Lab 3: SQL Injection

Shivali Mate

I. Introduction

THIS lab focused on exploring SQL injection vulnerabilities using WebGoat v2023.8 and WebGoat 7.1. SQL injection remains one of the most common and severe web application vulnerabilities, allowing attackers to manipulate queries executed by a database. Using poorly sanitized inputs, malicious payloads can alter the intended query logic and expose or modify sensitive information. Through a series of exercises, this lab demonstrated the fundamentals of SQL injection, including string and numeric injection techniques, as well as database backdoors, thereby highlighting real-world security challenges faced in industry and academia.

II. Methodology

The lab was divided into two major parts. First, SQL Injection (intro) was performed using WebGoat v2023.8, which had been installed in Lab 1. This provided a foundation for understanding the injection concept in modern versions of WebGoat. The environment was executed with the command:

sudo java -Dfile.encoding=UTF-8 -Dwebgoat.port=8080 -Dwebwolf.port=9090 -jar webgoat-2023.4.jar

Subsequently, WebGoat 7.1 was installed via Docker [2]. The reason for this installation was that WebGoat 8 (v2023.8) is a revolutionary but still developing version with limited exercises. WebGoat 7.1, being stable, provided additional lessons to strengthen training on SQL injection vulnerabilities. After setting up both environments, SQL Injection (intro) was first performed in WebGoat 8, followed by the exercises on Numeric SQL Injection, String SQL Injection, and Database Backdoor in WebGoat 7.1.

A. SQL Injection (Intro)

The WebGoat SQL Injection (intro) lesson is designed to demonstrate the fundamentals of injection attacks by exploiting input fields in a deliberately insecure web application. A SQL injection vulnerability arises when unsanitized user input is concatenated directly into SQL queries, allowing attackers to alter the intended logic of the query.

The lesson introduced several concepts, including retrieving hidden information from a database, appending additional queries through query chaining, using comment characters such as to nullify the rest of the original query, modifying existing data entries, and even dropping entire tables to compromise data integrity and availability.

Injection Payload that can be used are:

- ’ closes the intended input
- **or 1=1** ensures the condition is always true.
- **;** ends the statement.
- **–** comments out the remaining query.

A practical example was deleting the access_log table to compromise availability as shown in Fig. 1.

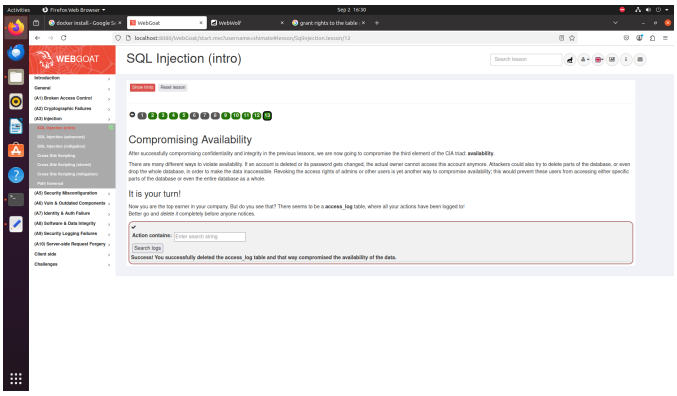

Fig. 1. Introduction to SQL Injection

B. String SQL Injection

String-based SQL injection occurs when user input enclosed in quotes is not properly sanitized. The exercise involved a staff listing page where the goal was to escalate privileges and log in as the administrator “Neville” while being an employee “Larry.”

Using Burp Suite to intercept requests, the following payload was injected into the password field:

’ or ’1’=’1

This successfully bypassed authentication, granting admin access as shown in Fig. 2.

C. Numeric SQL Injection

Numeric SQL injection exploits incorrectly validated numeric fields. In this exercise, after logging in as “Larry,” access to the admin “Neville’s” profile page was attempted. Injection payload used:

0 or 1=1

This forced the query to return true and allowed unauthorized access to the admin profile as shown in Fig. 3 documents the result.

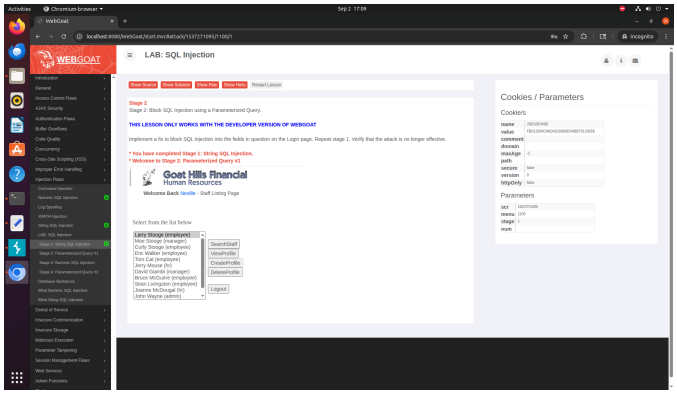

Fig. 2. String SQL Injection

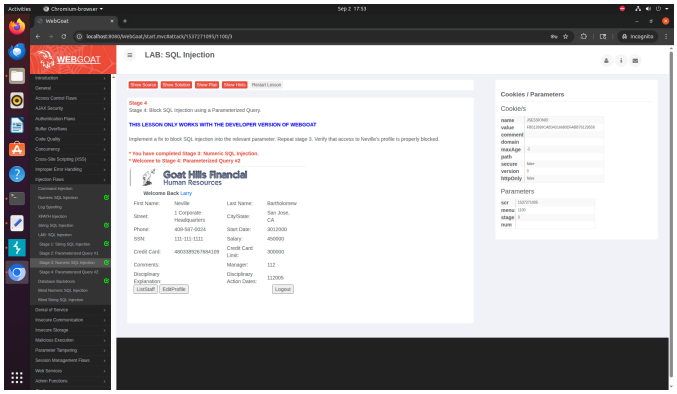

Fig. 3. Numeric SQL Injection

D. Database Backdoor

A database backdoor refers to hidden malicious functionality left within a database that allows unauthorized modifications. The lab exercise demonstrated updating sensitive information, such as user salaries. By exploiting the vulnerability, salaries could be increased, highlighting the risks posed by malicious actors who gain backdoor access, as demonstrated in Fig. 4

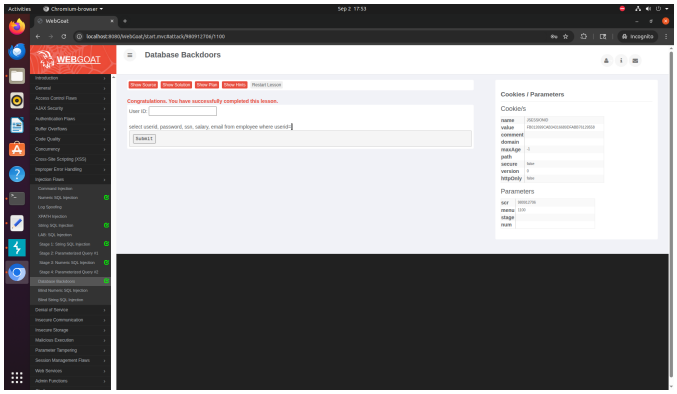

Fig. 4. Database Backdoor

III. Tools and Learning Experience

WebGoat 8 (v2023.8) provided modern, guided exercises enriched with detailed background information, but its limited coverage required supplementing with WebGoat 7.1, a stable release offering practical vulnerability exploitation scenarios. Burp Suite was instrumental in intercepting and manipulating HTTP requests, enabling injection payload testing.

While WebGoat 8's strength lies in comprehensive tutorials, its instability in covering all scenarios was a disadvantage. In contrast, WebGoat 7.1 provided reliable exercises but lacked extensive explanations. Combining both versions proved advantageous.

IV. Findings and Discussion

This lab deepened the understanding of SQL injection, one of the most critical OWASP Top 10 vulnerabilities. It illustrated how improper input sanitization leads to data exposure, privilege escalation, and even destructive database operations. Such knowledge is directly applicable in industry for secure software design, penetration testing, and auditing, while academically it highlights fundamental principles of secure system development.

V. Conclusion

This lab successfully demonstrated SQL injection techniques through exercises in both WebGoat v2023.8 and WebGoat 7.1. From basic payloads to backdoor exploitation, the lab reinforced the importance of input validation and query parameterization in preventing injection attacks. It also highlighted the role of training tools like WebGoat and interception proxies like Burp Suite in advancing cybersecurity education.

References

[1] SQL Tutorial - W3Schools https://www.w3schools.com/sql/default.asp
[2] WebGoat 7.1 – Docker Hub https://hub.docker.com/r/webgoat/webgoat-7.1/
[3] SQL Injection - W3Schools https://www.w3schools.com/sql/sql_injection.asp
[4] SQL Injection Overview - Microsoft Docs https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953(v=sql.105)
[5] WebGoat Wiki - GitHub https://github.com/WebGoat/WebGoat/wiki
[6] Docker Installation Guide - Docker Docs https://docs.docker.com/engine/install/ubuntu/