Secure Compilation
an extensive introduction

Marco Patrignani
9th October 2017

so a language note i presume you all watch got do you watch it with subtitles? turn the m off, do yourselves a favour

this is to say, that if you don’t understand stuff or if you have question just ask – even in italian
Secure Compilation

What is Secure Compilation?

Secure Compilation Criteria

Programming Languages Techniques for Secure Compilation

Security Architectures for Secure Compilation
What is Secure Compilation?

Secure Compilation
Compilation

Secure Compilation

2017-10-09

Compilation

buffer overflow
privilege escalation
code injection
τ {P} e {Q}
Mon
goto
PMA
ASLR
CM
PUMP
/six.osf/
two.osf/
zero.osf/
one.osf/
seven.osf/
-one.osf/
-zero.osf/
nine.osf
Compilation

Secure Compilation

2017-10-09

Compilation

source S

buffer overflow
privilege escalation
code injection

\{P\}
\{Q\}

Mon
goto

PMA
ASLR
CM
PUMP

/six.osf
/one.osf/seven.osf/
/one.osf/zero.osf/
/zero.osf/nine.osf

Compilation
Secure Compilation

Compilation

your brains are currently translating from english to italian
Compilation

- Source: $S$
- Target: $T$
- $\left[ \cdot \right]^S_T$

Secure Compilation

- Correct Compilation

Compilation

- Source $S$
- Target $T$

2017-10-09

- buffer overflow
- privilege escalation
- code injection

τ

{P}

e

{Q}

Mon

goto

PMA

ASLR

CM

PUMP
Correct Compilation

\[ [\cdot]_S^T \]

Secure Compilation

2017-10-09

Correct Compilation

buffer overflow
privilege escalation
code injection

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

Mon

go

PMA
ASLR
CM

PUMP

/six.osf/
two.osf/zero.osf/one.osf/seven.osf/-/one.osf/zero.osf/-/zero.osf/nine.osf

Secure Compilation

Correct Compilation
Correct Compilation

$\left[ \cdot \right]_S^T$

Secure Compilation

2017-10-09

Correct Compilation

buffer overflow
privilege escalation
code injection

τ {P} e {Q}

Mon

goto

PMA

ASLR

CM

PUMP

/six.osf

two.osf

/zero.osf

/one.osf

/seven.osf

/one.osf

/zero.osf

/zero.osf

/nine.osf
Secure Compilation

Secure Compilation

buffer overflow
privilege escalation
code injection

exploits: buffer overflow, privilege escalation
Secure Compilation

- Buffer overflow
- Privilege escalation
- Code injection

Mon

\([P,e,Q]_T[\cdot]^S_T\)

2017-10-09

Secure Compilation
Secure Compilation

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

Mon

goto

buffer overflow

privilege escalation

code injection

2017-10-09

Secure Compilation

Secure Compilation

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

Mon

goto

code injection

privilege escalation

buffer overflow

Secure Compilation
Secure Compilation

{P}e{Q}

Mon

[ \cdot ]^T$
goto

{P}e{Q}

Mon

2017-10-09

Secure Compilation

Secure Compilation

PMA

ASLR

CM

PUMP

buffer Overflow

privilege escalation

code injection

Mon
Memory Allocation Issues

Patrignani et al.'15'16

Java-like

O1
O2

Ext1
Ext2

Asm-like

[O1]^S
[O2]^S

Ext1
Ext2

Secure Compilation

2017-10-09

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 type checks

Isolated memory regions e.g., SGX enclaves

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

Patrignani et al.'15'16

Java-like

O1
O2
O4
O3

Ext1
Ext2

Asm-like

[O1]S
[O2]S

[O1]S
[O2]S

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

Secure Compilation

Memory Allocation Issues

Patrignani et al.'15'16

2017-10-09

...or?
Memory Allocation Issues

Patrignani et al.'15'16

O1 ∶ Account
O2
O4
O3
Java-like
Ext1
Ext2

return O3

Asm-like
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

Secure Compilation

2017-10-09

Memory Allocation Issues Patrignani et al.'15'16

O1
O2
O4
O3
Java-like
return O3

Ext1 Ext2

Asm-like

[O1]_S
[O2]_T
Ext1
Ext2
Secure Compilation

Memory Allocation Issues

Patrignani et al.’15’16

Java-like

Asm-like

O1
O2
O4
O3

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()

createAccount()
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

Java-like

O1
O2
O4
O3

Asm-like

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

return

[O3]S

Securable Compilation

2017-10-09

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
Memory Allocation Issues

Java-like

O1
O2
O4
O3

Ext1
Ext2

return 0x00C

Asm-like

0x001
0x005
0x009
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
Memory Allocation Issues

Patrignani et al.’15’16

Secure Compilation

2017-10-09

Isolated memory regions
e.g., SGX enclaves

• design
• implement
• ...or?

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

Issue: type violation
Solution: add dynamic typechecks

...
Memory Allocation Issues

Patrignani et al.’15’16

Secure Compilation

2017-10-09

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

Issue: type violation
Solution: add dynamic type checks

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

Patrignani et al.'15'16

Secure Compilation

● 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?
Issue: Oid guessing

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

Java-like

O1
O2
O4
O3

Asm-like

0x001 \rightarrow 1
0x005 \rightarrow 2

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
• design
• implement
• ...or?

Secure Compilation

2017-10-09
Memory Allocation Issues

---

**Java-like**

- `O1`
- `O2`
- `O4`
- `O3`

---

**Asm-like**

- `0x001 \rightarrow 1`
- `0x005 \rightarrow 2`

---

**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?

---

Secure Compilation

---

```
O1
O2
O4
O3
```

---

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

---

```
0x001 \rightarrow 1
0x005 \rightarrow 2
```

---

```
Ext1
Ext2
```

---

```
Ext1
Ext2
```

---
Memory Allocation Issues

Patrignani et al.'15'16

O1
O2
O4
O3

Java-like

Ext1
Ext2

Asm-like

0x001 \rightarrow 1
0x005 \rightarrow 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

Secure Compilation

2017-10-09
Secure Compilation

Memory Allocation Issues

Java-like

```
O1
O2
O4
O3
```

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

Ext1
Ext2

Asm-like

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

Ext1
Ext2

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

Patrickini et al.'15'16

Java-like

O1
O2
O4
O3

Asm-like

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

\[ \text{return 3} \]

Ext1
Ext2

0x001 \rightarrow O1
0x005 \rightarrow O2
0x009
0x00C \rightarrow O3

Ext1
Ext2

Secure Compilation

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

\[ \text{...or?} \]
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?

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

O1
O2
O4
O3

Ext1
Ext2

O1
O2
O4
O3

Ext1
Ext2

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Ext1
Ext2

0x001 ↔ 1
0x005 ↔ 2
0x009
0x00C ↔ 3

Ext1
Ext2
Memory Allocation Issues

O1: Account
O2: Pair
O4
O3

Java-like

Ext1
Ext2

1. createAccount()
return

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

Asm-like

Ext1
Ext2

1. createAccount()
return 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?

Secure Compilation

O1 Account
O2: Pair
O4
O3
Memory Allocation Issues

O1: Account
O2: Pair
O4
O3

Java-like

Ext1
Ext2

createAccount()

Asm-like

0x001 ➔ 1
0x005 ➔ 2
0x009
0x00C ➔ 3

Secure Compilation

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

O1: Account
O2: Pair
O3
O4

Java-like

Ext1
createAccount() Ext2

Asm-like

1.createAccount() return 3

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

Issue: type violation
Solution: add dynamic type checks

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

Asm-like

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

Java-like

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

Secure Compilation

...
Memory Allocation Issues

Java-like

O1: Account
O2: Pair
O4
O3

Java-like

O1: Account
O2: Pair
O4
O3

Asm-like

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

Secure Compilation

O1 Account
O2: Pair
O4
O3

0x001 ⇒ 1
0x005 ⇒ 2
0x009
0x00C ⇒ 3

2. createAccount()
Issue: type violation

Solution: add dynamic type checks
Memory Allocation Issues

O1: Account
O2: Pair
O4
O3

Java-like

Ext1
Ext2

Asm-like

Ext1
Ext2

Secure Compilation

O1 Account
O2: Pair
O4
O3

Java-like

Ext1
Ext2

Asm-like

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

\[ \text{createAccount()} \]

1.createAccount()
return 3

2.createAccount()
Memory Allocation Issues

Patrignani et al.'15'16

Java-like

O1: Account
O2: Pair
O4
O3

Ext1
Ext2

Asm-like

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

Secure Compilation

2017-10-09

Ext1 Ext2

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?
Isolated memory regions
e.g., SGX enclaves

efficiency issues are the price to pay
Memory Allocation Issues

- design
- implement

Secure Compilation

- design
- implement

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
Memory Allocation Issues

- design
- implement

...or?

Secure Compilation

- design
- implement
- ...or?
Guarantees

- How do we know we are right?

Secure Compilation

- Guarantees
Guarantees

- How do we know we are right?
- How can we know that $J \cdot K_S T$ is secure?

Prove $J \cdot K_S T$ to attain a secure compilation criterion
Show the security implications of the criterion

8
Guarantees

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

• How do we know we are right?
• How can we know that $J \cdot K$ is secure?
• What do we mean with secure?
Guarantees

Show the security implications of the criterion
Secure Compilation Criteria
Secure compilation is a broad field that received contributions from many researchers, many of which are in this room today. Around the end of the 90's came a line of works that contributed greatly to the idea we have nowadays of Secure Compilation.
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)

Many works using FA have been made and FA has also been studied by Parrow that tells us when it's possible.
Why does Fully Abstract Compilation entail security?

Secure Compilation

Fully Abstract Compilation Influence
Why does Fully Abstract Compilation entail security?
FAC ensures that a target-level attacker has the same power of a source-level one.
Compiler Full Abstraction

x = 1;     x = 0;
x += 2;    x += 2;
x  x

1. let's look at FA, informaly
x = 1;
++;

x = 0;

x = x + 2;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;

x = x + 1;
Compiler Full Abstraction

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

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

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

\[ \uparrow \quad \uparrow \]

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

Secure Compilation

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

\[ \downarrow \quad \downarrow \]

\[ \text{loadi } r_0 \quad 1 \quad \text{loadi } r_0 \quad 0 \]
\[ \text{inc } r_0 \quad = \quad \text{addi } r_0 \quad 2 \]
\[ \text{ret } r_0 \quad \text{ret } r_0 \]
1. note that the observers have different powers! the target one is not subject to what happens in the source, so if S has types, he’s not subject to them if the language allows him to, he could also jump in mid code! but if FA holds, either those bad things cannot arise/are prevented or they are also doable in the source
Why is FAC Secure?

- An attacker linking or injecting target code

Secure Compilation

1. The difference in observers power is at the root of the reasons why FA is a good def for SC
2. 2 reasons: what attacks are prevented, what properties are secured
Why is FAC Secure?

- is an attacker linking or injecting target code

Secure Compilation

1. First reason is: what is FA protecting from target level attacker
Why is FAC Secure?

- is an attacker linking or injecting target code
- is not constrained by source constructs

Secure Compilation

1. First reason is: what is FA protecting from target level attacker
Why is FAC Secure?

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

Secure Compilation

1. First reason is: what is FA protecting from target level attacker
Why is FAC Secure?

- An attacker linking or injecting target code
- Is not constrained by source constructs
- The co-implied equalities reduce to

• FAC protects against these attacks

FAC preserves these properties
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, 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
- \([P_1]^S_T \) and \([P_2]^S_T \) also have different secrets
- but they are equivalent

Why is FAC Secure?

Secure Compilation

- Why is FAC Secure?

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
- \([P_1]^S T\) and \([P_2]^S T\) also have different secrets
- but they are equivalent
- so the secret does not leak

Secure Compilation

Why is FAC Secure?

- \( P_1 \) 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

If the source has it.

Agten et al.’12, Abadi and Plotkin ’10, Jagadeesan et al.’11
Secure Compilation
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 '/one.osf/seven.osf]

1. Is that it? Well, no, fa has a number of shortcomings, unfortunately, which we started unveiling only recently.
Not All That Glitters is Gold

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

Secure Compilation

- Not All That Glitters is Gold

1. first off, when studied in an untyped target setting, it enforces nothing about modularity
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]

Secure Compilation

- No support for separate compilation
  [Patrignani et al.'16, Juglaret et al.'16]
- No support for undefined behaviour [Juglaret et al.'16]

1. secondly, it does not support languages with undefined behaviour
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

Secure Compilation

- Not All That Glitters is Gold

1. Some security properties are not upheld by it for example a simple declassification policy ... though someone may say that it’s a matter of changing the kind of equivalence to be preserved
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]

Secure Compilation

- 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]

1. Finally, the list of security properties it preserves is obtained by example, it is not clear if there should be more to that list, or what is definitively not there. For this, we have a paper under submission that defines what (a specific form of) fa preserves in terms of classes of hyperproperties, but I’ll not go into details here.
1. So: what now? This has potential. We can keep using FA as we’ve done and in similar settings: it’s fine!
Perspective on Foundations

Use Full Abstraction (with precautions)

Invent new definitions

Secure Compilation

1. So: what now? this has potential. we can keep using FA as we’ve done and in similar settings: it’s fine!
Perspective on Foundations

1. some of us are discussing more specific statements of SC just tailored to specific properties of interest
Invent new definitions

1. some of us have already started looking into this: catalin’s SCC, our TPC, though they all show a tight connection with fa
Perspective on Foundations

Use Full Abstraction (with precautions)

Invent new definitions

Ongoing work with:
Catalin Hritcu (INRIA)

Deepak Garg (MPI-SWS)

Secure Compilation

1. FA is not bad, but it can be improved: that’s what we’re after when we look at the foundations of this field
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
as we said, sc sits at the verge of PL and security. from the PL perspective, what is there to be challenged for SC?
In this part of the talk, i’ll focus on proof techniques for proving FA and the reason is that other definitions that we’ve seen also need to capture the additional power of target-level programs/contexts and reason about that. while the "how" it shows up in the definition may differ, most notions of secure


What PL Want

- better proof techniques
1. To prove FA, we need to prove the 2 parts of its coimplication. If you don’t see the observer don’t worry, it’s buried in here.
Proving FAC

\[ P_1 \approx_{ctx} P_2 \]

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

Secure Compilation

1. \( \leq \) is easy, it follows from compiler correctness
Secure Compilation

1. \( \Rightarrow \) is the hard one. Let us now look a bit more in detail why is this difficult and how can we do this.
1. well, here’s the deal: we need to prove something over all target contexts, which is a notoriously complex feat to attain so we do what any sensible person would do, replace $ceqT$ with something else.
Secure Compilation

1. and now there are 2 schools of thought
   - trace equivalence
   - logical relations
1. and now there are 2 schools of thought

- trace equivalence
- logical relations

Jagadeesan et al.,'11,
Agten et al.,'12,
Patrignani et al.,'15-16,
Juglaret et al.,'16,
Abadi et al.,'zero.osf/zero.osf'/zero.osf/one.osf'/zero.osf/two.osf',
Bugliesi et al.,'zero.osf/seven.osf',
Adao et al.,'zero.osf/six.osf',
Fournet et al.,'one.osf/three.osf',
Ahmed et al.,'/eight.osf'/one.osf/one.osf'/one.osf/four.osf'/one.osf/five.osf'/one.osf/six.osf'/one.osf/seven.osf',
Devriese et al.,'/one.osf/six.osf'/two.osf/zero.osf/one.osf/seven.osf'/one.osf/zero.osf'/zero.osf/nine.osf
\[ P_1 \approx_{ctx} P_2 \]

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

\[ P_1 \approx_{ctx} P_2 \]

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

Secure Compilation

Proving FAC

\[ P_1 \approx_{ctx} P_2 \]

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

Ahmed et al. 8'14'15'16'17,
Devriese et al. 16

桔adeesan et al. '/one.osf/one.osf,
Agten et al. '/one.osf/two.osf,
Patrignani et al. '/one.osf/five.osf'/one.osf/six.osf,
Juglaret et al. '/one.osf/six.osf
Abadi et al. '/zero.osf/zero.osf'/zero.osf/one.osf'/zero.osf/two.osf'
Bugliesi et al. '/zero.osf/seven.osf
Adao et al. '/zero.osf/six.osf
Fournet et al. '/one.osf/three.osf
Ahmed et al. '/eight.osf'/one.osf/one.osf'/one.osf/four.osf'/one.osf/five.osf'/one.osf/six.osf'/one.osf/seven.osf,
Devriese et al. '/one.osf/six.osf

Secure Compilation

Proving FAC
Proving FAC with Logical Relations

\[ P_1 \sim_{ctx} P_2 \]

Secure Compilation

1. Logical relations: what we need to prove in this case is
   start from \( P_1 \), relate it and its context to source counterparts
   obtain that \( P_2 \) terminates in the source
   relate this to \( P_2 \) terminating in the target
Proving FAC with Logical Relations

$P_1 \simeq_{ctx} P_2$

1. Logical relations: what we need to prove in this case is
   start from $P_1$, relate it and its context to source counterparts
   obtain that $P_2$ terminates in the source
   relate this to $P_2$ terminating in the target
Proving FAC with Logical Relations

\[ \langle C \rangle \downarrow \Rightarrow C \]

\( P_1 \sim [P_1]^s \)
\( \downarrow \Rightarrow C \]
\( C \]
\( \downarrow \Rightarrow C \]
\( [P_2]^s \)
\( P_2 \sim [P_2]^s \)

Approx. compiler security

Proving FAC with Logical Relations

Benton et al.
Hur et al.
Neis et al.

\[ P_1 \sim [P_1]^s \]
\( \downarrow \Rightarrow C \]
\( C \]
\( \downarrow \Rightarrow C \]
\( [P_2]^s \)
\( P_2 \sim [P_2]^s \)

1. Logical relations: what we need to prove in this case
   - start from \( P_1 \), relate it and its context to source counterparts
   - obtain that \( P_2 \) terminates in the source
   - relate this to \( P_2 \) terminating in the target

approx. compiler security
Proving FAC with Logical Relations

1. Logical relations: what we need to prove in this case

\[
\frac{\frac{1}{S}\left[\Pi_2\right]^{\Gamma_P}}{\frac{1}{S}\left[\Pi_1\right]^{\Gamma_P}} \nvdash \frac{1}{S}\left[\Pi_1\right]^{\Gamma_P} \quad \iff \quad \frac{1}{S}\left[\Pi_1\right]^{\Gamma_P} \nvdash \frac{1}{S}\left[\Pi_2\right]^{\Gamma_P}
\]

(approx. compiler security)

\[\ldots\]

\[
\frac{\frac{1}{S}\left[\Pi_2\right]^{\Gamma_P}}{\frac{1}{S}\left[\Pi_1\right]^{\Gamma_P}} \nvdash \frac{1}{S}\left[\Pi_1\right]^{\Gamma_P} \quad \iff \quad \frac{1}{S}\left[\Pi_1\right]^{\Gamma_P} \nvdash \frac{1}{S}\left[\Pi_2\right]^{\Gamma_P}
\]

(approx. compiler security)
Proving FAC with Logical Relations

1. Logical relations: what we need to prove in this case is
   start from P1, relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target
Proving FAC with Logical Relations

P1 \sim_{ctx} P2

\langle C \rangle_n[P1] \Downarrow \implies \langle C \rangle_n[P2] \Downarrow

\langle C \rangle_n \sim_n C

P1 \sim [P1]^S_T

\langle C \rangle_n \sim \langle C \rangle_n

P2 \sim [P2]^S_T

C[[P1]^S_T] \Downarrow \implies C[[P2]^S_T] \Downarrow

[P1]^S_T \sim_{ctx} [P2]^S_T

1. Logical relations: what we need to prove in this case is
   start from P1, relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target
Proving FAC with Logical Relations

P1 $\sim_{ctx}$ P2

$\langle C \rangle [P1]^{S} \sim_{n} [P2]^{S}$

C$[P1]^{S}$ $\downarrow_{n}$ $\Rightarrow$ C$[P2]^{S}$ $\downarrow$

$[P1]^{S}_{T} \sim_{ctx} [P2]^{S}_{T}$

$\langle C \rangle [P1]^{S}$ $\downarrow_{n}$ $\Rightarrow$ $C[\langle P2 \rangle^{S}]_{T}$ $\downarrow$

$\langle C \rangle [P1]^{S} \sim_{n} C[\langle P2 \rangle^{S}]_{T}$

$\langle C \rangle [P1]^{S}$ $\downarrow_{n}$ $\Rightarrow$ $C[\langle P2 \rangle^{S}]_{T}$ $\downarrow$

Secure Compilation

1. so we need to define a relation for normal terms for relating programs

approx. compiler security

P1 $\sim [P1]^{S}$ is obtained with standard techniques
Benton et al.’09’10
Hur et al.’11
Neis et al.’15
Proving FAC with Logical Relations

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

requires

• back-translation of terms

• reasoning at the type of back-translated terms

Secure Compilation

1. need to define a pseudo type and a relation at that pseudotype for back-translated terms
Secure Compilation

1. for example, we proved a fa compiler from stlc+fix to ulc

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

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

Approx. compiler security

Secure Compilation

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

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

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

\[ [P1]^S_T \simeq_{ctx} [P2]^S_T \]
Security Architectures for Secure Compilation
Security Architectures for SC

Secure Compilation

Security Architectures for SC
Security Architectures for SC

1. albeit formally, we devise compilers and we want them to be written and used. so let’s change perspective and after looking at how to ensure that these compilers are secure, let’ look at how to write them.
Security Architectures for SC

1. a large number of SC results have been made possible due to the developments of the security community

SGX Enclaves (aka PMA), ASLR, TAL, PUMP. Cheri are all examples of security architectures which support secure compilation and whose development allowed us to devise SC for them.
Secure Compilation

1. A large number of SC results have been made possible due to the developments of the security community.

SGX Enclaves (aka PMA), ASLR, TAL, PUMP. Cheri are all examples of security architectures which support secure compilation and whose development allowed us to devise SC for them.
Security Architectures for SC

Secure Compilation

- Security Architectures for SC

1. sec arch are needed because we simply can’t SC to our pcs without having to trust huge codebases so we rely on special machines that give us additional security primitives for the compiler to rely on
   - isolation / randomisation / types / tags / capabilities
   these let us devise secure compilers that have a minimal TCB, most times not including the OS, and
Security Architectures for SC

- ASLR [Abadi & Plotkin, Jagadeesan et al.]
- 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

Secure Compilation

Security Architectures for SC
Security Architectures for SC

- 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

Secure Compilation

Security Architectures for SC
Security Architectures for SC

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

Secure Compilation

Security Architectures for SC
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]

Reduced TCB
Efficiency

Secure Compilation

2017-10-09
Secure Compilation

Capability Machines: Cheri

1. now i want to discuss some of the most recent and interesting of these sec arch: cap machs cap machs bring the capability principle to hardware, so they are directly usable at the assembly level
Capability Machines: Cheri

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

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

A Cheri capability load rs rd cap
Unforgeable capabilities at the hardware level
Mature: has a FreeBSD port /
two.osf/four.osf

Secure Compilation

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation
in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses
so to load a word you need cap to R it,
to jump to an address you need cap to X it
Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al/one.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

Secure Compilation

- Capability Machines: Cheri

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,

to jump to an address you need cap to X it
Capability Machines: Cheri

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

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

Capabilities: subjects perform operations on objects if they have the right to do it. No right -> no operation.

In this case, subj = instructions, operations = r/w/x, objs are memory areas/addresses.

So to load a word you need cap to R it,

to jump to an address you need cap to X it.

Secure Compilation
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
- address ranges

A Cheri capability:
load rs rd cap

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port

Secure Compilation

capabilities: subjects perform operations on objects if they have the right to do it; no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,
to jump to an address you need cap to X it
Capability Machines: Cheri

Capability Machines

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

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap

Secure Compilation
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

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

Secure Compilation

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap
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

Secure Compilation

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap
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

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al'/one.osf/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

Secure Compilation

2017-10-09
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

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port

Secure Compilation

2017-10-09
Capability Machines: Cheri

- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al’/one.osf/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

Secure Compilation
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

Secure Compilation

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

Mature: has a FreeBSD port

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

Instructions: read/write/execute address ranges

A Cheri capability: load rs rd cap
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

Secure Compilation

- Capability Machines: Cheri
CM and Secure Compilation

- identify secure compartments

Secure Compilation

- CM and Secure Compilation

2017-10-09

• identify secure compartments
CM and Secure Compilation

• identify secure compartments
• wrap compiled code in code and data capabilities: isolation

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

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

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)
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/two.osf/five.osf
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

Implementation 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
Running! implemented by Tsampas
• motivations for secure compilation
• 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

Secure Compilation

—Conclusion

2017-10-09