Typing Illegal Information Flows as Program Effects

Ana Almeida Matos\textsuperscript{1}  José Santos\textsuperscript{2}

\textsuperscript{1}Instituto de Telecomunicações, Lisbon, Portugal
\textsuperscript{2}INRIA Sophia Antipolis Méditerranée, Antibes, France

The 19th CREST Open Workshop on Interference and Dependence
London, UK
April 30, 2012
## Table of Contents

1. The Problem
2. Relaxing IF Settings
3. Informative Type Systems versus Checking Type Systems
4. Comparison with flow relations
5. Applications and Future Work
Typing Illegal Information Flows as Program Effects

The Problem

How can we?

- Reason about illegal programs:
  - Order illegal programs
- Express arbitrary relaxations of an information flow policy
Typing Illegal Information Flows as Program Effects

Problem

How can we?

- Reason about illegal programs:
  - Order illegal programs
- Express arbitrary relaxations of an information flow policy

Approach

- Establish a base lattice $\Rightarrow$ strictest information flow policy
- Model illegal flows as kernels over the base lattice
- Assign to each program the strictest kernel that captures all its illegal flows
Flow Kernels

Kernels are computed iteratively

\[ \hat{r}_k [l_1, l_2](l) = \begin{cases} 
    k(l \sqcap l_2) & \text{if } l \preceq l_1 \\
    k(l) & \text{otherwise}
\end{cases} \]

- \( k \): original kernel
- \((l_1, l_2)\): new flow
Flow Kernels

Original Lattice

Illegal Flow: $l_3 \rightarrow l_5$
Flow Kernels

Original Lattice

Illegal Flow: $l_1 \rightarrow l_3$
Flow Kernels

Original Lattice

Illegal Flow: \( l_3 \rightarrow l_6 \)
Flow Kernels

**Original Lattice**

Illegal Flow: $l_1 \rightarrow l_2$
Syntax

- **Expressions**: $\lambda$-Calculus + Reference Creation + Thread creation
Language

Syntax

- **Expressions**: λ-Calculus + Reference Creation + Thread creation

Model

- **Model**: \( \langle P, S \rangle \rightarrow \langle P', S' \rangle \)
- **P**: initial pool of expressions
- **S**: initial memory
Syntax

- **Expressions**: $\lambda$-Calculus + Reference Creation + Thread creation

Model

- **Model**: $\langle P, S \rangle \rightarrow \langle P', S' \rangle$
- $P'$: **final** pool of expressions
- $S'$: **final** memory
Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>((\mathcal{L}, \Sigma, k, \Gamma, l))-bissimulation</td>
<td>A binary relation between programs that behave in the same way according to an observer at level (l).</td>
</tr>
<tr>
<td>(\approx_{\mathcal{L}, \Sigma, k}^{\Gamma, l})</td>
<td>The largest ((\mathcal{L}, \Sigma, k, \Gamma, l))-bissimulation.</td>
</tr>
<tr>
<td>((\mathcal{L}, \Sigma, k, \Gamma))-Noninterference</td>
<td>A pool of expressions (P) satisfies Noninterference with respect to a setting ((\mathcal{L}, \Sigma, k)) and a typing environment (\Gamma) if it satisfies (P \approx_{\mathcal{L}, \Sigma, k}^{\Gamma, l} P) for all security levels (l).</td>
</tr>
</tbody>
</table>
Relaxing IF Settings

Original IF Setting

- **Lattice:** $\mathcal{L}$
- **Labeling:** $\Sigma$
Relaxing IF Settings

Original IF Setting
- Lattice: $\mathcal{L}$
- Labeling: $\Sigma$

Relaxed IF Setting
- Lattice: $k(\mathcal{L})$
- Labeling: $k \circ \Sigma$
Relaxing IF Settings

Original IF Setting - Illegal

\[ \Sigma = \left\{ \begin{array}{c}
  a \mapsto l_2, b \mapsto l_3 \\
  c \mapsto l_5, d \mapsto l_7 \\
\end{array} \right\} \]

\[ ((c_{l_5} := (!b_{l_3})); (d_{l_7} := (!a_{l_2}))) \]
Relaxing IF Settings

Original IF Setting - Illegal

\[ \Sigma = \left\{ a \mapsto l_2, b \mapsto l_3, c \mapsto l_5, d \mapsto l_7 \right\} \]

\[ ((c_{l_5} := (!b_{l_3})); (d_{l_7} := (!a_{l_2}))) \]

Relaxed IF Setting - Legal

\[ \Sigma' = \left\{ a \mapsto l_7, b \mapsto l_5, c \mapsto l_5, d \mapsto l_5 \right\} \]

\[ ((c_{l_5} := (!b_{l_5})); (d_{l_7} := (!a_{l_7}))) \]
Information Flow Analysis

Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s, \tau \]

\( M \) is typable with **type** \( \tau \) and **security effect** \( s \) in the **typing context** \( \Gamma \) with respect to the IF setting \( \langle \mathcal{L}, \Sigma, k \rangle \).
Information Flow Analysis

Checking Type System

$$\Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s, \tau$$

$M$ is typable with type $\tau$ and security effect $s$ in the typing context $\Gamma$ with respect to the IF setting $\langle \mathcal{L}, \Sigma, k \rangle$.

Security Effect - $s$

- $s.r$: reading effect
- $s.w$: writting effect
- $s.t$: testing effect
Information Flow Analysis

Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s, \tau \]

Informative Type System

\[ \Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s, s_d \rangle, \tau \]

\( s_d \) is the **declassification effect** of \( M \).
### Information Flow Analysis

<table>
<thead>
<tr>
<th>Checking Type System</th>
<th>Informative Type System</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma \vdash^L_k \varnothing, \Sigma \ M : s, \tau ]</td>
<td>[ \Gamma \vdash L, \Sigma \ M : \langle s, s_d \rangle, \tau ]</td>
</tr>
</tbody>
</table>

- \[ \Gamma \]: a map from variables to security levels
Information Flow Analysis

Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L},\Sigma} M : s, \tau \]

Informative Type System

\[ \Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau \]

- \( \Gamma \): a map from variables to security levels
- \( \mathcal{L} \): Lattice of security levels
## Information Flow Analysis

### Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L},\Sigma} M : s, \tau \]

### Informative Type System

\[ \Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau \]

- **\( \Gamma \)**: a map from variables to security levels
- **\( \mathcal{L} \)**: lattice of security levels
- **\( \Sigma \)**: a map from references to security levels
Information Flow Analysis

Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s, \tau \]

Informative Type System

\[ \Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s, s_d \rangle, \tau \]

- \( \Gamma \): a map from variables to security levels
- \( \mathcal{L} \): lattice of security levels
- \( \Sigma \): a map from references to security levels
- \( s \): security effect
Information Flow Analysis

Checking Type System

\[ \Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s, \tau \]

- \( k \): parametrizing kernel

Informative Type System

\[ \Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s, s_d \rangle, \tau \]

- \( s_d \): declassification effect

- \( \Gamma \): a map from variables to security levels
- \( \mathcal{L} \): lattice of security levels
- \( \Sigma \): a map from references to security levels
- \( s \): security effect
Information Flow Analysis

Checking Type System - Assign Rule

\[
\begin{align*}
\Gamma \vdash_{\mathcal{L}, \Sigma} k^M : s_1, \theta \text{ ref}_l & \quad k(s_1.t) \sqsubseteq k(s_2.w) \\
\Gamma \vdash_{\mathcal{L}, \Sigma} k^N : s_2, \theta & \quad k(s_1.r), k(s_2.r) \sqsubseteq k(l) \\
\hline
\Gamma \vdash_{\mathcal{L}, \Sigma} k^M := N : s_1 \sqcup s_2 \sqcup s_l, \text{unit}
\end{align*}
\]

Where: \( s_l = \langle \bot, k(l), \bot \rangle \)
Information Flow Analysis

Checking Type System - Assign Rule

\[
\begin{align*}
\Gamma \vdash^k_{\mathcal{L}, \Sigma} M : s_1, \theta \text{ ref}_l & \quad k(s_1.t) \sqsubseteq k(s_2.w) \\
\Gamma \vdash^k_{\mathcal{L}, \Sigma} N : s_2, \theta & \quad k(s_1.r), k(s_2.r) \sqsubseteq k(l)
\end{align*}
\]

\[
\Gamma \vdash^k_{\mathcal{L}, \Sigma} M := N : s_1 \sqcup s_2 \sqcup s_l, \text{ unit}
\]

Where: \( s_l = \langle \bot, k(l), \bot \rangle \)

Informative Type System - Assign Rule

\[
\begin{align*}
\Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s_1, s_1^d \rangle, \theta \text{ ref}_l & \quad \Gamma \vdash_{\mathcal{L}, \Sigma} N : \langle s_2, s_2^d \rangle, \theta
\end{align*}
\]

\[
\Gamma \vdash_{\mathcal{L}, \Sigma} M := N : \langle s, s_d \rangle, \text{ unit}
\]

Where:
\[
\begin{align*}
s_d & = s_1^d \uplus s_2^d \uplus \{ (s_1.t, s_2.w), (s_1.r, l), (s_2.r, l) \} \\
s & = s_1 \sqcup s_2 \sqcup \langle \bot, l, \bot \rangle
\end{align*}
\]
Information Flow Analysis

Soundness

\[
\Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s, s_d \rangle, \tau \\
\downarrow \\
\Gamma \vdash_{\mathcal{L}, \Sigma}^{s_d} M : s', \tau
\]
Information Flow Analysis

**Soundness**

\[ \Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s, s_d \rangle, \tau \]

\[ \Downarrow \]

\[ \Gamma \vdash_{s_d}^{s_d} M : s', \tau \]

**Optimality**

\[ \Gamma \vdash_{\mathcal{L}, \Sigma}^k M : s_1, \tau \quad \text{and} \quad \Gamma \vdash_{\mathcal{L}, \Sigma} M : \langle s_2, s_d^2 \rangle, \tau \]

\[ \Downarrow \]

\[ k \sqsubseteq s_d^2 \]
Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals:** $\text{Pri}$
- **Security levels:** subsets of $\text{Pri}$
- **Security lattice:** $\langle \mathcal{P}(\text{Pri}), \supseteq \rangle$
- **Flow Relations:** binary relations on $\text{Pri}$
  - $(A, B) \in F$: information may flow from principal $A$ to principal $B$
Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals**: \( \text{Pri} \)
- **Security levels**: subsets of \( \text{Pri} \)
- **Security lattice**: \( \langle \mathcal{P}(\text{Pri}), \supseteq \rangle \)
- **Flow Relations**: binary relations on \( \text{Pri} \)

Remark

*Flow Relations correspond to the co-additive kernels on \( \text{Pri} \)*
Flow Relations as IF Setting Relaxations

Original IF Setting

\[ \text{Pri} = \{A, B, C\} \]

\[ \mathcal{P}(\text{Pri}) \]

\[
\begin{align*}
\{ \emptyset \} \\
\{ C \} & \quad \{ A \} & \quad \{ B \} \\
\{ A, C \} & \quad \{ B, C \} & \quad \{ A, B \} \\
\{ A, B, C \} \\
\end{align*}
\]
Flow Relations as IF Setting Relaxations

Original IF Setting

\[ \text{Pri} = \{A, B, C\} \]

\[ P(\text{Pri}) \]

\[ \{\emptyset\} \]

\[ \{A\} \]

\[ \{B\} \]

\[ \{C\} \]

\[ \{A, C\} \]

\[ \{A, B\} \]

\[ \{A, B, C\} \]

Relaxed IF Setting

\[ f = \{(B, A), (C, A)\} \]

\[ \uparrow_f (P(\text{Pri})) \]

\[ \{\emptyset\} \]

\[ \{A\} \]

\[ \{A, C\} \]

\[ \{A, B\} \]

\[ \{A, B, C\} \]
Flow Relations as IF Setting Relaxations

\[ \mathcal{P}(\text{Pri}) \]

\[
\begin{align*}
\{C\} & \rightarrow \{A\} \rightarrow \{B\} \\
\{A, C\} & \rightarrow \{B, C\} \\
\{A, B\} & \rightarrow \{A, B, C\}
\end{align*}
\]

\[(b_{\{A\}} := (!a_{\{B, C\}}))\]
Flow Relations as IF Setting Relaxations

**IF Setting**

\[ \mathcal{P}(\text{Pri}) \]

\[
\begin{align*}
\{C\} & \rightarrow \{A\} \\
\{A, C\} & \rightarrow \{A, B\} \\
\{A, B, C\} & \rightarrow \{A, B, C\}
\end{align*}
\]

**Declassification Effect**

\[ s_d \]

\[
\begin{align*}
\{C\} & \rightarrow \{A\} \\
\{A, C\} & \rightarrow \{A, B\} \\
\{A, B, C\} & \rightarrow \{A, B, C\}
\end{align*}
\]

\[(b_{\{A\}} := (!a_{\{B,C\}}))\]
Flow Relations as IF Setting Relaxations

Declassification Effect

Flow Relations below $s_d$

No optimality result for flow relations!

- $f_1, f_2 \preceq s_d$ and $f_1 \not\preceq f_2$ and $f_2 \not\preceq f_1$
Permissivity Contexts as Kernels

Ingredients

- Model the **permissivity context** under which a program executes as a kernel
- The permissivity context $\Rightarrow$ Relaxation of the original IF setting
- Permissivity contexts are allowed to change dynamically
Permissivity Contexts as Kernels

Ingredients

- Model the **permissivity context** under which a program executes as a kernel
- The permissivity context $\Rightarrow$ Relaxation of the original IF setting
- Permissivity contexts are allowed to change dynamically

Goal

Only the threads that respect **all** the permissivity contexts that were allowed during the program execution are allowed to terminate.
Permissivity Contexts as Kernels

Approach

- The current permissivity context - \( k_A \) - to configurations
- Add a mapping from thread names to their declassification effect - \( D \) - to configurations
- When the permissivity context changes remove the threads that are not compliant with it

Configurations

\[ \langle P, S, D, k_A \rangle \]

Changing the permissivity context

\[ \langle P, S, D, k_A \rangle \rightarrow \langle P', S, D, k_F \rangle \]
Future Work

Study new program constructs that *dynamically* interact with the permissivity context:

- Check if an expression is compliant with the current permissivity context
- Test the current permissivity context
- Kernels as values...
Thank You!

Questions...