Information Flow in a Simple Imperative Language

The Problem

- System with High and Low inputs, L $\leq$ H.
  H $\equiv$ secret/private/classified
- L users permitted to see L outputs.

Security policy: Confidentiality $\equiv$ “PROTECT SECRETS”, *i.e.*, L-outputs should not depend on H-inputs.

Dual policy: Integrity, *i.e.*, Licensed data is not influenced by Hacked data. No Hacked data should be used at a Licensed sink.

Formalize for programs written in a simple imperative languages.

- Noninterference (NI) [Goguen-Meseguer ’82]

“No matter how H inputs change, L outputs remain same”.

Examples

```
h := 42; l := 42;
l := h;

l := h; l := l - h;
h := l; l := h;

h := h mod 2; // High variable set
l  := 0; // Low variable set
if h = 1 then l := 1 // Implicit flow from high to low
    else skip

while h > 0 do
  l := l + 1;
  h := h - 1;
```

Syntax and typing rules

Syntax:

$$
\begin{aligned}
T &\coloneqq \textbf{int} \\
e &\coloneqq x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \\
S &\coloneqq x := e \mid \textsf{if } e \textsf{ then } S_1 \textsf{ else } S_2 \mid S_1;\ S_2
\end{aligned}
$$

Typing rules for expressions: $\Gamma \vdash e : T$

$$
\Gamma \vdash x : \Gamma\, x \qquad \Gamma \vdash n : \textbf{int} \qquad \frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 + e_2 : \textbf{int}}
$$

$$
\frac{\Gamma \vdash e_1 : \textbf{int} \qquad \Gamma \vdash e_2 : \textbf{int}}{\Gamma \vdash e_1 - e_2 : \textbf{int}}
$$

Typing rules for commands: $\Gamma \vdash S$

$$\frac{\Gamma, x : \mathsf{T} \vdash e : \mathsf{T}}{\Gamma, x : \mathsf{T} \vdash x := e} \qquad \frac{\Gamma \vdash e : \textbf{int} \qquad \Gamma \vdash S_1 \qquad \Gamma \vdash S_2}{\Gamma \vdash \textsf{if } e \textsf{ then } S_1 \textsf{ else } S_2}$$

$$\frac{\Gamma \vdash S_1 \qquad \Gamma \vdash S_2}{\Gamma \vdash S_1;\ S_2}$$

Semantics

$$\begin{array}{lcl} [\![\mathbf{int}]\!] & = & \mathit{Z} \\ [\![\Gamma]\!] & = & \{\eta \mid \mathit{dom}\ \eta = \mathit{dom}\ \Gamma \wedge \forall x \in \mathit{dom}\ \eta \bullet \eta\, x \in [\![\Gamma\, x]\!]\} \end{array}$$

Semantics of expressions: The meaning of an expression $\Gamma \vdash e : T$ is a function $[\![\Gamma]\!] \to [\![T]\!]$.

$$\begin{array}{lcl} [\![\Gamma \vdash x : T]\!]\eta & = & \eta x \\ [\![\Gamma \vdash n : \mathbf{int}]\!]\eta & = & n \\ [\![\Gamma \vdash e_1 + e_2 : \mathbf{int}]\!]\eta & = & \text{let } d_1 = [\![\Gamma \vdash e_1 : \mathbf{int}]\!]\eta \text{ in} \\ & & \text{let } d_2 = [\![\Gamma \vdash e_2 : \mathbf{int}]\!]\eta \text{ in } d_1 + d_2 \end{array}$$

Similary for $e_1 - e_2$.

Semantics of commands: The meaning of a command $\Gamma \vdash S$ is a function $[\![\Gamma]\!] \to [\![\Gamma]\!]$ that takes a store η, and returns a possibly updated store.

$$[\![\Gamma \vdash x := e]\!]\eta = \text{let } d = [\![\Gamma \vdash e : \mathsf{T}]\!]\eta \text{ in } [\eta \mid x \mapsto d]$$

$$\begin{aligned}
[\![\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2]\!]\eta \\
&= \text{let } b = [\![\Gamma \vdash e : \textbf{int}]\!]\eta \text{ in} \\
&\quad \text{if } b > 0 \text{ then } [\![\Gamma \vdash S_1]\!]\eta \text{ else } [\![\Gamma \vdash S_2]\!]\eta
\end{aligned}$$

$$[\![\Gamma \vdash S_1;\ S_2]\!]\eta = \text{let } \eta_1 = [\![\Gamma \vdash S_1]\!]\eta \text{ in } [\![\Gamma \vdash S_2]\!]\eta_1$$

What does being secure mean?

Suppose $\Gamma \vdash e : \mathrm{T}$ and suppose $\Gamma \vdash S$. Under what conditions are e, S secure? First, partition variables into H-variables and L-variables. Then:

- The value of a “low security” expression should not depend on H-variables. This is called *read confinement*.
- A “high security” command (*i.e.*, one that depends on the results of a “high expression”) should not assign to L-variables. This is called *write confinement*.

Security literature: “no read up” or “simple security” and “no write down” or “*-property”.

Formalization coming up ...

Checking information flow using security types.

- Label variables by security types, for example replace $x : T$ by $x : (T, \kappa)$ where κ is the security level.
- Syntax-directed typing rules specify conditions that ensure secure flow.
- Overt flows, like an assignment of an H-variable to an L-variable, are disallowed by the typing rule for assignment.
- Covert flows due to control flow are precluded via the typing rule for conditional.
- Technical machinery: Commands are given types $\mathtt{com}\ \kappa$ with the meaning that all assigned variables have at least level κ.

Security type system: Rules for expressions

General form: $\Delta \vdash e : (T, \kappa)$. Note: $L \leq H$.

$$\Delta \vdash x : \Delta\, x \qquad\qquad \Delta \vdash n : (\textbf{int}, \kappa)$$

$$\frac{\Delta \vdash e_1 : (\textbf{int}, \kappa) \qquad \Delta \vdash e_2 : (\textbf{int}, \kappa)}{\Delta \vdash e_1 + e_2 : (\textbf{int}, \kappa)}$$

$$\frac{\Delta \vdash e : (T, \kappa) \qquad \kappa \leq \kappa'}{\Delta \vdash e : (T, \kappa')}$$

Security type system: Rules for commands

General form: $\Delta \vdash S : (\texttt{com}\ \kappa)$.

$$\frac{\Delta, x : (T, \kappa) \vdash e : (T, \kappa)}{\Delta, x : (T, \kappa) \vdash x := e : (\texttt{com}\ \kappa)}$$

$$\frac{\Delta \vdash e : (\textbf{int}, \kappa) \qquad \Delta \vdash S_1 : (\texttt{com}\ \kappa) \qquad \Delta \vdash S_2 : (\texttt{com}\ \kappa)}{\Delta \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 : (\texttt{com}\ \kappa)}$$

$$\frac{\Delta \vdash S_1 : (\texttt{com}\ \kappa) \qquad \Delta \vdash S_2 : (\texttt{com}\ \kappa)}{\Delta \vdash S_1;\ S_2 : (\texttt{com}\ \kappa)}$$

$$\frac{\Delta \vdash S : (\texttt{com}\ \kappa_1) \qquad \kappa \leq \kappa_1}{\Delta \vdash S : (\texttt{com}\ \kappa)}$$

Examples revisited

Let $\Delta = [x : (\textbf{int}, \mathrm{H}), y : (\textbf{int}, \mathrm{L})]$.

```
x := 42 : (com H); y := 42 : (com L)
x := 42; y := 42 : (com L)
y := x    (* untypable *)

y := x; y := y - x;  (* untypable *)
x := y; y := x;  (* untypable *)

x := x mod 2;  (* (com H) *)
y  := 0;  (* (com L) *)
```

if `x = 1` then `y := 1`
   else skip
 `(* untypable: low assignment under high test *)`

```
while x > 0 do  (* x: (int, H) *)
  y := y + 1;  (* (com L) *)
  x := x - 1;  (* (com H) *)
 (* untypable: low assignment under high test *)
```

“Being secure” revisited

Want a notion of being “indistinguishable by L”. Define the following relation:

$$
\begin{aligned}
d \sim_{[\![T]\!]} d' &\iff d = d' \text{ for primitive types } T \\
\eta \sim_{[\![\Delta^\dagger]\!]} \eta' &\iff \forall (x : (T, \kappa)) \in \Delta \bullet \kappa = L \Rightarrow (\eta\, x) \sim_{[\![T]\!]} (\eta'\, x)
\end{aligned}
$$

Thus two stores are indistinguishable if the L-view of the stores are the same. That is, any change in the H-variables are invisible to the L-viewer (“attacker”).

Ultimately want *noninterference*: for any pair of initial stores that are indistinguishable for L, the two corresponding runs of the program yield final stores that are indistinguishable for L.

Safe expressions are read confined

Say that an expression or command is safe if it is typable using the security typing rules.

Lemma (safe expressions are read confined)

Suppose $\Delta \vdash e : (T, L)$ and $\eta \sim_{[\![\Delta^\dagger]\!]} \eta'$. If $d = [\![\Delta^\dagger \vdash e : T]\!]\eta$ and $d' = [\![\Delta^\dagger \vdash e : T]\!]\eta'$ then $d \sim_{[\![T]\!]} d'$.

The lemma says that if an expression can be typed $\Delta \vdash e : (T, L)$ then its meaning is the same in two L-indistinguishable stores.

Proof: The proof is by induction on a derivation of $\Delta \vdash e : (T, L)$ with cases on the last rule used.

All cases are easy, with only a small excitement in the subsumption rule.

Write confinement of commands

Lemma (write confinement of commands)

Suppose $\Delta \vdash S : (\mathtt{com}\ \kappa)$. For all η, if $\eta_0 = [\![\Delta^\dagger \vdash S]\!]\eta$ then

$$\kappa = H \Rightarrow \eta \sim_{[\![\Delta^\dagger]\!]} \eta_0$$

Proof: The proof is by induction on a derivation of $\Delta \vdash S : (\mathtt{com}\ \kappa)$ and by cases on the last rule used in the derivation.

Noninterference

Theorem (safe commands are noninterfering)

Suppose $\Delta \vdash S : (\mathtt{com}\ \kappa)$ and $\eta \sim_{[\![\Delta^\dagger]\!]} \eta'$. Let $\eta_0 = [\![\Delta^\dagger \vdash S]\!]\eta$ and $\eta_0' = [\![\Delta^\dagger \vdash S]\!]\eta'$. Then $\eta_0 \sim_{[\![\Delta^\dagger]\!]} \eta_0'$.

Proof: The proof is by induction on a derivation of $\Delta \vdash S : (\mathtt{com}\ \kappa)$ with cases on the last rule used in the derivation.

Could we say something more?

Suppose $\Delta \vdash S : (\mathtt{com}\ L)$ and $\eta \sim_{[\![\Delta^\dagger]\!]} \eta'$. If $\eta_0 = [\![\Delta^\dagger \vdash S]\!]\eta$, then *there exists* η_0', such that $\eta_0' = [\![\Delta^\dagger \vdash S]\!]\eta'$ and $\eta_0 \sim_{[\![\Delta^\dagger]\!]} \eta_0'$.

Dually, if $\eta_0' = [\![\Delta^\dagger \vdash S]\!]\eta'$, then *there exists* η_0, such that $\eta_0 = [\![\Delta^\dagger \vdash S]\!]\eta$ and $\eta_0 \sim_{[\![\Delta^\dagger]\!]} \eta_0'$.

Handling loops

Semantics of commands: The meaning of a command $\Gamma \vdash S$ is a function $[\![\Gamma]\!] \to [\![\Gamma]\!]_\perp$ that takes a store η, and returns a possibly updated store or returns $\perp$ which indicates divergence.

$$[\![\Gamma \vdash \text{while } e \text{ do } S]\!] \quad = \quad \mathrm{lub}\; f \quad \text{where}$$

$$f_0\, \eta \;\; = \perp$$

$$f_{i+1}\, \eta = \text{let } b = [\![\Delta^\dagger \vdash e : \texttt{bool}]\!]\eta \text{ in}$$

$$\text{if } b = 0 \text{ then } \eta \text{ else}$$

$$\text{let } \hat{\eta} = [\![\Delta^\dagger \vdash S]\!]\eta \text{ in } f_i\, \hat{\eta}$$

We assume that the (metalanguage) construct, let $x = e_1$ in e_2, is strict: If the value of e_1 is $\perp$ then that is the value of the entire let expression; otherwise, its value is the value of e_2 with x bound to the value of e_1.

Predicates

The predicate wconf on $[\![\Gamma]\!] \to [\![\Gamma]\!]_\perp$ is defined as:

$$\mathit{wconf}\ \mathsf{f} \quad \Longleftrightarrow \quad \forall \eta \in [\![\Gamma]\!] \bullet \mathsf{f}\eta \neq \perp\ \Rightarrow\ \eta \sim_{[\![\Gamma]\!]} \mathsf{f}\eta$$

The predicate nonint on $[\![\Gamma]\!] \to [\![\Gamma]\!]_\perp$ is defined as:

$$\mathit{nonint}\ \mathsf{f} \quad \Longleftrightarrow \quad \forall (\eta, \eta') \bullet (\eta \sim \eta') \wedge (\mathsf{f}\eta \neq \perp \neq \mathsf{f}\eta')\ \Rightarrow\ \mathsf{f}\eta \sim \mathsf{f}\eta'$$

Technical results revisited

Lemma (write confinement of commands) Suppose $\Delta \vdash S : (\mathtt{com}\ \kappa)$. Then $\kappa = H \Rightarrow \mathit{wconf}[\![\Delta^\dagger \vdash S]\!]$.

Theorem (safe commands are noninterfering) Suppose $\Delta \vdash S : (\mathtt{com}\ \kappa)$. Then $\mathit{nonint}[\![\Delta^\dagger \vdash S]\!]$.

The proofs of the above require additional lemmas for the while case:

1. $\forall i \bullet \mathit{wconf}\ f_i$
2. $(\forall i \bullet \mathit{wconf}\ f_i) \Rightarrow \mathit{wconf}\ (\mathrm{lub}\ f)$
3. $\forall i \bullet \mathit{nonint}\ f_i$
4. $(\forall i \bullet \mathit{nonint}\ f_i) \Rightarrow \mathit{nonint}\ (\mathrm{lub}\ f)$