Physical Model-Assisted Secure Software Execution Against Fault-Injection Attacks

Henian Li, Md. Habibur Rahman, Arunabho Basu, Rakibul Hassan, Farimah Farahmandi, Mark M. Tehranipoor
Department of Electrical and Computer Engineering, University of Florida

***Abstract*—Modern software running on vulnerable circuits such as SoCs and processors is increasingly susceptible to cyber and hardware attacks. While protections against cyberattacks are widely integrated across the software development and compilation cycle, physical attacks such as fault-injection attacks directly target hardware implementations. Such attacks bypass upper-level protections and compromise the confidentiality, integrity, and availability of modern chips. Since the hardware has already been fabricated, the cost of redesign is extremely high, hence hardening software/firmware against the identified physical vulnerabilities would be a recommended and less costly approach. However, existing compilers lack considerations and hardening techniques for physical attacks. In this paper, we address physical threats from fault injections at the software compilation stage. Our approach models fault-injection attacks based on physical design characteristics, facilitating secure software compilations with low-cost and effective physically-aware hardening techniques.**

***Index Terms*—fault injection; physical attacks; fault model; compiler; software countermeasures**

I. Introduction

With the emergence of the Internet of Things (IoTs) regime that promises exciting new applications from smart cities to connected autonomous vehicles, security and privacy have emerged as major design challenges. These advancements mean that modern software applications running on vulnerable circuits such as SoCs and processors are facing heightened risks from both cyber and hardware attacks. Traditionally, the software has been protected against cyberattacks integrated across the software development and compilation cycle [1]. However, physical attacks (e.g., fault injection attacks) could effectively bypass these upper-level software protections, threatening the system's confidentiality, integrity, and availability.

Unlike the awareness and sophisticated protections against cyberattacks, current software developers lack information on physical attacks. Besides, existing compilers such as GCC and Clang lack considerations and hardening techniques for physical attacks. Although there are a few explorations in the literature on software code simulations and hardenings against hardware attacks [2], these efforts do not consider realistic physical attack parameters and models. As a result, only a limited number of physically vulnerable scenarios can be captured by such software-side predictions. Because the vulnerable hardware has already been fabricated, it is inaccessible to software developers and it costs highly to do a redesign, exposing the software to risks from physical attacks.

Our Contributions: In this paper, for the first time to the best of our knowledge, we propose a cost-effective and scalable approach for addressing physical threats from fault injections at the software compilation stage. Our method leverages fault-injection attack modeling based on physical design characteristics, enabling secure software compilation through precise and effective physically-aware hardening techniques.

II. Background

A. Fault-Injection Attack

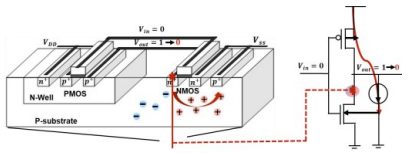


Fig. 1: Laser causing a single-event upset (SEU) [3].

Fault-injection techniques could maliciously alter the correct functionality of a computing device: Non-invasive fault-injection attacks include a clock or voltage glitching, while semi-invasive attacks such as laser fault-injection possess precise temporal and spatial control. Fig. 1 illustrates when the laser illuminations on an inverter excite extra electron-and-hole pairs, and these generated charges could potentially flip the logical value.

B. Compiler Hardening

Compiler hardening refers to the techniques used to improve the security and stability of compiled code, it has been extensively used to protect codes (e.g., in C/C++) against memory attacks and control flow integrity violation attacks. Most compilers, such as GCC and Clang, have built-in options to detect such vulnerabilities [1]. However, very limited attention has been given to identifying or addressing hardware vulnerabilities during the compilation process.

III. Physically-aware Software Compilation

The overall flow of our proposed framework shown in Fig. 2 consists of three major steps: 1) Fault-injection modeling, 2) Code Hardening, and 3) Evaluations.

Fault-Injection Modeling: We model the analog impacts of laser fault-injection attacks and correlate the physical model characteristics with the software. We then characterize the fault feasibility by modeling the attack impacts (e.g., output voltage distortion and delay variations) in SPICE simulations. Subsequently, we model the physical behavior of the attacks as laser-induced current injections within impacted standard

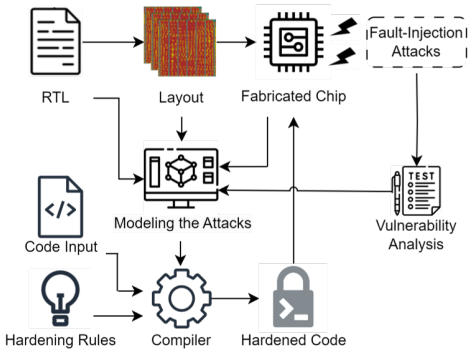
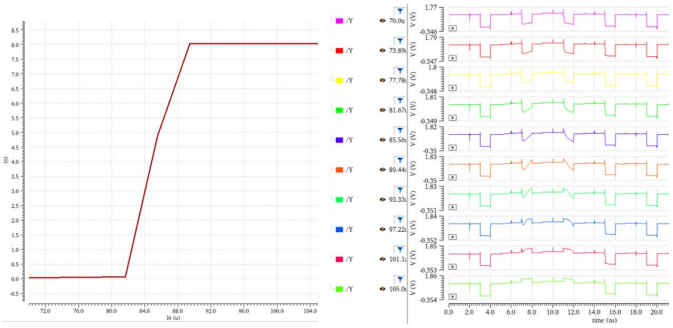


Fig. 2: Proposed physical model-assisted compilation flow.



(a) (b)

Fig. 3: Delay (a) and voltage (b) responses on AND2X1 standard cell when scaling the laser-induced current.

cells, therefore, the design's vulnerabilities can be determined and ranked by observing the current demand variations at VDD/VSS pins. The laser-induced current on impacted cells can be modeled as a double exponential current source, where $I(t)$ calculated by Equation 1 [4]:

$$I(t) = \frac{Q}{\tau_\alpha - \tau_\beta}(e^{-\frac{t}{\tau_\alpha}} - e^{-\frac{t}{\tau_\beta}}) \quad (1)$$

where Q is the number of charges, τ_α and τ_β are constants for establishing the ion track and charge collections.

Hardening Code Compilations: We perform two major steps to integrate physical attack models into the compilation process for software hardening: 1) Code feature extraction due to different code sections having varying vulnerable patterns. For example, if-condition checks, and the branched sections are critical for password authentications while not necessarily a threat for other applications. 2) Integration of hardening techniques: We develop a scalable hardening-rule database including instruction duplication, rearrangements of branch conditions, etc. These rules, combined with physical modeling information, are implemented in the form of GCC plug-ins to be integrated into regular software compilations. The final output is physically hardened assembly codes.

Evaluation: We evaluate the hardened codes' resiliency and the overhead on RISC-V processors and SoCs implemented on FPGAs. By observing the software execution's results on hardware under attack, we also validate and refine the attack models developed in the first step.

IV. Experimental Results

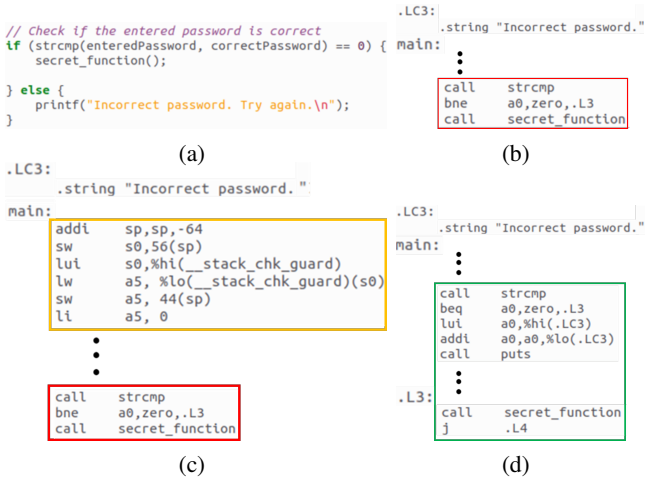
This section provides the results of laser fault-injection attack modeling and the hardened code. We developed hardening techniques on top of the RISCV-GCC compiler toolchain in Ubuntu 22.04 LTS, and used Cadence Spectre for SPICE simulations and laser modeling on Cadence 45 nm library.

Table I summarized the standard cells with the highest activities from the password-checking software. The second and third columns are the high-to-low delays without and with laser input, respectively. The delay increases while scaling up the laser intensity because the laser-induced current slows the voltage transitions, as shown in Fig. 3a. The third and fourth columns facilitate determining the threshold current for output distortion, which serves as a quantitative indicator to measure each cell's vulnerability level. Fig. 3b illustrates this trend of output distortions on AND2X1 cell when the laser intensity swept from 70 μA to 120 μA.

Based on the laser fault model applied to critical cells, we can identify that the password-checking if-condition branch in Fig. 4a is highly vulnerable. As shown in Fig. 4b, the red-highlighted instructions possess high activity in using the previously analyzed hardware resources, meanwhile, any fault

TABLE I: Output delay variations and laser current values when signal distortions start and end.

Gates	t_d (ps)	t_{d_laser} (ps)	I_{start} (μA)	I_{end} (μA)
AND2X1	11.94	22.52	130	146
OR2X1	27.35	50.12	119	131
AO21X1	22.80	40.95	122	134
NAND2X1	13.09	27.25	85	95
NOR2X1	8.71	38.22	116	147
INVX1	6.05	16.67	116	168



(a) (b) (c) (d)

Fig. 4: (a) source code in C, (b) assembly code compiled by GCC-RISCV, (c) code compiled with all GCC built-in security flags enabled, (d) hardened code from our proposed flow.

on these instruction addresses or content will lead to an instruction alteration or bypassing. E.g., skipping line *bne a0, zero, .L3* would skip the password comparison entirely, which leads to the execution of *secrect_function* regardless of the password entered. This physically vulnerable section remains unaddressed even when compiling with all GCC built-in security flags enabled (GCC protections added as in yellow in Fig. 4c). However, Fig. 4d demonstrates that the hardened code, compiled with our proposed approach, is resistant to fault-injection attacks. The vulnerable section is reorganized by our compiler plugin (highlighted in green), ensuring that skipping the critical line *beq a0, zero, .L3* would simply cause the program to branch to the scenario where an incorrect password is detected, thereby maintaining system security.

V. Conclusion

In this paper, we address physical threats from fault injections at the software compilation stage. Our approach models fault-injection attacks based on physical design characteristics, facilitating secure software compilations with low-cost and effective physically-aware hardening techniques. In the future, we plan to model other categories of physical attacks such as clock/voltage glitching and side-channel attacks, and extend the hardening database for this physically-aware compiler.

References

[1] GCC, the GNU Compiler Collection, "Program instrumentation options," https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html.

[2] L. Dureuil, G. Petiot, M.-L. Potet, T.-H. Le, A. Crohen, and P. de Choudens, "FISSC: a fault injection and simulation secure collection," in *SAFECOMP 2016 - The 35th International conference on Computer Safety, Reliability and Security*, vol. 9922, 2016, pp. 3–11.

[3] T. Farheen, S. Tajik, and D. Forte, "Spred: Spatially distributed laser fault injection resilient design," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023, pp. 1–8.

[4] F. Lu, G. D. Natale, M.-L. Flottes, B. Rouzeyre, and G. Hubert, "Layout-aware laser fault injection simulation and modeling: From physical level to gate level," in *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era*, 2014, pp. 1–6.