Secure Information Flow by Self-Composition

Gilles Barthe, Pedro D'Argenio and Tamara Rezk

EVEREST TEAM

INRIA SOPHIA ANTIPOLIS

http://www-sop.inria.fr/everest/

CSFW - June 2004

Outline of the Talk

- Non-Interference: 2 in 1
- Secure Flow: A Generalisation
- Possibilistic Security (non-deterministic)
- Characterization using CTL
- Final Considerations

Non Interference

Non Interference is a semantic property about any

TWO program executions

$$\left.\begin{array}{lcl} P(\vec{x_1}, \vec{y_1}) & \rightsquigarrow^{\star} & (\vec{x_2}, \vec{y_2}) \\ P(\vec{x'_1}, \vec{y'_1}) & \rightsquigarrow^{\star} & (\vec{x'_2}, \vec{y'_2}) \end{array}\right\} (\vec{x_1} = \vec{x'_1}) \Rightarrow (\vec{x_2} = \vec{x'_2})$$

x represents public

y represents confidential

Non-Interference

Type Systems

Non-Interference

Type Systems

- Static analysis to determine if a program is non-interfering

Non-Interference

Type Systems

- Static analysis to determine if a program is non-interfering
- Nice, but too conservative

Non-Interference

Type Systems

- Static analysis to determine if a program is non-interfering
- Nice, but too conservative
- E.g. this program is usually rejected as insecure $x := y; x := 0$

Non-Interference

Type Systems

- Static analysis to determine if a program is non-interfering
- Nice, but too conservative
- E.g. this program is usually rejected as insecure $x := y; x := 0$
- Even more conservative with features that are notoriously difficult to handle

Our approach

Non Interference as a semantic property of every

SINGLE program execution

- Based on the observation that NI can be reduced to a property about every single program execution
- Use verification logics (e.g. Programming Logics & Temporal Logics) and borrow all the know-how.

Non-Interference Revisited

$$\left.\begin{array}{lll} P(\vec{x_1},\vec{y_1}) & \rightsquigarrow^\star & (\vec{x_2},\vec{y_2}) \\ P(\vec{x'_1},\vec{y'_1}) & \rightsquigarrow^\star & (\vec{x'_2},\vec{y'_2}) \end{array}\right\} (\vec{x_1} = \vec{x'_1}) \Rightarrow (\vec{x_2} = \vec{x'_2})$$

NI can be rewritten using ";"

$$P; P[\vec{z'}/\vec{z}](\vec{x_1} \oplus \vec{x_1}', \vec{y_1} \oplus \vec{y_1}') \rightsquigarrow^\star (\vec{x_2} \oplus \vec{x_2}', \vec{y_2} \oplus \vec{y_2}')$$
$$\Rightarrow (\vec{x_1} = \vec{x'_1}) \Rightarrow (\vec{x_2} = \vec{x'_2})$$

$[\vec{z'}/\vec{z}]$ renames all variables with new names

Consequences of the Observation

$$P; P[\vec{z'}/\vec{z}](\vec{x_1} \oplus \vec{x_1}', \vec{y_1} \oplus \vec{y_1}') \rightsquigarrow^{\star} (\vec{x_2} \oplus \vec{x_2}', \vec{y_2} \oplus \vec{y_2}')$$

Because of soundness & completeness of Hoare Logic, it is equivalent to:

$$\{\vec{x} = \vec{x'}\}P; P[\vec{z'}/\vec{z}]\{\vec{x} = \vec{x'}\}$$

Example

if y_{secret}=0 then x_{public}:=0 else $x_{public} := 0$

Execution of this program is:

$$(x_{public} = 2, y_{secret} = 1) \quad \leadsto \quad (x_{public} = 0, y_{secret} = 1)$$

$$(x_{public} = 2, y_{secret} = 0) \quad \leadsto \quad (x_{public} = 0, y_{secret} = 0)$$

Example

if y_{secret}=0 then x_{public}:=0 else $x_{public} := 0$;
if y'_{secret}=0 then x'_{public}:=0 else $x'_{public} := 0$
Execution of this program is:

$$(x_{public} = 2, x'_{public} = 2, y_{secret} = 1, y'_{secret} = 0) \leadsto$$

$$(x_{public} = 0, x'_{public} = 2, y_{secret} = 1, y'_{secret} = 0) \leadsto$$

$$(x_{public} = 0, x'_{public} = 0, y_{secret} = 1, y'_{secret} = 0)$$

Example: Hoare Logic

Because of (relative) completeness of Hoare Logic, we can prove:

$$\{\vec{x} = \vec{x'}\}$$

if y_{secret}=0 then x_{public}:=0 else $x_{public} := 0$;

if y'_{secret}=0 then x'_{public}:=0 else $x'_{public} := 0$

$$\{\vec{x} = \vec{x'}\}$$

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, pin = 234234) \leadsto (acc = false, in = 222, pin = 234234)$$

$$(acc = false, in = 222, pin = 222) \leadsto (acc = true, in = 222, pin = 222)$$

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, \qquad\qquad) \leadsto (acc = false, in = 222, \qquad\qquad)$$

$$(acc = false, in = 222,) \leadsto (acc = true, in = 222,)$$

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, \qquad\qquad) \leadsto (acc = false, in = 222, \qquad\qquad)$$

$$(acc = false, in = 222,) \leadsto (acc = true, in = 222,)$$

- This program IS interfering, but . . .

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, \qquad\qquad) \rightsquigarrow (acc = false, in = 222, \qquad\qquad)$$

$$(acc = false, in = 222,) \rightsquigarrow (acc = true, in = 222,)$$

- This program IS interfering, but . . .
- is it really insecure? . . .

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, \qquad\qquad) \rightsquigarrow (acc = false, in = 222, \qquad\qquad)$$

$$(acc = false, in = 222,) \rightsquigarrow (acc = true, in = 222,)$$

- This program IS interfering, but . . .
- is it really insecure? . . . It depends on the security policy.

A Program that is interfering

if $in = pin$ then acc:=true else $acc := false$

Execution of this program is:

$$(acc = false, in = 222, \qquad\qquad) \rightsquigarrow (acc = false, in = 222, \qquad\qquad)$$

$$(acc = false, in = 222,) \rightsquigarrow (acc = true, in = 222,)$$

- This program IS interfering, but . . .
- is it really insecure? . . . It depends on the security policy.
- NI is too strong to characterize some security policies.

Secure Flow: A General Characterization

if $in = pin$ then acc:=true else $acc := false$

- The declassified information should only reveal whether the input code agrees with the PIN number or not.

Secure Flow: A General Characterization

if $in = pin$ then acc:=true else $acc := false$

- The declassified information should only reveal whether the input code agrees with the PIN number or not.
- $(\mu, \mu') \in I_1$ iff $\mu(in) = \mu(pin) \Leftrightarrow \mu'(in) = \mu'(pin)$

Secure Flow: A General Characterization

if $in = pin$ then acc:=true else $acc := false$

- The declassified information should only reveal whether the input code agrees with the PIN number or not.
- $(\mu, \mu') \in I_1$ iff $\mu(in) = \mu(pin) \Leftrightarrow \mu'(in) = \mu'(pin)$
- $(\mu, \mu') \in I_2$ iff $\mu(acc) = \mu(acc')$

Secure Flow: A General Characterization

if $in = pin$ then acc:=true else $acc := false$

- The declassified information should only reveal whether the input code agrees with the PIN number or not.
- $(\mu, \mu') \in I_1$ iff $\mu(in) = \mu(pin) \Leftrightarrow \mu'(in) = \mu'(pin)$
- $(\mu, \mu') \in I_2$ iff $\mu(acc) = \mu(acc')$
- In Hoare Logic:

$$\{(in = pin) \leftrightarrow (in' = pin')\}$$

if $in = pin$ then acc:=true else $acc := false$;

if $in' = pin'$ then acc':=true else $acc' := false$

$$\{acc = acc'\}$$

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)
- Termination Sensitive Security

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)
- Termination Sensitive Security
- Possibilistic Security (termination (in)sensitive)

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)
- Termination Sensitive Security
- Possibilistic Security (termination (in)sensitive)
- Any language, including languages with pointers, featuring a composition satisfying a couple of properties, including ";" and "$\|$".

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)
- Termination Sensitive Security
- Possibilistic Security (termination (in)sensitive)
- Any language, including languages with pointers, featuring a composition satisfying a couple of properties, including ";" and "$\|$".
- Any logic, including:
  0. Hoare Logic and Separation Logic
  0. LTL and CTL

Secure Flow: A General Characterization

- General Securities Policies, including declassification (delimited release)
- Termination Sensitive Security
- Possibilistic Security (termination (in)sensitive)
- Any language, including languages with pointers, featuring a composition satisfying a couple of properties, including ";" and "$\|$".
- Any logic, including:
  0. Hoare Logic and Separation Logic
  0. LTL and CTL
- Specification Languages and calculus:
  0. JML
  0. wp-calculus

Possibilistic Security (TS)

$$\left.\begin{array}{l} P(\vec{z_1}) \rightsquigarrow^{\star} (\vec{z_2}) \\ \text{and } \ \vec{z_1}, \vec{z_1'} \in I_1 \end{array}\right\} \Rightarrow \exists z_2' : P(\vec{z_1'}) \rightsquigarrow^{\star} (\vec{z_2'}) \ \text{ and } \ \vec{z_2}, \vec{z_2'} \in I_2$$

Possibilistic Security (TS)

$$\left.\begin{array}{l} P(\vec{z_1}) \leadsto^\star (\vec{z_2}) \\ \text{and } \ \vec{z_1}, \vec{z_1'} \in I_1 \end{array}\right\} \Rightarrow \exists z_2' : P(\vec{z_1'}) \leadsto^\star (\vec{z_2'}) \ \text{ and } \ \vec{z_2}, \vec{z_2'} \in I_2$$

Can be alternatively defined using ";" by:

$$P; P[\vec{z'}/\vec{z}](\vec{z_1}, \vec{z_1'}) \leadsto^\star P[\vec{z'}/\vec{z}](\vec{z_2}, \vec{z_1'}) \ \text{ and } \ \vec{z_1}, \vec{z_1'} \in I_1 \Rightarrow$$
$$\exists z_2' : P[\vec{z'}/\vec{z}](\vec{z_2}, \vec{z_1'}) \leadsto^\star (\vec{z_2}, \vec{z_2'}) \text{and } \ \vec{z_2}, \vec{z_2'} \in I_2$$

Characterizing Possibilistic Security with CTL

Extend $(Conf, \rightsquigarrow)$ with a function $Prop(P, c)$ to sets of atomic propositions:

- mid $\in Prop(P', c)$ iff $P' = P[z'/z]$ (middle of self-compose program)
- Ind[I] $\in Prop(P, c)$ iff $c(z), c(z') \in I$ (indistinguishability)
- end $\in Prop(P, c)$ iff c is a terminating confi guration (end of program)

Characterizing Possibilistic Security with CTL

$$P; P[\vec{z'}/\vec{z}](\vec{z_1}, \vec{z'_1}) \rightsquigarrow^\star P[\vec{z'}/\vec{z}](\vec{z_2}, \vec{z'_1}) \text{ and } \vec{z_1}, \vec{z'_1} \in I_1 \Rightarrow$$
$$\exists z'_2 : P[\vec{z'}/\vec{z}](\vec{z_2}, \vec{z'_1}) \rightsquigarrow^\star (\vec{z_2}, \vec{z'_2})\text{and } \vec{z_2}, \vec{z'_2} \in I_2$$

Then, in our characterization:

$$\mathrm{Ind}[I_1] \mapsto AG \,\mathrm{mid} \mapsto EF(\mathrm{end} \wedge \mathrm{Ind}[I_2])$$

Final Considerations

- Limited to branching temporal Logics (CTL, $\mathrm{CTL}^{\star}$, μ-calculus)
- LTL can also characterize both types of security BUT limited to determinism (The CTL formula AG($\ldots$EF) is not expressible in LTL)
- It can be done in wp-calculus+predicate logic with our technique but limited to determinism

Final Considerations

- Completeness allows to reuse known proof rules and automate or shorter proofs of NI.

Final Considerations

- Completeness allows to reuse known proof rules and automate or shorter proofs of NI.

$$\frac{\vec{y} : \mathsf{high},\ \vec{x} : \mathsf{low} \vdash P : \tau\ \mathsf{cmd}}{\{\vec{x} = \vec{x}'\}\ P; (P[\vec{x}', \vec{y}'/\vec{x}, \vec{y}])\ \{\vec{x} = \vec{x}'\}}$$

$$\frac{\begin{array}{c}\{\vec{x} = \vec{x}'\}\ P; P[\vec{x}', \vec{y}'/\vec{x}, \vec{y}]\ \{\vec{x} = \vec{x}'\} \\ \{\vec{x} = \vec{x}'\}\ Q; Q[\vec{x}', \vec{y}'/\vec{x}, \vec{y}]\ \{\vec{x} = \vec{x}'\}\end{array}}{\{\vec{x} = \vec{x}'\}\ (P; Q); (P; Q)[\vec{x}', \vec{y}'/\vec{x}, \vec{y}]\ \{\vec{x} = \vec{x}'\}}$$

Final Considerations

- Inmediate use of model checkers such as SMV or SPIN.
- An aside contribution is to provide a general method to check secure flow for languages which no type system is known (e.g. a language with pointers and arithmetic for pointers).

Final Considerations

- Related Work
  - Joshi and Leino 2000: characterisation of Possibilistic TS NI in the wp-calculus.
  - Darvas, Hahnle and Sands 2003: characterisation of Possibilistic NI in Dynamic Logic using self composition.
  - Amtoft and Banerjee 2004: information flow analysis in Logical Form
  - Giacobazzi and Matroaini 2004: abstract non-interference

Separation Logic

- $\{e \mapsto (\_, e_2)\}\ e.i := e_1\ \{e \mapsto (e_1, e_2)\}$
- x does not occur in e_1 or in e_2 then {empty } $x := \mathrm{cons}(e_1, e_2)$ $\{x \mapsto (e_1, e_2)\}$
- If x, x' and x'' are different and x does not occur in e, then $\{x{=}x' \wedge (e \mapsto (x'', e_2))\}\ x := e.1\ \{x{=}x'' \wedge (e \mapsto (x'', e_2))\}$

A predicate recursively defined in the Logic:

$$\begin{aligned} \mathrm{list}.[\,].p &= (p = nil) \\ \mathrm{list}.(x :: xs).p &= (\exists r : (p \mapsto (x, r)) * \mathrm{list}.xs.r) \end{aligned}$$

A Characterization in Separation Logic

Let $\mathbf{I}_{sl}$ be:

$$\exists \vec{xs}, \vec{xs}' : \left(\left(\bigwedge_{1 \le i \le n} \mathsf{list}.xs_i.x_i \right) * \left(\bigwedge_{1 \le i \le n} \mathsf{list}.xs'_i.x'_i \right) \right.$$

$$\left. \vec{xs} = \vec{xs} \right)$$

$\mathbf{I}_{sl}$ has two parts: the first part states the separation of the heap, the second one, the indistinguishability of the values.