Compositional Secure Compilation against Spectre

Marco Patrignani[1,2]

[1] CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Compositional Secure

Cor

Special thanks to

CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
    - Based on recent secure compilation theory

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
   - Based on recent secure compilation theory
   - Preservation of SNI: semantic notion of security

     Guarnieri *et al.* S&P'19

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
    - Based on recent secure compilation theory
    - Preservation of SNI: semantic notion of security Guarnieri *et al.* S&P'19
    - Source semantics is “standard”

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
   - Based on recent secure compilation theory
   - Preservation of SNI: semantic notion of security Guarnieri *et al.* S&P'19
   - Source semantics is "standard"
   - Target semantics is speculative
2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
   - Based on recent secure compilation theory
   - Preservation of SNI: semantic notion of security

     Guarnieri *et al.* S&P'19
   - Source semantics is "standard"
   - Target semantics is speculative
2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - Secure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
   - Based on recent secure compilation theory
   - Preservation of SNI: semantic notion of security

     Guarnieri *et al.* S&P'19
   - Source semantics is "standard"
   - (1) rget semantics is speculative
2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - Secure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

1. Formal framework for assessing security of Spectre v1 compiler countermeasures
   - Based on recent secure compilation theory
   - (2) Preservation of SNI: semantic notion of security

     Guarnieri *et al.* S&P'19
   - Source semantics is "standard"
   - (1) Target semantics is speculative
2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - Secure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

(3) rmal framework for assessing security of Spectre v1 compiler countermeasures

- Based on recent secure compilation theory
- (2) eservation of SNI: semantic notion of security Guarnieri *et al.* S&P’19
- Source semantics is “standard”
- (1) rget semantics is speculative

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - Secure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

(3) rmal framework for assessing security of Spectre v1 compiler countermeasures

- Based on recent secure compilation theory
- (2) eservation of SNI: semantic notion of security Guarnieri *et al.* S&P'19
- Source semantics is "standard"
- (1) rget semantics is speculative

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - (4) cure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

(3) rmal framework for assessing security of

(5) What about preserving multiple variants?

19

- Source semantics is "standard"
- (1) rget semantics is speculative

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures
   - (4) cure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

(3) rmal framework for assessing security of

(5) What about preserving multiple variants?

Composition (wip)

19

- Source semantics is "standard"

(1) rget semantics is speculative

2. Proofs of security and insecurity of existing Spectre v1 compiler countermeasures

(4) cure: Intel ICC, SLH(ish), SSLH

- Insecure: MSVC, Interprocedural SLH

Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

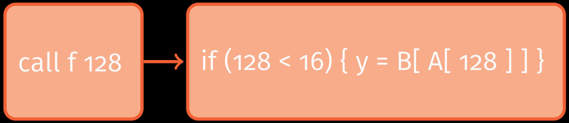
run 1: A.size = 16, A[128] = 3

call f 128

Speculative Semantics & SNI

$\textbf{void f (int x)} \mapsto \textbf{if}(\mathbf{x} < \mathbf{A}.\mathbf{size})\ \{\mathbf{y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$

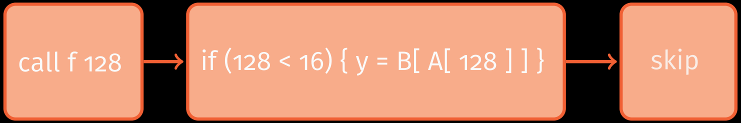
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

void f (int x) ↦ if(x < A.size) {y = B[A[x]]}

run 1: A.size = 16, A[128] = 3



call f 128

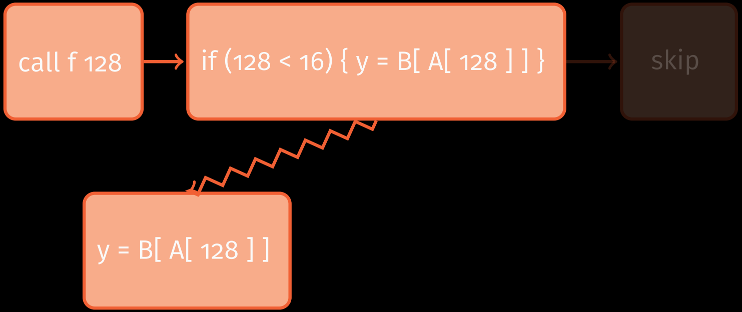
if (128 < 16) { y = B[A[128]] }

skip

Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

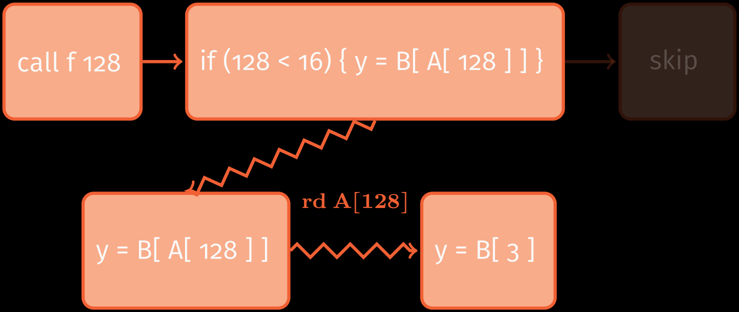
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

void f (int x) $\mapsto$ **if(x < A.size) {y = B[A[x]]}**

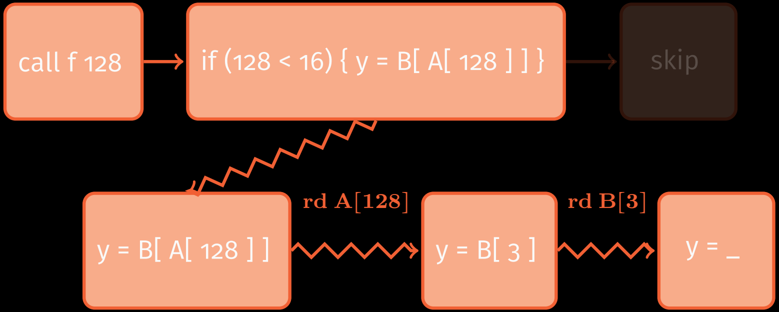
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

void f (int x) ↦ if(x < A.size) {y = B[A[x]]}

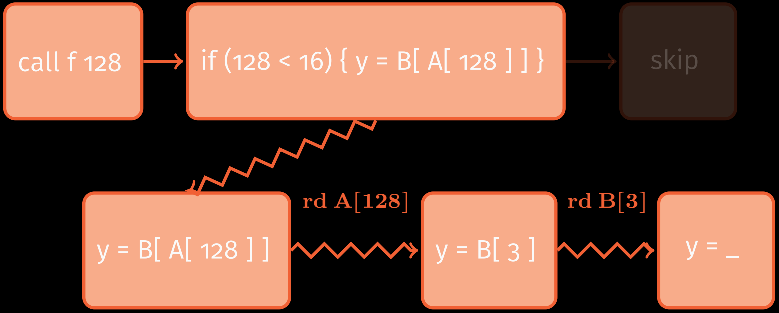
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

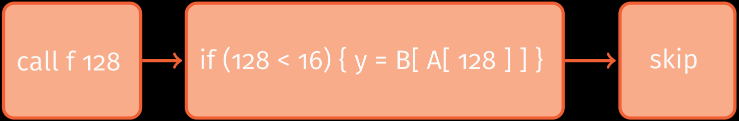
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3



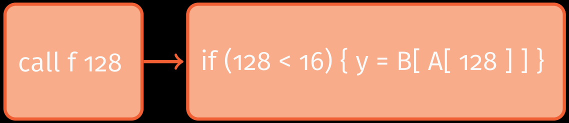
trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

Speculative Semantics & SNI

void f (int x) ↦ if(x < A.size) {y = B[A[x]]}

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



call f 128

if (128 < 16) { y = B[A[128]] }

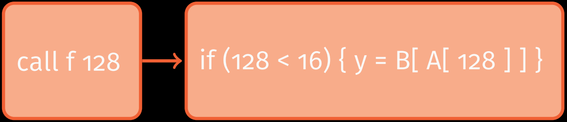
trace 1: **rd A[128]** **rd B[3]**

Speculative Semantics & SNI

void f (int x) ↦ if(x < A.size) {y = B[A[x]]}

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: rd A[128] rd B[3]

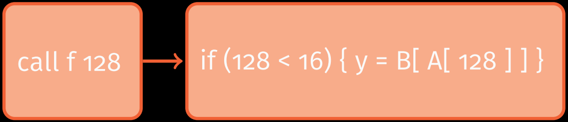
rd A[128]

Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



call f 128

if (128 < 16) { y = B[A[128]] }

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

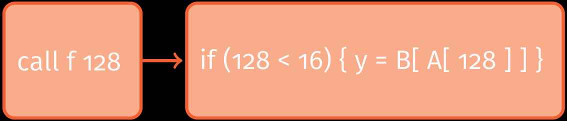
$\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$

Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$ different traces

trace 2: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$ $\Rightarrow$ SNI violation

Speculative Semantics & SNI

A program is SNI ($\vdash \mathbf{P} : \mathbf{SNI}$) if, given two runs from low-equivalent states:

- - if the non-speculative traces are low-equivalent
- - then the speculative traces are also low-equivalent

call f

trace 1: $\mathbf{rd}\ \mathbf{A}[128]$ $\mathbf{rd}\ \mathbf{B}[3]$ different traces

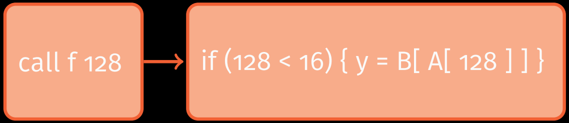
trace 2: $\mathbf{rd}\ \mathbf{A}[128]$ $\mathbf{rd}\ \mathbf{B}[7]$ ⇒ SNI violation

Speculative Semantics & SNI

void f (int x) ↦ **if**(**x** < **A**.**size**) {**y** = **B**[**A**[**x**]]}

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: **rd A**[**128**] **rd B**[**3**] different traces

trace 2: **rd A**[**128**] **rd B**[**7**] ⇒ SNI violation

Speculative Semantics & SNI

A program attains SNI robustly ($\vdash \mathbf{P} : \mathbf{RSNI}$) if it is $\mathbf{SNI}$ no matter what attacker $\mathbf{A}$ it links against.

$$\forall \mathbf{A}. \vdash \mathbf{A}\left[\mathbf{P}\right] : \mathbf{SNI}$$

call f

trace 1: $\mathbf{rd}\ \mathbf{A}[128]$ $\mathbf{rd}\ \mathbf{B}[3]$ different traces

trace 2: $\mathbf{rd}\ \mathbf{A}[128]$ $\mathbf{rd}\ \mathbf{B}[7]$ ⇒ SNI violation

Problems Problems Problems …

Problem: Proving compiler preserves RSNI is hard

Problems Problems Problems . . .

Problem: Proving compiler preserves RSNI is hard

Solution: overapproximate RSNI with a novel property: robust speculative safety (RSS)

Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

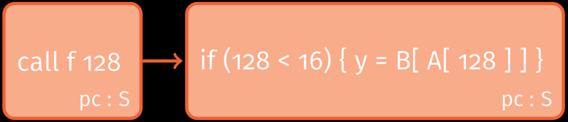
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



call f 128

pc : S

if (128 < 16) { y = B[A[128]] }

pc : S

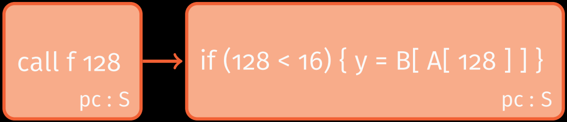
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



call f 128

pc : S

if (128 < 16) { y = B[A[128]] }

pc : S

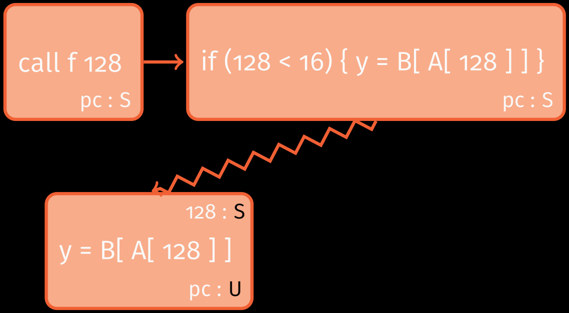
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



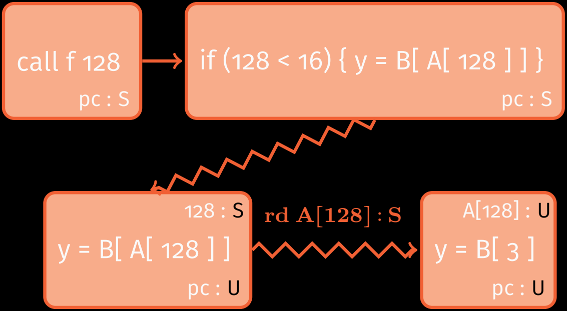
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



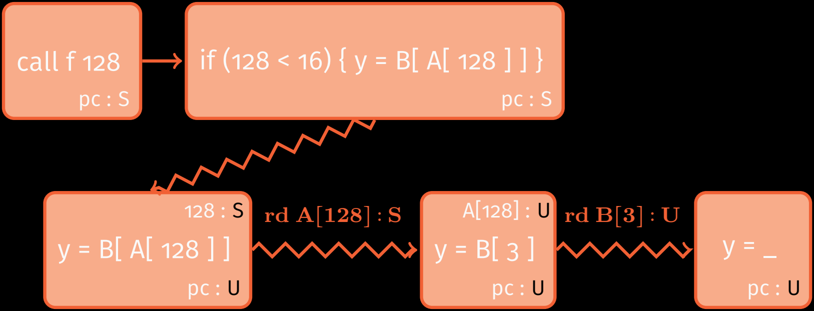
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



call f 128 pc : S

if (128 < 16) { y = B[A[128]] } pc : S

128 : S y = B[A[128]] pc : U

rd A[128] : S

A[128] : U y = B[3] pc : U

rd B[3] : U

y = _ pc : U

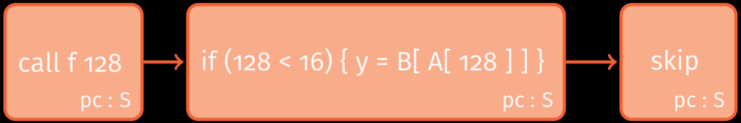
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



$\mathbf{rd\ A[128] : S}$ $\mathbf{rd\ B[3] : U}$

Speculative Safety (RSS)

Sema

A program is SS ($\vdash \mathbf{P} : \mathbf{SS}$) if its traces do not contain $\mathbf{U}$ actions

A program is SS robustly ($\vdash \mathbf{P} : \mathbf{RSS}$) if it is $\mathbf{SS}$ no matter what attacker $\mathbf{A}$ it links against.

call f 128

pc : S

if (128 < 16) { y = B[A[128]] }

pc : S

$\mathbf{rd\ A[128] : S}$ $\mathbf{rd\ B[3] : U}$

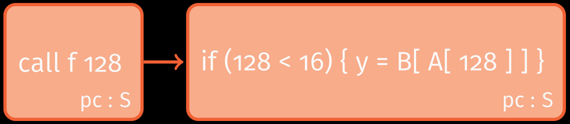
Speculative Safety (RSS)

Semantic-Irrelevant Taint Tracking

$\mathbf{void\ f\ (int\ x)} \mapsto \mathbf{if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U$ $S \sqcap U = S$ U does not flow to S



$\mathbf{rd\ A[128] : S}$ $\mathbf{rd\ B[3] : U}$

Robustness & Preserving NI-like Props

1. preserve for whole programs (e.g., CCT), cube-like proofs

Robustness & Preserving NI-like Props

1. preserve for whole programs (e.g., CCT), cube-like proofs
2. preserve safety overapproximation, for partial programs and robustly

Robustness & Preserving NI-like Props

1. preserve for whole programs (e.g., CCT), cube-like proofs
2. preserve safety overapproximation, for partial programs and robustly

Robustness pros and cons:

- ✓ realistic, (not) lossy, precise attacker + actions awareness
- ✗ coqability, precision, sometimes inefficient

RSS and RSNI

RSS overapproximates RSNI, so:

- in the $\mathbf{target}$: $\forall \mathbf{P}. \vdash \mathbf{P} : \mathbf{RSS} \Rightarrow \vdash \mathbf{P} : \mathbf{RSNI}$

RSS and RSNI

RSS overapproximates RSNI, so:

- in the $\mathbf{target}$: $\forall \mathbf{P}. \vdash \mathbf{P} : \mathbf{RSS} \Rightarrow \vdash \mathbf{P} : \mathbf{RSNI}$
- in the source: $\forall \mathsf{P}. \vdash \mathsf{P} : \mathsf{RSS} \Longleftrightarrow \vdash \mathsf{P} : \mathsf{RSNI}$ (recall, no speculative execution in source)

RSS-Preserving Compiler: RSSC & RSSP

$$\llbracket\cdot\rrbracket : \mathsf{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\ \vdash \mathsf{A}\left[\mathsf{P}\right] : \mathsf{RSS} \text{ and } \mathsf{RSS} \sim \mathbf{RSS}$$

$$\text{then } \forall \mathbf{A}.\ \vdash \mathbf{A}\left[\llbracket\mathsf{P}\rrbracket\right] : \mathbf{RSS}$$

RSS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \mathsf{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\ \vdash \mathsf{A}\left[\mathsf{P}\right] : \mathsf{RSS} \text{ and } \mathsf{RSS} \sim \mathbf{RSS}$$

$$\text{then } \forall \mathbf{A}.\ \vdash \mathbf{A}\left[[\![\mathsf{P}]\!]\right] : \mathbf{RSS}$$

$$[\![\cdot]\!] : \mathsf{RSSC} \overset{\text{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \rightsquigarrow \mathsf{m} \approx \mathbf{m}$$

$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

RSS-Preserving Compiler: RSSC **&** RSSP

$$[\![\cdot]\!] : \mathrm{RSSP} \overset{\mathrm{def}}{=} \text{ if } \forall \mathrm{A}.\ \vdash \mathrm{A}\left[\mathrm{P}\right] : \mathrm{RSS} \text{ and } \mathrm{RSS} \sim \mathbf{RSS}$$

$$\text{then } \forall \mathbf{A}.\ \vdash \mathbf{A}\left[[\![\mathrm{P}]\!]\right] : \mathbf{RSS}$$

$$[\![\cdot]\!] : \mathrm{RSSC} \overset{\mathrm{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}\left[[\![\mathrm{P}]\!]\right] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathrm{A}.\mathrm{A}\left[\mathrm{P}\right] \rightsquigarrow \mathrm{m} \approx \mathbf{m}$$

$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- $\forall A$: explicit attacker model (robustness)

RSS-Preserving Compiler: RSSC **&** RSSP

$$[\![\cdot]\!] : \mathsf{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\vdash \mathsf{A}\left[\mathsf{P}\right] : \mathsf{RSS} \text{ and } \mathsf{RSS} \sim \mathbf{RSS}$$

$$\text{then } \forall \mathbf{A}.\vdash \mathbf{A}\left[[\![\mathsf{P}]\!]\right] : \mathbf{RSS}$$

$$[\![\cdot]\!] : \mathsf{RSSC} \overset{\text{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}\left[[\![\mathsf{P}]\!]\right] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathsf{A}.\mathsf{A}\left[\mathsf{P}\right] \rightsquigarrow \mathsf{m} \approx \mathbf{m}$$

$\approx$= same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- $\forall A$: explicit attacker model (robustness)
- Proof: RSSC & RSSP are equivalent

  RSSC : clear security guarantees

  RSSP : simpler proofs

RSS-Preserving Compiler: RSSC **&** RSSP

$$[\![\cdot]\!] : \mathsf{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\ \vdash \mathsf{A}\,[\mathsf{P}] : \mathsf{RSS} \text{ and } \mathsf{RSS} \sim \mathbf{RSS}$$

$$\text{then } \forall \mathbf{A}.\ \vdash \mathbf{A}\,[[\![\mathsf{P}]\!]] : \mathbf{RSS}$$

$$[\![\cdot]\!] : \mathsf{RSSC} \overset{\text{def}}{=} \text{ if } \forall \mathbf{A}.\mathbf{A}\,[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m} \text{ then } \exists \mathsf{A}.\mathsf{A}\,[\mathsf{P}] \rightsquigarrow \mathsf{m} \approx \mathbf{m}$$

$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- $\forall A$: explicit attacker model (robustness)

Danger

of: RSSC & RSSP are equivalent

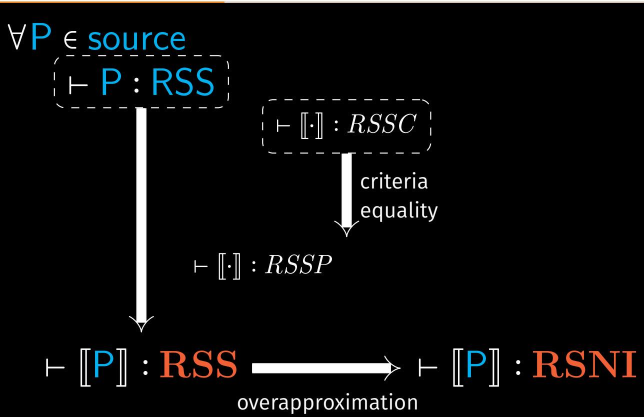
RSSC : clear security guarantees

RSSP : simpler proofs

Secure Compilation Framework for Spectre

$$\begin{tikzcd}
\forall P \in \text{source} & & \\
\boxed{\vdash P : \mathrm{RSS}} \arrow[dd] & \boxed{\vdash [\![\cdot]\!] : RSSC} \arrow[d, "\text{criteria equality}"] & \\
& \vdash [\![\cdot]\!] : RSSP & \\
\vdash [\![P]\!] : \mathbf{RSS} \arrow[rr, "\text{overapproximation}"'] & & \vdash [\![P]\!] : \mathbf{RSNI}
\end{tikzcd}$$

Secure Compilation Framework for Spectre



$\forall P \in$ source

$\vdash P : \mathsf{RSS}$

$\vdash [\![\cdot]\!] : RSSC$

criteria equality

$\vdash [\![\cdot]\!] : RSSP$

$\vdash [\![P]\!] : \mathbf{RSS} \longrightarrow \vdash [\![P]\!] : \mathbf{RSNI}$

overapproximation

- all source programs are trivially RSS

Secure Compilation Framework for Spectre

$$\begin{tikzcd}
\forall \mathsf{P} \in \mathsf{source} \\
\vdash \mathsf{P} : \mathsf{RSS} \arrow[ddd] & \boxed{\vdash [\![\cdot]\!] : \mathit{RSSC}} \arrow[d, "\text{criteria equality}"] & \\
 & \, & \\
 & \vdash [\![\cdot]\!] : \mathit{RSSP} & \\
\vdash [\![\mathsf{P}]\!] : \mathbf{RSS} \arrow[rr, "\text{overapproximation}"'] & & \vdash [\![\mathsf{P}]\!] : \mathbf{RSNI}
\end{tikzcd}$$

- all source programs are trivially RSS
- to show security: simply prove RSSC

Preservation or Enforcement?

Is it still preservation if the property is trivial in source?

Preservation or Enforcement?

Is it still preservation if the property is trivial in source?

Preservation can generalise the proof

Preservation or Enforcement?

Is it still preservation if the property is trivial in source?

Preservation can generalise the proof

Enforcement cannot work for classes (more on this later)

Security Spectrum

- 2 notions of RSS and RSNI (thus 2 $\mathbf{target}$s):
    - strong(+): no speculative leaks

Security Spectrum

- 2 notions of RSS and RSNI (thus 2 $\mathbf{target}$s):
    - strong(+): no speculative leaks
    - weak(-): allows speculative leaks of data retrieved non-speculatively

Security Spectrum

- 2 notions of RSS and RSNI (thus 2 targets):
  - strong(+): no speculative leaks
  - weak(-): allows speculative leaks of data retrieved non-speculatively

```
void get (int y)
  if (y < size) then
    temp = B[A[y]*512]
```

Violates + and -

```
void get (int y)
  x = A[y];
  if (y < size) then
    temp = B[x];
```

Violates +, Satisfies -

RSSC **for** `lfence`

$\mathsf{void\ f(int\ x)} \mapsto \mathsf{if}(\mathsf{x} < \mathsf{A.size})\{\mathsf{y} = \mathsf{B}[\mathsf{A}[\mathsf{x}]]\}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$

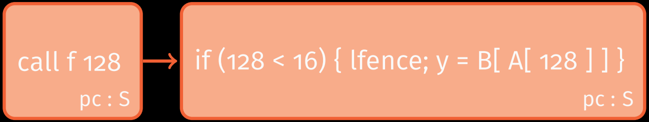
call f 128

pc : S

RSSC **for** `lfence`

$\mathsf{void\ f(int\ x)} \mapsto \mathsf{if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$$



call f 128

pc : S

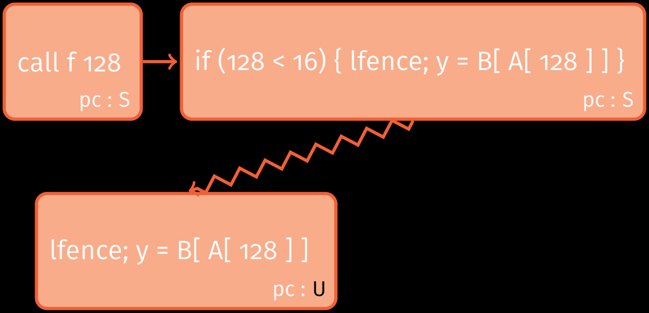
if (128 < 16) { lfence; y = B[A[128]] }

pc : S

RSSC **for** `lfence`

$\text{void f(int x)} \mapsto \text{if}(x < \text{A.size})\{y = B[A[x]]\}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\mathbf{x} < \textbf{A.size})\{\textbf{lfence; y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



call f 128

pc : S

if (128 < 16) { lfence; y = B[A[128]] }

pc : S

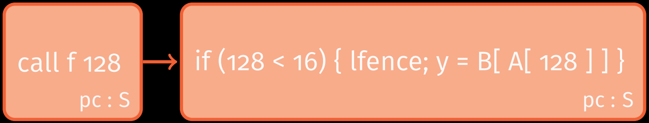
lfence; y = B[A[128]]

pc : U

RSSC **for** `lfence`

$\mathsf{void\ f(int\ x)} \mapsto \mathsf{if}(\mathsf{x} < \mathsf{A.size})\{\mathsf{y} = \mathsf{B}[\mathsf{A}[\mathsf{x}]]\}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if}(\mathbf{x} < \mathbf{A.size})\{\mathbf{lfence; y} = \mathbf{B}[\mathbf{A}[\mathbf{x}]]\}$



call f 128

pc : S

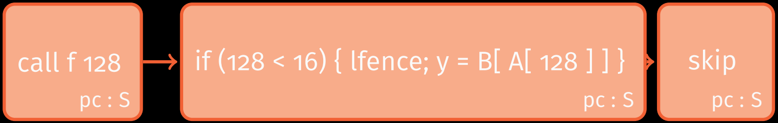
if (128 < 16) { lfence; y = B[A[128]] }

pc : S

RSSC **for** `lfence`

$\mathsf{void\ f(int\ x)} \mapsto \mathsf{if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$



RSSC **for** SLH

$\mathsf{void\ f(int\ x)} \mapsto \mathsf{if(x < A.size)\{y = B[A[x]]\}}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{y = B[mask(A[x])]\}}$

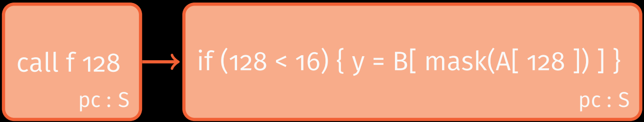
call f 128

pc : S

RSSC for SLH

$\text{void f(int x)} \mapsto \text{if}(x < \text{A.size})\{y = \text{B}[\text{A}[x]]\}$ // A.size=16, A[128]=3

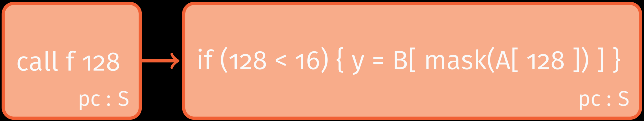
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\textbf{x} < \textbf{A.size})\{\textbf{y} = \textbf{B}[\textbf{mask}(\textbf{A}[\textbf{x}])]\}$



RSSC **for** SLH

void f(int x) $\mapsto$ if(x < A.size){y = B[A[x]]}   // A.size=16, A[128]=3

$[\![\cdot]\!]$ = **void f(int x)** $\mapsto$ **if(x < A.size){y = B[mask(A[x])]}**



call f 128

pc : S

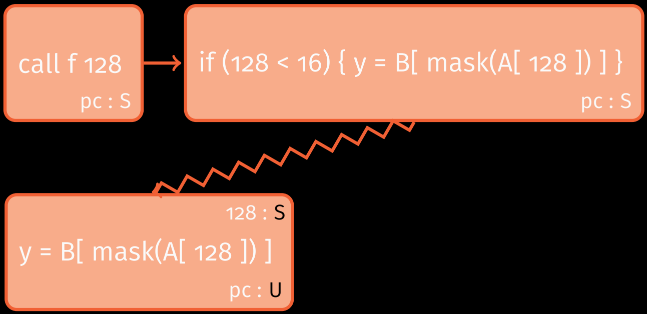
if (128 < 16) { y = B[mask(A[128])] }

pc : S

RSSC **for** SLH

$\text{void f(int x)} \mapsto \text{if}(x < \text{A.size})\{y = \text{B}[\text{A}[x]]\}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\textbf{x} < \textbf{A.size})\{\textbf{y} = \textbf{B}[\textbf{mask}(\textbf{A}[\textbf{x}])]\}$



call f 128

pc : S

if (128 < 16) { y = B[mask(A[128])] }

pc : S

128 : S

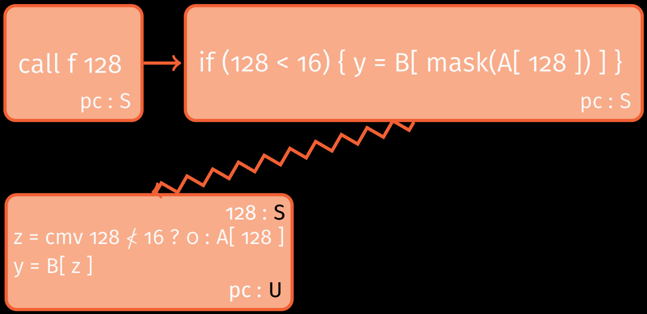
y = B[mask(A[128])]

pc : U

RSSC **for** SLH

$\text{void f(int x)} \mapsto \text{if}(x < \text{A.size})\{y = \text{B}[\text{A}[x]]\}$ // A.size=16, A[128]=3

$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\textbf{x} < \textbf{A.size})\{\textbf{y} = \textbf{B}[\textbf{mask}(\textbf{A}[\textbf{x}])]\}$



call f 128

pc : S

if (128 < 16) { y = B[mask(A[128])] }

pc : S

128 : S

z = cmv 128 $\not<$ 16 ? 0 : A[128]

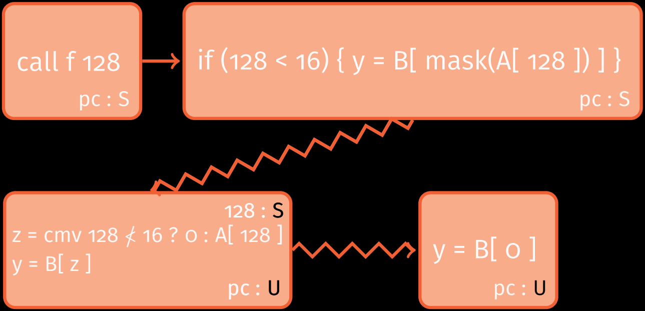
y = B[z]

pc : U

RSSC for SLH

void f(int x) ↦ if(x < A.size){y = B[A[x]]}   // A.size=16, A[128]=3

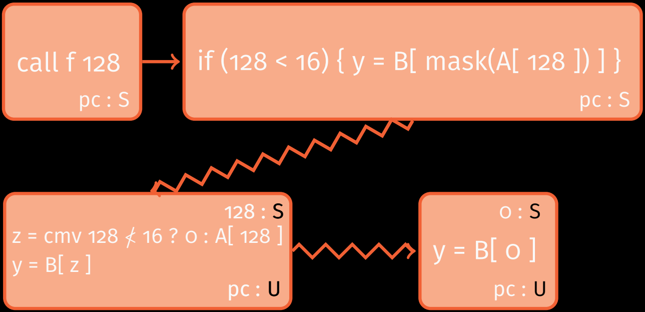
$\llbracket\cdot\rrbracket$ = **void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}**


call f 128
pc : S
if (128 < 16) { y = B[mask(A[128])] }
pc : S
128 : S
z = cmv 128 ≮ 16 ? 0 : A[128]
y = B[z]
pc : U
y = B[0]
pc : U

RSSC for SLH

void f(int x) ↦ if(x < A.size){y = B[A[x]]}   // A.size=16, A[128]=3

$[\![\cdot]\!]$ = void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}



call f 128
pc : S

if (128 < 16) { y = B[mask(A[128])] }
pc : S

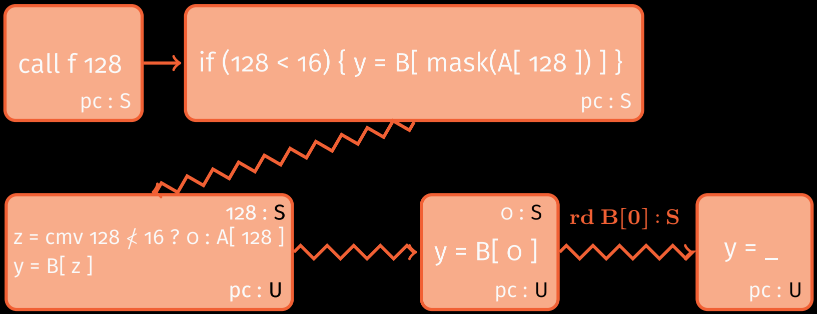
128 : S
z = cmv 128 $\nless$ 16 ? 0 : A[128]
y = B[z]
pc : U

0 : S
y = B[0]
pc : U

RSSC for SLH

void f(int x) ↦ if(x < A.size){y = B[A[x]]}   // A.size=16, A[128]=3

$[\![\cdot]\!]$ = **void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}**



RSSC for SLH

SLH preserves RSS- (and thus RSNI-) but not RSS+ (and thus not RSNI+)

Framework benefits: fine-grained analysis of countermeasures security

vo …28]=3

$[\![\cdot]\!]$ =

call f

pc : S

pc : S

pc : S

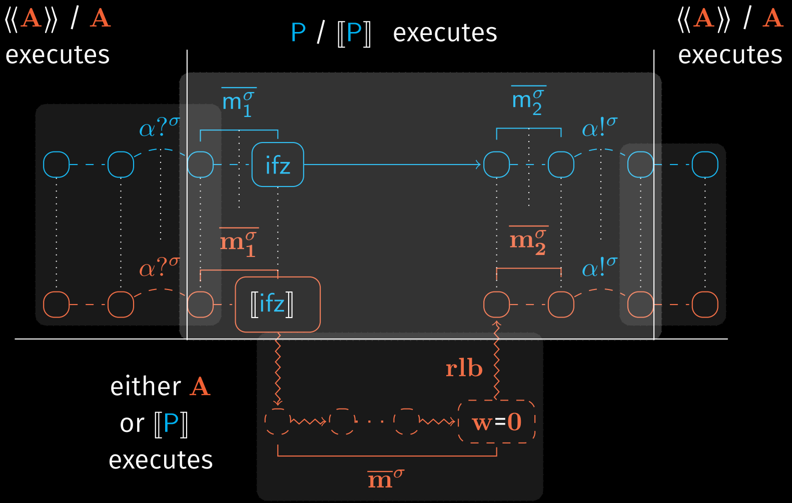
$\mathbf{rd\ B[0] : S}$

Insecurity Results

- MSVC is Insecure
- Non-interprocedural SLH is insecure

Both omit speculation barriers

Proofs Insight


$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
P / ⟦P⟧ executes
$\langle\!\langle \mathbf{A} \rangle\!\rangle$ / $\mathbf{A}$ executes
$\overline{m_1^\sigma}$
$\overline{m_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
ifz
$\overline{\mathbf{m}_1^\sigma}$
$\overline{\mathbf{m}_2^\sigma}$
$\alpha?^\sigma$
$\alpha!^\sigma$
⟦ifz⟧
rlb
either $\mathbf{A}$ or ⟦P⟧ executes
w=0
$\overline{\mathbf{m}^\sigma}$

Beyond V1 Protection

- $RSSP$ with V1 trace model = $RSSP_1$

Beyond V1 Protection

- $RSSP$ with V1 trace model = $RSSP_1$
- wh: $\vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : RSSP_1$ (produces V1-secure code)

Beyond V1 Protection

- $RSSP$ with V1 trace model = $RSSP_1$
- wh: $\vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : RSSP_1$ (produces V1-secure code)
- take $[\![\cdot]\!]^{\mathbf{T}}_{\mathbf{T}}$ that produces V4-secure code

Beyond V1 Protection

- $RSSP$ with V1 trace model = $RSSP_1$
- wh: $\vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : RSSP_1$ (produces V1-secure code)
- take $[\![\cdot]\!]^{\mathbf{T}}_{\mathbf{T}}$ that produces V4-secure code

- if $\vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} : RSSP_1$
- and $\vdash [\![\cdot]\!]^{\mathbf{T}}_{\mathbf{T}} : RSSP_4$
- what do we know about $\vdash \left[\!\!\left[[\![\cdot]\!]^{\mathsf{S}}_{\mathbf{T}} \right]\!\!\right]^{\mathbf{T}}_{\mathbf{T}} :?$

Composition Results

- “Unknown” (but expected(?)):

$$
\begin{aligned}
\text{if} \quad & \vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathsf{I}} : X && (RSSP_1) \\
\text{and} \quad & \vdash [\![\cdot]\!]^{\mathsf{I}}_{\mathbf{T}} : Y && (RSSP_4) \\
\text{then} \quad & \vdash \left[\!\!\left[[\![\cdot]\!]^{\mathsf{S}}_{\mathsf{I}} \right]\!\!\right]^{\mathsf{I}}_{\mathbf{T}} : X \cap Y &&
\end{aligned}
$$

Composition Results

- “Unknown” (but expected(?)):

$$\begin{aligned} \text{if} \quad & \vdash [\![\cdot]\!]^{\mathsf{S}}_{\mathsf{I}} : X && (RSSP_1) \\ \text{and} \quad & \vdash [\![\cdot]\!]^{\mathsf{I}}_{\mathbf{T}} : Y && (RSSP_4) \\ \text{then} \quad & \vdash \left[\!\!\left[[\![\cdot]\!]^{\mathsf{S}}_{\mathsf{I}} \right]\!\!\right]^{\mathsf{I}}_{\mathbf{T}} : X \cap Y \end{aligned}$$

- problem:

$$RSSP_1 \cap RSSP_4 \neq RSSP_1 \cup RSSP_4$$

Proposed Solution (wip)

Instrumentations:

- preserve some [class of] (hyper)property $\mathbf{X}$
- enforce a specific (hyper)property $\mathbf{Y}$

$$\vdash [\![\cdot]\!] \succ_{\mathbf{X}} \mathbf{Y}$$

Proposed Solution (wip)

Instrumentations:

- preserve some [class of] (hyper)property $\mathbf{X}$
- enforce a specific (hyper)property $\mathbf{Y}$

$$\vdash [\![\cdot]\!] \succ_{\mathbf{X}} \mathbf{Y}$$

Proposed Solution (wip)

Instrumentations:

- preserve some [class of] (hyper)property $\mathbf{X}$
- enforce a specific (hyper)property $\mathbf{Y}$

cannot enforce classes

$$\vdash [\![\cdot]\!] \succ_{\mathbf{X}} \mathbf{Y}$$

Instrumentations for Spectre (wip)

if $\vdash \llbracket \cdot \rrbracket^{S}_{I} : RSSP_1$

and $\vdash \llbracket \cdot \rrbracket^{I}_{\mathbf{T}} \succ_{RSSP_1} RSSP_4$

then $\vdash \left\llbracket \llbracket \cdot \rrbracket^{S}_{I} \right\rrbracket^{I}_{\mathbf{T}} : RSSP_1 \cup RSSP_4$

More Generally (wip)

- some optimisation passes may not preserve some property X (specific, not class)

More Generally (wip)

- some optimisation passes may not preserve some property X (specific, not class)
- we need later passes to enforce X

More Generally (wip)

- some optimisation passes may not preserve some property X (specific, not class)
- we need later passes to enforce X
- interesting (unknown(?)) metatheory, very interesting application

Questions?