Secure Compilation

Marco Patrignani
17th October 2018
Collaborators

and more whose image I couldn't find...
Collaborators

interrupt & ask questions

and more whose image I couldn’t find...
Contents

• High-level picture
  (i.e., yes, you should pay attention)
Contents

• High-level picture
  (i.e., yes, you should pay attention)

• Low-level details of a secure compiler
  (i.e., what some published work do)
Contents

• High-level picture
  (i.e., yes, you should pay attention)

• Low-level details of a secure compiler
  (i.e., what some published work do)

• Formal definitions of criteria for secure compilation
  (i.e., why is a secure compiler secure)
Content

- High-level picture
  (i.e., yes, you should pay attention)
- Low-level details of a secure compiler
  (i.e., what some published work do)
- Formal definitions of criteria for secure compilation
  (i.e., why is a secure compiler secure)
- Advanced proof techniques for secure compilation
  (i.e., how much greek gives me a q.e.d.)
What is Secure Compilation?
Compilation

source $S$

target $T$

[Image]

[buffer overflow, privilege escalation, code injection]

Enclaves

ASLR

Cap.

Mach.

Tag.

Mach.

/six.osf//two.osf/six.osf
Correct Compilation

source $S$

target $T$
Correct Compilation

- Buffer overflow
- Privilege escalation
- Code injection

- Enclaves
- ASLR
- Cap, Mach
- Tag, Mach

/six.osf//two.osf/six.osf
Correct Compilation

buffer overflow
privilege escalation
code injection

\[ \{P\} \text{enclaves} \]

Enclaves
ASLR
Cap.
Mach.
Tag.
Mach.

/six.osf//two.osf/six.osf
Correct Compilation

- Buffer overflow
- Privilege escalation
- Code injection

Mon. goto Enclaves

ASLR
Mach.
Tag.
Mach.
Correct Compilation

buffer overflow
privilege escalation
code injection

\{P\}
\{Q\}

Mon
goto
Enclaves
ASLR
Cap.
Mach.
Tag.
Mach.

/six.osf//two.osf/six.osf
Correct Compilation

J ⋅ K

buffer overflow

privilege escalation
code injection

τ

{ P }

{ Q }

Mon
goto

Enclaves

ASLR

Cap.

Mach.

Tag.

Mach.

/six.osf//two.osf/six.osf
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

Mon, goto Enclaves

ASLR, Cap, Mach, Tag, Mach

/six.osf//two.osf/six.osf
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

Mon

Goto

Enclaves

ASLR

Cap.

Mach.

Tag.

Mach.

/six.osf//two.osf/six.osf
Secure Compilation

- Code injection
- Privilege escalation
- Buffer overflow

\{P\}e\{Q\}

Mon
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

Mon

\{P\} e \{Q\}

\(\tau\)

[\cdot]

goto

\text{Mon}

\text{code injection}
\text{privilege escalation}
\text{buffer overflow}

\text{\$/six.osf//two.osf/six.osf}
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

\[ \{ \text{P} \} e \{ \text{Q} \} \]

Mon

\[ \tau \]

Goto

\{ \text{\_\_\_\_} \}
Secure Compilation

- Secure compilation
- Buffer overflow
- Privilege escalation
- Code injection
- Mon
- \(\{P\}e\{Q\}\)
- \(\tau\)
- Enclaves
- ASLR
- Cap.Mach.
- Tag.Mach.
- goto
Secure Compilation

- use security architectures to protect code
Secure Compilation

- use security architectures to protect code
- demonstrate that $[\cdot\cdot\cdot]$ attains security
Secure Compilation

• use security architectures to protect code
  SGX-like enclaves (coming up)
• demonstrate that $\mathcal{J} \cdot \mathcal{K}$ attains security
Secure Compilation

• use security architectures to protect code
  SGX-like enclaves (coming up)
• demonstrate that $\mathcal{J} \cdot \mathcal{K}$ attains security criteria and proof techniques (later)
Secure Compilation

• use security architectures to protect code
  SGX-like enclaves (coming up)
• demonstrate that $\mathcal{J}$ attains security
  criteria and proof techniques (later)

more generally

• **build** securely, don’t fix afterwards
Secure Compilation

- use security architectures to protect code
  SGX-like enclaves (coming up)
- demonstrate that \( J \cdot K \) attains security
criteria and proof techniques (later)

more generally

- **build** securely, don’t fix afterwards
- **understand** what ‘securely’ means
Example of a Secure Compiler

- source = Java-like language
Example of a Secure Compiler

- source = **Java-like language**
- target = **Assembly-like + isolation (sgx-likes)**
Example of a Secure Compiler

- source = **Java-like language**
- target = **Assembly-like + isolation (sgx-likes)**
- based on Agten *et al.*’12, Patrignani *et al.*’15’16

**Warning** fairly high level
Memory Allocation Issues

### Java-like

```java
O1.createAccount()
return
```

### Asm-like

```asm
O1.createAccount()
return
```

#### Issue: Oid guessing
- **Solution:** keep a map from Oid to random numbers

#### Issue: type violation
- **Solution:** add dynamic type checks

**Isolated memory regions**
- **e.g.,** SGX enclaves

**DSL**
- **design**
- **implement**
- **fund a startup in the Bay Area**

```
1.createAccount()
return 3
```

```
2.createAccount()
2.createAccount()
```
Memory Allocation Issues

Java-like

Ext1
Ext2

[O1]
[O2]

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic type checks

Asm-like

Ext1
Ext2

1.createAccount()
2.createAccount()
3.createAccount()

Isolated memory regions
  e.g., SGX enclaves

DSL
  • design
  • implement
  • fund a startup in the Bay Area

/one.osf
/two.osf
/three.osf
/nine.osf/two.osf/six.osf
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

O1
O2

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
  e.g., SGX enclaves

DSL
  • design
  • implement
  • fund a startup in the Bay Area

O1
O2

Memory Allocation Issues

Java-like

O1
O2
O4
O3

return O3

Asm-like

[O1]
[O2]

Ext1
Ext2

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic type checks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

[O1]
[O2]

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
- e.g., SGX enclaves

DSL
- design
- implement
- fund a startup in the Bay Area

```
O1::createAccount()
return

O2::createAccount()

O4::createAccount()
```
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic type checks

Isolated memory regions, e.g., SGX enclaves

Java-like

```
O1
O2
O4
O3
```

Asm-like

```
[O1]
[O2]
[O4]
[O3]
```

```
O1.createAccount()
return
```

```
O2.createAccount()
O4.createAccount()
O3.createAccount()
```

```
1.createAccount()
return 3
```

```
2.createAccount()
2.createAccount()
```

DSL

- design
- implement
- fund a startup in the Bay Area
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
- e.g., SGX enclaves

DSL
- design
- implement
- fund a startup in the Bay Area

Asm-like

Java-like
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

0x001
0x005
0x009
0x00C

return 0x00C

Ext1
Ext2

Issue: Oid guessing
Solution: keep a map
from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions

e.g., SGX enclaves

DSL

• design
• implement
• fund a startup in the Bay Area

/uni21A6/one.osf
/uni21A6/two.osf
/uni21A6/three.osf
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic type checks

Isolated memory regions, e.g., SGX enclaves

Java-like

O1
O2
O3
O4

Asm-like

0x001
0x005
0x009
0x00C

createAccount()
Memory Allocation Issues

Java-like

O1
O2
O4
O3

createAccount()

Asm-like

0x001
0x005
0x009
0x009
0x00C

createAccount()

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

nine.osf//two.osf/six.osf
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic type checks

Isolated memory regions e.g., SGX enclaves

Java-like

O1
O2
O4
O3

createAccount()

return

Asm-like

0x001
0x005
0x009
0x00C

createAccount()
Memory Allocation Issues

Issues:
- **Issue:** Oid guessing
  - **Solution:** keep a map from Oid to random numbers
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

Ext1
Ext2

Ext1
Ext2
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

1.createAccount()
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

0x001/uni21A6/one.osf
0x005/uni21A6/two.osf
0x009/uni21A6/three.osf

1.createAccount()
2.createAccount()
3.createAccount()
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Asm-like

0x001 ⟷ 1
0x005 ⟷ 2
0x009
0x00C ⟷ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

Ext1
Ext2

Ext1
Ext2
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves

Java-like

```
O1
O2
O4
O3
```

Asm-like

```
0x001 ↔ 1
0x005 ↔ 2
0x009 ↔ 3
```

```
J.O1.K.createAccount()
return
```
Memory Allocation Issues

- **Java-like**
  - `O1`
  - `O2`
  - `O4`
  - `O3`

- **Asm-like**
  - `0x001 ↔ 1`
  - `0x005 ↔ 2`
  - `0x009 ↔ 3`

**Issue:** Oid guessing
**Solution:** keep a map from Oid to random numbers

**Issue:** type violation
**Solution:** add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves
Memory Allocation Issues

Java-like

O1: Account
O2: Pair
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

Ext1
Ext2
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves

Java-like

O1: Account
O2: Pair
O4
O3

createAccount()

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Ext1
Ext2
Ext1
Ext2
Memory Allocation Issues

Java-like

O1: Account
O2: Pair
O3
O4

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves

DSL
- design
- implement
- fund a startup in the Bay Area

createAccount()
Memory Allocation Issues

Java-like

O1: Account
O2: Pair
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions, e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

Ext1
Ext2
Memory Allocation Issues

Issue: type violation

Solution: add dynamic typechecks
Memory Allocation Issues

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

Java-like

O1: Account
O2: Pair
O4
O3

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

2. createAccount()
Memory Allocation Issues

Java-like

01: Account
02: Pair
04
03

Asm-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL
• design
• implement
• fund a startup in the Bay Area

1.createAccount()
2.createAccount()
3.createAccount()
Memory Allocation Issues

O1: Account
O2: Pair

Isolated memory regions
e.g., SGX enclaves

0x001 ⇔ 1
0x005 ⇔ 2
0x009
0x00C ⇔ 3
Memory Allocation Issues

Java-like

O1: Account
O2: Pair
O4

ASM-like

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Issue: Oid guessing
Solution: keep a map from Oid to random numbers

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions e.g., SGX enclaves

DSL

• design
• implement
• fund a startup in the Bay Area

Ext1 Ext2

0x001/uni21A6/one.osf
0x005/uni21A6/two.osf
0x009/uni21A6/three.osf
Memory Allocation Issues

- design
- implement
- fund a startup in the Bay Area
1. Is this actually **secure**?
Concerns

1. Is this actually secure?
2. How efficient is this?
Concerns

1. Is this actually secure?
2. How efficient is this?

1. Yes!
1. Is this actually secure?
2. How efficient is this?

1. Yes! So prove it!
Concerns

1. Is this actually secure?
2. How efficient is this?

1. Yes! So prove it!
2. Not bad, but we can aim for better.
We need **criteria** for secure compilation, they:

- tell us what to **prove** about the **compiler** (e.g., compiler correctness, or type soundness criteria)
We need **criteria** for secure compilation, they:

- **tell us what to prove** about the **compiler** (e.g., compiler correctness, or type soundness criteria)
- **impact** efficiency
Concerns

We need **criteria** for secure compilation, they:

- tell us what to **prove** about the **compiler** (e.g., compiler correctness, or type soundness criteria)
- **impact** efficiency
- define security guarantees (what security properties they preserve)
Secure Compilation Criteria
The Origins of the Secure Compiler

Secure Implementation of Channel Abstractions

Martín Abadi
ma@pa.dec.com
Digital Equipment Corporation
Systems Research Center

Cédric Fournet
Cedric.Fournet@inria.fr
INRIA Rocquencourt

Georges Gonthier
Georges.Gonthier@inria.fr
INRIA Rocquencourt

Abstract

Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this spaces are on the same machine, and that a centralized operating system provides security for them. In reality, these address spaces could be spread across a network, and security could depend on several local operating systems and on cryptographic protocols across machines.

Theorem 1 The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes $P$ and $Q$,

$$P \approx Q \text{ if and only if } Env[[P]] \approx Env[[Q]]$$

From the join-calculus to the sjoin-calculus
The Origins of the Secure Compiler

they needed a definition that their implementation of secure channels via cryptography was secure
The Origins of the Secure Compiler

Fully Abstract Compilation (FAC)

**Theorem 1** The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes $P$ and $Q$,

$$P \approx Q \text{ if and only if } \text{Env}[[P]] \approx \text{Env}[[Q]]$$
Fully Abstract Compilation Influence

Typed Closure Conversion Preserves Observational Equivalence

Authentication primitives and their compilation

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Secure Compilation to Protected Module Architectures

Local Memory via Layout Randomization

On Modular and Fully-Abstract Compilations

Fully Abstract Compilation to JavaScript

Secure Implementations for Typed Session Abstraction

Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation

A Secure Compiler for ML Module

An Equivalence-Preserving CPS Translation via Multi-Language Semantics

On Modular and Fully-Abstract Compile
How does Fully Abstract Compilation entail security?
FAC and Security

FAC ensures that a target-level attacker has the same power of a source-level one
x = 1;
x += 2;
x
x = 0;
x += 2;
x
Compiler Full Abstraction

```
x = 1;
x++; = x+= 2;
x x
```
Compiler Full Abstraction

\[ x = 1; \quad x = 0; \]
\[ x++; \quad x += 2; \]
\[ x \]

\[ \text{loadi } r_0 1 \]
\[ \text{inc } r_0 \]
\[ \text{ret } r_0 \]

\[ \text{loadi } r_0 0 \]
\[ \text{addi } r_0 2 \]
\[ \text{ret } r_0 \]
Compiler Full Abstraction

\[ x = 1; \]
\[ x \quad ++; \quad = \quad x \quad += \quad 2; \]
\[ x \]

\[ \text{loadi } r_0 \quad 1 \quad = \quad \text{loadi } r_0 \quad 0 \]
\[ \text{inc } r_0 \quad = \quad \text{addi } r_0 \quad 2 \]
\[ \text{ret } r_0 \quad = \quad \text{ret } r_0 \]
x = 1;  \quad x = 0;

x += 1;  \quad =  \quad x += 2;

and  have different powers!

inc r_0  \quad =  \quad addi r_0 2

ret r_0  \quad =  \quad ret r_0
Why is FAC Secure?

- is an attacker linking or injecting target code
Why is FAC Secure?

- is an attacker linking or injecting target code
Why is FAC Secure?

- is an attacker linking or injecting target code
- is not constrained by source constructs
Why is FAC Secure?

• is an attacker linking or injecting target code
• is not constrained by source constructs
• the co-implied equalities reduce to
Why is FAC Secure?

- is an attacker linking or injecting target code
- FAC protects against target attacks
- the co-implied equalities reduce to
Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

Survey by Patrignani et al.'19, based on Agten et al.'12, Abadi and Plotkin '10, Jagadeesan et al.'11, Patrignani et al.'15'16
Why is FAC Secure?

- FAC protects against target attacks
- confidentiality:
  \[ P_1 = P_2 \iff [P_1] = [P_2] \]
- \( P_1 \) and \( P_2 \) have different secrets
- but they are equivalent
- if the source has it.

Survey by Patrignani et al.’19, based on Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11, Patrignani et al.’15’16
Why is FAC Secure?

- FAC protects against target attacks
- FAC ensures confidentiality
- FAC maintains integrity
- FAC ensures invariant definition
- FAC guarantees memory allocation
- FAC enforces well-bracketed control flow

Confidentiality:

\[ P_1 = P_2 \iff [P_1] = [P_2] \]

- \( P_1 \) and \( P_2 \) have different secrets
- but they are equivalent
- \([P_1]\) and \([P_2]\) also have different secrets
- but they are equivalent

Survey by Patrignani et al.’19, based on Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11, Patrignani et al.’15’16
Why is FAC Secure?

1. Confidentiality:
   \[ P_1 = P_2 \iff [P_1] = [P_2] \]
   - \( P_1 \) and \( P_2 \) have different secrets
   - but they are equivalent
   - \([P_1]\) and \([P_2]\) also have different secrets
   - but they are equivalent
   - so the secret does not leak

Survey by Patrignani et al.’19, based on Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11, Patrignani et al.’15’16
Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

If the source has it.

Survey by Patrignani et al.’19, based on Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11, Patrignani et al.’15’16
Why is FAC Secure?

1. confidentiality
2. integrity
3. • FAC preserves these properties
4. memory allocation
5. well-bracketed control flow

If the source has it.

Survey by Patrignani et al.’19, based on Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11, Patrignani et al.’15’16
Not All That Glitters is Gold

- No support for separate compilation
  [Patrignani et al.’16, Juglaret et al.’16]
• No support for separate compilation

[Patrignani et al.’16, Juglaret et al.’16]
Not All That Glitters is Gold

- No support for separate compilation
  [Patrignani et al.’16, Juglaret et al.’16]
- No support for undefined behaviour
  [Juglaret et al.’16]
Not All That Glitters is Gold

• No support for **separate compilation** [Patrignani *et al.*'16, Juglaret *et al.*'16]

• No support for **undefined behaviour** [Juglaret *et al.*'16]

• **Costly to enforce** [Patrignani and Garg ’19]
• No support for separate compilation
  [Patrignani et al.’16, Juglaret et al.’16]
• No support for undefined behaviour [Juglaret et al.’16]
• **Costly to enforce** [Patrignani and Garg ’19]
• Preserves hypersafety under certain conditions [Patrignani and Garg ’17]
• FAC is not precise about security
• FAC is not precise about security
• this affects efficiency and proof complexity
Alternatives

- FAC is not precise about security
- this affects efficiency and proof complexity
- in certain cases we want easier/more efficient alternatives
Alternatives

• FAC is not precise about security
• this affects efficiency and proof complexity
• in certain cases we want easier/more efficient alternatives

preserve classes of security (hyper)properties
Preserving Safety  Patrignani and Garg ’19, Abate et al.’18

• we have a source program with a safety property
• we have a source program with a safety property against any source attacker
• we have a source program with a safety property against any source attacker
• safety = integrity / weak secrecy / correctness
• we have a source program with a safety property against any source attacker
• safety = integrity / weak secrecy / correctness
• we want its compiled counterpart to have the same safety property
• we have a source program with a safety property against any source attacker
• safety = integrity / weak secrecy / correctness
• we want its compiled counterpart to have the same safety property against any target attacker
Preserving Safety  Patrignani and Garg ’19, Abate et al.’18

- property ($\pi$) = set of traces
  $\{t_1, t_2, \ldots\}$
- traces ($t$) = infinite sequences of observables
- prefixes ($m$) = finite sequences of observables
- $P \sim t = \text{program } P \text{ generates trace } t$
RSC : \( \forall \pi, \pi \in Safety. \pi \approx \pi. \forall P. \)
\[(\forall C_S, t. \ C_S[P] \Rightarrow t \Rightarrow t \in \pi) \Rightarrow (\forall C_T, t. \ C_T[[P]] \Rightarrow t \Rightarrow t \in \pi)\]
\( RSC \): \[ \forall \pi, \pi \in \text{Safety}. \pi \approx \pi. \forall P. \\
\left( \forall C_S, t. \ C_S [P] \sim t \Rightarrow t \in \pi \right) \Rightarrow \left( \forall C_T, t. \ C_T [\llbracket P \rrbracket] \sim t \Rightarrow t \in \pi \right) \]

\( PF-RSC \): \[ \forall P. \forall C_T. \forall m, m. m \approx m. \\
C_T [\llbracket P \rrbracket] \sim m \Rightarrow \exists C_S. C_S [P] \sim m \]
Preserving Safety  Patrignani and Garg ’19, Abate et al.’18

\[ RSC : \forall \pi, \pi \in Safety. \pi \approx \pi. \forall P. \]
\[ (\forall C_S, t. C_S[P] \sim t \Rightarrow t \in \pi) \]
\[ \Rightarrow (\forall C_T, t. C_T[\llbracket P \rrbracket] \sim t \Rightarrow t \in \pi) \]

\[ \Uparrow \]

\[ PF-RSC : \forall P. \forall C_T. \forall m, m. m \approx m. \]
\[ C_T[\llbracket P \rrbracket] \sim m \Rightarrow \exists C_S. C_S[P] \sim m \]
Not All That Glitters is Gold #2

- RSC leads to **more efficient** compiled code
Not All That Glitters is Gold #2

- RSC leads to more efficient compiled code
- RSC is simpler to prove than FAC
• RSC leads to more efficient compiled code
• RSC is simpler to prove than FAC
• but it’s weaker: no confidentiality
Not All That Glitters is Gold #2

- RSC leads to more efficient compiled code
- RSC is simpler to prove than FAC
- but it’s weaker: no confidentiality (weaker than existing FAC works)
Proof Techniques for Secure Compilation
Proving FAC

\[
P_1 \simeq_{ctx} P_2
\]

\[
[P_1] \simeq_{ctx} [P_2]
\]
Proving FAC

\[ P1 \sim_{ctx} P2 \]

\[ [P1] \sim_{ctx} [P2] \]
Proving FAC

\[ P1 \overset{\text{ctx}}{\sim} P2 \]

\[ \downarrow \]

\[ [P1] \overset{\text{ctx}}{\sim} [P2] \]
Proving FAC

\[ \text{P1} \sim_{\text{ctx}} \text{P2} \]

\[ \forall C. C[[\text{P1}]] \downarrow \iff C[[\text{P2}]] \downarrow \]
Proving FAC (History)

\[ P_1 \sim_{ctx} P_2 \]

\[ \Downarrow \]

\[ [P_1] \perp [P_2] \]
Proving FAC (History)

\[ P_1 \simeq_{ctx} P_2 \]

- Jagadeesan et al.’11,
- Agten et al.’12,
- Patrignani et al.’15’16,
- Juglaret et al.’16
Proving FAC (History)

\[ P_1 \sim_{ctx} P_2 \]

\[ \Downarrow \]

\[
\begin{bmatrix} P_1 \end{bmatrix} \sim \begin{bmatrix} P_2 \end{bmatrix}
\]
Proving FAC (History)

\[ P_1 \sim_{ctx} P_2 \]

Abadi et al.’00’01’02’
Bugliesi et al.’07
Adao et al.’06
Fournet et al.’13
Proving FAC (History)

\[ P_1 \approx_{ctx} P_2 \]

\[
\begin{align*}
\left[ P_1 \right] & \approx_n \left[ P_2 \right]
\end{align*}
\]
Proving FAC (History)

\[ P_1 \sim_{ctx} P_2 \]

Ahmed et al.’8’11’14’15’16’17,
Devriese et al.’16’17

\[ [P_1] \sim_n [P_2] \]
Proving FAC with Logical Relations

\[
P_1 \sim_{ctx} P_2
\]

\[
[P_1] \sim_{ctx} [P_2]
\]
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

\[ \mathbb{C}[\llbracket P_1 \rrbracket] \downarrow_n \Rightarrow \mathbb{C}[\llbracket P_2 \rrbracket] \downarrow \]

\[ \llbracket P_1 \rrbracket \sim_{ctx} \llbracket P_2 \rrbracket \]
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

\[
\langle \langle C \rangle \rangle_n \sim_n C \\
P_1 \sim_\_ \[P_1]\]
\]

(1)

\[
C\left[\left[P_1\right]\right] \downarrow_n ? \Rightarrow C\left[\left[P_2\right]\right] \downarrow_\_ \\
\left[P_1\right] \sim_{ctx} \left[P_2\right]
\]

approx. compiler security

COURSE coming up next semester (and next year)
Proving FAC with Logical Relations

\[ P_1 \sim_{\text{ctx}} P_2 \]

\[ \llbracket C \rrbracket_n \llbracket P_1 \rrbracket \Downarrow_\_ \]

\[ \llbracket P_1 \rrbracket \sim_{\text{ctx}} \llbracket P_2 \rrbracket \]

approx. compiler security
Proving FAC with Logical Relations

\[ \text{P1} \sim_{\text{ctx}} \text{P2} \]

\[ \langle \langle C \rangle \rangle_n [\text{P1}] \Downarrow \quad \Rightarrow \quad \langle \langle C \rangle \rangle_n [\text{P2}] \Downarrow \]

\[ \langle \langle C \rangle \rangle_n \sim_n C \quad \text{(1)} \]

\[ \text{C}[[\text{P1}]] \Downarrow_n \quad \Rightarrow \quad \text{C}[[\text{P2}]] \Downarrow \]

\[ [\text{P1}] \sim_{\text{ctx}} [\text{P2}] \]
Proving FAC with Logical Relations

\[ P_1 \equiv_{\text{ctx}} P_2 \]

\[ \langle C \rangle_n[P_1] \Downarrow \Rightarrow \langle C \rangle_n[P_2] \Downarrow \]  
(2)

\[ \langle C \rangle_n \sim_n C \]

\[ P_1 \sim_\square [P_1] \]
\[ P_2 \sim_\square [P_2] \]

\[ \langle C \rangle_n \sim_\square C \]

\[ C[[P_1]] \Downarrow_n \Rightarrow \quad C[[P_2]] \Downarrow \]

\[ [P_1] \sim_{\text{ctx}} [P_2] \]
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

\[ \langle C \rangle_n \sim_n \langle C \rangle_{\langle P_1 \rangle} \sim_{\langle C \rangle_{\langle P_2 \rangle}} \]

\[ P_1 \sim_{\langle P_1 \rangle} \]

**P1 \sim_{\langle P_1 \rangle}** is obtained with standard techniques

Benton *et al.*'09'10

Hur *et al.*'11

Neis *et al.*'15

\[ C[\langle P_1 \rangle] \downarrow_n \Rightarrow C[\langle P_2 \rangle] \downarrow_\sim \]

\[ \langle P_1 \rangle \sim_{ctx} \langle P_2 \rangle \]
Proving FAC with Logical Relations

\[ \langle\langle C\rangle\rangle_n \sim C \] requires

- back-translation of terms
- reasoning at the type of back-translated terms

\[ [P1] \sim_{ctx} [P2] \]
Proving FAC with Logical Relations

\[ \langle C \rangle_n \sim C \] requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation

\[ [P1] \sim_{\text{ctx}} [P2] \]
Proving FAC with Logical Relations

\[
\langle C \rangle_n \sim C \quad \text{requires}
\]

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too
Proving FAC with Logical Relations

\[ \langle \langle C \rangle \rangle_n \sim C \] requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too

\[ [P_1] \sim_{ctx} [P_2] \]
Proving FAC with Logical Relations

\[
\langle \langle C \rangle \rangle_n \sim C \text{ requires}
\]

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for RSC too
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

\[ \langle C \rangle_n[P_1] \Downarrow \Rightarrow \langle C \rangle_n[P_2] \Downarrow \]

\[ C[[P_1]] \Downarrow_n \Rightarrow C[[P_2]] \Downarrow \]

\[ [P_1] \sim_{ctx} [P_2] \]

COURSE
coming up next semester
(and next year)
Conclusion

• motivations for secure compilation
Conclusion

• motivations for secure compilation
• secure compilation criterion: fully abstract compilation
Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- secure compilation criterion: robustly-safe compilation
Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- secure compilation criterion: robustly-safe compilation
- proof techniques for secure compilation
Research Field Prospect

• secure compilation workshop: PrISC 3rd ed. (co-located with POPL)
• secure compilation classes: Winter quarter ’18-19, Spring quarter ’19-20 (?)
• introductory survey: Patrignani, Ahmed, Clarke. ACM CSUR ’19
• lots of challenging open problems to work on (talk to me!)
Questions?