June 5: Composition of Policies

- Problem
- Deterministic Noninterference
- Nondeducibility
- Generalized Noninterference
- Restrictiveness
Policy Composition

• **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
Example
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
  - \(\text{AS}(x)\): 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 $\text{AS}(X \cup Y)$:
  - $\text{AS}(X \cup Y)^+ = \{(Bob, Eve), (Bob, Lilith), (Bob, Alice), (Eve, Lilith), (Eve, Alice), (Lilith, Eve), (Lilith, Alice)\}$

- 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 $T: C \times \Sigma \to \Sigma$
  – Describes effect of executing command $c$ in state $\sigma$

• Output function $P: C \times \Sigma \to O$
  – 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>Input States $(H, L)$</th>
<th>(0,0)</th>
<th>(0,1)</th>
<th>(1,0)</th>
<th>(1,1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor0</td>
<td>(0,0)</td>
<td>(0,1)</td>
<td>(1,0)</td>
<td>(1,1)</td>
</tr>
<tr>
<td>xor1</td>
<td>(1,1)</td>
<td>(1,0)</td>
<td>(0,1)</td>
<td>(0,0)</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 $xor1$
- $c_s = ((Heidi,xor0),(Lucy,xor1),(Heidi,xor0))$
- Output is 011001
  - Shorthand for sequence $(0,1)(1,0)(0,1)$
Example

- $\text{proj}(\text{Heidi}, c_s, \sigma_0) = 011001$
- $\text{proj}(\text{Lucy}, c_s, \sigma_0) = 101$
- $\pi_{\text{Lucy}}(c_s) = (\text{Heidi}, \text{xor}0), (\text{Heidi}, \text{xor}1)$
- $\pi_{\text{Lucy,xor}1}(c_s) = (\text{Heidi}, \text{xor}0), (\text{Heidi}, \text{xor}1)$
- $\pi_{\text{Heidi}}(c_s) = (\text{Lucy}, \text{xor}1)$
Example

- $\pi_{\text{Lucy, xor 0}}(c_s) = (\text{Heidi, xor 0}), (\text{Lucy, xor 1}), (\text{Heidi, xor 1})$
- $\pi_{\text{Heidi, xor 0}}(c_s) = \pi_{\text{xor 0}}(c_s) = (\text{Lucy, xor 1}), (\text{Heidi, xor 1})$
- $\pi_{\text{Heidi, xor 1}}(c_s) = (\text{Heidi, xor 0}), (\text{Lucy, xor 1})$
- $\pi_{\text{xor 1}}(c_s) = (\text{Heidi, xor 0})$
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 \( s \in G' \),
  \[
  \text{proj}(s, c_s, \sigma_i) = \text{proj}(s, \pi_{GA}(c_s), \sigma_i)
  \]
- 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 $\{ \text{Heidi} \} \not\models \{ \text{Lucy} \}$ 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 $0_H0_L1_H$

• $\pi_{Heidi}(c_s) = (Lucy, xor1)$
  
  – So $proj(Lucy, \pi_{Heidi}(c_s), \sigma_0) = 0$

• $proj(Lucy, c_s, \sigma_0) = 0$

• So $\{ Heidi \} :| \{ Lucy \}$ is true
  
  – Makes sense; commands issued to change $H$ bit now do not affect $L$ bit
Security Policy

• Partitions systems into authorized, unauthorized states
• Authorized states have no forbidden interferences
• Hence a security policy is a set of noninterference assertions
  – See previous definition
Alternative Development

• System $X$ is a set of protection domains $D = \{ d_1, \ldots, d_n \}$

• When command $c$ executed, it is executed in protection domain $\text{dom}(c)$

• Give alternate versions of definitions shown previously
Output-Consistency

- $c \in C, \text{dom}(c) \in D$
- $\sim^{\text{dom}(c)}$ equivalence relation on states of system $X$
- $\sim^{\text{dom}(c)}$ output-consistent if
  $\sigma_a \sim^{\text{dom}(c)} \sigma_b \Rightarrow P(c, \sigma_a) = P(c, \sigma_b)$
- Intuition: states are output-consistent if for subjects in $\text{dom}(c)$, projections of outputs for both states after $c$ are the same
Security Policy

- $D = \{ d_1, \ldots, d_n \}$, $d_i$ a protection domain
- $r: D \times D$ a reflexive relation
- Then $r$ defines a security policy
- Intuition: defines how information can flow around a system
  - $d_i \rightarrow d_j$ means info can flow from $d_i$ to $d_j$
  - $d_i \rightarrow d_i$ as info can flow within a domain
Projection Function

- $\pi'$ analogue of $\pi$, earlier
- Commands, subjects absorbed into protection domains
- $d \in D, c \in C, c_s \in C^*$
- $\pi'_d(\nu) = \nu$
- $\pi'_d(c_sc) = \pi'_d(c_s)c$ if $dom(c)\cap d$
- $\pi'_d(c_sc) = \pi'_d(c_s)$ otherwise
- Intuition: if executing $c$ interferes with $d$, then $c$ is visible; otherwise, as if $c$ never executed
Noninterference-Secure

- System has set of protection domains $D$
- System is noninterference-secure with respect to policy $r$ if
  \[ P^*(c, T^*(c_s, \sigma_0)) = P^*(c, T^*(\pi'_d(c_s), \sigma_0)) \]
- Intuition: if executing $c_s$ causes the same transitions for subjects in domain $d$ as does its projection with respect to domain $d$, then no information flows in violation of the policy
Lemma

• Let $T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0)$ for $c \in C$
• If $\sim^d$ output-consistent, then system is noninterference-secure with respect to policy $r$
Proof

• $d = \text{dom}(c)$ for $c \in C$

• By definition of output-consistent,

$$T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0)$$

implies

$$P^*(c, T^*(c_s, \sigma_0)) = P^*(c, T^*(\pi'_d(c_s), \sigma_0))$$

• This is definition of noninterference-secure with respect to policy $r$
Unwinding Theorem

• Links security of sequences of state transition commands to security of individual state transition commands

• Allows you to show a system design is ML secure by showing it matches specs from which certain lemmata derived
  – Says *nothing* about security of system, because of implementation, operation, *etc.* issues
Locally Respects

• $r$ is a policy
• System $X$ locally respects $r$ if $\text{dom}(c)$ being noninterfering with $d \in D$ implies $\sigma_a \sim^d T(c, \sigma_a)$
• Intuition: applying $c$ under policy $r$ to system $X$ has no effect on domain $d$ when $X$ locally respects $r$
Transition-Consistent

- $r$ policy, $d \in D$
- If $\sigma_a \sim^d \sigma_b$ implies $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, system $X$ transition-consistent under $r$
- Intuition: command $c$ does not affect equivalence of states under policy $r$
Lemma

- $c_1, c_2 \in C, d \in D$
- For policy $r$, $\text{dom}(c_1) \text{rd}$ and $\text{dom}(c_2) \text{rd}$
- Then

$$T^*(c_1 c_2, \sigma) = T(c_1, T(c_2, \sigma)) = T(c_2, T(c_1, \sigma))$$

- Intuition: if info can flow from domains of commands into $d$, then order doesn’t affect result of applying commands
Unwinding Theorem

- Links security of sequences of state transition commands to security of individual state transition commands
- Allows you to show a system design is ML secure by showing it matches specs from which certain lemmata derived
  - Says *nothing* about security of system, because of implementation, operation, *etc.* issues
Locally Respects

- $r$ is a policy
- System $X$ locally respects $r$ if $\text{dom}(c)$ being noninterfering with $d \in D$ implies $\sigma_a \sim^d T(c, \sigma_a)$
- Intuition: applying $c$ under policy $r$ to system $X$ has no effect on domain $d$ when $X$ locally respects $r$
Transition-Consistent

- $r$ policy, $d \in D$
- If $\sigma_a \sim^d \sigma_b$ implies $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, system $X$ transition-consistent under $r$
- Intuition: command $c$ does not affect equivalence of states under policy $r$
Lemma

• \( c_1, c_2 \in C, d \in D \)
• For policy \( r \), \( \text{dom}(c_1)rd \) and \( \text{dom}(c_2)rd \)
• Then

\[
T^*(c_1c_2, \sigma) = T(c_1, T(c_2, \sigma)) = T(c_2, T(c_1, \sigma))
\]

• Intuition: if info can flow from domains of commands into \( d \), then order doesn’t affect result of applying commands
Theorem

• $r$ policy, $X$ system that is output consistent, transition consistent, locally respects $r$
• $X$ noninterference-secure with respect to policy $r$
• Significance: basis for analyzing systems claiming to enforce noninterference policy
  – Establish conditions of theorem for particular set of commands, states with respect to some policy, set of protection domains
  – Noninterference security with respect to $r$ follows
Proof

• Must show $\sigma_a \sim^d \sigma_b$ implies
  $$T^*(c_s, \sigma_a) \sim^d T^*(\pi'_d(c_s), \sigma_b)$$

• Induct on length of $c_s$

• Basis: $c_s = \nu$, so $T^*(c_s, \sigma) = \sigma$; $\pi'_d(\nu) = \nu$; claim holds

• Hypothesis: $c_s = c_1 \ldots c_n$; then claim holds
Induction Step

• Consider $c_sc_{n+1}$. Assume $\sigma_a \sim^d \sigma_b$ and look at $T^*(\pi'_d(c_sc_{n+1}), \sigma_b)$

• 2 cases:
  – $dom(c_{n+1})rd$ holds
  – $dom(c_{n+1})rd$ does not hold
$dom(c_{n+1}) \text{rd} \text{ Holds}$

$T^*(\pi'_d(c_sc_{n+1}), \sigma_b) = T^*(\pi'_d(c_s)c_{n+1}, \sigma_b)$

$= T(c_{n+1}, T^*(\pi'_d(c_s), \sigma_b))$

- by definition of $T^*$ and $\pi'_d$

$\cdot T(c_{n+1}, \sigma_a) \sim^d T(c_{n+1}, \sigma_b)$

- as $X$ transition-consistent and $\sigma_a \sim^d \sigma_b$

$\cdot T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s), \sigma_b))$

- by transition-consistency and IH
\( \text{dom}(c_{n+1}) \text{rd}\) Holds

\[
T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s)c_{n+1}, \sigma_b))
\]
– by substitution from earlier equality

\[
T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s)c_{n+1}, \sigma_b))
\]
– by definition of \( T^* \)

• proving hypothesis
\( \text{dom}(c_{n+1}) \) rd Does Not Hold

\[
T^*(\pi'_d(c_s c_{n+1}), \sigma_b) = T^*(\pi'_d(c_s), \sigma_b)
\]
- by definition of \( \pi'_d \)

\[
T^*(c_s, \sigma_b) = T^*(\pi'_d(c_s c_{n+1}), \sigma_b)
\]
- by above and IH

\[
T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T^*(c_s, \sigma_a)
\]
- as \( X \) locally respects \( r \), so \( \sigma \sim^d T(c_{n+1}, \sigma) \) for any \( \sigma \)

\[
T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s) c_{n+1}, \sigma_b))
\]
- substituting back

- proving hypothesis
Finishing Proof

• Take $\sigma_a = \sigma_b = \sigma_0$, so from claim proved by induction,

\[ T^*(c_s, \sigma_0) \sim^d T^*(\pi'_d(c_s), \sigma_0) \]

• By previous lemma, as $X$ (and so $\sim^d$) output consistent, then $X$ is noninterference-secure with respect to policy $r$
Access Control Matrix

• Example of interpretation
• Given: access control information
• Question: are given conditions enough to provide noninterference security?
• Assume: system in a particular state
  – Encapsulates values in ACM
ACM Model

- **Objects** $L = \{ l_1, \ldots, l_m \}$
  - Locations in memory
- **Values** $V = \{ v_1, \ldots, v_n \}$
  - Values that $L$ can assume
- **Set of states** $\Sigma = \{ \sigma_1, \ldots, \sigma_k \}$
- **Set of protection domains** $D = \{ d_1, \ldots, d_j \}$
Functions

• **value**: $L \times \Sigma \rightarrow V$
  
  - returns value $v$ stored in location $l$ when system in state $\sigma$

• **read**: $D \rightarrow 2^V$
  
  - returns set of objects observable from domain $d$

• **write**: $D \rightarrow 2^V$
  
  - returns set of objects observable from domain $d$
Interpretation of ACM

- Functions represent ACM
  - Subject $s$ in domain $d$, object $o$
  - $r \in A[s, o]$ if $o \in read(d)$
  - $w \in A[s, o]$ if $o \in write(d)$

- Equivalence relation:
  
  $[\sigma_a \sim^{\text{dom}(c)} \sigma_b] \iff [\forall l_i \in read(d)\text{ [ }value(l_i, \sigma_a) = value(l_i, \sigma_b)\text{ ]}]$

  - You can read the exactly the same locations in both states
Enforcing Policy $r$

- 5 requirements
  - 3 general ones describing dependence of commands on rights over input and output
    - Hold for all ACMs and policies
  - 2 that are specific to some security policies
    - Hold for most policies