Technical University of Denmark

Access Control Lab

02239 - Data Security

Group 7

Authors

Student Number:	Name:
s202789	Adamantios Alexandros Kounis
s202518	Christos Grigoriou
s213685	Nikolaos Karavasilis
s202074	Joachim Morken

November 19, 2021

Introduction

The current report is considered to be an extension of the previous Authentication Lab assignment. In the context of this lab we introduce two Access Control Policies (ACPs). To this end, a short description of the Access Control concept is presented to help the reader understand the problem. Recalling the previous authentication lab, an authenticated client is allowed to use services offered by the server. However, such systems are most commonly implemented in organizations where clients have different privileges depending on their hierarchical position. Therefore, constructing a proper ACP addresses the problem of how objects are accessed from different subjects with specific underlying access modes.

In this lab, we consider a small company which employs distinct users such as managers, janitors, service technicians, power and ordinary users with specific access rights to the system's objects. In our system this signifies that the manager is fully permitted to use the printer server, whereas the janitor, the technician and power users have less control rights. Regular users, have limited access rights to use basic printer server functions. Thus, on the client's side, access on services can vary from restricted use to full. On the server's side, the protective access control mechanism needs to be established appropriately to control which user or group of users is allowed to access the available printer services. Finally, this tailored access operates against vulnerability exploitation and intrusions.

An issue that needs special care while constructing an access control mechanism is the the degree of how fine or coarse the policy is going to be. With both incumbent and added privileged users and user groups, performance, efficiency and scalability are adequately handled. The next sections present the authentication lab extended in two access control approaches which are the *Access Control List* and the *Access Control Role Based* projects.

Access Control Lists

Firstly, we are considering the implementation of the Access Control List model. The solution is based on the access to the functionality defined by every single user of the system separately. Using this approach, see Table 1, it is possible to see how many users the system is consisting of as well as point to the rights that the users have when it comes to all the functionality of the system. The combination of a choice of a username as well as a method offered by the printer server, indicates either to a ✓ or a ✕, meaning authorised or not authorized respectively.

Operation / User	**Alice**	**Bob**	**Cecilia**	**David**	**Erica**	**Fred**	**George**
print	✓	✕	✓	✓	✓	✓	✓
queue	✓	✕	✓	✓	✓	✓	✓
topQueue	✓	✕	✓	✕	✕	✕	✕
start	✓	✓	✕	✕	✕	✕	✕
stop	✓	✓	✕	✕	✕	✕	✕
restart	✓	✓	✓	✕	✕	✕	✕
status	✓	✓	✕	✕	✕	✕	✕
readConfig	✓	✓	✕	✕	✕	✕	✕
setConfig	✓	✓	✕	✕	✕	✕	✕

Table 1: Access Control List

In our solution, the policy is saved in a .txt file which holds the same information as Table 1. The names appear on the first line of the file separated by space between them and then each following line starts with the method name to which the user can or can not have access to. Following the method name, again separated with space in between, there is a binary representation (1 or 0) defining whether the user has authority to access the functionality of which the row belongs to.

In more detail, at the time in which the server instance is being initiated, the server is reading the policy file and assigning its values into a 2-Dimensional Hash-Map. The first key holds the username of the user, which returns a second Hash-Map. The second key holds the name of the function of the system which then returns a Boolean value. At the time a user is attempting to login, if the login is successful, the server is generating a token for the user to use later on and it is also saving it alongside the username of the user that it belongs to.

Later on when the user attempts to take advantage of a functionality of the system, while passing on the token that was generated by the server, the system will first try to verify the token. Upon successful verification, the system will also compare the username, that comes along with the token, and the method which the user is trying to initiate with the 2-Dimensional Hash-Map to assess that the user has legitimate rights for accessing the function. Given that the Hash-Map returns True, the server is then allowed to proceed with the functionality of the selected operation. In any other case it returns to the user the following message *"Not Authenticated or User has no Authority"*. This indicates that the user has no authority to access the requested printer method, since the user is logged in already.

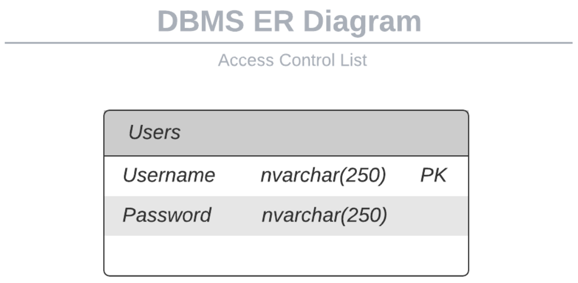


Figure 1: ER diagram (ACL)

Figure 1 represents the DBMS diagram, which is the same as in the previous assignment. It includes two columns, Username and Password which are both of type *nvarchar* with a character limit of 250. The Username is defined as a Primary Key to ensure that it is not possible to contain duplicates for the Username column.

Role Based Access Control

In this section we implemented Role Based Access Control. This implementation is based on a similar text file as in the previous section, with the difference being that a role is now associated with a set of possible operations. Therefore, it is essential that we know the role each user has, before allowing him to complete a printer method. Table 2 illustrates the system's subjects and their initial roles.

Subject	**Role**
Alice	Manager
Bob	Janitor, Service technician
Cecilia	Power user
David	User
Erica	User
Fred	User
George	User

Table 2: Initial Subjects and Roles

The hierarchy can be visualized in Figure 2 and described in the following manner: Managers are at the top, having access to all methods of the printer server. Under the manager we have the service technician, the janitor as well as the power user, each inheriting a subset of the possible operations a manager can perform. Finally we have the most simple role, namely a user. This user inherits only the two most basic access rights from the power user, namely the print and queue operations.

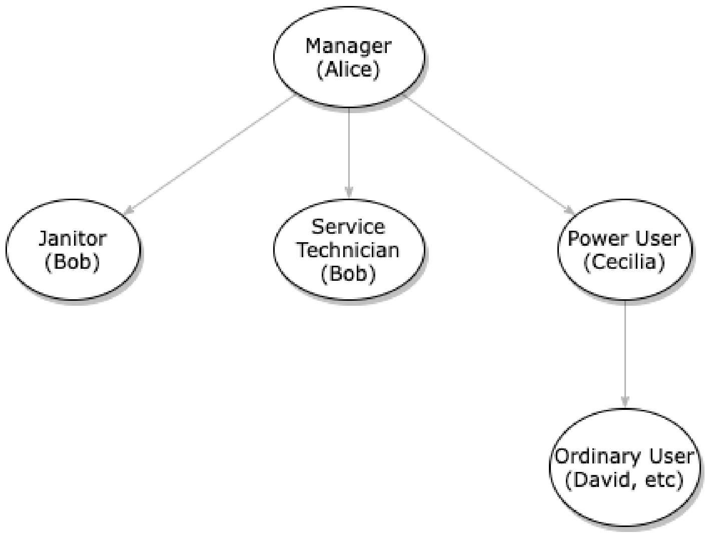


Figure 2: Role Hierarchy

Each role has permission to specific operations, as mentioned above, and is thoroughly illustrated in Table 3. The combination of a choice of a role as well as a method offered by the printer server points either to a ✓ or a ✗, meaning authorised or not authorized respectively.

The names appear on the first line of the file separated by space between them and then each following line starts with the method name to which the user can or can not have access to. Following the method name, again separated with space in between, there is a binary representation (1 or 0) defining whether the user has authority to access the functionality of which the row belongs to.

Operation / Role	**Manager**	**Janitor**	**Service Technician**	**Power User**	**User**
print	✓	✗	✗	✓	✓
queue	✓	✗	✗	✓	✓
topQueue	✓	✗	✗	✓	✗
start	✓	✓	✗	✗	✗
stop	✓	✓	✗	✗	✗
restart	✓	✓	✗	✓	✗
status	✓	✗	✓	✗	✗
readConfig	✓	✗	✓	✗	✗
setConfig	✓	✗	✓	✗	✗

Table 3: Role Based Access Policy

For the implementation of task 3, the policy was saved in a .txt file which holds the same information as Table 3. The file has the format of a two dimensional array, where the columns reflect the roles of the hierarchy and the rows the printer methods. In each array cell there is a binary representation (1 or 0) defining whether the specific role has authority to access the functionality of which the row belongs to. Furthermore, we store the roles of a user in the remote database, in which we also keep the users' credentials. When a user logs in, their roles is fetched from the remote database. As described above, a text file with the structure as Table 3 is loaded once the server starts. The fetched roles for the logged in user are simply compared to their corresponding roles in the text file, and thus we can create a data structure to store the logged in users' permissions. Similar to the ACL solution using the 2-Dimensional Hash-Map, except this time instead of using the username and function to retrieve the Boolean result, we use the role and the function. Considering that the users can have multiple roles assigned to them this function runs in a way that grants them access to the functions provided that at least one of the roles which they hold has the right to access the functionality. When a user attempts to perform an operation, the server simply checks the Hash-Map whether the user has the necessary permission. If permission is granted, the operation is executed, and vice versa if permission is not.

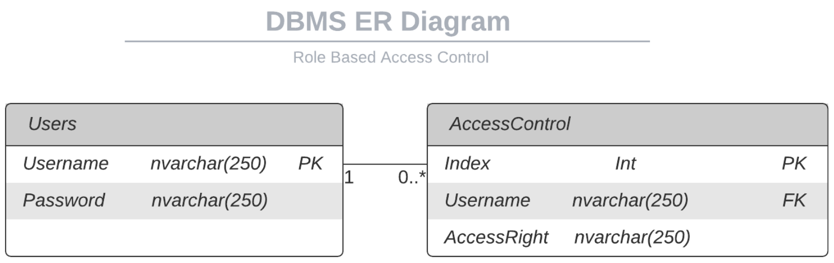

DBMS ER Diagram
Role Based Access Control
Users
Username nvarchar(250) PK
Password nvarchar(250)
1
0..*
AccessControl
Index Int PK
Username nvarchar(250) FK
AccessRight nvarchar(250)

Figure 3: ER diagram (RBAC)

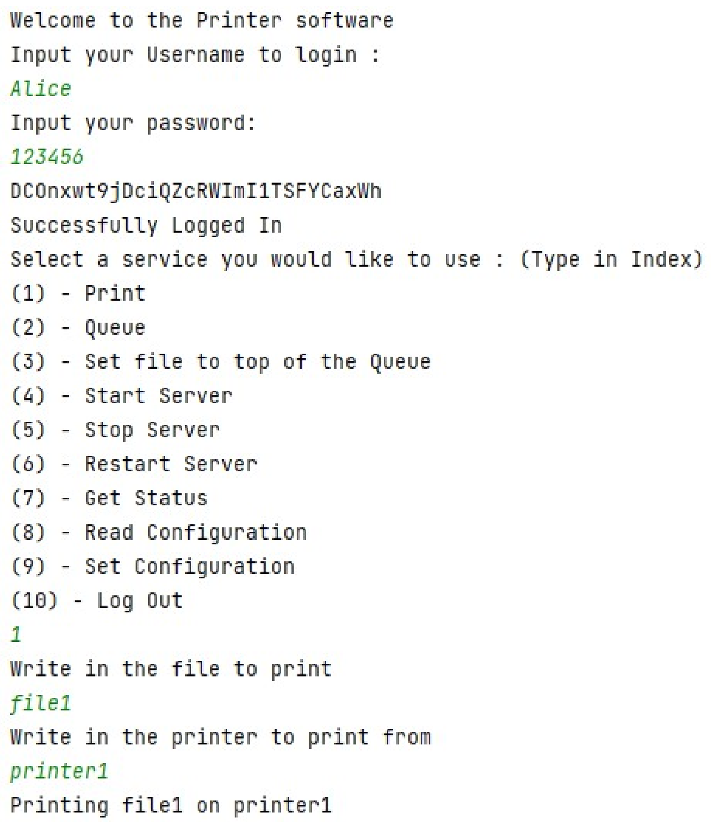
Figure 3 represents the DBMS Diagram, which contains the same Users table as in Figure 1, as well as an AccessControl. The table contains an Index which functions as a Primary Key, the *Username* is a foreign key to the *Username* of the Users table and the AccessRight column is meant to hold one of the roles defined in the .txt policy file. The relationship between the two tables in a 1 to 0..*, since our implementation supports a multi-role functionality for the Users.

Evaluation

This section demonstrates how access control is enforced in both prototypes of our project. Provided correct credentials[1] for users that have different position in the company's hierarchy, we observe that access is either granted or denied according to their access rights. Upon every service invocation by an authenticated user, the ACP is applied differently behind the scenes, for the ACL model by checking the user rights against the system operations and for the RBAC by checking the subject's role against the offered operations.

Concerning the client, the following screenshots illustrate how the system's requirements are met in both *ACL* and *RBAC* models. With respect to the server, messages of *Authority status: true* or *Authority status: false* denote the outcome of user invocation.

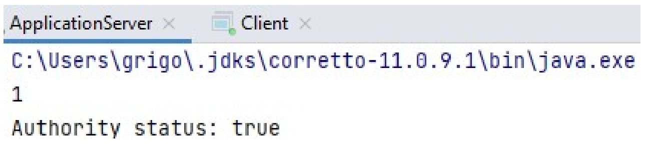
Screenshots from console output:
The services offered by the printer server are presented only in the first console screenshot and deliberately omitted in the following ones for saving space purposes.



```
Welcome to the Printer software
Input your Username to login :
Alice
Input your password:
123456
DCOnxwt9jDciQZcRWImI1TSFYCaxWh
Successfully Logged In
Select a service you would like to use : (Type in Index)
(1) - Print
(2) - Queue
(3) - Set file to top of the Queue
(4) - Start Server
(5) - Stop Server
(6) - Restart Server
(7) - Get Status
(8) - Read Configuration
(9) - Set Configuration
(10) - Log Out
1
Write in the file to print
file1
Write in the printer to print from
printer1
Printing file1 on printer1
```

Manager Alice *is allowed to use the Print service after a successful login*

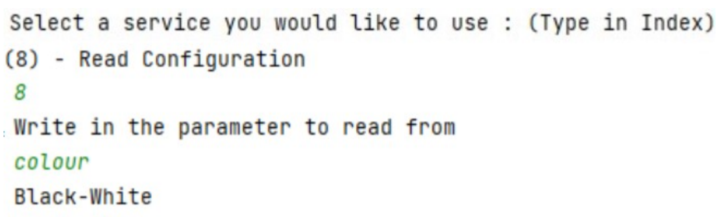
[1]In order to test our system's functionality one can provide:
username: The name of the user you want to log in with (e.g. Alice, Bob etc.)
password: 123456



ApplicationServer Client

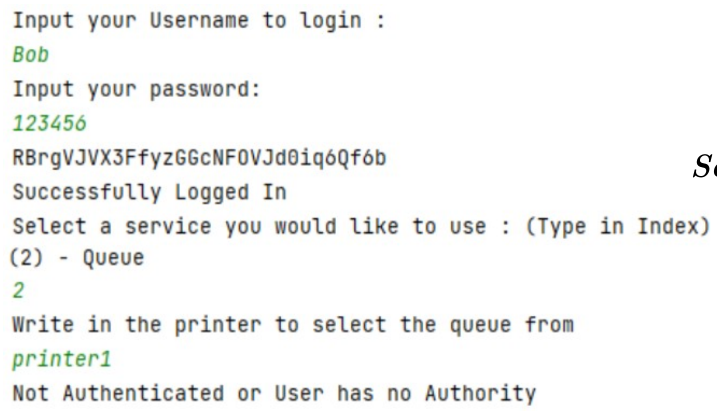
```
C:\Users\grigo\.jdks\corretto-11.0.9.1\bin\java.exe
1
Authority status: true
```

Alice's *invocation is allowed as illustrated on the server's console output*



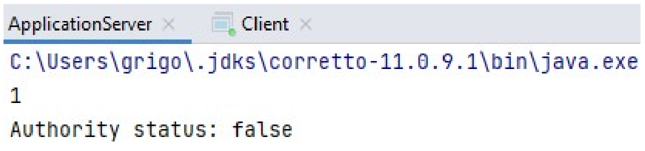
```
Select a service you would like to use : (Type in Index)
(8) - Read Configuration
8
Write in the parameter to read from
colour
Black-White
```

Manager Alice *is allowed to use the Read Configuration printer service*



```
Input your Username to login :
Bob
Input your password:
123456
RBrgVJVX3FfyzGGcNFOVJd0iq6Qf6b
Successfully Logged In
Select a service you would like to use : (Type in Index)
(2) - Queue
2
Write in the printer to select the queue from
printer1
Not Authenticated or User has no Authority
```

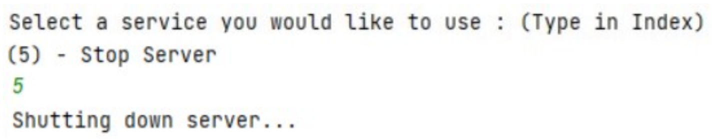
Service Technician - Janitor Bob *is not allowed to use the Queue printer service*



ApplicationServer Client

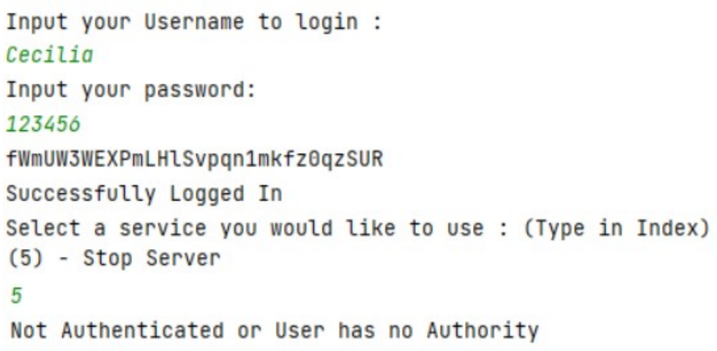
```
C:\Users\grigo\.jdks\corretto-11.0.9.1\bin\java.exe
1
Authority status: false
```

Bob's *invocation is not allowed as illustrated on the server's console output*



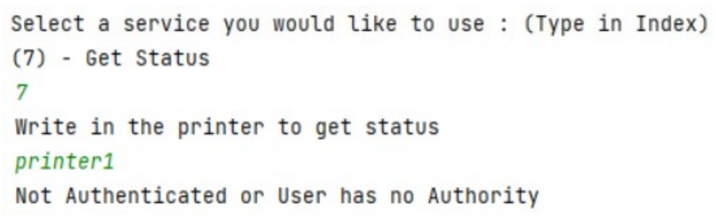
```
Select a service you would like to use : (Type in Index)
(5) - Stop Server
5
Shutting down server...
```

Janitor Bob *is allowed to use the Stop printer service*



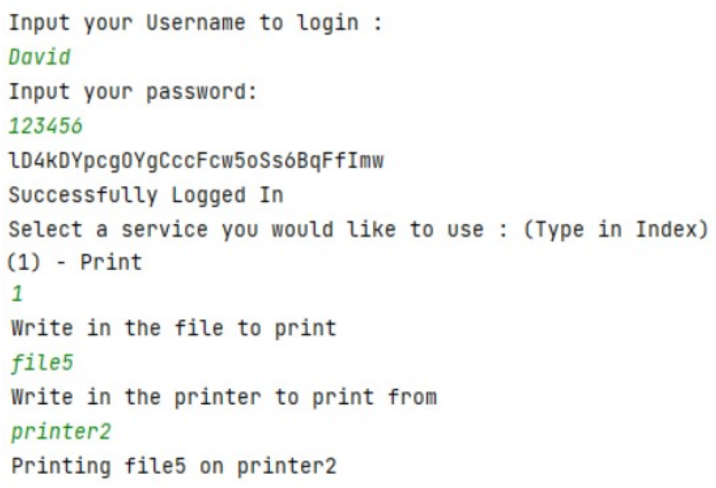
```
Input your Username to login :
Cecilia
Input your password:
123456
fWmUW3WEXPmLHlSvpqn1mkfz0qzSUR
Successfully Logged In
Select a service you would like to use : (Type in Index)
(5) - Stop Server
5
Not Authenticated or User has no Authority
```

Power User Cecilia *is not allowed to use the Stop printer service*



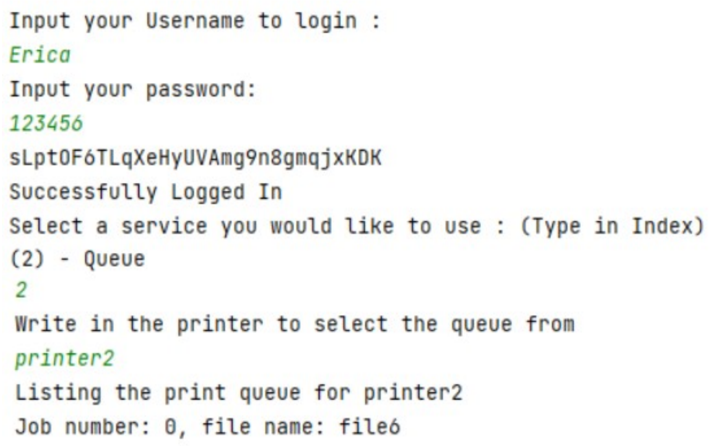
```
Select a service you would like to use : (Type in Index)
(7) - Get Status
7
Write in the printer to get status
printer1
Not Authenticated or User has no Authority
```

Power User Cecilia
is not allowed to use the
Status printer service



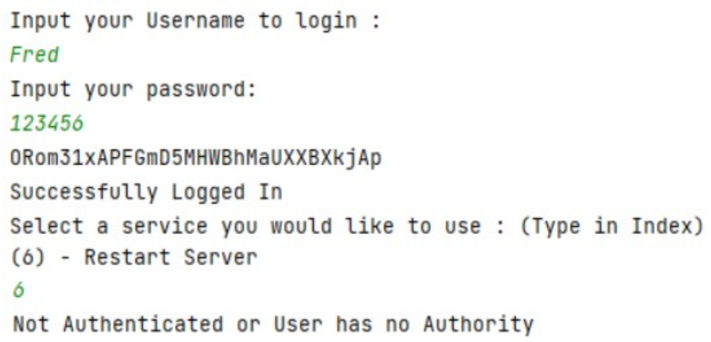
```
Input your Username to login :
David
Input your password:
123456
lD4kDYpcgOYgCccFcw5oSs6BqFfImw
Successfully Logged In
Select a service you would like to use : (Type in Index)
(1) - Print
1
Write in the file to print
file5
Write in the printer to print from
printer2
Printing file5 on printer2
```

User David
is allowed to use the
Print printer service



```
Input your Username to login :
Erica
Input your password:
123456
sLptOF6TLqXeHyUVAmg9n8gmqjxKDK
Successfully Logged In
Select a service you would like to use : (Type in Index)
(2) - Queue
2
Write in the printer to select the queue from
printer2
Listing the print queue for printer2
Job number: 0, file name: file6
```

User Erica
is allowed to use the
Queue printer service



```
Input your Username to login :
Fred
Input your password:
123456
ORom31xAPFGmD5MHWBhMaUXXBXkjAp
Successfully Logged In
Select a service you would like to use : (Type in Index)
(6) - Restart Server
6
Not Authenticated or User has no Authority
```

User Fred
is not allowed to use the
Restart printer service

System After Changes

The new ACP is outlined in Table 4 and Table 5 for ACL and RBAC respectively. To this end, George takes over Bob's responsibilities and becomes service technician while retaining his ordinary user rights. Bob's access in the system is ceased since he cannot get authenticated. Henry is hired in George's former position (ordinary user), and Ida is the new colleague of Cecilia.

Regarding the technical perspective, changes affect the **ACL** model where adjustments need to be done to the respective *.txt* file. More precisely, Bob is removed while new users (Henry and Ida) are added along with their respective rights in the given objects. Adjustments on new George access rights are also reflected in the system objects.

On the other hand, the *.txt* file of **RBAC** model remains intact due to its nature of accommodating just the roles of subjects. However, changes are reflected in the database since new users need to be added (Henry and Ida) both for role attachment and for the authentication process when system starts. George is assigned with a new role and thus he now has additional access rights. Finally Bob is removed from the database since he is not part of the company anymore and access in the system would now cause a violation in the policy.

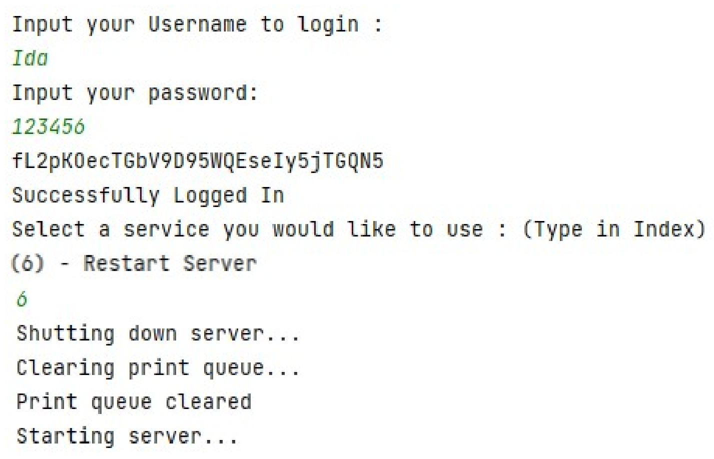
Operation / User	**Alice**	**Cecilia**	**David**	**Erica**	**Fred**	**George**	**Henry**	**Ida**
print	✓	✓	✓	✓	✓	✓	✓	✓
queue	✓	✓	✓	✓	✓	✓	✓	✓
topQueue	✓	✓	✗	✗	✗	✗	✗	✓
start	✓	✗	✗	✗	✗	✗	✗	✗
stop	✓	✗	✗	✗	✗	✗	✗	✗
restart	✓	✓	✗	✗	✗	✗	✗	✓
status	✓	✗	✗	✗	✗	✓	✗	✗
readConfig	✓	✗	✗	✗	✗	✓	✗	✗
setConfig	✓	✗	✗	✗	✗	✓	✗	✗

Table 4: Updated Access Control List

Subject	Role
Alice	Manager
Cecilia	Power user
David	User
Erica	User
Fred	User
George	Service Technician, User
Henry	User
Ida	Power user

Table 5: Updated Subjects and Roles

Screenshot of recently hired Power User Ida shows her access right to restart the printer server:



```
Input your Username to login :
Ida
Input your password:
123456
fL2pKOecTGbV9D95WQEseIy5jTGQN5
Successfully Logged In
Select a service you would like to use : (Type in Index)
(6) - Restart Server
6
Shutting down server...
Clearing print queue...
Print queue cleared
Starting server...
```

Power User Ida
is allowed to use the
Restart printer service

Discussion

After carefully specifying and implementing both approaches, it is clear to admit that using an ACP is a great mechanical process to implement in any application that resembles the printer server that we have looked into during the current and the previous assignments. Not having to hard-code the policy into the program, follows the clean code guidelines. Furthermore, it gives freedom to the user to open the policy text file and change the access rights without making any changes to the application.

After successfully implementing both approaches, we realized that both the ACL and RBAC serve their purpose sufficiently, although, one is more suitable in certain scenarios than the other. In more detail, given a scenario in which the rights of the users are changing frequently and there are not many users, if any, that share the same access rights, storing the information in an ACL format is more efficient. Rather than spending time constantly identifying schemes and hierarchies under which only one person might be falling under, it is far better to consider their identifier and specifically determine within the policy file what functionalities they are entitled to access. Naturally this has its own drawbacks, since the policy file needs to be constantly updated to match the changed criteria. Moreover, it can end up being rather too long in the event that a system is being used by a large number of users. ACL policy requires that user's information needs to be explicitly stated in the policy and not just the DBMS server. Therefore, this solution can become rather inconvenient when users have their accounts terminated or in the event of new users joining the system.

On the other hand, the role based access control constitutes a solution which needs a defined hierarchy within the system that is implementing it. Every user has a role value associated with their username and by utilizing that role, the user can access the functionality of the platform. These assigned role values are predetermined within the policy file. Arguably the biggest advantage of the RBAC is that if the hierarchy is well defined, any user, current or future, can fall under a role. As a consequence, the policy file does not need to undergo any change no matter how many users have been added to the system or had their profiles terminated from it. In the event that a user is being assigned a different role, the only place that a value needs to be changed is in the DBMS server under the access role column. That happens because, even the new rule with the new set of access rights would be already defined in the policy file, with no need for further changes on it.

Lastly, another advantage of the RBAC is the ease of changing the roles of the users without affecting the state of the application. Since the policy file is loaded with the initiation of the printer server instance, the policy is loaded into the system and remains constant until a new instance of the printer server is being created. Since the roles of each user are defined in the database, it is enough to change the value of the role there. Following that, if the user is logged into the system when that change occurs, they need to logout and login again. The change of role would be reflected without the server instance having to terminate and no data to be lost on the printers. In contrast, in the ACL implementation, in the event of a change in the policy file for a specific user, in order for them to access or be denied the newly assigned functionality, the whole print server instance would need to be terminated and restarted.

Role Based Access Control Clarification - *As mentioned earlier in the document, the reason why the user would need to logout and login again after the change, is because at login, the server creates a session token that is being associated locally with the role obtained from the database. This is particularly helpful, since a communication with the database about the role does not need to happen every time the user is accessing a functionality of the system. A change like this one occurs rather infrequently and reduces communication between server and Database to a constant for each user.*

Conclusion

In this report, we introduce the Access Control security concept to the print server prototype by developing two different mechanisms of Access Control Policies. These two policies, address the problem of offering different permission rights to different users of the printer. This is a necessary policy to have in place to restrict certain operations of the printer to a specific set of users or roles. Thus, the utter goal of this project is to minimize the security risk in likewise systems where resources are offered to individuals.

The first Access Control Policy we implemented is an Access Control List. This model relies on specifying permissions directly for a user. This approach results in a very flexible way of dealing with permissions for the users, as it is easy to manage the permissions text file and add or remove available operations for a user. However, this model might become cumbersome in systems that employ big number of users. In such cases where there are a lot of users to keep track of and there is not a clearly defined hierarchy, it might not be clear which users have access to which operations.

The other approach we implemented is Role-based Access Control. This model relies on specifying clearly defined roles along with the operations available to them. Users are then simply tied to a role (or roles) and adopt the role's corresponding operations. The benefit of this model is a concise and clear structure, which makes it easy to identify which users have access to which operations. It also offers an easy way of maintaining the permission rights, as the security agent can directly change a role of a user and thus their available operations changes. The drawback of this approach is that it is not very flexible since it is complex to create and assign a specific role for a user.

Both approaches are valid and capable of solving the underlying problem of defining different access rights for different users. While they achieve the same, they both offer different strengths and weaknesses. Access Control Lists are more convenient on small-scale user bases, offering the ease of quickly giving permission rights to a user without defining a hierarchy. On the other hand, Role-based Access Control thrives in centrally administrated environments with a larger user base, offering consistency, easy maintenance, and clearly defined roles in a hierarchy. Finally, project requirements are met, both in the initial tasks as well as in the final where changes take place.