Secure Compilation as Hyperproperties Preservation

Marco Patrignani Deepak Garg

Max Planck Institute for Software Systems

MAX-PLANCK-GESELLSCHAFT

The problem

What is a secure compiler?

what do you mean by secure?

I don't know

A Declassification Example

Language **S** has Bools

Language **T** has Nats

[true] = 1
[false] = 0

[.] : **S** -> **T**

[.] is a Compiler from **S** to **T**

λ x.
if interactions < 10
then x
else secret

[.]

λ x.
if x >1 **then** secret
elif interactions <10
then x
else secret

Is [.] secure?

no (but it is **FAC**!!)

A 2nd Example

What if the compiler produces this:

λ x. **if** x >1 **then** *fail*
elif interactions <10 **then** x **else** secret

Is [.] secure now?

yes

The Definition

T race
P reserving
C ompiler

Behaviour in **S** and **T** must be expressed as traces

- on "good" inputs, act "like" **S**

  so **TPC** implies compiler *correctness*!

- on "bad" ones, act *opaquely* (*fail*)

  fail must **not** be expressible in **S**

Why is this secure?

Hyperproperties

Clarkson and Schneider '10

formalise any program property

Interesting subclass: *hypersafety*: "something bad does not happen"

How does this relate to compilation?

First result

TPC preserves *hypersafety*!

If a **S** program has some hypersafety then its compilation also has it.

With a suitable hypersafety translation

We know the precise security implications of **TPC**!

What about related work?

like fully-abstract compilers

Second result

Most works in compiler security prove compiler full abstraction **FAC**

Abadi '00

T **P** **C** — any **TPC** is **FAC** → **F** **A** **C**

← Any **FAC** is **TPC** if an extra assumption holds

All existing **FAC** works uphold it!

We know the precise security implications of **FAC**!