ECS 289M Lecture 13

April 28, 2006

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”

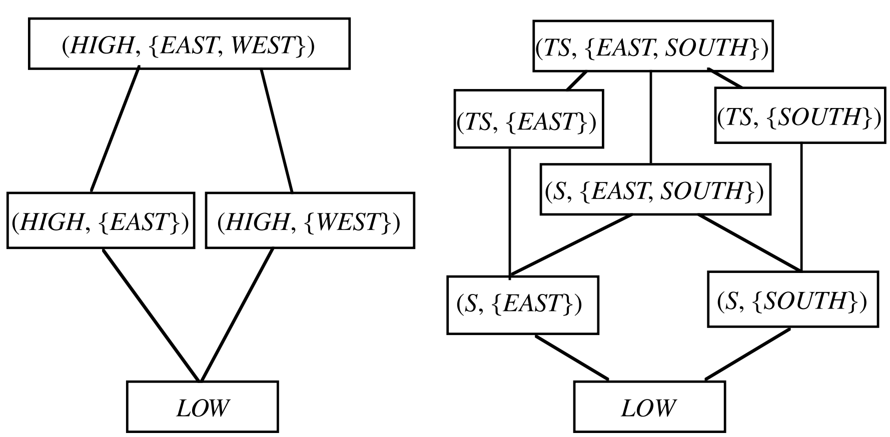
Composition of Bell-LaPadula

- 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 < 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
  - AS(*x*): access set of system *x*
- Set-up:
  - AS(X) = ∅
  - AS(Y) = { (Eve, Lilith), (Lilith, Eve) }
  - AS(X∪Y) = { (Bob, Eve), (Lilith, Alice), (Eve, Lilith), (Lilith, Eve) }

Solution (Gong & Qian)

- Compute transitive closure of AS(X∪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 σ
- Output function P: $C \times \Sigma \rightarrow O$
  - Output of machine when executng command c in state s
- Initial state is σ_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 = { Heidi, Lucy }
- Σ = { (0,0), (0,1), (1,0), (1,1) }
- C = { *xor0*, *xor1* }

	Input States (H, L)			
	(0,0)	(0,1)	(1,0)	(1,1)
xor0	(0,0)	(0,1)	(1,0)	(1,1)
xor1	(1,1)	(1,0)	(0,1)	(0,0)

Outputs and States

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

- *proj*(Heidi, c_s, σ_0) = 011001
- *proj*(Lucy, c_s, σ_0) = 101
- $\pi_{\text{Lucy}}(c_s)$ = (Heidi,*xor0*), (Heidi,*xor1*)
- $\pi_{\text{Lucy},xor1}(c_s)$ = (Heidi,*xor0*), (Heidi,*xor1*)
- $\pi_{\text{Heidi}}(c_s)$ = (Lucy,*xor1*)

Example

- $\pi_{\text{Lucy},xor0}(c_s)$ = (Heidi,*xor0*),(Lucy,*xor1*),(Heidi,*xor1*)
- $\pi_{\text{Heidi},xor0}(c_s) = \pi_{xor0}(c_s)$ = (Lucy,*xor1*),(Heidi, *xor1*)
- $\pi_{\text{Heidi},xor1}(c_s)$ = (Heidi, *xor0*), (Lucy, *xor1*)
- $\pi_{xor1}(c_s)$ = (Heidi, *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 $s \in G'$,

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

  – Written $A,G :| G'$

Example

- Let c_s = ((Heidi,*xor0*),(Lucy,*xor1*),(Heidi,*xor1*)) and σ_0 = (0, 1)
- Take G = { Heidi }, G' = { Lucy }, $A = \varnothing$
- $\pi_{\text{Heidi}}(c_s)$ = (Lucy,*xor1*)
  - So *proj*(Lucy, $\pi_{\text{Heidi}}(c_s)$, σ_0) = 0
- proj(Lucy, c_s, σ_0) = 101
- So { Heidi } :| { 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 $0_H 0_L 1_H$
- $\pi_{\text{Heidi}}(c_s)$ = (Lucy,*xor1*)
  - So *proj*(Lucy, $\pi_{\text{Heidi}}(c_s)$, σ_0) = 0
- proj(Lucy, c_s, σ_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 $dom(c)$
- Give alternate versions of definitions shown previously

Output-Consistency

- $c \in C$, $dom(c) \in D$
- $\sim^{dom(c)}$ equivalence relation on states of system X
- $\sim^{dom(c)}$ *output-consistent* if
  $\sigma_a \sim^{dom(c)} \sigma_b \Rightarrow P(c, \sigma_a) = P(c, \sigma_b)$
- Intuition: states are output-consistent if for subjects in *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
  - $d_i r d_j$ means info can flow from d_i to d_j
  - $d_i r d_i$ as info can flow within a domain

Projection Function

- π' analogue of π, earlier
- Commands, subjects absorbed into protection domains
- $d \in D$, $c \in C$, $c_s \in C^*$
- $\pi'_d(\nu) = \nu$
- $\pi'_d(c_s c) = \pi'_d(c_s)c$ if $dom(c)rd$
- $\pi'_d(c_s c) = \pi'_d(c_s)$ 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

- $d = dom(c)$ 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 $dom(c)$ 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, $dom(c_1)rd$ and $dom(c_2)rd$
- 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