Project 2

Chase Lane cdl52

Task W0

For my access control language I modeled it around a campus security policy. There are three files for every policy: users, groups, and delegation. The basic syntax is that users have permission(s) for a location(s), users belong to a group that has permission(s) for location(s), groups inherit permissions, and groups can administrate other groups. SD3 and RT model the system but the actual syntax/format of the files is completely unique. From this structure, indirection, administrative delegation, and role inheritance is made.

The program, as described later on, can be ran using scripts with set flags to automatically parse input files. The program also supports loading individual files one at a time and resetting the current policy from the command line without restarting/recompiling the program. A list of available commands can be viewed at any time in the terminal by typing “l”. The program supports advanced queries such as what permissions does user have, can user access this location, who has permission for this location, etc.. For demonstrating the administrative delegation, you have the ability to change into a mock role for the session and add/remove permissions to groups. These changes reflect across the system. Complex, randomly generated policies can also be made from the terminal for testing the system.

Task W1

Policy files are created as CSV so they can be easily modified in Excel or spreadsheet programs. Attributes can be multi-assigned using Excel’s features too for easier use. Three files are used: users.csv, groups.csv, and delegation.csv. For users and groups, the length of a given line determines what is read in. The syntax is as follows:

- **Groups**
  - For group permissions:
    - # Group, permission, location
      - Ex: Student,Total,Market
  - For role inheritance:
    - # Group, group it inherits
      - Ex: Student,Public
- **Users**
  - For user specific permissions:
    - # User, Permission, Location
      - Ex: Joe,Partial,Cathedral of Learning
  - For assigning group, indirection
    - # User, Group
      - Ex: Joe, Student
- **Delegation**
  - For giving group administrative power over another group
    - # Group, Administrates who
      - Ex: CS Admin,CS Faculty

By incorporating attributes of SD3 & RT and the separation of files, complex policies can be easily written including assigning many permissions at once, inheriting multiple groups of permissions at once, and administrating over many groups as one entity. The users file simulates the RT “←” attribute by assigning groups/users attributes all at once. The delegation file simulates the SD3 “$” by assigning groups the ability to oversee/administrate other groups. In this system, an administrator would easily be able to write a complex chain of command such as Teacher inherits from Student who inherits from Guest – Teacher,Student /n Student,Guest. Ordering of the files does not matter since the entire policy is read in at the same time.

Task C2

To run the test policy, use the test script or use the command “java driver -1 ./policies/test/groups.csv ./policies/test/users.csv ./policies/test/delegation.csv” The flag – 1 signifies that the policy should be loaded immediately. Alternatively, policy files can be individually loading by using the command (lowercase I) and selecting a file from there. The commands are:

- **c** - (change) change role into an administrative entity
- **d** - (debug) list current sizes of users/groups/locations policies
- **g** - (generate) generate random policies of varying size for testing
- **i** - (input) read in policy from input file
- **l** - (list) view available commands
- **q** - (query) search regarding the policy
- **r** - (reset) clear current policies
- **s** - (shutdown) exit program

For **queries**, type **q** and these options are shown:

- *Surround each search term in double quotation marks*
- **1** - What users are part of group ___?
- **2** - What permissions does user ___ have?
- **3** - What permissions does group ___ have?
- **4** - Can user ___ access ___?
- **5** - Who has ___ permission for location ___? (List users with specific permission (Restricted,Partial,Total) for location)
- **# fillIn fillIn(4&5 only)**

When executing a query, make sure to surround all inputs in double quotation marks. Some example queries on the test policy:

- What users are part of group student?
  - **1 “Student”**
    - In the users file, only John and Joey are assigned to group Student

```
----------------------------------------
Listing users part of Student
John
Joey
----------------------------------------
```

- What permissions does user Ricky have?
  - **2 “Ricky”**
    - Ricky is assigned group Guest, which inherits group Student, which also inherits group Public. This chain of permissions: Guest ← Student ← Public. Therefore, Ricky has his individual permissions + Guest, Student, and Public permissions

```
-----------------------------------
Permissions From Group: Guest
No Group Permissions

Inherited From Group: Student
[permission=Restricted | location=Hillman Library]
[permission=Specific | location=Cathedral of Learning]
[permission=Total | location=Market]

Inherited From Group: Public
[permission=Restricted | location=Cathedral of Learning]
[permission=Restricted | location=WPU]

Ricky has permissions:
-----------------------------------
```

- What permissions does group IT have?
  - **3 “IT”**
    - IT inherits group Guest. The chain of permissions:
      IT ← Guest ← Student ← Public
      Also administrates group Student

```
-----------------------------------
Permissions From Group: IT
[permission=Total | location=Hillman Library]
[permission=Total | location=Cathedral of Learning]

Inherited From Group: Guest
No Group Permissions

Inherited From Group: Student
[permission=Restricted | location=Hillman Library]
[permission=Specific | location=Cathedral of Learning]
[permission=Total | location=Market]

Inherited From Group: Public
[permission=Restricted | location=Cathedral of Learning]
[permission=Restricted | location=WPU]

Administrator Of Group: Student
-----------------------------------
```

- Can user Max access Cathedral of Learning?
  - **4 “Max” “Cathedral of Learning”**
    - Max is not part of a group, but he does have an individual permission with restricted access

```
-------------------------------------
Max has permission:
[permission=Restricted | location=Cathedral of Learning]
-------------------------------------
```

- Can user Max access WPU?
  - **4 “Max” “WPU”**
    - Max does not have this permission

```
----------------------------------------
No, they do not have permission for WPU
----------------------------------------
```

- Who has Restricted permission for Hillman Library?
  - **5 “Restricted” “Hillman Library”**
    - Group student has this permission Max has this individual permission
      Groups guest, IT, and Admin inherit Student and also have this permission

```
-----------------------------------
Max has individual permission:
[permission=Restricted | location=Hillman Library]
John is part of group Student which has permission
Ricky inherits group Student which has permission:
[permission=Restricted | location=Hillman Library]
Adam inherits group Student which has permission:
[permission=Restricted | location=Hillman Library]
Systems Administrator inherits group Student which has permission:
[permission=Restricted | location=Hillman Library]
Steve inherits group Student which has permission:
[permission=Restricted | location=Hillman Library]
Joey is part of group Student which has permission
-----------------------------------
```

- Who has Total permission for Cathedral of Learning?
  - **5 “Total” “Cathedral of Learning”**
    - Group IT and CS Admin have this permissions
      Admin inherits both IT and CS Admin

```
-----------------------------------
Adam is part of group IT which has permission
Systems Administrator inherits group IT which has permission:
[permission=Total | location=Cathedral of Learning]
Systems Administrator inherits group CS Admin which has permission:
[permission=Total | location=Cathedral of Learning]
Prof. Teach has individual permission:
[permission=Total | location=Cathedral of Learning]
Steve is part of group IT which has permission
Top Dog is part of group CS Admin which has permission
-----------------------------------
```

Task W3

Indirection is represented through the ability to assign multiple permissions to a group then assign that group to multiple individuals. You do this by first assigning group permissions in the groups file then assign that group to users in the users file.

- Examples (also show in queries 1-3 above):
  - In **groups file** there is group IT which has permissions:
    - IT,Total,Hillman Library
    - IT,Total,Cathedral of Learning
    - This example only have two permissions from this group, but you can assign as many as you want to a single group
  - In **users file** there are users who are assigned the role IT:
    - Adam,IT
    - Steve,IT
    - This means that now Adam and Steve both have permissions for Hillman Library and Cathedral of Learning without explicitly stating it

One of the major benefits of implementing an access control policy like this is the easy of use for an administrator. Instead of having to write x amount of permissions for x users, you can write it once for a role and assign that role to the users. This approach also makes it easy to see where the user is getting a permission from. If a user inherits multiple groups, we can see which group is giving them a specific permission. As an administrator, their life is made easy with role policies and decreases the amount of work needed when implementing new permissions. Just as permissions can be easily added, so can permissions be removed. Mainly the benefit of added indirection is to make the policy less convoluted and easier for the admin to use. It also allows policies to be made that can specifically distinguish groups of users such as CS Students and Econ Students will share a majority of permissions but can be identified based on their role.

A major drawback could be potentially assigning the wrong user a very high clearance role. A scenario where this might happen is a new employee gets hired and needs administrative powers but the admin gives the powers accidentally to a user with malicious intent. Therefore this malicious user can use their role powers to modify the policy or access areas their not supposed to.

Task W4

Administrative delegation is represented by the ability to assign groups of users the power to change the access policy of another group. You do this by modifying the delegation file and assigning these powers, examples:

- **c** Change into a role
  - **“IT”**
    - IT is Admin of group Student
      - Adding permission Restricted to Students to Test Location
      - **a “Student” “Restricted” “Test Location”**
      - Removing permission Restricted to Hillman Library from Students
      - **r “Student” “Restricted” “Hillman Library”**
    - Seeing changes in policy
      - **q**
        - List student permissions
        - **3 “Student”**
        - We can see the changes were successful and that they no longer have permission for Hillman Library and now have permission for Test Location

```
-----------------------------------
Permissions From Group: Student
[permission=Specific | location=Cathedral of Learning]
[permission=Total | location=Market]
[permission=Restricted | location=Test Location]
```

- **c** Change into a role
  - **“Admin”**
    - Admin is Admin of Groups IT and CS Admin
      - Adding permission Total to CS Admin to xyz123
      - **a “CS Admin” “Total” “xyz123”**
    - Seeing changes in policy
      - **q**
        - Who has permission Total to xyz123?
        - **5 “Total” “xyz123”**
        - We can see that CS Admin now has this permission and that also groups that inherit CS Admin get this permission too

```
-----------------------------------
Systems Administrator inherits group CS Admin which has permission:
[permission=Total | location=xyz123]
Top Dog is part of group CS Admin which has permission
-----------------------------------
```

A major benefit of implementing administrative delegation is the ability to give certain users certain administrative powers without giving them total access to the system. For example, a teacher only needs admin powers over students where an admin would have powers over everything and is a security risk if a teacher were to get those powers. Again this simplifies a system administrators job too because they can split the work they would have to do to modify the policy to different users. Consider a policy with 5 different main offices and each office has a varying policy. If office #2 wants a new group with new permissions, the sys admin can create a role for office 2 that can modify their subset of users. Basically give some users some access so they can’t tamper and mess with the overall system.

A drawback may include creating too much administrative overhead where the chain of command of who actually administrates who becomes convoluted. Another issue is still giving users too much power in the ability to affect the entire group policy they oversee. A potential solution to this is perhaps make it so they can only affect a subset of a specific policy or maybe only access specific locations. The security threat of a malicious insider is way too high to implement a bare bones interface like this in an actual access control policy. Lots of details need to be worked on to make sure no one user obtains too much power. Even the sys admin would need to be part of a group that only has certain permissions because it could be sensitive data stored in the policy. Adding additional administrator powers would make a search for insecure policies much more difficult. Again on this simple example its easy to see that the Admin oversees IT who oversees

Students but imagine on a policy with 10000 users and 2500 groups. Now the chain of command for that many groups would be way to complicated to write out. If an automated analysis were ran, it would also take a long time to work through the entire policy despite being stored in HashMaps with minimal oversight.

Task W5

Role inheritance is represented by the ability to have groups inherit the permissions of other groups. The groups inherit can be thought of recursively. In the test example, Guest ← Student and Student <- Public. While not explicitly stated in the file that Guest inherits Public, through the chain of permissions, Guest also inherits Public. Groups can also inherit multiple groups at once and this chain of permissions can grow fast. Examples:

- In **groups file**
  - To demonstrate how the chain of permissions Guest ← Student ← Public is created
    - Guest, Student (Line 15)
      - Guest inherits Student
    - Student,Public (Line 16)
      - Student inherits Public
    - Therefore, Guest inherits Public
  - Another example of a chain of permissions IT ← Guest ← Student ← Public is created
    - IT, Guest (Line 17)
    - Following the same logic above, IT inherits Student who inherits Public
- Demonstration
  - **q**
    - What permissions does group IT have?
    - **3 "IT"**
      - We can see the permissions that group IT has and then all the different permissions it inherits
  - When running queries like 4&5, it also shows where a user/group gets that permission from, which is useful as an administrator

```
-----------------------------------
Permissions From Group: IT
[permission=Total | location=Hillman Library]
[permission=Total | location=Cathedral of Learning]

Inherited From Group: Guest
No Group Permissions

Inherited From Group: Student
[permission=Specific | location=Cathedral of Learning]
[permission=Total | location=Market]
[permission=Restricted | location=Test Location]

Inherited From Group: Public
[permission=Restricted | location=Cathedral of Learning]
[permission=Restricted | location=WPU]

Administrator Of Group: Student
-----------------------------------
```

A major benefit of including role inheritance is to remove the redundancy when writing policies. Consider scenario where there are two groups of students: CS Students, Econ Students. Both groups have permissions to go to the food court and study hall but CS Students can access CS buildings and Econ students can access Econ buildings. A general group "Student" can be defined with permissions for food court and study hall which can then be inherited by both CS and Econ students. So instead of having to write the permissions twice, you can achieve the same goal with only one time. This problem grows exponentially when the chain of inherited permissions grows larger than 2, so as an administrator, their life becomes easy again in creating permissions. It definitely allows the creation of policies that would see impossible to create without this due to the complexity of some. In the example IT ← Guest ← Student ← Public, when a query is performed to find who has access to x, which belongs to Public, the query can be simply written recursively to find all the users in the chain of permissions easily. So writing the code query-wise, this definitely made it simpler.

A drawback is the potential confusion when trying to describe this policy and how to use it. It is not clear right away how the feature is intended to work, and it needs extra explaining how groups can inherit more than one and chain of permissions etc.. If I could've included a visual representation, I definitely would've. Another issue is confusion when writing a policy because you could accidentally write infinite policies like: x ← y ← z ← x. There would need to be a check in place to avoid this. Another issue is security threats because you could accidentally have a user-based role inherit an administrator-type role because the chain of command might be confusing.

Task C6

The terminal allows randomly generated policies to be created. The tricky part briefly touched on above was the ability to avoid infinity inheriting policies. To circumvent this, a Hashmap of already inherited policies for a give group was consulted when adding new groups to make sure the direction of inheritance only traveled one way.

- **g**
  - Generating policy with 10,000 users, 2,000 groups, 5,000 locations, and 50 complexity
    - Complexity refers to how often a group inherits from another group and users having individual permissions
      - A complexity of 100 would mean every time (except the off-chance it selects itself) a group inherits another group and a user has extra individual permissions
      - 0 would mean this never happens
    - **10000 2000 5000 50**
      - Files are outputted in ./policies/output
- **To run the tests**
  - run script stressTest.sh
  - or set the flag to -2
    - java driver -2
- **Overview of tests**
  - The complexity parameter is set to a constant 50% to ensure the benchmark is consistent
    - **g1** – 15 users, 5 groups, 10 locations, 50
    - **g2** – 100 users, 25 groups, 100 locations, 50
    - **g3** – 500 users, 100 groups, 1000 locations, 50
    - **g4** – 1000 users, 200 groups, 5000 locations, 50
    - **g5** – 10000 users, 1000 groups, 5000 locations, 50
    - **g6** – 10000 users, 1000 groups, 5000 locations, 90

Task W7

	Insert (s)	query
g1	.005639743	
g2	.010813702	
g3	.021923317	
g4	.016269245	
g5	.081349150	
g6	.072967193	

The result is not that surprising considering the implementation utilizes hashmaps which are insanely fast. What is interesting is how g5 and g6 which are similar policies besides complexity fluctuate a lot in time. There wasn't a consistent result in which ran faster although an average is listed above. The language would be reasonable to run on personal computers given the policy size isn't insanely large like a bank account system would use. If this policy were used for local files and that value is less than 50,000, it could have potential to be used. At large amounts of users, there is a problem with verifying the inheritance policy and stack overflow errors occur.