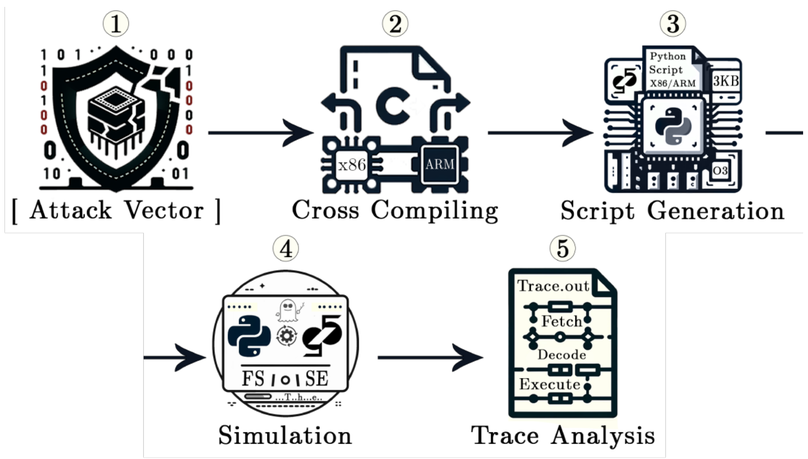
Decoding Attack Behaviors by Analyzing Patterns in Instruction-Based Attacks using gem5.

Muhammad Awais, Maria Muhstaq, Lirida Niviner, Florent Bruguiere, Jawad Haj Yahya, Pascal Boniot.

ABSTRACT

The diversity of Instruction Set Architectures (ISAs) presents opportunities and challenges in processor design, particularly for enhancing security, reliability, and performance. Recent vulnerabilities like Spectre and Meltdown underscore the need for robust hardware security. This paper uses gem5, a cycle-accurate simulation tool, to simulate the Spectre attack, with scripts developed for x86 and ARM architectures. By analyzing cache and branch prediction data from gem5 traces, specific attack patterns useful for detection were identified. Future work will expand the analysis to include more attack vectors and reproduce cache-based attacks like Flush+Reload to enhance mitigation strategies.



MOTIVATION AND CONTEXT

- **Microarchitectural side-channel Attacks:** exploit unintended information leakage from hardware components, such as caches or branch predictors, to infer sensitive data.
- **Out-of-order Execution:** is a performance-enhancing technique where a processor executes instructions based on the availability of execution units and resources rather than their original order in the program.
- **Branch Prediction:** Branch prediction can introduce security vulnerabilities, such as Spectre, by allowing attackers to exploit speculative execution to access sensitive data through side channels.
- **Gem5:** is a simulation tool that allows researchers to model and analyze processor architectures, including the security implications of various microarchitectural features and attacks.

EXPERIMENTS AND RESULTS

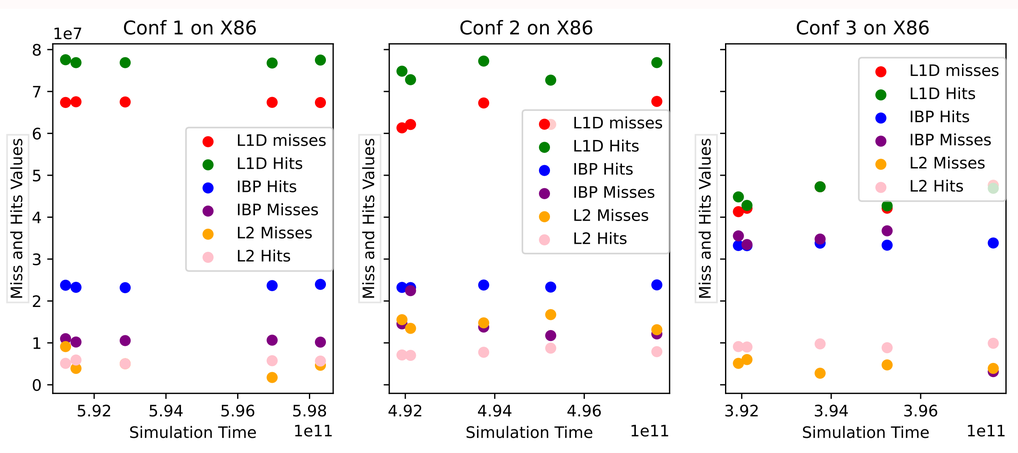
Cache Configuratoin

- Making cache with the configuration listed:

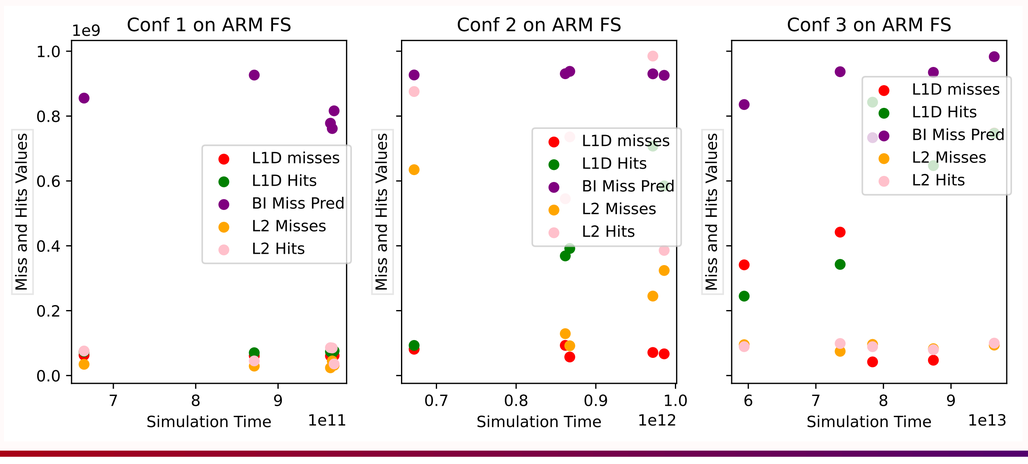
Conf No	L1 Instruction cache, L1 Data cache and L2 Cache Configuration							
	Caches	Assoc	Tag Latency	Data Latency	Resp Latency	Mshrs	Tgts-per-mshr	Size
Conf 1	L1 Ins	2	2	2	2	2	4	16KB
	L1 Data	2	2	2	2	2	4	64KB
	L2 Cache	4	4	4	2	4	8	128KB
Conf 2	L1 ins	4	2	2	2	4	8	32KB
	L1 Data	4	2	2	2	4	8	128KB
	L2 Cache	8	4	4	2	8	16	256KB
Conf 3	L1 ins	8	2	2	2	4	8	64KB
	L1 Data	8	2	2	2	4	8	256KB
	L2 Cache	16	4	4	2	8	16	1024KB

The configuration that we use the test our resutls is the `L1I` with size `16KB` initially and `L1D` with `64KB` with `L2 128KB`. the other cache conf are `L1I` with `32KB`, `l1D` with `128KB` with `L2` of `256KB`. We test our scripts with different cache assosiatitvity, Tag, Data and responce Latency.

- **X86 Specter: Cache configuration Results with the correct predictions.**



- **ARM v8 Specter: Cache configuration Results with the correct predictions.**



Speculative traing patteren.

- Case 1 (Speculative excution of the instruction)
- By analyzing the trace files after running the simulation script in the SE mode and FS mode in gem5.
- We analyze the Speculative excution pattrens and the instructions that are excuted in the O3 processor, patternss from the O3 PipeView utility of the gem5.

```
mov rsi, secret_index ; Index that triggers out-of-bounds
rsi, array1_size ; Check bounds
jae skip ; Jump if out-of-bounds
mov rdx, [rax + rsi*4] ; Speculatively execute out-of-bounds
shl rdx, 0xC
mov al, [rbx + rdx] ; Acc array2[array1[secret_index]*4096]
skip:
```

Branch prediction patteren.

- Case 2 (Miss prediciting the branch prediciton)

```
train_loop:
mov rdx, [rax + rsi*4] ; Load array1[i] into rdx
shl rdx, 0xC ; Multiply by 4096
mov al, [rbx + rdx] ; Access array2[array1[i]*4096]
add rsi, 1
cmp rsi, 100
jl train_loop
```

- First loading the `array[i]` the victim address, and the multiply by 4096 and then access the predicited `array2`.

Discussions.

- **Configuration Metrics and Simulation Time:** we run simulation multiple time with the conf 1 2 and 3 respectively in SE mode and also in FS Mode.
- **Performance analysis:** for the X86 we get the results with in the 5 mintues in SE mode and in 30 min FS mode, And for the ARM simulation it takes upto 5 hours form 1 run.
- In the analysis of the trace file we use the `m5.write(1, 'byte recovered,15)`, in the source file, and the in the trace file we make the patteren by analyzing the previous instructions

CONCLUSION AND FUTURE WORK

- The traces generated by gem5 provide extensive insights into microarchitectural behavior, which are highly beneficial for analysis.
- By developing additional patterns, we can create a comprehensive pattern vector to facilitate the development of diagnostic tools.
- In the future, we plan to conduct further side-channel attack simulations using gem5 to identify additional patterns

Contact : muhammad.awais@telecom-paris.fr http://www.telecom-paris.fr