Typing Illegal Information Flows as Program Effects

Ana Almeida Matos[1] José Santos[2]

[1]Instituto de Telecomunicações, Lisbon, Portugal

[2]INRIA Sophia Antipolis Méditerranée, Antibes, France

PLAS'2012
Beijing, China
June 15, 2012

Outline

Information Flow Security Analysis

General Goal

Prevent the execution of programs that leak information

What do we mean by information leak?

- **Information Flow Policy:**
  - A lattice of security levels: $\mathcal{L} = \langle L, \sqsubseteq, \sqcup, \sqcap, \bot, \top \rangle$
  - Assign a security level to each resource: $\Sigma : \mathcal{R}ef \rightarrow L$
- Information is not allowed to flow from high security levels to low security levels:
  - $l_1 \sqsubseteq l_2 \Rightarrow l_1 \rightarrow l_2$ - **Legal**
  - $l_1 \not\sqsubseteq l_2 \Rightarrow l_1 \rightarrow l_2$ - **Illegal**

Goal

We want to...

Be able to reason to reason about illegal programs

Why?

- Order illegal programs wrt their degree of illegality
- Express arbitrary policy relaxations

Goal

Standard Type Systems

- The type assigned to each program is generally not meaningfull
- Essentially there are only two types: **secure** and **insecure**

Our approach

- Type each program with a **summary** of the illegal flows that it may trigger:

Program declassification effect

Reinterpreting the Declassification Effect

Type interpretation

A summary of the illegal flows that may occur during the exectution of the program

Policy interpretation

- A relaxation of the original policy
- The strictest IF policy to which the program complies

Outline

Representing Illegal Flows

Kernels

The declassification effect is modeled as kernel on the original lattice

Kernel???

A kernel is a function k on a lattice which is:

- **contractant:** $k(l) \sqsubseteq l$
- **idempotent:** $k(k(l)) = k(l)$
- **monotone:** $l_1 \sqsubseteq l_2 \;\Rightarrow\; k(l_1) \sqsubseteq k(l_2)$

Informally, a kernel is a function on a lattice that...

maps each point in the lattice to itself or to a lower point preserving the original relative orders.

Representing Illegal Flows

Kernels

The declassification effect is modeled as kernel on the original lattice

Capturing illegal flows

- Kernel k is said to **admit** flow $l_1 \to l_2$ if: $k(l_1) \sqsubseteq k(l_2)$
- Given a set of illegal flows the **strictest kernel** that admits all its flows is always defined!

Why Kernels?

Kernels can be ordered in a meaningful way

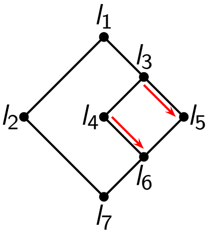
- $k_1 \sqsubseteq k_2$: k_1 collapses more levels down than k_2
- **Interpretation:** k_1 is equally or less strict than k_2

The set of all kernels is a lattice

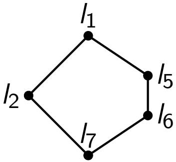
- Given two kernels k_1 and k_2, $k_1 \sqcap k_2$ is always defined
- $k_1 \sqcap k_2$ is the strictest kernel that is simulaneously less confidential than k_1 and k_2
- The **strictest kernel** maps each level to itself
- The **most permissive kernel** maps all levels to the bottom level

Kernels as Declassification Effects

Original Lattice



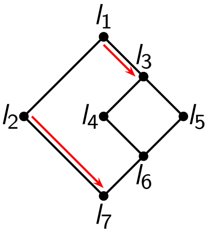
Illegal Flow: $l_3 \rightarrow l_5$



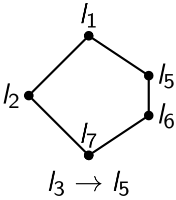
$l_3 \rightarrow l_5$

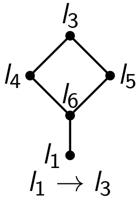
Kernels as Declassification Effects

Original Lattice



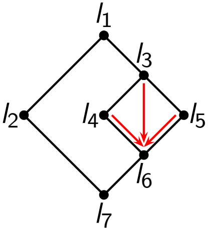
Illegal Flow: $l_1 \rightarrow l_3$



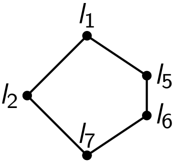


Kernels as Declassification Effects

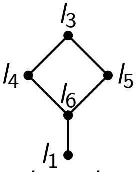
Original Lattice



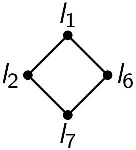
Illegal Flow: $l_3 \to l_6$



$l_3 \to l_5$



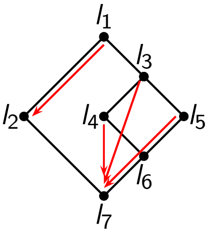
$l_1 \to l_3$



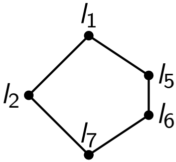
$l_3 \to l_6$

Kernels as Declassification Effects

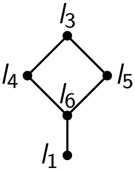
Original Lattice



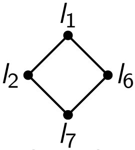
Illegal Flow: $l_1 \rightarrow l_2$



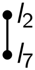
$l_3 \rightarrow l_5$



$l_1 \rightarrow l_3$



$l_3 \rightarrow l_6$



$l_1 \rightarrow l_2$

Relaxing IF Settings

Kernels as policy relaxations

- An illegal flow $l_1 \to l_2$ in the original setting is deemed legal by a given kernel k if:

$$k(l_1) = k(l_2)$$

Relaxing IF Settings

Kernels as policy relaxations

- An illegal flow $l_1 \to l_2$ in the original setting is deemed legal by a given kernel k if:

$$k(l_1) = k(l_2)$$

Original IF Setting

- **Lattice:** $\mathcal{L}$
- **Labeling:** Σ

Relaxing IF Settings

Kernels as policy relaxations

- An illegal flow $l_1 \to l_2$ in the original setting is deemed legal by a given kernel k if:

$$k(l_1) = k(l_2)$$

Original IF Setting

- **Lattice:** $\mathcal{L}$
- **Labeling:** Σ

$\xrightarrow{\mathbf{k}}$

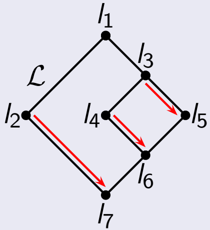
Relaxed IF Setting

- **Lattice:** $k(\mathcal{L})$
- **Labeling:** $k \circ \Sigma$

Relaxing IF Settings

Original IF Setting

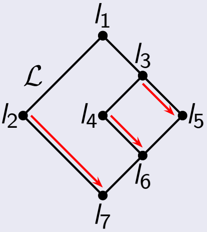
$$\Sigma = \left\{ \begin{array}{c} a \mapsto l_2, b \mapsto l_3 \\ c \mapsto l_5, d \mapsto l_7 \end{array} \right\}$$



Relaxing IF Settings

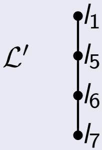
Original IF Setting

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



Relaxed IF Setting

$$\Sigma' = \left\{ \begin{array}{c} a \mapsto l_7, b \mapsto l_5 \\ c \mapsto l_5, d \mapsto l_5 \end{array} \right\}$$



Outline

Language

Syntax

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

Language

Syntax

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

Small-step operational semantics

- **Transitions:** $\langle P, S \rangle \rightarrow \langle P', S' \rangle$
- P: **pool of threads**
- S: **memory**

Security Property

$(\mathcal{L}, \Sigma, k, \Gamma)$-Noninterference

A pool of expressions P satisfies Noninterference with respect to a setting $(\mathcal{L}, \Sigma, k)$ and a typing environment Γ if it satisfies $P \approx^{\mathcal{L},\Sigma,k}_{\Gamma,l} P$ for all security levels l.

Where:

- $\approx^{\mathcal{L},\Sigma,k}_{\Gamma,l}$ is the **largest** $(\mathcal{L}, \Sigma, k, \Gamma, l)$-bissimulation

Information Flow Analysis

Checking Type System

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

Informative Type System

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

- Γ: **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

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

- Γ: **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

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

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **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

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

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **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**

- Γ: **a map from variables to security levels**
- $\mathcal{L}$: **lattice of security levels**
- Σ: **a map from references to security levels**
- s: **security effect**

Information Flow Analysis

Checking Type System

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

- The type system is parameterized with a fixed kernel: k
- The typing rules **constrain** the information flows that may take place and **type out** illegal programs

Information Flow Analysis

Checking Type System

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

- The type system is parameterized with a fixed kernel: k
- The typing rules **constrain** the information flows that may take place and **type out** illegal programs

Informative Type System

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

- The type system is not parameterized with any kernel
- The typing rules **update** the **declassification effect** to include the detected information flows

Information Flow Analysis

Checking Type System - Assign Rule

$$\frac{\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \theta \ \mathrm{ref}_l \qquad k(s_1.t) \sqsubseteq k(s_2.w) \qquad \Gamma \vdash^{k}_{\mathcal{L},\Sigma} N : s_2, \theta \qquad k(s_1.r), k(s_2.r) \sqsubseteq k(l)}{\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M := N : s_1 \sqcup s_2 \sqcup s_l, \mathrm{unit}}$$

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

Information Flow Analysis

Checking Type System - Assign Rule

$$\frac{\begin{array}{ll}\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \theta\ \mathsf{ref}_l & k(s_1.t) \sqsubseteq k(s_2.w) \\ \Gamma \vdash^{k}_{\mathcal{L},\Sigma} N : s_2, \theta & k(s_1.r), k(s_2.r) \sqsubseteq k(l)\end{array}}{\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M := N : s_1 \sqcup s_2 \sqcup s_l, \mathsf{unit}}$$

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

Informative Type System - Assign Rule

$$\frac{\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s_1, s_1^d \rangle, \theta\ \mathsf{ref}_l \quad \Gamma \vdash_{\mathcal{L},\Sigma} N : \langle s_2, s_2^d \rangle, \theta}{\Gamma \vdash_{\mathcal{L},\Sigma} M := N : \langle s, s_d \rangle, \mathsf{unit}}$$

Where:

$s_d = s_1^d \sqcup s_2^d \sqcup \upharpoonright \{(s_1.t, s_2.w), (s_1.r, l), (s_2.r, l)\}$

$s = s_1 \sqcup s_2 \sqcup \langle \bot, l, \bot \rangle$

Soundness

Checking Type System

If $\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$, then P satisfies $(\mathcal{L}, \Sigma, k, \Gamma)$-Noninterference

Informative Type System

If $\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$, then P satisfies $(\mathcal{L}, \Sigma, s_d, \Gamma)$-Noninterference

Soundness

Checking Type System

If $\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$, then P satisfies $(\mathcal{L}, \Sigma, k, \Gamma)$-Noninterference

Informative Type System

If $\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$, then P satisfies $(\mathcal{L}, \Sigma, s_d, \Gamma)$-Noninterference

Proof Sketch

$$\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$$
$$\Downarrow$$
$$\Gamma \vdash^{s_d}_{\mathcal{L},\Sigma} M : s', \tau$$

Soundness

Checking Type System

If $\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s, \tau$, then P satisfies $(\mathcal{L}, \Sigma, k, \Gamma)$-Noninterference

Informative Type System

If $\Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s, s_d \rangle, \tau$, then P satisfies $(\mathcal{L}, \Sigma, s_d, \Gamma)$-Noninterference

Optimality

$$\Gamma \vdash^{k}_{\mathcal{L},\Sigma} M : s_1, \tau \quad \textbf{and} \quad \Gamma \vdash_{\mathcal{L},\Sigma} M : \langle s_2, s_d^2 \rangle, \tau$$
$$\Downarrow$$
$$k \sqsubseteq s_d^2$$

Outline

Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals: Pri**
- **Security levels:** subsets of **Pri**
- **Security lattice:** $\langle \mathcal{P}(\mathbf{Pri}), \supseteq \rangle$
- **Flow Relations:** binary relations on **Pri**
  - $(A, B) \in f$: information may flow A to B

Flow Relations as IF Setting Relaxations

Ingredients

- **Set of Principals: Pri**
- **Security levels:** subsets of **Pri**
- **Security lattice:** $\langle \mathcal{P}(\mathbf{Pri}), \supseteq \rangle$
- **Flow Relations:** binary relations on **Pri**
  - $(A, B) \in f$: information may flow A to B

Remark

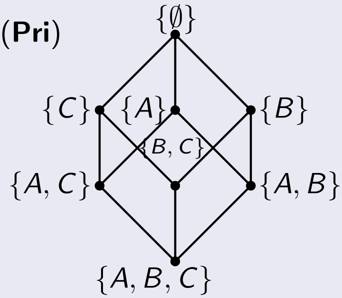
Flow Relations correspond to the co-additive kernels on Pri

Flow Relations as IF Setting Relaxations

Original IF Setting

$$\mathbf{Pri} = \{A, B, C\}$$

$\mathcal{P}(\mathbf{Pri})$

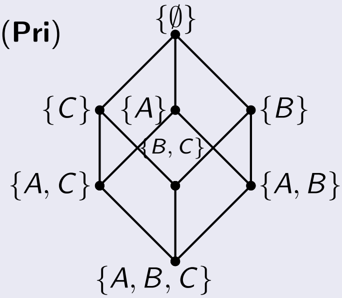


Flow Relations as IF Setting Relaxations

Original IF Setting

$$\mathbf{Pri} = \{A, B, C\}$$

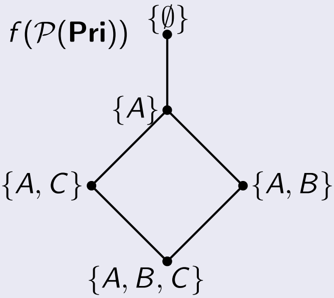
$\mathcal{P}(\mathbf{Pri})$



Relaxed IF Setting

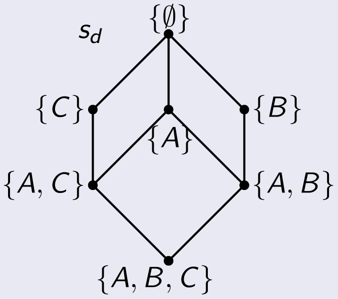
$$\mathbf{f} = \{(B, A), (C, A)\}$$

$f(\mathcal{P}(\mathbf{Pri}))$

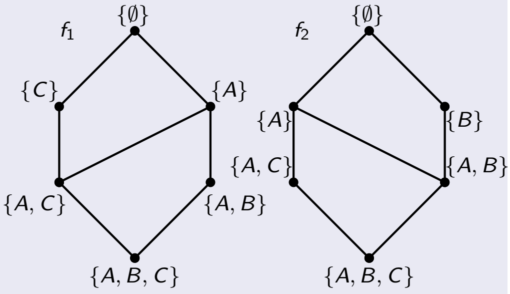


Flow Relations as IF Setting Relaxations

Declassification Effect



Flow Relations below s_d



No optimality result for flow relations!

- $f_1, f_2 \preccurlyeq s_d$ **and** $f_1 \not\preccurlyeq f_2$ **and** $f_2 \not\preccurlyeq f_1$

Outline

Permissivity Contexts as Kernels

Scenario

- **Permissivity context:** relaxation of the original IF setting
- Each program executes under a permissivity context
- Permissivity contexts are allowed to change dynamically

Permissivity Contexts as Kernels

Scenario

- **Permissivity context:** relaxation of the original IF setting
- Each program executes under a permissivity context
- Permissivity contexts are allowed to change dynamically

Our approach

- Model the permissivity context as a kernel
- Label each thread with the corresponding declassification effect

Permissivity Contexts as Kernels

Dynamic Enforcement Mechanism

When the permissivity context changes:

- **Abort** all threads which are not compatible with the new permissivity context
- To determine compatibility just compare the kernels

Security Property

All threads that are allowed to terminate verify Noninterference wrt **all** the permissivity contexts.

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

Thank You!

Questions?