ECS 235B Module 45
Compiler Based
Information Flow Mechanisms
Compiler-Based Mechanisms

- Detect unauthorized information flows in a program during compilation
- Analysis not precise, but secure
  - If a flow *could* violate policy (but may not), it is unauthorized
  - No unauthorized path along which information could flow remains undetected
- Set of statements *certified* with respect to information flow policy if flows in set of statements do not violate that policy
Example

```
if x = 1 then y := a;
else y := b;
```

• Information flows from $x$ and $a$ to $y$, or from $x$ and $b$ to $y$

• Certified only if $x \leq y$ and $a \leq y$ and $b \leq y$
  • Note flows for *both* branches must be true unless compiler can determine that one branch will *never* be taken
Declarations

• Notation:

\[ x: \text{int class } \{ A, B \} \]

means \( x \) is an integer variable with security class at least \( \text{lub}\{ A, B \} \), so \( \text{lub}\{ A, B \} \leq x \)

• Distinguished classes \textit{Low, High}
  • Constants are always \textit{Low}
Input Parameters

• Parameters through which data passed into procedure
• Class of parameter is class of actual argument

\[ i_p: \text{type\ class} \{ i_p \} \]
Output Parameters

• Parameters through which data passed out of procedure
  • If data passed in, called input/output parameter
• As information can flow from input parameters to output parameters, class must include this:

\[ o_p: \text{type class } \{ r_1, \ldots, r_n \} \]

where \( r_i \) is class of \( i \)th input or input/output argument
Example

```latex
proc \textit{sum}(x: \text{ int class} \{ A \});
    \textbf{var} \textit{out}: \text{ int class} \{ A, B \});
begin
    \textit{out} := \textit{out} + x;
end;
• Require $x \leq \textit{out}$ and $\textit{out} \leq \textit{out}$
```
Array Elements

• Information flowing out:
  
  \[ ... := a[i] \]

  Value of \( i \), \( a[i] \) both affect result, so class is \( \text{lub}\{ a[i], i \} \)

• Information flowing in:
  
  \[ a[i] := ... \]

• Only value of \( a[i] \) affected, so class is \( a[i] \)
Assignment Statements

\( x := y + z; \)

- Information flows from \( y, z \) to \( x \), so this requires \( \text{lub\{ } y, z \text{ } \} \leq x \)

More generally:

\( y := f( x_1, \ldots, x_n ) \)

- the relation \( \text{lub\{ } x_1, \ldots, x_n \text{ } \} \leq y \) must hold
Compound Statements

\[ x := y + z; \quad a := b * c - x; \]

- First statement: \( \text{lub}\{ y, z \} \leq x \)
- Second statement: \( \text{lub}\{ b, c, x \} \leq a \)
- So, both must hold (i.e., be secure)

More generally:

\[ S_1; \quad \ldots \quad S_n; \]

- Each individual \( S_i \) must be secure
Conditional Statements

if \( x + y < z \) then \( a := b \) else \( d := b \ast c - x \); end

• Statement executed reveals information about \( x, y, z \), so \( \text{lub}\{ x, y, z \} \leq \text{glb}\{ a, d \} \)

More generally:

if \( f(x_1, ..., x_n) \) then \( S_1 \) else \( S_2 \); end

• \( S_1, S_2 \) must be secure

• \( \text{lub}\{ x_1, ..., x_n \} \leq \text{glb}\{ y \mid y \text{ target of assignment in } S_1, S_2 \} \)
Iterative Statements

while $i < n$ do begin $a[i] := b[i]; \ i := i + 1; \$ end

• Same ideas as for “if”, but must terminate

More generally:
while $f(x_1, \ldots, x_n)$ do $S$;

• Loop must terminate;
• $S$ must be secure
• $\lub\{ x_1, \ldots, x_n \} \leq \glb\{ y \mid y \text{ target of assignment in } S \}$
Goto Statements

• No assignments
  • Hence no explicit flows
• Need to detect implicit flows
• *Basic block* is sequence of statements that have one entry point and one exit point
  • Control in block *always* flows from entry point to exit point
Example Program

\[\text{proc } tm(x: \text{array}[1..10][1..10] \text{ of integer class } \{x\};\]
\[\quad \text{var } y: \text{array}[1..10][1..10] \text{ of integer class } \{y\});\]
\[\text{var } i, j: \text{integer class } \{i\};\]
\[\text{begin}\]
\[\quad b_1 \quad i := 1;\]
\[\quad b_2 \quad \text{L2: if } i > 10 \text{ goto L7;}\]
\[\quad b_3 \quad j := 1;\]
\[\quad b_4 \quad \text{L4: if } j > 10 \text{ then goto L6;}\]
\[\quad b_5 \quad y[j][i] := x[i][j]; \quad j := j + 1; \quad \text{goto L4};\]
\[\quad b_6 \quad \text{L6: i := i + 1; goto L2;}\]
\[\quad b_7 \quad \text{L7:}\]
\[\text{end;}\]
Flow of Control

\[ b_1 \rightarrow b_2 \rightarrow b_3 \rightarrow b_4 \rightarrow b_5 \rightarrow b_6 \rightarrow b_7 \]

- \( i > n \) from \( b_2 \) to \( b_7 \)
- \( i \leq n \) from \( b_2 \) to \( b_7 \)
- \( j > n \) from \( b_6 \) to \( b_4 \)
- \( j \leq n \) from \( b_4 \) to \( b_6 \)
- \( j \leq n \) from \( b_4 \) to \( b_5 \) and back to \( b_4 \)
Immediate Forward Dominators

• Idea: when two paths out of basic block, implicit flow occurs
  • Because information says *which* path to take

• When paths converge, either:
  • Implicit flow becomes irrelevant; or
  • Implicit flow becomes explicit

• *Immediate forward dominator* of basic block $b$ (written IFD($b$)) is first basic block lying on all paths of execution passing through $b$
IFD Example

• In previous procedure:
  • $\text{IFD}(b_1) = b_2$ one path
  • $\text{IFD}(b_2) = b_7$ $b_2 \rightarrow b_7$ or $b_2 \rightarrow b_3 \rightarrow b_6 \rightarrow b_2 \rightarrow b_7$
  • $\text{IFD}(b_3) = b_4$ one path
  • $\text{IFD}(b_4) = b_6$ $b_4 \rightarrow b_6$ or $b_4 \rightarrow b_5 \rightarrow b_6$
  • $\text{IFD}(b_5) = b_4$ one path
  • $\text{IFD}(b_6) = b_2$ one path
Requirements

• $B_i$ is set of basic blocks along an execution path from $b_i$ to $\text{IFD}(b_i)$
  • Analogous to statements in conditional statement

• $x_{i_1}, ..., x_{i_n}$ variables in expression selecting which execution path
  containing basic blocks in $B_i$ used
  • Analogous to conditional expression

• Requirements for secure:
  • All statements in each basic blocks are secure
  • lub{$x_{i_1}, ..., x_{i_n}$} ≤ glb{$y \mid y \text{ target of assignment in } B_i$}
Example of Requirements

\[
\begin{align*}
& b_1 \quad j > n \\
& b_2 \quad i > n \\
& b_3 \quad i \leq n \\
& b_4 \quad j > n \\
& b_5 \quad j \leq n \\
& b_6 \quad \text{lub}\{\text{Low, } i\} \leq i \\
& b_7 \quad \text{Low} \leq i
\end{align*}
\]

\[
\begin{align*}
& b_1 \quad i := 1; \\
& b_2 \quad \text{L2: if } i > 10 \text{ goto L7; } \\
& b_3 \quad j := 1; \\
& b_4 \quad \text{L4: if } j > 10 \text{ then goto L6; } \\
& b_5 \quad y[j][i] := x[i][j]; \quad j := j + 1; \text{ goto L4; } \\
& b_6 \quad \text{L6: } i := i + 1; \text{ goto L2; } \\
& b_7 \quad \text{L7: }
\end{align*}
\]
Example of Requirements

• Within each basic block:
  \[ b_1: \text{Low} \leq i \quad b_3: \text{Low} \leq j \quad b_6: \text{lub}\{ \text{Low}, i \} \leq i \]
  \[ b_5: \text{lub}\{ x[i][j], i, j \} \leq y[j][i] \}; \text{lub}\{ \text{Low}, j \} \leq j \]
• Combining, \text{lub}\{ x[i][j], i, j \} \leq y[j][i] \}
• From declarations, true when \text{lub}\{ x, i \} \leq y

• \( B_2 = \{ b_3, b_4, b_5, b_6 \} \)
  • Assignments to \( i, j, y[j][i] \); conditional is \( i \leq 10 \)
  • Requires \( i \leq \text{glb}\{ i, j, y[j][i] \} \)
  • From declarations, true when \( i \leq y \)
Example (continued)

• $B_4 = \{ b_5 \}$
  • Assignments to $j$, $y[j][i]$; conditional is $j \leq 10$
  • Requires $j \leq \text{glb}\{ j, y[j][i] \}$
  • From declarations, means $j \leq y$

• Result:
  • Combine $\text{lub}\{ x, i \} \leq y$; $i \leq y$; $i \leq y$
  • Requirement is $\text{lub}\{ x, i \} \leq y$
Procedure Calls

\[ \text{tm}(a, b); \]

From previous slides, to be secure, \( \text{lub}\{x, i\} \leq y \) must hold

- In call, \( x \) corresponds to \( a \), \( y \) to \( b \)
- Means that \( \text{lub}\{a, i\} \leq b \), or \( a \leq b \)

More generally:

\[ \text{proc } \text{pn}(i_1, \ldots, i_m: \text{int}; \text{ var } o_1, \ldots, o_n: \text{int}); \begin{align*} \text{begin} & \quad S \\ \text{end}; \end{align*} \]
- \( S \) must be secure
- For all \( j \) and \( k \), if \( i_j \leq o_k \), then \( x_j \leq y_k \)
- For all \( j \) and \( k \), if \( o_j \leq o_k \), then \( y_j \leq y_k \)
Exceptions

\begin{verbatim}
proc copy(x: integer class { x });
    var y: integer class Low);
var sum: integer class { x };
    z: int class Low;
begin
    y := z := sum := 0;
    while z = 0 do begin
        sum := sum + x;
        y := y + 1;
    end
end
\end{verbatim}
Exceptions (cont)

• When sum overflows, integer overflow trap
  • Procedure exits
  • Value of $sum$ is MAXINT/$y$
  • Information flows from $y$ to $sum$, but $sum \leq y$ never checked

• Need to handle exceptions explicitly
  • Idea: on integer overflow, terminate loop
    
    ```
    on integer_overflow_exception sum do z := 1;
    ```
  • Now information flows from $sum$ to $z$, meaning $sum \leq z$
  • This is false ($sum = \{ x \}$ dominates $z = Low$)
Infinite Loops

```plaintext
proc copy(x: integer 0..1 class { x });
    var y: integer 0..1 class Low);
begin
    y := 0;
    while x = 0 do
        (* nothing *);
        y := 1;
end
• If x = 0 initially, infinite loop
• If x = 1 initially, terminates with y set to 1
• No explicit flows, but implicit flow from x to y
```
Semaphores

Use these constructs:

\[ \text{wait}(x): \quad \text{if } x = 0 \text{ then block until } x > 0; \quad x := x - 1; \]

\[ \text{signal}(x): \quad x := x + 1; \]

- \(x\) is semaphore, a shared variable
- Both executed atomically

Consider statement

\[ \text{wait}(\text{sem}); \quad x := x + 1; \]

- Implicit flow from \(\text{sem}\) to \(x\)
  - Certification must take this into account!
Flow Requirements

• Semaphores in *signal* irrelevant
  • Don’t affect information flow in that process

• Statement \( S \) is a *wait*
  • \( \text{shared}(S) \): set of shared variables read
    • Idea: information flows out of variables in \( \text{shared}(S) \)
  • \( \text{fglb}(S) \): \( \text{glb} \) of assignment targets *following* \( S \)
  • So, requirement is \( \text{shared}(S) \leq \text{fglb}(S) \)

• begin \( S_1; \ldots; S_n \) end
  • All \( S_i \) must be secure
  • For all \( i, \text{shared}(S_i) \leq \text{fglb}(S_i) \)
Example

begin
    \[ x := y + z; \quad (* S_1 *) \]
    \[ \text{wait}(sem); \quad (* S_2 *) \]
    \[ a := b \times c - x; \quad (* S_3 *) \]
end

• Requirements:
  • \( \text{lub}\{y, z\} \leq x \)
  • \( \text{lub}\{b, c, x\} \leq a \)
  • \text{sem} \leq a
    • Because \( \text{fglb}(S_2) = a \) and \( \text{shared}(S_2) = \text{sem} \)
Concurrent Loops

• Similar, but wait in loop affects all statements in loop
  • Because if flow of control loops, statements in loop before wait may be executed after wait

• Requirements
  • Loop terminates
  • All statements $S_1, ..., S_n$ in loop secure
  • $\text{lub}\{ \text{shared}(S_1), ..., \text{shared}(S_n) \} \leq \text{glb}(t_1, ..., t_m)$
    • Where $t_1, ..., t_m$ are variables assigned to in loop
Loop Example

```
while i < n do begin
    a[i] := item;    (* S_1 *)
    wait(sem);       (* S_2 *)
    i := i + 1;      (* S_3 *)
end
```

• Conditions for this to be secure:
  
  • Loop terminates, so this condition met
  • $S_1$ secure if \( \text{lub}\{i, \text{item}\} \leq a[i] \)
  • $S_2$ secure if \( \text{sem} \leq i \) and \( \text{sem} \leq a[i] \)
  • $S_3$ trivially secure
cobegin/coend

cobegin

\[ \begin{align*}
    x & := y + z; & (\ast S_1 \ast) \\
    a & := b \times c - y; & (\ast S_2 \ast)
\end{align*} \]

coend

• No information flow among statements
  • For \( S_1 \), \( \operatorname{lub}\{y, z\} \leq x \)
  • For \( S_2 \), \( \operatorname{lub}\{b, c, y\} \leq a \)

• Security requirement is both must hold
  • So this is secure if \( \operatorname{lub}\{y, z\} \leq x \land \operatorname{lub}\{b, c, y\} \leq a \)
Soundness

• Above exposition intuitive

• Can be made rigorous:
  • Express flows as types
  • Equate certification to correct use of types
  • Checking for valid information flows same as checking types conform to semantics imposed by security policy
Quiz

In the certification of iterative statements such as a while statement, why is the condition that the loop terminate necessary?

1. If it were not present, the certification mechanism could not determine if the program will halt
2. If it were not present, then whether the loop terminates or not will cause an unauthorized leak of information
3. If it were not present, the certification mechanism could not use the requirements for the conditional (if) statement
4. It is not necessary