.

[2] Md Manjurul Ahsan, M. A. Parvez Mahmud, Pritom Kumar Saha, Kishor Datta Gupta, and Zahed Siddique. 2021. Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance. *Technologies* 9, 3 (2021), 52. https://doi.org/10.3390/technologies9030052

[3] Gabriele Ara, Tommaso Cucinotta, and Agostino Mascitti. 2022. Simulating execution time and power consumption of real-time tasks on embedded platforms. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22)*. Association for Computing Machinery, 491–500. https://doi.org/10.1145/3477314.3507030

[4] Tiyash Basu, Kartik Aggarwal, Chundong Wang, and Sudipta Chattopadhyay. 2020. An exploration of effective fuzzing for side-channel cache leakage. *Software Testing, Verification and Reliability* 30, 1 (2020), e1718.

[5] Gavin Black, Varghese Mathew Vaidyan, and Gurcan Comert. 2024. Evaluating Large Language Models for Enhanced Fuzzing: An Analysis Framework for LLM-Driven Seed Generation. *IEEE Access* 12 (2024), 156065–156081. https://doi.org/10.1109/ACCESS.2024.3484947

[6] Tegan Brennan, Seemanta Saha, and Tevfik Bultan. 2020. JVM fuzzing for JIT-induced side-channel detection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, 1011–1023. https://doi.org/10.1145/3377811.3380432

[7] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD '00)*. Association for Computing Machinery, 93–104. https://doi.org/10.1145/342009.335388

[8] Yuanliang Chen, Yu Jiang, Fuchen Ma, Jie Liang, Mingzhe Wang, Chijin Zhou, Xun Jiao, and Zhuo Su. 2019. EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, 1967–1983.

[9] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. 2019. Outlier detection using isolation forest and local outlier factor. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems (RACS '19)*. Association for Computing Machinery, 161–168. https://doi.org/10.1145/3338840.3355641

[10] Maialen Eceiza, Jose Luis Flores, and Mikel Iturbe. 2023. Improving fuzzing assessment methods through the analysis of metrics and experimental conditions. *Computers & Security* 124 (2023), 102946.

[11] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. 2020. AFL++ : Combining Incremental Steps of Fuzzing Research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*. USENIX Association.

[12] Philipp Fuxen, Murad Hachani, Jonas Schmidt, Philipp Zaumseil, and Rudolf Hackenberg. 2023. Side Channel Monitoring for Fuzz Testing of Future Mobility Systems. *CLOUD COMPUTING 2023* (06 2023), 15–20.

[13] Google. 2018. Fuzzer-test-suite. https://github.com/google/fuzzer-test-suite

[14] Halcy0nic. 2023. https://github.com/Halcy0nic/CVE-2022-34913/tree/main

[15] Yi Han, Matthew Chan, Zahra Aref, Nils Ole Tippenhauer, and Saman Zonouz. 2022. Hiding in Plain Sight? On the Efficacy of Power Side Channel-Based Control Flow Monitoring. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, 661–678.

[16] Shaobo He, Michael Emmi, and Gabriela Ciocarlie. 2020. ct-fuzz: Fuzzing for Timing Leaks . In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE Computer Society, 466–471. https://doi.org/10.1109/ICST46399.2020.00063

[17] Wenqiang Jin, Srinivasan Murali, Huadi Zhu, and Ming Li. 2021. Periscope: A Keystroke Inference Attack Using Human Coupled Electromagnetic Emanations. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. Association for Computing Machinery, 700–714. https://doi.org/10.1145/3460120.3484549

[18] Kazimierz Krosman, Janusz Sosnowski, and Piotr Gawkowski. 2021. Object oriented time series exploration: Applied to power consumption analysis of embedded systems. *Expert Systems with Applications* 184 (2021), 115531.

[19] Github Security Lab. 2022. https://github.com/antonio-morales/Fuzzing101

[20] Sahar Lazim Qaddoori and Qutaiba Ibrahim Ali. 2023. An embedded and intelligent anomaly power consumption detection system based on smart metering. *IET Wireless Sensor Systems* 13, 2 (2023), 75–90.

[21] Seongmin Lee, Shreyas Minocha, and Marcel Böhme. 2025. Accounting for Missing Events in Statistical Information Leakage Analysis . In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 217–228. https://doi.org/10.1109/ICSE55347.2025.00018

[22] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. IEEE Computer Society, 413–422. https://doi.org/10.1109/ICDM.2008.17

[23] Kaizheng Liu, Ming Yang, Zhen Ling, Yue Zhang, Chongqing Lei, Junzhou Luo, and Xinwen Fu. 2024. RIoTFuzzer: Companion App Assisted Remote Fuzzing for Detecting Vulnerabilities in IoT Devices. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. Association for Computing Machinery, 2341–2354. https://doi.org/10.1145/3658644.3670342

[24] Yan Long, Qinhong Jiang, Chen Yan, Tobias Alam, Xiaoyu Ji, Wenyuan Xu, and Kevin Fu. 2024. EM Eye: Characterizing Electromagnetic Side-channel Eavesdropping on Embedded Cameras. In *Network and Distributed Systems Security (NDSS) Symposium*. https://doi.org/10.14722/ndss.2024.24552

[25] Zhengxiong Luo, Junze Yu, Feilong Zuo, Jianzhong Liu, Yu Jiang, Ting Chen, Abhik Roychoudhury, and Jiaguang Sun. 2023. Bleem: Packet Sequence Oriented Fuzzing for Protocol Implementations. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, 4481–4498.

[26] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. In *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*. https://doi.org/10.14722/ndss.2024.24556

[27] Tao Ni, Xiaokuan Zhang, and Qingchuan Zhao. 2023. Recovering Fingerprints from In-Display Fingerprint Sensors via Electromagnetic Side Channel. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, 253–267. https://doi.org/10.1145/3576915.3623153

[28] Shirin Nilizadeh, Yannic Noller, and Corina S. Păsăreanu. 2019. DiffFuzz: differential fuzzing for side-channel analysis. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, 176–187. https://doi.org/10.1109/ICSE.2019.00034

[29] Yannic Noller. 2018. Differential program analysis with fuzzing and symbolic execution. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. Association for Computing Machinery, 944–947. https://doi.org/10.1145/3238147.3241537

[30] Yannic Noller and Saeid Tizpaz-Niari. 2021. QFuzz: quantitative fuzzing for side channels. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021)*. Association for Computing Machinery, 257–269. https://doi.org/10.1145/3460319.3464817

[31] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. 2023. SoK: Deep Learning-based Physical Side-channel Analysis. *ACM Comput. Surv.* 55, 11, Article 227 (Feb. 2023), 35 pages. https://doi.org/10.1145/3569577

[32] Chathura Rajapaksha, Leila Delshadtehrani, Manuel Egele, and Ajay Joshi. 2023. SIGFuzz: A Framework for Discovering Microarchitectural Timing Side Channels. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. https://doi.org/10.23919/date56975.2023.10136966

[33] Dipayan Saha, Shams Tarek, Katayoon Yahyaei, Sujan Kumar Saha, Jingbo Zhou, Mark Tehranipoor, and Farimah Farahmandi. 2024. LLM for SoC Security: A Paradigm Shift. *IEEE Access* 12 (2024), 155498–155521. https://doi.org/10.1109/ACCESS.2024.3427369

[34] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. 2019. Leveraging Electromagnetic Side-Channel Analysis for the Investigation of IoT Devices. *Digital Investigation* 29 (2019), S94–S103.

[35] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitriy Vyukov. 2012. AddressSanitizer: A Fast Address Sanity Checker. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, 309–318.

[36] Dokyung Song, Felicitas Hetzelt, Dipanjan Das, Chad Spensky, Yeoul Na, Stijn Volckaert, Giovanni Vigna, Christopher Kruegel, Jean Pierre Seifert, and Michael Franz. 2019. PeriScope: An Effective Probing and Fuzzing Framework for the Hardware-OS Boundary. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019 (26th Annual Network and Distributed System Security Symposium, NDSS 2019)*. The Internet Society. https://doi.org/10.14722/ndss.2019.23176 Publisher Copyright: © NDSS 2019.All rights reserved.; 26th Annual Network and Distributed System Security Symposium, NDSS 2019 ; Conference date: 24-02-2019 Through 27-02-2019.

[37] Philip Sperl and Konstantin Böttinger. 2019. Side-Channel Aware Fuzzing. In *Computer Security – ESORICS 2019: 24th European Symposium on Research in Computer Security, Luxembourg, September 23–27, 2019, Proceedings, Part I*. Springer-Verlag, 259–278. https://doi.org/10.1007/978-3-030-29959-0_13

[38] Evgeniy Stepanov and Konstantin Serebryany. 2015. MemorySanitizer: Fast detector of uninitialized memory use in C++ . In *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE Computer Society, 46–55. https://doi.org/10.1109/CGO.2015.7054186

[39] KPC Team. 2023. Overclocking for Raspberry Pi 3 Model B with LN2. https://xdevs.com/article/rpi3_oc/

[40] Fei Yan, Rushan Wu, Liqiang Zhang, and Yue Cao. 2023. SPIDER: Speeding up Side-Channel Vulnerability Detection via Test Suite Reduction. *Tsinghua Science and Technology* 28, 1 (2023), 47–58. https://doi.org/10.26599/TST.2021.9010078