Hardware-Software Contracts for Secure Speculation

Jan Reineke @ Universität des Saarlandes

Joint work with
Marco Guarnieri, Pepe Vila @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK

Supported by the European Research Council and an Intel Strategic Research Alliance (ISRA)
The Need for HW/SW Contracts
ISA: Benefits

High-level language

Instruction set architecture (ISA)

Microarchitecture

Can program independently of microarchitecture

Can implement arbitrary optimizations as long as ISA semantics are obeyed
Inadequacy of the ISA: Side channels

High-level language → Instruction set architecture (ISA) → Microarchitecture

Impossible to program securely cryptographic algorithms?
sandboxing untrusted code?

No guarantees about side channels

Can implement arbitrary insecure optimizations as long as ISA is implemented correctly
**A Way Forward: HW/SW Security Contracts**

Can program **securely** on top of contract **independently** of microarchitecture

\[ \text{HW/SW contract} = \text{ISA} + X \]

Succinctly captures possible information leakage

Can implement **arbitrary insecure optimizations** as long as contract is obeyed
A Concrete Challenge: Spectre
Exploits speculative execution

Almost all modern CPUs are affected
Example: Spectre v1 Gadget

1. if (x < A.size)
2. y = A[x]
3. z = B[y*512]
4. end

x is out of bounds

Executed speculatively

Access “secret” A[x]

Transmit A[x] via data cache
Hardware Countermeasures

InvisiSpec: Making Speculative Execution Invisible in the Cache

Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adi Weller
University of Illinois at Urbana-Champaign
myan8, jchoi42, skarlat2@illinois.edu

NDA: Preventing Speculative Execution Attacks at Their Source

Kevin Loughlin
University of Michigan

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

Ofir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Stefanos Kaxiras
Uppsala University
stefanos.kaxiras@itolu.se

CleanupSpec: An “Undo” Approach to Safe Speculation

Gururaj Saileshwar
Georgia Institute of Technology
gururaj.s@gatech.edu

Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data

Moinuddin K. Qureshi
Georgia Institute of Technology
moin@gatech.edu

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Magnus Sjölander
Norwegian University of Science and Technology
wman

Artem Khyzha
Tel Aviv University
arkhyzha@mail.tau.ac.il

Mengjia Yan
University of Illinois at Urbana-Champaign
myan8@illinois.edu

Josep Torrellas
University of Illinois at Urbana-Champaign

Christopher W. Fletcher
University of Michigan
Examples

1. \texttt{if (x < A\_size)}
2. \texttt{y = A[x]}
3. \texttt{z = B[y*512]}
4. \texttt{end}

Delay loads until they can be retired
[Sakalis et al., ISCA’19]

Delay loads until they cannot be squashed
[Sakalis et al., ISCA’19]

Taint speculatively loaded data + delay tainted loads
[STT and NDA, MICRO’19]
Examples

1. $y = A[x]$
2. if ($x < A_{size}$) 
3. $z = B[y*512]$
4. end

Delay loads until they can be retired  
[Sakalis et al., ISCA'19]

Delay loads until they cannot be squashed  
[Sakalis et al., ISCA'19]

Taint speculatively loaded data + delay tainted loads  
[STT and NDA, MICRO’19]
What security properties do HW countermeasures enforce?

How can we program securely?
A Proof of Concept

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware–Software Contracts for Secure Speculation
S&P (Oakland) 2021
Hardware-Software Contracts
HW/SW Contracts for Secure Speculation

Secure Programming

Constant-time

Sandboxing

Desiderata:
- simple
- mechanism-independent
- precise
- at “ISA level”

HW/SW Contracts for Secure Speculation

Hardware Countermeasures

No speculation

Load Delay

Taint Tracking

No countermeasures
HW/SW Contracts for Secure Speculation

Contracts specify which program executions a side-channel adversary cannot distinguish

Contract traces: $\langle p, \sigma \rangle$

“What leaks” about an execution
Contracts

Contract
ISA + observations

Contract traces: \( (p, \sigma) \)

Hardware
Formal model of processor

Hardware traces: \( (p, \sigma) \)

Contract satisfaction

Hardware \( p \) satisfies contract \( \square \) if for all programs \( p \) and arch. states \( \sigma, \sigma' \):

if \( \square(p, \sigma) = \square(p, \sigma') \) then \( \square(p, \sigma) = \square(p, \sigma') \)
Contracts for Secure Speculation

\[
\text{Contract} = \text{Execution Mode} \cdot \text{Observer Mode}
\]

How are programs executed? What is visible about the execution?
Contracts for Secure Speculation

Contract =

Execution Mode · Observer Mode

seq — sequential execution

spec — mispredict branch instructions
Contracts for Secure Speculation

Contract = Execution Mode \cdot Observer Mode

pc — only program counter
ct — pc + addr. of loads and stores
arch — ct + loaded values
A Lattice of Contracts

Leaks “everything”

seq-arch

Leaks all data accessed non-speculatively

spec-arch

seq-ct

Leaks “nothing”

seq-ct+spec-pc

Leaks addresses of non-spec loads/stores/instruction fetches

spec-ct

Leaks addresses of all loads/stores/instruction fetches
Example: seq-ct

1. if \((x < A\_size)\)
2. \(y = A[x]\)
3. \(z = B[y]\)
4. end

Assume \(x < A\_size\)

1. \(pc = 2\)
2. load \(A+x\)
3. load \(B+A[x]\)
Example: seq-arch

1. if ($x < A_{\text{size}}$)
2. \( y = A[x] \)
3. \( z = B[y] \)
4. end

Assume $x < A_{\text{size}}$
Example: spec-ct

Assume $x > A_{\text{size}}$

1. if ($x < A_{\text{size}}$)
2. $y = A[x]$
3. $z = B[y]$
4. end
Hardware Countermeasures
A Simple Processor

3-stage pipeline
(fetch, execute, retire)

Speculative and out-of-order execution

Parametric in branch predictor and memory hierarchy

Different schedulers for different countermeasures
Disabling Speculative Execution

*Instructions* are executed *sequentially*: (fetch, execute, retire)*

No speculative leaks

Satisfies *seq-ct*
Eager Load Delay [Sakalis et al., ISCA’19]

Security guarantees?
Eager Load Delay [Sakalis et al., ISCA’19]

\[
\text{if } (x < A\text{\_size}) \\
\quad z = A[x] \\
\quad y = B[z]
\]

\[A[x] \text{ and } B[z] \text{ delayed until } x < A\text{\_size} \text{ is resolved}\]

🥳 No speculative leaks 😁
Eager Load Delay \cite{Sakalis et al., ISCA’19}

\[ z = A[x] \]
\[ \text{if } (x < A\_size) \]
\[ y = B[z] \]

**B[z]** delayed until \( x < A\_size \) is resolved

🥳 No speculative leaks 🎉
Eager Load Delay [*Sakalis et al., ISCA’19*]

\[
z = A[x] \\
\text{if } (x < A_{\text{size}}) \\
\quad \text{if } (z == 0) \\
\text{skip}
\]

*Observation*: Can only leak data accessed non-speculatively

- If \((z == 0)\) is not delayed
- Program speculatively leaks \(A[x]\) 😞
- Satisfies \textit{seq-arch}
- Satisfies \textit{seq-ct+spec-pc}
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

Security guarantees?

Taint speculatively loaded data

Delay tainted operations
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

if \( x < A_{\text{size}} \)
\[
\begin{align*}
  z &= A[x] \\
  y &= B[z]
\end{align*}
\]

\( A[x] \) tainted as *unsafe*

\( B[z] \) *delayed* until

\( A[x] \) is safe

![🥳](https://i.imgur.com/3Q5zQ5.png)

No specula leaks 😁
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

\[ z = A[x] \]
\[ \text{if } (x < A\text{size}) \]
\[ y = B[z] \]

\( A[x] \) tagged as \textit{safe} \n\( B[z] \) \textit{not delayed} \n
Program speculatively leaks \( A[x] \)

Also satisfies \texttt{seq-arch}
No Countermeasure [The World until 2018]

\[
\text{if } (x < A_{\text{size}}) \\
z = A[x] \\
y = B[z]
\]

Leaks addressed of speculative and non-speculative accesses

Satisfies spec-ct
Security Guarantees

- seq-arch
- spec-arch
- seq-ct
- seq-ct+spec-ct
- seq-ct+spec-pc
- spec-ct

- Load Delay
- Taint Tracking
- no speculation
- no countermeasure
Two Flavors of Secure Programming

Constant-time

Sandboxing
Two Flavors of Secure Programming

Constant-time

Sandboxing
Program $p$ is non-interferent wrt contract $[[\cdot]]$ and policy $\pi$ if for all arch. states $\sigma, \sigma'$: if $\sigma \approx_\pi \sigma'$ then $[[p]](\sigma) = [[p]](\sigma')$.

**Theorem**

If $p$ is non-interferent wrt contract $[[\cdot]]$ and policy $\pi$, and hardware $\{\cdot\}$ satisfies $[[\cdot]]$, then $p$ is non-interferent wrt hardware $\{\cdot\}$ and policy $\pi$. 
Sandboxing

Programs *never access* secret memory locations (out-of-sandbox)

**Traditional SB** wrt policy $\pi \equiv$ non-interference wrt *seq-arch* and $\pi$

**General SB** wrt $\pi$ and $[[\cdot]] \equiv$

Traditional SB wrt $\pi +$ non-interference wrt $\pi$ and $[[\cdot]]$
## Checking Sandboxing

<table>
<thead>
<tr>
<th>seq-ct</th>
<th>Traditional sandboxing (= non-interference wrt <strong>seq-arch</strong>)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>seq-arch</strong></td>
<td>Traditional sandboxing</td>
</tr>
<tr>
<td>spec-ct</td>
<td>... + weak SNI</td>
</tr>
</tbody>
</table>
Constant-time Programming

Control flow and memory accesses do not depend on secrets

**Traditional CT** wrt policy $\pi \equiv$ non-interference wrt seq-ct and $\pi$

**General CT** wrt $\pi$ and $[\cdot]$ $\equiv$ non-interference wrt $[\cdot]$ and $\pi$
# Checking Constant-time Programming

## General constant-time

<table>
<thead>
<tr>
<th>seq-ct</th>
<th>Traditional constant-time (= non-interference wrt seq-ct)</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-arch</td>
<td>Non-interference wrt seq-arch</td>
</tr>
<tr>
<td>spec-ct</td>
<td>... + Spec. non-interference</td>
</tr>
</tbody>
</table>

[Spectector, S&P’20]

No access to secrets!
Work in progress:
Contracts meet “the real world”
Contracts for Real ISAs + Real CPUs

- 3-stage pipeline
- Manual proof
- Toy ISA (6 instr.) + observer modes

Microarchitecture

- Register transfer level designs
- Automatic proof
- Real ISA + observer modes

Contract
Contracts for Real ISAs + Real CPUs

Microarchitecture

- Register transfer level designs
- Open-source RISC-V cores

Contract

- Automatic proof
- SMT solvers + Invariant inference
- Real ISA + observer modes
- Separate ISA from observer mode
Separating Observer from ISA satisfaction

**Contract satisfaction**
Hardware satisfies contract if for all programs $p$ and arch. states $\sigma$, $\sigma'$: if $\ hinges(p, \sigma) = hinges(p, \sigma')$ then $\ hinges(p, \sigma) = hinges(p, \sigma')$

**ISA satisfaction** + **Observer satisfaction**
Observer Inference

Given:
- Microarchitecture

Wanted:
- Weakest Observer

Observer Refinement

Trivial Observer

Microarchitecture

Candidate Observer

Observer Verification

Counterexample = Program + Pair of inputs

✓
Conclusions
Need to rethink **hardware-software contracts** with security in mind

*Find out more in our paper:*

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware–Software Contracts for Secure Speculation
S&P (Oakland) 2021