ECS 289M Lecture 17

May 10, 2006

Confinement Flow Model

- (I, O, *confine*, $\rightarrow$)
  - $I = (SC_I, \leq_I, join_I)$
  - O set of entities
  - $\rightarrow$: $O \times O$ with $(a, b) \in \rightarrow$ (written $a \rightarrow b$) iff information can flow from a to b
  - for $a \in O$, $confine(a) = (a_L, a_U) \in SC_I \times SC_I$ with $a_L \leq_I a_U$
    - Interpretation: for $a \in O$, if $x \leq_I a_U$, info can flow from x to a, and if $a_L \leq_I x$, info can flow from a to x
    - So a_L lowest classification of info allowed to flow out of a, and a_U highest classification of info allowed to flow into a

Assumptions, *etc*.

- Assumes: object can change security classes
  - So, variable can take on security class of its data
- Object *x* has security class $\underline{x}$ currently
- Note transitivity *not* required
- If information can flow from *a* to *b*, then *b* dominates *a* under ordering of policy *I*:

  $(\forall\ a, b \in O)[\ a \rightarrow b \Rightarrow a_L \leq_I b_U\ ]$

Example 1

- SC_I = { U, C, S, TS }, with U $\leq_I$ C, C $\leq_I$ S, and S $\leq_I$ TS
- *a*, *b*, $c \in O$
  - confine(*a*) = [C, C]
  - confine(*b*) = [S, S]
  - confine(*c*) = [TS, TS]
- Secure information flows: $a \rightarrow b$, $a \rightarrow c$, $b \rightarrow c$
  - As $a_L \leq_I b_U$, $a_L \leq_I c_U$, $b_L \leq_I c_U$
  - Transitivity holds

Example 2

- SC_I, $\leq_I$ as in Example 1
- $x, y, z \in O$
  - confine(x) = [C, C]
  - confine(y) = [S, S]
  - confine(z) = [C, TS]
- Secure information flows: $x \to y$, $x \to z$, $y \to z$, $z \to x$, $z \to y$
  - As $x_L \leq_I y_U$, $x_L \leq_I z_U$, $y_L \leq_I z_U$, $z_L \leq_I x_U$, $z_L \leq_I y_U$
  - Transitivity does not hold
    - $y \to z$ and $z \to x$, but $y \to x$ is false, because $y_L \leq_I x_U$ is false

Transitive Non-Lattice Policies

- Q = (S_Q, $\leq_Q$) is a *quasi-ordered set* when $\leq_Q$ is transitive and reflexive over S_Q
- How to handle information flow?
  - Define a partially ordered set containing quasi-ordered set
  - Add least upper bound, greatest lower bound to partially ordered set
  - It's a lattice, so apply lattice rules!

In Detail …

- $\forall x \in S_Q$: let $f(x) = \{ y \mid y \in S_Q \wedge y \le_Q x \}$
  - Define $S_{QP} = \{ f(x) \mid x \in S_Q \}$
  - Define $\le_{QP} = \{ (x, y) \mid x, y \in S_Q \wedge x \subseteq y \}$
    - S_{QP} partially ordered set under $\le_{QP}$
    - f preserves order, so $y \le_Q x$ iff $f(x) \le_{QP} f(y)$
- Add upper, lower bounds
  - $S_{QP}' = S_{QP} \cup \{ S_Q, \varnothing \}$
  - Upper bound $ub(x, y) = \{ z \mid z \in S_{QP} \wedge x \subseteq z \wedge y \subseteq z \}$
  - Least upper bound $lub(x, y) = \cap ub(x, y)$
    - Lower bound, greatest lower bound defined analogously

And the Policy Is …

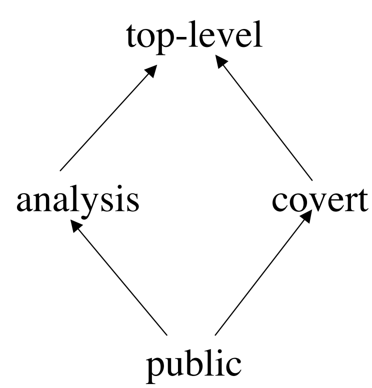
- Now $(S_{QP}', \le_{QP})$ is lattice
- Information flow policy on quasi-ordered set emulates that of this lattice!

Nontransitive Flow Policies

- Government agency information flow policy (on next slide)
- Entities public relations officers PRO, analysts A, spymasters S
  – *confine*(PRO) = { public, analysis }
  – *confine*(A) = { analysis, top-level }
  – *confine*(S) = { covert, top-level }

Information Flow

- By confinement flow model:
  – PRO ≤ A, A ≤ PRO
  – PRO ≤ S
  – A ≤ S, S ≤ A
- Data *cannot* flow to public relations officers; not transitive
  – S ≤ A, A ≤ PRO
  – S ≤ PRO is *false*



Transforming Into Lattice

- Rough idea: apply a special mapping to generate a subset of the power set of the set of classes
  - Done so this set is partially ordered
  - Means it can be transformed into a lattice
- Can show this mapping preserves ordering relation
  - So it preserves non-orderings and non-transitivity of elements corresponding to those of original set

Dual Mapping

- $R = (SC_R, \leq_R, join_R)$ reflexive info flow policy
- $P = (S_P, \leq_P)$ ordered set
  - Define *dual mapping* functions l_R, h_R: $SC_R \rightarrow S_P$
    - $l_R(x) = \{ x \}$
    - $h_R(x) = \{ y \mid y \in SC_R \wedge y \leq_R x \}$
  - S_P contains subsets of SC_R; $\leq_P$ subset relation
  - Dual mapping function *order preserving* iff

$$(\forall a, b \in SC_R)[\, a \leq_R b \Leftrightarrow l_R(a) \leq_P h_R(b) \,]$$

Theorem

Dual mapping from reflexive info flow policy R to ordered set P order-preserving

Proof sketch: all notation as before

($\Rightarrow$) Let $a \leq_R b$. Then $a \in l_R(a)$, $a \in h_R(b)$, so $l_R(a) \subseteq h_R(b)$, or $l_R(a) \leq_P h_R(b)$

($\Leftarrow$) Let $l_R(a) \leq_P h_R(b)$. Then $l_R(a) \subseteq h_R(b)$. But $l_R(a) = \{ a \}$, so $a \in h_R(b)$, giving $a \leq_R b$

Info Flow Requirements

- Interpretation: let *confine*(x) = $\{ \underline{x}_L, \underline{x}_U \}$, consider class $\underline{y}$
  - Information can flow from x to element of $\underline{y}$ iff $\underline{x}_L \leq_R \underline{y}$, or $l_R(\underline{x}_L) \subseteq h_R(\underline{y})$
  - Information can flow from element of $\underline{y}$ to x iff y $\leq_R \underline{x}_U$, or $l_R(\underline{y}) \subseteq h_R(\underline{x}_U)$

Revisit Government Example

- Information flow policy is *R*
- Flow relationships among classes are:

public $\le_R$ public
public $\le_R$ analysis
analysis $\le_R$ analysis
public $\le_R$ covert
covert $\le_R$ covert
public $\le_R$ top-level
covert $\le_R$ top-level
analysis $\le_R$ top-level
top-level $\le_R$ top-level

Dual Mapping of *R*

- Elements l_R, h_R:

l_R(public) = { public }
h_R(public) = { public }
l_R(analysis) = { analysis }
h_R(analysis) = { public, analysis }
l_R(covert) = { covert }
h_R(covert) = { public, covert }
l_R(top-level) = { top-level }
h_R(top-level) = { public, analysis, covert, top-level }

confine

- Let p be entity of type PRO, a of type A, s of type S
- In terms of P (not R), we get:
  - *confine*(p) = [{ public }, { public, analysis }]
  - *confine*(a) = [{ analysis },
    { public, analysis, covert, top-level }]
  - *confine*(s) = [{ covert },
    { public, analysis, covert, top-level }]

And the Flow Relations Are …

- $p \rightarrow a$ as $l_R(p) \subseteq h_R(a)$
  - $l_R(p)$ = { public }
  - $h_R(a)$ = { public, analysis, covert, top-level }
- Similarly: $a \rightarrow p$, $p \rightarrow s$, $a \rightarrow s$, $s \rightarrow a$
- ***But*** $s \rightarrow p$ ***is false*** as $l_R(s) \not\subseteq h_R(p)$
  - $l_R(s)$ = { covert }
  - $h_R(p)$ = { public, analysis }

Analysis

- $(S_P, \leq_P)$ is a lattice, so it can be analyzed like a lattice policy
- Dual mapping preserves ordering, hence non-ordering and non-transitivity, of original policy
  - So results of analysis of $(S_P, \leq_P)$ can be mapped back into $(SC_R, \leq_R, join_R)$

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;
```

- Info flows from *x* and *a* to *y*, or from *x* and *b* to *y*
- Certified only if $\underline{x} \le \underline{y}$ and $\underline{a} \le \underline{y}$ and $\underline{b} \le \underline{y}$
  - Note flows for *both* branches must be true unless compiler can determine that one branch will *never* be taken

Declarations

- Notation:

```
x: int class { A, B }
```

  means *x* is an integer variable with security class at least *lub*{ A, B }, so *lub*{ A, B } $\le \underline{x}$
- Distinguished classes *Low*, *High*
  - Constants are always *Low*

Input Parameters

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

i_p: ***type* class** { $\mathtt{i_p}$ }

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: ***type* class** { r_1, …, r_n }

where r_i is class of *i*th input or input/output argument

Example

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

- Require <u>*x*</u> ≤ <u>*out*</u> and <u>*out*</u> ≤ <u>*out*</u>

Array Elements

- Information flowing out:

  ```
  … := a[i]
  ```

  Value of *i*, *a*[*i*] both affect result, so class is lub{ <u>*a*[*i*]</u>, <u>*i*</u> }
- Information flowing in:

  ```
  a[i] := …
  ```

- Only value of *a*[*i*] affected, so class is <u>*a*[*i*]</u>

Assignment Statements

x := *y* + *z*;

- Information flows from *y*, *z* to *x*, so this requires lub{ $\underline{y}$, $\underline{z}$ } ≤ $\underline{x}$

More generally:

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

- the relation lub{ $\underline{x}_1$, …, x_n } ≤ $\underline{y}$ must hold

Compound Statements

x := *y* + *z*; *a* := *b* * *c* – *x*;

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

More generally:

$S_1; \ldots S_n;$

- Each individual S_i must be secure

Conditional Statements

```
if x + y < z then a := b else d := b * c – x; end
```

- The statement executed reveals information about *x*, *y*, *z*, so lub{ $\underline{x}$, $\underline{y}$, $\underline{z}$ } ≤ glb{ $\underline{a}$, $\underline{d}$ }

More generally:

```
if f(x1, …, xn) then S1 else S2; end
```

- S_1, S_2 must be secure
- lub{ $\underline{x}_1$, …, $\underline{x}_n$ } ≤ glb{$\underline{y}$ | *y* 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(x1, …, xn) do S;
```

- Loop must terminate;
- *S* must be secure
- lub{ $\underline{x}_1$, …, $\underline{x}_n$ } ≤ glb{$\underline{y}$ | *y* target of assignment in *S* }

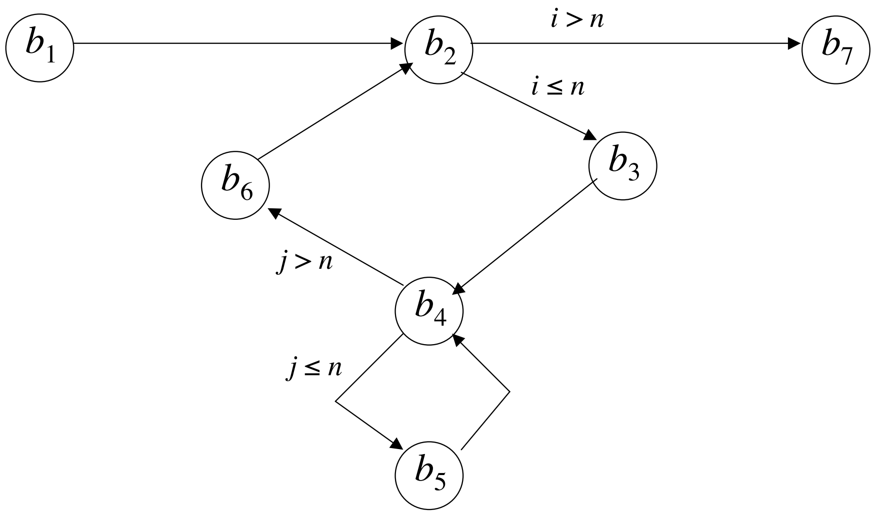
Goto Statements

- 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

```
proc tm(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 {i};
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][i] := x[i][j]; j := j + 1; goto L4;
b6 L6:   i := i + 1; goto L2;
b7 L7:
end;
```

Flow of Control



IFDs

- 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:
  - 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

Requirements

- B_i is set of basic blocks along an execution path from b_i to IFD(b_i)
  - Analogous to statements in conditional statement
- x_{i1}, …, x_{in} 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{ $\underline{x}_{i1}$, …, $\underline{x}_{in}$ } ≤ glb{ $\underline{y}$ | y target of assignment in B_i }

Example of Requirements

- Within each basic block:

  b_1: *Low* ≤ $\underline{i}$ b_3: *Low* ≤ $\underline{j}$ b_6: lub{ *Low*, $\underline{i}$ } ≤ $\underline{i}$

  b_5: lub{ $\underline{x[i][j]}$, $\underline{i}$, $\underline{j}$ } ≤ $\underline{y[j][i]}$ }; lub{ *Low*, $\underline{j}$ } ≤ $\underline{j}$
  - Combining, lub{ $\underline{x[i][j]}$, $\underline{i}$, $\underline{j}$ } ≤ $\underline{y[j][i]}$ }
  - From declarations, true when lub{ $\underline{x}$, $\underline{i}$ } ≤ $\underline{y}$
- $B_2 = \{b_3, b_4, b_5, b_6\}$
  - Assignments to *i*, *j*, *y*[*j*][*i*]; conditional is *i* ≤ 10
  - Requires $\underline{i}$ ≤ glb{ $\underline{i}$, $\underline{j}$, $\underline{y[j][i]}$ }
  - From declarations, true when $\underline{i}$ ≤ $\underline{y}$

Example (continued)

- $B_4 = \{ b_5 \}$
  - Assignments to *j*, *y*[*j*][*i*]; conditional is *j* ≤ 10
  - Requires $\underline{j}$ ≤ glb{ $\underline{j}$, $\underline{y[j][i]}$ }
  - From declarations, means $\underline{i}$ ≤ $\underline{y}$
- Result:
  - Combine lub{ $\underline{x}$, $\underline{i}$ } ≤ $\underline{y}$; $\underline{i}$ ≤ $\underline{y}$; $\underline{i}$ ≤ $\underline{y}$
  - Requirement is lub{ $\underline{x}$, $\underline{i}$ } ≤ $\underline{y}$