Assignment 1: Automatic SQL Injection

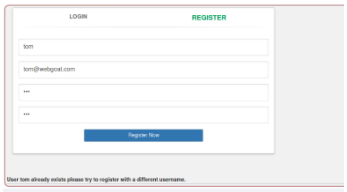
Thangmuthu Balaji

I. Introduction

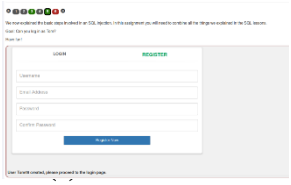
The goal of this lab was to explore various SQL injection techniques by engaging with a vulnerable application and revealing the structure and data of its underlying database. We started by identifying the entry fields that were susceptible on the "Register New User" page, then proceeded to methodically list the table and column names within the database, and finally extracted specific user details, such as usernames and passwords. In the following sections, I will outline each step in detail, including the Python scripts I utilized and the results of each attack.

II. Finding out Vulnerability

Initially, I aimed to find out which input fields in the "Register New User" form were susceptible to SQL injection. To do this, I took a systematic approach by using a Python script that tried to submit different payloads to the various fields in the registration form. By sending specifically designed SQL queries that would return either "true" or "false" responses, I could see how the server reacted to each input. The main objective was to identify which field might be vulnerable based on the application's responses. It turned out that the "Register User" page itself was the weak point. The plan was to analyze the server's responses to figure out the vulnerability: if a payload led to a response saying "Username already exists," it indicated a "true" condition, meaning the input was validated against existing records. Conversely, if the response stated "New username created," it signified a "false" condition, suggesting that the input was treated as unique and the payload had bypassed the checks. By closely monitoring these responses, I could pinpoint which fields were at risk for SQL injection. The screenshots show the user registration process and how I tested the fields for vulnerabilities.

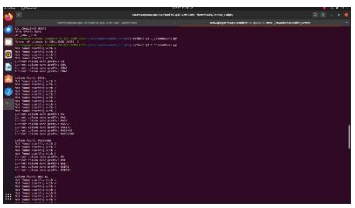


(a) it already exists

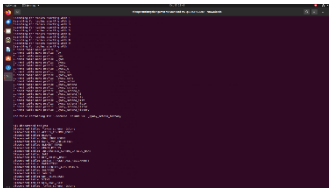


III. Finding the Table names

After I found a weak spot in the system, my next move was to discover the names of all the tables in the database. This step is crucial since having the table names is key to pulling out data. I used a different Python script to make this process automatic. the method of searching the database involved systematically listing out the names of tables. The script operated by making educated guesses about the prefixes of table names and then trying various combinations until it discovered a valid name. This is similar to trying different keys on a keychain until you find the right one—each query aimed to check if a table with a certain prefix was present, and if it was, the script would keep building on that prefix until it uncovered the complete name. From the terminal output displayed in the screenshots, we can see that the script successfully identified tables like ACCESS CONTROL USERS and CHALLENGE USERS.



(a) Finding table names



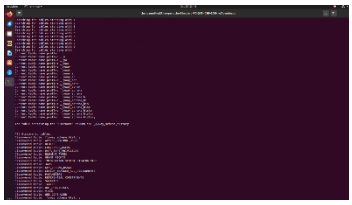
(b) Table names found

IV. Finding Column Names

After figuring out the names of the tables, I started looking into the columns each table had because those columns hold the actual data. My code checked all the tables that included a PASSWORD column. I discovered that the CHALLENGE USERS table had a PASSWORD column. Then, I split the task into two parts: first, I needed to find out how many columns were in a specific table, and second, I had to identify the names of those columns. The script I used to count the columns worked by running queries with different numbers of column placeholders until it found one that worked, which told me how many columns there were. After I got the count, the next script was all about figuring out the names of those columns. Similar to the table search, this script tried various prefixes until it found the right column names. The output in the terminal showed that I successfully identified column names like EMAIL, PASSWORD, and USERID. These columns were really important because they contained credentials and user identifiers that could be misused later.

V. Finding the Usernames

Once I had the table and column details, I proceeded to extract the real data, concentrating mainly on the usernames. I utilized another Python script to compile a list of the usernames that were saved CHALLENGEUSERTABLES



(a) Finding column names

I managed to find the usernames by guessing each character individually until I revealed the complete name. This involved writing SQL queries that would check for specific conditions. When I got a character right, the server would respond accordingly, allowing me to continue to the next character. Through this technique, I was able to discover usernames such as alice, eve, larry, and tom. Each time I made a correct guess, I got closer to uncovering the entire username, which helped me compile a list of users in the target database.

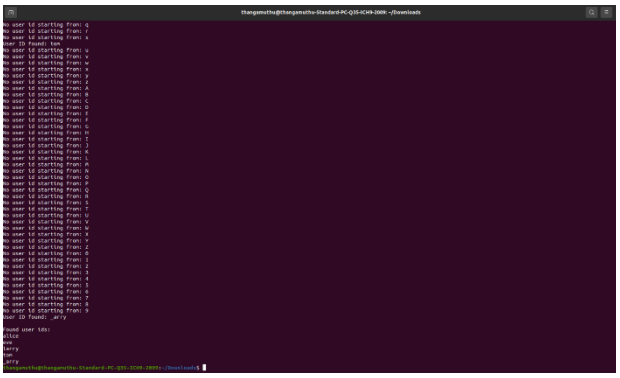


Fig. 4: Finding the Usernames

VI. Finding the Password

Finally, I focused on finding Tom's password. This required an even more detailed approach, as passwords are typically more sensitive than usernames. The Python script for this task used a brute-force technique to reveal Tom's password character by character. Much like the process for enumerating usernames, this script sent SQL queries that tried different possible characters, checking if each character was correct before moving on to the next one. The process began with querying the first character of the password, then the second, and so on, until the entire password was revealed. As seen in the screenshots, the script eventually discovered Tom's password to be thisisasecretfortomonly. This demonstrated how a lack of proper input sanitization and parameterized queries can allow attackers to use brute-force techniques to access highly sensitive information.

VII. Conclusion

During the lab, I was concentrated on learning how attackers take advantage of SQL injection weaknesses to illegally access databases. Each Python script was designed to automate a certain step in the attack process—identifying vulnerable spots

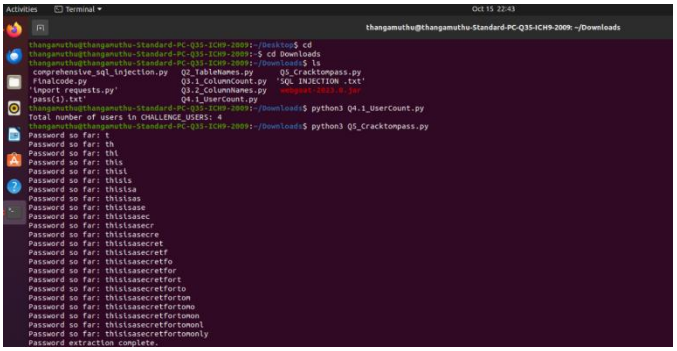


Fig. 5: Password has been Cracked

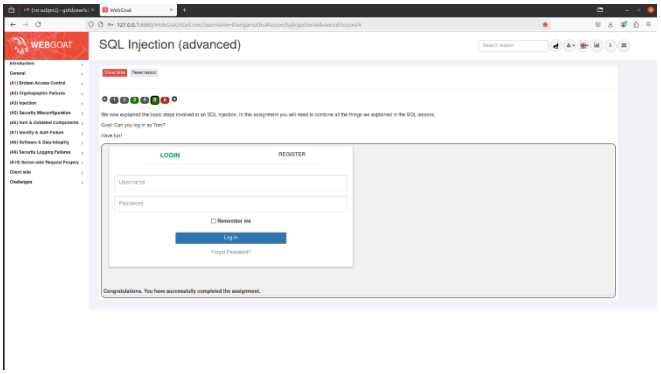


Fig. 6: Completion Webgoat Page

by carefully following each step, I showed how someone could take advantage of a weak application, starting from finding security flaws to using them to get important data. This lab really highlighted how crucial it is to have strong security measures in place, like checking user input and using parameterized queries, to keep web applications safe from SQL injection attacks.