Language-based enforcement of Noninterference

Anupam Datta
CMU

Fall 2007-08
Security issues with C

- Cyclone: A typed language with syntax similar to C, but provides memory safety
  - No buffer overflows, format string vulnerabilities, dangling pointers,...
  - Perfectly “good” Cyclone program can reveal secrets by printing them on the screen
Information flow

◆ Security definition
  • Non-interference [Goguen-Meseguer82]
◆ Embodiment in a programming language
  • [Denning-Denning77]
  • [Volpano-Smith-Irvine96]
◆ Extending Java with information flow control
  • Jif [Myers-Liskov97]
Definition of Security

- Non-interference (idea)

No information flows from high inputs to low outputs
**Example**

If $x = 1$ then $y := 1$ else $y := 0$

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>Yes</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>Yes</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>No</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Formal definition

A system $S$ is said to be non-interfering from High to Low iff:

$$\forall tr \in I^*, c \in I \cdot Output_L(S, tr, c) = Output_L(S, purge_{HI}(tr), c)$$

- System is deterministic finite state machine: takes input and transitions to next state producing output
- Trace $tr$ is a sequence of inputs and outputs (high & low)
- $Output_L(S, tr, c)$: low output of system $S$ when input $c$ is applied to the state corresponding to trace $tr$
- $purge_{HI}(tr)$: returns a trace with all high inputs in $tr$ removed
Specification and Enforcement

◆ Approach
  • Use a typed programming language
  • Types represent security levels
    - HI, LO, ...
  • Subtyping captures partial order in the lattice of security levels
    - LO \leq HI
  • Type system captures allowed information flows
  • Soundness theorem
    - Well-typed programs satisfy non-interference
Language Definition

- Syntax
- Type System
- Operational Semantics

- Soundness Theorem
  - Well typed programs satisfy non-interference
We consider a core block-structured language described below. It consists of phrases, which are either expressions $e$ or commands $c$:

$\begin{align*}
(\text{phrases}) \quad p & ::= \quad e \mid c \\
(\text{expressions}) \quad e & ::= \quad x \mid l \mid n \mid e + e' \mid e - e' \mid e = e' \mid e < e' \\
(\text{commands}) \quad c & ::= \quad e := e' \mid c; c' \mid \text{if } e \text{ then } c \text{ else } c' \mid \\
& \quad \text{while } e \text{ do } c \mid \text{letvar } x := e \text{ in } c
\end{align*}$

Metavariable $x$ ranges over identifiers, $l$ over locations (addresses), and $n$ over integer literals. Integers are the only values. We use 0 for false and 1 for true, and assume that locations are well ordered.
Syntax (II)

The types of the core language are stratified as follows.

\[
\begin{align*}
\text{(data types)} & \quad \tau ::= s \\
\text{(phrase types)} & \quad \rho ::= \tau \mid \tau \ var \mid \tau \ cmd
\end{align*}
\]

Metavariabe \( s \) ranges over the set \( SC \) of security classes, which is assumed to be partially ordered by \( \leq \). Type \( \tau \ var \) is the type of a variable and \( \tau \ cmd \) is the type of a command.
Type System (I)

Typing judgment

\[ \lambda; \gamma \vdash p : \rho \]

where \( \lambda \) is a location typing and \( \gamma \) is an identifier typing. The judgment means that phrase \( p \) has type \( \rho \), assuming \( \lambda \) prescribes types for locations in \( p \) and \( \gamma \) prescribes types for any free identifiers in \( p \). An identifier typing is a finite function mapping identifiers to \( \rho \) types; \( \gamma(x) \) is the \( \rho \) type assigned to \( x \) by \( \gamma \). Also, \( \gamma[x : \rho] \) is a modified identifier typing that assigns type \( \rho \) to \( x \) and assigns type \( \gamma(x') \) to any identifier \( x' \) other than \( x \). A location typing is a finite function mapping locations to \( \tau \) types. The notational conventions for location typings are similar to those for identifier typings.
Type system (II)

\[
\begin{align*}
\text{(INT)} & : & \lambda; \gamma \vdash n : \tau \\
\text{(VAR)} & : & \lambda; \gamma \vdash x : \tau \text{ var} \quad \text{if } \gamma(x) = \tau \text{ var} \\
\text{(VARLOC)} & : & \lambda; \gamma \vdash l : \tau \text{ var} \quad \text{if } \lambda(l) = \tau \\
\text{(ARITH)} & : & \lambda; \gamma \vdash e : \tau, \\
& & \lambda; \gamma \vdash e' : \tau \\
& & \lambda; \gamma \vdash e + e' : \tau \\
\text{(R-VAL)} & : & \lambda; \gamma \vdash e : \tau \text{ var} \\
& & \lambda; \gamma \vdash e : \tau \\
\text{(ASSIGN)} & : & \lambda; \gamma \vdash e' : \tau \\
& & \lambda; \gamma \vdash e := e' : \tau \text{ cmd}
\end{align*}
\]
Type System (III)

\[\begin{align*}
\text{(COMPOSE)} & \quad \frac{\lambda; \gamma \vdash c : \tau \ \cmd,}{} \\
& \quad \frac{\lambda; \gamma \vdash c' : \tau \ \cmd}{}
\quad \lambda; \gamma \vdash c ; c' : \tau \ \cmd
\end{align*}\]

\[\begin{align*}
\text{(IF)} & \quad \frac{\lambda; \gamma \vdash e : \tau,}{} \\
& \quad \frac{\lambda; \gamma \vdash c : \tau \ \cmd,}{} \\
& \quad \frac{\lambda; \gamma \vdash c' : \tau \ \cmd}{}
\quad \lambda; \gamma \vdash \textbf{if } e \textbf{ then } c \textbf{ else } c' : \tau \ \cmd
\end{align*}\]

\[\begin{align*}
\text{(WHILE)} & \quad \frac{\lambda; \gamma \vdash e : \tau,}{} \\
& \quad \frac{\lambda; \gamma \vdash c : \tau \ \cmd}{}
\quad \lambda; \gamma \vdash \textbf{while } e \textbf{ do } c : \tau \ \cmd
\end{align*}\]

\[\begin{align*}
\text{(LETVAR)} & \quad \frac{\lambda; \gamma[x : \tau \ \var] \vdash c : \tau' \ \cmd}{} \\
& \quad \lambda; \gamma \vdash \textbf{letvar } x := e \textbf{ in } c : \tau' \ \cmd
\end{align*}\]
Example

if $x = 1$ then $y := 1$ else $y := 0$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$y$</td>
<td>NI</td>
</tr>
<tr>
<td>$H$</td>
<td>$H$</td>
<td>Yes</td>
</tr>
<tr>
<td>$L$</td>
<td>$L$</td>
<td>Yes</td>
</tr>
<tr>
<td>$H$</td>
<td>$L$</td>
<td>No</td>
</tr>
<tr>
<td>$L$</td>
<td>$H$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Will justify rows 1 & 2
Example with types

For example, suppose $\gamma(x) = \gamma(y) = H \, var$. By the preceding typing rule for assignment, we have $\gamma \vdash y := 1 : H \, cmd$ and $\gamma \vdash y := 0 : H \, cmd$. This means that each statement can be placed in a context where high information is implicitly known through the guard of a conditional statement. An example is

\[
\text{if } x = 1 \text{ then } y := 1 \text{ else } y := 0. \]

With $\tau = H$, the secure flow typing rule for conditionals gives

\[
\gamma \vdash \text{if } x = 1 \text{ then } y := 1 \text{ else } y := 0 : H \, cmd
\]

Key rules used are (ASSIGN) and (IF)
Type System (IV)

Figure 3. Subtyping rules
Example

if \( x = 1 \) then \( y := 1 \) else \( y := 0 \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( y )</td>
<td>( \text{NI} )</td>
</tr>
<tr>
<td>( H )</td>
<td>( H )</td>
<td>Yes</td>
</tr>
<tr>
<td>( L )</td>
<td>( L )</td>
<td>Yes</td>
</tr>
<tr>
<td>( H )</td>
<td>( L )</td>
<td>No</td>
</tr>
<tr>
<td>( L )</td>
<td>( H )</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Will justify rows 3 & 4
Example with types

Suppose $x: L \text{ var}$ and $y: H \text{ var}$

Two options
1. Use \texttt{(SUBTYPE)} to infer $x: H \text{ var}$
2. Use \texttt{(CMD-)} to infer $H \text{ cmd} \subseteq L \text{ cmd}$ followed by \texttt{(SUBTYPE)} to infer $y := 1: L \text{ cmd}$ and $y := 0: L \text{ cmd}$

$x: H \text{ var}$ and $y: L \text{ var}$ is not well-typed as expected
Operational Semantics (I)

• $\mu$ is memory, a function from locations to values

• $\mu(l)$ is contents of location $l$

• Judgments

\[
\begin{align*}
\text{(BASE)} & \quad \mu \vdash n \Rightarrow n \\
\text{(CONTENTS)} & \quad \mu \vdash l \Rightarrow \mu(l) \quad \text{if } l \in \text{dom}(\mu) \\
\text{(ADD)} & \quad \mu \vdash e \Rightarrow n, \quad \mu \vdash e' \Rightarrow n' \\
& \quad \mu \vdash e + e' \Rightarrow n + n' \\
\text{(UPDATE)} & \quad \mu \vdash e \Rightarrow n, \quad l \in \text{dom}(\mu) \\
& \quad \mu \vdash l := e \Rightarrow \mu[l := n] \\
\text{(SEQUENCE)} & \quad \mu \vdash c \Rightarrow \mu', \quad \mu' \vdash c' \Rightarrow \mu'' \\
& \quad \mu \vdash c; c' \Rightarrow \mu'' \\
\text{(BRANCH)} & \quad \mu \vdash e \Rightarrow 1, \quad \mu \vdash c \Rightarrow \mu' \\
& \quad \mu \vdash \text{if } e \text{ then } c \text{ else } c' \Rightarrow \mu' \\
& \quad \mu \vdash e \Rightarrow 0, \quad \mu \vdash c' \Rightarrow \mu' \\
& \quad \mu \vdash \text{if } e \text{ then } c \text{ else } c' \Rightarrow \mu'
\end{align*}
\]
(LOOP)
\[\begin{align*}
\mu \vdash e & \Rightarrow 0 \\
\mu \vdash \text{while } e \text{ do } c & \Rightarrow \mu
\end{align*}\]
\[\begin{align*}
\mu \vdash e & \Rightarrow 1, \\
\mu \vdash c & \Rightarrow \mu', \\
\mu' \vdash \text{while } e \text{ do } c & \Rightarrow \mu'' \\
\mu \vdash \text{while } e \text{ do } c & \Rightarrow \mu''
\end{align*}\]

(BINDVAR)
\[\begin{align*}
\mu \vdash e & \Rightarrow n, \\
l & \text{is the first location not in } \text{dom}(\mu), \\
\mu[l := n] \vdash [l/x]c & \Rightarrow \mu' \\
\mu \vdash \text{letvar } x := e \text{ in } c & \Rightarrow \mu' - l
\end{align*}\]
Soundness Theorem

Theorem 6.8 (Type Soundness) Suppose

(a) $\lambda \vdash c : \rho$  \hspace{1cm} $c$ is well-typed
(b) $\mu \vdash c \Rightarrow \mu'$  \hspace{1cm} execution one
(c) $\nu \vdash c \Rightarrow \nu'$  \hspace{1cm} execution two
(d) $\text{dom}(\mu) = \text{dom}(\nu) = \text{dom}(\lambda)$
(e) $\nu(l) = \mu(l)$ for all $l$ such that $\lambda(l) \leq \tau$ \hspace{1cm} the same low inputs

Then $\nu'(l) = \mu'(l)$ for all $l$ such that $\lambda(l) \leq \tau$. \hspace{1cm} the same low outputs
Definition of Security

- Non-interference (idea)

No information flows from high inputs to low outputs
What next?

- A more useful security typed programming language
  - Jif: Java + information flow
  - Examples of secure web applications

- What is the “right” security definition?
  - Issues with non-interference
Thanks

Questions?