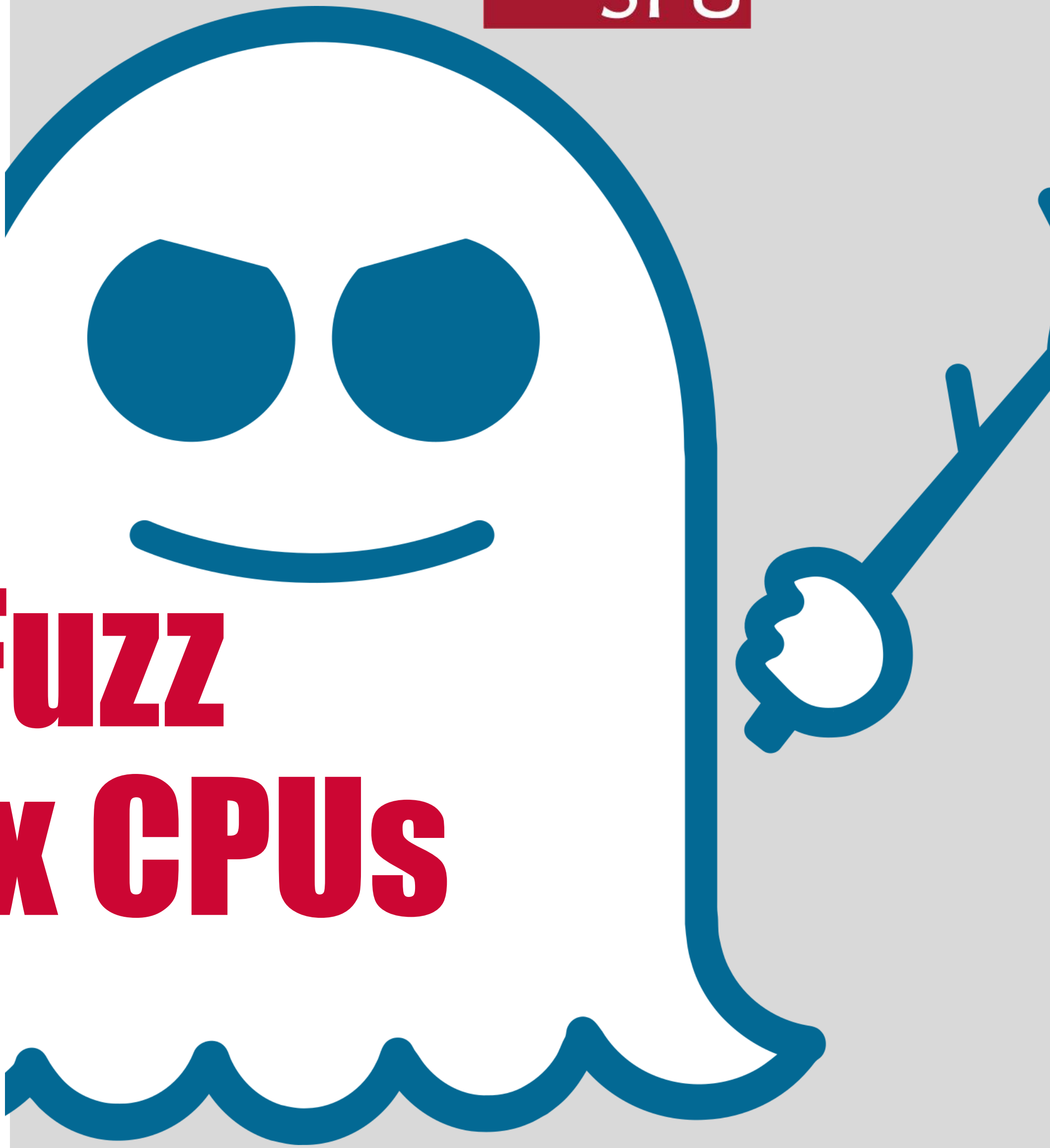
Detecting Microarchitectural Vulnerabilities via Fuzz Testing of White-box CPUs



Bo (Brian) Fu
Simon Fraser University

Alaa R. Alameldeen
Simon Fraser University

Gururaj Saileshwar
NVIDIA Research

Existing Work – Revizor (ASPLOS '22)

Idea: Fuzz µarch with generated test cases

- Randomly generated programs
  - Bounded memory accesses to a sandbox
- Randomly generated architectural state ("Input")
  - Register values and memory contents

Speculation Contracts

- Specify expected µarch side effects
- Augment ISA with "allowed" speculation clauses

Trace-generation Framework

- Contract Traces
  - Expected observations from given contract; via Unicorn emulator
- Hardware Traces
  - Actual observations from hardware; via Prime+Probe

	Observation Clause	Execution Clause
Load	`expose: ADDRESS`	None
Store	`expose: ADDRESS`	None
Cond. Jump	None	`speculate: if(INVERTED_CONDITION){ IP = IP + TARGET}`
Other	None	None

Example: MEM-COND
-Observe memory addresses, collect for both correct and mispredicted paths

$$Contract(Prog, Data) = Contract(Prog, Data')$$
$$\Longrightarrow Attack(Prog, Data, Ctx) = Attack(Prog, Data', Ctx)$$

For the same program, test with different inputs, and check if traces match.
If NOT: Contract violation, Side-Channel found!

Credit for both figures: O. Oleksenko, C. Fetzer, B. Kopf, and M. Silberstein, "Revizor: testing ¨ black-box cpus against speculation contracts," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2022

ISSUE:

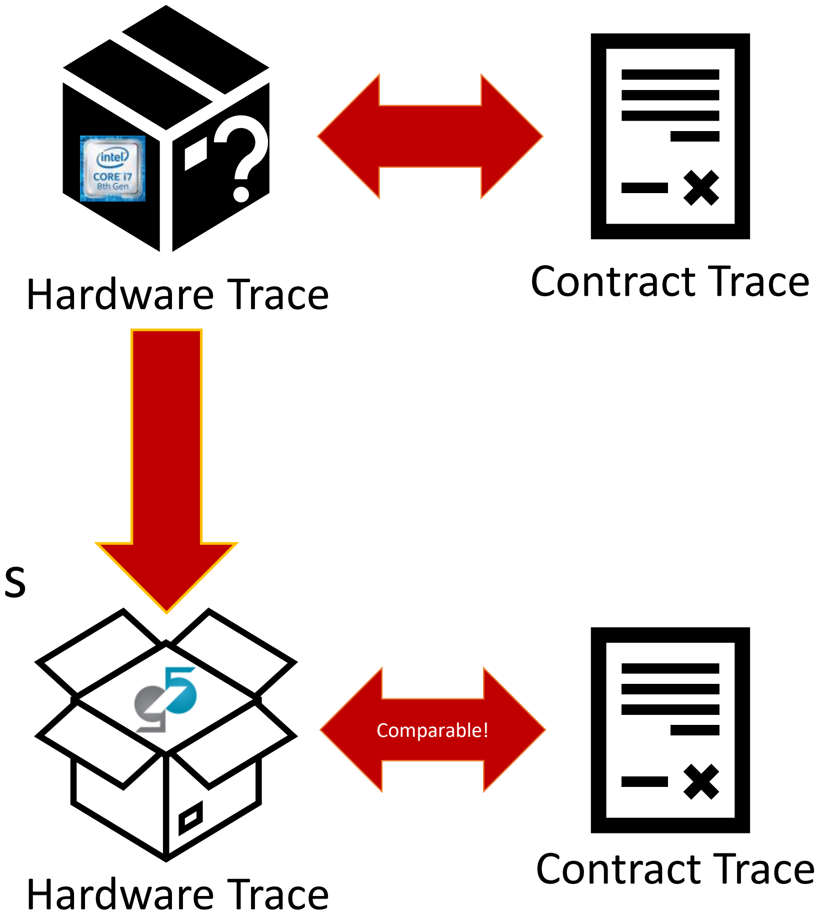
- No relation between contract and hardware traces directly
- No insight into how hardware trace is generated

$\Rightarrow$ Can only construct an existence proof of a vulnerability

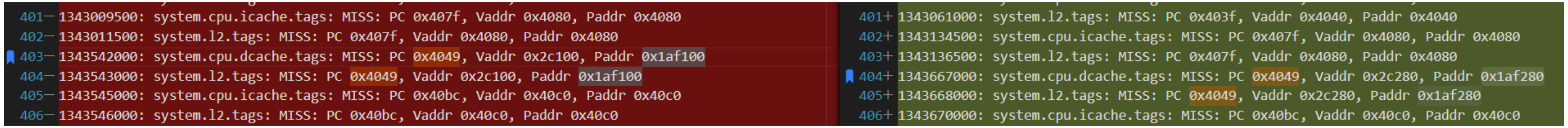
SOLUTION - <u>White-box system</u>:

- Can understand exactly how hardware trace is generated;
  - E.g. Which cache block miss caused traces to differ
- Can correlate contract (expected) and hardware (actual) traces
- Much easier to understand what is wrong

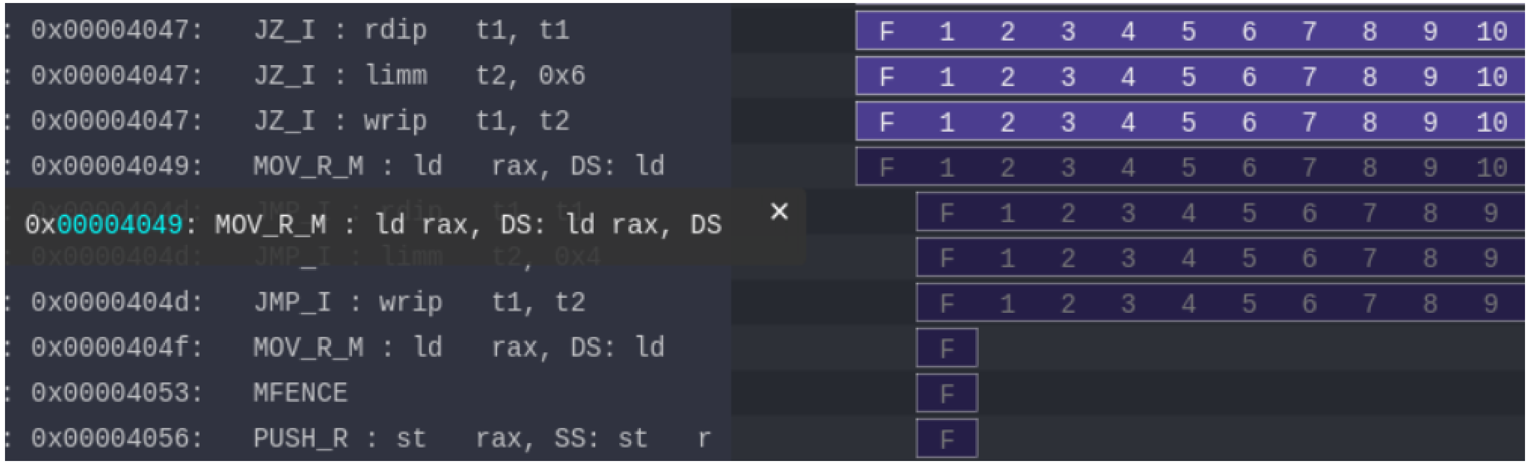
Importantly: Provides a method to validate (or disprove) existing speculation defenses e.g. Invisi-Spec, CleanupSpec



Example – Spectre v1 (Spec. Store Bypass):



For 2 different inputs: Can see which memory addresses were accessed by the transient miss; 0x2c100 for the RHS, 0x2c280 for the LHS (VAddr)



Can be sure a contract violation occurs & results are not false positive.

Great speedup vs. original Revizor, which must repeatedly check to ascertain findings are real.

Can also measure other µarch structures i.e. LSQ Unit, TLB, BTB, etc. & look for possible side channels there! (via Speculation Contract Violation)

Goals & Future of our work:

- Implement our changes in existing gem5 implementations of speculative defenses (Invisi-Spec, CleanupSpec, etc.) and validate them, finding existing known vulnerabilities (and maybe unknown ones!)

- Smart mutation of generated programs
  - Don't test previously tested µarch states/contexts
  - Can directly set initial µarch context as another input

- Detect vulnerabilities in future proposed defenses
  - Create an extensible fuzzing framework that any gem5-based defense may be plugged into, after running a script to make our required modifications (within gem5)