Department of Computer Engineering

University of Peradeniya

CO527 Advanced Database Systems

Lab Number : 05
Topic : Database Security
Lab Date : 28th May 2020 from 2.00 PM to 4.00 PM
Due Date : 07th June 2020 before 11:55 PM

1. Objective

At the end of this lab session, you will learn how to control access to the databases itself and how to control access to the data objects within the database.

2. Introduction

Database security discusses techniques for securing databases against a variety of threats. One way of ensuring this is by providing access privileges to authorised users. We will be looking at how commercial DBMS provides different types of security.

3. Security Types

3.1 Authentication

When you first attempt to access an instance or database, the authentication system will try to determine if you are who you say you are and this is done by the underlying operating system to verify your user IDs and passwords. Almost all DBMS products use some form of username and password security.

3.2 Authorisation

DBMS products limit certain actions on certain objects to certain users or groups (also called roles). This is to determine what operations a user can perform within a database. Authorisation consists of the privileges, roles, and label-based access control credentials.

3.3 Privileges

So far, you have been issuing all database commands as the database administrator which has privileges to access all the utilities, data, and database objects. It is important that users be only given privileges that are necessary to complete their tasks.Types of privileges:

- ALL
- SELECT
- INSERT, DELETE, UPDATE
- CREATE, ALTER, DROP

3.4 GRANT and REVOKE

- **GRANT** – grant users with privileges.

```
GRANT privileges [columns]
ON object
TO 'user'@'localhost' [IDENTIFIED BY 'password'] [WITH GRANT OPTION]
```

Example 1:

```
GRANT ALL
ON db.*
TO dbuser IDENTIFIED BY 'userpassword'
```

- **REVOKE** – remove privileges

```
REVOKE privilege
ON object
FROM user
```

Example 2:

```
REVOKE INSERT
ON db.*
FROM 'user'@'localhost';
```

- Some other useful MySQL commands:
  - CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';
  - DROP USER 'user1'@'localhost';
  - To view grants for super user or root:

```
SHOW GRANTS;
```

- To view grants for any user [user]:
  SHOW GRANTS FOR [user];

- To view user table in mysql:
  SELECT USER,HOST from MYSQL.USER;

- Once you have finalized the permissions that you want to set up for your new users, always be sure to reload all the privileges issuing the following command:
  FLUSH PRIVILEGES;

4. In Class Exercise

In the following scenario a new member has joined you and you are the super user of the database. We will look at how to assign specific authorities and privileges to the new member to safeguard the security of the database.

1. Create database company security.
2. Load the given company security.sql file to the company security database.
3. Create a new user ‘user1’ within the MySQL shell.
4. Login to MySQL with a new user account and password and see if the new user has any authorities or privileges to the database.
5. Make sure the new user has only read only permission to ‘Employee’ table.
6. Now allow ‘user1’ to query the followings: SELECT * FROM Employee; INSERT into Employee(...)VALUES(...). What happens? Fix the problem.
7. From user1 create a view WORKS ON1(Fname,Lname,Pno) on EMPLOYEE and WORKS ON. (Note: You will have to give permission to user1 on CREATE VIEW). Give another user ‘user2’ permission to select tuples from WORKS ON1(Note: user2 will not be able to see WORKS ON or EMPLOYEE).
8. Select tuples from user2 account. What happens?
9. Remove privileges of user1 on WORKS ON and EMPLOYEE. Can user1 still access WORKS ON1? What happened to WORKS ON1? Why?

4.1 SQL Injection Attacks

SQL injection attack occurs when data from the user is used to modify a SQL statement.

Example:
Suppose that a simplistic authentication procedure issues the following query and checks to see if any rows were returned:

```
SELECT  *
FROM    users
WHERE   username = ‘jake’ and PASSWORD = ‘jakespasswd’;
```

The attacker can try to change (or manipulate) the SQL statement, by changing it as follows:

```
SELECT  *
FROM    users
WHERE   username = 'jake' and (PASSWORD = 'jakespasswd? or 'x' = 'x');
```

As a result, the attacker who knows that 'jake' is a valid login of some user is able to log into the database system as 'jake' without knowing his password and is able to do everything that 'jake' may be authorized to do to the database system.

Since you do not have any authentication related tables in your database, you can try the following two queries to see what happens in an SQL injection.

```
SELECT  *
FROM    employee
WHERE  ssn=999887777;
```

```
SELECT  *
FROM    employee
WHERE  ssn=999887777 or 'x'='x';
```

Observe the result. What happens?

5. Assignment

Consider the relational database schema provided. Suppose that all the relations were created by (and hence are owned by) user X, who wants to grant the following privileges to user accounts A, B, C, D, and E:

I. Account A can retrieve or modify any relation except DEPENDENT and can grant any of these privileges to other users.
II. Account B can retrieve all the attributes of EMPLOYEE and DEPARTMENT except for Salary, Mgr ssn, and Mgr start date.
III. Account C can retrieve or modify WORKS ON but can only retrieve the Fname, Minit, Lname, and Ssn attributes of EMPLOYEE and the Pname and Pnumber attributes of PROJECT.
IV. Account D can retrieve any attribute of EMPLOYEE or DEPENDENT and can modify DEPENDENT.
V. Account E can retrieve any attribute of EMPLOYEE but only for EMPLOYEE tuples that have Dno = 3.

Write SQL statements to grant these privileges. Use views where appropriate.

6. Submission

Submissions have to be done on or before 05.06.2020. Submit a single E_1X_XXX_lab05.zip file including all your .sql scripts and explanations in a .pdf file.