ECS 235B, Lecture 19

February 22, 2019
Composition of Policies

• Two organizations have two security policies
• They merge
  • How do they combine security policies to create one security policy?
  • Can they create a coherent, consistent security policy?
The Problem

• Single system with 2 users
  • Each has own virtual machine
  • Holly at system high, Lara at system low so they cannot communicate directly

• CPU shared between VMs based on load
  • Forms a *covert channel* through which Holly, Lara can communicate
Example Protocol

• Holly, Lara agree:
  • Begin at noon
  • Lara will sample CPU utilization every minute
  • To send 1 bit, Holly runs program
    • Raises CPU utilization to over 60%
  • To send 0 bit, Holly does not run program
    • CPU utilization will be under 40%

• Not “writing” in traditional sense
  • But information flows from Holly to Lara
Policy vs. Mechanism

• Can be hard to separate these

• In the abstract: CPU forms channel along which information can be transmitted
  • Violates *-property
  • Not “writing” in traditional sense

• Conclusion:
  • Bell-LaPadula model does not give sufficient conditions to prevent communication, or
  • System is improperly abstracted; need a better definition of “writing”
Composition of Bell-LaPadula

• Why?
  • Some standards require secure components to be connected to form secure (distributed, networked) system

• Question
  • Under what conditions is this secure?

• Assumptions
  • Implementation of systems precise with respect to each system’s security policy
Issues

• Compose the lattices

• What is relationship among labels?
  • If the same, trivial
  • If different, new lattice must reflect the relationships among the levels
Example
Analysis

• Assume S < HIGH < TS
• Assume SOUTH, EAST, WEST different

Resulting lattice has:
• 4 clearances (LOW < S < HIGH < TS)
• 3 categories (SOUTH, EAST, WEST)
Same Policies

• If we can change policies that components must meet, composition is trivial (as above)
• If we cannot, we must show composition meets the same policy as that of components; this can be very hard
Different Policies

• What does “secure” now mean?
• Which policy (components) dominates?
• Possible principles:
  • Any access allowed by policy of a component must be allowed by composition of components (*autonomy*)
  • Any access forbidden by policy of a component must be forbidden by composition of components (*security*)
Implications

• Composite system satisfies security policy of components as components’ policies take precedence

• If something neither allowed nor forbidden by principles, then:
  • Allow it (Gong & Qian)
  • Disallow it (Fail-Safe Defaults)
Example

• System X: Bob can’t access Alice’s files
• System Y: Eve, Lilith can access each other’s files
• Composition policy:
  • Bob can access Eve’s files
  • Lilith can access Alice’s files
• Question: can Bob access Lilith’s files?
Solution (Gong & Qian)

• Notation:
  • \((a, b)\): \(a\) can read \(b\)'s files
  • \(\text{AS}(x)\): access set of system \(x\)

• Set-up:
  • \(\text{AS}(X) = \emptyset\)
  • \(\text{AS}(Y) = \{(\text{Eve, Lilith}), (\text{Lilith, Eve})\}\)
  • \(\text{AS}(X \cup Y) = \{(\text{Bob, Eve}), (\text{Lilith, Alice}), (\text{Eve, Lilith}), (\text{Lilith, Eve})\}\)
Solution (Gong & Qian)

• Compute transitive closure of $AS(X \cup Y)$:
  • $AS(X \cup Y)^+ = \{(Bob, Eve), (Bob, Lilith), (Bob, Alice), (Eve, Lilith), (Eve, Alice), (Lilith, Eve), (Lilith, Alice)\}$

• Delete accesses conflicting with policies of components:
  • Delete (Bob, Alice)
  • (Bob, Lilith) in set, so Bob can access Lilith’s files
Idea

• Composition of policies allows accesses not mentioned by original policies
• Generate all possible allowed accesses
  • Computation of transitive closure
• Eliminate forbidden accesses
  • Removal of accesses disallowed by individual access policies
• Everything else is allowed
• Note: determining if access allowed is of polynomial complexity
Information Flow

• Basics and background
  • Entropy
• Non-lattice flow policies
• Compiler-based mechanisms
• Execution-based mechanisms
• Examples
  • Privacy and cell phones
  • Firewalls
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:
  • \( \text{PRO} \leq A \), \( A \leq \text{PRO} \)
  • \( \text{PRO} \leq S \)
  • \( A \leq S \), \( S \leq A \)

• Data cannot flow to public relations officers; not transitive
  • \( S \leq A \), \( A \leq \text{PRO} \)
  • \( S \leq \text{PRO} \) 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

• $R = (SC_R, \leq_R, \text{join}_R)$ 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 information 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 $a \leq_R b$
Information 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
  - public $\leq_R$ covert
  - public $\leq_R$ top-level
  - analysis $\leq_R$ analysis
  - covert $\leq_R$ covert
  - 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:
  - \( \text{confine}(p) = [ \{ \text{public} \}, \{ \text{public, analysis} \} ] \)
  - \( \text{confine}(a) = [ \{ \text{analysis} \}, \{ \text{public, analysis, covert, top-level} \} ] \)
  - \( \text{confine}(s) = [ \{ \text{covert} \}, \{ \text{public, analysis, covert, top-level} \} ] \)
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 $l_R(s) \not\subseteq h_R(p)$
  • $l_R(s) = \{ \text{covert} \}$
  • $h_R(p) = \{ \text{public, analysis} \}$
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 information flow policy if flows in set of statements do not violate that policy
Example

```c
if x = 1 then y := a;
else y := b;
```

- Information 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 \( lub\{ A, B \} \), so

\[ lub\{ A, B \} \leq x \]

• 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

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

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

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

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

• Require \( x \leq out \) and \( out \leq out \)
```
Array Elements

• Information flowing out:
  \[ ... := a[i] \]
  Value of \( i \), \( a[i] \) both affect result, so class is \( \text{lub}\{a[i], i\} \)

• 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

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

- 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 $d := b * c - x$; end

• Statement executed reveals information about $x, y, z$, so lub{$x, y, z$} $\leq$ 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

• lub{$x_1, \ldots, x_n$} $\leq$ glb{$y$ | $y$ 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 $f(x_1, \ldots, x_n)$ do $S$;

• Loop must terminate;
• $S$ must be secure
• lub$\{ x_1, \ldots, x_n \} \leq$ glb$\{ y \mid y$ 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

```pascal
proc tm(x: array[1..10][1..10] of integer class {x};
    var y: array[1..10][1..10] of integer class {y});

var i, j: integer class {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

\[ b_1 \rightarrow b_2 \quad i \leq n \quad i > n \rightarrow b_7 \]

\[ b_6 \quad j > n \quad b_3 \]

\[ b_4 \quad j \leq n \quad b_5 \]

February 22, 2019

ECS 235B, Foundations of Computer and Information Security
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 $\text{IFD}(b)$) 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
  • $\text{lub}\{x_{i1}, ..., x_{in}\} \leq \text{glb}\{y | y \text{ target of assignment in } B_i\}$
Example of Requirements

• Within each basic block:
  
  \( b_1: \text{Low} \leq i \)  
  \( b_3: \text{Low} \leq j \)  
  \( b_6: \text{lub}\{ \text{Low}, j \} \leq i \)  
  
  \( b_5: \text{lub}\{ x[i][j], i, j \} \leq y[j][i] \}; \text{lub}\{ \text{Low}, i \} \leq i \)

• 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 $j \leq \text{glb}\{ j, y[j][i] \}$
  - 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

\( \text{tm}(a, b); \)

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

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

More generally:

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

• \( S \) 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 \( y_j \leq y_k \)
Exceptions

```plaintext
proc copy(x: integer class { x });
    var y: integer class Low;

var sum: integer class { x };
    z: int class Low;

begin
    y := z := sum := 0;
    while z = 0 do begin
        sum := sum + x;
        y := y + 1;
    end
end
```
Exceptions (cont)

• When sum overflows, integer overflow trap
  • Procedure exits
  • Value of $x$ is $\text{MAXINT}/y$
  • Information flows from $y$ to $x$, but $x \leq y$ never checked

• Need to handle exceptions explicitly
  • Idea: on integer overflow, terminate loop
    
    ```
    on integer_overflow_exception sum do z := 1;
    ```

  • Now information flows from $sum$ to $z$, meaning $sum \leq z$
  • This is false ($sum = \{x\}$ dominates $z = \text{Low}$)