Secure information flow for sequential programs

Ilaria Castellani

INDES team, INRIA Sophia Antipolis

Master UBINET, Secure diffuse computing, Lecture 7b

November 25, 2009

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.

Language-based security

- Information: contained in "objects", used by "subjects".
- Objects have security levels, e.g. high = secret, low = public.
- Secure information flow (for confidentiality): no flow towards lower or incomparable levels.

$x_L := y_H$	not secure
$z_H := y_H \ ; \ x_L := 0$	secure

- Imperative languages:
  - Subjects = programs. Objects = variables. Tools:
  - Security property based on a semantic equivalence;
  - Type systems to statically ensure the property.

The Volpano-Smith-Irvine (VSI) approach

Lattice model of information flow: based on work by Bell and LaPadula, Denning and others in the late 70's.

Each variable x has a security level $sec(x) = \ell$. Security levels form a lattice (which represents the security policy, e.g. confidentiality).

Example. The simplest nontrivial lattice is $\{L, H\}$, with $L \leq H$.

$sec(x) = H$	secret information
$sec(x) = L$	public information

Information flow from x to y is secure when $sec(x) \leq sec(y)$

$\Rightarrow$ the only secure flow is upward flow

The language BabyIMP

Variables x, y, z, values v, v' and expressions e, e':

$$e ::= x \mid v \mid e \; op \; e'$$

Boolean and integer expressions, built with standard operations.

Syntax of commands (or programs) c, d:

$$c,\, d \quad ::= \quad \texttt{nil} \mid x := e \mid c\,;\, d \mid \texttt{if}\ (e)\ \texttt{then}\ \{c\}\ \texttt{else}\ \{d\}$$

Small-step semantics, defined on configurations $\langle c, \sigma \rangle$, where c is a command and σ is a state (a mapping from variables to values):

$$\langle c, \sigma \rangle \to \langle c', \sigma' \rangle$$

Semantics of expressions

State $\sigma : Var \to Val$

$[\![e]\!]\sigma$: semantics of expression e in state σ, defined by:

$$
\begin{array}{lcll}
[\![x]\!]\sigma & = & \sigma(x) & (variables) \\
[\![v]\!]\sigma & = & v & (constants) \\
[\![e \; op \; e']\!]\sigma & = & [\![e]\!] \; op \; [\![e']\!] &
\end{array}
$$

Operational semantics of BabyIMP (1/2)

Rules for assignment and sequential composition:

$$\text{(Assign-Op)} \quad \frac{[\![e]\!]\sigma = v}{\langle x := e, \sigma\rangle \to \langle \texttt{nil}, \sigma[v/x]\rangle}$$

$$\text{(Seq-Op1)} \quad \frac{}{\langle \texttt{nil}; c, \sigma\rangle \to \langle c, \sigma\rangle}$$

$$\text{(Seq-Op2)} \quad \frac{\langle c, \sigma\rangle \to \langle c', \sigma'\rangle}{\langle c\,; d, \sigma\rangle \to \langle c'; d, \sigma'\rangle}$$

Operational semantics of BabyIMP (2/2)

Rules for conditional:

$$\text{(Cond-Op1)} \quad \frac{[\![e]\!]\sigma = tt}{\langle \texttt{if } (e) \texttt{ then } \{c\} \texttt{ else } \{d\}, \sigma\rangle \to \langle c, \sigma\rangle}$$

$$\text{(Cond-Op2)} \quad \frac{[\![e]\!]\sigma = ff}{\langle \texttt{if } (e) \texttt{ then } \{c\} \texttt{ else } \{d\}, \sigma\rangle \to \langle d, \sigma\rangle}$$

Big step semantics of BabyIMP

A more abstract semantics, used to define a semantic equivalence $\approx$ on programs, as well as the security property.

Big step semantics:

$\langle c, \sigma\rangle \Downarrow \sigma'$: when run on state σ, c terminates with state σ'

$$\langle c, \sigma\rangle \Downarrow \sigma' \quad \stackrel{\text{def}}{=} \quad \exists\, \langle c_0, \sigma_0\rangle, \ldots, \langle c_n, \sigma_n\rangle,\ n \geq 0 \quad \text{such that}$$

$$\langle c, \sigma\rangle = \langle c_0, \sigma_0\rangle \rightarrow \cdots \rightarrow \langle c_n, \sigma_n\rangle = \langle \texttt{nil}, \sigma'\rangle$$

Semantic equivalence:

$$c \approx d \quad \textit{if} \quad \forall \sigma.\ \langle c, \sigma\rangle \Downarrow \sigma' \ \Leftrightarrow\ \langle d, \sigma\rangle \Downarrow \sigma'$$

Security property for BabyIMP

Noninterference: the final value of variables of a given level does not depend on the initial value of higher or incomparable variables.

ℓ-equality on states:

$$\sigma_1 =_\ell \sigma_2 \text{ if } \mathtt{dom}(\sigma_1) = \mathtt{dom}(\sigma_2) \text{ and}$$
$$(\, x \in \mathtt{dom}(\sigma_i) \;\wedge\; sec(x) \leq \ell \,) \;\Rightarrow\; \sigma_1(x) = \sigma_2(x)$$

ℓ-noninterference:

A command c is ℓ-noninterferent if $\forall \sigma_1, \sigma_2$ such that $\sigma_1 =_\ell \sigma_2$:

$$(\; \exists \sigma_1' \,.\, \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \exists \sigma_2' \,.\, \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \;) \;\Rightarrow\; \sigma_1' =_\ell \sigma_2'$$

Security:

A command c is secure if it is ℓ-noninterferent for any level ℓ.

BabyIMP security examples

Security lattice $\{L, H\}$ with $L \leq H$ and $sec(x) = H, sec(y) = L$.

Insecure programs

$y_L := x_H$ direct leak

$\texttt{if } (x_H = 0) \texttt{ then } \{y_L := 0\} \texttt{ else } \{y_L := 1\}$ indirect leak

Secure programs

$x_H := 0 \,;\, y_L := 0$

$x_H := 0 \,;\, y_L := x_H$

$\texttt{if } (x_H = 0) \texttt{ then } \{y_L := 0\} \texttt{ else } \{y_L := 0\}$ equal branches

BabyIMP security: exercises

Exercise 1.

Security lattice $\{L, H\}$ with $L \leq H$ and $sec(x) = H, sec(y) = L$.

Question 1a. Is the following program secure?

$$\texttt{if } (x_H = 0) \quad \texttt{then } \{\texttt{if } (x_H = 0) \texttt{ then } \{y_L := 0\} \texttt{ else } \{\texttt{nil}\}\}$$
$$\texttt{else } \{\texttt{if } (x_H = 0) \texttt{ then } \{y_L := 0\} \texttt{ else } \{\texttt{nil}\}\}$$

Hint: the two branches of the conditional are equal but not secure.

Question 1b.

Is a conditional with two secure equal branches always secure?

BabyIMP security: exercises (ctd)

Exercise 2.

Assume a lattice with 5 elements: Public, Secret, Alice, Bob, Carol, and with order: Alice $\leq$ Bob and $\forall k.\ \mathsf{Public} \leq k \ \wedge\ k \leq \mathsf{Secret}$.

Suppose that $sec(x_H) = \mathsf{Secret}$, $sec(y_L) = \mathsf{Public}$, $sec(z_A) = \mathsf{Alice}$, $sec(z_B) = \mathsf{Bob}$ and $sec(z_C) = \mathsf{Carol}$.

Question 2a. Are the following programs secure?

1. $\texttt{if}\ (z_A \neq z_B)\ \texttt{then}\ \{z_B := z_A\}\ \texttt{else}\ \{\texttt{nil}\}$
2. $\texttt{if}\ (z_A \neq z_C)\ \texttt{then}\ \{z_B := z_A\}\ \texttt{else}\ \{\texttt{nil}\}$
3. $\texttt{if}\ (z_A \neq z_C)\ \texttt{then}\ \{z_B := z_A\}\ \texttt{else}\ \{z_B := z_C\}$
4. $\texttt{if}\ (z_A < z_C)\ \texttt{then}\ \{x_H := z_A\}\ \texttt{else}\ \{x_H := z_C\}$

BabyIMP security: exercises (ctd)

Question 2b. The following program:

$$z_C := z_B \ ; \ z_A := z_C$$

is not secure because the first command is not C-noninterferent and the second command is not A-noninterferent.

If we allowed flows between two incomparable levels (by slightly changing the definition of $=_\ell$, can you see how?), then the two commands $z_C := z_B$ and $z_A := z_C$ would be secure but the whole program would still be insecure. Can you explain why?

Hint: the program leaks information from Bob to Alice.

The language IMP

Extension of BabyIMP with while-loops.

Syntax of programs (or commands) c, d:

$$c,\, d \quad ::= \quad \texttt{nil} \;\mid\; x := e \;\mid\; c\,;\, d \;\mid\; \texttt{if}\ (e)\ \texttt{then}\ \{c\}\ \texttt{else}\ \{d\} \;\mid$$

$$\texttt{while}\ (e)\ \texttt{do}\ \{c\}$$

Abbreviation: $\texttt{loop}\ c \stackrel{\text{def}}{=} \texttt{while}\ (tt)\ \texttt{do}\ \{c\}$

Operational semantics of IMP

Previous rules + two new rules for while-loops:

$$\text{(While-Op1)} \qquad \frac{[\![e]\!]\sigma = tt}{\langle \texttt{while}\ (e)\ \texttt{do}\ \{c\}, \sigma\rangle \to \langle c\,;\ \texttt{while}\ (e)\ \texttt{do}\ \{c\}, \sigma\rangle}$$

$$\text{(While-Op2)} \qquad \frac{[\![e]\!]\sigma = ff}{\langle \texttt{while}\ (e)\ \texttt{do}\ \{c\}, \sigma\rangle \to \langle \texttt{nil}, \sigma\rangle}$$

Big step semantics of IMP

While loops introduce the possibility of divergent behaviours.

Big step semantics:

$\langle c, \sigma \rangle \Downarrow \sigma'$: when run on state σ, c terminates with state σ'

$\langle c, \sigma \rangle \Uparrow$: when run on state σ, c diverges

Formally, $\langle c, \sigma \rangle \Uparrow \stackrel{\text{def}}{=} \nexists \sigma' . \langle c, \sigma \rangle \Downarrow \sigma'$

Examples

$\forall \sigma . \sigma(x) > 1 : \langle \texttt{while}\ (x \neq 1)\ \texttt{do}\ \{x := x + 1\}, \sigma \rangle \Uparrow$

$\forall c, \forall \sigma : \langle \texttt{loop}\ c\ , \sigma \rangle \Uparrow$

Security property for IMP

Because of while-loops, noninterference (NI) has to be refined.

Two possibilities:

- termination-insensitive NI: ignores divergent computations. Requires same result only if program converges in both states.
- termination-sensitive NI: requires the same termination behaviour on each pair of equivalent states.

Example. The program

$$\texttt{while}\ (x_H \neq 0)\ \texttt{do}\ \{\texttt{nil}\}\ ;\ \ y_L := 0$$

is secure for termination-insensitive NI (whenever it terminates, it yields $y_L = 0$), but not for termination-sensitive NI.

Security property for IMP (ctd)

1. termination-insensitive NI: ignores divergent computations.

   Command c is ℓ-TI-noninterferent if $\forall \sigma_1, \sigma_2$ s.t. $\sigma_1 =_\ell \sigma_2$:

   $$(\exists \sigma_1' \,.\, \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \exists \sigma_2' \,.\, \langle c, \sigma_2 \rangle \Downarrow \sigma_2') \Rightarrow \sigma_1' =_\ell \sigma_2'$$

2. termination-sensitive NI: requires same termination behaviours.

   Command c is ℓ-TS-noninterferent if $\forall \sigma_1, \sigma_2$ s.t. $\sigma_1 =_\ell \sigma_2$:

   $$\exists \sigma_1' \,.\, \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \Rightarrow (\exists \sigma_2' \,.\, \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \wedge \sigma_1' =_\ell \sigma_2')$$

Property 2 stronger than Property 1. They coincide in BabyIMP.

IMP security examples

Security lattice $\{L, H\}$ with $L \leq H$ and $sec(x) = H, sec(y) = L$.

TI-secure but not TS-secure programs

$$\texttt{while } (x_H \neq 0) \texttt{ do } \{\texttt{nil}\} \texttt{ ; } y_L := 0$$

$$\texttt{if } (x_H = 0) \texttt{ then } \{\texttt{nil}\} \texttt{ else } \{\texttt{loop}\} \texttt{ ; } y_L := 0$$

Termination leaks: ignored by TI-security but not by TS-security.
May arise even in conditionals with equal secure high branches:

$$\begin{array}{l} \texttt{if } (x_H \leq 1) \quad \texttt{then } \{\texttt{while } (x \neq 1) \texttt{ do } \{x := x + 1\}\} \\ \qquad\qquad\quad \texttt{else } \{\texttt{while } (x \neq 1) \texttt{ do } \{x := x + 1\}\} \texttt{ ;} \\ y_L := 0 \end{array}$$