Information flow security and safety in multiparty sessions

Ilaria Castellani
(INRIA Sophia Antipolis)

with Sara Capecchi and Mariangiola Dezani-Ciancaglini
(TORINO University)

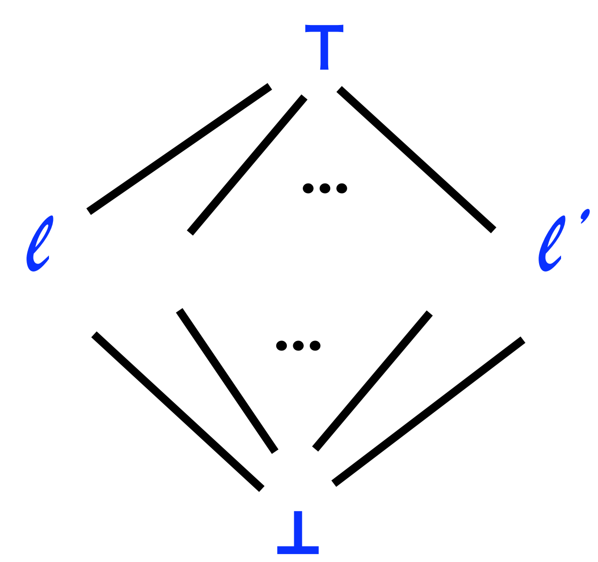
BETTY Summer School   Lovran, June 30 - July 4, 2014

General goal

Information flow control in multiparty sessions, to preserve confidentiality of participants' data

A finite lattice of security levels :

levels assigned to variables and values



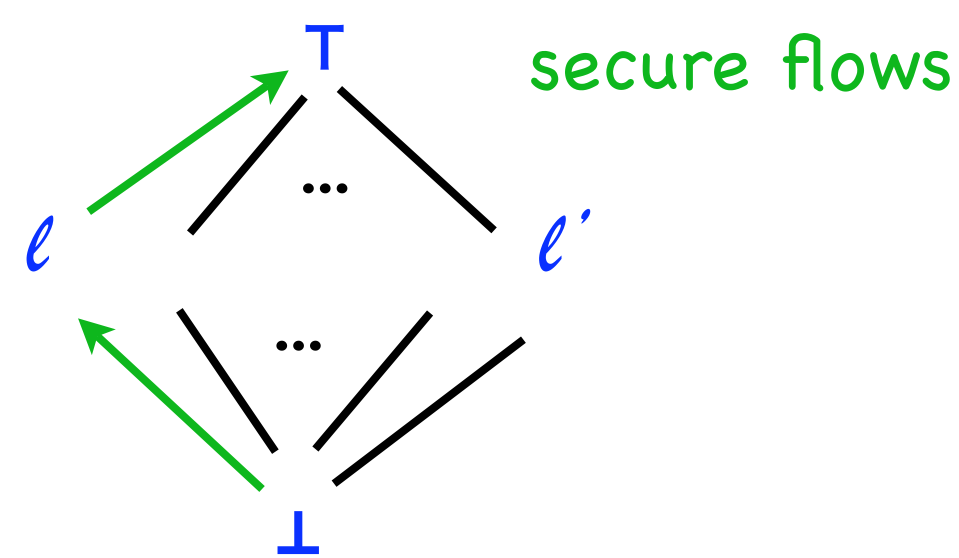
Secure information flow (SIF): the input or output of a value v^{ℓ} should only depend on inputs of values $v_0^{\ell_0}$ with $\ell_0 \leq \ell$

General goal

Information flow control in multiparty sessions, to preserve confidentiality of participants' data

A finite lattice of security levels :

levels assigned to variables and values



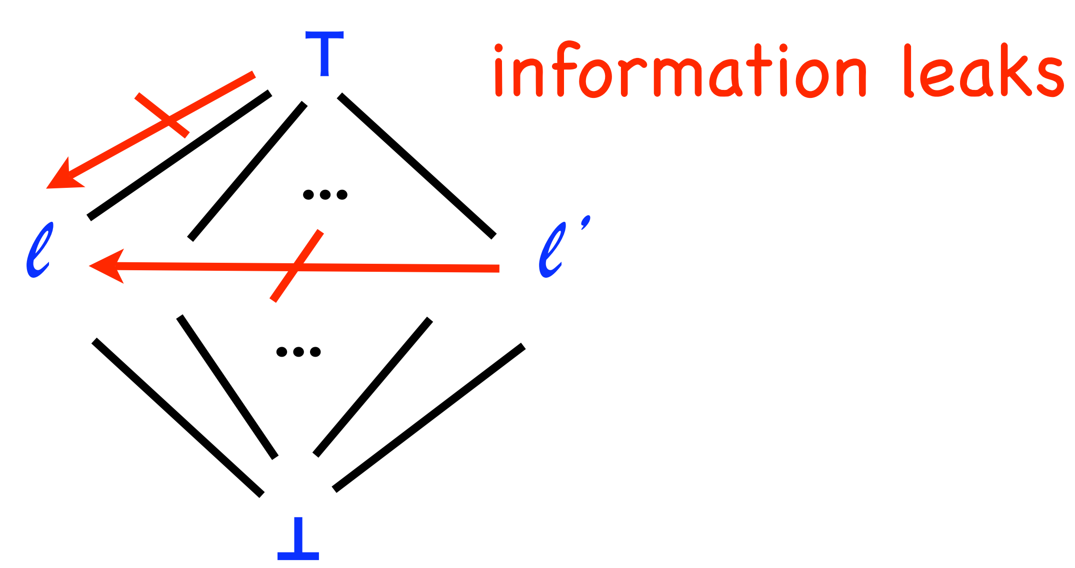
Secure information flow (SIF): the input or output of a value v^{ℓ} should only depend on inputs of values $v_0^{\ell_0}$ with $\ell_0 \leq \ell$

General goal

Information flow control in multiparty sessions, to preserve confidentiality of participants' data

A finite lattice of security levels :

levels assigned to variables and values



Secure information flow (SIF): the input or output of a value v^{ℓ} should only depend on inputs of values $v_0^{\ell_0}$ with $\ell_0 \leq \ell$

Sessions

- Session: abstraction for "structured communication"

  a particular activation of a service, with:

  - fixed number of participants, with predefined roles
  - fixed types for exchanged data
  - fixed order for interactions (unless independent)

Private conversation following a specified protocol

Security in sessions

Private conversation following a specified protocol

$\Downarrow$

Expectation: security should be easier to achieve!

- Private session channels => no external leaks
- Disciplined behaviour => fewer internal leaks

Tracking information leaks

How to prevent / detect information leaks ?

- Typing (prevention): security-enhanced session types
- Safety (detection): induced by a monitored semantics
- Security (detection): behavioural property based on observational equivalence / bisimulation

Tracking information leaks

How to prevent / detect information leaks ?

- Typability (prevention): security-enhanced session types

  $\Downarrow$
- Safety (detection): induced by a monitored semantics

  $\Downarrow$
- Security (detection): behavioural property based on observational equivalence / bisimulation

Tracking information leaks

How to prevent / detect information leaks ?

- Typability (prevention): security-enhanced session types

  $\Downarrow \; \not\Uparrow$

- Safety (detection): induced by a monitored semantics

  $\Downarrow \; \not\Uparrow$

- Security (detection): behavioural property based on observational equivalence / bisimulation

Tracking information leaks

How to prevent / detect information leaks ?

- Typability (prevention): security-enhanced session types

$$\Downarrow \; \not\Uparrow$$

- Safety (detection): induced by a monitored semantics

$$\Downarrow \; \not\Uparrow$$

- Security (detection): behavioural property based on observational equivalence / bisimulation

3 increasingly precise ways to track information leaks

Classical approach to SIF

How to prevent / detect information leaks ?

- Typability (prevention): security types

$\Downarrow \quad \not\Uparrow$

- Security (detection): behavioural property based on observational equivalence / bisimulation

Approach pioneered by Volpano, Smith, Irvine [VSI96]

Overview

Part 1: A quick tour on secure information flow, from imperative languages to process calculi

Security session calculus

Part 2: security, types

- security property
- security type system
- typability => security

Part 3: safety

- monitored semantics
- safety property
- safety => security

Part 4: future directions

Part 1

A quick tour on secure information flow (SIF)

Secure information flow

Why does it matter?

Secure information flow

Why does it matter?

Techniques for data protection

- Encryption: secures data transmission on channels, but not what happens with them on destination
- Access control: controls who may directly access data, but not their further propagation

Secure information flow

Techniques for data protection

- **Encryption:** secures data transmission on channels, but not what happens with them on destination
- **Access control:** controls who may directly access data, but not their further propagation
- **Secure information flow:** controls data propagation throughout the system

=> end-to-end protection of data confidentiality

Language based security

Use programming language techniques to specify and enforce security properties of programs.

Language-based approach pioneered by Volpano, Smith and Irvine:

- Sequential imperative language:

  [**VSI96**] D. Volpano, G. Smith and C. Irvine. *A Sound Type System for Secure Flow Analysis*, J. of Computer Security, 1996.

- Multi-threaded imperative language:

  [**SV98**] G. Smith and D. Volpano. *Secure information flow in a multi-threaded imperative language*, POPL'98.

- A good survey:

  [**SM03**] A. Sabelfeld and A. Myers. *Language-based information flow security*, IEEE J. Selected areas in communications, 2003.

SIF: imperative languages

- Information: contained in "objects", used by "subjects".
- Objects have security levels forming a lattice, for instance:

$$H = \text{high} = \text{secret} \qquad L = \text{low} = \text{public}$$

- Secure information flow: no flow from high to low objects.

$$y_L := x_H \qquad \text{not secure}$$

$$z_H := x_H \; ; \; y_L := 0 \qquad \text{secure}$$

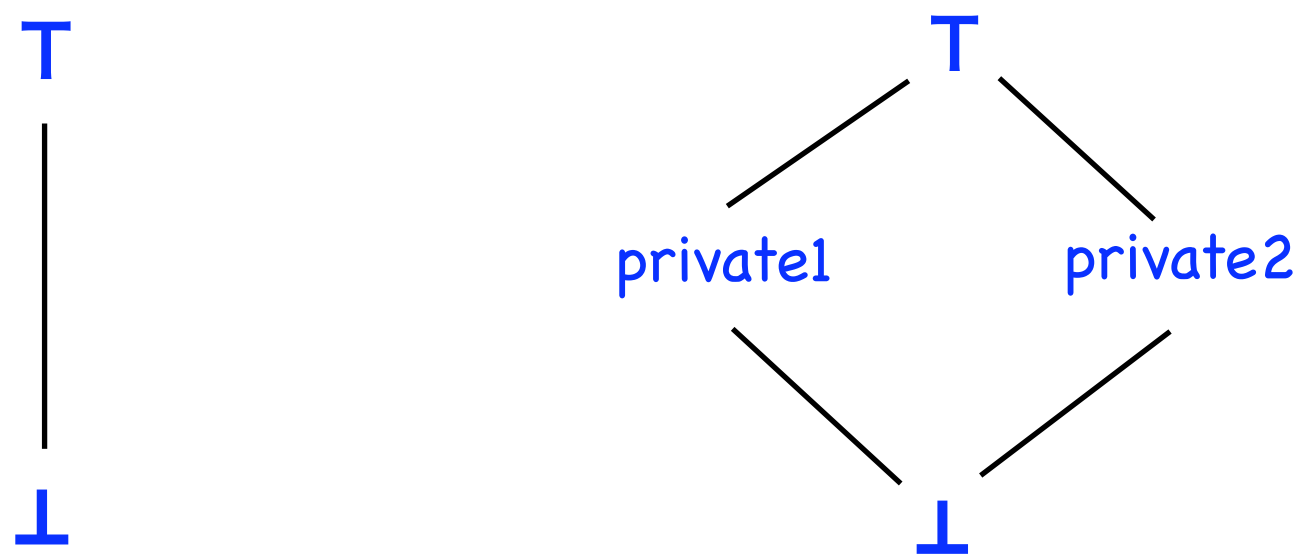
- Imperative languages:
  - Subjects = programs. Objects = variables.
  - Language techniques:

$$\begin{cases} \text{behavioural equivalence to formalise security property} \\ \text{type system to statically ensure it} \end{cases}$$

SIF: imperative languages

Lattice model [Bell & LaPadula 73], [Denning 76] :

lattice $(\mathcal{S}, \leq)$ of **security levels** for variables.



SIF: imperative languages

Noninterference [Goguen & Meseguer 82] :

high-level variables *do not interfere* with low-level variables.

Meaning in a **sequential imperative language**:

> The *final* value of a low variable y_L does not depend on the *initial* value of any high variable x_H.

SIF: imperative languages

Leak-freedom would be a better name!

Noninterference [Goguen & Meseguer 82] :

high-level variables *do not interfere* with low-level variables.

Meaning in a **sequential imperative language**:

The *final* value of a low variable y_L does not depend on the *initial* value of any high variable x_H.

Public outputs should not depend on private inputs

SIF: imperative languages

- **Explicit flow :** $y_L := x_H$

- **Implicit flow :**

  $\texttt{if } x_H \texttt{ then } y_L := tt \texttt{ else } y_L := \mathit{ff}$

The value of x_H is copied into y_L.

SIF: imperative languages

- **Explicit flow :** $y_L := x_H$

- **Implicit flow :**

  $\texttt{if}\ x_H\ \texttt{then}\ y_L := tt\ \texttt{else}\ y_L := f\!f$

Types

lower bound for writes

$\Gamma \vdash P \; : \; \tau$

$\mathrm{Ex} : \Gamma \vdash (x_H := y_L) : H \qquad \Gamma \vdash ((x_H := y_L);(y_L := z_L)) : L$

SIF: imperative languages

- **Explicit flow :** $y_L := x_H$

- **Implicit flow :**

  $\texttt{if } x_H \texttt{ then } y_L := tt \texttt{ else } y_L := f\!f$

Types

lower bound for writes

$$\Gamma \vdash P \ : \ \tau$$

$$\mathrm{Ex} : \Gamma \vdash (x_H := y_L) : H \quad \Gamma \vdash ((x_H := y_L);(y_L := z_L)) : L$$

Rule for conditional: level of condition $\leq$ levels of branches

SIF: imperative languages

Termination leaks

$$\texttt{while } x_H \texttt{ do nil} \; ; \; y_L := \mathit{ff}$$

$$\texttt{if } x_H \texttt{ then nil else loop} \; ; \; y_L := \mathit{ff}$$

In both programs: depending on the value of x_H the 1st component will either terminate or loop. In the latter case y_L will never be updated.

Leaks due to different termination behaviours after a high test

SIF: imperative languages

Termination leaks

$$\texttt{while } x_H \texttt{ do nil ; } y_L := \mathit{ff}$$

$$\texttt{if } x_H \texttt{ then nil else loop ; } y_L := \mathit{ff}$$

-> may be ignored in sequential case, using termination-insensitive noninterference

-> cannot be ignored in concurrent case!

Example on next slide

SIF: parallel imp. languages

$$P = \alpha \parallel \beta \parallel \gamma, \text{ where :}$$

$$\gamma : \texttt{if } PIN = 0 \texttt{ then } t_\alpha := tt \texttt{ else } t_\beta := tt$$

$$\alpha : \texttt{while } t_\alpha = \mathit{ff} \texttt{ do nil} \ ; \ r := 1 \ ; \ t_\beta := tt$$

$$\beta : \texttt{while } t_\beta = \mathit{ff} \texttt{ do nil} \ ; \ r := 0 \ ; \ t_\alpha := tt$$

$$\Gamma = PIN, t_\alpha, t_\beta : H, \quad r : L$$

$$\Gamma \vdash \gamma : H, \quad \Gamma \vdash \alpha, \beta : L$$

SIF: parallel imp. languages

$$P = \alpha \parallel \beta \parallel \gamma, \text{ where :}$$

$$\gamma : \texttt{if } PIN = 0 \texttt{ then } t_\alpha := tt \texttt{ else } t_\beta := tt$$
$$\alpha : \texttt{while } t_\alpha = \mathit{ff} \texttt{ do nil} \;;\; r := 1 \;;\; t_\beta := tt$$
$$\beta : \texttt{while } t_\beta = \mathit{ff} \texttt{ do nil} \;;\; r := 0 \;;\; t_\alpha := tt$$

$$\Gamma \;=\; PIN, t_\alpha, t_\beta : H, \quad r : L$$

$\Gamma \vdash \gamma : H, \quad \Gamma \vdash \alpha, \beta : L$ each thread is typable

Problem: if $t_\alpha = t_\beta = \mathit{ff}$, PIN is copied into r!

$\Rightarrow$ P *well-typed* but **not interference-free**.

SIF: parallel imp. languages

$P = \alpha \parallel \beta \parallel \gamma, \text{ where :}$

$\gamma :$ `if` $PIN = 0$ `then` $t_\alpha := tt$ `else` $t_\beta := tt$

$\alpha :$ `while` $t_\alpha = f\!f$ `do nil ;` $r := 1$; $t_\beta := tt$

$\beta :$ `while` $t_\beta = f\!f$ `do nil ;` $r := 0$; $t_\alpha := tt$

termination leaks
cannot be ignored
anymore

$\Gamma \; = \; PIN, t_\alpha, t_\beta : H, \quad r : L$

$\Gamma \vdash \gamma : H \, , \quad \Gamma \vdash \alpha, \beta : L$

Problem: if $t_\alpha = t_\beta = f\!f$, PIN is copied into r !

$\Rightarrow$ P *well-typed* but **not interference-free**.

SIF: parallel imp. languages

$$P = \alpha \parallel \beta \parallel \gamma, \text{ where :}$$

$$\gamma : \texttt{if } PIN = 0 \texttt{ then } t_\alpha := tt \texttt{ else } t_\beta := tt$$

$$\alpha : \underline{\texttt{while } t_\alpha = \mathit{ff} \texttt{ do nil ; } r := 1} \; ; \; t_\beta := tt$$

$$\beta : \underline{\texttt{while } t_\beta = \mathit{ff} \texttt{ do nil ; } r := 0} \; ; \; t_\alpha := tt$$

$$\Gamma \; = \; PIN, t_\alpha, t_\beta : H, \quad r : L$$

$$\Gamma \vdash \gamma : H \, , \quad \Gamma \vdash \alpha, \beta : L$$

termination leaks cannot be ignored anymore

NB Program P terminates, but depending on the value of PIN it executes $r := 1$ and $r := 0$ in a different order.

SIF: parallel imp. languages

$$P = \alpha \parallel \beta \parallel \gamma, \text{ where :}$$

$$\gamma : \texttt{if } PIN = 0 \texttt{ then } t_\alpha := tt \texttt{ else } t_\beta := tt$$
$$\alpha : \underline{\texttt{while } t_\alpha = \mathit{ff} \texttt{ do nil ; } r := 1} \; ; \; t_\beta := tt$$
$$\beta : \underline{\texttt{while } t_\beta = \mathit{ff} \texttt{ do nil ; } r := 0} \; ; \; t_\alpha := tt$$

$$\Gamma \; = \; PIN, t_\alpha, t_\beta : H, \quad r : L$$
$$\Gamma \vdash \gamma : H \,, \quad \Gamma \vdash \alpha, \beta : L$$

termination leaks cannot be ignored anymore

The termination behaviour of one thread may be modified by another thread running in parallel.

SIF: double types

Solution to deal with termination leaks

$$\texttt{while}\ x_H\ \texttt{do}\ \texttt{nil}\ ;\ y_L := \mathit{ff}$$

$$\texttt{if}\ x_H\ \texttt{then}\ \texttt{nil}\ \texttt{else}\ \texttt{loop}\ ;\ y_L := \mathit{ff}$$

Proposal by Boudol and C. [BC01], Smith [Smi01]: use double types

$$\Gamma \vdash P : (\tau, \sigma)$$

lower bound for writes

upper bound for reads

Rule for $(P_1; P_2)$: read level of P_1 $\leq$ write level of P_2

Bisimulation for PARIMP

Standard small-step semantics for PARIMP:

$$\langle P, s\rangle \rightarrow \langle P', s'\rangle$$

Bisimulation on programs: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any state s:

If $< P_1 , s > \longrightarrow < P_1' , s' >$, then there exist P_2' such that

$< P_2 , s > \longrightarrow^* < P_2' , s' >$ and $P_1' \mathscr{R} P_2'$

Bisimilarity: $P_1 \simeq P_2$ if $P_1 \mathscr{R} P_2$ for some bisimulation $\mathscr{R}$

Security for PARIMP

Standard small-step semantics for PARIMP:

$$\langle P, s\rangle \rightarrow \langle P', s'\rangle$$

Security (noninterference) is based on Low-bisimulation, an adaptation of bisimulation where instead of assuming a single observer one assumes a set of $\mathcal{L}$-observers, one for each downward-closed set $\mathcal{L}$ of security levels.

Examples: $\mathcal{L} = \{\bot\}$, $\mathcal{L} = \{\bot, private_1, private_2\}$

$\Gamma\mathcal{L}$-observation

Lattice of security levels : $(\mathcal{S}, \leq)$ $\mathcal{L} \subseteq \mathcal{S}$ downward-closed

Type environment : $\Gamma : Var \to \mathcal{S}$

$\Gamma\mathcal{L}$-observer : sees only variables of level in $\mathcal{L}$

State : $s : Var \to Val$

$\Gamma\mathcal{L}$-equality of states (indistinguishability of states by $\Gamma\mathcal{L}$-observer):

$$s_1 =^{\Gamma}_{\mathcal{L}} s_2 \quad \mathit{if} \quad \forall x \in Var \quad (\Gamma(x) \in \mathcal{L} \Rightarrow s_1(x) = s_2(x))$$

NB If $\mathcal{L} = \mathcal{S}$, then $=^{\Gamma}_{\mathcal{L}}$ reduces to state equality.

Noninterference for PARIMP

$\Gamma\mathscr{L}$-bisimulation on programs: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any pair of states s_1, s_2 such that $s_1 =^{\Gamma}_{\mathscr{L}} s_2$:

$$\text{If } < P_1 , s_1 > \longrightarrow < P_1' , s_1' >, \text{ then there exist } P_2', s_2' \text{ such that}$$

$$< P_2 , s_2 > \longrightarrow^* < P_2' , s_2' >, \text{ where } s_1' =^{\Gamma}_{\mathscr{L}} s_2' \text{ and } P_1' \mathscr{R} P_2'$$

$\Gamma\mathscr{L}$-bisimilarity: $P_1 \simeq^{\Gamma}_{\mathscr{L}} P_2$ if $P_1 \mathscr{R} P_2$ for some $\Gamma\mathscr{L}$-bisimulation $\mathscr{R}$

$\simeq^{\Gamma}_{\mathcal{L}}$: indistinguishability of programs by $\Gamma\mathcal{L}$-observer

NB If $\mathcal{L} = \mathcal{S}$, then $\simeq^{\Gamma}_{\mathcal{L}}$ reduces to ordinary bisimilarity $\simeq$

Noninterference for PARIMP

$\Gamma\mathscr{L}$-bisimulation on programs: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any pair of states s_1, s_2 such that $s_1 =^{\Gamma}_{\mathscr{L}} s_2$:

$$\text{If } < P_1 \,,\, s_1 > \longrightarrow < P_1' \,,\, s_1' >, \text{ then there exist } P_2', s_2' \text{ such that}$$

$$< P_2 \,,\, s_2 > \longrightarrow^* < P_2' \,,\, s_2' >, \text{ where } s_1' =^{\Gamma}_{\mathscr{L}} s_2' \text{ and } P_1' \mathscr{R} P_2'$$

$\Gamma\mathscr{L}$-bisimilarity: $P_1 \simeq^{\Gamma}_{\mathscr{L}} P_2$ if $P_1 \mathscr{R} P_2$ for some $\Gamma\mathscr{L}$-bisimulation $\mathscr{R}$

$\Gamma\mathscr{L}$-security: P is $\Gamma\mathscr{L}$-*secure* if $P \simeq^{\Gamma}_{\mathscr{L}} P$

A program is secure for the $\Gamma\mathcal{L}$-observer if no variation in variables outside $\mathcal{L}$ has an effect on variables inside $\mathcal{L}$

Noninterference for PARIMP

$\Gamma\mathscr{L}$-bisimulation on programs: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any pair of states s_1, s_2 such that $s_1 =^{\Gamma}_{\mathscr{L}} s_2$:

$$\text{If } < P_1 \,,\, s_1 > \longrightarrow < P_1' \,,\, s_1' >, \text{ then there exist } P_2', s_2' \text{ such that}$$

$$< P_2 \,,\, s_2 > \longrightarrow^* < P_2' \,,\, s_2' >, \text{ where } s_1' =^{\Gamma}_{\mathscr{L}} s_2' \text{ and } P_1' \mathscr{R} P_2'$$

$\Gamma\mathscr{L}$-bisimilarity: $P_1 \simeq^{\Gamma}_{\mathscr{L}} P_2$ if $P_1 \mathscr{R} P_2$ for some $\Gamma\mathscr{L}$-bisimulation $\mathscr{R}$

$\Gamma\mathscr{L}$-security: P is $\Gamma\mathscr{L}$-*secure* if $P \simeq^{\Gamma}_{\mathscr{L}} P$

Example (need for considering all sets $\mathcal{L}$)

If $\bot < \ell < \top$, then $y_\ell := x_\top$ is $\{\bot\}$-secure but not $\{\bot, \ell\}$-secure

Noninterference for PARIMP

$\Gamma\mathscr{L}$-bisimulation on programs: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any pair of states s_1, s_2 such that $s_1 =^{\Gamma}_{\mathscr{L}} s_2$:

$$\text{If } < P_1 \, , \, s_1 > \longrightarrow < P_1' \, , \, s_1' >, \text{ then there exist } P_2', s_2' \text{ such that}$$

$$< P_2 \, , \, s_2 > \longrightarrow^* < P_2' \, , \, s_2' >, \text{ where } s_1' =^{\Gamma}_{\mathscr{L}} s_2' \text{ and } P_1' \mathscr{R} P_2'$$

$\Gamma\mathscr{L}$-bisimilarity: $P_1 \simeq^{\Gamma}_{\mathscr{L}} P_2$ if $P_1 \mathscr{R} P_2$ for some $\Gamma\mathscr{L}$-bisimulation $\mathscr{R}$

$\Gamma\mathscr{L}$-security: P is $\Gamma\mathscr{L}$-*secure* if $P \simeq^{\Gamma}_{\mathscr{L}} P$

A program is Γ-secure if it is $\Gamma\mathcal{L}$-secure for every $\mathcal{L}$

NB In the following Γ will be generally omitted.

SIF: process calculi

- Subjects = processes. Objects = channels $a, b, c \ldots$

$a_H(x).\,\overline{b}_L\langle x\rangle$ — not secure

- Data flow and control flow are closely intertwined:

$a_H(x).\,\overline{b}_L\langle v\rangle \qquad a_H(x).\,\overline{b}_L \qquad a_H.\,\overline{b}_L\langle v\rangle$ — secure?

Warning ! Can be used to implement indirect insecure flows:

$$(a_H(x).\,\texttt{if}\ x\ \texttt{then}\ \overline{c}_H\ \texttt{else}\ \overline{d}_H \mid (\,c_H.\,\overline{b}_L\langle 0\rangle + d_H.\,\overline{b}_L\langle 1\rangle\,))\setminus\{c_H, d_H\}$$

CCS with security

Simple security (BNDC) [Focardi-Gorrieri'01]

Channels are partitioned into high channels $\mathcal{H}$ and low channels $\mathcal{L}$.

$\mathcal{P}r^{\mathcal{H}}_{\mathsf{syn}}$: set of syntactically high processes, with all channels in $\mathcal{H}$.

Bisimulation-based Non Deducibility on Compositions (BNDC)

P is secure with respect to $\mathcal{H}$, $P \in \mathsf{BNDC}_{\mathcal{H}}$, if for every $\Pi \in \mathcal{P}r^{\mathcal{H}}_{\mathsf{syn}}$:

$$(\nu\mathcal{H})(P \mid \Pi) \approx (\nu\mathcal{H})P$$

Examples.

$a_H \,.\, b_L$	$a_H + b_L$	not secure
$a_H \mid b_L$	$a_H \,.\, b_L + b_L$	secure

Choosing $\Pi = \overline{a_H}$ for the first two, we get $(\nu\mathcal{H})(P \mid \Pi) \not\approx (\nu\mathcal{H})P$.

CCS with security

Simple security (BNDC) [Focardi-Gorrieri'01]

Channels are partitioned into high channels $\mathcal{H}$ and low channels $\mathcal{L}$.

$\mathcal{P}r^{\mathcal{H}}_{\text{syn}}$: set of syntactically high processes, with all channels in $\mathcal{H}$.

Bisimulation-based Non Deducibility on Compositions (BNDC)

P is secure with respect to $\mathcal{H}$, $P \in \mathsf{BNDC}_{\mathcal{H}}$, if for every $\Pi \in \mathcal{P}r^{\mathcal{H}}_{\text{syn}}$:

$$(\nu\mathcal{H})(P \mid \Pi) \approx (\nu\mathcal{H})P$$

Examples.

$a_H \,.\, b_L$	$a_H + b_L$	not secure	occurrence of a_H depends on high environment
$a_H \mid b_L$	$a_H \,.\, b_L + b_L$	secure	

Choosing $\Pi = \overline{a_H}$ for the first two, we get $(\nu\mathcal{H})(P \mid \Pi) \not\approx (\nu\mathcal{H})P$.

2 sources of insecurity: in $a_H \,.\, b_L$ occurrence of a_H enables b_L
in $a_H + b_L$ occurrence of a_H discards b_L

CCS with security

Several other NI properties (mostly surveyed in FG05)

- "Venice school": Focardi and Gorrieri [FG01], Focardi and Rossi [FR02], Bossi, Focardi, Piazza and Rossi [BFPR04], Focardi, Rossi and Sabelfeld [FRS05], ...
- Castellani [Cas07]

NB All references are given at the end of the talk

pi-calculus with security

A variety of approaches:

- Honda, Vasconcelos, Yoshida [HVY00], Honda and Yoshida [HY02], [HY07]
- Pottier [Pot02]
- Hennessy and Riely [HR02], Hennessy [Hen04]
- Crafa and Rossi [CR05]
- Kobayashi [Kob05]

Mostly for pi-calculus with synchronous communication

Part 2
Security and Types

Back to sessions

Our approach: mix of classical LBS approach
and process calculi approaches

Sessions with asynchronous communication
=> messages stored in queues

Bisimulation equivalence: queues are the "observables"
-> play the role of memories in classical LBS approach

Tracking information leaks

1st kind of leak: high input followed by low action

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle$$

in some initiated session s, participant 1 waits for a top level value from participant 2

then participant 1 sends a bottom level value to participant 3

Security levels for variables and values, not for session channels (more on this later)

Tracking information leaks

1st kind of leak: high input followed by low action

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle$$

Insecure because:

- if the high environment provides a value for $x^{\top}$ then the low observer sees $\mathsf{true}^{\perp}$
- otherwise, the process is blocked and the low observer sees the empty behaviour

Tracking information leaks

1st kind of leak: high input followed by low action

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle$$

occurrence of input depends on high environment

Lock (blocked input) => new kind of termination leak

cf Dezani's lecture

3 ways to track leaks

1st kind of leak: high input followed by low action

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle$$

- Typability (prevention): any "syntactic leak" is bad ✗
- Safety (local detection): any "semantic leak" is bad ✗
- Security (global detection): any "global semantic leak", detectable by observing the overall process, is bad ✗

Rejected by all analyses, both static and semantic

Syntactic vs semantic leaks

What if the execution never reaches the leak ?

$$\nu(a)(a[1](\alpha).\ s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle)$$

Syntactic vs semantic leaks

What if the execution never reaches the leak ?

$$\nu(a)(a[1](\alpha).\ s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle)$$

- Typability (prevention): no syntactic leak ✘

Syntactic vs semantic leaks

What if the execution never reaches the leak ?

$$\nu(a)(a[1](\alpha).\ s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle)$$

- Typability (prevention): no syntactic leak ✗
- Safety (local detection): no local semantic leak ✓
- Security (global detection): no global semantic leak ✓

Syntactic vs semantic leaks

What if the execution never reaches the leak ?

$$\nu(a)(a[1](\alpha).\ s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp}\rangle)$$

- Typability (prevention): no syntactic leak ✗
- Safety (local detection): no local semantic leak ✓
- Security (global detection): no global semantic leak ✓

Level drop in dead code does not appear at semantic level

Local vs global semantic leaks

2nd kind of leak: high conditional with ≠ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]$$

$$|\ \ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

Local vs global semantic leaks

2nd kind of leak: high conditional with ≠ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if}\ x^{\top}\ \text{then}\ s[1]!\langle 3, \text{true}^{\perp}\rangle\ \text{else}\ s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]$$

$$|\ \ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

Since participant 2 sends a value to participant 1, the input on s[1] is guaranteed to occur.

Depending on whether $x^{\top}$ is true or false, the low observer will see two different values.

Local vs global semantic leaks

2nd kind of leak: high conditional with $\neq$ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]$$

$$|\ \ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

Since participant 2 sends a value to participant 1, the input on s[1] is guaranteed to occur.

Depending on whether $x^{\top}$ is true or false, the low observer will see two different values.

Classical example of implicit information flow in conditionals

Local vs global semantic leaks

2nd kind of leak: high conditional with $\neq$ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \mathsf{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \mathsf{false}^{\perp}\rangle\ ]$$

$$\mid\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

Since participant 2 sends a value to participant 1, the input on s[1] is guaranteed to occur.

Depending on whether $x^{\top}$ is true or false, the low observer will see two different values.

Warning: this example holds for synchronous communication. More care has to be taken for asynchronous communication.

Local vs global semantic leaks

2nd kind of leak: high conditional with ≠ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]$$

$$|\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

asynchronous communication
=> messages stored in queues

"high part" of the queue may be changed/increased/decreased between send and receive (=> message of 2 may be withdrawn!)

=> the input on s[1] is actually not guaranteed. In asynchronous case, even this seemingly well-behaved process is insecure:

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \text{true}^{\perp}\rangle\ \ |\ s[2]!\langle 1, v^{\top}\rangle$$

Local vs global semantic leaks

2nd kind of leak: high conditional with ≠ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]$$

$$\mid\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]$$

asynchronous communication
=> messages stored in queues

"high part" of the queue may be changed/increased/decreased between send and receive (=> message of 2 may be withdrawn!)

=> the input on s[1] is actually not guaranteed. In asynchronous case, even this seemingly well-behaved process is insecure:

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \text{true}^{\perp}\rangle \ \mid\ s[2]!\langle 1, v^{\top}\rangle$$

needs to be persistent

Local vs global semantic leaks

2nd kind of leak: high conditional with $\neq$ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]^{\infty}$$

$$|\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]^{\infty}$$

persistent output

asynchronous communication
=> messages stored in queues

"high part" of the queue may be changed/increased/decreased between send and receive (=> message of 2 may be withdrawn!)

Notation

P^{∞}: a new copy of P is grafted at the end of each branch

Local vs global semantic leaks

2nd kind of leak: high conditional with $\neq$ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \text{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]^{\infty}$$

$$|\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]^{\infty}$$

asynchronous communication
=> messages stored in queues

Since 2 is persistently sending a message to 1, the input on s[1] is guaranteed to occur.

Since high messages may be changed/added/subtracted in the queue, 1 can input different values for $x^{\top}$ and the low observer will see two different values.

Local vs global semantic leaks

2nd kind of leak: high conditional with ≠ low branches

$$[\ s[1]?(2, x^{\top}).\ \text{if}\ x^{\top}\ \text{then}\ s[1]!\langle 3, \text{true}^{\perp}\rangle\ \text{else}\ s[1]!\langle 3, \text{false}^{\perp}\rangle\ ]^{\infty}$$

$$|\ \ [\ s[2]!\langle 1, v^{\top}\rangle\ ]^{\infty}$$

- Typability (prevention): no syntactic leak ✗
- Safety (local detection): no semantic leak ✗
- Security (global detection): no global semantic leak ✗

Local vs global semantic leaks

What if the high conditional has equal low branches?

$$[\ s[1]?(2, x^{\top}).\ \text{if } x^{\top} \text{ then } s[1]!\langle 3, \mathsf{true}^{\perp}\rangle \text{ else } s[1]!\langle 3, \mathsf{true}^{\perp}\rangle\ ]^{\infty}$$

$$\mid\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]^{\infty}$$

- Typability (prevention): no syntactic leak ✗
- Safety (local detection): no local semantic leak ✗
- Security (global detection): no global semantic leak ✓

The $\perp$-observer sees no difference between the branches

Multiparty sessions

[Honda, Yoshida, Carbone POPL'08]

Multiparty session: activation of an n-ary service a

$$\underbrace{\bar{a}[n]}_{\text{arity}} \mid a[1](\alpha_1).P_1 \mid \cdots \mid a[n](\alpha_n).P_n$$

roles

initiator $\bar{a}[n]$: starts a new session on service a when there are n suitable participants

Multiparty sessions

[Honda, Yoshida, Carbone POPL'08]

Multiparty session: activation of an n-ary service a

$$\bar{a}[n] \mid a[1](\alpha_1).P_1 \mid \cdots \mid a[n](\alpha_n).P_n \longrightarrow$$

$$(\nu s) < P_1\{s[1]/\alpha_1\} \mid ... \mid P_n\{s[n]/\alpha_n\} \, , \, s : \varepsilon >$$

initiator $\bar{a}[n]$: starts a new session on service a
when there are n suitable participants

Security session calculus

- Security levels ℓ, ℓ', forming a finite lattice $(\mathscr{S}, \leq)$.
- Services a^ℓ, b^ℓ, with an *arity* n and a security level ℓ.
- Sessions s, s' (activations of services). At n-ary session initiation, creation of private name s and channels with role $s[\mathrm{p}]$, $\mathrm{p} \in \{1, \ldots, n\}$.

value	v	$::=$	$\mathsf{true} \mid \mathsf{false} \mid \ldots$
expression	e	$::=$	$x^\ell \mid v^\ell \mid \mathsf{not}\ e \mid e\ \mathsf{and}\ e' \mid \ldots$
channel	c	$::=$	$\alpha \mid s[\mathrm{p}]$

Security session calculus

- Security levels ℓ, ℓ', forming a finite lattice $(\mathscr{S}, \leq)$.
- Services a^ℓ, b^ℓ, with an *arity* n and a security level ℓ.
- Sessions s, s' (activations of services). At n-ary session initiation, creation of private name s and channels with role $s[\mathrm{p}]$, $\mathrm{p} \in \{1, \ldots, n\}$.

$$
\begin{array}{llll}
\text{value} & v & ::= & \mathsf{true} \mid \mathsf{false} \mid \ldots \\
\text{expression} & e & ::= & x^\ell \mid v^\ell \mid \mathsf{not}\ e \mid e\ \mathsf{and}\ e' \mid \ldots \\
\text{channel} & c & ::= & \alpha \mid s[\mathrm{p}]
\end{array}
$$

Security levels for variables and values, not for session channels (because participants use the same channel for all interactions)

Syntax: processes

P	$::=$	$\overline{a}^{\ell}[n]$	n-ary session initiator
	$\mid$	$a^{\ell}[\mathrm{p}](\alpha).P$	p-th session participant
	$\mid$	$c!\langle \Pi, e\rangle.P$	value send
	$\mid$	$c?(\mathrm{p}, x^{\ell}).P$	value recv
	$\mid$	$c \oplus^{\ell} \langle \Pi, \lambda\rangle.P$	selection
	$\mid$	$c \&^{\ell}(\mathrm{p}, \{\lambda_i : P_i\}_{i\in I})$	branching
	$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
	$\mid$	$\mathbf{0} \mid P \mid Q \mid (\nu a^{\ell})P \mid \ldots$	π-calculus ops

Syntax: processes

$$
\begin{array}{lcll}
P & ::= & \overline{a}^{\ell}[n] & n\text{-ary session initiator} \\
 & | & a^{\ell}[p](\alpha).P & \text{p-th session participant} \\
 & | & c!\langle\Pi, e\rangle.P & \text{value send} \\
 & | & c?(\mathrm{p}, x^{\ell}).P & \text{value recv} \\
 & | & c\oplus^{\ell}\langle\Pi, \lambda\rangle.P & \text{selection} \\
 & | & c\&^{\ell}(\mathrm{p}, \{\lambda_i : P_i\}_{i\in I}) & \text{branching} \\
 & | & \text{if } e \text{ then } P \text{ else } Q & \text{conditional} \\
 & | & \mathbf{0} \mid P \mid Q \mid (\nu a^{\ell})P \mid \ldots & \pi\text{-calculus ops}
\end{array}
$$

Security levels on services (shared channels) and choice operators are needed to deal with indirect leaks (see examples later on)

Syntax: processes

P	$::=$	$\overline{a}^{\ell}[n]$	n-ary session initiator
	$\mid$	$a^{\ell}[p](\alpha).P$	p-th session participant
	$\mid$	$c!\langle\Pi, e\rangle.P$	value send
	$\mid$	$c?(\mathrm{p}, x^{\ell}).P$	value recv
	$\mid$	$c\oplus^{\ell}\langle\Pi, \lambda\rangle.P$	selection
	$\mid$	$c\&^{\ell}(\mathrm{p}, \{\lambda_i : P_i\}_{i\in I})$	branching
	$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
	$\mid$	$\mathbf{0} \mid P \mid Q \mid (\nu a^{\ell})P \mid \ldots$	π-calculus ops

Security and types are studied in [CCD14a] for a more general calculus, with delegation and declassification.

Runtime syntax: queues

Asynchronous communication: messages stored in queues

H	$::=$	$H \cup \{s : h\} \mid \emptyset$	**Q**-set
h	$::=$	$m \cdot h \mid \varepsilon$	queue
m	$::=$	$(\mathrm{p}, \Pi, \vartheta)$	message in transit
ϑ	$::=$	$v^{\ell} \mid \lambda^{\ell}$	message content

Independent message commutation:

$$(\mathrm{p}, \Pi, \vartheta) \cdot (\mathrm{p}', \Pi', \vartheta') \cdot h \equiv (\mathrm{p}', \Pi', \vartheta') \cdot (\mathrm{p}, \Pi, \vartheta) \cdot h$$

$$\text{if } \mathrm{p} \neq \mathrm{p}' \text{ or } \Pi \cap \Pi' = \emptyset$$

Semantics: configurations

In the semantics, **Q**-sets will be the observable part of process behaviour

$\Rightarrow$ need to be separated from the rest of the process.

Configurations $C ::= <P\,,\,H> \;\mid\; (\nu\tilde{r})<P\,,\,H> \;\mid\; C \,\|\, C$

Reduction semantics:

transitions of the form $<P\,,\,H> \longrightarrow (\nu\tilde{r})<P'\,,\,H'>$

Semantics: computational rules

Session initiation:

$$a^{\ell}[1](\alpha_1).P_1 \mid \ldots \mid a^{\ell}[n](\alpha_n).P_n \mid \bar{a}^{\ell}[n] \longrightarrow$$

$$(\nu s) < P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\} \; , \; s : \varepsilon > \qquad \text{[Link]}$$

Value exchange:

$$< s[\mathrm{p}]!\langle \Pi, e \rangle.P \; , \; s : h > \longrightarrow < P \; , \; s : h \cdot (\mathrm{p}, \Pi, v^{\ell}) > \qquad (e \downarrow v^{\ell}) \qquad \text{[Send]}$$

$$< s[\mathrm{q}]?(\mathrm{p}, x^{\ell}).P \; , \; s : (\mathrm{p}, \mathrm{q}, v^{\ell}) \cdot h > \longrightarrow < P\{v^{\ell}/x^{\ell}\} \; , \; s : h > \qquad \text{[Rec]}$$

Semantics: choice

Selection / branching:

$$< s[\mathrm{p}] \oplus^{\ell} \langle \Pi, \lambda_k \rangle .P \;,\; s : h > \longrightarrow < P \;,\; s : h \cdot (\mathrm{p}, \Pi, \lambda_k{}^{\ell}) > \qquad \text{[Label]}$$

$$< s[\mathrm{q}] \&^{\ell} (\mathrm{p}, \{\lambda_i : P_i\}_{i \in I}) \;,\; s : (\mathrm{p}, \mathrm{q}, \lambda_k{}^{\ell}) \cdot h > \longrightarrow < P_k \;,\; s : h > \quad (k \in I) \qquad \text{[Branch]}$$

Security

Observation defined as usual wrt a downward-closed set of levels $\mathscr{L}$.

What is $\mathscr{L}$-observable in $(\nu\tilde{r}) < P\ ,\ H >$? Messages of level $\ell \in \mathscr{L}$ in H.

$\Longrightarrow$ session queues play the role of memories in imperative languages

$\mathscr{L}$-projection of $\mathbf{Q}$-sets

$$(\mathrm{p}, \Pi, \vartheta) \Downarrow \mathscr{L} = \begin{cases} (\mathrm{p}, \Pi, \vartheta) & \text{if } lev(\vartheta) \in \mathscr{L} \\ \varepsilon & \text{otherwise} \end{cases}$$

extended pointwise to named queues and $\mathbf{Q}$-sets (NB: $s : \varepsilon$ not observed)

$\mathscr{L}$-equality of $\mathbf{Q}$-sets: $H =_{\mathscr{L}} K$ if $H \Downarrow \mathscr{L} = K \Downarrow \mathscr{L}$

Security of processes

$\mathscr{L}$-bisimulation on processes: symmetric relation $\mathscr{R}$ such that $P_1 \mathscr{R} P_2$ implies, for any pair of monotone H_1, H_2 such that $H_1 =_{\mathscr{L}} H_2$ and each $< P_i \ , \ H_i >$ is saturated:

If $< P_1 \ , \ H_1 > \longrightarrow (\nu \tilde{r}) < P_1' \ , \ H_1' >$, then there exist P_2', H_2' such that

$< P_2 \ , \ H_2 > \longrightarrow^* \equiv (\nu \tilde{r}) < P_2' \ , \ H_2' >$, where $H_1' =_{\mathscr{L}} H_2'$ and $P_1' \mathscr{R} P_2'$

$\mathscr{L}$-equivalence: $P_1 \simeq_{\mathscr{L}} P_2$ if $P_1 \mathscr{R} P_2$ for some $\mathscr{L}$-bisimulation $\mathscr{R}$

$\mathscr{L}$-security: P is $\mathscr{L}$-*secure* if $P \simeq_{\mathscr{L}} P$

Security: P is *secure* if it is $\mathscr{L}$-*secure* for any $\mathscr{L}$

Examples of information leaks

High input followed by low action

(*) $$s[2]?(1,x^{\top}).\text{if } x^{\top} \text{ then } s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0} \text{ else } \mathbf{0}$$
$$\mid\ s[3]?(2,z^{\top}).s[3]!\langle 4,\text{true}^{\perp}\rangle.\mathbf{0} \mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

(*) Assuming input on s[2] to be guaranteed by persistent output on s[1].
Same hypothesis in the following series of examples.

Examples of information leaks

High input followed by low action

1st thread
not session typable!

$$s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0}\ \mathsf{else}\ \mathbf{0}$$

$$\mid\ s[3]?(2,z^\top).s[3]!\langle 4,\mathsf{true}^\perp\rangle.\mathbf{0}\ \mid\ s[4]?(3,y^\perp).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

Session types => same interactive behaviour in the two branches

Examples of information leaks

High input followed by low action

1st thread
not session typable!

$$s[2]?(1,x^{\top}).\text{if } x^{\top} \text{ then } s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0} \text{ else } \mathbf{0}$$

$$\mid\ s[3]?(2,z^{\top}).s[3]!\langle 4,\text{true}^{\perp}\rangle.\mathbf{0} \mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

Session types => same interactive behaviour in the two branches

=> Session types help preventing indirect leaks

Examples of information leaks

High input followed by low action

$$s[2]?(1,x^{\top}).\text{if } x^{\top} \text{ then } s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0} \text{ else } P^{\infty}$$

$$\mid\ s[3]?(2,z^{\top}).s[3]!\langle 4,\text{true}^{\perp}\rangle.\mathbf{0} \mid s[4]?(3,y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

P^{∞}: some infinite sequential behaviour

Examples of information leaks

High input followed by low action

1st thread
not session typable

$$s[2]?(1,x^\top).\text{if } x^\top \text{ then } s[2]!\langle 3,\text{true}^\top\rangle.\mathbf{0} \text{ else } P^\infty$$

$$\mid\ s[3]?(2,z^\top).s[3]!\langle 4,\text{true}^\perp\rangle.\mathbf{0} \mid s[4]?(3,y^\perp).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

P^∞: some infinite sequential behaviour

Examples of information leaks

High input followed by low action

1st thread
not session typable

$$s[2]?(1,x^{\top}).\text{if } x^{\top} \text{ then } s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0} \text{ else } P^{\infty}$$

$$\mid\ s[3]?(2,z^{\top}).s[3]!\langle 4,\text{true}^{\perp}\rangle.\mathbf{0} \mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

P^{∞}: some infinite sequential behaviour

Session types help uniformising termination behaviours of branches
=> they help preventing classical termination leaks

Examples of information leaks

High input followed by low action

session typable

$$s[2]?(1,x^{\top}).\text{if } x^{\top} \text{ then } s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0} \text{ else } (\nu b^{\ell})b^{\ell}[1](\beta).s[2]!\langle 3,\text{true}^{\top}\rangle.\mathbf{0}$$

$$\mid\ s[3]?(2,z^{\top}).s[3]!\langle 4,\text{true}^{\perp}\rangle.\mathbf{0} \mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

deadlock!

Session types: not enough to prevent all termination leaks => need to strengthen them with constraints for deadlock-freedom

NB This example shows that, unless we have deadlock freedom, we cannot avoid the security requirement in the rule for input

Need for levels on services

Service calls may induce (insecure) information flows

$$s[2]?(1, x^{\top}).\mathsf{if}\ x^{\top}\ \mathsf{then}\ \bar{b}[2]\ \mathsf{else}\ \mathbf{0}$$

$$\mid\ b[1](\beta_1).\beta_1!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0} \mid\ b[2](\beta_2).\beta_2?(1, y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

Need for levels on services

Service calls may induce (insecure) information flows

$\Longrightarrow$ necessary to add security levels on services

$$s[2]?(1, x^{\top}).\text{if } x^{\top} \text{ then } \bar{b}^{?}[2] \text{ else } \mathbf{0}$$

$$\mid\ b^{?}[1](\beta_1).\beta_1!\langle 2, \text{true}^{\perp}\rangle.\mathbf{0} \mid\ b^{?}[2](\beta_2).\beta_2?(1, y^{\perp}).\mathbf{0}$$

No possible security level for b making this process typable.

Adding levels on services rules out this kind of indirect leak

Need for levels on choice/labels

Selections may induce (insecure) information flows

$$s[2]?(1,x^{\top}).\mathsf{if}\ x^{\top}\ \mathsf{then}\ s[2]\oplus\langle 3,\lambda\rangle.\mathbf{0}\ \mathsf{else}\ s[2]\oplus\langle 3,\lambda'\rangle.\mathbf{0}$$
$$\mid\ s[3]\&(2,\{\lambda : s[3]!\langle 4,\mathsf{true}^{\perp}\rangle.\mathbf{0},\lambda' : s[3]!\langle 4,\mathsf{false}^{\perp}\rangle.\mathbf{0}\})$$
$$\mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

Insecure process: low level value exchange depending on high test

No possible security level for λ,λ' that allows typing this process.

Adding levels on choice and labels
rules out this kind of indirect leak

Type system

Service type: G^{ℓ}, where

- G is a global type, describing the whole protocol of the service
- ℓ is the meet of all security levels appearing in G

$$
\begin{array}{llll}
\text{Global} & G & ::= & \mathsf{p} \rightarrow \Pi : \langle S^{\ell} \rangle . G \\
 & & \mid & \mathsf{p} \rightarrow \Pi : \{\lambda_i : G_i\}^{\ell}_{i \in I} \\
 & & \mid & \mu \mathbf{t}.G \mid \mathbf{t} \mid \mathsf{end}
\end{array}
$$

$$
\begin{array}{llll}
\text{Sorts} & S & ::= & \mathsf{bool} \mid \ldots
\end{array}
$$

Type system

Session type: describes a participant's contribution to the session.

$$
\begin{array}{llll}
T & ::= & !\langle \Pi, S^{\ell} \rangle ; T & \mid \quad ?(\mathrm{p}, S^{\ell}) ; T \\
 & \mid & \oplus^{\ell} \langle \Pi, \{\lambda_i : T_i\}_{i \in I} \rangle & \mid \quad \&^{\ell} (\mathrm{p}, \{\lambda_i : T_i\}_{i \in I}) \\
 & \mid & \mu \mathbf{t}.T & \mid \quad \mathbf{t} \\
 & \mid & \mathsf{end} &
\end{array}
$$

Typing rules for processes

Typing judgments for processes:

$$\Gamma \vdash_{\ell} P \triangleright \Delta$$

- Γ (standard type environment) maps variables to sort types or service types and services to service types
- Δ (process environment) maps session channels to session types
- security level ℓ is a lower bound for all levels in communications (input/output or selection/branching) of P

Some typing rules

usual subtyping
for security

$$\frac{\Gamma \vdash_{\ell} P \triangleright \Delta \quad \ell' \leq \ell}{\Gamma \vdash_{\ell'} P \triangleright \Delta} \lfloor \textsc{SubS} \rfloor$$

$$\frac{\Gamma, u : G^{\ell} \vdash_{\ell} P \triangleright \Delta, \alpha : G \upharpoonright \mathrm{p}}{\Gamma, u : G^{\ell} \vdash_{\ell} u[\mathrm{p}](\alpha).P \triangleright \Delta} \lfloor \textsc{MAcc} \rfloor$$

Typing rule for I/O

not a constraint, since
one can take $\ell' = \bot$

$$\frac{\Gamma \vdash e : S^{\ell} \quad \Gamma \vdash_{\ell'} P \triangleright \Delta, c : T \quad \ell' \leq \ell}{\Gamma \vdash_{\ell'} c!\langle \Pi, e \rangle.P \triangleright \Delta, c : !\langle \Pi, S^{\ell} \rangle; T} \lfloor \text{SEND} \rfloor$$

real constraint, since
type of x^{ℓ} is invariant

$$\frac{\Gamma, x^{\ell} : S^{\ell} \vdash_{\ell} P \triangleright \Delta, c : T}{\Gamma \vdash_{\ell} c?(\mathrm{p}, x^{\ell}).P \triangleright \Delta, c :?(\mathrm{p}, S^{\ell}); T} \lfloor \text{RCV} \rfloor$$

Analogies with PARIMP

Rule $\lfloor \text{RCV} \rfloor$ for input prefix

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp} \rangle \qquad ✗$$

input prefix level $\leq$ communication level of P

Rule for sequential composition

$$P_1 \,;\, P_2 = \ (\mathtt{while}\ x^{\top} \mathtt{do\ nil}) \,;\, y^{\perp} := \mathsf{true} \qquad ✗$$

read level of P_1 $\leq$ write level of P_2

Analogies with PARIMP

Rule $\lfloor \text{Rcv} \rfloor$ for input prefix

termination leak

$$s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\perp} \rangle$$

✘

input prefix level $\leq$ communication level of P

Rule for sequential composition

termination leak

$$P_1\,; P_2 = \ (\texttt{while}\ x^{\top}\,\texttt{do nil})\,;\ y^{\perp} := \mathsf{true}$$

✘

read level of P_1 $\leq$ write level of P_2

Typing rule for conditional

$$\begin{cases} \text{Usual session type requirement: equal session types for branches} \\ \text{Usual security requirement: equal security levels for test and branches} \end{cases}$$

$$\frac{\Gamma \vdash e : \mathsf{bool}^{\ell} \quad \Gamma \vdash_{\ell} P \triangleright \Delta \quad \Gamma \vdash_{\ell} Q \triangleright \Delta}{\Gamma \vdash_{\ell} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

Typing rule for conditional

Usual session type requirement: equal session types for branches

Usual security requirement: equal security levels for test and branches

$$\frac{\Gamma \vdash e : \mathsf{bool}^{\ell} \quad \Gamma \vdash_{\ell} P \triangleright \Delta \quad \Gamma \vdash_{\ell} Q \triangleright \Delta}{\Gamma \vdash_{\ell} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

In combination with [Rcv], this rule can be relaxed, by allowing any level ℓ' for the tested expression.

Typing rule for conditional

$\begin{cases} \text{Usual session type requirement: equal session types for branches} \\ \text{Usual security requirement: equal security levels for test and branches} \end{cases}$

$$\frac{\Gamma \vdash e : \mathsf{bool}^{\ell} \quad \Gamma \vdash_{\ell} P \triangleright \Delta \quad \Gamma \vdash_{\ell} Q \triangleright \Delta}{\Gamma \vdash_{\ell} \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta}$$

In combination with [Rcv], this rule can be relaxed, by allowing any level ℓ' for the tested expression.

$$s[1]?(2, x^{\top}).\ \mathsf{if}\ x^{\top}\ \mathsf{then}\ s[1]!\langle 3, \mathsf{true}^{\perp}\rangle\ \mathsf{else}\ s[1]!\langle 3, \mathsf{false}^{\perp}\rangle$$

already excluded by Rule [Rcv]

Soundness

Soundness of the type system

If P is typable, then $P \simeq_{\mathscr{L}} P$ for all downward-closed $\mathscr{L}$.

Soundness

Soundness of the type system

If P is typable, then $P \simeq_{\mathscr{L}} P$ for all downward-closed $\mathscr{L}$.

Secure but not typable processes:

$$[\ s[1]?(2, x^{\top})\ .\ s[1]!\langle 2, \mathsf{true}^{\perp}\rangle\ ]^{\infty}\ \mid\ [\ s[2]!\langle 1, v^{\top}\rangle\ .\ s[2]?(1, y^{\perp})\ ]^{\infty}$$

A local insecurity may be sanitised by its context

Soundness

Soundness of the type system

If P is typable, then $P \simeq_{\mathscr{L}} P$ for all downward-closed $\mathscr{L}$.

Secure but not typable processes:

$$[\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle \,]^{\infty} \;\mid\; [\, s[2]!\langle 1, v^{\top}\rangle \,.\, s[2]?(1, y^{\perp}) \,]^{\infty}$$

A local insecurity may be sanitised by its context

$$\nu(a)(a[1](\alpha).\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle \,)$$

deadlock

$$[\, s[1]?(2, x^{\top}).\ \mathsf{if}\ x^{\top}\ \mathsf{then}\ s[1]!\langle 3, \mathsf{true}^{\perp}\rangle\ \mathsf{else}\ s[1]!\langle 3, \mathsf{true}^{\perp}\rangle \,]^{\infty}$$

$$\mid\ [\, s[2]!\langle 1, v^{\top}\rangle \,]^{\infty}$$

secure high conditional

previously discussed examples

Compositionality issues

Security is not decompositional:

a secure program may have insecure components

secure but not typable:

$$[\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle \,]^{\infty} \;\mid\; [\, s[2]!\langle 1, v^{\top}\rangle \,.\, s[2]?(1, y^{\perp}) \,]^{\infty}$$

A local insecurity may be sanitised by its context

Security is not compositional:

the composition of secure programs may be insecure

another example of deadlock, secure but not typable:

$$\bar{a}^{\perp}[2] \;\mid\; \underline{a^{\perp}[1](\alpha_1) \,.\, b^{\perp}[1](\beta_1)} \,.\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle$$

$$\mid\; \bar{b}^{\perp}[2] \;\mid\; \underline{b^{\perp}[2](\beta_2) \,.\, a^{\perp}[2](\alpha_2)} \,.\, \mathbf{0}$$

(solvable) deadlock due to inverse service calls

Compositionality issues

Security is not decompositional:

secure but not typable:

$$[\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle \,]^{\infty} \;\mid\; [\, s[2]!\langle 1, v^{\top}\rangle \,.\, s[2]?(1, y^{\perp}) \,]^{\infty}$$

A local insecurity may be sanitised by its context

Security is not compositional:

another example of deadlock, secure but not typable:

$$\begin{aligned} &\bar{a}^{\perp}[2] \;\mid\; \underline{a^{\perp}[1](\alpha_1) \,.\, b^{\perp}[1](\beta_1)} \,.\, s[1]?(2, x^{\top}) \,.\, s[1]!\langle 2, \mathsf{true}^{\perp}\rangle \\ \mid\; &\bar{b}^{\perp}[2] \;\mid\; b^{\perp}[2](\beta_2) \,.\, a^{\perp}[2](\alpha_2) \,.\, \mathbf{0} \\ &\qquad\;\;\mid\; \underline{a^{\perp}[2](\alpha_2) \,.\, b^{\perp}[2](\beta_2)} \,.\, \mathbf{0} \end{aligned}$$

deadlock solved by
adding a component
=> insecurity appears

Part 3
Information Flow Safety

Monitored semantics

Idea: lift to the semantic level the requirements of the security type system.

Technique: each parallel component is controlled by a monitor, which records the level of inputs along the component's computation and checks its subsequent communications against this level.

=> blocks execution when a local leak is detected.

Monitored semantics

Monitored processes (where $\mu \in \mathscr{S}$):

$$M ::= P^{\rceil \mu} \mid M \mid M \mid (\nu \tilde{r})M \mid \mathsf{def}\ D\ \mathsf{in}\ M$$

Monitored transitions

$$< M\ ,\ H > \multimap\!\!\rightarrow (\nu \tilde{s}) < M'\ ,\ H' >$$

Error predicate

$$< M\ ,\ H > \dagger$$

New structural rules:

$$(P_1 \mid P_2)^{\rceil \mu} \equiv P_1^{\rceil \mu} \mid P_2^{\rceil \mu} \qquad\qquad C\dagger \ \wedge\ C \equiv C' \quad \Longrightarrow \quad C'\dagger$$

Monitored semantics rules

Conditional:

$$\text{if } e \text{ then } P \text{ else } Q^{\rceil\mu} \multimap P^{\rceil\mu} \qquad \text{if } e \downarrow \text{true}^{\ell}$$

$$\text{if } e \text{ then } P \text{ else } Q^{\rceil\mu} \multimap Q^{\rceil\mu} \qquad \text{if } e \downarrow \text{false}^{\ell}$$

Value input:

$$\text{if } \mu \leq \ell \quad \text{then} < s[\mathrm{q}]?(\mathrm{p}, x^{\ell}).P^{\rceil\mu} \;, s : (\mathrm{p}, \mathrm{q}, v^{\ell}) \cdot h > \multimap < P\{v/x\}^{\rceil\ell} \;, s : h >$$

$$\text{else} < s[\mathrm{q}]?(\mathrm{p}, x^{\ell}).P^{\rceil\mu} \;, s : (\mathrm{p}, \mathrm{q}, v^{\ell}) \cdot h > \dagger$$

Security requirements of typing rules lifted to semantic rules
=> only checked in reachable states of processes.

Monitored semantics rules (ctd)

Session initiation:

$$a^{\ell}[1](\alpha_1).P_1^{\rceil\mu_1} \mid \ldots \mid a^{\ell}[n](\alpha_n).P_n^{\rceil\mu_n} \mid \bar{a}^{\ell}[n]^{\rceil\mu_{n+1}} \multimap\!\!\!\longrightarrow$$

$$(\nu s) < P_1\{s[1]/\alpha_1\}^{\rceil\ell} \mid \ldots \mid P_n\{s[n]/\alpha_n\}^{\rceil\ell}\ ,\ s:\varepsilon >$$

$$\text{if } \bigsqcup_{i\in\{1\ldots n+1\}} \mu_i \leq \ell$$

Example

$$s[2]?(1,x^{\top}).\mathsf{if}\ x^{\top}\ \mathsf{then}\ \bar{b}^{\ell}[2]\ \mathsf{else}\ \mathbf{0}$$

$$\mid\ b^{\ell}[1](\beta_1).\beta_1!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0} \mid b^{\ell}[2](\beta_2).\beta_2?(1,y^{\perp}).\mathbf{0}$$

Execution blocks at session initiation if $\top \not\leq \ell$, otherwise it blocks before the exchange of the low value.

Safety

Let $|M|$ be the process obtained by erasing all monitoring levels in M.

Monitored process safety:

M is safe if for any monotone H such that $< |M| \, , \, H >$ is saturated:

If $< |M| \, , \, H > \longrightarrow (\nu\tilde{r}) < P \, , \, H' >$

then $< M \, , \, H > \multimap\!\!\!\rightarrow (\nu\tilde{r}) < M' \, , \, H' >$, where $|M'| = P$ and M' is safe.

Process safety: A process P is safe if $P^{\rceil\perp}$ is safe.

Main results

Safety implies absence of run-time errors

If P is safe, then every monitored computation:

$$< P^{\rceil\perp} , \emptyset > = < M_0 , H_0 > \multimap\!\!\!\rightarrow \cdots \multimap\!\!\!\rightarrow (\nu \tilde{r_k}) < M_k , H_k >$$

is such that $\neg < M_k , H_k > \dagger$.

Safety implies security

If P is safe, then P is $\mathscr{L}$-secure for any down-closed set of levels $\mathscr{L}$.

Main results (ctd)

Absence of run-time errors does not imply safety

Not safe

$$
\begin{array}{lcl}
P & = & \bar{a}^{\ell}[1] \mid a^{\ell}[1](\alpha_1).P_1 \mid a^{\ell}[2](\alpha_2).P_2 \\
P_1 & = & \alpha_1!\langle 2, \mathsf{true}^{\top}\rangle.\alpha_1?(2, x^{\top}).\mathbf{0} \\
P_2 & = & \alpha_2?(1, z^{\top}).\mathsf{if}\ z^{\top}\ \mathsf{then}\ \alpha_2!\langle 1, \mathsf{false}^{\top}\rangle.\mathbf{0}\ \mathsf{else}\ \alpha_2!\langle 1, \mathsf{true}^{\perp}\rangle.\mathbf{0}
\end{array}
$$

Security does not imply safety

Not safe

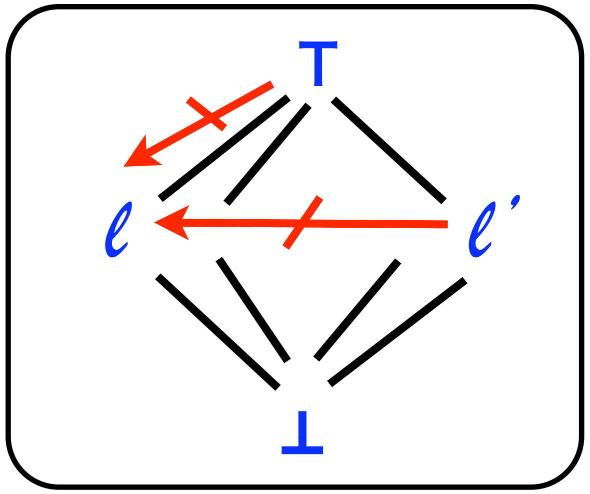
$$[\ s[1]?(2, x^{\top}).\ \mathsf{if}\ x^{\top}\ \mathsf{then}\ s[1]!\langle 3, \mathsf{true}^{\perp}\rangle\ \mathsf{else}\ s[1]!\langle 3, \mathsf{true}^{\perp}\rangle\ ]^{\infty}$$

$$\mid\ [\ s[2]!\langle 1, v^{\top}\rangle\ ]^{\infty}$$

Part 4

Conclusion and future directions

Summary of results



2 main kinds of information leaks:

1) receive $x^{\top}$; send $v^{\perp}$

2) if $e^{\top}$ then send $v_1^{\perp}$ else send $v_2^{\perp}$

3 increasingly precise ways to track information leaks

Type system (prevention): rejects any syntactic leak in the program

$\Downarrow \quad \not\Uparrow$

Safety (local detection): blocks computation when reaching a leak

$\Downarrow \quad \not\Uparrow$

Security (global detection): rejects globally detectable leaks only

Summary of results (ctd)

Interplay between session types and security types, and between lock freedom and leak freedom (*)

Session types help preventing indirect leaks and termination leaks

Input rule => security requirement in conditional rule may be lifted

Lock freedom => security requirement in input rule could be lifted (keeping the usual requirement in conditional rule)

(*) Already noted by Kobayashi [Kob05] for pi-calculus + usage types

Future directions

-> Towards secure data manipulation in web services

-> Towards flexible, adaptable, communication protocols

- Monitored semantics with labelled transitions, returning informative error messages to the programmer

- Security session calculi with reconfiguration/adaptation mechanisms, in reaction to security violations

- Security session calculi with reputations for principals, based on their security-abiding behaviour

References

This lecture

[CCD14a]
Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Tamara Rezk. Session types for access and information flow control, CONCUR'10, LNCS 6269, 2010. Full version to appear in Inf. and Comp.

[CCD14b]
Sara Capecchi, Ilaria Castellani and Mariangiola Dezani-Ciancaglini. Information Flow Safety in Multiparty Sessions, EXPRESS'11, EPTCS, 16-30, vol. 64, 2011. Full version to appear in MSCS.

Papers available on Lovran school web site

References

Related work: SIF in imperative languages

[VSI96] D. Volpano, G. Smith and C. Irvine. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security* 4(3):167–187, 1996.

[VS98] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. *Proceedings of POPL '98*, ACM Press, pages 355–364, 1998.

[SS00] A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200-214, 2000.

[BC01] G. Boudol and I. Castellani. Noninterference for Concurrent Programs. In *Proceedings of ICALP'01*, volume 2076 of *LNCS*, pages 382-395, Springer-Verlag, 2001.

[Smi01] G. Smith. A new type system for secure information flow. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 115–125, 2001.

[BC02] G. Boudol and I. Castellani. Noninterference for Concurrent Programs and Thread Systems. *Theoretical Computer Science* 281(1): 109-130, 2002.

[SM03] A. Sabelfeld and A. C. Myers, Language-based information-flow security. *IEEE Journal on Selected Areas in Communications* 211:5-19, 2003.

References

Related work: SIF on CCS

[FG01] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In *Foundations of Security Analysis and Design - Tutorial Lectures* (R. Focardi and R. Gorrieri, Eds.), volume 2171 of *LNCS*, Springer, 2001.

[FR02] R. Focardi and S. Rossi. Information flow security in dynamic contexts. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 2002.

[BFPR04] A. Bossi, R. Focardi, C. Piazza and S. Rossi. Verifying persistent security properties. *Computer Languages, Systems and Structures* 30(3-4): 231-258, 2004.

[FRS05] R. Focardi, S. Rossi and A. Sabelfeld. Bridging Language-Based and Process Calculi Security. In *Proceedings of FoSSaCs'05*, volume 3441 of *LNCS*, Springer-Verlag, 2005.

[Cas07] I. Castellani. State-oriented Noninterference for CCS. *Electr. Notes Theor. Comput. Sci.* 194(1): 39-60, 2007.

References

Related work: SIF on pi-calculus

[HVY00] K. Honda, V. Vasconcelos and N. Yoshida. Secure information flow as typed process behavior. In *Proceedings of ESOP'00*, volume 1782 of *LNCS*, pages 180-199, Springer-Verlag, 2000.

[HY02] K. Honda and N. Yoshida. A uniform type structure for secure information flow. To appear in *ACM TOPLAS*. Extended abstract in *Proceedings of POPL'02*, pages 81-92, January, 2002.

[Pot02] F. Pottier. A Simple View of Type-Secure Information Flow in the π-Calculus. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 320–330, 2002.

[HR02] M. Hennessy and J. Riely. Information flow vs resource access in the asynchronous pi-calculus. *ACM TOPLAS* 24(5): 566-591, 2002.

[Hen04] M. Hennessy. The security π-calculus and noninterference. *Journal of Logic and Algebraic Programming* 63(1): 3-34, 2004.

[CR05] S. Crafa and S. Rossi. A theory of noninterference for the π-calculus. In *Proceedings of Symp. on Trustworthy Global Computing TGC'05*, volume 3705 of *LNCS*, Springer-Verlag, 2005.

[Kob05] N. Kobayashi. Type-based Information Flow Analysis for the Pi-Calculus. *Acta Informatica* 42(4-5): 291-347, 2005.

Thank you!