Levi Pfantz

CS 750 Project Report

12/15/2020

Here is the access control matrix for my program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Users / Folders | System | Shared | Engineer\_group | Accounting\_group |
| Admin | RW | RW | RW | RW |
| Bob |  | RW | R |  |
| Johnny |  | RW | RW | R |
| Charles |  | RW |  | RW |

As you can see there is a pretty large data leak with Admin having complete access to high privilege (system) and low privilege (shared) resources. I went this way because it is representative of many systems in the real world.

It’s implemented with a capabilities list like so:

Admin: { (System, RW), (Shared RW), (Engineer\_group RW), (Accounting\_group RW) }

Bob: { (Shared RW), (Engineer\_group R) }

Johnny: { (Shared RW), (Engineer\_group RW), (Accounting\_group R) }

Charles: { (Shared RW), (Accounting\_group RW) }

In reality I use something inspired by Linux and 3 (11) is RW, 2 (10) is R, 1(01) is W. So my capabilities list is more like this:

Admin: { (System,3), (Shared 3), (Engineer\_group 3), (Accounting\_group 3) }

Bob: { (Shared 3), (Engineer\_group 2) }

Johnny: { (Shared 3), (Engineer\_group 3), (Accounting\_group 2) }

Charles: { (Shared3), (Accounting\_group 3) }

Also, here are their passwords:

Admin: root

Bob: veggieburger

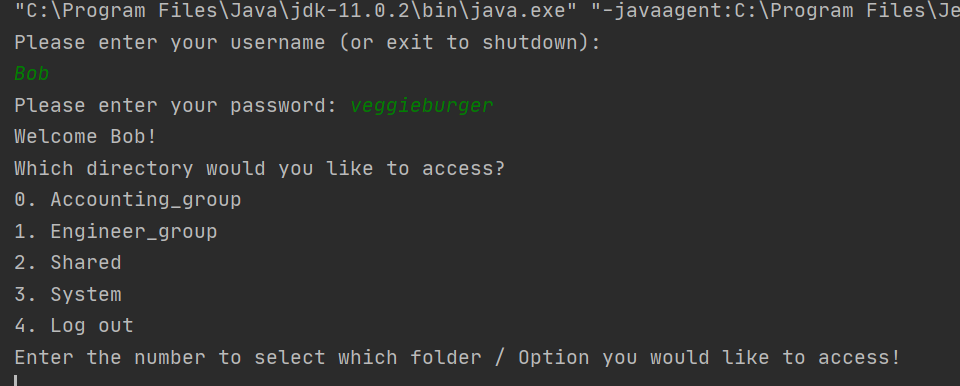
Johnny: weakpassword

Charles: 1234

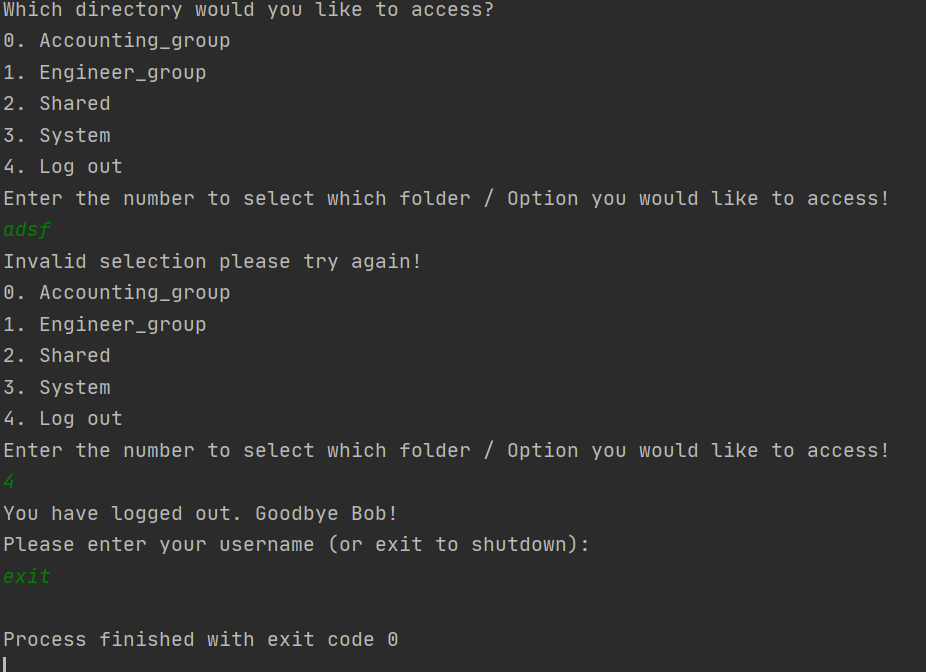
My code has four parts. First of all is a capabilities object class. This class simply stores a folder name string and a permission int. Second of all is a User class which stores a username, password, user ID, and a list of capabilities objects. Thirdly is the AccessControllSystem which contains a list of users and handles things like logins and checking if a user has access to a specific folder.

Finally, the main.java file has methods that call each other or operate in while loops to perform the necessary operations and string it all together.

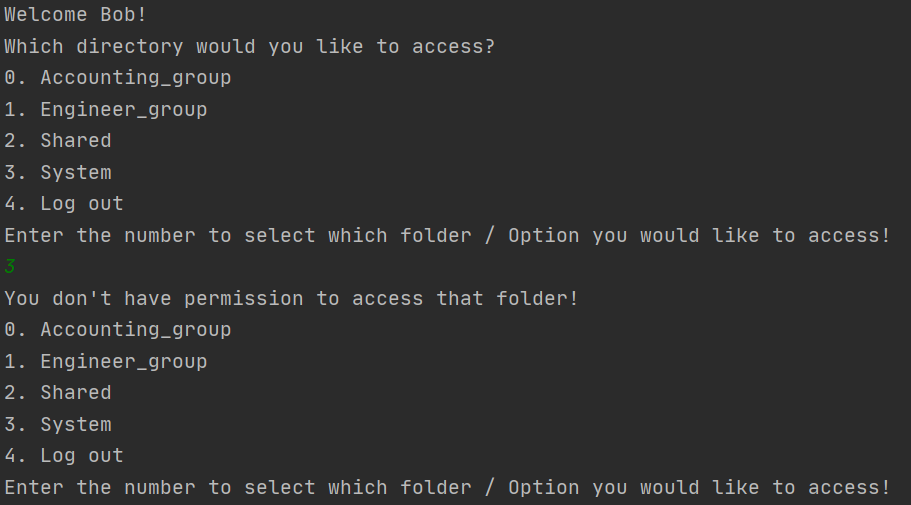
The first screen when someone starts up the program is a login screen.



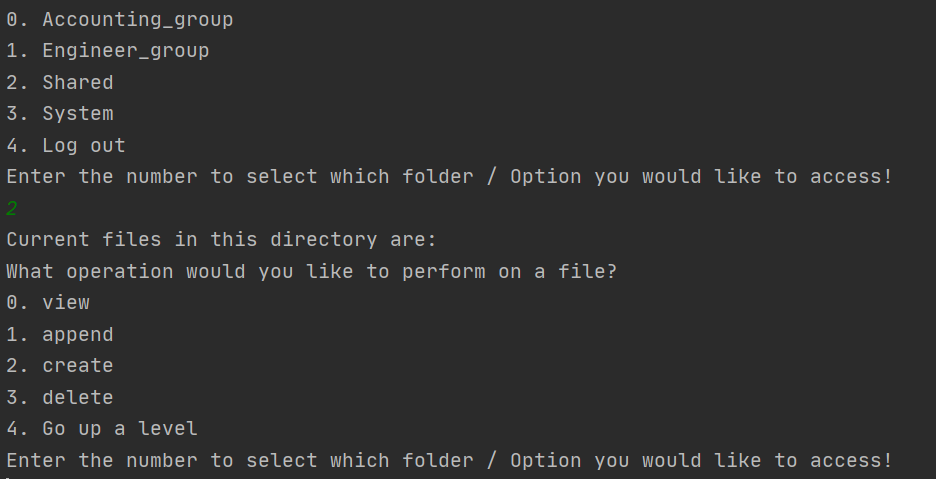
This screen also gives them the option to exit the program, the screen after it gives them the option to log out and they can then log in as someone else and all user input is validated so that out of bounds or inputting strings when ints are expected don’t crash the program or cause unexpected behavior.



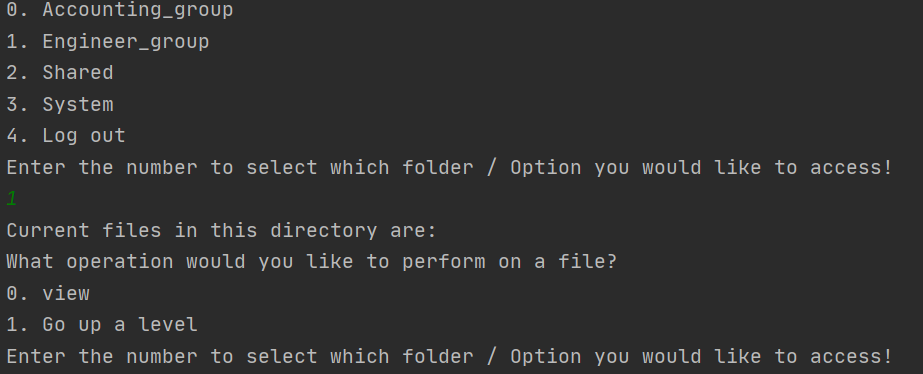
If a tries to access a folder they don’t have permission to access they are simply denied and prompted to try again.



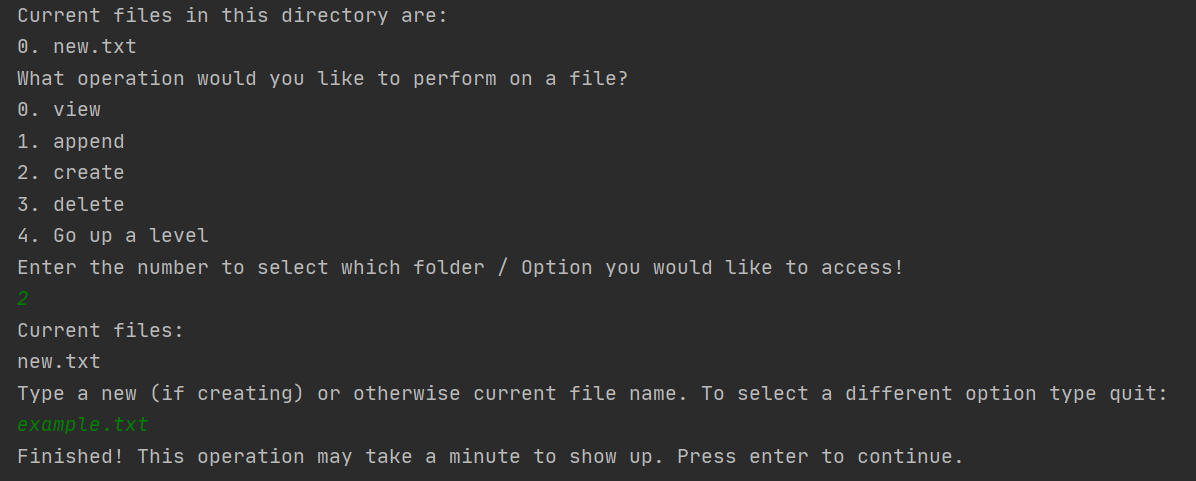
If they access a folder for which they have full permission, then they are given all the possible options (including the ability to go back to the last menu.)



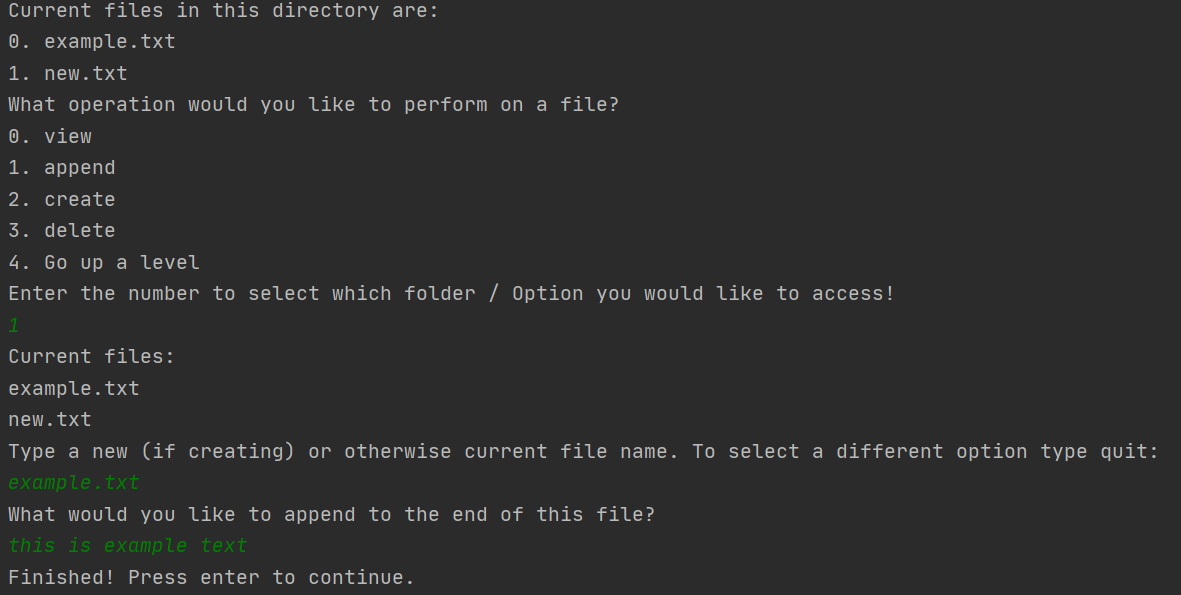
If they access folder for which they only have read permissions then I decided to only allow them to pick from the appropriate options instead of showing all options and then denying ones they don’t have privilege for. As you can see here. Bob can only perform view within the engineer\_group folder.



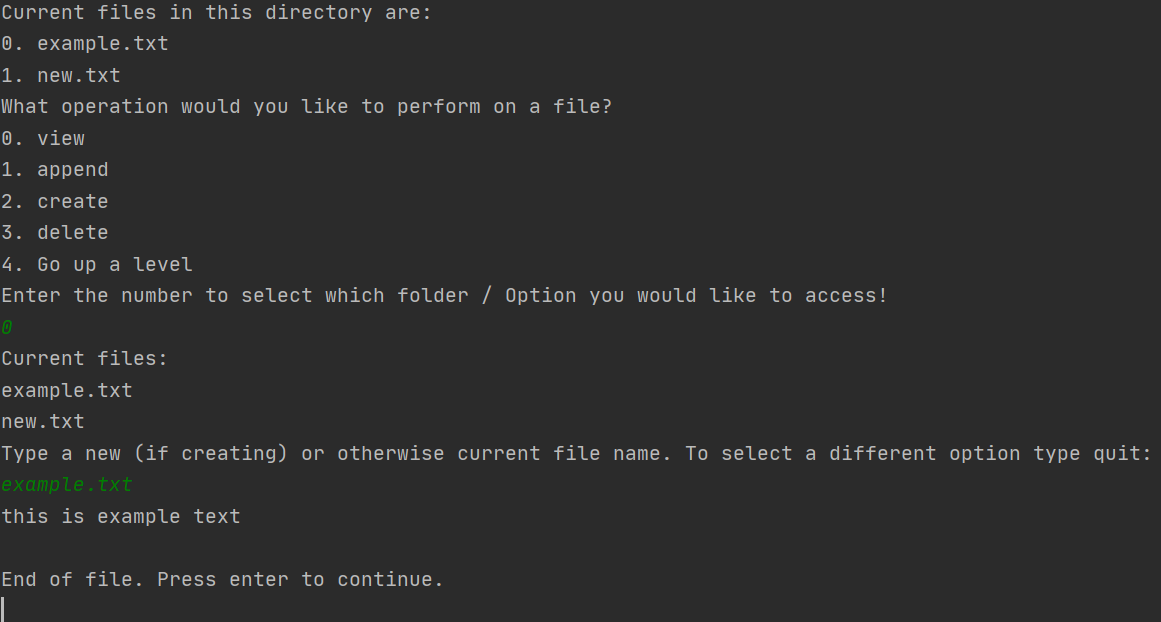
If Bob changes to a folder, he has full access rights to like shared then he can perform all the operations like create:



Append:



View:



And deletion. There are checks in place with all operations to make sure the file does (or as appropriate doesn’t) exist. There is also a check to make sure that the user input something because otherwise it would point towards whatever folder they are in and they would have the option to say delete that folder.

