Hardware-Software Contracts for Safe and Secure Systems

Jan Reineke @ Università des Saarlandes

Joint work with
Marco Guarnieri, Pepe Vila @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK
Andreas Abel, Sebastian Hahn, Valentin Touzeau @ Saarland University

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

Applications implement data transformations:
  e.g. payroll processing

Hardware:
  • isolated, on-site
  • limited interaction with environment

IBM System 360/30

HW/SW Contract: Instruction Set Architecture
ISA Abstraction

High-level languages

Compiler

Instruction set architecture (ISA)

Implementation

Microarchitecture
ISA Abstraction: Benefits

Instruction set architecture (ISA)

Can program independently of microarchitecture

Can implement arbitrary optimizations as long as ISA semantics are obeyed
"Modern" (?) Computing

Applications are:

- **Data-driven**: e.g. deep neural networks
- **Distributed**: e.g. locally + in the cloud
- **Open**: e.g. untrusted code in the browser
- **Real-time**: interacting with the physical environment

What are the implications for HW/SW contracts?
Inadequacy of the ISA + current μArchitectures: Real-time Systems

Programs do not have a **timed semantics**
Programs have **no control** over timing

**Instruction set architecture (ISA)**

Can implement arbitrary **unpredictable** optimizations as long as ISA semantics are obeyed
State-of-the-art:
Handcrafted Microarchitectural Timing Models

Instruction set architecture (ISA) → Refinement → Microarchitectural timing model → Manual Modeling → Microarchitecture

Models are
- limited to particular microarchitectures
- probably incorrect
- yield expensive or imprecise analysis

models timing behavior
- still no control over timing

unpredictable
Wanted: Timed HW/SW Contracts

Programs have a **timed semantics**
Programs have **control** over timing

**Timed** Instruction Set Architecture

Admit **wide range** of high-performance microarchitectural implementations
Wanted: Timed HW/SW Contracts

Some answers:

S. Hahn and J. Reineke:
Design and Analysis of SIC:
A Provably Timing-Predictable Pipelined Processor Core
RTSS 2018
Inadequacy of the ISA + current μArchitectures: Side-channel security

**Impossible** to program securely on top of ISA cryptographic algorithms? sandboxing untrusted code?

*Instruction set architecture (ISA)*

No guarantees about side channels

Can implement arbitrary **insecure** optimizations as long as ISA semantics are obeyed
A Way Forward: HW/SW Security Contracts

Hardware-Software Contract = ISA + X

Can program securely on top contract independently of microarchitecture

Can implement arbitrary insecure optimizations as long as contract is obeyed

Succinctly captures possible information leakage
A Proof of Concept

M. Guarnieri, B. Köpf, J. Reineke, and P. Vila
Hardware–Software Contracts for Secure Speculation
S&P (Oakland) 2021
Almost all modern CPUs are affected.
Example: Spectre v1 Gadget

1. **x** is out of bounds
2. Executed speculatively
3. Leaks **A[x]** via data cache

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

InvisiSpec: Making Speculative Execution Invisible in the Cache

Mengjia Yan¹, Jiho Choi¹, Dimitrios Skarlatos, Adi Acar
University of Illinois at Urbana-Champaign
{myan8, jchoi42, skarlat2}@illinois.edu

Offir Weisse
University of Michigan

Thomas F. Wenisch
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

Kevin Loughlin
University of Michigan

NDA: Preventing Speculative Execution Attacks at Their Source

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

Stefanos Kaxiras
Uppsala University
stefanos.kaxiras@it.uu.se

Lazaros Sakalis
University of Uppsala

Magnus Själander
Norwegian University of Science and Technology

Trondheim, Norway

CleanupSpec: An “Undo” Approach to Safe Speculation

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

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

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

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

Josep Torrellas
University of Illinois at Urbana-Champaign

Christopher W. Fletcher
University of Illinois at Urbana-Champaign

Artem Khyzha
Tel Aviv University
artkhyzha@mail.tau.ac.il
Examples

1. if \( x < A_{size} \)
2. \( y = A[x] \)
3. \( z = B[y\times512] \)
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]
Examples

1. \( y = A[x] \)
2. if \( x < A_{\text{size}} \)
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?
Hardware-software contracts
Contracts for side-channel-free systems

Contracts specify which program executions a side-channel adversary can distinguish

Goals:
• Basis for secure programming
• Captures assumptions on microarchitectural implementations
• Capture security guarantees of HW in a simple, mechanism-independent manner
Ingredients

Instruction Set Architecture
Arch. states: $\sigma$
Arch. semantics: $\sigma \xrightarrow{\cdot} \sigma'$

Microarchitecture
Hardware states: $\langle \sigma, \mu \rangle$
Hardware semantics: $\langle \sigma, \mu \rangle \Rightarrow \langle \sigma', \mu' \rangle$

Adversary model
$\mu$Arch traces: $\{p\}(\sigma) = \mu_0\mu_1\ldots\mu_n$
**Contracts**

*Observations* expose security-relevant $\mu$Arch events

**Contract**

A deterministic, labelled semantics $\xrightarrow{\tau}$ for the ISA

Contract traces: $\llbracket p \rrbracket(\sigma) = \tau_1\tau_2\ldots\tau_n$

**Contract satisfaction**

Hardware $\{ \cdot \}$ satisfies contract $\llbracket \cdot \rrbracket$ if for all programs $p$ and arch. states $\sigma, \sigma'$: if $\llbracket p \rrbracket(\sigma) = \llbracket p \rrbracket(\sigma')$ then $\{ p \} (\sigma) = \{ p \} (\sigma')$
Contracts for secure speculation

Contract = Execution Mode • 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 · Observer Mode

\[ \text{pc} \quad \text{— only program counter} \]

\[ \text{ct} \quad \text{— pc} + \text{addr. of loads and stores} \]

\[ \text{arch} \quad \text{— ct} + \text{loaded values} \]
access-control checks, we make the following distinction: a program is declared high. To account for programs accessing controlled resources, we focus on one important aspect: the contract interface under the contract, i.e., no information about high memory locations leaks into the contract's traces. Observations, and means that we continue the computation. Speculation barriers trigger a rollback by setting for the correct branch, following by composition of Definitions 1 and 3: if for all initial arch. states the speculative window on fetches is reduced by 1 (rules S), where information about high memory locations is treated as non-interferent w.r.t. the contract. Configurations are stacks of speculative windows w.r.t. vanilla-sandboxed w.r.t. the contract. Proposition 3. General sandboxing means that a program is vanilla-sandboxed is equivalent to executing under the arch. semantics of the sandbox affects what a program exposing the value of accessed high memory locations.

Leaks all data accessed non-speculatively

Leaks “everything”

Leaks “nothing”

Leaks addresses of all loads/stores/instruction fetches

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

A lattice of contracts
Hardware countermeasures
A simple processor

Speculative and out-of-order execution

Formalized as operational semantics
⇒ describing how processor’s state changes
(inspired by [Cauligi et al., PLDI'20; Vassena et al., 2020])

Parametric in branch predictor and memory hierarchy
Disabling speculative execution

*Instructions* are executed *sequentially*

No speculative leaks

Satisfies *seq-ct* contract
Eager load delay \cite{Sakalis et al., ISCA’19}

Security guarantees?
Eager load delay \cite{Sakalis et al., ISCA’19}

\begin{verbatim}
if (x < A_size)  
z = A[x]  
y = B[z]
\end{verbatim}

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

🥳 No speculative leaks 😁
Eager load delay [Sakalis et al., ISCA’19]

\[
z = A[x]
\]

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

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

No speculative leaks 😊
Eager load delay [Sakalis et al., ISCA’19]

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

Observation: Can only leak data accessed non-speculatively

\[ \text{if } (z == 0) \text{ is not delayed} \]

Program speculatively leaks \[ A[x] \]

Satisfies \textit{seq-arch} contract
Hardware taint-tracking [Yu et al. 2019, Weisse et al. 2019]

- **Taint** speculatively loaded data
- **Delay** tainted operations

Security guarantees?
Hardware taint-tracking [Yu et al. 2019, Weisse et al. 2019]

```
if (x < A_size)
z = A[x]
y = B[z]
```

\[ A[x] \text{ tainted as } \textit{unsafe} \]
\[ B[z] \text{ delayed until } \]
\[ A[x] \text{ is safe} \]

🥳 No speculative leaks 😁
Hardware taint-tracking [Yu et al. 2019, Weisse et al. 2019]

\[ z = A[x] \]

if \( x < A\text{size} \)

\[ y = B[z] \]

\( A[x] \) tagged as *safe*

\( B[z] \) *not delayed*

Program speculatively

leaks \( A[x] \) 😞
No countermeasures [The World until 2018]

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

Leaks addressed of speculative and non-speculative accesses

Satisfies spec-ct contract
Security guarantees

\[ [\cdot\cdot]\text{seq}_{\text{arch}} \rightarrow [\cdot\cdot]\text{spec}_{\text{arch}} \]

\[ [\cdot\cdot]\text{seq}_{\text{arch}} \rightarrow [\cdot\cdot]\text{seq-spec}_{\text{ct-pc}} \]

\[ [\cdot\cdot]\text{seq-spec}_{\text{ct-pc}} \rightarrow [\cdot\cdot]\text{spec}_{\text{ct}} \]

\[ [\cdot\cdot]\text{spec}_{\text{ct}} \rightarrow [\cdot\cdot]\text{seq}_{\text{ct}} \]

\[ [\cdot\cdot]\text{seq}_{\text{ct}} \rightarrow [\cdot\cdot]\text{seq} \]

\[ [\cdot\cdot]\text{seq} \rightarrow \{\cdot\cdot\}\text{seq} \]

\[ \{\cdot\cdot\}\text{seq} \rightarrow \{\cdot\cdot\}\text{loadDelay} \]

\[ \{\cdot\cdot\}\text{loadDelay} \rightarrow \{\cdot\cdot\}\text{tt} \]

\[ \{\cdot\cdot\}\text{tt} \]

As shown in Example 3, Concretely, the array \( y \) beqz \( z \) is reduced by 1 (rules S + T - E).
Secure programming
Program $p$ is **non-interferent** wrt contract $\llbracket \cdot \rrbracket$ and policy $\pi$ if for all arch. states $\sigma, \sigma'$: if $\sigma \approx_\pi \sigma'$ then $\llbracket p \rrbracket(\sigma) = \llbracket p \rrbracket(\sigma')$.

If $p$ is **non-interferent** wrt contract $\llbracket \cdot \rrbracket$ and policy $\pi$, and hardware $\{ \cdot \}$ satisfies $\llbracket \cdot \rrbracket$, then $p$ is **non-interferent** wrt hardware $\{ \cdot \}$ and policy $\pi$. Specify secret data.
Two flavors of secure programming

Constant-time

Sandboxing
**Constant-time programming**

Control-flow and memory accesses do not depend on secrets

**Traditional CT wrt policy $\pi \equiv$ non-interference wrt $\textsf{seq}_{ct}$ and $\pi$**

**General CT wrt $\pi$ and $\textsf{seq}$ $\equiv$ non-interference wrt $\textsf{seq}$ and $\pi$**
Sandboxing

Programs never access high memory locations (out-of-sandbox)

**Traditional SB** wrt policy $\pi \equiv$ non-interference wrt $\llbracket \cdot \rrbracket_{\text{arch}}^{seq}$ and $\pi$

**General SB** wrt $\pi$ and $\llbracket \cdot \rrbracket$ $\equiv$

Traditional SB wrt $\pi$ + non-interference wrt $\pi$ and $\llbracket \cdot \rrbracket$
# Checking secure programming

<table>
<thead>
<tr>
<th>Constant-time</th>
</tr>
</thead>
</table>
| \[
\begin{array}{c}
\text{\llbracket \cdot \rrbracket}^{seq}_{ct} \\
\text{\llbracket \cdot \rrbracket}^{seq}_{arch} \\
\text{\llbracket \cdot \rrbracket}^{spec}_{ct}
\end{array}
\] |
| Traditional CT |
| \[
\text{NI wrt } \llbracket \cdot \rrbracket^{seq}_{arch}
\] |
| \[
\text{\ldots + Spec. Non-Interference [Spectector, S&P’20]}
\] |
Checking secure programming

**Sandboxing**

\[
\begin{array}{ccc}
\llbracket \cdot \rrbracket_{ct}^{seq} & \text{Traditional SB} & \text{Traditional SB} \\
\llbracket \cdot \rrbracket_{arch}^{seq} & \text{Traditional SB} & \text{... + weak SNI} \\
\llbracket \cdot \rrbracket_{ct}^{spec} & & \\
\end{array}
\]
Conclusions
Need to rethink **hardware-software contracts** with security and safety in mind!

Keep abstractions **as simple as possible**, but not more

*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