What is Secure Compilation?

summer semester 18-19, block

Marco Patrignani\textsuperscript{1,2}
Practicalities

- Monday, Tuesday, Wednesday, Friday, Monday, Tuesday

Type of course: lectures, presentations

Course goal:
- Understand background and motivation behind SC
- Learn reasoning techniques for SC
- Know the most recent developments in SC

Evaluation: presentations, reports.
Practicalities

• Monday, Tuesday, Wednesday, Friday, Monday, Tuesday
• 4h30, 2 breaks remind me
Practicalities

- Monday, Tuesday, Wednesday, Friday, Monday, Tuesday
- 4h30, 2 breaks remind me
- Type of course: lectures + presentations
Practicalities

- Monday, Tuesday, Wednesday, Friday, Monday, Tuesday
- 4h30, 2 breaks remind me
- Type of course: lectures + presentations
- Course goal:
  - understand background and motivation behind SC
  - learn reasoning techniques for SC
  - know the most recent developments in SC
Practicalities

- Monday, Tuesday, Wednesday, Friday, Monday, Tuesday
- 4h30, 2 breaks remind me
- Type of course: lectures + presentations
- Course goal:
  - understand background and motivation behind SC
  - learn reasoning techniques for SC
  - know the most recent developments in SC
- Evaluation: presentations, reports.
• SC is a very active research field with many unsolved difficult problems to work on
Practicalities

• SC is a very active research field with many unsolved difficult problems to work on (for some questions there is no answer yet)
Practicalities

• SC is a very active research field with many unsolved difficult problems to work on (for some questions there is no answer yet)

• Pose questions
Practicalities

- SC is a very active research field with many unsolved difficult problems to work on (for some questions there is no answer yet)
- Pose questions
- Course flavour: formal methods.
Practicalities

• SC is a very active research field with many unsolved difficult problems to work on (for some questions there is no answer yet)
• Pose questions
• Course flavour: formal methods.
• You think how to bridge the gap between formality and practicality
A Note on Flavour

Formal methods give you the tools to reason about things and to reason about the motivation why things are done in a certain way.
• Develop a super toy formal compiler
Couse Outline

- Develop a super toy formal compiler
- Prove it is correct, understand why it is not secure
Course Outline

- Develop a super toy formal compiler
- Prove it is correct, understand why it is not secure
- Prove that it is Fully Abstract via Backtranslations
Couse Outline

- Develop a super toy formal compiler
- **Prove** it is correct, understand why it is not secure
- Prove that it is **Fully Abstract** via Backtranslations
- Understand why Full Abstraction **yields** security
Course Outline

- Develop a super toy formal compiler
- Prove it is correct, understand why it is not secure
- Prove that it is Fully Abstract via Backtranslations
- Understand why Full Abstraction yields security
- Prove that it is Robustly Safe
Course Outline

- Develop a super toy formal compiler
- Prove it is correct, understand why it is not secure
- Prove that it is Fully Abstract via Backtranslations
- Understand why Full Abstraction yields security
- Prove that it is Robustly Safe
- Understand why Robust Compilation yields security
Problems

• Programming *abstractions* are not preserved by compilers (linkers etc) (security is an abstraction)
• Programming **abstractions** are not preserved by compilers (linkers etc) (security is an abstraction)
• what does preserving abstractions mean?
Problems

• Programming abstractions are not preserved by compilers (linkers etc) (security is an abstraction)

• what does preserving abstractions mean?

• what tools are there to preserve abstractions?
Solutions

• Study what preserving abstractions means via secure compilation criteria
Solutions

• Study what preserving abstractions means via secure compilation criteria
• Devise efficient enforcement mechanisms to attain security
Solutions

• Study what preserving abstractions means via secure compilation criteria
• Devise efficient enforcement mechanisms to attain security
• Prove compilers can use these mechanisms for security
Recommended Reading

- http://drops.dagstuhl.de/opus/volltexte/2018/
- https://blog.sigplan.org/2019/07/01/secure-compilation/
A First Example

(borrowed from Catalin Hritcu)
HACL* verified cryptographic library

~100,000 LOC in F*

HACL* library
HACL* verified cryptographic library, in practice

- HACL* library: ~100,000 LOC in F*
- Firefox web browser: 16,000,000+ LOC in C/C++

160x
HACL* verified cryptographic library, in practice

HACL* library

~100,000 LOC in F*

KreMLin + CompCert

ASM

16,000,000+ LOC in C/C++

Firefox web browser

GCC

ASM

160x
HACL* verified cryptographic library, in practice

Insecure interoperability: linked code can read and write data and code, jump to arbitrary instructions, smash the stack, ...
A Second Example

Rust

Asm

P₁
P₂
... ᴘn

[P₁]
[P₂]
... [Pn]
P'
A Second Example

Rust

Asm

\[ y = \&\text{mut} \ P_1 \]

\[ \begin{array}{c}
P_2 \\
\ldots \\
P_n \\
\end{array} \]

\[ \begin{array}{c}
P \\\n[P_1] \\
[P_2] \\
\ldots \\
[P_n] \\
P' \\
\end{array} \]
A Second Example

Rust

```rust
y = &mut P_1
```

used linearly

```
P_2
...
```

P_n

Asm

```
P
[P_1]
P_2
...
P_n
```

P'
A Second Example

\[
y = \& \text{mut} \\
P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n
\]

Rust

Asm

\[
[y = \& \text{mut}]
\]

\[
[P_1] \rightarrow [P_2] \rightarrow \ldots \rightarrow [P_n] \rightarrow P'
\]
A Second Example

\[ y = \&\text{mut} \]

\[ P_1 \quad P_2 \quad \ldots \quad P_n \]

Rust

Asm

\[ \text{violate linearity} \]
A Second Example

Preserve the security properties of

\[ y = &\text{mut} \]

\[ P_1 \]

\[ P_2 \]

\[ \ldots \]

\[ P_n \]

Rust

Asm

\[
\begin{bmatrix}
y = &\text{mut} \\
[P_1]
\end{bmatrix}
\]

\[
\begin{bmatrix}
[P_2]
\end{bmatrix}
\]

\[ \ldots \]

\[
\begin{bmatrix}
P_n
\end{bmatrix}
\]

\[ P' \]
A Second Example

Preserve the security properties of

\[
y = \&\text{mut } P_1 \rightarrow P_2 \rightarrow \ldots \rightarrow P_n
\]

Rust

Asm

\[
[y = \&\text{mut}]
\]

\[
[P_1] \rightarrow [P_2] \rightarrow \ldots \rightarrow [P_n]
\]

when interoperating with

P \rightarrow [P_1] \rightarrow [P_2] \rightarrow \ldots \rightarrow [P_n] \rightarrow P'
A Second Example

Preserve the security properties of when interoperating with PL sec (e.g., no side channels)
A Second Example

Correct compilation

\[
y = \&\text{mut} \quad P_1 \quad P_2 \quad \ldots \quad P_n
\]

Rust

Asm

\[
[y = \&\text{mut}] \\
[P_1] \\
[P_2] \\
\ldots \\
[P_n] \\
\]

P

P'

\[\]
A Second Example

Correct compilation

Rust

Asm

respect linearity
A Second Example

Secure compilation

Rust

Asm
Enable source-level security reasoning

Rust

Asm

\[ y = \&\text{mut} \]

\[ P_1 \]

\[ P_2 \]

\[ \ldots \]

\[ P_n \]

\[ P \]

\[ [y = \&\text{mut}] \]

\[ [P_1] \]

\[ [P_2] \]

\[ \ldots \]

\[ [P_n] \]

\[ P' \]