Hardware-Software Contracts for Safe and Secure Systems

Jan Reineke @ Universität 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 the European Research Council and an Intel Strategic Research Alliance (ISRA)
The Need for HW/SW Contracts
"Stone-age" Computing

Applications implemented data transformations:
e.g. payroll processing
"Stone-age" Computing

Applications implemented data transformations:
  e.g. payroll processing

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

IBM System 360/30

Author: ArnoldReinhold  License: CC BY-SA 3.0
"Stone-age" Computing

Applications implemented 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

Can program **independently** of microarchitecture

Instruction set architecture (ISA)

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
"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
Inadequacy of the ISA + current \( \mu \)Architectures: Real-time Systems

Instruction set architecture (ISA)

Can implement arbitrary \textcolor{red}{unpredictable} optimizations as long as ISA semantics are obeyed

Abstracts from time
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 → unpredictable

models timing behavior + still no control over timing
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
Wanted: Timed HW/SW Contracts

Timed Instruction Set Architecture

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

Programs have a **timed semantics** that is **efficiently predictable**
Programs have **control** over timing

**Timed** Instruction Set Architecture

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

Some answers:

D. Bui, E. Lee, I. Liu, H. Patel, and J. Reineke: Temporal Isolation on Multiprocessing Architectures  
DAC 2011

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

Instruction set architecture (ISA) No guarantees about side channels
Inadequacy of the ISA + current μArchitectures: Side-channel security

Instruction set architecture (ISA)

Can implement arbitrary **insecure** optimizations as long as ISA semantics are obeyed

No guarantees about side channels
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

Succinctly captures possible information leakage
**A Way Forward: HW/SW Security Contracts**

Hardware-Software Contract = ISA + X

Succinctly captures possible information leakage

Can implement *arbitrary insecure optimizations* as long as contract is obeyed
A Way Forward: HW/SW Security Contracts

Hardware-Software Contract = ISA + X

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

Succinctly captures possible information leakage

Can implement **arbitrary insecure optimizations** as long as contract is obeyed
A Concrete Challenge: Spectre
Almost all modern CPUs are affected

Exploits *speculative execution*

Example: Spectre v1 Gadget

1. if \((x < A\_size)\)
2. \(y = A[x]\)
3. \(z = B[y*512]\)
4. end
Example: Spectre v1 Gadget

1. \( x \) is out of bounds

```
1. if \( (x < A_{\text{size}}) \)
2. \( y = A[x] \)
3. \( z = B[y*512] \)
4. end
```
Example: Spectre v1 Gadget

1. x is out of bounds

1. if \( x < A\_size \)
2. \( y = A[x] \)
3. \( z = B[y\times512] \)
4. end

2. Executed speculatively
**Example: Spectre v1 Gadget**

1. \( x \) is out of bounds

2. Executed speculatively

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

3. Leaks \( A[x] \) via data cache
Hardware Countermeasures
Hardware Countermeasures

InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan¹, Jiho Choi¹, Dimitrios Skarlatos, Adam Morrison*, Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign  *Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu
Hardware Countermeasures

InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan¹, Jiho Choi¹, Dimitrios Skarlatos, Adam Morrison*, Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign  *Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletch, torrella}@illinois.edu

CleanupSpec: An “Undo” Approach to Safe Speculation

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

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

InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy

Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher W. Fletcher, and Josep Torrellas
University of Illinois at Urbana-Champaign  Tel Aviv University
{myan8, jchoi42, skarlat2}@illinois.edu, mad@cs.tau.ac.il, {cwfletcher, torrella}@illinois.edu

CleanupSpec: An “Undo” Approach to Safe Speculation

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

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

Christos Sakalis
Uppsala University
christos.sakalis@it.uu.se

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

Magnus Själander
Norwegian University of Science and Technology
Trondheim, Norway
magnus.sjalander@ntnu.no

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

Alexandra Jimborean
Uppsala University
alexia.jimborean@it.uu.se

Alberto Ros
University of Murcia
Murcia, Spain
aros@ditlec.um.es
Hardware Countermeasures

InvisiSpec: Making Speculative Execution Invisible in the Cache

Mengjia Yan¹, Jiho Choi¹, Dimitrios Skarlatos, Adria Swenson
University of Illinois at Urbana-Champaign
{myan8, jchoi42, skarlat2}@uiuc.edu

Kevin Loughlin
University of Michigan

Ofir Weisse
University of Michigan

Ian Neal
University of Michigan

Baris Kasikci
University of Michigan

NDA: Preventing Speculative Execution Attacks at Their Source

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

Thomas F. Wenisch
University of Michigan

Efficient Invisible Speculative Execution through Selective Delay and Value Prediction

Christos Sakalis
Uppsala University
Christos.Sakalis@it.uu.se

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

Alberto Ros
University of Murcia
aros@um.es

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

Norwegian University of Science and Technology
Trondheim, Norway
magnus.sjalander@ntnu.no

CleanupSpec: An “Undo” Approach to Safe Speculation

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

Alexandra Jimborean
Uppsala University
alexandra.jimborean@it.uu.se
InvisiSpec: Making Speculative Execution Attacks at Their Source

Invisible in the Cache

Hardware Countermeasures

Efficient Invisible Speculative Execution and Value Prediction

Selective Invisible Speculative Execution for Speculatively Accessed Data

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

NDA: Preventing Speculative Execution Attacks

"Undo" Approach to Safe Speculation

Speculative Access Prevention

CleanupSpec: An "Undo" Approach to Safe Speculation

Mengjie Yan, Jibo Chen, Dimitrios Skjoldan, Avr.
Jan N. Neide
Thomas F. Wenisch
Carl Seaman
Georgia Institute of Technology
University of Illinois at Urbana-Champaign
University of Michigan
Georgia Institute of Technology

InvisiSpec: Making Speculative Execution Attacks at Their Source

Kevin Langlin
University of Michigan

Christopher W. Fletcher

Armkhzya@nlluac.uk

Mengjie Yan, Jibo Chen, Dimitrios Skjoldan, Avr.
Jan N. Neide
Thomas F. Wenisch
Carl Seaman
Georgia Institute of Technology
University of Illinois at Urbana-Champaign
University of Michigan
Georgia Institute of Technology

Efficient Invisible Speculative Execution and Value Prediction

Selective Invisible Speculative Execution for Speculatively Accessed Data

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

NDA: Preventing Speculative Execution Attacks

"Undo" Approach to Safe Speculation

CleanupSpec: An "Undo" Approach to Safe Speculation

Mengjie Yan, Jibo Chen, Dimitrios Skjoldan, Avr.
Jan N. Neide
Thomas F. Wenisch
Carl Seaman
Georgia Institute of Technology
University of Illinois at Urbana-Champaign
University of Michigan
Georgia Institute of Technology
Examples

1. \texttt{if (x < A\_size)}
2. \texttt{y = A[x]}
3. \texttt{z = B[y\times512]}
4. \texttt{end}
Examples

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

1. if \( (x < A\_size) \)

2. \( y = A[x] \)

3. \( z = B[y*512] \)

4. end
Examples

1. if \((x < A\_\text{size})\)
2. \(y = A[x]\)
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]
Examples

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

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

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

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
HW/SW Contracts for Secure Speculation

- No countermeasures
- Load Delay
- Taint Tracking
- No speculation
HW/SW Contracts for Secure Speculation

Secure Programming

Constant-time

Sandboxing

HW/SW Contracts for Secure Speculation

Hardware Countermeasures

No speculation

Load Delay

Taint Tracking

No countermeasures
HW/SW Contracts for Secure Speculation

Secure Programming

Desiderata: simple, mechanism-independent, precise

Constant-time

Sandboxing

HW/SW Contracts for Secure Speculation

Hardware Countermeasures

No speculation

Load Delay

Taint Tracking

No countermeasures

22
Ingredients of a Formalization
Ingredients of a Formalization

Instruction Set Architecture
Arch. states: $\sigma$
Arch. semantics: $\sigma \rightsquigarrow \sigma'$
Ingredients of a Formalization

**Instruction Set Architecture**
Arch. states: $\sigma$
Arch. semantics: $\sigma \rightarrow \sigma'$

**Microarchitecture**
Hardware states: $\langle \sigma, \mu \rangle$
Hardware semantics: $\langle \sigma, \mu \rangle \Rightarrow \langle \sigma', \mu' \rangle$
Ingredients of a Formalization

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

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

Adversary model
$\mu$Arch traces: $\Xi p \Xi (\sigma) = \mu_0 \mu_1 \ldots \mu_n$
Contracts
A deterministic, labelled semantics $\tau$ for the ISA
Contracts

Observations expose security-relevant $\mu$Arch events

Contract

A deterministic, labelled semantics for the ISA
Contracts

Observations expose security-relevant $\mu$Arch events

Contract

A deterministic, labelled semantics $\tau$ for the ISA

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

*Observations* expose security-relevant \( \mu \text{Arch events} \)

**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 \( \llbracket p \rrbracket(\sigma) = \llbracket p \rrbracket(\sigma') \)
Contracts for Secure Speculation
Contracts for Secure Speculation

Contract =
Execution Mode · Observer Mode
Contracts for Secure Speculation

**Contract** = Execution Mode · Observer Mode

How are programs executed?
Contracts for Secure Speculation

Contract = Execution Mode · Observer Mode

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

\[ \text{Contract} = \text{Execution Mode} \cdot \text{Observer Mode} \]
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
Contracts for Secure Speculation

Contract =  
Execution Mode · Observer Mode

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

\[
\begin{align*}
\text{seq-arch} & \quad \quad \text{seq-ct} \\
\downarrow & \quad \quad \downarrow \\
\text{spec-arch} & \quad \quad \text{seq-ct+spec-pc} \\
\quad & \quad \quad \quad \quad \quad \quad \text{spec-ct} \\
\quad & \quad \quad \quad \quad \quad \quad \quad \text{⊤}
\end{align*}
\]
A Lattice of Contracts

seq-arch \(\downarrow\) spec-arch

seq-ct \(\uparrow\) seq-ct+spec-pc \(\uparrow\) spec-ct \(\uparrow\) \(\top\)

Leaks "everything"
A Lattice of Contracts

Leaks “everything”

Leaks “nothing”
A Lattice of Contracts

Leaks "everything"

Leaks "nothing"

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

Leaks "everything"

Leaks all data accessed non-speculatively

seq-arch

seq-ct

seq-ct+spec-pc

spec-ct

spec-arch

Leaks "nothing"

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

Leaks "everything"
A Lattice of Contracts

Leaks "everything"

Leaks "nothing"

Leaks addresses of non-spec. loads/stores/instructions on fetches

Leaks addresses of all loads/stores/instructions on fetches

Leaks all data accessed non-speculatively
Hardware Countermeasures
A Simple Processor
A Simple Processor

3-stage pipeline
(fetch, execute, retire)
A Simple Processor

3-stage pipeline
(fetch, execute, retire)

Speculative and out-of-order execution
A Simple Processor

3-stage pipeline
(fetch, execute, retire)

Speculative and out-of-order execution

Parametric in branch predictor and memory hierarchy
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
Disabling Speculative Execution

Instructions are executed **sequentially**: (fetch, execute, retire)\(^*\)
Disabling Speculative Execution

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

No speculative leaks 😞
Disabling Speculative Execution

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

No speculative leaks

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

Delaying loads until all sources of speculation are resolved
Eager Load Delay \([\text{Sakalis et al., ISCA’19}]\)

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

if \( x < A\_size \)
  \( z = A[x] \)
  \( y = B[z] \)
Eager Load Delay \textit{[Sakalis et al., ISCA’19]}

\begin{verbatim}
if (x < A_size)
  z = A[x]
  y = B[z]
\end{verbatim}
Eager Load Delay [Sakalis et al., ISCA’19]

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

\(A[x]\) and \(B[z]\) delayed until \(x < A_{\text{size}}\) is resolved
Eager Load Delay \cite{Sakalis2019}

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

\begin{itemize}
\item $A[x]$ and $B[z]$ delayed until $x < A_{size}$ is resolved
\end{itemize}

🥳 No speculative leaks 🥳
Eager Load Delay \[\text{[Sakalis et al., ISCA’19]}\]

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

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

No speculative leaks 😂
Eager Load Delay \[\text{[Sakalis et al., ISCA'19]}\]

\[z = A[x]\]
\[
\text{if } (x < A_{\text{size}})
\]
\[
\text{if } (z == 0)
\]
\[
\text{skip}
\]
Eager Load Delay [Sakalis et al., ISCA’19]

\[ z = A[x] \]
\[ \text{if } (x < A\_size) \]
\[ \text{if } (z == 0) \]
\[ \text{skip} \]
Eager Load Delay [Sakalis et al., ISCA’19]

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

\[ \text{if} \ (z == 0) \text{ is not delayed} \]
Eager Load Delay [Sakalis et al., ISCA’19]

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

\* if \( (z==0) \) is *not* delayed

Program speculatively

leaks \( A[x] \) 😞
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

if \((z == 0)\) is not delayed

Program speculatively leaks \(A[x]\) 😞
Eager Load Delay \cite{Sakalis et al., ISCA’19}

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

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

Program speculatively leaks \( A[x] \)

Observation: Can only leak data accessed non-speculatively

Satisfies seq-arch

Satisfies seq-ct+spec-pc
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

*Taint* speculatively loaded data
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

- Propagate taint through computation
- Taint speculatively loaded data
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

- **Taint** speculatively loaded data
- **Propagate taint** through computation
- **Delay** tainted operations
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

Taint speculatively loaded data

Security guarantees?

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

\[
\text{if} \ (x < A\_size) \\
\quad z = A[x] \\
\quad y = B[z]
\]
Taint Tracking \cite{Yu_2019, Weisse_2019}

\[
\text{if } (x < A\_size) \\
z = A[x] \\
y = B[z]
\]
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

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

\(A[x]\) tainted as \textit{unsafe} \\
\(B[z]\) \textit{delayed} until \\
\(A[x]\) is safe
Taint Tracking \cite{yu2019taint, weisse2019taint}

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

\begin{itemize}
  \item \(A[x]\) tainted as \textit{unsafe}
  \item \(B[z]\) \textit{delayed} until
  \item \(A[x]\) is safe
\end{itemize}

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

\[
z = A[x]
\]

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

\[
y = B[z]
\]
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

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

\( B[z] \) \textit{not delayed}
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

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

- \(A[x]\) tagged as safe
- \(B[z]\) not delayed

Program speculatively leaks \(A[x]\) 😞
Taint Tracking [Yu et al. 2019, Weisse et al. 2019]

$$z = A[x]$$

if ($$x < A\_size$$)

$$y = B[z]$$

A[$$x$$] tagged as *safe*

B[$$z$$] *not delayed*

Program speculatively leaks A[$$x$$] 😞

Also satisfies seq-arch
No Countermeasures \textit{[The World until 2018]}

\begin{align*}
\text{if } (x < A_{\text{size}}) \\
& \quad z = A[x] \\
& \quad y = B[z]
\end{align*}
No Countermeasures *[The World until 2018]*

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

Leaks addressed of speculative and non-speculative accesses
No Countermeasures \textit{[The World until 2018]}

Leaks addressed of speculative and non-speculative accesses

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

Satisfies \texttt{spec-ct}
Security Guarantees

seq-arch

spec-arch

seq-ct

seq-ct+spec-pc

spec-ct
Security Guarantees

- seq-arch
- spec-arch

seq-ct

seq-ct+spec-pc

spec-ct

no speculation
Security Guarantees

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

Load Delay

no speculation
Security Guarantees

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

- no speculation
- Load Delay
- Taint Tracking
Security Guarantees

seq-arch

spec-arch

seq-ct

seq-ct+spec-pc

spec-ct

Load Delay

Taint Tracking

no speculation

no countermeasure
Secure Programming
Secure Programming: Foundations
Program $p$ is non-interferent wrt contract $[[\cdot]]$ and policy $\pi$ if for all arch. states $\sigma, \sigma'$: if $\sigma \simeq_\pi \sigma'$ then $[[p]](\sigma) = [[p]](\sigma')$
Secure Programming: Foundations

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 $\llbracket p \rrbracket(\sigma) = \llbracket p \rrbracket(\sigma')$

Specify secret data
Secure Programming: Foundations

Specify secret data

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 $\llbracket p \rrbracket(\sigma) = \llbracket p \rrbracket(\sigma')$
Program $p$ is non-interferent wrt contract $[[\cdot]]$ and policy $\pi$ if for all arch. states $\sigma$, $\sigma'$: if $\sigma \approx_\pi \sigma'$ then $[[p]](\sigma) = [[p]](\sigma')$
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 $\pi$
Two Flavors of Secure Programming

Constant-time

Sandboxing
Two Flavors of Secure Programming

Constant-time

Sandboxing
Two Flavors of Secure Programming

Constant-time

Sandboxing
Constant-time Programming
Constant-time Programming

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

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

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

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

Programs never access high 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 Secure Programming

<table>
<thead>
<tr>
<th></th>
<th>Constant-time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>seq-ct</strong></td>
<td>Traditional constant-time (= non-interference wrt <strong>seq-ct</strong>)</td>
</tr>
<tr>
<td><strong>seq-arch</strong></td>
<td>Non-interference wrt <strong>seq-arch</strong></td>
</tr>
<tr>
<td><strong>spec-ct</strong></td>
<td>… + Spec. non-interference [Spectector, S&amp;P’20]</td>
</tr>
</tbody>
</table>
Checking Secure Programming

<table>
<thead>
<tr>
<th></th>
<th>Constant-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-ct</td>
<td>Traditional constant-time ((=) non-interference wrt seq-ct)</td>
</tr>
<tr>
<td>seq-arch</td>
<td>Non-interference wrt seq-arch</td>
</tr>
<tr>
<td>spec-ct</td>
<td>... + Spec. non-interference [Spectector, S&amp;P’20]</td>
</tr>
</tbody>
</table>
## Checking Secure Programming

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

| seq-ct     | Traditional sandboxing  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(= non-interference wrt seq-arch)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>seq-arch</th>
<th>Traditional sandboxing</th>
</tr>
</thead>
</table>

| spec-ct    | ... + weak SNI          |
Conclusions
Need to rethink **hardware-software contracts**
with security and safety in mind!
Need to rethink **hardware-software contracts** with security and safety in mind!

Should strive for **simple** and **mechanism-independent** contracts.
Need to rethink **hardware-software contracts** with security and safety in mind!

Should strive for **simple** and **mechanism-independent** contracts.

*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