Spectector: Principled detection of speculative information flows

M. Guarnieri, B. Köpf, J. F. Morales, J. Reineke, A. Sánchez

To appear at ***IEEE Symposium on Security and Privacy 2020***

How can we reason about speculative leaks?

Speculative semantics

- Parametric in branch predictor
- Execute mispredicted branches for fixed number of steps
- Backtrack on wrong decisions

Attacker model

Attacker observes:

- locations of ***memory accesses***
- ***branch/jump*** targets
- ***start/end*** speculative execution

Speculative non-interference

Compares a program's leakage w.r.t. two semantics:

- Standard, non-speculative semantics (proxy for intended program behavior)
- Speculative semantics (proxy for speculative leaks)

Formally:

A program **P** is ***speculatively non-interferent*** if

∀ program states ***s*** and ***s′***,

$$\mathbf{P}_{\text{non-spec}}(s) = \mathbf{P}_{\text{non-spec}}(s') \Rightarrow \mathbf{P}_{\text{spec}}(s) = \mathbf{P}_{\text{spec}}(s')$$

Automatically detecting speculative leaks!

Spectector

```
    mov rax, A_size
    mov rcx, x
    cmp rcx, rax
    jae END
L1: mov rax, A[rcx]
    mov rax, B[rax]
```

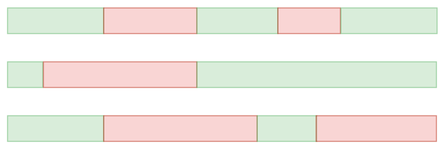
Assembly program

x64 to μASM

```
    rax <- A_size
    rcx <- x
    beqz rcx<rax, END
L1: load rax, A + rcx
    load rax, B+rax
END:
```

μASM program

Symbolic execution



Symbolic traces

Speculative leaks detection

Security decision

Case study: compiler countermeasures

Goals:

- Determine if speculative non-interference captures speculative leaks on 15 variants of SPECTRE v1
- Assess Spectector's precision

How:

Analyze 240 microbenchmarks obtained by compiling 15 variants of SPECTRE v1 with Clang, Intel ICC, and Microsoft Visual C++

Results:

Ex.	Vcc						Icc				Clang					
	Unp		Fen 19.15		Fen 19.20		Unp		Fen		Unp		Fen		Slh	
	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2	-O0	-O2
01	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
02	○	○	●	●	●	●	○	○	●	●	○	○	●	●	●	●
03	○	○	●	○	●	●	○	○	●	●	○	○	●	●	●	●
04	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
05	○	○	●	○	●	○	○	○	●	●	○	○	●	●	●	●
06	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
07	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
08	○	●	○	●	○	●	○	●	●	●	○	●	●	●	●	●
09	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
10	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	○
11	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
12	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
13	○	○	○	○	○	○	○	○	●	●	○	○	●	●	●	●
14	○	○	○	○	●	●	○	○	●	●	○	○	●	●	●	●
15	○	○	○	○	○	○	○	○	●	●	○	○	●	●	○	●

Case study: Xen Project hypervisor

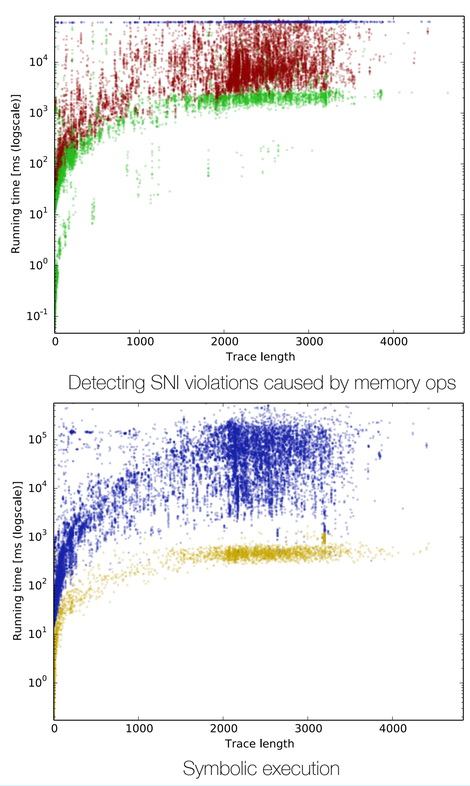
Goal:

Assess Spectector's scalability

How:

Compare time spent on discovering new symbolic paths with the time spent on checking SNI

Results:



Symbolic execution

Symbolic execution versus checking SNI

Supported by Intel Strategic Research Alliance (ISRA) "Information Flow Tracking across the Hardware-Software Boundary"

Available at: ***https://spectector.github.io***

Contact: ***marco.guarnieri@imdea.org***

We're ***hiring*** interns and PhD students!