

# LANGUAGE-BASED SECURITY – SUMMARY

Static analysis of **information flow** in programs:

- ▶ a **programming language**: syntax and semantics.
- ▶ a confidentiality policy: a **lattice of security levels**.
- ▶ a **security policy**: no illegal flow of information.
- ▶ a **type system** for information flow.
- ▶ a type **soundness** result.

# CONTENTS

---

- ▶ A simple imperative language.
- ▶ Concurrent threads.
- ▶ A [higher-order](#) imperative language.
- ▶ [Declassification](#).

A higher-order imperative language

# FUNCTIONAL PROGRAMS

---

3  
(1/2)

- ▶ ML, SCHEME: programs are expressions with a result possibly  $\neq ()$ , **expressions are programs** to execute, possibly with side-effects and non-termination.
- ▶ Compared with imperative + concurrent programs:
  - ▶▶ we **still have**: boolean values, conditional branching, termination  $()$ , assignments, sequential composition, while loops (derived), thread creation.
  - ▶▶ we get: **functions** ( $\lambda$ -abstractions) – esp. procedures –, **application** of functions and procedures to arguments (= expressions), dynamically created **memory locations** (with scope), **recursion**.

# FUNCTIONAL PROGRAMS

4  
(2/2)

## Syntax – ML-like

$$\begin{aligned} V, W \dots & ::= x \mid u_{\ell, \theta} \mid \lambda x M \mid tt \mid ff \mid () \quad \text{values} \\ M, N \dots & ::= V \mid (\text{if } M \text{ then } N \text{ else } N') \mid (MN) \\ & \mid M ; N \mid (\text{ref}_{\ell, \theta} N) \mid (! N) \mid (M := N) \\ & \mid \rho x V \mid (\text{thread } M) \end{aligned}$$

## Comments:

- ▶ values are **terminated** expressions.
- ▶  $u_{\ell, \theta}$  is any memory location, or **reference**, with security level  $\ell$  and type  $\theta$ .
- ▶  $\lambda x M$  may also be written  $(\text{fun } x \rightarrow M)$ .
- ▶  $(\text{ref}_{\ell, \theta} N)$  creates a **fresh** reference with initial value  $N$ .
- ▶  $(! N)$  is **dereferencing**, to get the value of reference  $N$  (previously “ $x$ ”).
- ▶  $\rho x V$  is  $V$  where  $x$  is **recursively** bound to  $V$ .

## SOME NOTATIONS

---

- ▶  $(\lambda x M N)$  also written **(let  $x = N$  in  $M$ )**.
- ▶  $M ; N$  could be derived as **(let  $z = N$  in  $M$ )** where  $z$  does not occur in  $M$ .
- ▶ **(while  $M$  do  $N$ )** is now derived as

$$(\rho w \lambda x (\text{if } M \text{ then } N ; (w x) \text{ else } x) ())$$

- ▶  $\rho x x$  is written **loop**.

As for the imperative + concurrent language:

$$(M, \mu) \xrightarrow{N} (M', \mu')$$

Evaluation contexts:

$$\begin{aligned} \mathbf{E} ::= & [\cdot] \mid (\text{if } \mathbf{E} \text{ then } M \text{ else } N) \mid (\mathbf{E}N) \mid (V\mathbf{E}) \\ & \mid \mathbf{E}; N \mid (\text{ref}_{\ell, \theta} \mathbf{E}) \mid (!\mathbf{E}) \mid (\mathbf{E} := N) \mid (V := \mathbf{E}) \end{aligned}$$

with

$$\frac{(M, \mu) \xrightarrow{N} (M', \mu')}{(\mathbf{E}[M], \mu) \xrightarrow{N} (\mathbf{E}[M'], \mu')}$$

plus the rules for parallel composition ( $T \parallel T'$ ).

# OPERATIONAL SEMANTICS

7  
(2/2)

$$\frac{}{((\text{if } tt \text{ then } M \text{ else } N), \mu) \xrightarrow{\emptyset} (M, \mu)}$$

$$\frac{}{((\text{if } ff \text{ then } M \text{ else } N), \mu) \xrightarrow{\emptyset} (N, \mu)}$$

$$\frac{}{((\lambda x M V), \mu) \xrightarrow{\emptyset} (\{x \mapsto V\} M, \mu)}$$

$$\frac{}{(V ; N, \mu) \xrightarrow{\emptyset} (N, \mu)}$$

$$\frac{}{((\text{ref}_{\ell, \theta} V), \mu) \xrightarrow{\emptyset} (u_{\ell, \theta}, \mu \cup \{u_{\ell, \theta} \mapsto V\})} \quad u \text{ fresh}$$

$$\frac{}{((! u_{\ell, \theta}), \mu) \xrightarrow{\emptyset} (\mu(u_{\ell, \theta}), \mu)}$$

$$\frac{}{((u_{\ell, \theta} := V), \mu) \xrightarrow{\emptyset} (\emptyset, \mu[u_{\ell, \theta} := V])}$$

$$\frac{}{(\varrho x V, \mu) \xrightarrow{\emptyset} (\{x \mapsto \varrho x V\} V, \mu)}$$

$$\frac{}{((\text{thread } N), \mu) \xrightarrow{N} (\emptyset, \mu)}$$

Still **direct**, **indirect** and **termination leaks** – but: more evaluation contexts  $\Rightarrow$  more opportunities.

- ▶ **Application** ( $MN$ )
- ▶ **non termination** of  $M$  may prevent low writing in  $N$ , e.g. with  $M = (\text{if } !u_H \text{ then } \lambda x x \text{ else loop})$  and  $N = (v_L := tt)$ .
- ▶ if applying  $M$  has **low writing effect**, e.g.  $M = \lambda x (v_L := x)$ , there may be a direct flow, e.g. with  $N = (!u_H)$ .
- ▶ if  $M$  has low writing effect, **reading it from a secret location** may result in an indirect flow, e.g. with  $M = (!u_H)$  and  $\mu(u_H) = \lambda x (v_L := V)$  and  $\nu(u_H) = \lambda x (v_L := W)$ .
- ➡ We record the **effect** of (applying) a function in its type.

# INFORMATION FLOW REVISITED

---

9  
(2/2)

- ▶ **Assignment** ( $M := N$ )
- ▶ **non termination** of  $M$  may prevent low writing in  $N$ , as in  $(MN)$ .
- ▶ if  $M$  is **read from a secret location**, e.g.  $M = (!u_H)$ , there may be an indirect flow, if the value read is a low reference, e.g. with  $\mu(u_H) = v_L$  and  $\nu(u_H) = w_L$ .
- ▶ **Creation** of a **low** reference ( $\text{ref}_L N$ ): what if  $N$  reads secret information, e.g.  $N = (!u_H)$ ?

# TYPING INFORMATION FLOW – Again! <sup>10</sup> (1/5)

Programs = expressions  $\Rightarrow$  we record

- ▶  $r$ , the confidentiality level = an upper bound of the reading level,
- ▶  $w$ , a lower bound of the writing level,
- ▶  $t$ , an upper bound on the levels of reading that could influence termination.

↳ typing judgements:

$$\Gamma \vdash M : (r, w, t), \tau$$

where

$$\tau, \theta, \sigma \dots ::= \text{bool} \mid \text{unit} \mid \theta \text{ref}_\ell \mid (\sigma \xrightarrow{(r,w,t)} \tau) \mid \dots$$

In the typing context  $x : (\ell, \theta) \rightsquigarrow x : \theta \text{ref}_\ell$ .

# TYPING INFORMATION FLOW

11

(2/5)

The “security effects”  $s = (r, w, t)$  are ordered componentwise:

$$\begin{aligned}(r, w, t) \leq (r', w', t') &\Leftrightarrow_{\text{def}} r \leq r' \ \& \ w \geq w' \ \& \ t \leq t' \\(r, w, t) \sqcup (r', w', t') &= (r \sqcup r', w \sqcap w', t \sqcup t') \\ \perp = (\perp, \top, \perp) &\ \& \ \top = (\top, \perp, \top)\end{aligned}$$

Typing rules:

$$\begin{array}{c} \overline{\Gamma, x : \tau \vdash x : \perp, \tau} \\ \Gamma \vdash \lambda x M : \perp, (\sigma \xrightarrow{s} \tau) \end{array} \qquad \begin{array}{c} \overline{\Gamma \vdash u_{\ell, \theta} : \perp, \theta \text{ ref}_{\ell}} \\ \Gamma \vdash () : \perp, \text{unit} \end{array}$$

More generally: **unit** ( $\sim$  “void”) is the type of “commands”, and the return type of procedures.

# TYPING INFORMATION FLOW

12

(3/5)

$$\overline{\Gamma \vdash tt : \perp, \text{bool}}$$

$$\overline{\Gamma \vdash ff : \perp, \text{bool}}$$

Termination of conditional branching may depend on the reading level of the predicate:

(if !  $u_H$  then () else loop)

$\Rightarrow$  the rule is

$$\frac{\Gamma \vdash M : (r, w, t), \text{bool} \quad \Gamma \vdash N_i : (r_i, w_i, t_i), \tau}{\Gamma \vdash (\text{if } M \text{ then } N_0 \text{ else } N_1) : s \sqcup s_0 \sqcup s_1 \sqcup (\perp, \top, r), \tau} \quad r \leq w_0 \sqcap w_1$$

where  $s = (r, w, t)$  and  $s_i = (r_i, w_i, t_i)$ .

# TYPING INFORMATION FLOW

(4/5)

**Termination** of the application  $(MN)$  may depend on the reading level of both the function and the argument:  $M = (!u_H)$  and  $N = ()$ , or  $M = \lambda x(x())$  and  $N = (!u_H)$  with  $\mu(u_H) = \lambda x()$  and  $\nu(u_H) = \lambda x \text{loop}$ .

$$\frac{\Gamma \vdash M : s_0, (\sigma \xrightarrow{s_1} \tau) \quad \Gamma \vdash N : s_2, \sigma}{\Gamma \vdash (MN) : s_0 \sqcup s_1 \sqcup s_2 \sqcup (\perp, \top, r_0 \sqcup r_2), \tau} \quad t_0 \leq w_2, r_0 \sqcup r_2 \leq w_1$$

$$\frac{\Gamma \vdash M : s, \sigma \quad \Gamma \vdash N : s', \tau}{\Gamma \vdash M ; N : s \sqcup s', \tau} \quad t \leq w' \qquad \frac{\Gamma \vdash N : s, \theta}{\Gamma \vdash (\text{ref}_{\ell, \theta} N) : s, \theta \text{ref}_{\ell}} \quad r \leq \ell$$

# TYPING INFORMATION FLOW

14

(5/5)

Dereferencing builds the [reading level](#):

$$\frac{\Gamma \vdash N : s, \theta \text{ ref}_\ell}{\Gamma \vdash (!N) : s \sqcup (\ell, \top, \perp), \theta}$$

Assignment builds the [writing level](#):

$$\frac{\Gamma \vdash M : s, \theta \text{ ref}_\ell \quad \Gamma \vdash N : s', \theta}{\Gamma \vdash (M := N) : s \sqcup s' \sqcup (\perp, \ell, \perp), \text{unit}} \quad t \leq w', r \sqcup r' \leq \ell$$

$$\frac{\Gamma, x : \tau \vdash V : s, \tau}{\Gamma \vdash \rho x V : s, \tau}$$

$$\frac{\Gamma \vdash M : s, \tau}{\Gamma \vdash (\text{thread } M) : (\perp, w, \perp), \text{unit}}$$

# EXAMPLE

(1/2)

Derived typing for

$$(\text{while } M \text{ do } N) =_{\text{def}} (\varrho y \lambda x (\text{if } M \text{ then } N ; (yx) \text{ else } x)())$$

Assume  $\Gamma \vdash M : s_0, \text{bool}$  and  $\Gamma \vdash N : s_1, \text{unit}$ , and let  $s = s_0 \sqcup s_1 \sqcup (\perp, \top, r_0)$  and  $\Delta = \Gamma, y : (\text{unit} \xrightarrow{s} \text{unit}), x : \text{unit}$ . Then

$$\frac{\frac{\vdots}{\Delta \vdash N : s_1, \text{unit}} \quad \frac{\frac{\Delta \vdash y : \perp, (\text{unit} \xrightarrow{s} \text{unit}) \quad \Delta \vdash x : \perp, \text{unit}}{\Delta \vdash (yx) : s, \text{unit}}}{\Delta \vdash N ; (yx) : s, \text{unit}}}{\Delta \vdash N ; (yx) : s, \text{unit}} \quad t_1 \leq w_0 \sqcap w_1$$

Therefore

$$\frac{\frac{\vdots}{\Delta \vdash M : s_0, \text{bool}} \quad \frac{\vdots}{\Delta \vdash N ; (yx) : s, \text{unit}} \quad \frac{\vdots}{\Delta \vdash x : \perp, \text{unit}}}{\Delta \vdash (\text{if } M \text{ then } N ; (yx) \text{ else } x) : s, \text{unit}} \quad \left\{ \begin{array}{l} t_1 \leq w_0 \sqcap w_1 \\ r_0 \leq w_0 \sqcap w_1 \end{array} \right.$$

# EXAMPLE

(2/2)

Finally, if we let  $P = (\text{if } M \text{ then } N ; (yx) \text{ else } x)$ , we get

$$\frac{\frac{\frac{\vdots}{\Delta \vdash P : s, \text{unit}}}{\Gamma, y : \text{unit} \xrightarrow{s} \text{unit} \vdash \lambda x P : \perp, \text{unit} \xrightarrow{s} \text{unit}}}{\Gamma \vdash \rho y \lambda x P : \perp, \text{unit} \xrightarrow{s} \text{unit}} \quad \frac{}{\Gamma \vdash () : \perp, \text{unit}}}{\Gamma \vdash (\text{while } M \text{ do } N) : s, \text{unit}} \quad \left\{ \begin{array}{l} t_1 \leq w \\ r_0 \leq w_0 \sqcap w_1 \end{array} \right.$$

that is

$$\frac{\Gamma \vdash M : s_0, \text{bool} \quad \Gamma \vdash N : s_1, \text{unit}}{\Gamma \vdash (\text{while } M \text{ do } N) : s_0 \sqcup s_1 \sqcup (\perp, \top, r_0)} \quad r_0 \sqcup t_1 \leq w_0 \sqcap w_1$$

## SECURITY REVISITED – Again!

---

To take into account the [creation of references](#):

$$\mu =^{\leq \ell} \nu \Leftrightarrow_{\text{def}} \forall u_{\ell', \theta} \in \text{dom}(\mu) \cap \text{dom}(\nu). \ell' \leq \ell \Rightarrow \mu(u_{\ell', \theta}) = \nu(u_{\ell', \theta})$$

Then:

- ▶  $M$  is [strongly secure](#) iff for any  $\ell$  there exists an  $\ell$ -bisimulation  $\mathcal{R}$  such that  $M \mathcal{R} M$ .
- ▶  $\mathcal{R}$  is an  [\$\ell\$ -bisimulation](#) iff  $M \mathcal{R} N$  implies either
  - ▶▶  $M$  and  $N$  are both  $\ell$ -high, i.e.  $M, N \in \text{High}_\ell$ , or
  - ▶▶ if  $(M, \mu) \xrightarrow{S} (M', \mu')$  &  $\mu =^{\leq \ell} \nu$  with  $\text{dom}(\mu' - \mu) \cap \text{dom}(\nu) = \emptyset$  then  $S \mathcal{R} S$  and there exist  $N'$  and  $\nu'$  with  $\text{dom}(\nu' - \nu) = \text{dom}(\mu' - \mu)$  such that  $(N, \nu) \xrightarrow{S} (N', \nu')$  with  $N' \mathcal{R} N'$  and  $\mu' =^{\leq \ell} \nu'$ .

# TYPE SOUNDNESS

---

Theorem: a typable program is strongly secure.

Proof: see Almeida Matos & Boudol 05.

A more advanced topic:  
declassification  
(current research)

## A SERIOUS PROBLEM: DECLASSIFICATION

---

Non-interference is **impractical**: there are (common and useful) programs that **intentionally leak** information, e.g. a **password checker**

$$\begin{aligned}
 P(\textit{user\_id}, \textit{pwd}) &= \text{if } \text{!(}\textit{pwd\_db.user\_id})_{\textit{Secret}} = \textit{pwd} \\
 &\text{then } \textit{output}_{\textit{Public}} := \text{"yes"} \\
 &\text{else } \textit{output}_{\textit{Public}} := \text{"no"}
 \end{aligned}$$

yet  $\textit{Secret} \not\subseteq \textit{Public}$ , otherwise

$$P'(\textit{user\_id}, \textit{pwd}) = \textit{output}_{\textit{Public}} := \text{"!(}\textit{pwd\_db.root})_{\textit{Secret}}\text{"}$$

would be legal! Another standard [counter-]example is **encryption** (Jones & Lipton 75).

➡ **Question**: is it possible to have “language-based information-flow security” while allowing declassification? **Requires** a new security policy.

# ISSUES

---

(1) **What**, or **how much** information is it secure to disclose?

For instance, revealing  $1/2^n$  bit, or  $x_{Public} := e_k(! y_{Secret})$  where  $e_k$  is “one-way” (encryption), are ok (depending on the computing power of the “attacker”, and the “value” of the information).

(2) **How** to accommodate intentionally declassifying programs in a language-based security approach?

For instance,  $P$  and  $P'$  are different w.r.t. question (1), but the same w.r.t. (2): both disclose information from level *Secret* to level *Public* – but not the same amount!

# Our APPROACH to DECLASSIFICATION

---

## What?

The [programmer](#) has responsibility for solving this question (a proof-of-program activity).

## How?

The (designer of the) [programming language](#) has to provide an answer to this one.

↳ we address question “**how?**” – language design.

## Our SOLUTION

---

Keep the good stuff of language-based security: information flow analysis by means of [security lattices](#), [non-interference](#), [sound type systems](#)...

... but: make it [local](#) to subprograms,

introducing a [local flow declaration](#) construct ([flow  \$F\$  in  \$P\$](#) ).

For instance, the password checker assumes the  $Secret \leq Public$  flow relation holds, but this should only hold locally, not for all programs:

$$P(user\_id, pwd) = (\text{flow } Secret \prec Public \text{ in } \dots)$$

Another example – a service selling digital information:

$$(\text{if } paid \text{ then } (\text{flow } A \prec B \text{ in } y_B := !x_A) \text{ else } \dots)$$

# SECURITY LATTICES REVISITED

---

- ▶ A set  $\mathcal{P}$  of **principals** is given (e.g. *Alice, Bob...*).
- ▶ **Security levels** are sets of principals  $\ell \subseteq \mathcal{P}$ , assigned to memory locations: who has access.
- ▶  $\ell \supseteq \ell'$  means  $\ell'$  is more **restrictive** than  $\ell$ . Information may flow from  $\ell$  to  $\ell'$ .
- ▶ **Flow relations**  $F \subseteq \mathcal{P} \times \mathcal{P}$  determine **preorders**

$$\ell \preceq_F \ell' \iff_{\text{def}} \forall q \in \ell' \exists p \in \ell. pF^*q$$

with **meet**  $\ell \sqcup \ell'$  and **join**

$$\ell \sqcup \ell' = \{ q \mid \exists p \in \ell \exists p' \in \ell'. pF^*q \ \& \ p'F^*q \}$$

Up to now:  $\leq = \preceq_G$  where  $G$  is a given **global confidentiality policy**.

# The LANGUAGE

---

## Syntax

$$M, N \dots ::= \dots \mid (\text{flow } F \text{ in } M)$$

Operational semantics: small-step transitions

$$(M, \mu) \xrightarrow[F]{N} (M', \mu')$$

where  $F$  is the **local** flow policy that holds for this step, and extends a given global flow policy, with

$$\frac{(M, \mu) \xrightarrow[G]{N} (M', \mu')}{((\text{flow } F \text{ in } M), \mu) \xrightarrow[F \cup G]{N} ((\text{flow } F \text{ in } M'), \mu')}$$

$$\frac{}{((\text{flow } F \text{ in } V), \mu) \xrightarrow[\emptyset]{\emptyset} (V, \mu)}$$

# The NON-DISCLOSURE POLICY

(1/2)

- ▶ **Intuition:** if  $(M, \mu) \xrightarrow[F]{N} (M', \mu')$  then  $M$  is allowed to read the input memory according to the global flow policy  $G$  **extended with  $F$** .
- ▶ Refined memory equality, **relative to a flow relation:**

$$\mu = \overset{F}{\preceq} \nu \quad \Leftrightarrow_{\text{def}} \quad \forall u_{\ell', \theta} \in \text{dom}(\mu) \cap \text{dom}(\nu) \\ \ell' \preceq_F \ell \Rightarrow \mu(u_{\ell', \theta}) = \nu(u_{\ell', \theta})$$

Notice that

$$F \subseteq F' \ \& \ \mu = \overset{F'}{\preceq} \nu \Rightarrow \mu = \overset{F}{\preceq} \nu$$

- ▶  $M$  satisfies the **non-disclosure policy** w.r.t.  $G$  iff for any  $\ell$  there exists a  $(G, \ell)$ -bisimulation  $\mathcal{R}$  such that  $M \mathcal{R} M$ .

# The NON-DISCLOSURE POLICY

(2/2)

$\mathcal{R}$  is a  $(G, \ell)$ -bisimulation iff  $M \mathcal{R} N$  implies either

- ▶  $M$  and  $N$  are both  $(G, \ell)$ -high, i.e.  $M, N \in \mathcal{H}igh_{G, \ell}$ , or
- ▶ if  $(M, \mu) \xrightarrow[F]{S} (M', \mu')$  &  $\mu = \preceq_{F \cup G^\ell} \nu$  with  $\text{dom}(\mu' - \mu) \cap \text{dom}(\nu) = \emptyset$  then  $S \mathcal{R} S$  and there exist  $N'$  and  $\nu'$  with  $\text{dom}(\nu' - \nu) = \text{dom}(\mu' - \mu)$  such that  $(N, \nu) \xrightarrow[F]{S} (N', \nu')$  with  $N' \mathcal{R} N'$  and  $\mu' = \preceq_{G^\ell} \nu'$ .

Remark:  $M$  **strongly secure** w.r.t.  $\leq = \preceq_G \Rightarrow M$  satisfies the non-disclosure policy.

Conversely:  $M$  satisfies the non-disclosure policy and **does not use declassification** (i.e. only perform transitions  $\xrightarrow[\emptyset]{N}$ )  $\Rightarrow M$  strongly secure.

# TYPING DECLASSIFICATION

---

## Judgements

$$F; \Gamma \vdash M : s, \tau$$

where  $F$  is the current flow relation that is in force when evaluating  $M$ , used to **check information flow**.

**Typing rules:** replace  $\leq$  by  $\preceq_F$ , and similarly for the meet and join, plus

$$\frac{F, F'; \Gamma \vdash M : s, \tau \quad r \preceq_{F' \cup F} r' \quad t \preceq_{F' \cup F} t' \preceq_{F'} r'}{F'; \Gamma \vdash (\text{flow } F \text{ in } M) : (r', w, t'), \tau}$$

**Type soundness:** Any expression that is typable in the context of the flow relation  $G$ , that is  $G; \Gamma \vdash M : s, \tau$ , satisfies the non-disclosure policy w.r.t.  $G$ .

Proof: see Almeida Matos & Boudol 05.

## SOME FURTHER TOPICS

---

- ▶ [exceptions](#): Vincent Simonet's PhD Thesis.
- ▶ [continuations](#): Steve Zdancewic's PhD Thesis.
- ▶ [objects](#): Myers, Barthe and Serpette, Banerjee and Nauman.
- ▶ [low-level](#) languages: Barthe and Rezk.
- ▶ [process calculi](#): Focardi and Gorrieri, Honda.
- ▶ [probabilistic](#) non-interference: Sabelfeld and Sands.
- ▶ [quantified](#) information flow.
- ▶ [distributed](#) programming – relation with authentication, cryptographic protocols?
- ▶ [mobile](#) code.
- ▶ [etc.](#)

## SOME READINGS

---

- ▶ A [book](#): “Computer Security, Art and Science”, Matt Bishop, Addison-Wesley (2003).
- ▶ A [PhD Thesis](#): “Programming Languages for Information Security”, Steve Zdancewic (2002).
- ▶ A [survey](#): “Language-Based Information-Flow Security”, Andrei Sabelfeld and Andrew Myers (IEEE J. on Selected Areas in Communications 21-1, 2003). Available from Andrei’s web page.
- ▶ My own work:  
<http://www-sop.inria.fr/mimosa/>