Short Paper: Efficient Verification of Fault Injection Attack Protections at Binary Level*

Noémie Bénard[1], Sébastien Bardin[1], Frédéric Recoules[1], Yanis Sellami[1]

Université Paris-Saclay, CEA List, France

[1][firstname].[lastname]@cea.fr

Abstract

Advanced attackers can exploit targeted physical perturbations on a system to modify the behavior of an otherwise safe program. This is called a fault injection attack. While analysis techniques exist both at source level and microarchitectural level to evaluate whether successful fault injection attacks are possible, those are respectively unadapted to represent accurate microarchitecural fault effects and prohibitively expansive. Program analysis at binary level offers a good compromise. In this work, we extend the adversarial reachability of Binsec to propose a more general and more mature binary-level fault-aware symbolic execution as a Binsec plugin by making use of the recently introduced capability to hook the decoding phase, allowing the full support of the instruction skip fault model. We validate this approach by analyzing the effect of compiler-introduced countermeasures against various fault models in CompCert and CompaSec, and observe that some of them remain efficient for stronger attack models.

1 Introduction

Fault injection attacks [14] are a class of attacks that use a physical perturbation on the system running a program to obtain an adversarial effect leading to an execution that does not exist in the semantic of the original program. By leveraging this effect, an attacker can exploit such unruly executions to extract secret data or bypass protection mechanisms. High-security components, particularly on protected embedded systems, must provide some level of countermeasures against such attacks. In order to evaluate the quality of these countermeasures, one solution is to analyze the programs running on the system to detect possible fault attacks. While an evaluation or simulation of the full microarchitecural state over a program execution permits to precisely assess when a fault leads to an observable effect [4], this type of analysis tend to remain prohibitively expansive. Moreover, it requires a detailed knowledge of the microachitecture of the evaluated target, which may not be available for proprietary systems. Conversely, software analyses tend to be significantly more efficient but may be unable to represent correctly the microarchitectural effect of a fault injection attack [13, 11], particularly when the analyzer works at a high level of abstraction (e.g. source code).

To obtain the best of both world, we believe that designing software-level analyses aware of the micro-architectural effects needed to represent the investigated fault attacks (the evaluated fault model) is a good compromise. Previous work [9, 8] has demonstrated that we can construct efficient binary-level formal analyzer to detect fault injection attacks in protected software, including in the WooKey bootloader [1], as well as the capabilities of the method to scale well for multifault attacks. This was implemented in the binary-level symbolic execution Binsec [7, 6, 2]. However, this work was limited to value fault injection and test inversions, and deeply interleaved with the core of the underlying symbolic execution analyzer. These limitations

*This work was supported by ANR PIA funding: ANR-20-IDEES-0002. This work was supported by the SECUREVAL project, under the reference ANR-22-PECY- 0005.

Table 1: Evaluation result/Expected result of countermeasures against several fault injection attack models, for up to two faults, on examples of the Lazart benchmark for a RISCV target. ✓ denotes a safe/expected safe example and ✗ an unsafe/expected unsafe example; #1 denotes a fault budget of 1 and #2 a fault budget of 2.

	Test Inversion						Instruction Skip						Call Skip	
	gcc		CompCert		CompaSec		gcc		CompCert		CompaSec		CompCert	
Program	#1	#2	#1	#2	#1	#2	#1	#2	#1	#2	#1	#2	#1	#2
`ark`	✗	✗	✓/✓	✗/✗	✗/✗	✗/✗	✗	✗	✗/✗	✗/✗	✗/✓	✗/✓	✓/✓	✓/✗
`fu`	✗	✗	✓/✓	✗/✗	✗/✗	✗/✗	✗	✗	✗/✗	✗/✗	✗/✓	✗/✓	✓/✓	✓/✗
`memcmps`	✗	✗	✓/✓	✗/✗	✗/✗	✗/✗	✗	✗	✗/✗	✗/✗	✗/✓	✗/✓	✓/✓	✓/✗

render the related tool hard to maintain, hard to extend, and unable to manage more precise microarchitectural effects. In particular, it does not benefit from recent improvement of its underlying engine Binsec [7, 2], particularly the full support for ARM and RISCV architectures, as well as performance improvements. We aim to extend this work to design a more parametric and extensible fault-injection aware binary-level symbolic execution.

2 Generalized Adversarial Reachability Binsec Plugin

As a first step towards this goal, we propose to leverage the newest plugin mechanism of Binsec, to design an adversarial reachability analysis able to handle efficiently both value faults (including arbitrary value injection and mask flips) and control-flow faults (including test inversion and instruction skip) for transient, persistent and permanent scenarios.

Adversarial reachability [9] is a framework designed to reason about fault injection attacks during a symbolic execution [5]. In practice, the faults are introduced in the path predicates in a forkless manner to prevent path explosion, typically with *if-then-else* expressions. This permits to keep the strategy scalable, even with reasonably high multifault budgets (up to 10 faults). We extend this strategy to implement persistent and permanent faults, as well as encodings for previously unsupported control-flow fault models: full instruction skips. In our setup, most instruction skip faults are expressed in a forkless manner, including most branching instructions.

The architecture that permits this implementation is supported by the new plugin mechanism of Binsec, which introduces the capability to patch the intermediate representation generated in the decoding stage of the symbolic execution engine, without any modification to the core of Binsec. We also demonstrate that the same mechanism can be used to remediate to unwanted binary code decoding optimizations when they conflict with the sound application of a fault model if applied afterwards, in particular in the context of ARM IT blocks.

3 Validating Fault Injection Attacks Countermeasures

As a validation, we have used this implementation to verify the correct application of hardening strategies against fault injection attacks, including the hardening of CompCert [12] and the CompaSec [10] countermeasure against instruction skips. The results of these evaluations appear to validate the claims of CompCert and part of the claims of CompaSec. We also show that some of the protection appear to be sufficient in practice to protect against slightly stronger fault models than the ones they claim to protect against. The detailed results of this evaluation are presented in Table 1, where we compare the result of the symbolic execution with the expected result for the hardenings for several programs of the Lazart [13, 3] benchmark for a RISCV target.

References

[1] https://wookey-project.github.io/.

[2] https://binsec.github.io/.

[3] https://gricad-gitlab.univ-grenoble-alpes.fr/securitytools/lazart/-/tree/main/use-cases.

[4] Fault-resistant partitioning of secure cpus for system co-verification against faults. 2024.

[5] Cristian Cadar and Koushik Sen. Symbolic execution for software testing: three decades later. *Commun. ACM*, 56(2), February 2013.

[6] Robin David, Sébastien Bardin, Thanh Dinh Ta, Laurent Mounier, Josselin Feist, Marie-Laure Potet, and Jean-Yves Marion. BINSEC/SE: A dynamic symbolic execution toolkit for binary-level analysis. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER*, 2016.

[7] Adel Djoudi and Sébastien Bardin. BINSEC: binary code analysis with low-level regions. In *TACAS 2015*. Springer, 2015.

[8] Soline Ducousso. *Moving code analysis from safety to security : attacker model*. Theses, Université Grenoble Alpes, December 2023.

[9] Soline Ducousso, Sébastien Bardin, and Marie-Laure Potet. Adversarial reachability for program-level security analysis. In *ESOP 2023*. Springer Nature, 2023.

[10] Johannes Geier, Lukas Auer, Daniel Mueller-Gritschneder, Uzair Sharif, and Ulf Schlichtmann. Compasec: A compiler-assisted security countermeasure to address instruction skip fault attacks on risc-v. ASPDAC '23, 2023.

[11] Sébastien Michelland, Christophe Deleuze, and Laure Gonnord. From low-level fault modeling (of a pipeline attack) to a proven hardening scheme. In *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction*, CC 2024, 2024.

[12] Basile Pesin, Sylvain Boulmé, David Monniaux, and Marie-Laure Potet. Formally verified hardening of c programs against hardware fault injection. CPP '25. Association for Computing Machinery, 2025.

[13] Marie-Laure Potet, Laurent Mounier, Maxime Puys, and Louis Dureuil. Lazart: A symbolic approach for evaluation the robustness of secured codes against control flow injections. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 2014.

[14] Bilgiday Yuce, Patrick Schaumont, and Marc Witteman. Fault attacks on secure embedded software: Threats, design, and evaluation. *Journal of Hardware and Systems Security*, 2, 06 2018.