

Chapter 15 INFORMATION FLOW

- Access controls can constrain the rights of a user.
- However, they cannot constrain the flow of information about a system.
- When a system has a security policy regulating information flow, the system must ensure that the information flows do not violate the constraints of the policy!
- Two types of mechanisms support how the information flows are checked:
 - Compile-time mechanisms
 - Runtime mechanisms
- These mechanisms have been implemented in several systems.
- Their effectiveness is demonstrated.

Chapter 15 INFORMATION FLOW

15.1 Basics and Background

- Information flow policies define the way information moves throughout a system.
- These policies are typically designed to preserve:
 - Confidentiality of data
 - Integrity of data
- Any confidentiality and integrity policy embodies an information flow policy.
- The Bell-LaPadula model is a confidentiality policy.
- Each confidentiality policy is also called an information flow policy.
- **EXAMPLE:**
 - The Bell-LaPadula model a lattice-based information flow policy.
 - Given 2 compartments (i.e., security levels) A and B , information can flow from an object in A to a subject in B iff $B \text{ dom } A$.
 - Example:
 - George is cleared into security level (SECRET, {NUC, EUR}).
 - DocA is classified as (CONFIDENTIAL, {NUC}).
 - DocB is classified as (SECRET, {EUR, US}).
 - DocC is classified as (SECRET, {EUR}).

George dom DocA as CONFIDENTIAL \leq SECRET and {NUC} \subseteq {NUC, EUR}.

George $\neg\text{dom}$ DocB as {EUR, US} \subseteq {NUC, EUR}.

George dom DocC as SECRET \leq SECRET and {EUR} \subseteq {NUC, EUR}.

Chapter 15 INFORMATION FLOW

- **NOTATION**
 - x : a variable in a program.
 - \underline{x} : information flow class of x
- **EXAMPLE: A system using the Bell-LaPadula model**
 - The variable x , which holds data in the compartment (TS, {NUC, EUR}), is set to 3.
 - $x=3$ and $\underline{x} = (\text{TS}, \{\text{NUC}, \text{EUR}\})$.
- **DEFINITION: The command sequence c causes a flow of information from x to y if, after execution of c , some information about the value of x before c was executed can be deduced from the value of y after c was executed.**
- **EXAMPLE:**
 - $y := x$; after the execution of this statement, information about the value of x in the initial state can be deduced from the value of y .
 - $y := x / z$; this statement reveals some information about x but not as much as the first statement.
- **EXAMPLE:**
 - Consider the statement
 - if $x = 1$ then $y := 0$;
 - else $y := 1$;
 - This statement does not explicitly assign the value of x to y .
 - Assume x is equally likely to be either 0 or 1.
 - If $x = 1$, then $y = 0$, and vice versa, so value of y depends on x .
 - Hence, information flows from x to y .

Chapter 15 INFORMATION FLOW

- **DEFINITION:** An implicit flow of information occurs when information flows from x to y without an explicit assignment of the form $y:=f(x)$, where $f(x)$ is an arithmetic expression with the variable x .
- The flow of information occurs not because of an assignment of the value of x but because of a flow of control based on the value of x .
 - This demonstrates that analyzing programs for assignments to detect information flow is not enough.
 - To detect all flows, implicit flows must be examined!
- **Information Flow Models and Mechanisms**
 - An information flow policy is a security policy that describes the authorized paths along which that information can flow.
 - Each model associates a label (representing a security class) with information and with entities containing that information.
 - Each model has rules about conditions under which information can flow throughout the system.
- **Notation**
 - $\underline{x} \leq \underline{y}$ means “information can flow from an element in class of x to an element in class of y .”

Chapter 15 INFORMATION FLOW

15.2 Compiler-Based Mechanisms

- These mechanisms detect unauthorized information flows in a program during compilation.
 - They determine if the information flows in a program could violate a given information flow policy.
 - This determination is not precise but secure.
 - If a flow could violate the policy, it is unauthorized.
 - No unauthorized path along which information could flow remains undetected.
- **DEFINITION:** A set of statements is certified with respect to an information flow policy if the information flow within that set of statements do not violate the policy.
- **EXAMPLE:**
 - Consider the statement
 if $x = 1$ then $y := a$;
 else $y := b$;
 - Information flows from x and a to y , or from x and b to y .
 - The statement is certified only if $\underline{x} \leq \underline{y}$ and $\underline{a} \leq \underline{y}$ and $\underline{b} \leq \underline{y}$.

Chapter 15 INFORMATION FLOW

- **Declarations**
 - x : integer class $\{A, B\}$: states that x is an integer variable and that data from security classes A and B may flow into x .
 - Viewing the security classes as a lattice, x 's class must be at least $\text{lub}\{A, B\}$, so $\text{lub}\{A, B\} \leq x$.
 - Two distinguished classes are Low (greatest lower bound) and High (least upper bound) of the lattice.
 - All the constants are of class Low.
- **Information can be passed into or out of a procedure through parameters.**
 - Classification of parameters
 - input parameters
 - output parameters
 - input/output parameters
 - (* input parameters: i_s ; output parameters: o_s ; input/output parameters: io_s *)
proc something (i_1, \dots, i_k ; var o_1, \dots, o_m ; io_1, \dots, io_n);
var l_1, \dots, l_k ; (* local variables *)
begin
 S (* body of procedure *)
end;
- **Input parameters**
 - Parameters through which data is passed into the procedure.
 - The class of an input parameter is the class of the actual argument: i_s : type class $\{i_s\}$
- **Output parameters**
 - Parameters through which data is passed out of the procedure.
 - r_1, \dots, r_p : the set of input and input/output parameters
 - As information can flow from input parameters to output parameters, the declaration for the type must capture this: o_s : type class $\{r_1, \dots, r_p\}$.
- **Input/Output parameters**
 - The input/output parameters are like output parameters, except that the initial value (as input) affects the allowed security classes.
 - r_1, \dots, r_p : the set of input and input/output parameters:
 - io_s : type class $\{r_1, \dots, r_p, io_1, \dots, io_k\}$

Chapter 15 INFORMATION FLOW

- **EXAMPLE**

- Consider the following procedure for adding two numbers.

```
proc sum(x: int class {A};  
      out: int class {A,B});  
begin  
  out := out + x;  
end;
```

- We require $\underline{x} \leq \underline{out}$ and $\underline{out} \leq \underline{out}$ (because \leq is reflective).
- The above declarations deal only with basic types (integers, characters, floating point numbers, etc.).
- Nonscalar types (arrays, records, variant records, etc.) also contain information.
- **EXAMPLE**
 - Consider the array *a*: array 1 .. 100 of int;
 - For information flows from $\underline{a[i]}$, the class involved is $\text{lub } \{\underline{a[i]}, i\}$.
 - Information flows from $\underline{a[i]}$ and from *i*.
 - For information flows into $\underline{a[i]}$, the class involved is $\underline{a[i]}$.
 - Information flows into $\underline{a[i]}$ affect only the value in $\underline{a[i]}$ (not the value in *i*).

Chapter 15 INFORMATION FLOW

- **Program Statements**
 - **Assignment Statements**
 - **Compound Statements**
 - **Conditional Statements**
 - **Iterative Statements**
 - **Goto Statements**
 - **Procedure Calls**
 - **Function calls**
 - **Input/Output Statements**
- **We will consider each of these types of statements separately with two exceptions.**
 - **Function calls** can be modeled as procedure calls by treating the return value of the function as an output parameter of the procedure.
 - **Input/Output statements** can be modeled as assignment statements in which the value is assigned to (or assigned from) a file.

Chapter 15 INFORMATION FLOW

- **Assignment Statements**

- An assignment statement has the form
$$y := f(x_1, \dots, x_n),$$
 x_i and y are variables and f is some function of those variables.
- Information flows from each of the x_i 's to y .
- The requirement for the information flow to be secure is $\text{lub } \{\underline{x}_1, \dots, \underline{x}_n\} \leq \underline{y}$.
- **EXAMPLE**
 - Consider the assignment statement $x := y + z$;
 - The requirement for the information flow to be secure is $\text{lub } \{\underline{y}, \underline{z}\} \leq \underline{x}$.

- **Compound Statements**

- A compound statement has the form
$$\begin{array}{l} \text{begin} \\ \quad S_1; \\ \quad \dots \\ \quad S_n; \\ \text{end;} \end{array}$$
- If the information flow in each of the statements is secure, then the information flow in the compound statement is secure.
- **EXAMPLE:**
 - Consider the statements
$$\begin{array}{l} \text{begin} \\ \quad x := y + z; \\ \quad a := b * c - x; \\ \text{end;} \end{array}$$
 - The requirements for the information flow to be secure are:
 - $\text{lub } \{\underline{y}, \underline{z}\} \leq \underline{x}$ for S_1 .
 - $\text{lub } \{\underline{b}, \underline{c}, \underline{x}\} \leq \underline{a}$ for S_2 .

Chapter 15 INFORMATION FLOW

- **Conditional Statements**

- A conditional statement has the form

if $f(x_1, \dots, x_n)$ then

S_1 ;

else

S_2 ;

end;

- Either S_1 or S_2 may be executed, so both must be secure.
- The selection of either S_1 or S_2 imparts information about the values of x_1, \dots, x_n , so information must be able to flow from these variables to any targets of assignments in S_1 or S_2 .
- This is possible iff the lowest class of the targets dominates the highest class of the variables x_1, \dots, x_n .
- The requirements for the information flow to be secure are:
 - S_1 secure
 - S_2 secure
 - $\text{lub } \{x_1, \dots, x_n\} \leq \text{glb } \{y / y \text{ is the target of an assignment in } S_1 \text{ or } S_2\}$

Chapter 15 INFORMATION FLOW

– EXAMPLE

- Consider the statements

if $x + y < z$ then

$a := b$;

else

$d := b * c - x$;

end;

- The requirements for the information flow to be secure are:

$\underline{b} \leq \underline{a}$ for S_1

$\text{lub } \{\underline{b}, \underline{c}, \underline{x}\} \leq \underline{d}$ for S_2

- However, the statement that is executed depends on the value of x , y , and z .
- Hence, the information also flows from x , y , and z to d and a .
- So, the requirements are $\text{lub } \{\underline{b}, \underline{c}, \underline{x}\} \leq \underline{d}$, $\underline{b} \leq \underline{a}$, and $\text{lub } \{\underline{x}, \underline{y}, \underline{z}\} \leq \text{glb } \{\underline{a}, \underline{d}\}$.

Chapter 15 INFORMATION FLOW

- **Iterative Statements**

- An iterative statement has the form

while $f(x_1, \dots, x_n)$ do
 S;

- The requirements for the information flow to be secure are:

Iterative statement terminates

S is secure

$\text{lub } \{\underline{x}_1, \dots, \underline{x}_n\} \leq \text{glb } \{\underline{y} \mid y \text{ is the target of an assignment in } S\}$

- **EXAMPLE:**

- Consider the statements

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

- This loop terminates.
- Consider the compound statement that has 2 statements.
- The first statement is secure if $\underline{i} \leq \underline{a[i]}$ and $\underline{b[i]} \leq \underline{a[i]}$.
- The second statement is secure because $\underline{i} \leq \underline{i}$.
- Hence, the compound statement is secure if $\text{lub } \{\underline{i}, \underline{b[i]}\} \leq \underline{a[i]}$.
- Finally, $\underline{a[i]}$ and \underline{i} are targets of assignments in the body of the loop.
- So, $\text{lub } \{\underline{i}, \underline{n}\} \leq \text{glb } \{\underline{a[i]}, \underline{i}\}$. Putting these together, we have $\text{lub } \{\underline{b[i]}, \underline{i}, \underline{n}\} \leq \text{glb } \{\underline{a[i]}, \underline{i}\}$.

Chapter 15 INFORMATION FLOW

- **Goto Statements**

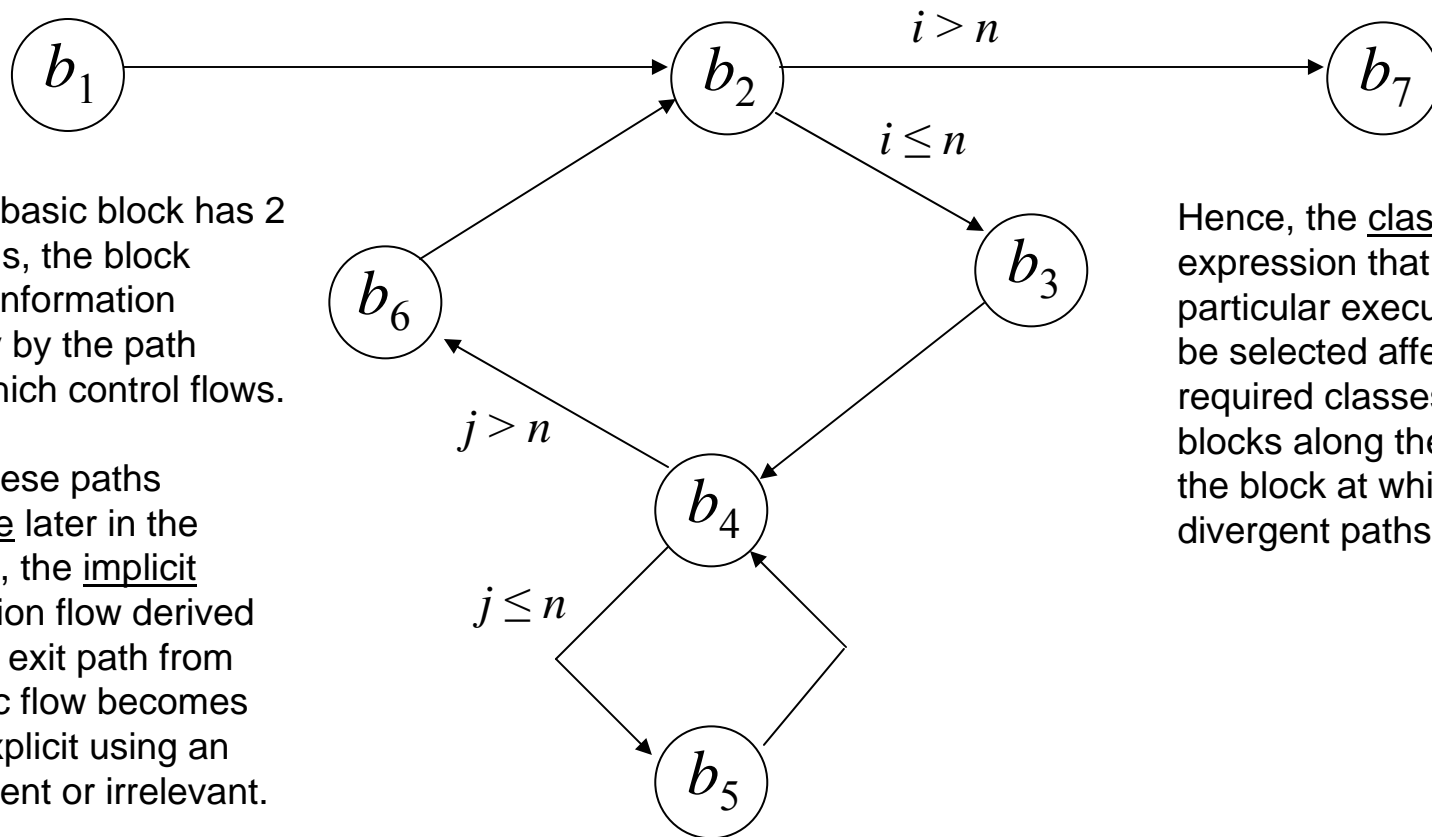
- A goto statement contains no assignments, so no explicit flows of information occurs.
- Implicit flows may occur. Analysis detects these flows.
- **DEFINITION:** A basic block is a sequence of statements in a program that has one entry point and one exit point.
 - Control in each block always flows from entry point to exit point.
- **EXAMPLE:**
 - Consider the following code fragment.

```
proc transmatrix(x: array[1..10][1..10] of int class {x};
                 var y: array[1..10][1..10] of int class {y});
var i, j: int class {tmp};
begin
  b1 i := 1;
  b2 L2: if i > 10 goto L7;
  b3 j := 1;
  b4 L4: if j > 10 then goto L6;
  b5 y[j][j] := x[j][j]; j := j + 1; goto L4;
  b6 L6: i := i + 1; goto L2;
  b7 L7:
end;
```

There are 7 basic blocks.
Control within a basic block flows from the first line to the last.
Analyzing the flow of control within a program is therefore equivalent to analyzing the flow of control among the program's basic blocks.

Chapter 15 INFORMATION FLOW

The flow of control among the basic blocks of the body of the procedure transmatrix.



When a basic block has 2 exit paths, the block reveals information implicitly by the path along which control flows.

When these paths converge later in the program, the implicit information flow derived from the exit path from the basic flow becomes either explicit using an assignment or irrelevant.

Hence, the class of the expression that causes a particular execution path to be selected affects the required classes of the blocks along the path up to the block at which the divergent paths converge.

Chapter 15 INFORMATION FLOW

- **DEFINITION:** An immediate forward dominator of a basic block b is the first block that lies on all paths of execution that pass through b : $IFD(b)$.
- **EXAMPLE:** In the procedure transmatrix:
 - $IFD(b_1) = b_2$ one path
 - $IFD(b_2) = b_7$ $b_2 \rightarrow b_7$ or $b_2 \rightarrow b_3 \rightarrow b_6 \rightarrow b_2 \rightarrow b_7$
 - $IFD(b_3) = b_4$ one path
 - $IFD(b_4) = b_6$ $b_4 \rightarrow b_6$ or $b_4 \rightarrow b_5 \rightarrow b_6$
 - $IFD(b_5) = b_4$ one path
 - $IFD(b_6) = b_2$ one path
- Computing the information flow requirement for the set of blocks along the path is to apply the logic for the conditional statement.
 - B_i : the set of basic blocks along an execution path from b_i to $IFD(b_i)$
 - The endpoints are excluded. Why?
 - Analogous to statements in conditional statement.
 - x_{i1}, \dots, x_{in} : the set of variables in the expression that selects the execution path containing basic blocks in B_i .
 - Analogous to conditional expression.
 - The requirements for the program's information flows to be secure are:
 - All statements in each basic blocks are secure.
 - $\text{lub}\{ \underline{x}_{i1}, \dots, \underline{x}_{in} \} \leq \text{glb}\{ \underline{y} \mid y \text{ target of assignment in } B_i \}$.

Chapter 15 INFORMATION FLOW

- **Procedure Calls**

- A procedure has the form

```
proc procname(i1,...,im : int; var o1,..., on : int);  
begin  
  S  
end;
```

- The information flow in the body of S must be secure.
- x_1, \dots, x_m and y_1, \dots, y_n : actual input and input/output parameters, respectively.

- The requirements for the information flow to be secure are:

S is secure.

For $j = 1, \dots, m$ and $k = 1, \dots, n$, if $\underline{i}_j \leq \underline{o}_k$, then $\underline{x}_j \leq \underline{y}_k$

For $j = 1, \dots, n$ and $k = 1, \dots, n$, if $\underline{o}_j \leq \underline{o}_k$, then $\underline{y}_j \leq \underline{y}_k$

- **EXAMPLE:**

- Consider the procedure transmatrix.
- The body of the procedure is secure w.r.t. information flow when $\text{lub } \{\underline{x}, \underline{tmp}\} \leq \underline{y}$.
- This indicates that the formal parameters x and y have the information flow relationship $\underline{x} \leq \underline{y}$.
- Now, suppose a program contains the call transmatrix (a, b).
- Hence, this call is secure w.r.t. information flow iff $\underline{a} \leq \underline{b}$.

Chapter 15 INFORMATION FLOW

- **Exceptions and Infinite Loops**

- Exceptions can cause information to overflow.

```
proc copy (x: int class { x }; var y: int class Low)
var sum: int 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
```

- When sum overflows, integer overflow trap occurs.
- If the trap is not handled, the procedure exists.
 - Value of x is $MAXINT/y$, where $MAXINT$ is the largest integer representable as an *int* on the system.
 - Information flows from y to x , but the flow relationship $\underline{x} \leq \underline{y}$ is not checked at any point.
- If exceptions are handled explicitly, the compiler can detect problems like this.
- Denning supplies a solution.
- **EXAMPLE:**
 - Ignoring the exception in the previous example would cause the program to loop indefinitely.
 - The programmer would like the loop to terminate when the exception occurred.
 - on overflowexception sum do z := 1;
 - this line causes the information to flow from sum to z , meaning that $\underline{sum} \leq \underline{z}$.
 - z is Low and \underline{sum} is $\{x\}$: This is incorrect ($\underline{sum} = \{x\}$ dominates $\underline{z} = \text{Low}$).

Chapter 15 INFORMATION FLOW

15.3 Execution-Based Mechanisms

- The goal of an execution-based mechanism is to prevent an information flow that violates policy.
- Checking the flow requirements of explicit flows achieves this result for statements involving explicit flows.
- For example, before the assignment $y = f(x_1, \dots, x_n)$ is executed, the execution mechanism verifies that $\text{lub} \{ \underline{x}_1, \dots, \underline{x}_n \} \leq \underline{y}$.
 - If the condition is satisfied, the assignment proceeds.
 - A naïve approach is to check information flow conditions whenever an explicit flow occurs.
 - Implicit flows complicate checking.
- **EXAMPLE:**
 - x and y are variables.
 - The requirement for certification for a particular statement “ $y \text{ op } x$ ” is that $\underline{x} \leq \underline{y}$.
 - The conditional statement
 if $x = 1$ then $y := a$;
causes a flow from x to y .
 - Suppose that when $x \neq 1$, $\underline{x} = \text{High}$ and $\underline{y} = \text{Low}$.
 - If flows were verified only when explicit, and $x \neq 1$, the implicit flow would not be checked.
 - Hence, the statement may be incorrectly certified as complying with the information flow policy.