Quis Custodiet Ipsos Custodes?

Jasmin Blanchette, Lukas Bulwahn, Andreas Lochbihler, Denis Lohner, Tobias Nipkow, Gregor Snelting, Daniel Wasserrab

PROGRAMMING PARADIGMS GROUP

theorem nonInterferenceSecurity:
  assumes "[cf₁] ≡ₜ [cf₂]" and "(-High-) ∉ HRB-slice (CFG-node (-Low-))" and "valid-edge a"
  and "sourcenode a = (-High-)" and "targetnode a = n" and "kind a = (λs. True)\"" and "n ≡ c"
  and "final c" and "⟨c,[cf₁]⟩ ⇒ ⟨c',s₁⟩" and "⟨c,[cf₂]⟩ ⇒ ⟨c',s₂⟩"
  shows "s₁ ≡ₜ s₂"

proof –
  from High-target-Entry-edge obtain ax where "valid-edge ax" and "sourcenode ax = (-Entry-)"
  and "targetnode ax = (-High-)" and "kind ax = (λs. True)\"" by blast
  from `n ≡ c` `⟨c,[cf₁]⟩ ⇒ ⟨c',s₁⟩` obtain n₁ as₁ cf₁ where "n ≡ₜ as₁⁻→ₜ n₁" and "n₁ ≡ c'" and "transfers (kinds as₁) [[cf₁,undefined]]"
  and "transfers (kinds as₁) [[cf₁,undefined]] = cf₁" and "map fst cf₁ = s₁" by(fastsimp dest:fundamental-property)
  from `n ≡ₜ as₁⁻→ₜ n₁` `valid-edge a` `sourcenode a = (-High-)` `targetnode a = n` `kind a = (λs. True)\""
  have "(-High-)⁻→ₜ-a#as₁⁻→ₜ n₁" by(fastsimp intro:Cons-path simp:vp-def valid-path-def)
  from `final c` `n₁ ≡ c` obtain a₁ where "valid-edge a₁" and "sourcenode a₁ = n₁" and "targetnode a₁ = (-Low-)" and "kind a₁ = (id)"
  by(fastsimp dest:final-edge-Low)
Who will guard the Guards? 
Many software security analysis algorithms are published without soundness proof, some with a manual proof only.
Who will guard the Guards?
Many software security analysis algorithms are published without soundness proof, some with a manual proof only

Vision of our Project:
- provide machine-checked proofs for IFC algorithms
- reach a new level of reliability in language based security (LBS)
- develop new techniques to validate the underlying language description
- integrate semantics, theorem provers and program analysis with LBS

Ultimate Goal: automatically generate an executable, completely machine-verified, PDG-based IFC tool
Starting Point and Goals

KIT: Joana
PDG-based IFC for Java

TUM: Jinja
Java semantics in Isabelle
Starting Point and Goals

KIT: Joana
PDG-based IFC for Java

TUM: Jinja
Java semantics in Isabelle

Project Idea
1. verify the PDG-based IFC algorithm using Isabelle
2. support verification by innovative counter example generators
A tiny PDG

```java
1  a = input();
2  while (n>0) {
3    x = input();
4    if (x>0)
5      b = a;
6    else
7      c = b;
8  }
9  z = c;
```

**Slicing Theorem:**

- no path $x \rightarrow^* y \implies$ information flow $x \rightarrow y$ impossible
- $\exists$ path $x \rightarrow^* y \implies$ potential information flow $x \rightarrow y$

Precise PDG construction for full Java is **very complex**
requires precise points-to analysis
scales to ca. 100 kLOC
Interprocedural PDG-based IFC is correct

- low-equality non-interference
- slicing theorem
- PDG / SDG
- control flow graph
- While
- JinjaThreads
- CoreC++
Interprocedural PDG-based IFC is correct

- low-equality non-interference
- slicing theorem
- PDG / SDG
- control flow graph
- While
- JinjaThreads
- CoreC++

input & output declassification
interprocedural!
call graph & points-to analysis concurrency & memory model
Counter-Example Generation

**Idea:** Find errors in definitions & theorems early
Generate counter-examples for incorrect theorems automatically!

- **Nitpick** translate HOL formula to propositional logic
  hand it to a SAT solver
  generally applicable, requires a lot of fine tuning

- **Quickcheck** evaluate the formula
  test data generation:
  - random
  - exhaustive with intelligent generators
  - symbolic execution + narrowing
  fast, but requires executability
Results

KIT:
- PDGs & slicing for full Java bytecode
  [FSE ’03, PASTE ’04, SCAM ’07a, TPHOLs’ 08, Hamm ’09, JASE ’09a]
- path conditions in PDGs: necessary conditions for information flow
  [SAS ’96, ICSE ’02, TOSEM ’06, SCAM ’07b, PLAS ’08, JASE ’09b]
- IFC for full Java based on PDGs
  [ISSSE ’06, ISOLA ’06, PLAS ’08, IJIS ’09, PLAS ’09, Verify ’10]
- Semantics for Java and C++
  [OOPSLA ’06, FOOL ’08, ESOP ’10, ITP ’11]

TUM:
- Nitpick
  [TAP ’09, TAP ’10, ITP ’10, IJCAR ’10, LPAR ’10, PPDP ’11, FroCoS ’11]
- Quickcheck
  [SEFM ’03, TPHOLs ’09, ICLP ’11, ITP ’11, FroCoS ’11]
Ongoing Work in Quis Custodiet

- Isabelle proof for full algorithm including points-to, threads & memory model
- automatically generate an executable, completely machine-verified, PDG-based IFC tool
- extend and engineer Nitpick & Quickcheck application to Quis Custodiet theorems

Quis Custodiet Ipsos Custodes?
Ongoing Work in Quis Custodiet

- Isabelle proof for full algorithm including points-to, threads & memory model
- automatically generate an executable, completely machine-verified, PDG-based IFC tool
- extend and engineer Nitpick & Quickcheck application to Quis Custodiet theorems

Quis Custodiet Ipsos Custodes?
Isabelle!