

On typing information flow

Gerard Boudol

Kumar Avijit

15-819 Logics and Languages for Security
November 5, 2007

Motivation : The information flow problem

- ▶ Information flow deals with deciding if execution of a certain program fragment depends on another value/input/program.
- ▶ As such, the problem is undecidable.
- ▶ Conservative analyses tend to severely limit expressivity.
 - ▶ Declassification is often used to get around.
- ▶ Covert channels like termination leaks are hard to check.

A bird's eye view of the paper

- ▶ Study information flow in the context of a declassification.
- ▶ Work with a concurrent, higher order language with a store as being the source of information transfer.
- ▶ Define what is meant by a safe information flow in presence of declassification.
- ▶ Build a type and effect system to admit only programs safe wrt information flow.
 - ▶ Cut down on some of the conservative analysis from a previous paper. [Boudol et al. *Non-interference for concurrent programs and thread systems*, TCS 281(1), 2002]

Lattice of principals

- ▶ Let \mathcal{P} be the set of principals.
- ▶ The memory locations are each assigned a *security level*.
- ▶ A security level tells which principals are allowed to know the data stored in the location.

$$l \subseteq \mathcal{P}$$

ϕ : Most private; \mathcal{P} : Most public

- ▶ A pair (p, q) specifies a flow of information from p to q .
- ▶ A *flow policy* is a binary relation over \mathcal{P} .

Ordering of security levels

- ▶ A level l' is at least as confidential as another level l if whoever can read a location at l' can also read a location at l .

$$l \preceq_F l' \text{ iff } \forall q \in l'. \exists p \in l. (p, q) \in F^* \quad (1)$$

- ▶ Meet: $l \wedge_F l' = l \cup l'$
- ▶ Join: $l \vee_F l' = \{q \mid \exists p \in l, p' \in l'. pF^*q, p'F^*q\}$

Ordering of security levels

- ▶ A level l' is at least as confidential as another level l if whoever can read a location at l' can also read a location at l .

$$l \preceq_F l' \text{ iff } \forall q \in l'. \exists p \in l. (p, q) \in F^* \quad (1)$$

- ▶ Meet: $l \wedge_F l' = l \cup l'$
- ▶ Join: $l \vee_F l' = \{q \mid \exists p \in l, p' \in l'. pF^*q, p'F^*q\}$
- ▶ Notice that meets and joins are with respect to a given flow policy.
- ▶ The security levels form a prelattice.

Some Notation

- ▶ A process P is a pool of expressions being evaluated.

$$P, Q ::= M \mid (P \parallel Q).$$

- ▶ A configuration is a process being evaluated under a heap:
 (P, μ) .

Typing expressions

- ▶ Expressions return a value, and may possibly write to memory locations.

Typing expressions

- ▶ Expressions return a value, and may possibly write to memory locations.

$$u := M \quad (2)$$

The value returned by M must not be more secret than what u can hold.

- ▶ The *read-effects* of M characterize the security level of its return value.

Typing expressions

- ▶ Expressions return a value, and may possibly write to memory locations.

$$u := M \quad (2)$$

The value returned by M must not be more secret than what u can hold.

- ▶ The *read-effects* of M characterize the security level of its return value.

$$\text{if } M \text{ then } u := N_0 \text{ else } u := N_1 \quad (3)$$

- ▶ Crude Assumption: All information on which the execution of the branches depend, may flow to the locations they write to.
- ▶ The *write effects* provide the lower bound on the security levels of locations being written to.

Typing expressions

- ▶ A new effect other than read and write effects is needed to type termination leaks.

Typing expressions

- ▶ A new effect other than read and write effects is needed to type termination leaks.

$$\text{while } !u_H \text{ do } (); v := ff \quad (4)$$

- ▶ The while loop does not write to any location. But...

Typing expressions

- ▶ A new effect other than read and write effects is needed to type termination leaks.

$$\text{while } !u_H \text{ do } (); v := ff \quad (4)$$

- ▶ The while loop does not write to any location. But...
- ▶ The termination of while loop carries information about the high location u_H .

Typing expressions

- ▶ A new effect other than read and write effects is needed to type termination leaks.

$$\text{while } !u_H \text{ do } (); v := ff \quad (4)$$

- ▶ The while loop does not write to any location. But...
- ▶ The termination of while loop carries information about the high location u_H .

The 'termination effect' of $\text{while } !u_H \text{ do } ()$ is H .

Typing expressions

- ▶ A new effect other than read and write effects is needed to type termination leaks.

$$\text{while } !u_H \text{ do } (); v := ff \quad (4)$$

- ▶ The while loop does not write to any location. But...
- ▶ The termination of while loop carries information about the high location u_H .

The 'termination effect' of $\text{while } !u_H \text{ do } ()$ is H .

- ▶ So v should be at a higher security level than H .

Typing judgment

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

where

- ▶ G is the current flow policy.
- ▶ $s = (s.c, s.w, s.t)$ is the security effect.

Typing judgment

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

where

- ▶ G is the current flow policy.
- ▶ $s = (s.c, s.w, s.t)$ is the security effect.
- ▶ Effects can be ordered covariantly on read and termination effects, and contravariantly on write effects.

Typing judgment

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

where

- ▶ G is the current flow policy.
- ▶ $s = (s.c, s.w, s.t)$ is the security effect.
- ▶ Effects can be ordered covariantly on read and termination effects, and contravariantly on write effects.
 - ▶ $s \Upsilon_F s' = (s.c \Upsilon_F s'.c, s.w \wedge_G s'.w, s.t \Upsilon_F s'.t)$
- ▶ Notation: $\perp = (\perp, \top, \perp)$, $\top = (\top, \perp, \top)$

Typing functions

$$\tau \rightarrow_F^s \sigma$$

- ▶ s records the *latent* effects: the effects that may happen when the function is applied to an argument.
- ▶ F records the *latent flow relation* which is assumed to hold when the function is applied.

$$\frac{F; \Gamma, x : \tau \vdash M : s, \sigma}{G; \Gamma \vdash \lambda x. M : \tau \rightarrow_F^s \sigma}$$

$$\lambda x. (x := v_H) : \theta \text{ ref}_L \rightarrow_{GU\{(H,L)\}}^{(\perp, L, \perp)} \text{unit}$$

Typing functions

$$\tau \rightarrow_F^s \sigma$$

- ▶ s records the *latent* effects: the effects that may happen when the function is applied to an argument.
- ▶ F records the *latent flow relation* which is assumed to hold when the function is applied.

$$\frac{F; \Gamma, x : \tau \vdash M : s, \sigma}{G; \Gamma \vdash \lambda x. M : \tau \rightarrow_F^s \sigma}$$

$$\lambda x. (x := v_H) \quad \not\vdash \theta \text{ ref}_L \rightarrow_G^{(\perp, L, \perp)} \text{unit}$$

Typing values

The effects of a value are always \perp .

$$\overline{F; \Gamma \vdash u_{l, \theta} : \perp, \theta \text{ ref}_l}$$

$$\overline{F; \Gamma, x : \tau \vdash x : \perp, \tau}$$

$$\overline{F; \Gamma \vdash () : \perp, \text{unit}}$$

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

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

Typing Conditionals

if M then N_0 else N_1

Let $F; \Gamma \vdash M : s, \text{bool}$, $F; \Gamma \vdash N_i : s_i, \tau$.

Typing Conditionals

if M then N_0 else N_1

Let $F; \Gamma \vdash M : s, \text{bool}$, $F; \Gamma \vdash N_i : s_i, \tau$.

Possible information flows:

- ▶ Reads from M into writes of N_0, N_1 :

Typing Conditionals

if M then N_0 else N_1

Let $F; \Gamma \vdash M : s, \text{bool}$, $F; \Gamma \vdash N_i : s_i, \tau$.

Possible information flows:

- ▶ Reads from M into writes of N_0, N_1 : $s.c \preceq_F s_0.w \wedge_G s_1.w$.
- ▶ Information about reads in M is not leaked by termination of the if-statement only when both N_0 and N_1 are terminating.

if ! $u_{H, \text{bool}}$ then loop else ()

Typing Conditionals

if M then N_0 else N_1

Let $F; \Gamma \vdash M : s, \text{bool}$, $F; \Gamma \vdash N_i : s_i, \tau$.

Possible information flows:

- ▶ Reads from M into writes of N_0, N_1 : $s.C \preceq_F s_0.W \wedge_G s_1.W$.
- ▶ Information about reads in M is not leaked by termination of the if-statement only when both N_0 and N_1 are terminating.

if ! $u_{H, \text{bool}}$ then loop else ()

$$\frac{\begin{array}{c} F; \Gamma \vdash M : s, \text{bool} \\ F; \Gamma \vdash N_i : s_i, \tau \quad s.C \preceq_F s_0.W \wedge_G s_1.W \end{array}}{F; \Gamma \vdash \text{if } M \text{ then } N_0 \text{ else } N_1 : s \vee s_0 \vee s_1 \vee (\perp, \top, t), \tau}$$

Typing application

$(M N)$

Let $G; \Gamma \vdash M : s, \tau \xrightarrow{F}^{s'} \sigma$, $G; \Gamma \vdash N : s'', \tau$. Possible information flows are:

- ▶ Assume M is evaluated first:

Typing application

(M N)

Let $G; \Gamma \vdash M : s, \tau \xrightarrow{F}^{s'} \sigma$, $G; \Gamma \vdash N : s'', \tau$. Possible information flows are:

- ▶ Assume M is evaluated first: $s.t \preceq_G s''.w$

Typing application

$(M N)$

Let $G; \Gamma \vdash M : s, \tau \xrightarrow{F}^{s'} \sigma$, $G; \Gamma \vdash N : s'', \tau$. Possible information flows are:

- ▶ Assume M is evaluated first: $s.t \preceq_G s''.w$
- ▶ Reads from M and N may flow into body of function:

Typing application

$(M N)$

Let $G; \Gamma \vdash M : s, \tau \xrightarrow{F}^{s'} \sigma$, $G; \Gamma \vdash N : s'', \tau$. Possible information flows are:

- ▶ Assume M is evaluated first: $s.t \preceq_G s''.w$
- ▶ Reads from M and N may flow into body of function:
 $s.c \curlywedge s''.c \preceq_G s'.w$.

Typing application

(M N)

Let $G; \Gamma \vdash M : s, \tau \rightarrow_F^{s'} \sigma$, $G; \Gamma \vdash N : s'', \tau$. Possible information flows are:

- ▶ Assume M is evaluated first: $s.t \preceq_G s''.w$
- ▶ Reads from M and N may flow into body of function:
 $s.c \Upsilon s''.c \preceq_G s'.w$.
- ▶ Termination of body of function might depend on reads from M and N .

$$\frac{G; \Gamma \vdash M : s, \tau \rightarrow_F^{s'} \sigma \quad G; \Gamma \vdash N : s'', \tau \quad s.t \preceq_G s''.w \quad s.c \Upsilon s''.c \preceq_G s'.w}{G, F; \Gamma \vdash (MN) : s \Upsilon s' \Upsilon s'' \Upsilon (\perp, \top, s.c \Upsilon s''.c), \sigma}$$

Typing flow declarations

flow F in M

Let $F, G; \Gamma \vdash M : s, \tau$.

Typing flow declarations

flow F in M

Let $F, G; \Gamma \vdash M : s, \tau$. Subsumption can be applied to the read and termination effects wrt. the new policy $F \cup G$.

$$\frac{F, G; \Gamma \vdash M : s, \tau \quad s.c \preceq_{G \cup F} c \quad s.t \preceq_{G \cup F} t}{G; \Gamma \vdash \text{flow } F \text{ in } M : (c, s.w, t), \tau}$$

Typing flow declarations

flow F in M

Let $F, G; \Gamma \vdash M : s, \tau$. Subsumption can be applied to the read and termination effects wrt. the new policy $F \cup G$.

$$\frac{F, G; \Gamma \vdash M : s, \tau \quad s.c \preceq_{G \cup F} c \quad s.t \preceq_{G \cup F} t}{G; \Gamma \vdash \text{flow } F \text{ in } M : (c, s.w, t), \tau}$$

Notice that there is no subsumption for write effects: $w \not\preceq_{G \cup F} s.w$!

Typing Sequential Computations

$M;N$

Let $G; \Gamma \vdash M : s, \tau$, $G; \Gamma \vdash N : s', \tau'$. Possible information flows:

- ▶ Only termination from M can be captured by N :

Typing Sequential Computations

$M;N$

Let $G; \Gamma \vdash M : s, \tau$, $G; \Gamma \vdash N : s', \tau'$. Possible information flows:

- ▶ Only termination from M can be captured by N : $s.t \preceq s'.w$.
- ▶ No information read in M directly flows into the result, except what flows through writes and termination.

$$\frac{G; \Gamma \vdash M : s, \tau \quad G; \Gamma \vdash N : s', \tau' \quad s.t \preceq s'.w}{G; \Gamma \vdash M; N : (\perp, s.w, s.t) \Upsilon s', \tau'}$$

Typing reads and writes

$$\frac{F; \Gamma \vdash M : s, \theta \quad s.r \preceq l}{F; \Gamma \vdash \text{ref}_{l, \theta} M : (\perp, s.w, s.t), \theta \text{ ref}_l}$$

$$\frac{F; \Gamma \vdash M : s, \theta \text{ ref}_l}{F; \Gamma \vdash !M : s \Upsilon (l, \top, \perp), \theta}$$

Typing threads

- ▶ Reads inside the thread cannot escape except via writes.
- ▶ Termination of a thread cannot be tracked.

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

The non-interference property

- ▶ Classic non-interference property: A program does not leak information from 'High' locations into 'Low' locations if the high locations do not affect the low locations.

The non-interference property

- ▶ Classic non-interference property: A program does not leak information from 'High' locations into 'Low' locations if the high locations do not affect the low locations.
- ▶ **How to modify this notion in presence of declassification?**

The non-interference property

- ▶ Classic non-interference property: A program does not leak information from 'High' locations into 'Low' locations if the high locations do not affect the low locations.
- ▶ **How to modify this notion in presence of declassification?**
- ▶ A process is safe if at each step of execution, it satisfies the non-interference condition (wrt the then policy).
- ▶ The 'then' policy of a process P during an execution step is denoted as $(P, \mu) \rightarrow_F (P', \mu')$.

Effects of local declassifications

`flow {(H, M)} in $u_M := v_H$`

- ▶ Starting from two heaps identical in L part, the result is identical in L parts.

Effects of local declassifications

$\text{flow } \{(H, M)\} \text{ in } u_M := v_H$

- ▶ Starting from two heaps identical in L part, the result is identical in L parts. Starting from two heaps identical in M parts, the result is not identical in M parts.

Effects of local declassifications

$\text{flow } \{(H, M)\} \text{ in } u_M := v_H$

- ▶ Starting from two heaps identical in L part, the result is identical in L parts. Starting from two heaps identical in M parts, the result is not identical in M parts.
- ▶ The current policy affects how heaps are considered identical. Thus $(L = 1, M = 1, H = 2)$ and $(L = 1, M = 1, H = 3)$ are not M -identical.

Effects of local declassifications

flow $\{(H, M)\}$ in $u_M := v_H$

- ▶ Starting from two heaps identical in L part, the result is identical in L parts. Starting from two heaps identical in M parts, the result is not identical in M parts.
- ▶ The current policy affects how heaps are considered identical. Thus $(L = 1, M = 1, H = 2)$ and $(L = 1, M = 1, H = 3)$ are not M -identical.

$$\mu \simeq^{F,I} \nu \text{ iff } \forall u, \theta \in \mu \cap \nu. I' \preceq_F I \implies \mu(u) = \nu(u)$$

Towards a non-interference definition

- ▶ A process may be thought of as being “safe”, if, operating under policy $F \cup G$, for any level l , starting from two $(F \cup G)$, l -identical heaps, the resulting heaps are also $(F \cup G)$, l -identical.

Towards a non-interference definition

- ▶ A process may be thought of as being “safe”, if, operating under policy $F \cup G$, for any level l , starting from two $(F \cup G)$, l -identical heaps, the resulting heaps are also $(F \cup G)$, l -identical.
- ▶ Notice that if $\mu \simeq^{(F \cup G), l} \nu$, then $\mu \simeq^{G, l} \nu$.

Towards a non-interference definition

- ▶ A process may be thought of as being “safe”, if, operating under policy $F \cup G$, for any level l , starting from two $(F \cup G)$, l -identical heaps, the resulting heaps are also $(F \cup G)$, l -identical.
- ▶ Notice that if $\mu \simeq^{(F \cup G), l} \nu$, then $\mu \simeq^{G, l} \nu$.
- ▶ In other words, a process P satisfies non-disclosure if for any level l , and $(F \cup G)$, l -identical heaps μ, ν , whenever

$$(P, \mu) \rightarrow_{(F \cup G)} (P', \mu')$$

then there exists a (P'', ν') , s.t.

$$(P, \nu) \xrightarrow{*}_{(F \cup G)} (P'', \nu')$$

,
and $\mu' \simeq^{(F \cup G), l} \nu'$, and P' and P'' simulate each other likewise.

Theorem

Soundness of type system:

A well-typed program in the context of a global flow policy G satisfies non-disclosure with respect to G .

Comments

- ▶ Why do we need to relax the definition of non-disclosure to require the resultant heaps to be identical wrt just G .
- ▶ Why is concurrency needed? Is there more to information flow in a concurrent setting than in a sequential setting?
- ▶ Which is a better way to specify reclassification policies:
 $p \preceq q$ or $I \preceq I'$?