

Assignment 1

Chetan Sahrudhai Kimidi - ckimidi

Abstract—Exploiting SQL Advanced Injection and finding out Tom's password, using a custom automatic script.

I. INTRODUCTION

In this assignment, I have written an automatic script, which sends HTTP PUT requests to the WebGoat server, and injects into the susceptible username field on the registration page, thereby obtaining various data like table names, columns in a particular table, values of columns such as passwords and such, to automate the process of finding out Tom's password, for the task 5 in SQL Advanced Injection, of WebGoat v2023.4.

II. ASKING THE SERVER TRUE/FALSE QUESTIONS

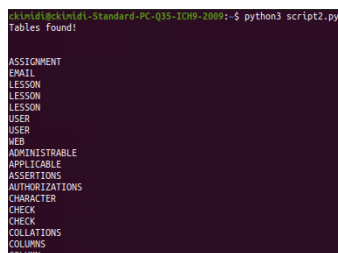
Initially, I gave various inputs in both the login and registration forms, to check the susceptibility of the fields. I found out that, the username field in the registration form, was injectable, since when I tried to sign up/register as some new person, the account was being created, but when I tried to register as username 'tom', using some random email and mobile, it was returning that 'tom' already exists, as seen in [1].

So, in my script, I made use of this susceptibility in a way such that the script is injecting the SQL query into the registration username field, along with filling in other details of the form.

III. ASKING THE SERVER FOR TABLE NAMES

First, to find out the actual number of the tables present in the database, as suggested in the hints, I input a query into the username field, repeatedly. I found out there were 123 tables in the database, as seen in [1] and Fig. 1.. The query was -

'tom' AND (select count(*) from information_schema.tables) = 123; - -



```

C:\Users\ckimidi\Standard-PC-Q35-1000-10000> python3 script2.py
Tables found!
ASSIGNMENT
EMAIL
LESSON
LESSON
LESSON
USER
USER
WEB
ADMINSTRABLE
APPLICABLE
ASSERTIONS
AUTHORIZATIONS
CHARACTER
CHECK
CHECK
COLLATIONS
COLUMN
COLUMN

```

Fig. 1. 123 tables

Then, I realized that I don't need all these tables, but I need only the tables which would have a column called 'password', which was a calculated guess (check section VII), for the

column name. So, in the manual approach, I modified my input into the username field like -

'tom' AND (select count(table_name) from information_schema.columns where column_name like 'PASSWORD') = 6; - -

I found out through the Web about a similar view, called information_schema.columns, which contains all the columns in the database, with their corresponding table names. So, using this information, I found out there were 6 tables with a 'password' column, as shown in the above query and as seen in [1]. I decided to work on those 6 tables, since it was better than to working with 123 tables. Later, I modified the input such that it would show me the true or false case based on letters of the tablename, using the functionality of substring, as follows (here, X is any letter of the alphabet, case-sensitive) -

'tom' AND (substring((select TABLE_NAME from INFORMATION_SCHEMA.COLUMNS where COLUMN_NAME = 'PASSWORD'),1,1) LIKE 'X');- -

IV. AUTOMATING THE APPROACH

I learnt about HTTP PUT requests, how they work, what cookies are, the Python requests module and related things, to write a script, which would repeatedly query the server, using the username field, such that a desired output is achieved. The same script could be used to find out all the tables in the database, or the number of tables which have a 'password' column, or the password of 'tom' and so on, depending upon the query in it.

The basic outline of the script is a nested-loop mechanism, where the outer loop iterates over a defined range, depending on the case, for instance (0,125), when you want to get all the 123 tables in the database etc. The inner loop actually contains the PUT request, in a try-except clause, with a vital if-else condition, which is basically used for the increments of counters, and appending the correct characters which were found until now.

The source code can be found here, [3].

V. FINDING OUT TARGET TABLE AND ITS COLUMNS

After finally writing my script, I resumed where I stopped in my manual approach, which was finding out the letters of names of tables, which contained a 'password' column. The difference was that, I modified my input a bit (automating the letter to be used with the like ' ' operator), and used it in the payload in my script, which is passed in the request data header, so that it can be repeatedly queried in the nested loop. As seen in [2], I found out the table names, which were CHALLENGE_USERS SQL_CHALLENGE_USERS

USER_DATA_TAN
USER_SYSTEM_DATA
WEB_GOAT_USER

Then, to find the target table (the table with a user 'tom' and the respective password), I wanted to check the columns of the obtained tables, so that if I found any columns like 'username', along with password, it might be the target table, if it contains username 'tom'. So, I modified the query in the payload as follows, starting with the first table obtained i.e. CHALLENGE_USERS -

'tom' AND (substring((select COLUMN_NAME from INFORMATION_SCHEMA.COLUMNS where TABLE_NAME = 'CHALLENGE_USERS'),1,1) LIKE 'tom'));

I obtained three columns, which were 'userid', 'email' and 'password', as seen in Fig. 2.

```
ckimidi@ckimidi-Standard-PC-Q35-ICH9-2009:~$ python3 script2.py
USERID
EMAIL
PASSWORD
```

Fig. 2. Columns in CHALLENGE_USERS

VI. USERNAMES IN TARGET TABLE'S USER COLUMN

As CHALLENGE_USERS consisted of the 'userid' and 'password' columns, I wanted to check all the usernames present, to see if Tom was one of them. To do this, I modified the query as follows -

'tom' AND (substring((select USERID from 'CHALLENGE_USERS'),1,1) LIKE 'tom'));

I found out there were 4 users, namely Larry, Tom, Alice and Eve, as seen in Fig. 3.

```
ckimidi@ckimidi-Standard-PC-Q35-ICH9-2009:~$ python3 script2.py
larry
tom
alice
eve
```

Fig. 3. Users in CHALLENGE_USERS

VII. FINDING TOM'S PASSWORD

Initially, I was experimenting with a manual approach, where I tried to find out the length of the password, without knowing any of this table name or columns information. I found out it was 23 characters long [1], by playing with greater than, lesser than conditions between ranges of numbers, from 0 to 100. My query was -

'tom' AND length(password)=23;

My idea was to use the concept of referencing the password locally in the query, since it is a validation, and the password would not need its table name specified. And luckily, it worked which led me to knowing not only that the password is 23

long, but also the target table would have a column called 'password' exactly.

Finally, now with that gathered knowledge and knowing 'tom' was present in CHALLENGE_USERS, and that the same table has a column called 'password', I modified my query as follows -

'tom' AND (substring((select PASSWORD from 'CHALLENGE_USERS' where USERID='tom'),1) LIKE 'thisisasecretfortomonly'));

I found out that the password was 'thisisasecretfortomonly', as seen in Fig. 4 which was 23 characters long. I entered this password in the login form, and the challenge was completed successfully.

```
ckimidi@ckimidi-Standard-PC-Q35-ICH9-2009:~$ python3 script2.py
thisisasecretfortomonly
```

Fig. 4. TOM'S PASSWORD

VIII. DETAILS OF THE APPROACH

Briefly describe how you designed your automatic tool/script, what technical challenges you solved and how your tool/script is able to help you do the hacking in an automatic manner.

I started from the basic script of just a PUT request to the server, to check if I could establish successful contact with the server. Then, I started using a loop in the wrong way, without realizing that I needed one loop for sending the parameters in the request, while one had to repeatedly iterate for the first loop to repeat. After fixing this, I had some issues with figuring out the delimiter syntactical passing into the username field, in the payload part of the data header. Also, I had challenges in figuring out how to setup the conditional statement, based on the response obtained from the request.

My script is pretty much helpful, since it does almost all of the work in determining the output, when I input the right query. **Overall, which part of your hacking is automatic and which part is manual?**

The request-response cycle, figuring out the output and appending it into a data structure, checking for various characters and letters, is all automatic in my script. The output which I want, for instance, the total tables, or tom's password etc. depends on the query I input into the header. So, that part is manual, where I just change the input query, for different use cases. This can be further developed into a tool, where all the steps, sequentially, can feed into each other, and there is no need of manual intervention and the password or any other data can be found totally automated.

Why do you need an automatic tool? Can you do such hacking manually?

It is clearly evident why we need an automatic tool, since it is not humanely and manually possible and extremely time-consuming, to find out and repeatedly type the same query again and again, around 2×26^n times, while the script can cruise through all of that with ease.

IX. CONCLUSION

This concludes the assignment, and shows that I was able to successfully write a script which automates the process of finding out Tom's password, and SQL Advanced Injection, in general.

REFERENCES

- [1] Milestone 1
- [2] Milestone 2
- [3] Source Code
- [4] Blind SQL Injection
- [5] Python Requests