ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

March 28th 2023

Journées du GT-MFS

To Appear
USENIX Security 2023

Lesly-Ann Daniel	Marton Bognar	Job Noorman	Sébastien Bardin	Tamara Rezk	Frank Piessens
KU Leuven	KU Leuven	KU Leuven	CEA List	INRIA	KU Leuven

Speculative execution is powerful ☺ …

```
char A[16]

if (idx < 16)
  x = load A[idx]
  compute(x)
```

Speculate instead of stalling!

Good prediction: performance gain!

Bad prediction (transient executions): revert changes and continue.

Processor speculates on branch targets, store-to-load dependencies, etc.

… but leads to Spectre attacks ☹

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

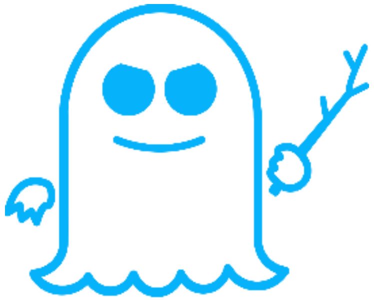
Mispredicted with `idx = 16`

`x = secret`

Leaks `secret` to cache!

Changes to *microarchitectural state* (e.g. cache) are not reverted!

Idea. *Force victim to leak secret data during transient execution and recover them with microarchitectural attacks*

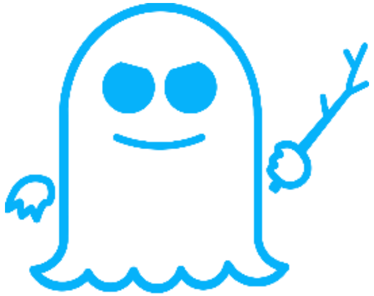


Constant-Time vs Spectre?

Even constant-time programs are vulnerable to Spectre ☹!

Constant-time

- Protection against (non-transient) microarchitectural attacks
- No secret-dependent control flow & memory accesses
- Used in many cryptographic implementations



Constant-Time in the Spectre Era

- Speculative semantics for software defenses

→ Hard to reason about

→ Accommodate new speculation mechanisms?

Secure Speculation for Constant-Time!

Developers should not care about speculations

Hardware should not speculatively leak secrets

But still be efficient and enables speculation

Hardware defense:

Secure speculation for constant-time!

How do I know that my defense works?

Hardware-Software Contracts for Secure Speculation

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

Formalize hardware leakage as a **contract**

Software side	Hardware side
Program secure software wrt. contract • Secure software design • Verification • Compilation	Hardware complies with contract • *Formally express guarantees of hardware defenses*

Hardware-Software Contracts for Secure Speculation

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

Formalize hardware leakage as a **contract**

Software side	Hardware side
Pro	es

No secure speculation for constant-time programs!

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

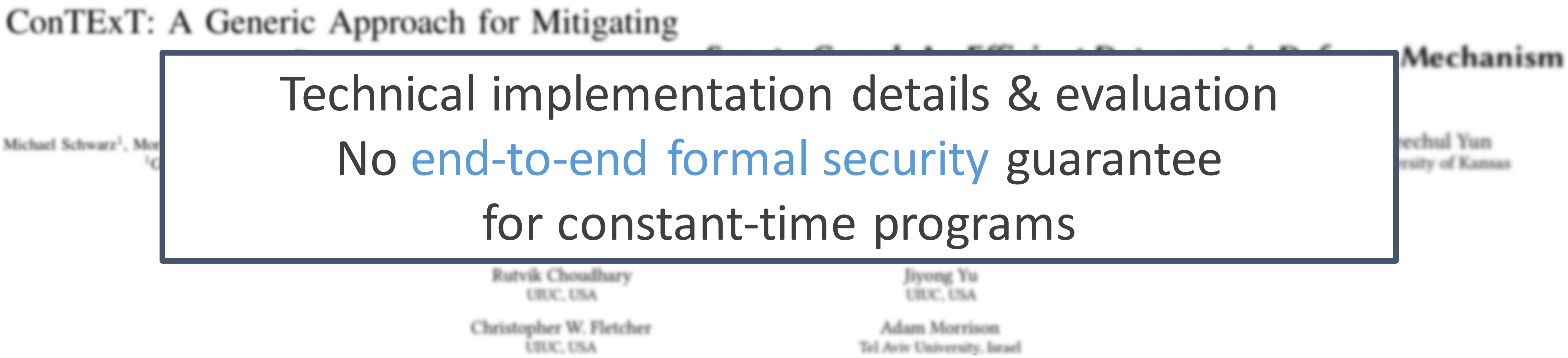
Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy

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

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

Challenges

- Account for all existing speculation mechanisms
- Account for futuristic speculation mechanisms
- Account for declassification
- Adapt HW/SW contract framework for these new features
- Evaluation: hardware costs?

Our contributions

ProSpeCT: Formal processor model with HST

- Proof: constant-time programs do not leak secrets
- Generic: 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

Spectre-v1. Exploit branch prediction

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

ProSpeCT

Developer annotates secret memory

Consider `idx = 16`

Illustration with Spectre-v1

Spectre-v1. Exploit branch prediction

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

ProSpeCT

Developer annotates secret memory

Prediction

`x = secret:H`

Consider `idx = 16`

Illustration with Spectre-v1

Spectre-v1. Exploit branch prediction

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

Consider `idx = 16`

ProSpeCT

Developer annotates secret memory

Prediction

`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)
```

Akin to Load Value Prediction

No defense

Attacker injects `x = 16`

`y = secret`

`secret` is transiently leaked!

Illustration with LVI

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

ProSpeCT

Developer annotates secret memory

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)
```

ProSpeCT

Developer annotates secret memory

Attacker injects `x = 16`

`y = secret:H`

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 annotates secret memory

Attacker injects `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

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

architectural state *microarchitectural context* *declassification trace*

ProSpeCT: Generic formal processor model for HST

Semantics of out-of-order speculative processor with HST

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

Abstract microarchitectural context μ

+ Functions $update, predict, next$

$\Bigg\{$ Attacker observations

Attacker influence

ProSpeCT: Generic formal processor model for HST

Semantics of out-of-order speculative processor with HST

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

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

{ Attacker observations
Attacker influence

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 declassification).

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 (with declassification).

For all constant-time program up to declassification

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

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

Execution 1: `secret=0`

Execution 2: `secret=1`

Predict load value to 0

`x = 0 (?); y = 4`

`x = 0 (?); y = 4`

Load Prediction: Rollback correct executions?

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

Execution 1: `secret=0`

Execution 2: `secret=1`

Predict load value to 0

`x = 0 (?); y = 4` | `x = 0 (?); y = 4`

Resolve

`x = 0; y = 4` | `x = 1`

Commit if secret = 0 vs Rollback if secret ≠ 0

⟹ *Implicit resolution-based channel*

Load Prediction: Rollback correct executions?

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

Execution 1: `secret=0`

Execution 2: `secret=1`

	Execution 1	Execution 2
Predict load value to 0	`x = 0 (?); y = 4`	`x = 0 (?); y = 4`
Resolve	`x = 0:H`	`x = 1:H`

Solution: Always rollback when actual value is secret

Implementation and Evaluation

Implementation

Prototype Risc-V implementation

- Firsts synthesizable implementation
- On top of Proteus modular RiSC-V processor
- Open-sourced on github!
- Limitation
  - Only branch prediction
  - Secrets not forwarded *at all* during speculation (conservative)

Evaluation: Labelling Secrets

Inform hardware about secrets?

Secret are labelled in source and co-located in binary
Boundaries stored in CSRs

- Currently supporting up to 2 separate regions
- Easy to change

Evaluation: is annotation easy?

Need to mark secret in source
Need avoid stack spilling!

	LoC	S	A_m	A_a	I	Description
`djbsort` [86]	246	`L`	3	0	6	Constant-time sort
`sha256` [59]	1795	`L`	34	0	6	Hash function
`chacha20` [59]	1864	`L`	51	0	6	Encryption
`curve25519` [59]	3026	`H`	9	67	0	Elliptic curve

Evaluation: Hardware

Hardware implementation

- Proteus is written in SpinalHDL
- ≈5000 lines of Scala code
- Changes for ProSpeCT: ≈ 400 lines

Hardware costs

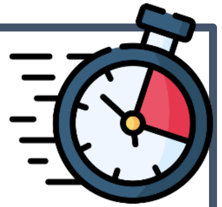
- LUTs: 16,847 → 19,728 (+17%)
- Registers: 11,913 → 12,600 (+6%)
- Critical path: 30.1 ns → 30.7 ns (+2%)

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	110%	125%	136%	145%



Conclusion

Results similar to [1]

Precise annotation + restricted secret computations = Low overhead

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

Conclusion

Check our paper on arXiv!

Hardware Secrecy Tracking

Software informs hardware about secret

Strong security guarantees

ProSpeCT $\Longrightarrow$ *end-to-end security for constant-time programs*

Low overhead

ProSpeCT $\Longrightarrow$ *no runtime overhead on public data*

Future Work?

Formal model

- Cryptographic security down to the hardware?

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 / Formal methods?

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