ECS 235B, Lecture 24

March 8, 2019

Analyzing Covert Channels

- Policy and operational issues determine how dangerous it is
  - What follows assumes a policy saying all covert channels are a problem
- *Amount* of information that can be transmitted affects how serious a problem a covert channel is
  - 1 bit per hour: probably harmless in most circumstances
  - 1,000,000 bits per second: probably dangerous in most circumstances
  - Begin here . . .

Measuring Capacity

- Intuitively, difference between unmodulated, modulated channel
  - Normal uncertainty in channel is 8 bits
  - Attacker modulates channel to send information, reducing uncertainty to 5 bits
  - Covert channel capacity is 3 bits
    - Modulation in effect fixes those bits

Formally

- Inputs:
  - *A* input from Alice (sender)
  - *V* input from everyone else
  - *X* output of channel
- Capacity measures uncertainty in *X* given *A*
- In other terms: maximize

$$I(A; X) = H(X) - H(X \mid A)$$

  with respect to *A*

Noninterference and Covert Channels

- If *A*, *V* are independent and *A* noninterfering with *X*, then $I(A; X) = 0$
- Why? Intuition is that *A* and *X* are independent
  - If so, then only *V* affects *X* (noninterference)
  - So information from *A* cannot affect *X* unless *A* influences *V*
  - But *A* and *V* are independent, so information from *A* does not affect *X*
- But noninterference is not necessary

Example: Noninterference Not Necessary

- System has 1 bit of state; 3 inputs I_A, I_B, I_C; one output O_X
- Each input flips state, and state's value is then output
  - System initially in state 0
- *w* sequence of inputs corresponding to output $x(w) = length(w) \bmod 2$
  - I_A not noninterfering as deleting its inputs may change output
- Define terms
  - *W* random variable corresponding to length of input sequences
  - *A* random variable corresponding to length of input sequences contributed by I_A; *V* random variable corresponding to other contributions; *A*, *V* independent
  - *X* random variable corresponding to output state

Two Cases

- $V = 0$; then as $W = (A + V) \bmod 2$, $W = A$, and so A, W not independent; neither are A, X. So if $V = 0$, $I(A, X) \neq 0$
- I_B, I_C produce inputs such that p(V=0) = p(V=1) = 0.5; then

$$p(X=x) = p(V=x, A=0) + p(V = 1 - x, A = 1)$$

Because A, V independent, this becomes

$$p(X=x) = p(V=x, A=0) + p(V = 1 - x)p(A = 1)$$

and so $p(X=x) = 0.5$. Also,

$$p(X=x \mid A=a) = p(X = (a + x) \bmod 2) = 0.5$$

establishing A, X independent; so $I(A, X) = 0$

Meaning

- Note A, X noninterfering, and I(A; X) = 0
- So covert channel capacity is 0 if either of the following hold:
  - Input is noninterfering with output; or
  - Input comes from independent sources, all possible values from at least one source are equally probable

Example (More Formally)

- If A, V independent, take $p=p(A=0)$, $q=p(V=0)$:
  - $p(A=0,V=0) = pq$
  - $p(A=1,V=0) = (1-p)q$
  - $p(A=0,V=1) = p(1-q)$
  - $p(A=1,V=1) = (1-p)(1-q)$
- So
  - $p(X=0) = p(A=0,V=0)+\mathrm{p}(A=1,V=1) = pq + (1-p)(1-q)$
  - $p(X=1) = p(A=0,V=1)+\mathrm{p}(A=1,V=0) = (1-p)q + p(1-q)$

Example (*con't*)

- Also:
  - $p(X=0|A=0) = q$
  - $p(X=0|A=1) = 1-q$
  - $p(X=1|A=0) = 1-q$
  - $\text{p}(X=1|A=1) = q$
- So you can compute:
  - $H(X) = -[(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)]$
  - $H(X|A) = -q \lg q - (1-q) \lg (1-q)$
  - $I(A;X) = H(X)-H(X|A)$

Example (*con't*)

- So $I(A; X) = -[pq + (1-p)(1-q)] \lg [pq + (1-p)(1-q)] -$
  $[(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)] +$
  $q \lg q + (1-q) \lg (1-q)$
- Maximum when p = 0.5; then

$$I(A;X) = 1 + q \lg q + (1-q) \lg (1-q) = 1 - H(V)$$

- So, if $q = 0$ (meaning V is constant) then $I(A;X) = 1$
- Also, if $q = p = 0.5$, $I(A;X) = 0$

Analyzing Capacity

- Assume a noisy channel
- Examine covert channel in MLS database that uses replication to ensure availability
  - 2-phase commit protocol ensures atomicity
  - *Coordinator* process manages global execution
  - *Participant* processes do everything else

How It Works

- Coordinator sends message to each participant asking whether to abort or commit transaction
  - If any says “abort”, coordinator stops
- Coordinator gathers replies
  - If all say “commit”, sends commit messages back to participants
  - If any says “abort”, sends abort messages back to participants
  - Each participant that sent commit waits for reply; on receipt, acts accordingly

Exceptions

- Protocol times out, causing party to act as if transaction aborted, when:
  - Coordinator doesn’t receive reply from participant
  - Participant who sends a commit doesn’t receive reply from coordinator

Covert Channel Here

- Two types of components
  - One at *Low* security level, other at *High*
- Low component begins 2-phase commit
  - Both *High*, *Low* components must cooperate in the 2-phase commit protocol
- *High* sends information to *Low* by selectively aborting transactions
  - Can send abort messages
  - Can just not do anything

Note

- If transaction *always* succeeded except when *High* component sending information, channel not noisy
  - Capacity would be 1 bit per trial
  - But channel noisy as transactions may abort for reasons *other* than the sending of information

Analysis

- X random variable: what *High* user wants to send
  - Assume abort is 1, commit is 0
  - $p = p(X=0)$ probability *High* sends 0
- A random variable: what *Low* receives
  - For noiseless channel $X = A$
- n+2 users
  - Sender, receiver, n others that act independently of one another
  - q probability of transaction aborting at any of these n users

Basic Probabilities

- Probabilities of receiving given sending
  - $p(A=0|X=0) = (1-q)^n$
  - $p(A=1|X=0) = 1-(1-q)^n$
  - $p(A=0|X=1) = 0$
  - $p(A=1|X=1) = 1$
- So probabilities of receiving values:
  - $p(A=0) = p(1-q)^n$
  - $p(A=1) = 1-p(1-q)^n$

More Probabilities

- Given sending, what is receiving?
  - $p(X=0 | A=0) = 1$
  - $p(X=1 | A=0) = 0$
  - $p(X=0 | A=1) = p[1-(1-q)^n] / [1-p(1-q)^n]$
  - $p(X=1 | A=1) = (1-p) / [1-p(1-q)^n]$

Entropies

You can compute these:

- $H(X) = -p \lg p - (1-p) \lg (1-p)$
- $H(X|A) = -p[1-(1-q)^n] \lg p - p[1-(1-q)^n] \lg [1-(1-q)^n] +$
  $[1-p(1-q)^n] \lg [1-p(1-q)^n] - (1-p) \lg (1-p)$
- $\mathrm{I}(\mathrm{A};\mathrm{X}) = -p(1-q)^n \lg p + p[1-(1-q)^n] \lg [1-(1-q)^n] -$
  $[1-p(1-q)^n] \lg [1-p(1-q)^n]$

Capacity

- Maximize this with respect to p (probability that *High* sends 0)
  - Notation: $m = (1-q)^n$, $M = (1-m)^{(1-m)}$
  - Maximum when $p = M / (Mm+1)$
- Capacity is:

$$I(A;X) = \frac{Mm \lg p + M(1-m) \lg (1-m) + \lg (Mm+1)}{(Mm+1)}$$

Mitigation of Covert Channels

- Problem: these work by varying use of shared resources
- One solution
  - Require processes to say what resources they need before running
  - Provide access to them in a way that no other process can access them
- Cumbersome
  - Includes running (CPU covert channel)
  - Resources stay allocated for lifetime of process

Alternate Approach

- Obscure amount of resources being used
  - Receiver cannot distinguish between what the sender is using and what is added
- How? Two ways:
  - Devote uniform resources to each process
  - Inject randomness into allocation, use of resources

Uniformity

- Variation of isolation
  - Process can’t tell if second process using resource
- Example: KVM/370 covert channel via CPU usage
  - Give each VM a time slice of fixed duration
  - Do not allow VM to surrender its CPU time
    - Can no longer send 0 or 1 by modulating CPU usage

Randomness

- Make noise dominate channel
  - Does not close it, but makes it useless
- Example: MLS database
  - Probability of transaction being aborted by user other than sender, receiver approaches 1
    - $q \to 1$
  - $I(A; X) \to 0$
  - How to do this: resolve conflicts by aborting increases q, or have participants abort transactions randomly

Problem: Loss of Efficiency

- Fixed allocation, constraining use
  - Wastes resources
- Increasing probability of aborts
  - Some transactions that will normally commit now fail, requiring more retries
- Policy: is the inefficiency preferable to the covert channel?

Example

- Goal: limit covert timing channels on VAX/VMM
- “Fuzzy time” reduces accuracy of system clocks by generating random clock ticks
  - Random interrupts take any desired distribution
  - System clock updates only after each timer interrupt
  - Kernel rounds time to nearest 0.1 sec before giving it to VM
    - Means it cannot be more accurate than timing of interrupts

Example

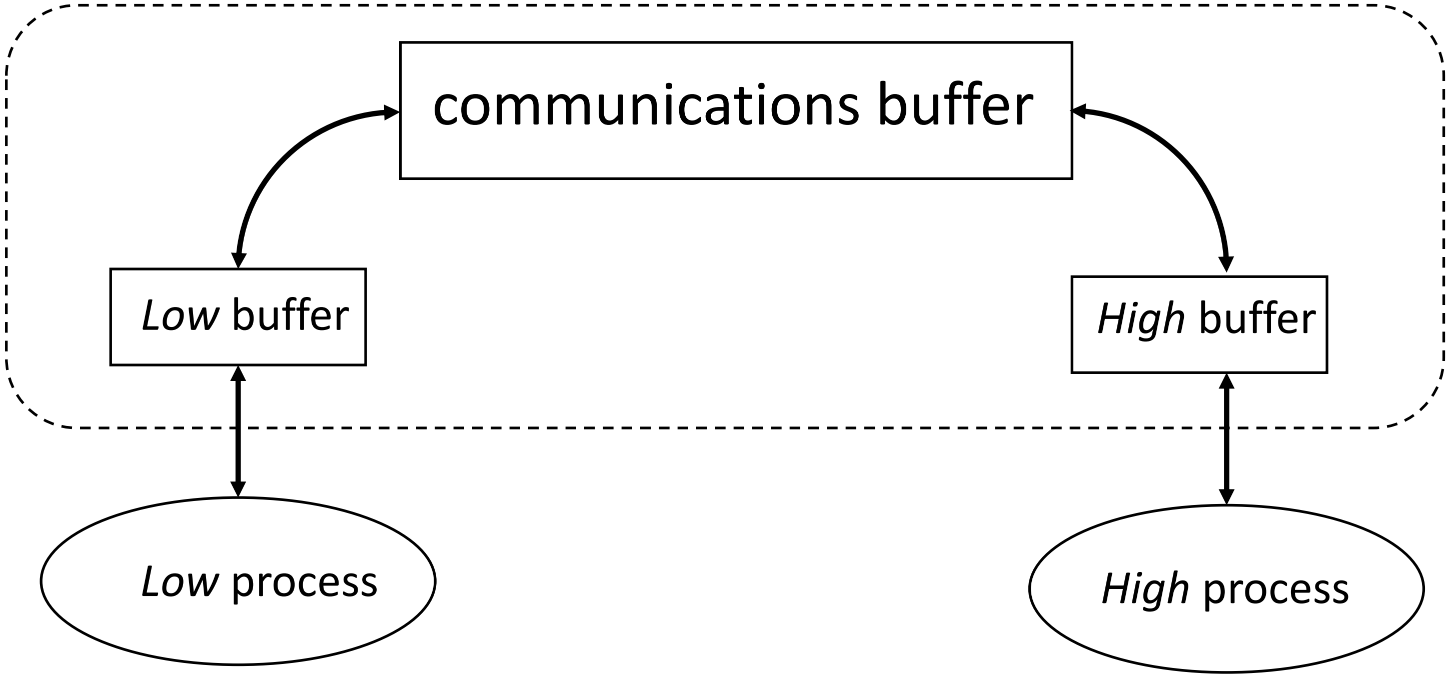
- I/O operations have random delays
- Kernel distinguishes 2 kinds of time:
  - *Event time* (when I/O event occurs)
  - *Notification time* (when VM told I/O event occurred)
    - Random delay between these prevents VM from figuring out when event actually occurred)
    - Delay can be randomly distributed as desired (in security kernel, it’s 1–19ms)
  - Added enough noise to make covert timing channels hard to exploit

Improvement

- Modify scheduler to run processes in increasing order of security level
  - Now we're worried about "reads up", so …
- Countermeasures needed only when transition from *dominating* VM to *dominated* VM
  - Add random intervals between quanta for these transitions

The Pump

- Tool for controlling communications path between *High* and *Low*



Details

- Communications buffer of length *n*
  - Means it can hold up to *n* messages
- Messages numbered
- Pump ACKs each message as it is moved from *High* (*Low*) buffer to communications buffer
- If pump crashes, communications buffer preserves messages
  - Processes using pump can recover from crash

Covert Channel

- Low fills communications buffer
  - Send messages to pump until no ACK
  - If *High* wants to send 1, it accepts 1 message from pump; if *High* wants to send 0, it does not
  - If *Low* gets ACK, message moved from *Low* buffer to communications buffer $\Rightarrow$ *High* sent 1
  - If *Low* doesn't get ACK, no message moved $\Rightarrow$ *High* sent 0
- Meaning: if *High* can control rate at which pump passes messages to it, a covert timing channel

Performance vs. Capacity

- Assume *Low* process, pump can process messages more quickly than *High* process
- L_i random variable: time from *Low* sending message to pump to *Low* receiving ACK
- H_i random variable: average time for *High* to ACK each of last n messages

Case1: $E(L_i) > H_i$

- *High* can process messages more quickly than *Low* can get ACKs
- Contradicts above assumption
  - Pump must be delaying ACKs
  - *Low* waits for ACK whether or not communications buffer is full
- Covert channel closed
- Not optimal
  - Process may wait to send message even when there is room

Case 2: $E(L_i) < H_i$

- *Low* sending messages faster than *High* can remove them
- Covert channel open
- Optimal performance

Case 3: $E(L_i) = H_i$

- Pump, processes handle messages at same rate
- Covert channel open
  - Bandwidth decreased from optimal case (can’t send messages over covert channel as fast)
- Performance not optimal

Adding Noise

- Shown: adding noise to approximate case 3
  - Covert channel capacity reduced to 1/*nr* where *r* time from *Low* sending message to pump to *Low* receiving ACK when communications buffer not full
  - Conclusion: use of pump substantially reduces capacity of covert channel between *High*, *Low* processes when compared to direct connection

Key Points

- Confinement problem central to computer security
  - Arises in many contexts
- Many approaches to handle it
  - Each has benefits and drawbacks
- Covert channels are hard to close
  - But their capacity can be measured and reduced

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

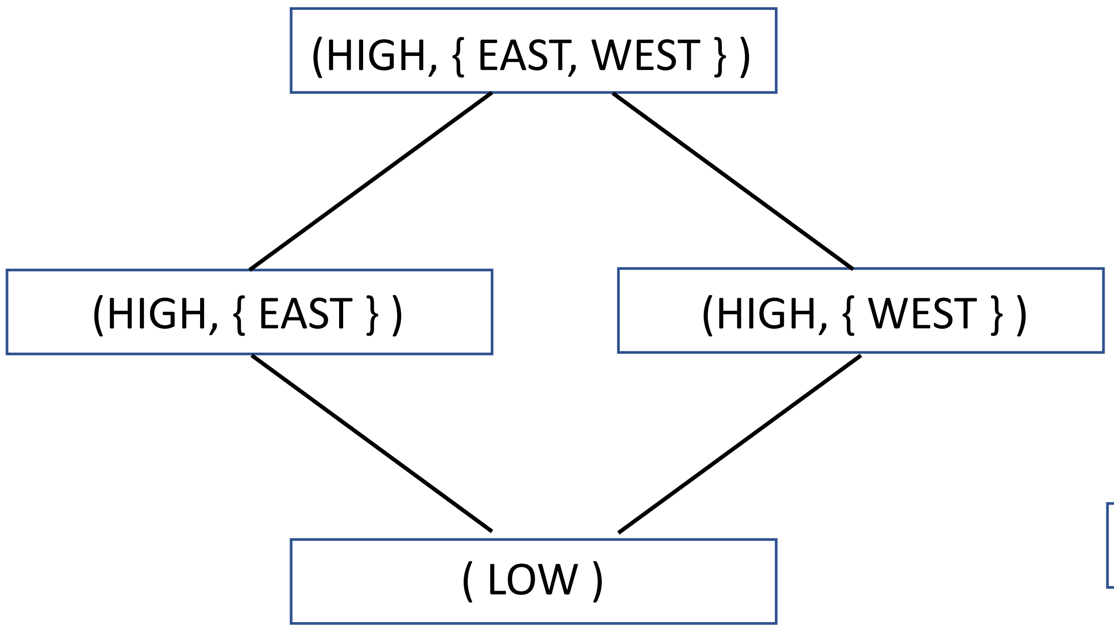
Composition of Bell-LaPadula

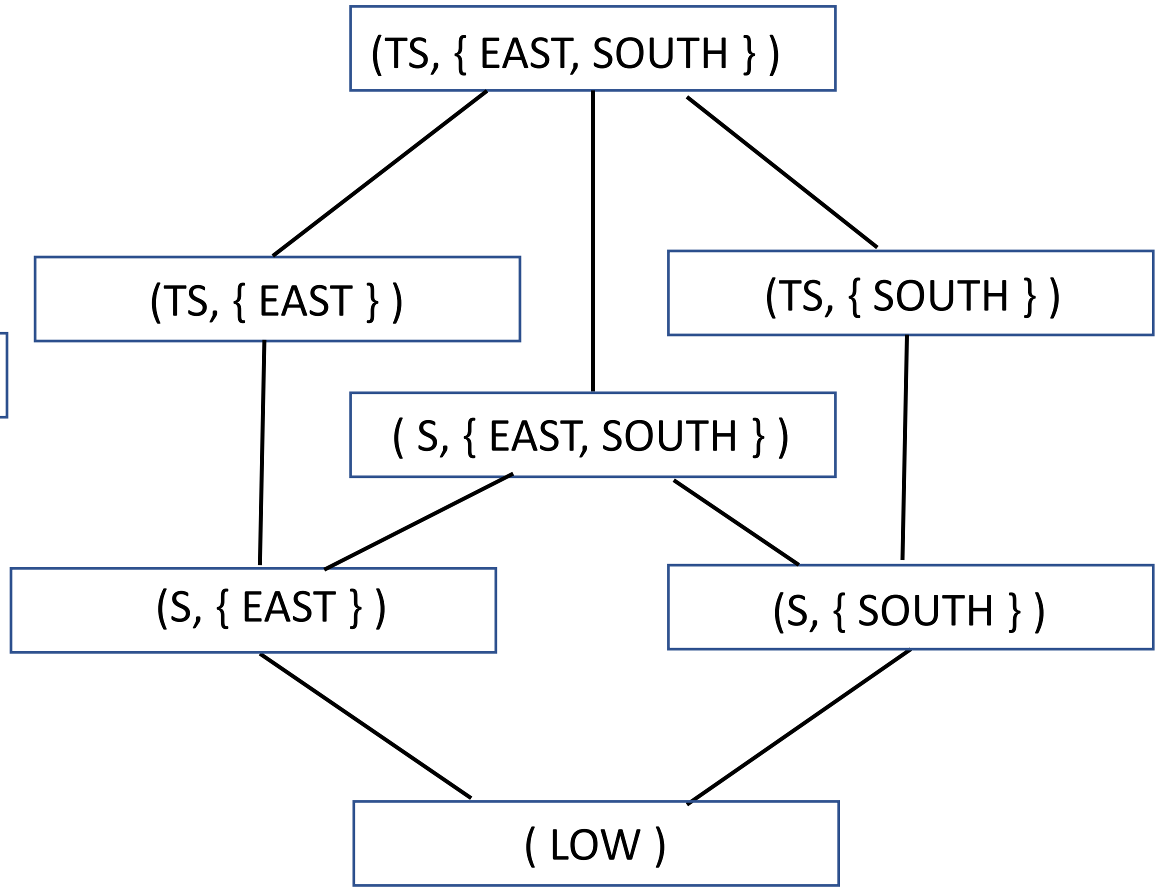
- Why?
  - Some standards require secure components to be connected to form secure (distributed, networked) system
- Question
  - Under what conditions is this secure?
- Assumptions
  - Implementation of systems precise with respect to each system’s security policy

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


(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) = ∅
  - 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($X \cup Y$):
  - $AS(X \cup Y)^+$ = { (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 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: 2-Bit Machine

- 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^*: $C^* \times \Sigma \rightarrow O$ similarly