February 27, 2014

- Information flow
- Information flow policies
  - Non-transitive
  - Transitive non-lattice
- Compiler-based mechanisms
- Execution-based mechanisms

Entropy and Information Flow

- Idea: info flows from x to y as a result of a sequence of commands c if you can deduce information about x before c from the value in y after c
- Formally:
  - s time before execution of c, t time after
  - $H(x_s \mid y_t) < H(x_s \mid y_s)$
  - If no y at time s, then $H(x_s \mid y_t) < H(x_s)$

Example 1

- Command is $x := y + z$; where:
  - $0 \le y \le 7$, equal probability
  - $z = 1$ with prob. 1/2, $z = 2$ or 3 with prob. 1/4 each
- s state before command executed; t, after; so
  - $H(y_s) = H(y_t) = -8(1/8) \lg (1/8) = 3$
  - $H(z_s) = H(z_t) = -(1/2) \lg (1/2) - 2(1/4) \lg (1/4) = 1.5$
- If you know x_t, y_s can have at most 3 values, so $H(y_s \mid x_t) = -3(1/3) \lg (1/3) = \lg 3$

Example 2

- Command is
  - **if** $x = 1$ **then** $y := 0$ **else** $y := 1$;

  where:
  - x, y equally likely to be either 0 or 1
- $H(x_s) = 1$ as x can be either 0 or 1 with equal probability
- $H(x_s \mid y_t) = 0$ as if $y_t = 1$ then $x_s = 0$ and vice versa
  - Thus, $H(x_s \mid y_t) = 0 < 1 = H(x_s)$
- So information flowed from x to y

Implicit Flow of Information

- Information flows from x to y without an *explicit* assignment of the form $y := f(x)$
  - $f(x)$ an arithmetic expression with variable x
- Example from previous slide:
  - **if** $x = 1$ **then** $y := 0$
    else $y := 1$;
- So must look for implicit flows of information to analyze program

Notation

- $\underline{x}$ means class of x
  - In Bell-LaPadula based system, same as "label of security compartment to which x belongs"
- $\underline{x} \le \underline{y}$ means "information can flow from an element in class of x to an element in class of y"
  - Or, "information with a label placing it in class $\underline{x}$ can flow into class $\underline{y}$"

Information Flow Policies

Information flow policies are usually:

- reflexive
  - So information can flow freely among members of a single class
- transitive
  - So if information can flow from class 1 to class 2, and from class 2 to class 3, then information can flow from class 1 to class 3

Non-Transitive Policies

- Betty is a confident of Anne
- Cathy is a confident of Betty
  - With transitivity, information flows from Anne to Betty to Cathy
- Anne confides to Betty she is having an affair with Cathy's spouse
  - Transitivity undesirable in this case, probably

Transitive Non-Lattice Policies

- 2 faculty members co-PIs on a grant
  - Equal authority; neither can overrule the other
- Grad students report to faculty members
- Undergrads report to grad students
- Information flow relation is:
  - Reflexive and transitive
- But some elements (people) have no "least upper bound" element
  - What is it for the faculty members?

Confidentiality Policy Model

- Lattice model fails in previous 2 cases
- Generalize: policy $I = (SC_I, \leq_I, join_I)$:
  - SC_I set of security classes
  - $\leq_I$ ordering relation on elements of SC_I
  - $join_I$ function to combine two elements of SC_I
- Example: Bell-LaPadula Model
  - SC_I set of security compartments
  - $\leq_I$ ordering relation *dom*
  - $join_I$ function *lub*

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

- $SC_I = \{ \text{U}, \text{C}, \text{S}, \text{TS} \}$, with $\text{U} \le_I \text{C}$, $\text{C} \le_I \text{S}$, and $\text{S} \le_I \text{TS}$
- $a, b, c \in O$
  - confine(a) = [C, C]
  - confine(b) = [S, S]
  - confine(c) = [TS, TS]
- Secure information flows: $a \to b$, $a \to c$, $b \to c$
  - As $a_L \le_I b_U$, $a_L \le_I c_U$, $b_L \le_I c_U$
  - 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: $x \rightarrow y$, $x \rightarrow z$, $y \rightarrow z$, $z \rightarrow x$, $z \rightarrow y$
  - 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
    - $y \rightarrow z$ and $z \rightarrow x$, but $y \rightarrow x$ 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 $f(x) = \{ y \mid y \in S_Q \wedge y \leq_Q x \}$
  - Define $S_{QP} = \{ f(x) \mid x \in S_Q \}$
  - Define $\leq_{QP} = \{ (x, y) \mid x, y \in S_{QP} \wedge x \subseteq y \}$
    - S_{QP} partially ordered set under $\leq_{QP}$
    - f preserves order, so $y \leq_Q x$ iff $f(x) \leq_{QP} f(y)$
- 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 …

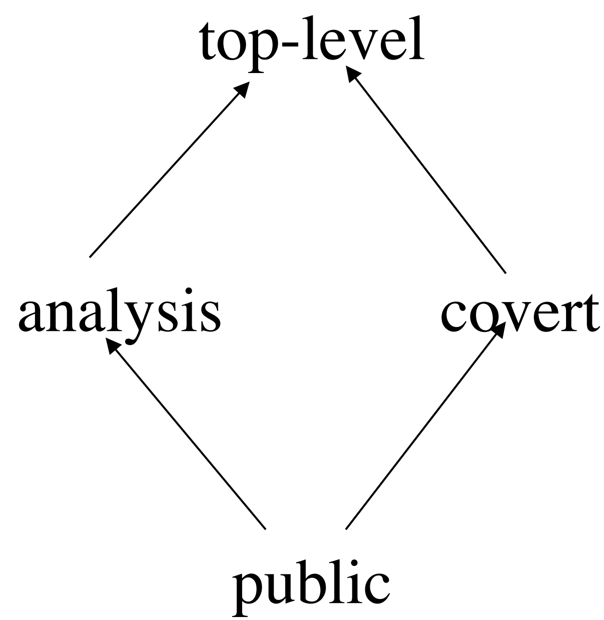
- Now $(S_{QP}', \leq_{QP})$ is lattice
- Information flow policy on quasi-ordered set emulates that of this lattice!

Non-Transitive 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 $\mathit{confine}(x) = \{ \underline{x}_L, \underline{x}_U \}$, 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 $\underline{y} \leq_R \underline{x}_U$, 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 $\leq_R$ public	
public $\leq_R$ analysis	analysis $\leq_R$ analysis
public $\leq_R$ covert	covert $\leq_R$ covert
public $\leq_R$ top-level	covert $\leq_R$ top-level
analysis $\leq_R$ top-level	top-level $\leq_R$ 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 an information flow policy if the flows in the 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 $\underline{x} \leq \underline{y}$ and $\underline{a} \leq \underline{y}$ and $\underline{b} \leq \underline{y}$
  - 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 $lub\{ A, B \} \leq \underline{x}$
- 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

$$i_p\text{: } \textbf{\textit{type}}\ \textbf{class}\ \{\ i_p\ \}$$

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:

  $\boldsymbol{o_p}$**: *type* class {** r_1**, ...,** r_n **}**

  where r_i is class of ith input or input/output argument

Example

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

- Require $\underline{x} \leq \underline{out}$ and $\underline{out} \leq \underline{out}$

Array Elements

- Information flowing out:

  `... := a[i]`

  Value of i, $a[i]$ both affect result, so class is lub{ $\underline{a[i]}$, $\underline{i}$ }
- Information flowing in:

  `a[i] := ...`
- Only value of $a[i]$ affected, so class is $\underline{a[i]}$

Assignment Statements

$$\texttt{x := } \underline{y} \texttt{ + z;}$$

- Information flows from y, z to x, so this requires $lub(\underline{y}, \underline{z}) \le \underline{x}$

More generally:

$$\texttt{y := f(}x_1\texttt{, ..., }x_n\texttt{)}$$

- the relation $lub(\underline{x}_1, \ldots, x_n) \le \underline{y}$ must hold

Compound Statements

`x := y + z; a := b * c – x;`

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

More generally:

$$S_1; \; ...; \; S_n;$$

- Each individual S_i must be secure

Conditional Statements

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

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

More generally:

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

- S_1, S_2 must be secure
- $lub(\underline{x}_1, \ldots, \underline{x}_n) \le$
  $glb(\underline{y} \mid y \text{ target of assignment in } S_1, S_2)$

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:

$$\texttt{while } f(x_1, \ldots, x_n) \texttt{ do } S;$$

- Loop must terminate;
- S must be secure
- $lub(\underline{x}_1, \ldots, \underline{x}_n) \le glb(\underline{y} \mid y \text{ target of assignment in } S)$

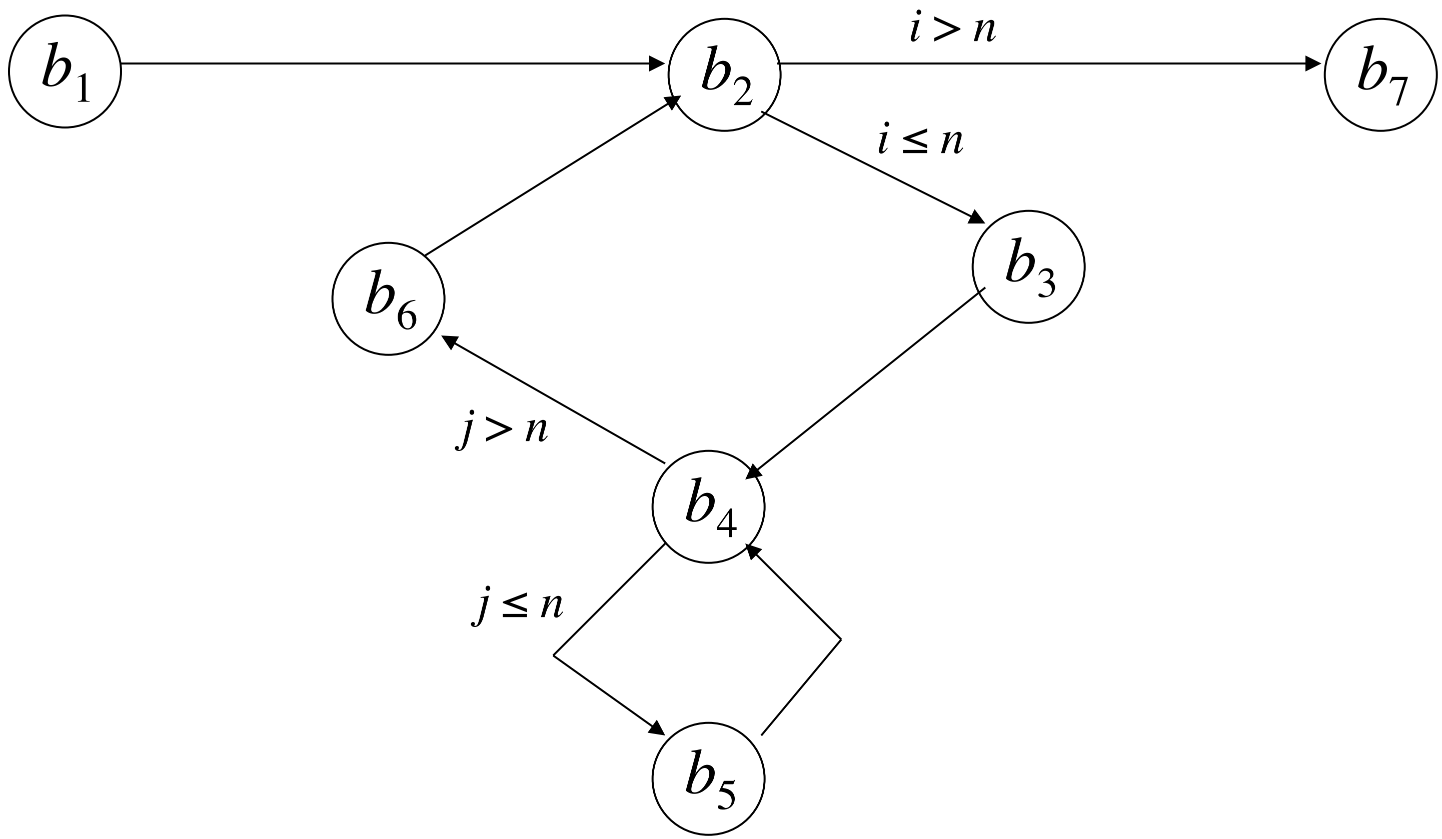
Goto Statements

- 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

```
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 then 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 a basic block *b* (written IFD(*b*)) is the 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 the set of basic blocks along an execution path from b_i to IFD(b_i)
  - Analogous to statements in conditional statement
- $x_{i1}, \ldots, x_{in}$ variables in expression selecting which execution path containing basic blocks in B_i used
  - Analogous to conditional expression
- Requirements for being secure:
  - All statements in each basic blocks are secure
  - $lub(\underline{x}_{i1}, \ldots, \underline{x}_{in}) \leq glb\{ \underline{y} \mid y \text{ target of assignment in } B_i \}$

Example of Requirements

- Within each basic block:

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

  b_5: $lub(\underline{x[i][j]}, \underline{i}, \underline{j}) \le \underline{y[j][i]}$; $lub(Low, \underline{j}) \le \underline{j}$

  – Combining, $lub(\underline{x[i][j]}, \underline{i}, \underline{j}) \le \underline{y[j][i]}$

  – From declarations, true when $lub(\underline{x}, \underline{i}) \le \underline{y}$

- $B_2 = \{b_3, b_4, b_5, b_6\}$
  – Assignments to $i, j, y[j][i]$; conditional is $i \le 10$

  – Requires $\underline{i} \le glb(\underline{i}, \underline{j}, \underline{y[j][i]})$

  – From declarations, true when $\underline{i} \le \underline{y}$

Example (continued)

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

Procedure Calls

```
tm(a, b);
```

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

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

More generally:

$\texttt{proc } pn(i_1, \ldots, i_m\texttt{: int; var } o_1, \ldots, o_n\texttt{: int)}$
$\texttt{begin } S \texttt{ end;}$

- S must be secure
- For all j and k, if $\underline{i_j} \leq \underline{o_k}$, then $\underline{x_j} \leq \underline{y_k}$
- For all j and k, if $\underline{o_j} \leq \underline{o_k}$, then $\underline{y_j} \leq \underline{y_k}$