Week 3 Homework due on Friday Oct 5, 18:59 Hour

Group 5

Wong Ann Yi (1004000)
Liu Bowen (1004028)
Tan Chin Leong Leonard (1004041)

Exercise 1

Consider a modification of the 2-bit machine of Classwork Exercise 1 (Week 3), where Holly can only alter the H bit and Lucy only the L bit. (Assume that the projection function for Lucy only shows the outputs she is supposed to see, and therefore has length equal to the number of commands issued by her.) Draw the corresponding state machine. Is this system secure? If yes, explain why. If not, show a counter example.

Answer:

Assuming:

S= {Holly, Lucy}

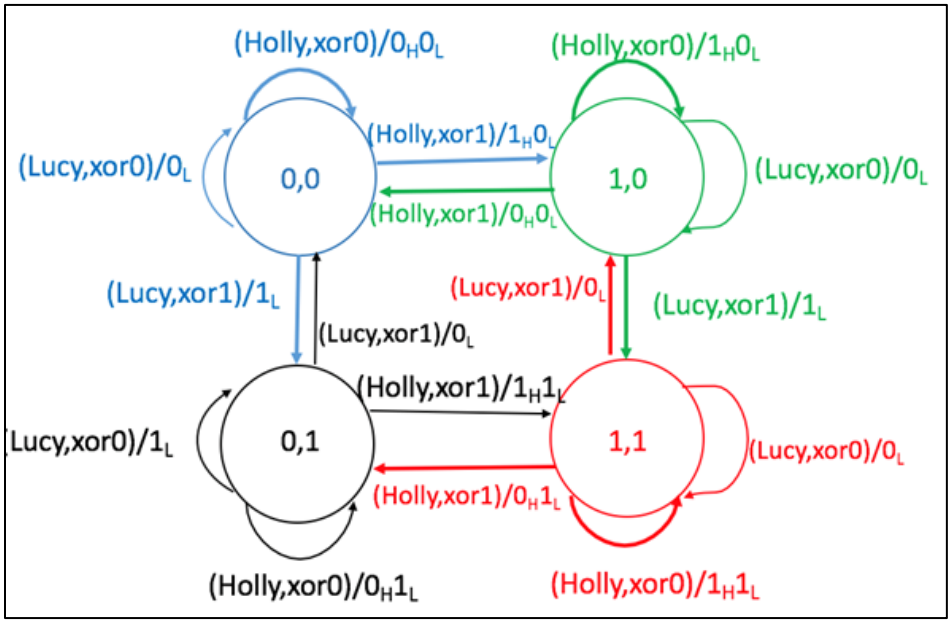
$\sum$= *{(0,0), (0,1), (1,0), (1,1)}*

C= {*xor0*, *xor1*}

The state transition function diagram is:

Commands	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)

The Finite State Machine is: (do note that Holly has the security clearance to read both H and L bits while Lucy can only ready L bits).



The above is one way to strengthen the security of this system which is to modify the commands so that Holly can alter only the high bits and Lucy only the low bits. Using the same 2-bit machine and consider the sequence c_S = (Holly, xor0), (Lucy, xor1), (Holly, xor1). Given an initial state of (0,0), the output is $0_H1_L1_H$ which correspond to Holly's and Lucy's command sequence (where the subscripts indicate the security level of the output).

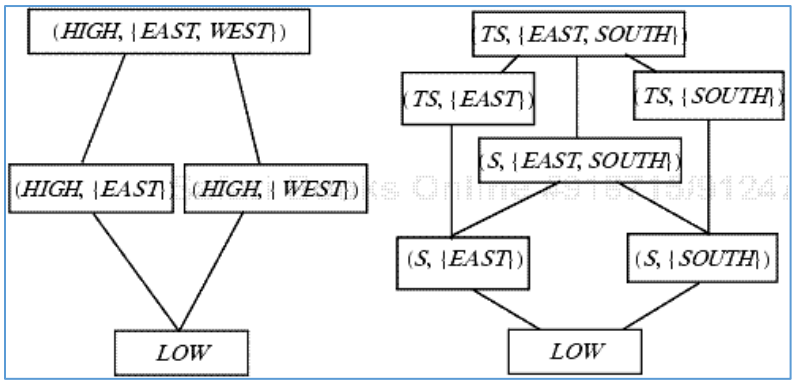
Applying Definition 8-4 (from Bishop's), we use G = {Holly}, G' = {Lucy}, and A = Ø. Then $\pi_{Holly}(c_S)$ = (Lucy, xor1), so proj(Lucy, $\pi_{Holly}(c_S)$, σ_0) = (1). Now we have proj(Lucy, c_S, σ_0) = proj(Lucy, $\pi_{Holly}(c_S)$, $\sigma 0$), and therefore { Holly } :| { Lucy } holds. In other words, subjects Holly and Lucy cannot interfere each other during the execution of the commands.

The system is therefore secured under this condition because the command made by Holly and her inputs and outputs are not accessible to Lucy. As a result, Lucy is unable to deduce any information about Holly.

Exercise 2

Draw the lattice described in the following example of Bishop's (Section 8.1.1).

EXAMPLE: Consider two systems with policies modeled by the Bell-LaPadula Model. One, system allie, has two security levels, LOW and HIGH, and two categories, EAST and WEST. The other, system son, has three security levels, LOW, S, and TS, and two categories, EAST and SOUTH. (The two EAST categories have the same meaning, as do the two LOW security levels.) Figure below shows the lattices of these two systems. The relevant issues are (1) how S and TS compare with HIGH and (2) how SOUTH compares with EAST and WEST.



Assume that HIGH corresponds to a level between S and TS, and that SOUTH is a category disjoint from EAST and WEST. Then the composed lattice has four security levels (LOW, S, HIGH, and TS) and three categories (EAST, WEST, and SOUTH).

2) The following do not exist: S should not have WEST, HIGH should not have SOUTH

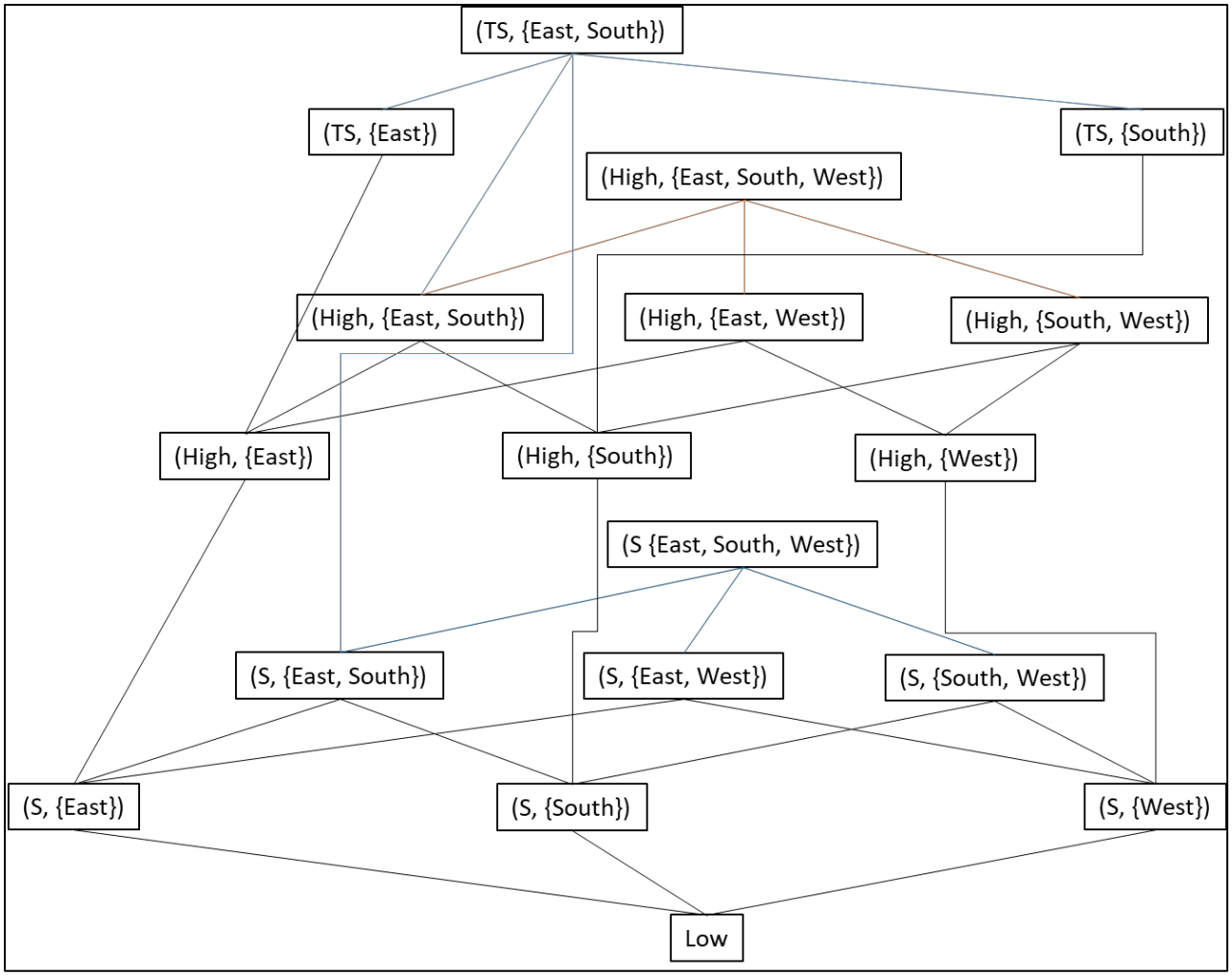
Answer:

We will apply Gong and Qian's Axiom 8-1 and 8-2 when developing the composition lattice which states that any access allowed or forbidden by the security policy of a component must be allowed or forbidden respectively by the composition of the components.

In addition, Gong and Qian[1] also developed an algorithm to determine the security of an access by computing the transitive closure of the allowed accesses under each component's policy and under the composition policy. Then all forbidden accesses in this set are deleted.

Applying the above, the resulting composition lattice is in the below figure.

[1] Extracted from Computer Security: Art and Science by Matt Bishop, Chapter 8.1 The Problem


(TS, {East, South})
(TS, {East})
(TS, {South})
(High, {East, South, West})
(High, {East, South})
(High, {East, West})
(High, {South, West})
(High, {East})
(High, {South})
(High, {West})
(S {East, South, West})
(S, {East, South})
(S, {East, West})
(S, {South, West})
(S, {East})
(S, {South})
(S, {West})
Low

3) It will not be detected during compile as the individual conditional statements are correct but do not show any direct relationship between x and y.

Exercise 3

Consider the following code snippet:

```
y := 0;
z := 0;
if x = 0 then z := 1;
if z = 0 then y := 1;
```

Assume x is a secret (high) variable, and y, z are public (low). Is there an information flow from x to y? If so why, and could have this been detected at compile or runtime?

Answer:

According to the code snippet, when $x = 0$ then $y = 0$ as $z = 1$, otherwise, $y = 1$ when $x \neq 0$. Therefore, there is an information flow from x to y as the value of y can be determined by the initial value of x at the end of sequence. This conforms to Definition 16-1 (Chapter 16.1.1. from Bishop's) which states that "information flows from variable x to y if the value of y after the commands allows one to deduce information about the value of x before the commands were run".

Compiler-based mechanisms check whether the information flows throughout the program are authorized and whether they violate a given information policy[2]. In the above case, the compiler mechanism detects the information flow because the flow from x to y is not secure. This is because $y \leq x$ (as x is classified as secret while y is low). Since the information flow from x to y is not secure, the compiler mechanism will raise flag and not certify the program. The compiler mechanism has an advantage over a run-time mechanism as it first verifies that a program is secure before it executes; it does not impact execution speed; and certification process has higher accuracy and easily understood[3]

The runtime or execution based mechanism is to prevent an information flow that violates flow policy. The mechanism checks the flow requirements whenever an explicit flow occurs. Therefore, if we apply the execution based mechanism to the above code snippet, it would check that the flow policy states $y \leq x$ because x is classified secret and y is classified low. It would inhibit the execution if explicitly x=0 causes information to flow to z and to y. Therefore, execution based mechanism would detect the flow in the above example and to prevent its execution.

---

[2] Computer Security: Art and Science by Matt Bishop, Chapter 16.3 Compiler-Based Mechanisms

[3] A Lattice Model of Secure Information Flow by Dorothy E. Denning, published in 1976 by ACM in volume 19 number 5

4) Answer is so confusing. Also Auth_long is supposed to be insecure as compared to auth as it aids bruteforce.

Exercise 4

Let auth_long be a modification of the password checker of Classwork Exercise 1 (Week 2), which, for efficiency reasons will now work as follows: if a username is found matching the input u, then the password p will be compared character by character with the stored password p'. If a character does not match, then the program will stop executing. For instance, if supplied password p is f0und4710n5 and the correct password p' is f0undat10n4l, then the password checker will stop executing after comparing the first 6 characters of p'.

Is there an information flow from d to the user's output? How is this different from the case of auth (of Week 2)?

Answer:

Firstly, we convert the auth_long modification to pseudo-code, assuming that the matched pair (u, p) should be tagged as (u_i, p_i), $1 \leq i \leq N$ where N represents the number of username-password pair in database and u_i is corresponding to p_i

```
if u_i in d

  then for j in length(p_i)

        if p_i[j] in d
              then output := 1
        else
              stop executing
else
stop executing
```

Is there an information flow from d to the user's output? How is this different from the case of auth (of Week 2)?

There is no information flow from d to output except when validation is successful and the user can access the account. When the username or password is incorrect, the program will stop executing and it will not return a value for output. Hence no information of d is transmitted to the output.

In contrast, the auth program in week 2 classwork would have revealed a 0 as output in the event of an incorrect username and password pairs. An attacker can create a list of usernames and passwords and use brute force method to find a right set of username and password.

The other difference is that the auth program from Week 2 checks the validity of username first, followed by password and finally the existence of the username and password pair in the database. Hence, if username is invalid, the program will output 0 immediately. If the username is correct and the password is wrong, the program will output 0 after a delay. Therefore, an attacker can realize whether the username or password is in the database from the time taken for the output to appear.

However, in auth_long program in Week 3, the program will stop executing immediately without any output if the validation fails. In addition, the password is validated one character at a time and the program will immediately stop when the following character fails the validation. The attacker will not be able to determine whether the username or password is in the database because there is very short or negligible time delay for the attacker to make a deduction. Auth_long (from week 3) is more secure and efficient than auth (from week 2).

However, given the availability of high performing computing resources in modern days, an attacker can still hack into an account by using brute force, dictionary and key logging attack methods. Therefore, organizations should deploy ways to improve the authentication program such as: to limit the login attempts and implement password reset policy. Impose blockage on accounts which reach the login limit and block IP addresses which exhibit unusual login attempts. Lastly, having users with access to very sensitive information use multi-factor authentication (in addition to password) is important to strengthen the security of an account