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:
  - $0 \leq y \leq 7$, 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
  - $H(y_s) = H(y_t) = -8(1/8) \log_2 (1/8) = 3$
  - $H(z_s) = H(z_t) = -(1/2) \log_2 (1/2) - 2(1/4) \log_2 (1/4) = 1.5$

- If you know $x_t$, $y_s$ can have at most 3 values, so
  $H(y_s \mid x_t) = -3(1/3) \log_2 (1/3) = \log_2 3$
Example 2

• Command is
  – \textbf{if} \( x = 1 \) \textbf{then} \( y := 0 \) \textbf{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

• \( x \) means class of \( x \)
  – In Bell-LaPadula based system, same as “label of security compartment to which \( x \) belongs”

• \( x \leq y \) 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 \( x \) can flow into class \( y \)”
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

- \((I, O, \text{confine}, \rightarrow)\)
  - \(I = (SC_I, \leq_I, \text{join}_I)\)
  - \(O\) 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 \(a \in O\), \(\text{confine}(a) = (a_L, a_U) \in SC_I \times SC_I\) 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 $I$:
  \[(\forall a, b \in O)[ a \rightarrow b \Rightarrow a_L \leq_I b_U ]\]
Example 1

- $SC_I = \{ U, C, S, TS \}$, with $U \leq_I C$, $C \leq_I S$, and $S \leq_I TS$
- $a, b, c \in O$
  - $\text{confine}(a) = [ C, C ]$
  - $\text{confine}(b) = [ S, S ]$
  - $\text{confine}(c) = [ TS, TS ]$
- 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 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 …

• \( \forall x \in S_Q: \text{let } f(x) = \{ y \mid y \in S_Q \land y \leq_Q x \} \)
  - Define \( S_{QP} = \{ f(x) \mid x \in S_Q \} \)
  - Define \( \leq_{QP} = \{ (x, y) \mid x, y \in S_{QP} \land x \subseteq y \} \)
    - \( S_{QP} \) partially ordered set under \( \leq_{QP} \)
    - \( f \) preserves order, so \( y \leq_Q x \text{ iff } f(x) \leq_{QP} f(y) \)

• 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 \( lub(x, y) = \bigcap ub(x, y) \)
    - 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
  – \( \text{confine}(\text{PRO}) = \{ \text{public, analysis} \} \)
  – \( \text{confine}(\text{A}) = \{ \text{analysis, top-level} \} \)
  – \( \text{confine}(\text{S}) = \{ \text{covert, top-level} \} \)
Information Flow

• By confinement flow model:
  – $\text{PRO} \leq A$, $A \leq \text{PRO}$
  – $\text{PRO} \leq S$
  – $A \leq S$, $S \leq A$

• Data \textit{cannot} flow to public relations officers; not transitive
  – $S \leq A$, $A \leq \text{PRO}$
  – $S \leq \text{PRO}$ is \textit{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

- $R = (SC_R, \leq_R, \text{join}_R)$ reflexive info flow policy
- $P = (S_P, \leq_P)$ ordered set
  - Define dual mapping functions $l_R, h_R: SC_R \rightarrow S_P$
    - $l_R(x) = \{ x \}$
    - $h_R(x) = \{ y \mid y \in SC_R \land y \leq_R x \}$
  - $S_P$ contains subsets of $SC_R$; $\leq_P$ subset relation
  - Dual mapping function order preserving iff
    \[(\forall a, b \in SC_R)[ a \leq_R b \iff l_R(a) \leq_P h_R(b) ]\]
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 \( \text{confine}(x) = \{ x_L, x_U \} \), consider class \( y \)

  – Information can flow from \( x \) to element of \( y \) iff \( x_L \leq_R y \), or \( l_R(x_L) \subseteq h_R(y) \)

  – Information can flow from element of \( y \) to \( x \) iff \( y \leq_R x_U \), or \( l_R(y) \subseteq h_R(x_U) \)
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 $l_R$, $h_R$:
  
  \[ l_R(\text{public}) = \{ \text{public} \} \]
  \[ h_R(\text{public}) = \{ \text{public} \} \]
  \[ l_R(\text{analysis}) = \{ \text{analysis} \} \]
  \[ h_R(\text{analysis}) = \{ \text{public, analysis} \} \]
  \[ l_R(\text{covert}) = \{ \text{covert} \} \]
  \[ h_R(\text{covert}) = \{ \text{public, covert} \} \]
  \[ l_R(\text{top-level}) = \{ \text{top-level} \} \]
  \[ h_R(\text{top-level}) = \{ \text{public, analysis, covert, top-level} \} \]
confine

• Let $p$ be entity of type PRO, $a$ of type A, $s$ of type S

• In terms of $P$ (not $R$), we get:
  
  $confine(p) = [ \{ \text{public} \}, \{ \text{public, analysis} \} ]$
  $confine(a) = [ \{ \text{analysis} \}, \{ \text{public, analysis, covert, top-level} \} ]$
  $confine(s) = [ \{ \text{covert} \}, \{ \text{public, analysis, covert, top-level} \} ]$
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 $l_R(s) \not\subseteq h_R(p)$
  - $l_R(s) = \{ \text{covert} \}$
  - $h_R(p) = \{ \text{public, analysis} \}$
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 \((SC_R, \leq_R, join_R)\)
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 $x \leq y$ 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 \( Low, High \)
  – Constants are always \( Low \)
Input Parameters

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

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

\[
\text{proc } \text{sum}(x: \text{ int class } \{ \text{ A } \});
\]
\[
\quad \text{var } \text{out}: \text{ int class } \{ \text{ A, B } \});
\]
\[
\text{begin}
\]
\[
\quad \text{out} := \text{out} + x;
\]
\[
\text{end};
\]
\[
\cdot \text{ Require } x \leq \text{out and out} \leq \text{out}
\]
Array Elements

• Information flowing out:
  \[ \ldots := a[i] \]

Value of \( i \), \( a[i] \) both affect result, so class is \( \text{lub}\{ a[i], i \} \)

• Information flowing in:
  \[ a[i] := \ldots \]

• 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

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

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

More generally:

\[ S_1; \ldots; S_n; \]

- Each individual \( S_i \) must be secure
Conditional Statements

\[
\text{if } x + y < z \text{ then } a := b \ \text{else} \ d := b \times c - x;
\]

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

More generally:

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

- \(S_1, S_2\) 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

\begin{verbatim}
while i < n do begin
  a[i] := b[i]; i := i + 1; end
\end{verbatim}

- Same ideas as for “if”, but must terminate

More generally:

\begin{verbatim}
while f(x_1, \ldots, x_n) do S;
\end{verbatim}

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

\[
\text{proc } \text{tm}(x: \text{array}[1..10][1..10] \text{ of int class } \{x\}; \\
    \text{ var } y: \text{array}[1..10][1..10] \text{ of int class } \{y\}); \\
\text{ var } i, j: \text{int} \{i\}; \\
\text{begin} \\
    b_1 \quad i := 1; \\
    b_2 \quad L2: \text{if } i > 10 \text{ then goto } L7; \\
    b_3 \quad j := 1; \\
    b_4 \quad L4: \text{if } j > 10 \text{ then goto } L6; \\
    b_5 \quad y[j][i] := x[i][j]; \quad j := j + 1; \quad \text{goto } L4; \\
    b_6 \quad L6: \quad i := i + 1; \quad \text{goto } L2; \\
    b_7 \quad L7: \\
\text{end};
\]
Flow of Control

\[ b_1 \rightarrow b_2 \]
\[ b_2 \rightarrow b_3 \]
\[ b_2 \rightarrow b_6 \]
\[ b_6 \rightarrow b_4 \]
\[ b_4 \rightarrow b_5 \]
\[ b_5 \rightarrow b_2 \]
\[ b_3 \rightarrow b_7 \]

\[ i \leq n \]
\[ i > n \]
\[ j \leq n \]
\[ j > n \]
IFDs

• Idea: when two paths out of basic block, implicit flow occurs
  – Because information says \textit{which} path to take
• When paths converge, either:
  – Implicit flow becomes irrelevant; or
  – Implicit flow becomes explicit
• \textit{Immediate forward dominator} of a basic block \( b \)
  (written \( \text{IFD}(b) \)) 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
  - $\text{lub}(x_{i1}, \ldots, x_{in}) \leq \text{glb}\{ y \mid y \text{ target of assignment in } B_i \}$
Example of Requirements

- Within each basic block:
  \[
  b_1: Low \leq i \hspace{1cm} b_3: Low \leq j \hspace{1cm} b_6: \text{lub}\{Low, i\} \leq i
  \]
  \[
  b_5: \text{lub}(x[i][j], i, j) \leq y[j][i]; \text{lub}(Low, j) \leq j
  \]
  - Combining, \( \text{lub}(x[i][j], i, j) \leq y[j][i] \)
  - From declarations, true when \( \text{lub}(x, i) \leq y \)

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

• Result:
  
  – Combine $\text{lub}(x, i) \leq y; i \leq y; i \leq y$
  
  – Requirement is $\text{lub}(x, i) \leq y$
Procedure Calls

\[ tm(a, b); \]

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

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

More generally:

\[
\text{proc } pn(i_1, \ldots, i_m: \text{int}; \text{ var } o_1, \ldots, o_n: \text{int})
\begin{align*}
\text{begin } S \text{ end;}
\end{align*}
\]

- \( S \) must be secure
- For all \( j \) and \( k \), if \( i_j \leq o_k \), then \( x_j \leq y_k \)
- For all \( j \) and \( k \), if \( o_j \leq o_k \), then \( y_j \leq y_k \)