50.042 FCS Summer 2024
Lecture 18 – Information Flow I

Felix LOH
Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

With selected materials adapted from: *Computer Security: Art and Science, by M. Bishop (2nd Edition)*

Information flow models: motivation

- Access controls can constrain the rights of a user, but they cannot constrain the flow of information about a system
- So we’ll introduce information flow models and information flow policies that abstract the essence of security policies

Information flow policy

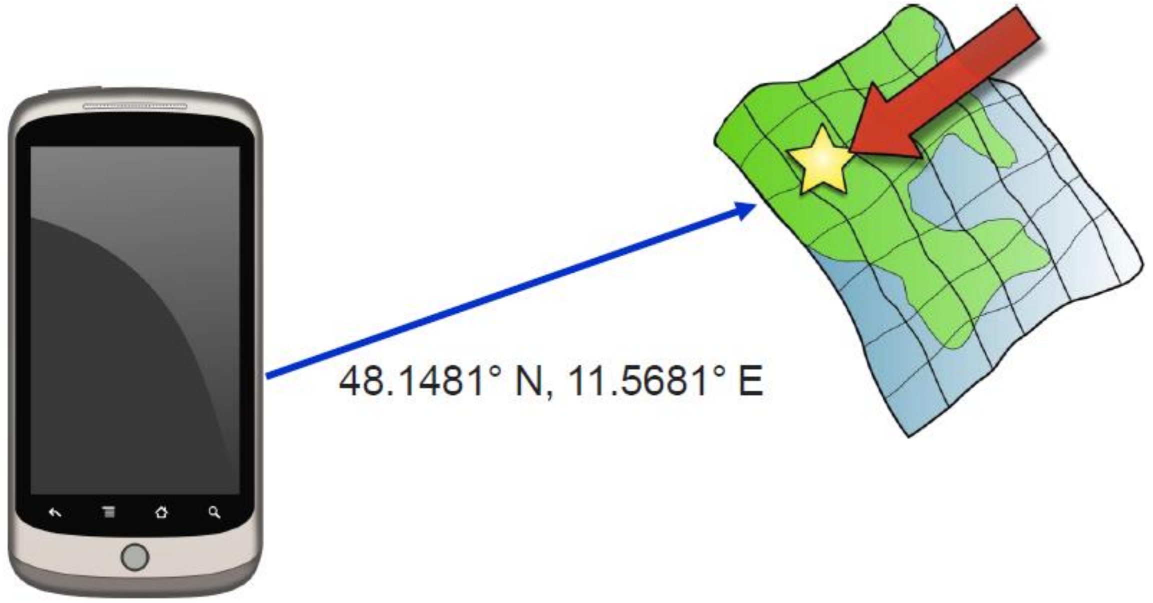
- Information flow policies define the way information moves throughout a system.
- We can design information flow policies to prevent information from flowing to a user who is not authorized to receive it (confidentiality)
- Or, alternatively, only to processes that are no more trustworthy than the data (integrity)

Information flow policy

- For this lecture, we are going to look at security through an alternative viewpoint
- Basically, the idea is that a system is secure if groups of subjects cannot *interfere* with one another

- So we'll view a system as a state machine (i.e. a FSM)

Information flow – example



- E.g. consider a navigation app
  - The app wants the coordinates of your current location in order to provide a service; in other words, the app needs access to your current location
  - Are the coordinates transmitted to untrusted third parties afterwards?

Practical interpretation of unwanted flows

- Exploiting a vulnerability that alters data is an <u>integrity</u> violation

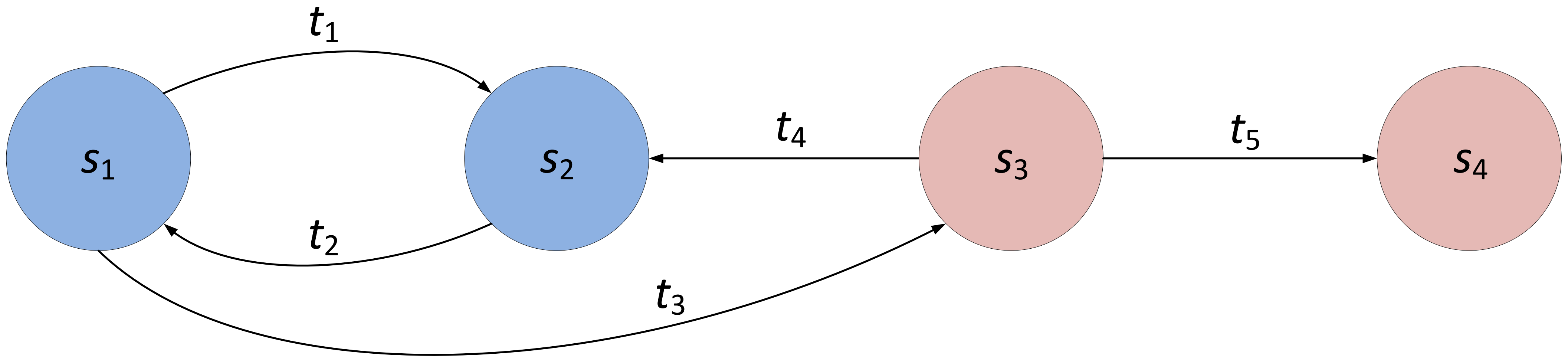
- An attack that leaks information is a <u>confidentiality</u> violation

Unauthorized flows: example

$$\begin{tikzcd}
s_1 \arrow[r, bend left, "t_1"] \arrow[rr, bend right=40, "t_3"'] & s_2 \arrow[l, bend left, "t_2"] & s_3 \arrow[l, "t_4"'] \arrow[r, "t_5"] & s_4
\end{tikzcd}$$

- Consider a system represented by the FSM above
  - It consists of 4 states and 5 transitions (or edges)
- Suppose a security policy partitions the 4 states shown above into a set of <u>authorized states</u> $A = \{s_1, s_2\}$ and a set of <u>unauthorized states</u> $UA = \{s_3, s_4\}$

Unauthorized flows: example



- This system is **not** secure, because regardless of which authorized state it starts in (either s_1 or s_2), the entity can enter an unauthorized state, because of the edge/transition t_3
- However, if the transition t_3 is removed, then the system would be secure, because the entity cannot enter an unauthorized state from an authorized state

Information flow model

- Now we'll discuss the details of an information flow model…
- As mentioned earlier, let's define a system as a state machine (FSM), consisting of the following sets:
  - Subjects $S = \{ s_i \}$
  - States $\Sigma = \{ \sigma_i \}$
  - Outputs $O = \{ o_i \}$
  - Commands $Z = \{ z_i \}$
  - State transition commands $C = S \times Z$

- Note that we don't need to define any inputs, because either the inputs select the commands to be executed, or the inputs themselves can be encoded in the set of state transition commands

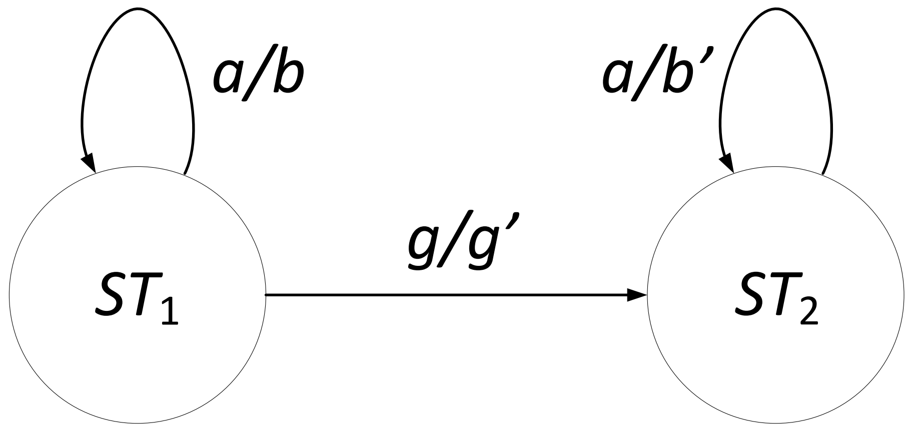
Information flow model – functions

- State transition function T: $C \times \Sigma \rightarrow \Sigma$
  - This function describes the effect of executing a state transition command c when in state σ
  - If the system is initially in state σ_0, the next state is σ_1

- Output function P: $C \times \Sigma \rightarrow O$
  - This function describes the output of the FSM when executing a state transition command c in state σ

Information flow model – functions

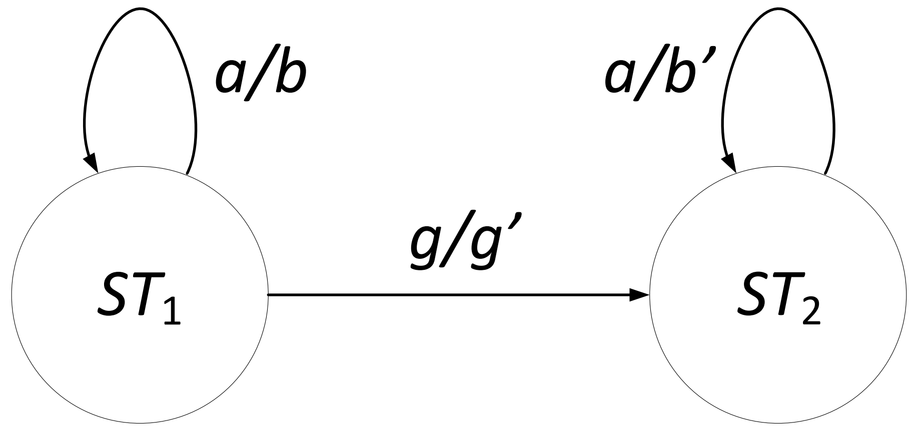
- Semantics example…


$$\begin{tikzcd} ST_1 \arrow[loop above, "a/b"] \arrow[r, "g/g'"] & ST_2 \arrow[loop above, "a/b'"] \end{tikzcd}$$

- Suppose we have the Mealy FSM above, described as follows:
  - The initial state, $\sigma_0 = ST_1$
  - The state transition function, *T*, takes a state transition command (i.e. an input) and returns the next state
  - The output function, *P*, takes a state transition command and shows the associated output
  - Note: the next state and the output are both dependent on the input and the current state σ

Information flow model – functions

- Semantics example, continued…



- So, $T(a, ST_1) = ST_1$ and $T(g, ST_1) = ST_2$
- Likewise, $P(a, ST_1) = b$ and $P(a, ST_2) = b'$

Information flow model – states and outputs

- For the state transition function, T: $C \times \Sigma \rightarrow \Sigma$ is inductive with respect to its second argument:

  $T(c_0, \sigma_0) = \sigma_1$

  $T(c_{i+1}, \sigma_{i+1}) = T(c_{i+1}, T(c_i, \sigma_i))$, for $i = 0, 1, 2, \ldots$

- Let C^* be the set of possible <u>sequences</u> of state transition commands in C. Then we define a function T^* as follows:

  T^*: $C^* \times \Sigma \rightarrow \Sigma$ and $c^* = c_0, \ldots, c_n$ implies that

  $T^*(c^*, \sigma_0) = T(c_n, T(c_{n-1}, \ldots, T(c_0, \sigma_0) \ldots)) = \sigma_{n+1}$

- For some initial state σ_i, we can generalize the above definition to:

  $T^*(c^*, \sigma_i) = T(c_n, T(c_{n-1}, \ldots, T(c_0, \sigma_i) \ldots)) = \sigma_{i+n+1}$

Information flow model – states and outputs

- There is a similar definition with respect to the <u>outputs</u> (also inductive with respect to its second argument)

- Let O^* be the set of possible <u>sequences</u> of outputs after executing some sequence $c^* \in C^*$. Then we define a function P^* as follows :

  $P^*: C^* \times \Sigma \rightarrow O^*$ and $c^* = c_0, \ldots, c_n$ implies that

  $P^*(c^*, \sigma_0) = o_1, o_2, \ldots, o_n, o_{n+1}$

- For some initial state σ_i, we can generalize the above definition to:

  $P^*(c^*, \sigma_i) = o_{i+1}, o_{i+2}, \ldots, o_{i+n}, o_{i+n+1}$

Information flow model – example

- A 2-bit FSM…
- 2 bits of state information: *H* (high bit), *L* (low bit)
  - The system state is (*H*, *L*), where *H*, *L* ∈ {0, 1}
- 2 users: Holly (high privilege), Lucy (low privilege)
  - Holly can **read** both the high and low bit information
  - Lucy can only **read** the low bit information
- Output is a 2-bit string that concatenates the 2 state bits (*H* is the MSB) of the destination state
- 2 commands: *xor0*, *xor1*
  - Each command bitwise XORs the two state bits with 0 or 1 respectively, regardless of whether Holly or Lucy issued the command

Information flow model – example

- Subjects, *S* = {Holly, Lucy}
- States, *Σ* = {(0, 0), (0, 1), (1, 0), (1, 1)}
- Outputs, *O* = {00, 01, 10, 11}
- Commands, *Z* = {*xor0*, *xor1*}
- *C* = {(Holly, *xor0)*, (Holly, *xor1*), (Lucy, *xor0)*, (Lucy, *xor1*)}
- State transition function:

Commands	**Current State (*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)

Information flow model – example

- Let σ_0 = (0, 1)
- 3 commands issued:
  1. Holly issues *xor0*
  2. Lucy issues *xor1*
  3. Holly issues *xor1*

- So c^* = (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)
- Output $P^*(c^*, \sigma_0)$ = 01, 10, 01
  - Holly can see both output bits: 01, 10, 01
  - But Lucy should only see the low bit: 1, 0, 1
  - This provides a rationale for a definition for a projection function…

Information flow model – projection

- $T^*(c^*, \sigma_i)$: sequence of state transitions for a system, starting from state σ_i, resulting in some final state σ_{i+n+1}
- $P^*(c^*, \sigma_i)$: sequence of corresponding outputs for a system, starting from state σ_i

- Then we have:

  $proj(s, c^*, \sigma_i)$: the sequence of outputs in $P^*(c^*, \sigma_i)$ that subject s <u>is authorized to see</u>

  - This sequence of outputs is in the same order as the sequence of $P^*(c^*, \sigma_i)$
  - Basically a projection of outputs for subject s
  - Intuition: the projection function removes outputs that s is not supposed to see

Information flow model – projection example

- From our earlier example:
- $\sigma_0 = (0, 1)$
- c^* = (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)
- $P^*(c^*, \sigma_0)$ = 01, 10, 01

- Then *proj*(Holly, c^*, σ_0) = 01, 10, 01 and *proj*(Lucy, c^*, σ_0) = 1, 0, 1
  - Holly can see both the high and low bit outputs
  - Lucy **cannot** see the high bit outputs

Information flow model – purge

- Now let’s define another function for our model – the purge function
- $G \subseteq S$, G is a subset of subjects
- $A \subseteq Z$, A is a subset of commands

- Then we have:
- $\pi_G(c^*)$: subsequence of c^* with all elements (s, z), $s \in G$ deleted
- $\pi_A(c^*)$: subsequence of c^* with all elements (s, z), $z \in A$ deleted
- $\pi_{G, A}(c^*)$: subsequence of c^* with all elements (s, z), $s \in G$ and $z \in A$ deleted
- Note: here, “deleted” means that the element is removed from the observable sequence (the “deleted” command has still been executed)

Information flow model – purge examples

- Again, from our earlier example:
- $\sigma_0 = (0, 1)$
- c^* = (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)

- Then we have:
- $\pi_{Lucy}(c^*)$ = (Holly, *xor0*), (Holly, *xor1*)
- $\pi_{Lucy,\ xor1}(c^*)$ = (Holly, *xor0*), (Holly, *xor1*)
- $\pi_{Lucy,\ xor0}(c^*)$ = (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)

Information flow model – purge examples

- Also:
- $\pi_{Holly}(c^*)$ = (Lucy, *xor1*)
- $\pi_{Holly,\ xor1}(c^*)$ = (Holly, *xor0*), (Lucy, *xor1*)
- $\pi_{Holly,\ xor0}(c^*)$ = (Lucy, *xor1*), (Holly, *xor1*)

- $\pi_{xor0}(c^*)$ = (Lucy, *xor1*), (Holly, *xor1*)
- $\pi_{xor1}(c^*)$ = (Holly, *xor0*)

Information flow model – non-interference

- Basic intuition of non-interference, using our earlier example:
  - There is **no** interference if the set of outputs that Lucy can observe corresponds to the set of commands that she can observe

- Formal definition:

  Let G, $G' \subseteq S$ with $G \neq G'$, and $A \subseteq Z$. The subjects in G executing commands in A are **non-interfering** with subjects in G' (which is written as $A, G :| G'$), if and only if for all $c^* \in C^*$, and for all $s \in \boldsymbol{G'}$,

  $proj(s, c^*, \sigma_i) = proj(s, \pi_{\boldsymbol{G}, A}(c^*), \sigma_i)$

Information flow model – non-interference and security

- With the formal definition of non-interference presented, we can now say that:
- A system is *secure* (with respect to the subjects in *G’* and the commands in *A*), if and only if every subject in *G* is *non-interfering* with all subjects in *G’*

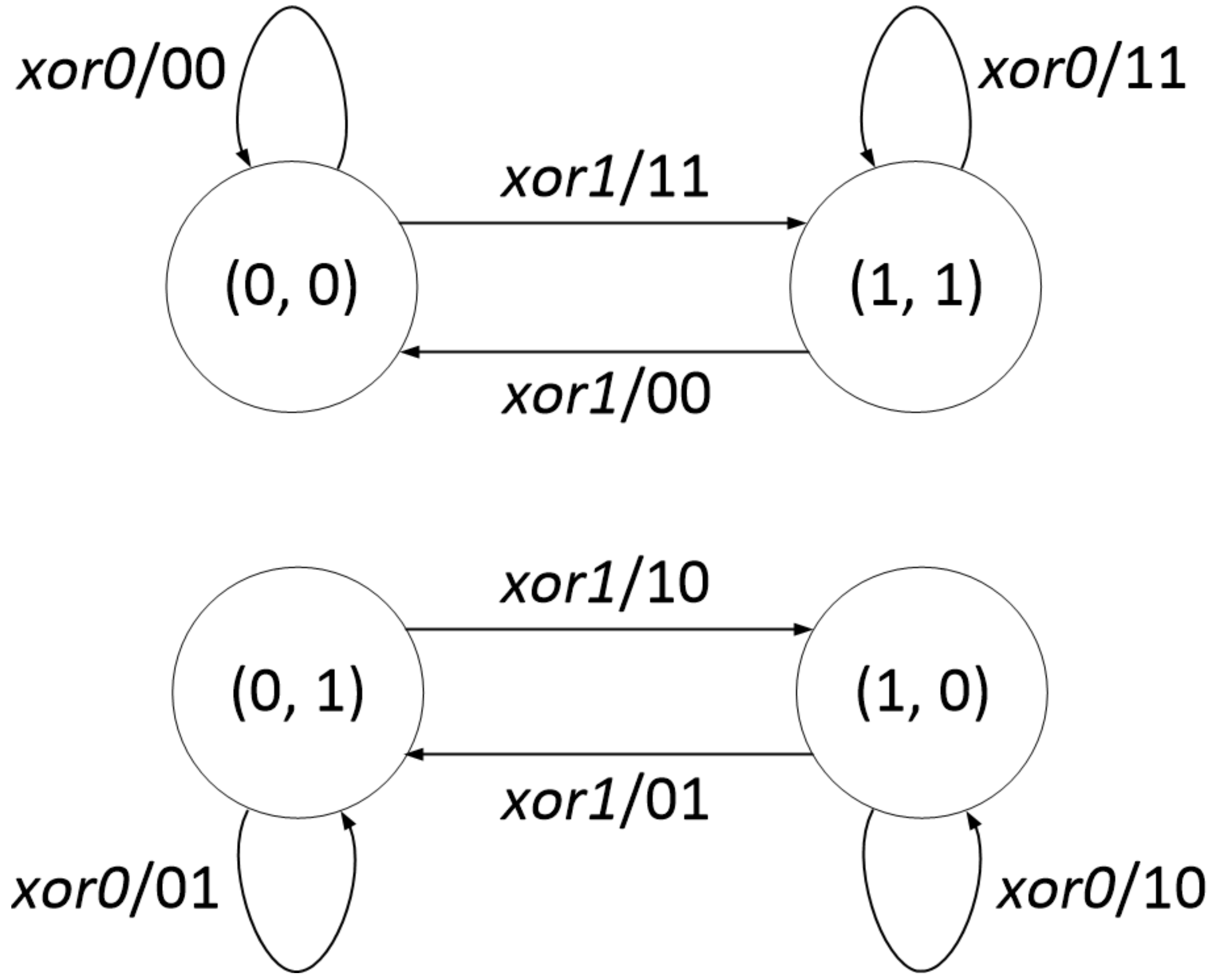
- So in a secure system, it is not possible for a subject in *G’* to deduce information about a subject in *G*
- Note that it is possible for a subject in ***G*** to deduce information about a subject in ***G’***
  - We can think of subjects in *G* as system administrators with elevated privileges, while subjects in *G’* are just ordinary users

Exercise (using our earlier example)

1) Draw the state diagram for the system represented by the 2-bit FSM we discussed in the previous slides.
2) State whether this system is secure (or not secure) with respect to Lucy, i.e. *G’* = {Lucy}.
   - If it is secure, explain why.
   - If it is not secure, show a counter example illustrating the case where the system is insecure.

Exercise (using our earlier example)

1) The state diagram is as follows:



Exercise (using our earlier example)

2) This system is <u>not secure</u> with respect to Lucy. A counterexample is as follows:

- $\sigma_0 = (0, 1)$
- $c^* =$ (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)
- $P^*(c^*, \sigma_0) = 0\boxed{1}, 1\boxed{0}, 0\boxed{1}$

- Let G = {Holly}, G' = {Lucy}, $A = Z$ = {*xor0*, *xor1*}
- Then $\pi_{\text{Holly}, A}(c^*)$ = (Lucy, *xor1*)   purge Holly $c^* = (\text{Lucy}, \text{xor1}) \Rightarrow 0$
- We have *proj*(Lucy, $\underline{c^*}$, $\underline{\sigma_0}$) = 1, 0, 1
- But *proj*(Lucy, $\pi_{\text{Holly}, A}(c^*)$, σ_0) = *proj*(Lucy, (Lucy, *xor1*), σ_0) = 0
- So *proj*(Lucy, c^*, σ_0) $\neq$ *proj*(Lucy, $\pi_{\text{Holly}, A}(c^*)$, σ_0)

Exercise (using our earlier example)

- $proj(\text{Lucy}, c^*, \sigma_0) \neq proj(\text{Lucy}, \pi_{\text{Holly}, A}(c^*), \sigma_0)$
- Thus, the statement {*xor0*, *xor1*}, {Holly} :| {Lucy} is <u>false</u>, which implies that Holly is interfering with Lucy

- This is because the commands issued to change the *H* bit also affect the *L* bit

- In other words, even though Lucy should not be aware of Holly's existence, Lucy can tell that there some other user exists (Holly), from Lucy's observation of the system's output sequence

Fixing the insecure system in the example

- How can we modify the system, so that it is secure (with respect to Lucy)?

- Let's alter the system, such that Holly can only modify the *H* bit and Lucy can only modify the *L* bit
- Also, we will modify the output of the system, such that a security level is associated with each bit of the output:
  - Holly can see bits with security level 'H' and 'L'
  - Lucy can only see bits with security level 'L'
  - The security level of each output bit is indicated by a subscript 'H' or a subscript 'L' for that bit

Fixing the insecure system in the example

- New state transition function:

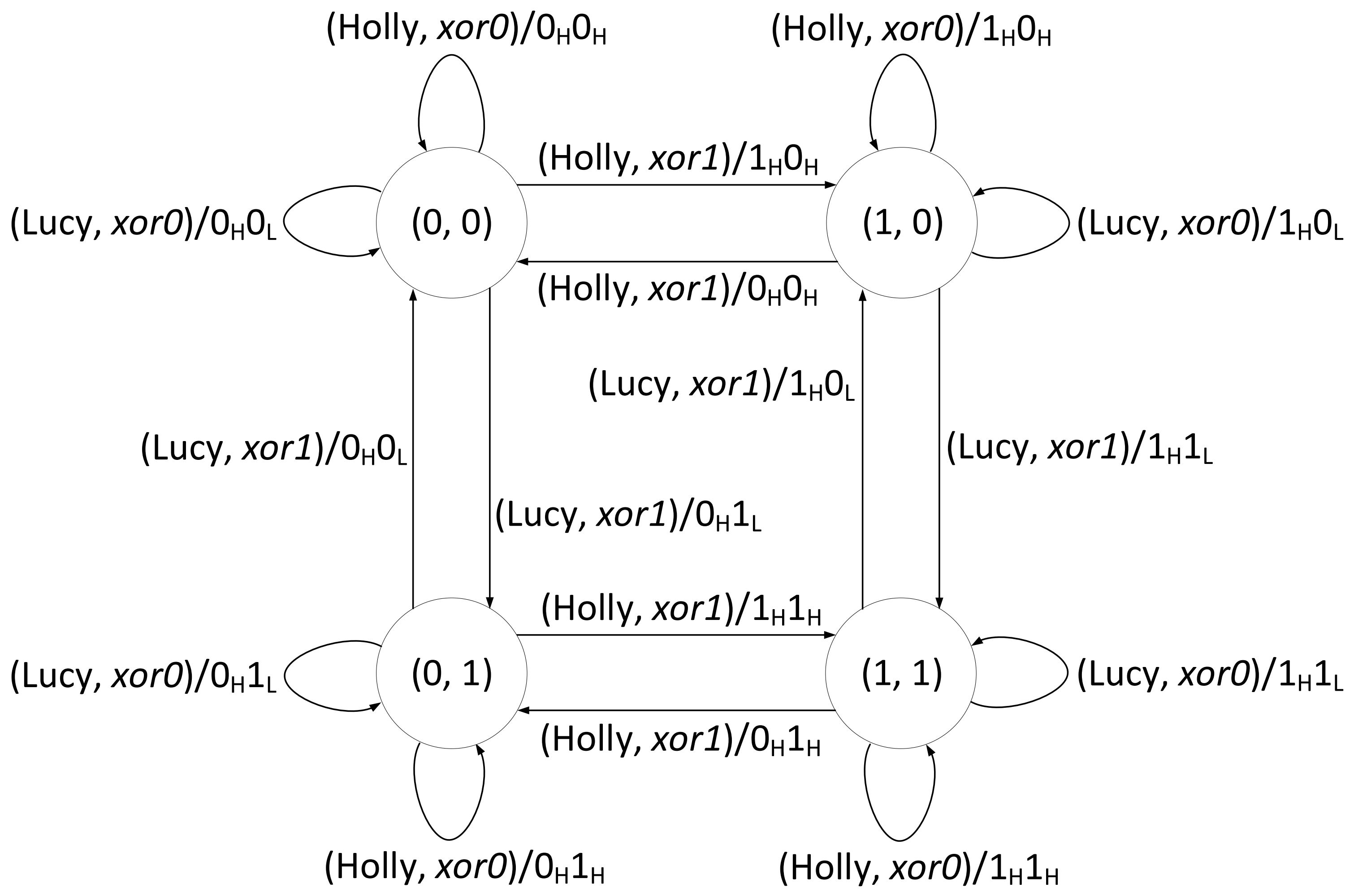
Commands	Current State (*H*, *L*)			
	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Holly, *xor0*	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Holly, *xor1*	(1, 0)	(1, 1)	(0, 0)	(0, 1)
Lucy, *xor0*	(0, 0)	(0, 1)	(1, 0)	(1, 1)
Lucy, *xor1*	(0, 1)	(0, 0)	(1, 1)	(1, 0)

only change H bit $\rightarrow$

only chng L bit $\rightarrow$

Fixing the insecure system in the example

- State diagram for the new FSM:



Fixing the insecure system in the example

- $\sigma_0 = (0, 1)$
- $c^* =$ (Holly, *xor0*), (Lucy, *xor1*), (Holly, *xor1*)
- $P^*(c^*, \sigma_0) = 0_H 1_H, 0_H 0_L, 1_H 0_H$

- Let $G = \{\text{Holly}\}$, $G' = \{\text{Lucy}\}$, $A = Z = \{xor0, xor1\}$
- Then $\pi_{\text{Holly}, A}(c^*) =$ (Lucy, *xor1*)
- Now we have $proj(\text{Lucy}, c^*, \sigma_0) = 0_L$
- $proj(\text{Lucy}, \pi_{\text{Holly}, A}(c^*), \sigma_0) = proj(\text{Lucy}, (\text{Lucy}, xor1), \sigma_0) = 0_L$
- So now $proj(\text{Lucy}, c^*, \sigma_0) = proj(\text{Lucy}, \pi_{\text{Holly}, A}(c^*), \sigma_0)$

Fixing the insecure system in the example

- So this system is secure with respect to Lucy, for the chosen σ_0 and *c**
- It can be shown that this system will also be secure with respect to Lucy for all possible initial states σ_0 and command sequences *c**

- Thus, for this system, the statement {*xor0*, *xor1*}, {Holly} :| {Lucy} is <u>true</u>, which means that Holly is non-interfering with Lucy

- The commands executed by Holly and the resulting outputs of the system are not visible to Lucy. As a result, Lucy is unable to deduce any information about Holly in this system

A more precise definition of information flow

- Now, we know how to determine whether a system is secure
- So how do we develop mechanisms to detect and stop flows of information that violate a security policy?

- In the next lecture, we will <u>precisely</u> define information flow, then we can discuss mechanisms to detect and stop flows of information that violate a security policy