Assignment 1

Venkata Diwakar Reddy Kashireddy

***Abstract*—This report explains how to perform a blind SQL injection using automated python script/tool to extract necessary information from a target database. A blind SQL injection is a technique that attackers use to ask the database true or false questions and determines the answer based on the applications response.**

I. Introduction

In this lab, we completed a automated blind SQL injection using python script/tool on a server to find information in the database. We used tool to find the table names, columns, users and Tom's password. We studied blind SQL injection and executed it on WebGoat 8.2.2.

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. Blind SQL (Asking True or False)

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.

First, I found out which text-box out of both Login and Register page is vulnerable to SQL injection. I found a text box that provided responses — either True or False from database based on the input I provided. For instance, when checking with the "Register" text box, it would indicate whether a user was already registered or not

Secondly, I formed an true or false question for the database and injected it in the user field of the Register page.

```
tom\' and (Select count(table name) from
information schema.tables where table
name like 'A%') > 0;-
```

If the server returned that the user already exists, then it indicates that the statement is also true.

Fig. 1. Server indicating the existence of table with name starting with 'A'.

IV. Automation

In this lab, we were asked to develop an automated tool for blind sql injection using the requests module in Python. I sent queries to the target URL (http://localhost:8080/WebGoat/SqlInjectionAdvanced/challenge) using the requests module from Python, attaching the appropriate parameters and HTTP headers (cookie, which I obtained using developer tools). The headers, payload, submitting the request, and receiving the answer were all handled manually as part of the automated program. Using a for loop, I tried different payload combinations. Depending on the tasks we were expected to complete, the payload differed. In order to try every possible combination, I used a for loop and recursion. When all the following strings (updated strings: after adding another character) returned false, I used a flag to keep track of the string I passed and added the string to the list. I chose the register page as the injection point because it would respond with true or false for the given input from the username field. So, I added an true or false SQL statement to figure out the table names.

V. Table Names

In MySQL, the INFORMATION_SCHEMA.TABLES view allows you to get information about all tables and views within the database. This was included in the SQL command to retrieve all the table names. By default, it will display this information for each and every database table and view. For this particular task, I tried using blind SQL injection to discover all of the tables' names. I used the below sql injection command along with the http payload to attack the field.

```
sql_injection = 'tom\' and (Select count
(table_name) from information_
schema.tables where table_name like \'{}%\'
ESCAPE \'$\')> 0;--'.format(current_prefix)
http_payload = {
    'username_reg': sql_injection,
    'email_reg': 'kvdr@gmail.com',
    'password_reg': '000',
    'confirm_password_reg': '000'
}
```

Since, we are using brute force attack on the application, it ran for a very long time. The below image shows the table names of all the tables in the database.

Fig. 2. Table Names

VI. Column Names

From the table names I gathered, the "SQL_CHALLENGE_USERS" table seemed to be the one containing the user "tom's password. I crafted another automation script aimed at identifying all the column names within that table. To achieve this, I leaned on the "INFORMATION_SCHEMA.COLUMNS" view. This will show the names of all the columns for every table and view in the database. Essentially, it offers a structured overview of the database's schema. I used the below sql command to inject to get the names of the columns.

```
sql_injection = 'tom\' and (Select count
(COLUMN_NAME) from information_schema.
columns where table_name like \'SQL_CHAL
LENGE_USERS\' and column_name like\'{}%\'
ESCAPE \'$\') > 0;--'.format(partial_name)
```

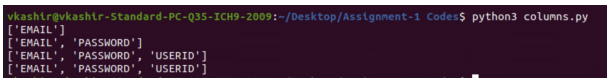

Fig. 3. Column Names

VII. Usernames

From the information I gathered from the column names, it was evident in which the usernames were located.Here, to get all the users from the tables, I executed the below sql command using blind injection to get all the user names from the column. The below shows the usernames present in the table.

```
sql_injection = 'tom\' and (Select count
(USERID) from SQL_CHALLENGE_USERS where
USERID like \'{}%\' ESCAPE \'$\')> 0;--'.
format(prefix)
```

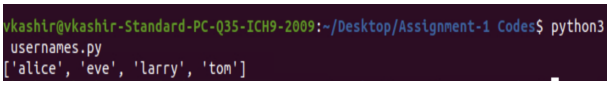

Fig. 4. User Names

VIII. Password

I used the substring function to guess each character of Tom's password from the USERS table. I used brute force on all the character to check if the character in that position is correct or wrong. The program took few minutes to run as we are using brute force with the sql injection command and the end output as below.

```
sql_injection = 'tom\' AND substring(pass
word,{},1)=\'{}'.format(pass_char_index,
all_chars[char_index])
```

Fig. 5. Tom's password

IX. Challenges

I encountered difficulties while trying to use the Python requests package, especially when passing characters in the sql injection command without them being interpreted as regex symbols. I overcame this by using the ESCAPE $ćharacter in the command.

X. Conclusion

In this lab, I successfully developed an automated tool for blind SQL injection using Python's requests module, a technique employed to extract database information by posing true/false questions to an application with hidden error messages and SQL vulnerabilities. I targeted a specific URL, sending queries with proper parameters and headers, automating the entire process. Utilizing for loops and recursion, I adjusted sql injection commands based on tasks, such as retrieving table names, columns names, usernames through true/false responses. The program ran for an extended period while attempting to brute-force table names in the MySQL database. Finally, I got Tom's password based on information gathered before. This lab underlines the significance of securing web applications against vulnerabilities and the ethical use of hacking techniques to detect and address them.

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/
[8] Information Schema
https://www.mssqltips.com/sqlservertutorial/196/
information-schema-tables/.