Exorcising Spectres with Secure Compilers

Marco Patrignani[1,2]

Marco Guarnieri[3]

[1]
CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

[2]
Stanford
University

[3] institute
imdea

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) 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
   - Secure: Intel ICC, SLH(ish), SSLH
   - Insecure: MSVC, Interprocedural SLH

Contributions & Outline

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

- ased 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 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
   - (5) secure: 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

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

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

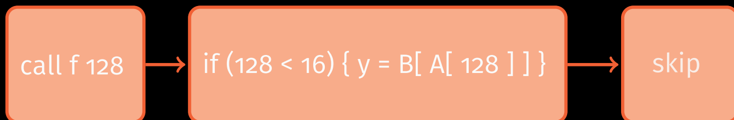
call f 128

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

Speculative Semantics & SNI

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

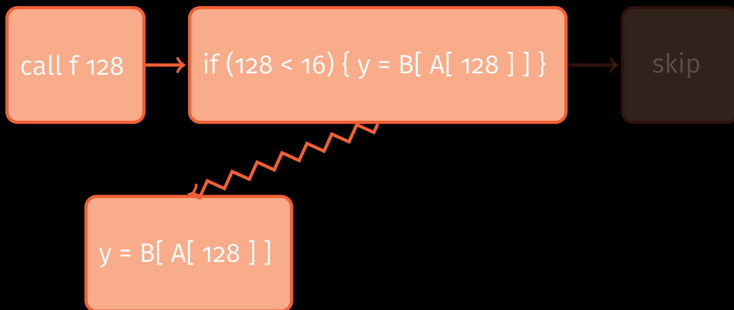
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

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

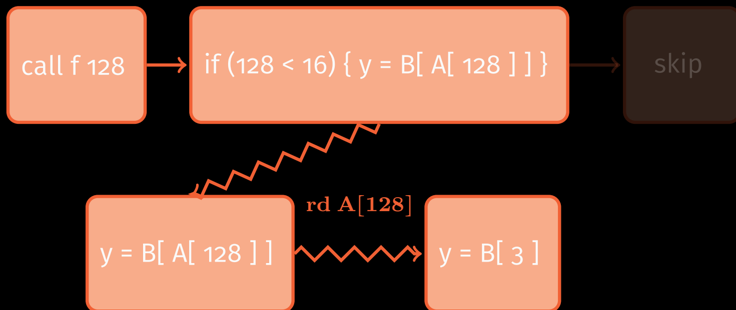
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

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

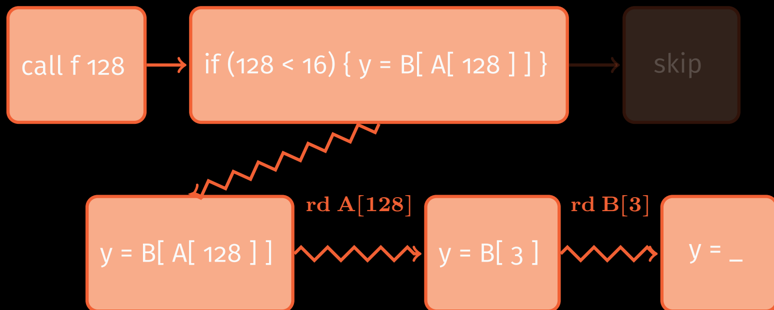
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

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

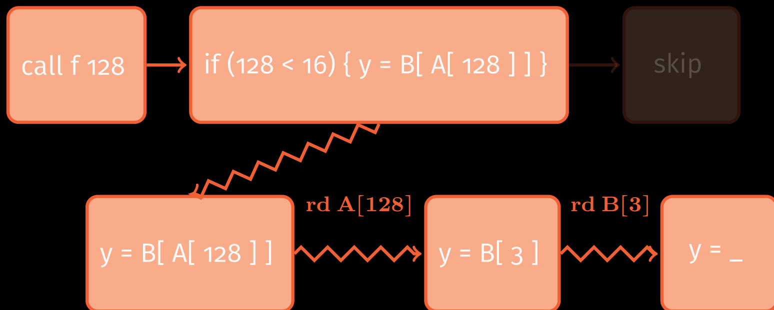
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

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

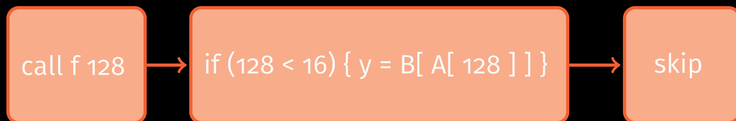
run 1: A.size = 16, A[128] = 3



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



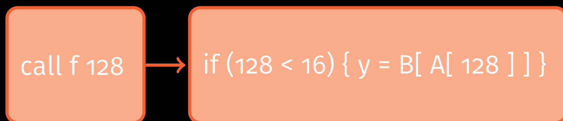
trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

Speculative Semantics & SNI

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

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

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



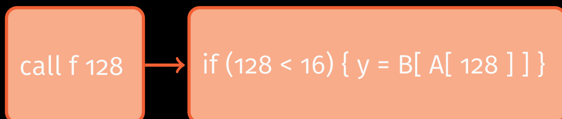
trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

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

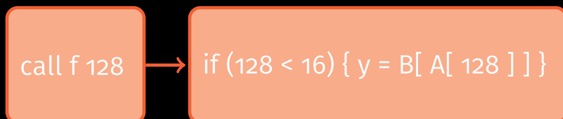
$\mathbf{rd\ A[128]}$

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

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



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

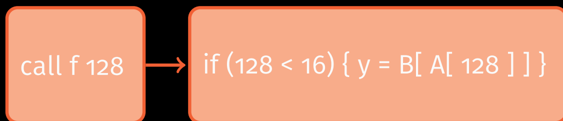
$\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$

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:

- assuming 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

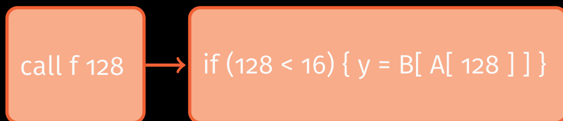
trace 2: $\mathbf{rd}\ \mathbf{A}[128]$ $\mathbf{rd}\ \mathbf{B}[7]$ $\Rightarrow$ SNI violation

Speculative Semantics & SNI

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

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

Problems Problems Problems ...

Problem: Proving compiler preserves SNI is hard

Problems Problems Problems ...

Problem: Proving compiler preserves SNI is hard

Solution: overapproximate SNI with a novel property: speculative safety (SS)

Speculative Safety (SS): 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

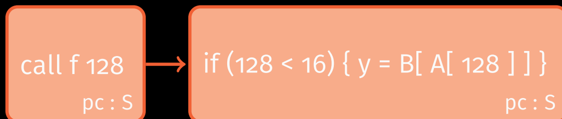
pc : S

Speculative Safety (SS): Taint Tracking

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

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

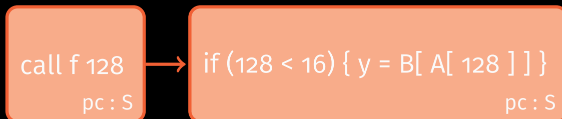


Speculative Safety (SS): Taint Tracking

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

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

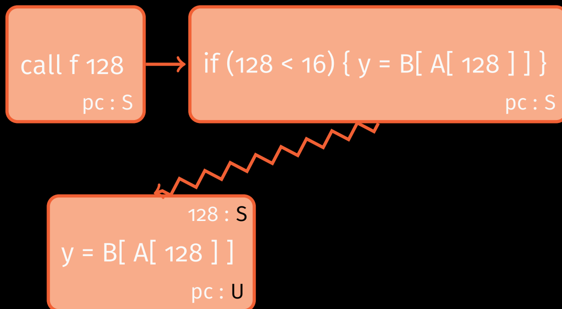


Speculative Safety (SS): Taint Tracking

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

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

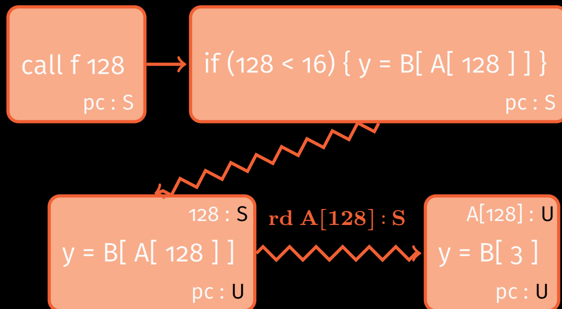


Speculative Safety ($\mathcal{SS}$): Taint Tracking

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

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

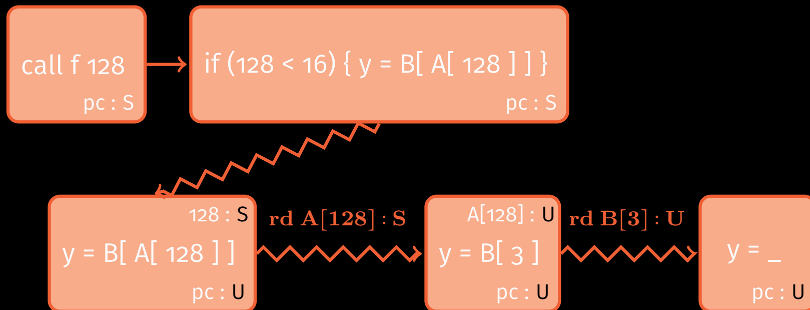


Speculative Safety (SS): Taint Tracking

void f (int x) ↦ 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

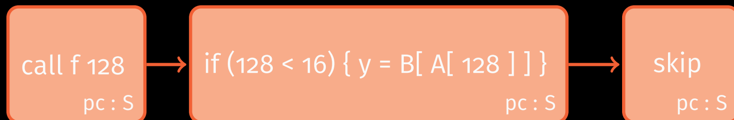


Speculative Safety (SS): Taint Tracking

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

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



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

Speculative Safety (SS): 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

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

call f 128
pc : S

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

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

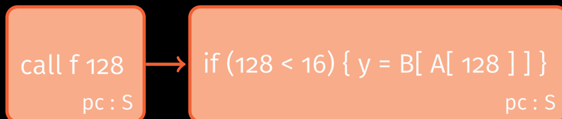
$\mathbf{rd\ B[3] : U}$

Speculative Safety (SS): Taint Tracking

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

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



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

SS and SNI

SS overapproximates SNI, so:

- in the $\mathbf{target}$: $\forall \mathbf{P} \vdash \mathbf{P} : \mathbf{SS} \Rightarrow \mathbf{P} : \mathbf{SNI}$

SS and SNI

SS overapproximates SNI, so:

- in the **target**: $\forall \mathbf{P} \vdash \mathbf{P} : \mathbf{SS} \Rightarrow \mathbf{P} : \mathbf{SNI}$
- in the source: $\forall \mathsf{P} \vdash \mathsf{P} : \mathsf{SS} \iff \mathsf{P} : \mathsf{SNI}$
  (recall, no speculative execution in source)

SS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \mathsf{RSSP} \overset{\text{def}}{=} \text{ if } \forall \mathsf{A}.\mathsf{A}\,[\mathsf{P}] : SS \text{ then } \forall \mathbf{A}.\mathbf{A}\,[[\![\mathsf{P}]\!]] : SS$$

SS-Preserving Compiler: RSSC & RSSP

$$[\![\cdot]\!] : \mathrm{RSSP} \stackrel{\text{def}}{=} \text{if } \forall \mathsf{A}. \mathsf{A}\left[\mathsf{P}\right] : SS \text{ then } \forall \mathbf{A}. \mathbf{A}\left[[\![\mathsf{P}]\!]\right] : SS$$

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

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

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

$$[\![\cdot]\!] : \mathsf{RSSP} \stackrel{\mathsf{def}}{=} \text{ if } \forall \mathsf{A}.\mathsf{A}[\mathsf{P}] : SS \text{ then } \forall \mathbf{A}.\mathbf{A}[[\![\mathsf{P}]\!]] : SS$$

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

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

- $\forall$ attackers: explicit attacker model

  robustness

SS-Preserving Compiler: RSSC & RSSP

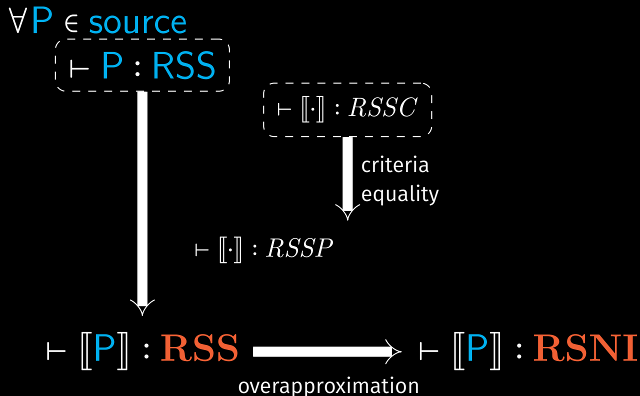
$$[\![\cdot]\!] : \text{RSSP} \stackrel{\text{def}}{=} \text{if } \forall A. A[P] : SS \text{ then } \forall \mathbf{A}. \mathbf{A}[[\![P]\!]] : SS$$

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

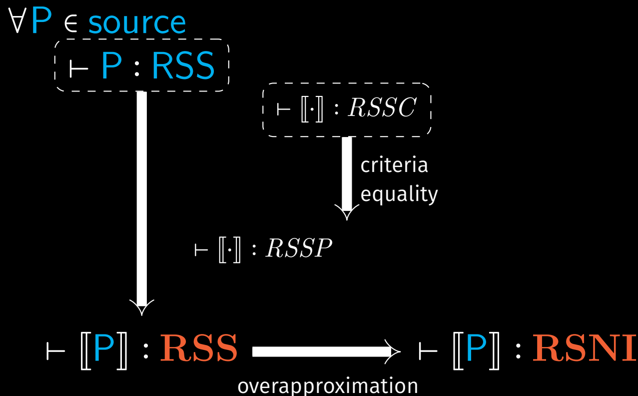
$\approx =$ same traces, plus $\mathbf{S}$ actions in $\mathbf{m}$

- $\forall$ attackers: explicit attacker model
  robustness
- Proof: RSSC & RSSP are equivalent
  RSSC : clear security guarantees
  RSSP : simpler proofs

Secure Compilation Framework for Spectre

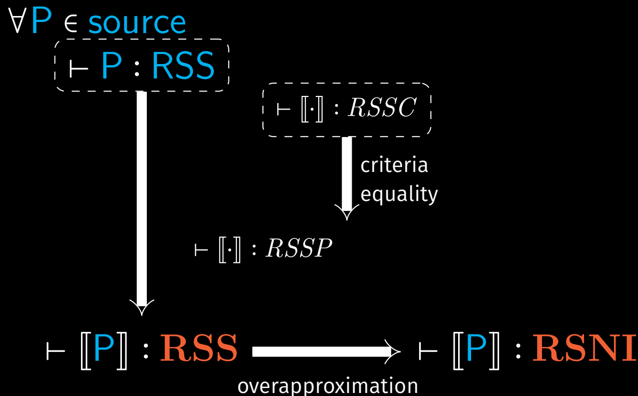


Secure Compilation Framework for Spectre



- dashed premises are already discharged

Secure Compilation Framework for Spectre



- dashed premises are already discharged
- to show security: simply prove $RSSP$

Security Spectrum

- 2 notions of SS and SNI (thus 2 targets):
  - strong(+): no speculative leaks

Security Spectrum

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

Security Spectrum

- 2 notions of SS and SNI (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`

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

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

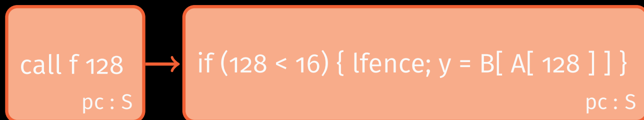
call f 128

pc : S

RSSC for `lfence`

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

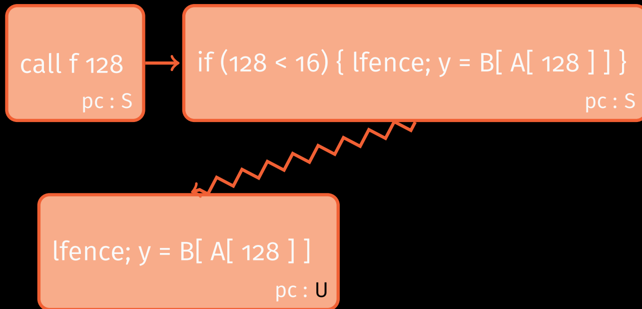
$[\![\cdot]\!] = \textbf{void f(int x)} \mapsto \textbf{if}(\textbf{x} < \textbf{A.size})\{\textbf{lfence; y} = \textbf{B}[\textbf{A}[\textbf{x}]]\}$



RSSC **for** `lfence`

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

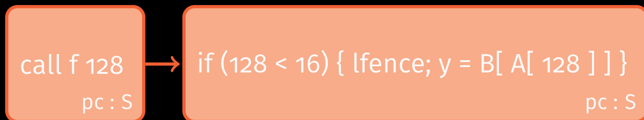
$[\![\cdot]\!]$ = $\mathbf{void\ f(int\ x)} \mapsto \mathbf{if(x < A.size)\{lfence; y = B[A[x]]\}}$



RSSC **for** `lfence`

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

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



RSSC for `lfence`

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

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

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

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

call f 128

pc : S

RSSC **for** SLH

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

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

call f 128
pc : S

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

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])]}**

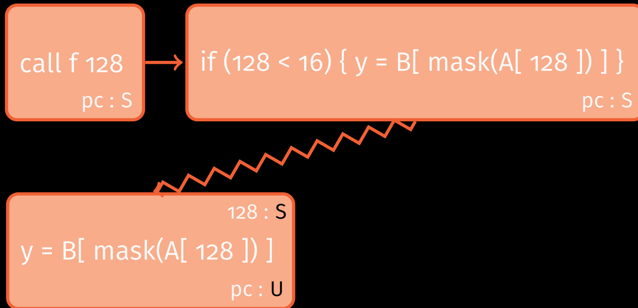
call f 128
pc : S

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

RSSC **for** SLH

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

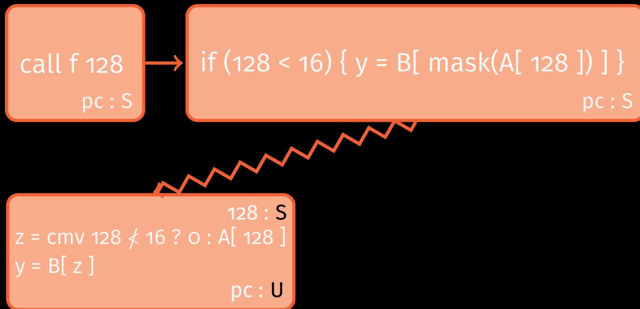
⟦·⟧ = **void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}**



RSSC for SLH

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

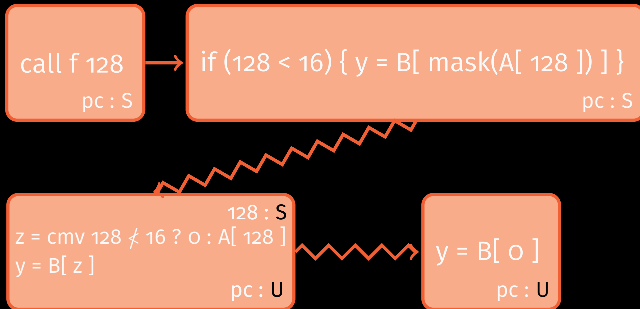
⟦·⟧ = **void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}**



RSSC **for** SLH

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

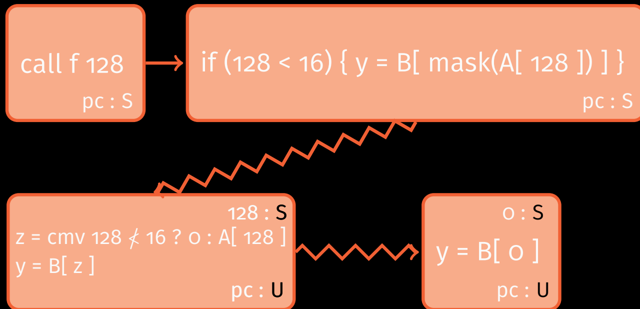
⟦·⟧ = void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}



RSSC for SLH

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

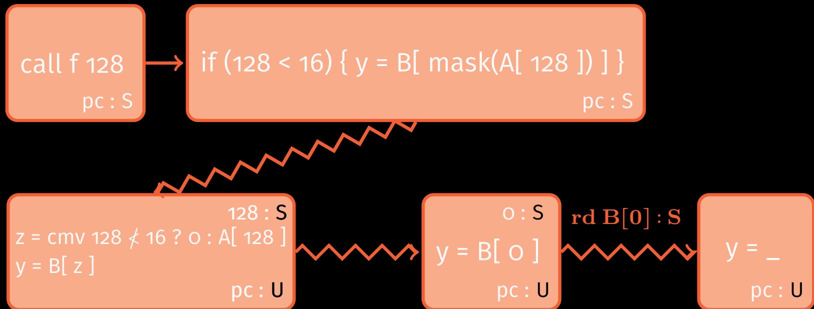
⟦·⟧ = void f(int x) ↦ if(x < A.size){y = B[mask(A[x])]}



RSSC **for** SLH

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

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



RSSC for SLH

SLH preserves SS- (and thus SNI-) but not SS+ (and thus not SNI+)

Framework benefits: fine-grained analysis of countermeasures security

Insecurity Results

- MSVC is Insecure
- Non-interprocedural SLH is insecure

Both omit speculation barriers

Proofs Insight