ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

November 8th 2022

Under submission

Lesly-Ann Daniel
KU Leuven

Marton Bognar
KU Leuven

Job Noorman
KU Leuven

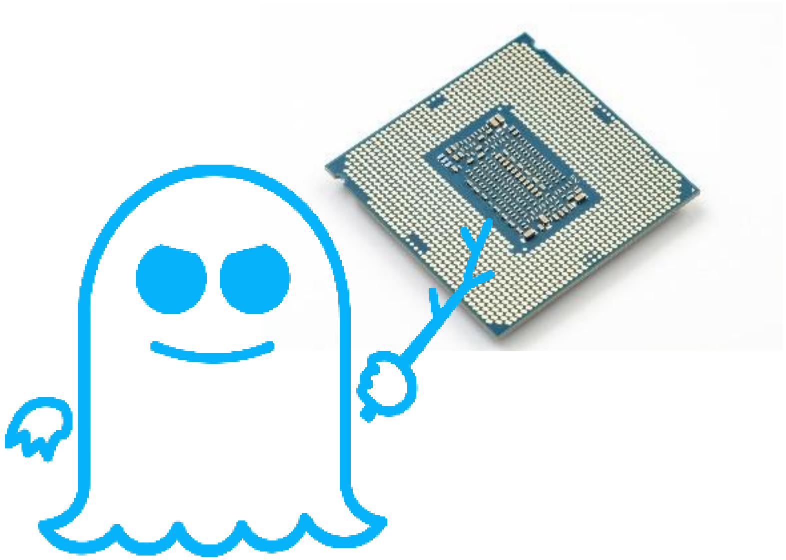
Sébastien Bardin
CEA List

Tamara Rezk
INRIA

Frank Piessens
KU Leuven

Spectre attacks

- Speculative out-of-order execution is powerful
- Speculation may lead to transient executions
- Transient executions are reverted at architectural level
- But *not the microarchitectural state* (e.g. cache)



Spectre attacks (2018)

Idea. *Force victim to encode secret data in cache during transient execution & recover them with microarchitectural attacks*

Hardware-Software Contracts

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri*, Boris Köpf†, Jan Reineke‡, and Pepe Vila*
*IMDEA Software Institute †Microsoft Research ‡Saarland University

Formally reason about defenses & Enable hardware-software co-design

Foundational Framework

- Secure software design, verification and compilation
- *Formally express guarantees of hardware defenses*

Hardware-Software Contracts

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri[*], Boris Köpf[†], Jan Reineke[‡], and Pepe Vila[*]
[*]*IMDEA Software Institute* [†]*Microsoft Research* [‡]*Saarland University*

Formally reason about defenses & Enable hardware-software co-design

Foundational Framework

No hardware defense studied in the paper enables secure speculation for constant-time programs!

Secure Speculation for Constant-Time?

Constant-time Programming

Protection against (non-transient) microarchitectural attacks

- Used in many cryptographic implementations
- No secret-dependent control flow & memory accesses

Constant-Time in the Spectre Era

- Speculative semantics for software defenses
  → Hard to reason about & accommodate new speculation mechanisms?

- Hardware defense: disable speculation
  → Not acceptable

Secure Speculation for Constant-Time

Hardware defense

Efficient: enables speculation

Constant-time programs do not leak

Developer can ignore speculation

Hardware Secrecy Tracking

Hardware Secrecy Tracking (HST)

- Inform hardware of what is secret
- Track secret taint in hardware
- Do not leak tainted values during speculation

ConTExT: A Generic Approach for Mitigating Spectre

Michael Schwarz[1], Moritz Lipp[1], Claudio Canella[1], Robert Schilling[1,2], Florian Kargl[1], Daniel Gruss[1]
[1]Graz University of Technology [2]Know-Center GmbH

SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

Jacob Fustos, University of Kansas
Farzad Farshchi, University of Kansas
Heechul Yun, University of Kansas

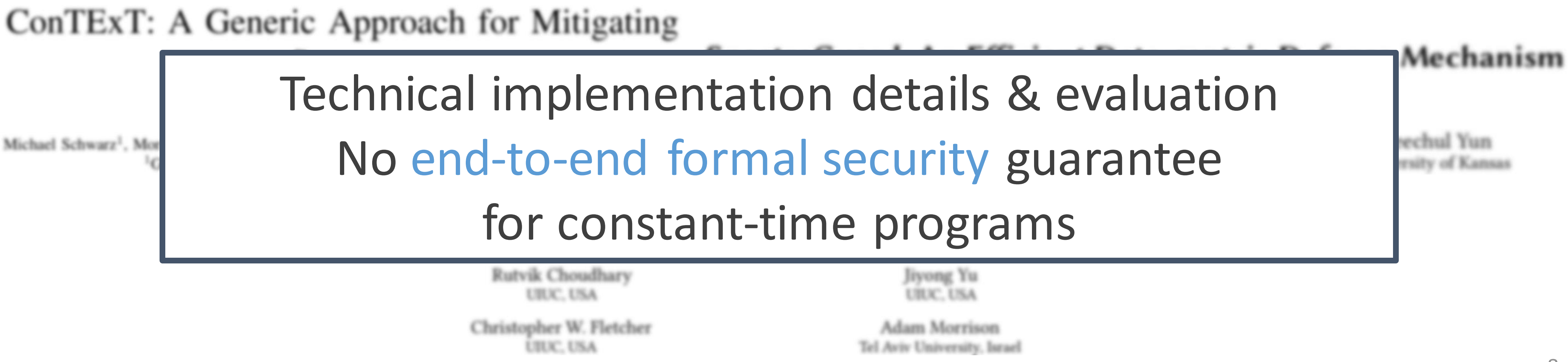
Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy

Rutvik Choudhary, UIUC, USA
Jiyong Yu, UIUC, USA
Christopher W. Fletcher, UIUC, USA
Adam Morrison, Tel Aviv University, Israel

Hardware Secrecy Tracking

Hardware Secrecy Tracking (HST)

- Inform hardware of what is secret
- Track secret taint in hardware
- Do not leak tainted values during speculation



ConTExT: A Generic Approach for Mitigating

Mechanism

Michael Schwarz[1], Mo

echul Yun

rsity of Kansas

Rutvik Choudhary
UIUC, USA

Jiyong Yu
UIUC, USA

Christopher W. Fletcher
UIUC, USA

Adam Morrison
Tel Aviv University, Israel

Technical implementation details & evaluation
No end-to-end formal security guarantee
for constant-time programs

What we propose

ProSpeCT: Formal processor model with HST

- Generic: wide range of speculation mechanisms

Proof that CT programs do not leak secrets

- All Spectre variants + LVI
- Allows for *declassification*

First to consider Load Value Speculation

- Novel insight: sometimes need to rollback *correct* speculations for security

Implementation in a RISC-V microarchitecture

- First synthesizable implementation
- Evaluation: hardware cost, performance, annotations

ProSpeCT
Secure Speculation for Constant-Time

Illustration with Spectre-v1

Spectre-v1. Exploit branch prediction

```
char A[16]
char secret
if (idx < 16)
  x = load A[idx]
  leak(x)
```

Consider `idx = 16`

No defense

Mispredicted

`x = secret`

`secret` is transiently leaked !

Illustration with Spectre-v1

```
char A[16]  // public memory
char secret // secret memory
if (idx < 16)
  x = load A[idx]
  leak(x)
```

Consider `idx = 16`

ProSpeCT

Developer annotate secret memory

Mispredicted

`x = secret:H`

`secret` is not forwarded to `leak`

Illustration with LVI

LVI. Inject values at faulting loads

```
char A[16]
char secret
x = load idx
y = load A[x]
leak(y)
```

No defense

Attacker inject `x = 16`

`y = secret`

`secret` is transiently leaked!

Akin to Load Value Prediction

Illustration with LVI

```
char A[16]  // public memory
char secret // secret memory
x = load idx
y = load A[x]
leak(y)
```

Akin to Load Value Prediction

ProSpeCT

Developer annotate secret memory

Attacker inject `x = 16`

`y = secret:H`

`secret` is not forwarded to `leak`

Design Choices

Software side	Hardware side
• Label secret memory • Constant-time program • Secret written to public memory is declassified	• Track security labels • Secrets do not speculatively flow to insecure instructions • Predictions do not leak secrets

Code without secret $\Longrightarrow$ free speculation

Constant-time programs $\Longrightarrow$ only block mispredictions

ProSpeCT: Generic formal processor model for HST

Semantics of out-of-order speculative processor with HST

$$(a, \mu) \xrightarrow{d} (a', \mu')$$

- *declassification trace* (d)
- *microarchitectural context* (μ')
- *architectural state* (a')

Abstract microarchitectural context μ
+ Functions $update, predict, next$

{ Observations of attacker
  Influence of attacker

At each step: μ is updated with *all* public values
→ predictions can depend on any public value

Secure Speculation for Constant-Time Policy

Security (no decl). *For all constant-time program (architectural semantics)*

if $a_0 =_{public} a'_0$ and $(a_0, \mu) \longrightarrow^n (a_n, \mu_n)$

then $(a'_0, \mu) \longrightarrow^n (a'_n, \mu'_n)$ and $\mu_n = \mu'_n$

Architectural semantics = hardware software security contract

Secure Speculation for Constant-Time Policy

Security (decl). *For all constant-time program up to declassification*

$$\text{if } a_0 =_{public} a_0' \text{ and } (a_0, \mu) \xrightarrow{d}{}^{n} (a_n, \mu_n)$$

$$\text{then } (a_0', \mu), d \hookrightarrow^{n} (a_n', \mu_n') \text{ and } \mu_n = \mu_n'$$

Declassify ciphertext while still protecting plaintext

Load Prediction: Rollback correct executions?

```
char secret // secret memory
x = load secret
y = x + 4
```

	Fetch	Predict 0	Resolve prediction	
secret=0	x = load secret y = x + 4	x = 0 y = 4	x = 0 y = 4	*Implicit resolution-based channel*
secret=1	x = load secret y = x + 4	x = 0 y = 4	x = 1	Commit / Rollback can be distinguished

Load Prediction: Rollback correct executions?

```
char secret // secret memory
x = load secret
y = x + 4
```

	Fetch	Predict 0	Resolve prediction
secret=0	x = load secret y = x + 4	x = 0 y = 4	x = 0:H
secret=1	x = load secret y = x + 4	x = 0 y = 4	x = 1:H

Always rollback when actual value is secret

Implementation and Evaluation

Implementation

Prototype Risc-V implementation

- On top of Proteus modular RiSC-V processor
- Will be open-sourced
- Limitation
  - Only branch prediction
  - Secrets not forwarded *at all* during speculation (conservative)

Evaluation

Preliminary Evaluation

- Hardware cost
- Labelling secrets
- Runtime overhead

Synthesized on FPGA

– LUTs: +9.6%

– Register: +4.8%

– Critical path: +3.3%

4 primitives (HACL*)

– Annotate public/secret

– Ensure no secret spilled

– Stack public in 3 cases

– Easy: ≤1h/primitive

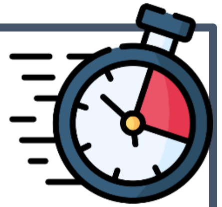
Runtime Overhead

Benchmark [1]

- Amount of secret
- Speculation-heavy public computations / crypto

spec/crypto	25/75	50/50	75/25	90/10
None	100%	100%	100%	100%
Secret	100%	100%	100%	100%
All	109%	125%	136%	145%

Conclusion



Results similar to [1]

Low overhead when secret annotation is precise and restricted part of code compute on secrets

[1] Jacob Fustos, Farzad Farshchi, and Heechul Yun. "SpectreGuard: An Efficient Data-Centric Defense Mechanism against Spectre Attacks". In: DAC. 2019

Conclusion

Hardware Secrecy Tracking

Software informs hardware about secret

Strong security guarantees

ProSpeCT ⟹ end-to-end security for constant-time programs

Low overhead

ProSpeCT ⟹ no runtime overhead on public data

Credit

Icons made by **Freepik**
from www.flaticon.com

Diamond icons created by **Vectors Market** – Flaticon
www.flaticon.com/free-icons/diamond

Hard work icon created by **monkik** – Flaticon
www.flaticon.com/free-icons/hard-work

Backup

Future Work

Formal model

- Express details of existing HST defenses in our model

Compiler-support

- Separate secret from public memory
- Ensure no unintentional declassification

Validate RISC-V implementation

- Contract-based CPU testing (e.g., Revizor, Scam-V)?
- Hardware-fuzzing / Model checking?

Secure Speculation for Constant-Time Policy

Security without declassification:

If program is constant-time (sequential semantics), then secrets do not leak to μ in our hardware (speculative) semantics

Security with declassification

If program is constant-time up to declassification (sequential semantics), secrets do not leak to μ (speculative semantics).