Lecture 11

- Role-Based Access Control
- Composition of policies
- Noninterference
  - Access control matrix interpretation

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
  - *trans*(*r*): set of authorized transactions for *r*
- Active role of subject *s*: role *s* is currently in
  - *actr*(*s*)
- Authorized roles of a subject *s*: set of roles *s* is authorized to assume
  - *authr*(*s*)
- *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*:

  $$(\forall s \in S)(\forall t \in T)\ [canexec(s, t) \rightarrow actr(s) \neq \varnothing]$$

  – If s can execute a transaction, it has a role
  – This ties transactions to roles
- *Rule of role authorization*:

  $$(\forall s \in S)\ [actr(s) \subseteq authr(s)]$$

  – Subject must be authorized to assume an active role (otherwise, any subject could assume any role)

Axiom

- *Rule of transaction authorization*:

$$(\forall s \in S)(\forall t \in T)$$
$$[canexec(s, t) \rightarrow t \in trans(actr(s))].$$

  – 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:

$$(\forall s \in S)[\ r' \in authr(s) \wedge r > r' \rightarrow r \in authr(s)\ ]$$

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:

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

Multiple Policies

- 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"

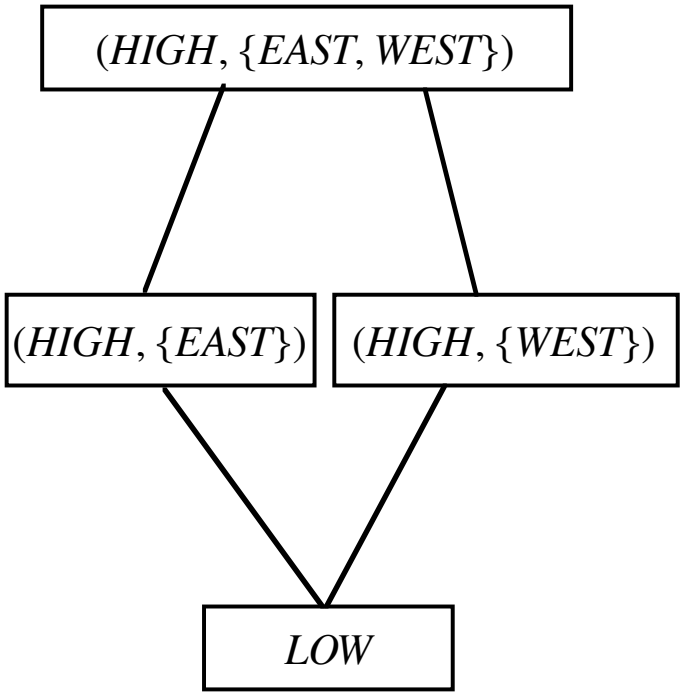
Composition of Bell-LaPadula

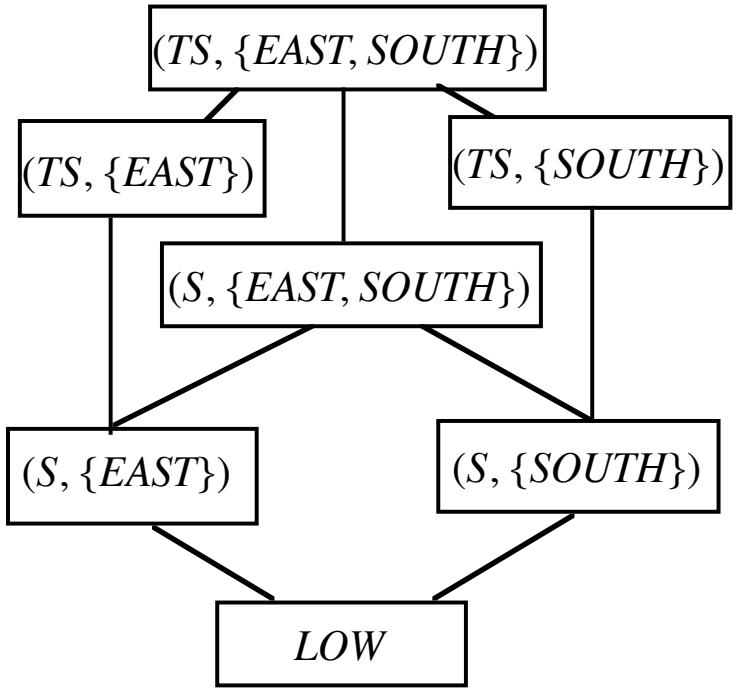
- Why?
  - Some standards require secure components to be connected to form secure (distributed, networked) system
- Question
  - Under what conditions is this secure?
- Assumptions
  - Implementation of systems precise with respect to each system's security policy

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


(HIGH, {EAST, WEST})
(HIGH, {EAST})
(HIGH, {WEST})
LOW


(TS, {EAST, SOUTH})
(TS, {EAST})
(TS, {SOUTH})
(S, {EAST, SOUTH})
(S, {EAST})
(S, {SOUTH})
LOW

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) = $\varnothing$
  - 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∪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 executing command c in state σ
- 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: xor_0, xor_1 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 = { xor_0, xor_1 }

	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, σ_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 xor_0
  - Lucy applies xor_1
  - Heidi applies xor_1
- $c_s = ((\text{Heidi}, xor_0), (\text{Lucy}, xor_1), (\text{Heidi}, xor_0))$
- 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
- $\pi_{\text{Lucy}}(c_s) = ((\text{Heidi}, xor_0), (\text{Heidi}, xor_1))$
- $\pi_{\text{Lucy},xor1}(c_s) = ((\text{Heidi}, xor_0), (\text{Heidi}, xor_1))$
- $\pi_{\text{Heidi}}(c_s) = ((\text{Lucy}, xor_1))$

Example

- $\pi_{\text{Lucy},xor0}(c_s) = ((\text{Heidi}, xor_0), (\text{Lucy}, xor_1), (\text{Heidi}, xor_1))$
- $\pi_{\text{Heidi},xor0}(c_s) = \pi_{xor0}(c_s) = ((\text{Lucy}, xor_1), (\text{Heidi}, xor_1))$
- $\pi_{\text{Heidi}, xor1}(c_s) = ((\text{Heidi}, xor_0), (\text{Lucy}, xor_1))$
- $\pi_{xor1}(c_s) = ((\text{Heidi}, xor_0))$

Noninterference

- Intuition: Set of outputs Lucy can see corresponds to set of inputs she can see, there is no interference
- Formally: $G, G' \subseteq \mathrm{S}, G \neq G'$; $A \subseteq Z$; 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, xor_0), (Lucy, xor_1), (Heidi, xor_1)) and σ_0 = (0, 1)
- Take G = { Heidi }, G' = { Lucy }, $A = \varnothing$
- $\pi_{\text{Heidi}}(c_s)$ = ((Lucy, xor_1))
  - So *proj*(Lucy, $\pi_{\text{Heidi}}(c_s)$, σ_0) = 0
- proj(Lucy, c_s, σ_0) = 101
- So { Heidi } :l { 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 $0_H 0_L 1_H$
- $\pi_{\text{Heidi}}(c_s) = ((\text{Lucy}, xor1))$
  - So $proj(\text{Lucy}, \pi_{\text{Heidi}}(c_s), \sigma_0) = 0$
- $\text{proj}(\text{Lucy}, c_s, \sigma_0) = 0$
- So { Heidi } :I { 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, $\mathrm{d} \in D$
- 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

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 $c_s c_{n+1}$. Assume $\sigma_a \sim^d \sigma_b$ and look at $T^*(\pi'_d(c_s c_{n+1}), \sigma_b)$
- 2 cases:
  - $dom(c_{n+1})rd$ holds
  - $dom(c_{n+1})rd$ does not hold

$dom(c_{n+1})rd$ Holds

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

– by definition of T^* and π'_d

- $T(c_{n+1}, \sigma_a) \sim^d T(c_{n+1}, \sigma_b)$
  – as X transition-consistent and $\sigma_a \sim^d \sigma_b$
- $T(c_{n+1}, T^*(c_s, \sigma_a)) \sim^d T(c_{n+1}, T^*(\pi'_d(c_s), \sigma_b))$
  – by transition-consistency and IH

$dom(c_{n+1})rd$ 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

$dom(c_{n+1})rd$ Does Not Hold

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

- by definition of π'_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 σ

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

- 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 σ
- *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:

$$[\sigma_a \sim^{dom(c)} \sigma_b] \Leftrightarrow [\ \forall l_i \in read(d)$$
$$[\ value(l_i, \sigma_a) = value(l_i, \sigma_b)\ ]\ ]$$

  - 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

Enforcing Policy *r*: First

- Output of command *c* executed in domain *dom*(*c*) depends only on values for which subjects in *dom*(*c*) have read access

$$\sigma_a \sim^{dom(c)} \sigma_b \Rightarrow P(c, \sigma_a) = P(c, \sigma_b)$$

Enforcing Policy *r*: Second

- If c changes l_i, then c can only use values of objects in $read(dom(c))$ to determine new value

$$[\ \sigma_a \sim^{dom(c)} \sigma_b \textit{ and } (value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a) \textit{ or } value(l_i, T(c, \sigma_b)) \neq value(l_i, \sigma_b))\ ] \Rightarrow value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$$

Enforcing Policy *r*: Third

- If c changes l_i, then $dom(c)$ provides subject executing c with write access to l_i

$$value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a) \Rightarrow l_i \in write(dom(c))$$

Enforcing Policies *r*: Fourth

- If domain *u* can interfere with domain *v*, then every object that can be read in *u* can also be read in *v*
- So if object *o* cannot be read in *u*, but can be read in *v*; and object *o′* in *u* can be read in *v*, then info flows from *o* to *o′*, then to *v*

Let $u, v \in D$; then $urv \Rightarrow read(u) \subseteq read(v)$

Enforcing Policies r: Fifth

- Subject s can write object o in v, subject s' can read o in u, then domain v can interfere with domain u

$$l_i \in read(u) \; and \; l_i \in write(v) \Rightarrow vru$$

Theorem

- Let *X* be a system satisfying the five conditions. The *X* is noninterference-secure with respect to r
- Proof: must show *X* output-consistent, locally respects *r*, transition-consistent
  - Then by unwinding theorem, theorem holds

Output-Consistent

- Take equivalence relation to be $\sim^d$, first condition *is* definition of output-consistent

Locally Respects *r*

- Proof by contradiction: assume $(dom(c), d) \notin r$ but $\sigma_a \sim^d T(c, \sigma_a)$ does not hold
- Some object has value changed by *c*:

  $\exists\, l_i \in read(d)\ [\ value(l_i, \sigma_a) \neq \text{value}(l_i, T(c, \sigma_a))\ ]$
- Condition 3: $l_i \in write(d)$
- Condition 5: $dom(c)rd$, contradiction
- So $\sigma_a \sim^d T(c, \sigma_a)$ holds, meaning *X* locally respects *r*

Transition Consistency

- Assume $\sigma_a \sim^d \sigma_b$
- Must show

$$\text{value}(l_i, T(c, \sigma_a)) = \textit{value}(l_i, T(c, \sigma_b))$$

for $l_i \in \textit{read}(d)$

- 3 cases dealing with change that c makes in l_i in states σ_a, σ_b

Case 1

- $value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a)$
- Condition 3: $l_i \in write(dom(c))$
- As $l_i \in read(d)$, condition 5 says $dom(c) rd$
- Condition 4 says $read(dom(c)) \subseteq read(d)$
- As $\sigma_a \sim^{d} \sigma_b$, $\sigma_a \sim^{dom(c)} \sigma_b$
- Condition 2:

$$value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$$

- So $T(c, \sigma_a) \sim^{dom(c)} T(c, \sigma_b)$, as desired

Case 2

- $value(l_i, T(c, \sigma_b)) \neq value(l_i, \sigma_b)$
- Condition 3: $l_i \in write(dom(c))$
- As $l_i \in read(d)$, condition 5 says $dom(c)rd$
- Condition 4 says $read(dom(c)) \subseteq read(d)$
- As $\sigma_a \sim^{d} \sigma_b$, $\sigma_a \sim^{dom(c)} \sigma_b$
- Condition 2:

$$value(l_i, T(c, \sigma_a)) = value(l_i, T(c, \sigma_b))$$

- So $T(c, \sigma_a) \sim^{dom(c)} T(c, \sigma_b)$, as desired

Case 3

- Neither of the previous two
  - $value(l_i, T(c, \sigma_a)) = value(l_i, \sigma_a)$
  - $value(l_i, T(c, \sigma_b)) = value(l_i, \sigma_b)$
- Interpretation of $\sigma_a \sim^d \sigma_b$ is:

  for $l_i \in read(d)$, $value(l_i, \sigma_a) = value(l_i, \sigma_b)$
- So $T(c, \sigma_a) \sim^d T(c, \sigma_b)$, as desired
- In all 3 cases, X transition-consistent

Policies Changing Over Time

- Problem: previous analysis assumes static system
  - In real life, ACM changes as system commands issued
- Example: $w \in C^*$ leads to current state
  - *cando*(w, s, z) holds if s can execute z in current state
  - Condition noninterference on *cando*
  - If ¬*cando*(w, Lara, "write *f*"), Lara can't interfere with any other user by writing file *f*

Generalize Noninterference

- $G \subseteq S$ group of subjects, $A \subseteq Z$ set of commands, p predicate over elements of C^*
- $c_s = (c_1, \ldots, c_n) \in C^*$
- $\pi''(\nu) = \nu$
- $\pi''((c_1, \ldots, c_n)) = (c_1', \ldots, c_n')$
  - $c_i' = \nu$ if $p(c_1', \ldots, c_{i-1}')$ and $c_i = (s, z)$ with $s \in G$ and $z \in A$
  - $c_i' = c_i$ otherwise

Intuition

- $\pi''(c_s) = c_s$
- But if p holds, and element of c_s involves both command in A and subject in G, replace corresponding element of c_s with empty command ν
  - Just like deleting entries from c_s as $\pi_{A,G}$ does earlier

Noninterference

- $G, G' \subseteq S$ groups of subjects, $A \subseteq Z$ set of commands, p predicate over C^*
- Users in G executing commands in A are noninterfering with users in G' under condition p iff, for all $c_s \in C^*$, all $s \in G'$, $proj(s, c_s, \sigma_i) = proj(s, \mathrm{p}''(c_s), \sigma_i)$
  - Written $A,G :| G'$ **if** p

Example

- From earlier one, simple security policy based on noninterference:

  $\forall(s \in S)\ \forall(z \in Z)$

  $[\ \{z\}, \{s\} :|\ S$ **if** $\neg cando(w, s, z)\ ]$
- If subject can't execute command (the $\neg cando$ part), subject can't use that command to interfere with another subject

Another Example

- Consider system in which rights can be passed
  - $pass(s, z)$ gives s right to execute z
  - $w_n = v_1, \ldots, v_n$ sequence of $v_i \in C^*$
  - $prev(w_n) = w_{n-1}$; last(wn) = v_n

Policy

- No subject s can use z to interfere if, in previous state, s did not have right to z, and no subject gave it to s

$\{ z \}, \{ s \} :| S$ **if**

$$[\neg cando(prev(w), s, z) \wedge$$
$$[cando(prev(w), s', pass(s, z)) \Rightarrow$$
$$\neg last(w) = (s', pass(s, z))]]$$

Effect

- Suppose $s_1 \in S$ can execute $pass(s_2, z)$
- For all $w \in C^*$, $cando(w, s_1, pass(s_2, z))$ true
- Initially, $cando(\nu, s_2, z)$ false
- Let $z' \in Z$ be such that (s_3, z') noninterfering with (s_2, z)
  - So for each w_n with $v_n = (s_3, z')$,

$$cando(w_n, s_2, z) = cando(w_{n-1}, s_2, z)$$

Effect

- Then policy says for all $s \in S$

$$proj(s, ((s_2, z), (s_1, pass(s_2, z)), (s_3, z'), (s_2, z)), \sigma_i) =$$

$$proj(s, ((s_1, pass(s_2, z)), (s_3, z'), (s_2, z)), \sigma_i)$$

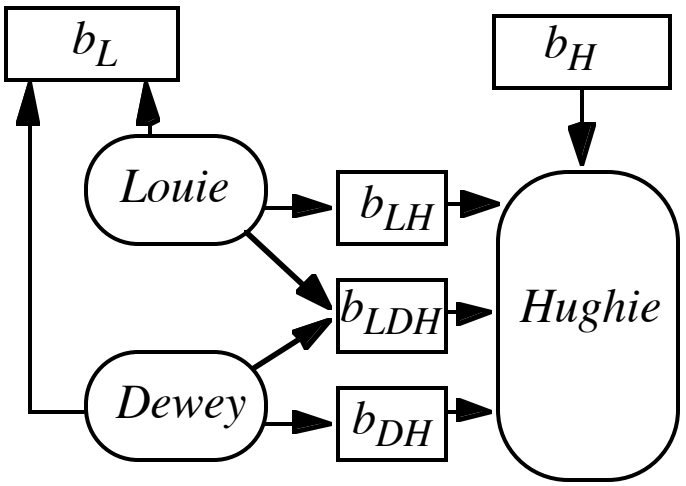
- So s_2's first execution of z does not affect any subject's observation of system

Policy Composition I

- Assumed: Output function of input
  - Means deterministic (else not function)
  - Means uninterruptability (differences in timings can cause differences in states, hence in outputs)
- This result for deterministic, noninterference-secure systems

Compose Systems

- Louie, Dewey LOW
- Hughie HIGH
- b_L output buffer
  - Anyone can read it
- b_H input buffer
  - From HIGH source
- Hughie reads from:
  - b_{LH} (Louie writes)
  - b_{LDH} (Louie, Dewey write)
  - b_{DH} (Dewey writes)



Systems Secure

- All noninterference-secure
  - Hughie has no output
    - So inputs don't interfere with it
  - Louie, Dewey have no input
    - So (nonexistent) inputs don't interfere with outputs