Secure compilation of speculative-constant-time programs

Roméo La Spina

Internship supervised by Vincent Laporte (LORIA)
ENS Rennes

May 10th - July 16th, 2021

Motivations

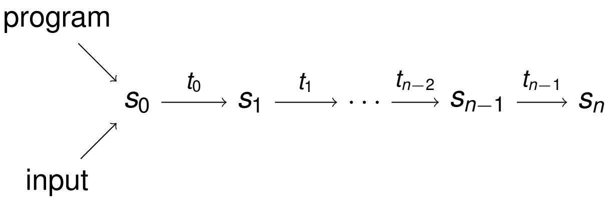
Warning, speculative execution is efficient but vulnerable...
Do you really trust your compiler?

A vulnerability caused by speculative execution...

```
fn PHT(stack u64[8] a b, reg u64 x) → reg u64 {
  reg u64 i r;
  if (x < 8) {          // Speculatively bypass check
    i = a[(int) x];     // Speculatively read secrets
    r = b[(int) i];     // Secret-dependent access
  }
  return r;
}
```

Semantics

State = code + current environment
Small-step semantics:



Adversarial Semantics

The attacker partially control the execution...

$\Rightarrow$ In addition, a directive (attack) and an observation (leak)

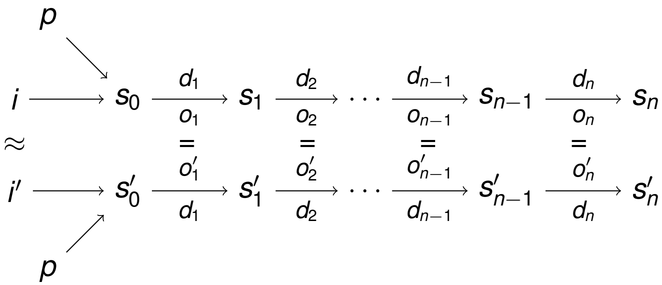
$$s \xrightarrow[d]{o} s'$$

Example : rule for condition

$$\frac{}{S(\text{if } e \text{ then } c_\top \text{ else } c_\bot; c, \ldots) \xrightarrow[\text{force } b_f]{\text{branch } [\![e]\!]_\rho, b_f} S(c_{b_f}; c, \ldots)}\ [\textsc{Cond}]$$

Speculative Constant-Time (SCT)

A program p is SCT w.r.t. to an equivalence relation $\approx$ if...



Both executions are indistinguishable for the attacker!

A first approach

Question. Given a SCT source program, is the compiled program still SCT?

Problem. The attacker gives a list of directives on the target program. How to reflect its impact on source program?

	Source		Target
Code	`if 0 then x := 3` `else x := 1`	$\Rightarrow$	`x := 1`
Directives	force $\bot$; step	$\Leftarrow$	step
Observations	branch $\bot$,$\bot$; $\bullet$	$\Rightarrow$	$\bullet$

Program transformation

A transformation take a source program p, and produces three objects collectively noted $[\![p]\!]$:

- a target program

$$[\![p]\!]$$

- a directive transformer (from target dirs. to source dirs.):

$$[\![p]\!](dl_t) = dl_s$$

- an observation transformer (from source obs. to target obs.):

$$[\![p]\!](ol_s) = ol_t$$

Correctness

$$p,\ i \longrightarrow s_0 \xrightarrow[\llbracket p \rrbracket(dl)[0]]{ol[0]} s_1 \xrightarrow[\llbracket p \rrbracket(dl)[1]]{ol[1]} \cdots \xrightarrow[\llbracket p \rrbracket(dl)[n-1]]{ol[n-1]} s_n$$

$$\Downarrow$$

$$\llbracket p \rrbracket,\ i \longrightarrow s'_0 \xrightarrow[dl[0]]{\llbracket p \rrbracket(ol)[0]} s'_1 \xrightarrow[dl[1]]{\llbracket p \rrbracket(ol)[1]} \cdots \xrightarrow[dl[n-1]]{\llbracket p \rrbracket(ol)[n-1]} s'_n$$

SCT-preservation

SCT-preservation

Definition. A transformation $[\![\cdot]\!]$ *preserves SCT* iff, for any equivalence relation φ that guarantees safety for source program, and for all p,

$$SCT_\varphi(p) \Rightarrow SCT_\varphi([\![p]\!])$$

Theorem. Every correct transformation preserves SCT.
Proof. In Coq...

Application

- Design of a toy language, including Fence instructions
- Coq proof of correctness (and therefore SCT-preservation) of two simple transformations:
  - Branch elimination:

```
if 0 then x := 3 else x := 1
              ⇓
           x := 1
```

  - Array concatenation (with `a` of length `n`):

```
a[0] := 1; b[1] := 2
         ⇓
c[0] := 1; c[n+1] := 2
```

Conclusion

- First approach to design SCT-preserving compilers
- Transformations adding speculation? (when the compiler need to introduce Fences)

Speculative Constant-Time

Speculative Constant-Time (SCT)

Definition. A program p is SCT with respect to a low-equivalence relation φ (denoted $SCT_\varphi(p)$) iff for all $dl, i_1, i_2, s'_1, s'_2, ol_1, ol_2$,

$$\varphi(i_1, i_2) \Rightarrow p(i_1) \xrightarrow[dl]{ol_1}{}^{*} s'_1 \Rightarrow p(i_2) \xrightarrow[dl]{ol_2}{}^{*} s'_2 \Rightarrow ol_1 = ol_2$$

Same public values $\rightarrow$ same observations.

Properties

Correctness

Definition. A transformation $[\![\cdot]\!]$ is *correct* iff for all dl, p, i, s', ol,

$$p(i) \xrightarrow[[\![p]\!](dl)]{ol}{}^{*} s' \Rightarrow \exists s'', [\![p]\!](i) \xrightarrow[dl]{[\![p]\!](ol)}{}^{*} s''$$

SCT-preservation

Definition. A transformation $[\![\cdot]\!]$ *preserves SCT* iff, for any equivalence relation φ that guarantees safety for source program, and for all p,

$$SCT_{\varphi}(p) \Rightarrow SCT_{\varphi}([\![p]\!])$$