Lecture 20: Information Flow Control
Information flow policies

Can flow to: Alice

Automatic deduction of policies!
Labels represent policies
Labels represent policies
Noninterference

[Goguen and Meseguer 1982]

An interpretation of noninterference for a program:

- Changes on $H$ inputs should not cause changes on $L$ outputs.
Today: Information Flow Control

- **Goal**: Enforce IF policies that tag variables in a program.
- There is a mapping $\Gamma$ from variables to labels, which represent desired IF policies.
- The enforcement mechanism should ensure that a given program and a given $\Gamma$ satisfy noninterference.
Information Flow Control

Inputs

Variable

Program

Label

Outputs
Information Flow Control: fixed $\Gamma$

- $\Gamma$ remains the same during the analysis of the program.
- The mechanism checks that $\Gamma$ satisfies noninterference.
- The program is rejected, if at least one red arrow appears in the program.
Information Flow Control: flow-sensitive $\Gamma$

- $\Gamma$ may change during the analysis of the program.
- The mechanism deduces $\Gamma(x)$, $\Gamma(y)$, $\Gamma(z)$ such that noninterference is satisfied.
- The program is never rejected.
Enforcing IF policies

- **Static mechanism**
  - Checking and/or deduction of labels before execution.

- **Dynamic mechanism**
  - Checking and/or deduction of labels during execution.

- **Hybrid mechanism**
  - Combination of static and dynamic.
STATIC TYPE CHECKING

fixed $\Gamma$
A simple programming language

e ::= x | n | e1+e2 | ...

c ::= x := e
    | if e then c1 else c2
    | while e do c
    | c1; c2
Checking an assignment

\[ x := y \]

Examples for confidentiality

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

Does this assignment satisfy NI?

Does this assignment satisfy NI?
Checking an assignment

Assignments cause *explicit* information flows.

\[ x := y \]

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

\[ x := y \]

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

**MLS for confidentiality**

“no read up”:

\( S \) may read \( O \) iff \( \text{Label}(O) \sqsubseteq \text{Label}(S) \)

“no write down”:

\( S \) may write \( O' \) iff \( \text{Label}(S) \sqsubseteq \text{Label}(O') \)
Checking an assignment

\[ x := y \]

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

**MLS for confidentiality**

“no read up”:

C may read \( y \) iff \( \text{Label}(y) \sqsubseteq \text{Label}(C) \)

“no write down”:

C may write \( x \) iff \( \text{Label}(C) \sqsubseteq \text{Label}(x) \)
Checking an assignment

\[ x := y + z \]

It satisfies NI, if \( \Gamma(y) \subseteq \Gamma(x) \) and \( \Gamma(z) \subseteq \Gamma(x) \).

It satisfies NI, if \( \Gamma(y+z) \subseteq \Gamma(x) \).
Operator for combining labels

- For each \( \ell \) and \( \ell' \), there should exist label \( \ell \sqcup \ell' \), such that:
  - \( \ell \sqsubseteq \ell \sqcup \ell' \), \( \ell' \sqsubseteq \ell \sqcup \ell' \), and
  - if \( \ell \sqsubseteq \ell'' \) and \( \ell' \sqsubseteq \ell'' \), then \( \ell \sqcup \ell' \sqsubseteq \ell'' \).
- \( \ell \sqcup \ell' \) is called the **join** of \( \ell \) and \( \ell' \).
- Operator \( \sqcup \) is associative and commutative.
Checking an assignment

\[ x := y + z \]

It satisfies NI, if \( \Gamma(y) \cup \Gamma(z) \subseteq \Gamma(x) \).
The set of labels and relation \( \sqsubseteq \) define a lattice, with join operator \( \sqcup \).

---

Lattice of labels

- **Conf**, \{\}
- **Secret**, \{\}
- **Secret**, \{nuc\}
- **Secret**, \{crypto\}
- **Secret**, \{nuc, crypto\}
- **Conf**, \{nuc\}
- **Conf**, \{nuc, crypto\}
- **Conf**, \{crypto\}

\[ \sqsubseteq \]

\[ \sqcup \]

\[ \top \]

\[ \bot \]
Checking an if-statement

if $z > 0$ then
  $x := 1$
else
  $x := 0$

Examples for confidentiality

<table>
<thead>
<tr>
<th>$\Gamma(x)$ is L.</th>
<th>$\Gamma(z)$ is L.</th>
<th>Does this if-statement satisfy NI?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Gamma(x)$ is H.</td>
<td>$\Gamma(z)$ is L.</td>
<td>🙄</td>
</tr>
</tbody>
</table>
Checking an if-statement

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

Conditional commands (e.g., if-statements and while-statements) cause \textit{implicit} information flows.
Context

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

They reveal information about \( z > 0 \).

Introduce a context label \( ctx \)

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

if $z > 0$ then
    $x := 1$
else
    $x := 0$

Introduce a context label $ctx$

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

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

Implicit flow

Explicit flow
Typing system for IF control

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

Example of typing rule from Java or OCaml:

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

Judgement $\Gamma \vdash e : \ell$

According to mapping $\Gamma$, expression $e$ has type (i.e., label) $\ell$.

Constant: $\Gamma \vdash n : \bot$

Variable: $\Gamma \vdash x : \Gamma(x)$

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

if $\Gamma \vdash e : \ell$

and $\Gamma \vdash e' : \ell'$
Typing rules for expressions

Expression: \( \Gamma \vdash e + e' : \ell \sqcup \ell' \)

if \( \Gamma \vdash e : \ell \)
and \( \Gamma \vdash e' : \ell' \)

\[\Gamma \vdash e + e' : \ell \sqcup \ell'\]

Inference rule:

Premises \(\Gamma \vdash e : \ell \quad \Gamma \vdash e' : \ell'\)

Conclusion \(\Gamma \vdash e + e' : \ell \sqcup \ell'\)
Example

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

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

Judgement $\Gamma, ctx \vdash c$

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

\[
\Gamma, \, \text{ctx} \vdash x := e
\]

if \( \Gamma \vdash e : \ell \)

and \( \ell \sqcup \text{ctx} \subseteq \Gamma(x) \)

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

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

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

\[
\begin{array}{c}
\Gamma \vdash 1 : \bot \\
\bot \cup \Gamma(z) \cup L \subseteq \Gamma(x) \cup L \subseteq \Gamma(x)
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash z > 0 : \Gamma(z) \\
\Gamma, \Gamma(z) \cup L \vdash x := 1 \\
\Gamma, \Gamma(z) \cup L \vdash x := 0
\end{array}
\]

\[
\Gamma, L \vdash \text{if } z > 0 \text{ then } x := 1 \text{ else } x := 0
\]
Static type system

Assignment-Rule:

\[ \Gamma \vdash e : \ell \quad \ell \sqcup ctx \sqsubseteq \Gamma(x) \]
\[ \Gamma, ctx \vdash x := e \]

If-Rule:

\[ \Gamma \vdash e : \ell \quad \Gamma, \ell \sqcup ctx \vdash c1 \quad \Gamma, \ell \sqcup ctx \vdash c2 \]
\[ \Gamma, ctx \vdash \text{if } e \text{ then } c1 \text{ else } c2 \]

While-Rule:

\[ \Gamma \vdash e : \ell \quad \Gamma, \ell \sqcup ctx \vdash c \]
\[ \Gamma, ctx \vdash \text{while } e \text{ do } c \]

Sequence-Rule:

\[ \Gamma, ctx \vdash c1 \quad \Gamma, ctx \vdash c2 \]
\[ \Gamma, ctx \vdash c1; c2 \]
Soundness of type system

$$\Gamma, ctx \vdash c \implies c \text{ satisfies NI}$$
Limitations of the type system
This type system does not prevent leaks through covert channels.

Example of covert channel:

```plaintext
while s != 0 do { //nothing }

p:=1
```

where `s` is a secret variable (i.e., $\Gamma(s)=H$) and `p` is a public variable (i.e., $\Gamma(p)=L$).
A solution

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

\[
\begin{array}{c}
\Gamma \vdash e : \bot \\
\hline
\Gamma, ctx \vdash c \\
\end{array}
\]

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

- Now, type correctness implies termination sensitive NI.
- But, the enforcement mechanism becomes overly conservative.
- Another solution? Research!
This type system is not complete.

- $c$ satisfies noninterference $\nRightarrow \Gamma, ctx \vdash c$
  - There is a command $c$, such that noninterference is satisfied, but $c$ is not type correct.

**Example 1:**
- $\Gamma(x) = H, \Gamma(y) = L$
- $c$ is $\text{if } x>0 \text{ then } y:=1 \text{ else } y:=1$
- $c$ satisfies noninterference, because $x$ does not leak to $y$.
- $c$ is not type correct, because $\Gamma(x) \nsubseteq \Gamma(y)$. 
This type system is not complete.

• Example 2:
  • $\Gamma(x) = H, \Gamma(y) = L$
  • $c$ is if 1=1 then y:=1 else y:=x
  • $c$ satisfies noninterference, because $x$ does not leak to $y$.
  • $c$ is not type correct, because $\Gamma(x) \not\subseteq \Gamma(y)$.

• So, this type system is conservative. It has false negatives:
  • There are programs that are not type correct, but that satisfy noninterference.
Can we build a complete mechanism?

- Is there an enforcement mechanism for information flow control that has no false negatives?
  - 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>1 then c; l:=2 else skip
  ```
  - If we could precisely decide whether this program is secure, we could decide whether \( c \) terminates!
Can we build a mechanism with fewer false positives?

Switch from static to dynamic mechanisms!