Enforcing information flow by combining static and dynamic analysis

Andrew Bedford, Josée Desharnais, Théophane G. Godonou, Nadia Tawbi

Université Laval, Québec, Canada

Université Laval 2014

Plan

Context

- Enforcing security mechanisms must control how information flows through the system.
- No information can flow from a private source to a public destination.
- Static mechanisms, usually type-based, are conservative, they reject programs in case of doubt.
- Our approach is type-based data-flow sensitive
    - The program is safe: all the executions satisfy the information flow policy
    - The program may be unsafe: instrumentation is needed.
    - The program is unsafe: rejection.

The non-interference property

- *Non-interference* essentially means that a variation of program input associated with a given security level does not cause variation of output of lower security level.
- We have to forbid explicit flow and implicit flow.

The core language

$$
\begin{array}{lll}
\textit{(instructions)} & p ::= & e \mid c \\
\textit{(expressions)} & e ::= & x \mid n \mid nch \mid e_1 \ \mathbf{op} \ e_2 \\
\textit{(commands)} & c ::= & x := e \mid \\
 & & \mathbf{skip} \mid \\
 & & \mathbf{if} \ e \ \mathbf{then} \ c_1 \ \mathbf{else} \ c_2 \ \mathbf{end} \mid \\
 & & \mathbf{while} \ e \ \mathbf{do} \ c \ \mathbf{end} \mid \\
 & & c_1 ; c_2 \mid \\
 & & \mathbf{receive_c} \ x_1 \ \mathbf{from} \ x_2 \mid \\
 & & \mathbf{receive_n} \ x_1 \ \mathbf{from} \ x_2 \mid \\
 & & \mathbf{send} \ x_1 \ \mathbf{to} \ x_2
\end{array}
$$

The language

- Programs are sequential.
- Programs communicate via channels.
- An external observer can see only the channels not the variables.
- The channels are assigned a priori security levels, types.
- The variables security levels depend on the security level of their content.

Plan

Some dynamic rules I

(ASSIGN)	$\dfrac{\langle e, \mu\rangle \rightarrow_e v}{\langle \mathbf{x := e}, \mu\rangle \rightarrow \mu[x \mapsto v]}$
(RECEIVE-CONTENT)	$\dfrac{x_2 \in \mathrm{dom}(\mu) \qquad read(\mu(x_2)) = n}{\langle \mathbf{receive_c}\ x_1\ \mathbf{from}\ x_2, \mu\rangle \rightarrow \mu[x_1 \mapsto n]}$
(RECEIVE-NAME)	$\dfrac{x_2 \in \mathrm{dom}(\mu) \qquad read(\mu(x_2)) = nch}{\langle \mathbf{receive_n}\ x_1\ \mathbf{from}\ x_2, \mu\rangle \rightarrow \mu[x_1 \mapsto nch]}$
(SEND)	$\dfrac{x_1 \in \mathrm{dom}(\mu)}{\langle \mathbf{send}\ x_1\ \mathbf{to}\ x_2, \mu\rangle \rightarrow \mu, update(\mu(x_2), \mu(x_1))}$

Table : A few rules of the structural operational semantics

Some dynamic rules II

(CONDITIONAL)	$$\frac{\langle e, \mu\rangle \to_e n \qquad n \neq 0}{\langle \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ end}, \mu\rangle \to \langle c_1, \mu\rangle}$$
	$$\frac{\langle e, \mu\rangle \to_e n \qquad n = 0}{\langle \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ end}, \mu\rangle \to \langle c_2, \mu\rangle}$$
(LOOP)	$$\frac{\langle e, \mu\rangle \to_e n \qquad n = 0}{\langle \textbf{while } e \textbf{ do } c \textbf{ end}, \mu\rangle \to \mu}$$
	$$\frac{\langle e, \mu\rangle \to_e n \qquad n \neq 0}{\langle \textbf{while } e \textbf{ do } c \textbf{ end}, \mu\rangle \to \langle c; \textbf{while } e \textbf{ then } c \textbf{ end}, \mu\rangle}$$
(SEQUENCE)	$$\frac{\langle c_1, \mu\rangle \to \mu'}{\langle c_1; c_2, \mu\rangle \to \langle c_2, \mu'\rangle}$$

Table : Structural operational semantics

Explicit or implicit flow

Reject programs leading to explicit or implicit flow.

```
1.   x := highValue;
2.   send x to publicChannel
```

```
1.   if highValue then
2.     x := 1
3.   else
       x := 2
4.   end;
5.   send x to publicChannel
```

Flow sensitivity

Take into account data flows.

1. $\textbf{receive}_\mathbf{c}\ x\ \textbf{from}\ \texttt{privateChannel};$
2. $\textbf{receive}_\mathbf{n}\ c\ \textbf{from}\ \texttt{publicChannel};$
3. $\textbf{receive}_\mathbf{c}\ x\ \textbf{from}\ c;$
4. $x := 0;$
5. $\textbf{send}\ x\ \textbf{to}\ \texttt{publicChannel};$

Unknown flow

Do not reject programs leading to unknown flows.

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;
2. **receive**$_{\mathbf{n}}$ c **from** `publicChannel`;
3. **send** x **to** c

Blocked channel

Reject programs leading to more subtle implicit flow.

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;
2. **if** $\mathtt{x} > 0$ **then**
3. c:= `publicChannel1`;
   else
4. c:= `publicChannel2`
   end;
5. **send** *lowValue* **to** c

Plan

The typing principles

- The security types are defined as follows:

$$\begin{array}{ll} \textit{(data types)} & \tau ::= \; L \mid U \mid H \mid B \\ \textit{(instruction types)} & \rho ::= \tau \; val \mid \tau \; chan \mid \tau \; cmd \end{array}$$

  $L \sqsubseteq U \sqsubseteq H \sqsubseteq B$, $B \not\sqsubseteq H \not\sqsubseteq U \not\sqsubseteq L$
- U is assigned when the type cannot be determined statically.
- B is used to block a channel without rejecting the program.
- A special variable *_instr* indicates the need of instrumentation or not.
- Channels are assigned a priori security types namely H or L.
- Variables security types depend on the security type of their content.

The non-interference property revisited

Definition

(Non-interference) A program P satisfies non-interference if, for any memories μ and ν that are τ-equivalent and that agree on values of type $\tau' \sqsubseteq \tau$, the memories μ' and ν' produced by running P on μ and ν are also τ-equivalent (provided that both runs terminate successfully).

Definition

(τ-equivalence) Two memories μ and ν are τ-equivalent, written $\mu \sim_\tau \nu$, if $\forall x \in dom(\mu) \cap dom(\nu) : (x$ is a channel of type $\tau' chan \wedge \tau' \sqsubseteq \tau) \Rightarrow \mu(x) =_{ch} \nu(x)$.

where τ is a security level.

Some typing rules

(ASSIGN -CHAN_S)	$\dfrac{\Gamma, pc \vdash e : \tau\ chan}{\Gamma, pc \vdash x := e : \tau\ cmd, \Gamma \sqcup [\_instr \mapsto HL^L_L(pc,\tau)] \dagger [x \mapsto HL^B_\tau(pc,\tau) chan]}$
(RECEIVE-CONTENT_S)	$\dfrac{\Gamma(x_2) = \tau\ chan}{\Gamma, pc \vdash \mathbf{receive_c}\ x_1\ \mathbf{from}\ x_2 : (\tau \sqcup pc)\ cmd, \Gamma \dagger [x_1 \mapsto (\tau \sqcup pc) val]}$
(RECEIVE-NAME_S)	$\dfrac{\Gamma(x_2) = \tau\ chan}{\Gamma, pc \vdash \mathbf{receive_n}\ x_1\ \mathbf{from}\ x_2 : \tau\ cmd, \Gamma \sqcup [\_instr \mapsto HL^L_\tau(pc,\tau)] \dagger [x_1 \mapsto HL^B_{U \sqcup \tau}(pc,\tau) chan]}$
(SEND_S)	$\dfrac{\Gamma(x_1) = \tau_1\ \alpha \quad \Gamma(x_2) = \tau\ chan \quad \neg((\tau_1 \sqcup pc) = H \wedge \tau = L) \quad \tau \neq B}{\Gamma, pc \vdash \mathbf{send}\ x_1\ \mathbf{to}\ x_2 : \tau\ cmd, \Gamma \sqcup [\_instr \mapsto HL^U_L(\tau_1 \sqcup pc, \tau)]}$

Table : Typing rules 1

Some typing rules II

(CONDITIONAL_S)
$$\frac{\Gamma, pc \vdash e : \tau_0\, val \qquad \begin{array}{c}\Gamma, (pc \sqcup \tau_0) \vdash c_1 : \tau_1\; cmd, \Gamma' \\ \Gamma, (pc \sqcup \tau_0) \vdash c_2 : \tau_2\; cmd, \Gamma''\end{array} \qquad \Gamma' \sqcup \Gamma'' \sqsupset \bot}{\Gamma, pc \vdash \textbf{ if } e \textbf{ then } c_1 \textbf{ else } c_2 \textbf{ end} : (\tau_1 \sqcap \tau_2)\; cmd, \Gamma' \sqcup \Gamma''}$$

(LOOP1_S)
$$\frac{\Gamma, pc \vdash e : \tau_0\, val \qquad \Gamma, (pc \sqcup \tau_0) \vdash c : \tau\;\; cmd, \Gamma' \qquad \Gamma = \Gamma \sqcup \Gamma' \sqsupset \bot}{\Gamma, pc \vdash \textbf{while } e \textbf{ do } c \textbf{ end} : \tau\;\; cmd, \Gamma \sqcup \Gamma'}$$

(LOOP2_S)
$$\frac{\begin{array}{c}\Gamma, (pc \sqcup \tau_0) \vdash c : \tau\;\; cmd, \Gamma' \quad \Gamma \neq \Gamma \sqcup \Gamma' \sqsupset \bot \\ \Gamma, pc \vdash e : \tau_0\, val \quad \Gamma \sqcup \Gamma', (pc \sqcup \tau_0) \vdash \textbf{while } e \textbf{ do } c \textbf{ end} : \tau'\; cmd, \Gamma''\end{array}}{\Gamma, pc \vdash \textbf{while } e \textbf{ do } c \textbf{ end} : \tau'\; cmd,\;\; \Gamma''}$$

(SEQUENCE_S)
$$\frac{\Gamma, pc \vdash c_1 : \tau_1\; cmd, \Gamma' \qquad \Gamma', pc \vdash c_2 : \tau_2\; cmd, \Gamma''}{\Gamma, pc \vdash c_1; c_2 : (\tau_1 \sqcap \tau_2)\; cmd, \Gamma''}$$

Table : Typing rules2

Explicit flow

1. $x :=$ `highValue`;
2. **send** x **to** `publicChannel`

Explicit flow

1. $x :=$ `highValue`;
   $[x \mapsto H\ val]$
2. **send** x **to** `publicChannel`

Explicit flow

1. $x :=$ `highValue`;
   $[x \mapsto H\ val]$
2. **send** x **to** `publicChannel`
   Reject

Implicit flow

```
1.  if highValue then
2.    x := 1
3.  else
      x := 2
4.  end;

5.  send x to publicChannel
```

Implicit flow

1. **if** `highValue` **then**
2. $x := 1$
3. **else**
   $x := 2$
4. **end**;
   $[x \mapsto H\ val]$
5. **send** x **to** `publicChannel`

Implicit flow

1. **if** `highValue` **then**
2. $x := 1$
3. **else**
   $x := 2$
4. **end**;
   $[x \mapsto H\ val]$
5. **send** x **to** `publicChannel`
   Reject

Flow sensitivity

1. $\textbf{receive}_{\textbf{c}}\ x$ **from** `privateChannel`;
2. $\textbf{receive}_{\textbf{n}}\ c$ **from** `publicChannel`;
3. $\textbf{receive}_{\textbf{c}}\ x$ **from** c;
4. $x := 0$;
5. **send** x **to** `publicChannel`;

Flow sensitivity

1. $\textbf{receive}_\textbf{c}\ x\ \textbf{from}\ \texttt{privateChannel};$
   $[x \mapsto H\ val]$
2. $\textbf{receive}_\textbf{n}\ c\ \textbf{from}\ \texttt{publicChannel};$
3. $\textbf{receive}_\textbf{c}\ x\ \textbf{from}\ c;$
4. $x := 0;$
5. $\textbf{send}\ x\ \textbf{to}\ \texttt{publicChannel};$

Flow sensitivity

1. $\mathbf{receive_c}\ x\ \mathbf{from}\ \mathtt{privateChannel};$
   $[x \mapsto H\ val]$
2. $\mathbf{receive_n}\ c\ \mathbf{from}\ \mathtt{publicChannel};$
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. $\mathbf{receive_c}\ x\ \mathbf{from}\ c;$
4. $x := 0;$
5. $\mathbf{send}\ x\ \mathbf{to}\ \mathtt{publicChannel};$

Flow sensitivity

1. **receive$_{\mathbf{c}}$** x **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **receive$_{\mathbf{n}}$** c **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. **receive$_{\mathbf{c}}$** x **from** c;
   $[c \mapsto U\ chan, x \mapsto U\ val]$
4. $x := 0$;

5. **send** x **to** `publicChannel`;

Flow sensitivity

1. $\textbf{receive}_{\mathbf{c}}\ x$ **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. $\textbf{receive}_{\mathbf{n}}\ c$ **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. $\textbf{receive}_{\mathbf{c}}\ x$ **from** c;
   $[c \mapsto U\ chan, x \mapsto U\ val]$
4. $x := 0$;
   $[x \mapsto L\ val]$
5. **send** x **to** `publicChannel`;

Flow sensitivity

1. $\textbf{receive}_{\textbf{c}}\ x$ **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. $\textbf{receive}_{\textbf{n}}\ c$ **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. $\textbf{receive}_{\textbf{c}}\ x$ **from** c;
   $[c \mapsto U\ chan, x \mapsto U\ val]$
4. $x := 0$;
   $[x \mapsto L\ val]$
5. **send** x **to** `publicChannel`;
   Accept

Flow sensitivity

```
receivec h from privateChannel;
e := 0;
x1 := 0;
x2 := 0;
x3 := 0;
while e < 5 do

  send x3 to publicChannel;
  x3 := x2;
  x2 := x1;
  x1 := h;
  e := e+1
end
```

Flow sensitivity

```
receivec h from privateChannel;
e := 0;
x1 := 0;
x2 := 0;
x3 := 0;
while e < 5 do
```
$[h \mapsto H\ val, e \mapsto\ L\ val, x1 \mapsto H\ val, x2 \mapsto H\ val, x3 \mapsto H\ val]$
```
  send x3 to publicChannel;
  x3 := x2;
  x2 := x1;
  x1 := h;
  e := e+1
end
```

Flow sensitivity

```
receivec h from privateChannel;
e := 0;
x1 := 0;
x2 := 0;
x3 := 0;
while e < 5 do
```
$[h \mapsto H\ val, e \mapsto\ L\ val, x1 \mapsto H\ val, x2 \mapsto H\ val, x3 \mapsto H\ val]$
```
  send x3 to publicChannel;
  x3 := x2;
  x2 := x1;
  x1 := h;
  e := e+1
end
```
Reject

Unknown flow

1. $\textbf{receive}_{\textbf{c}}\ x$ **from** `privateChannel`;
2. $\textbf{receive}_{\textbf{n}}\ c$ **from** `publicChannel`;
3. **send** x **to** c

Unknown flow

1. $\textbf{receive}_{\textbf{c}}\ x\ \textbf{from}\ \texttt{privateChannel};$
   $[x \mapsto H\ val]$
2. $\textbf{receive}_{\textbf{n}}\ c\ \textbf{from}\ \texttt{publicChannel};$
3. $\textbf{send}\ x\ \textbf{to}\ c$

Unknown flow

1. $\textbf{receive}_{\textbf{c}}\ x$ **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. $\textbf{receive}_{\textbf{n}}\ c$ **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. **send** x **to** c

Unknown flow

1. $\textbf{receive}_{\textbf{c}}\ x$ **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. $\textbf{receive}_{\textbf{n}}\ c$ **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. **send** x **to** c
   Tag for need of instrumentation

Blocked channel

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;

2. **if** `x` $>$ `0` **then**
3. c:= `publicChannel1`;

else
4. c:= `publicChannel2`

end;

5. **send** *lowValue* **to** c

Blocked channel

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **if** $\mathtt{x} > \mathtt{0}$ **then**
3. $c := \mathtt{publicChannel1}$;

   else

4. $c := \mathtt{publicChannel2}$

   end;

5. **send** *lowValue* **to** c

Blocked channel

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **if** $\mathtt{x} > 0$ **then**
3. c:= `publicChannel1`;
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   else
4. c:= `publicChannel2`

   end;

5. **send** *lowValue* **to** c

Blocked channel

1. $\textbf{receive}_{\mathbf{c}}\ x\ \textbf{from}\ \texttt{privateChannel};$
   $[x \mapsto H\ val]$
2. $\textbf{if}\ \texttt{x} > 0\ \textbf{then}$
3. $c := \texttt{publicChannel1};$
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   $\textbf{else}$
4. $c := \texttt{publicChannel2}$
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   $\textbf{end};$

5. $\textbf{send}\ lowValue\ \textbf{to}\ c$

Blocked channel

1. $\textbf{receive}_\textbf{c}\ x$ **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **if** $\mathtt{x} > 0$ **then**
3. $c := \mathtt{publicChannel1}$;
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   else
4. $c := \mathtt{publicChannel2}$
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   end;
   $[x \mapsto H\ val, c \mapsto B\ chan]$
5. **send** $lowValue$ **to** c

Blocked channel

1. **receive**$_c$ x **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **if** `x` $>$ `0` **then**
3. $c :=$ `publicChannel1`;
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   else
4. $c :=$ `publicChannel2`
   $pc \mapsto H, [x \mapsto H\ val, c \mapsto B\ chan]$
   end;
   $[x \mapsto H\ val, c \mapsto B\ chan]$
5. **send** *lowValue* **to** c
   Reject

Inference algorithm refines the type system

$\texttt{Infer}(g_e, i, pc, c) =$

case c of

$\vdots$

receive$_n$ x_1 **from** x_2 :

$\tau = \texttt{evalT}(g_e(x_2))$

$\_instr_t = \texttt{HL}(L, \tau, pc, \tau)$

$\_instr_{t2} = g_e(\_instr)$

$x1_t = \texttt{HL}(B, \sup(U, \tau), pc, \tau)$

$G(i) = \texttt{updateEnv}(\texttt{updateEnv}(g_e, \_instr, \sup(\_instr_t, \_instr_{t2})), x_1, x1_t \;\; chan)$

return $(G(i), i + 1)$

send x_1 **to** x_2 :

$\tau_1 = \texttt{evalT}(g_e(x_1))$

$\tau = \texttt{evalT}(g_e(x_2))$

$\_instr_t = \texttt{HL}(U, L, \sup(\tau_1, pc), \tau)$

$\_instr_{t2} = g_e(\_instr)$

if $((\tau \neq B)$ *and* $\neg(\sup(\tau_1, pc) = H$ *and* $\tau = L))$

then $G(i) = \texttt{updateEnv}(g_e, \_instr, \sup(\_instr_t, \_instr_{t2}))$

else **fail**

return $(G(i), i + 1)$

$\vdots$

Plan

Instrumentation algorithm

```
Instrument: cmd * int → int
Instrument(c, i) =     case c of
⋮
    receive_n x1 from  x2 :
       IC = IC∧ "receive_n x1 from x2 ";
       if  (G(i)(x2) != L chan)
        then  IC = IC∧ " if TypeOf_Channel(x1) = L chan and TypeOf_Channel(x2) = H chan
            then updateEnv(G(i), x1, B chan)
            else updateEnv(G(i), x1, TypeOf_Channel(x1))
            end "
         else IC = IC∧ "updateEnv(G(i), x1, TypeOf_Channel(x1))"
         end
       IC = IC∧ " updateEnv(g_M, x1, G(i)(x1)); "
    return (i + 1)
    send x1 to x2 :
          IC = IC∧ " tau = TypeOf_Expression(x2) ; tau1 = TypeOf_Expression(x1);
            if(((tau = L chan) and (sup(evalT(tau1), top(pc)) = H)) or (tau = B chan))
            then fail else send x1 to x2 end; "
    return (i + 1)
.
```

Example: unknown flow

1. **receive**$_{\mathbf{c}}$ x **from** `privateChannel`;
   $[x \mapsto H\ val]$
2. **receive**$_{\mathbf{n}}$ c **from** `publicChannel`;
   $[x \mapsto H\ val, c \mapsto U\ chan]$
3. **send** x **to** c
   Tag for need of instrumentation

Instrumented example

```
    push(L, pc);
1.  receive_c x from privateChannel;
    updateEnv(g_M, x, G(1)(x));
2.  receive_n c from publicChannel;
     updateEnv(G(2), c, TypeOf_Channel(c));
     updateEnv(g_M, c, G(2)(c));
     tau = TypeOf_Expression(c);
     tau1 = TypeOf_Expression(x);
     if ((tau = L chan) and sup(evalT(tau1), top(pc)) = H) or
     (tau = B chan) then fail; else
5.     send x to c
```

Plan

Conclusion

Contribution

- The definition of a sound type system that captures lack of information in a program at compile-time.
- A multi-valued type analysis:
  - The program is safe: all the executions satisfy the information flow policy
  - The program may be unsafe: instrumentation is needed.
  - The program is unsafe: rejection

  leading to fewer false positives and lighter overhead

Future Work

- Dynamic code.
- Non-termination and concurrency.
- Leverage to a real language.

Thanks

Questions?