Assignment number: 09

Roll Number: **4958**

A) Title of the assignment:

Simulation of SQL Injection.

B) Description of SQL Injection attack and various methods (various input strings) that cause such attack.

SQL injection is a common attack vector used to run malicious SQL code to query the backend database and manipulate it to access information that was not intended to be displayed.

This information may include a variety of information related to that domain and also may have some sensitive data belonging to a company or private details of their customers. The attacker can manipulate the data in such a way that it causes change in the behavior of the application.

It is one of the most used attack vectors in order to retrieve sensitive company data.

The typical process of the attack is to insert a malicious SQL query into the front-end input fields which are to be processed by the database. Once the weaknesses of the entire system are exposed it can be used to query the database in a direct format.

This can lead to:

- Exposure to highly confidential data- passport/credit-card/etc.
- Authorization details being exposed.

- Deletion of entire data store, and/ corruption of database, execution of Operating System commands.
- Entire system can get compromised.

✻ Some of the methods of attack are as follows:

1. Using Line comments like:

   ```
   SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'
   ```

   This comments the entire part after the – hence retrieving all the admin records from the table.

2. Stacking Queries-Union/Drop/Alter

   ```
   SELECT * FROM members WHERE username = 'admin' AND password = 'password’; DROP TABLE members /*
   ```

   or

   ```
   SELECT * FROM `city`; ALTER TABLE student RENAME to states;/*
   ```

   Execution of more than one query in a transaction.

3. Get information about the database like:

   ```
   SELECT * FROM information_schema.tables;
   ```

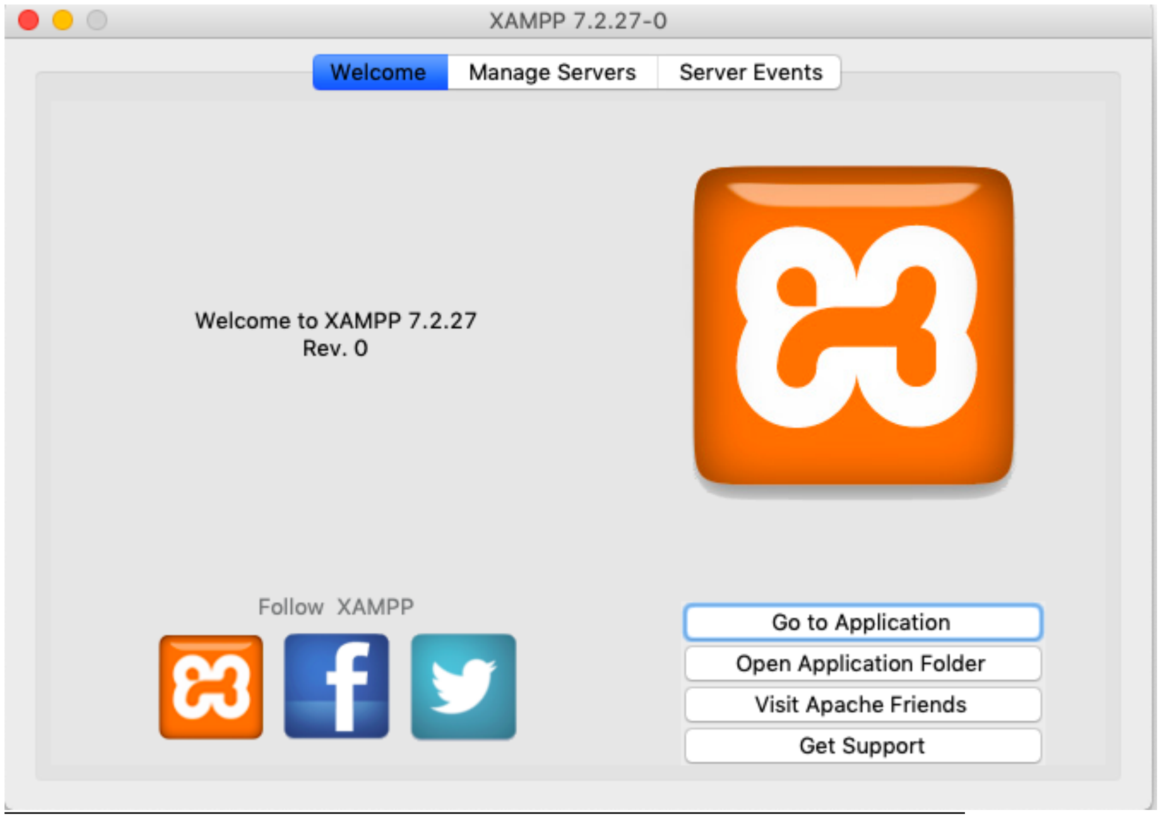
   Gives all the information about the database like the tables/columns present in it.

C) Installation steps of your system Environment:

1. OS Configuration:

   macOS 10.14.6

2. Local server: XAMPP



Welcome to XAMPP 7.2.27
Rev. 0

Figure: XAMPP Server

3. Database- SQL database provided by phpMyAdmin functionality of XAMPP. Acts as our remote database for local testing purposes.

4. Database Name: inject

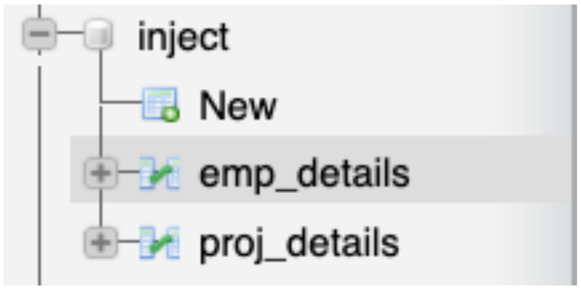


Figure: Database Structure

It has two tables:

(1)“emp_details” : which has the following structure

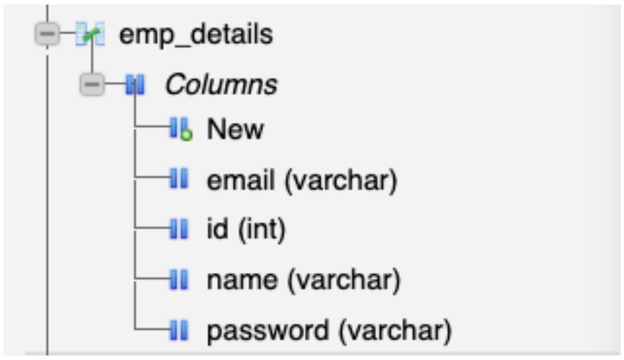


Figure: table 1 structure

(2) “proj_details”: which has the following structure

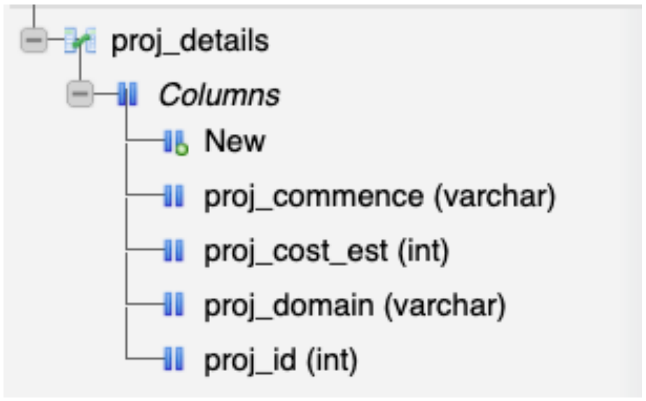
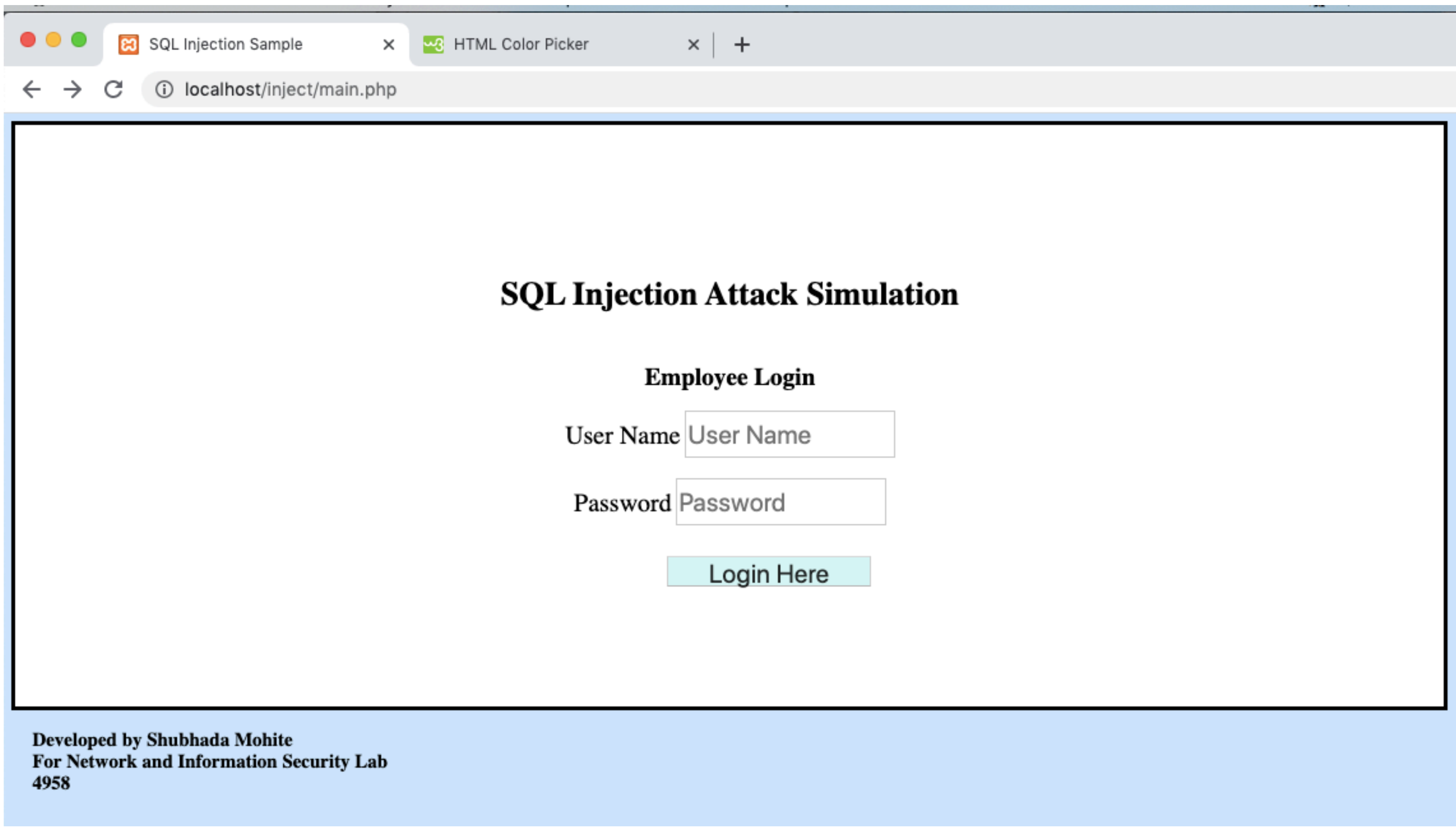


Figure: table 2 structure

The overall website purpose is for the employees of the firm to login and see the project details that they are currently working on. Only legitimate users can see the project.

The admin has the right to see all the employee details and the project details.

The login page is accessed using the php files on the local apache server:
http://localhost/inject/main.php



SQL Injection Attack Simulation

Employee Login

User Name

Password

Developed by Shubhada Mohite
For Network and Information Security Lab
4958

Figure: Employee Login Page

D) Demonstration of actual SQL Injection Attack

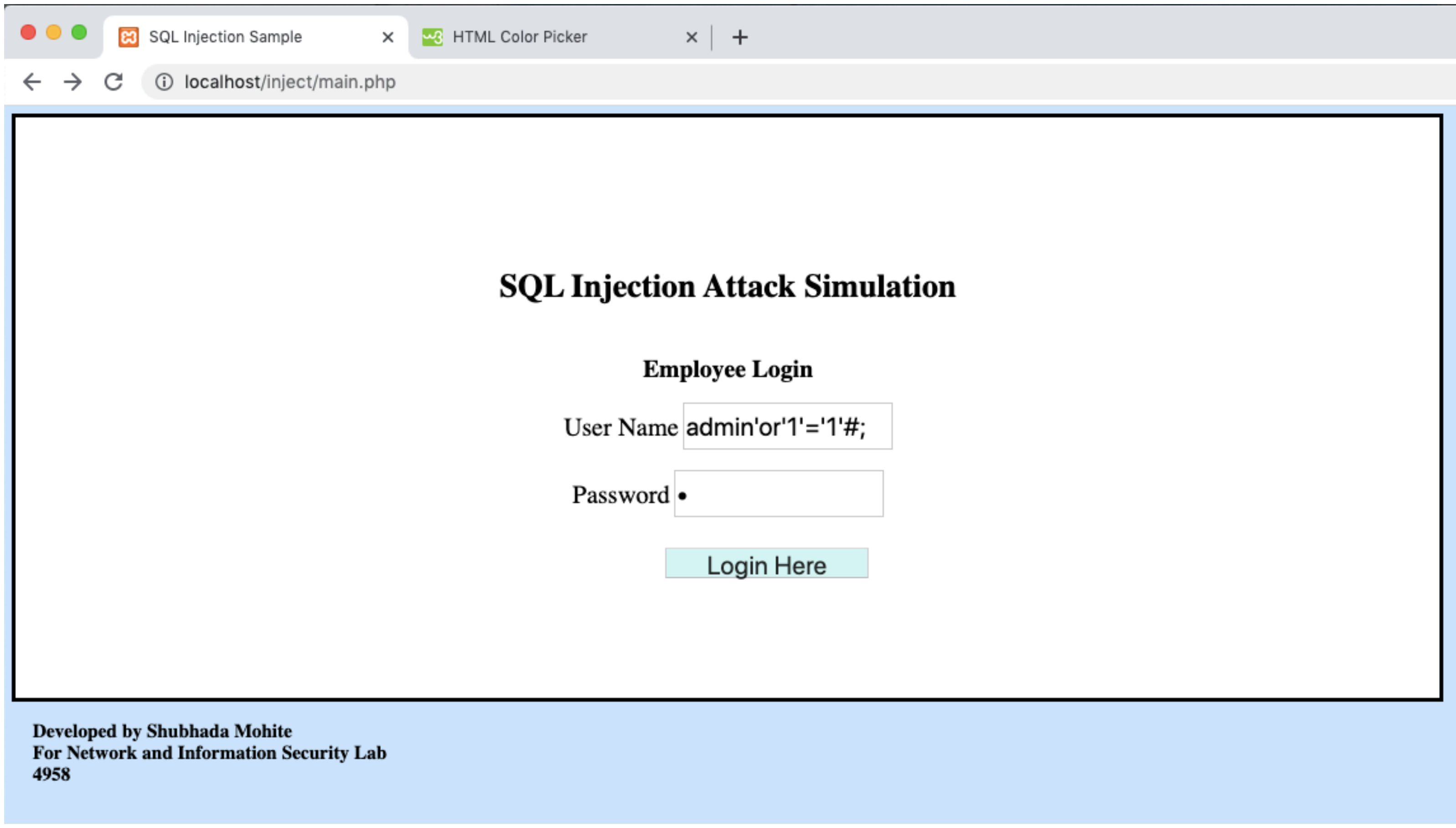
<u>1) Bypassing the login screen</u> /using 'OR' logic in the query

E.g: **SELECT * FROM emp_details WHERE ItemNumber = 9999 OR 1=1**

Since the 1=1 part of the query will always be true and thus makes the entire part after the WHERE clause true.

This query ends up returning all the entries from the table.

This logic can be used to bypass the login screen, or to gain illegitimate access to the database.



SQL Injection Attack Simulation

Employee Login

User Name admin'or'1'='1'#;

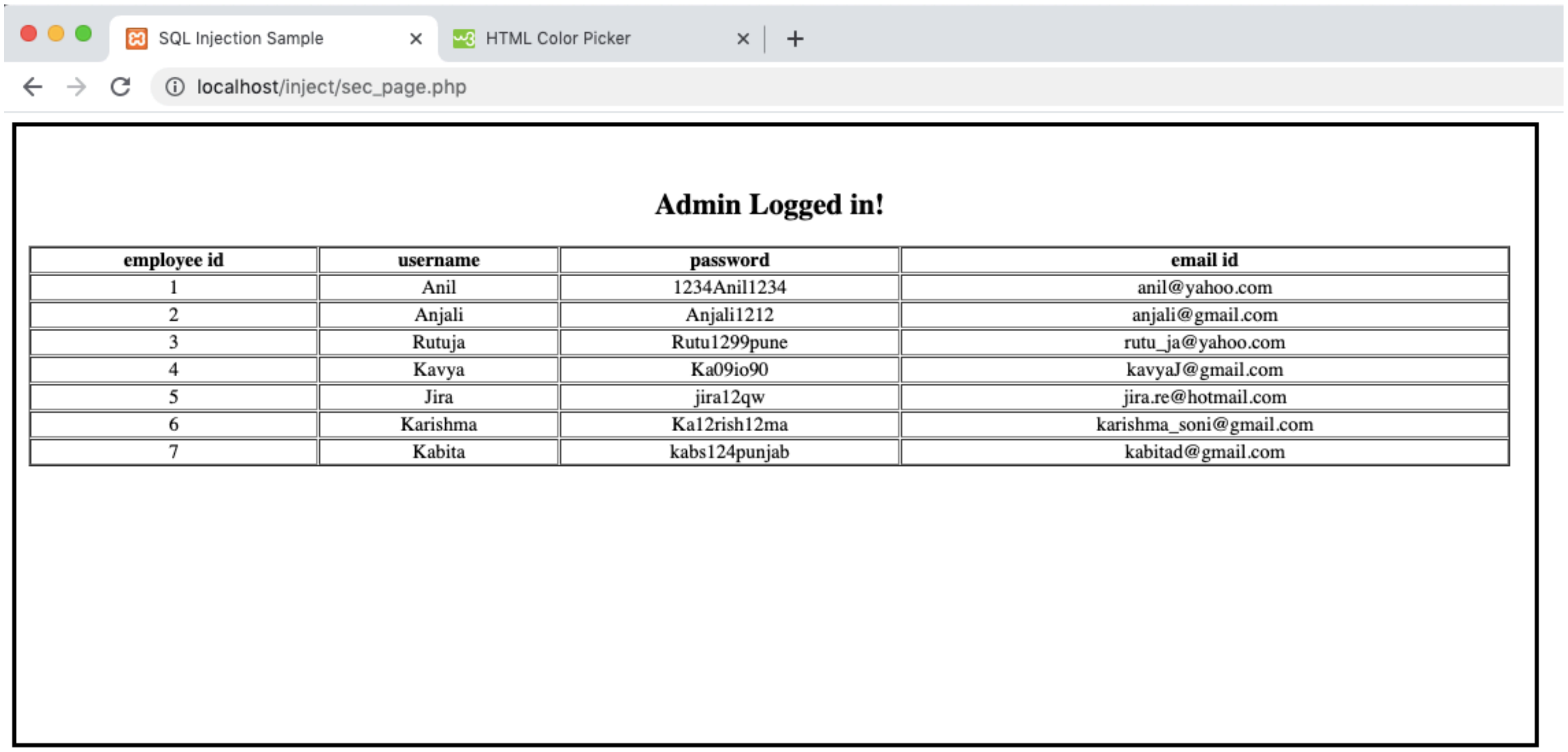
Password •

Developed by Shubhada Mohite
For Network and Information Security Lab
4958

Figure: SQL Injection Attack Scenario 1

After pressing the button, the effective query becomes as follows:

```
select * from emp_details where 1;
```


SQL Injection Sample

HTML Color Picker

localhost/inject/sec_page.php

Admin Logged in!

employee id	username	password	email id
1	Anil	1234Anil1234	anil@yahoo.com
2	Anjali	Anjali1212	anjali@gmail.com
3	Rutuja	Rutu1299pune	rutu_ja@yahoo.com
4	Kavya	Ka09io90	kavyaJ@gmail.com
5	Jira	jira12qw	jira.re@hotmail.com
6	Karishma	Ka12rish12ma	karishma_soni@gmail.com
7	Kabita	kabs124punjab	kabitad@gmail.com

Figure: SQL Injection Attack Result 1

Thus returning all the records of the table which otherwise is only accessible to the admin.

2) Stacking Queries - executing more than one queries in one session/transaction.

```
SELECT * FROM emp_details WHERE ItemNumber = 9999; DROP TABLE emp_details
SELECT * FROM emp_details WHERE id = 9999; RENAME TABLE emp_details
TO student;
```

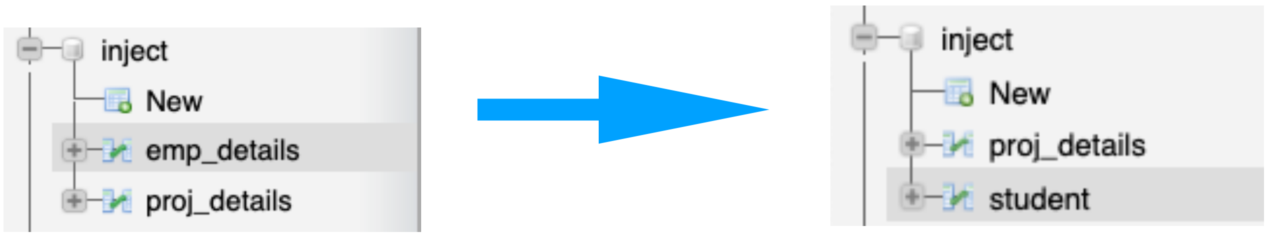


Figure: SQL Injection Attack Result 1

As shown in the figure above the malicious query changes the name of the table from emp_details to student.

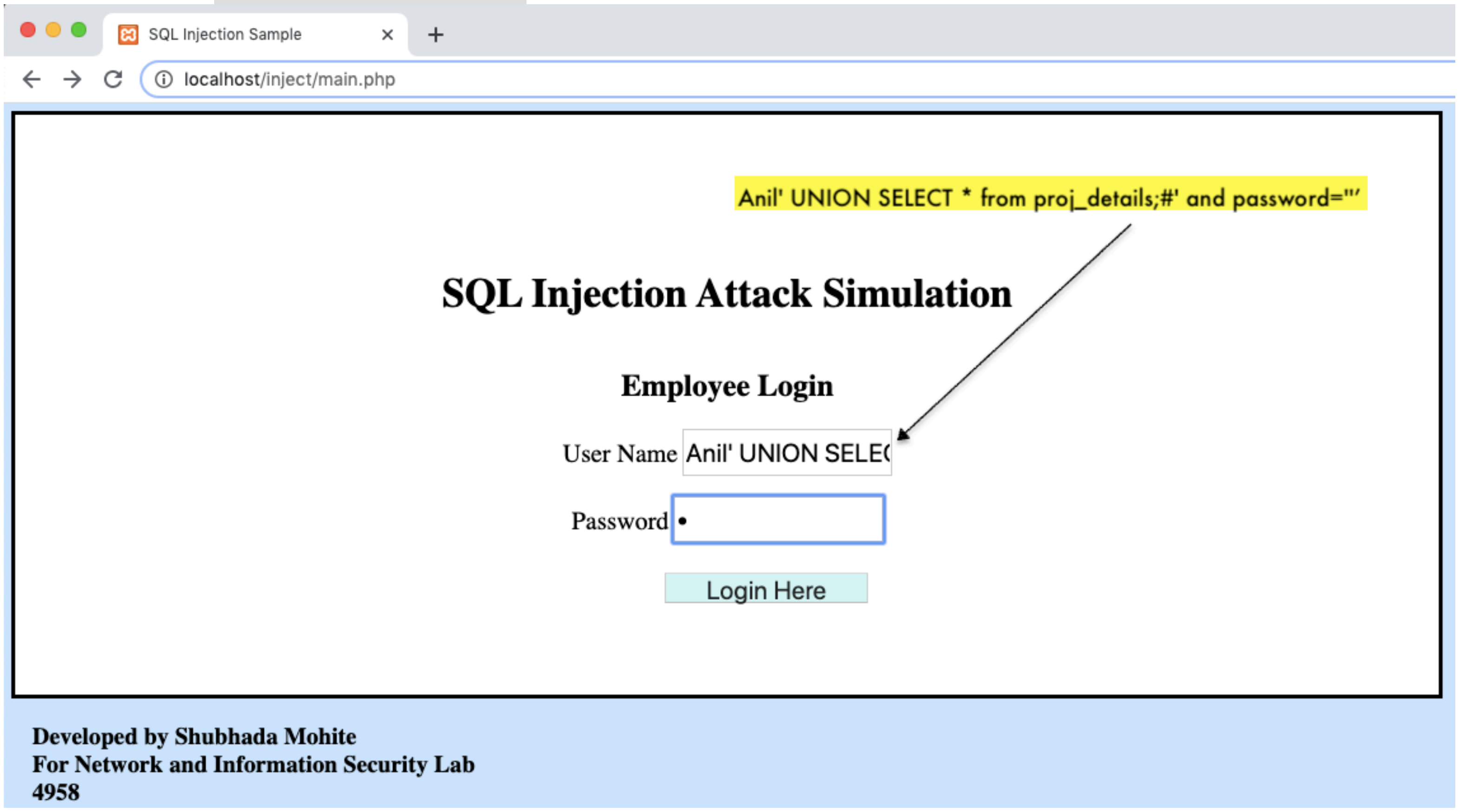
This can be used to drop other tables and change the schema of the tables as well.

The semicolon helps to separate two fields/statements and thus after execution of the first part of the separator any queries after it are executed which result in making changes to the schema of the table. Primarily make changes to the tables without having the appropriate permissions.

<u>3) Union based injections</u>- can be used to return records from other tables in the database.

Anil an employee at the firm wants to see the projects the firm is currently working on which is confidential information known by the admin, he doesn't have to know the credentials to see the projects:

select * from `emp_details` where name='Anil' UNION SELECT * from proj_details;#' and password= ''



SQL Injection Sample

localhost/inject/main.php

Anil' UNION SELECT * from proj_details;#' and password=''

SQL Injection Attack Simulation

Employee Login

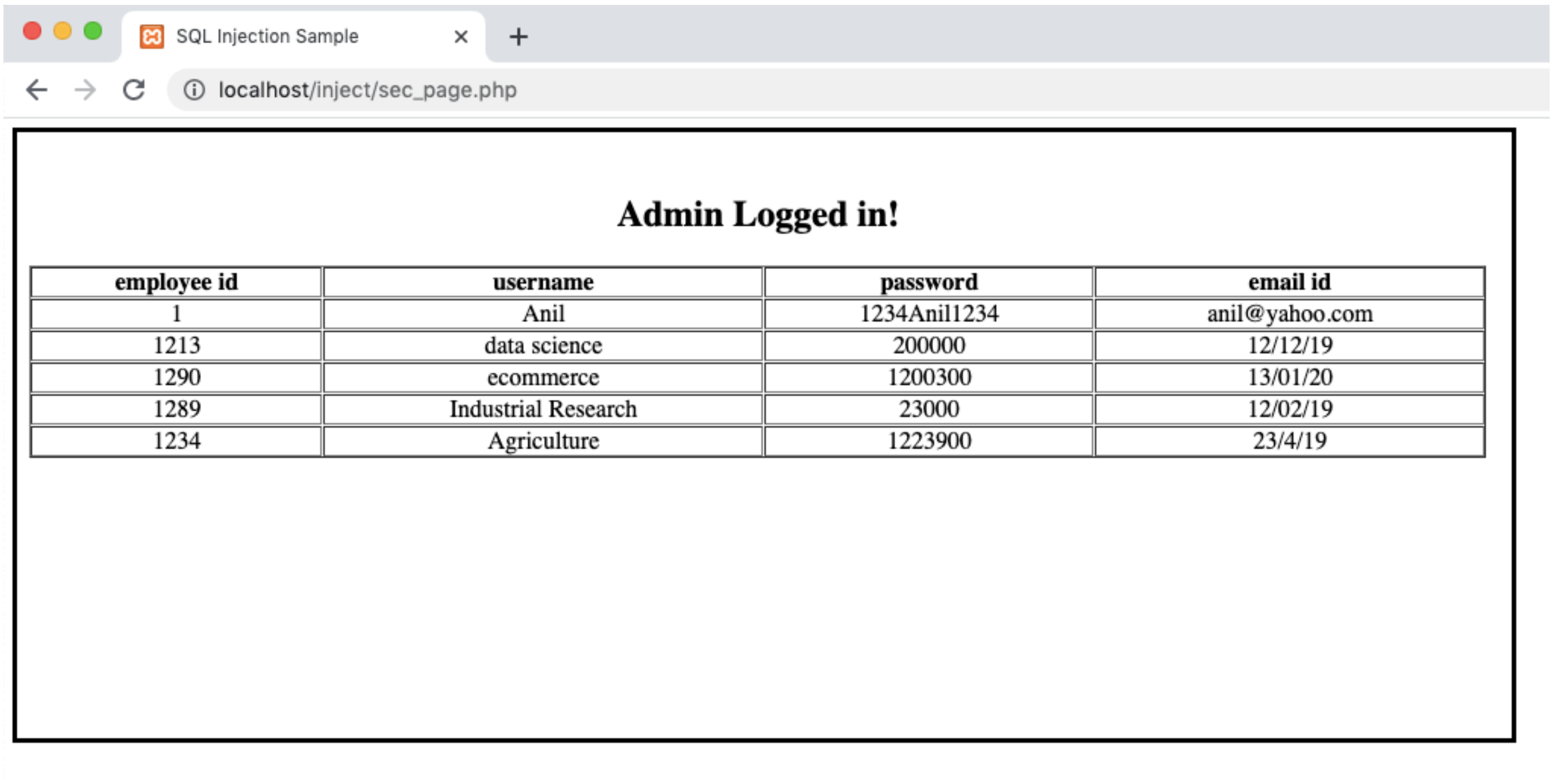
User Name Anil' UNION SELEC

Password •

Developed by Shubhada Mohite
For Network and Information Security Lab
4958

Figure: SQL Injection Attack Scenario 2

This results in a table shown below containing all the records from proj_details along with Anil's details:


SQL Injection Sample

localhost/inject/sec_page.php

Admin Logged in!

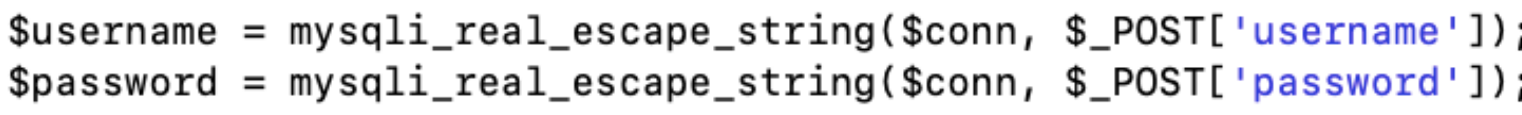
employee id	username	password	email id
1	Anil	1234Anil1234	anil@yahoo.com
1213	data science	200000	12/12/19
1290	ecommerce	1200300	13/01/20
1289	Industrial Research	23000	12/02/19
1234	Agriculture	1223900	23/4/19

Figure: SQL Injection Attack Result 2

E) Demonstration of preventing an SQL injection Attack

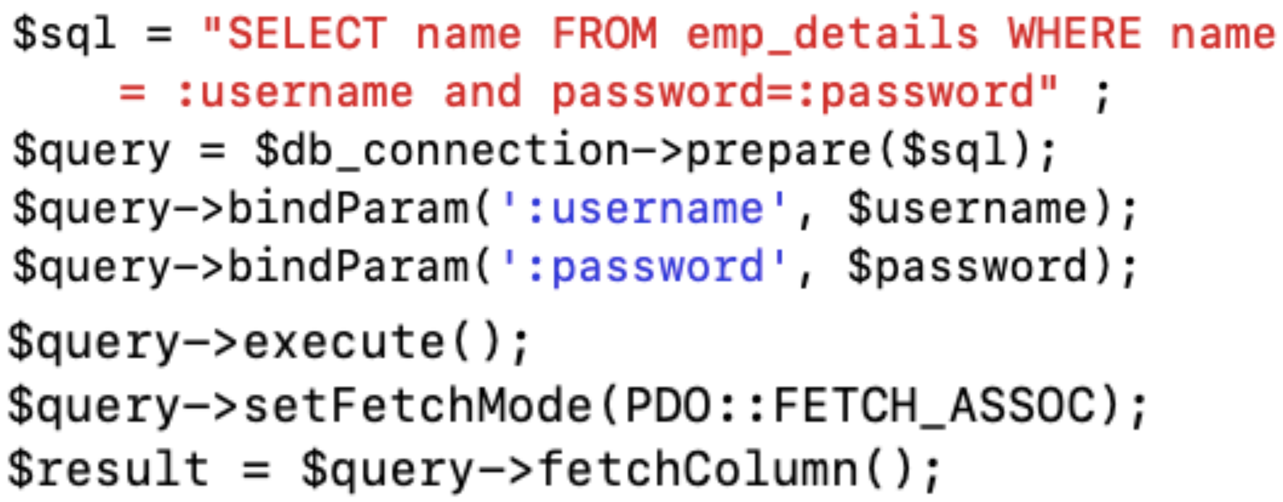
1) Using mysqli_real_escape_string

This function helps escape special characters in a string for use in SQL queries, thus provides a means of sanitising the user input before executing the sql query.


```
$username = mysqli_real_escape_string($conn, $_POST['username']);
$password = mysqli_real_escape_string($conn, $_POST['password']);
```

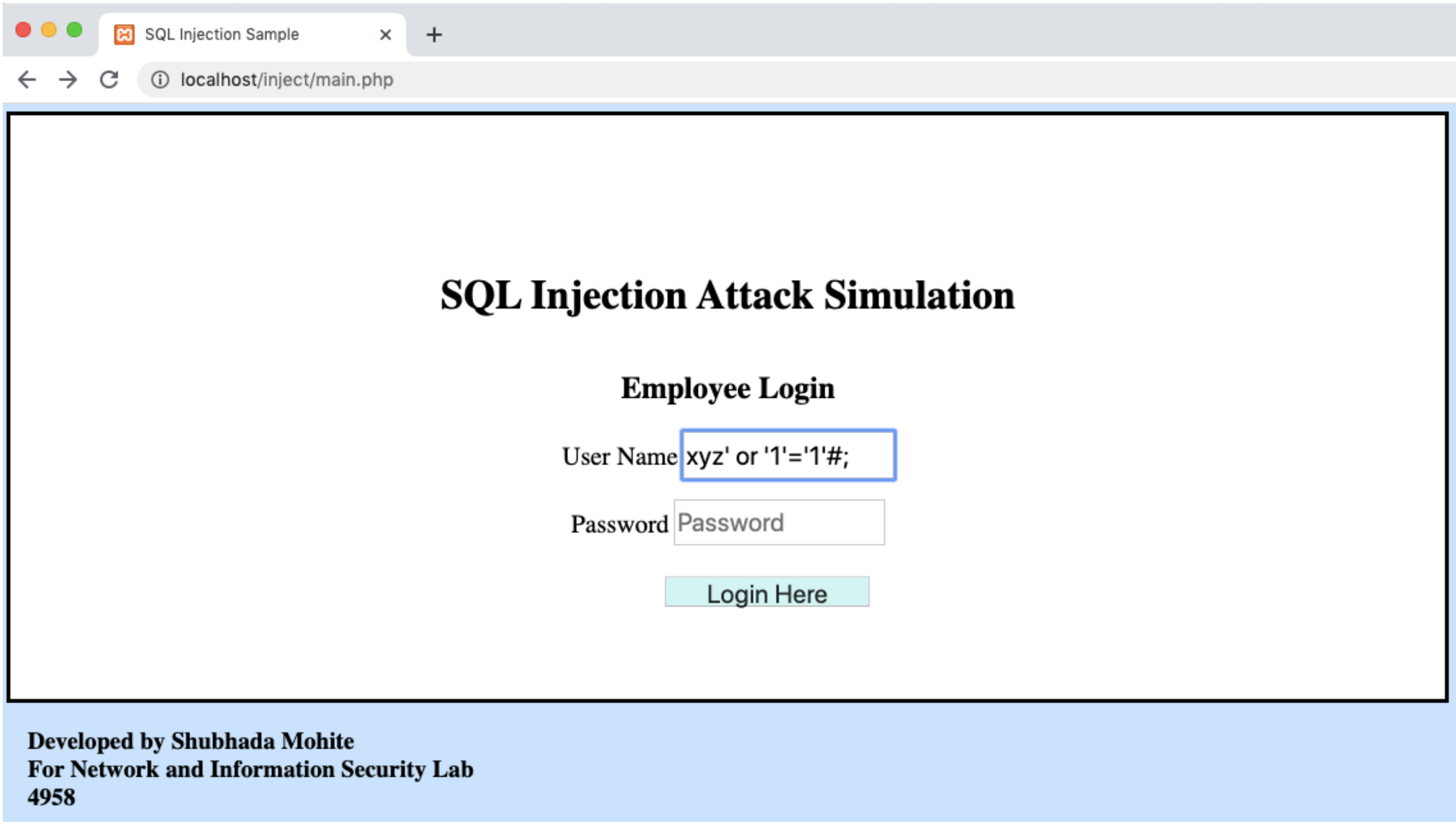
2) Using Parametrized queries:

Provides a way of pre-compiling the SQL query beforehand so that on supplying user inputs to this query it will substitute them and proceed for execution. This makes it possible for the database to distinguish input data from the actual code. A concept called as php Data Objects/ PDO provides a method for achieving this. This also uses PDOStatement::bindParam() allowing to bind a parameter to the specified variable name.



```
$sql = "SELECT name FROM emp_details WHERE name
    = :username and password=:password" ;
$query = $db_connection->prepare($sql);
$query->bindParam(':username', $username);
$query->bindParam(':password', $password);
$query->execute();
$query->setFetchMode(PDO::FETCH_ASSOC);
$result = $query->fetchColumn();
```

✻ After the prevention logic was added, making an attempt of any attack like this:



SQL Injection Attack Simulation

Employee Login

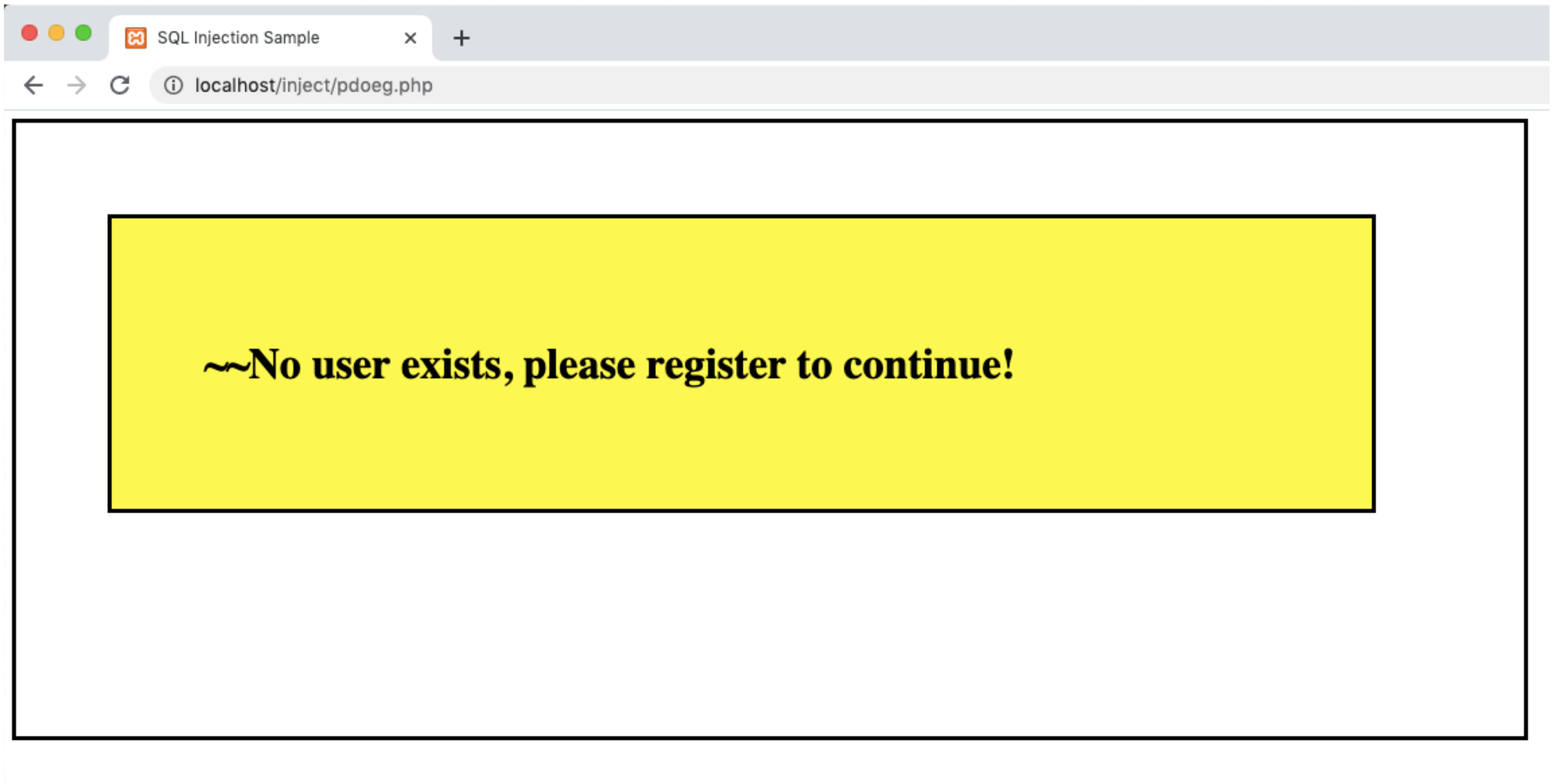
User Name xyz' or '1'='1'#;

Password Password

Developed by Shubhada Mohite
For Network and Information Security Lab
4958

Figure: SQL Injection Attack Scenario 3 After Prevention Logic was Applied

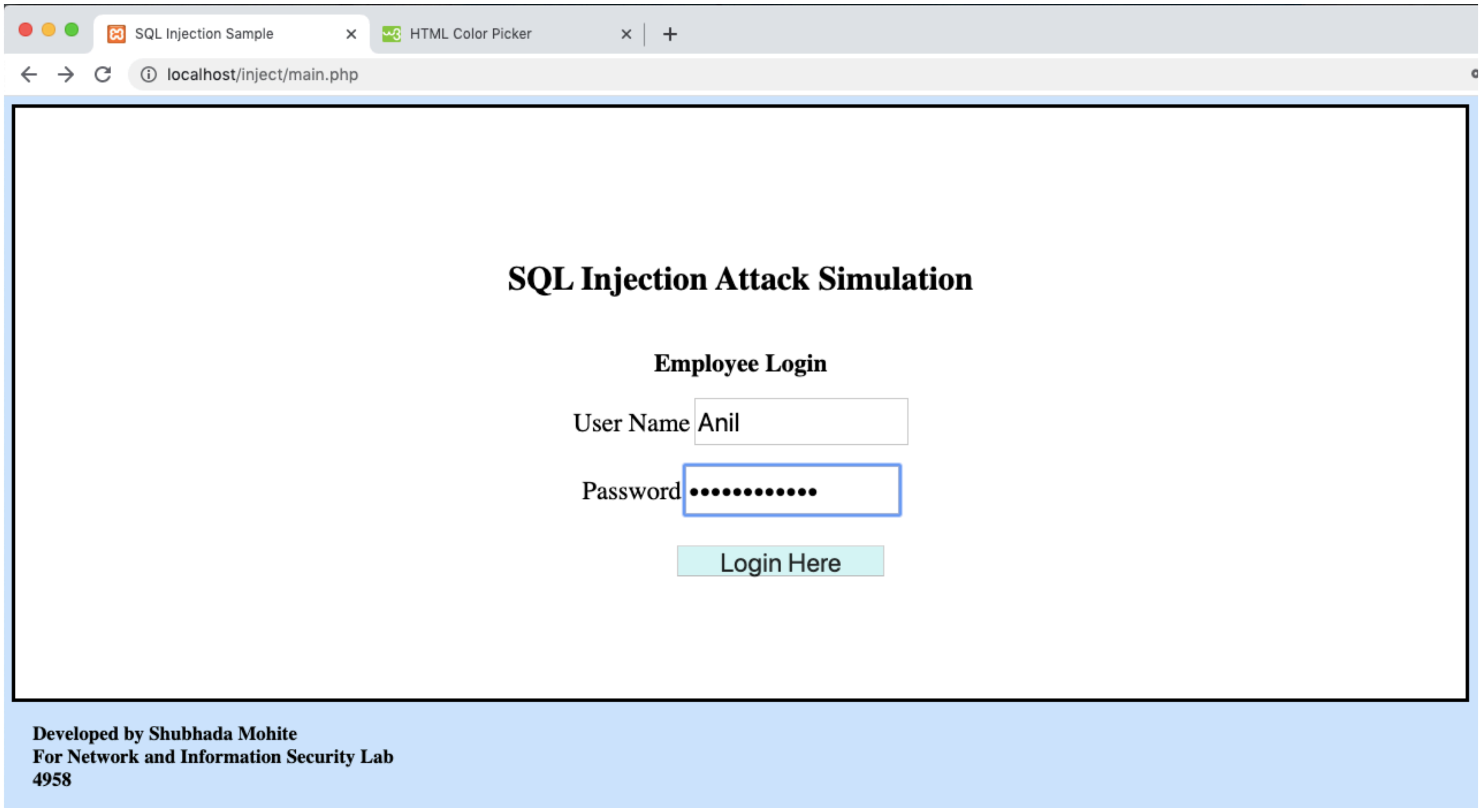
✻ Login page can’t be bypassed as shown in the Figure below:



~~No user exists, please register to continue!

Figure: SQL Injection Attack Result 3 After Prevention Logic was Applied

✻ Trying to login as an employee after prevention logic was applied to the code:



SQL Injection Attack Simulation

Employee Login

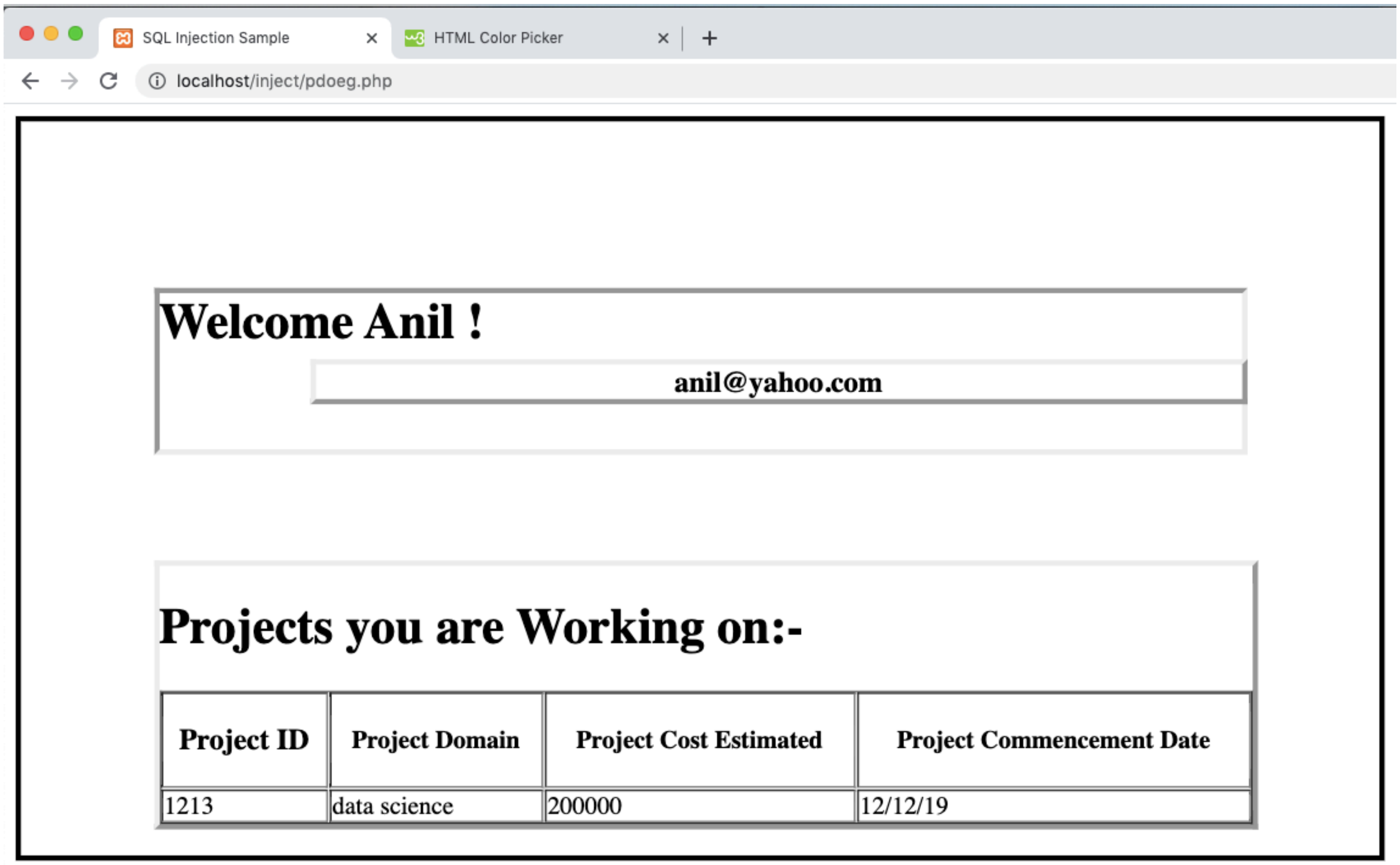
User Name Anil

Password •••••••••••••

Developed by Shubhada Mohite
For Network and Information Security Lab
4958

Figure: Ideal Login Scenario by Legitimate User Anil

Since PDO was used to mitigate sql injection attempts, Anil can see his own projects post login as he is a legitimate user of the system, as shown in the figure below:-



Welcome Anil !

anil@yahoo.com

Projects you are Working on:-

Project ID	Project Domain	Project Cost Estimated	Project Commencement Date
1213	data science	200000	12/12/19

Figure: Ideal Scenario after Legitimate User Anil was allowed to log-in, he can now view his projects.