Exorcising Spectres with Secure Compilers

(wip)

Marco Patrignani[1,2] Marco Guarnieri[3]

9th March 2020

1

2

3

Talk Outline

Slides courtesy of M.G.

Foundations of Secure Compilation

Exorcism via RSC

Spectre v1

Slides courtesy of M.G.

Speculative execution + branch prediction

Size of array **A**

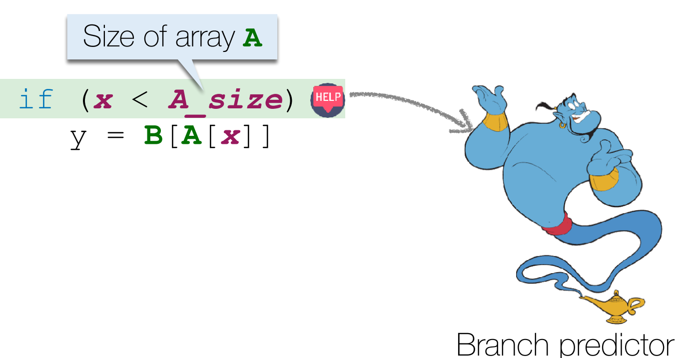
```
if (x < A_size)
    y = B[A[x]]
```

Speculative execution + branch prediction

Size of array **A**

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

Speculative execution + branch prediction



Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

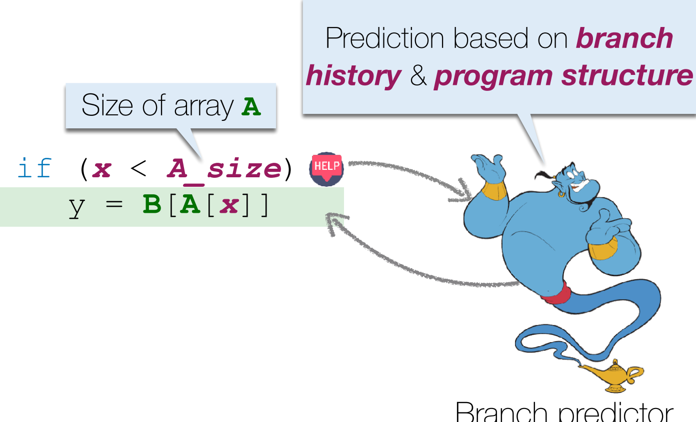
Size of array **A**

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

HELP

Branch predictor

Speculative execution + branch prediction



Speculative execution + branch prediction

Prediction based on ***branch history*** & ***program structure***

Size of array **A**

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

Wrong predicton? ***Rollback changes***!

- ✔ Architectural (ISA) state
- ✖ Microarchitectural state

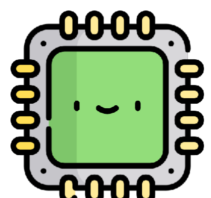
Branch predictor

Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

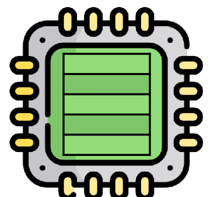
Spectre V1

```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```



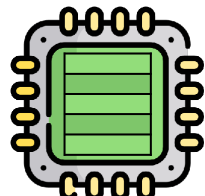
Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



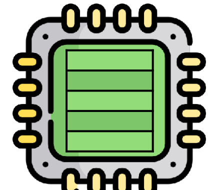
Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

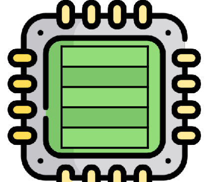


Spectre V1

A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

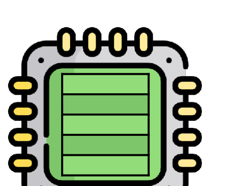


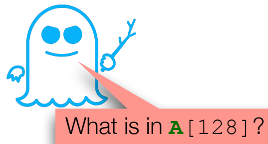
Spectre V1

A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```





What is in A[128]?

Spectre V1

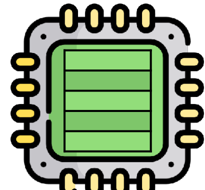
A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

What is in A[128]?

1) Training



Spectre V1

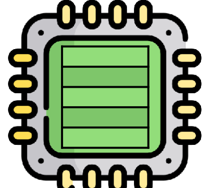
A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

What is in A[128]?

1) Training



Spectre V1

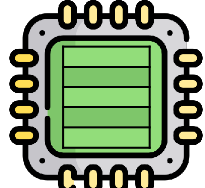
A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

What is in A[128]?

1) Training f(0);



Spectre V1

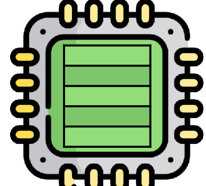
A_size=16

B B[0] B[1] ...

What is in A[128]?

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

1) Training f(0);f(1);

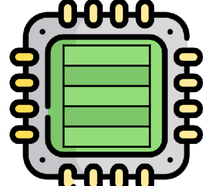


Spectre V1

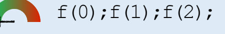
A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



What is in A[128]?



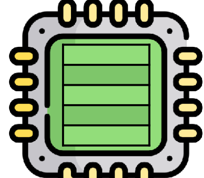
1) Training f(0);f(1);f(2); ...

Spectre V1

A_size=16

B B[0] B[1] ...

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



What is in A[128]?

1) Training f(0);f(1);f(2); …

2) Prepare cache

Spectre V1

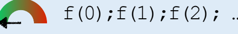
`A_size`=16

B: B[0] B[1] ...

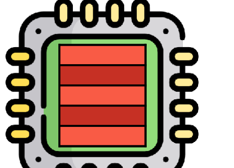
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

What is in `A[128]`?

1) Training



2) Prepare cache



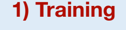
Spectre V1

A_size=16

B | B[0] | B[1] | ...

What is in A[128]?

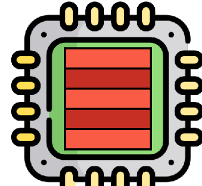
```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```



1) Training f(0);f(1);f(2); …

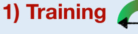
2) Prepare cache

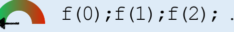
3) Run with x = 128



Spectre V1

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

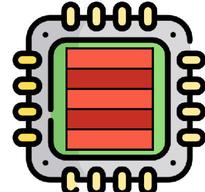




1) Training f(0);f(1);f(2); …

2) Prepare cache

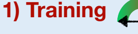
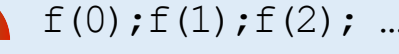
3) Run with *x* = 128



Spectre V1

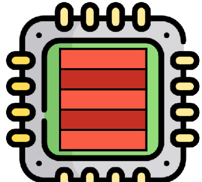
What is in A[128]?

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

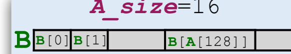
1) Training



2) Prepare cache

3) Run with x = 128



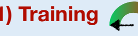
Spectre V1




```
void f(int x)
  if (x < A_size)
      y = B[A[x]]
```

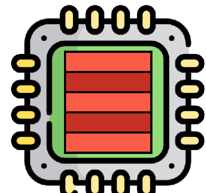
What is in A[128]?

1) Training f(0);f(1);f(2); …

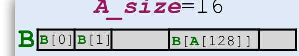


2) Prepare cache

3) Run with x = 128

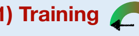
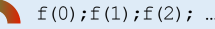


Spectre V1



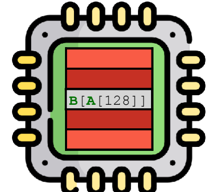

```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```

What is in A[128]?

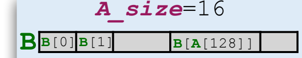
1) Training  

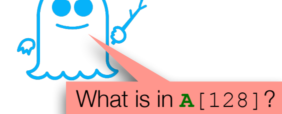
2) Prepare cache

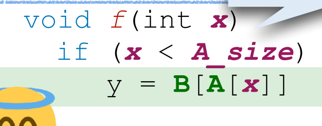
3) Run with x = 128

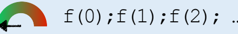


Spectre V1



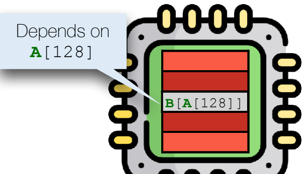




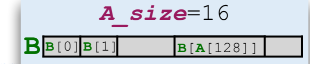
1) Training 

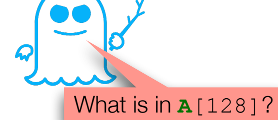
2) Prepare cache

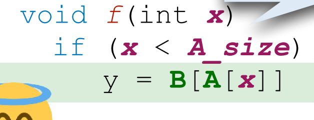
3) Run with x = 128



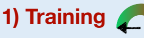
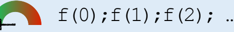
Spectre V1





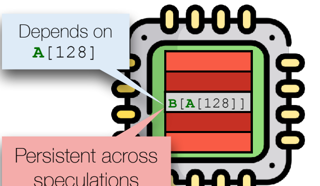


```
void f(int x)
  if (x < A_size)
    y = B[A[x]]
```

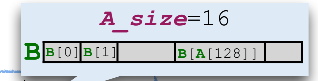
1) Training   `f(0);f(1);f(2); …`

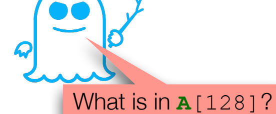
2) Prepare cache

3) Run with `x = 128`



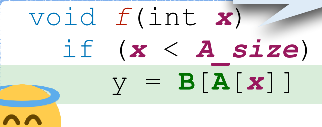
Spectre V1





What is in A[128]?

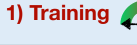
```
void f(int x)
  if (x < A_size)
     y = B[A[x]]
```



Depends on A[128]

B[A[128]]

Persistent across speculations

1) Training  f(0);f(1);f(2); …

2) Prepare cache

3) Run with x = 128

4) Extract from cache

Compiler-level countermeasures

Compiler-level countermeasures

For ***Spectre V1***

Injecting speculation barriers

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

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

- In x86, ***LFENCE*** act as ***speculation barrier***
- Compiler injects LFENCE after each branch instruction
    - Microsoft Visual C++
    - Intel ICC
- Effectively ***stop speculative execution***!

Speculative load-hardening (SLH)

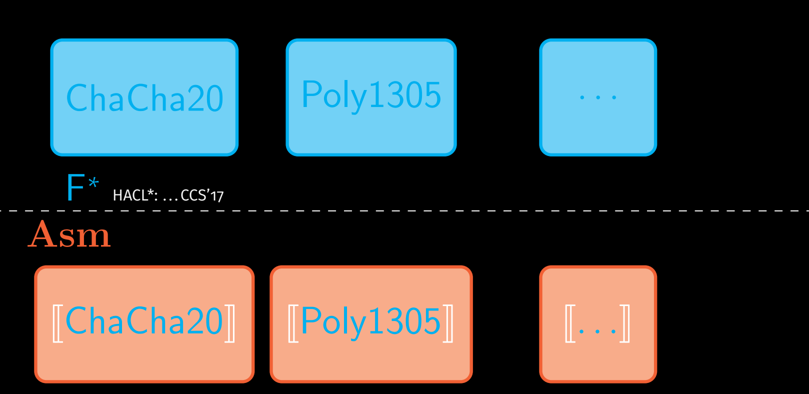
```
if (x < A_size)
    y = B[A[x]]
```

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

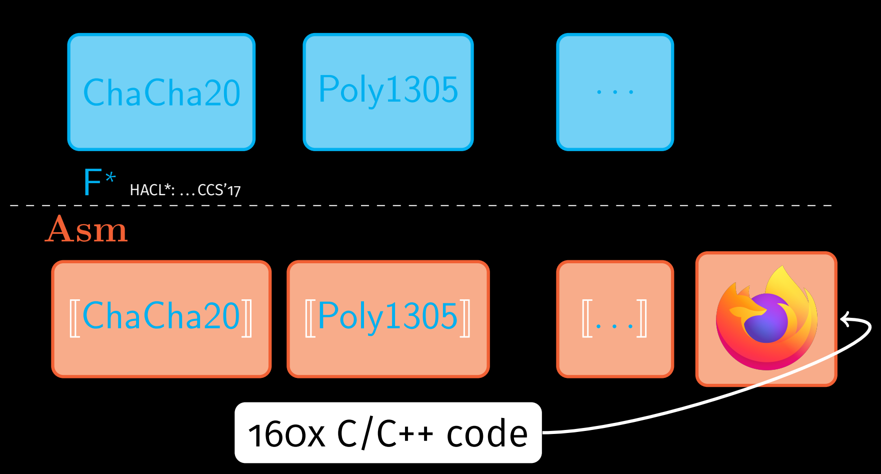
- Injects ***data dependencies*** and ***masking operations***
    - Combines ***conditional moves*** and ***binary operations***
- Stops ***speculative leaks***
- Does not block speculative execution!
- Implemented in Clang

Foundations of Secure Compilation

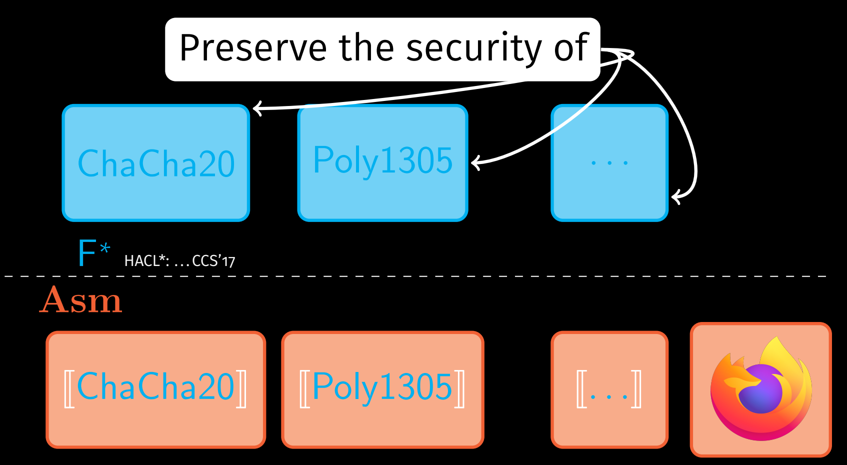
Secure Compilation Goals



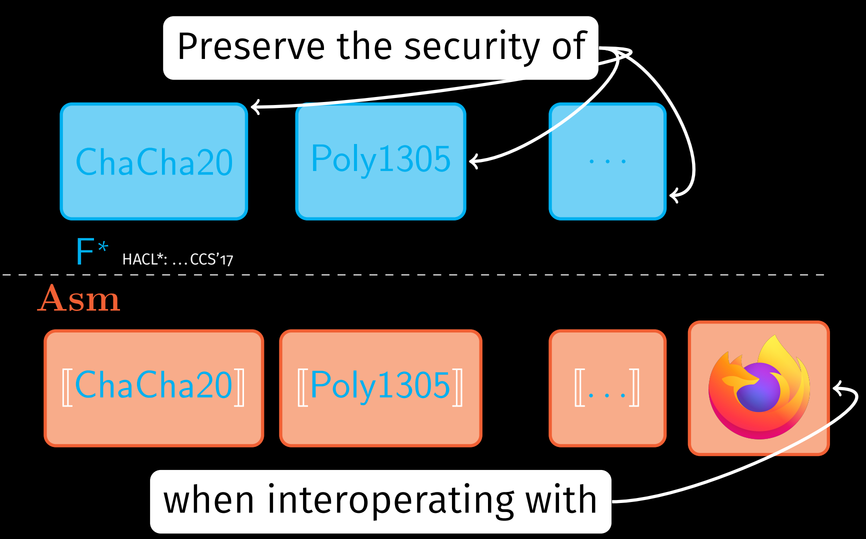
Secure Compilation Goals



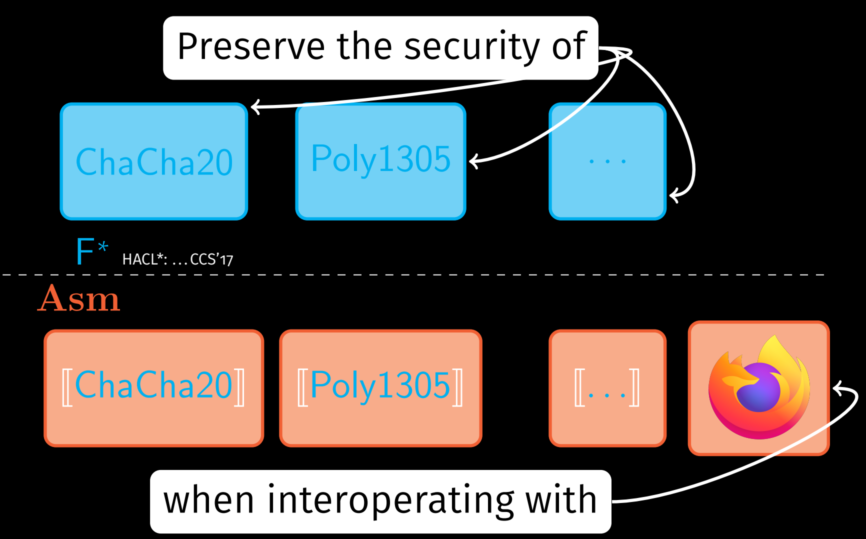
Secure Compilation Goals



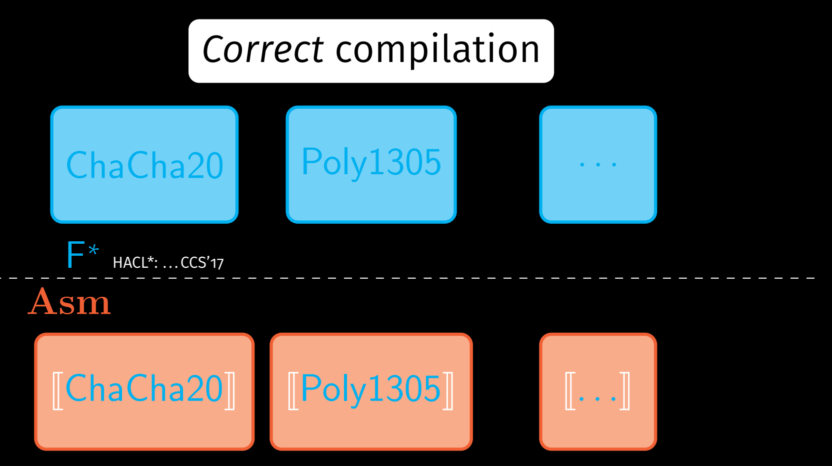
Secure Compilation Goals



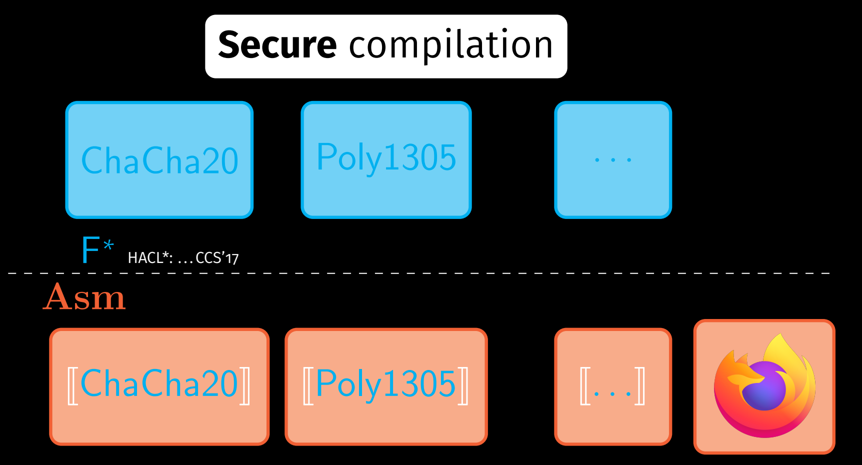
Secure Compilation Goals



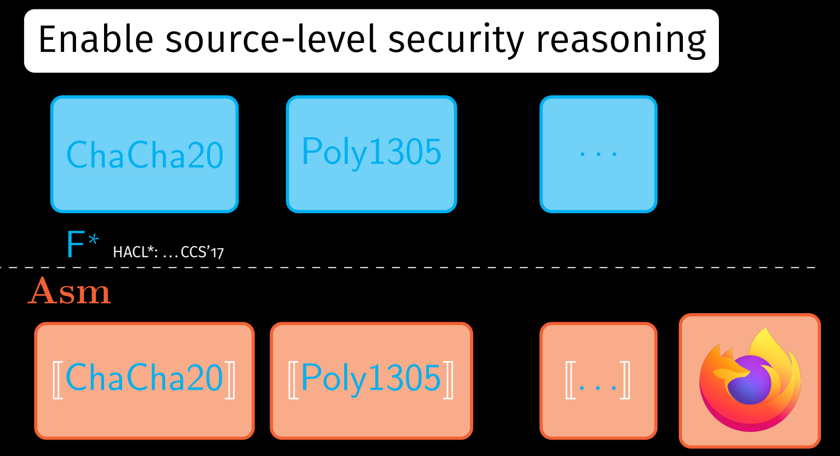
Secure Compilation Goals



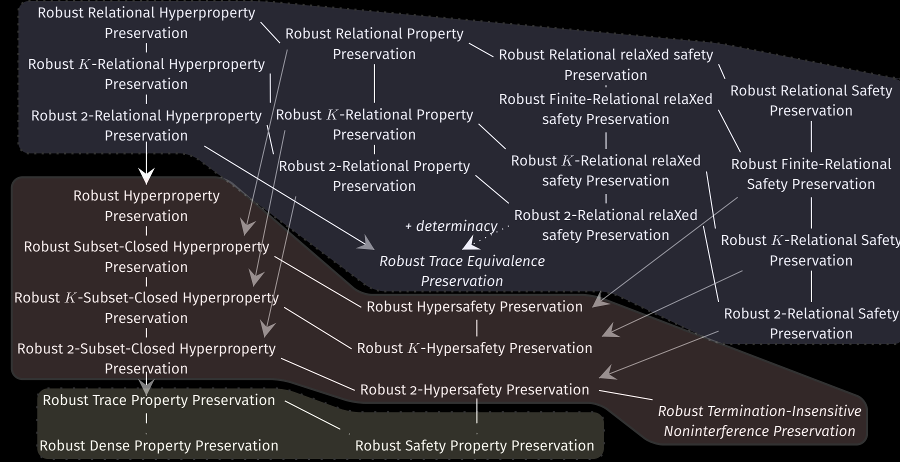
Secure Compilation Goals



Secure Compilation Goals



Robust Compilation Criteria "Journey Beyond Full Abstraction …" CSF'19



Robust Compilation Criteria "Journey Beyond Full Abstraction ..." CSF'19

Relational Hyperproperties

Robust Relational Hyperproperty Preservation

Robust K-Relational Hyperproperty Preservation

Robust 2-Relational Hyperproperty Preservation

Robust Relational Property Preservation

Robust K-Relational Property Preservation

Robust 2-Relational Property Preservation

Robust Relational relaXed safety Preservation

Robust Finite-Relational relaXed safety Preservation

Robust K-Relational relaXed safety Preservation

Robust 2-Relational relaXed safety Preservation

Robust Relational Safety Preservation

Robust Finite-Relational Safety Preservation

Robust K-Relational Safety Preservation

Robust 2-Relational Safety Preservation

+ *determinacy*

Robust Trace Equivalence Preservation

Hyperproperties

Robust Hyperproperty Preservation

Robust Subset-Closed Hyperproperty Preservation

Robust K-Subset-Closed Hyperproperty Preservation

Robust 2-Subset-Closed Hyperproperty Preservation

Robust Hypersafety Preservation

Robust K-Hypersafety Preservation

Robust 2-Hypersafety Preservation

Robust Termination-Insensitive Noninterference Preservation

Trace Properties

Robust Trace Property Preservation

Robust Dense Property Preservation

Robust Safety Property Preservation

Exorcism via RSC

Goal

1. formalise `lfence` & `SLH` compilers

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbb{T}$ must capture speculative execution ($\rightsquigarrow$)

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbb{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC

Goal

1. formalise `lfence` & `SLH` compilers
2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC
5. prove the compilers attain RSSC

Goal Up Next

2. $\mathbf{T}$ must capture speculative execution ($\rightsquigarrow$)
3. need a safety property capturing vulnerability to Spectre v1: SS
4. adapt RSC to preserve SS: RSSC

Speculative Semantics 101 "Spectector …" S&P'20

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

call f 128

Speculative Semantics 101 "Spectector ..." S&P'20

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

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

Speculative Semantics 101 “Spectector …” S&P’20

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

call f 128 → if (128 < 16) { y = B[A[128]] } → skip

Speculative Semantics 101 "Spectector …" S&P'20

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

call f 128 → if (128 < 16) { y = B[A[128]] } → skip

y = B[A[128]]

Speculative Semantics 101 "Spectector …" S&P'20

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

call f 128 → if (128 < 16) { y = B[A[128]] } → skip

y = B[A[128]] ⇝ rdA[128] ⇝ y = B[3]

Speculative Semantics 101 "Spectector ..." S&P'20

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

call f 128

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

skip

y = B[A[128]]

rdA[128]

y = B[3]

rdB[3]

y = _

Speculative Semantics 101 "Spectector ..." S&P'20

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

call f 128 → if (128 < 16) { y = B[A[128]] } → skip

y = B[A[128]] —rdA[128]→ y = B[3] —rdB[3]→ y = _

Speculative Semantics 101 "Spectector ..." S&P'20

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

call f 128 → if (128 < 16) { y = B[A[128]] } → skip

$\mathbf{rdA[128]}$ $\mathbf{rdB[3]}$

Speculative Safety (SS): Taint Tracking

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

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

call f 128
pc : S

Speculative Safety (SS): Taint Tracking

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

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

call f 128
pc : S

→

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

Speculative Safety (SS): Taint Tracking

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

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

call f 128
pc : S

→

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

Speculative Safety (SS): Taint Tracking

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

integrity lattice: $S \subset U$ $S \sqcap U = S$ 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

Speculative Safety (SS): Taint Tracking

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

integrity lattice: $S \subset U$　$S \sqcap U = S$　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

$\mathbf{rdA[128] : S}$

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

Speculative Safety (SS): Taint Tracking

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

integrity lattice: $S \subset U$ $S \sqcap U = S$ 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

$\mathbf{rdA[128] : S}$

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

$\mathbf{rdB[3] : U}$

y = _
pc : U

Speculative Safety (SS): Taint Tracking

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

integrity lattice: $S \subset U$ $S \sqcap U = S$ 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

$\mathbf{rdA[128] : S}$

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

$\mathbf{rdB[3] : U}$

y = _
pc : U

Speculative Safety (SS): Taint Tracking

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

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

call f 128
pc : S

→

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

→

skip
pc : S

$\mathbf{rdA[128] : S}$ $\mathbf{rdB[3] : U}$

SS-Preserving Compiler: RSSC & RSSP

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

SS-Preserving Compiler: RSSC & RSSP

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

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

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

SS-Preserving Compiler: RSSC & RSSP

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

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

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

- RSSC & RSSP are equivalent

$\mathcal{SS}$-Preserving Compiler: RSSC & RSSP

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

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

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

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)

SS-Preserving Compiler: RSSC & RSSP

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

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

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

- RSSC & RSSP are equivalent
- `lfence` : RSSC because it has no speculation ($\mathbf{pc}$: $\mathbf{S}$ always)
- `SLH` : RSSC because masking taints as $\mathbf{S}$

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

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

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

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

call f 128
pc : S

→

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

RSSC **for** `lfence`

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

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

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`

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

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

call f 128
pc : S

→

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

RSSC for `lfence`

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

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

call f 128
pc : S

→

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

→

skip
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

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

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

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

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

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

call f 128
pc : S

→

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

RSSC for SLH

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

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

call f 128
pc : S

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

128 : S
y = B[mask(A[128])]
pc : U

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

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

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

128 : S
z = cmv 128 $\not<$ 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 ≮ 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

⟦·⟧ = **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

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

rd B[0] : S

y = _
pc : U

RSSC **for** SLH

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

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

call f 128 → if (128 < 16) { y = B[mask(A[128])] } → skip

pc : S | pc : S | pc : S

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

Questions?