Lab 3.pdf

Sonal Chandrakant Hasbi

I. Introduction

The purpose of this lab was to gain practical experience with SQL injection vulnerabilities using WebGoat v2023.8 and WebGoat 7.1. SQL injection (SQLi) is a critical web application vulnerability that occurs when untrusted input is directly concatenated into SQL queries without proper sanitization. By completing both numeric and string injection challenges, as well as exploring stages of SQLi exploitation, I learned how attackers manipulate queries to bypass authentication, retrieve hidden data, and establish persistence through backdoors.

II. Methodology and Key Steps

Task 1: WebGoat v2023.8 – SQL Injection (Intro)

1) Launched WebGoat with the following command:
2) sudo java -Dfile.encoding=UTF-8 -Dwebgoat.port=8080 -Dwebwolf.port=9090 -jar webgoat-2023.4.jar
3) Navigated to the "Injection → SQL Injection (Intro)" section.
4) We had Multiple tasks to complete :
   - First, We had a basic task of understanding how SQL works by retrieving the department of employee Bob Franco from the Employee table (Fig. 1)

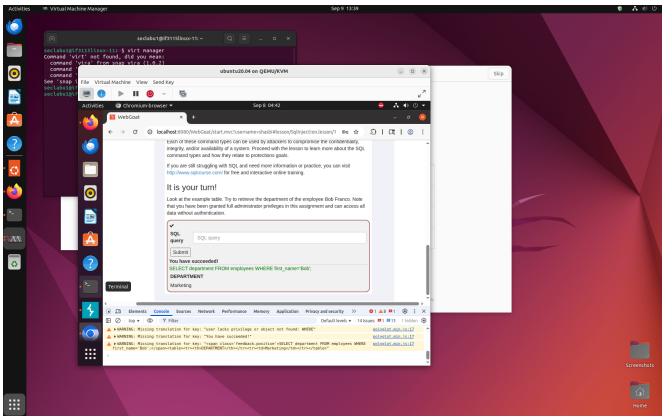

Fig. 1: Task 1-SQL Injection Intro

- Second task was to Manipulate data from the table Using Data Manipulation language (DML) (Fig. 2) .
- Third task was to understand Data Definition Language modify the schema by adding the column "phone"(varchar(20)) to the "employees" table (Fig. 3) .
- Fourth task was to understand Data control language to implement access control in a database by granting rights to "grant_rights" table to the user "unauthorized_user"
- Fifth task was to try string SQL injection by retrieving information of all users from the "users" table without having to know any specific username to get complete access of the list.(Fig. 5).
- Sixth task was to understand Numeric SQL injection by concatenating a number to build a dynamic query. (Fig. 6) .
- Seventh task was to compromise integrity by query chaining- the task given was to change the salary of the employee in the main database(Fig. 7) I did so by writing the command : Smith';UPDATE employees SET salary=99999 WHERE last_name='Smith'–
  And the TAN value was given as 3SL99A.
- Eighth task was compromising Confidentiality through string SQL injection : saw internal info that we should not have access to employee name: 0 OR

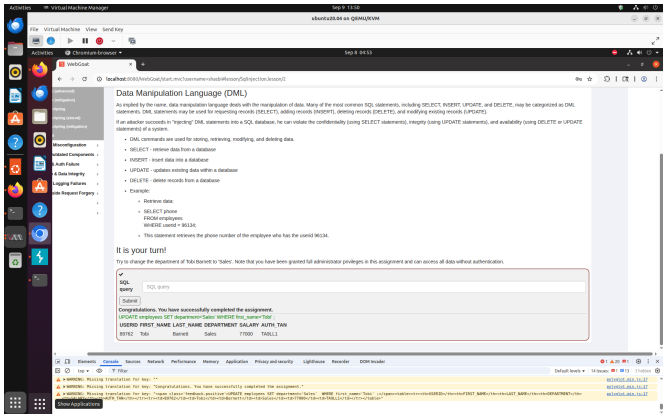

Fig. 2: Task 2-SQL Injection Intro

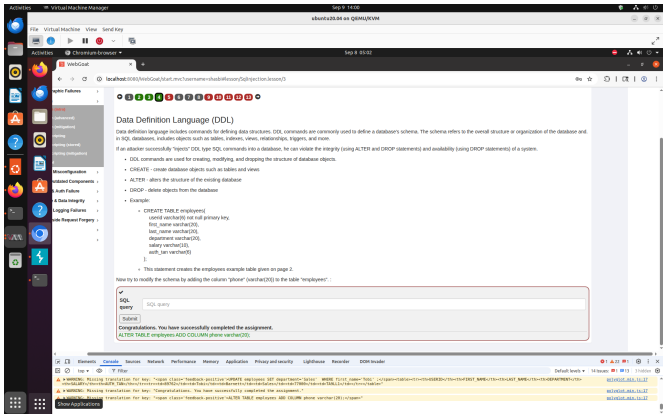

Fig. 3: Task 3-SQL Injection Intro

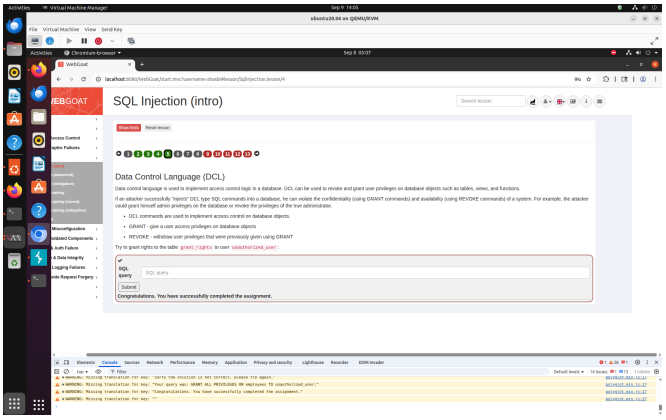

Fig. 4: Task 4-SQL Injection Intro

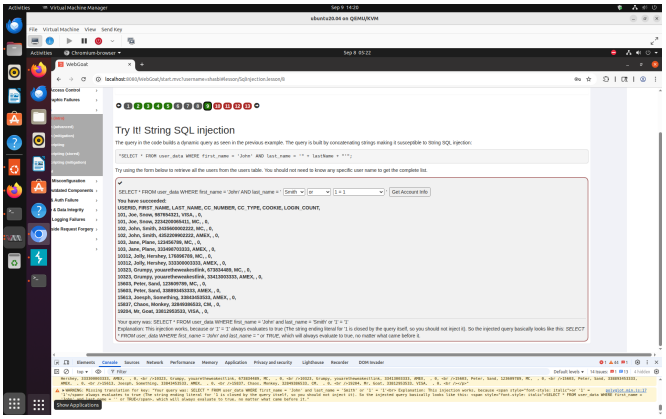

Fig. 5: Task 5-SQL Injection Intro

’1’=’1’–

- ninth task was to erase the access log table so that theres no proof of access logged (Fig. 8). I did so by inputting the command: " SELECT * FROM access_log WHERE action LIKE % ’ ; DROP TABLE access_log;– "

5) Observed how these queries altered the underlying SQL execution, confirming injection vulnerabilities.

Task 2: Installing WebGoat 7.1

1) Pulled the Docker image:
   - docker pull webgoat/webgoat-7.1
   - docker run -p 8081:8080 webgoat/webgoat-7.1
2) Accessed the application via http://localhost:8081/WebGoat.
3) Verified successful setup by logging into the platform.

Task 3: WebGoat 7.1 Injection Flaws

A deliberately insecure application used to demonstrate common web vulnerabilities. I completed the following modules:

- Numeric SQL Injection
  - Input: 101 OR 1=1
  - Result: Successfully extracted the weather data from the data base using SQL Injection. (Fig. 9), proving numeric injection.

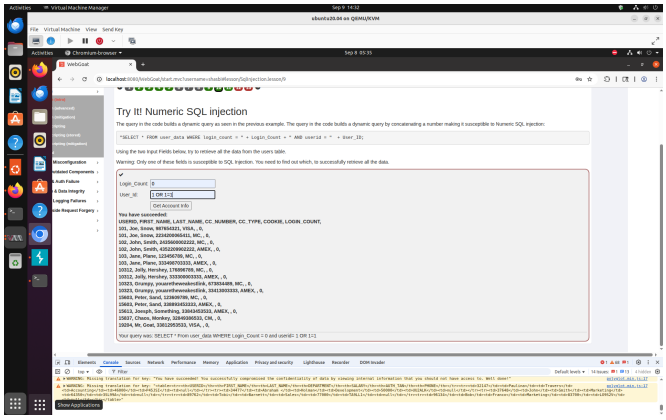

Fig. 6: Task 6-SQL Injection Intro

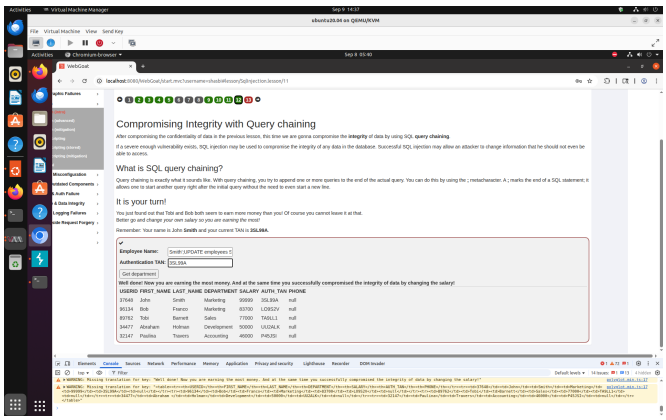

Fig. 7: Task 7-SQL Injection Intro

- String SQL Injection
  - Input: Last_name=’Smith’
  - Result: Successfully displayed the cerdit card information of John Smith, proving string SQL injection.(Fig. 10)
- SQL Injection Lab (Stage 1 and Stage 3)
  - Stage 1: Injected 101 OR 1=1 to enumerate tables.
  - Stage 3: Numeric SQL injection - by loggin in using SQL Injection as Larry and then extracting Naval profile by using the command: employee_id=101 or 1=1 order by employee_id dec
- Database Backdoor (Stage 1)
  - Created a backdoor user by executing an injection that inserted a new admin account into the database by using the command: 101;UPDATE employee SET salary=9999 .
  - Confirmed persistence by logging in with the injected credentials. (Fig. 11)
- Database Backdoor (Stage 2)
  - Understood how to use a Vulnerable field to inject the DB work or Backdoor: 101;CREATE TRIGGER myBackDoor BEFORE INSERT ON employee FOR EACH ROW BEGIN UPDATE employee SET email="john@hackme.com" WHERE userid= NEW.userid

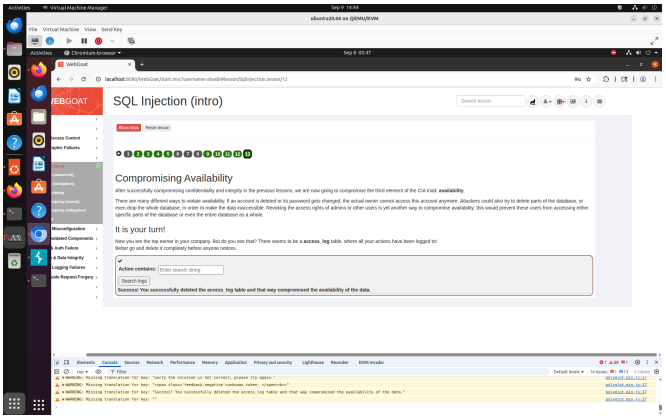

Fig. 8: Task 9-SQL Injection Intro

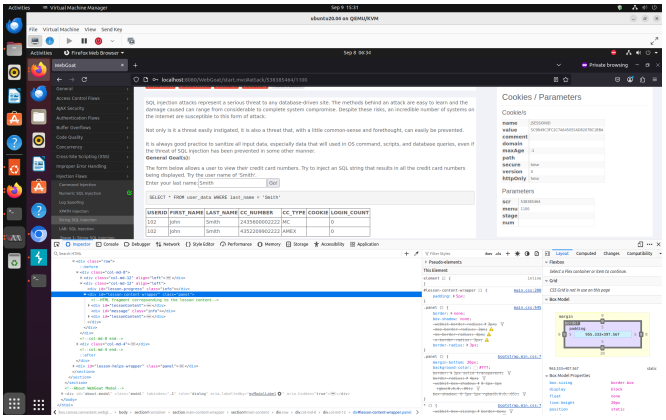

Fig. 10: String SQL Injection

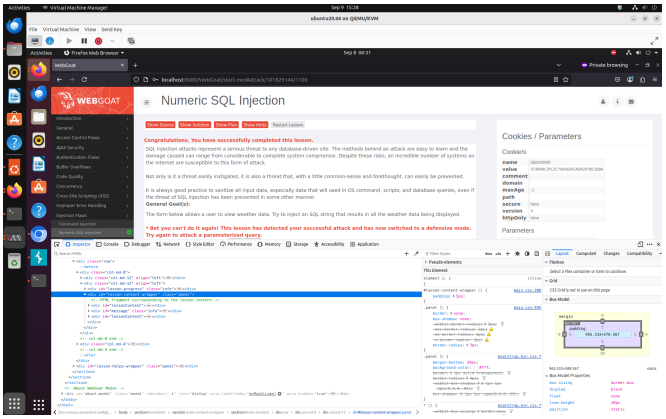

Fig. 9: Numeric SQL Injection

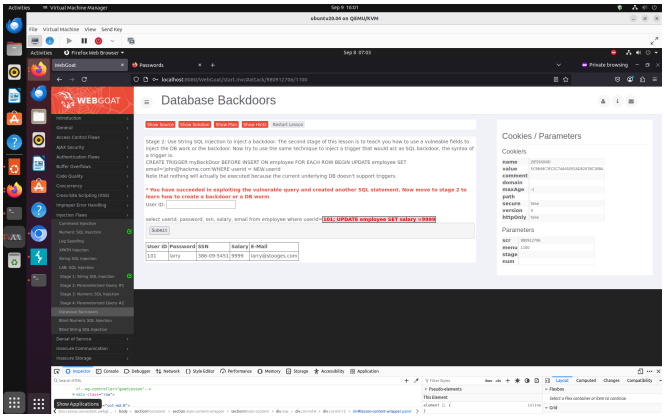

Fig. 11: Database Backdoor (Stage 1)

Tools and Knowledge Learned

1) WebGoat v2023.8 and 7.1: Platforms for simulating real-world vulnerabilities in a safe environment.
2) SQL Injection techniques: Numeric, string-based, UNION SELECT, and persistence (backdoors).
3) Docker: Simplified deployment of vulnerable applications.
4) Key concepts: Importance of parameterized queries, prepared statements, and least-privilege access.

Challenges and Improvements

1) Some payloads required multiple trials due to filtering mechanisms.
2) WebGoat 8 lacked certain lessons, reinforcing the importance of maintaining older stable versions for full coverage.
3) In real-world systems, web application firewalls (WAFs) and query sanitization must be combined for layered defense.

Conclusion

This lab reinforced my understanding of SQL injection attacks by allowing me to exploit vulnerable queries in both modern (WebGoat 8) and stable (WebGoat 7.1) environments. I gained practical skills in constructing injection payloads, identifying risks, and understanding the remediation strategies organizations must adopt. The exercises highlighted the criticality of secure coding practices such as prepared statements, input validation, and least-privilege access.

References

[1] WebGoat Documentation: https://github.com/WebGoat/WebGoat/wiki
[2] W3Schools SQL Tutorial: https://www.w3schools.com/sql/default.asp
[3] Microsoft SQL Injection Overview: https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms161953(v=sql.105)