جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

أكاديمية كاوست
KAUST ACADEMY

DAY 3

Web Application Security Lab

Dr. Ali Hassan
Instructional Professor, CEMSE Division,
King Abdullah University of Science and Technology,
Contact: ali.hassan.1@kaust.edu.sa

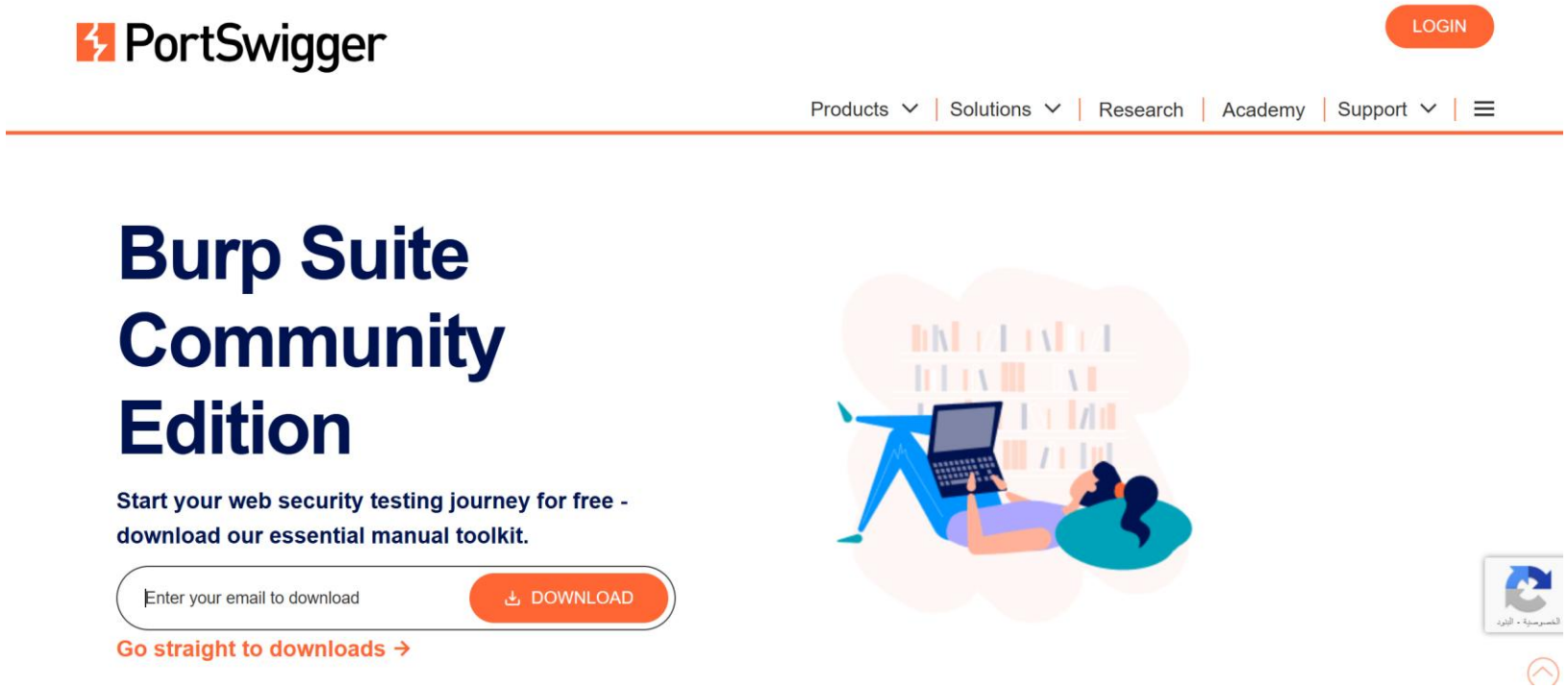
PortSwigger

BURP SUITE

- Burp Suite is a widely used tool for testing the security of web applications.
- It helps security professionals identify vulnerabilities like SQL injection, XSS, and authentication flaws.
- It acts as a proxy between the user and the target website, allowing users to intercept, modify, and analyze HTTP/HTTPS requests and responses for testing.
- It has a free, community edition which we'll be using to solve the labs.
- You can either download Burp Suite from the official website or use the pre-installed version available in Kali Linux.

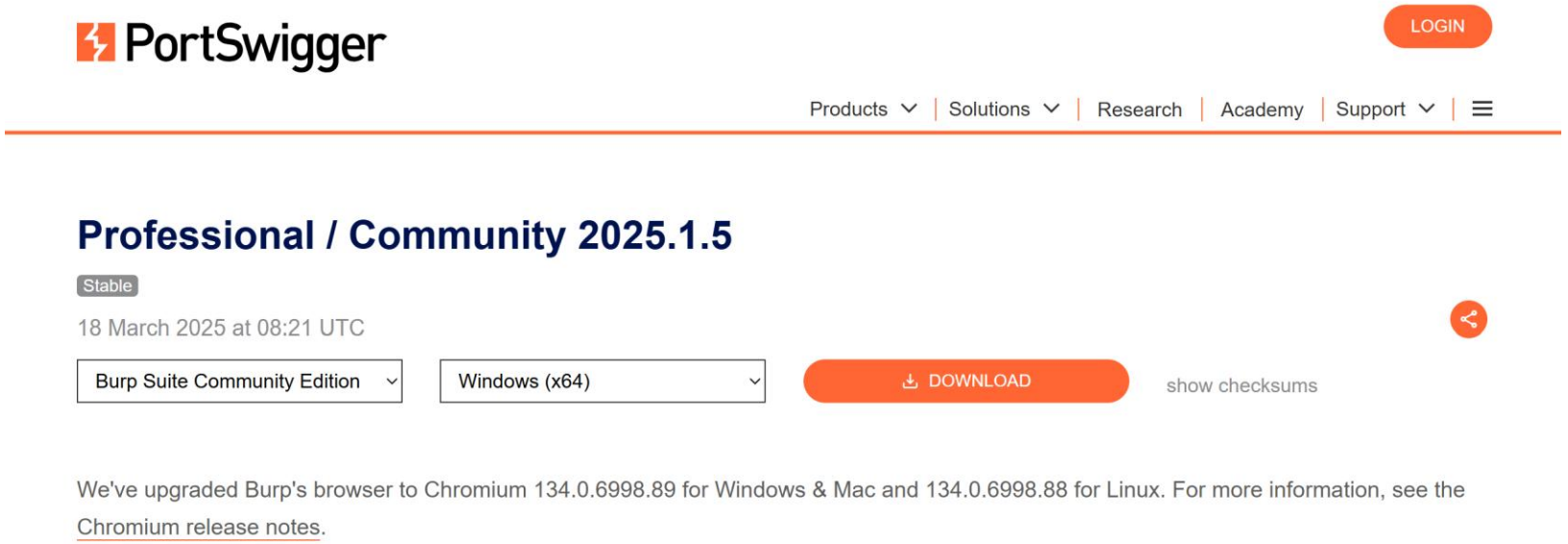
BURP SUITE

- **Step 1:** Follow this link https://portswigger.net/burp/communitydownload
- **Step 2:** Enter your email and click download



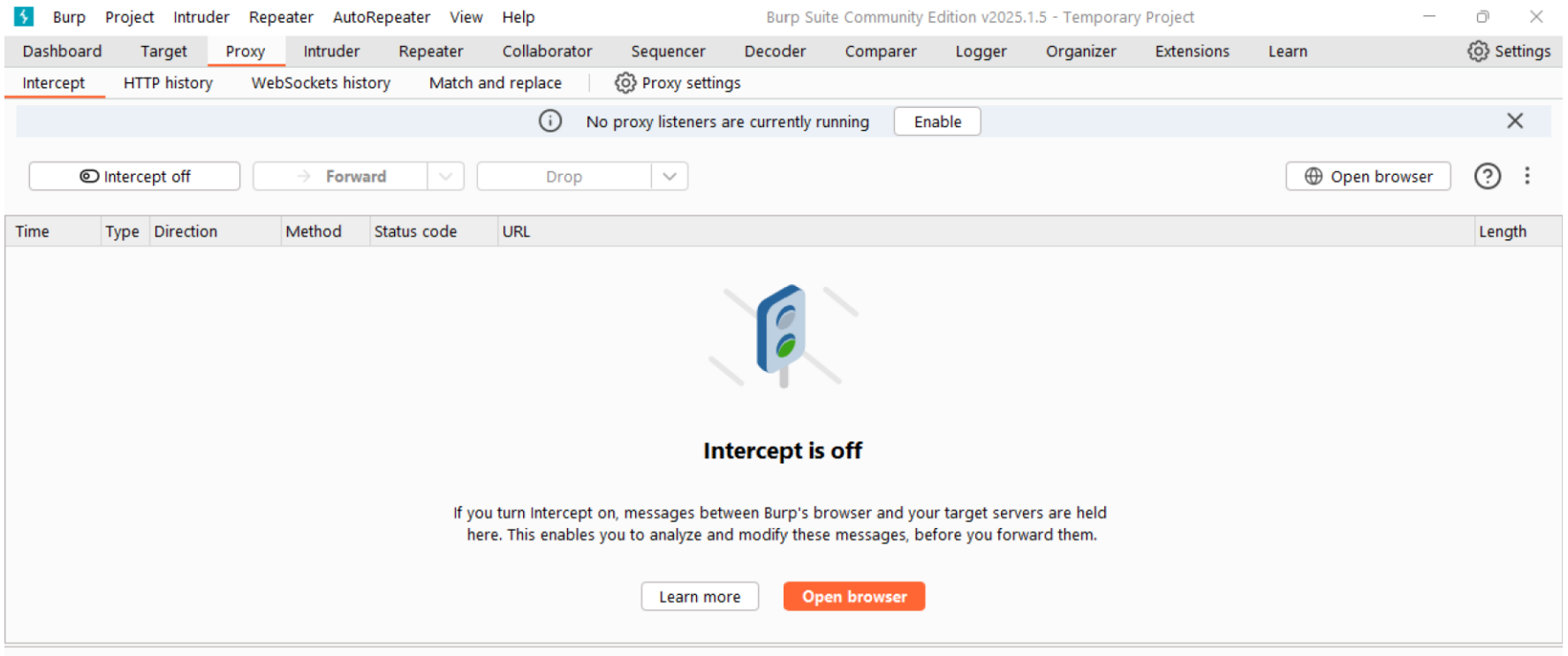
BURP SUITE

- **Step 3:** Select Community Edition and your operating system then click download
- **Step 4:** Open the installer and follow the on-screen instructions to complete the installation
- Once installed, launch Burp Suite and start hacking!



BURP SUITE

- Burp Suite consists of several essential tabs that help security professionals analyze and test web applications for vulnerabilities.
- In the next slide, we'll see a brief overview of the key tabs and their functions:

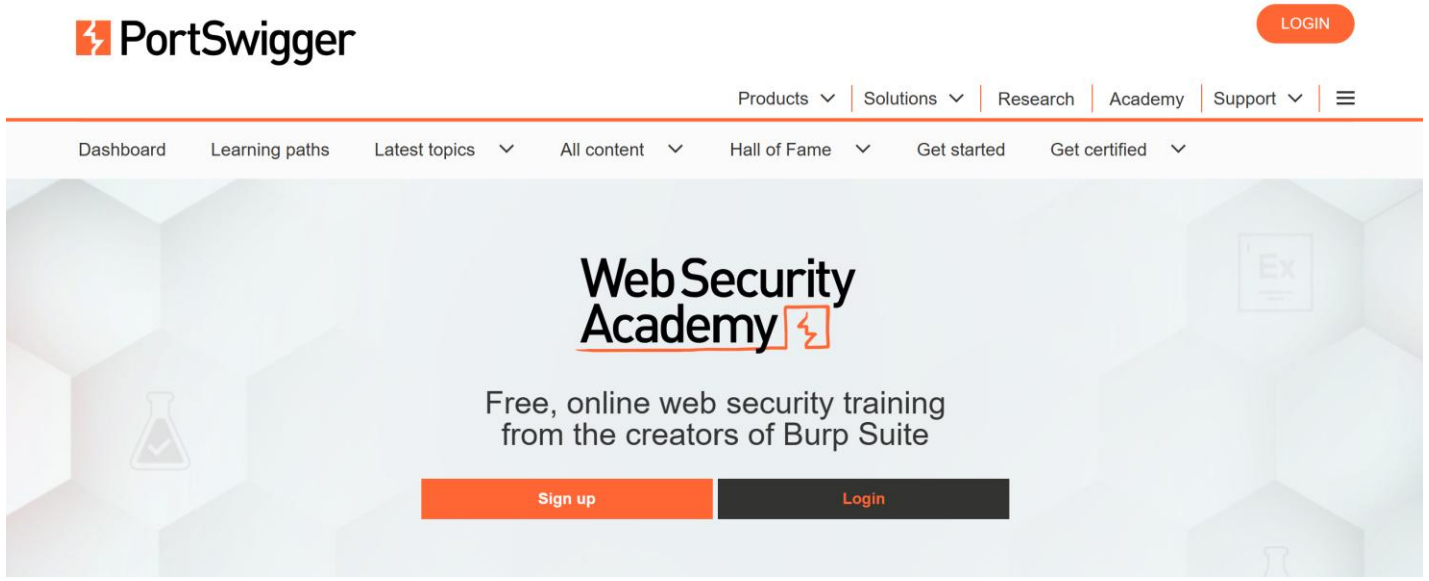


BURP SUITE

- **Target** – Displays a site map of the target application, showing all endpoints and directories discovered during interaction with the website.
- **Proxy** – Acts as an intercepting proxy between your browser and the target, allowing you to capture and modify HTTP/S requests and responses.
- **Intruder** – Automates attacks by sending multiple requests with different inputs from the defined payload, useful for brute-force attacks.
- **Repeater** – Allows manual testing by modifying and resending requests to analyze responses and identify security vulnerabilities.
- **Decoder** – Helps encode, decode, and hash data, useful for analyzing encoded data observed in requests/responses.
- **Extender** – Lets users add extensions and scripts to enhance Burp Suite's functionality.

BURP SUITE

- PortSwigger Academy is a free online platform offering web security training, created by the developers of Burp Suite. It provides interactive labs and learning resources to help users understand and exploit web vulnerabilities.
- Visit the academy https://portswigger.net/web-security
- Create an account and start hacking!



AUTHENTICATION LABS

- Guided Lab:

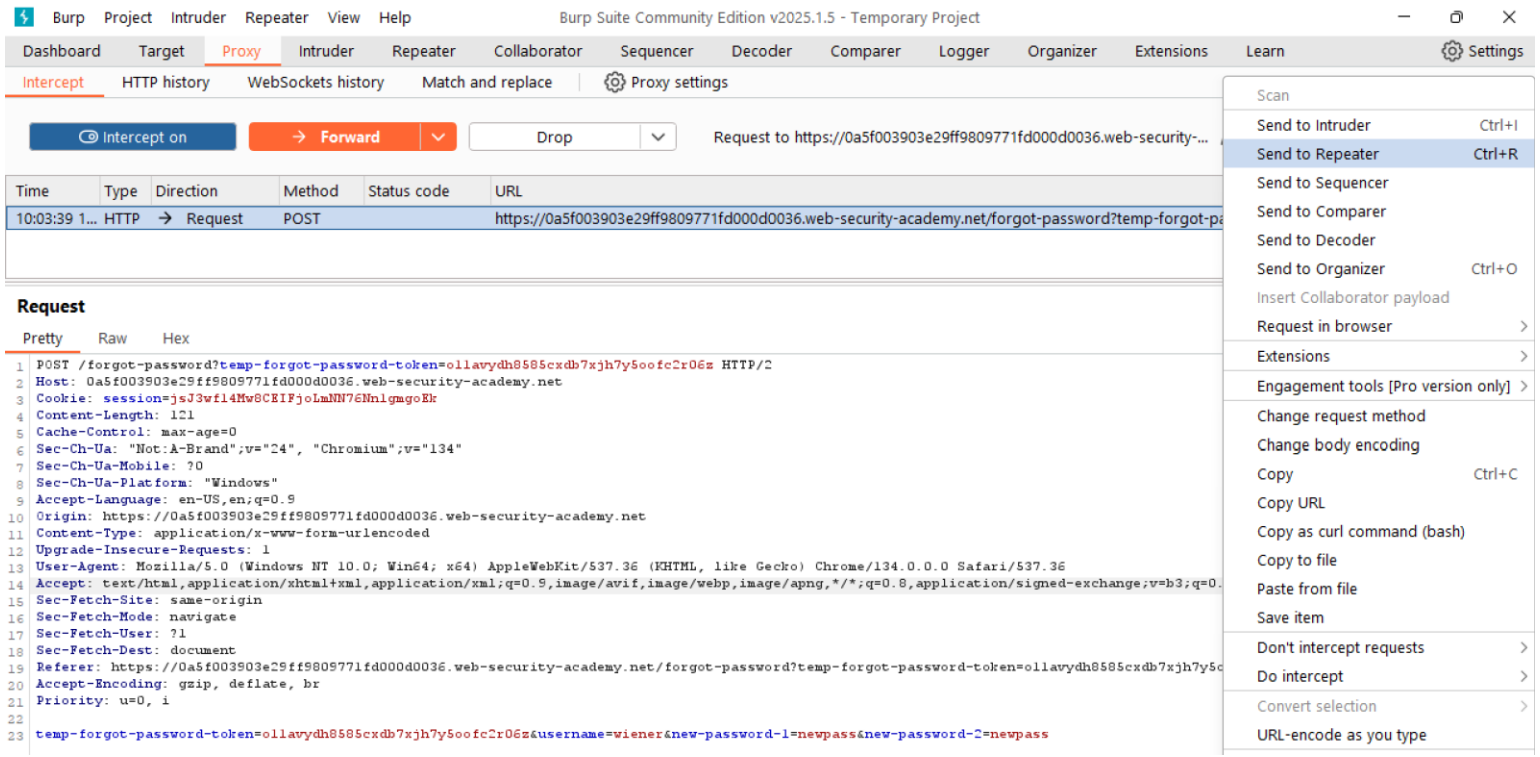
https://portswigger.net/web-security/authentication/other-mechanisms/lab-password-reset-broken-logic

- Challenge Lab:

https://portswigger.net/web-security/authentication/password-based/lab-username-enumeration-via-subtly-different-responses

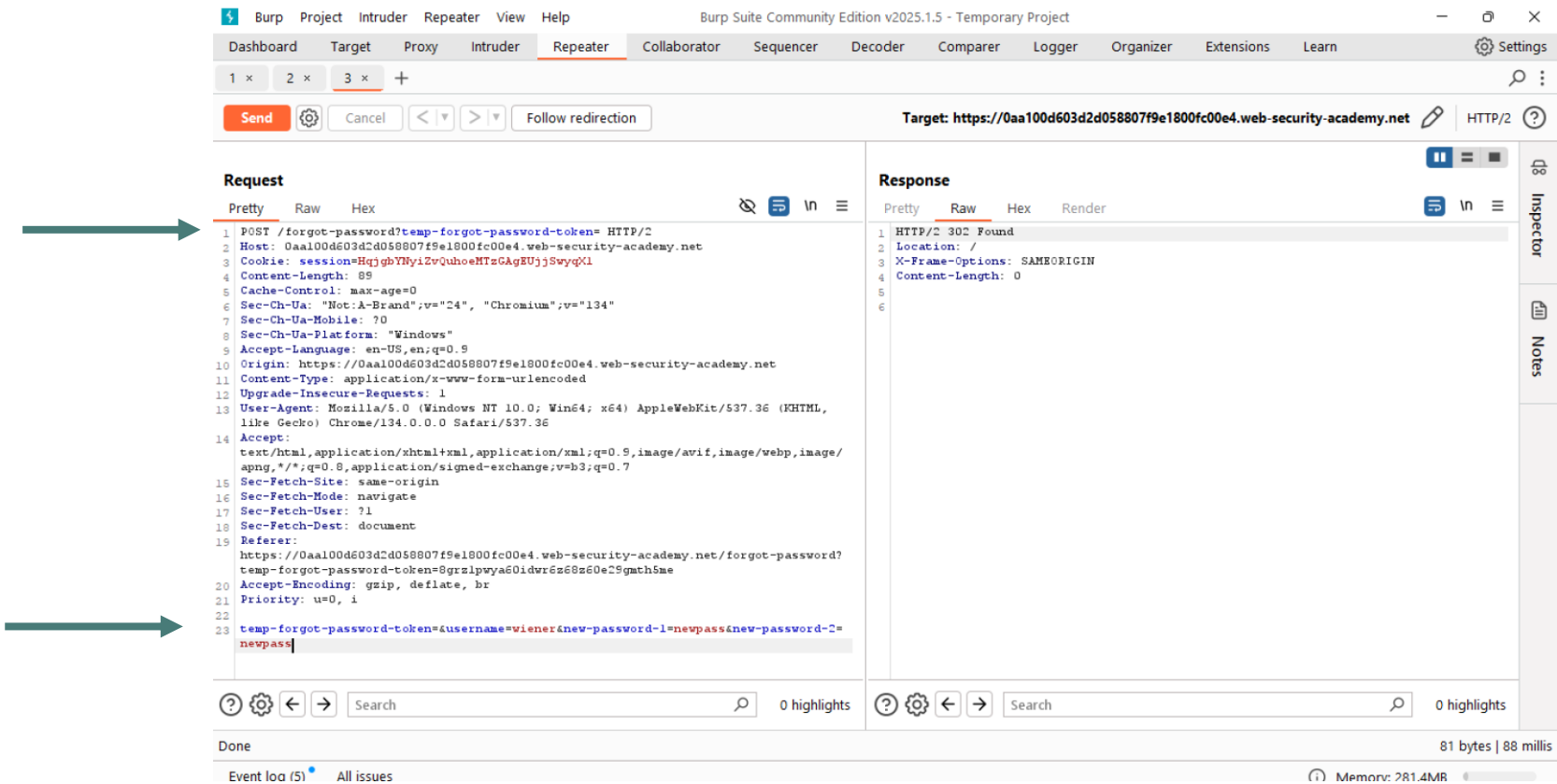
AUTHENTICATION LAB: PASSWORD RESET BROKEN LOGIC

- Since we're looking for authentication flaws, observe the login and forget password pages.
- Send the password reset POST request to Repeater.



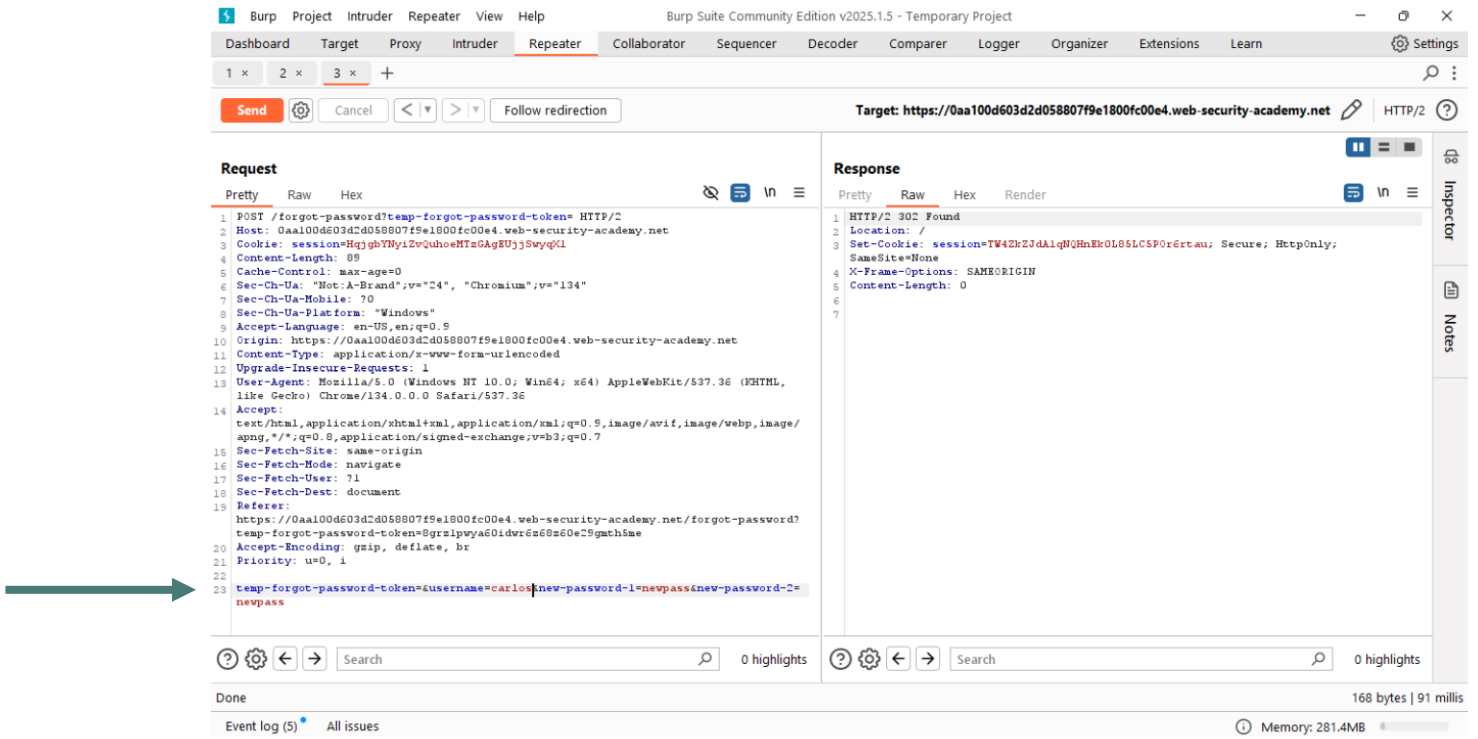
AUTHENTICATION LAB: PASSWORD RESET BROKEN LOGIC

- Test by removing the temp-forgot-password-token from both the URL and body. Observe that the reset still succeeds, confirming broken logic in token validation!



AUTHENTICATION LAB: PASSWORD RESET BROKEN LOGIC

- Initiate another reset and modify the POST request in Burp Repeater—clear the token values, change the username to carlos, and set a new password.
- Login as carlos using the new password to solve the lab.

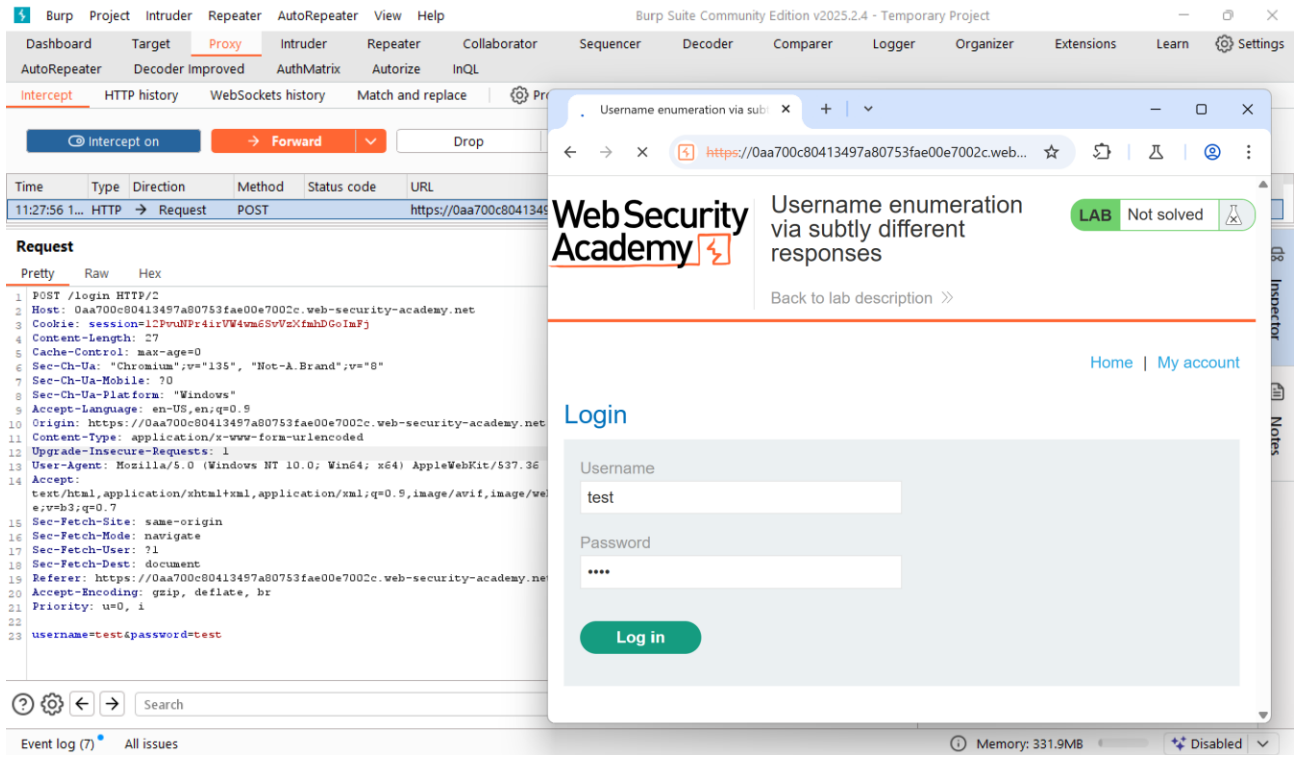


AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- To solve this lab, our goal is to identify a valid username and password combination that allows us to log in successfully.
- This typically involves two main techniques: enumeration and brute-force attacks.
- Enumeration helps us determine which usernames exist on the system, while brute-force is used to guess the correct password for a known username.
- The target account has a predictable username and password.
- The lab provides you with usernames and passwords wordlists to use.
- To carry out the attack efficiently, we'll use Burp Suite, a powerful web security testing tool.

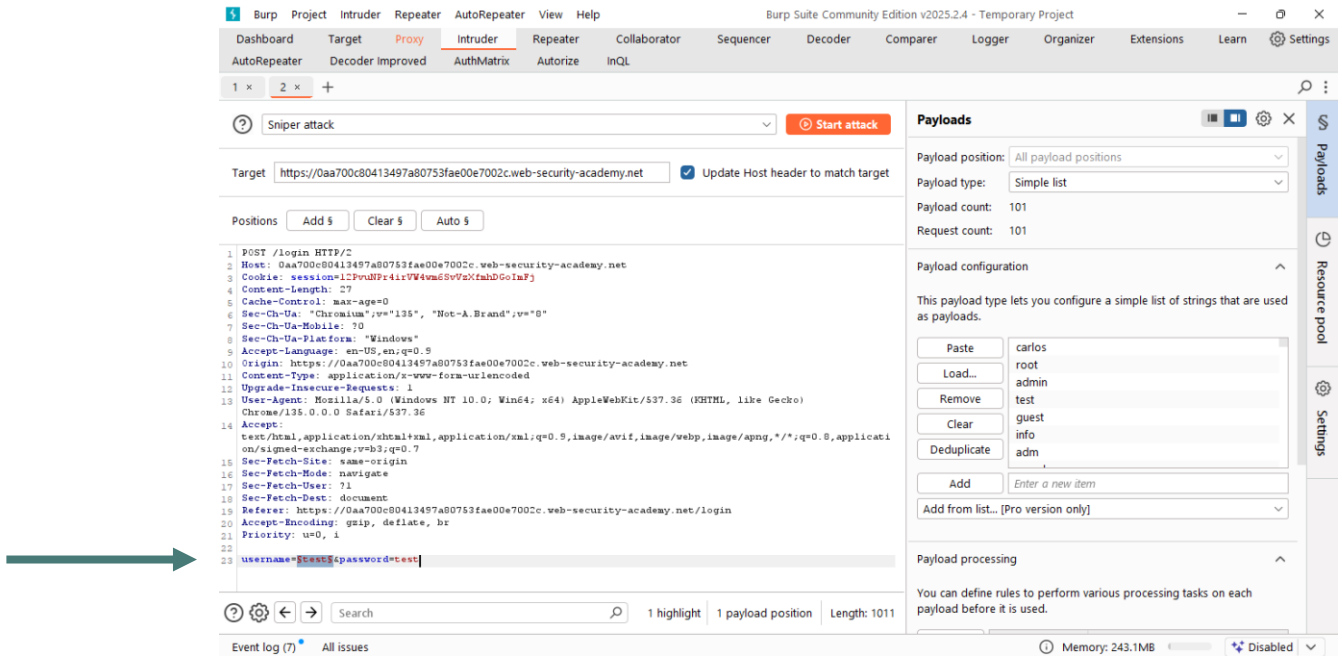
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- With Burp Suite browser is running, submit an invalid username and password.
- Send the captured login post request to Intruder by right-clicking on it and select Send to Intruder



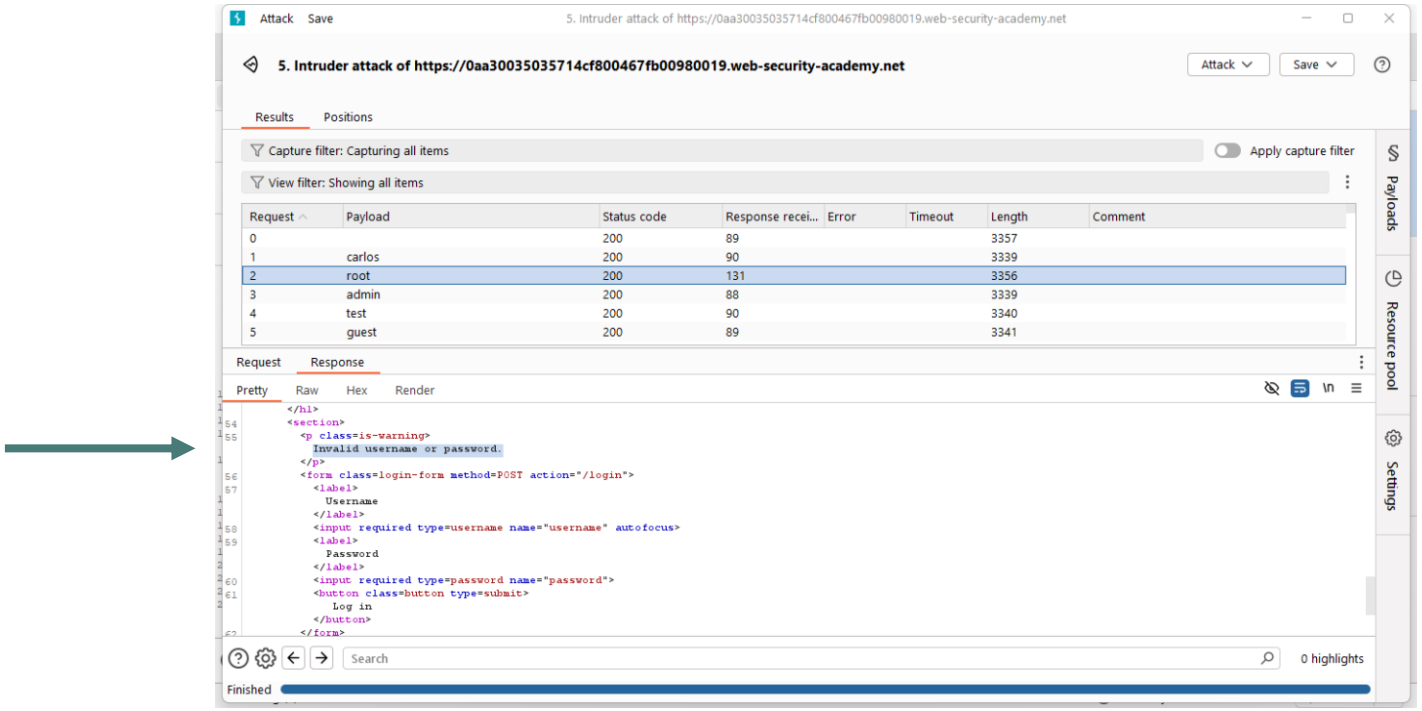
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Mark the username as a payload position.
- In the Payloads side panel, make sure that the Simple list payload type is selected and add the list of candidate usernames.
- Now click Start attack!



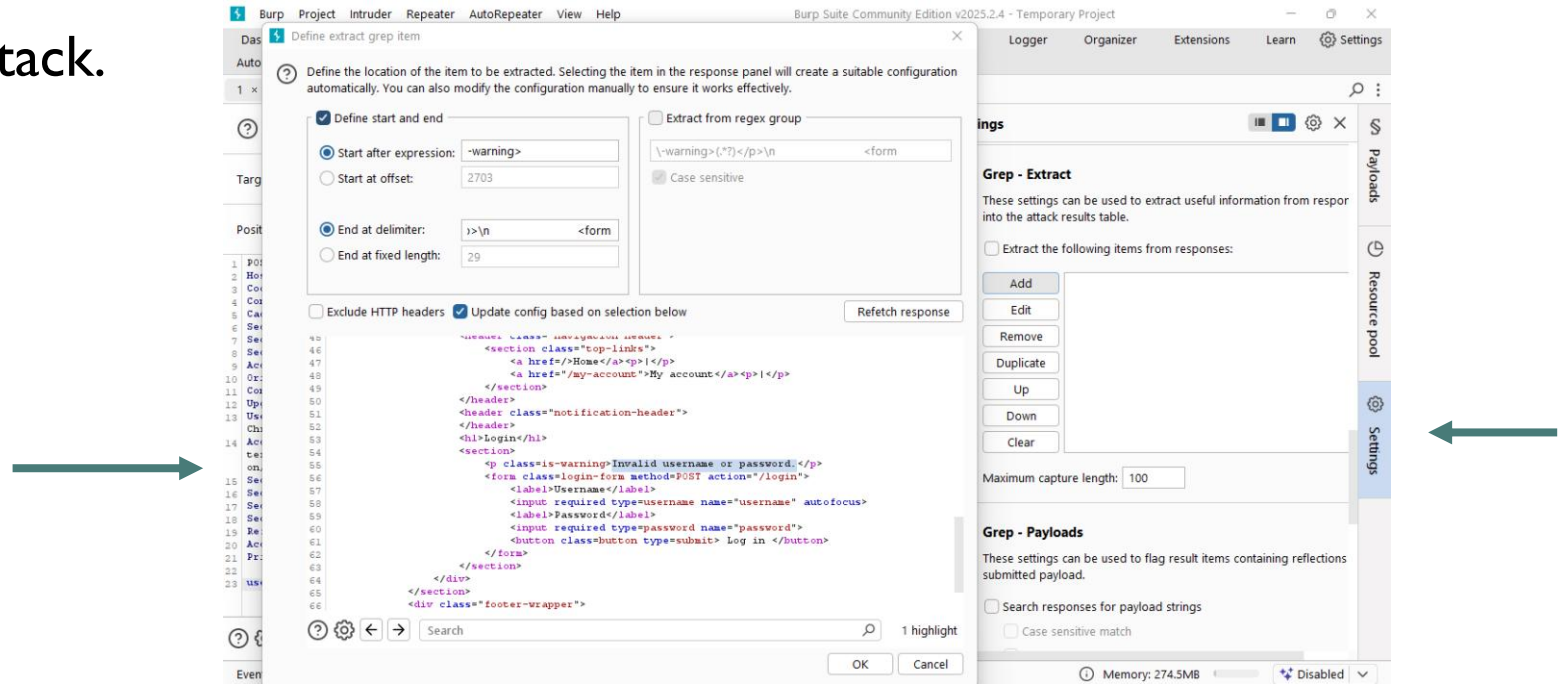
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Going through the brute-force results, you'll notice that most of the responses return the same message: "Invalid username or password."
- To narrow down potential valid usernames, we need to detect subtle variations in the response content.



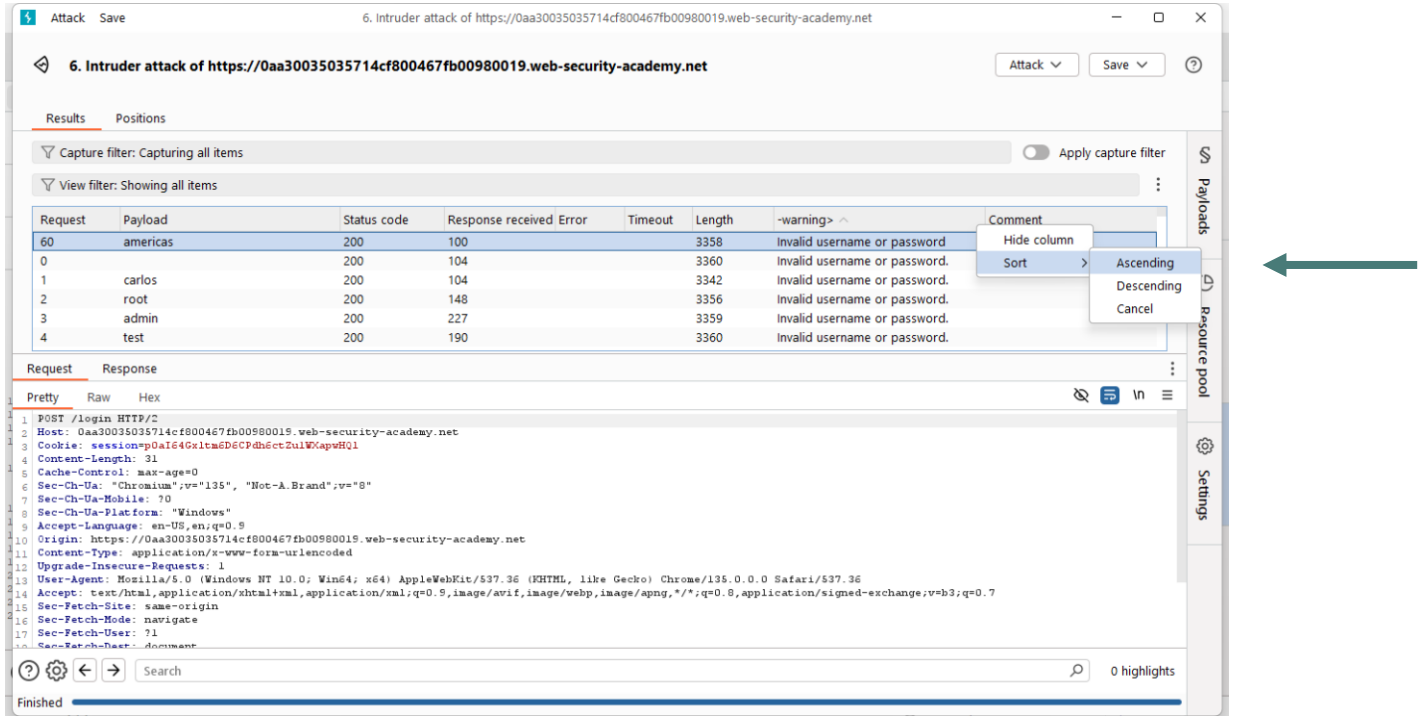
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Discard the previous attack and return to the Intruder tab.
- Navigate to the Settings side panel, click Add under Grep - Extract to define a new extraction rule, then scroll through the response to locate the "Invalid username or password." message, highlight it with your mouse to auto-fill the settings
- Click OK and start the attack.



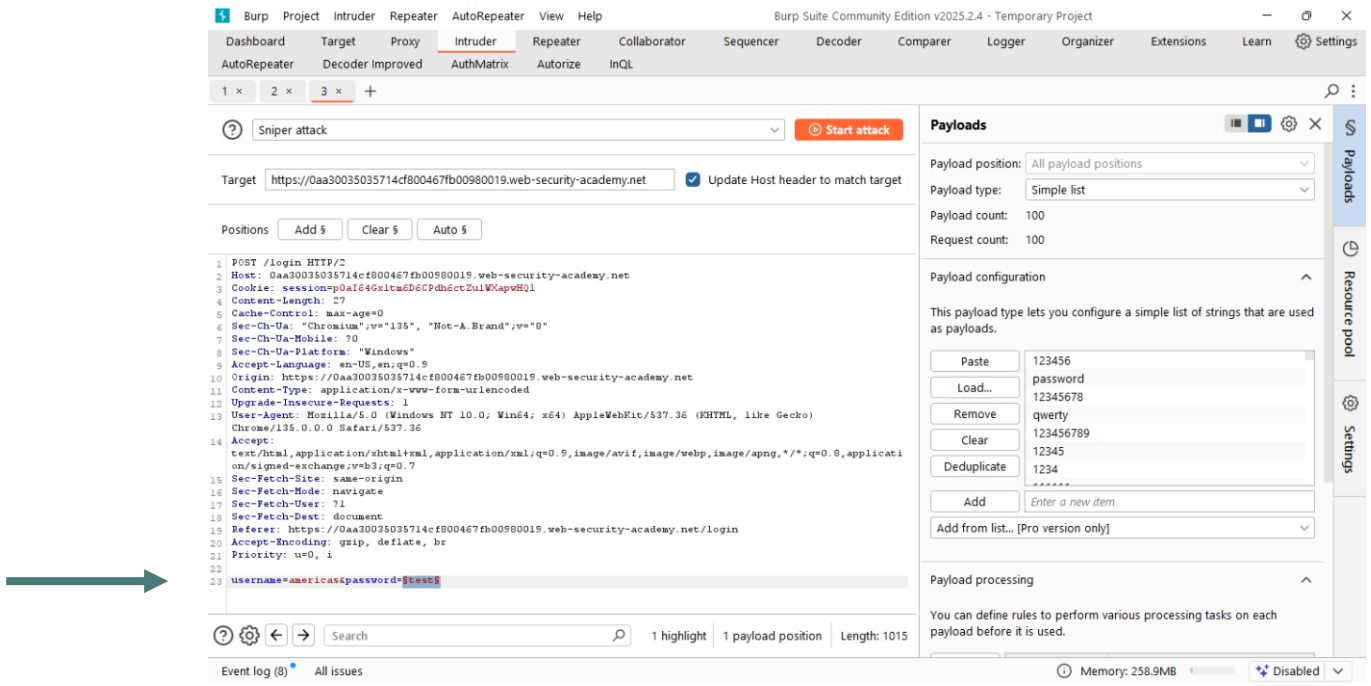
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Once the attack is complete, sort the results by the extracted message column and observe that one entry is subtly different.
- Upon closer inspection, you'll notice a slight typo in the error message — instead of ending with a period, it has a trailing space; make a note of the associated username!



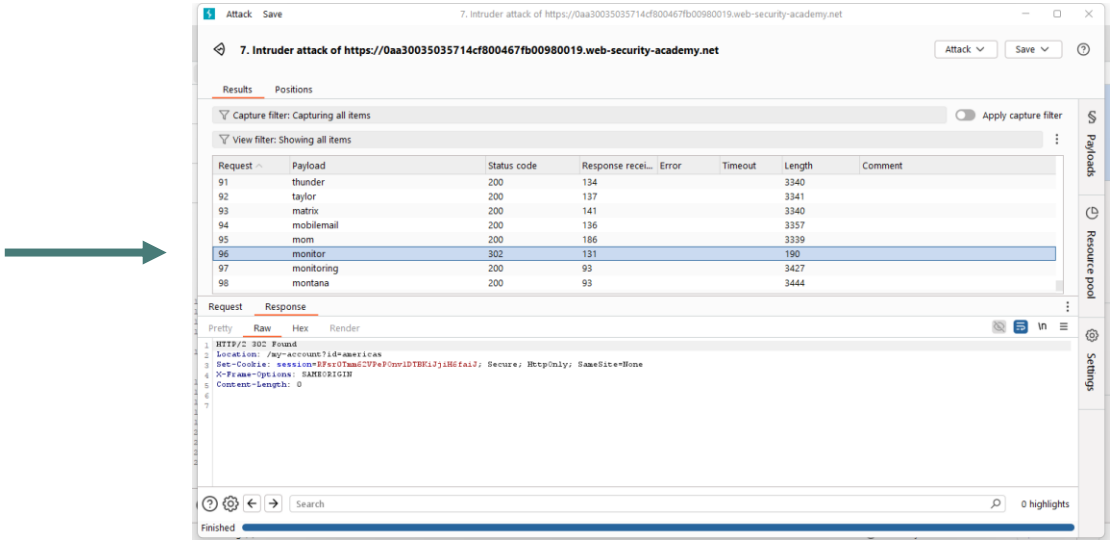
AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Close the results window and return to the Intruder tab, then replace the request body with the identified username and add a payload marker around the password value.
- Next, in the Payloads panel, clear the username list, load the provided list of passwords, and start the attack.



AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- Once the attack is complete, look for a request that received a 302 Found response — this indicates a successful login attempt, as the server is redirecting to the account page. Make a note of the associated password.
- Log in using the username and password that you identified and access the user account page to solve the lab.
- This lab demonstrates how subtle differences in server responses can be leveraged to enumerate valid usernames and perform targeted brute-force attacks.



ACCESS CONTROL LABS

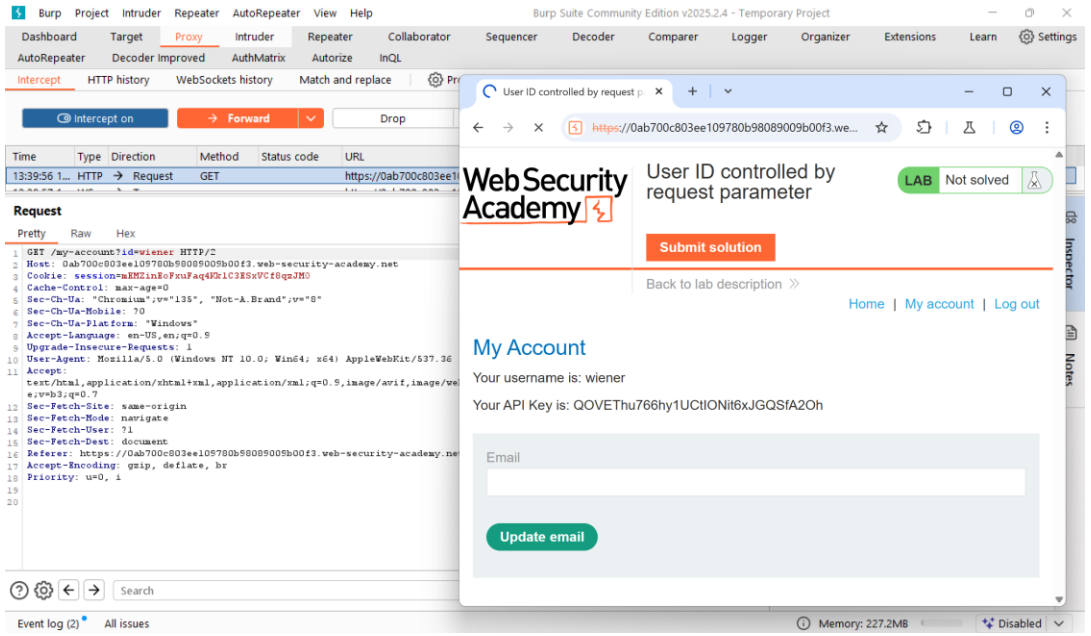
- Guided Lab: https://portswigger.net/web-security/access-control/lab-user-id-controlled-by-request-parameter
- Guided Lab: https://portswigger.net/web-security/access-control/lab-user-role-controlled-by-request-parameter

Broken Function Level Authorization

- Challenge Lab: https://portswigger.net/web-security/access-control/lab-insecure-direct-object-references
- Challenge Lab (Optional): https://portswigger.net/web-security/access-control/lab-unprotected-admin-functionality

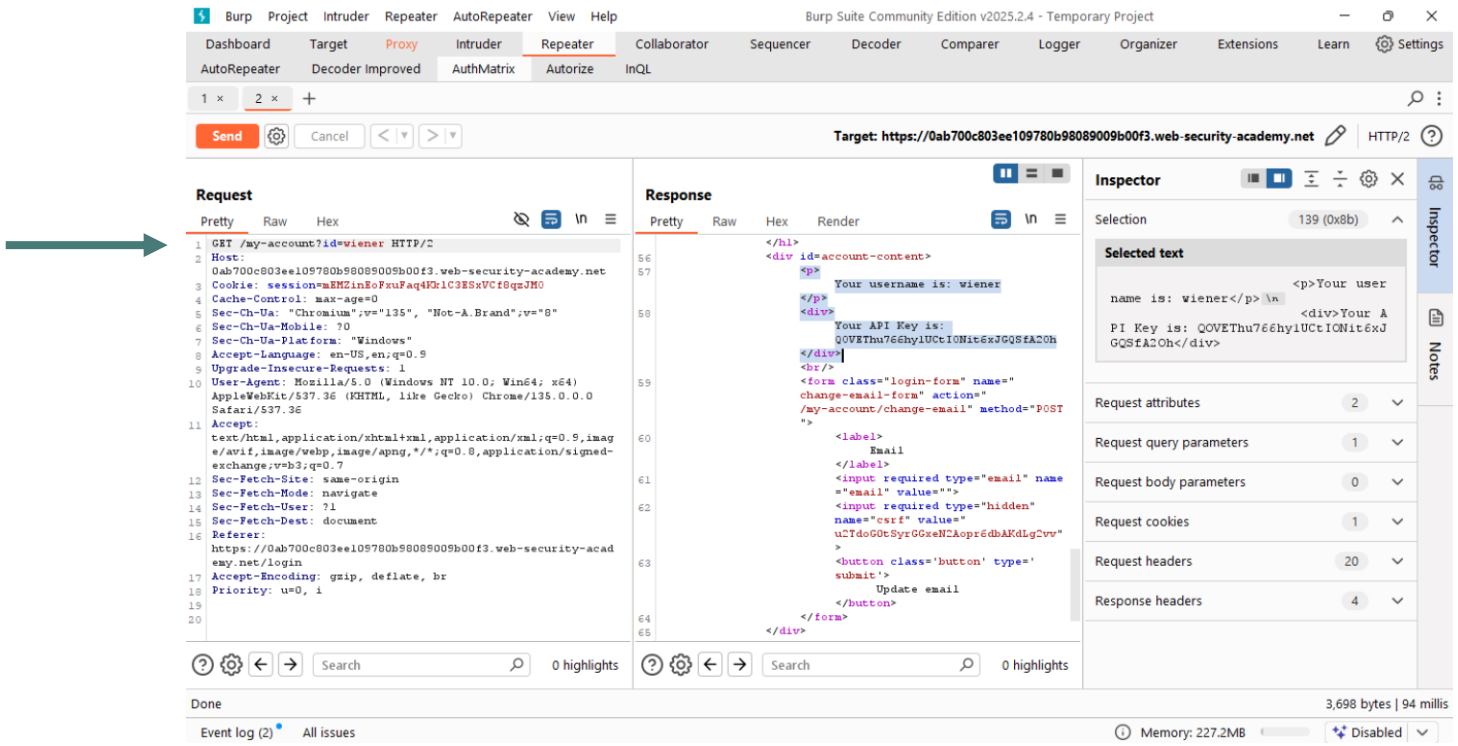
ACCESS CONTROL LAB: USER ID CONTROLLED BY REQUEST PARAMETER

- This lab demonstrates a horizontal privilege escalation vulnerability in the user account page.
- The vulnerability occurs when a user with a certain level of privileges can access other users' data at the same privilege level by manipulating request parameters.
- Log in using the credentials wiener:peter. Then, capture the account page request in Burp Suite.



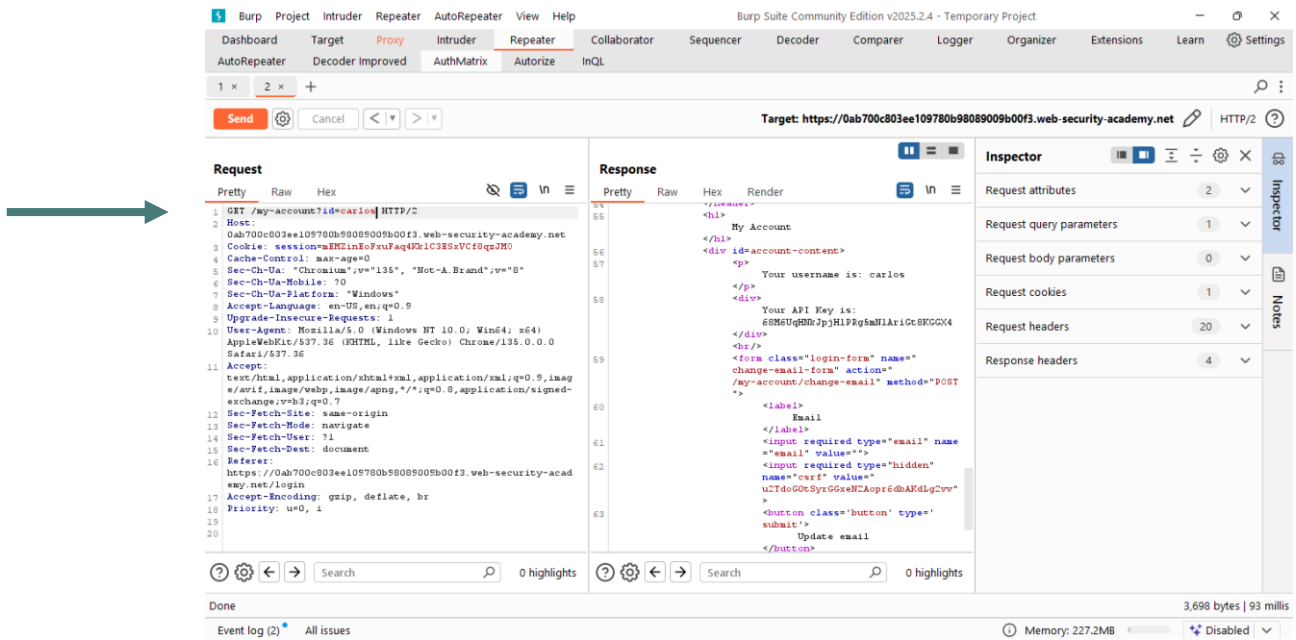
ACCESS CONTROL LAB: USER ID CONTROLLED BY REQUEST PARAMETER

- Send the request to the Repeater tab.
- Note that the URL contains your username in the "id" parameter.



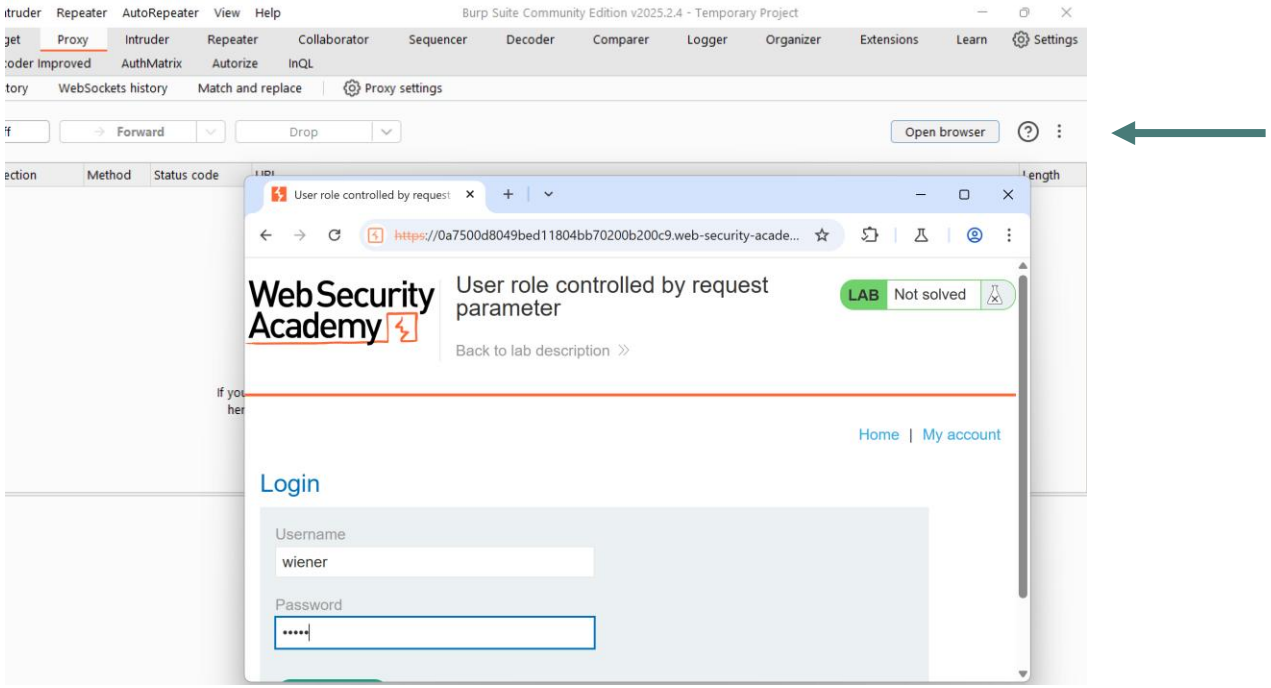
ACCESS CONTROL LAB: USER ID CONTROLLED BY REQUEST PARAMETER

- Modify the id parameter in the captured request to carlos to access his account page and retrieve his API key.
- This works because the application does not perform any backend verification to ensure that the requesting user is authorized to view another user's data.
- To solve the lab, obtain the API key for the user carlos and submit it as the solution.



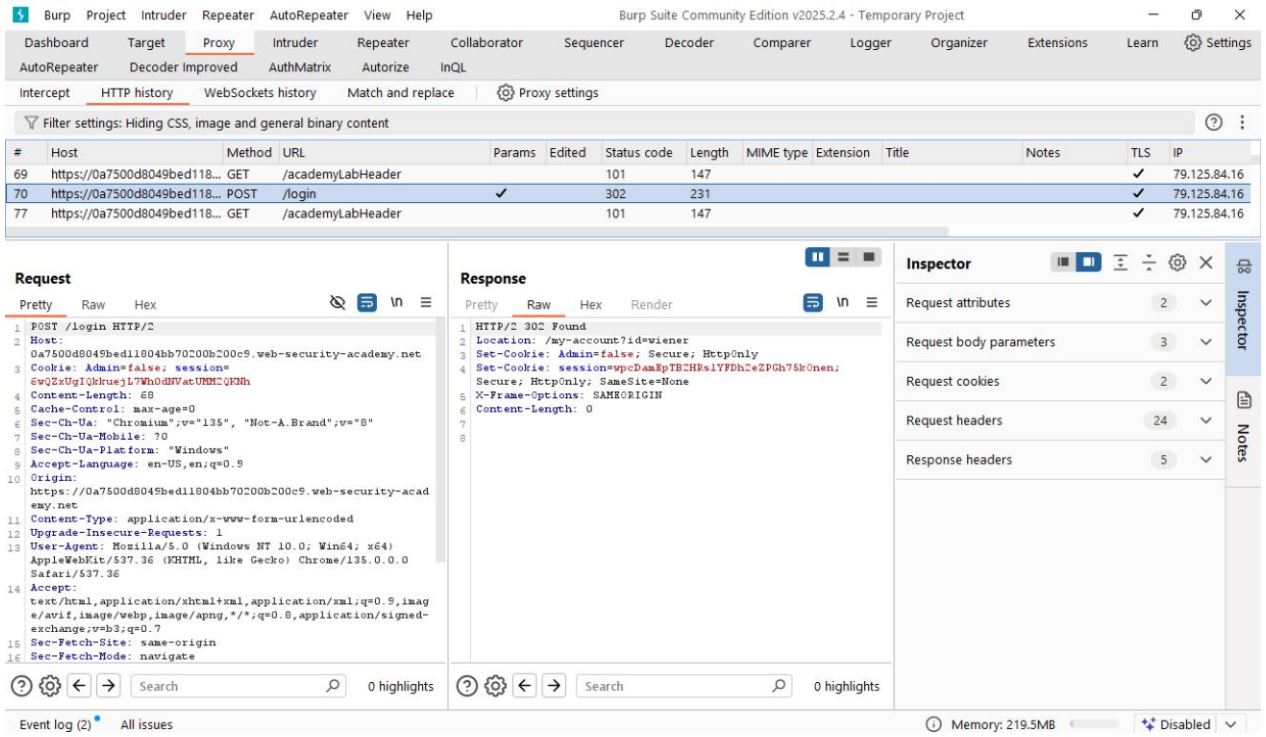
ACCESS CONTROL LAB: USER ROLE CONTROLLED BY REQUEST PARAMETER

- This lab demonstrates a broken access control vulnerability where access control is determined based on a cookie that we could alter.
- Our goal is to access the admin panel. Access the lab using the in-built web browser in Burp.
- Log in using the credentials wiener:peter.



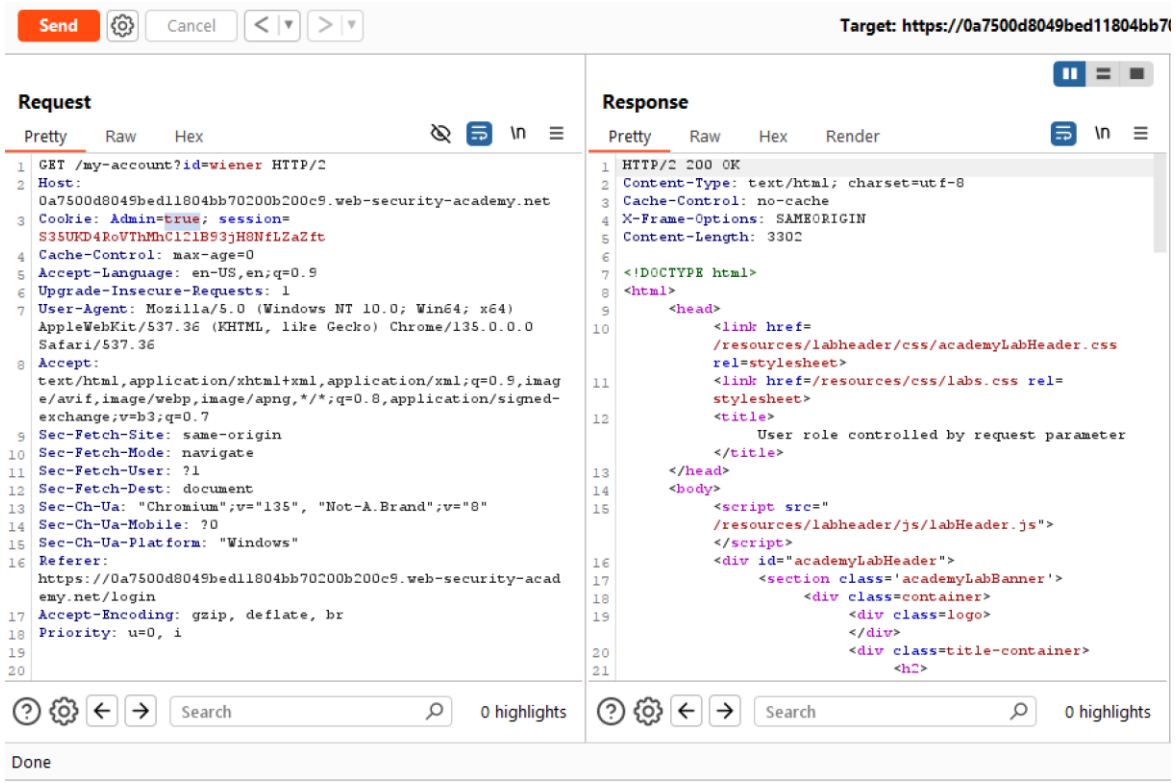
ACCESS CONTROL LAB: USER ROLE CONTROLLED BY REQUEST PARAMETER

- All your requests are going through Burp in the HTTP History tab under the Proxy tab.
- Find the login request and analyze the request
- Notice that you get to Set-Cookie. The Admin cookie is set to false!



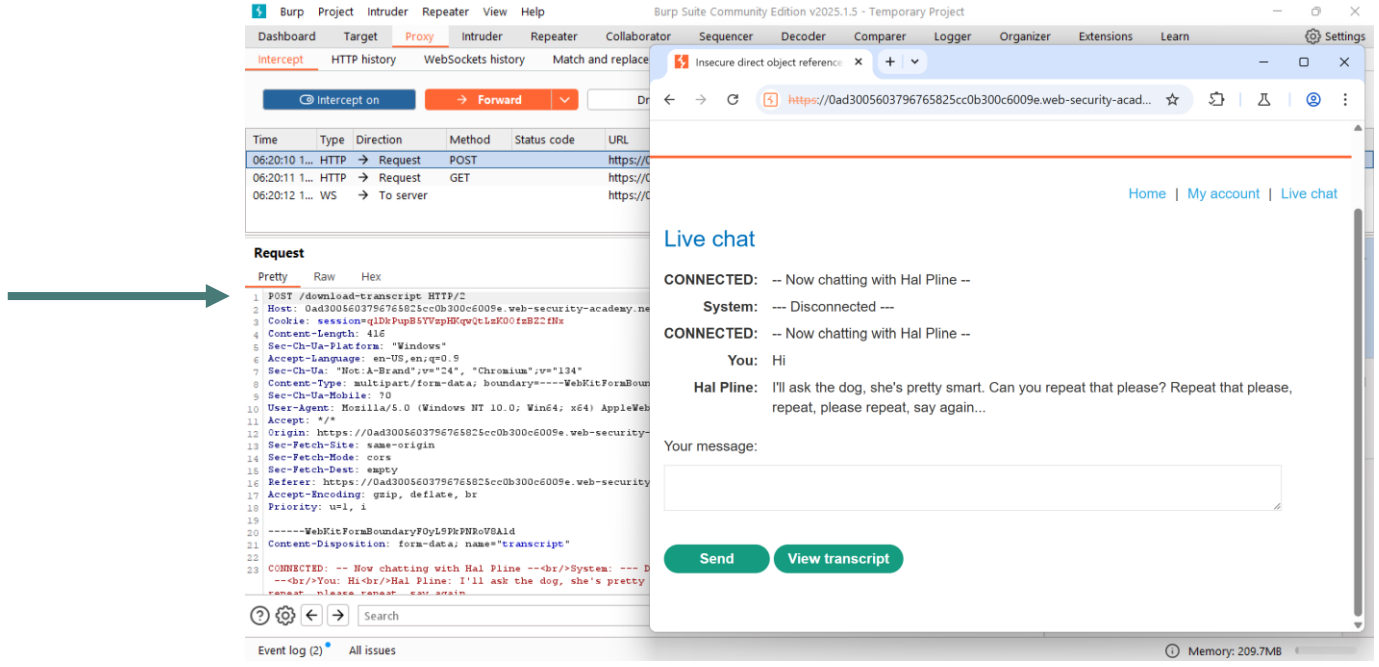
ACCESS CONTROL LAB: USER ROLE CONTROLLED BY REQUEST PARAMETER

- Now, find the account request in the HTTP History tab and send it to Repeater.
- Set the Admin cookie to true. We got access to the admin panel because this cookie sets the access control in the backend.
- Delete the user carlos to solve the solve!



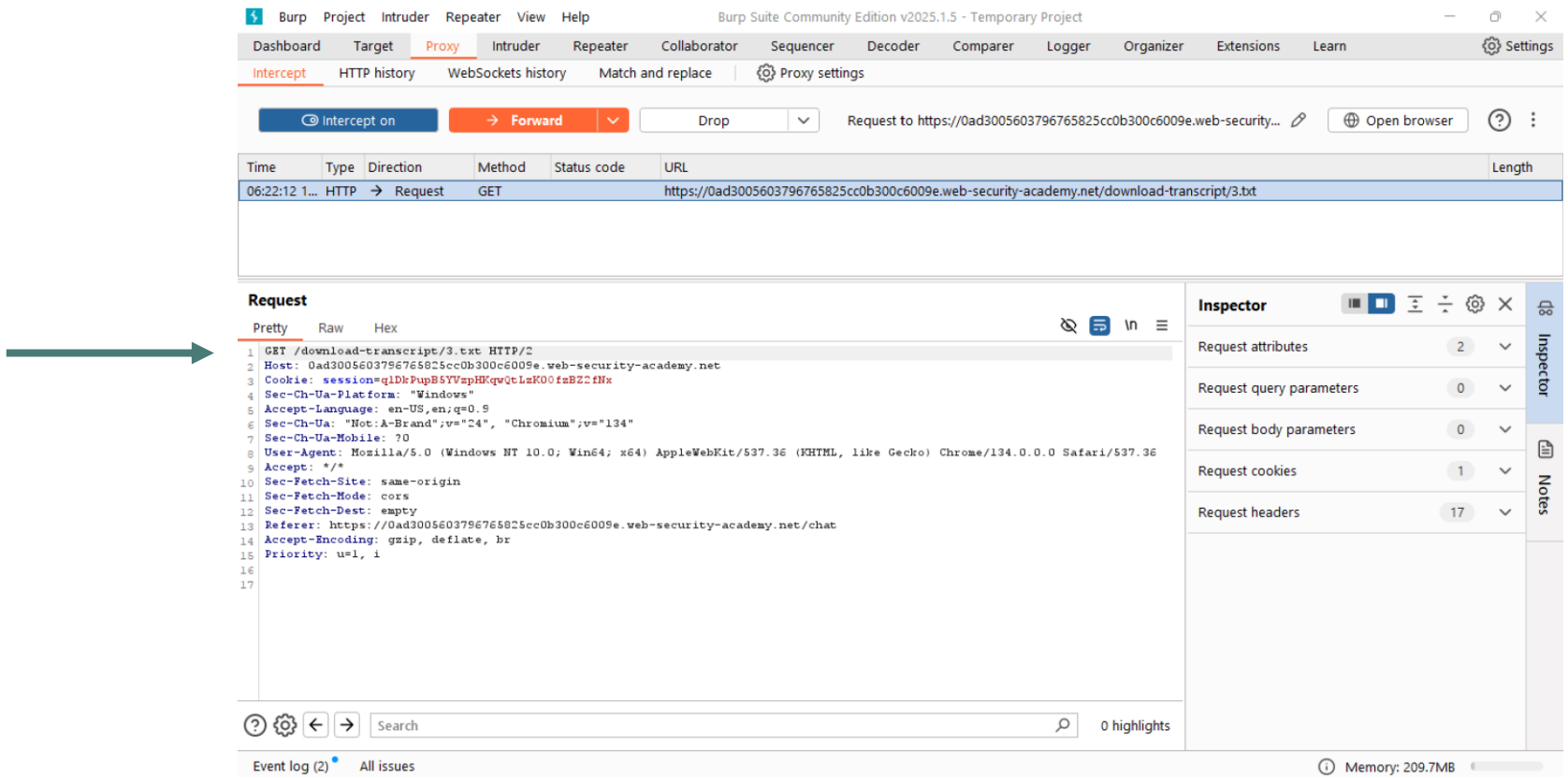
ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- This lab contains a login page and a live chat page. Since we're required to find the password of user carlos, let's interact with the live chat page in hope to find an IDOR.
- Send a message in the live chat. Turn on your burp proxy then select View transcript to capture the request.
- Notice there's /download-transcript endpoint, but no reference to an object.



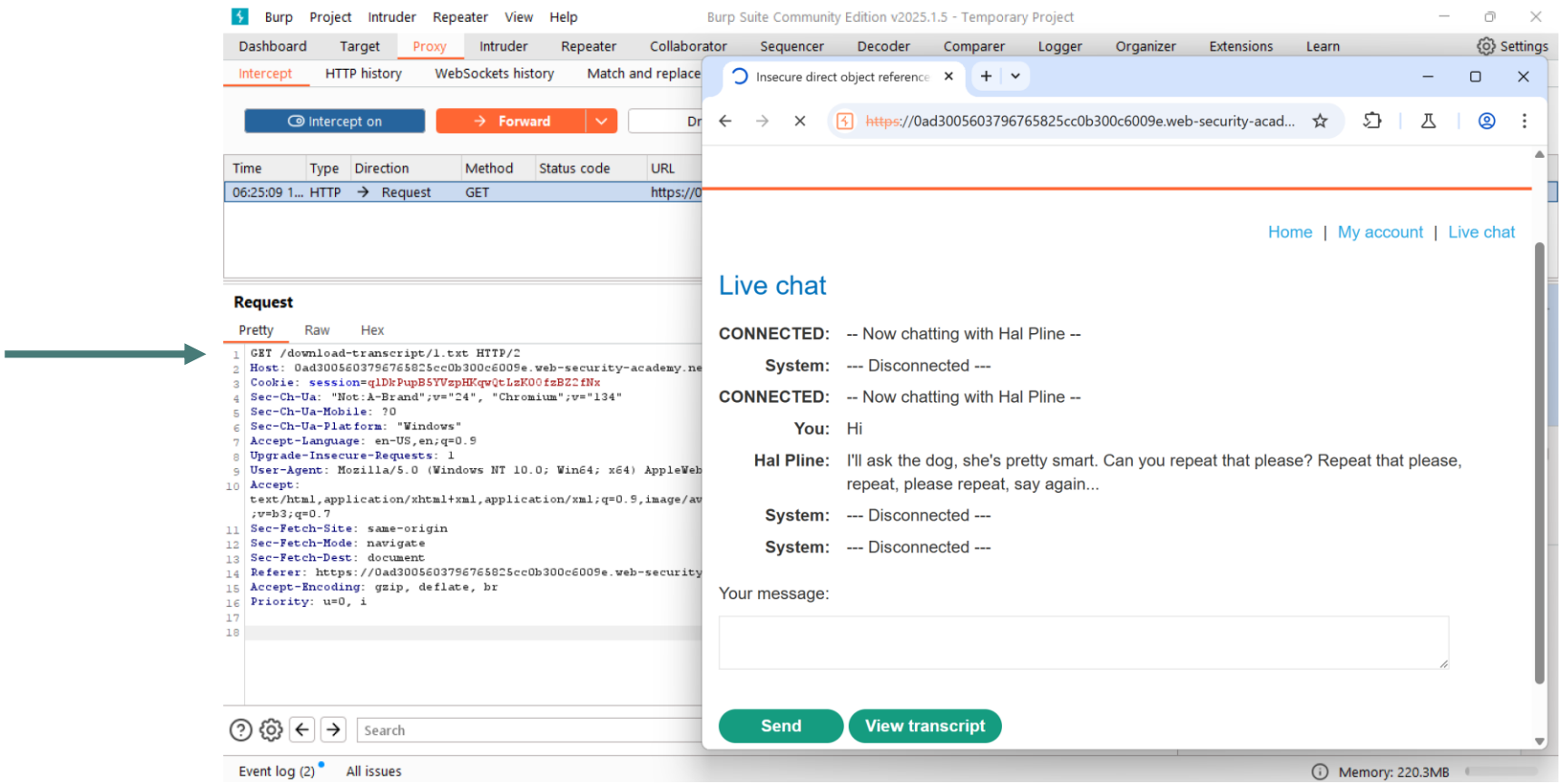
ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- Forward the requests then we'll reach an endpoint /download-transcript/3.txt. Observe that the transcripts are text files assigned a filename containing an incrementing number.



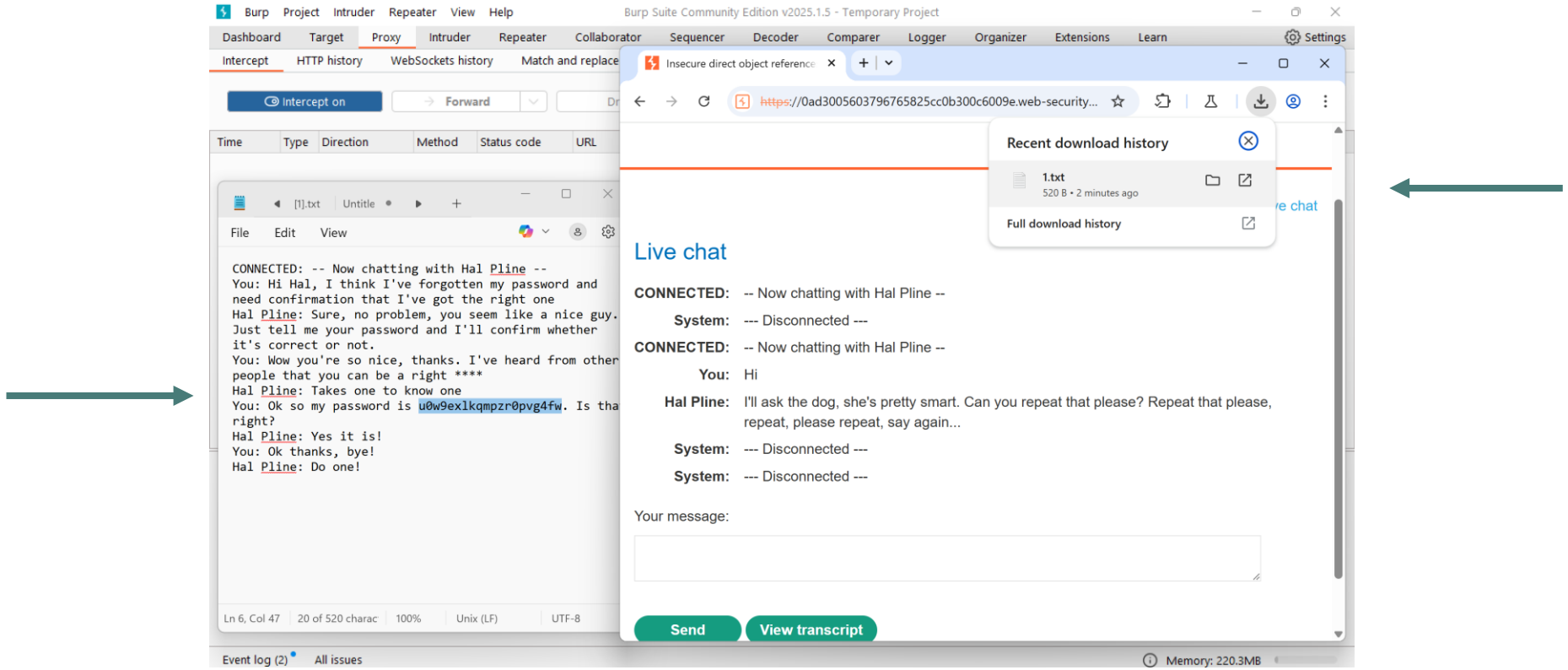
ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- Change the filename to 1.txt this might uncover a weakness in enforcing access controls in the View transcript functionality.



ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- Review the text file. Notice a password within the chat transcript. Return to the main lab page and log in using the stolen credentials.
- Login as carlos using the found password to solve the lab.

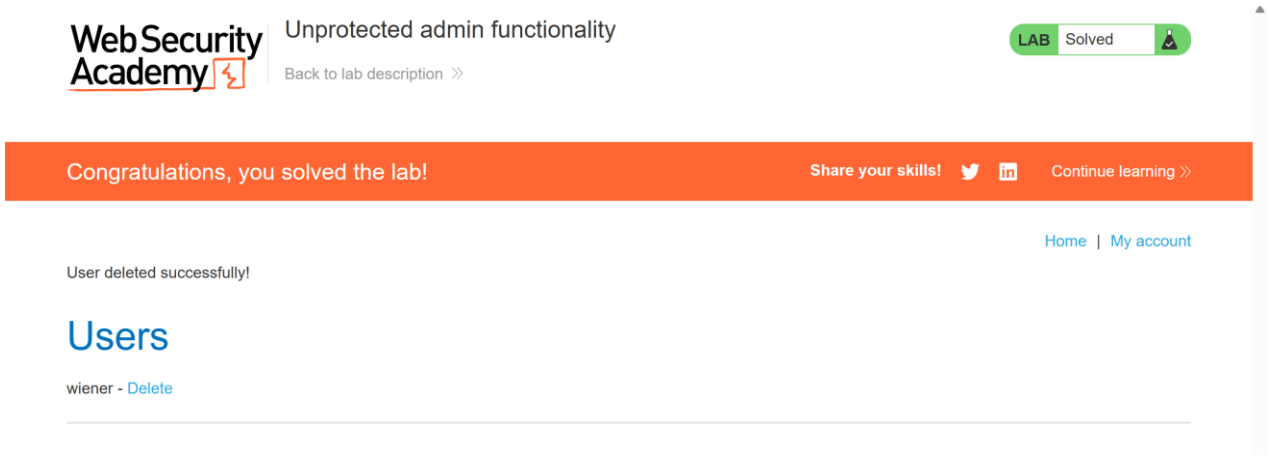


ACCESS CONTROL LAB: UNPROTECTED ADMIN FUNCTIONALITY

- This lab involves an unprotected admin panel, which allows unauthorized users to perform administrative actions.
- To begin, visit the lab URL and append /robots.txt to view the file, which lists paths that web crawlers should avoid.
- You'll notice a Disallow entry revealing the location of the admin panel.
- Replace /robots.txt in the URL with /administrator-panel to access the admin interface. From there, you're able to delete the user carlos, even though your account lacks admin privileges.

ACCESS CONTROL LAB: UNPROTECTED ADMIN FUNCTIONALITY

- Solve the lab by deleting the user carlos.
- This lab illustrates a Broken Function Level Authorization vulnerability, where the application fails to enforce proper access control at the function level.
- As a result, regular users can directly access and execute administrative functions simply by knowing or guessing the endpoint.

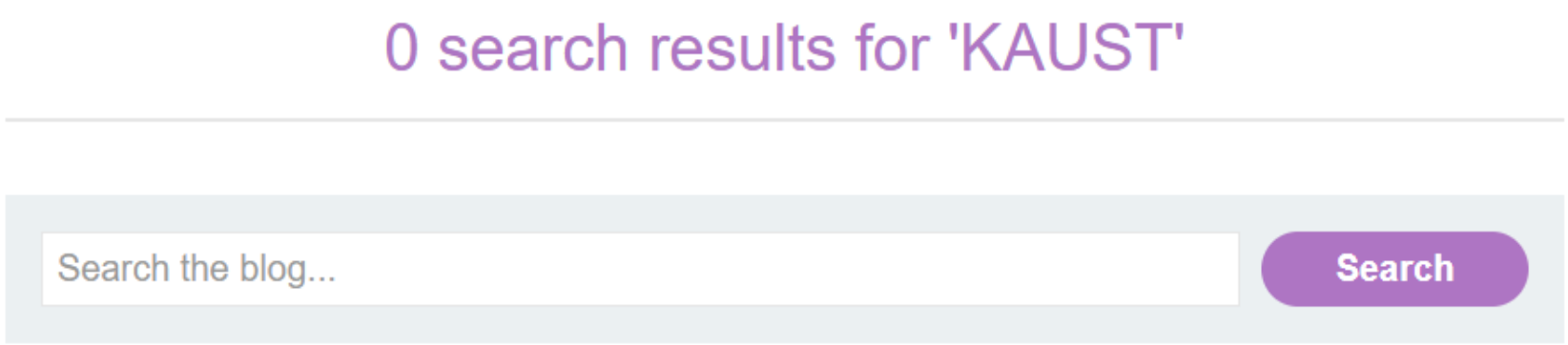


CROSS-SITE SCRIPTING: XSS LAB

- We will start with the simple guided labs first, then level up your skills!
- Guided Labs:
  - ✓Reflected XSS: https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded
  - ✓Stored XSS: https://portswigger.net/web-security/cross-site-scripting/stored/lab-html-context-nothing-encoded
  - ✓Document Object Model (DOM) XSS (Optional): https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink

XSS LAB: REFLECTED XSS

- Launch the Lab: Click the "Access the lab" button to begin.
- Explore the Interface, you'll see a search bar at the top and a list of posts, each with a "View Post" button.
- Enter a simple keyword in the search bar, for example, try searching for "kaust" and observe the results.
- Notice how the page reflected "kaust", that's a good sign of reflected XSS vulnerability



XSS LAB: REFLECTED XSS

- Now, let's test for a basic XSS vulnerability using one of the most widely used payloads:

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

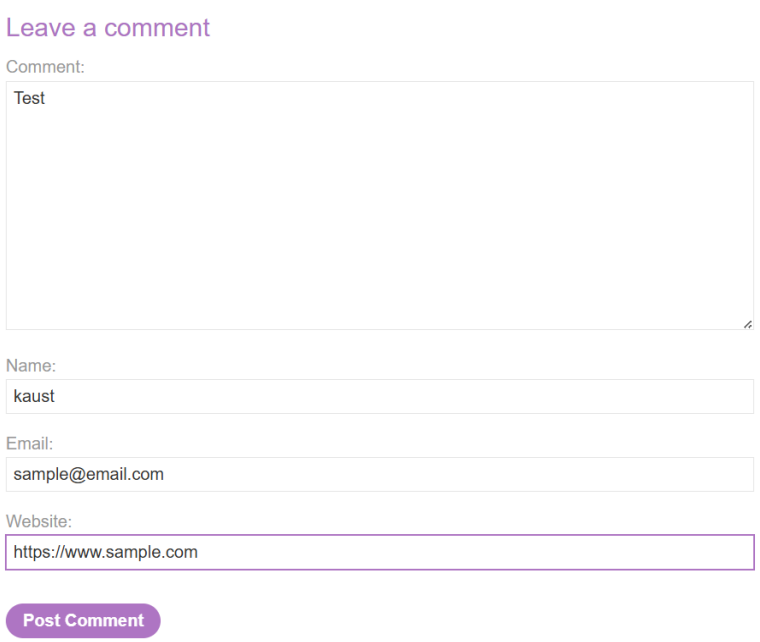
- Paste this into the search bar and hit enter.
- Success! You Triggered an Alert!
- If a JavaScript alert appears saying 1, it means the input is being improperly handled, the site is vulnerable to XSS!
- Click OK to dismiss the popup.

XSS LAB: STORED XSS

- Now, we'll explore another type of XSS vulnerability: Stored XSS, where the payload is saved on the server instead of being immediately reflected on the page.
- Fire up the lab and let's take a look around.
- This time, it's another blog website. At first glance, there's no visible input field on the homepage, and just like the previous lab. You'll see a list of posts, each with a "View Post" button.
- Click on the first post to view its details.
- Scroll to the bottom of the page, you'll find a comment input section we can test.
- Add some test info and submit it.
- **Note**: When testing websites, **never use your real name or personal data!**

XSS LAB: STORED XSS

- You'll get a message saying: "Thank you for your comment."
- Now go back to the blog post and you'll notice that your comment is displayed in the comments section. This means it's likely visible to other visitors as well.



XSS LAB: STORED XSS

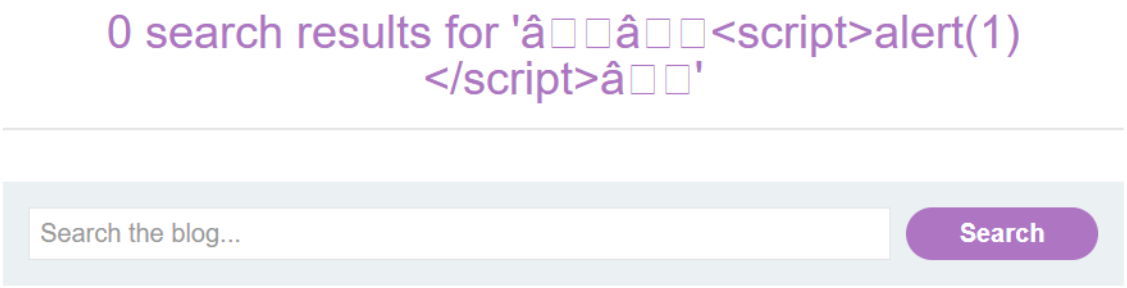
- Now let's test the comment field with our XSS payload:

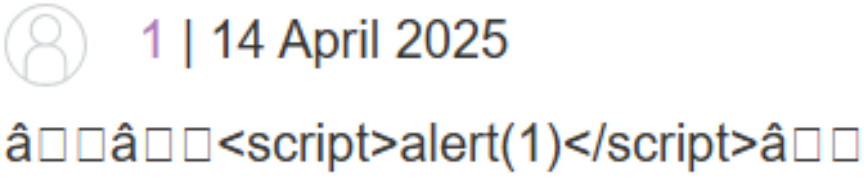
  <script>alert(1)</script>

- BOOM! The page is vulnerable to stored XSS attack!

XSS LAB: DOCUMENT OBJECT MODEL (DOM) XSS

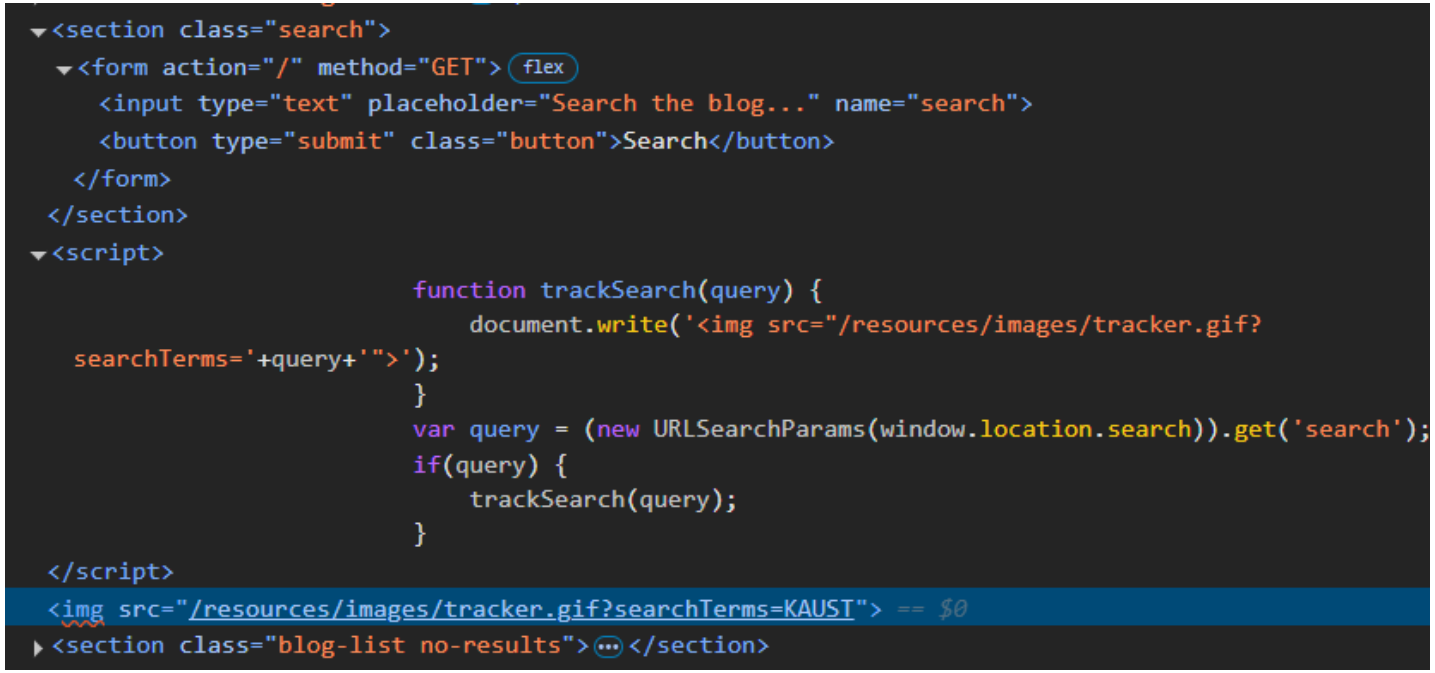
- For the final guided lab, we'll detect and exploit a DOM-based XSS vulnerability. As always, start by inspecting the page.
- It's another blog website. Try injecting our usual payload into the search bar, and also into the comment section of one of the posts. Nothing shows up, right?





XSS LAB: DOCUMENT OBJECT MODEL (DOM) XSS

- If this ever happens, backtrack and investigate. Check everything, the HTTP request using Burp Suite. Or, simply, inspect the page source code after submitting the input.
- Let's do what we did earlier: Enter "kaust" in the search bar, then right-click on the search bar and select "Inspect".
- You'll notice that the word "kaust" is now inserted inside an <img> tag.



XSS LAB: DOCUMENT OBJECT MODEL (DOM) XSS

- Using this information, we can craft a new payload.
- To exploit this: Close the current <img> tag properly using >, then inject a new image tag with an invalid source and an onerror handler to trigger our script.

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

- Submit the payload in the search bar
- Awesome! We've now confirmed that the site is vulnerable to DOM-based XSS.
- This worked because we escaped the existing DOM context and inserted our own malicious code into the page's structure. That's why it called DOM XSS!

XSS LAB

- Challenge Labs (Optional):

  - Reflected XSS: https://portswigger.net/web-security/cross-site-scripting/contexts/lab-html-context-with-most-tags-and-attributes-blocked
  - Stored XSS: https://portswigger.net/web-security/cross-site-scripting/contexts/lab-onclick-event-angle-brackets-double-quotes-html-encoded-single-quotes-backslash-escaped
  - DOM XSS: https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-document-write-sink-inside-select-element

XSS LAB: REFLECTED XSS

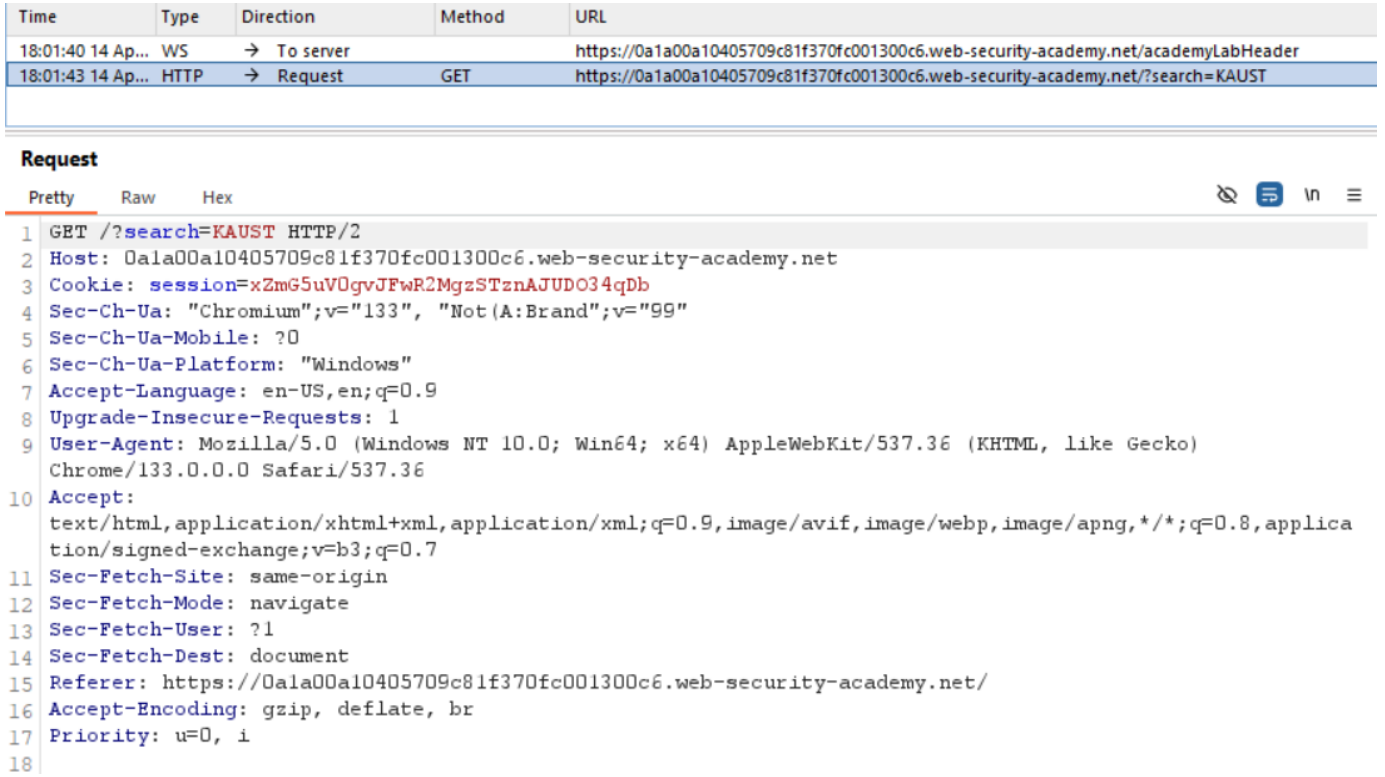
- Now let's get to the next level!
- This is our blog website, try testing the input field with the same payloads we used before, for example:

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

- You'll notice an error that says, "Tag is not allowed" This happens because the tag you used is being filtered.
- From this error, we can tell that some tags are blocked, but not all — otherwise, the error would have said something like "No tags allowed."

XSS LAB: REFLECTED XSS

- In order to find the correct tag, Fire up Burp Suite and Let's Intrude!
- Copy and paste the lab URL into Burp Suite's browser (or configure your proxy).
- Type anything into the search bar and intercept the request.



Time	Type	Direction	Method	URL
18:01:40 14 Ap...	WS	→ To server		https://0a1a00a10405709c81f370fc001300c6.web-security-academy.net/academyLabHeader
18:01:43 14 Ap...	HTTP	→ Request	GET	https://0a1a00a10405709c81f370fc001300c6.web-security-academy.net/?search=KAUST

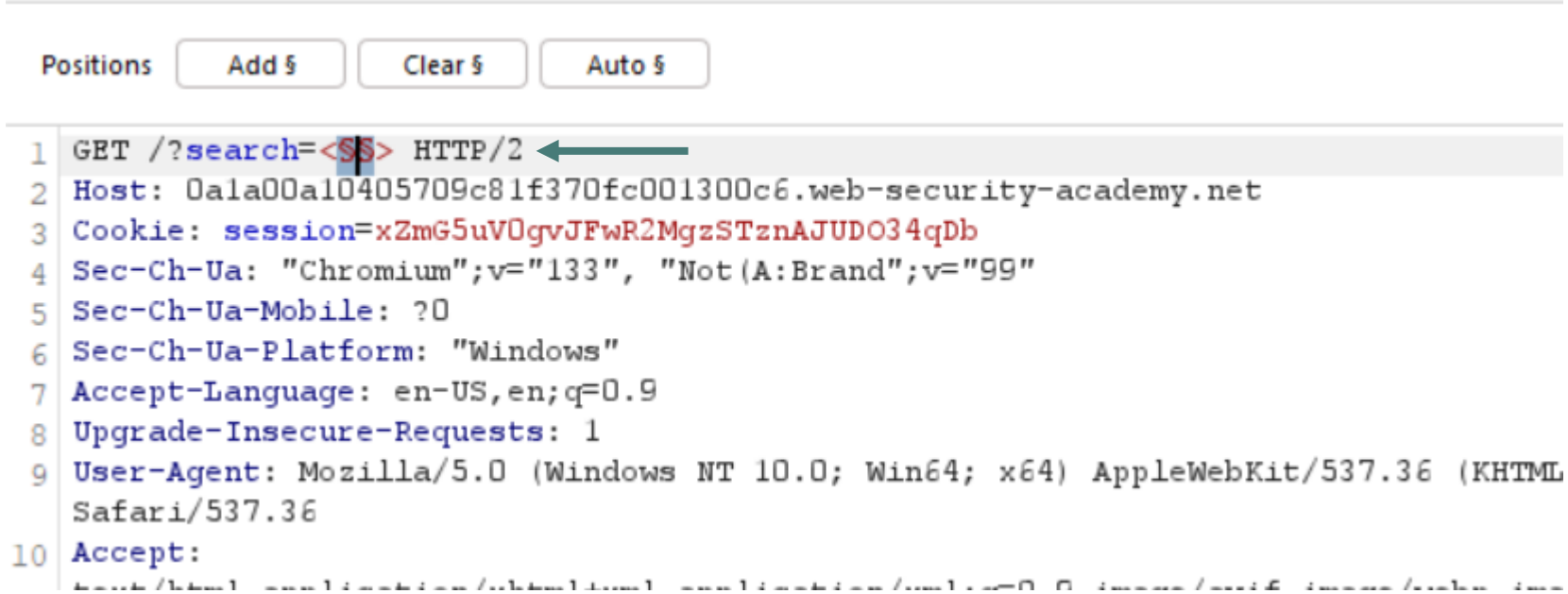
Request

Pretty Raw Hex

```
1  GET /?search=KAUST HTTP/2
2  Host: 0a1a00a10405709c81f370fc001300c6.web-security-academy.net
3  Cookie: session=xZmG5uV0gvJFwR2MgzSTznAJUDO34qDb
4  Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Windows"
7  Accept-Language: en-US,en;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/133.0.0.0 Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applica
   tion/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: https://0a1a00a10405709c81f370fc001300c6.web-security-academy.net/
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18
```

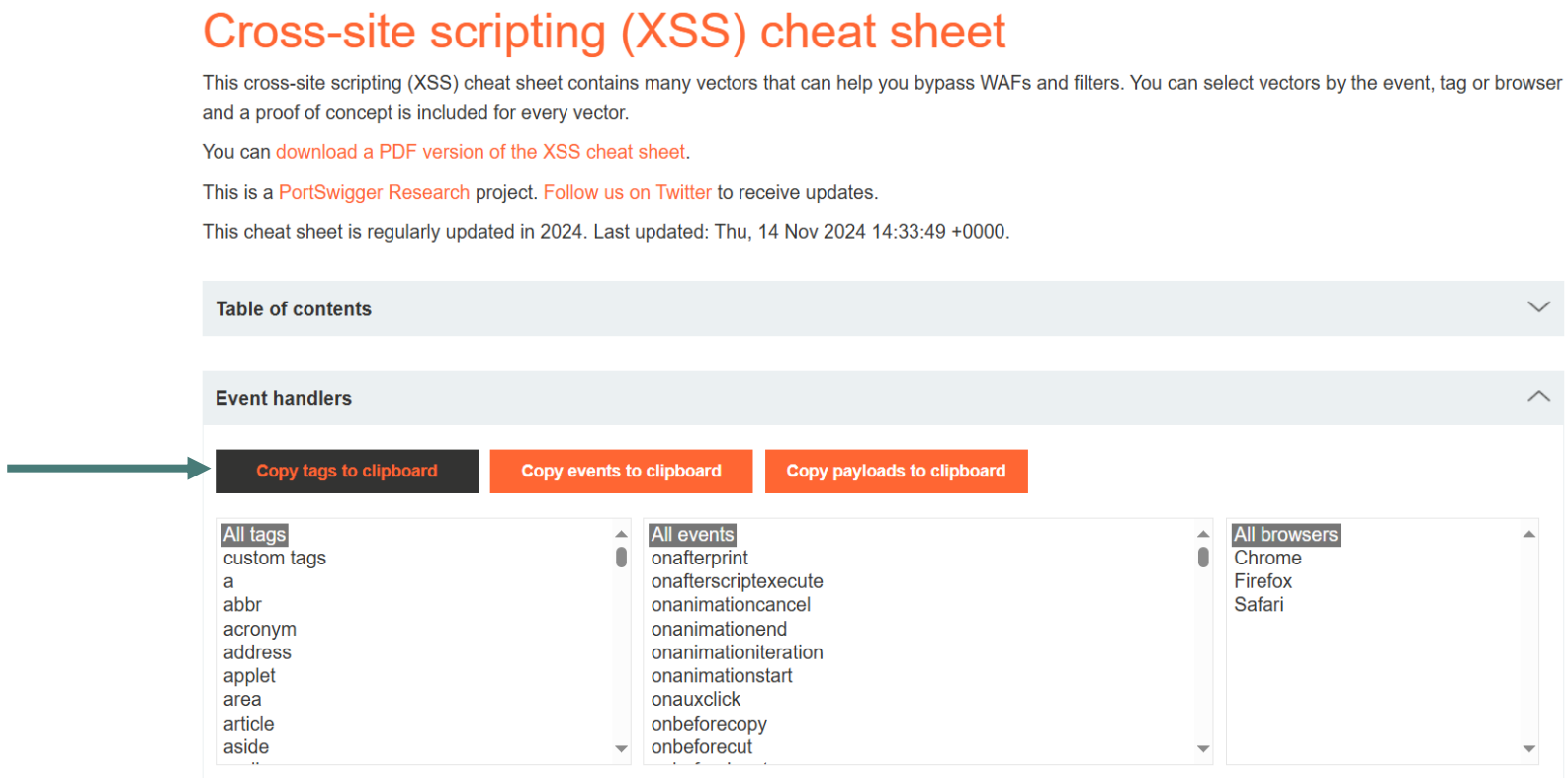
XSS LAB: REFLECTED XSS

- Send the intercepted request to Intruder.
- In the request, replace your input (e.g., "kaust") with angle brackets <>.
- Place the cursor between them and click Add § to mark the payload position.



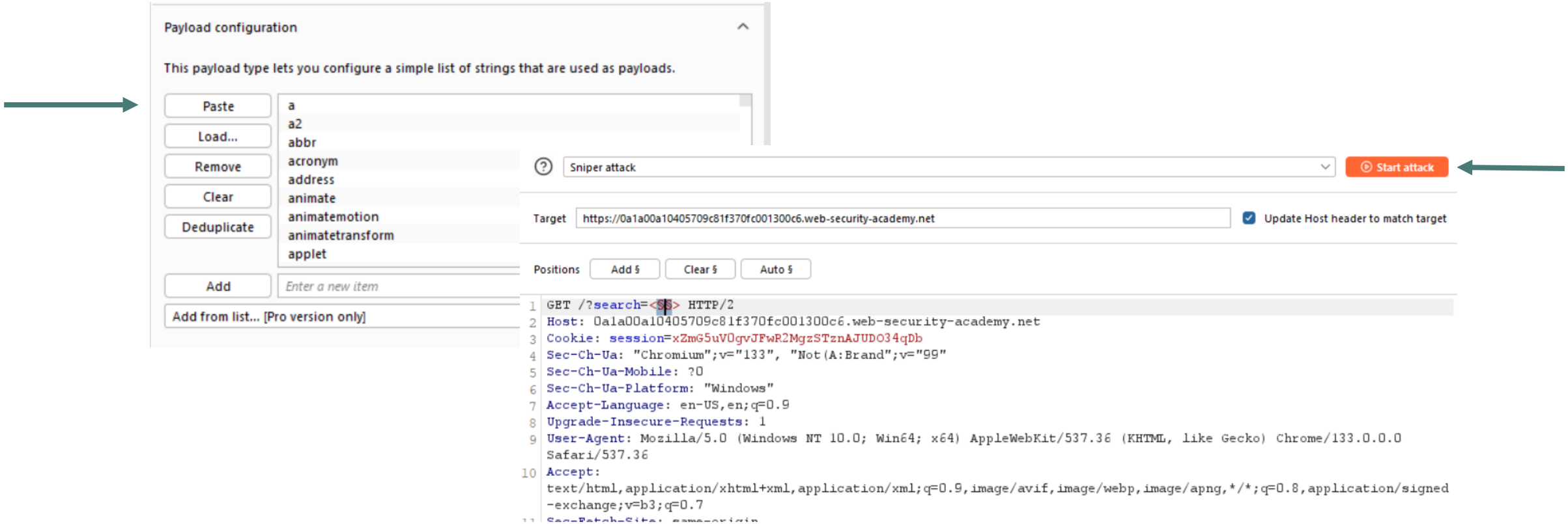
XSS LAB: REFLECTED XSS

- Then use the PortSwigger Cheat sheet to get the list of tags for XSS , by visiting this link: Cross-Site Scripting (XSS) Cheat Sheet - 2024 Edition | Web Security Academy



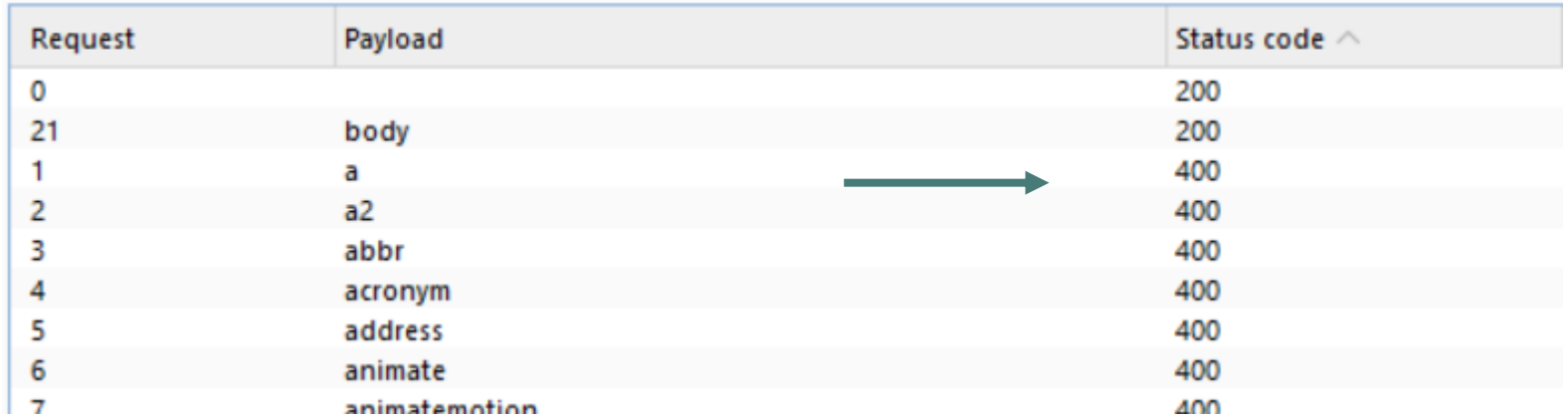
XSS LAB: REFLECTED XSS

- In Burp Suite, navigate to the payloads tap, go to the payload configuration section and past your payload
- Hit Start Attack and wait for the results.



XSS LAB: REFLECTED XSS

- Notice that most payloads caused 404, which is filtered out, except for the <body> tag, try to search for the body tag, nothing is shown or reflected in the page as a result! meaning it's accepted but not rendered. Perfect.
- Now get back to the cheat sheet and copy events to your clipboard, then Clear the previous payloads list and paste the events list, this will tell us which function we can use to trigger the XSS vulnerability.



Request	Payload	Status code
0		200
21	body	200
1	a	400
2	a2	400
3	abbr	400
4	acronym	400
5	address	400
6	animate	400
7	animatemotion	400

XSS LAB: REFLECTED XSS

- Update the request input section to match a function inside the body tag

  <body%20=1>

- The 20% means a space in URL encoding, place the cursor before the = sign and click add

- This time, most return 404 — but onresize returns 200.
  That means it works, and we can use it to exploit our XSS!

Positions | Add § | Clear § | Auto §

```
1 GET /?search=<body%20§§=1> HTTP/2
2 Host: 0ab300600487985980af03cc00c0002b.web-security-academy.net
3 Cookie: session=y7zTMHKKNATuic8IZwgKIOLgxchTUG1a
4 Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="99"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Accept-Language: en-US,en;q=0.9
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  Safari/537.36
```

Request	Payload	Status code	Response received
0		200	431
10	onbeforeinput	200	115
13	onbeforetoggle	200	174
17	oncancel	200	159
23	oncontentvisibilityautostatechange	200	246
24	oncontentvisibilityautostatechange(hidden)	200	166
33	ondragexit	200	118
46	onformdata	200	121
74	onpointercancel	200	116
85	onratechange	200	129
88	onresize	200	1384
90	onscrollend	200	147

XSS LAB: REFLECTED XSS

- These information would tell us that the crafted payload should look something like this:

```
"><body onresize=print()>"
onload=this.style.width='100px'>
```

- Go to exploit server (this is your server as a hacker) and create an iframe to insert the payload:

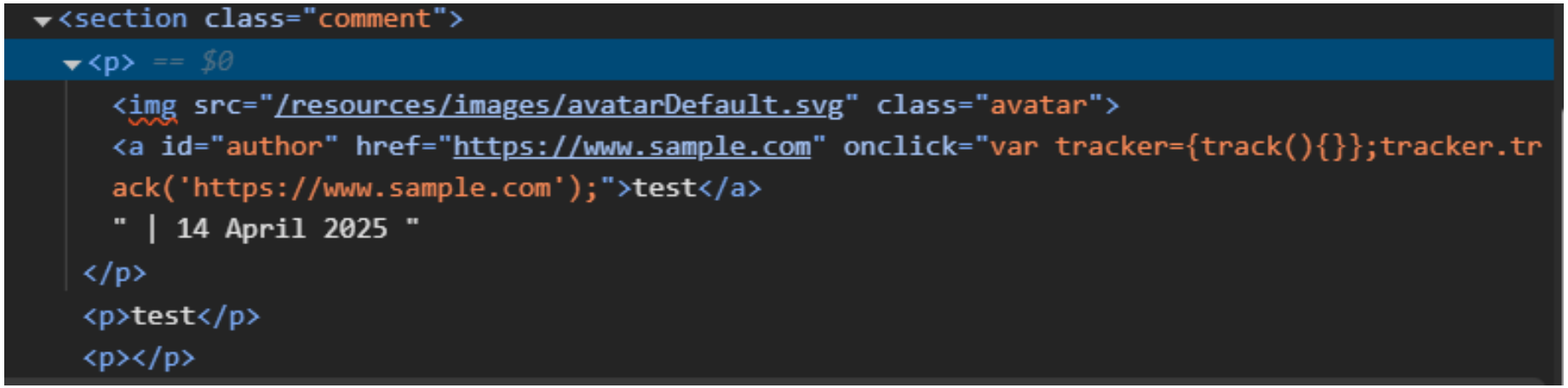
```
<iframe src="https://YOUR-LAB-ID.web-security-
academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E"
onload=this.style.width='100px'>
```

- Payload replace "YOUR-LAB-ID" with your lab ID from the lab URL

- Save and boom! You solved the lab!

XSS LAB: STORED XSS

- Start by posting a comment in the "website" input field, enter "https://sample.com" and submit. This will help us track where our input shows up later.
- Observe the page source code, you'll see your input (https://sample.com) is placed inside a tracker() function, which itself is inside an onclick event handler.



XSS LAB: STORED XSS

- Now, as the lab title, you may try:
  - angle brackets < >
  - double quotes "
  - single quotes '
  - HTML encoding
  - or even backslashes \
- But none of them will work.
- That's because the input is being escaped, meaning unsafe characters are being filtered or neutralized.

XSS LAB: STORED XSS

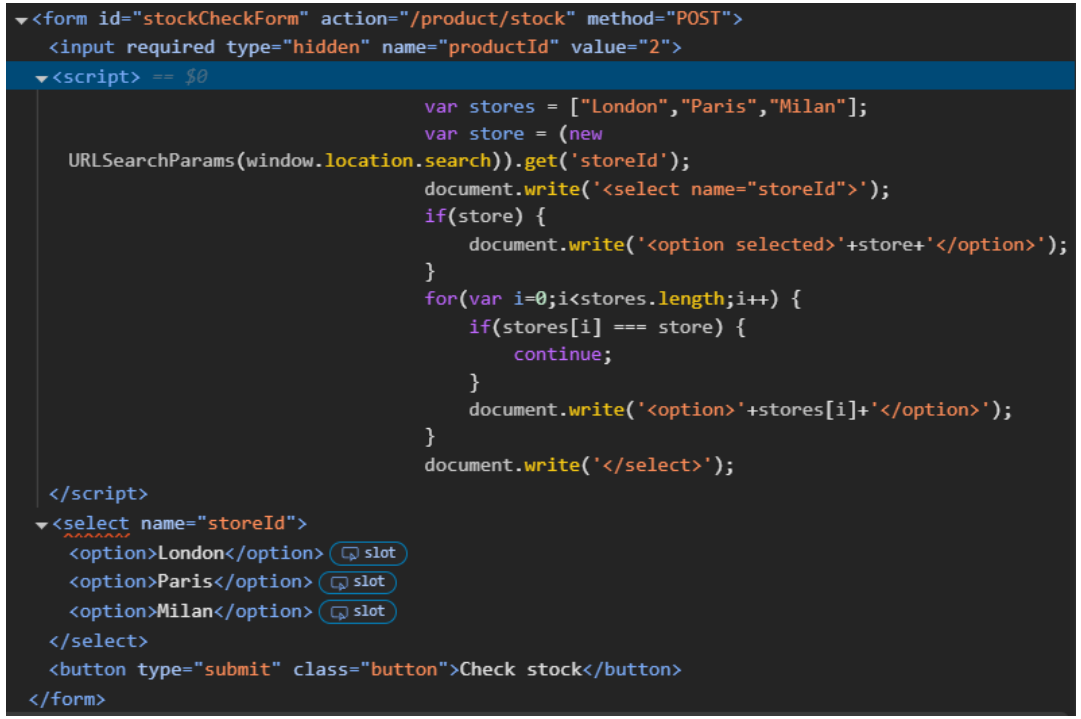
- So why not try something that looks safe, but still breaks the logic? (maybe malicios URL)?
- Try this:

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

- This is crafted to:
  - Break out of the string the app wraps your input in
  - Inject a valid JavaScript statement
  - Trigger an alert
- Click the link that appears on your comment...
- Boom! Alert box triggered!

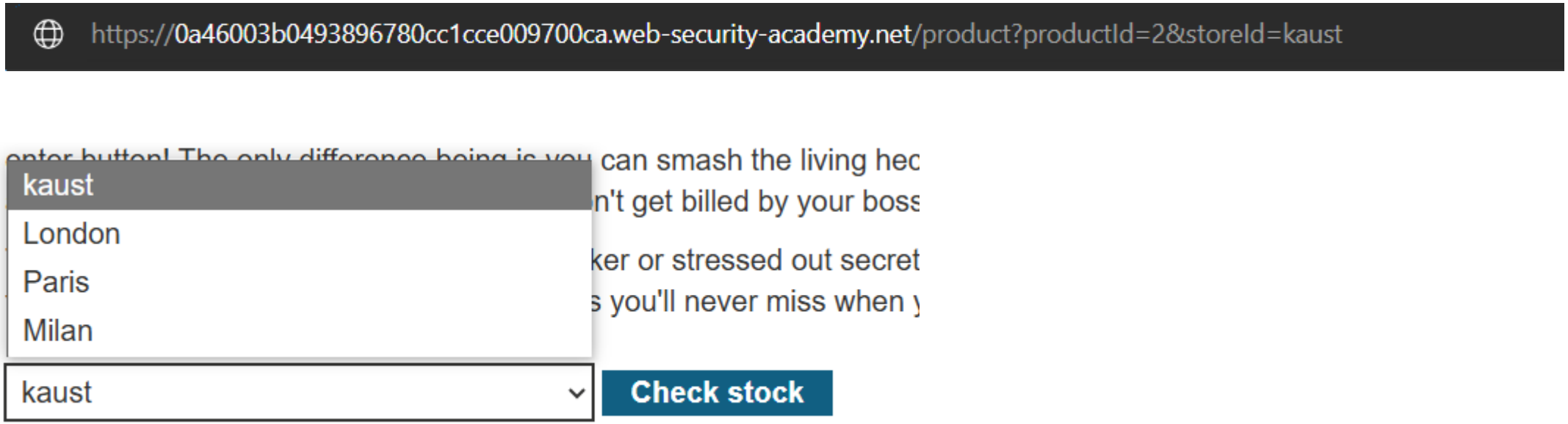
XSS LAB: DOM XSS

- Examine the website as we used to do, notice that there is no explicit input field that we can manipulate, so we will navigate to one of the product page
- At the bottom of the page, notice a dropdown menu used to check the number of unit in each stock location
- Check the source code, the JavaScript extracts the `storeId` value from the URL (using `location.search`). Then uses `document.write` to insert a new `<option>` into a dropdown menu (a `<select>` element) for stock checking, as u see when changing the stock location.



XSS LAB: DOM XSS

- Now Modify the page URL by appending a storeId parameter any random string, then check the drop-down menu again, you'll notice that your input to the URL parameter appears as a new option in the drop-down menu!



XSS LAB: DOM XSS

- Now let's break out of the HTML structure and inject our script.
- Update the URL to this payload:

?productId=1&storeId="></select><img src=1 onerror=alert(1)>

- "> closes the current <option> tag
- </select> closes the dropdown
- <img src=1 onerror=alert(1)> injects an image that fails to load and triggers the alert
- BOOM! Load the URL and the alert pops up, that means you successfully triggered an XSS via *query parameter injection*.
- This worked because your input wasn't sanitized and got directly written into the DOM via document.write.

SQL INJECTION LAB

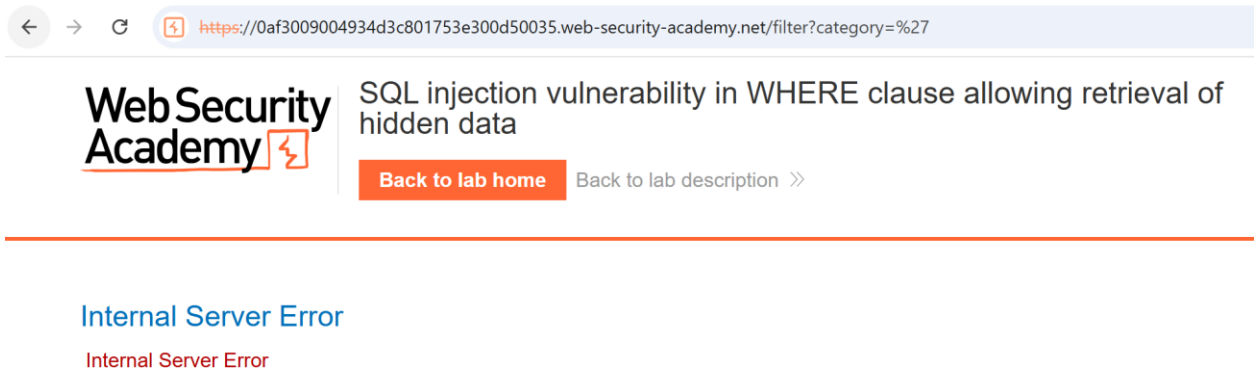
- Challenge Labs: https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data

- Challenge Labs (Optional): https://portswigger.net/web-security/sql-injection/lab-login-bypass

SQL LAB: VULNERABILITY IN WHERE CLAUSE ALLOWING RETRIEVAL OF HIDDEN DATA

- The lab simulates a shopping site where you can filter products based on category.
- Try one of the most used SQL characters, the single quote character, in the category parameter
- We get an internal server error which could indicate that the website contains an sql vulnerability. The left single quote resulted in a syntax error.

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

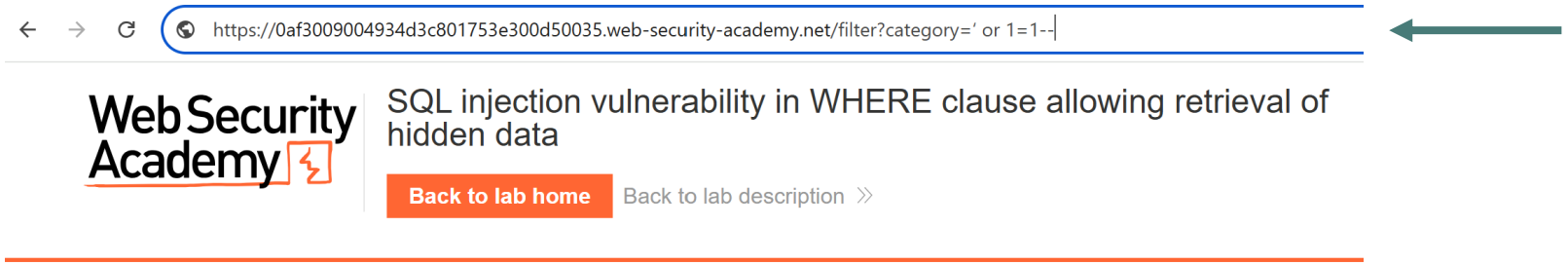


SQL LAB: VULNERABILITY IN WHERE CLAUSE ALLOWING RETRIEVAL OF HIDDEN DATA

- Now, we need the application to respond with hidden data instead of an error.
- We'll add a comment field to ignore the rest of the query, so it doesn't run into an error.

```
SELECT * FROM products WHERE category = ''--' AND released = 1
```

- This time, we don't get an error! The category parameter is set to nothing, so we don't get any products information.
- To solve the lab, we need to display all products in the application.
- Use the following SQL injection payload in the category parameter: `' or 1=1--`
- The conditional statement 1=1 will evaluate to true so the query will display all the rows in the products table.



SQL LAB: VULNERABILITY ALLOWING LOGIN BYPASS

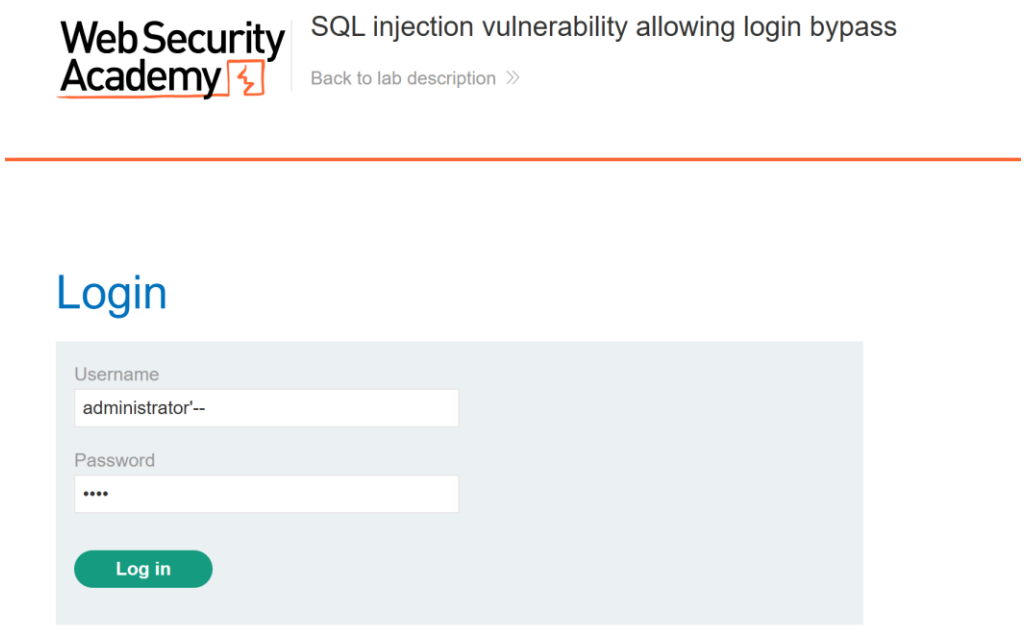
- Unlike the previous exercise, we're not given the SQL query, so we'll try to find it out by fuzzing the application.
- Since we're trying to log in to the application as an administrator user, it's common to think how we can ignore the password field.
- We'll try to put SQL characters in the username to get the application to ignore checking the password.
- Modify the username parameter, giving it the value: administrator'--
- The single quotation closes the username field, and the comment character skips the remaining part of the query

SQL LAB: VULNERABILITY ALLOWING LOGIN BYPASS

- The SQL query in the backend is probably similar to this:

- `SELECT firstname FROM users WHERE username='username' and password='password'`

- Log in as an administrator to solve the lab.

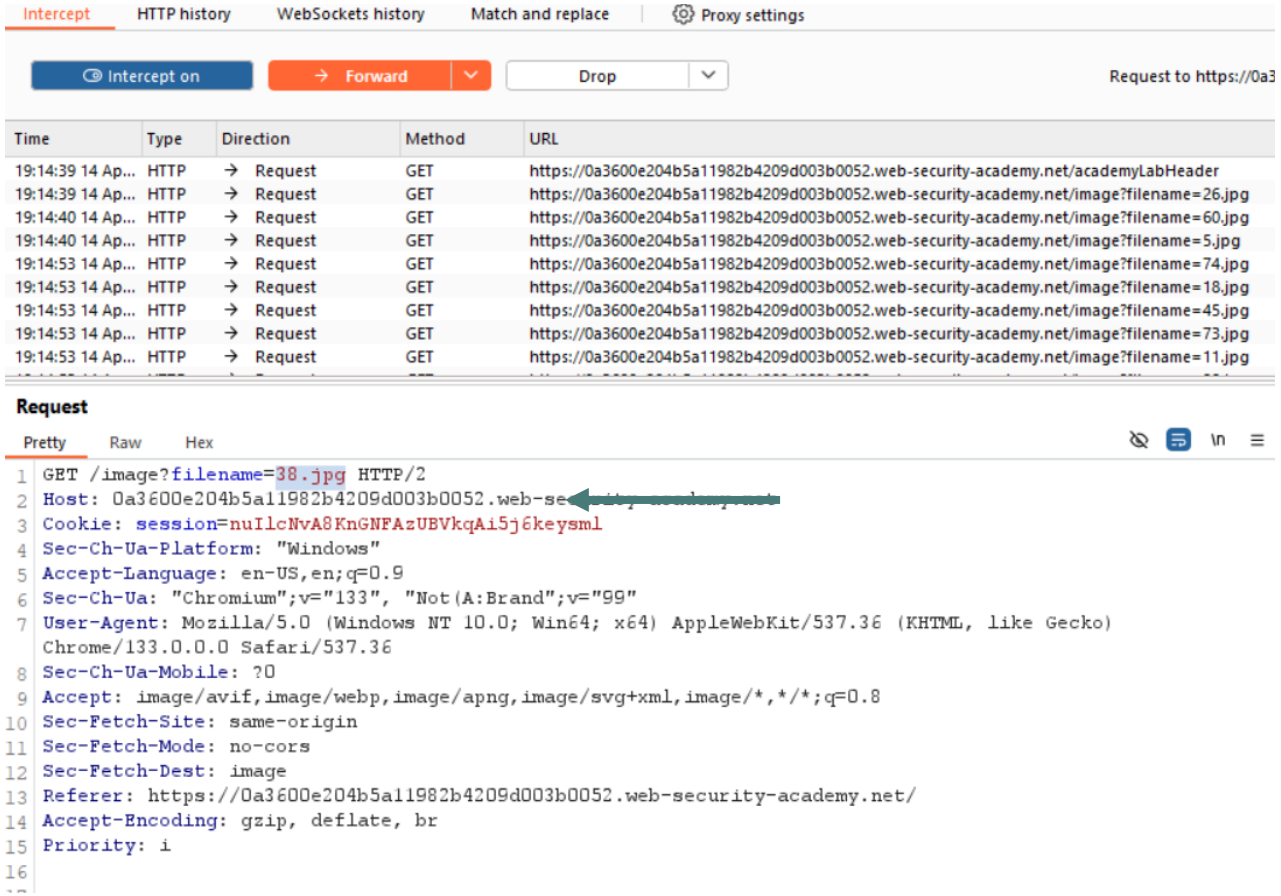


PATH TRAVERSAL LAB

- Lab: Lab: File path traversal, simple case | Web Security Academy
- In this lab, we'll explore the Path Traversal vulnerability which allows an attacker to access files outside the intended directory.
- The website looks like a shopping site displaying products, each with a "View Product" button.
- When you click on a product, you'll notice that the URL updates with a numeric parameter, this means the site is retrieving specific data for that product dynamically.
- But this isn't what we care about just yet...

PATH TRAVERSAL LAB

- Copy the lab's URL.
- Open Burp Suite and intercept the request.
- Refresh the home page, you'll see multiple requests being made to load product details and images
- Check the filename parameter, That's the one we want! The filename parameter is taking a file name, and that's exactly what we can exploit!



PATH TRAVERSAL LAB

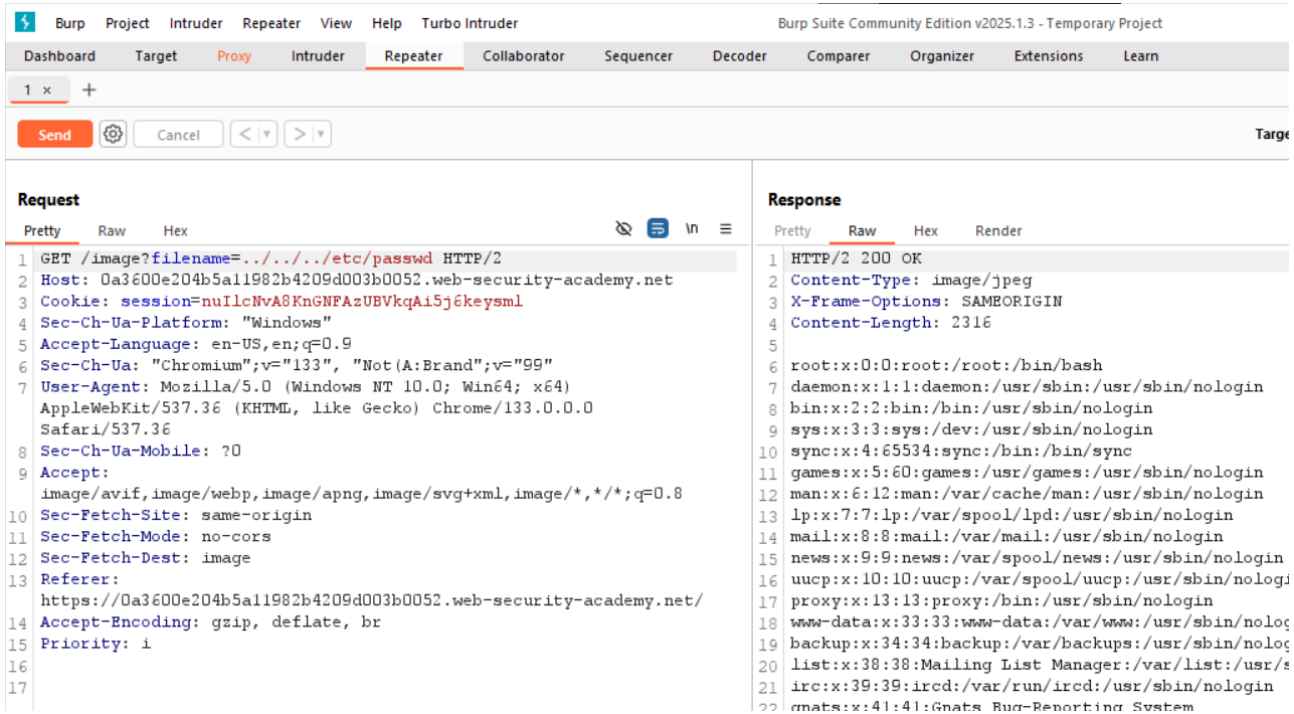
- Send the image request with the filename param to Repeater.
- Now, let's attempt a path traversal attack.
- Let's try to breach the password file of the server by typing this payload instead of the image name:

```
../../../etc/passwd
```

- Hit send, and If successful,

the server will respond with the contents of the /etc/passwd file.

a common Linux file that lists user accounts. we've just exploited a Path Traversal Vulnerability to access sensitive server data!



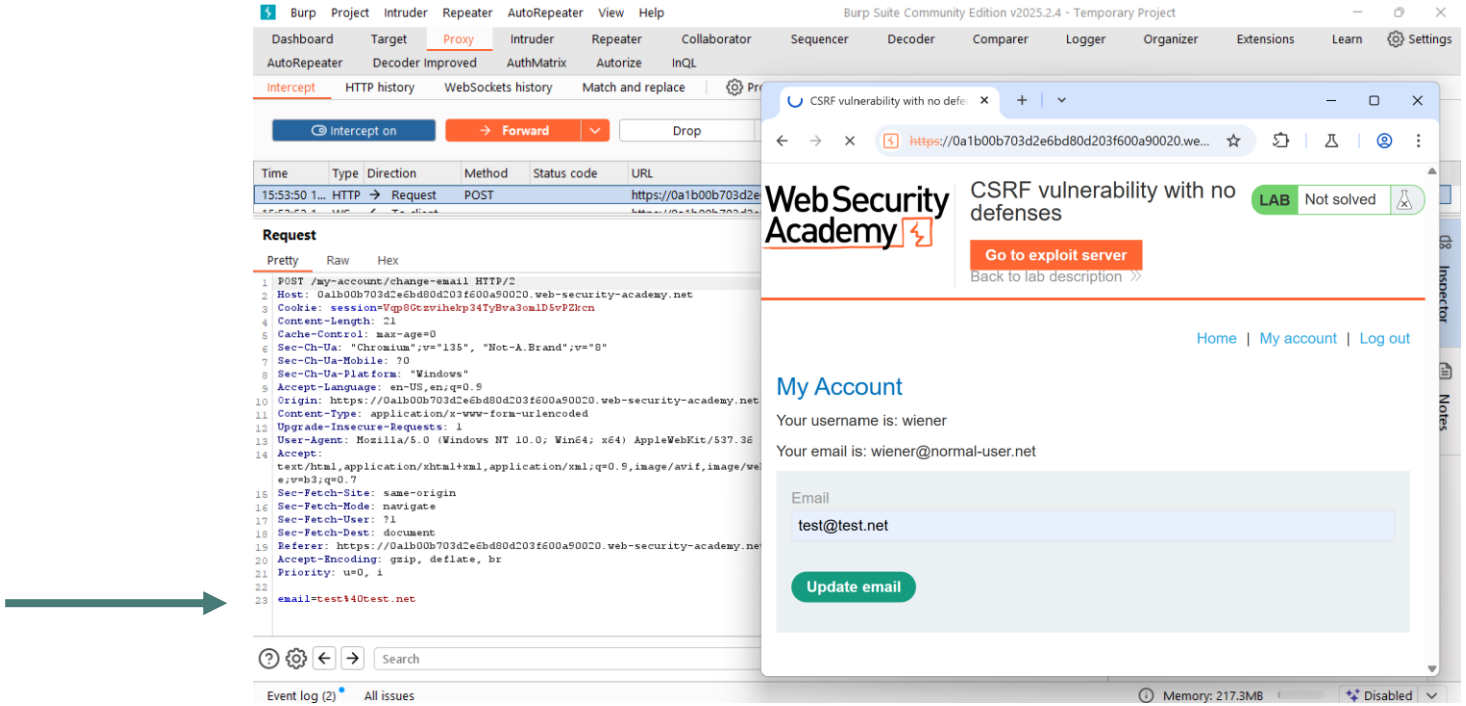
CROSS-SITE REQUEST FORGERY: CSRF LABS

- Challenge Lab: https://portswigger.net/web-security/csrf/lab-no-defenses

- Challenge Lab (Optional): https://portswigger.net/web-security/csrf/bypassing-token-validation/lab-token-validation-depends-on-request-method

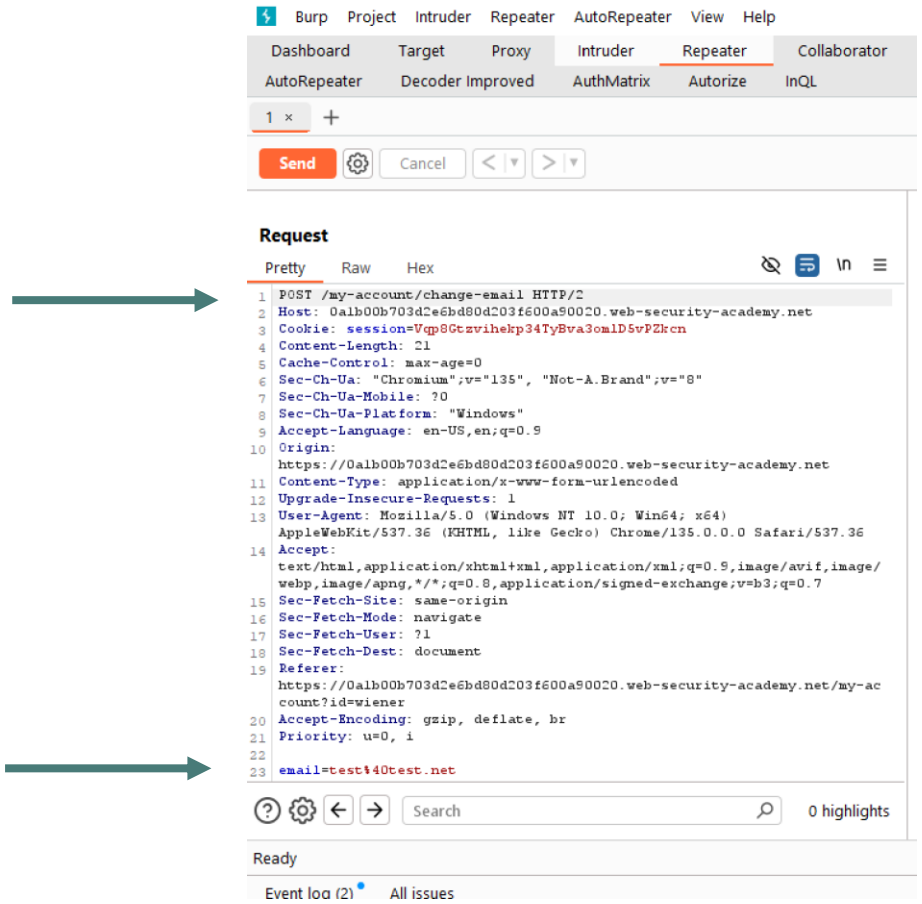
CSRF LAB: CSRF VULNERABILITY WITH NO DEFENSES

- Access the lab and use the given credentials wiener:peter.
- The account page contains an email change functionality which is vulnerable to CSRF. Turn on your proxy then update your email address.
- Send the email change request to Repeater.



CSRF LAB: CSRF VULNERABILITY WITH NO DEFENSES

- The request has an action (email change), cookie-based session handling, and predictable request parameter (email) which could possible be vulnerable to CSRF.

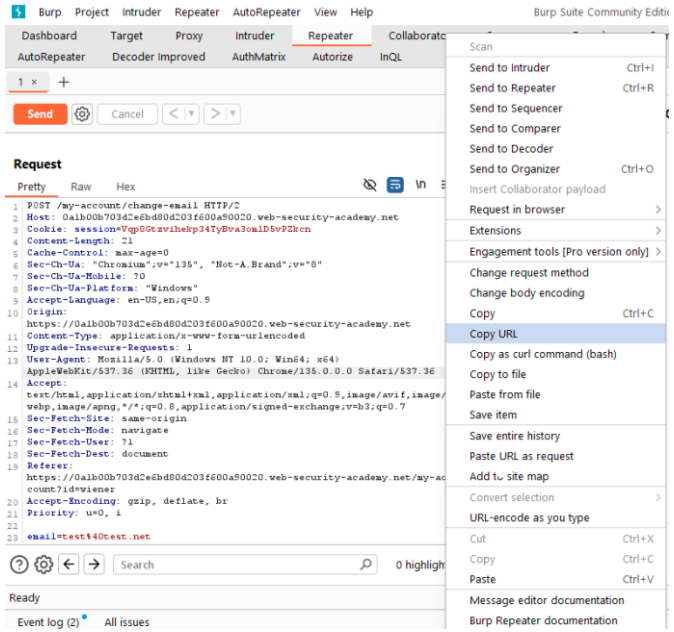


CSRF LAB: CSRF VULNERABILITY WITH NO DEFENSES

- To exploit this vulnerability, use the following CSRF script.

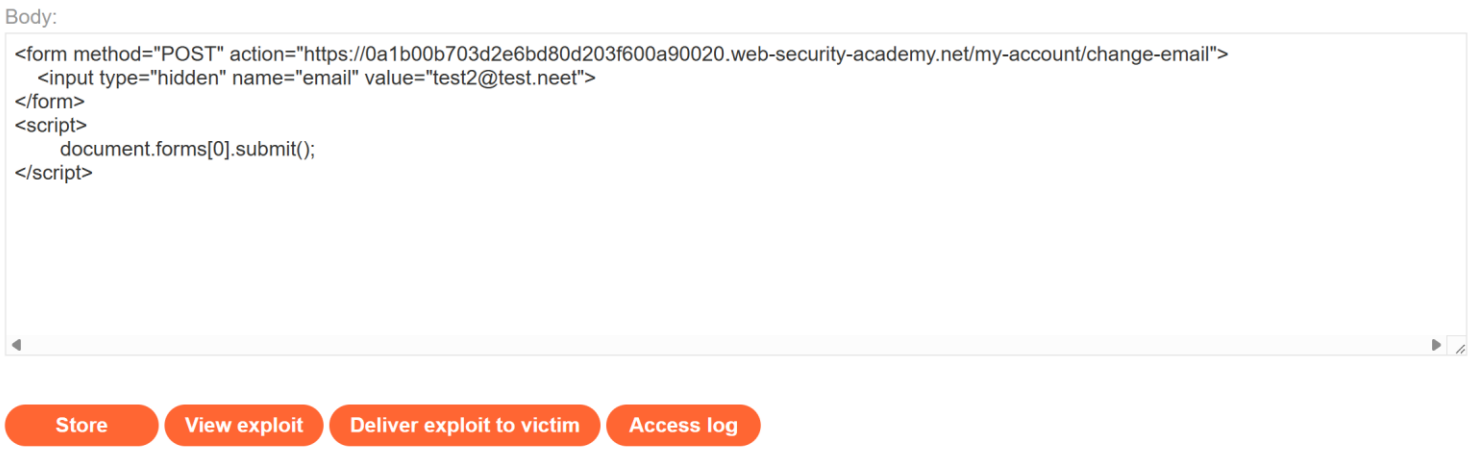
```
<form method="POST" action="https://YOUR-LAB-ID.web-security-academy.net/my-account/change-email"> <input type="hidden" name="email" value="anything%40web-security-academy.net"> </form>
<script> document.forms[0].submit(); </script>
```

- You can get the request URL by right-clicking and selecting "Copy URL".



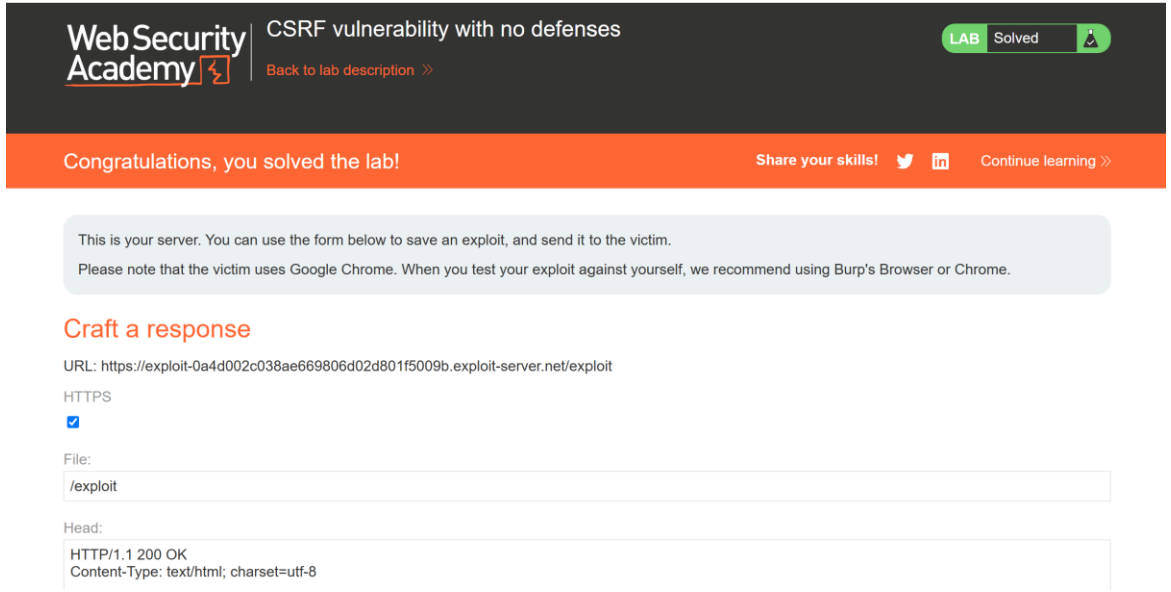
CSRF LAB: CSRF VULNERABILITY WITH NO DEFENSES

- Go to the exploit server, paste your exploit script into the "Body" section, and click "Store". Change the email address in your exploit so that it doesn't match your own.
- The exploit uses a hidden HTML form with a preset email input and a script that automatically submits the form, triggering the email change request as soon as the page loads.
- Click "Deliver exploit to victim" to solve the lab.



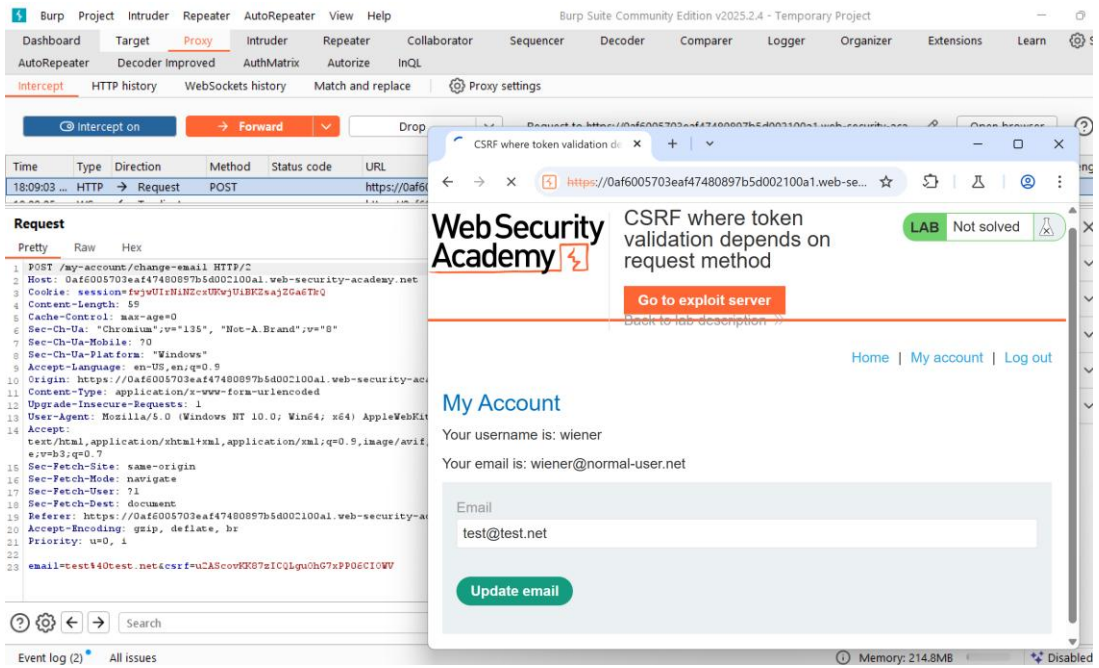
CSRF LAB: CSRF VULNERABILITY WITH NO DEFENSES

- This lab demonstrates a CSRF vulnerability, where the email change functionality lacks proper protection to verify that requests are made intentionally by the authenticated user.
- As a result, an attacker can trick a logged-in user into unknowingly submitting a request to change their account email by embedding malicious HTML in a third-party site.



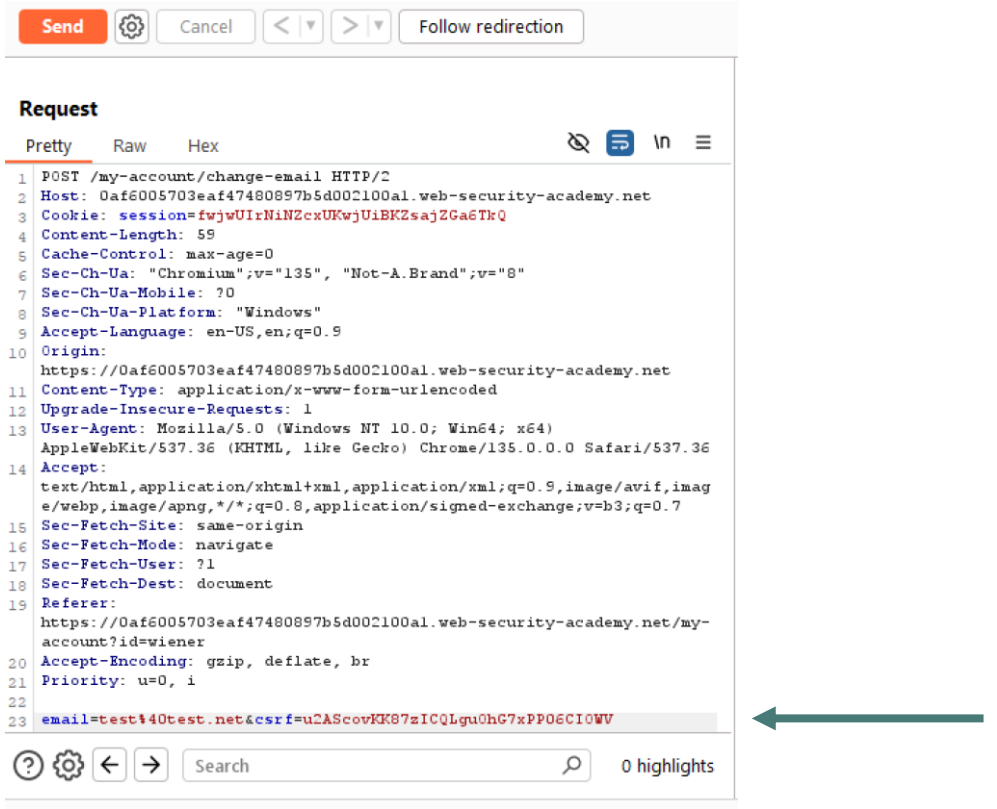
CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- Access the lab and use the given credentials wiener:peter.
- The account page contains an email change functionality which is vulnerable to CSRF. It attempts to block CSRF attacks but only applies defenses to certain types of requests.
- Turn on your proxy then update your email address. Send the email change request to Repeater.



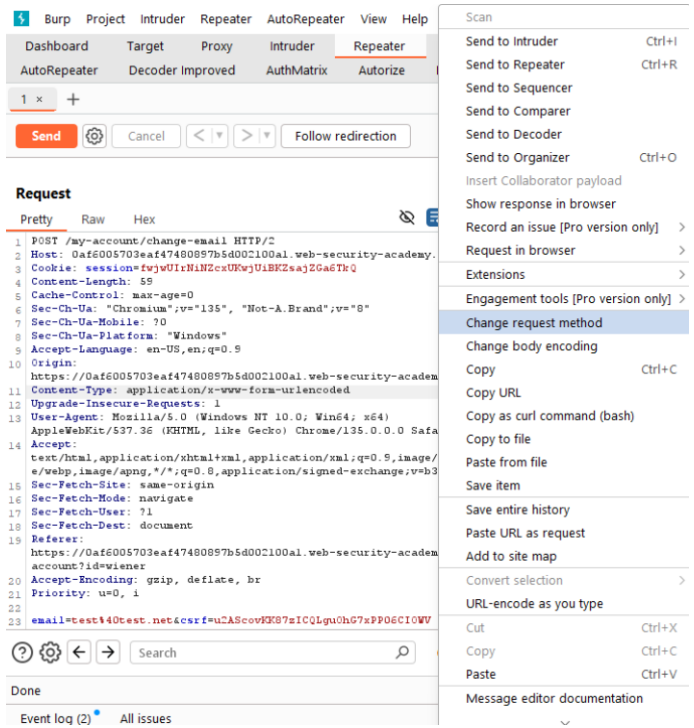
CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- There's a CSRF token in the request body which at first glance makes it seem like this is not vulnerable to CSRF.
- However, depending on the implementation of the functionality behind the CSRF token generation, it might be vulnerable.



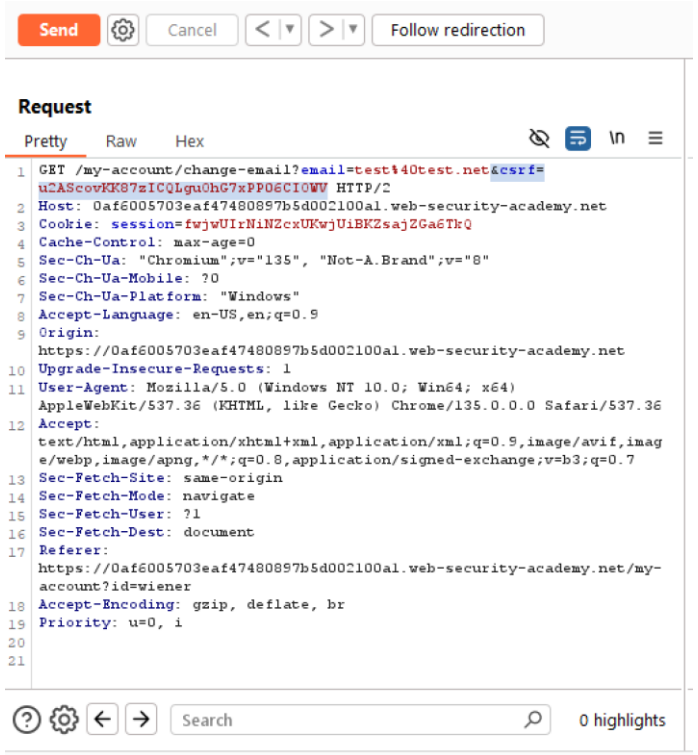
CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- One of the CSRF testing techniques, is to change the request method from POST to GET.
- Sometimes CSRF is set only in POSTS method because they're used to send data to the application. If you place a CSRF token in a GET method, it doesn't really matter because it usually only outputs data from the application.



CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- If the application allow using a GET method to send data to the application, then this might be vulnerable to CSRF.
- Remove the CSRF token and observe that the CSRF token is no longer verified.



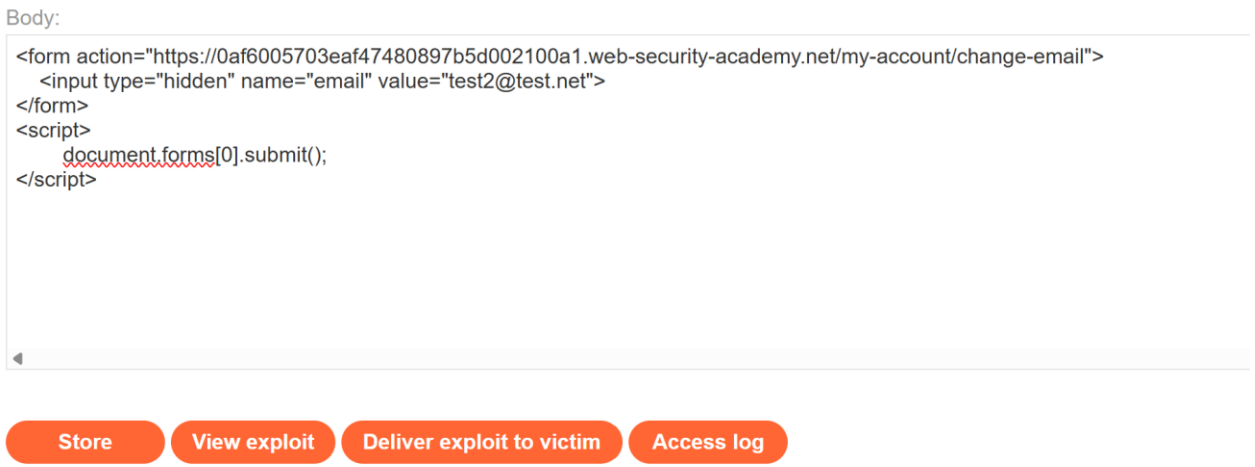
CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- To exploit this vulnerability, use the following CSRF script.

```
<form action="https://YOUR-LAB-ID.web-security-academy.net/my-
account/change-email"> <input type="hidden" name="email"
value="anything%40web-security-academy.net"> </form> <script>
document.forms[0].submit(); </script>
```

CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- Go to the exploit server, paste your exploit script into the "Body" section, and click "Store". Change the email address in your exploit so that it doesn't match your own.
- The exploit uses a hidden HTML form with a preset email input and a script that automatically submits the form, triggering the email change request as soon as the page loads.
- Click "Deliver exploit to victim" to solve the lab.



Questions?