Enhancing the RISC-V Trace Encoder to verify the control-flow and code execution integrity

Anthony Zgheib[1,2,3*], Pierre-Alain Moellic[1,2,3*], Olivier Potin[1†], Jean-Baptiste Rigaud[1†] and Jean-Max Dutertre[1†]

[1]Mines Saint-Etienne, CEA Tech, Centre CMP, F - 13541 Gardanne, France
[2]CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne, F-13541 Gardanne, France
[3]Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France

Abstract

This paper presents control-flow and code execution integrity solutions for programs running on RISC-V cores. Our solutions are based on the RISC-V Trace Encoder (TE) that provides information about the execution path of the user's program. A first approach is compliant with the RISC-V TE standard. It detects instruction skip attacks on function calls and their returns and, attacks on branch instructions. A second approach implies an evolution of the TE specifications that permits to detect more complex fault models as the corruption of any discontinuity instruction. It covers more security properties as the program execution integrity. Our TE-based solutions were implemented on an IBEX RISC-V core and have efficiently detected simulated and experimental Fault Injection Attacks (FIA). Our verification systems do not modify the RISC-V Instruction Set Architecture (ISA), the compilation process or the user code.

Introduction

Control-Flow Integrity (CFI) [1] verification schemes are used to verify that a program is correctly executed during runtime and not altered by software or hardware attacks. It checks that its execution follows a path known to be correct in the application Control Flow Graph (CFG). In our research, we exploit the features of the Trace Encoder (TE) to verify the CFI of a program on a RISC-V core. Two approaches are proposed. One is consistent with the RISC-V TE standard [2] covering partially the CFI. The second implies an evolution of the TE – thanks to its open-source specifications — detecting more complex fault models and covering the program CFI. Based on the enhanced TE, we designed a Code and Control-Flow Integrity (CCFI) solution verifying the integrity of the executed code till the Instruction Fetch stage of a RISC-V core. However, the state-of-the-art has shown that a fault can also be injected into any micro-architectural block affecting the program execution [3]. Therefore, we developed a Control-Flow and Execution Integrity (CFEI) approach verifying the microarchitectural control signals to guarantee the program's correct execution.

Trace Encoder

The RISC-V TE [2] is an execution flow tracer that compresses at runtime the sequence of discontinuity instructions executed by the RISC-V core into trace packets. These packets sent to a debug tool allow developers to check the path followed by the program. By having access to the program binary, developers can reconstruct the program flow as depicted in Figure 1 (top). A packet defined by the TE standard [2] contains information about the path followed by the program since the last emitted packet. It is sent after fulfilling one of the TE seven conditions [2] (for example, after executing an uninferable discontinuity — return instruction). The TE alone is used for debugging purposes and allows neither CFI, CCFI nor CFEI verification.

TE-based solutions

Figure 1 (bottom) shows the Trace Verifier (**TV**) — the core of our TE-based solutions. It verifies the coherence between the program's executed path and the expected path based on its CFG. The **TV** compares TE packet contents with CFG metadata generated from our static analysis program. The metadata defined by all discontinuity instructions with their addresses and the addresses of subsequent discontinuities are stored in the **TV** memory. The **TV** based on the TE standard detects instruction skip attacks on function calls, on their returns and on branch instructions (by having a Shadow Stack in the **TV** and accessing the TE 32-bit instruction bus).

The TE standard enhancement consists in sending a packet after each discontinuity instruction, and not only by meeting the seven conditions of the actual standard [2]. Thus, the packet verification latency is decreased. With this solution **TV-CFI**, corruption of any discontinuity instruction is additionally covered by comparing the executed discontinuity to the one

*anthony.ZGHEIB@cea.fr, pierre-alain.MOELLIC@cea.fr
†olivier.potin@emse.fr, rigaud@emse.fr, dutertre@emse.fr

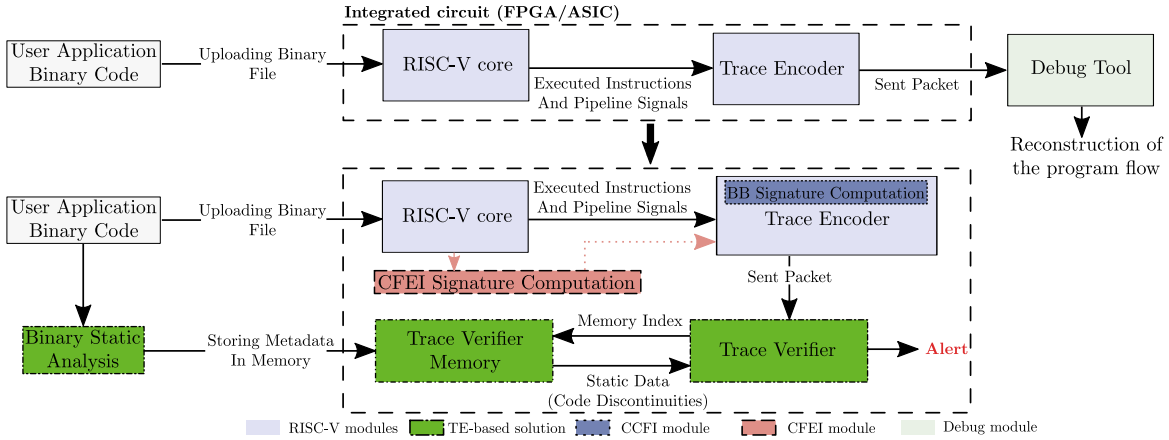

Figure 1: *A schematic of the RISC-V + TE (top) and verification solution (bottom)*

stored in the metadata. Additional signature computation modules could be added to generate signatures of Basic Block (BB) instructions or pipeline signals (cf. Figure 1). A signature sent to the **TV** via a dedicated TE packet permits CCFI or CFEI verification (expected signatures are stored in the **TV** memory). We implemented our solutions on an IBEX, a 32-bit RISC-V core with a 2-stage pipeline, and reported their efficiency against simulated Fault Injection Attacks (FIA) on "Embench-IOT" benchmarks and against experimental FIA using Electromagnetic (EM) pulses on a FPGA board. Table 1 shows the average overhead costs of our solutions. The **TV-CFEI** dedicated memory is the largest storing signatures of BB pipeline signals for the maximum coverage.

Discussion

Our solutions could be implemented to other RISC-V cores compatible with the TE. The study of the targeted architecture is required for signature calculation (instructions or control signals). The standard enhancement is done while respecting the retro-compatibility of the TE. It can run in a normal [2], CFI, CCFI or CFEI mode. Indirect calls destinations which cannot be known from the static analysis are not covered. A perspective is to avoid these calls by modifying the user code or the compiler. Moreover, the designer may need to just cover a sensitive section of the code (e.g. authentication function). It could be done by using the TE filter (cf. Chapter 5 of the TE standard [2]). It specifies the lower and higher addresses where packets need to be generated. Activating this functionality reduces drastically static data size and TV memory cost. In our core implementation, the branch prediction feature was disabled. Enabling this option emits a specific packet after a discontinuity instruction with content defined by the TE standard. The TV could be enhanced to operate in this mode.

Table 1: *TE-based solutions average overhead costs.*

	TV	**TV-CFI**	**TV-CCFI**	**TV-CFEI**
Code Size (%)	0	0	0	0
Performance (%)	0	0	0	0
Hardware Area (%)	15,9	17,2	27,9	35,1
TV Memory Size (%)	4,29	4,29	6,25	7,81

Conclusion

This paper proposes control-flow verification systems for programs executed on a RISC-V core. These solutions are based on the RISC-V Trace Encoder, a debug feature that allows to capture the execution path of a program. We developed a solution following the TE standard where skip attacks on function calls and returns are detected. We also proposed an enhancement to the standard for detecting more complex fault models as the corruption of any executed instruction. Solutions were experimented on an IBEX core both in simulation and under EM FIA. Our countermeasures do not generate performance overheads. They do not modify the RISC-V ISA, compiler or the user code. Only hardware overheads are reported. In our future work, we aim to enhance our solutions to handle interruptions and core exceptions.

References

[1] Martín Abadi et al. “Control-flow integrity principles, implementations, and applications”. In: *ACM Transactions on Information and System Security* 13.1 (Nov. 6, 2009).

[2] RISC-V. *Efficient Trace for RISC-V*. Nov. 2020. URL: `https://github.com/riscv/riscv-trace-spec`.

[3] Johan Laurent et al. “Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Mar. 2019.