Information Flow Control with Errors

Andreas Gampe and Jeffery von Ronne

Department of Computer Science
The University of Texas at San Antonio
Scripted Apps, Risks

- Javascript in
  - Facebook
  - IPhone, Android
  - Windows Metro

- Ruby, Python, . . .

- Profile
- Contacts
- Usage data
Security Approach

- Confidentiality
- Usage Control / Information Flow Control
- Security lattice, e.g., \( \{L, H\} \)
- Runtime monitoring
- Static enforcement
  - Logic
  - Static Analysis
  - Type Systems
The Problem

- Javascript very dynamic

- Approaches
  - Linear Types (Kehrt & Aldrich, FOOL’08)
  - Recency Types (Heidegger & Thiemann, FOOL’09)
  - Singleton Types (Zhao, FOOL’10)

- Richards (PLDI’10): practive vs. theory
Our Approach

- Drop error-freeness as goal
- Pluggable type system
- Handle errors in security type system
- Ensure that errors do not leak information
Based on Abadi & Cardelli
Includes extension (inspired by Liquori)
Errors: method not found on invocation

All syntactical elements labeled
Reduction includes explicit rules for error states
Traditional Type System Setup

- Subtyping allows to use an object in a more general context
  
  *If we expect a point, we can use a colored point. We only “forget” the color functionality.*

- Underapproximate set of methods
Our Setup

- Overapproximate types, add error type

\[
\begin{align*}
&[\text{foo : int, baz : double}] & [\text{foo : int, bar : bool}] & [\text{foo : int, baz : String}] \\
\downarrow & \quad & \downarrow & \quad & \downarrow \\
&[\text{foo : int}] & [\text{bar : bool}] & [\text{foo : int, baz : String}] \\
\downarrow & \quad & \quad & \quad & \downarrow \\
&\varepsilon
\end{align*}
\]

- Only enforces the \textit{consistent} usage of all methods
Consequences

- Code can call unknown methods
- Allows incomplete objects to be typed
- Only one rule for override & extension
- Diamond types are not needed (cf. Liquori)
Security Proof

Preservation
↓
Subst. Reduction
↓
Low Progress
↓
Noninterference

Typed terms reduce to typed terms

Term & term with substitution reduce similarly

Low-typed terms are values or can progress

Low result similar for different high inputs
Indistinguishability by bisimulation

- Indistinguishability by behaviour
  - Two objects equivalent if all low calls have equivalent results
  - Coinductive definition
  - Establish with bisimulation
    - Similar to noninterference proof itself
Type Inference

- Structural inference
- Adapted from Palsberg
  1. Term
  2. Constraint System
  3. Constraint Graph
  4. Type Automaton

  ▶ Subtyping invered $\rightarrow$ some constraints invered

- Security typing: incrementally propagate from type environment
Future Work

- Imperative version
- Delegation / explicit prototypes
- Method deletion
Thanks!

Questions?