

Information flow

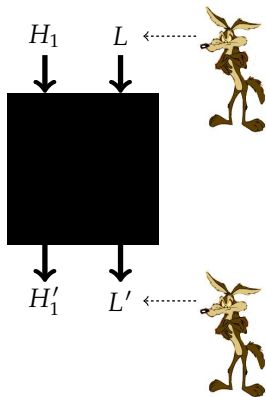
David Pichardie
Thomas Jensen

Master recherche Université Rennes 1
Module SDL
November 2011

Information flow type system

Non-interference

"Low-security behavior of the program is not affected by any high-security data." Goguen & Meseguer 1982

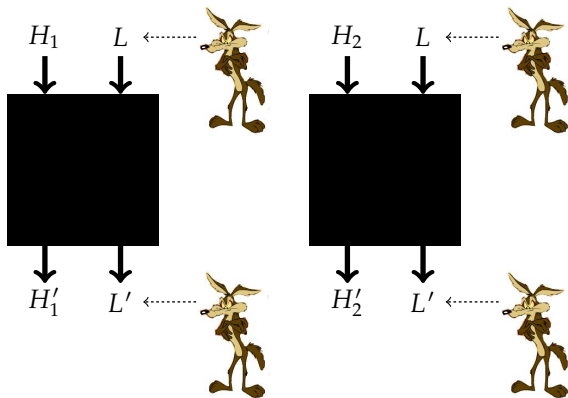


High = confidential

Low = public

Non-interference

"Low-security behavior of the program is not affected by any high-security data." Goguen & Meseguer 1982



High = confidential

Low = public

Motivation

Today we will present a simple information flow type system¹ and prove it enforces the semantic non-interference property on well typed programs.

1. D. Volpano and G. Smith, *A Type-Based Approach to Program Security*, Theory and Practice of Software Development, 1997.

Program syntax

We consider a standard WHILE language but we mix arithmetic and boolean expressions.

Expr ::=	n	$n \in \mathbb{Z}$
	x	$x \in \mathbb{V}_H \uplus \mathbb{V}_L$
	Expr o Expr	$o \in \{+, -, \times, \dots\}$
	Expr c Expr	$c \in \{=, \neq, <, \leq, \dots\}$
	Expr b Expr	$b \in \{\text{and, or}\}$
Stm ::=	$x :=$ Expr	
	if Expr then Stm else Stm	
	while Expr do Stm	
	Stm ; Stm	

The set of variable is partitioned in two disjoint sets :

- ▶ \mathbb{V}_H : high (or secret) variables
- ▶ \mathbb{V}_L : low (or public) variables

Secure programs

Intuitively², a program is *secure* (or *non interfering*) if the final values of low variables do not depend on the initial values of the high variables.

Examples : are this programs secure or not ?

- 1 `h := 1`
- 2 `l := h`
- 3 `if (h1>h2) then {l := 1} else {l := 2}`
- 4 `while (h) do { l := l+1 }; l := 0`

2. This notion will be defined formally when presenting the semantics of the language.

A lattice of security levels

We consider here only two kind of informations (low and high), but the information flow policy can be defined as a lattice of *security levels*.



We note \sqsubseteq the corresponding partial order.

We say there is a flow of information from x to y if the value of the variable y depends on the value of the variable x .

If x (resp. y) is of level k_x (resp. k_y), the flow is

- ▶ licit if $k_x \sqsubseteq k_y$
- ▶ illicit if $k_x \not\sqsubseteq k_y$

A simple information flow type system (1/2)

Non-interference can be enforced by an *information flow type system*

Typing judgment for expressions : $e \in \mathbf{Expr}$, $\tau \in \{L, H\}$

$$\vdash e : \tau$$

Meaning : the expression e depends only of variable of level τ or lower.

Typing rules :

$$\text{CONST} \frac{}{\vdash n : L} \quad \text{VAR} \frac{x \in \mathbb{V}_\tau}{\vdash x : \tau} \quad \text{BINOP} \frac{\vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash e_1 \circ e_2 : \tau}$$

$$\text{EXP-SUBTYP} \frac{\vdash e : \tau_1 \quad \tau_1 \sqsubseteq \tau_2}{\vdash e : \tau_2}$$

Exemple

A type derivation for $\vdash h + 1 : H$

$$\text{BINOP} \frac{\text{VAR} \frac{h \in \mathbb{V}_H}{\vdash h : H} \quad \text{EXP-SUBTYP} \frac{\text{CONST} \frac{}{\vdash 1 : L} \quad L \sqsubseteq H}{\vdash 1 : H}}{\vdash h + 1 : H}$$

Exercise : is there another type derivation for $\vdash h + 1 : H$?

A simple information flow type system (2/2)

Typing judgment for statements : $S \in \mathbf{Stm}$, $\tau \in \{L, H\}$

$$\vdash S : \tau$$

Meaning : the only variables modified by statement S are of level τ or higher.

Typing rules :

$$\begin{array}{c} \text{ASSIGN} \frac{x \in \mathbb{V}_\tau \quad \vdash e : \tau}{\vdash x := e : \tau} \quad \text{SEQ} \frac{\vdash S_1 : \tau \quad \vdash S_2 : \tau}{\vdash S_1 ; S_2 : \tau} \\ \\ \text{IF} \frac{\vdash e : \tau \quad \vdash S_1 : \tau \quad \vdash S_2 : \tau}{\vdash \text{if } e \text{ then } S_1 \text{ else } S_2 : \tau} \quad \text{WHILE} \frac{\vdash e : \tau \quad \vdash S : \tau}{\vdash \text{while } e \text{ do } S : \tau} \\ \\ \text{STM-SUBTYP} \frac{\vdash S : \tau_2 \quad \tau_1 \sqsubseteq \tau_2}{\vdash S : \tau_1} \end{array}$$



The subtype relation on statements is *contravariant* !

Exercise

Try to type the following statements (give a derivation type, if possible) :

`if (l) then h := l else l := 0`

`if (h) then h := l else l := 0`

A natural semantics

State = **Var** \rightarrow \mathbb{Z}

$\llbracket \cdot \rrbracket \in$ **Expr** \rightarrow **State** \rightarrow \mathbb{Z} (semantics of expression)

$(\cdot, \cdot) \Downarrow \cdot \subseteq$ (**Stm** \times **State**) \times **State** (semantics of statement)

$$\llbracket n \rrbracket s = \mathcal{N}[\llbracket n \rrbracket]$$

$$\llbracket x \rrbracket s = s(x)$$

$$\llbracket e_1 + e_2 \rrbracket s = \llbracket e_1 \rrbracket s + \llbracket e_2 \rrbracket s$$

...

$$(x := e, s) \Downarrow s[x \mapsto \llbracket e \rrbracket s] \quad \frac{(S_1, s) \Downarrow s' \quad (S_2, s') \Downarrow s''}{(S_1; S_2, s) \Downarrow s''}$$

$$\frac{(S_1, s) \Downarrow s' \quad \llbracket e \rrbracket s = 1}{(\text{if } e \text{ then } S_1 \text{ else } S_2, s) \Downarrow s'}$$

$$\frac{(S_2, s) \Downarrow s' \quad \llbracket e \rrbracket s = 0}{(\text{if } e \text{ then } S_1 \text{ else } S_2, s) \Downarrow s'}$$

$$\frac{(S, s) \Downarrow s' \quad (\text{while } e \text{ do } S, s') \Downarrow s'' \quad \llbracket e \rrbracket s = 1}{(\text{while } e \text{ do } S, s) \Downarrow s''}$$

$$\frac{\llbracket e \rrbracket s = 0}{(\text{while } e \text{ do } S, s) \Downarrow s}$$

The observational power of an attacker

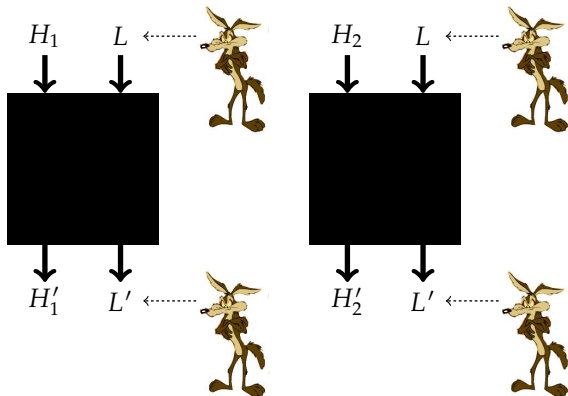
The attacker only see low variable before and after executions.

We model his observational power with an *indistinguishability* relation $\sim \subseteq \mathbf{State} \times \mathbf{State}$ between states.

$$s_1 \sim s_2 \text{ iff } \forall x \in \mathbb{V}_L, s_1(x) = s_2(x)$$

Non-interference

"Low-security behavior of the program is not affected by any high-security data." Goguen & Meseguer 1982

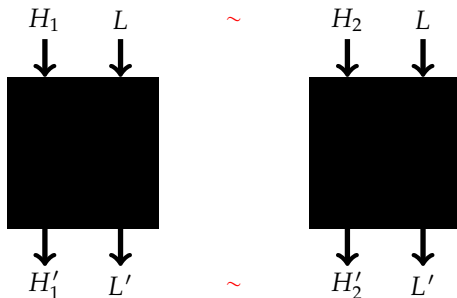


High = confidential

Low = public

Non-interference

"Low-security behavior of the program is not affected by any high-security data." Goguen & Meseguer 1982



$$\forall s_1, s_2, s'_1, s'_2, \quad s_1 \sim s_2 \wedge (P, s_1) \Downarrow s'_1 \wedge (P, s_2) \Downarrow s'_2 \implies s'_1 \sim s'_2$$

High = confidential

Low = public

Type soundness

A statement S is said *non interferent* iff
$$\left. \begin{array}{l} s_1 \sim s_2 \\ (S, s_1) \Downarrow s'_1 \\ (S, s_2) \Downarrow s'_2 \end{array} \right\} \text{implies } s'_1 \sim s'_2$$

Theorem

Every typable statement (i.e. such that $\exists \tau, \vdash S : \tau$) is non interferent.

Type soundness proof : step 1

We need a new set of typing rules

$$\text{CONST}' \frac{}{\vdash_s n : \tau} \quad \text{VAR}' \frac{x \in \mathbb{V}_\tau \quad \tau \sqsubseteq \tau'}{\vdash_s x : \tau'} \quad \text{BINOP} \frac{\vdash_s e_1 : \tau \quad \vdash_s e_2 : \tau}{\vdash_s e_1 \circ e_2 : \tau}$$

$$\text{ASSIGN}' \frac{x \in \mathbb{V}_\tau \quad \vdash_s e : \tau \quad \tau' \sqsubseteq \tau}{\vdash_s x := e : \tau'} \quad \text{SEQ} \frac{\vdash_s S_1 : \tau \quad \vdash_s S_2 : \tau}{\vdash_s S_1 ; S_2 : \tau}$$

$$\text{IF}' \frac{\vdash_s e : \tau \quad \vdash_s S_1 : \tau \quad \vdash_s S_2 : \tau \quad \tau' \sqsubseteq \tau}{\vdash_s \text{if } e \text{ then } S_1 \text{ else } S_2 : \tau'}$$

$$\text{WHILE}' \frac{\vdash_s e : \tau \quad \vdash_s S : \tau \quad \tau' \sqsubseteq \tau}{\vdash_s \text{while } e \text{ do } S : \tau'}$$

This type system is *syntax-directed* : at most one rule can be used for each statement (or expression).

Type soundness proof : step 1

Lemma (Sub-typing property)

For all $e \in \mathbf{Expr}$, $\tau, \tau' \in \{L, H\}$, $\vdash_s e : \tau$ and $\tau \sqsubseteq \tau'$ implies $\vdash_s e : \tau'$.

For all $S \in \mathbf{Stm}$, $\tau \in \{L, H\}$, $\vdash_s S : \tau'$ and $\tau \sqsubseteq \tau'$ implies $\vdash_s S : \tau$.

Proof. By induction on the typing judgment.

The new system is equivalent to the previous one.

Lemma

For all $e \in \mathbf{Expr}$, $\tau \in \{L, H\}$, $\vdash e : \tau$ implies $\vdash_s e : \tau$.

For all $S \in \mathbf{Stm}$, $\tau \in \{L, H\}$, $\vdash S : \tau$ implies $\vdash_s S : \tau$.

Proof. By induction on the typing judgment.

Lemma

For all $e \in \mathbf{Expr}$, $\tau \in \{L, H\}$, $\vdash_s e : \tau$ implies $\vdash e : \tau$.

For all $S \in \mathbf{Stm}$, $\tau \in \{L, H\}$, $\vdash_s S : \tau$ implies $\vdash S : \tau$.

Proof. By induction on the typing judgment.

Type soundness proof : step 2

Lemma (Low expressions)

For all $e \in \mathbf{Expr}$, $s_1, s_2 \in \mathbf{State}$, if $s_1 \sim s_2$ and $\vdash_s e : L$ then, $\llbracket e \rrbracket_{s_1} = \llbracket e \rrbracket_{s_2}$.

Proof. By induction on e .

Lemma (Side-effect of high statements)

For all $S \in \mathbf{Stm}$, $s, s' \in \mathbf{State}$, if $(S, s) \Downarrow s'$ and $\vdash_s S : H$ then $s \sim s'$.

Proof. By induction on the judgment $(S, s) \Downarrow s'$.

Type soundness proof : final step

Theorem (Type soundness)

For all $S \in \mathbf{Stm}$, $s_1, s_2, s'_1, s'_2 \in \mathbf{State}$, $\tau \in \{L, H\}$, if $s_1 \sim s_2$, $(S, s_1) \Downarrow s'_1$, $(S, s_2) \Downarrow s'_2$ and $\vdash S : \tau$ then $s'_1 \sim s'_2$.

Proof. By induction on the judgment $(S, s_1) \Downarrow s'_1$ and case analysis on $(S, s_2) \Downarrow s'_2$.

Conclusions

- ▶ Type checking is computable but non-interference is not
Exercice : give an example of non-interferent program that is not typable.
- ▶ We have ignored some information channels :
 - ▶ timing channels

```
if h>0 then l:=0 else { h:=h+1; l:=0 }
```

measuring the run-time of this program may reveal secret informations.

- ▶ termination channels

```
while h>0 do skip
```

- ▶ Non-inteference is sometimes a too strong property.
Example : a password checker always reveals some secret.

Information flow challenge

`http://ifc.hvergi.net/`

Declassification

Giving (some) information away

Code should not leak sensitive information.

However, some applications **intentionally** leak some confidential information :

- ▶ password checking
- ▶ statistics
- ▶ ...

Need for controlled information release or **declassification**

Controlling information release

Declassification release might compromise confidentiality.

Ensure that secrets are not leaked via release mechanisms.

Information release violates non-interference !

- ▶ cannot rely on previous type system to ensure security

What security guarantees for programs with declassification ?

An operator for declassification

We introduce a binary operator `declassify(exp, lvl)` that takes as arguments

- ▶ an expression *exp*
- ▶ a security level *lvl* such as high, low...

and declassifies the result of *exp* to the level *lvl*.

For example, (h_i are secrets, `avg`, `n` are low variables)

```
avg := declassify((h_1 + ... + h_n)/n, low)
```

Rejected by non-interference.

How to ensure that we are not declassifying more than intended ?

Delimited release

Principle :

Only release declassified data and no further information

Intuition : expression exp can be declassified in statement S if

all environments that are indistinguishable through exp
are undistinguishable through S .

Definition :

Assume

$$s_1 \sim s_2 \text{ and } (S, s_1) \Downarrow s'_1 \text{ and } (S, s_2) \Downarrow s'_2$$

The we must have

$$\llbracket exp \rrbracket_{s_1} = \llbracket exp \rrbracket_{s_2} \Rightarrow s'_1 \sim s'_2$$

Examples

Accepts

```
avg := declassify((h_1 + ... + h_n)/n, low)
```

Rejects

```
h2:=h1;...; hn:=h1;  
avg:=declassify((h1+...+hn)/n,low);
```

because it is semantically equivalent to

```
avg:=h1;
```

Type system for declassification

Idea :

prevent new information from flowing into variables used in declassifying expressions

Intuition : `exp` should not contain high variables other than `h` in

`h := exp ; ... ; declassify(h,low);`

Typing rules

$$\text{EXP-DECLASS} \frac{\vdash e : l', D}{\vdash \mathbf{declassify}(e, l) : l, \text{Vars}(e)}$$

$$\text{CMD-ASG} \frac{\vdash e : l', D}{\vdash x := e : \{x\}, D}$$

$$\text{CMD-SEQ} \frac{\vdash S_1 : U_1, D_1 \quad S_2 : U_2, D_2 \quad U_1 \cap D_2 = \emptyset}{\vdash S_1; S_2 : U_1 \cup U_2, D_1 \cup D_2}$$