Proving the Correctness of Type-Based Information Flow Analyses: Some Approaches

François Pottier, INRIA, April 2002

Outline

1. Type-based information flow analysis
2. A modular proof technique, with instances based on
   - a labelled λ-calculus;
   - a labelled π-calculus;
   - a bracketed λ-calculus.
3. Conclusion

The "confinement problem" (Lampson, 1973)

- Information systems (computers) run *multiple* processes,
- on behalf of *multiple* users,
- which read and write *multiple*, *shared* bodies of data.

It is often desirable to control the *flow* of information through these systems, so as to preserve data *secrecy* or *integrity*.

Access control vs. information flow control

- *Access control*, a widespread, authentication-based mechanism, only restricts the *initial* release of data. Thus, it requires *trust*, which is often misplaced, especially in *programs*.
- *Information flow control*, on the other hand, aims to restrict *every single* flow of information, *obviating* the need for trust, *provided* the whole system is within control.

Language-based information flow control

- The system is (viewed as) a *program*,
- whose *semantics* it is easy to reason about,
- making a *static* analysis possible, *provided* its code is available beforehand.

Defining information flow

Consider a sequential program P of one input and one output. P allows (some) information to flow from its input to its output if varying the former causes the latter to vary, that is, if the latter *depends* on the former:

$$\exists xy \quad P(x) \neq P(y)$$

The negation is called *non-interference* (Goguen and Meseguer, 1982):

$$\forall xy \quad P(x) = P(y)$$

More generally, if P is a process with one input and $\approx$ is a notion of process equivalence, then

$$\forall xy \quad P(x) \approx P(y)$$

states that there is no flow of information from P's input to "the observer".

Setting up an information flow policy

Bell & LaPadula (1973) and Denning (1975) suggest adopting a security lattice $(\mathcal{L}, \leq)$, and assigning a level $\underline{x}$ to every variable x (or, more generally, to every *piece* of input or output). Then, a flow from x to y is legal iff $\underline{x} \leq \underline{y}$.

Sample lattices:

- $\{\mathrm{L}, \mathrm{H}\}$ allows distinguishing public vs. secret (or trusted vs. untrusted) data.
- The *powerset* of a set of principals allows telling who may consult (or who produced) a piece of data.
- Taking the *product* of several such lattices allows forming composite policies, such as the military classification lattice.

Specifying a program

Given a fixed lattice, say $\{\text{L} \leq \text{H}\}$, one may ascribe *types* to programs:

$$P : \mathsf{int}^{\text{L}} \times \mathsf{int}^{\text{H}} \rightarrow \mathsf{int}^{\text{L}} \times \mathsf{int}^{\text{H}}$$

Such a specification, which *relies upon* and *enriches* the original type structure, *encodes* a non-interference assertion:

$$\forall k, k_1, k_2 \quad \text{fst}\ (P\ (k, k_1)) = \text{fst}\ (P\ (k, k_2))$$

Denning (1975) points out that the same assertion can be encoded in a *polymorphic*, *lattice-independent* manner:

$$P : \forall \ell, h[\ell \leq h].\mathsf{int}^{\ell} \times \mathsf{int}^{h} \rightarrow \mathsf{int}^{\ell} \times \mathsf{int}^{h}$$

Thus, information flow analysis is about *detecting dependencies*.

A modular proof technique

A type-based information flow analysis can be proved correct as follows:

- define an instrumented semantics, that is, a *dynamic* dependency analysis;
- prove it correct;
- define a type system for it, that is, a *static* approximation of it;
- prove it correct (*subject reduction*);
- derive a non-interference statement from the above.

The first part is about making dependencies explicit, and need not be concerned with types. The second part is a standard type preservation argument (albeit for a non-standard semantics).

A labelled λ-calculus

A dynamic dependency analysis proposed by Abadi, Lampson & Lévy (1996).

$$e ::= k \mid x \mid \lambda x.e \mid (e\ e) \mid \ell : e \qquad (\ell \in \mathcal{L})$$

Operational semantics:

$$\begin{array}{rcll} (\lambda x.e_1)\, e_2 & \to & e_1[e_2/x] & (\beta) \\ (\ell : e_1)\ e_2 & \to & \ell : (e_1\ e_2) & (\mathit{lift}) \end{array}$$

Informal correctness statement: if $e \to^\star f$, then f does not *depend* on any sub-term of e which carries a label not found in f. That is, the *prefix* of e obtained by *pruning* all such sub-terms still produces f.

Correctness

Prefixes are defined by augmenting expressions with a hole $\_$. Write $e \preceq e'$ if e is obtained from e' by replacing some sub-terms with holes.

Monotonicity. Let e, e' be prefixes such that $e \preceq e'$. If f is an expression such that $e \rightarrow^\star f$, then $e' \rightarrow^\star f$.

Let $\lfloor e \rfloor$ be the prefix of e where every sub-term labelled $\textsc{h}$ has been pruned. (Assuming $\mathcal{L} = \{\textsc{l} \leq \textsc{h}\}$, for simplicity.)

Stability. Assume e is a prefix and f is an expression. If $e \rightarrow^\star f$ and $\lfloor f \rfloor = f$, then $\lfloor e \rfloor \rightarrow^\star f$.

A (simple) type system for the labelled λ-calculus

Types are given by $t ::= \mathsf{int}^{\ell} \mid (t \to t)^{\ell}$, and equipped with a *subtyping* relation which extends the order on security levels. The predicate $\ell \lhd t$ is defined by $\ell \lhd \mathsf{int}^{\ell'} \iff \ell \lhd (t \to t')^{\ell'} \iff \ell \leq \ell'$.

Here are a few typing rules of interest:

$$\textsc{Sub}\ \frac{\Gamma \vdash e : t \qquad t \leq t'}{\Gamma \vdash e : t'} \qquad \textsc{Label}\ \frac{\Gamma \vdash e : t \qquad \ell \lhd t}{\Gamma \vdash (l : e) : t} \qquad \textsc{App}\ \frac{\Gamma \vdash e_1 : (t \to t')^{\ell} \quad \Gamma \vdash e_2 : t \qquad \ell \lhd t'}{\Gamma \vdash e_1\, e_2 : t'}$$

Subject reduction. $\vdash e : t$ and $e \to e'$ imply $\vdash e' : t$.

Non-interference statements

Non-interference. If $\vdash e : \mathsf{int}^{\mathrm{L}}$ and $e \rightarrow^{\star} v$, then $\lfloor e \rfloor \rightarrow^{\star} v$.

Proof. By subject reduction, $\vdash v : \mathsf{int}^{\mathrm{L}}$. By examination of the typing rules, $\lfloor v \rfloor = v$. By stability, $\lfloor e \rfloor \rightarrow^{\star} v$.

Non-interference. If $\vdash e, f : \mathsf{int}^{\mathrm{L}}$ and $\lfloor e \rfloor = \lfloor f \rfloor$, then either both e and f diverge, or both produce the same value.

Proof. By the above, monotonicity, and progress.

A labelled π-calculus

A π-calculus where (say) messages are labelled:

$$P ::= 0 \mid (P \mid P) \mid \nu x.P \mid !P \mid x(\tilde{y}).P \mid \ell : \bar{x}\langle\tilde{v}\rangle$$

Operational semantics:

$$x(\tilde{y}).P \mid \ell : \bar{x}\langle\tilde{v}\rangle \to \ell \bullet P[\tilde{v}/\tilde{y}]$$

The prefix ordering is generated by $0 \leq P$, erasure by $\lfloor \mathrm{H} : \bar{x}\langle\tilde{v}\rangle \rfloor = 0$.

Monotonicity. If $P \to Q$ and $P \leq P'$, then $P' \to \cdot \geq Q$.

Stability (1). If $P \to Q$, then $\lfloor P \rfloor \to \cdot \geq \lfloor Q \rfloor$.

Stability (*). If $P \to^\star Q$, then $\lfloor P \rfloor \to^\star \cdot \geq \lfloor Q \rfloor$.

A type system for the labelled π-calculus

Leaving out the details, assume that types are given by $t ::= \langle \tilde{t} \rangle^{\ell}$. Assume the type system satisfies the following two properties:

Subject reduction. $P \to Q$ and $\Gamma \vdash P$ imply $\Gamma \vdash Q$.

Barb preservation. If $x : \langle\rangle^{\mathrm{L}} \vdash P$ and $P \downarrow_x$, then $\lfloor P \rfloor \downarrow_x$.

Non-interference statements

Non-interference. If $x : \langle\rangle^{\mathrm{L}} \vdash P$ and $P \Downarrow_x$, then $\lfloor P \rfloor \Downarrow_x$.

Proof. Assume $P \rightarrow^\star P'$ and $P' \downarrow_x$. By subject reduction and barb preservation, $\lfloor P' \rfloor \downarrow_x$. Furthermore, by stability, $\lfloor P \rfloor \rightarrow^\star \cdot \geq \lfloor P' \rfloor$. This implies $\lfloor P \rfloor \Downarrow_x$.

Non-interference. If $x : \langle\rangle^{\mathrm{L}} \vdash P, Q$ and $\lfloor P \rfloor = \lfloor Q \rfloor$ then $P \approx_{may} Q$.

Proof. By the above and monotonicity.

Labels have limits

In a programming language with state, the statement $\textsf{if}\ !x = 0\ \textsf{then}\ z := 1$ causes information to flow from x to z, *even* when it is skipped. As a result, in such a situation, it appears difficult to conceive a labelled semantics.

In the π-calculus, building a *bisimulation*-based non-interference result on top of a labelled semantics appears a bit awkward.

An alternative to labels: brackets

Until now, we have used labels and prefixes to reason about arbitrary data:

$$(\text{L} : (\lambda xy.y))\ (\text{H} : \_) \to^\star \text{L} : (\lambda y.y)$$

This implied $(\lambda xy.y)\ 27 \to^\star \lambda y.y$ and $(\lambda xy.y)\ 68 \to^\star \lambda y.y$ by *monotonicity* and *erasure*. Instead, it is possible to reason directly about *two* processes that share some structure:

$$(\lambda xy.y)\ \langle 27 \mid 68 \rangle \to^\star \lambda y.y$$

with the same consequences, this time via *projection*.

A "bracketed" λ-calculus: λ^2

The language λ^2 is defined as an extension of λ. (Brackets cannot be nested.)

$$e \quad ::= \quad \ldots \mid \langle e \mid e \rangle$$

A λ^2 term encodes a pair of λ terms. For instance, $\langle e_1 \mid e_2 \rangle\, e$ and $\langle e_1\, e \mid e_2\, e \rangle$ both encode the pair $(e_1\, e, e_2\, e)$.

Brackets encode the *differences* between two programs, i.e. their "secret" parts.

Two *projection* functions map a λ^2 term to the two λ terms which it encodes. In particular, $\lfloor \langle e_1 \mid e_2 \rangle \rfloor_i = e_i$, for $i \in \{1, 2\}$.

Semantics of λ^2

As in the labelled λ-calculus, each language construct is typically dealt with by *two* reduction rules: a standard one, and one that moves (*lifts*) brackets out of the way. The latter are no-ops w.r.t. the term's projections.

$$\begin{array}{rclr} (\lambda x.e)\, v & \rightarrow & e[x \Leftarrow v] & (\beta_v) \\ \langle v_1 \mid v_2 \rangle\, v & \rightarrow & \langle v_1\, \lfloor v \rfloor_1 \mid v_2\, \lfloor v \rfloor_2 \rangle & (\mathit{lift}) \end{array}$$

Soundness. Let $i \in \{1, 2\}$. If $e \rightarrow e'$, then $\lfloor e \rfloor_i \rightarrow \lfloor e' \rfloor_i$.

Completeness. Assume $\lfloor e \rfloor_i \rightarrow^\star v_i$ for all $i \in \{1, 2\}$. Then, $e \rightarrow^\star v$ for some value v.

In short, brackets cannot *corrupt* or *block* reduction.

A type system for λ

The bracketed calculus can now be seen as a tool to attack the non-interference proof, provided we can *control* the proliferation of brackets during reduction.

We again do so using a type system which *marks* every bracket as "secret":

$$\frac{\Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t \qquad \textsc{h} \lhd t}{\Gamma \vdash \langle e_1 \mid e_2 \rangle : t}\ \textsc{Bracket}$$

No-low-brackets. If $v : \mathsf{int}^{\textsc{l}}$ then $\lfloor v \rfloor_1 = \lfloor v \rfloor_2$.

Subject reduction. $\vdash e : t$ and $e \rightarrow e'$ imply $\vdash e' : t$.

Non-interference

Non-interference. Assume $x : \mathsf{int}^{\mathrm{H}} \vdash e : \mathsf{int}^{\mathrm{L}}$, where e is a λ expression. If, for all $i \in \{1, 2\}$, $v_i : \mathsf{int}^{\mathrm{H}}$ and $e[x \Leftarrow v_i] \rightarrow^\star v'_i$ hold, then $v'_1 = v'_2$.

Proof. Define $v = \langle v_1 \mid v_2 \rangle$. Then, $v : \mathsf{int}^{\mathrm{H}}$ holds. By substitution, $e[x \Leftarrow v] : \mathsf{int}^{\mathrm{L}}$. Now, $\lfloor e[x \Leftarrow v] \rfloor_i$ is $e[x \Leftarrow v_i]$, which, by hypothesis, reduces to v'_i. By *completeness*, there exists a value v' such that $e[x \Leftarrow v] \rightarrow^\star v'$. Then, by *soundness*, we have $\lfloor v' \rfloor_i = v'_i$ for all $i \in \{1, 2\}$. Subject reduction yields $v' : \mathsf{int}^{\mathrm{L}}$. No-low-brackets yields $\lfloor v' \rfloor_1 = \lfloor v' \rfloor_2$.

In short: the task is to prove $e[x \Leftarrow v_1]$ and $e[x \Leftarrow v_2]$ "*bisimilar*", i.e. to relate their reducts. λ^2 and its type system provide a structured description of the bisimulation *invariant*.

This approach can be transported to the π-calculus.

Conclusion

I have sketched several *two-step* approaches to establishing the correctness of a type-based information flow analysis, separating a purely *dynamic* analysis, on the one hand, and a *static* approximation, on the other hand.

- these approaches yield manageable, modular proofs;
- on the down-side, not all analyses can be decomposed in this way – the dynamic analysis may require some type information, introducing a circularity.

References

More details and references to related work can be found within:

- François Pottier and Sylvain Conchon, *Information flow inference for free*, ICFP'00.
- François Pottier and Vincent Simonet, *Information flow inference for ML*, POPL'02.
- François Pottier, *A Simple View of Type-Secure Information Flow in the π-Calculus*, CSFW'02.
- Syvain Conchon, *Secure information flow in the Join-Calculus*, manuscript, 2002.