CS 5430
Information-Flow Control

Elisavet Kozyri
Spring 2019
Information Flow Control

• We are given:
  – a lattice \( \langle L, \sqsubseteq \rangle \) of labels,
  – a program, and
  – an *environment* \( \Gamma \) that maps variables to labels.

• Threat model: Attacker knows \( L \) inputs, knows program code, and can observe \( L \) outputs.
Information Flow Control

• Executions of the program should cause only allowed flows (with respect to the lattice).

• The program is expected to satisfy noninterference (NI):
  – Different H inputs, keeping L inputs fixed, should not cause different L outputs.
Information Flow Control

• How can we check that a program satisfies NI?

• Design an *enforcement mechanism*:
  – takes as an input the program, and
  – outputs whether that program satisfies NI.
Enforcing NI

• **Static mechanism**
  – Checking program before execution.

• **Dynamic mechanism**
  – Checking program during execution.

• **Hybrid mechanism**
  – Combination of static and dynamic.
Static enforcement mechanism

\[
\begin{align*}
x & := 2; \\
y & := x + z; \\
\text{if } y > 0 \text{ then} & \quad z := 10; \\
& \quad x := x - 1; \\
\text{else} & \quad z := w
\end{align*}
\]
Static enforcement mechanism

\[
\begin{align*}
x &:= 2; \\
y &:= x + z; \\
\text{if } y > 0 \text{ then} & \\
\quad z &:= 10; \\
\quad x &:= x - 1; \\
\text{else} & \\
\quad z &:= w
\end{align*}
\]
Static enforcement mechanism

\[
x := 2;
y := x+z;
\]

if \( y > 0 \) then

\[
z := 10;
x := x - 1;
\]

else

\[
z := w
\]
Static enforcement mechanism

\[
x := 2; \\
y := x+z; \\
\text{if } y > 0 \text{ then} \\
\quad z := 10; \\
\quad x := x - 1; \\
\text{else} \\
\quad z := w
\]
Static enforcement mechanism

\[ x := 2; \]
\[ y := x + z; \]
\[ \text{if } y > 0 \text{ then} \]
\[ z := 10; \]
\[ x := x - 1; \]
\[ \text{else} \]
\[ z := w \]
Programs are written using this syntax:

\[ e ::= x \mid n \mid e_1+e_2 \mid \ldots \]

\[ c ::= x := e \]
\[ | \text{if } e \text{ then } c_1 \text{ else } c_2 \]
\[ | \text{while } e \text{ do } c \]
\[ | c_1; \ c_2 \]
Checking an assignment

Assignments cause explicit flows of values.

\[ x := y \]

It satisfies NI, if \( \Gamma(y) \subseteq \Gamma(x) \).
Checking an assignment

If $\Gamma(y) \subseteq \Gamma(x)$, then it satisfies NI.

$x := y$

Examples for confidentiality

<table>
<thead>
<tr>
<th>$\Gamma(x)$ is L.</th>
<th>$\Gamma(y)$ is L.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does this assignment satisfy NI?</td>
<td>smiley face</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\Gamma(x)$ is H.</th>
<th>$\Gamma(y)$ is L.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does this assignment satisfy NI?</td>
<td>smiley face</td>
</tr>
</tbody>
</table>
Checking an assignment

\[ x := y + z \]

It satisfies NI, if \( \Gamma(y+z) \sqsubseteq \Gamma(x) \).

It satisfies NI, if \( \Gamma(y) \sqcup \Gamma(z) \sqsubseteq \Gamma(x) \).
Checking an if-statement

\[
\text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2
\]

Conditional commands (e.g., if-statements and while-statements) cause \textit{implicit} flows of values.
if y>0 then x:=1 else x:=2

They reveal information about y>0.
Context label \emph{ctx}

\[
\text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2
\]

Their \emph{ctx} is \( \Gamma(y) \).
Context label $ctx$

if $y > 0$ then $x := 1$ else $x := 2$

Check if $ctx \sqsubseteq \Gamma(x)$, where $ctx = \Gamma(y)$. 
Context label $ctx$

if $y > 0$ then $x := e$ else $x := 2$

Check if $ctx \sqcup \Gamma(e) \subseteq \Gamma(x)$.

Implicit flow

Explicit flow
Checking an if-statement

If $\Gamma(y) \sqcup \Gamma(z) \sqsubseteq \Gamma(x)$, then it satisfies NI.

```plaintext
if y>0 then x:=z+1 else x:=z+2
```

Examples for confidentiality

<table>
<thead>
<tr>
<th>$\Gamma(x)$ is L.</th>
<th>$\Gamma(y)$, $\Gamma(z)$ are L.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma(x)$, $\Gamma(z)$ are H.</td>
<td>$\Gamma(y)$ is L.</td>
</tr>
</tbody>
</table>

Does this if-statement satisfy NI?
Checking an if-statement

if \( z > 0 \) then
  if \( y > 0 \) then \( x := 1 \) else \( x := 2 \)
else
  \( x := 3 \)

Its \( ctx \) is \( \Gamma(z) \).

Their \( ctx \) is \( \Gamma(z) \sqcup \Gamma(y) \).

Checking an if-statement

if \( z > 0 \) then
  if \( y > 0 \) then \( x := 1 \) else \( x := 2 \)
else
\( x := 3 \)

Check if \( ctx \sqsubseteq \Gamma(x) \),
where \( ctx = \Gamma(z) \cup \Gamma(y) \).

Check if \( ctx \sqsubseteq \Gamma(x) \),
where \( ctx = \Gamma(z) \).
A type system for information flow control

• Static
• Fixed environment $\Gamma$
• Labels as types
  – Label $\Gamma(x)$ is the type of $x$.
• Typing rules for all possible commands.
• **Goal**: type-correctness $\Rightarrow$ noninterference
We are already familiar with type systems!

Example of typing rule from Java:

\[
x + y : \text{int}
\]
assuming \(x : \text{int}\) and \(y : \text{int}\)
Typing rules for expressions

Judgement $\Gamma \vdash e : \ell$

- According to environment $\Gamma$, expression $e$ has type (i.e., label) $\ell$. 
Typing rules for expressions

Expression: $\Gamma \vdash e + e' : \ell \sqcup \ell'$
assuming $\Gamma \vdash e : \ell$ and $\Gamma \vdash e' : \ell'$

Inference rule:

Premises $\Gamma \vdash e : \ell$  $\Gamma \vdash e' : \ell'$
Conclusion $\Gamma \vdash e + e' : \ell \sqcup \ell'$
Typing rules for expressions

Constant: \[ \Gamma \vdash n : \bot \]

Variable: \[ \Gamma (x) = \ell \quad \Gamma \vdash x : \ell \]

*Bottom*: the least restrictive label.
Example

• Let $\Gamma(x) = L$ and $\Gamma(y) = H$.
• What is the type of $x + y + 1$?
• Proof tree:

\[
\begin{align*}
\Gamma(x) &= L \\
\Gamma(y) &= H
\end{align*}
\]
Example

• Let $\Gamma(x) = L$ and $\Gamma(y) = H$.
• What is the type of $x + y + 1$?
• Proof tree:

\[
\begin{array}{c}
\Gamma(x) = L \\
\hline
\Gamma \vdash x : L
\end{array}
\quad\quad
\begin{array}{c}
\Gamma(y) = H \\
\hline
\Gamma \vdash y : H
\end{array}
\]
Example

• Let $\Gamma(x) = L$ and $\Gamma(y) = H$.
• What is the type of $x+y+1$?

Proof tree:

\[
\frac{
\Gamma(x) = L \quad \Gamma(y) = H
}{
\Gamma \vdash x : L \quad \Gamma \vdash y : H
}
\]

$\Gamma \vdash 1 : L$
Example

• Let $\Gamma(x) = L$ and $\Gamma(y) = H$.
• What is the type of $x + y + 1$?
• Proof tree:

\[
\begin{align*}
\Gamma(x) &= L & \Gamma(y) &= H \\
\Gamma \vdash x : L & \quad \Gamma \vdash y : H & \quad \Gamma \vdash 1 : L \\
\hline
\Gamma \vdash x + y + 1 : H
\end{align*}
\]
Typing rules for commands

Judgement $\Gamma, ctx \vdash c$

- According to environment $\Gamma$, and context label $ctx$, command $c$ is type correct.
Assignment rule

\[
\begin{align*}
\Gamma & \vdash e : \ell \\
\ell \cup \text{ctx} & \subseteq \Gamma(x) \\
\hline
\Gamma,\text{ctx} & \vdash x := e
\end{align*}
\]
If-rule

\[ \Gamma \vdash e : \ell \quad \Gamma, \ell \sqcup \text{ctx} \vdash c_1 \quad \Gamma, \ell \sqcup \text{ctx} \vdash c_2 \]

\[ \Gamma, \text{ctx} \vdash \text{if e then } c_1 \text{ else } c_2 \]
If-rule (example)

\[ \Gamma \vdash y > 0 : \Gamma(y) \]

\[ \Gamma, \Gamma(y) \sqcup L \vdash x := 1 \]

\[ \Gamma, \Gamma(y) \sqcup L \vdash x := 2 \]

\[ \Gamma, L \vdash \text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2 \]

What is the relation between \( \Gamma(x) \) and \( \Gamma(y) \), such that the above judgement can be proved?
If-rule (example)

\[ \Gamma \vdash y > 0 : \Gamma(y) \]
\[ \Gamma, \Gamma(y) \cup L \vdash x := 1 \]
\[ \Gamma, L \vdash \text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2 \]

What is the relation between \(\Gamma(x)\) and \(\Gamma(y)\), such that the above judgement can be proved?
If-rule (example)

\[ \Gamma(y) \sqsubseteq \Gamma(x) \]

\[ \Gamma, L \vdash \text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2 \]

What is the relation between \( \Gamma(x) \) and \( \Gamma(y) \), such that the above judgement can be proved?
If-rule (example)

\[
\begin{align*}
\Gamma, \Gamma(z) \cup L \vdash & \text{if } y > 0 \text{ then } x := 1 \quad \text{else } x := 2 \\
\Gamma \vdash & z > 0 : \Gamma(z) \\
\Gamma, L \vdash & \text{if } z > 0 \text{ then } \{ \text{if } y > 0 \text{ then } x := 1 \text{ else } x := 2 \} \quad \text{else } \{ x := 3 \}
\end{align*}
\]
while-rule

\[
\Gamma \vdash e : \ell \\
\Gamma, \ell \sqcup ctx \vdash c
\]

\[
\Gamma, ctx \vdash \text{while } e \text{ do } c
\]
Sequence rule

\[
\Gamma,ctx \vdash c_1 \quad \Gamma,ctx \vdash c_2
\]

\[
\Gamma,ctx \vdash c_1; c_2
\]
Sequence rule (example)

\[
\Gamma, \ell \sqcup \Gamma(e) \vdash x := 1 \quad \Gamma, \ell \sqcup \Gamma(e) \vdash x := 2
\]

\[
\Gamma, \ell \vdash \text{if } e \text{ then } \{x := 1\} \text{ else } \{x := 2\}
\]

\[
\Gamma, \ell \vdash \text{if } e \text{ then } \{x := 1\} \text{ else } \{x := 2\}; x := 3
\]
Theorem

Type correctness $\Rightarrow$ Noninterference
• If a program does not satisfy NI, then type checking fails.

  – Example, where \( \Gamma(x) = L \) and \( \Gamma(y) = H \):

\[
\begin{align*}
\Gamma \vdash y : H & \quad H \sqcup L \subseteq \Gamma(x) \\
\Gamma, L \vdash x := y & \quad \text{Fails} \\
\end{align*}
\]

• But, if type checking fails, then the program might or might nor satisfy NI.

\[
\begin{align*}
\Gamma \vdash y*0 : H & \quad H \sqcup L \subseteq \Gamma(x) \\
\Gamma, L \vdash x := y*0 & \quad \text{Satisfies NI} \\
\end{align*}
\]
Soundness and completeness

The type system is *sound*:

Type correctness $\Rightarrow$ Noninterference

but not *complete*:

Noninterference $\nRightarrow$ Type correctness
This type system has false positives.
Can we build a complete mechanism?

- Is there an enforcement mechanism for information flow control that has no false positives?
  - A mechanism that rejects only programs that do not satisfy noninterference?
- No! [Sabelfeld and Myers, 2003]
  - “The general problem of confidentiality for programs is undecidable.”
  - The halting problem can be reduced to the information flow control problem.
  - Example:
    ```
    if h>0 then c; l:=2 else skip
    ```
  - If we could precisely decide whether this program is secure, we could decide whether c terminates!
But we can build a mechanism with fewer false positives.
This type system does not prevent leaks through termination channels.

Example of termination channel:

```plaintext
while h != 0 do skip;
1:=2
```

where `h` is a high variable and `1` is a low variable.

- The program leaks over termination channel.
  - It does not satisfy *termination sensitive* noninterference.

- But, the program is type correct.
  - It satisfies (vanilla) noninterference.
A solution

• To prevent covert channels due to infinite loops,
• strengthen the typing rule for while-statement, to allow only low guard expression:

\[
\frac{\Gamma \vdash e : \bot}{\Gamma, ctx \vdash \text{while } e \text{ do } c}
\]

• Now, type correctness implies termination sensitive NI.
• But, the enforcement mechanism becomes overly conservative.
• Another solution?