FISSA: Fault Injection Simulation for Security Assessment

William PENSEC
UMR 6285, Lab-STICC
Université Bretagne Sud
Lorient, France
william.pensec@univ-ubs.fr

Vianney LAPÔTRE
UMR 6285, Lab-STICC
Université Bretagne Sud
Lorient, France
vianney.lapotre@univ-ubs.fr

Guy GOGNIAT
UMR 6285, Lab-STICC
Université Bretagne Sud
Lorient, France
guy.gogniat@univ-ubs.fr

***Abstract*—This work introduces FISSA, an open-source software tool designed to simplify the creation of fault injection campaigns using popular HDL simulation tools. FISSA comprises two key modules, each encapsulating an existing HDL simulator. The first module generates TCL scripts to automate fault injection according to user specifications, while the second module facilitates fault analysis, allowing users to evaluate design's security. This approach seamlessly integrates fault injection simulations into the design workflow. To showcase FISSA's effectiveness, the paper presents a case study assessing the robustness of a Dynamic Information Flow Tracking mechanism within a RISC-V processor across fault injection scenarios.**

***Index Terms*—Hardware security, RISC-V, DIFT, Fault Injections Attacks, Open-Source tool, Vulnerability Assessment**

I. Introduction

Internet of Things (IoT) devices have transformed the way we interact with our environment. These interconnected devices seamlessly collect, exchange, and analyse data, enhancing efficiency and convenience across various domains (medical sensors, automotive, intelligent security systems). However, this increased connectivity also raises concerns about potential physical attacks, such as fault injection attacks (FIA) [1] gaining attention in recent years.

Many studies have shown the vulnerabilities of critical systems against FIAs. In [2], authors demonstrate the possibility to recover computed secret data using FIA targeting hidden registers on the RISC-V Rocket processor. In [3], Electromagnetic Fault Injection (EMFI) attack is used to recover an AES key by targeting the cache hierarchy and the MMU. Finally, authors of [4] have shown that one can combine side-channel attacks (SCA) and FIAs to bypass the PMP mechanism in a RISC-V processor. It is therefore necessary, when designing a circuit, to assess its sensitivity to FIA as soon as possible.

The rest of the paper is structured as follows. Section II introduces FISSA, our fault injection tool. Section III describes our experimental setup for fault injection simulations and presents the results of the different fault injections campaigns with and without protections. Finally, Section IV concludes the work and draws some perspectives.

II. FISSA - Fault Injection Simulation for Security Assessment

This section presents our open-source tool, FISSA, to help circuit designers to analyse, throughout the design cycle, the sensitivity to FIA of the developed circuit. Figure 1 presents the software architecture of FISSA. It consists of 3 different modules: *TCL Generator*, *Fault Injection Simulator* and *Analyser*. The first and third modules correspond to a set of Python classes.

The *TCL Generator* takes a configuration file and a target file to generate a set of parameterised TCL scripts. These scripts are customised according to the provided configuration file and are used to execute the fault injection campaign.

The *Fault Injection Simulator* conducts the fault injection campaign using input files generated by the *TCL Generator*. The simulator simulates these input files on a circuit design via HDL files and memory initialisation files to generate output JSON result files. To accomplish this, it relies on an existing HDL simulator such as Questasim, or Vivado.

The *Analyser* assesses the results of the simulations and generates a collection of files that enables designers to examine the effects of fault injection on their designs using various pieces of information.

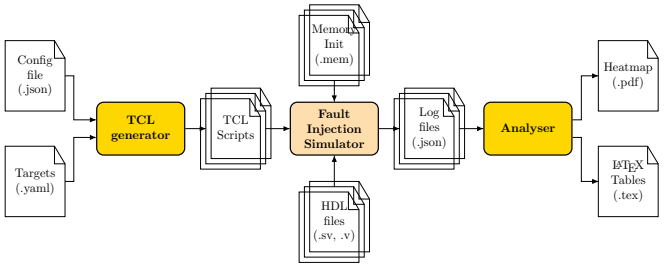

Fig. 1. Software architecture of FISSA

The tool requires a set of targets (i.e. hardware elements in which a fault should be injected), the fault model and the considered injection window(s) which identifies the period(s), in number of clock cycles, in which fault injections are performed. Then, it runs a first simulation with no fault injected, which is used as a reference for comparison with the following simulations to determine end-of-simulation statuses.

TABLE I
LOGICAL FAULT INJECTION SIMULATION CAMPAIGNS RESULTS FOR SINGLE BIT-FLIP IN TWO REGISTERS AT A GIVEN CLOCK CYCLES

		Crash	Silent	Delay	Single Error Correction	Double Errors Detection	Success	Total
Buffer overflow	No protection	0	45,097	1,503	–	–	1,406 (2.93%)	48,006
	Hamming Code	0	0	575	67,829	–	452 (0.66%)	68,856
	SECDED	0	2,436	0	59,424	11,616	0 (0.00%)	73,476

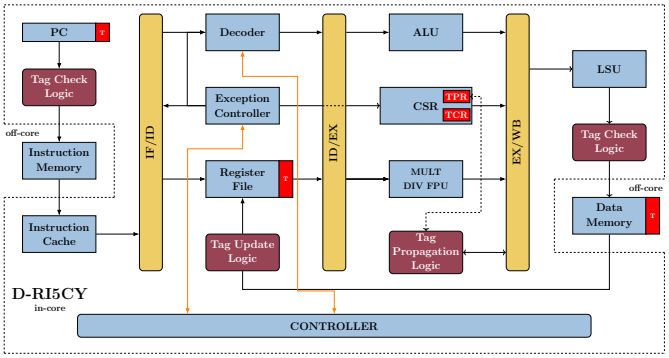

Fig. 2. D-RI5CY processor architecture overview

Then, for each target, each fault model and for each clock cycle within the injection window, the simulation is executed, and the logs are stored in a dedicated file.

III. EXPERIMENTAL SETUP AND RESULTS

This section presents a case study to demonstrate the interest of the proposed solution. It focuses on the evaluation of the robustness of the Dynamic Information Flow Tracking (DIFT) mechanism integrated to the D-RI5CY [5] processor.

The D-RI5CY core is a 4-stage in-order 32-bit RISC-V processor. It introduces an DIFT mechanism to protect the processor against software attacks such as buffer overflows or SQL injections. Figure 2 presents an overview of the D-RI5CY processor. DIFT-related modules are highlighted in red. These modules allow storing, propagating and checking tags during the execution of an application. The security policy is configured through two CSRs (*Configuration and Status Register*) named TPR (*Tag Propagation Register*) and TCR (*Tag Check Register*). The *Tag Update Logic* module is used to initialise or update the tag in the register file according to the tagged data. Then, when a tag is propagated in the pipeline, the *Tag Propagation Logic* module propagates tags according to the security policy defined in the TPR. Once a tag has been propagated and its data has been sent out of the pipeline, the *Tag Check Logic* modules check that it conforms to the security policy defined in the TCR. If not, an exception is raised.

We rely on FISSA, described in Section II, to study the behaviour of the DIFT D-RI5CY mechanism against different fault injection scenarios taking into account three versions of the D-RI5CY: an unprotected, a Hamming Code implemented version and a Hamming Code with a parity bit (SECDED) version for double errors detections. For the use case, we consider a software application in which a buffer overflow can be exploited to perform a Return-Oriented Programming attack (ROP). Thanks to the DIFT mechanism, such attack is detected and stopped.

Table I shows the results obtained from this use case on the three implemented protections. We consider, for our fault model, an attacker able to perform one single bit-flip in two registers at a given clock cycle. We will only consider bit-flips into DIFT-related registers, as we want to show the limit of the DIFT against FIA. The results presented in Table I reveal that employing the unprotected version will lead to 2.93% of successes when attacking the registers, while this number goes down to 0.66% with Hamming Code. With less than 1% of successes, we could stop the evaluation there, but in real world, an attacker would need only 1 success to perform and obtain access to the device. To harden our DIFT mechanism, we choose to implement SECDED, with this protection we can correct single error and detect double errors. For that we use, the same implementation as Hamming Code but with a simple extra bit. Then, all single faults are corrected depending on our implementation and all double bit errors are detected. The last 2,436 silent faults happen when there is double fault in parity bit of SECDED or in the TPR and TCR unprotected part as all the TPR and TCR are not used.

IV. CONCLUSION

In this work, we present a configurable open-source tool, FISSA, to automate fault injection campaigns simulation. We illustrate, quickly, FISSA thanks to a case study. FISSA can be used with well-known HDL simulators such as Questasim. It generates TCL scripts for simulation execution and produces JSON log files for security analysis.

For future work, we plan to support new HDL simulators (Vivado, Verilator), extend the fault models supported and improve integration into the design workflow.

REFERENCES

[1] H. Bar-El *et al.*, "The Sorcerer's Apprentice Guide to Fault Attacks," *Proceedings of the IEEE*, 2006. DOI: 10.1109/JPROC.2005.862424.

[2] J. Laurent *et al.*, "Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures," in *Design, Automation & Test in Europe Conference (DATE)*, 2019. DOI: 10.23919/DATE.2019.8715158.

[3] T. Trouchkine *et al.*, "Electromagnetic Fault Injection Against a Complex CPU, toward new Micro-architectural Fault Models," *Journal of Cryptographic Engineering*, 2021. DOI: 10.1007/s13389-021-00259-6.

[4] S. Nashimoto *et al.*, "Bypassing Isolated Execution on RISC-V using Side-Channel-Assisted Fault-Injection and Its Countermeasure," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021. DOI: 10.46586/tches.v2022.i1.28-68.

[5] C. Palmiero *et al.*, "Design and implementation of a dynamic information flow tracking architecture to secure a RISC-V core for IoT applications," in *High Performance Extreme Computing*, 2018. DOI: 10.1109/HPEC.2018.8547578.