Lecture 25
November 21, 2022
Semaphores

Use these constructs:

\[
\begin{align*}
\text{wait}(x): & \quad \text{if } x = 0 \text{ then block until } x > 0; \quad x := x - 1; \\
\text{signal}(x): & \quad x := x + 1;
\end{align*}
\]

- \( x \) 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*
  • shared(S): set of shared variables read
    • Idea: information flows out of variables in shared(S)
  • fglb(S): glb of assignment targets following S
  • So, requirement is shared(S) ≤ fglb(S)

• begin S₁; ... Sₙ end
  • All Sᵢ must be secure
  • For all i, shared(Sᵢ) ≤ fglb(Sᵢ)
Example

```
begin
  x := y + z;       (* S₁ *)
  wait(sem);        (* S₂ *)
  a := b * c − x;   (* S₃ *)
end
```

• Requirements:
  • lub{ y, z } ≤ x
  • lub{ b, c, x } ≤ a
  • sem ≤ a
    • Because fglb(S₂) = a and shared(S₂) = 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

```plaintext
while i < n do begin
    a[i] := item;     (* S₁ *)
    wait(sem);        (* S₂ *)
    i := i + 1;       (* S₃ *)
end
```

- Conditions for this to be secure:
  - Loop terminates, so this condition met
  - S₁ secure if lub{ i, item } ≤ a[i]
  - S₂ secure if sem ≤ i and sem ≤ a[i]
  - S₃ trivially secure
cobegin/coend

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

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

• Security requirement is both must hold
  • So this is secure if lub\{ $y$, $z$ \} $\leq x \land$ 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
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:
    $\text{if } PC \leq x \text{ then } x := x + 1 \text{ else } \text{skip}$

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

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

• Same as:

\[
\begin{align*}
\text{if } x &= 0 \text{ then } \\
&\quad \text{if } x \leq PC \text{ then } PC := n \text{ else } skip \\
\text{else } &\quad \text{if } PC \leq x \text{ then } x := x - 1 \text{ else } skip
\end{align*}
\]
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

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: Initial Setting

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>PC</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>
<td>—</td>
</tr>
</tbody>
</table>
Example Execution: Step 1

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>PC</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Low</td>
<td>—</td>
<td>Low ≤ x</td>
</tr>
</tbody>
</table>

```plaintext
if x = 0 then goto 4 else x := x - 1
```

```plaintext
if x = 0 then begin
    push(PC, PC); PC := lub(PC, x); PC := n;
end else if PC ≤ x then
    x := x - 1
else
    skip;
```
### Example Execution: Step 2

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>PC</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Low</td>
<td>—</td>
<td>$Low \leq x$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>$z$</td>
<td>(3, Low)</td>
<td>$PC \leq y$</td>
</tr>
</tbody>
</table>

**if** $z = 0$ **then** goto 6 **else** $z := z - 1$

```plaintext
if $z = 0$ then begin
    push($PC$, $PC$); $PC := \text{lub}\{PC, z\}$; $PC := n$;
end else if $PC \leq z$ then
    $z := z - 1$
else
    skip;
```
Example Execution: Step 3

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

y := y + 1

\[ \text{if } PC \leq y \text{ then } y := y + 1 \text{ else } \text{skip} \]
Example Execution: Step 4

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$z$</th>
<th>PC</th>
<th>$\underline{PC}$</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>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Low</td>
<td>—</td>
<td>Low ≤ $x$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>$z$</td>
<td>(3, Low)</td>
<td>$\underline{PC}$ ≤ $y$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>$z$</td>
<td>(3, Low)</td>
<td></td>
</tr>
</tbody>
</table>

return

```
pop(\underline{PC}, \underline{PC});
```
Example Execution: Step 5

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

**halt**

```
if program stack empty then halt
```
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 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 ≤ 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
Integrity Mechanisms

• The above also works with Biba, as it is mathematical dual of Bell-LaPadula

• All constraints are simply duals of confidentiality-based ones presented above
Example 1

For information flow of assignment statement:

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

the relation $\text{glb}\{ x_1, \ldots, x_n \} \geq y$ must hold

• Why? Because information flows from $x_1, \ldots, x_n$ to $y$, and under Biba, information must flow from a higher (or equal) class to a lower one.
Example 2

For information flow of conditional statement:

\[
\text{if } f(x_1, \ldots, x_n) \text{ then } S_1; \text{ else } S_2; \text{ end;}
\]

then the following must hold:

• \(S_1, S_2\) must satisfy integrity constraints

• \(\text{glb}\{ x_1, \ldots, x_n \} \geq \text{lub}\{y \mid y \text{ target of assignment in } S_1, S_2 \}\)
Example Information Flow Control Systems

• Privacy and Android Cell Phones
  • Analyzes data being sent from the phone
• Firewalls
Privacy and Android Cell Phones

• Many commercial apps use advertising libraries to monitor clicks, fetch ads, display them
  • So they send information, ostensibly to help tailor advertising to you

• Many apps ask to have full access to phone, data
  • This is because of complexity of permission structure of Android system

• Ads displayed with privileges of app
  • And if they use Javascript, that executes with those privileges
  • So if it has full access privilege, it can send contact lists, other information to others

• Information flow problem as information is flowing from phone to external party
Analyzing Android Flows

• Android based on Linux
  • App executables in bytecode format (Dalvik executables, or DEX) and run in Dalvik VM
  • Apps event driven
  • Apps use system libraries to do many of their functions
  • Binder subsystem controls interprocess communication

• Analysis uses 2 security levels, *untainted* and *tainted*
  • No categories, and *tainted* < *untainted*
TaintDroid: Checking Information Flows

• All objects tagged *tainted* or *untainted*
  • Interpreters, Binder augmented to handle tags

• Android native libraries trusted
  • Those communicating externally are *taint sinks*

• When untrusted app invokes a taint sink library, taint tag of data is recorded

• Taint tags assigned to external variables, library return values
  • These are assigned based on knowledge of what native code does

• Files have single taint tag, updated when file is written

• Database queries retrieve information, so tag determined by database query responder
TaintDroid: Checking Information Flows

- Information from phone sensor may be sensitive; if so, *tainted*
  - TaintDroid determines this from characteristics of information
- Experiment 1 (2010): selected 30 popular apps out of a set of 358 that required permission to access Internet, phone location, camera, or microphone; also could access cell phone information
  - 105 network connections accessed *tainted* data
  - 2 sent phone identification information to a server
  - 9 sent device identifiers to third parties, and 2 didn’t tell user
  - 15 sent location information to third parties, none told user
  - No false positives
TaintDroid: Checking Information Flows

• Experiment 2 (2012): revisited 18 out of the 30 apps (others did not run on current version of Android)
  • 3 still sent location information to third parties
  • 8 sent device identification information to third parties without consent
    • 3 of these did so in 2010 experiment
    • 5 were new
  • 2 new flows that could reveal tainted data
  • No false positives
Firewalls

- Host that mediates access to a network
  - Allows, disallows accesses based on configuration and type of access

- Example: block Conficker worm
  - Conficker connects to botnet, which can use system for many purposes
    - Spreads through a vulnerability in a particular network service
  - Firewall analyze packets using that service remotely, and look for Conficker and its variants
    - If found, packets discarded, and other actions may be taken
  - Conficker also generates list of host names, tried to contact botnets at those hosts
    - As set of domains known, firewall can also block outbound traffic to those hosts
Filtering Firewalls

- Access control based on attributes of packets and packet headers
  - Such as destination address, port numbers, options, etc.
  - Also called a *packet filtering firewall*
  - Does not control access based on content
  - Examples: routers, other infrastructure systems
Proxy

- Intermediate agent or server acting on behalf of endpoint without allowing a direct connection between the two endpoints
  - So each endpoint talks to proxy, thinking it is talking to other endpoint
  - Proxy decides whether to forward messages, and whether to alter them
Proxy Firewall

• Access control done with proxies
  • Usually bases access control on content as well as source, destination addresses, etc.
  • Also called an *applications level* or *application level firewall*
• Example: virus checking in electronic mail
  • Incoming mail goes to proxy firewall
  • Proxy firewall receives mail, scans it
  • If no virus, mail forwarded to destination
  • If virus, mail rejected or disinfected before forwarding
Example

- Want to scan incoming email for malware
- Firewall acts as recipient, gets packets making up message and reassembles the message
  - It then scans the message for malware
  - If none, message forwarded
  - If some found, mail is discarded (or some other appropriate action)
- As email reassembled at firewall by a mail agent acting on behalf of mail agent at destination, it’s a proxy firewall (application layer firewall)
Stateful Firewall

• Keeps track of the state of each connection
• Similar to a proxy firewall
  • No proxies involved, but this can examine contents of connections
  • Analyzes each packet, keeps track of state
  • When state indicates an attack, connection blocked or some other appropriate action taken