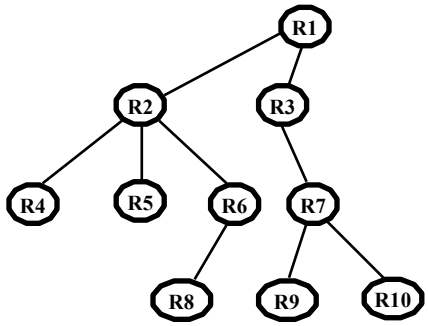
Homework 7, Due Date*: 12:00pm (noon) 11/08/2019, Cutoff Date: 12:00pm (noon) 11/10/2019**
****Submission will NOT be accepted after the cutoff deadline**
Submission: (1) upload your .java file(s) and .txt input files on Blackboard (NO email submission please!), (2) see Task II.

Part I. Review Questions for reading the textbook (No submission; however, the relevant contents will be included in Exams as True/False, Multiple-Choices, and/or Filling-in-the-Blanks questions for up to 20% of the total points.)

- **Textbook, Page 134**
  4.8 Briefly define the four RBAC models of Figure 4.9a.
  4.9 List and define the four types of entities in a base model RBAC system.
  4.10 Describe three types of role hierarchy constraints.
  4.11 In the NIST RBAC model, what is the difference between SSD and DSD?
- **Textbook, Page 353**
  10.1 Define buffer overflow.
  10.2 List the three distinct types of locations in a processes address space that buffer over-flow attacks typically target.
  10.3 What are the possible consequences of a buffer overflow occurring?
  10.4 What are the two key elements that must be identified in order to implement a buffer overflow?
  10.5 What types of programming languages are vulnerable to buffer overflows?
  10.6 Describe how a stack buffer overflow attack is implemented. 10.7 Define shellcode .
  10.8 What restrictions are often found in shellcode, and how can they be avoided?
  10.9 Describe what a NOP sled is and how it is used in a buffer overflow attack.
  10.10 List some of the different operations an attacker may design shellcode to perform.
  10.11 What are the two broad categories of defenses against buffer overflows?
  10.12 List and briefly describe some of the defenses against buffer overflows that can be used when compiling new programs.
  10.13 List and briefly describe some of the defenses against buffer overflows that can be implemented when running existing, vulnerable programs.

Part II: Task I (90%). Write a Java program to implement a simplified NIST RBAC model. (The topology and examples here are provided for you to test your program. The instructor will use similar but different test cases to test your program. Your program will be evaluated according to how much it can accomplish and the test cases that it can pass INSTEAD OF code review.)

1. Data structures: it is up to you to choose **appropriate data structures** to maintain *role hierarchy*, *user-role matrix*, *role-object matrix*, etc.



2. Role Hierarchy: this program must support role hierarchy defined in NIST RBAC.

2.1 This program reads a role hierarchy from file "*roleHierarchy.txt*". Each line in this file is formatted as "<ascendant> <descendant>". For example, for the role hierarchy on the right, the first several lines in roleHiearchy.txt is

```
R8          R6
R9          R7
R10         R7
.....
```

2.2 The contents of this text file must be **validated** to enforce the "limited role hierarchies" defined in NIST RBAC (See the slide titled "**Hierarchical RBAC**"). For an **invalid** file, (1) **close** the file, (2) **display** "invalid line is found in roleHierarchy.txt: line <line # of the first invalid line>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to **2.2**. **Continue** if it's **valid**.

2.3 Display the role hierarchy **top down**. If you may display it as a tree, it is great. Otherwise, the following format is fine.

```
R1--->R2, R3
R2--->R4, R5, R6
R3--->R7
.....
```

3. Objects: This program reads all ***resource objects*** from file "*resourceObjects.txt*", which contains ONE line. For example,

```
F1    F2    F3    F4    P1    P2    P3    D1    D2
```

3.1 Validate objects for duplication. For an **invalid** file, (1) **close** the file, (2) **display** "duplicate object is found: < object>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to **3.1**. **Continue** if it's **valid**.

3.2 Display the current *role-object matrix* with null entries. If necessary, you may break this matrix into multiple sub-matrices for displaying by limiting up to 5 columns per sub-matrix. For example,

```
          R1    R2    R3    R4    R5
R1
.....
          R6    R7    R8    R9    R10
R1
.....
... ... ... ... ... ... ...
```

4. Granting Permissions to Roles with Inheritance:

4.1 This program must support role inheritance described as "the upper role includes all of the access rights of the lower role, as well as other access rights not available to the lower role" in the slide titled "**Role Hierarchy**". In **4.3**, **4.4**, and **4.5**, whenever a permission is granted to a role, it must be **inherited** by **all** the **descendants** of this role.

4.2 Redundancy must be avoided. For example, when adding "read" by R2 to F1, if "read" is already there, don't add it again.

4.3 Update the *role-object matrix* to grant "control" by EACH role to itself. (See **4.1**). E.g., R8, R6, R2, and R1 "control" R8.

4.4 Update the *role-object matrix* to grant "own" by the descendent of EACH role to it. (See **4.1**). E.g., R6, R2, and R1 "own" R8.

4.5 This program also reads permissions to roles from file "*permissionsToRoles.txt*". Each line in this file is formatted as "<role> <access right> <object>", where a permission is an access right to an object. For example, after reading the following two lines, R6, R2, and R1 have "write*" to F3, and R2 and R1 have "seek" to D1.

```
R6      write*          F3
R2      seek            D1
```

4.6 Display the current *role-object matrix* in the same format as what is used in **step 3.2**.

5. SSD: This program reads ***mutually exclusive*** (when *n* = 2) and ***cardinality constraints*** (when n>= 3) from file "*roleSetsSSD.txt*".

5.1 Each line includes a constraint given by the value of *n* and a set of roles. For example,

```
3       R2      R4      R5      R6      R8
2       R1      R2      R4
.....
```

Note: ***n*** MUST be >= 2 to be valid. For an **invalid** case, (1) **close** the file, (2) **display** "invalid line is found in roleSetsSSD.txt: line <line # of the first invalid line>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to Step **5**. **Continue** if it's **valid.**

5.2 Display all the constraints, one in a line. For example,

```
Constraint 1, n = 3, set of roles = {R2, R4, R5, R6, R8}
…
```

6. This program constructs the *user-role matrix* by reading file *usersRoles.txt*. Each line is formatted as "<user> <list of roles>".

6.1 While reading each line and updating the *user-role matrix*, the **SSD constraints** configured in Step **5 must be enforced**. It is **also invalid** if two or more lines contain the same user. For an **invalid** file, (1) **close** the file, (2) **display** "invalid line is found in usersRoles.txt: line <line # of the first invalid line> due to <constraint #? or duplicated user>, enter any key to read it again" to allow user to fix the problem, (3) once reading any user input, go back to Step **6**. **Continue** if it's **valid.**

```
U1      R2      R5
.....
```

6.2 Display the *user-role matrix*. For example,

```
        R1      R2      R3      R4      R5      R6      R7      R8      R9      R10
U1              +                       +
.....
```

7. Query for a given User, a given pair of User and Object, or a given combination of (User, Access Right, Object)

7.1 Display the following messages to get three user inputs: **`user`**, **`object`**, and **`accessRight`**.

```
Please enter the user in your query:
Please enter the object in your query (hit enter if it's for any):
Please enter the access right in your query (hit enter if it's for any):
```

7.2 if **`user`** isn't in the *user-role matrix*, display "invalid user, try again." and go back to Step 7.1. Otherwise, **continue**.

7.3 if **`object`** and **`accessRight`** are empty, **display** all the objects that this user has access rights to as "<object> <list of access rights to this object>", and then **go to** Step 7.7. Otherwise, **continue**. To keep it simple, if a user has the same or different lists of access rights to the same object due to different roles assumed by this user, you don't have to merge them. For example,

```
R4      control, own
F1      read, write, execute
R4      own
…
```

7.4 if **`object`** isn't in the *user-role matrix*, display "invalid object, try again." and go back to Step 7.1. Otherwise, continue.

7.5 if **`accessRight`** is empty, **display** all the access rights this user has to this object and **go to** Step 7.7. Otherwise, continue.

7.6 check whether this **`user`** has the given **`accessRight`** to this **`object`**. If yes, display "authorized", otherwise, display "rejected". Continue.

7.7 Display "Would you like to continue for the next query?" If the user inputs "yes", go back to Step 7.1. Otherwise, exit.

Task II (10%): **Test your program on cs3750a.**

1. MAKE a directory "**HW07**" under your home directory on **cs3750a.msdenver.edu**.
2. UPLOAD, COMPILE, and TEST your program under "**HW07**" on **cs3750a.msdenver.edu**.
3. SAVE ***ALL your .txt input files*** and a ***file named testResults.txt*** under "**HW07**" on cs3700a.msudenver.edu, which captures the outputs of your program when you test it. You can use the following commands to redirect the standard output (stdout) to a **file** on UNIX, Linux, or Mac, and view the contents of the file

```
java prog_name_args | tee testResults.txt //copy stdout to the .txt file
cat file-name     //display the file's contents.
```