Technical University of Denmark

02239

Data Security

Access Control Lab

Authors:	*Student Nr.:*
Jeff Gyldenbrand	s202790
Marcus Pagh	s151714

December 20, 2021

Contents

1 Introduction

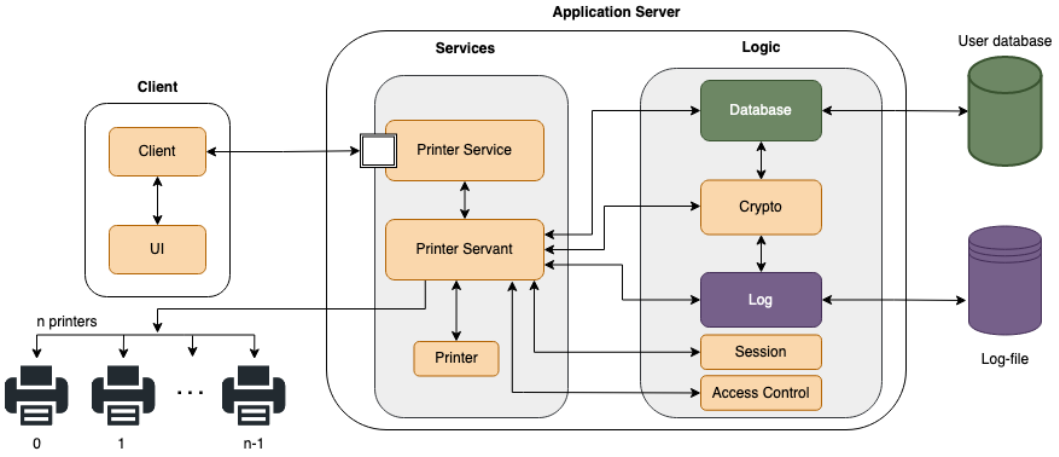

Figure 1.1: Overview of the system

Often in server/client setups not all clients are equal. It would therefore behove the organization to have some kind of measure to restrict access for certain clients. Throughout this assignment two ways different means of achieving said behavior is explored.

The simplest way to restrict user access is by using Access Control Lists (ACL). ACL works by storing a list of all users and their respective permissions. Any action taken by said user is checked against the list of permissions and will only be executed if the user in question has the particular permission in the list.

The second way this assignment is regulating client access is by implementing Role Based Access Control (RBAC). RBAC is similar to ACL in the sense that they both keep track of permissions in a separate list. However, while ACL appoints specific legal actions to each user, RBAC works by categorising permissions into specific roles that each user can be appointed to.

2 Access Control Lists

In the first part of the assignment, we have been given explicit client permissions and to accompany these new restrictions, an instance of ACL has been implemented. The list is shown in fig 2.1 and the table is added to the database as shown in fig 2.3. Whenever any method involving one of the restricted actions is invoked, the list is consulted before any logic is executed. If and only if the current user has the necessary permission(s) will the action be executed.

user	print	queue	topQueue	start	stop	restart	status	readConfig	setConfig
jeff	1	1	1	1	1	1	1	1	1
alice	1	1	1	1	1	1	1	1	1
bob	0	0	0	1	1	1	1	1	1
cecilia	1	1	1	0	0	1	0	0	0
david	1	1	0	0	0	0	0	0	0
erica	1	1	0	0	0	0	0	0	0
fred	1	1	0	0	0	0	0	0	0
george	1	1	0	0	0	0	0	0	0

Figure 2.1: Permissions table as implemented in database before company restructuring (Visualized).

user	print	queue	topQueue	start	stop	restart	status	readConfig	setConfig
jeff	1	1	1	1	1	1	1	1	1
alice	1	1	1	1	1	1	1	1	1
henry	1	1	0	0	0	0	0	0	0
cecilia	1	1	1	0	0	1	0	0	0
david	1	1	0	0	0	0	0	0	0
erica	1	1	0	0	0	0	0	0	0
fred	1	1	0	0	0	0	0	0	0
george	1	1	0	1	1	1	1	1	1
ida	1	1	1	0	0	1	0	0	0

Figure 2.2: Permissions table as implemented in database after company restructuring(Visualized).

```
CREATE TABLE permissions (
user varchar(20),
print boolean(0, 1),
queue boolean(0, 1),
topQueue boolean(0, 1),
start boolean(0, 1),
stop boolean(0, 1),
restart boolean(0, 1),
status boolean(0, 1),
readConfig boolean(0, 1),
setConfig boolean(0, 1)
);
```

Figure 2.3: Permissions table as implemented in database.

3 Role Based Access Control

In the second part of the assignment we are asked to extrapolate the previously defined permissions and group them into roles in a way that imposes the same restrictions but bound to predetermined types of relationships. The result of said grouping is shown in fig 3.1. Notice the extra permission "admin". This is not part of the original permission list and will be explained later.

roles	print	queue	topQueue	start	stop	restart	status	readConfig	setConfig	admin
manager	1	1	1	1	1	1	1	1	1	1
power-user	1	1	1	0	0	1	0	0	0	0
service-tech	0	0	0	1	1	1	1	1	1	0
janitor	0	0	0	0	0	1	0	0	0	0
user	1	1	0	0	0	0	0	0	0	0

Figure 3.1: Role table as implemented in database (Visualized).

In order to utilize RBAC instead of ACL, the database table for permissions is altered to hold the list of fig 3.1. The new database table is structured as shown in fig 3.4. Furthermore the users table are added an extra row, 'roles', to allow users to be assigned specific rows. The users table with assigned roles are shown in fig 3.2

user	password	salt	roles
jeff	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	manager
alice	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	manager
bob	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	janitor, service-tech
cecilia	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	power-user
david	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
erica	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
fred	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
george	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user

Figure 3.2: Role table as implemented in database before company restructuring (Visualized).

user	password	salt	roles
jeff	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	manager
alice	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	manager
cecilia	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	power-user
david	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
erica	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
fred	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user
george	`HashedPasswordCensoredForLatexPurposes|`	22-10-2021:21.18zz	user, service-tech
henry	`HashedPasswordCensoredForLatexPurposes|`	saltyMcSalty222	user
ida	`HashedPasswordCensoredForLatexPurposes|`	saltyMcSalty222	power-user

Figure 3.3: User table as implemented in database after company restructuring (Visualized).

```
CREATE TABLE permissions (
roles varchar(200),
print boolean(0, 1),
queue boolean(0, 1),
topQueue boolean(0, 1),
start boolean(0, 1),
stop boolean(0, 1),
restart boolean(0, 1),
status boolean(0, 1),
readConfig boolean(0, 1),
setConfig boolean(0, 1),
admin boolean(0, 1)
);
```

Figure 3.4: Role table as implemented in database.

Whenever any method involving one of the restricted actions is invoked, the particular clients role is consulted before that role is is then looked up in the permissions table. If and only if the current user has a role with the necessary permission(s) will the action be executed.

4 Evaluation

4.1 ACL

Requirements	**Satisfied**
Print server prototype modified to enforce a ACL-policy	yes
The ACL is specified in a external file	yes
All registered user are included in the file	yes
The external file is loaded when the server starts	yes
Can handle the company restructuring of the final task	yes
Manual UI	no[1]

Figure 4.1: Requirements for ACL

1

Considering the situation "Bob leaves the company and George takes over the responsibilities as service technician. At the same time, two new employees are

[1]Automated tests are instead mocking user-interactions with feedback still being printed in terminal. Tests are made in such a way that any sequence of user interactions can easily be put together by copy-pasting the necessary actions from the current tests.

hired: Henry, who should be granted the privileges of an ordinary user, and Ida who is a power user and should be given the same privileges as Cecilia." multiple things are happening.

The first part "Bob leaves the company and George takes over the responsibilities as service technician." is done relatively easy by manually altering the database rows of the permissions table involving Bob and George. George's row is simply altered so every 0 where Bob has a 1 is flipped to a 1. Now George will have every permission Bob had in addition to his own (As Bob was only a service technician and not a user, he did not have the permission to print, as George did. George retains this ability while being granted extra permissions).

The second part "two new employees are hired: Henry, who should be granted the privileges of an ordinary user, and Ida who is a power user and should be given the same privileges as Cecilia." is more tricky as it involves adding extra users. The easiest way to do this would be to manually copying a row of a user with a known password and altering the username and permission to correspond to the new employees. This however would impose severe security risks as the passwords would not be secret. Instead we have opted to allow anyone to sign up and create their own user, by choosing their own username and password. New users are created with zero permissions, so in addition to Henry and Ida manually signing up, the admin would have to also alter the rows in the permissions table of the database, just like (s)he did with George to set the corresponding permissions for the new employees.

The altered table corresponding to the specified situation is shown in fig 2.2. For reference, the table before the restructuring can be found in fig 2.1.

4.2 RBAC

Requirements	**Satisfied**
Print server prototype modified to enforce a RBAC-policy	yes
The RBAC is specified in a external file	yes
All registered user are included in the file	yes
The external file is loaded when the server starts	yes
Can handle the company restructuring of the final task	yes
Manual UI	no[1]

Figure 4.2: Requirements for RBAC

2

Considering the same situation "Bob leaves the company and George takes over the responsibilities as service technician. At the same time, two new employees are hired: Henry, who should be granted the privileges of an ordinary user, and Ida who

is a power user and should be given the same privileges as Cecilia." again with the RBAC version, the changes are implemented a bit differently.

Again, in order to let George take upon himself the responsibility of the service technician, an admin can manually alter the database table permissions, but instead of flipping ones and zeroes, (s)he must now simply add "service-tech" to the roles of George. Again Bobs row can either be removed entirely or simply stripped of all roles.

However, to satisfy our own curiosity as well as refraining from doing the same thing once again, we have added the extra column, 'admin', to the permissions table and granted this permission only to the role of 'manager'. The purpose of the 'admin' permission is to allow editing the external database file directly from the UI instead of doing so manually. The 'admin' permission allows the creation of new users as well as (de)assigning roles.

Once a manager has made the appropriate changes to account for the specified situation, the permissions table shown in fig 3.1 remains the same as before, while the users table shown in fig 3.2 is altered to look as shown in fig 3.3

5 Discussion

At first glance one might think that ACL allows for more flexibility than RBAC, since with ACL any and all actions can either be allowed or disallowed for any specific users, while with RBAC such flexibility is restricted to fit certain roles. However by looking at it a little bit longer it is possible to conclude that both methods can actually provide the exact same functionality. If no role correspond to the necessary permissions, roles can either be combined or a new role can simply be created. The only difference is how they are implemented and managed. While a small company with fluid roles might enjoy the ability to define specific client permissions, a larger company with a lot of users coming and going, it might impose fewer mistakes and a more fluent process when the roles are already predetermined.

With the small scenario given to us, it is essentially up to personal preference as there is no real difference. However, when it comes to the design choices we made together with the two versions, a few key take aways does come up.

For instance with ACL we opted to let any new user self-sign up while with RBAC we opted to only allow new users to be created by a manager. There is no particular reason that we did not do it the other way around or just did the same thing twice. While allowing users to self-sign up makes up vulnerable to flooding attacks, only letting managers create new users might cost more in terms of availability. Also, if allowing self-sign up with RBAC, if the admin were to decide the initial permissions should change, by simply changing the permissions of the initial role, the changes would instantly apply to previously self-signed up users. With ACL such a change

would only apply to users signing up after the changes are made.

With a lot of different actions ACL might be more prone to user-errors, while RBAC is more intuitive for the less techy manager when new clients or employees need appropriate roles. Therefore we opted to let managers (de)assign user roles directly from the UI.

With RBAC we also opted to only allow users with the 'admin' permission to create new users. While our implementation requires the manager to manually specify both password and salt, in any real-life situation these should obviously be randomly generated and somehow securely transferred to the user in question. This does however provide an easy way of managing initial roles in situations where these are often not the same.

6 Conclusion

While both implementations satisfied all requirements and offer the same amount of flexibility, in conclusion RBAC is probably the preferred choice in most situations as it is more intuitive to use for the less techy person, provides the same amount of flexibility if needed as well as providing a faster way of (de)assigning permissions which is a great benefit in situations with a lot of change in clients.

However ACL does have its place when custom roles are the rule rather than the exception.

7 Setup and running the project

- **The software used in this project:**
  - **IDE:** Eclipse[1]. Version: 2020-12 (4.18.0).
  - **Database:** SQLite[2].

7.1 How to edit external file

As in the previous lab, we have opted to store user data in a database. As we continue this lab by implementing permissions, the database-file now also serves as the "external policy file, or another form of external media, that is loaded when the print server starts".

As we have chosen to use SQLite, this external DB-file does however look confusing in comparison to a plain-text file, if opened by a regular text editor.

In order to manually change permissions, in both the ACL and RBAC versions, the file should be opened and edited by a viewer capable of rendering such a file. While something like PHPMyAdmin might be the most well-known, we did not bundle this with our program. If you have no editor capable of editing this file in a satisfactory

matter, we suggest using the online tool found at `https://extendsclass.com/sqlite-browser.html`

7.2 Running the projects

Run the project:

- Install Eclipse to your system.
- In Eclipse: File > Import > General > Existing Projects into Workspace > Select archive file > Locate the zipped project (final_projects.zip) > Finnish
- Two projects appears: **PrintServer-acl** and **PrintSercer-rbac**
- Run src/main/java/test/Main.java in each of the projects
- *if localport is used pick a new port number*

When application is running:

- You will get the option to
  (1) run the test program
  (2) exit the application
- For both projects, the test will run through all operation on each of the users to demonstrate which users are allowed to do what.
- The **log** is printed to either the folder /**log**/ or at root depending on the operating system it is running on. We did'nt have time to fix this.

All mentioned functionality has been implemented, although the manual UI has not been prioritized. Every possible action is shown in the automated tests, which has been structured in such a way that any sequence of client actions can easily be put together. Any and all output are still shown in the terminal as if the user had been interacting live.

8 Appendix

8.1 Illustrations

References

[1] Eclipse. "IDE". In: (). URL: `https://www.eclipse.org/downloads/packages/installer`.

[2] SQLite. "database". In: (). URL: `https://github.com/xerial/sqlite-jdbc/releases`.