Information Assurance: Homework 6

Due October 26, 2009.

1. The following policy is enforced in a business:
   - Employees can access and update their own personal data. They can access their own salary information.
   - Managers can access personal and salary data about people that report to them
   - Managers can update salary information for people who report to them.

Consider a specific case with the following entities:

- Alice reports to Bob.
- Bob reports to Carol.

The grading on this question was fairly generous. You may have gotten full or most of the points but not answered the question fully. Review the grader comments and the comments here.

a. Define the rights involved and create an Access Control Matrix to encode the protection state for this scenario.

One way of expressing this. Right R = read or access. Right W = write or update. Right M = manage. Tracking the manage right isn't strictly necessary

	Alice PD	***Alice Salary***	***Bob PD***	***Bob Salary***	***Carol PD***	***Carol Salary***	***Alice***	***Bob***	***Carol***
Alice	*RW*	*R*							
Bob	*R*	*RW*	*RW*	*R*			*M*		
Carol			*R*	*RW*	*RW*	*R*		*M*	

b. Write the following command in the HRU model make_manager(s1, s2) – Make s1 a manager of s2.

As noted in the newsgroup, this command needs a couple additional arguments for the personal data and the salary objects.

make_manager(s1, s2, pd, sal)
enter M in A[s1, s2]
enter R in A[s1, pd]
enter R,W in A[s1, s]

c. Another rule is added to the policy. A manager can only change an employee's salary information if reviewed by their manager. Update the ACM to reflect the protection state with this new rule.

There was a lot of variance here. I was trying to give a broader question to think about how ACM's can be used in different ways. I'd do this by adding a review right, V. Our managers no longer have the default right to update their reports' salaries. Then our ACM from part a becomes:

	Alice PD	***Alice Salary***	***Bob PD***	***Bob Salary***	***Carol PD***	***Carol Salary***	***Alice***	***Bob***	***Carol***
Alice	*RW*	*R*							
Bob	*R*	*R*	*RW*	*R*			*M*		
Carol		*V*	*R*	*R*	*RW*	*R*		*M*	

We add a new one-time write right, T. After reviewing Bob's suggestion for the salary change, Carol can grant Bob a one time write right to create protection state shown in the second ACM

	Alice PD	***Alice Salary***	***Bob PD***	***Bob Salary***	***Carol PD***	***Carol Salary***	***Alice***	***Bob***	***Carol***
Alice	*RW*	*R*							
Bob	*R*	*RT*	*RW*	*R*			*M*		
Carol		*V*	*R*	*R*	*RW*	*R*		*M*	

This transition could be expressed as the following command

```
set_reviewed_salary_write(s1, s2, sal)
   if V in A[s1, sal] then
      enter T in [s2, sal]
```

d. Express the ACM as a set of access control lists.

You could have indicated one of the ACMs from part c. I was originally assuming the ACM from part a. The ACL's in that case would be

ACL(Alice PD) = (Alice, {RW}), (Bob, {R})
ACL(Alice Salary) = (Alice, {R}), (Bob, {RW})
ACL(Bob PD) = (Bob, {RW}), (Carol, {R})
ACL(Bob Salary) = (Bob, {R}), (Carol, {RW})
ACL(Carol PD) = (Carol, {RW})
ACL(Carol Salary) = (Carol, {R})
ACL(Alice) = (Bob, {M})
ACL(Bob) = (Carol, {M})
ACL(Carol) = empty

2. In this question you will work through evaluating labeled access following the Bell-LaPadula confidentiality model and Strict Biba integrity model. For the first two sections consider the following labeled entities:

Subject	Object	Label
Alice	Plan1	L1
Bob	Plan2	L2
Carol	Plan3	L3
Dave	Plan4	L4
Ellen	Plan9	L5

The labels follow a complete ordering L1 > L2 > L3 > L4 > L5.

I find the rules for execution in Biba fairly confusing. There are actually three labeled objects involved, the invoking process, the new process, and the binary file used to initialize the new process. I think in the Biba rule, S2 is the new process. So it is saying that one process can not spawn a higher integrity process. In this exercise it seems that we are more addressing files.

a. Interpret the labels as security labels in the simplified Bell-LaPadula model. Fill the the access column with the access that BLP would give each subject to the corresponding object: read, append (also mentioned in lecture as a pure write).

Subject	Object	Access?
Alice	Plan4	*Read*
Bob	Plan2	*Read, Append*
Ellen	Plan3	*Append*
Dave	Plan9	*Read*

b. Now interpret the labels as integrity labels in the strict Biba model. Fill the access column with the access that strict Biba would give each subject to the corresponding object: read, write, execute.

Subject	Object	Access?
Alice	Plan4	*Write, Execute*
Bob	Plan2	*Read, Write, Execute*
Ellen	Plan3	*Read*
Dave	Plan9	*Write, Execute*

c. Now consider the case where the labels have categories in addition to the completely ordered levels. We add categories alpha and omega. The new label assignments are:

Subject	Subject Label	Object	Object Label
Alice	L1:{alpha}	Plan1	L1:{alpha}
Bob	L2:{alpha,omega}	Plan2	L2:{omega}
Carol	L3:{omega}	Plan3	L3:{alpha, omega}
Dave	L4:{omega}	Plan4	L4:{alpha}
Ellen	L5:{alpha}	Plan9	L5:{omega}

Interpret these labels according to the Bell-LaPadula Model. Fill the the access column with the access that BLP would give each subject to the corresponding object: read, append (also mentioned in lecture as a pure write).

Subject	Object	Access?
Alice	Plan2	*No access*
Bob	Plan2	*Read*
Ellen	Plan4	*Append*
Dave	Plan9	*Read*

d. In class we only discussed the simple form of labels for Biba, but we mentioned the model could be extended to use the level and category labels as used in Bell-LaPadula. Now interpret the labels as integrity labels in the strict Biba model. Fill the access column with the access that strict Biba would give each subject to the corresponding object: read, write, execute.

Subject	Object	Access?
Alice	Plan2	*No access*
Bob	Plan2	*Write, execute*
Ellen	Plan4	*Read*
Dave	Plan9	*Write, execute*

3. Suppose a database for a department store contains an 'employee' table listing all employees' names, e-mail addresses, SSNs, salaries, hiring dates, and departments. The employee table rows for three employees is shown below

Name	Email	SSN	Salary	Hired	Department
Alice	alice@mart.com	xxx-xx-xxxx	$20.00	01/01/97	Appliance
Bob	bob@mart.com	yyy-yy-yyyy	$15.00	07/11/05	Shoes
Carol	carol@mart.com	zzz-zz-zzzz	$12.00	11/11/08	Hardware

   a. Suppose you are the database administrator. Your company has a policy that each employee can see the names, e-mails, and hiring dates of all other employees in the same department. Show the SQL statements for these three employees to enforce this policy.

Strictly speaking, you should express this as three separate statements, one for each employee, e.g.:

grant select(Name, Email, Hired) on employee where Department = 'Appliance' to 'Alice';

It is also acceptable to express as a parameterized user name and department name, e.g.,

grant select(Name, Email, Hired) on employee where Department = ***EmployeeDepartment*** *to* ***Employee****;*

You could also express this as a view and then a grant. The exact syntax is not important as long as the key parts are there.

   b. The company policy states that every employee should be able to view all fields about themselves in the 'employee' table. Show the SQL statements you would use to enforce this policy.

Again, could express the rules for each employee, or parameterize one set of rules. Could be expressed as a grant rule directly or as a view and grant combination, e.g.,

*create view alice_private as select * from employee where name = 'Alice';*
grant select on alice_private to Alice;

*create view bob_private as select * from employee where name = 'Bob';*
grant select on bob_private to Bob;

*create view carol_private as select * from employee where name = 'Carol';*
grant select on carol_private to Carol;

c. The company policy further states that an employee may choose to share this information with other employees of the company. How would you amend your answer in part b to enable an employee to allow other employees to view his or her non-public information in the 'employee' table?

Several ways one could do about this. The one I was looking for was amending the grant statements from part b to add the with grant option. e.g.,

grant select on alice_private to Alice with grant option;

This means that Alice can turn around and use another grant statement to pass on the select right to others.

You could also redesign the table, so there is a "allow_view_by_others" field.