InjectV: Modeling Fault Injection Attacks in RISC-V Simulation Environment

InjectV: Modeling Fault Injection Attacks in RISC-V Simulation Environment

Niccolò Lentini[1], Giorgio Fardo[1,2], Stefano Di Carlo[1], Alessandro Savino[1]
[1]Control and Computer Engineering Department, Politecnico di Torino, Turin, Italy
[2]Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
niccolo.lentini@polito.it, giorgio.fardo@cea.fr, stefano.dicarlo@polito.it, alessandro.savino@polito.it

***Abstract*—Fault Injection Attacks (FIAs) are a significant threat to hardware security, capable of compromising systems by inducing malicious faults in computation or storage. Evaluating resilience against such attacks is challenging due to the high cost, complexity, and limited availability of physical fault experiments, particularly during pre-silicon development. Architectural-level simulation offers a developer-oriented, white-box perspective for systematic vulnerability assessment. This paper introduces InjectV, a fault injection attack framework for RISC-V platforms built on the gem5 simulator. InjectV enables precise, guided fault injection at security-critical execution points, such as control-flow decisions, counters, and comparisons, allowing systematic exploration of attack vectors. It currently supports transient fault injection in registers and memory, broadening its ability to simulate diverse attack scenarios. Experimental results on security benchmarks from the FISSC suite, including hardened variants of the VerifyPIN application, demonstrate InjectV's ability to effectively identify fault-injection points, achieving a 95.8% time-saving advantage over traditional fault injection approaches.**

***Index Terms*—Fault Injection Attacks, RISCV, Gem5, Simulation, VerifyPin**

I. Introduction

Nowadays, Fault Injection Attacks (FIAs) are a well-established class of hardware security threats where carefully timed and localized faults can be used to corrupt control flow, bypass authentication checks, or extract sensitive information from otherwise protected systems [1], [2], [3], [4]. Unlike random faults traditionally studied in reliability analysis, FIAs are intentional and targeted. Attackers exploit specific execution points, where a single transient fault can invalidate a protection mechanism or alter program semantics. This shifts the focus of FIA resilience analysis, so the challenge becomes restricting and guiding the analysis toward those trigger points rather than exhaustively exploring arbitrary fault locations.

Evaluating resilience against FIAs on real hardware is expensive, difficult to reproduce, and often infeasible during early design phases. Moreover, exhaustive exploration of the fault space on physical devices is intractable due to the combinatorial explosion of fault parameters, including timing, location, and target state. As a result, developers lack practical tools to systematically assess fault vectors and software countermeasures before deployment [5].

This work was supported by project COLTRANE-V funded by the Ministero dell'Università e della Ricerca within the PRIN 2022 program (D.D.104 - 02/02/2022).

Simulation-based fault injection offers a cost-effective, repeatable alternative to physical testing, enabling scalable security evaluation during early design. However, existing literature lacks tools specifically optimized for large-scale attack-oriented FIA analysis.

This paper presents **InjectV**, a gem5-based [6] RISC-V fault-injection framework that combines preprocessing-driven identification of security-relevant execution points with guided register and memory fault campaigns. RISC-V is targeted due to its increasing adoption in embedded and security-critical systems [7], [8], [9], [10], while gem5 provides a modular full-system simulation infrastructure that can be extended to other instruction set architectures. Experimental results on FISSC VerifyPIN benchmarks show that InjectV enables scalable attack-oriented exploration and achieves a 24× efficiency gain over random injection.

II. Background

A FIA is a physical attack in which an adversary deliberately induces faults to perturb hardware components such as registers, memory, or control logic during program execution, to force the system to deviate from its intended operation. FIAs violate security through transient corruptions, as early research demonstrates that even minimal, targeted faults can fully compromise secret material [11], [12], [13], [14].

Since most faults are masked or cause benign crashes [15], effective evaluation requires identifying specific software–architecture contexts where perturbations become exploitable behaviors. To achieve this, analysis must be system-aware and emulate the full execution stack. We therefore utilize the gem5 full-system simulator [6] to provide the precise, reproducible control necessary to study complex attack vectors across the hardware–software interface.

III. Related Works

Existing fault-injection tools provide fine-grained control over fault models, but offer limited support for relating faults to security-relevant program semantics [16], [17]. They therefore emphasize broad fault-space coverage rather than attacker-guided injections at critical branches, comparisons, or counters.

Physical fault-injection studies provide realistic evidence of attack feasibility. Dutertre et al. [18] showed that electromagnetic injection can induce multiple consecutive instruction

skips and evaluated the resulting model on a PIN-code verification algorithm. Their work also led to the Fault Injection and Simulation Secure Collection (FISSC) benchmark suite [19], which provides hardened program variants with explicit security objectives. Complementary software-level approaches, such as the formal models proposed by Moro et al. [20], reason about instruction skips and control-flow perturbations to derive and assess countermeasures.

Simulation and emulation frameworks improve scalability and reproducibility. Within gem5, tools such as GemFI [21], gemV-tool [22], and gem5-MARVEL [23] support large-scale fault-injection campaigns and detailed architectural or microarchitectural resilience studies. However, these frameworks primarily target reliability-oriented analysis, focusing on coverage, fault vulnerability, and dependability metrics rather than adversarial strategies and workload-level security outcomes. QEMU-based approaches [24] and binary-level tools such as FaultFinder [25] offer faster exploration, but trade off detailed timing fidelity, microarchitectural modeling, or full-system visibility. InjectV addresses this gap by combining gem5 full-system simulation with preprocessing-driven identification of potential attack trigger points. Unlike approaches centered on uniform fault-space exploration, InjectV guides register and memory fault campaigns toward security-relevant execution points while preserving reproducible scheduling and end-to-end analysis across the hardware-software stack.

IV. Methodology

InjectV is structured around a systematic preprocessing stage that isolates potentially exploitable execution windows and guides the following fault-injection campaign. The framework operates on the architectural state within a full-system RISC-V gem5 simulation. The current implementation targets the RISC-V ISA supported by gem5 full-system simulation and operates at the architectural state level. Therefore, the methodology is not tied to a specific core implementation, although the available microarchitectural detail depends on the selected gem5 CPU model.

Figure 1 illustrates the execution flow of the fault injection campaign. The overall workflow consists of four main steps:

1) **Preparation phase:** the system is booted once, and a checkpoint is created after the operating system completes initialization. Subsequent fault-injection experiments start directly from this checkpoint, avoiding repeated boot, reducing trace size, and simplifying simulation and result collection.

2) **Golden and Divergent baseline establishment:** a fault-free execution trace, called the *golden baseline*, is generated to represent the normal application execution.[1] Optionally, one or more *divergent baselines* are collected to represent legitimate executions following alternative control-flow paths, e.g., verification success against failure. These traces are then provided to the preprocessing phase.

[1]A trace denotes the ordered sequence of executed instructions collected during gem5 simulation, including simulation tick, program counter, disassembled instruction, accessed architectural registers, and, when available, memory access information.

3) **Preprocessing phase:** this phase represents the key novelty of the approach. It correlates the execution traces with the target binary to identify security-relevant execution points and to generate a reduced set of candidate Candidate Injection Points (CIPs).

4) **Fault injection phase:** this phase is managed by a campaign manager, which parallelizes the gem5 simulations, parses and activates the faults specified in the fault files generated in the previous step, classifies the outcome of each fault-injection run, generates the differences with respect to the golden trace, and produces the final report.

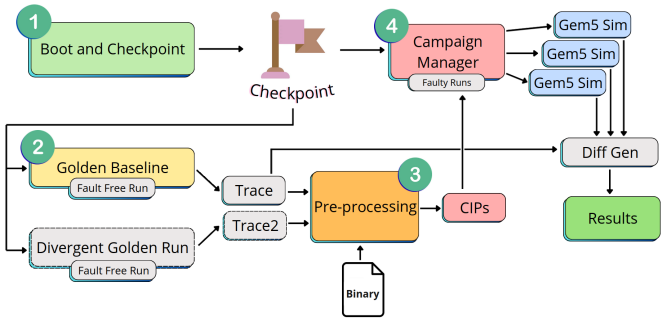

Fig. 1. Execution workflow of the proposed fault injection framework.

Threat and Attack Model: The framework assumes a realistic physical attacker with local access to the device, aiming to bypass security mechanisms using techniques such as laser or voltage/clock glitching.

InjectV abstracts these effects as deterministic, transient bit-level corruptions injected at a specific simulation tick, targeting either CPU architectural registers or physical memory words. This models the software-visible behavior of modern physical fault attacks.

Preprocessing and Candidate Injection Points (CIP) Generation: As highlighted in Section II, untargeted injection yields predominantly masked or non-actionable faults. To explicitly reduce the exploration space, the preprocessing stage produces a list of CIPs.

The framework cross-analyzes the execution trace (and a divergent trace, when provided) with the target ELF binary to systematically identify security-relevant execution points based on the following architectural characteristics:

Control-Flow and Branch Behavior: the framework parses the assembly code extracted from the target binary file to identify conditional branch instructions. Combining this information with the execution trace provides the simulation ticks at which instructions are executed, allowing the framework to determine whether a branch was taken and to locate critical branch evaluations together with their temporal injection window.

Data Dependencies and Write-Before-Use: InjectV analyzes register data liveness by identifying write-before-use patterns directly from the execution trace and binary image. For each register write, the tool determines how many subsequent instructions are executed before the value is first consumed.

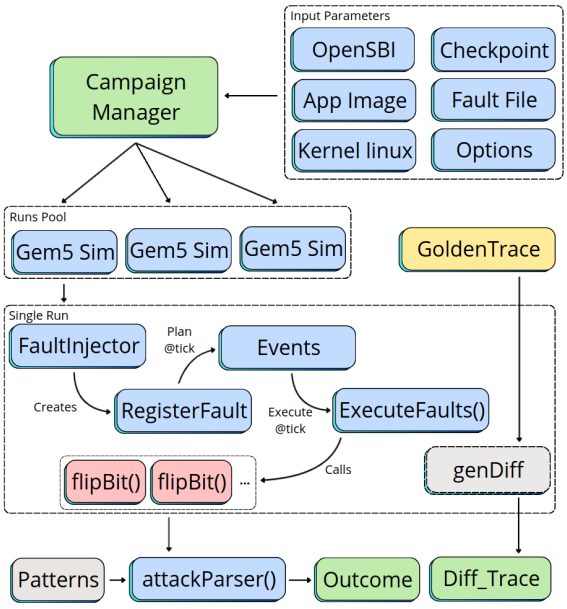

Fig. 2. Campaign Manager workflow.

If this instruction distance falls within a user-defined window, the location is considered a valid candidate injection point.

Differential Execution Divergence: when a divergent execution trace is provided, the framework compares it with the golden trace to identify the first diverging execution point and correlates it with the corresponding instruction and simulation tick, e.g., valid versus invalid PIN code verification.

Instruction-Level Susceptibility and Logic Flips: the framework analyzes the opcodes of instructions extracted from the binary and observed in the execution trace to identify operations whose semantics can be altered through bit-level perturbations. Candidate instructions are evaluated according to the fault model parameters, including the maximum number of bits that may flip and whether flipped bits must be contiguous. Instructions requiring fewer flipped bits and contiguous bit patterns are ranked as better candidates, enabling the identification of instruction logic flips, e.g., `beq` ↔ `bne`, or transformations that convert security-relevant instructions into NOPs. This process outputs a JSON manifest of CIPs, each uniquely identified by a temporal coordinate, i.e., the simulation tick, a spatial target, either a physical memory address or a CPU register, and a viable fault model.

Guided Injection Strategy: The generated CIP manifest is used by the framework to craft targeted fault injections automatically. The campaign manager then uses the resulting fault list to schedule each injection in a dedicated gem5 simulation as part of a precise campaign.

Scalable Campaign Orchestration and Outcome Classification: InjectV decouples the fault-injection mechanics from an automated *Campaign Management* layer. Rather than repeatedly restarting the evaluation environment, InjectV groups experiments into discrete, independent campaigns that can be run in parallel. The Campaign Manager launches multiple isolated gem5 instances that resume from a pre-computed stable checkpoint, delivering **zero OS boot overhead** and improving throughput.

Eventually, InjectV includes an attack-oriented classification stage that separates system-level execution issues from application-level outcomes resulting from the executed workload. **Host-Level Supervision (TIMEOUT):** runs exceeding a timeout derived from the golden, fault-free, execution time are terminated by the host and classified as **TIMEOUT**. **Target-Level Outcome Classification (CRASH, NO_EFFECT, UNKNOWN):** runs that finish within the timeout are classified by parsing guest-visible telemetry like terminal output and explicit failure markers.

Clear OS/application failures, such as kernel panic, are labeled **CRASH**. Runs whose outputs match the golden baseline are labeled **NO_EFFECT**. If the observed output does not match any expected pattern, the run is labeled **UNKNOWN** and flagged for follow-up analysis.

Finally, an execution is labeled **SUCCESS** *only* when the injected perturbation satisfies a user-defined, application-dependent security predicate.

V. Results

In this section, we describe the experimental campaign used to evaluate InjectV.

`VerifyPIN` is the benchmark workload used to evaluate InjectV on a realistic, security-relevant control flow. It implements a PIN code verification routine that compares a user-provided PIN against a stored reference while enforcing authentication checks and attempt counters. `VerifyPIN` is part of the FISSC [19], a suite of C programs designed to study fault-injection attacks on applications hardened by means of countermeasures, such as Hardened booleans, Backup copy, Double test, Step counter, and programming features, such as Inlined calls and Fixed time loop. FISSC provides several `VerifyPIN` variants with progressively increasing levels of software hardening.

This setup enables a controlled comparison between an unprotected implementation and progressively hardened verification flows under identical simulation conditions. For each `VerifyPIN` version, the InjectV preprocessing stage produced, on average, approximately 5,000 candidate fault injections. These were executed in a large-scale simulation campaign and classified according to the workload-driven outcome categories.

To properly classify **SUCCESS**, the user-defined predicate parses the final trace status line reporting three `VerifyPIN` variables: countermeasure activation, authentication success, and the remaining PIN attempts. A run is classified as **SUCCESS** when the final state indicates authentication bypass, countermeasure-triggered authentication, or a reset of the protection counter enabling further attempts. Runs that deviate from the golden execution but do not satisfy this predicate are labeled **PARTIAL_SUCCESS**. This class captures application-visible deviations that are not directly exploitable according to the selected security objective. This predicate-based definition distinguishes generic divergence from ex-

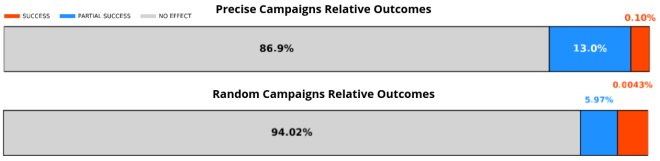

Fig. 3. Relative Outcomes Distribution.

ploitability and allows campaign outcomes to be interpreted in terms of security impact.

To assess the guided preprocessing stage, we compared it with random exploration campaigns that uniformly sample the fault space in time and space. To ensure a fair comparison, random campaigns used the same number of injections and the same register/memory fault mode distribution as the guided ones. Overall, 46,067 $\times$ 2 simulations were executed on 37 cores, with 33.6GB peak RAM usage, 30 seconds average runtime per simulation, and around 20 hours for diff generation.

Guided vs Random Fault Exploration: The results of the guided and random campaigns were analyzed across the temporal distribution of injections, the spatial distribution across memory addresses, and the spatial distribution across architectural registers. For each injection, outcomes were categorized consistently with the scheme defined in Section IV. The campaigns confirm the effectiveness of the guided approach. As reported in Fig. 3, InjectV discovered 48 successful security violations, 27 from memory corruption and 21 from register perturbation, whereas random injection discovered only 2. The temporal analysis in Fig. 4 highlights a clear difference between guided and random exploration. Guided injections cluster around execution regions where the PIN value is processed and the verification logic executes, while random exploration distributes injections uniformly across the execution timeline producing mostly *NO_EFFECT* outcomes.

Figures 5 and 6 show that guided injections concentrate on a restricted set of sensitive memory regions and registers. In particular, all successful memory corruptions fall along the PIN verification path, while register `x15` accounts for 16 of the 21 successful register-based violations. Conversely, random exploration targets the memory space and register file uniformly, producing mainly *NO_EFFECT* outcomes.

Across the different `VerifyPIN` versions, the number of successful attacks remains similar. This suggests that the guided preprocessing stage often identifies control-flow decisions that directly govern authentication, allowing faults to bypass higher-level countermeasures. This motivates future work on attack strategies targeting specific software protections.

Eventually, success rate comparison between random and guided campaigns reveals a significant disparity in efficiency. To achieve the same number of success as the guided campaign, a random approach would require an estimated 1,104,000 injections, representing a 24$\times$ increase in computational effort. Consequently, the preprocessing methodology presented in this paper achieves a 95.8% time-saving advantage over a brute-force random injection strategy.

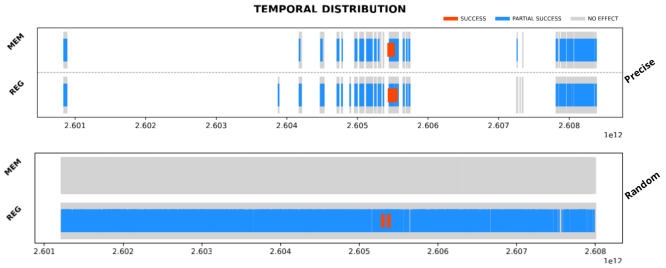

Fig. 4. Temporal distribution of injections for guided and random campaigns.

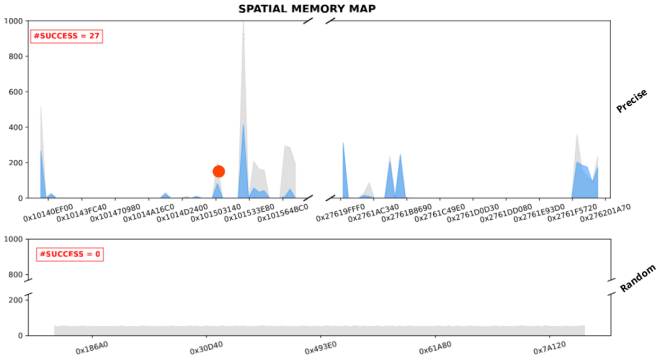

Fig. 5. Spatial Distribution of memory injections.

VI. Conclusion

This paper introduced InjectV, an attack-centric fault-injection framework for full-system RISC-V simulation built on gem5. By using preprocessing to identify security-relevant execution points, InjectV reduces the search space for exploitable vulnerabilities and achieves a 24$\times$ analysis efficiency improvement over random injection. Future work will extend the framework to multi-fault scenarios and broader attack models.

To support open science in research, InjectV is publicly available at the official GitHub repository: https://github.com/smilies-polito/injectv.

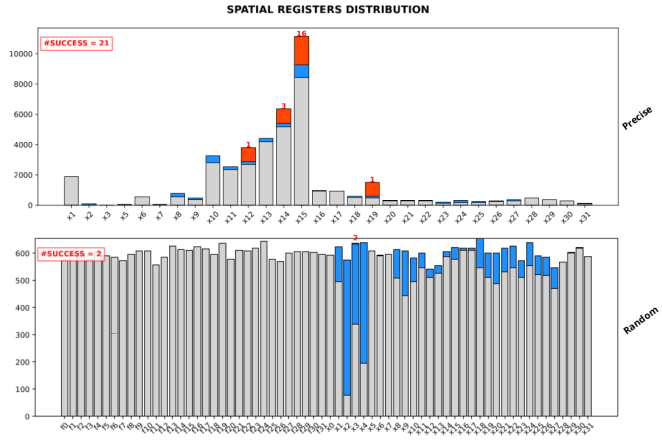

Fig. 6. Spatial Distribution of register injections.

REFERENCES

[1] A. Gangolli, Q. H. Mahmoud, and A. Azim, "A systematic review of fault injection attacks on iot systems," *Electronics*, vol. 11, no. 13, p. 2023, 2022.

[2] R. Boulifa, G. Di Natale, and P. Maistri, "Countermeasures against fault injection attacks in processors: A review," *Information*, vol. 16, no. 4, p. 293, 2025.

[3] M. Portolan, A. Savino, R. Leveugle, S. Di Carlo, A. Bosio, and G. Di Natale, "Alternatives to fault injections for early safety/security evaluations," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–10.

[4] P. Ravi, S. S. Roy, S. Bhasin, and A. Chattopadhyay, "Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 2, pp. 1–54, 2024.

[5] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[6] J. Lowe-Power, E. Carmean, M. Hsu *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[7] M. Werner, R. Schilling, T. Unterluggauer, and S. Mangard, "Protecting risc-v processors against physical attacks," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1136–1141.

[8] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, "A security risc: Microarchitectural attacks on hardware risc-v cpus," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2321–2338.

[9] J. Anders, P. Andreu, B. Becker, S. Becker, R. Cantoro, N. I. Deligiannis, N. Elhamawy, T. Faller, C. Hernandez, N. Mentens, M. N. Rizi, I. Polian, A. Sajadi, M. Sauer, D. Schwachhofer, M. S. Reorda, T. Stefanov, I. Tuzov, S. Wagner, and N. Zidarič, "A survey of recent developments in testability, safety and security of risc-v processors," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–10.

[10] B. Farnaghinejad, D. Bellizia, A. Dolmeta, G. Masera, A. Porsia, A. Ruospo, S. Di Carlo, A. Savino, and E. Sanchez, "Power side-channel vulnerabilities of a risc-v cryptography accelerator integrated into cva6 via core-v extension interface (cv-x-if)," in *2025 IEEE International Test Conference (ITC)*, 2025, pp. 233–242.

[11] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology — EUROCRYPT '97*, ser. Lecture Notes in Computer Science, vol. 1233. Springer, 1997, pp. 37–51.

[12] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO '99*, ser. Lecture Notes in Computer Science, vol. 1666. Springer, 1999, pp. 388–397.

[13] M. Joye and M. Tunstall, *Fault Analysis in Cryptography*. Springer, 2012.

[14] L. Teixeira *et al.*, "A survey of fault attacks," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–28, 2019.

[15] M. Moradi, B. Van Acker, and J. Denil, "Failure identification using model-implemented fault injection with domain knowledge-guided reinforcement learning," *Sensors*, vol. 23, no. 4, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/4/2166

[16] M. Alonso, D. Andreu, R. Canal, S. Di Carlo, O. Chatzopoulos, C. Chenet, J. Costa, A. Girones, D. Gizopoulos, G. Papadimitriou, E. Morancho, B. Otero, and A. Savino, "Special session: Security and ras in the computing continuum," in *2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2024, pp. 1–6.

[17] E. Magliano, A. Savino, and S. Di Carlo, "Real-time embedded system fault injector framework for micro-architectural state based reliability assessment," *Journal of Electronic Testing*, vol. 41, no. 2, pp. 193–208, 2025. [Online]. Available: https://doi.org/10.1007/s10836-025-06170-w

[18] J.-M. Dutertre, A. Menu, O. Potin, J.-B. Rigaud, and J.-L. Danger, "Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures," *Microelectronics Reliability*, vol. 121, p. 114133, 2021.

[19] FISSC Consortium, "Fault injection and simulation secure collection (fissc)," https://lazart.gricad-pages.univ-grenoble-alpes.fr/fissc/, 2024.

[20] N. Moro, K. Heydemann, E. Encrenaz, and B. Robisson, "Formal verification of a software countermeasure against instruction skip attacks," in *Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2014, pp. 1–10.

[21] K. Parasyris, C. Tziantzoulis, C. Antonopoulos, and N. Bellas, "Gemfi: A fault injection tool for studying the behavior of applications on unreliable substrates," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014.

[22] J. Doe *et al.*, "gemv-tool: A comprehensive soft error reliability estimation tool," *Electronics*, vol. 12, no. 22, 2023.

[23] O. Chatzopoulos, G. Papadimitriou, V. Karakostas, and D. Gizopoulos, "gem5-marvel: Microarchitecture-level resilience analysis of heterogeneous soc architectures," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[24] Y. B. Bekele, D. B. Limbrick, and J. C. Kelly, "A survey of qemu-based fault injection tools & techniques for emulating physical faults," *IEEE Access*, vol. 11, pp. 62 662–62 673, 2023.

[25] K. Murdock, M. Thompson, and D. Oswald, "Faultfinder: Lightning-fast, multi-architectural fault injection simulation," in *Workshop on Attacks and Solutions in Hardware Security (ASHES)*, 2024.