Secure Compilation

an extensive introduction

Marco Patrignani

9th October 2017
Collaborators
Contents

What is Secure Compilation?

Secure Compilation Criteria

Programming Languages Techniques for Secure Compilation

Security Architectures for Secure Compilation
What is Secure Compilation?
Compilation

source S
Compilation

source $S$

$[\cdot]_S^T$

buffer overflow
privilege escalation
code injection

Mon
goto

PMA
ASLR
CM

/six.osf/
Compilation

source $S$

target $T$

\[ \begin{bmatrix} \cdot \end{bmatrix}^S_T \]
Correct Compilation

\[
\begin{bmatrix}
\cdot \\
\cdot 
\end{bmatrix}^S_T
\]
Correct Compilation

\[
\left[ \cdot \right]^S_T
\]
Correct Compilation
Correct Compilation

\[
\begin{bmatrix}
\cdot & \cdot \\
\end{bmatrix}_{ST}
\]

buffer overflow
privilege escalation
code injection

\{P\} \rightarrow \{Q\}

Mon

goto

PMA
ASLR
CM
PUMP
Correct Compilation

[ . ]^{ST}

buffer overflow
privilege escalation
code injection

Mon

goto

PMA

ASLR

CM

PUMP

/six.osf
Secure Compilation

buffer overflow
privilege escalation
code injection

\[ \underbrace{\ldots}^{S} \]

\[ \underbrace{\ldots}^{T} \]
Secure Compilation

\[
[ \cdot ]^{ST}
\]

code injection
privilege escalation
buffer overflow
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

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

\[ \tau \]

\[ [ \cdot ]_{S_T} \]
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection
- τ

\{P\}e\{Q\}

\[ S_T \]

goto

Mon

PMA

ASLR

CM

PUMP

/six.osf
Secure Compilation

\[ \begin{bmatrix} \lfloor \cdot \rfloor^S_T \end{bmatrix} \]

goto

\{P\}e\{Q\}

Mon

\[ \tau \]

\{P\}e\{Q\}

PMA

ASLR

CM

PUMP

buffer overflow

privilege escalation

code injection
Memory Allocation Issues

Patrignani et al.’15’16

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

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
• design
• implement
...or?

/one.osf/five.osf/six.osf

Java-like

Ext1
Ext2

O1
O2

Asm-like

Ext1
Ext2

[O1] [O2]
Memory Allocation Issues

Patrignani et al.’15’16

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
• design
• implement
• ...or?

\[
\begin{bmatrix}
[O1] & [O2]
\end{bmatrix}
\]
Memory Allocation Issues

Patrignani et al.’15’16

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

• design
• implement
• ... or?
Memory Allocation Issues

Patrignani et al.’15’16

O1
O2
O4
O3

return

Ext1
Ext2

O1

Java-like

O2

O4

O3


Ext1
Ext2

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
- design
- implement
- or?
Memory Allocation Issues

Patrignani et al.’15’16

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

• design
• implement
Memory Allocation Issues

Patrignani et al.’15’16

Java-like

O1
O2
O4
O3

Ext1
Ext2

Asm-like

[O1]S
[O2]S
[O4]S
[O3]S

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
• design
• implement
• ...or?
Memory Allocation Issues

Patrignani et al.'15'16

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

• design
• implement
• ...or?

/one.osf
/two.osf
/three.osf
/five.osf
/six.osf

O1
O2
O4
O3

Java-like

Asm-like

Ext1
Ext2

return

[O3]^{s}_{t}

/O4
/O3

[O1]^{s}_{t}

[O2]^{s}_{t}

[O1]^{s}_{t}

return

[O3]^{s}_{t}
Memory Allocation Issues

Patrignani et al.’15’16

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

• design
• implement
• ...or?

/ seven.osf
Memory Allocation Issues

Patrignani et al.’15’16

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
• design
• implement
• ... or?

Java-like

O1
O2
O4
O3

Asm-like

0x001
0x005
0x009
0x00C

0x009 .createAccount()
Memory Allocation Issues

Patrignani et al.’15’16

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
...or?

0x001
0x005
0x009
0x00C

createAccount()

createAccount()

createAccount()
Memory Allocation Issues

Patrignani et al.’15’16

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

Issue: type violation
Solution: add dynamic typechecks

Asm-like
Ext1  Ext2

Java-like

Ext1  Ext2

0x001
0x005
0x009
0x00C

createAccount()
Issue: Oid guessing

Solution: keep a map from Oid to random numbers
Memory Allocation Issues

Patrignani et al.'15'16

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

- design
- implement

...or?
Memory Allocation Issues

Patrignani et al.’15’16

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

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
- design
- implement
- ... or?
Memory Allocation Issues

Patrignani et al.'15'16

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

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
- design
- implement

...or?

0x001 \rightarrow 1
0x005 \rightarrow 2
0x009
0x00C
Memory Allocation Issues

Patrignani et al.’15’16

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

Issue: type violation
Solution: add dynamic typechecks

Isolated memory regions
- design
- implement

...or?

Java-like
- O1
- O2
- O4
- O3

Asm-like
- 0x001 ↔ 1
- 0x005 ↔ 2
- 0x009
- 0x00C ↔ 3
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
• design
• implement
• ...or?

Asm-like

\[
\begin{array}{c}
0x001 \leftrightarrow 1 \\
0x005 \leftrightarrow 2 \\
0x009 \\
0x00C \leftrightarrow 3
\end{array}
\]

Java-like

\[
\begin{array}{c}
O1 \\
O2 \\
O4 \\
O3
\end{array}
\]
Memory Allocation Issues  Patrignani et al.’15’16

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
- design
- implement
- ... or?
Memory Allocation Issues

Patrignani et al.’15’16

O1: Account
O2: Pair
O4
O3

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

• design
• implement
• ...or?

/one.osf
/two.osf
/three.osf
/four.osf
/five.osf
/six.osf
/seven.osf
Memory Allocation Issues

Patrignani et al.'15'16

Java-like

O1: Account
O2: Pair
O4
O3

createAccount()

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 type checks

Isolated memory regions e.g., SGX enclaves

• design
• implement
• ...or?
Memory Allocation Issues

Patrignani et al.’15’16

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
• design
• implement
• ...or?
Memory Allocation Issues  Patrignani et al.'15'16

- **Java-like**:
  - O1: Account
  - O2: Pair
  - 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
  - • design
  - • implement
  - • or?
Issue: type violation

Solution: add dynamic typechecks
Memory Allocation Issues

Patrignani et al.’15’16

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
- design
- implement
- ... or?

/one.osf/five.osf/six.osf
Memory Allocation Issues

Patrignani et al.’15’16

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
• design
• implement
...or?
Memory Allocation Issues

Patrignani et al.’15’16

Isolated memory regions
e.g., SGX enclaves

O₁: Account
O₂: Pair

0x001 \rightarrow 1
0x005 \rightarrow 2
0x009
0x00C \rightarrow 3

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

Issue: type violation
Solution: add dynamic typechecks

Asm-like

Ext1
Ext2

Ext1
Ext2
Memory Allocation Issues  Patrignani et al.’15’16

• design
• implement

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

• design
• implement
• design
• implement
• ...or?
Guarantees

• How do we know we are right?
Guarantees

• How do we know we are right?
• How can we know that $\cdot \left[ \begin{array}{c} S \\ T \end{array} \right]$ is secure?
Guarantees

Prove \( [\cdot]^S_T \) to attain a secure compilation criterion
Guarantees

• How do we know we are right?
• How can we know that $[\cdot]^S_T$ is secure?
• What do we mean with secure?
Show the security implications of the criterion
Secure Compilation Criteria
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

From the join-calculus to the sjoin-calculus

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

\[ P \approx Q \text{ if and only if } \text{Env}[[P]] \approx \text{Env}[[Q]] \]
they needed a definition that their implementation of secure channels via cryptography was secure
The main question they had (and we still have):

what are good correctness criteria for secure compilers?
The Origins of the Secure Compiler

The main question they had (and we still have):
what are good correctness criteria for secure compilers?

Fully Abstract Compilation (FAC)

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

$$P \approx Q \iff \text{Env}[\llbracket P \rrbracket] \approx \text{Env}[\llbracket Q \rrbracket]$$
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

Fully Abstract Compilation to JavaScript

Fully Abstract Compilation via Universal Embedding
Why does Fully Abstract Compilation entail security?
Because FAC ensures that a \textit{target-level} attacker has the same power as a \textit{source-level} one.
x = 1;
x += 1;
x = 0;
x += 2;
Compiler Full Abstraction

\[
x = 1; \\
x = 0; \\
x + +; = x += 2; \\
x
\]
Compiler Full Abstraction

\[
\begin{align*}
\text{x} &= 1; \\
\text{x} &= 0; \\
\text{x} &+ +;
\end{align*}
\]

\[
\begin{align*}
\text{x} &= \text{0+2;} \\
\text{x} &= \text{x++2;} \\
\text{x} &= \text{x=x+=2;} \\
\text{x} &= \text{x=x+=2;} \\
\text{x} &= \text{x=x+=2;}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 1} \\
\text{inc r0} \\
\text{ret r0}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 0} \\
\text{addi r0 2} \\
\text{ret r0}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 1} \\
\text{inc r0} \\
\text{ret r0}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 0} \\
\text{addi r0 2} \\
\text{ret r0}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 1} \\
\text{inc r0} \\
\text{ret r0}
\end{align*}
\]

\[
\begin{align*}
\text{loadi r0 0} \\
\text{addi r0 2} \\
\text{ret r0}
\end{align*}
\]
Compiler Full Abstraction

\[
\begin{align*}
x &= 1; & x &= 0; \\
x &= x++; & = x += 2; \\
\end{align*}
\]

\[
\begin{align*}
\text{loadi } r_0 & \ 1 & \text{loadi } r_0 & \ 0 \\
\text{inc } r_0 & = \text{addi } r_0 & \ 2 \\
\text{ret } r_0 & & \text{ret } r_0 \\
\end{align*}
\]
Compiler Full Abstraction

\[
\begin{align*}
    x &= 1; & x &= 0; \\
    x &= ++; & x &= += 2;
\end{align*}
\]

and have different powers!

\[
\begin{align*}
    \text{inc } r_0 &= \text{addi } r_0 \ 2 \\
    \text{ret } r_0 &= \text{ret } r_0
\end{align*}
\]
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?

- An attacker linking or injecting target code is not constrained by source constructs.
- FAC protects against these 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

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Why is FAC Secure?

- FAC protects against these attacks:
  1. confidentiality
  2. integrity
  3. invariant definition
  4. memory allocation
  5. well-bracketed control flow

Confidentiality:

\[ P_1 = P_2 \iff [P_1]^S_T = [P_2]^S_T \]

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

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesam et al.’11
Why is FAC Secure?

1. One.
2. Two.
3. Three.
4. Four.
5. Five.

confidentiality:

\[ P_1 = P_2 \iff [P_1]^S_T = [P_2]^S_T \]

• **P1** and **P2** have different secrets
• but they are equivalent
• \([P_1]^S_T\) and \([P_2]^S_T\) also have different secrets
• but they are equivalent

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Why is FAC Secure?

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

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Why is FAC Secure?

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

If the source has it.

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Why is FAC Secure?

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

• FAC preserves these properties

If the source has it.

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Not All That Glitters is Gold

- 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]
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
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
- Preserves hypersafety under certain conditions [Patrignani and Garg ’17]
Perspective on Foundations

Use Full Abstraction
(with precautions)
Perspective on Foundations

Invent new definitions

Use Full Abstraction (with precautions)
Perspective on Foundations

- Improve Full Abstraction
- Invent new definitions
- Use Full Abstraction (with precautions)
Perspective on Foundations

Invent new definitions
Perspective on Foundations

Invent new definitions

Ongoing work with:
Catalin Hritcu (INRIA)
Deepak Garg (MPI-SWS)
What More does Secure Compilation Offer?

- study language techniques for proofs
- implement secure compilers to new security architectures
What More does Secure Compilation Offer?

• study language techniques for proofs
• implement secure compilers to new security architectures
Programming Languages Techniques for Secure Compilation
• better proof techniques
Proving FAC

\[
P_1 \sim_{ctx} P_2
\]

\[
\llbracket P_1 \rrbracket^S_T \sim_{ctx} \llbracket P_2 \rrbracket^S_T
\]
Proving FAC

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

\[ [P_1]^S_T \triangleq_{\text{ctx}} [P_2]^S_T \]
Proving FAC

\[ P_1 \overset{\sim}{\approx}_{ctx} P_2 \]

\[ \downarrow \]

\[ \left[ P_1 \right]_T^S \overset{\sim}{\approx}_{ctx} \left[ P_2 \right]_T^S \]
Proving FAC

\[ \forall C. C[\left[ P1 \right]_T^S] \Downarrow \iff C[\left[ P2 \right]_T^S] \Downarrow \]
Proving FAC

\[ P_1 \sim_{ctx} P_2 \]

\[ [P_1]^S_T \pm [P_2]^S_T \]
Proving FAC

\[ P_1 \sim_{ctx} P_2 \]

Jagadeesan et al.'11,
Agten et al.'12,
Patrignani et al.'15'16,
Juglaret et al.'16
Proving FAC

\[ P_1 \sim_{ctx} P_2 \]

\[ \Downarrow \]

\[ [P_1]^S_T \sim [P_2]^S_T \]
Proving FAC

\[ P_1 \sim_{ctx} P_2 \]

Abadi et al.'00'01'02'
Bugliesi et al.'07
Adao et al.'06
Fournet et al.'13
Proving FAC

\[ P_1 \sim_{ctx} P_2 \]

\[ \Downarrow \]

\[ [P_1]^{S_T} \sim_n [P_2]^{S_T} \]
Proving FAC

\[ P_1 \simeq_{ctx} P_2 \]

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

\[ [P_1]^S_T \simeq_n [P_2]^S_T \]
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

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

approx. compiler security
Proving FAC with Logical Relations

\[ P_1 \approx_{ctx} P_2 \]

\[ C \left[ \llbracket P_1 \rrbracket^S_T \right] \Downarrow \_ \Rightarrow C \left[ \llbracket P_2 \rrbracket^S_T \right] \Downarrow \_
\]

approx. compiler security

\[ \llbracket P_1 \rrbracket^S_T \approx_{ctx} \llbracket P_2 \rrbracket^S_T \]

Benton et al. /zero.osf/nine.osf/one.osf/zero.osf
Hur et al. /one.osf/one.osf
Neis et al. /one.osf/five.osf

\[ C \left[ \llbracket P_1 \rrbracket^S_T \right] \Downarrow \_ \Rightarrow C \left[ \llbracket P_2 \rrbracket^S_T \right] \Downarrow \_
\]

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

\[ P_1 \sim_{ctx} P_2 \]

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

\[ P_1 \sim_\_ [P_1]^S_T \]

\[ C[[[P_1]^S_T]] \downarrow_n \Rightarrow C[[[P_2]^S_T]] \downarrow_\_ \]

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

\[ P_1 \sim_{ctx} P_2 \]

\[ \langle C \rangle_n[P_1] \downarrow_\_ \]

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

\[ P_1 \sim_\_ [P_1]^S_T \]  \hspace{1cm} (1)

\[ C[[P_1]^S_T] \downarrow_n \Rightarrow C[[P_2]^S_T] \downarrow_\_ \]

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

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

\[ \langle \text{C} \rangle_n[\text{P1}] \Downarrow \Rightarrow \langle \text{C} \rangle_n[\text{P2}] \Downarrow \quad (2) \]

\[ \langle \text{C} \rangle_n \sim_n \text{C} \]

\[ \text{P1} \sim_{\_} [\text{P1}]^S_T \quad (1) \]

\[ \text{C}[[[\text{P1}]^S_T]] \Downarrow_n \Rightarrow \text{C}[[[\text{P2}]^S_T]] \Downarrow_\_ \]

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

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

\[ \langle \langle \mathbf{C} \rangle \rangle_n[\mathbf{P1}] \downarrow \Rightarrow \langle \langle \mathbf{C} \rangle \rangle_n[\mathbf{P2}] \downarrow \quad (2) \]

\[ \langle \langle \mathbf{C} \rangle \rangle_n \sim_n \mathbf{C} \]

\[ \mathbf{P1} \sim_\_ [\mathbf{P1}]_S^T \]

\[ \langle \langle \mathbf{C} \rangle \rangle_n \sim_\_ \mathbf{C} \]

\[ \mathbf{P2} \sim_\_ [\mathbf{P2}]_S^T \]

\[ \mathbf{C}[\mathbf{P1}]_S^T \downarrow_n \Rightarrow \mathbf{C}[\mathbf{P2}]_S^T \downarrow \]

\[ [\mathbf{P1}]_S^T \sim_{ctx} [\mathbf{P2}]_S^T \]
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

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

\[ P_1 \sim_? \llbracket P_1 \rrbracket_T^S \]

\[ \llbracket C \llbracket \llbracket P_1 \rrbracket_T^S \rrbracket_T \downarrow_n \Rightarrow \llbracket C \llbracket \llbracket P_2 \rrbracket_T^S \rrbracket_T \downarrow_? \]

\[ \llbracket P_1 \rrbracket_T^S \sim_{ctx} \llbracket P_2 \rrbracket_T^S \]

\( \text{P1} \sim \llbracket P1 \rrbracket_T^S \) is obtained with standard techniques

Benton et al.'09'10
Hur et al.'11
Neis et al.'15

approx. compiler security
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

\[
\left[ P_1 \right]_T^S \sim_{ctx} \left[ P_2 \right]_T^S
\]
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

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

\[
\begin{align*}
\langle C \rangle_n \sim C & \text{ requires} \\
\text{• back-translation of terms} \\
\text{• reasoning at the type of} \\
\text{back-translated terms} \\
\text{• needed for all kinds of} \\
\text{back-translation} \\
\text{• needed for alternative criteria too}
\end{align*}
\]
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 alternative criteria too

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

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

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for alternative criteria too
Security Architectures for Secure Compilation
Security Architectures for SC
Security Architectures for SC

Security Architectures:

• ASLR [Abadi & Plotkin '/one.osf/zero.osf, Jagadeesan et al. '/one.osf/one.osf]
• Intel SGX-like enclaves [Agten et al. '/one.osf/two.osf, Patrignani et al. '/one.osf/three.osf,'/one.osf/six.osf]
• Typed Assembly Languages [Ahmed et al. '/one.osf/four.osf]
• Tagged Architectures (Pump) [Juglaret et al. '/one.osf/six.osf]
• Capability Machines [Tsampas et al. '/one.osf/seven.osf, WIP]
Security Architectures for SC

- ASLR
  - Abadi & Plotkin
  - Jagadeesan
- Intel SGX-like enclaves
  - Agten
  - Patrignani
- Typed Assembly Languages
  - Ahmed
- Tagged Architectures (Pump)
  - Juglaret
- Capability Machines
  - Tsampas
  - WIP

Reduced TCB

Efficiency

Security Architectures for SC

Security Architectures:

• ASLR [Abadi & Plotkin /one.osf/zero.osf, Jagadeesan et al.]

• Intel SGX-like enclaves [Agten et al. /one.osf/two.osf, Patrignani et al. /one.osf/three.osf, /one.osf/six.osf]

• Typed Assembly Languages [Ahmed et al. /one.osf/four.osf]

• Tagged Architectures (Pump) [Juglaret et al. /one.osf/six.osf]

• Capability Machines [Tsampas et al. /one.osf/seven.osf, WIP]

Reduced TCB Efficiency /two.osf/three.osf
Security Architectures for SC

Security Architectures:

• ASLR
  - Abadi & Plotkin et al. [1]
  - Jagadeesan et al. [2]

• Intel SGX-like enclaves
  - Agten et al. [3]
  - Patrignani et al. [4]

• Typed Assembly Languages
  - Ahmed et al. [5]

• Tagged Architectures (Pump)
  - Juglaret et al. [6]

• Capability Machines
  - Tsampas et al. [7]
  - WIP

Reduced TCB Efficiency
Security Architectures for SC

- ASLR
  - Abadi & Plotkin
  - Jagadeesan
- Intel SGX-like enclaves
  - Agten et al.
  - Patrignani et al.
- Typed Assembly Languages
  - Ahmed et al.
- Tagged Architectures (Pump)
  - Juglaret et al.
- Capability Machines
  - Tsampas et al.

Reduced TCB Efficiency
Security Architectures for SC

- ASLR
- Intel SGX-like enclaves
- Typed Assembly Languages
- Tagged Architectures (Pump)
- Capability Machines

Reduced TCB Efficiency
Security Architectures for SC

Security Architectures:

- ASLR
- Intel SGX-like enclaves
- Typed Assembly Languages
- Tagged Architectures (Pump)
- Capability Machines
Security Architectures for SC

Security Architectures:

• **ASLR**  [Abadi & Plotkin ’10, Jagadeesan et al.’11]
• **Intel SGX-like enclaves**  [Agten et al.’12, Patrignani et al.’13,’16]
• **Typed Assembly Languages**  [Ahmed et al.’14]
• **Tagged Architectures (Pump)**  [Juglaret et al.’16]
• **Capability Machines**  [Tsampas et al.’17, WIP]
Capability Machines: CHERI

Capability Machines

- Hardware support for fine-grained Capabilities
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities

Cheri (MIPS extension, FPGA) [Woodruff et al’/one.osf/two.osf/four.osf]

- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Capability Mantra:

subjects
perform operations
on objects
if they have rights

Mature: has a FreeBSD port /two.osf/four.osf
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
  - Cheri (MIPS extension, FPGA)
  - Tagged memory
  - Aligned memory
  - Capability registers file
  - Capability instructions

Capability Mantra:

- subjects
- perform operations
- on objects
- if they have rights

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
  - Cheri (MIPS extension, FPGA)
  - Tagged memory
  - Aligned memory
  - Capability registers file
  - Capability instructions

Capability Mantra:
- subjects
- perform operations on objects
- if they have rights

- instructions
- read/write
- execute

Mature: has a FreeBSD port
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities

   Cheri (MIPS extension, FPGA) [Woodruff et al.]

   Tagged memory

   Aligned memory

   Capability registers file

   Capability instructions

   A Cheri capability

   

   instructions

   read/ write/ execute

   address ranges

   Capability Mantra:

   subjects

   perform operations

   on objects

   if they have rights
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al’14]
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al.’14]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

A Cheri capability

- permissions
- ...
- length
- base address

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port
Capability Machines: Cheri

Capability Machines

• Hardware support for fine-grained Capabilities
• Cheri (MIPS extension, FPGA) [Woodruff et al’14]
• Tagged memory
• Aligned memory
• Capability registers file
• Capability instructions

Capability Mantra: subjects perform operations on objects if they have rights

instructions read/ write/ execute address ranges

A Cheri capability load rs rd cap

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) \cite{Woodruffetal14}
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al']
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Capability Mantra:
subjects
perform operations
on objects
if they have rights
instructions
read/ write/
execute
address
ranges
A Cheri capability
load rs rd cap

Unforgeable capabilities at the hardware level
Mature: has a FreeBSD port [’14]
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al', four.osf]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Capability Mantra:
subjects perform operations on objects if they have rights
instructions read/ write/ execute address ranges
A Cheri capability
load rs rd cap

Unforgeable capabilities at the hardware level
Mature: has a FreeBSD port
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al.][1]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Capability Mantra:

- subjects perform operations on objects if they have rights
- instructions read/ write/ execute
- address ranges

A Cheri capability:

```
load rs rd cap
```

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port [two.osf/four.osf]
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al.'14]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Unforgeable capabilities at the hardware level
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al’14]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Unforgeable capabilities at the hardware level
Mature: has a FreeBSD port
CM and Secure Compilation

• identify secure compartments
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: isolation
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: isolation
- capabilities regulate access to methods: public/private
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: isolation
- capabilities regulate access to methods: public/private
- capabilities regulate access to objects: shared/local
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: isolation
- capabilities regulate access to methods: public/private
- capabilities regulate access to objects: shared/local
- support dynamic security policies (runtime modification of accesses)
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data
- capabilities: isolation
  - capabilities regulate access to methods:
    - public/private
  - capabilities regulate access to objects:
    - shared/local
- support dynamic security policies
  (runtime modification of accesses)

More efficient than existing results

Support unprecedented security paradigms

Running! implemented by Tsampas
CM and Secure Compilation

• identify secure compartments
• wrap compiled code in code and data capabilities: isolation
  • capabilities regulate access to methods: public/private
  • capabilities regulate access to objects: shared/local
• support dynamic security policies (runtime modification of accesses)

More efficient than existing results
Support unprecedented security paradigms
CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: isolation
- capabilities regulate access to methods: public/private
- capabilities regulate access to objects: shared/local
- support dynamic security policies (runtime modification of accesses)

More efficient than existing results
Support unprecedented security paradigms
Running! implemented by Tsampas
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
- proof techniques for fully abstract compilation
Conclusion

• motivations for secure compilation
• secure compilation criterion: fully abstract compilation
• proof techniques for fully abstract compilation
• secure compilation to capability machines
Conclusion