Pre-silicon Security Analysis of RISC-V Processors to Fault Injection Attacks

Damien Couroussé[1] and Mathieu Jan[2*]

[1]Univ. Grenoble Alpes, CEA, List, F-38000 Grenoble, France
[2]Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

This talk proposal showcases the sensitivity of processor microarchitectures to fault injection attacks, which threat hardware and software security. Therefore, security analysis must consider both the hardware and software models of the target system. Furthermore, fault injection requires an exhaustive analysis of all possible injection locations, resulting in unprecedented complexity. We present our methodology and two tools developed for this purpose. In particular, our approach has enabled us to identify a new vulnerability in the OpenTitan secure core.

Introduction

Context. Fault injection allows an attacker to move the target processor out of its expected functioning bounds, such that the target system reaches unexpected hardware and software states or follows unexpected execution paths. Reaching such unexpected states is then leveraged in attacks for leaking secrets, escalating privileges, etc.

An analysis at the hardware level can show that a module is functionally incorrect due to the perturbation induced by fault injections. Such approaches are sufficient for the robustness analysis of standalone components such as cryptographic IPs, but usually, the exploitation of a fault injection, in an attack, involves software. On the other side, a pure software analysis struggles to model many subtle behavioral effects induced by fault injection [1].

Recent research has highlighted the need to consider the consequences of fault injection in the processor micro-architecture [2]. However, such effects, induced by faults in the processor microarchitecture, can only be leveraged in an attack by specific software conditions, in particular the sequence of program instructions executed, such that the fault effects propagate until the attack target is reached.

Contributions. Our team has implemented pre-silicion tools [3, 4] able to: 1) identify exploitable vulnerabilities at the software level based on these interactions between a software and a microarchitecture, or 2) formally prove the security, for a given attacker model, of a system embedding hardware/software countermeasures against fault injections. These tools implement a formal methodology that has proved effective in finding microarchitectural vulnerabilities and/or proving the robustness, for a given fault model, of various RISC-V based processors [5]. For instance, we apply this methodology to the OpenTitan secure element and formally prove the security of its processor's HW countermeasure to single bit-flip injections [6].

*damien.courousse@cea.fr, mathieu.jan@cea.fr

We thank Simon Tollec, Karine Heydemann and Mihail Asavoae for their contributions to this work. This work was funded in part by the French National Research Agency (ANR) under the ARSENE project (ANR-22-PECY-0004) and under the France 2030 program (ANR-22-PTCC-0001, ANR-24-RRII-0004), and by Key Digital Technologies Joint Undertaking (KDT JU) under the TRISTAN project (101095947).

µArchiFI

µArchiFI [5] generates a formal transition system from a processor hardware description in Verilog, a binary software program, and an attacker model comprising the fault model. First, the user can simulate the execution of the target program, compiled for the corresponding ISA, on the hardware design to set the initial state of the hardware right before the instruction sequence to analyze formally. Then, the user needs to specify the attacker model comprising the goal φ, the maximal number of faults N, and the fault model (location, timing, and fault effects). This model is automatically integrated into the system through a specific pass. The attacker's goal can also be specified into the hardware design using the SystemVerilog Assertion subset supported by Yosys [7]. Finally, the µArchiFI tool produces a transition system in Smt-lib or Btor2 format. The faulty transition system can be verified using external model-checking tools compatible with these input formats, like AVR [8], Pono [9] or BtorMC [10].

When the model checker finds a counterexample, a VCD file reports precisely where the fault is injected and when the attacker's goal is reached. However, understanding the propagation of faults and their consequences requires human expertise, but this task can be facilitated by external tools that perform differential traces comparison against a reference model. µArchiFI was used to analysis the CV32E40P

RISC-V processor and its secure flavor the CV32E40S.

k-Fault-Resistant Partitioning

A major challenge of µArchiFI lies in the state space generated by the modeling of processor's behavior executing a sequence of instructions and under a fault model. A monolithic resolution approach combining all these elements can only handle the analysis of hundreds of machine instructions executed on an in-order 4- or 5-stage processor, for a single fault injection. Our extension k-Fault-Resistant Partitioning (k-FRP) [6] solves the fault propagation problem when assessing redundancy-based hardware countermeasures. In a first step, a hardware verification step converts an input RTL design, after synthesis, to a cycle-accurate bit-accurate circuit model. The fault model describing all possible fault injections is then derived from the input fault model and the produced circuit model. k-FRP then formally proves, at the gate level, whether hardware redundancy-based countermeasures can capture up to k faults injected by an attacker, thus labeling it as k-fault secure, and this independently of the program being executed. When k-FRP fails to label a HW counter-measure as k-fault secure, the user can inspect the verification logs for failure analysis.

The proven security guarantees by k-FRP in the first step can then reduce the remaining hardware attack surface when introducing the software in a second step. This second step is a system verification process that analyzes program executions to detect if an attacker can reach his goal. This verification is performed by considering only the faults that have not been formally proven, at the first step, to be detected by hardware protections. The software and hardware co-verification takes as input the hardware design, a binary program, the attack order, the attacker goal, similarly to µArchiFI but on a reduced fault model.

k-FRP was used to analyze the k-fault security of a development version of the fault-hardened RISC-V Ibex processor used in the OpenTitan secure element [11]. We discovered a new vulnerability to a single fault injection in the processor's register file, demonstrated its possible exploitation in software secured programs, and verified the security of the proposed fix. We also verified the robustness of the OpenTitan secure element running the first step of a secureboot process, a previously intractable verification.

Related Work

Classical verification methods like simulation are used but they are often not exhaustive, and it is often difficult to highlight corner cases. Formal techniques were first dedicated to analyze cryptographic circuits with equivalence checking. However, they do not handle sequential verification since the design to analyze is unrolled to perform equivalence checking, and thus, tools cannot analyze software. In comparison, µArchiFI does not support advanced technological netlists, but supports any Verilog or SystemVerilog design by plugging our translation pass into the Yosys tool. In addition, we take advantage of a simplified word-level netlist to bridge the gap with the software and facilitate the analysis of the transitional system. We also keep the sequential logic instead of unrolling and flattening the whole design to use model-checking verification techniques.

On the other hand, approaches at the binary level propose methodologies to analyze the robustness of software programs. However, these works do not consider the execution platform, and the generic fault models used are sometimes inadequate to model microarchitectural implementation details.

Finally, commercial tools offer all the building blocks required for such a fault injection analysis, but their closed nature prevents users from integrating them into the same verification framework. SystemVerilog Assertion (SVA), supported by tools such as Synopsys VC Formal or Siemens QuestaVerify, could define the attacker's goal, but is not suitable for fault modeling. On the other hand, tools such as Cadence JasperGold offer support for fault injection but do not consider software. In short, none of these tools address the verification of software and hardware against fault injection.

References

[1] J. Laurent et al. "Bridging the Gap between RTL and Software Fault Injection". In: *ACM TECS* (2021).

[2] Simon Tollec et al. "Exploration of Fault Effects on Formal RISC-V Microarchitecture Models". In: *FDTC*. 2022.

[3] *µArchiFI*. https://github.com/CEA-LIST/uArchiFI. 2024.

[4] *k-FRP*. https://github.com/CEA-LIST/Fault-Resistant-Partitioning. 2024.

[5] Simon Tollec et al. "µARCHIFI: Formal Modeling and Verification Strategies for Microarchitectural Fault Injections". In: *FMCAD*. 2023.

[6] Simon Tollec et al. "Fault-Resistant Partitioning of Secure CPUs for System Co-Verification against Faults". In: *TCHES* (2024).

[7] Claire Xenia Wolf. *Yosys Open Synthesis Suite*. https://github.com/YosysHQ/yosys. 2016.

[8] Aman Goel and Karem Sakallah. "AVR: Abstractly Verifying Reachability". In: *TACAS*. 2020.

[9] Makai Mann et al. "Pono: A Flexible and Extensible SMT-Based Model Checker". In: *CAV*. 2021.

[10] Aina Niemetz et al. "Btor2 , BtorMC and Boolector 3.0". In: *Computer Aided Verification*. 2018.

[11] Scott Johnson et al. "Titan: enabling a transparent silicon root of trust for cloud". In: *Hot Chips*. 2018.