Provably Secure Compilers

Matteo Busi

Dipartimento di Informatica, Università di Pisa

matteo.busi@di.unipi.it - pages.di.unipi.it/busi

Today's agenda

- Goals & Motivations
- Correctness...isn't it enough?
- Notions of security
- What about **compiler** security?
    - Full abstraction
    - Other notions
- Alternative approaches
    - Secure translation validation
    - Hardware-based solutions

Compiler security?

(Roughly) the goal:
Show that a given compiler $[\![\cdot]\!]$ preserves the security properties of the source programs.

- Is this even relevant?
- Is this a real-world thing?

Indeed ...

```
pin := read_secret();
if (check(pin))
   // OK!

pin := 0; // overwrite the pin
```

↓

```
pin := read_secret();
if (check(pin))
   // OK!
```

Does the optimization:

- preserves the semantics?
- what about security?

Notions of security

- Fundamental question:
  - how would you define security of a **program**?
- At least two ways:
  - trace properties
  - hyperproperties

Security: trace properties

- Monday's refresher:
  - Observables in $\mathcal{O}$
  - Behaviour of p as $\mathcal{B}(p) \subseteq \mathcal{O}^* \cup \mathcal{O}^\omega$
- Now:
  - $\mathcal{B}(p)$ is a set of traces
  - A **trace property** is $P \in Prop \triangleq \wp(\mathcal{O}^\omega)$
    - p satisfies a trace property ($p \vDash P$) *iff* $\mathcal{B}(p) \subseteq P$

Security: trace properties (cont.)

We can identify two relevant classes of trace properties:

- **Safety properties:** roughly that something bad will never happen
    - e.g. *Chinese-wall policy:* "a program never writes to the network after having read from a file."
- **Liveness properties:** roughly that something good will eventually happen
    - e.g. *Guaranteed service:* "every request is eventually satisfied."

Security: trace properties (cont.)

Why are they nice?

- Nice properties:
  - **Theorem:** $\forall P \in \mathit{Prop}.\, (\exists S \in \mathit{Safety}, L \in \mathit{Liveness}.\, P = S \cap L)$
- Relevant for security
- (😉 We pretend that they are) easy to understand

Correctness and trace properties

Recall *refinement* as seen on monday:

$$[\![\cdot]\!] \text{ is correct if } \forall s \in S.\, \mathcal{B}(s) \supseteq \mathcal{B}([\![s]\!])$$

Refinement preserves **all** the trace properties (e.g. the chinese-wall policy above)!

Theorem: If $P \in Prop$, $[\![\cdot]\!]$ correct and $s \models P$, then $[\![s]\!] \models P$.

Proof: blackboard.

Security: hyperproperties

Trace properties are not enough 😩

- e.g. *non-interference*: two executions that differ on secret inputs must be indistinguishable to untrusted users

Hyperproperties to the rescue:

- **Idea:** the set of allowed systems
- $\mathbf{P} \in \mathbf{HP} \triangleq \wp(\wp(\mathcal{O}^\omega)) = \wp(Prop)$,
- $p \vDash \mathbf{P}$ *iff* $\mathcal{B}(p) \in \mathbf{P}$
- we can now express properties involving multiple traces!

Security: hyperproperties (cont.)

Again, two relevant classes of hyperproperties:

- **hypersafeties** and **hyperliveness** roughly as above
- subsume trace properties
- still with the same nice properties
- relevant for security!
- **Cons:** not easy anymore! 😩

Correctness and hyperproperties

Consider the subset-closed ($\mathbf{SSC}$) hyperproperties

- i.e. $\mathbf{P} \in \mathbf{SSC}$ if $P \in \mathbf{P}$ and $P' \subseteq P$, then $P' \in \mathbf{P}$

Theorem:
If $\mathbf{P} \in \mathbf{SSC}$, $[\![\cdot]\!]$ correct and $s \vDash \mathbf{P}$, then $[\![s]\!] \vDash \mathbf{P}$.

Proof: blackboard.

Remark:

- Observables are *still* arbitrary, thus
- no preservation if the considered (hyper)property cannot be expressed using $\mathcal{O}$

Where are the attackers?

Security needs attackers!

- Up to now: implicit and passive attackers, that could just *see* (!) the observables

Let's see...

A CATtacker! 🐱

Ok, Seriously... Attackers?

From now onwards:

- Recall that contexts are programs with an hole (denote as C_S and C_T + plug-in operator $[\cdot]$)
- The active attacker
  - provides **context** of execution
  - observes the actions (as before)

Compiler security: full abstraction

Full abstraction (FA):

- standard concept in the field of semantics
- first way to define secure compilation

Definition:

- Assume behavioural equivalence: $s_1 \simeq s_2$ (i.e. equi-convergence)
- A compiler $[\![\cdot]\!]$ is FA iff $\forall s_1, s_2 \in S \,.\, s_1 \simeq s_2 \Leftrightarrow [\![s_1]\!] \simeq [\![s_2]\!]$.

Compiler security: full abstraction (cont.)

- **Correctness:** $s_1 \simeq s_2 \Leftarrow [\![s_1]\!] \simeq [\![s_2]\!]$
- **Security:** $s_1 \simeq s_2 \Rightarrow [\![s_1]\!] \simeq [\![s_2]\!]$
- Both are complex to prove
    - esp. the second one
        - contrapositive: $[\![s_1]\!] \not\simeq [\![s_2]\!] \Rightarrow s_1 \not\simeq s_2$
        - usually to be shown via **back-translation**, i.e. "transform" a context distinguishing the two compiled programs into a context distinguishing their source counterparts

Issues with full abstraction

FA is nice and pretty strong if used correctly, but has some issues:

- Difficult to prove a compiler (not) to be FA
- FA compilers may produce inefficient code
- Mainstream compilers are not usually FA

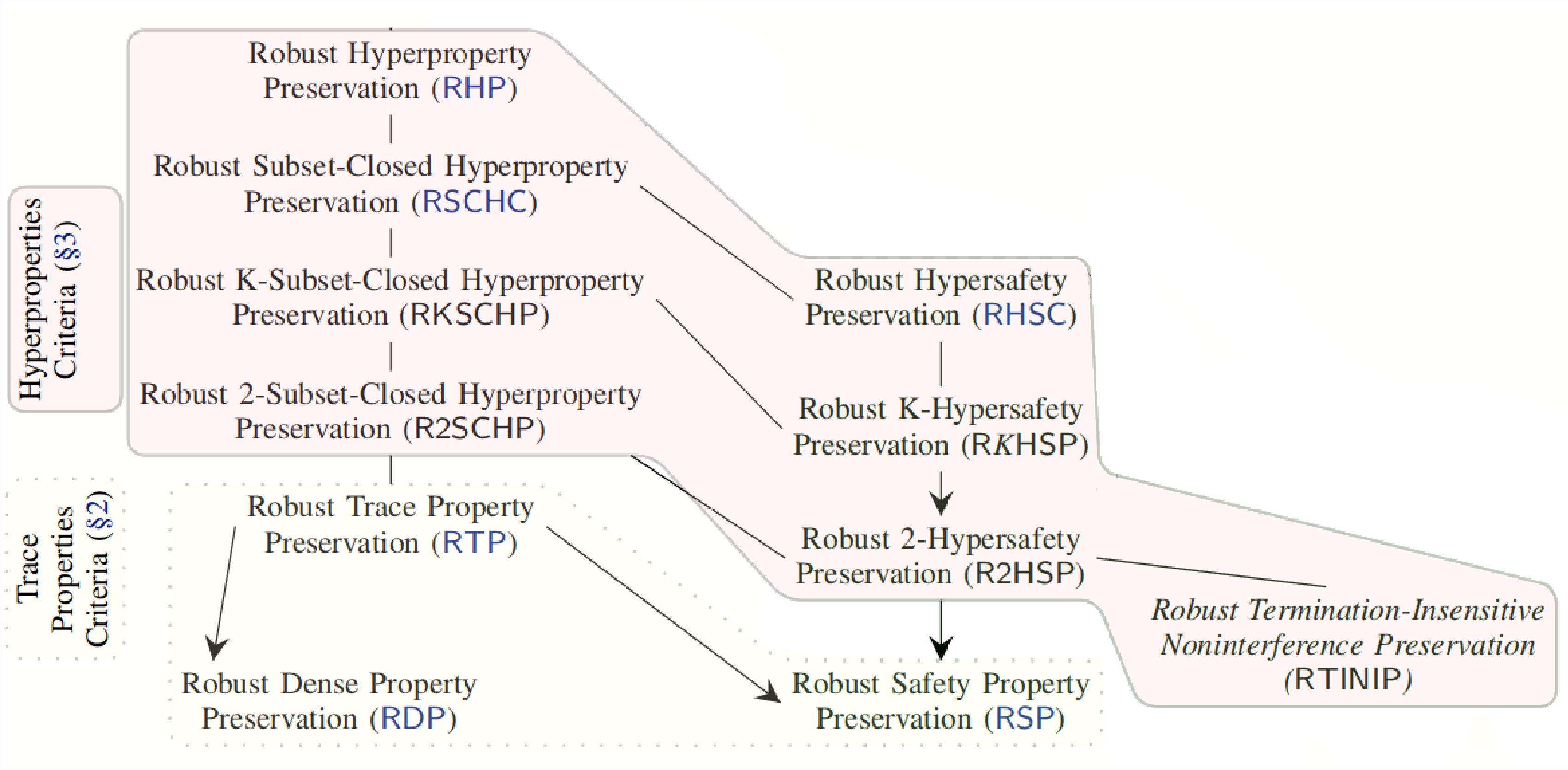
Other notions of security

Recently, **robust hyperproperty preservation (RHP)** have been proposed.
A compiler is RHP whenever

$$\forall \mathbf{P} \in \mathbf{F}, \boldsymbol{s} \in S. \left(\forall C_S. C_S[\boldsymbol{s}] \vDash \mathbf{P}\right) \Rightarrow \left(\forall C_T. C_T[[\![\boldsymbol{s}]\!]] \vDash \mathbf{P}\right)$$

i.e. it preserves all the hyperproperties in the set $\mathbf{F}$.

RHP is not alone 😄



(from https://arxiv.org/abs/1807.04603)

Question: where's FA? - Tricky question! (see Sec. 5 of [6])

Other approaches

Many possible alternative approaches to compiler security:

- Non-robust approaches, i.e. w/o contexts
- Secure translation validation
    - Lift the notion of translation validation to secure compilation
    - Under investigation: which principles are more suitable?
- Hardware-based approaches
    - Enclaves:
        - Intel SGX, Sancus, ...
    - Micro-policies based architectures

Concluding remarks

- Compiler security means **preservation** of some (hyper)property
  - This allows to reason at source level to rule out attacks at the target!
- As for correctness, many principles
  - Full abstraction, w. many applications (e.g. proof of security for mitigations against micro-architectural attacks)
  - New and emerging principles
- Of course, many other approaches in the literature
- No working examples in the slides
  - Things get complex even for very simple languages

The End

If you want to have a chat about secure compilation

just ask Prof. Degano or contact me 🙂

Bibliography

Surveys

[1]. Marco Patrignani, Amal Ahmed, and Dave Clarke. "Formal approaches to secure compilation: A survey of fully abstract compilation and related work." ACM CSUR 51.6 (2019): 125.

[2]. Matteo Busi and Letterio Galletta. "A Brief Tour of Formally Secure Compilation." ITASEC 2019.

Bibliography (cont.)

Secure, non-robust compilation

[3]. Gilles Barthe, Benjamin Grégoire, and Vincent Laporte. "Secure compilation of side-channel countermeasures: the case of cryptographic “constant-time”." IEEE CSF 2018.

[4]. Gilles Barthe, Benjamin Grégoire, and Vincent Laporte. "Secure compilation of side-channel countermeasures: the case of cryptographic “constant-time”." IEEE CSF 2018.

[5]. Jonathan Protzenko, et al. "Verified low-level programming embedded in F." ACM ICFP 2017.

Bibliography (cont.)

Recent ideas and advances

[6]. Carmine Abate, et al. "Journey beyond full abstraction: Exploring robust property preservation for secure compilation." IEEE CSF 2019.

[7]. Dominique Devriese, Marco Patrignani, and Frank Piessens. "Parametricity versus the universal type." ACM POPL 2017.

[8]. PriSC 2020 program, https://popl20.sigplan.org/home/prisc-2020

[9]. PriSC 2019 program, https://popl19.sigplan.org/track/prisc-2019

[10]. Matteo Busi, Pierpaolo Degano and Letterio Galletta. "Translation Validation for Security Properties." https://arxiv.org/abs/1901.05082