Lecture #14

• Access control models
  – ORCON, RBAC
• Information flow
  – Noninterference
• Problem: organization creating document wants to control its dissemination
  – Example: Secretary of Agriculture writes a memo for distribution to her immediate subordinates, and she must give permission for it to be disseminated further. This is “originator controlled” (here, the “originator” is a person).
Requirements

- Subject $s \in S$ marks object $o \in O$ as ORCON on behalf of organization $X$. $X$ allows $o$ to be disclosed to subjects acting on behalf of organization $Y$ with the following restrictions:
  1. $o$ cannot be released to subjects acting on behalf of other organizations without $X$’s permission; and
  2. Any copies of $o$ must have the same restrictions placed on it.
DAC Fails

- Owner can set any desired permissions
  - This makes 2 unenforceable
MAC Fails

• First problem: category explosion
  – Category $C$ contains $o$, $X$, $Y$, and nothing else. If a subject $y \in Y$ wants to read $o$, $x \in X$ makes a copy $o'$. Note $o'$ has category $C$. If $y$ wants to give $z \in Z$ a copy, $z$ must be in $Y$—by definition, it’s not. If $x$ wants to let $w \in W$ see the document, need a new category $C'$ containing $o$, $X$, $W$.

• Second problem: abstraction
  – MAC classification, categories centrally controlled, and access controlled by a centralized policy
  – ORCON controlled locally
Combine Them

- The owner of an object cannot change the access controls of the object.
- When an object is copied, the access control restrictions of that source are copied and bound to the target of the copy.
  - These are MAC (owner can’t control them)
- The creator (originator) can alter the access control restrictions on a per-subject and per-object basis.
  - This is DAC (owner can control it)
DRM

- Goal is to protect information on a disk
- “Owner” is actually “licensee”
  - You don’t own the content
  - Owner (copyright holder) can constrain what you can do with it
How Not to Do It

- User must install special program to play content
- Program also modified kernel to:
  - Prevent your CD copying software from working (by using a blacklist)
  - Monitors running applications always (even when no CD in drive)
  - Places hidden files on system
  - Allows you to make 3 copies using their software (and none with yours)
  - Weakens kernel so bad folks can exploit this (unintentional)
RBAC

• Access depends on function, not identity
  – Example:
    • Allison, bookkeeper for Math Dept, has access to financial records.
    • She leaves.
    • Betty hired as the new bookkeeper, so she now has access to those records
  – The role of “bookkeeper” dictates access, not the identity of the individual.
Definitions

• Role $r$: collection of job functions
  – $\text{trans}(r)$: set of authorized transactions for $r$

• Active role of subject $s$: role $s$ is currently in
  – $\text{actr}(s)$

• Authorized roles of a subject $s$: set of roles $s$ is authorized to assume
  – $\text{authr}(s)$

• $\text{canexec}(s, t)$ iff subject $s$ can execute transaction $t$ at current time
Axioms

• Let $S$ be the set of subjects and $T$ the set of transactions.

• Rule of role assignment:
  $$(\forall s \in S)(\forall t \in T) \ [\text{canexec}(s, t) \rightarrow \text{actr}(s) \neq \emptyset].$$
  – If $s$ can execute a transaction, it has a role
  – This ties transactions to roles

• Rule of role authorization:
  $$(\forall s \in S) \ [\text{actr}(s) \subseteq \text{authr}(s)].$$
  – Subject must be authorized to assume an active role
    (otherwise, any subject could assume any role)
Axiom

• Rule of transaction authorization:
  \[(\forall s \in S)(\forall t \in T)\]
  \[ [\text{canexec}(s, t) \rightarrow t \in \text{trans}(\text{actr}(s))] \].
  – If a subject \( s \) can execute a transaction, then the transaction is an authorized one for the role \( s \) has assumed.
Containment of Roles

• Trainer can do all transactions that trainee can do (and then some). This means role $r$ contains role $r'(r > r')$. So:

$$(\forall s \in S)[ r' \in \text{authr}(s) \land r > r' \rightarrow r \in \text{authr}(s) ]$$
Separation of Duty

- Let $r$ be a role, and let $s$ be a subject such that $r \in auth(s)$. Then the predicate $meauth(r)$ (for mutually exclusive authorizations) is the set of roles that $s$ cannot assume because of the separation of duty requirement.

- Separation of duty:

  $$(\forall r_1, r_2 \in R) \ [ r_2 \in meauth(r_1) \rightarrow \ [ (\forall s \in S) \ [ r_1 \in authr(s) \rightarrow r_2 \notin authr(s) ] ] ]$$
Overview

• Problem
  – Policy composition

• Noninterference
  – HIGH inputs affect LOW outputs

• Nondeducibility
  – HIGH inputs can be determined from LOW outputs

• Restrictiveness
  – When can policies be composed successfully
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
• Conclusions:
  – 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
Analysis

• Assume $S < \text{HIGH} < \text{TS}$
• Assume SOUTH, EAST, WEST different
• Resulting lattice has:
  – 4 clearances ($\text{LOW} < S < \text{HIGH} < \text{TS}$)
  – 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
  – \(AS(x)\): access set of system \(x\)

• Set-up:
  – \(AS(X) = \emptyset\)
  – \(AS(Y) = \{ (Eve, Lilith), (Lilith, Eve) \} \)
  – \(AS(X \cup Y) = \{ (Bob, Eve), (Lilith, Alice), (Eve, Lilith), (Lilith, Eve) \} \)
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
Interference

• Think of it as something used in communication
  – Holly/Lara example: Holly interferes with the CPU utilization, and Lara detects it—communication

• Plays role of writing (interfering) and reading (detecting the interference)
Model

- System as state machine
  - Subjects $S = \{s_i\}$
  - States $\Sigma = \{\sigma_i\}$
  - Outputs $O = \{o_i\}$
  - Commands $Z = \{z_i\}$
  - State transition commands $C = S \times Z$

- Note: no inputs
  - Encode either as selection of commands or in state transition commands
Functions

• State transition function $T: C \times \Sigma \rightarrow \Sigma$
  – Describes effect of executing command $c$ in state $\sigma$

• Output function $P: C \times \Sigma \rightarrow O$
  – Output of machine when executing command $c$ in state $s$

• Initial state is $\sigma_0$
Example

- Users Heidi (high), Lucy (low)
- 2 bits of state, $H$ (high) and $L$ (low)
  - System state is $(H, L)$ where $H, L$ are 0, 1
- 2 commands: $xor0, xor1$ do xor with 0, 1
  - Operations affect both state bits regardless of whether Heidi or Lucy issues it
Example: 2-bit Machine

- $S = \{ \text{Heidi, Lucy} \}$
- $\Sigma = \{ (0,0), (0,1), (1,0), (1,1) \}$
- $C = \{ \text{xor0, xor1} \}$

<table>
<thead>
<tr>
<th></th>
<th>Input States $(H, L)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor0</td>
<td>(0,0)</td>
</tr>
<tr>
<td>xor1</td>
<td>(1,1)</td>
</tr>
</tbody>
</table>
Outputs and States

- $T$ is inductive in first argument, as
  \[ T(c_0, \sigma_0) = \sigma_1; \ T(c_{i+1}, \sigma_{i+1}) = T(c_{i+1}, T(c_i, \sigma_i)) \]

- Let $C^*$ be set of possible sequences of commands in $C$

- $T^*: C^* \times \Sigma \rightarrow \Sigma$ and
  \[ c_s = c_0 \ldots c_n \Rightarrow T^*(c_s, \sigma_i) = T(c_n, \ldots, T(c_0, \sigma_i) \ldots) \]

- $P$ similar; define $P^*$ similarly
Projection

• $T^*(c_s, \sigma_i)$ sequence of state transitions
• $P^*(c_s, \sigma_i)$ corresponding outputs
• $proj(s, c_s, \sigma_i)$ set of outputs in $P^*(c_s, \sigma_i)$ that subject $s$ authorized to see
  – In same order as they occur in $P^*(c_s, \sigma_i)$
  – Projection of outputs for $s$
• Intuition: list of outputs after removing outputs that $s$ cannot see
Purge

- \( G \subseteq S \), \( G \) a group of subjects
- \( A \subseteq Z \), \( A \) a set of commands
- \( \pi_G(c_s) \) subsequence of \( c_s \) with all elements \((s, z), s \in G \) deleted
- \( \pi_A(c_s) \) subsequence of \( c_s \) with all elements \((s, z), z \in A \) deleted
- \( \pi_{G,A}(c_s) \) subsequence of \( c_s \) with all elements \((s, z), s \in G \) and \( z \in A \) deleted
Example: 2-bit Machine

- Let $\sigma_0 = (0,1)$
- 3 commands applied:
  - Heidi applies $xor0$
  - Lucy applies $xor1$
  - Heidi applies $xor1$
- $c_s = ((\text{Heidi}, xor0), (\text{Lucy}, xor1), (\text{Heidi}, xor0))$
- Output is 011001
  - Shorthand for sequence (0,1)(1,0)(0,1)
Example

• $proj(\text{Heidi}, c_s, \sigma_0) = 011001$
• $proj(\text{Lucy}, c_s, \sigma_0) = 101$
• $\pi_{\text{Lucy}}(c_s) = (\text{Heidi,xor0}), (\text{Heidi,xor1})$
• $\pi_{\text{Lucy,xor1}}(c_s) = (\text{Heidi,xor0}), (\text{Heidi,xor1})$
• $\pi_{\text{Heidi}}(c_s) = (\text{Lucy,xor1})$
Example

• $\pi_{\text{Lucy}, \text{xor0}}(c_s) = (\text{Heidi}, \text{xor0}), (\text{Lucy}, \text{xor1}), (\text{Heidi}, \text{xor1})$

• $\pi_{\text{Heidi}, \text{xor0}}(c_s) = \pi_{\text{xor0}}(c_s) = (\text{Lucy}, \text{xor1}), (\text{Heidi}, \text{xor1})$

• $\pi_{\text{Heidi}, \text{xor1}}(c_s) = (\text{Heidi}, \text{xor0}), (\text{Lucy}, \text{xor1})$

• $\pi_{\text{xor1}}(c_s) = (\text{Heidi}, \text{xor0})$
Noninterference

- Intuition: Set of outputs Lucy can see corresponds to set of inputs she can see, there is no interference.
- Formally: $G, G' \subseteq S, G \neq G'; A \subseteq Z$; Users in $G$ executing commands in $A$ are noninterfering with users in $G'$ iff for all $c_s \in C^*$, and for all $s \in G'$,

  $$\text{proj}(s, c_s, \sigma_i) = \text{proj}(s, \pi_{G,A}(c_s), \sigma_i)$$

  - Written $A, G :| G'$
Example

• Let $c_s = ((\text{Heidi}, \text{xor}0), (\text{Lucy}, \text{xor}1), (\text{Heidi}, \text{xor}1))$
  and $\sigma_0 = (0, 1)$

• Take $G = \{ \text{Heidi} \}$, $G' = \{ \text{Lucy} \}$, $A = \emptyset$

• $\pi_{\text{Heidi}}(c_s) = (\text{Lucy}, \text{xor}1)$
  – So $\text{proj}(\text{Lucy}, \pi_{\text{Heidi}}(c_s), \sigma_0) = 0$

• $\text{proj}(\text{Lucy}, c_s, \sigma_0) = 101$

• So $\{ \text{Heidi} \} :| \{ \text{Lucy} \}$ is false
  – Makes sense; commands issued to change $H$ bit also affect $L$ bit
Example

• Same as before, but Heidi’s commands affect $H$ bit only, Lucy’s the $L$ bit only
• Output is $0_H0_L1_H$
• $\pi_{\text{Heidi}}(c_s) = (\text{Lucy}, \text{xor1})$
  – So $\text{proj}(\text{Lucy}, \pi_{\text{Heidi}}(c_s), \sigma_0) = 0$
• $\text{proj}(\text{Lucy}, c_s, \sigma_0) = 0$
• So $\{ \text{Heidi} \} :\not\{ \text{Lucy} \}$ is true
  – Makes sense; commands issued to change $H$ bit now do not affect $L$ bit