02239 - Data Security

Access Control Lab

s171783 - Onat Buyukakkus
s181314 - Altug Tosun

14.11.2018

Contents

1 Introduction

In most of the applications of client/server architectures, access control is a problem that needs to be addressed. Access control allows users with different permissions to invoke different remote methods. With this way, some users are able to benefit from some features of the system according to their access. In the context of this assignment, within an enterprise business there are often lots of tools and accounts being used day to day by people within the company, such as printing services. If we wanted some feature of these services to be accessible by only certain people we would need to implement many new services. Instead we can assign access or roles for the users and check those when they want to invoke a remote method.

In this assignment we will extend Authentication Lab with the functionality of the print server so that the server will support client access levels for specific remote methods. In Authentication Lab, all users in the database are able to invoke all remote methods on the server. In order to increase usability of the print server, we will define what actions can be taken by which users, since then the server will be able to support a wider range of users.

Our solution will include both the Access Control List (ACL) and Role Based Access Control (RBAC) approaches. In the ACL approach we will add access for different methods for each user in the database and whenever a user tries to invoke a method, we will check with our database whether that user is able to invoke that method or not.

In the RBAC approach, we will create roles and add roles of each users in the database. Then add access for different methods for each role in the database and whenever a user tries to invoke a method, we will check with our database whether that user's role is able to invoke that method or not.

We will implement both approaches as two different projects so that we can compare those two. Since our implementation will be built on top of Authentication Lab, so all communication between the client and the server takes place on a secure network and each user is authenticated.

2 Access Control Lists

Access Control List approach consists of assigning a list of permissions to each user individually. To be able to implement that, we have added a new table `Permissions`. Context of the table is in Table 1.

username	permission
Alice	CAN_RESTART
Alice	CAN_START
Alice	CAN_STOP
Alice	CAN_PRINT
Alice	CAN_READ_CONFIG
Alice	CAN_WRITE_CONFIG
Alice	CAN_READ_STATUS
Alice	CAN_READ_QUEUE
Alice	CAN_EDIT_QUEUE
Bob	CAN_RESTART
Bob	CAN_START
Bob	CAN_STOP
Bob	CAN_READ_CONFIG
Bob	CAN_WRITE_CONFIG
Bob	CAN_READ_STATUS
Cecilia	CAN_RESTART
Cecilia	CAN_PRINT
Cecilia	CAN_READ_QUEUE
Cecilia	CAN_EDIT_QUEUE
David	CAN_PRINT
David	CAN_READ_QUEUE
Erica	CAN_PRINT
Erica	CAN_READ_QUEUE
Fred	CAN_PRINT
Fred	CAN_READ_QUEUE
George	CAN_PRINT
George	CAN_READ_QUEUE

Table 1

Every time a user invokes a remote method, after authenticating user, we check whether that user has permission to run that method. Code snippets for `print` method and database access checking method `hasAccess` are in Figure 1 and 2.

```java
    public String print(String filename, String printer, User user) throws IOException, NoSuchAlgorithmException {
        if(!login(user)) {
            this.logger.info("Authentication failed for method: print user: " + user.getUsername());
            return "Authentication failed.";
        }
        if(!hasAccess(user, "CAN_PRINT")) {
            this.logger.info("Access failed for method: print user: " + user.getUsername());
            return "Access failed.";
        }
        this.logger.info("Authentication and access successful for method: print user: " + user.getUsername());
        PrintJob printJob = new PrintJob(filename, printer);
        this.printQueue.add(new Pair<Integer, PrintJob>(this.jobId++,printJob));
        return "File added to queue.";
    }
```

Figure 1: Java code for `print` method

```java
    public Boolean hasAccess(User user, String access) {
        return SQLiteJDBC.hasAccess(user.getUsername(), access);
    }

    public static Boolean hasAccess(String username, String access) {
        Connection c = null;
        Statement stmt = null;
        boolean hasAccess = false;

        try {
            Class.forName("org.sqlite.JDBC");
            c = DriverManager.getConnection("jdbc:sqlite:printaccess1.db");
            c.setAutoCommit(false);

            stmt = c.createStatement();
            String sql = "SELECT EXISTS(SELECT 1 FROM Permissions WHERE username= '" + username + "' and permission= '"
                     + access + "') AS bool;";
            ResultSet rs = stmt.executeQuery(sql);

            if (rs.next()) {
                hasAccess = rs.getBoolean("bool");
            }

            rs.close();
            stmt.close();
            c.close();
        } catch (Exception e) {
            System.err.println(e.getClass().getName() + ": " + e.getMessage());
            System.exit(0);
        }
        return hasAccess;
    }
```

Figure 2: Java code for `hasAccess` method

3 Role Based Access Control

Role Based Access Control approach consists of assigning roles to each user. Each role has different permissions. We haven't gone with a role hierarchy since roles are not interdependently related to each other. This allows for a much simpler scheme.

We have added two new tables `UserRoles` and `UserRolePermissions`. Context of these two tables is in Table 2.

role	permission
1	CAN_RESTART
1	CAN_START
1	CAN_STOP
1	CAN_PRINT
1	CAN_READ_CONFIG
1	CAN_WRITE_CONFIG
1	CAN_READ_STATUS
1	CAN_READ_QUEUE
1	CAN_EDIT_QUEUE
2	CAN_RESTART
2	CAN_START
2	CAN_STOP
2	CAN_READ_CONFIG
2	CAN_WRITE_CONFIG
2	CAN_READ_STATUS
3	CAN_RESTART
3	CAN_PRINT
3	CAN_READ_QUEUE
3	CAN_EDIT_QUEUE
4	CAN_PRINT
4	CAN_READ_QUEUE

role	name
1	manager
2	service_technician
3	power_user
4	regular_user

Table 2

We have added a column for each user `role`. With this implementation we are able to create new roles or change permissions of existing roles without having to change users. Just like in ACL implementation in Section 2, every time a user invokes a remote method, after authenticating user, we check whether that user role has permission to run that method. Code snippets for `print` method and database access checking method `hasAccess` are in Figure 3 and 4.

```java
public String print(String filename, String printer, User user) throws IOException, NoSuchAlgorithmException {
    if(!login(user)) {
        this.logger.info("Authentication␣failed␣for␣method:␣print␣user:␣" + user.getUsername());
        return "Authentication␣failed.";
    }
    if(!hasAccess(user, "CAN_PRINT")) {
        this.logger.info("Access␣failed␣for␣method:␣print␣user:␣" + user.getUsername());
        return "Access␣failed.";
    }
    this.logger.info("Authentication␣and␣access␣successful␣for␣method:␣print␣user:␣" + user.getUsername());
    PrintJob printJob = new PrintJob(filename, printer);
    this.printQueue.add(new Pair<Integer, PrintJob>(this.jobId++,printJob));
    return "File␣added␣to␣queue.";
}
```

Figure 3: Java code for `print` method

```java
    public Boolean hasAccess(User user, String access) {
        return SQLiteJDBC.hasAccess(user.getUsername(), access);
    }

    public static Boolean hasAccess(String username, String access) {
        Connection c = null;
        Statement stmt = null;
        boolean hasAccess = false;

        try {
            Class.forName("org.sqlite.JDBC");
            c = DriverManager.getConnection("jdbc:sqlite:printaccess2.db");
            c.setAutoCommit(false);

            stmt = c.createStatement();
            String sql = "SELECT␣EXISTS(SELECT␣1␣FROM␣UserRolePermissions,␣Users␣WHERE␣Users.username=␣'" + username + "'␣and␣Users.role=␣UserRolePermissions.role␣and␣UserRolePermissions.permission␣=␣'"
                     + access + "')␣AS␣bool;";
            ResultSet rs = stmt.executeQuery(sql);

            if (rs.next()) {
                hasAccess = rs.getBoolean("bool");
            }

            rs.close();
            stmt.close();
            c.close();
        } catch (Exception e) {
            System.err.println(e.getClass().getName() + ":␣" + e.getMessage());
            System.exit(0);
        }
        return hasAccess;
    }
```

Figure 4: Java code for `hasAccess` method

4 Evaluation

4.0.1 ACL

Before changes

If we create try to invoke methods with two users that one has access for the method and one has not, only the user with the permission will be able to invoke the method. As an example we have tried to start the print server with two users (Erica and Alice), only Alice is be able to start the server. Also, Bob is not able to print since he has no access meanwhile Fred can access print method. You can see the test we have created in Figure 5.

```java
        Service printService = (Service) Naming.lookup("rmi://localhost:5099/printaccess1");
        printService.start(erica);
        printService.start(alice);

        printService.print("file1", "printer1", fred);
        printService.print("file2", "printer2", bob);

        printService.queue(fred);
```

Figure 5: Java code for ACL test before changes

We have created log files on the server side for each remote method. Whenever a remote method is invoked, log file is appended with either authentication/access success or fail. Figure 6 shows the log file for code snippet in Figure 5.

```
    Nov 21, 2018 3:40:55 PM server.ServiceImpl <init>
    INFO: Server started.
    Nov 21, 2018 3:41:10 PM server.ServiceImpl start
    INFO: Access failed for method: start user: Erica
    Nov 21, 2018 3:41:10 PM server.ServiceImpl start
    INFO: Authentication and access successful for method: start user: Alice
    Nov 21, 2018 3:41:10 PM server.ServiceImpl print
    INFO: Authentication and access successful for method: print user: Fred
    Nov 21, 2018 3:41:10 PM server.ServiceImpl print
    INFO: Access failed for method: print user: Bob
    Nov 21, 2018 3:41:10 PM server.ServiceImpl queue
    INFO: Authentication and access successful for method: queue user: Fred
```

Figure 6: Log file for example test of ACL before changes

After changes

When Bob leaves the company and George takes over the responsibilities as service technician, first we needed to update `Permission` table in order to change George's permissions which is a hustle since we need to update every single permission for every single method. Then we needed to delete Bob from the database.

When two new employees Henry and Ida hired, again we needed to add every single permission for every single method for both employees in the database. Again which is not practical at all is very error prone. So final table is formed in Table 3.

username	permission
Alice	CAN_RESTART
Alice	CAN_START
Alice	CAN_STOP
Alice	CAN_PRINT
Alice	CAN_READ_CONFIG
Alice	CAN_WRITE_CONFIG
Alice	CAN_READ_STATUS
Alice	CAN_READ_QUEUE
Alice	CAN_EDIT_QUEUE
George	CAN_RESTART
George	CAN_START
George	CAN_STOP
George	CAN_READ_CONFIG
George	CAN_WRITE_CONFIG
George	CCAN_READ_STATUS

username	permission
Cecilia	CAN_EDIT_QUEUE
Cecilia	CAN_RESTART
Cecilia	CAN_PRINT
Cecilia	CAN_READ_QUEUE
David	CAN_READ_QUEUE
David	CAN_PRINT
Erica	CAN_READ_QUEUE
Erica	CAN_PRINT
Fred	CAN_READ_QUEUE
Fred	CAN_PRINT
Henry	CAN_READ_QUEUE
Henry	CAN_PRINT
Ida	CAN_RESTART
Ida	CAN_PRINT
Ida	CAN_READ_QUEUE
Ida	CAN_EDIT_QUEUE

Table 3

When we try to test changes we have done with code snippet in Figure 7 we got the logs in Figure 8.

```
        Service printService = (Service) Naming.lookup("rmi://localhost:5099/printaccess1");
        printService.start(erica);
        printService.start(alice);

        printService.print("file1", "printer1", henry);
        printService.print("file2", "printer2", george);

        printService.queue(ida);
```

Figure 7: Java code for ACL test after changes

```
    Nov 21, 2018 5:42:55 PM server.ServiceImpl <init>
    INFO: Server started.
    Nov 21, 2018 5:44:10 PM server.ServiceImpl start
    INFO: Access failed for method: start user: Erica
    Nov 21, 2018 5:44:10 PM server.ServiceImpl start
    INFO: Authentication and access successful for method: start user: Alice
    Nov 21, 2018 5:44:10 PM server.ServiceImpl print
    INFO: Authentication and access successful for method: print user: Henry
    Nov 21, 2018 5:44:10 PM server.ServiceImpl print
    INFO: Access failed for method: print user: George
    Nov 21, 2018 5:44:10 PM server.ServiceImpl queue
    INFO: Authentication and access successful for method: queue user: Ida
```

Figure 8: Log file for example test of ACL after changes

4.0.2 RBAC

Before changes

Since the only implementation difference between ACL and RBAC is on the database, we were able to test RBAC access permissions with the same code snippet in Figure 5.

Just as with ACL, we were able to give access to users for different methods if only they have the permission to invoke that method. Again we have created log files on the server side for each remote method. Figure 9 shows the log file for the test.

```
    Nov 21, 2018 4:12:44 PM server.ServiceImpl <init>
    INFO: Server started.
    Nov 21, 2018 4:13:06 PM server.ServiceImpl start
    INFO: Authentication and access successful for method: start user: Alice
    Nov 21, 2018 4:13:06 PM server.ServiceImpl start
    INFO: Access failed for method: start user: Erica
    Nov 21, 2018 4:13:06 PM server.ServiceImpl print
    INFO: Authentication and access successful for method: print user: Fred
    Nov 21, 2018 4:13:06 PM server.ServiceImpl print
    INFO: Access failed for method: print user: Bob
    Nov 21, 2018 4:13:06 PM server.ServiceImpl queue
    INFO: Authentication and access successful for method: queue user: Fred
```

Figure 9: Log file for example test of RBAC before changes

After changes

When Bob leaves the company and George takes over the responsibilities as service technician, we only needed to remove Bob from the database and change the role of George which is way more easier and less error prone than changes in the ACL.

When two new employees Henry and Ida hired, again we only needed to roles for both employees in the database. Again which is very practical and is not error prone at all. `UserRoles` and `UserRolesPermissions` tables in Table 2 are not changed at all. Only three lines are changed in the `User` table which can be seen in Table 4.

username	password	salt	role
George	ok1ePXas6rMS0Ccfe5610eYB4...	VY1Li1FRE5c=	2
Henry	yWDvfJ54qo8EIPFR14VCGhpI...	a1J2glQyBNY=	4
Ida	bzEkb6Ii88Llg4mMKCuldmPW...	2gT/i30RkAo=	3

Table 4

When we try to test changes we have done with code snippet in Figure 7 we got the logs in Figure 10.

```
    Nov 21, 2018 6:18:33 PM server.ServiceImpl <init>
    INFO: Server started.
    Nov 21, 2018 6:20:10 PM server.ServiceImpl start
    INFO: Access failed for method: start user: Erica
    Nov 21, 2018 6:20:10 PM server.ServiceImpl start
    INFO: Authentication and access successful for method: start user: Alice
    Nov 21, 2018 6:20:10 PM server.ServiceImpl print
    INFO: Authentication and access successful for method: print user: Henry
    Nov 21, 2018 6:20:10 PM server.ServiceImpl print
    INFO: Access failed for method: print user: George
    Nov 21, 2018 6:20:10 PM server.ServiceImpl queue
    INFO: Authentication and access successful for method: queue user: Ida
```

Figure 10: Log file for example test of RBAC after changes

As it can be seen in Section 2, 3 and above tests, we were able to satisfy following requirements:

- **ACL**
  - We were able to add user permissions individually in the database.
  - We were able to check if given user has permission to run remote method every time methods are invoked. This means that we were able to use ACL to give access to users.
  - When someone leaves the company and another employee takes over the responsibilities of the left employee, we needed to update the database for the employee to change each permission.
  - When a new employee is hired, we needed to add permissions for each method for the new employee.
- **RBAC**
  - We were able to introduce roles to the database, add roles to existing users and add permissions for each role.
  - We were able to check if given user's role has permission to run remote method every time methods are invoked. This means that we were able to use RBAC to give access to users according to their roles.
  - When someone leaves the company and another employee takes over the responsibilities of the left employee, we only needed to change the role of the employee that took over.
  - When a new employee is hired, instead of adding permissions for each method, we only needed to add the role of the new employee.

5 Discussion

After adapting final task with both ACL and RBAC by updating our database, we have concluded following:

Advantages of ACL

- ACLs are superior where the security is tied to an individual, apart from his or her role with the company.
- In the ACL model the data owner can decide who has access (if he has that permission on the data) and add or remove people from the list.

In the context of this assignment, for the print server, there is no advantages to use ACL instead of RBAC as far as we can see. But, if we think about for a different context, there might be some advantages to use ACL instead of RBAC. If every user has different rights in the system and we require refined roles for each user, such as a military system, it is more convenient to use ACL for two main reasons. Firstly, there will be no use of the roles that we use in RBAC. Secondly, access is given at the discretion of some user in ACL so that we will have roles more confidentially.

Advantages of RBAC

- The main benefit of RBAC over ACL, is ease of management - in principle you have a very few roles, centrally administered, no matter how many users, and its just a question of granting each user the correct role; as opposed to ACL, where for each new user (or change in user, or deletion, etc), you have to go around to all the resources user needs access to and add them to the list.
- RBAC is more flexible, easy to manage and less error prone compared to ACL.

Therefore, it would be more flexible and easy to manage the users in our print server. Because, there will be a lot of users that is going to have same rights and permissions in the hierarchy. So, it is not sensible to record every access that user have in the Access Control List. We just need to keep user's roles and what is the rights of that role. In this way, when somebody leaves the company and we need replace this person with somebody else, we just change the roles of the users in the system rather than adding and deleting roles one by one from list.

6 Conclusion

As shown in Section 4, our implementation successfully gives access to users with permission for specific methods when invocation. For ACL, this is done by storing permissions linked to each user in database. For RBAC, this is done by storing permissions linked to roles and roles linked to users. So, we were able to satisfy access requirement with ACL and RBAC.

After adapting changes, as discussed in Section 5 we have decided that for this specific assignment, it is more efficient and effective to use RBAC.

In terms of outlining possible future work, it would be beneficial to implement dependency relations between roles in RBAC model for more complex access control schemes. That way roles would inherit permissions and it would be more flexible.