ECS 235B Module 52
Execution Based
Information Flow Mechanisms
Execution-Based Mechanisms

• Detect and stop flows of information that violate policy
  • Done at run time, not compile time
• Obvious approach: check explicit flows
  • Problem: assume for security, \( x \leq y \)
    
    \[
    \text{if } x = 1 \text{ then } y := a; \]
  • When \( x \neq 1, \) \( x = \text{High}, y = \text{Low}, a = \text{Low}, \) appears okay—but implicit flow violates condition!
Fenton’s Data Mark Machine

• Each variable has an associated class
• Program counter (PC) has one too
• Idea: branches are assignments to PC, so you can treat implicit flows as explicit flows
• Stack-based machine, so everything done in terms of pushing onto and popping from a program stack
Instruction Description

• $skip$: instruction not executed

• $push(x, x)$: push variable $x$ and its security class $x$ onto program stack

• $pop(x, x)$ : pop top value and security class from program stack, assign them to variable $x$ and its security class $x$ respectively
Instructions

• $x := x + 1$ (increment)
  • Same as:
    ```
    if $PC \leq x$ then $x := x + 1$ else skip
    ```

• if $x = 0$ then goto $n$ else $x := x - 1$ (branch and save PC on stack)
  • Same as:
    ```
    if $x = 0$ then begin
      push$(PC, PC)$; $PC :=$ lub{$PC$, $x$}; $PC := n$;
    end else if $PC \leq x$ then
      $x := x - 1$
    else
      skip;
    ```
More Instructions

- \textbf{if'} \ x = 0 \ \textbf{then} \ \textbf{goto} \ n \ \textbf{else} \ x := x - 1 \ (\text{branch without saving PC on stack})

- Same as:

\begin{verbatim}
if x = 0 then
  if x \leq PC then PC := n else skip
else
  if PC \leq x then x := x - 1 else skip
\end{verbatim}
More Instructions

• **return** (go to just after last *if*)
  • Same as:
    \[
    \text{pop}(PC, \ PC);
    \]

• **halt** (stop)
  • Same as:
    \[
    \text{if program stack empty then halt}
    \]
  • Note stack empty to prevent user obtaining information from it after halting
Example Program

```plaintext
1  if x = 0 then goto 4 else x := x - 1
2  if z = 0 then goto 6 else z := z - 1
3  halt
4  z := z - 1
5  return
6  y := y - 1
7  return
```

Initially x = 0 or x = 1, y = 0, z = 0
Program copies value of x to y
## Example Execution

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>PC</th>
<th>stack</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Low</td>
<td>—</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Low</td>
<td>—</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>$z$</td>
<td>(3, Low)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>$z$</td>
<td>(3, Low)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>Low</td>
<td>—</td>
</tr>
</tbody>
</table>
Handling Errors

• Ignore statement that causes error, but continue execution
  • If aborted or a visible exception taken, user could deduce information
  • Means errors cannot be reported unless user has clearance at least equal to that of the information causing the error
Variable Classes

• Up to now, classes fixed
  • Check relationships on assignment, etc.

• Consider variable classes
  • Fenton’s Data Mark Machine does this for $PC$
  • On assignment of form $y := f(x_1, ..., x_n)$, $y$ changed to $\text{lub}\{x_1, ..., x_n\}$
  • Need to consider implicit flows, also
Example Program

(* Copy value from x to y. Initially, x is 0 or 1 *)
proc copy(x: integer class { x });
    var y: integer class { y };
var z: integer class variable { Low };
begin
    y := 0;
    z := 0;
    if x = 0 then z := 1;
    if z = 0 then y := 1;
end;

• z changes when z assigned to
• Assume y < x (that is, x strictly dominates y; they are not equal)
Analysis of Example

• $x = 0$
  • $z := 0$ sets $z$ to Low
  • if $x = 0$ then $z := 1$ sets $z$ to 1 and $z$ to $x$
  • So on exit, $y = 0$

• $x = 1$
  • $z := 0$ sets $z$ to Low
  • if $z = 0$ then $y := 1$ sets $y$ to 1 and checks that lub{Low, $z$} $\leq y$
  • So on exit, $y = 1$

• Information flowed from $x$ to $y$ even though $y < x$
Handling This (1)

• Fenton’s Data Mark Machine detects implicit flows violating certification rules
Handling This (2)

• Raise class of variables assigned to in conditionals even when branch not taken
• Also, verify information flow requirements even when branch not taken
• Example:
  • In if \( x = 0 \) then \( z := 1 \), \( z \) raised to \( x \) whether or not \( x = 0 \)
  • Certification check in next statement, that \( z \leq y \), fails, as \( z = x \) from previous statement, and \( y < x \)
Handling This (3)

• Change classes only when explicit flows occur, but all flows (implicit as well as explicit) force certification checks

• Example
  • When $x = 0$, first if sets $z$ to Low, then checks $x \leq z$
  • When $x = 1$, first if checks $x \leq z$
  • This holds if and only if $x = \text{Low}$
    • Not possible as $y < x = \text{Low}$ by assumption and there is no class that Low strictly dominates
Quiz

Should a statement that causes an error be ignored, and execution continue?

1. Yes; if the program is aborted or a visible exception is taken, the user could deduce information about values in the program

2. Yes; such a statement cannot be certified and so it must be ignored

3. No; the user must be informed lest they draw an incorrect conclusion about values in the program

4. No; the user’s clearance may allow them to see that an error occurred