CPS 633 – Computer Security, Lab 2

Due date: October 28, 2013 @ 9 AM via Blackboard

Background: The main purpose of this lab is for you to gain hands-on experience with access control schemes, and in particular *Discretionary Access Control (DAC)*, and *Role-Based Access Control (RBAC)*. You are to write programs to generate and update an authorization table, and to enforce, i. e. output possible access rights to a list of resource, when queried. You are also asked to comment on a number of questions on these access control schemes, and how they can be improved.

The implemented access control scheme is to manage users and resources in a typical classroom. For simplicity, we assume that the objects are simple documents files with read or write access privileges, and subjects are users, which are categorized as the *instructor*, *Teaching Assistants (TAs)*, and *students*.

The number of documents should be set to be variable, but has to at least include the following: a Course Management Form (CMF) file, an Announcement Page (AP) file, a Lab Information (LI) file, an indexed Lab Group Reports (LGRs) file , a Discussion Page (DP) file, a Lab Marks (LMs) file, and a Master Course Mark (MCM) file. You are to also assume that there is at least one TA, and that TAs are also students. There is also one instructor in this setting.

1 Discretionary Access Control (DAC)

The basic idea for this part of the lab is to implement a DAC access policy for the class setting described above. You are to write programs in C, based on the following requirements and objectives:

- A program that creates and can modify an access matrix for the documents listed above (and possibly more), and allows for an administrator to assign read and write to that document for the users of the system. For instance, it should allow for the MCM to be readable and writeable only to the instructor, the CMF to be readable and writeable to the instructor and readable to the rest of the users, the AP to be readable and writeable to all, the LI to be readable and writeable to the instructor and the TA(s), and readable to all other users, the LMs file to be readable and writeable to the instructor and the TA(s), and the indexed LGRs to be readable and writeable to the instructor, the TA(s), and a subset of students (a group of up to 4 students).

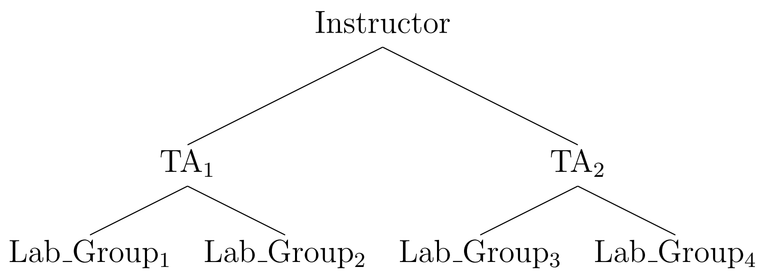
- A program that can interface with the created access matrix, and output an *Access Control List (ACL)* for a given document, or a *Capability Ticket* for a given user. You should also be able to set a *default* or *public* option to a given ACL. It has to also be able to look up and list the access right to a specific document for a given user. Here and throughout this lab, we assume that this particular user has already been authenticated by the system.

2 Role-Based Access Control (RBAC)

In this part of the lab, you will implement an RBAC access policy for the same class setting. Use the same set of users and permissions, i. e. read or write as above. Define and use three roles: **instructor**, **TA**, **student**, and **Lab_Group**$_i$, where i is the number of lab groups needed to accommodate the number of students (recall that each group can have at most 4 students). The goal of this part is to implement $RBAC_0$, $RBAC_1$, $RBAC_2$, and $RBAC_3$ for this class setting.

1. Write a program to implement $RBAC_0$ access policy. Students can be enrolled into TA or Lab_Group$_i$ roles. Modify your program from the last part to allow for access rights to the course documents to be assigned to roles. Combine this with the assignment of user to roles to output possible access rights to a given document for a specific user. Define two sessions: **Lab 1** and **Lab 2**, and allow for assignment of (possibly different) students to different lab groups. Use this to restrict access to lab reports for specific groups to only students who are on those groups, the TA(s), and the instructor.

2. Implement $RBAC_1$ extension of the above by adding a role hierarchy to the class:



For illustration, here we have assumed as an example that there are 2 TAs and 4 lab groups. Your role hierarchy should ensure that the inheritance is preserved. For example, the instructor should have all the privileges of any TA or any students, and any TA should have all the privileges of lab groups assigned to him or her.

3. An important concept of RBAC is that of *constraints.* Implement an $RBAC_2$ extension of your $RBAC_0$ that enforces the following **mutually exclusive** and **cardinality** constraints:

   - A user assigned to the role of the instructor cannot be also assigned to the roles of students, TAs, or lab group members. TAs cannot be assigned to roles of students or lab group members. Students cannot be assigned to be member of more than one group.
   - Only one user can be assigned the role of the instructor. At most 4 students can be assigned to a given lab group. The number of users assigned to the role of TAs cannot exceed the number of users assigned to the role of students.

4. Combine your implementations in the first 3 parts to enforce an $RBAC_3$ access policy.

3 Word Problems

- An important part of DAC is the ability of a system subject to enable another subject to have similar access privileges to the system resource. Discuss what you have to do to implement this kind of abilities in your implementation of DAC in part 1, and what would some of the typical security precautions or concerns that have to be taken into account?

- Suppose your system is to allow students or lab groups to 'read' their own (lab group) marks in the LM file. How would you enable this in your implementation?

- In a (large) system, it is possible to have conflicting constraints or hierarchy role assignment, when setting up an RBAC access centre. Given an example of such conflicts, and discuss how it can be handled.

- In an RBAC access policy implementation, it is possible to define a **public** role. Would you define such a role for our setting - discuss some of the possible advantages and disadvantages of such implementation?

4 Deliverables

You should submit a .zip file containing a report detailing your finding, observations, screenshots and answers to word questions in section 3. Text and document files must be in .txt, .doc, .docx or .pdf format.The .zip file should also contain all the programs and functions implemented in this lab, including scripts, source codes, any files created/required, together with a README file with instructions on how to build/use your implementations, if it's not self-evident. Your programs should be written in C, and be well commented. Once you have submitted your lab, the TAs will schedule a demo session for your group. All members of your group have to attend this demo session, answer questions about the lab and your implementation. Your (individual) grade for this lab will be partially based on your performance in this demo. Failure to be present for the demo or lack of knowledge about what is handed in will result in a grade of zero.