Lecture 17

- Nondeducibility
- Composition and restrictiveness
- What is identity
- Multiple names for one thing
- Different contexts, environments
- Pseudonymity and anonymity

Nondeducibility

- Noninterference: do state transitions caused by high level commands interfere with sequences of state transitions caused by low level commands?
- Really case about inputs and outputs:
  - Can low level subject deduce *anything* about high level outputs from a set of low level outputs?

Example: 2-Bit System

- *High* operations change only *High* bit
  - Similar for *Low*
- $\sigma_0 = (0, 0)$
- Commands (Heidi, xor_1), (Lara, xor_0), (Lara, xor_1), (Lara, xor_0), (Heidi, xor_1), (Lara, xor_0)
  - Both bits output after each command
- Output is: 00 10 10 11 11 01 01

Security

- Not noninterference-secure w.r.t. Lara
  - Lara sees output as 0001111
  - Delete *High* and she sees 00111
- But Lara still cannot deduce the commands deleted
  - Don't affect values; only lengths
- So it is deducibly secure
  - Lara can't deduce the commands Heidi gave

Event System

- 4-tuple (E, I, O, T)
  - E set of events
  - $I \subseteq E$ set of input events
  - $O \subseteq E$ set of output events
  - T set of all finite sequences of events legal within system
- E partitioned into H, L
  - H set of *High* events
  - L set of *Low* events

More Events …

- $H \cap I$ set of *High* inputs
- $H \cap O$ set of *High* outputs
- $L \cap I$ set of *Low* inputs
- $L \cap O$ set of *Low* outputs
- T_{Low} set of all possible sequences of *Low* events that are legal within system
- π_L: $T \rightarrow T_{Low}$ projection function deleting all *High* inputs from trace
  - *Low* observer should not be able to deduce anything about *High* inputs from trace $t_{Low} \in T_{low}$

Deducibly Secure

- System deducibly secure if, for every trace $t_{Low} \in T_{Low}$, the corresponding set of high level traces contains every possible trace $t \in T$ for which $\pi_L(t) = t_{Low}$
  - Given any t_{Low}, the trace $t \in T$ producing that t_{Low} is equally likely to be *any* trace with $\pi_L(t) = t_{Low}$

Example

- Back to our 2-bit machine
  - Let xor0, xor1 apply to both bits
  - Both bits output after each command
- Initial state: (0, 1)
- Inputs: $1_H 0_L 1_L 0_H 1_L 0_L$
- Outputs: 10 10 01 01 10 10
- Lara (at *Low*) sees: 001100
  - Does not know initial state, so does not know first input; but can deduce fourth input is 0
- Not deducibly secure

Example

- Now xor_0, xor_1 apply only to state bit with same level as user
- Inputs: $1_H 0_L \; 1_L 0_H \; 1_L 0_L$
- Outputs: 10 11 11 10 11
- Lara sees: 01101
- She cannot deduce *anything* about input
  - Could be $0_H 0_L \; 1_L 0_H \; 1_L 0_L$ or $0_L 1_H \; 1_L 0_H \; 1_L 0_L$ for example
- Deducibly secure

Security of Composition

- In general: deducibly secure systems not composable
- *Strong noninterference*: deducible security + requirement that no *High* output occurs unless caused by a *High* input
  - Systems meeting this property *are* composable

Example

- 2-bit machine done earlier does not exhibit strong noninterference
  - Because it puts out *High* bit even when there is no *High* input
- Modify machine to output only state bit at level of latest input
  - *Now* it exhibits strong noninterference

Problem

- Too restrictive; it bans some systems that are *obviously* secure
- Example: System *upgrade* reads *Low* inputs, outputs those bits at *High*
  - Clearly deducibly secure: low level user sees no outputs
  - Clearly does not exhibit strong noninterference, as no high level inputs!

Remove Determinism

- Previous assumption
  - Input, output synchronous
  - Output depends only on commands triggered by input
    - Sometimes absorbed into commands …
  - Input processed one datum at a time
- Not realistic
  - In real systems, lots of asynchronous events

Generalized Noninterference

- Nondeterministic systems meeting noninterference property meet *generalized noninterference-secure property*
  - More robust than deducible security because minor changes in assumptions affect whether system is deducibly secure

Example

- System with *High* Holly, *Low* lucy, text file at *High*
  - File fixed size, symbol b marks empty space
  - Holly can edit file, Lucy can run this program:

```
while true do begin
     n := read_integer_from_user;
     if n > file_length or char_in_file[n] = b then
            print random_character;
     else
            print char_in_file[n];
end;
```

Security of System

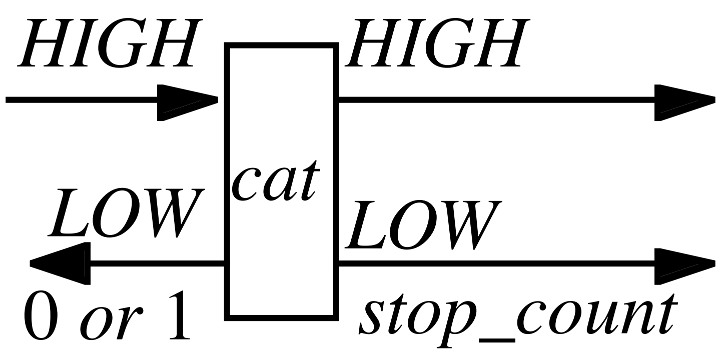
- Not noninterference-secure
  - High level inputs—Holly's changes—affect low level outputs
- *May* be deducibly secure
  - Can Lucy deduce contents of file from program?
  - If output meaningful ("This is right") or close ("Thes is riqht"), yes
  - Otherwise, no
- So deducibly secure depends on which inferences are allowed

Composition of Systems

- Does composing systems meeting generalized noninterference-secure property give you a system that also meets this property?
- Define two systems (*cat*, *dog*)
- Compose them

First System: *cat*

- Inputs, outputs can go left or right
- After some number of inputs, *cat* sends two outputs
  - First *stop_count*
  - Second parity of *High* inputs, outputs

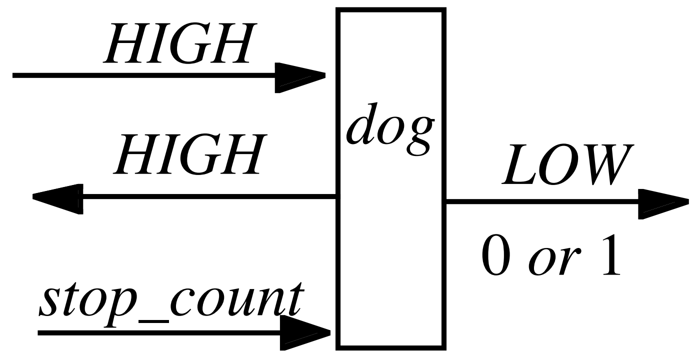


Noninterference-Secure?

- If even number of *High* inputs, output could be:
  - 0 (even number of outputs)
  - 1 (odd number of outputs)
- If odd number of *High* inputs, output could be:
  - 0 (odd number of outputs)
  - 1 (even number of outputs)
- High level inputs do not affect output
  - So noninterference-secure

Second System: *dog*

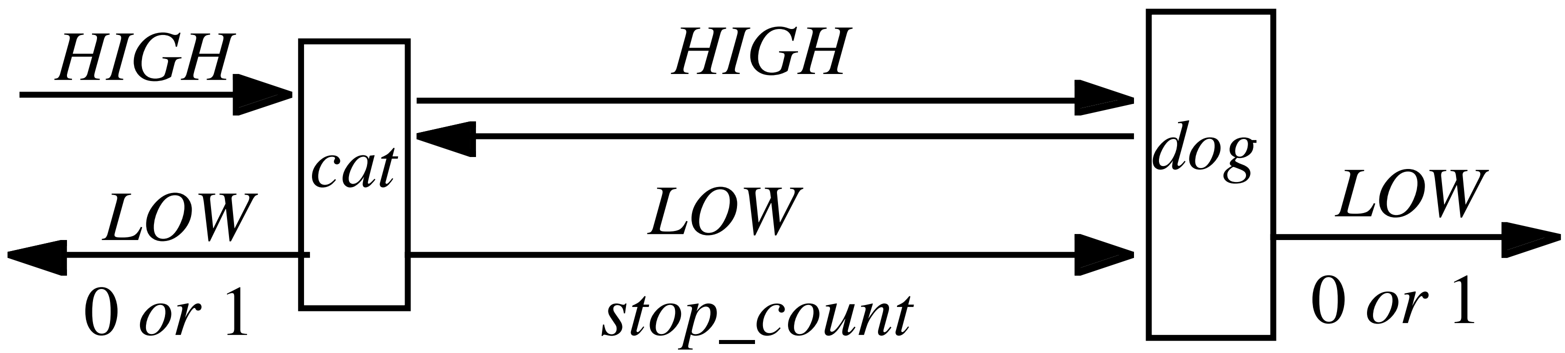
- High outputs to left
- Low outputs of 0 or 1 to right
- *stop_count* input from the left
  - When it arrives, *dog* emits 0 or 1



Noninterference-Secure?

- When *stop_count* arrives:
  - May or may not be inputs for which there are no corresponding outputs
  - Parity of *High* inputs, outputs can be odd or even
  - Hence *dog* emits 0 or 1
- High level inputs do not affect low level outputs
  - So noninterference-secure

Compose Them



- Once sent, message arrives
  - But *stop_count* may arrive before all inputs have generated corresponding outputs
  - If so, even number of *High* inputs and outputs on *cat*, but odd number on *dog*
- Four cases arise

The Cases

- *cat*, odd number of inputs, outputs; *dog*, even number of inputs, odd number of outputs
  - Input message from *cat* not arrived at *dog*, contradicting assumption
- *cat*, even number of inputs, outputs; *dog*, odd number of inputs, even number of outputs
  - Input message from *dog* not arrived at *cat*, contradicting assumption

The Cases

- cat, odd number of inputs, outputs; dog, odd number of inputs, even number of outputs
  - dog sent even number of outputs to cat, so cat has had at least one input from left
- cat, even number of inputs, outputs; dog, even number of inputs, odd number of outputs
  - dog sent odd number of outputs to cat, so cat has had at least one input from left

The Conclusion

- Composite system *catdog* emits 0 to left, 1 to right (or 1 to left, 0 to right)
  - Must have received at least one input from left
- Composite system *catdog* emits 0 to left, 0 to right (or 1 to left, 1 to right)
  - Could not have received any from left
- So, *High* inputs affect *Low* outputs
  - Not noninterference-secure

Feedback-Free Systems

- System has n distinct components
- Components c_i, c_j connected if any output of c_i is input to c_j
- System is *feedback-free* if for all c_i connected to c_j, c_j not connected to any c_i
  - Intuition: once information flows from one component to another, no information flows back from the second to the first

Feedback-Free Security

- *Theorem*: A feedback-free system composed of noninterference-secure systems is itself noninterference-secure

Some Feedback

- *Lemma*: A noninterference-secure system can feed a high level output *o* to a high level input *i* if the arrival of *o* at the input of the next component is delayed until *after* the next low level input or output
- *Theorem*: A system with feedback as described in the above lemma and composed of noninterference-secure systems is itself noninterference-secure

Why Didn't They Work?

- For compositions to work, machine must act same way regardless of what precedes low level input (high, low, nothing)
- *dog* does not meet this criterion
  - If first input is *stop_count*, *dog* emits 0
  - If high level input precedes *stop_count*, *dog* emits 0 or 1

State Machine Model

- 2-bit machine, levels *High*, *Low*, meeting 4 properties:

1. For every input i_k, state σ_j, there is an element $c_m \in C^*$ such that $T^*(c_m, \sigma_j) = \sigma_n$, where $\sigma_n \neq \sigma_j$
   - T^* is total function, inputs and commands always move system to a different state

Property 2

- There is an equivalence relation $\equiv$ such that:
  - If system in state σ_i and high level sequence of inputs causes transition from σ_i to σ_j, then $\sigma_i \equiv \sigma_j$
  - If $\sigma_i \equiv \sigma_j$ and low level sequence of inputs $i_1, \ldots, i_n$ causes system in state σ_i to transition to σ_i', then there is a state σ_j' such that $\sigma_i' \equiv \sigma_j'$ and the inputs $i_1, \ldots, i_n$ cause system in state σ_j to transition to σ_j'
- $\equiv$ holds if low level projections of both states are same

Property 3

- Let $\sigma_i \equiv \sigma_j$. If high level sequence of outputs $o_1, \ldots, o_n$ indicate system in state σ_i transitioned to state σ_i', then for some state σ_j' with $\sigma_j' \equiv \sigma_i'$, high level sequence of outputs $o_1', \ldots, o_m'$ indicates system in σ_j transitioned to σ_j'
  - High level outputs do not indicate changes in low level projection of states

Property 4

- Let $\sigma_i \equiv \sigma_j$, let c, d be high level output sequences, e a low level output. If ced indicates system in state σ_i transitions to σ_i', then there are high level output sequences c' and d' and state σ_j' such that $c'ed'$ indicates system in state σ_j transitions to state σ_j'
  - Intermingled low level, high level outputs cause changes in low level state reflecting low level outputs only

Restrictiveness

- System is *restrictive* if it meets the preceding 4 properties

Composition

- Intuition: by 3 and 4, high level output followed by low level output has same effect as low level input, so composition of restrictive systems should be restrictive

Composite System

- System M_1's outputs are M_2's inputs
- μ_{1i}, μ_{2i} states of M_1, M_2
- States of composite system pairs of M_1, M_2 states (μ_{1i}, μ_{2i})
- e event causing transition
- e causes transition from state (μ_{1a}, μ_{2a}) to state (μ_{1b}, μ_{2b}) if any of 3 conditions hold

Conditions

1. M_1 in state μ_{1a} and *e* occurs, M_1 transitions to μ_{1b}; *e* not an event for M_2; and $\mu_{2a} = \mu_{2b}$
2. M_2 in state μ_{2a} and *e* occurs, M_2 transitions to μ_{2b}; *e* not an event for M_1; and $\mu_{1a} = \mu_{1b}$
3. M_1 in state μ_{1a} and *e* occurs, M_1 transitions to μ_{1b}; M_2 in state μ_{2a} and *e* occurs, M_2 transitions to μ_{2b}; *e* is input to one machine, and output from other

Intuition

- Event causing transition in composite system causes transition in at least 1 of the components
- If transition occurs in exactly one component, event must not cause transition in other component when not connected to the composite system

Equivalence for Composite

- Equivalence relation for composite system

  $(\sigma_a, \sigma_b) \equiv_C (\sigma_c, \sigma_d)$ iff $\sigma_a \equiv \sigma_c$ and $\sigma_b \equiv \sigma_d$
- Corresponds to equivalence relation in property 2 for component system

Identity

- *Principal*: a unique entity
- *Identity*: specifies a principal
- *Authentication*: binding of a principal to a representation of identity internal to the system
  - All access, resource allocation decisions assume binding is correct

Files and Objects

- Identity depends on system containing object
- Different names for one object
  - Human use, *eg*. file name
  - Process use, *eg*. file descriptor or handle
  - Kernel use, *eg*. file allocation table entry, inode

More Names

- Different names for one context
  - Human: aliases, relative *vs*. absolute path names
  - Kernel: deleting a file identified by name can mean two things:
    - Delete the object that the name identifies
    - Delete the name given, and do not delete actual object until *all* names have been deleted
- Semantics of names may differ

Example: Names and Descriptors

- Interpretation of UNIX file name
  - Kernel maps name into an inode using iterative procedure
  - Same name can refer to different objects at different times without being deallocated
    - Causes race conditions
- Interpretation of UNIX file descriptor
  - Refers to a specific inode
  - Refers to same inode from creation to deallocation

Example: Different Systems

- Object name must encode location or pointer to location
  - *rsh*, *ssh* style: *host*:*object*
  - URLs: *protocol*://*host*/*object*
- Need not name actual object
  - *rsh*, *ssh* style may name pointer (link) to actual object
  - URL may forward to another host

Users

- Exact representation tied to system
- Example: UNIX systems
  - Login name: used to log in to system
    - Logging usually uses this name
  - User identification number (UID): unique integer assigned to user
    - Kernel uses UID to identify users
    - One UID per login name, but multiple login names may have a common UID

Multiple Identities

- UNIX systems again
  - Real UID: user identity at login, but changeable
  - Effective UID: user identity used for access control
    - Setuid changes effective UID
  - Saved UID: UID before last change of UID
    - Used to implement least privilege
    - Work with privileges, drop them, reclaim them later
  - Audit/Login UID: user identity used to track original UID
    - Cannot be altered; used to tie actions to login identity

Groups

- Used to share access privileges
- First model: alias for set of principals
  - Processes assigned to groups
  - Processes stay in those groups for their lifetime
- Second model: principals can change groups
  - Rights due to old group discarded; rights due to new group added

Roles

- Group with membership tied to function
  - Rights given are consistent with rights needed to perform function
- Uses second model of groups
- Example: DG/UX
  - User *root* does not have administration functionality
  - System administrator privileges are in *sysadmin* role
  - Network administration privileges are in *netadmin* role
  - Users can assume either role as needed