Assignment 1
CS-333
Fall 2013

Due Date: 9/28/2012 at 11:59 pm (No extensions)

Goals:

1. To become better acquainted with Role Based Access Control (RBAC).
2. To understand the benefits and drawbacks of using RBAC.
3. To understand the challenges involved in implementing RBAC enforcement mechanisms.
4. To develop appreciation for the RBAC model and its numerous applications.

Overview

In this assignment you will develop a mechanism for enforcing RBAC policies. When launched, your program shall read and parse the following two files:

1. **user-role assignment (URA) file:** stores user-role assignments.
2. **permission-role assignment (PRA) file:** stores permission-role assignments.

The program then displays login prompt:

```
login:
```

Next, the user enters his/her username and presses <enter>. The program then checks if the user is present in the URA file (based on the parsed information). If not so, then the program reports an error and re-displays the prompt:

```
login:  mikhail
ERROR: user mikhail is not in the database!

login:
```

Otherwise, the program logs the user in, and displays the command prompt:

```
login:  mikhail
Welcome mikhail!
cmd>
```

From here, the user may issue commands of the form `<action> <object>`. For example:

```
cmd> read GradeBook
```

After the user presses `<ENTER>`, the program then determines to what roles the user e.g. `mikhail` is assigned to (i.e. based on the parsed information from the URA file), and checks whether any of these roles have the permission in question e.g. `read GradeBook` (i.e. based on the PRA file). If not so, then the program displays an error:

```
cmd> view GradeBook
Access denied:  you are not authorized to perform this action!
cmd>
```

Otherwise, the program prints `Access granted by virtue of roles:` and the roles assigned to the logged in user that authorize the action:

```
cmd> view CourseSyllabus
Access granted by virtue of roles:  Professor Lecturer
```

In the above example user `mikhail` is assigned to roles Professor and Lecturer which have permission `view CourseSyllabus`.

Technical Details

URA file shall be named `URA.txt` and shall have the following format:

```
<userid1> <role1>
<userid1> <role2>
<userid2> <role1>
.
.
.
```

For example:

```
mikhail Professor
```

```
mikhail Lecturer
bozo Clown
.
.
.
```

PRA file shall be named `PRA.txt` and have the following format:

```
<role1> <action1> <object1>
<role1> <action2> <object1>
<role2> <action3> <object3>
.
.
.
```

For example:

```
Student view GradeBook
clown throw Pies
Student register Course
Professor modify Gradebook
Professor view CourseSyllabus
Lecturer view CourseSyllabus
.
.
.
```

Both files shall be read automatically by the program when started.

Coding Tips

If you are using C++, you may want to use the `multimap` data structure supplied by C++ Standard Template Library (STL), in order to implement associations between users and roles, and roles and permissions. Please see the sample file `multimap.cpp` on Titanium.

If you are using Java, you may want to use the `Map` class provided by the Java libraries. If you are using Python, you may want to use a Python dictionary. Please see the sample file `multimap.java` on Titanium.

BONUS

Extend your RBAC enforcement mechanism to incorporate a role hierarchy. Recall, in RBAC,

if role R_1 is senior to role R_2, then role R_1 inherits all the permissions of role R_2. For example, if role *Dean* is senior to role *DeptChair* which in turn is senior to role $Professor$, then role *Dean* inherits all permissions of roles $Professor$ and *DeptChair*.

Required changes:

In addition to URA and PRA files, your program shall also read the role hierarchy file, which shall be named `HR.txt`. The file shall have the following format:

```
<senior role1> <junior role1>
<senior role1> <junior role2>
<senior role2> <junior role3>
<senior role4> <junior role5>
.
.
.
```

For example:

```
Undergrad Student
Grad Student
DeptChair Professor
.
.
.
```

Hence, given URA file:

```
Joe Undergrad
John Grad
```

and PRA file:

```
Student view Transcripts
Undergrad register UndergradCourse
Grad register GradCourse
```

the following query will work as follows (assuming Joe is currently logged in):

```
cmd> view Transcripts
Access granted by virtue of roles:  Undergrad
```

This is because Joe is assigned to role `Undergrad` which has permission `register UndergradCourse` and also inherits permission `view Transcripts` from role `Student`.

SUBMISSION GUIDELINES:

- This assignment may be completed using C, C++, Java, or Python.

- You may work in groups of 2.
- Please hand in your source code. electronically (do not submit .o or executable code) through **TITANIUM**. You must make sure that this code compiles and runs correctly.
- Write a README file (text file, do not submit a .doc file) which contains
  - Your name and email address.
  - The programming language you use (e.g. C/C++/Java)
  - How to execute your program.
  - Whether you implemented the extra credit.
  - Anything special about your submission that we should take note of.
- Place all your files under one directory with a unique name (such as `p1-[userid]` for assignment 1, e.g. `p1-mgofman1`).
- Tar the contents of this directory using the following command. `tar cvf [directory_name].tar [directory_name]` E.g. `tar -cvf p1-mgofman1.tar p1-mgofman1/`
- Use TITANIUM to upload the tared file you created above.

Grading guideline:

- Program compiles: 10'
- Correct output: 80'
- README file: 5'
- Correct format of URA and PRA files: 5'
- Bonus 15'
- Late submissions shall be penalized 10%. No assignments shall be accepted after 24 hours.

Academic Honesty:

Academic Honesty: All forms of cheating shall be treated with utmost seriousness. You may discuss the problems with other students, however, you must write your **OWN codes and solutions**. Discussing solutions to the problem is **NOT** acceptable (unless specified otherwise). Copying an assignment from another student or allowing another student to copy your work **may lead to an automatic F for this course**. Moss shall be used to detect plagiarism in programming assignments. If you have any questions about whether an act of collaboration may be treated as academic dishonesty, please consult the instructor before you collaborate. Details posted at http://www.fullerton.edu/senate/documents/PDF/300/UPS300-021.pdf.