Confinement Flow Model

- $(I, O, confine, \to)$
  - $I = (SC_I, \leq_i, \text{join}_i)$
  - $O$ 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 $x$ currently
- Note transitivity *not* required
- If information can flow from $a$ to $b$, then $b$ dominates $a$ under ordering of policy $I$: $\forall a, b \in O \left[ a \rightarrow b \Rightarrow a_L \leq_I b_U \right]$

Example 1

- $SC_I = \{ U, C, S, TS \}$, with $U \leq_I C$, $C \leq_I S$, and $S \leq_I TS$
- $a, b, c \in O$
  - $\text{confine}(a) = [C, C]$
  - $\text{confine}(b) = [S, S]$
  - $\text{confine}(c) = [TS, TS]$
- 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
- \( x, y, z \in O \)
  - \( \text{confine}(x) = [C, C] \)
  - \( \text{confine}(y) = [S, S] \)
  - \( \text{confine}(z) = [C, TS] \)
- 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 \textit{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 $f(x) = \{ y \mid y \in S_Q \land y \leq_Q x \}$
  - Define $S_{QP} = \{ f(x) \mid x \in S_Q \}$
  - Define $\leq_{QP} = \{ (x, y) \mid x, y \in S_Q \land x \subseteq y \}$
    • $S_{QP}$ 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
  - $S_{QP}' = S_{QP} \cup \{ S_Q, \emptyset \}$
  - Upper bound $ub(x, y) = \{ z \mid z \in S_{QP} \land x \subseteq z \land y \subseteq z \}$
  - 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!
Nontransitive Flow Policies

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

Information Flow

- By confinement flow model:
  - \( \text{PRO} \leq \text{A}, \text{A} \leq \text{PRO} \)
  - \( \text{PRO} \leq \text{S} \)
  - \( \text{A} \leq \text{S}, \text{S} \leq \text{A} \)
- Data cannot flow to public relations officers; not transitive
  - \( \text{S} \leq \text{A}, \text{A} \leq \text{PRO} \)
  - \( \text{S} \leq \text{PRO} \text{ is false} \)
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 $I_R, h_R$: $SC_R \rightarrow S_P$
    - $I_R(x) = \{ x \}$
    - $h_R(x) = \{ y \mid y \in SC_R \land y \leq_R x \}$
  - $S_P$ contains subsets of $SC_R$; $\leq_P$ subset relation
  - Dual mapping function order preserving iff
    $(\forall a, b \in SC_R)[ a \leq_R b \Leftrightarrow I_R(a) \leq_P h_R(b) ]$
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 $a \leq_R b$
Revisit Government Example

- Information flow policy is $R$
- Flow relationships among classes are:
  
  \[
  \begin{align*}
  \text{public} & \leq_R \text{public} \\
  \text{public} & \leq_R \text{analysis} & \text{analysis} & \leq_R \text{analysis} \\
  \text{public} & \leq_R \text{covert} & \text{covert} & \leq_R \text{covert} \\
  \text{public} & \leq_R \text{top-level} & \text{covert} & \leq_R \text{top-level} \\
  \text{analysis} & \leq_R \text{top-level} & \text{top-level} & \leq_R \text{top-level}
  \end{align*}
  \]

Dual Mapping of $R$

- Elements $I_R$, $h_R$:
  \[
  \begin{align*}
  I_R(\text{public}) & = \{ \text{public} \} \\
  h_R(\text{public}) & = \{ \text{public} \} \\
  I_R(\text{analysis}) & = \{ \text{analysis} \} \\
  h_R(\text{analysis}) & = \{ \text{public}, \text{analysis} \} \\
  I_R(\text{covert}) & = \{ \text{covert} \} \\
  h_R(\text{covert}) & = \{ \text{public}, \text{covert} \} \\
  I_R(\text{top-level}) & = \{ \text{top-level} \} \\
  h_R(\text{top-level}) & = \{ \text{public}, \text{analysis}, \text{covert}, \text{top-level} \}
  \end{align*}
  \]
confine

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

And the Flow Relations Are ...

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

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 \( x \leq y \) and \( a \leq y \) and \( b \leq y \)
  – Note flows for both branches must be true unless compiler can determine that one branch will never be taken

Declarations

• Notation:

\[
x: \text{int class } \{ A, B \}
\]

means \( x \) is an integer variable with security class at least \( \text{lub}\{ A, B \} \), so \( \text{lub}\{ A, B \} \leq x \)

• Distinguished classes \( \text{Low, High} \)
  – Constants are always \( \text{Low} \)
Input Parameters

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

\[ i_p : \text{type class} \{ i_p \} \]

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:

\[ o_p : \text{type class} \{ r_1, \ldots, r_n \} \]

where \( r_i \) is class of \( i \)th input or input/output argument
Example

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

Array Elements

• Information flowing out:
  ... := a[i]
  Value of i, a[i] both affect result, so class is lub{ a[i], i }
• Information flowing in:
  a[i] := ...
• Only value of a[i] affected, so class is a[i]
Assignment Statements

\[ x := y + z; \]

- Information flows from \( y, z \) to \( x \), so this requires \( \text{lub}\{ y, z \} \leq x \)

More generally:

\[ y := f(x_1, \ldots, x_n) \]

- the relation \( \text{lub}\{ x_1, \ldots, x_n \} \leq y \) must hold

Compound Statements

\[ x := y + z; \ a := b \times c - x; \]

- First statement: \( \text{lub}\{ y, z \} \leq x \)
- Second statement: \( \text{lub}\{ b, c, x \} \leq a \)
- 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 \( d := b \cdot c - x \); end

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

More generally:
if \( f(x_1, \ldots, x_n) \) then \( S_1 \) else \( S_2 \); end

- \( S_1, S_2 \) must be secure
- \( \text{lub}\{ x_1, \ldots, x_n \} \leq \text{glb}\{ y \mid y \text{ target of assignment in } S_1, S_2 \} \)

**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:
while \( f(x_1, \ldots, x_n) \) do \( S \);

- Loop must terminate;
- \( S \) must be secure
- \( \text{lub}\{ x_1, \ldots, x_n \} \leq \text{glb}\{ y \mid y \text{ target of assignment in } S \} \)
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

```plaintext
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(b₁) = b₂ one path
  – IFD(b₂) = b₇ → b₇ or b₂ → b₃ → b₆ → b₂ → b₇
  – IFD(b₃) = b₄ one path
  – IFD(b₄) = b₆ → b₆ or b₄ → b₅ → b₆
  – IFD(b₅) = b₄ one path
  – IFD(b₆) = b₂ 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
• $x_{i1}, \ldots, x_{in}$ 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{ $x_{i1}, \ldots, x_{in}$ } $\leq$
    \begin{align*}
      \text{glb} \{ y \mid y \text{ target of assignment in } B_i \}
    \end{align*}
Example of Requirements

• Within each basic block:
  $b_1$: $\text{Low} \leq i$
  $b_2$: $\text{Low} \leq j$
  $b_3$: lub{ $\text{Low}, i$ } $\leq i$
  $b_4$: lub{ $x[i][i], i$ } $\leq y[i][i]$; lub{ $\text{Low}, i$ } $\leq i$
  - Combining, lub{ $x[i][i], i$ } $\leq y[i][i]$}
  - From declarations, true when lub{ $x, i$ } $\leq y$

• $B_2 = \{ b_3, b_4, b_5, b_6 \}$
  - Assignments to $i, j, y[i][i]$; conditional is $i \leq 10$
  - Requires $i \leq \text{glb} \{ i, j, y[i][i] \}$
  - From declarations, true when $i \leq y$

Example (continued)

• $B_4 = \{ b_5 \}$
  - Assignments to $j, y[j][i]$; conditional is $j \leq 10$
  - Requires $j \leq \text{glb} \{ i, y[i][i] \}$
  - From declarations, means $i \leq y$

• Result:
  - Combine lub{ $x, i$ } $\leq y$; $i \leq y$; $i \leq y$
  - Requirement is lub{ $x, i$ } $\leq y$