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

\begin{verbatim}
  \textbf{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;
\end{verbatim}
Syntax and typing rules

Syntax:

\[ T ::= \text{int} \]
\[ e ::= x \mid n \mid e_1 + e_2 \mid e_1 - e_2 \]
\[ S ::= x := e \mid \text{if } e \text{ then } S_1 \text{ else } S_2 \mid S_1 ; S_2 \]

Typing rules for expressions: \( \Gamma \vdash e : T \)

- \( \Gamma \vdash x : \Gamma x \)
- \( \Gamma \vdash n : \text{int} \)
  - \( \Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \)
    - \( \Gamma \vdash e_1 + e_2 : \text{int} \)
  - \( \Gamma \vdash e_1 - e_2 : \text{int} \)
Typing rules for commands: \( \Gamma \vdash S \)

\[
\frac{
\Gamma, x : T \vdash e : T \\
\Gamma, x : T \vdash x := e
}{
\Gamma, x : T \vdash x := e
}
\]

\[
\frac{
\Gamma \vdash e : \text{int} \\
\Gamma \vdash S_1 \\
\Gamma \vdash S_2
}{
\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2
}
\]

\[
\frac{
\Gamma \vdash S_1 \\
\Gamma \vdash S_2
}{
\Gamma \vdash S_1; S_2
}
\]
Semantics

\[
\begin{align*}
\llbracket \text{int} \rrbracket &= \mathbb{Z} \\
\llbracket \Gamma \rrbracket &= \{ \eta \mid \text{dom } \eta = \text{dom } \Gamma \land \forall x \in \text{dom } \eta \cdot \eta x \in \llbracket \Gamma \cdot x \rrbracket \}
\end{align*}
\]

Semantics of expressions: The meaning of an expression $\Gamma \vdash e : T$ is a function $\llbracket \Gamma \rrbracket \to \llbracket T \rrbracket$.

\[
\begin{align*}
\llbracket \Gamma \vdash x : T \rrbracket \eta &= \eta x \\
\llbracket \Gamma \vdash n : \text{int} \rrbracket \eta &= n \\
\llbracket \Gamma \vdash e_1 + e_2 : \text{int} \rrbracket \eta &= \text{let } d_1 = \llbracket \Gamma \vdash e_1 : \text{int} \rrbracket \eta \text{ in } \\
&\quad \text{let } d_2 = \llbracket \Gamma \vdash e_2 : \text{int} \rrbracket \eta \text{ in } d_1 + d_2
\end{align*}
\]

Similarly for $e_1 - e_2$. 
Semantics of commands: The meaning of a command $\Gamma \vdash S$ is a function $[\Gamma] \rightarrow [\Gamma]$ that takes a store $\eta$, and returns a possibly updated store.

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

$$[\Gamma \vdash \text{if } e \text{ then } S_1 \text{ else } S_2] \eta$$

$$= \text{let } b = [\Gamma \vdash e : \text{int}] \eta \text{ in }$$

$$\text{if } b > 0 \text{ then } [\Gamma \vdash S_1] \eta \text{ else } [\Gamma \vdash S_2] \eta$$

$$[\Gamma \vdash S_1; S_2] \eta = \text{let } \eta_1 = [\Gamma \vdash S_1] \eta \text{ in } [\Gamma \vdash S_2] \eta_1$$
What does being secure mean?

Suppose $\Gamma \vdash e : T$ 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 $\kappa$ 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 $\text{com} \ \kappa$ with the meaning that all assigned variables have at least level $\kappa$. 
Security type system: Rules for expressions

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

\[
\begin{align*}
\Delta &\vdash x : \Delta \ x \\
\Delta &\vdash n : (\text{int}, \kappa) \\
\Delta &\vdash e_1 : (\text{int}, \kappa) \quad \Delta &\vdash e_2 : (\text{int}, \kappa) \\
\hline
\Delta &\vdash e_1 + e_2 : (\text{int}, \kappa) \\
\Delta &\vdash e : (T, \kappa) \quad \kappa \leq \kappa' \\
\hline
\Delta &\vdash e : (T, \kappa')
\end{align*}
\]
Security type system: Rules for commands

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

\[
\Delta, x : (T, \kappa) \vdash e : (T, \kappa) \\
\Delta, x : (T, \kappa) \vdash x := e : (\text{com } \kappa)
\]

\[
\Delta \vdash e : (\text{int}, \kappa) \quad \Delta \vdash S_1 : (\text{com } \kappa) \quad \Delta \vdash S_2 : (\text{com } \kappa) \\
\Delta \vdash \text{if } e \text{ then } S_1 \text{ else } S_2 : (\text{com } \kappa)
\]

\[
\Delta \vdash S_1 : (\text{com } \kappa) \quad \Delta \vdash S_2 : (\text{com } \kappa) \\
\Delta \vdash S_1 ; S_2 : (\text{com } \kappa)
\]

\[
\Delta \vdash S : (\text{com } \kappa_1) \quad \kappa \leq \kappa_1 \\
\Delta \vdash S : (\text{com } \kappa)
\]
Examples revisited

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

$x := 42 : (\text{com H}); y := 42 : (\text{com L})$

$x := 42; y := 42 : (\text{com L})$

$y := x \quad (* \text{untypable} *)$

$y := x; y := y - x; \quad (* \text{untypable} *)$

$x := y; y := x; \quad (* \text{untypable} *)$

$x := x \text{mod} 2; \quad (* (\text{com H}) *)$

$y := 0; \quad (* (\text{com L}) *)$

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:

$$d \sim_{[T]} d' \iff d = d' \text{ for primitive types } T$$

$$\eta \sim_{[\Delta^+]} \eta' \iff \forall (x : (T, \kappa)) \in \Delta \bullet \kappa = L \Rightarrow (\eta x) \sim_{[T]} (\eta' x)$$

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.
Lemma (write confinement of commands)

Suppose $\Delta \vdash S : (\text{com } \kappa)$. For all $\eta$, if $\eta_0 = \llbracket \Delta^\dagger \vdash S \rrbracket \eta$ then

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

**Proof:** The proof is by induction on a derivation of $\Delta \vdash S : (\text{com } \kappa)$ and by cases on the last rule used in the derivation.
Theorem (safe commands are noninterfering)

Suppose $\Delta \vdash S : (\text{com } \kappa)$ and $\eta \sim_{[\Delta^+]} \eta'$. Let $\eta_0 = [[\Delta^+ \vdash S]]\eta$ and $\eta'_0 = [[\Delta^+ \vdash S]]\eta'$. Then $\eta_0 \sim_{[\Delta^+]} \eta'_0$.

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

Could we say something more?

Suppose $\Delta \vdash S : (\text{com } L)$ and $\eta \sim_{[\Delta^+]} \eta'$. If $\eta_0 = [[\Delta^+ \vdash S]]\eta$, then there exists $\eta'_0$, such that $\eta'_0 = [[\Delta^+ \vdash S]]\eta'$ and $\eta_0 \sim_{[\Delta^+]} \eta'_0$.

Dually, if $\eta'_0 = [[\Delta^+ \vdash S]]\eta'$, then there exists $\eta_0$, such that $\eta_0 = [[\Delta^+ \vdash S]]\eta$ and $\eta_0 \sim_{[\Delta^+]} \eta'_0$. 
Handling loops

Semantics of commands: The meaning of a command $\Gamma \vdash S$ is a function $[\Gamma] \rightarrow [\Gamma] \bot$ that takes a store $\eta$, and returns a possibly updated store or returns $\bot$ which indicates divergence.

$$[[\Gamma \vdash \text{while } e \text{ do } S]] = \text{lub } f$$

where

$$f_0 \eta = \bot$$

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

if $b = 0$ then $\eta$ 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 $\bot$ 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 $e_1$. 
The predicate \( wconf \) on \([\Gamma] \rightarrow [\Gamma]_\bot\) is defined as:

\[
wconf f \iff \forall \eta \in [\Gamma] \bullet f\eta \neq \bot \Rightarrow \eta \sim [\Gamma] f\eta
\]

The predicate \( nonint \) on \([\Gamma] \rightarrow [\Gamma]_\bot\) is defined as:

\[
nonint f \iff \forall (\eta, \eta') \bullet (\eta \sim \eta') \land (f\eta \neq \bot \neq f\eta') \Rightarrow f\eta \sim f\eta'
\]
Technical results revisited

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

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

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

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