PER-based certification of secure information flow

(Work in progress)

Andrzej Filinski Ken Friis Larsen Thomas Jensen

University of Copenhagen and INRIA Rennes

Workshop on Foundations of Computer Security

June 22, 2020

Background and motivation

- **Goal:** formally certifying *safety* and *security* of potentially malicious (or just buggy) mobile code.
  - E.g., User-supplied kernel extensions for network-packet or syscall inspection (eBPF).
- **"Safety":** protecting host's own memory integrity from code.
  - E.g., code may only read packet, and read/write scratch space.
  - "No safety-policy violation is *reachable*."
  - PCC approach: native code + Floyd/Hoare-style safety proof.
    - Complex safety policies expressible.
    - In principle *complete*: all actually safe code is certifiably so.
      (Up to limits of formal reasoning about integer arithmetic.)
- **"Security":** preventing *leakage* of potentially sensitive data made available to code by host.
  - E.g., code may only look at certain packet fields/aspects.
  - "No security-policy violation is *observable*."
  - Variety of information-flow logics/analyses exist.
    - Often only coarse policies (e.g. high/low-security variables).
    - Generally *incomplete*: actually secure code often uncertifiable.

PER-based safety&security policies

- *Uniform framework* for expressing safety and security.
  - **Partial**: some states are *impossible* at given program point
  - **Equivalence Relation**: some states must be *indistinguishable* at given program point.

- Can express complex security policies as relations on *two instances* of state, e.g.,
  - May observe everything about variable x, but only whether variable y is 0 or non-0.
    - $x_1 = x_2 \wedge (y_1 = 0 \Leftrightarrow y_2 = 0)$.
  - May observe whether purported checksum of sensitive data in packet is *correct*, but not the actual value of either.
    - $(\sum_{i=1}^{|p_1.d|} p_1.d[i]) \,\%\, 2^{32} = p_1.c \Leftrightarrow (\sum_{i=1}^{|p_2.d|} p_2.d[i]) \,\%\, 2^{32} = p_2.c$.
  - May observe all data in packet body, as long as header fields satisfy some conditions.
    - $(p1.evil = p2.evil) \wedge (p_1.evil \Rightarrow p_1.payload = p_2.payload)$.

- Crucially: can express and argue safety&security of code in terms of its semantics/meaning only, *not* its form.

Certifying compliance with policy

- In most safety-critical applications (e.g., eBPF), mobile code must terminate in *bounded* time (to protect host *availability*).
  - May disregard termination-based information leakage.
- Correctness assertions of form $\{P\}\ c\ \{Q\}$, where P and Q are PERs, not merely predicates, on state space.
- *Semantic* correctness: functional meaning of program is a *PER morphism*, i.e., maps precondition-related inputs to postcondition-related outputs.
- *Provable* correctness: reducible to ordinary Floyd/Hoare logic (with *ghost variables* for relating inputs to outputs).
  - *Weakest* precondition: needed observability of inputs, to support postcondition-allowed observation of outputs.
  - Must be semantically implied by *actual* precondition.
  - Most of required infrastructure already present in plain PCC.
    - In particular, no significant enlargement of TCB needed.
  - Can rephrase many type-based IF logics as special cases.
- More soon!