Lab 4.pdf

Sonal Chandrakant Hasbi

I. Introduction

SQL injection is one of the most common and dangerous web application vulnerabilities. This lab provided hands-on experience with exploiting SQL injection flaws using intentionally vulnerable applications hosted on WebGoat. The goal was to understand advanced injection techniques, including UNION-based and blind injections, and reflect on countermeasures like parameterized statements and input sanitization.

II. Tools Used

- WebGoat v2023.8 and WebGoat 7.1: Platforms for practicing injection vulnerabilities.
- Linux VM: Hosting the environment for offensive security practice.

III. Methodology

A. Part I: Advanced SQL Injection on WebGoat v2023.8

I navigated to `(A3) Injection > SQL Injection (Advanced)` in WebGoat. Below is a sample SQL injection used to retrieve data (Fig. 1) :

Quesry : SELECT * FROM user_data WHERE last_ name = UNION SELECT userid,user_name, password, cookie, nuil as f1, null as f2, null as f3 FROM user_system_data;-'

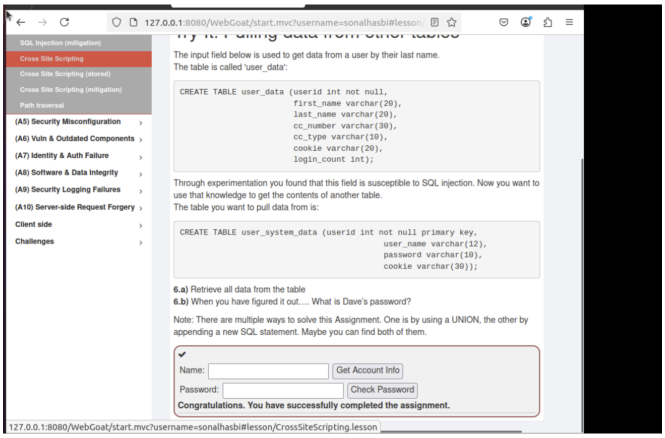

Fig. 1: Advanced SQL Injection

B. Part II: Blind SQL Injection on WebGoat 7.1

Blind Numeric SQL Injection: In this section, numeric payloads were used to guess database content, I tried multiple numbers like >3000, <2000 to first figure out the range of the number and then narrowed it down by trying various combinations to finally get the account number as '2364' (Fig. 2) :

Query : 101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444')=2364)

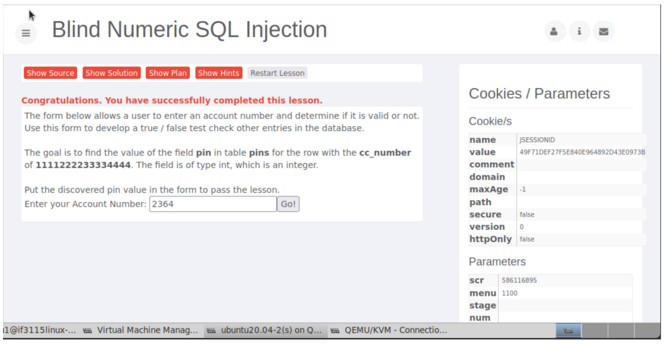

Fig. 2: Blind Numeric SQL Injection

Blind String SQL Injection: The same logic applied to string-based conditions where I tried multiple alphabets to realize that the valid account number is 'Jill' (Fig. 3) :

Quesry : 101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='1111222233334444'), 1, 1) = 'J');

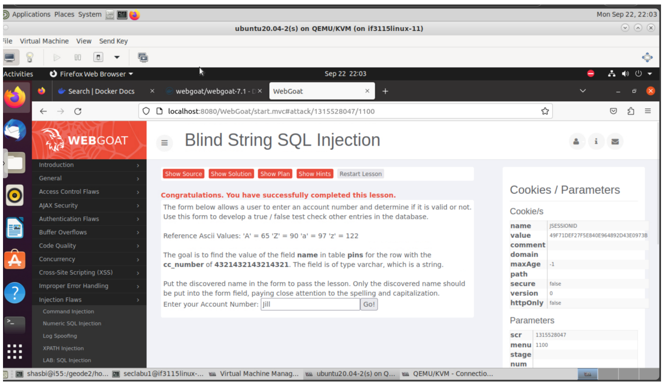

Fig. 3: Blind String SQL Injection

IV. Findings and Discussion

During the execution of the lab exercises, several key findings emerged:

- SQL Query Accuracy: The constructed SQL queries returned accurate results, indicating a correct understanding of SELECT, WHERE, JOIN, and UNION clauses.
- Security Insight: An important takeaway was the potential for SQL injection if user input is not properly sanitized. This reinforces the importance of using prepared statements and parameterized queries in real-world applications.
- Data Relationships: Exploring JOIN operations between tables helped illustrate how relational databases maintain normalized structures and the importance of primary/foreign key relationships.

- Error Debugging: Mistakes such as mismatched column types in JOIN or syntax errors in UNION queries provided learning moments on how to interpret MySQL error messages.

Overall, the lab exercise provided a practical understanding of database querying fundamentals and highlighted best practices for security and data integrity.

V. Conclusion

This lab demonstrated how attackers exploit both traditional and blind SQL injection vulnerabilities. It emphasized the critical need for using prepared statements, validating input, and never trusting client-side data. Understanding how these attacks work deepens our ability to defend systems effectively.

VI. References

[1] W3Schools SQL UNION: https://www.w3schools.com/sql/sql_union.asp

[2] OWASP Parameterization: https://owasp.org/www-community/Query_Parameterization_Cheat_Sheet

[3] Prepared Statements: https://en.wikipedia.org/wiki/Prepared_statement

[4] SQL Injection Prevention in PHP: https://stackoverflow.com/questions/60174/how-can-i-prevent-sql-injection-in-php

[5] Minted in LaTeX: https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted