Lab 4: Advanced SQL Injections

Thangamuthu Balaji

I. INTRODUCTION

LAB 4 dives into advanced SQL injection methods using the WebGoat v2023.8 and WebGoat 7.1 setups. The lab includes tasks like pulling data from various tables via SQL injection and executing blind SQL injection attacks. These activities are designed to strengthen the grasp of how attackers can alter database queries to access unauthorized data and highlight the importance of secure coding techniques to prevent these vulnerabilities.

II. PULLING DATAS FROM OTHER TABLES

In this exercise, I conducted a SQL injection attack to retrieve data from two different database tables using the UNION SQL command. The query used was designed to combine data from multiple sources: ' UNION SELECT userid, username, password, cookie, null as f1, null as f2, null as f3 FROM user_system_data;-- . By exploiting vulnerabilities in input validation, this query allowed me to access sensitive information such as usernames, passwords, and cookies from the user_system_data table. This illustrates how attackers can leverage poorly validated inputs to extract data from multiple tables within a database.

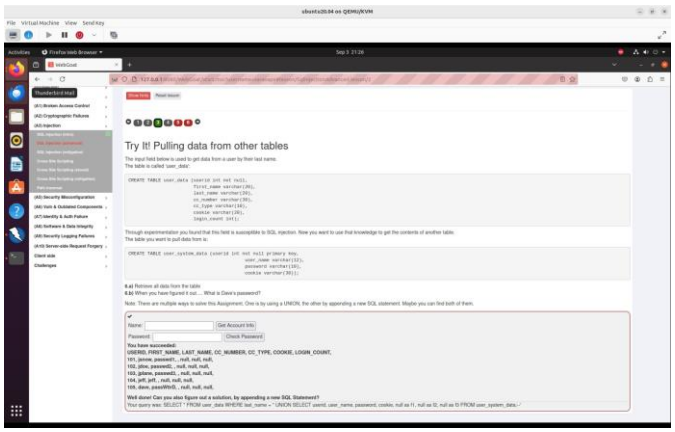


Fig. 1. Pulling Data From Other Tables

III. OPTIONAL - LOGIN AS TOM

I tried to utilize the SQL injection vulnerability in the login form to get in as the user TOM for this optional job. I was able to get around the authentication mechanism and log in as Tom by inserting a malicious string into the username filed, illustrating the dangers associated with unclean input fields. Thisisasecretfor-today was the password.

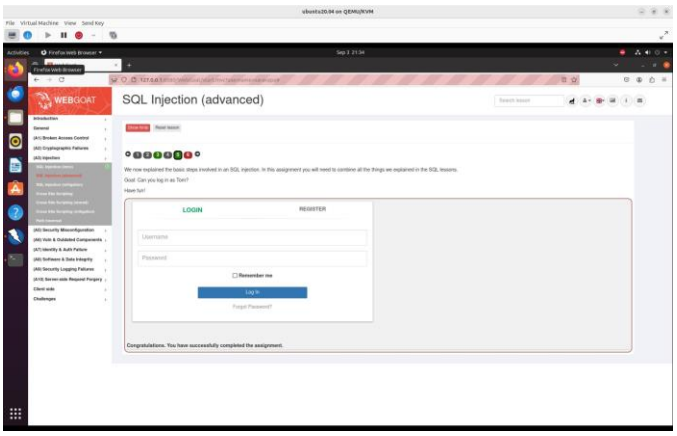


Fig. 2. Login as Tom

IV. BLIND NUMERIC SQL INJECTION

In this task, I tried to figure out what a secret database field was hiding by guessing its value through a series of yes/no questions. The question I went with was: 101 AND ((SELECT pin FROM pins WHERE cc number='1111222233334444') ¿ 10000);The query provided a true or false response based on whether the condition was met, returning either "Account number is valid" or "Invalid account number.

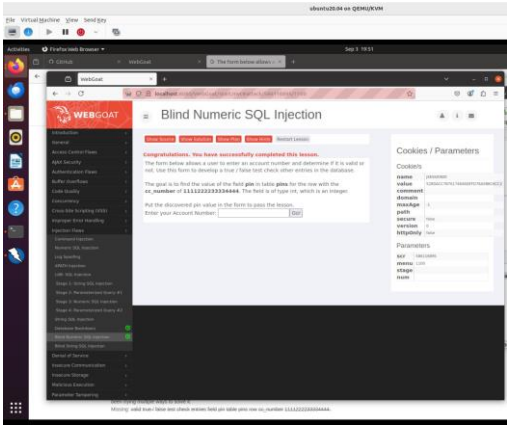
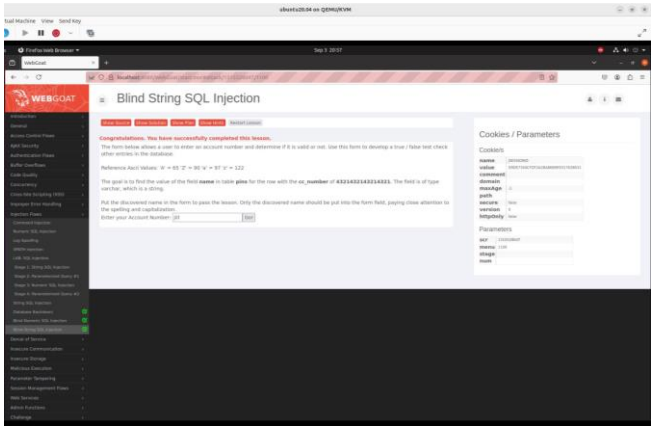


Fig. 3. Blind Numeric SQL Injection

V. BLIND STRING SQL INJECTION

In a way that's comparable to the numerical injection, I employed an anonymous SQL injection to uncover a concealed string value stored in the database.

101 AND (SUBSTRING((SELECT name FROM pins WHERE cc_number='4321432143214321'), 1, 1) ¡= 'N'); was the query I used. Depending on whether the first letter in the name field was less than or equal to "N," this query returned true or false. I was able to figure out that "Jill" was the proper name by altering the query's letter and going through the procedure again. This technique breaks down the query into individual character comparisons to show how an attacker might retrieve string data.



.

Fig. 4. Blind String SQL Injection

VI. CONCLUSION

Lab 4 offered practical training in sophisticated SQL injection methods, concentrating on extracting information from various tables and executing covert SQL injections on both numerical and textual data. These activities showcased the potential of SQL injection to breach both organized and unorganized data, highlighting the necessity of validating inputs and using queries with parameters to safeguard against these weaknesses.

REFERENCES

[1] W3Schools SQL UNION Tutorial -https://www.w3schools.com/sql/sql_union.asp
[2] OWASP Query Parameterization Cheat Sheet
[3] SQL Injection Prevention in PHP (StackOverflow) -https://stackoverflow.com/questions/60174/how-can-i-prevent-sql- injection-in-php
[4] MSDN Parameterized Queries - https://msdn.microsoft.com/en-us/library/cc296201(SQL.90).aspx
[5] Blind SQL Injection Explanation (Acunetix)