

Lab 2: SQL Injection

Objective and Set Up:

The goal of this lab is to help you understand how you can discover, attack, and mitigate vulnerabilities through SQL injection.

You will be using a Vocareum Lab environment, which allows you to perform actions securely, safely, and directly through the course portal.

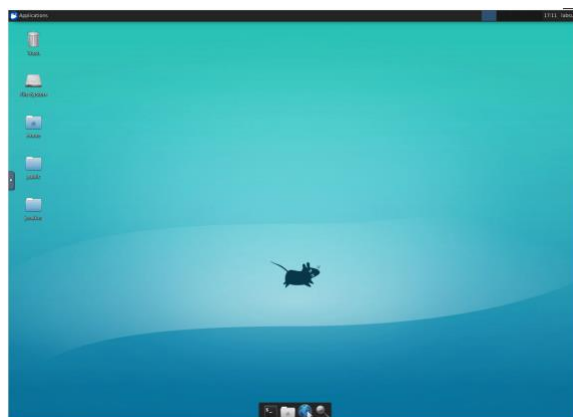
NOTES:

- *This lab is a follow-up on lab #1, so much of the set-up and introduction will be the same. You can proceed directly to Part 1 if you successfully completed the first lab.*
- *We will be using a simulated case study of a fictional bank called BankCorp.*
- *In this exercise, you don't need to develop any new code. The code and solutions are provided in this instruction file - you should be able to complete the lab even with minimal coding experience.*
- *This assignment is optional and is not graded, but it is highly recommended you complete it.*
- *Please note that this lab is for educational purposes only. As a student enrolled in Stanford's Advanced Cybersecurity Program, we trust that you will use the information from this lab for the greater benefit of cyber security.*

Environment:

Once you click on the lab, an interactive “computer desktop” with a blue background will appear. **Please wait 10 seconds after you launch the application for the database to initialize.**

(If the “computer desktop” doesn't appear automatically, you might need to hit “Connect” in the noVNC pop-up application window. Please, note that it might take a few seconds to start the environment depending on your internet connection.)



There should be 4 items in your dock at the bottom of your lab desktop:

- **Terminal emulator** - It emulates a terminal. You don't need to use the terminal for purposes of this lab.
- **File Manager** - It emulates a finder or a file manager on your computer. You can use it for browsing and viewing files. You don't need to use the file manager for purposes of this lab.
- **Web Browser** - You can browse the internet as you would do on your own device. You will use the web browser to navigate to the website of the fictional bank BankCorp, to test its functionalities, and perform a simulated attack.
- **Application Finder** - The important applications are already in your dock, but if you are unable to locate them, you can use the Application Finder to find and launch applications.

In addition, there are 5 icons on your desktop. These are included to simulate a computer desktop and can help in case you would like to search any additional applications or files. If you follow the lab instructions carefully, you won't need to use them.

- Trash
- File system
- "Home" of your file manager
- "Public folder" of your file manager
- "Jenkins" of your file manager

Introduction

BankCorp is a fictional bank. You have been hired as a whitehat hacker to find web vulnerabilities in a BankCorp's customer login website which is used by customers to handle personal banking related transactions. The bank is offering a reward to the whitehat hacker who finds the most vulnerabilities and can identify the magnitude that they can be exploited in their platform. We have outlined the steps below to help guide you to find web vulnerabilities in BankCorp and understand the potential risks associated with it.

Part 1 - Discovery

In the first lab, you helped BankCorp identify vulnerabilities in cross-site scripting. Now, you are wondering if BankCorp sanitizes their inputs to avoid SQL injections and other attacks.


You log back in the BankCorp's application. This application allows you to see how much money you have in your BankCorp account. For this demo let's assume that our username is janedoe. To login to it, please:

1. Open the web browser from your lab desktop (double-click on the icon of the globe at the bottom of the screen; if you are unable to locate the web browser, search "Web Browser" in the application finder).
2. Enter **localhost:4444** in the search box. This will open the fictional BankCorp website.



3. Log in using your fictional username and password
 - **Username:** janedoe
 - **Password:** abcd1234

As a result, you should be able to see how much money you have in your BankCorp account and the query that was executed



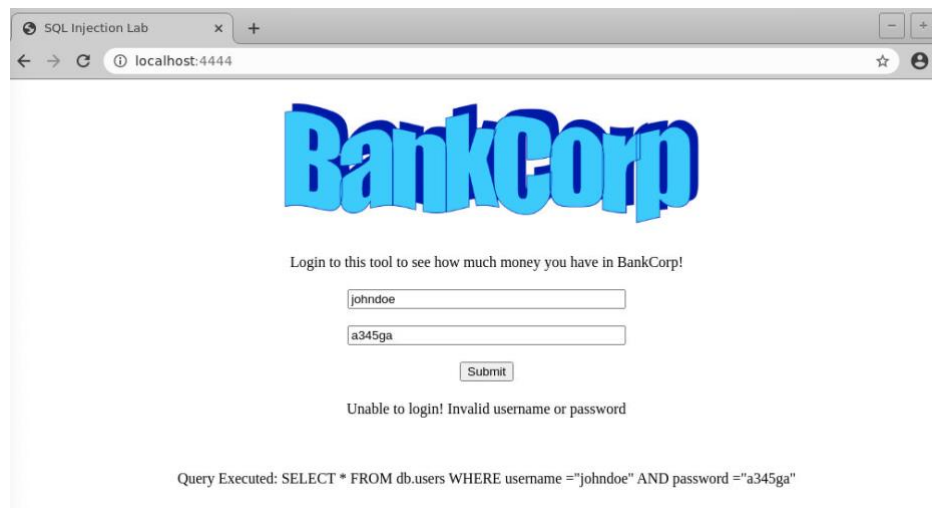
The image shows a web application interface for BankCorp. At the top is a large blue 3D logo that says "BankCorp". Below the logo is a text prompt: "Login to this tool to see how much money you have in BankCorp!". There are two input fields: the first contains "janedoe" and the second contains "abcd1234". A "Submit" button is located below the password field. Below the button, the text "Username: janedoe Password: abcd1234 Funds: 1,000" is displayed. At the bottom, a line of text shows the executed SQL query: "Query Executed: SELECT * FROM newdb.users WHERE username = 'janedoe' AND password = 'abcd1234'" data-bbox="211 166 642 380"/>

Now let's try to log in as another user

4. Enter these fields and click submit

- **Username:** johndoe
- **Password:** a345ga

The response should show that you are unable to log in as the username or password is incorrect



The image shows the same BankCorp login interface as before, but with the inputs "johndoe" and "a345ga". Below the "Submit" button, a new message appears: "Unable to login! Invalid username or password". The SQL query at the bottom is now: "Query Executed: SELECT * FROM db.users WHERE username = 'johndoe' AND password = 'a345ga'" data-bbox="209 546 779 786"/>

The query shows that BankCorp's website accepts the username and password that the user inputs. From the query, we are however not sure if BankCorp sanitizes their inputs to conform to security requirements. Let's discover if they actually do sanitize their inputs by not passing in SQL specific characters

5. Enter these fields and click submit

- **Username:** =
- **Password:** =

As a result, your query should look like this:

```
Query Executed: SELECT * FROM newdb.users WHERE username ="=" AND password ="="
```

We are seeing that we are able to inject untrusted commands into the query and modify it, which could result in an SQL injection attack by malicious actors. Clearly BankCorp is not sanitizing their inputs! Let's see how attackers could use this fact to perform an SQL injection attack.

Part 2 - Attack

Now that we know that BankCorp does not sanitize their inputs, we can manipulate the inputs in the username and password to fetch all users in the application and view their funds.

1. Instead of logging in as a user, enter these fields and click submit

- Username: " or ""="
- Password: " or ""="

As a result, you should be able to see all of the users in BankCorp and the funds associated with each account.



Login to this tool to see how much money you have in BankCorp!

```
Username: janedoe Password: abcd1234 Funds: 1,000
Username: johndoe Password: xyz987 Funds: 2,364
Username: kevindoe Password: 334c3s Funds: 1,000,000
```

We are able to see the usernames and passwords and perform this action, because the condition at the end of the **WHERE** clause will always be executed as **true**

```
username = "" or ""="" AND password = "" or ""=""
```

This simple action showed us how vulnerable BankCorp's website is. Now that we have the attention from BankCorp, let's give them the best recommendations to prevent an SQL injection attack.

Part 3 - Mitigation

With the current architecture, the value from the input box gets immediately sent to the BankCorp server hosted on localhost:4444. The BankCorp server then performs the query without sanitizing the inputs to provide the information shown in Part 2. To mitigate this attack, we need to clean and scrub user inputs to prevent it from exploiting security holes. This means that we need to use prepared statements with parameterized queries, and escape all user input.

Mitigation Step #1: Prepared Statements

A prepared statement allows you to define the SQL statement first, then pass in the parameters later. This prevents a sql injection because sql commands passed in as parameters are read as actual variables instead of sql commands. To include this mitigation, we would change this statement in the BankCorp server from

```
db.execute(`SELECT * FROM ${DB_CONFIG.database}.users WHERE username  
=${username}" AND password ="${password}"` )
```

to

```
db.execute(`SELECT * FROM ${DB_CONFIG.database}.users WHERE username =@0 AND  
password =@1, [username, password])
```

Mitigation Step #2: Input Sanitization & Whitelisting

If we set requirements on the username or password to only contain letters a - z, A - Z, and numbers 0 - 9, we can add logic to the BankCorp server to prevent the server from accepting any characters besides those listed. This can be performed using the Regex statement below

```
var regex = [^A-Za-z0-9]+  
var isValidUsername = regex.exec(usernameInputFromUser);  
if (!isValidUsername) print("This user has entered a character that is not allowed")
```

Part 4 - Conclusion

Thank you for following along with us during this lab. This only scratches the surface of the dangers with injection attacks, and we hope that you will carefully follow other aspects of this issue to secure your web applications from other possible future attacks.