Lab 3: SQL Injections

Thangamuthu Balaji

I. Introduction

Lab 3 foundational concepts of SQL injection and its impact on web application security. and the use of various security and development tools. The main objectives included understanding the basics of SQL injection, installing and configuring tools like WebGoat, and testing vulnerabilities through real-world web application security scenarios. This report outlines the steps taken during the lab, the tools used, and the knowledge gained, with a specific focus on performing SQL injection attacks and identifying security flaws in web applications.

II. SQL INTRO

In this task, I used a SQL query to get specified data from the given database. The purpose was to determine the department of employee "Bob Franco" from the workers table.

To do this, I created a SELECT query using the pattern SELECT department FROM workers WHERE first_name = 'Bob' AND last_name = 'Franco'. The query successfully returned the department, illustrating how SQL commands can be used to find and alter data in relational databases.

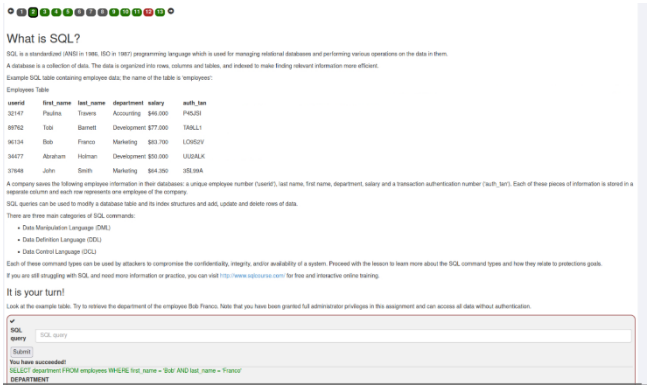


Fig. 1. SQL INTRO

III. STRING SQL injection

In this task, I used a String SQL Injection to retrieve all user data from the user_data table without requiring unique user credentials. The vulnerable query dynamically generated the last_name condition depending on user input. I took advantage of this by entering Smith as the final name, along with the condition 1 = 1, which always returns true.

Because 1 = 1 is always true, the query returned all records in the user_data table, including sensitive data such user IDs, credit card numbers, and login information. This attack emphasizes the dangers of poor input validation and shows how SQL injection may be leveraged to disclose sensitive data by leveraging string concatenation flaws.

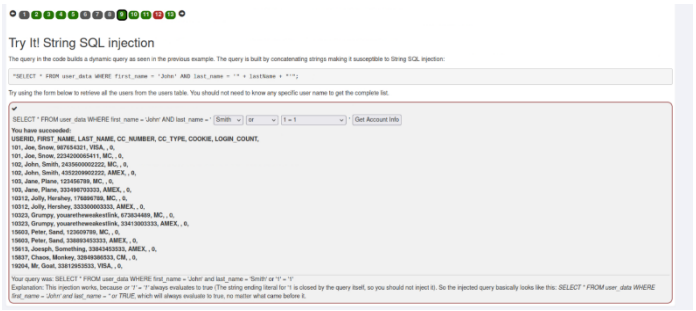


Fig. 2. STRING SQL Injection

IV. Numeric SQL Injection

In this experiment, I used a numeric SQL injection vulnerability in the WebGoat application. The dangerous query was: "SELECT * FROM user_data WHERE login_count = " + Login_Count + " AND userid = " + User_ID;". By entering the number 5 AND userid = user OR 1=1 in the User_Id column, I was able to circumvent authentication and access all entries from the user_data table. The criterion 1=1 is always true, therefore the query returns the full dataset rather than just one individual.

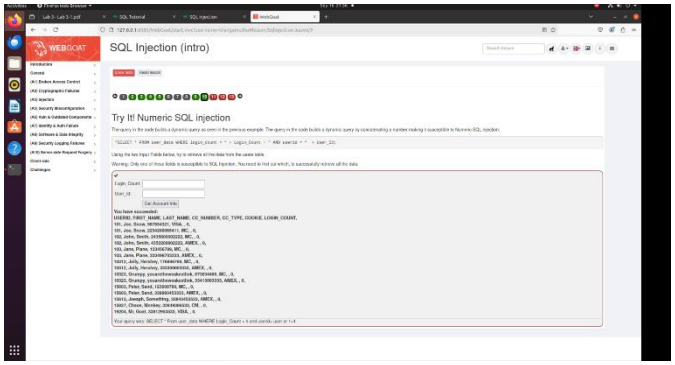


Fig. 4. Numeric SQL injection

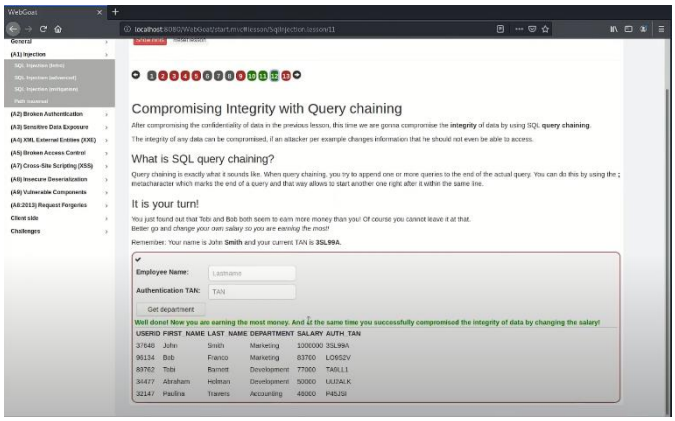


Fig. 5. Integrity Affect

V. Integrity Affect

In this example, I modified the system's "John Smith" salary data using SQL query chaining. I was able to raise my pay to the maximum amount and beat out Bob and Tobi by adding a second query using the ; character after the first one. This type of SQL injection occurs when there is insufficient input validation, leading to the execution of several queries. The example shows how chaining numerous searches in a single input might allow attackers to undermine the integrity of a database.

VI. Confidentiality Affect

I used a string-based SQL injection vulnerability in this work to get access to private employee data. I was able to extract the full employee information, complete with names, departments, and salaries, by entering a wildcard character (*) in the Employee Name field and an arbitrary number in the TAN field. Because this exploit made it possible for me to examine internal information that should have been limited, it jeopardized the system's confidentiality and brought attention to the dangers of using unclean string inputs in SQL queries.

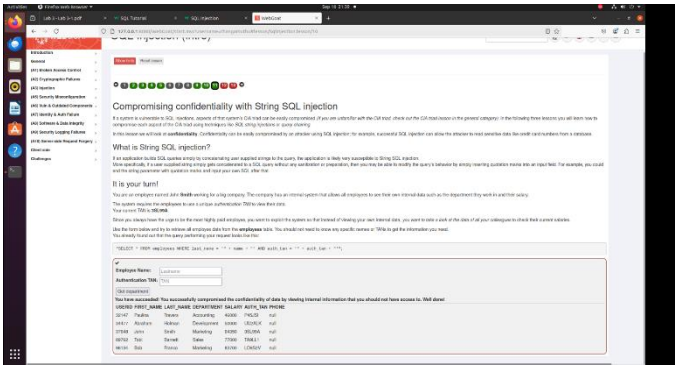
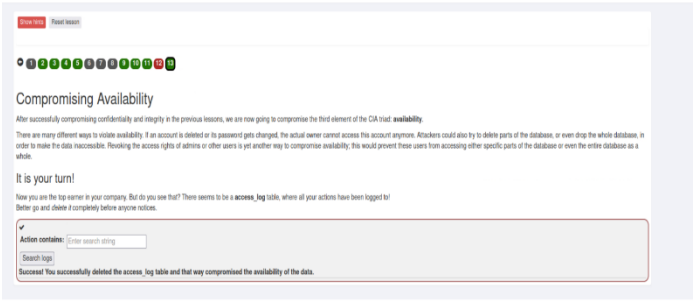


Fig. 6. Confidentiality SQL injection

VII. AVALIABLITY Affect



Deleting the access log table successfully compromised the availability of the data. This action prevented anyone from accessing or reviewing the records of actions performed, potentially hindering accountability and the ability to track and investigate any unauthorized activities.

Conclusion:

My understanding of web application security has improved as a result of finishing the WebGoat SQL Injection class. It gave me the opportunity to investigate in real life how hackers use SQL vulnerabilities to access databases without authorization. I discovered ways to stop these kinds of attacks by finding vulnerabilities in SQL queries. These methods included employing parameterized queries and doing appropriate input validation. Through practical experience, I was able to identify and address vulnerabilities in real-world apps, which strengthened my foundation in safe coding and application protection.

References

[1] l https://www.w3schools.com/sql/default.aspURL Encoding: https://www.w3schools.com/tags/ref urlencode. ASP
[2] https://www.w3schools.com/sql/sql_injection.asp
[3] FoxyProxy: https: //addons.mozilla.org/en-US/firefox/addon/ foxyproxy- standard