SELECTIVE SPECULATION: EXPLORING INSTRUCTION BARRIERS FOR TRANSIENT EXECUTION MITIGATION

Herinomena ANDRIANATREHINA
Supervisors: Ronan LASHERMES, Simon ROKICKI, Joseph PATUREL

Abstract

- Spectre [7] mitigation through selective speculation fences on **RISC-V cores**.
- Hybrid hardware/software mitigation enables fine grained control over speculative execution.
- The solution is implemented on NaxRiscV[1] out-of-order core in combination with LLVM[2] compiler.
- Explore the usage policies and semantics of a barrier instruction.

Context

```
if (condition) {
   y = tab[*secret_address];
}
```

Speculation gadgets	Acquisition gadget	Disclosure gadget
The if statement will trigger the speculative execution of the rest of the code, it is the speculation gadget.	The value stored at secret address is going to be speculatively loaded into a register.	Accessing the tab array using the speculative data will leave a measurable trace in architectural level.

Existing mitigations

Software-based

- **Locking speculative execution:** Retpoline mitigation [5] adds an infinite loop in the speculative path. (Prevent Spectre-BTB attacks)
- **Reducing the attack window:** LFENCE/JMP Mitigation [3] inserts a serialization barrier in front of the spectre gadget.
- **Preventing the illegal access :** SLH mitigation [6] poisons loads with condition predicates. (Prevent Spectre-PHT attacks)

Hardware-based

- **Insulating the speculative state:** Duplicating all microarchitectural registers and rolling them back in case of mispeculation.
- **Detecting and preventing data leakage:** SpecTerminator[4] detects the presence of a leak and denies information disclosure.

These approaches provide a fixed implementation and do not permit the application of any policy other than the predefined one. The reproducibility of the hardware-based approach has also become a limitation.

Fence.spec rd, rs1/Fence.ser rd, rs1 mitigation semantics

Fence

Full Wait — RD ≠ X0, RS1 = X0
- Rs1 depends on all previous registers in program order.
- Any new instruction consuming rd depends on fence.

Full Fence — RD = X0, RS1 ≠ X0
- Fence depends on rs1.
- All following registers in program order dependent on fence.

Minimal fence — RD ≠ X0, RS1 ≠ X0
- Fence depends on rs1.
- Any new instruction consuming rd depends on fence.

Fence all — RD = X0, RS1 = X0
- Rs1 depends on all previous registers in program order.
- All subsequent instructions in the program order depend on fence.

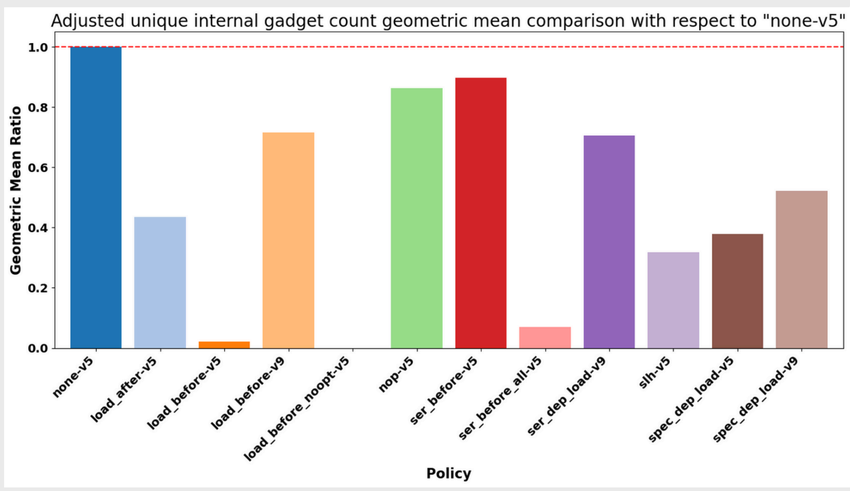
Fence.spec

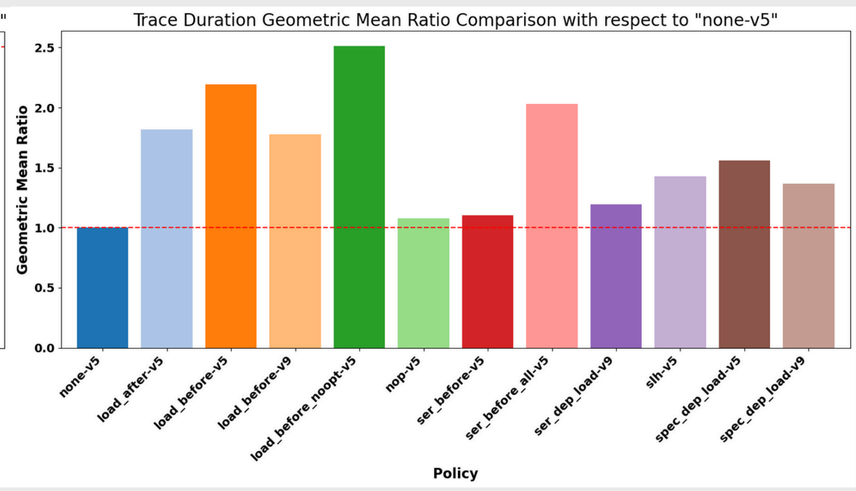
- Fence.spec is unable to complete its execution while in speculative mode.
- Ensures that the rd register is not used by other instructions in speculative mode.

Fence.ser

- Fence.ser introduces new data dependencies.
- Fence.ser can complete its execution in speculative mode.
- Somehow similar to LFENCE.

Preliminary results





Note:

- Two different implementations (V5 and V9) of fence.spec are included in NaxRiscV.
- The bar labeled 'none-V5' is used as the baseline.

Observation:

- The two figures illustrate the **trade-off between performance and security**, showing that as we gain more in security, we lose performance.

  e.g: load_before_V5 (red bin) is the most secure but the most expensive also.
- ***With Fence.spec:***
  - **Load_before** (orange bin) policy prevents unauthorized access during speculative execution.
  - **Load_after** (light blue bin) ensures that any target register provided by load instructions cannot be used speculatively. This policy prevents information leaks, but does not cover unauthorized access.

Conclusion

- Working on LLVM enables exploration of various protection policies with a fence instruction.
- The different fence.spec/ser options provide a comprehensive evaluation of the performance/security costs associated with different types of barrier instructions.

References

[1] **NaxRiscv**. https://github.com/SpinalHDL/NaxRiscv.Accessed: 2024-02-29.
[2] Chris Lattner and Vikram Adve. "**LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation**".
[3] Alyssa Milburn et al. **You Cannot Always Win the Race:Analyzing the LFENCE/JMP Mitigation for Branch Target Injection**.
[4] Hai et al. Jin. "**SpecTerminator: Blocking Speculative Side Channels Based on Instruction Classes on RISC-V**". In: ACM Trans. Archit. Code Optim. (2023)
[5] Abdul Kadir et al. "**Retpoline Technique for Mitigating Spectre Attack**". In: ICEEE 2019. 2019, pp. 96–101.
[6] Zhiyuan Zhang et al. "**Ultimate SLH: Taking Speculative Load Hardening to the Next Level**.
[7] Paul et al. Kocher. "**Spectre Attacks: Exploiting Speculative Execution**".

This work was supported by a grant from the French State, managed by the National Research Agency under the France 2030 Plan, reference